Artificial Intelligence-Powered Classification of Flora in Vineyards

Ana Catarina Antunes Corceiro

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletrotécnica e de Computadores
(2º ciclo de estudos)

Orientador: Prof. Doutor Pedro Miguel de Figueiredo Dinis Oliveira Gaspar
Co-orientador: Mestre Nuno José Matos Pereira

Covilhã, outubro de 2023

Declaração de Integridade

Eu, Ana Catarina Antunes Corceiro, que abaixo assino, estudante com o número de inscrição M10933 de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o **Código de Integridades da Universidade da Beira Interior**.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 06/10 /2023

Assinado por: **Ana Catarina Antunes Corceiro**
Num. de Identificação: 15539371
Data: 2023.10.06 13:41:54+01'00'

CHAVE MÓVEL

(assinatura conforme Cartão de Cidadão ou preferencialmente

assinatura digital no documento original se naquele mesmo formato

Dedicatória

Ao meu avô, Joaquim Antunes.

Agradecimentos

Foi um ano de desafios e superação do qual eu nunca me irei esquecer. Definitivamente contribuiu para o meu desenvolvimento profissional e sobretudo pessoal.

A realização desta dissertação não seria possível sem o apoio, conselhos e conhecimento que diversas pessoas me transmitiram. Desta forma, expresso o meu mais profundo agradecimento a todos aqueles que me permitiram alcançar este sonho.

Aos meus pais, Adelina e Inácio, o meu mais sincero obrigado, por acreditarem em mim e nas minhas capacidades, incutindo-me sempre princípios fundamentais para a minha vida académica, profissional e especialmente pessoal. Um obrigado nunca será suficiente para agradecer todo o esforço e sacrifícios demonstrado que me permitiu chegar aqui hoje e ser quem sou. Sem vocês nada disto seria possível. E se sou quem sou hoje é a vocês que o devo. Por isso, mãe e pai, o meu mais sincero obrigado!

À minha família, em especial à Margarida por todo o apoio e amizade e conselhos que me deu toda a minha vida. Também ao meu avô, que desde sempre acreditou em mim e me apoiou da melhor forma possível. Um muito obrigado a todos!

Ao meu orientador, Professor Doutor Pedro Miguel de Figueiredo Dinis Oliveira Gaspar, que me deu a possibilidade de trabalhar neste tema, por toda a ajuda, conselhos e disponibilidade. Um obrigado por todo o conhecimento e apoio prestado não só no desenvolvimento desta dissertação, mas também ao longo de todo o meu percurso académico.

Ao meu co- orientador Mestre Nuno Pereira, que sempre demonstrou prontidão em auxiliar-me, por todo o apoio e conhecimento que me deu ao longo deste percurso. Graças aos seus incentivos e opiniões foi possível tornar o trabalho mais notável e foi também possível experienciar uma aprendizagem enriquecedora. Sem ele a realização deste trabalho teria sido impossível.

À Doutora Khadijeh Alibabaei, por todo o apoio e por me ter ajudado no início deste projeto.

Não menos importante, quero agradecer ao meu namorado Rodrigo que sempre esteve ao meu lado, nos momentos bons e maus. Se consegui superar os desafios ao longo deste ano foi também graças a ele e a todo o apoio, amizade e amor que me transmitiu. Foi um ano de partilha de conhecimentos, ajuda, gargalhadas e choros, mas conseguimos chegar a este momento graças um ao outro. Agradecer-te por tudo não é suficiente nem é um gesto que se escreve, é algo que se partilha ao longo da vida. No entanto, muito obrigado por tudo Rodrigo!

Em especial à Matilde por todo o apoio, disponibilidade e amizade! E aos meus amigos que sempre estiveram ao meu lado.

Por fim, deixar um agradecimento à Universidade da Beira Interior, à Faculdade de Engenharia pela disponibilização do espaço e equipamentos necessários para a realização deste trabalho. E, a todos os docentes durante o meu percurso académico, obrigado pelos conhecimentos transmitidos.

Muito obrigado a todos do fundo do meu coração!

Resumo

O crescimento populacional global em ritmo acelerado tem exercido uma considerável pressão sobre o setor agrícola, que se vê obrigado a crescer 70% para atender à crescente necessidade de produtos alimentares. O maior desafio com que produtores agrícolas se deparam está ligado à existência de plantas que se tornam infestantes. Estas, não só competem com os cultivos por recursos vitais, como também representam ameaças económicas e ambientais significativas. No contexto da gestão abrangente da vinha, o controlo da vegetação torna-se especialmente crítico, influenciando a prevalência de várias pragas e enfatizando a necessidade de métodos de controlo de plantas ambientalmente sustentáveis. Apesar dos herbicidas terem sido inicialmente empregues com o objetivo de controlar o crescimento de plantas indesejadas, o seu uso excessivo e indiscriminado desses químicos provoca poluição ambiental e desenvolvimento de resistências por parte da vegetação infestante.

Para lidar com estes desafios, o sector agrícola tem vindo a apostar em modelos de Machine Learning (ML). Os avanços na Inteligência Artificial, particularmente nas Redes Neurais Convolucionais, emergiram como uma solução predominante para resolver problemas relacionados com este tipo de plantas. A Agricultura de Precisão aproveitou a tecnologia de algoritmos de ML como recurso para classificação de imagens para distinguir plantas cultivadas e indesejáveis.

Este trabalho pretende contribuir para a promoção da agricultura sustentável e para o avanço da classificação de imagens no domínio da classificação de plantas indesejáveis. Os objetivos principais envolvem o desenvolvimento de algoritmos utilizando redes neurais convolucionais para classificação de flora usando a estrutura PyTorch, com foco na otimização dos hiperparâmetros.

Uma revisão abrangente da área enfatiza a diversidade e a singularidade das plantas, bem como os métodos de aquisição de dados, índices de vegetação utilizados, métricas de avaliação, a redes neurais convolucionais bem como os seus PyTorch modelos. Para além disto, faz a análise de avanços recentes em modelos e métodos de ML para detetar e classificar ervas daninhas para melhorar a sustentabilidade das culturas agrícolas.

Os resultados destacam o excelente desempenho dos modelos MaxVit, ShuffleNet e EfficientNet, especialmente quando confrontados com um conjunto de dados expandido. A escolha dos hiperparâmetros, incluindo taxa de aprendizagem, configuração das camadas e redução de peso, influenciou significativamente a exatidão do modelo. Ao testar os modelos em uma aplicação web, o EfficientNet_B1 e o EfficientNet_B5 obtiveram uma exatidão excecional de 96.15%, destacando-se entre todos os modelos.

Estes modelos têm o potencial de revolucionar a agricultura, impulsionando a produtividade e a sustentabilidade ambiental. Ao integrar esses modelos em soluções tecnológicas, os agricultores podem monitorizar a saúde das plantas, identificar pragas e plantas indesejáveis, e otimizar o uso de recursos naturais como água e fertilizantes. A automação na deteção de

problemas nas plantações reduz o desperdício das culturas e minimiza o uso de herbicidas, promovendo uma agricultura ecológica. Além disso, ao prever padrões climáticos e de crescimento das plantas, os agricultores podem otimizar seus calendários de plantio e colheita, maximizando a produtividade. A tecnologia também é crucial para a agricultura de precisão, permitindo tratamento individualizado das plantas e redução do desperdício de recursos. Ao criar sistemas de monitorização em tempo real, os agricultores podem tomar decisões informadas, melhorando a adaptabilidade das culturas diante de desafios climáticos. Em última análise, ao incorporar modelos PyTorch, a agricultura torna-se mais eficiente, minimiza o desperdício e reduz os impactos ambientais, contribuindo para um futuro mais verde e resiliente para a agricultura.

Palavras-chave

Machine learning; Classificação de plantas; Redes Neurais Convolucionais; Agricultura de precisão; Pytorch.

Abstract

Rapid global population growth has put considerable pressure on the agricultural sector, which is forced to grow by 70 % in order to meet the growing demand for food. The biggest challenge facing agricultural producers is the existence of plants that become weeds. These not only compete with crops for vital resources, but also pose significant economic and environmental threats. In the context of comprehensive vineyard management, vegetation control becomes especially critical, influencing the prevalence of various pests and emphasising the need for environmentally sustainable plant control methods. Although herbicides were initially used to control the growth of unwanted plants, the excessive and indiscriminate use of these chemicals leads to environmental pollution and the development of resistance on the part of weed vegetation.

In order to deal with these challenges, the agricultural sector has been relying on ML models. Rapid advances in Artificial Intelligence (AI), particularly Convolutional Neural Networks (CNN), have emerged as a predominant solution for solving problems related to this type of plant. Precision Agriculture (PA) has taken advantage of ML algorithm technology as a resource for classifying images to distinguish cultivated and undesirable plants.

This work aims to contribute to the promotion of sustainable agriculture and to the advancement of image classification in the field of plant classification. The main objectives involve the development of algorithms using CNN for classification using the PyTorch framework, with a focus on hyperparameter optimisation.

A comprehensive review of the area emphasises plant diversity and uniqueness, as well as data acquisition methods, vegetation indices used, evaluation metrics, CNN as well as its PyTorch models. It also analyses recent advances in ML models and methods for detecting and classifying weeds to improve the sustainability of agricultural crops.

The results highlight the excellent performance of the MaxVit, ShuffleNet and EfficientNet models, especially when faced with an expanded dataset. The choice of hyperparameters, including learning rate, layer configuration and weight reduction, significantly influenced model accuracy. When testing the models in a web application, EfficientNet_B1 and EfficientNet_B5 obtained an exceptional accuracy of 96.15 per cent, standing out among all the models.

These models have the potential to revolutionise agriculture, increasing productivity and environmental sustainability. By integrating these models into technology solutions, farmers can monitor crop health, identify pests and undesirable plants, and optimise the use of natural resources such as water and fertiliser. Automating the detection of crop problems reduces crop waste and minimises the use of herbicides, promoting organic farming. In addition, through the prediction of weather conditions and plant growth patterns, farmers can optimise their planting and harvesting schedules to maximise productivity. Technology is also crucial to precision farming, enabling individualised treatment of crops and a reduction in wasted resources. By creating real-time monitoring systems, farmers can make informed decisions and improve crop

adaptability in the face of climatic challenges. Ultimately, by incorporating PyTorch models, farming becomes more efficient, minimises waste and reduces environmental impact, contributing to a greener and more resilient future for agriculture.

Keywords

Machine learning; Plants classification; CNN; Precision Agriculture; PyTorch.

Index

Dedicatória ..... **i**
Agradecimentos ..... **i**
Resumo ..... **iii**
Abstract ..... **v**
Index ..... **vii**
List of Figures ..... **ix**
List of Tables ..... **xv**
Nomenclature ..... **xvii**

1. Introduction **1**
1.1. Background 1
1.2. Study Problem and Relevance 3
1.3. Objectives and contributions 5
1.4. Document organization 6
2. State of the Art **7**
2.1. Plant Agrobiodiversity 7
2.2. Data acquisition 12
2.3. Vegetation indices 14
2.4. Performance Metrics 16
2.5. Neural Networks Overview 20
2.6. Loss Function and Optimizers 26
2.7. Neural Network Architectures 28
2.8. Experimental Studies 34
2.8.1. Disease Detection 34
2.8.2. Weed Detection 39
2.8.3. Weed Classification 55
2.8.4. Fruit Detection 65
2.9. Discussion 70
2.10. Conclusive Remarks 74
3. Materials and Methods **77**
3.1. PyTorch Models 77
3.2. Data Collection and Sample Preparation 78
3.3. Algorithm Development 80
3.4. Simulation of the Algorithms 86
4. Experimental Results **89**
4.1. Experiment 1: Simulating PyTorch Classification Models 89

4.1.1. MaxVit, ShuffleNet and EfficientNet Models in depth 101
4.2. Experiment 2: Evaluating Model Performance Across Epochs and Establishing Baselines 107
4.3. Experiment 3: Exploring Model Performance with an Expanded Image Dataset..... 114
4.4. Experiment 4: Testing Models in a Web Application- Real-world Performance Evaluation 123
4.5. Discussion of results..... 131
5. Conclusions 135
5.1. General Conclusions..... 135
5.2. Suggestions of Future Work..... 137
References 139

Appendix A..... 149

List of Figures

Figure 1. Example of the two types of leaves according to the plant group. (a) Broad leaves, and (b) Narrow leaves.....9
Figure 2. Example of the two types of plants according to the habitat. (a) Terrestrial; (b) Aquatic; (c) Indifferent. (Diego *et al.,* 2021), (UTAD, 2023), (Flora-On | Flora de Portugal, 2022)..10
Figure 3. Electromagnetic spectrum. Adapted from (Ferlic, 2019).....13
Figure 4. Spectrum of electromagnetic waves in the infrared (Charlton *et al.,* 2017). 13
Figure 5. Area under the ROC Curve (Rushikanjaria, 2022).....20
Figure 6. Typical architecture of a Multilayer Perceptron. 21
Figure 7. Typical architecture of a Recurrent Neural Network. 22
Figure 8. Convolutional neural network architecture. 23
Figure 9. Convolutional operation. (*Deep Learning*, n.d.). 24
Figure 10. Pooling operation. (Mishra, 2020). 25
Figure 11. Representation of a convolutional neural network with two hidden layers (*Using Deep Learning Models / Convolutional Neural Networks*, n.d.). 26
Figure 12. A visual representation of the input fruit's divided into unhealthy (yellow) and diseased (black) portions (Azgomi *et al.,* 2022).....36
Figure 13. In the healthy region, an illustration of segmentation and fusion. Visible image (a), IR image (b), Visible ground truth (c), IR ground truth (d), Fusion ground truth (e), Visible SegNet estimate (f), IR SegNet estimation (g), and Fusion of segmentation results (h). A region affected by mildew is used as an example for segmenting and fusing. Visible image (i), Infrared image (j), Visible ground truth (k), IR ground truth (l), Fusion ground truth (m), Visible SegNet estimation (n), (o): IR SegNet estimation (o), Fusion of segmentation results (p), (Adapted from Kerkech *et al.,* 2020).....37
Figure 14. Results of the suggested approach. Original pictures (a), real-world images (b), FCN output (c), and U-Net output (d), (Adapted from Ma *et al.,* 2019).....41
Figure 15. The first row of photos in the PSPNet results are the original photographs, the second row is the predicted output, and the last image is the actual image. The paddy is in the first line, followed by the broadleaved weed in the second, the paddy in the third, the broadleaved weed (blue), and the sedges weed in the fourth (Kamath *et al.,* 2022). 43
Figure 16. Results of the qualitative segmentation of the CWFID and weeds dataset. The labels of each sub image show the optimization that was performed to the model to achieve the segmentation, with the exclusion of input and ground truth (Assunção *et al.,* 2022). 46
Figure 17. Application in real-time results in the segmentation of weeds (Assunção *et al.,* 2022). 47
Figure 18. Four sampling frames' examples of results. Images taken on-location; manually classifying observed data; and classifying images using the OBIA approach are shown in (A),

(B), and (C), respectively. (1) Correct categorization; (2) Underestimation of weeds; (3) Negative mistakes; and (4) False Positive mistakes (Peña *et al.,* 2015).....48
Figure 19. FCN classification results using several pre-trained CNNs. Actual UAV picture, ground truth results, and results from FCN-AlexNet, FCN-VGG16, and FCN-GoogleNet are shown in (a), (b), and (c), respectively (Huang *et al.,* 2018). 49
Figure 20. On the top, there are several examples of UAV picture classification using models made from unsupervised data in spinach fields, and bean fields at the bottom. In blue there are crops, in red are weeds, and in white uncertain decision (Bah *et al.,* 2018).....50
Figure 21. Results of the SBWD algorithm's : (a) The initial RGB image; (b) The segmented plants obtained using the ExG method; (c) The outcome of the morphological filtering of small objects; (d) The segmented sugar beets obtained using the SBWD algorithm; (e) The weeds removed from the segmented sugar beets obtained using the SBWD algorithm; and (f) The result of the SBWD algorithm displaying the false negatives, weeds, and sugar beets (Bakhshipour & Jafari, 2018). 58
Figure 22. The seven frames' qualitative outcomes (row-wise). The first three columns contain input data for CNN, whereas the fourth and fifth columns display ground facts and probability forecasts of the suggested model (Sa *et al.,* 2018).....59
Figure 23. The results of vegetable detection using the CenterNet detection model under varied situations are shown in the first line. The final segmentation findings with vegetable sections highlighted in red boxes are shown in the second line. (Jin *et al.,* 2021). 60
Figure 24. The impact of RTFD model detection on the tomato and strawberry datasets. The blue circles denote locations of inaccurate or missing detections, while the blue arrows act as suggestive indicator symbols. Red, orange/yellow, and light blue colours represented full, half-mature, and immature strawberries, respectively. (Mao *et al.,* 2022).....66
Figure 25. An example of an object detection result from each of the three neural networks—Mask R-CNN, YOLOv2 and YOLOv3—for each type of grape. same colour does not imply connect ion. (Adapted from Santos *et al.,* 2017).....67
Figure 26. Detection samples for the Royal Time, Sweet Dream, and Catherine varieties of peaches are shown on the left, middle, and right, respectively. (Adapted from Assunção *et al.* 2022). 68
Figure 27. Illustration of the five species used for image classification task.....79
Figure 28. Dataset split for model training and validation: visualizing species-specific distribution. The blue line represents the number of images for train of each specie, and the orange one, the number of images for validation of each specie. 80
Figure 29. Description of the steps for algorithm development. 84
Figure 30. Description of the steps for inference algorithm development.....85
Figure 32. Highest accuracy achieved of the all the models simulated with the first and small dataset.....93

Figure 33. Loss (a) and accuracy (b) results from training (blue) and validation (orange) using MaxVit model. The simulation was set with the following parameters: 200 epochs, learning rate of 0.001, weight decay variable and linear number of layers. 94
Figure 34. Loss (a) and accuracy (b) results during train (blue) and validation (orange) from ShuffleNet_v2_x0_5, with 200 epochs, a learning rate of 0.0001, a default weight decay, and a linear number of layers.....94
Figure 35. Loss (a) and accuracy (b) results obtained in train (blue) and validation (orange) from ShuffleNet_v2_x1_0 model. The simulation ran with 200 epochs, a learning rate of 0.001, default weight decay, and a linear number of layers. 95
Figure 36. Loss (a) and accuracy (b) results from train (blue) and validation (orange) ShuffleNet_v2_x1_5. The simulation was executed, achieving success with a configuration of 200 epochs, a learning rate set at 0.001, variable weight decay, and a linearly number of layers. 95
Figure 37. Loss (a) and accuracy (b) results from train (blue) and validation (orange) in ShuffleNet_v2_x2_0 model. The simulation was carried out with a configuration of 200 epochs, a learning rate of 0.001, variable weight decay, and a linear number of layers. 96
Figure 38. Loss (a) and accuracy (b) results, depicted for both training (blue) and validation (orange), for the EfficientNet_B0 model. The simulation was executed with 200 epochs, a learning rate of 0.001, default weight decay, and a linear number of layers. 96
Figure 39. Loss (a) and accuracy (b) results, for both training (blue) and validation (orange), in the EfficientNet_B1 model. The hyperparameters used were 200 epochs, a learning rate of 0.0001, a default weight decay, and a linear number of layers.....97
Figure 40. For the EfficientNet_B2 model, the loss (a) and accuracy (b) results are presented, with training data displayed in blue and validation data in orange. The simulation was conducted with a setup that included 200 epochs, a learning rate of 0.001, variable weight decay, and a sequential number of layers. 98
Figure 41. Regarding the EfficientNet_B3 model, the results for loss (a) and accuracy (b), for training data (blue) and validation (orange). The simulation was carried out with a setup comprising 200 epochs, a learning rate of 0.001, default weight decay, and a linear number of layers. 98
Figure 42. Results of the EfficientNet_B4 model, for loss (a) and accuracy (b) concerning both the training data (blue) and validation data (orange). The simulation was performed with a configuration that included 200 epochs, a learning rate of 0.001, variable weight decay, and a linear number of layers.....99
Figure 43. The outcomes of the EfficientNet_B5 model presented for both loss (a) and accuracy (b), encompassing both the training data (blue) and validation data (orange). The simulation was executed with a setup that consisted of 200 epochs, a learning rate of 0.0001, default weight decay, and a sequential number of layers. 100
Figure 44. The results for both loss (a) and accuracy (b) of the EfficientNet_B6 model are displayed, covering both the training data (blue) and validation data (orange). The simulation

was conducted with 200 epochs, a learning rate set at 0.001, default weight decay, and a linear number of layers. 100
Figure 45. ShuffleNet V2 architecture: (a) basic unit of ShuffleNetV2; (b) Basic unit for spatial down-sampling (Liu *et al.,* 2019). 102
Figure 46. The fundamental idea behind channel shuffle (Chen *et al.,* 2022)..... 102
Figure 47. MaxVit architecture (Tu *et al.,* 2022). 104
Figure 48. EfficientNet architecture (Tan & Le, 2020). 107
Figure 49. Loss (a) and accuracy (b) results, displayed for both training (blue) and validation (orange) for the MaxVit model. The simulation was conducted with the following parameters: 25 epochs, a learning rate of 0.001, variable weight decay, and a linear progression in the number of layers. 108
Figure 50. Loss (a) and accuracy (b) outcomes for training (blue) and validation (orange) are depicted for the ShuffleNet_v2_x0_5 model. This model was trained over 25 epochs, with a learning rate of 0.0001, default weight decay, and a linear progression in the number of layers. 108
Figure 51. Loss (a) and accuracy (b) results for training (blue) and validation (orange) from ShuffleNet_v2_x1_0. The simulation ran with 25 epochs, a learning rate of 0.0001, default weight decay, and a linear progression in the number of layers..... 109
Figure 52. Loss (a) and accuracy (b) results from train (blue) and validation (orange) ShuffleNet_v2_x1_5 model with 25 epochs, learning rate of 0.001, variable weight decay and a linear layer architecture. 109
Figure 53. Loss (a) and accuracy (b) results from ShuffleNet_v2_x2_0 during training (blue) and validation (orange), using 25 epochs, a learning rate of 0.001, flexible weight decay, and a linear layer architecture. 110
Figure 54. Results from EfficientNet_B0 training (blue) and validation (orange) phases utilizing 25 epochs, a learning rate of 0.001, default weight decay, and a linear layer architecture are shown in loss (a) and accuracy (b). 110
Figure 55. Loss (a) and accuracy (b) results of the EfficientNet_B1 training (blue) and validation (orange) phases using 25 epochs, a learning rate of 0.0001, default weight decay, and a linear layer architecture..... 111
Figure 56. The results for both loss (a) and accuracy (b) of the EfficientNet_B2 model are visualized, encompassing both the training data (blue) and validation data (orange). The simulation was carried out using 25 epochs, a learning rate of 0.001, variable weight decay, and a sequential layer structure..... 111
Figure 57. The loss (a) and accuracy (b) results for the EfficientNet_B3 model are displayed, covering both the training data (blue) and validation data (orange). The simulation was conducted with 25 epochs, a learning rate of 0.001, default weight decay, and a linear layer structure. 112

Figure 58. The EfficientNet_B4 model's loss (a) and accuracy (b) results are shown, covering both the training data (blue) and validation data (orange). With 25 iterations, a learning rate of 0.001, variable weight decay, and a linear layer structure, the simulation was run.....113
Figure 59. The EfficientNet_B5 model's accuracy (b) and loss (a) results are shown, taking into account both the training data (blue) and validation data (orange). A sequential layer structure was used, a learning rate of 0.0001, default weight decay, and 25 iterations of the simulation were run.....113
Figure 60. The accuracy (b) and loss (a) results for the EfficientNet_B6 model are presented, considering both the training data (blue) and validation data (orange). The simulation was conducted with 25 iterations, utilizing a learning rate of 0.001, incorporating default weight decay, and employing a linear layer structure. 114
Figure 61. Description of the division of the new dataset for train and validation of the models. 115
Figure 62. Model performance using the accuracy metric of all the best twelve models with the new expanded Dataset.....116
Figure 63. Loss (a) and accuracy (b) results from MaxVit model during training (blue) and validation (orange). The setup employed the new dataset, 200 epochs, learning rate of 0.001, variable weight decay and a liner architecture.....116
Figure 64. Results of the ShuffleNet_v2_x0_5 network using 200 epochs, the new dataset, a learning rate of 0.0001, a default weight decay, and a linear number of layers for loss (a) and accuracy (b) during train (blue) and validation (orange).....117
Figure 65. Loss (a) and accuracy (b) results for the ShuffleNet_v2_x1_0 model during training (in blue) and validation (in orange). The simulation was carried out with the new dataset, over 200 epochs with a learning rate of 0.001, default weight decay, and a linear layer configuration.....117
Figure 66. Results for the ShuffleNet_v2_x1_5 model's loss (a) and accuracy (b) during training (blue) and validation (orange). With the new dataset, over 200 iterations, a learning rate of 0.001, variable weight decay, and a linear layer configuration, the simulation was run. ... 118
Figure 67. Results of the ShuffleNet_v2_x2_0 model's train (blue) and validation (orange) phases for loss (a) and accuracy (b). A linear number of layers, 200 epochs, a learning rate of 0.001, a variable weight decay, and the new dataset were used to run the simulation. 118
Figure 68. The training and validation outcomes for the EfficientNet_B0 model, depicted in blue and orange respectively, for loss (a) and accuracy (b). This simulation was conducted with a linear layer configuration, 200 epochs, utilizing a learning rate of 0.001, incorporating variable weight decay, and employing the new dataset.....119
Figure 69. The results of both training and validation for the EfficientNet_B1 model, represented as blue and orange lines for loss (a) and accuracy (b) respectively, were obtained. This experiment used the new dataset, a linear layer configuration, 200 epochs, with a learning rate of 0.0001, and incorporated default weight decay.....119

Figure 70. The results of both the training and validation phases for the EfficientNet_B2 model, depicted as blue and orange lines representing loss (a) and accuracy (b) respectively, have been acquired. This experiment employed the new dataset, featured a sequential layer setup, 200 epochs, utilized a learning rate of 0.001, and included variable weight decay. 120
Figure 71. The EfficientNet_B3 model's training and validation results are represented by blue and orange lines, which stand for loss (a) and accuracy (b), respectively. This experiment made use of the new dataset, had 200 epochs, linear layers, a learning rate of 0.001, and default weight decay. 120
Figure 72. The training and validation results for the EfficientNet_B4 model are shown as blue and orange lines, which stand for loss (a) and accuracy (b), respectively. Utilizing the new dataset, 200 epochs, linear layers, a learning rate of 0.001, and variable weight decay were all used in this experiment. 121
Figure 73. The training and validation results for the EfficientNet_B5 model are presented graphically, with loss (a) shown in blue and accuracy (b) in orange. These results were obtained using the new dataset and involved a 200 epoch training. The model configuration included sequential layers, a learning rate of 0.0001, and a default weight decay. 122
Figure 74. The training and validation results for the EfficientNet_B6 model are visually represented in the following way: loss (a) is represented by the blue curve, while accuracy (b) is depicted by the orange curve. These results were derived from a training process involving 200 epochs, using a new dataset. The model architecture consisted of linear layers, employing a learning rate of 0.001, and employing default weight decay. 122
Figure 75. Web interface for image classification. 124

List of Tables

Table 1. Representation of a confusion matrix. 18
Table 2. Overview of research work in the "Disease Detection" field. 38
Table 3. Overview of research works in the "Weed Detection" field. 52
Table 4. Overview of research works in the "Weed Classification" field. 62
Table 5. Overview of research works in the "Fruit Detection" field. 69
Table 6. Classification models and model builders available in PyTorch that were used for the simulations. 78
Table 7. Hyperparameter combinations for model simulation for exploring the configuration. 88
Table 8. Loss and accuracy performance across PyTorch classification models. 90
Table 9. Performance evaluation of the top twelve models: 25 epochs simulation results. 107
Table 10. Performance evaluation of the top twelve models: 200 epochs training with the new dataset. 115
Table 11. Accuracy results following the testing phase, including the count of correctly classified images and the count of incorrectly classified images 124
Table 12.Confusion matrix of the MaxVit model. 125
Table 13. Confusion matrix of the ShuffleNet_v2_x0_5 model 126
Table 14. Confusion matrix of the ShuffleNet_v2_x1_0 model. 126
Table 15. Confusion matrix of the ShuffleNet_v2_x1_5 model. 127
Table 16. Confusion matrix of the ShuffleNet_v2_x2_0 model 127
Table 17. Confusion matrix of the EfficientNet_B0 model. 128
Table 18. Confusion matrix of the EfficientNet_B1 model. 128
Table 19. Confusion matrix of the EfficientNet_B2 model. 129
Table 20. Confusion matrix of the EfficientNet_B3 model. 129
Table 21. Confusion matrix of the EfficientNet_B4 model. 130
Table 22. Confusion matrix of the EfficientNet_B5 model. 130
Table 23. Confusion matrix of the EfficientNet_B6 model. 131

Nomenclature

General symbols:

$\widehat{m}_n$	Bias-corrected estimates of the first moment of the gradients;
$\widehat{v}_n$	Bias-corrected estimates of the second moment of the gradients;
θ_n	Parameters being optimized at interation n;
F	Resulting Feature Map;
f	True probability distribution over classes;
I	Height;
I	Input Image;
J	Width;
k	Cohen's Kappa Coefficient;
K	Filter;
L	Cross Entropy Loss;
P	Pooled Feature Map;
p	Size of the pooling window;
P_a	Probability of Agreement;
$P_{correct}$	Probability of True Values;
pH	Hydrogen Potential;
$P_{incorrect}$	Probability of Incorrect Values;
P_r	Probability of Random Agreement;
t	Predicted probability distribution over classes;
Y	Classes Labels;
α	Learning rate;
β	Constant Constant Coefficients from EfficientNet;
γ	Constant Coefficients from EfficientNet;
φ	Compound Coefficient from EfficientNet;
ε	Constant;
θ	Constant Coefficients from EfficientNet.

Low indices:

spp.	Species;

Acronyms:

2-D-CNN	Two dimensional convolutional neural network;
3-D-CNN	Three dimensional convolutional neural network;
ACC	Accuracy;
ADAM	Adaptive Moment Estimation;
AI	Artificial Intelligence;
ANN	Artificial neural network;
ATV	All-Terrain Vehicles;
AU-ROC	Area Under the ROC Curve;
AutoML	Automated machine learning;
CED	Canny Edge Detector;
CNN	Convolution Neural Networks;
CO2	Carbon dioxide;
CPU	Central Process Unit;
CUDA	Compute Unified Device Architecture;
CWFID	Crop Weed Field Image Dataset;
DL	Deep Learning;
DM	Depth multiplier hyperparameters;
DNN	Deep Neural Network;
DRGV	Douro red grape variety;
EFSA	European Food Safety Authority;
EU MACP	European Union Coordinated Control Programme;
ExG	Excess Greenness Indice;
ExGR	Excess green minus excess red;
ExR	Excess Red Indice;
FAO	Food and Agriculture Organization of the United Nations;
FCN	Fully convolutional network;
FLOP	Floating-point;
FN	False Negatives;
FoA	Frequency of appearance;
FP	False Positives;
FPN	Feature Pyramid Network;
FPR	False positive rate;
FPS	Frames per second;
FR	Field Robots;
GA	Genetic algorithms;

GBIF	Global Biodiversity Information Facility;
GIS	Geographic Information Systems;
GLCM	Gray-level co-occurrence matrix;
GMP	Gray-scale Morphology Processing;
GNDVI	Green Normalised Difference Vegetation Indice;
GPS	Global Positioning Systems;
GT	Ground truth;
GWO	Grey wolf optimizer;
HE	Histogram Equalization;
HOG	Histograms of oriented gradients;
ICF	Independent components filters;
ICT	Information and Communication Technology;
IoT	Internet of Things;
IoU	Intersection Over Union;
IR	Infrared;
JULE	Joint unsupervised learning;
KNN	K-nearest neighbours algorithm;
LBP	Local binary pattern;
LiDAR	Light Detection and Range;
LMA	Levenberg–Marquardt algorithm;
LPE	Leaf Patch Extraction;
LRN	Local Response Normalisation;
LSA	Leaf segmentation algorithm;
MAV	Micro Aerial Vehicle;
Max-SA	Multi-axis self-attention;
MaxVit	Multi-Axis Vision Transformer;
MBConv	Mobile Inverted Bottleneck Convolution;
mIoU	Mean Intersection Over Union;
ML	Machine Learning;
MLP	Multilayer Perceptron;
MSE	Mean Squared Error;
NDI	Normalized Difference Indice;
NDRE	Normalised Difference Red Edge;
NDVI	Normalised Difference Vegetation Indice;
NIR	Near-Infrared;
O2	Oxygen;

OBIA Object-based image analysis;
OS Output stride hyperparameters;
P Precision;
PA Precision Agriculture;
PCA Principal Component Analysis;
R Recall;
R-2-D-CNN Region-based 2-dimensional convolutional neural networks;
R-3-D-CNN Region-based 3-dimensional convolutional neural networks;
R-CNN Region-based Convolutional Neural Network;
RE Red Edge;
ReLU Rectified Linear Unit;
ResNet Residual Network;
ResNeXt Residual Neural Network with Next;
RF Random forest;
RGB Red- Green- Blue (Visible light);
RMSprop The Root Mean Square Propagation;
RNN Recurrent Neural Network;
ROI Region of interest;
RTFD Real-Time Fruit Detection model;
RVI Ratio Vegetation Indice;
SBWD Shape-based weed detection;
SCA Sine cosine algorithm;
SE Squeeze-and-excitation Networks;
SfM Structure from-Motion;
SGD Stochastic Gradient Descent;
SLIC Simple linear iterative clustering;
SSD Single-shot detector;
SSWM Site-Specific Weed Management;
SVM Support vector machine;
SWIR Short-Wave Infrared Radiation;
TMG Tensorflow Model Garden;
TN True Negatives;
TNR True Negative Rate;
TP True Positives;
TPR True Positive Rate;
TPU Tensor processing unit;

UAV	Unmanned Aerial Vehicle;
UN	United Nations;
USA	United States of America;
VGG	Visual Group Geometry;
VI	Vegetation Indice;
ViT	Vision Transformer;
WGISD	Embrapa Wine Grape Instance Segmentation Dataset;
YOLO	You only live once.

1. Introduction

The whole world is changing drastically. The dynamics of our society are changing at an unprecedented rate because of the rise in global populations, the growth of urban areas, and changes in fertility rates. It is anticipated that most of the world's population will reside in metropolitan regions.

However, the increase in urbanization also carries with it a growing problem, a rise in food demand. Failure to achieve this challenge will have far-reaching effects on the world.

This study embarks on a journey to confront this multifaceted problem. We explore the realm of advanced neural networks to provide innovative solutions for food production, with a special emphasis on their application in agriculture. Within the agricultural sector, one of the most worrying challenges revolves around the management of unwanted vegetation, presenting a focal issue that this work aims to address comprehensively.

The key goal of this dissertation is to create a decision-making algorithm using CNN models capable of accurately classifying unessential vineyard plants in agricultural fields.

This research significantly advances the AI in agriculture and vineyard vegetation management, representing a noteworthy technological and efficient achievement in agriculture.

1.1. Background

The world's population started to increase during the Industrial Revolution, mainly as a result of the improvements in agriculture and medicine. Currently available estimations show that population growth is fast, averaging 80 million people each year, at a rate of 1.13%. According to the United Nations (UN), 8.1 billion people will populate the world by 2025 (Tripathi *et al.,* 2019; Nations, n.d.).

In addition, these large growths occur mainly in developing countries due to the increase in longevity, migration, and fertility rates. In contrast, developed countries present slow growth. Furthermore, urbanisation has been accelerating, and by 2050 it is expected that more than 70% of the world's population will reside in cities (Nations, n.d.).

Portugal’s population increased in urban areas while decreasing in the rural regions. This happens due to the search for better job offers and, consequently, quality of life. Therefore, the rural population has been decreasing over the years, reaching 33.2% of the Portuguese population, and 66.8% in the urban zones, in 2021 (FAOSTAT, 2022).

Considering this, the demand for food must be intensified. Food production should increase by 70% (Tripathi *et al.,* 2019). However, agriculture faces several significant obstacles, such as climate change, rising groundwater, vegetation, water loss, pests, diseases, pollution, deteriorating soil, high pump irrigation costs, the shift to a biobased economy from a fuel-based one, and finally, the declining availability of freshwater as demand increases (Wang *et al.,* 2019).

According to Wang et al. (2019), flora with infesting potential, known as weeds, directly compete with vegetables and fruits for water, growth space, sunlight, and nutrients, making them vulnerable to insects and illnesses and resulting in an average loss of 34% of productivity.

Weed control is one of the main variables in crop output, and manual weeding is the most traditional method in agriculture. However, it is high-labour cost, inefficient and time-consuming. Besides that, the population is migrating to urban zones, and consequently, labour is decreasing in the rural regions. Although mechanical weeding methods are much more labour- and time-efficient than manual weeding, they might potentially harm crops (Wang *et al.,* 2019).

On the other hand, the use of chemical strategies may affect plants, soil, animals, and the ecosystem's well-being. Pesticides are all substances directed to prevent, destroy, repel, or mitigate any pest, they were initially used during World War II, primarily to defoliate forests (Hasan *et al.,* 2021).

They are classified according to the pests they attack, in this case, they are called herbicides. Herbicides are a chemical used to control weeds. Therefore, these products spray evenly the whole field, and the problems that result from the use of herbicides are environmental and health. Weed management is an important phase in sustainable agriculture. However, treating them has become a challenge, as well as herbicides (Britannica, 2022).

The use of these chemicals has been increasing over the years with some insignificant decreases, and in 2020 1.397,465.09 tonnes of herbicides were consumed (FAOSTAT, 2022).

According to information from the Food and Agriculture Organization of the United Nations (FAO), from 1990 to 2020 the average herbicide consumption by continent is the following: the first place belongs to the United States of America (USA) with 56,6%, followed by Asia with 21.7%, Europe with 16,8%, Oceania with 2,8%, and in the end, Africa with 2%. The top 10 nations in the world for agricultural herbicide usage in 2020 are as follows: the USA came in first with a consumption of about 256000 tonnes. Brazil followed them in the consumption of herbicides, using around 234000 tonnes. After that, Argentina, China, Canada, Australia, Russia, Colombia, France, and Malaysia with, respectively, 230.12, 109.01, 59.43, 43.79, 41.21, 29.26, 29.18, and 26150 tonnes (FAOSTAT, 2022).

A vineyard management system must include weed control. Since they are often considerably closer to the ground, plants on the vineyard floor have an impact on the prevalence of other pests, namely, nematodes, animals, mites, insects, and diseases. Before any new vines are planted, a vegetation control programme should be implemented. Before vines are established, it is simpler to handle the harder-to-control plants (especially perennials). By competing with vines for nutrients, water, and sunlight, unwanted plants restrict vine development and productivity (UC IPM, 2015).

In locations where vine root development is constrained by shallow or compacted soil, weed competition is most intense in the first few years following planting. Nearby the trunk, weeds compete with vine growth directly and offer animals a decent home. These underground creatures damage or destroy young vines by feeding on the roots. Furthermore, dried-out weeds might provide a significant fire risk (UC IPM, 2015).

Herbicides often eliminate the kind of weeds specified on their label when applied correctly. However, this does not happen. Herbicides are used in excessive amounts and specific herbicides are not used for different types of weeds (UC IPM, 2015).

Knowing this, both the world areas harvested grapes and Portugal area harvested grapes are decreasing compared to the production of grapes. According to FAOSTAT, in 2021 Portugal's area harvested grapes was 175,670 tonnes, and the production was 853,380 tonnes. Globally, the world area harvested was 6,950.930 tonnes, and the production was 78,034,332 tonnes (FAOSTAT, 2022).

The best solution to these issues is PA. This technique is a management method that helps to increase agricultural yield while minimising water and fertiliser losses and having a negative environmental impact. It accomplishes this by utilising a variety of cutting-edge information, communication, and analytical methods (Sishodia *et al.,* 2020).

Currently, improvements in information and communication technology (ICT) can help in agriculture. Emerging technologies including the Internet of Things (IoT), Global positioning systems (GPS), remote sensing, Geographic information systems (GIS), big data analysis, and AI are auspicious tools being used to improve agricultural operations and inputs to increase productivity (Sishodia *et al.,* 2020).

Site-specific weed Management (SSWM) typically uses remote sensing to map weed patches in agricultural regions. Based on their distinctive spectral fingerprints, cultivated plants and weeds can be distinguished or classified. In recent years, the very accurate and effective image classification offered by ML algorithms has substantially aided weed mapping (Sishodia *et al.,* 2020).

In conclusion, these technologies are expected to change agriculture since they allow decisions to be made in days as opposed to weeks. Additionally, they promise a large decrease in costs and an increase in output (Tsouros *et al.,* 2019).

1.2. Study Problem and Relevance

Nowadays, it is important to have a special concern about climate change, water resources, and chemical consumption.

In recent years, the main selection factor has been herbicides. The high selection pressure exerted by these products, combined with their improper use, has helped in the selection of biotypes resistant, becoming the main current problem related to weed management. Resistance is the capacity acquired by a group of individuals within a population (biotype) to survive and reproduce after exposure to the herbicide that controls other individuals of the same species (spp.) (Carvalho, 2013).

One of the most crucial elements for crop growth is water. As a result, some vegetation competes with vegetables and fruit for water. Amongst others, they can steal the water from the field consequently causing crops to stop growing. Furthermore, variations in the average world temperature are closely related to climate change. This decreases the amount of water that is

available, therefore growers must adopt a water-saving strategy. To accomplish this objective, it is crucial to maintain the fields free of infestation flora (Nations, n.d.).

The use of herbicides is one method of vegetation control. On the other hand, even though these chemicals effectively get rid of weeds, they might contaminate food or water sources. To address these challenges, several European nations have begun to restrict the use of pesticides in agriculture (Tamirat Wato, 2020).

According to the European Food Safety Authority (EFSA), in 2020 the European Union Coordinated Control Programme (EU MACP) analysed 12,077 samples of random products, including carrots, cauliflowers, dried beans, kiwi fruits, onions, oranges, pears, potatoes, bovine liver, poultry fat, brown rice, and rye grain. According to the analyses, it was concluded that 98.2% were within legal limits. This means, that in 8278 samples, around 68.5% were found to have no quantifiable levels of residues. However, 4% of these 3590 products, restrained concentrations of one or more residues that were lower or equal to the permitted levels. Finally, the 209 samples remaining, about 1.7%, contained residues exceeding the legal maximum (EFSA, 2022).

Due to these issues, vegetation control must be environmentally friendly, and precise agriculture technology must be applied. Plant identification and computation processes must be employed to decrease the adverse impacts of herbicides on the environment and to make the best use of them (Espejo-Garcia et al. 2020).

To counteract these challenges, new technologies are being developed. Including, the application of robots and/or machines (Alibabaei *et al.,* 2022).

In this case, an SSWM's main objective is to spray weed spots exclusively and/or modify herbicide applications in response to species composition and/or vegetation density (Wang *et al.,* 2019). Additionally, after determining weed spots, other techniques that do not use chemical compounds can be used, such as electrical discharge, high temperature, and UV light, among others.

The advance of sub-areas in AI, for instance, speech generation and recognition, natural language processing, multi-agent systems, vision, image and video generation, planning, robotics integration of vision and motor control, and decision-making was made possible by the field's rapid advancements (Yang *et al.,* 2018).

Artificial intelligence has received a lot of attention as decision-making algorithms have improved. Machine Learning, a subfield of AI, uses computational techniques to turn real-world data into useable models and decision support. This means that ML gives machines the ability to imitate intelligent human behaviour, including the ability to learn from past mistakes and advance. Finally, *Deep Learning* (DL) is a branch of machine learning. DL algorithms are relatively more complicated than those of conventional ML models. The layers in a network that lie between the input(s) and output(s) are known as hidden layers. A deep network contains more hidden layers than a shallow network, which only has one. Deep neural networks are capable of learning the characteristics of data and addressing additional challenges by using multiple hidden layers (Alibabaei *et al.,* 2022).

The primary topic of image processing study nowadays consists of DL, which has found successful applications across several industries. DL models are utilized in agriculture to identify weeds and diagnose plant illnesses (Espejo-Garcia *et al.,* 2020).

In the agricultural sector, CNN is now the most used deep learning method. This subset of deep neural networks is frequently used to assess visual images (Islam *et al.,* 2021).

Automatic classification of plants can be crucial to weed control and help improve the sustainability of agriculture, reducing the consumption of herbicides and/or choosing wisely the type and quantity of them or leading the way to use other flora removal techniques that don't make use of chemicals (Hasan *et al.,* 2021).

1.3. Objectives and contributions

The worldwide population has been growing over the years, and, in consequence, the production of the agricultural sector must keep pace with this growth. However, an elevated percentage of the food is lost due to weeds. So, knowing this, the main objective of this work is to develop and apply a decision-making algorithm using AI that would be able to correctly classify dispensable vineyard flora in the fields.

The study aims to respond to some subjects like the best type of images used in tasks of classification. Also, the best performance metrics and which ones and how should be used, as well as which types of models are better for this purpose. And, if all the vegetation is considered weeds or not.

The primary objectives of this dissertation are to develop CNN models for flora classification using PyTorch and explore the optimal architecture, hyperparameters, and training strategies to achieve high accuracy, and to understand their strengths and weaknesses: (1) Conduct performance evaluations of the developed CNN models, measuring accuracy, and loss. (2) Conduct a comprehensive comparative analysis of the different CNN models developed. And (3) select the best-performing CNN architecture based on the evaluation metrics and hyperparameters. (4) Create a user-friendly application that applies the best CNN model developed. And (5) ensure that the application can take input data, process it using the chosen model, and provide meaningful results or predictions to the user.

This dissertation significantly advances the fields of AI in agriculture and vineyard weed management. The creation of CNN models specifically designed for classifying vineyard weeds is a big technological accomplishment. These models could improve the effectiveness and precision of weed management techniques.

The research in this dissertation lessens the environmental impact of pesticide use by enabling more accurate and focused weed management, which supports the larger objective of sustainable agriculture. Additionally, by automating weed classification tasks, less manual labour is needed for field scouting and monitoring. Furthermore, cost savings and environmental advantages result from this.

This dissertation sets out an objective to revolutionize weed categorization in vineyards by using the potential of convolutional neural networks.

1.4. Document organization

The research for this dissertation was developed in two stages. Firstly, a theoretical investigation to acquire knowledge about the subject being studied. Lastly, consists in the development of a practical component to assure that the knowledge that was first studied may be applied.

This dissertation is divided into six main chapters. These five sections support and develop the theme in depth.

The first chapter, “Introduction”, performs the background of the theme under study and presents the main objectives.

The second chapter, “State of the Art”, contains all the literature revision necessary to develop a decision-making algorithm in real-time. Firstly, it was made a brief description of plant agrobiodiversity, due to the importance of the characteristics of the flora to do its classification. Furthermore, the types of data acquisition that could be employed in different practices are presented. Also, an introductory material about vegetation indices, that could be applied in different approaches is described, as well as a description of the models that can be used for this work. Consequently, the most common performance metrics to evaluate the models are described and compared. Following, the initial concepts related to the CNN, as well as the PyTorch models, loss and optimizer functions are presented. In particular, the adaptive moment estimation Adam optimizer and the Cross-entropy loss function are described in more detail. In the last, related works are presented, thirty-three different types of research, segregated into disease detection, weed detection, weed classification, and fruit detection were studied.

In the third chapter, “Materials and Methods”, for the progress of this work, data acquisition and preparation, along with the architectures simulated, algorithm development and its simulation are explained.

In the fourth chapter, “Experimental Results”, it is realized the validation of the algorithms proposed for vineyard flora classification is divided into case studies. Also, the results of all experiments are presented and analysed. The first experiment considered all the models simulated. The second one analysed the performance of the twelve best models across epochs and established baselines. The third experiment explored the performance with a new dataset. And, finally, the fourth experiment, tested the algorithms in a web application.

The fifth, and the last main chapter,” Conclusion”, presented the main conclusions of the realized work.

2. State of the Art

This work aims to study an economical and sustainable alternative to current weed management mechanisms, that can be applied in a decision support system for weed classification, which can be included in an automatic control device.

Weed management is not an easy task. It has high labour costs, is inefficient and is time-consuming. As a result, image analysis is an essential area of study for the agricultural industry (Wang *et al.,* 2019; Kamilaris & Prenafeta-Boldú, 2018).

Remote sensing is frequently recognised as one of the furthermost significant technologies for PA and Smart Farming. In the last 35 years, it has provided efficient solutions by being widely utilised to monitor harvested fields. Remote sensing can track a variety of crop and vegetation factors employing images acquired at different wavelengths (Tsouros *et al.,* 2019).

However, when this is considered, several significant questions emerge. This work's objective is to classify weeds using artificial intelligence, but 'What are weeds?', or 'How can we realize which models' performance is better?'. As well as 'What type of images can be useful for the studies?', and 'Which type of models and performance metrics other authors used?'.

In this chapter, answers to these questions will be given. The first subchapter introduces different notions used in this work. Agrobiodiversity will be studied, followed by vegetation indices, data acquisition, and the performance metrics that will be employed to determine the effectiveness or quality of the model.

A revision of literature related to this work is presented in chronological order, from 2015 to 2022, where numerous methods to classify and detect weeds will be analysed.

In the end, after analysing this content, it will be possible to conclude which is the best model to adopt, the type of image, and the metric, to be applied later in future practical work.

2.1. Plant Agrobiodiversity

Ecologically speaking, the plants that are currently regarded as weeds have developed from the Earth's beginnings and before humans. As humanity transitioned from nomadic lifestyles to settled agricultural practices, the cultivation of crops and domestication of animals gradually emerged. As a result, with the development of agriculture, mankind started to cultivate the land for subsistence. This process was followed by the selection of wild species that could be consumed and cultivated, originating with the domestication of several plant and animal species, like maize (*Zea mays*), (MacLaren *et al.,* 2020).

In agricultural systems, biodiversity contributes to various ecosystem services. These services encompass soil formation, nutrient cycling, climate regulation, hydrological process management, and control of the proliferation of unwanted organisms. This useful component of biodiversity has particular importance in the context of multifunctional and sustainable agriculture (Gaião *et al.,* 2019).

However, when there is a lack of control of plant species in habitats for human use, such as agriculture, these plants can have a weed potential. An infestation occurs when these species become dominant or begin to interfere with human activity carried out in these places by competing for water and nutrients with productive species (e.g., vines or others). Usually, this happens when the ecosystem is already in environmental imbalances. Although people continue to use some terms, like invasive plants, spontaneous plants, and weeds, there is yet no notion established for these situations. In general, some authors believe that weeds can have beneficial effects on humans and/or the environment. In this instance, they believe it should not be referred to as a weed (Carvalho, 2013).

Several factors can contribute to such occurrences, including the absence of natural consumers for these potentially weedy species, selection pressure due to herbicide application (leading to herbicide-resistant species), imbalanced nutrient levels in the soil, or the scarcity of beneficial plant species that would naturally compete with them. In monocultures, this risk is heightened, as the ecosystem in general is unbalanced (MacLaren *et al.,* 2020).

Plants better adapted to soil cultivation were selected, which was the first way of natural selection of species from the agricultural environment. Additionally, due to the high genetic variability, plants dispersed their diaspores (seeds and propagules), and, as a result, became specialised in the colonisation of agroecosystems with high soil disturbance (Fernández-Aparicio *et al.,* 2020).

Stress is a result of variables connected to the scarcity of natural resources such as light, water, and nutrients, and it prevents the growth and development of plants. Disturbance is related to factors that alter the soil's plant cover including floods, fires, the cutting down of trees, weeding, and soil disturbance (Carvalho, 2013).

The properties of aggressiveness of plant species, as well as the environmental conditions of a certain area, including abiotic factors (climate, water availability, among others), influence the poll of species able to colonise a certain agroecosystem, particularly in locations with considerable disturbance. Aggressiveness may be interpreted as a plant's capacity to colonise and persist in a certain environment (Fernández-Aparicio *et al.,* 2020).

Competition is an interaction between living beings in which there is an injury to both individuals involved. Normally, the competition is described for the plant-plant interaction in which there is a limitation of some environmental resources required for growth and plant development. This means that the first plant is directly preventing the second one to grow and develop, characterizing direct interference. Competition can also arise from an interaction between plants and other living beings that use the same resources as vegetables (Carvalho, 2013).

Mainly in the plant-plant interaction, the main resources subject to competition are water, nutrients, light, and space; there may also be a limitation of gases (Carbon dioxide (CO_2) and oxygen (O_2), mainly). Therefore, competition will only occur when at least one resource is constrained in the middle. However, it is considered the main cause of the reduction of productivity in crops (Carvalho, 2013).

Furthermore, environmental variables and characteristics of each plant species have a role in the introduction or invasion of new places (adaptive capacity). As a result, to successfully colonise new areas, a plant must overcome environmental biological and adaptive limitations and be able to adapt to its new environment (Carvalho, 2013).

It is important to enhance that the term infestation refers to a process that allows for quick colonisation by allowing for rapid reproduction, intense production and facilitated dissemination of reproductive structures, storage of viable diaspores in the soil, and germination and establishment of plants in the area (Carvalho, 2013).

Furthermore, to control plants that have the potential of infestation, it is important to understand their habitat, type of flora, and origin, to decide the different management options as well as how and when to employ them, according to the types of management already being practised at a certain land. Thus, it becomes relevant to identify the pool of species present in each area.

Native or indigenous plants are those that occurr naturally in a particular region, ecosystem, or habitat without human introduction. Endemic plants are plants that are found in only one location on the planet, and nowhere else. On the other hand, invasive or allochthonous plants are those that grow in places other than their original habitat and settle and reproduce in an uncontrolled way (Fernández-Aparicio *et al.,* 2020).

Different categories can be used to group plants. One of the most used categorizations is taxonomic. The taxonomic classification is based on the grouping of plants with similar characteristics, which can easily be related to ecosystem functioning and services. Correct identification of the species presents at a given location provides specific knowledge of their functions or infesting potential, allowing one to choose and adopt different management strategies (Carvalho, 2013).

This classification separates plants into two large groups: the broad leaves, plants with a wide leaf limbo and peninérvea nervation; and the narrow leaves, plants with a narrow leaf limbo and parallelalelinérvea nervation. Figure 1 represents an example of each type of herb about their plant group, (a) narrow leaves, like *Lolium rigidum*, and (b) broad leaves, such as E*rodium botrys*.

a) Narrow leaves b) Broad leaves

Figure 1. Example of the two types of leaves according to the plant group. (a) Broad leaves, and (b) Narrow leaves.

According to the habitat, plants can be terrestrial, aquatic, or able to develop in both environments (indifferent), (Carvalho, 2013). Figure 2 represents an example of each type of weed about its habitat, (a) terrestrial, (b) aquatic, and (c) indifferent. The first example represents a vineyard, the second one a *Lemna minor*, and the last, a *Juncus capitatus*.

According to the life cycle, plants can be classified as monocarpic or polycarpic, and based on their habit of growth as herbaceous, shrubs and subshrubs, arboreal, vines, parasites, epiphytes, and hemiepiphytes. Each type of plant has its characteristics and importance as a crop or weed. Monocarpic plants complete their life cycle within one or two years, while polycarpic plants take more than two years (Carvalho, 2013).

a) Terrestrial b) Aquatic c) Indifferent

Figure 2. Example of the two types of plants according to the habitat. (a) Terrestrial; (b) Aquatic; (c) Indifferent. (Diego *et al.,* 2021), (UTAD, 2023), (Flora-On | Flora de Portugal, 2022).

The management of parasitic weeds is difficult because there are few sources of crop resistance, and it is difficult to use selective enough controlling strategies to kill the weeds without hurting the crop to which they are physically and biochemically connected (Gaião *et al.,* 2019).

The mechanical control method is based on the use of some instruments that cut plants, like hands, hoes, or electric brush cutters (Gaião *et al.,* 2019).

The physical control method is based on the use of some practice that exerts physical influence on weeds. Namely, flooding, which consists of the use of water for control. Also, solarization in which the soil is covered with polyethylene film causes an increase in temperature (Gaião *et al.,* 2019).

Biological control is based on the action of natural enemies (fungi, insects, bacteria, viruses, birds, fish) able to reduce weed populations and thus their ability to compete with crops (Gaião *et al.,* 2019).

The chemical control method is based on the use of chemicals aimed at killing vegetation. In the main crops, cultivated extensively, herbicides are the main control method (Gaião *et al.,* 2019).

Disadvantages and Advantages

The presence of some species in cultivated areas might be dangerous for plants, namely, fruits and vegetables. The existence of this vegetation results in productivity reduction because losses

vary by species and can make harvesting problematic. In this way, the potential worth of the land may be decreased depending on the species and population density in the region. In agricultural areas, species that are difficult to control, such as *Cyperus* spp., and *Cynodon dactylon* can reduce the land value (Carvalho, 2013).

Furthermore, they can cause losses in agricultural product quality. The presence of residues, at the time of the harvest, in addition to impurities, can increase the use of water while processing the products, which is negative for several factors: excessive water use, cost raises, and possibly favouring deterioration. In addition, there is the issue of contamination of seed lots. For example, the presence of *Oryza sativa* in seed lots of rice (Carvalho, 2013).

Plants are potential hosts for pests, diseases, nematodes, mites, bacteria, and viruses, therefore being an inoculum source of these organisms in cultures of commercial interest. For example, *Sida* spp., and *Bemisia Tabaci*, are hosts of aphids (Fernández-Aparicio *et al.,* 2020).

In addition, a crop with a high presence of infestation flora is more difficult to manage than others with few. Additionally, the expense of vegetation management raises the cost of production in the region. Its presence at harvest time can bring operational disruptions, delaying the harvesting process and, consequently, increasing losses and production costs (Carvalho, 2013).

Invasive aquatic plants can damage irrigation canals, hydroelectric dams, and fish-producing lakes. Due to the high evapotranspiration, there is an increased loss of water and oxygen consumption in the presence of *Eichornia Crassipes*, which can result in fish mortality, and a decrease in the amount of water in reservoirs, among others (Fernández-Aparicio *et al.,* 2020).

Plants with spikes, for example, *Solanum viarum*, or with diaspores equipped with spiky structures, like *Cenchrus echinatus*, as well as poisonous or irritating plants such as *Conium maculatum* and *Ricinus communis*, can be a nuisance in agricultural operations such as manual harvesting and other management tasks (Fernández-Aparicio *et al.,* 2020).

On the other hand, this flora plays an important role in maintaining biodiversity, as vegetation cover produces positive benefits to the soil, namely, improves soil structure and stability, preserves humidity, prevents water loss by evaporation, and decreases the superficial draining (reducing erosion), (Guzmán *et al.,* 2019).

As they continue to grow and die, the plants also assist the soil by adding more organic matter to it. That is one justification for allowing, preserving, and enhancing them in the surrounding areas of agricultural land. Furthermore, increases soil fertility and enrichment (Guzmán *et al.,* 2019).

Some species also act as a diagnostic tool by offering an abundance of data about the nutrient balance of our soil through their existence and growth patterns, like Stellaria media. Also, they are bioindicators, some flowers indicate the type of pH (hydrogen potential) of the soil, like *Rumex Acetosella* (Fernández-Aparicio *et al.,* 2020).

Furthermore, some flora may give information about the soil moisture, some plants grow up in wet areas, like *Rumex* spp., while others grow up in dry spaces, namely, A*chillea millefolium* (Guzmán *et al.,* 2019).

Also, cover crops provide a habitat for beneficial insects or arthropods, including natural enemies of grapevine pests, and improve the landscape (Guzmán *et al.,* 2019).

Another approach to benefit from the existence of weeds is through their use in medicine, like *Stellaria media* and *Trifolium pratense*. Some weeds are also consumed by humans and/or animals as food, such as *Amaranthus* spp., and *Trifolium* spp. (Guzmán *et al.,* 2019).

2.2. Data acquisition

For DL systems whose objective is weed detection and classification, a large amount of labelled data is essential. Therefore, collecting sufficient data for future investigation is crucial in the initial phase.

It is possible to define image acquisition as the process of getting an image from different sources. This may be accomplished via hardware devices like cameras. The camera sensors' function is to take high-resolution images in both time and space, which may be used to monitor a range of vegetation-related properties (Islam et al. 2021).

Numerous categories of sensors are used on a range of platforms to collect data in diverse modalities. These photos can be captured using unmanned aerial vehicles (UAV), field robots (FR), all-terrain vehicles (ATV), micro aerial vehicles (MAV), cameras, satellite photographs, and open datasets (Hasan *et al.,* 2021).

Datasets are available for free on the internet in addition to employing sensors, such as The Salinas, and Indiana Pines (Yang et al. 2018); the Broadleaf Dataset, and DeepWeed (Ferreira *et al.,* 2019).

Each kind of sensor can keep an eye on a variety of characteristics of the plant, such as its colour, texture, and geometric form. Some sensors can even measure radiation wavelengths. These sensors' data may be used to measure vital agricultural traits during the many growth phases, such as plant biomass, soil moisture, and vegetation health (Tsouros *et al.,* 2019).

Nowadays, for agriculture, the sensors entrenched in cameras for data acquisition correspond to one of the following four categories: multispectral, visible light (RGB- Red, Green, Blue), thermal, and hyperspectral.

Figure 3 shows the electromagnetic spectrum, which consists in the distribution of the electromagnetic waves, visible and not visible, according to the frequency and wavelength characteristic of each radiation (Tsouros *et al.,* 2019).

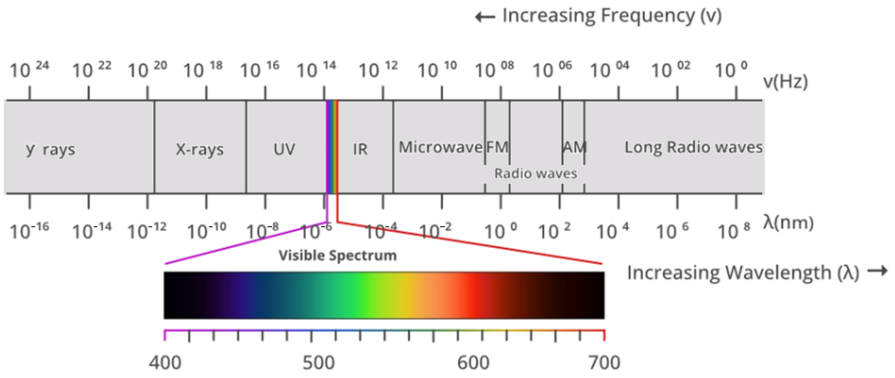


Figure 3. Electromagnetic spectrum. Adapted from (Ferlic, 2019).

The most typical sensors used for PA tasks are those that detect visible light. They can take high-resolution pictures and are inexpensive and simple to use. Additionally, the information acquired needs to be processed simply. It collects visible light with wavelengths in a range of [400~700 nm (nanometers)] converts that to an electrical signal, and then organizes that information to extract images (Tsouros *et al.,* 2019). The wavelength of the visible light is possible to see in Figure 3.

RGB sensors are routinely adjusted to collect data on radiation in additional bands, most typically the Red Edge (RE) or Infrared (IR) band. One of the original optical filters is typically replaced with one that allows the perception of the near-infrared (NIR) channel to create hybrid sensors, such as NIR-RGB. The wavelength of the IR is presented in Figure 3, at the right of visible light. Its wavelength has a range of approximately [750- 2500 nm], (Tsouros *et al.,* 2019).

The visible spectrum as well as wavelengths outside of it, such as short-wave infrared radiation (SWIR), near-infrared radiation, and others, are collected by multispectral sensors. The infrared division has five partitions, the near, the short, the mid, the long, and the very long wave. The wavelength of the SWIR is shown in Figure 4, right next to NIR (Charlton *et al.,* 2017).

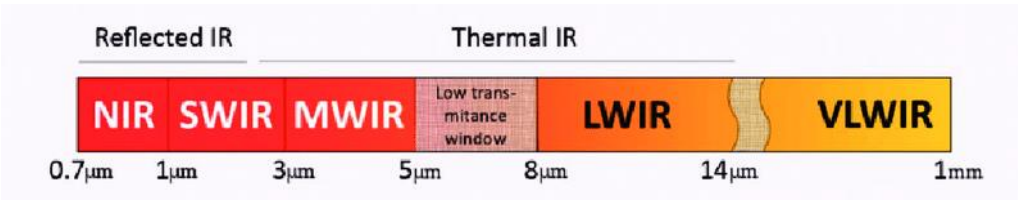


Figure 4. Spectrum of electromagnetic waves in the infrared (Charlton *et al.,* 2017).

Multispectral sensors involve the acquisition of near-infrared, visible, and short-wave infrared images, wavelengths with increments of 20 nm and a range of [400 - 1100 nm].

Hyperspectral sensors evaluate a broad range of light rather than merely giving each pixel a main colour, wavelengths having a [400 - 1100 nm] range, divided into 1 nm increments.

Data on the spectrum absorption and reflection of the vegetation on various bands may be obtained using multispectral and hyperspectral imaging techniques. Numerous vegetation indices may be calculated using spectral information, and various crop or weed characteristics can be tracked using them (Tsouros *et al.,* 2019).

Thermal infrared sensors measure the temperature of the materials and create images of them based on this data instead of their physical attributes. Infrared sensors and optical lenses are used by thermal cameras to gather infrared energy. To visualize temperature, thermal cameras typically transform the radiation they focus on and detect into a grayscale image. Many sensors used for thermal imaging may also provide coloured pictures. These instances usually depict warmer things as yellow and colder ones as blue. However, for particularly specific applications, such as irrigation control, this sort of sensor is employed. As a result, they are therefore rarely utilised in PA (Tsouros *et al.,* 2019).

Other types of sensors, such as light detection and range (LiDAR) sensors, are also an option in addition to those already listed. When used in conjunction with other sensors, these objects, also known as distance sensors, could be used by vehicles such as robots to navigate the field. LiDAR sensors analyse the phase difference between a laser beam that is emitted and one that is reflected. Plant detection with LiDAR works because chlorophyll reflects in NIR wavelength, and the green colours of vegetation lead to a high reflection value (Machleb *et al.,* 2020).

2.3. Vegetation indices

Applications for remote sensing typically handle vegetation indices (VI). They are recognised as being very helpful for monitoring the growth and well-being of crops in both qualitative and quantitative vegetation studies. Indicators of vegetation depend on how well a plant can absorb electromagnetic radiation (Tsouros *et al.,* 2019).

Using information from individual pictures or after the construction of orthophotos depicting the whole crop, these mathematical transformations of the electromagnetic spectrum may be found by scattering and absorption in different bands. Numerous elements, such as plant biochemical and physical features, environmental conditions, soil background characteristics, and moisture substance, have an impact on the reflectance in different bands. These translations can deliver precise spatial and temporal data on the crops being watched in agriculture (Tsouros *et al.,* 2019).

Every environment has unique and complicated qualities that must be considered when employing a VI. To detect vegetation, each VI has a unique blend of reflectance in several bands.

The primary idea is to merge reflections from various bands to reduce the "noise" from outside influences (lighting, calibration, soil, and atmosphere properties). The creation of straightforward plant indicators that can associate RGB data with several spectral bands, such as

NIR and RE, has dramatically improved the ability to recognise green, healthy vegetation. For instance, whereas chlorophyll absorbs visible radiation up to the red, it significantly reflects radiation in the NIR band. With this method, the soil in an image may be used to distinguish plants (Tsouros *et al.,* 2019).

The Red and NIR channels of radiation, which are used to construct the Ratio Vegetation Indices (RVI) as well as Normalised Difference Vegetation Indices (NDVI), are designed to highlight the distinction between the plant and the soil. The interaction between the two regions' reflections enables the removal of interference from elements that have an identical impact on each zone's radiation.

Two primary categories may be employed to distinguish between VIs based on visible spectrum data and vegetation indices based on multispectral or hyperspectral data (Tsouros *et al.,* 2019).

As one of the multispectral plant indices, the distinction between soil and vegetation is the RVI, which is exhibited in Equation (1). It is also vulnerable to the ground's visual characteristics.

$$\mathrm{RVI} = \frac{NIR}{R} \tag{1}$$

The near-infrared and visible light reflected from the vegetation is what determines the NDVI, which is a progression of RVI and appears in Equation (2). It is the most popular and often used indices. Numerous crops may be easily assessed for growth and health because diseased or sparse foliage reflects more visible light and less near-infrared radiation. This is based on NIR reflectance and red absorption that is influenced by chlorophyll (Tsouros *et al.,* 2019).

$$NDVI = \frac{NIR - R}{R + NIR} \tag{2}$$

Based on NDVI, other indices have been created. The Normalised Difference Red Edge (NDRE), demonstrated in Equation (3), uses the NDVI approach to normalise the ratio of NIR light to RE radiation. Equation (4) exhibits this for the Green Normalised Difference Vegetation Indices (GNDVI) for NIR and Green bands.

$$NDRE = \frac{NIR - RE}{RE + NIR} \tag{3}$$

$$\mathrm{GNDVI} = \frac{NIR - G}{NIR + G} \tag{4}$$

The two indices that are most often used for RGB pictures are the Excess Greenness Index (ExG) and the Normalised Difference Index (NDI). The fundament of ExG is the concept that soil is the only background component and that plants exhibit a high percentage of greenness. By increasing the radiation in the green channel and reducing the radiation in the red and blue channels, it may be calculated, as given by Equation (5).

$$\mathrm{ExG} = 2*G-R-B \tag{5}$$

The NDI was developed to distinguish between background images of dirt and debris and plants by using only the green and red channels, as demonstrated in Equation (6). A collection of the most popular VIs (Tsouros *et al.*, 2019).

$$NDI = \frac{G - R}{G + R} \tag{6}$$

2.4. Performance Metrics

Performance metrics or evaluation metrics are measuring tools that are used to evaluate the effectiveness or quality of the model. These performance measures enable researchers to analyse how successfully the simulation handled the data that was gathered. By altering the hyper-parameters, the model's performance may be enhanced (Kamilaris & Prenafeta-Boldú, 2018).

Machine learning tasks are separated into binary and multiclass classification, regression, clustering, anomaly detection, ranking, recommendation, forecasting, image classification, and object detection. Howbeit, not every metric can be applied to every task, thus it is essential to know which ones to employ. There are several assessment metrics used for both tasks (Sunasra, 2019).

Binary classification is a supervised task which determines which of two classes (categories) a particular instance of data belongs to. A set of labelled instances with each label being one of the numbers 0 or 1 serves as the input to a classification algorithm. The goal is to build a model that can accurately assign new data points to one of these two classes based on their features (Luis Quintanilla, 2022).

Multiclass classification is a supervised task where data is categorized into three or more distinct classes or categories, with each data point belonging to a single class. It involves training a model on labelled data to predict the correct class for new, unlabelled instances (Luis Quintanilla, 2022).

Regression is a supervised learning procedure with a continuous output that seeks to find the connections involving the independent and dependent variables. A quantitative or discrete value is what a predictive regression model anticipates. It is based on comparing predicted and actual data to find the difference (Sunasra, 2019).

Clustering is an unsupervised task that is employed to group similar data points into clusters or segments based on inherent patterns or similarities. It helps identify hidden structures within datasets without predefined labels. Common algorithms include K-Means and hierarchical clustering (Luis Quintanilla, 2022).

Anomaly detection is a task that identifies rare or unusual instances in a dataset that deviate significantly from the norm. This uses Principal Component Analysis (PCA) to build an anomaly detection model. It aims to pinpoint data points that are potential outliers or anomalies, which may signify errors, fraud, or unusual events (Luis Quintanilla, 2022).

Ranking is the process of arranging items or documents in a specific order based on their relevance or importance to a given query or context. It aims to provide users with a ranked list of results, making it easier to find the most relevant information or items (Luis Quintanilla, 2022).

Recommendation task refers to the process of suggesting items, products, or content to users based on their preferences, behaviour, or historical interactions. It leverages algorithms to analyse user data and make personalized suggestions to enhance user experiences (Luis Quintanilla, 2022).

Forecasting is a data analysis technique used to predict future values or trends based on historical data and patterns (Luis Quintanilla, 2022).

Object detection is a supervised task that involves identifying and locating multiple objects within an image or video stream, as well as drawing bounding boxes around them to specify their positions (Sunasra, 2019).

Image classification is a supervised task where the goal is to assign a single class label to an entire image or a specific region or object within an image. It involves determining what objects or categories are present in the given image or region without providing precise location information. Object classification is a fundamental building block in various computer vision tasks (Luis Quintanilla, 2022).

In a classification task, the category or classes of data are chosen using training data. Before categorising the new data according to the training, the model learns from the supplied dataset. Classification issues, such as image categorization and detection, are one of the most researched subjects worldwide (Sunasra, 2019).

One of the most understandable and straightforward measures for determining the correctness of the model is the confusion matrix. It is employed for problems involving classification in which the output might include two or more different classifications (Performance Metrics in Machine Learning, 2021).

Confusion Matrix is a tabular demonstration of model predictions and ground-truth (GT) labels. GT labels are specific and accurate annotations. The matrix rows represent the actual values, and the columns represent the expected values. The predicted and the actual values, both present two potential classes: positive or negative. It is helpful to visualise both the sorts of errors made in predicting the class of objects as well as the faults produced by the model. One of the evaluation factors is represented by each cell in the confusion matrix: True Positive (TP) is the proportion of positive class samples that the model properly predicted; The True Negative (TN) measure displays the number of negative class samples that the model correctly predicted; The number of samples from the negative class that the model incorrectly predicted is known as a false positive (FP), whereas the number of samples from the positive class that the model incorrectly predicted is known as a false negative (FN), (Sunasra, 2019), (Hasan *et al.,* 2021).

Table 2 shows a visual scheme of a general confusion matrix, where is possible to observe these relations between each type of value.

Table 1. Representation of a confusion matrix.

		Predicted Values	
		Negative	Positive
Real Values	Negative	TN	FP
	Positive	FN	TP

Accuracy (acc) is measured as the proportion of right predictions divided by the total number of predictions and is defined as the degree of closeness to the true value, as shown in Equation (7). The accuracy measure is a good option when the target classes in the data are distributed. On the other hand, should never be used as a measure when the target variable classes in the data are mostly one class. This is the most used metric to evaluate the DL models. The results vary between zero and one, and the closer to one one (100%) the better (Hasan *et al.,* 2021).

$$acc = \frac{\mathrm{TP} + \mathrm{TN}}{TP + FP + TN + FN} \tag{7}$$

The degree to which a process or instrument will repeat the same value is known as precision (P), in other words, it is the degree of reproducibility. It is the fraction between TP values, and the sum of TP and FP, as shown in Equation (8).

The results vary between zero and one. Precision provides information about how well the model is performing in terms of false positives. To minimize these values, the focus should be to make the metric as close to one (100%) as possible (Hasan *et al.,* 2021; Kamilaris & Prenafeta-Boldú, 2018).

$$P = \frac{\mathrm{TP}}{TP + FP} \tag{8}$$

Recall (R), sensitivity or True Positive Rate (TPR), is essentially the ratio of real positives to all ground truth positives. Specifically, it is a product's capacity to accurately identify the truth. It is the fraction of TP values, from the sum of TP and FN values, given by Equation (9). The results vary between zero and one. Recall reveals how well a classifier performs in terms of false negatives. To minimise these values, the focus should be to make the metric as close to one (100%) as possible (Swift *et al.,* 2020; Kamilaris & Prenafeta-Boldú, 2018).

$$R = \frac{\mathrm{TP}}{TP + FN} \tag{9}$$

The ability of a test to accurately identify negative findings is known as specificity or True Negative Rate (TNR). From the total of the TN and FP values, it is the percentage of TN values, as

shown in Equation (10). The results vary between zero and one (Swift *et al.,* 2020; Kamilaris & Prenafeta-Boldú, 2018).

$$TNR = \frac{\text{TN}}{TN + FP} \tag{10}$$

F1-score consists in the harmonic between P and R. A superior metric denotes strong recall and accuracy. It presents a good balance between precision and recall, presenting how precise and sensitive the system is. The performance of the model improves with increasing F1 Score. It consists of the fraction of the multiplication of P and R for the sum of P and R, and in the end, multiply the fraction for 2, as shown in Equation (11). The results vary between zero and one (Performance Metrics in Machine Learning, 2021; Kamilaris & Prenafeta-Boldú, 2018).

$$F1 - score = 2 \times \frac{P \times R}{P + R} \tag{11}$$

Kappa Coefficient Measures (k) is the level of agreement between actual values and predictions. The results vary between zero and one. A result less than 0.4 is typically regarded as poor. A kappa of >0.75 indicates excellent agreement, whereas values between 0.4 to 0.75 are considered moderate to good (Hasan *et al.,* 2021). Cohen's Kappa is calculated by subtracting the probability of agreement (P_a) from the probability of random agreement (P_r), which is calculated as the chance of agreement divided by 1, presented in Equation (12). In equation (13), it determines the value of the probability of agreement. In Equation (14) is calculated the probability of the true values and in Equation (15), the probability of the incorrect values. In Equation (16) it defines the value of the probability of random agreement (Hasan *et al.,* 2021).

$$k = \frac{P_a - P_r}{1 - P_r} \tag{12}$$

$$P_a = \frac{\text{TP} + \text{TN}}{Total} \tag{13}$$

$$P_{correct} = \frac{\text{TP} + \text{FN}}{Total} * \frac{\text{TP} + \text{FP}}{Total} \tag{14}$$

$$P_{incorrect} = \frac{\text{FP} + \text{TN}}{Total} * \frac{\text{TN} + \text{FN}}{Total} \tag{15}$$

$$P_r = P_{correct} + P_{incorrect} \tag{16}$$

The performance of a classification model is shown in a graph called the area under the receiver operating characteristic curve (AU-ROC or AUC) at various threshold levels, as is shown in Figure 5. The performance across all thresholds is calculated using the ROC, which is a probability curve that plots sensitivity with a False Positive Rate (FPR). It then provides an aggregate metric. The results vary between zero and one. Instead of focusing on the forecasts' absolute values, this metric should be used to assess how well they are ranked. In other words, a

model with 100% incorrect predictions will have an AUC of 0, whereas models with 100% right predictions will have an AUC of 1 (Rushikanjaria, 2021).

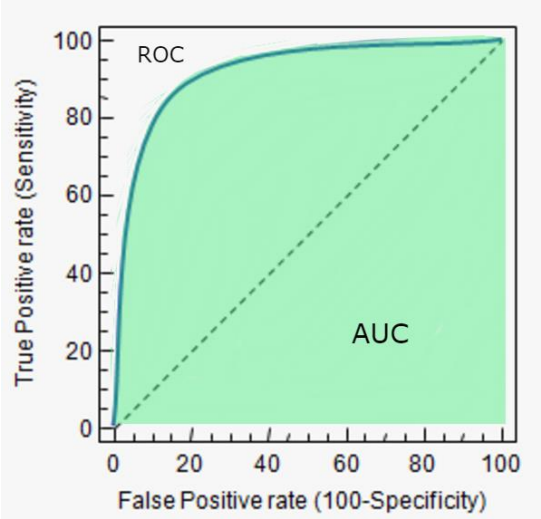


Figure 5. Area under the ROC Curve (Rushikanjaria, 2022).

2.5. Neural Networks Overview

Computer vision is advancing rapidly, owing in part to the significant role played by DL. It is a constituent of machine learning and finds widespread application in tasks such as image detection and categorization. Moreover, DL draws its theoretical underpinnings from the field of artificial neural networks (ANN). These networks are structured with multiple layers to emulate the hierarchical information processing observed in the human brain. By modelling the human brain, DL efficiently collects multilayer feature information in order to retrieve the essential feature information of images, texts, and other data. It is a procedure that moves from low-level feature extraction to high-level feature combining (Lv *et al.,* 2022; O'Shea & Nash, 2015).

ANNs are primarily made up of a large number of interconnected computational nodes (neurons), which collaborate to learn from the input in order to maximise the final output (O'Shea & Nash, 2015).

ANN typically develops the corresponding network structure in accordance with the different methods by which neurons are built.

A neural network is a type of operating model made up of several neurons linked to one another. Each neuron in the model serves as a processing function, and the weighted connections between different nodes indicate the ANN's memory capacity. The connection type, weight, and excitation function of various nodes all influence the output. The neural network model is also primarily built in accordance with some algorithm or function to express a certain logical action (Lv *et al.,* 2022; O'Shea & Nash, 2015).

A simple neural network model typically consists of an information input layer, a hidden layer, and an output layer for computation results. Multiple neurons can be found on several levels. The network's neurons each can accept input signals, process them, and provide an output signal. It comprises a collection of synapses that are weighted, an adder for adding input data that is weighted according to each synaptic strength, and an activation function for controlling the amplitude of the neuron's output (Lv *et al.*, 2022).

Multilayer feedforward networks, also known as multilayer perceptrons (MLPs), are a type of artificial neural network made up of multiple layers of linked nodes known as neurons (Zhang, 2018).

An MLP has an input layer, one or more hidden layers, and an output layer. Except for the input layer, each layer is made up of numerous neurons or units. Every neuron in one layer is linked to every other neuron in the following layer because neurons in successive levels are completely coupled. Each of these connections may have a varying strength or weight, but they are not all created equal (Zhang, 2018). A typical design is depicted in Figure 6, along with the connections between the neurons.

Each neuron in an MLP applies an activation function to the weighted sum of its inputs. The neurons are arranged into layers. The network gains non-linearity from the activation function, which enables it to learn intricate connections between inputs and outputs (Zhang, 2018).

An optimization technique is often used to iteratively change the weights and biases of the network to minimise a loss function during the training phase of an MLP. The gradients of the loss function with respect to the network's weights and biases are calculated using the backpropagation algorithm, a common method for training MLPs. The MLP's hyperparameters include the number of hidden layers and the number of neurons in each layer. These hyperparameters must be carefully selected based on the difficulty of the task, the quantity of training data available, and other factors (Zhang, 2018).

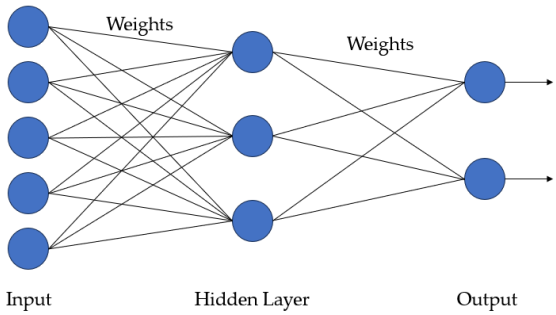


Figure 6. Typical architecture of a Multilayer Perceptron.

The most popular types of DL models are Recurrent Neural Networks (RNN), and CNN.

RNN and MLP are two groups of network topologies that are fundamentally distinct from one another (Zhang, 2018).

RNN are a subclass of ANN created especially for handling sequential input. RNNs feature feedback connections, which, in contrast to conventional feedforward neural networks, enable them to remember information from previous inputs and utilise it to predict or decide depending on the current input. The capacity of RNNs to work on data sequences, where each element in the sequence is processed one at a time, is its important characteristic (Zhang, 2018).

The network's memory is the hidden state, which stores details about previous inputs. The next recurrent unit in the sequence receives the output of the previous one as input. An optimisation approach is used to update the recurrent units' weights during the training of an RNN (Zhang, 2018).

The network receives the input at each time step, and the hidden state is then modified depending on the new input and the old hidden state. The output at the current time step is thus affected by the updated hidden state. At each time step, the hidden state is updated with data from the current input and the prior hidden state. An optimisation method is used to update the network's parameters, including its weights and biases (Zhang, 2018). A typical design of the RNN is depicted in Figure 7.

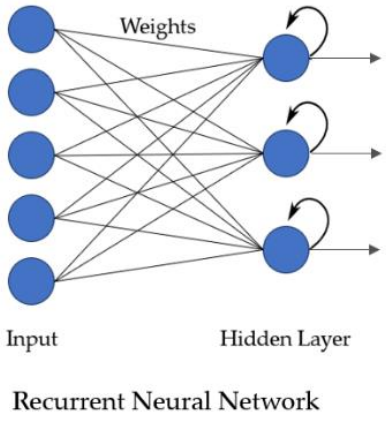


Figure 7. Typical architecture of a Recurrent Neural Network.

CNN is a common type of deep learning model network structure. CNN, as opposed to standard ML, are more suited for processing image and time series data, particularly for image classification and language recognition (Lv *et al.,* 2022). Visual data is represented in binary form by a digital picture. Pixel values are used to specify the brightness and colour of each pixel, and the pixels are organised in a grid-like pattern (Lv *et al.,* 2022).

Convolutional layers, pooling layers, and fully linked layers make up a standard CNN architecture. The convolutional layer completes the feature extraction process, while the feature mapping duty is handled by the pooling layer. The overall neural network structure and the entire connection layer are comparable. Although the nodes in this layer are linked to those in the layer

above, they are not connected. CNN also feature a data input layer and a result output layer, much like other neural networks (Lv *et al.*, 2022). The basic structure of the convolution neural network is shown in Figure 8.

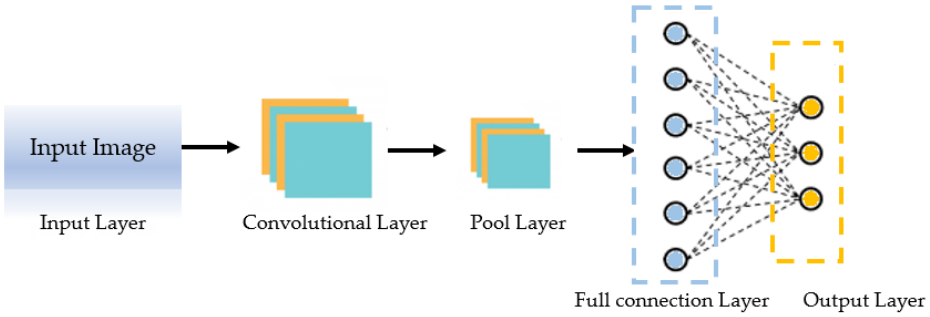


Figure 8. Convolutional neural network architecture.

The input layer in CNN contains the image data.

The foundation component of the CNN is the convolutional layer. It carries most of the computational load on the network. Filters are applied by convolutional layers to the input picture to find local characteristics and spatial patterns. This layer is used to extract features from the input and has a series of kernels, the parameters of which must be learnt (*Deep Learning*, n.d.; Lv *et al.*, 2022).

The convolution action between an input picture and a filter (also known as a kernel) can be expressed mathematically, as presented in Equation (17), where I is the input image, the filter is K, and the resulting feature map is F. The filter K moves one stride at a time (stride) through the input picture I and element-wise multiplication is done between the filter and the appropriate region of the input image at each place. The value at that location in the feature map F is created by adding the outcomes of these multiplications (*Deep Learning*, n.d.; Lv *et al.*, 2022).

$$F = I * K \tag{17}$$

A discrete mathematical representation of the convolution process is represented in Equation (18).

$$F(x,y) = (K * I)(i,j) = \sum_i \sum_j I(x+i, y+j) \cdot K(i,j) \tag{18}$$

Where, F(x,y)F(x,y) is the value at position (x,y)(x,y) in the feature map. I(x+i,y+j)I(x+i,y+j) is the value at position (x+i,y+j)(x+i,y+j) in the input image. K(i,j)K(i,j) is the value at position (i,j)(i,j) in the filter. And, j and i are the dimensions (height and width) of the filter (*Deep Learning*, n.d.).

An example of the convolution operations is represented in Figure 9.

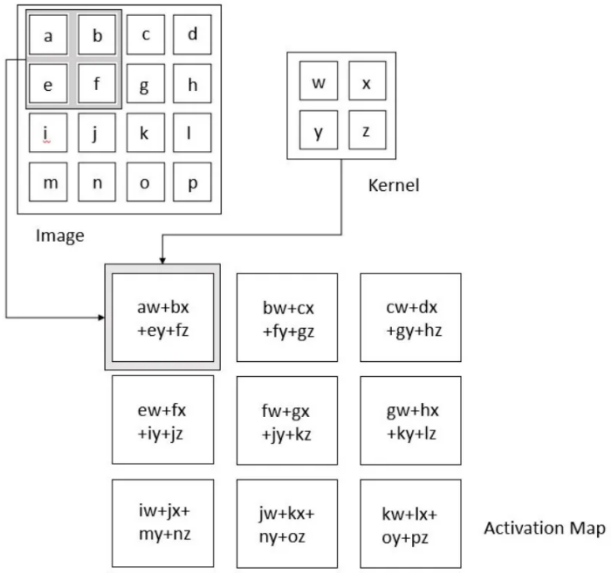


Figure 9. Convolutional operation. (*Deep Learning*, n.d.).

By calculating an aggregate statistic from the surrounding outputs, the pooling layer substitutes for the network's output at certain points. This aids in decreasing the representation's spatial size, which lowers the amount of computation and weights needed. Each slice of the representation is subjected to the pooling procedure separately (Mishra, 2020; Lv *et al.,* 2022).

During this process, the pooling window moves in steps over the input feature map. The largest value found in the window is chosen at each point and positioned at that location in the pooled feature map (Mishra, 2020).

The feature map may be downsampled, made smaller, and yet maintain the most important data inside each pooling zone by using max pooling. As a result, the input data may be more succinctly represented while maintaining key characteristics (Mishra, 2020).

The maximum pooling operation for a single pooling zone may be expressed mathematically as follows. The max pooling procedure inserts the highest value possible into the resultant pooled feature map, given a pooling window or area of size p. The abbreviation F is for the input feature map, P for the pooled feature map, and p for the size of the pooling window (Mishra, 2020).

Thus, P=maxpool(F,p) is a representation of the max pooling procedure. The mathematical representation of the max pooling operation is represented in Equation (19).

$$\mathrm{P}(\mathrm{x},\mathrm{y}) = max_{i=0}^{p-1} max_{j=0}^{p-1} \mathrm{F}(\mathrm{x} * \mathrm{p} + \mathrm{i}, \mathrm{y} * \mathrm{p} + \mathrm{j}) \quad (19)$$

Where, P(x,y)P(x,y) is the value at position (x,y)(x,y) in the pooled feature map. F(x·p+i,y·p+j) represents the value at the corresponding position in the input feature map FF. And, p is the size of the pooling window (Mishra, 2020). Figure 10 shows a representation of the pooling operation (Mishra, 2020).

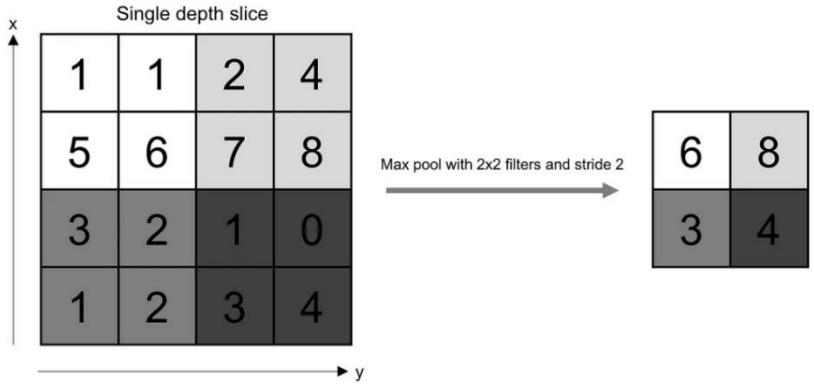


Figure 10. Pooling operation. (Mishra, 2020).

The last layers are made up of completely interconnected layers that use the features that the preceding layer retrieved to provide class probabilities or scores. All of the neurons in the layer above are totally linked to this layer (Mishra, 2020).

CNN also features a data input layer and a result output layer, much like other neural networks. The convolution layer serves as the primary means by which the CNN does its computations, and the convolution kernel found inside serves as the fundamental component of the CNN model. Convolution check is used by the convolution layer to convolute the input picture and retrieve the image's identifying information (Tan & Le, 2020).

The pixels near the edge of the image have minimal impact on the output results, and the convolution operation will progressively reduce the size of the images. In an image, there is typically a significant connection between nearby pixels (Lv *et al.,* 2022).

The primary function of the convolution kernel is to collect local characteristics from the picture and deliver them to a high level for integration processing. Due to the bottom feature of the image being independent of its position, it is possible to reduce the number of parameters in the neural network by using the shared weight property of the convolution kernel in addition to using the same convolution check to extract the relevant features. This increases the effectiveness of the network model's training process (Lv *et al.,* 2022).

So, in short, the first stage is the outcome of a local convolution of the preceding layer (the kernel has trainable weights), and the second stage is a "max-pooling stage," where the number of units is significantly reduced by keeping only the maximum response of a few units from the first stage. This process is repeated according to the number of hidden layers. The last layer is often a completely linked layer after a few concealed levels. Each unit takes input from all the units of the layer below and has a unit for each class that the network predicts (Lv *et al.,* 2022).

Figure 11 shows a schematic representation of a CNN with two hidden layers.

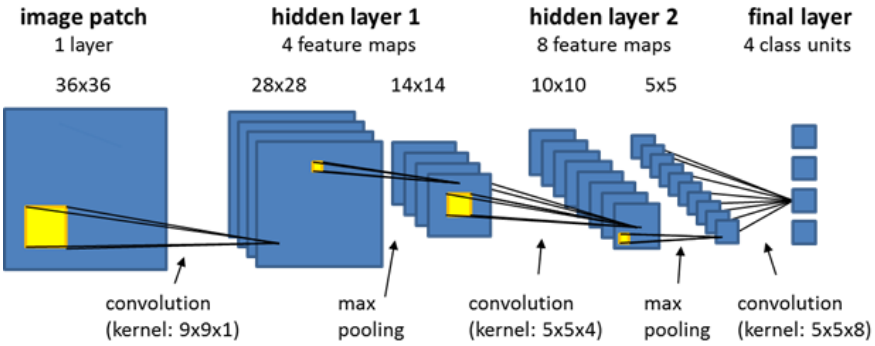


Figure 11. Representation of a convolutional neural network with two hidden layers (*Using Deep Learning Models / Convolutional Neural Networks*, n.d.).

2.6. Loss Function and Optimizers

Algorithms for machine learning must include loss functions, commonly referred to as cost functions or objective functions. They serve as an indicator of how well a machine learning model is working by putting a number on the discrepancy between the algorithm's predictions and the actual target values (Fu *et al.,* 2023).

A loss function is primarily used to measure the inaccuracy or difference between the predicted values of the model and the actual target values. It offers a single numerical number that demonstrates how effectively or poorly the model is working with a certain dataset (Fu *et al.,* 2023).

Secondly, ML models are trained using loss functions, which direct the optimisation procedure. By iteratively changing the model's parameters (such as the weights and biases in neural networks), the objective is to minimise the loss function (Fu *et al.,* 2023).

Different loss functions are needed for various ML applications and methods. The Mean Squared Error (MSE), which is used in regression tasks and calculates the average squared difference between predicted and actual data (Fu *et al.,* 2023).

The dissimilarity between predicted probability and actual binary labels is measured by the Binary Cross-Entropy Loss, which is used for binary classification problems (Fu *et al.,* 2023).

The Categorical Cross-Entropy Loss, employed in multi-class (more than two) classification problems, calculates the difference between actual one-hot encoded class labels and anticipated probability distributions(Fu *et al.,* 2023).

Additionally, Hinge Loss can be utilised with support vector machine (SVM) and certain linear classifiers. It provides a margin of distinction to encourage accurate categorization (Fu *et al.,* 2023).

Cross-entropy loss, often known as "cross-entropy" or "log loss," is a frequently employed loss function in deep learning and machine learning, notably for classification issues. The cross-entropy loss measures how well the predicted probabilities match the true probabilities. When the cross-entropy loss is low, a model has a good fit because the projected probabilities are near

to the real probabilities. In contrast, the loss increases when they are considerably dissimilar, suggesting a poor match (Fu *et al.,* 2023).

The categorical cross-entropy loss is defined as shown in Equation (20).

$$\mathrm{L(t,x)} = -\sum_i ti \ \log fi \tag{20}$$

Where y is the true probability distribution over classes, and f is the predicted probability distribution over classes. L(t,f) is the categorical cross-entropy loss. And, $\sum_i$ is the sum of all classes (Fu *et al.,* 2023).

In ML, optimizers are key players in the training of models by minimising or maximising a specified objective function, which is often a loss function that measures the effectiveness of the model on a task. The task of optimizers is to iteratively change the model's parameters until the loss is minimised. To train ML models to produce precise predictions or judgements, this optimisation process is crucial (Habib & Qureshi, 2022).

ML frequently employs a variety of optimisation methods, each with unique properties and benefits. Popular optimizers include Stochastic Gradient Descent (SGD), where the settings are updated using gradients calculated on tiny, randomly chosen mini-batches of data (Habib & Qureshi, 2022).

Adam combines the advantages of the RMSprop and momentum approaches. Based on previous gradients and squared gradients, it adjusts the learning rate (Habib & Qureshi, 2022).

The Root Mean Square Propagation (RMSprop) modifies the learning rates for each parameter separately, which can mitigate gradient descent's drawbacks (Habib & Qureshi, 2022).

Momentum increases the current parameter update by a small portion of the previous update. In situations when the cost function is very curved, it aids in accelerating convergence (Habib & Qureshi, 2022).

Adagrad uses historical gradients to adjust the learning rate for each parameter, which is useful for sparse data (Habib & Qureshi, 2022).

Adam is a popular ML optimisation strategy, especially for deep neural network training. The advantages of the momentum and RMSprop approaches are combined in this modification of the SGD optimisation process. Adam is renowned for its capacity to adaptively alter the learning rate for each parameter, which frequently leads to more rapid convergence and improved training results (Habib & Qureshi, 2022).

The resilience and adaptability of Adam are well known. It can tolerate noisy or sparse gradients. It is frequently used as the default optimizer in several deep learning frameworks, including TensorFlow and PyTorch, and has gained popularity for optimising deep neural networks (Habib & Qureshi, 2022).

The parameter update rule for Adam is demonstrated in Equation (21).

$$\theta_{n+1} = \theta_n \ - \frac{\alpha}{\sqrt{\hat{v}_n} + \varepsilon} * \hat{m}_n \tag{21}$$

Where θ_n represents the parameters being optimized at iteration t. The α is the learning rate. The ε is a small constant. And $\hat{m}_n$ and $\hat{v}_n$ are the bias-corrected estimates of the first moment and second moment of the gradients, respectively.

2.7. Neural Network Architectures

For classification, PyTorch has different models available, with or without pre-trained weights. Here are some notable CNN architectures commonly used, offered by Pyorch, and that will be studied to evaluate the best model, for image classification tasks.

AlexNet is a network structure with 8 layers, comprising three fully linked layers and five convolutional layers. Following the convolutional layers are the max-pooling layers, which reduce the spatial dimensionality by down sampling the feature maps. Rectified linear units (ReLU) served as the network's activation function. ReLU adds non-linearity and aids in solving the vanishing gradient issue, making it possible to train deeper networks effectively. Additionally, a normalization method known as Local Response Normalisation (LRN) was used following ReLU activation. By encouraging competition between nearby neurons, LRN improves the network's capacity to generalize and adapt to changes in input. AlexNet uses overlapping max pooling, which is different from conventional pooling methods. This method created overlapping areas by doing pooling operations with a stride that was less than the pool size, hence minimizing the loss of spatial information. Dropout regularization was used by AlexNet to avoid overfitting. During training, dropout randomly removes a portion of the neurons, driving the network to learn more durable and universal properties (Wang, 2003).

VGG (Visual Geometry Group) is a popular CNN architecture known for its simplicity and effectiveness in image classification. VGG employs a deep architecture with multiple convolutional layers and max-pooling layers. The key idea is to stack small-sized filters (typically 3x3) to increase the network's depth and capture complex features. VGG's architecture follows a consistent pattern, with 3x3 filters and 2x2 max-pooling layers. This regularity simplifies implementation and training. Although VGG is primarily known for image classification, its pre-trained models have also been used for other computer vision tasks, including object detection and segmentation. Vgg-16 and Vgg-19 networks have been popular in recent years for VggNet models (Wang, 2003).

ConvNeXt uses group convolutions, a method that divides input channels into various groups and applies different filters to each group. As a result, the network can simultaneously collect several feature types, which encourages efficient feature learning. Path aggregation, which includes merging data from many convolutional pathways, is the main emphasis of the ConvNeXt architecture. By using a concatenation process, ConvNeXt combines features picked up by many groups, allowing the network to gather a variety of complementary data. It introduces the idea of cardinality, which describes how many groups there are in group convolutions. The network may regulate the complexity and capacity of the model using the cardinality parameter. It strikes a

compromise between the overall network's computational efficiency and the trade-off between the representational strength of different groups. Similar to the well-known ResNet design, ConvNeXt has a bottleneck structure. A series of convolutions with dimensions of 1x1, 3x3, and 1x1 make up the bottleneck structure. This architecture enables the network to successfully capture both low-level and high-level information while reducing the number of parameters and computational cost. It also offers many variations with various depths and capacities (Simonyan & Zisserman, 2015).

The GoogLeNet architecture introduced inception modules to capture multi-scale characteristics. These modules consist of parallel convolutional processes with filter sizes of 1x1, 3x3, and 5x5, along with a max-pooling layer. By using multiple filter sizes, the network can gather local and global information effectively. To reduce computational cost and parameters, 1x1 convolutional layers were employed for dimensionality reduction. These layers decreased the number of input channels before larger filter sizes, enabling efficient processing in subsequent layers. Auxiliary classifiers were added to address the vanishing gradient problem during training. These classifiers, consisting of fully connected and 1x1 convolutional layers, provided additional gradients and encouraged the learning of discriminative features. Instead of fully connected layers, GoogLeNet employed global average pooling at the network's end. This technique computed the average value of each feature map across spatial dimensions, resulting in a fixed-size feature vector for classification and reduced overfitting. GoogLeNet had 22 layers and used a higher number of filters per layer (Simonyan & Zisserman, 2015).

DenseNet creates dense connectivity by establishing feed-forward connections between every layer. This indicates that all levels before it directly feeds data to the layer above it. This connection structure improves gradient flow and mitigates the vanishing gradient issue by facilitating feature reuse and allowing information to travel over shorter channels. It is made up of dense blocks, which represent a network's stack of layers. A dense block's layers are all linked to each other, enabling the concatenation of features. By mixing local and global information, dense blocks promote feature reuse and help the network learn more discriminative features. Between dense blocks, transition layers are added to limit the amount of feature maps and spatial dimensions. 1x1 convolutional layers are frequently used as transition layers, which are then followed by average pooling. They decrease the amount of feature maps, which helps to compress the data and improve computing performance. A growth rate hyperparameter introduced by DenseNet controls how many feature mappings are added to each layer inside a dense block. The growth rate controls the network's capacity and complexity, enabling a variable trade-off between model size and performance. Batch normalisation and ReLU activation functions, which are incorporated into DenseNet, increase the network's stability and nonlinearity. Each layer's input is normalised by batch normalisation, which enhances training efficiency. In transition layers, it also introduces a hyperparameter compression factor. By lowering the number of feature maps by a certain ratio, the compression factor lowers computing complexity while maintaining crucial data. It enables model size control while retaining computational efficiency (Ma *et al.,* 2018).

A neural network design called MobileNet makes use of depth-wise separable convolutions, which are made up of a depth wise convolution and a point-wise convolution. This factorization approach maintains spatial and channel interdependence while reducing computational cost and parameter count. It is appropriate for devices with limited resources since it seeks to reduce model parameters while keeping adequate accuracy. To manage the number of channels and layers, respectively, MobileNet offers a width multiplier and a depth multiplier, permitting trade-offs between model size and accuracy. Pre-trained models developed using transfer learning and fine-tuning on massive datasets like ImageNet are offered, allowing for quicker development and deployment (Ma *et al.,* 2018).

SqueezeNet is a compact neural network architecture that achieves high performance with fewer parameters. It uses 1x1 convolutional filters to reduce the number of input channels, reducing computational complexity. The model incorporates skip connections to improve gradient flow and address the vanishing gradient problem. SqueezeNet also utilizes aggressive down sampling and efficient fire modules to further reduce parameters. It strikes a balance between model size and accuracy, making it suitable for resource-constrained devices (Ma *et al.,* 2018).

ShuffleNet V2 is a convolutional neural network architecture that incorporates a channel shuffle operation to enable information exchange between channels. This operation is performed within Shuffle units, which consist of pointwise and depth wise convolutions. The channel shuffle promotes information flow and enhances the network's representational power. To reduce the computational cost, ShuffleNet V2 employs group convolutions in the depth wise convolution, reducing parameters and computation. It also employs a feature map alignment technique called channel split and concat to preserve information during down sampling. The architecture is designed hierarchically, capturing features at different abstraction levels, and uses different block configurations to balance accuracy and efficiency (Ma *et al.,* 2018).

The compound scaling mechanism used by EfficientNet scales the network's depth, width, and resolution consistently. The scaling of these dimensions is controlled by a compound coefficient. EfficientNet strikes a balance between model capacity and computing efficiency by scaling all facets of the network. The network's breadth and depth are measured in terms of the number of channels or filters present in each layer, respectively. By scaling these dimensions appropriately, EfficientNet permits more intricate representations while keeping computation costs under control. It adjusts the resolution of the input picture, allowing the network to gather more minute features. Improved accuracy may result from higher resolution input, but this comes at a higher processing cost. The compound scaling approach strikes the ideal balance between computing speed and resolution. Building blocks for MobileNetV2 that are depth-wise separable convolutions and inverted residual blocks are adjusted for use in EfficientNet. These building components effectively capture geographical and channel-wise information and are computationally efficient. The channel-wise feature responses in this model are adaptively recalibrated using squeeze-and-excitation (SE) blocks. SE blocks enhance valuable traits while suppressing unhelpful ones, increasing the strength of representation. To determine the ideal

scaling coefficients for depth, breadth, and resolution, it uses an automated machine learning (AutoML) based methodology (Radosavovic *et al.,* 2020).

EfficientNetV2 improves upon the compound scaling of EfficientNet. It introduces a new scaling method that optimizes network scaling coefficients to improve efficiency further. This model employs bottleneck residual blocks inspired by ResNet. These blocks use a combination of 1x1 and 3x3 convolutions to capture complex patterns. The use of bottlenecks reduces computational cost and allows deeper networks to be trained more efficiently. SE blocks are placed at the output of each block, enhancing the network's representational capacity. And it introduces stochastic depth, where a random subset of layers is dropped during training. This regularization technique enables the training of very deep networks more effectively (Radosavovic *et al.,* 2020).

The MaxViT architecture is a groundbreaking development in computer vision, merging Transformer and convolutional techniques to enhance image processing. Unlike traditional Vision Transformers (ViT) that break down images into patches for global analysis, MaxViT employs multi-axis attention, capturing both local and global interactions among these patches. This enables detailed examination of fine and coarse-grained properties in input data. MaxViT integrates key components, starting with a convolutional stem for initial image processing, followed by multi-axis self-attention blocks (Max-SA). Max-SA employs Mobile Inverted Bottleneck Convolution (MBConv) blocks with depthwise separable convolutions, reducing computational costs while preserving performance. The inverted bottleneck concept augments channel numbers before convolution, allowing the model to discern intricate patterns without significantly increasing computational demands. Within Max-SA, the SE module fine-tunes channel-wise features, focusing on informative channels. The attention mechanism combines fixed-size local attention windows for specific spatial focus and sparse global attention grids operating on a grid-like structure. These grids calculate attention scores, guiding position-specific attention. This method enhances object and pattern recognition by understanding connections and dependencies in the input data (Tu *et al.,* 2022).

RegNet is a neural network architecture that employs a systematic approach to network design. It explores a large design space of architecture while adhering to architectural design principles to guide the search for high-performing models. It incorporates compound scaling, which uniformly scales the depth, width, and resolution of the network, allowing for easy adjustment of model capacity and computational cost. RegNet introduces an adaptive scaling rule to find the optimal balance between model size and accuracy based on desired constraints. It emphasizes network width and depth, varying the number of channels and layers to explore different architectural configurations. To improve generalization and prevent overfitting, RegNet employs regularization techniques such as weight decay, dropout, and stochastic depth. It can be trained using standard deep learning optimization techniques like SGD with momentum, benefiting from learning rate schedules, data augmentation, and batch normalization. Overall, RegNet provides a framework for designing high-performing models with efficient computational characteristics (Radosavovic *et al.,* 2020).

Inception V3, inspired by GoogLeNet, introduces the concept of inception modules featuring dimensionality reduction and parallel convolutional operations. These modules employ 1x1 convolutions and filters of various sizes (1x1, 3x3, and 5x5) to capture local and global information effectively. Factorization techniques replace the 5x5 convolution with two consecutive 3x3 convolutions, reducing parameters while preserving the receptive field. Batch normalization is incorporated to normalize inputs and improve convergence, gradient flow, and network stability. Reduction blocks are added between inception modules to downscale spatial dimensions using 1x1 convolutions and max pooling, increasing channel capacity. Auxiliary classifiers are used at intermediate levels during training to address the vanishing gradient problem, provide regularization, and enhance gradient flow. Inception V3 combines these features to create a powerful and efficient convolutional neural network architecture (Szegedy *et al.,* 2015).

MNASNet is a neural network architecture that utilizes neural architecture search (NAS) to discover optimal network configurations. It focuses on reducing model size and improving computational efficiency for deployment on resource-constrained devices. MNASNet employs depth-wise separable convolutions, which reduce complexity while maintaining spatial and channel relationships. The design incorporates mobile-specific constraints for optimized mobile deployment. The neural architecture search process explores a wide range of architectures to identify high-performing configurations. MNASNet aims to provide compact and efficient neural networks suitable for mobile devices (Tan *et al.,* 2019).

ResNet is a deep neural network architecture that addresses the vanishing gradients problem in deep networks. It introduces skip connections or shortcuts that allow gradients to flow easily during backpropagation. The fundamental building block, called a residual block, consists of convolutional layers with a skip connection that adds the original input to the output. This enables the network to learn residual mappings and focus on the residual error. ResNet is renowned for its ability to build extremely deep networks by stacking residual blocks, allowing for hundreds or thousands of layers. Deeper networks can learn more complex features, leading to improved accuracy. ResNet also introduced pre-activation, where batch normalization and activation functions are applied before convolutional layers, aiding optimization. To reduce computational complexity, ResNet incorporates bottleneck structures in some residual blocks, using 1x1 and 3x3 convolutions to decrease and then increase the dimensionality. This approach reduces parameters and computations (Boesch, 2023).

SwinTransformer is a cutting-edge computer vision model that combines the power of Transformers and CNNs to excel in various visual tasks. It introduces a hierarchical design with shifted windows, enabling efficient processing of image patches. By using shifted windows instead of traditional self-attention, SwinTransformer captures global and local information effectively. The model consists of multiple stages, each containing a hierarchical transformer encoder. This design allows the model to capture information at different scales and resolutions. SwinTransformer incorporates a shifted window self-attention mechanism within each stage, facilitating parallel processing and reducing computational complexity. To incorporate positional information, SwinTransformer utilizes window-based position embedding, encoding spatial

relationships at the patch level. This ensures accurate modelling of object locations and improves overall performance. Furthermore, SwinTransformer employs a layer-wise feature map shifting strategy to enhance information propagation and gradient flow across different network layers. This facilitates efficient learning and optimization. SwinTransformer has demonstrated exceptional performance in tasks such as image classification and object detection. It strikes a balance between accuracy and computational efficiency, making it suitable for various computing systems and devices with different resource constraints (Liu *et al.,* 2021).

The Vision Transformer is a deep learning model that applies the transformer architecture to computer vision tasks. It treats images as sequences of patches and uses a transformer encoder to capture global dependencies among these patches. ViT replaces traditional convolutional layers with self-attention layers and feed-forward networks to process the patch embedding. It has achieved impressive results in image classification, object detection, and image segmentation tasks. However, to handle high-resolution images, a hybrid approach called Hybrid Vision Transformer combines the Vision Transformer with a CNN stem for local feature extraction (Dosovitskiy *et al.,* 2021).

Wide ResNet is an enhanced version of the original ResNet architecture designed to improve both performance and efficiency in DL tasks. In Wide ResNet, the number of channels in each convolutional layer is increased significantly compared to the original ResNet. This increase in width makes the network larger. It utilizes residual blocks, which consist of two or more convolutional layers. These blocks use shortcut connections to address the vanishing gradient problem, allowing the model to be very deep. The basic building block of Wide ResNet consists of two convolutional layers with Batch Normalization and ReLU activation, followed by a shortcut connection. The residual connections ensure that gradients can flow easily through the network during training. By increasing the width, Wide ResNet captures more diverse features at each layer. This is particularly helpful in tasks where capturing fine-grained details is essential, such as fine-grained object recognition. It comes in different configurations denoted as WRN-X-Y, where X represents the widening factor (how much wider the network is compared to the original ResNet), and Y represents the depth of the network (the number of residual blocks), (Zagoruyko & Komodakis, 2017).

ResNeXT (Residual Neural Network with Next) is a sophisticated neural network architecture introduced by Microsoft Research Asia. It's designed to improve the efficiency and accuracy of deep learning models, specifically in the field of computer vision. ResNeXT builds upon the foundation of the original ResNet (Residual Network) by introducing a new element called cardinality. The cardinality in ResNeXT refers to the number of parallel paths within each residual block. Unlike traditional architectures where the convolutional layers learn all the features together, ResNeXT allows the network to split into several parallel pathways (or groups). Each group captures a distinct set of features from the input data. Within each residual block, multiple parallel paths operate in parallel, allowing ResNeXT to learn a diverse range of features simultaneously. This parallelism enhances the network's ability to capture different aspects of the data. The parallel pathways are combined, promoting feature diversity within the network. The

introduction of cardinality enables the model to efficiently learn a variety of feature representations. By allowing the network to explore different feature spaces in parallel, ResNeXT can capture nuanced patterns and representations from the data, leading to improved performance on complex tasks (Xie et al., 2017).

2.8. Experimental Studies

In agriculture, weed management is a decisive procedure. Automation in agriculture has been made possible by technological developments in hardware and software. With automated technologies, the weeding activity is managed electronically, minimizing human involvement, and maximizing the machine's power (Sujaritha *et al.,* 2017).

Modern agricultural procedures establish computer vision knowledge in weeding. Computer vision is characterised as a technique, method, or system for automatically running and managing a mechanical process or equipment. Other AI subfields, such as machine learning and deep learning, can be used in this procedure (Littman *et al.,* 2021).

The most common characteristics used in earlier literature on this topic were colour, structure, spectral content, and texture (Sujaritha *et al.,* 2017). Realizing this, the works that are developed, use this type of physical qualities to their work, and correlate them with different forms of artificial intelligence.

Over the years, plenty of different studies whose objective is classification and/or detection based on images of the field are being developed and improved. Some of the techniques currently in use in weed detection in agricultural areas are described afterwards.

In addition to weed detection, AI is being used in a wide range of other fields. Examples of these applications include the identification of plant and leaf diseases as well as the detection and identification of fruits and vegetables. It is important to note, that the term “weeds” will be applied because the authors used it.

2.8.1. Disease Detection

Plant illnesses can decrease agriculture production and compromise global food security. Spraying pesticides uniformly throughout the crop area is the most typical method of controlling pests and diseases. This technique uses a lot of pesticides, which uses a lot of money and resources and has a large impact on the environment. Reduced resource consumption and adverse effects on the environment can be achieved through a ML system that indicates the precise time, location, and damaged crops and can spray pesticides only on those plants (Alibabaei *et al.,* 2022).

Identifying disease in particular fruits

Afonso et al. (2019) aimed to use DL to categorize healthy or blackleg-infected potato plants. The colour images were taken with the help of an industrial RGB camera installed on a tractor. Furthermore, using a train-test ratio of 80:20. the collection of photos was randomly divided into a test set of 106 photos (60 healthy, and 46 blackleg diseased), and a training set of 426 images

(218 healthy, 208 blackleg ill). The CNN classifier was independently benchmarked using the test set. Two deep CNN with residual networks, ResNet18 and ResNet50 were trained using RGB images of both healthy and diseased plants. The project used transfer learning for network weight initialization and the Adam optimizer for weight optimization with a model that has previously been trained on the ImageNet dataset. A batch size of 12 was used to train both networks throughout 100 epochs. The percentage of photos accurately categorized as having blackleg illness or being truly healthy is displayed in a confusion matrix. ResNet18 was experimentally the best network, with 94% of the pictures properly characterized. However, in ResNet50 only 82% were classified precisely. For the healthy class, recall was 83% and precision was 85%.

In Assunção et al. (2020), a deep neural network (DNN) that can work on mobile devices to categorize healthy peach fruits (healthy, rot, mildew, and scab) was demonstrated. In this study, the authors evaluated the results of the illness's classification in a dataset of peach fruit illnesses using transfer learning, data augmentation, and CNN with the model MobileNetV2, which was trained on the ImageNet dataset. The APPIZZÊRE, Forestry Images, PlantVillage, Pacific Northwest Pest Management Handbooks, Utah State University, and the University of Georgia provided the RGB images required to create the peach dataset. The ImageNet dataset served as the model's initial training set. A fully connected layer was trained on the APPIZZERE dataset for the peach illness to take into consideration the variations in the source and target data pictures. Using the Keras API, the model was enlarged to include a fully linked layer and a Softmax activation function.

The classifications of rot and mildew each had an F1-score of 96%, whereas scab disease had the highest F1-score of 100%. Healthy fruit was categorized with an F1 score of 94%. For the model's overall performance, the macro-average F1-score was 96%. The model accurately classifies all disease classes, which is essential for disease investigation for infection and control. These accomplishments demonstrate CNN's potential for identifying fruit illnesses with less training data. Additionally designed to function with portable devices is the model.

According to Azgomi et al. (2022), a cheap approach for detecting apple disease has been developed. A MLP neural network used the properties extracted from the photos as its input and produced the specified classes (classes of scab, bitter rot, black rot, and healthy fruits) as its output. The images were captured with a digital camera. This method used colour and texture, the ground truth, which represents a feature, is present in both spectra. The fruits in this setup should be placed individually in a container such as a lightbox before being shot. Images were reduced after being taken to accelerate the processing algorithms' execution. Considering the lighting circumstances, pre-processing was not applied to the photographs to change the brightness of the image pixels to enhance their quality or remove noise. The following phase was clustering photos based on zones with comparable attributes. In the current investigation, the RGB and L*a*b* models were used to segment the normal and damaged fruit samples. The K-Means algorithm was used in the study of photo clustering. The pictures were consequently first converted from RGB to L*a*b*. Then, the colour-containing pixels a* and b* were separated and

utilized as the K-Means algorithm's inputs. The method was designed to build these clusters using the Euclidean distance, and it was programmed to repeat three times.

In the next step, colour and texture features were extracted. Colour is a characteristic that is extremely important in the identification of diseases. SVM categorization was done semi-automatically. The illness is then identified by examining the characteristics of the selected clusters. To improve the process, make it entirely autonomous, and explore the possibility of raising the generated system's accuracy, an artificial neural network (ANN) was used. The apples with three distinct illnesses and healthy apples were the outcomes of the research, which employed a MLP with features as the input. Using the Levenberg–Marquardt algorithm (LMA), the network was trained. In the end, this system ought to be able to analyse the photos and determine the sickness and whether the fruit is unhealthy.

The neural network trained with 60% of the data was then assessed for accuracy using a variety of designs. The greatest accuracy achieved was 73.7% using a two-layer construction with eight neurons in the first layer and eight neurons in the second layer. Figure 12 shows an input of apple fruits that includes both healthy and diseased areas. The healthy area is highlighted in yellow in the middle image, while the damaged area is depicted in black in the right image.

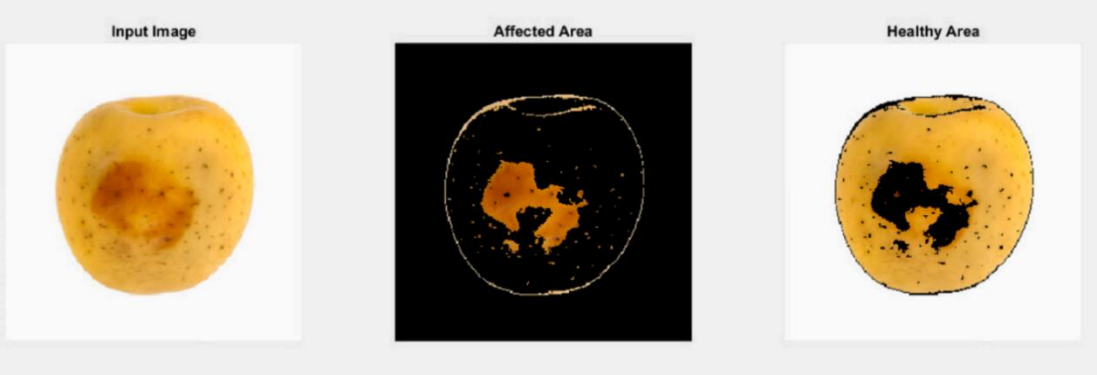


Figure 12. A visual representation of the input fruit's divided into unhealthy (yellow) and diseased (black) portions (Azgomi *et al.*, 2022).

Identifying diseases in farming areas

In Kerkech *et al.* (2020), the recommended method uses a DL segmentation methodology using UAV pictures to detect mildew disease in vineyards. A UAV configured with an IR sensor and an RGB sensor set up for automatic lighting was used to gather the data.

A semi-automatic approach for pixel-wise tagging was utilized in the first stage. A LeNet5 network then assigned each block a classification for pre-labelling. In the second step, visible and IR images are divided into four groups using the SegNet architecture: shadow, ground, healthy, and symptomatic vine. In the third and final stage, segmentations are combined to create a disease map. For the RGB and IR pictures, respectively, two models were trained. The two models'

segmentation outputs were also combined in 2 distinct ways. When a symptom appears on both RGB and IR pictures, the condition "Fusion AND" is thought to be present. The symptom is diagnosed in the second scenario, referred to as "fusion by the union," if it can be observed in either the RGB or the IR picture and is denoted by the symbol "fusion OR".

With accuracy rates of 85.13% and 78.72%, respectively, the RGB image-based model fared better than the infrared image-based model. Additionally, with an accuracy of 92.23% and 82.80%, respectively, the model fusion OR outperformed the fusion AND. SegNet's runtime on a UAV picture was calculated to be 140s (seconds) for visible and infrared images. The two segmented images can be combined in less than two seconds. Figure 13 shows a SegNet segmentation example along with a comparison of the fusion to the real data. The visible and infrared estimates and the fusion are identical in the first one, which has the range [a-h], and does not display samples of the symptom class, thus it is healthy. However, it is feasible to discern parallels between the visible and infrared symptoms in the second area [i- p], which is almost fully contaminated by mildew.

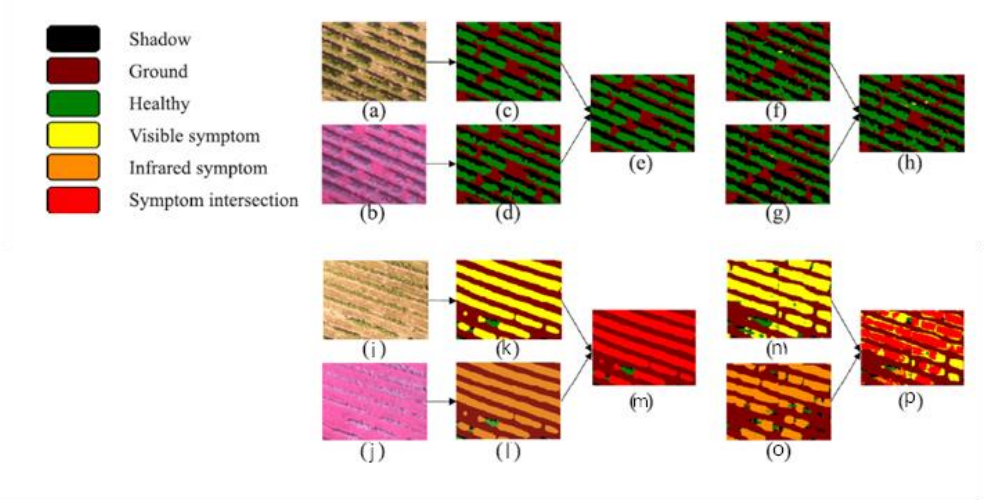


Figure 13. In the healthy region, an illustration of segmentation and fusion. Visible image (a), IR image (b), Visible ground truth (c), IR ground truth (d), Fusion ground truth (e), Visible SegNet estimate (f), IR SegNet estimation (g), and Fusion of segmentation results (h). A region affected by mildew is used as an example for segmenting and fusing. Visible image (i), Infrared image (j), Visible ground truth (k), IR ground truth (l), Fusion ground truth (m), Visible SegNet estimation (n), (o): IR SegNet estimation (o), Fusion of segmentation results (p), (Adapted from Kerkech *et al.,* 2020).

Table 2 gathers an overview of research work published in the "Disease Detection" field. The table includes the reference, the application, the data, model, and metric applied. The table also includes a column describing the model performance.

Table 2. Overview of research work in the "Disease Detection" field.

References	Application	Data applied	Model applied	Metric applied	Model performance
Afonso *et al.* (2019)	Detect potatoes disease.	RGB camera.	Deep CNN: ResNet18, ResNet50.	Precision Recall	According to the study's conclusions, ResNet18 may be a reliable CNN for detecting potatoes infected with blackleg disease in the field. Additionally, it is possible to anticipate an improvement in detection performance with larger datasets and data augmentation.
Assunção *et al.* (2020)	Detect peaches disease.	Datasets.	CNN	F1-score	The model successfully identifies all illness classes. These accomplishments demonstrate CNN's potential for identifying fruit illnesses with less training data. Additionally designed to function with portable devices is the model.
Azgomi *et al.* (2022)	Detect apples disease.	Digital camera.	MLP ANN	Accuracy	The two-layer structure with eight neurons in the first layer and eight neurons in the second layer, with a score of 73.7%, earned the highest accuracy.
Kerkech *et al.* (2020)	Detect grapevines disease.	UAV system with a RGB sensor.	CNN: SegNet.	Accuracy	The model created using RGB photographs outperformed the one created using infrared images. One of the limitations of the study is the small size of the training sample, which has a negative impact on the performance of deep learning segmentation.

2.8.2. Weed Detection

In addition to illness, plants that become weeds are considered a common threat to the production of food. These are plants that are seen to be undesirable in a certain location because they might rob crops of sunshine and water, harming the crops and the economy. Weed management has several challenges, including the fact that it is challenging to identify and separate them from crops. With fewer costs, side effects, and environmental issues, DL algorithms can enhance weed identification and classification. This technology might be used by robots that find and go after weeds (Alibabaei *et al.,* 2022)

Weed detection in individual plants

In accordance with Sujaritha *et al.* (2017), weeds have been identified in sugar cane fields using fuzzy real-time classifiers. A robotic prototype for weed identification was developed employing a Raspberry Pi and suitable input/output subsystems, including two separate cameras, small light sources, and motors with a power supply. There is a chance that during the robot's movement, field impediments might cause a deviation from the planned path. An automatic image classification system was constructed, that extracts leaf textures and employs a fuzzy real-time classification technique. The presented weed detection system consisted of the following steps: input of the field images, greenness identification, leaf texture extraction, computation of features, feature vector generation, and classification. Two processes make up the processing system. The first phase applies a colour extraction-based algorithm to photos taken by camera one, and the second process uses images taken by camera 2 to extract texture characteristics and use a Fuzzy Real-Time Classification Algorithm to detect weeds in crop rows.

The suggested robot prototype successfully recognizes the sugarcane crop among nine different weed types. With a detection accuracy of 92.9% and a processing time of 0.02 s, the system successfully detects weeds.

In Milioto *et al.* (2018) a modified encoder-decoder CNN was used in conjunction with data from a 4-channel RGB and NIR camera to produce a novel way for classificate crops and weeds. The number of convolutional layers was decreased to reduce time inference, and more vegetation indices (14 channels) were added to the input for more accurate identification. Three fields from a dataset were used. The networks were trained using three different sets of inputs, comprising 14 channels of vegetation indices, including RGB, ExG, Excess Red indice (ExR), and NDI, as well as RGB and NIR pictures and RGB and RGB and NIR images. In order to provide CNN with more inputs, the authors first compute a large number of vegetative indicators and alternative representations that are often employed in plant categorization. To give a per-pixel semantic categorization of the input data, the authors secondly developed their own semantic segmentation network.

The proposed model was trained on photos from a single field and tested on images from all fields to examine the generalization capability of the model. The model works better, according to the authors, when more channels are fed into CNN. The RGB channel network converges to the

final accuracy of 95%, 15% faster than the NIR channel network. The model's precision for weeds was 98.16%, accuracy for crops was 94.74%, and performance for crops was 95.09%. The method achieved recall rates of 94.17% for crops and 94.79% for weeds. The intersection over the union was 80.8%.

Lottes *et al.* (2018) created a sequential model encoder-decoder fully convolutional network (FCN) for weed identification in sugar beet fields. The dataset was obtained using a 4-channel RGB+NIR camera installed on a field robot called BoniRob. The recommended model was applied to the input photos to extract features. The sequential model employed 3D convolution to assess five shots in a series, creating a sequence code that was then used to learn sequential information about the weeds in the five photos. It was feasible to employ image sequences to implicitly represent local geometry thanks to an addition called the sequential module. This combination enhances generalization performance even if the visual characteristics or developmental stage of the plants alter between training and test times.

Based on FCN, Ma *et al.* (2019) created a SegNet image segmentation method for weeds and rice seedlings in the paddy field during the seedling stage. The model was then compared to one known as U-Net after that. RGB colour photographs of a rice paddy field with young plants were taken for this investigation. 20% of the samples were used as the test dataset, while about 80% of the samples were utilized as the training dataset. With the help of SegNet, which was developed using a symmetric structure for encoding and decoding, multiscale information may be extracted more precisely. This AI method can directly extract properties from the original RGB photographs as well as categorize and identify the pixels in paddy field images that correspond to the rice, background, and weeds. The major goal of this study is to evaluate how well the suggested method performs in comparison to a U-Net model.

The recommended method effectively identified pixels in pictures of weeds and unusually shaped rice seedlings seen in paddy areas. The FCN and U-Net techniques have average accuracy rates of 89.5% and 70.8%, respectively. Comparing the experimental results for the FCN based on SegNet and U-Net to the original and ground truth images are exhibited in Figure 14. Blue is used to represent rice, brown for weeds, and greyscale for the background.

In order to identify weeds in a soybean field, Ferreira *et al.* (2019) examine the performance of unsupervised deep clustering algorithms on actual weed datasets. Two current unsupervised deep clustering techniques (DeepCluster) are Deep Clustering for Unsupervised Learning of Visual Features and Joint Unsupervised Learning of Deep Representations and Image Clusters (JULE). The convolutional layer, batch normalisation layer, ReLU, and pooling layer are the layers that make up the JULE. In order to extract features for the DeepCluster model, which was built using AlexNet and Visual Group Geometry 16 (VGG) as a baseline, K-means was used as the clustering approach. In this study, two datasets were employed. The first one, known as the Grass-Broadleaf dataset, was captured in a soybean plantation in Brazil. In addition to 17,509 labelled images of eight nationally significant weed species native to eight distinct locations in northern Australia, the second dataset, DeepWeeds, has a larger class of negative cases.

JULE outperformed DeepCluster in terms of accuracy and NMI when the two clustering methods were compared. For the first dataset in JULE, the MNI and acc outcomes were 28% and 65. %, respectively, for 80 clusters. For 160 clusters in the second sample, the MNI and acc findings were 8% and 25.9%, respectively. On the other hand, for the first dataset, DeepCluster's MNI and ACC outcomes for 160 clusters were 0.41% and 87%, respectively. MNI and ACC outcomes for the second dataset were 26% and 51.6%, respectively, for 320 clusters.

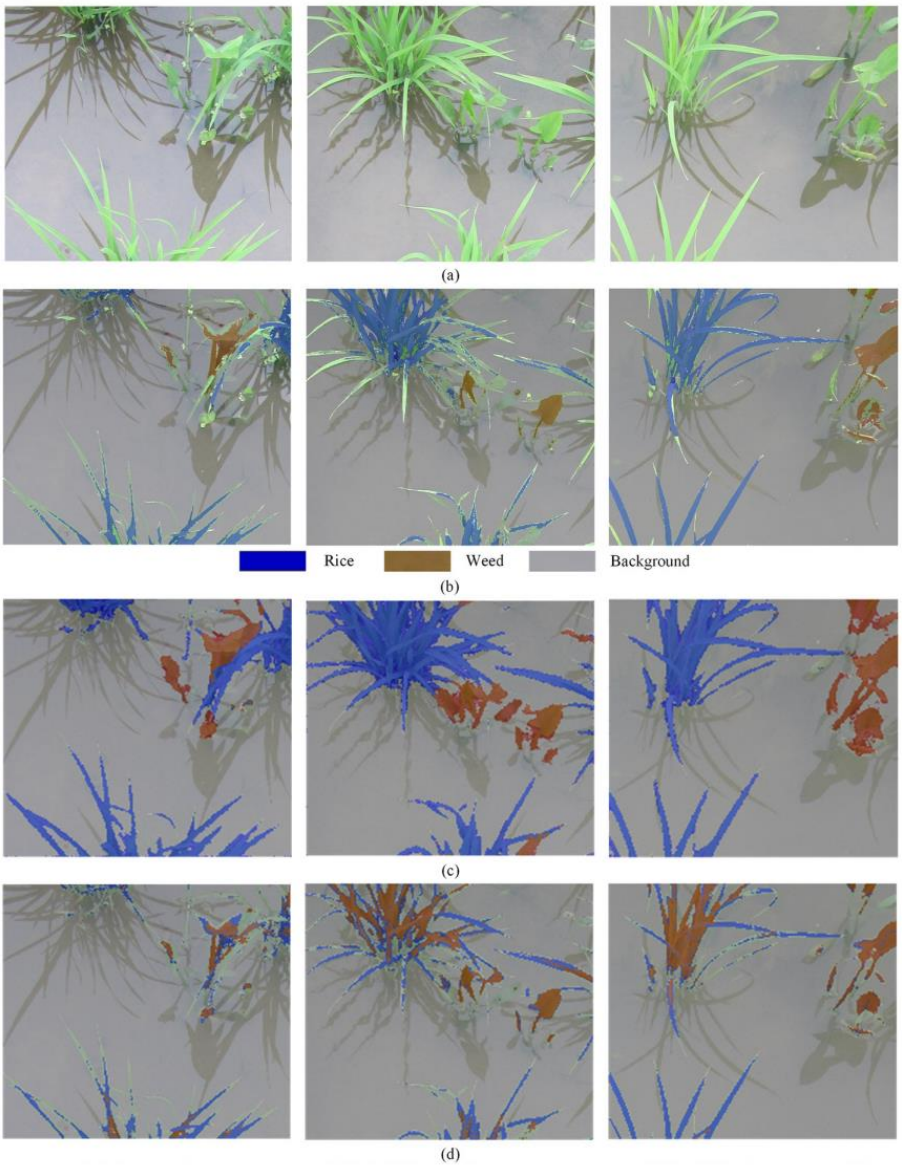


Figure 14. Results of the suggested approach. Original pictures (a), real-world images (b), FCN output (c), and U-Net output (d), (Adapted from Ma *et al.,* 2019).

In Wang *et al.* (2020), an encoder-decoder deep learning network was used to analyse pixel-wise semantic segmentation of crops and weeds. The two datasets for the study—oilseed and sugar beetroot—were gathered under various lighting conditions. To decrease the effects of the varied

lighting settings, three image enhancement techniques—Histogram Equalization (HE), Auto Contrast, and Deep Photo Enhancer—were investigated. A variety of input representations, including various colour space transformations and colour indices, were compared in order to enhance the input to the network. Colour spaces and vegetation indices such as NDI, NDVI, ExG, ExR, and excess green minus excess red (ExGR) were used to train the models. The results highlight the importance of NIR for successful segmentation in low-light situations, showing that although the presence of NIR information greatly increased segmentation accuracy, photos lacking NIR information did not enhance segmentation outcomes. The segmentation results for weed detection produced in this work by combining deep learning and picture enhancement methods were promising.

For the sugar beetroot dataset, the model trained using NIR images got a mean intersection over union (mIoU) of 87.13%. For the Oilseeds dataset, only RGB photos were used to train the models, and the image-based model outperformed the other models with a mIoU of 88.91%. It was the best with a 96.12% accuracy rate.

Kamath *et al.* (2022) paddy crop and two different types of weeds were segmented using semantic segmentation models PSPNet, UNet, and SegNet. The paddy field image collection is made up of pictures that were captured there using two different digital cameras and two different sources. The images were stored in RGB colour space. To keep solely the greenery, the background of earth and water was deleted. Two datasets were created; Dataset-1 had solely weed plants, whereas Dataset-2 contained paddy crop and weed images. A conventional CNN network serves as the foundation for semantic segmentation models. The notion of encoder-decoder structure is used in the image segmentation models that were developed for this study.

PSPNet developed a segmentation architecture based on the ResNet-50 base model. From the basis of the network, a feature map for PSPNet was generated. The scale of this feature map was then decreased. Convolution was applied to these pooled feature maps before feature maps were upscaled and concatenated. Segmented outputs are produced when a final convolution layer is used.

The segmentation architecture was built using the ResNet-50 base model, and the encoder-decoder framework employed by the UNet design comprises encoder-decoder layers that are symmetrical to one another. Skip connections, which are extra connections that connect sampling levels with later down sampling layers, were employed in this model. After down sampling, the segmentation boundaries are rebuilt with the use of skip connections, producing a more accurate output picture.

The VGG16 network and the encoder network used by the SegNet model are topologically identical. Each encoder layer has a matching decoder layer, and then each pixel receives class probabilities from a multi-class SoftMax classifier. There are no skip connections on SegNet.

Each pixel in pictures was identified with a classification from one of four categories using the playment.io programme: Background-0. Broadleaved weed-1, Sedges-2, and Paddy-3. The Keras, Tensorflow, and image-segmentation-keras APIs were used to implement the various models.

In terms of efficiency, PSPNet fared better than SegNet and UNet. The frequency weighted intersection over union (IoU) for PSPNet was 93%, and the mean IoU was 72%. In contrast, the mIoU for SegNet and UNet was 82% and 60%, respectively. The frequency weighted IoU was 74% and 38% at the end. Figure 15 shows the outcomes of the suggested model using PSPNet. The first row of pictures corresponds to the original photographs, the second row shows the projected output, and the last image is the actual ground truth image. The first line describes a paddy, and the second one describes a broadleaf weed. The third image shows a broadleaf weed (blue) and paddy (yellow) after that. The final one shows how sedge weed grows. Sedge weeds were challenging to identify, whereas paddy and broadleaved weeds were easy to recognize. The similarities between sedges and paddy may help to explain this loss.

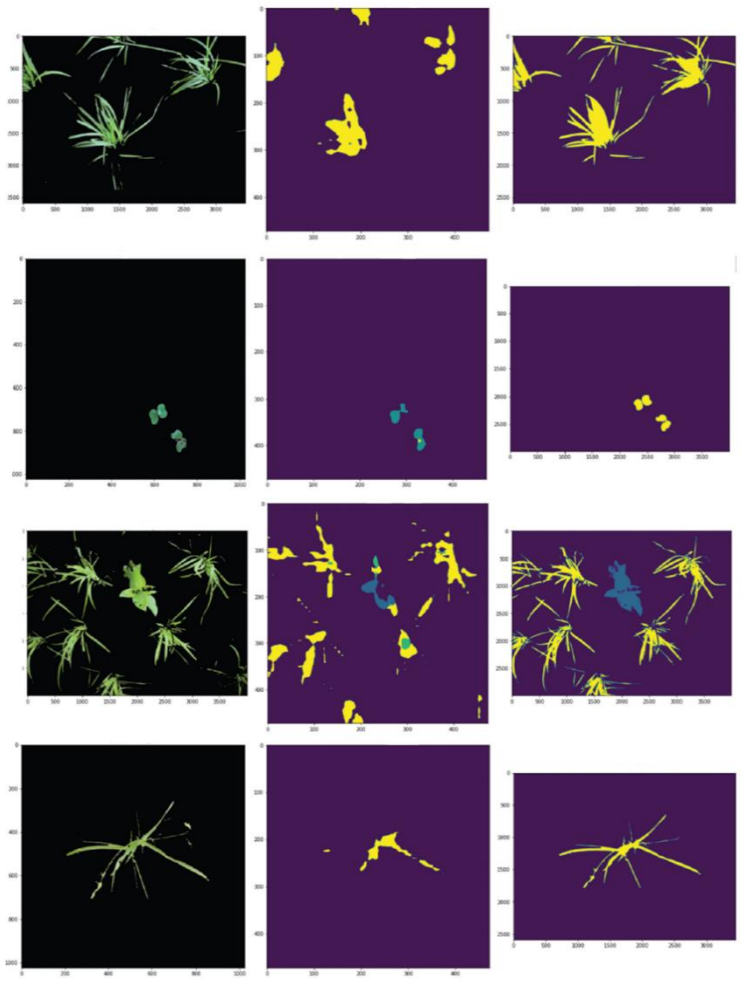


Figure 15. The first row of photos in the PSPNet results are the original photographs, the second row is the predicted output, and the last image is the actual image. The paddy is in the first line, followed by the broadleaved weed in the second, the paddy in the third, the broadleaved weed (blue), and the sedges weed in the fourth (Kamath *et al.,* 2022).

An application of a network model based on a Faster Region-based Convolutional Neural Network (R-CNN) for weed identification in photos of cropping regions was developed by Mu *et al.* in 2022. Additionally, the recognition accuracy of another model was enhanced by fusing the first model with the Feature Pyramid Network (FPN). The V2 Plant Seedlings dataset, which contains pictures taken in various weather conditions, was utilized for the images. The images were split and processed in order to separate the green component of the plant from the non-green parts. To separate the plants, the resultant greyscale images were converted into binary images using the Otsu approach. Following processing, distinct pictures of the plants were created. Convolutional features are shared using the Faster R-CNN deep learning network model, and feature extraction is carried out by merging the ResNeXt network with FPN in order to increase the model's weed identification detection accuracy.

The experimental findings demonstrate that by merging the ResNeXt feature extraction network with the FPN network, the Faster R-CNN-FPN deep network model achieves a higher recognition accuracy. Contrarily, the prototype using FPN recorded an accuracy of 95.61%, recall of 87.26%, F1-value of 91.24%, IoU of 93.7%, and detection time of 330 milliseconds (ms). The results from both models were successful. The model without FPN generated the following results for the same metrics: 92.4%, 85.2%, 88.65%, 89.6%, and 319 ms.

Assunção *et al.* (2022) investigated the effects on segmentation performance and inference time of optimizing the weed-specific semantic segmentation model at model DeeplabV3 with a MobileNetV2 backbone.

This essay is divided into three parts. In the first one, two datasets were used. The Crop Weed Field Image Dataset dataset (CWFID), which comprises crops (carrots) and weeds, was used to train and test the models. The Weeds dataset was employed in the real-time application idea, and Sony DSC-RX100M2 cameras were used to capture the photos. A handmade annotation (ground truth mask) was created for each pixel of each picture. Using the DeepLabV3 model, the fundamental model was trained on a desktop computer. A Jetson Nano edge device was then used for deployment.

By using crop and weed photographs for the model's training and validation, the second step of the process may be explained. In this study, tests were carried out utilizing the Tensorflow Model Garden (TMG) framework, the MobilenetV2 backbone, and the DeepLabV3 semantic segmentation model. The model was optimized both before and after training by selecting various model hyperparameters and using model quantization. The main objective is to take the supplied image's characteristics and extract them. To achieve the performance necessary for the application, the MobilinetV2 depth multiplier (DM) and output stride (OS) hyperparameters were modified (e.g., lightweight, and short inference time).

The analyses conducted in this study used DM values of 1.0 and 0.5. The ratio of the encoder's final output feature map size to that of the input picture size is known as the OS hyperparameter. Since this hyperparameter impacts segmentation accuracy and inference time, values of 8, 16, and 32 were chosen for OS to investigate the trade-off between accuracy and inference time.

TMG offered a fine-tuning approach to train the DeepLabV3 model's last layers using a fresh segmentation dataset. Pretrained models using the Cityscape, ADE20k, and PASCAL datasets served as the basis for DeepLabV3's training. A transfer learning technique was examined using these three datasets. During training, the TMG framework produced a number of checkpoint files.

Finally, the checkpoint files were converted into a frozen graph using a tool (script) that is part of the TMG framework. The frozen graph was finally changed (optimized) to run on the Tensorflow Real-Time engine using the TensorRT class converter.

The computational complexity of the models was measured in terms of floating-point operations (FLOPs), where addition and multiplication were both counted as one FLOP. The performance of pixel-wise segmentation was measured using the mIOU.

The semantic segmentation model was used in the final test, which was performed by the robotic orchard rover developed by (Veiros *et al.,* 2022). In this study, a system for spraying pesticides on weeds was used to assess the accuracy and feasibility of a computer-vision framework.

A Raspberry Pi v2 camera module with an 8-megapixel Sony IMX219 sensor was used to take video pictures. The three actuators that the Jetson Nano device controls are the herbicide container, a pressure motor, a DC motor that applies pressure to it, a manipulator motor, a stepper motor that moves the axis of the Cartesian manipulator, a nozzle relay, a relay that opens and closes the spray valve, and spray nozzle.

The result of the computer vision model (green dots) was a segmented image comprising the centroid of each weed (X, Y). These coordinates were converted to the manipulator's Cartesian manipulator coordinates (X, Y). The (X', Y') coordinates were then utilized by the control algorithm to identify the sequence of movements required to position the spray nozzle on the identified weed and execute the spray. To locate the reference point of the weeds for mechanism control, the Jetson Nano gadget used the computer-vision model. The nozzle was placed on a specific plant using a set of weed references, and the Cartesian manipulator then sprayed the spray.

The segmentation performance mIOU decreased by 14.7% while using a model hyperparameter DM of 0.5 and the TensorRT framework compared to a DM of 1.0 and no TensorRT, according to the study's findings for the second test. For OS = 8 and DM = 1.0. the model with the best segmentation performance has a mIOU of 75%. The model that performed the worst, with a mIOU of 64%, had a DM of 0.5 and an OS of 32.

In the CWFID and weeds dataset, the outcomes were also contrasted with the initial segmentation work. A mIOU of 75% was attained in the test with OS = 8 and DM = 1.0. and a mIOU of 64% was attained with OS = 32 and DM = 0.5.

Figure 16 shows the appropriate segmentation quality results. Variations in the segmentation performance (quality) for DM and OS were brought on by different hyperparameters.

Figure 17 shows the results of the application in real-time. In the upper-left corner is a segmentation that doesn't include plants. The input weed image is presented together with the segmentation results (output) that were related in the following images. The green dots in the

middle of each segmented area signify the center of gravity of the weeds. The results show how well and precisely the strategy may be applied. Given that the hyperparameters DM and OS allow for the management of the trade-off between segmentation accuracy and inference time, DeepLabV3 has demonstrated to be a model for segmentation tasks that is very adaptable. It was also accurate and useful to spray weeds in real-time. The video demonstration shows how the system successfully aimed the nozzle at all target weeds and applied the spray. This result illustrates how small models with great predicted accuracy may be improved.

Given that the hyperparameters OS and DM may be used to control the trade-off between segmentation accuracy and inference time, DeepLabV3 has shown to be an extraordinarily adaptable model for segmentation tasks. Additionally, successful results were obtained when the model was fine-tuned using the datasets from Cityscapes, PASCAL, and ADE20K. However, cityscapes produced considerably better outcomes.

Real-time weed spraying was also accurate and useful. The segmentation model built into the edge device provided all the information (position references) needed to control the mechanism. As seen in the video presentation, the system accurately aimed the nozzle at all target weeds and applied the spray. This result illustrates the potential for advancements in the development of small models with high projected accuracy.

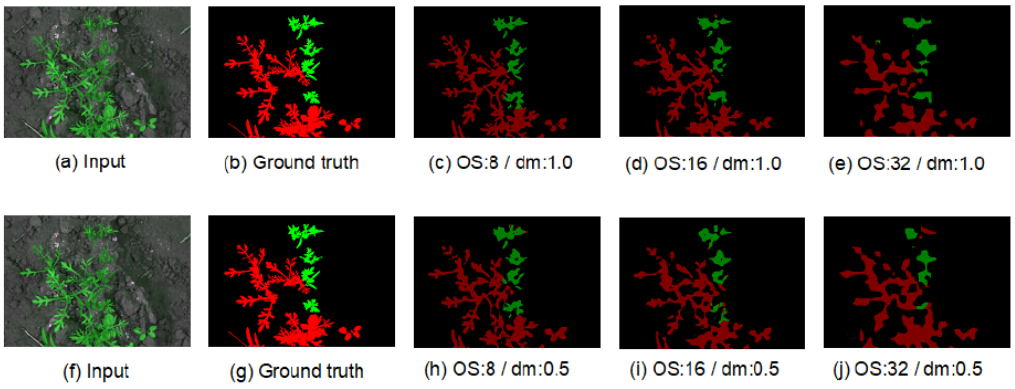


Figure 16. Results of the qualitative segmentation of the CWFID and weeds dataset. The labels of each sub image show the optimization that was performed to the model to achieve the segmentation, with the exclusion of input and ground truth (Assunção *et al.,* 2022).

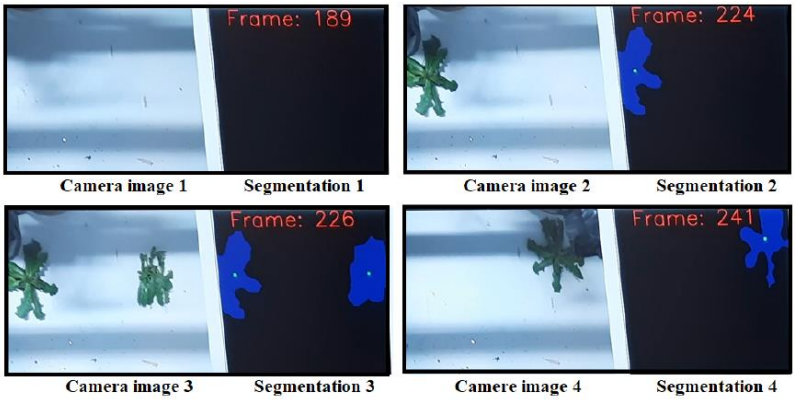


Figure 17. Application in real-time results in the segmentation of weeds (Assunção *et al.,* 2022).

Weed detection in areas of crops

Peña *et al.* (2015) created research to assess the effectiveness and limitations of early weed seeding identification using remotely sensed data obtained by an unmanned aerial vehicle equipped with visible and multispectral sensors. The objectives of the work included choosing the best sensor to enhance vegetation (crop and weed) and bare soil class discrimination as affected by the vegetation indices used, developing and testing an algorithm for crop and weed patch detection using object-based image analysis (OBIA), and selecting the best UAV flight configuration for the altitude, date of flight, and sensor type (visible-light vs. visible-light + near-infrared multispectral).

Two distinct cameras that were installed separately in a quadcopter were used to capture the remote photos. The pictures were captured on three dates, date 1 (44 days after seeding), date 2 (50 days after seeding) and date 3 (57 days after seeding). Four distinct heights were performed for each camera on each date: 40. 60. 80. and 100 meters (m).

The OBIA method coupled hierarchical relationships between analytic levels with object-based features including spectral values, location, and orientation. To properly recognize crop rows, the system was trained to categorize vegetation outside of crop rows as weeds utilizing a dynamic and auto-adaptive classification procedure.

The colour infrared images collected at 40m and on date 2 (50 days after sowing), when plants had 5–6 true leaves, showed the highest weed identification accuracy of up to 91%. At this flight level, the photos taken before date 2 performed noticeably better than the ones taken after. At higher flying altitudes, the multispectral camera performed better, but at lower altitudes, visible light cameras performed better. The errors, which are brought on by the higher heights, are the result of the spectrum mixing of sunflowers and bare soil components that occurred at the margins of the crop rows.

In Figure 18, some findings and comparisons are displayed. On-the-ground images are shown in line (A), manual categorization of observed data is shown in line (B), and image classification using the OBIA method is shown in line (C). The model was divided into four types, the number

of correct frames (1); underestimated weeds (2), frames with weed infestations in which the OBIA system spotted some weed plants but missed others; false negative (3), weed-infested frames in which no weeds were detected; and false positive (4), frames in which weeds were overestimated.

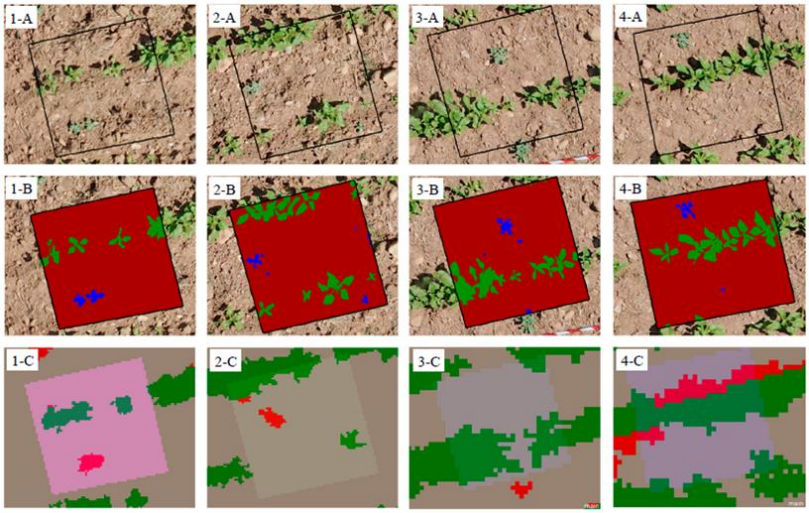


Figure 18. Four sampling frames' examples of results. Images taken on-location; manually classifying observed data; and classifying images using the OBIA approach are shown in (A), (B), and (C), respectively. (1) Correct categorization; (2) Underestimation of weeds; (3) Negative mistakes; and (4) False Positive mistakes (Peña *et al.,* 2015).

To identify weeds and rice crops, Huang *et al.* (2018) used UAV Phantom 4 pictures to accurately map the weed cover in rice fields. It was recommended to use the FCN approach to map the weeds in the gathered pictures. During the training stage, the FCN network receives the image-label combinations from the training set that match pixels-to-pixels. The output image is utilized to calculate the loss as an objective function together with the ground truth label when the network transforms the input image into an output image of the same size. The chain rule for derivatives is then used to determine the gradient of the objective function with respect to the weights of each network module, and the gradient descent algorithm is used to modify the network parameters. The training technique outlined above will be continued until the maximum number of iterations is reached or until the loss is below a threshold, whichever comes first. Until the loss is below a certain level, or the allotted number of iterations has been met, the training phase will continue. The trained FCN network will map the validation picture into a prediction class map with per-pixel classification during the validation step.

The results of the experiments showed that the FCN technology performed quite well. The system's overall accuracy was 93.5%, its weed detection accuracy was 88.3%, and it had an IoU of 75.2%, showing that it can produce accurate weed cover maps for the UAV pictures under examination.

Figure 19 allows you to see the FCN outcomes using several pre-trained CNNs. Real UAV photos are shown in (a), the ground truth representation is shown in (b), and results from FCN-AlexNet, FCN-with VGG16, and FCN-GoogLeNet, respectively, are shown in images from (c) to (e).

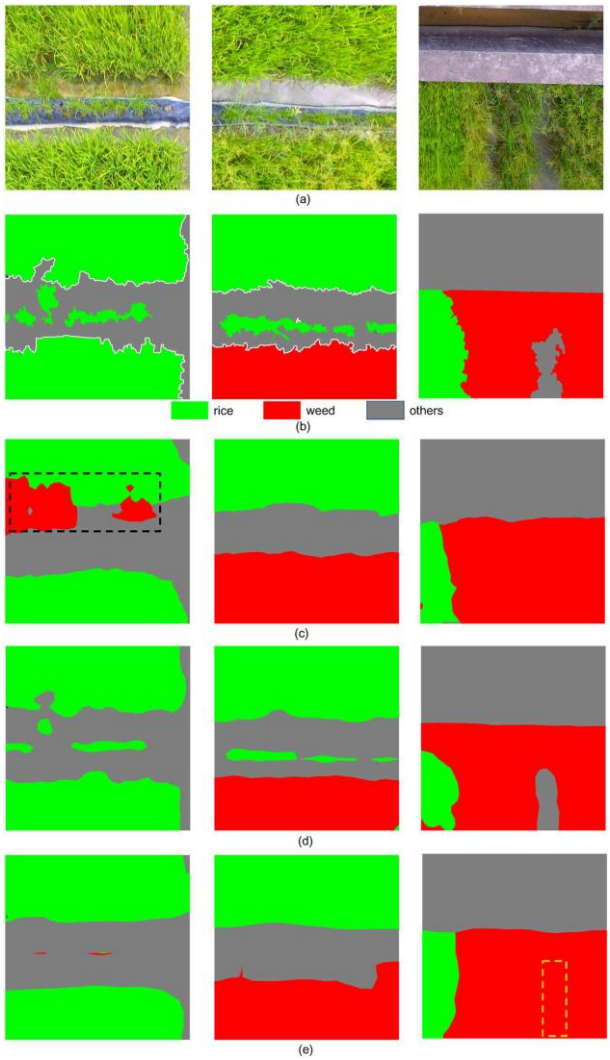


Figure 19. FCN classification results using several pre-trained CNNs. Actual UAV picture, ground truth results, and results from FCN-AlexNet, FCN-VGG16, and FCN-GoogleNet are shown in (a), (b), and (c), respectively (Huang *et al.*, 2018).

By combining CNNs with a variety of unsupervised training datasets, Bah *et al.* (2018) created a fully automated learning system for weed detection in bean and spinach fields using UAV pictures. This method, which makes use of k-mean clustering, results in super-pixels. The recommended technique has three main phases. The first level enables inter-row weed identification so that crop rows may be automatically recognized. They believed that weeds were plants that grew between the rows of crops and that crops were placed in neat rows. Following the application of the Hough transform to establish the rate of plant rows on the skeleton, a marker

was created, and the plant rows were defined using the simple linear iterative clustering (SLIC) technique. K-mean clustering was used in this method to create superpixels. The training dataset then consists of inter-row weeds. In the last phase, a CNN model that can identify weeds and crops in pictures was developed. Using the unsupervised training dataset, ResNet18 was trained to recognize weeds in images.

To compare the performance of the model, SVM and random forest (RF) models were also employed. SVM and RF are outperformed overall by ResNet18 in both supervised and unsupervised learning methods. The model's accuracy was 94.5% and its kappa value was 0.912.

Figure 20 shows two examples of picture categorization in bean fields at the bottom and spinach fields at the top. On the left, you can see the samples that were obtained using a sliding window without a crop line or any background features. The plants are either crops, weeds, or ambiguous options, as indicated by the blue, red, and white dots, respectively. On the right, the backdrop details are shown, along with the weeds that were detected after applying the crop line, highlighted in red.

The results show that the encoder-decoder with a sequential model increases the module's F1-score by about 11–14% when compared to the encoder-decoder FCN. With an F1-score of 92.3%, the proposed model beat encoder-decoder FCN without a sequential model.

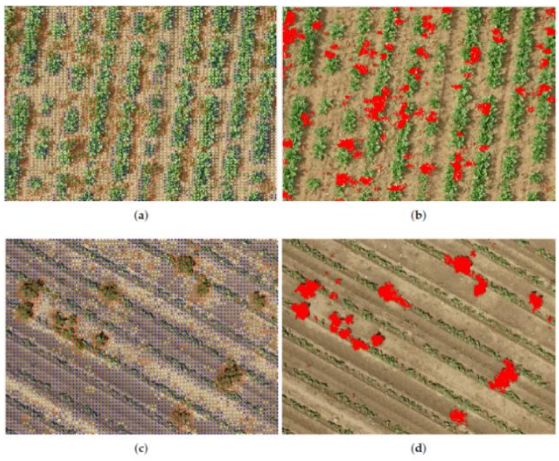


Figure 20. On the top, there are several examples of UAV picture classification using models made from unsupervised data in spinach fields, and bean fields at the bottom. In blue there are crops, in red are weeds, and in white uncertain decision (Bah *et al.,* 2018).

According to Osorio *et al.* (2020), deep learning image processing and multispectral pictures taken by a drone formed the foundation for three weed estimate approaches that were developed. An NDVI index was used in conjunction with these techniques. The first method is based on SVM and employs histograms of oriented gradients (HOG) as a feature descriptor. An item is represented as a histogram of directed gradients based on the magnitude and direction of the gradient from a specific set of pixel blocks. This sort of trait makes it feasible to pinpoint the exact

geometric shape of a plant. A mask that is produced by NDVI hides the ground and other features that are unrelated to vegetation. Histograms of oriented gradients are utilized to obtain these objects' attributes, which are subsequently fed into a support vector machine that has previously been trained. Using the SVM, it can be determined whether the discovered objects belong to the lettuce class.

For object recognition in the second one, CNN based on You Only Live Once version 3 (YOLOv3) was employed. After the model has been trained to detect the crop, an algorithm uses bounding box coordinates from the model to delete crop samples from the image. The image is then binarized using a green filter, making the pixels that lack vegetation black and the pixels that the green filter accept white. To make it easier to determine the proportion of weeds in each photograph, vegetation that does not match the crop is highlighted.

The last one used CNN with masks to get instance segmentation for each crop. RCNN uses the "selective search for object recognition" technique to extract 2000 regions from the image. They give information to a convolutional neural network, in this case, Inception V2, which extracts traits that an SVM uses to place the object in the proper category.

The techniques' F1-scores for crop detection were 88%, 94%, and 94%, respectively, based on the criteria that were employed. It was 79%, 89%, and 89% accurate. It was 83%, 98%, and 91% sensitive. There was 0%, 91%, and 98% specificity. Lastly, the accuracy was 95%, 91%, and 94%.

In order to identify weeds in crops, Islam *et al.* (2021) employed three different types of approaches: RF, K-nearest neighbours' algorithm (KNN), and SVM. An Australian chilli farm provided the RGB camera that was used to capture the photos by a UAV, which were subsequently pre-processed utilizing image processing techniques. The photos underwent pre-processing to produce Ortho mosaic images. The authors determined vegetation indicators such as the normalized red band, normalized green band, and normalized blue band by extracting the reflectance of the red, green, and blue bands. After being extracted and chosen, image features were then applied to classification models. MATLAB was used to extract the features from the pre-processed images as well as to simulate machine learning-based techniques.

The experimental findings show that RF outperformed the other classifiers. The efficiency of RF and SVM as classifiers for weed detection from UAV photos is so obvious. For RF, KNN, and SVM, the accuracy scores were 96%, 62.8%, and 94%, respectively. Recall and specificity were 95.1% and 88.7%, 62.1% and 81.9%, and 91.1% and 89% with RF, KNN, and SVM, respectively. For RF, KNN, and SVM, respectively, the accuracy, FPR, and kappa coefficient were 94.9%, 5.7%, and 87.8%; 62.4%, 18%, and 36.4%; and 90.8%, 8%, and 82.5%.

As previously, Table 3 gathers an overview of research work published in the "Weed Detection" field. The table includes the reference, the application, the data, model and metric applied. The table also includes a column describing the model performance.

Table 3. Overview of research works in the "Weed Detection" field.

REFERENCES	Application	Data Used	Model Used	Metric Used	Model Performance
Sujaritha *et al.* (2017)	Weeds in sugar cane fields detection.	Digital cameras.	Fuzzy real-time classifier.	Accuracy	The prototype made a more accurate distinction between nine different weed species in less time. The robot was prompted to deviate from its intended course while it was travelling by the field impediments.
Milioto *et al.* (2018)	Crop-weed sugar beet detection (VIs added to the input).	RGB and NIR camera.	Mask R-CNN.	Accuracy IoU Precision Recall	The model's performance increased by adding a new channel to its input. The RGB channel network converges to final accuracy of 95% 15% faster than the NIR channel network.
Lottes *et al.* (2018)	Crop- weed detection in the sugar field.	RGB and NIR camera.	Encoder–Decoder FCN: DenseNet.	F1-score	With an F1-score of 92.3%, the proposed model beat encoder-decoder FCN without a sequential model, random forests, and vanilla FCN in accurately identifying crops in all growth phases.
Ma *et al.* (2019)	Detection of weeds and rice seedlings.	RGB camera.	FCN: SegNet.	Accuracy	The recommended method effectively identified pixels in pictures of weeds and unusually shaped rice seedlings seen in paddy areas. The SegNet method led to a classification with excellent accuracy.
Ferreira *et al.* (2019)	Detection of weeds in a soybean field (Unsupervised clustering).	Datasets.	JULE. DeepCluster.	Accuracy NMI	JULE and DeepCluster both performed better than expected. The findings from these datasets suggest that clustering and unsupervised learning may be successfully applied to problems in agriculture.

Wang *et al.* (2020)	Detection of crops of sugar, weeds, and oilseeds.	Datasets.	Encoder-decoder CNN.	Accuracy IoU	The results demonstrate the use of NIR data for precise segmentation in low-light conditions. The addition of NIR data significantly improved segmentation accuracy.
Kamath *et al.* (2022)	Weed detection in paddy crops.	Digital camera.	Semantic segmentation models: PSPNet, UNet, SegNet.	IoU	PSPNet outperformed SegNet and UNet in terms of efficiency. The mean IoU is between 70% and 80%, while the frequency weighted IoU is between 80% and 90%.
Mu *et al.* (2022)	Weed detection in maize, sugar beet, and wheat crops.	Dataset.	FPN. Faster R-CNN: ResNeXt.	Accuracy Recall F1-Score IoU	The experimental results show that the Faster R-CNN-FPN deep network model obtains a greater recognition accuracy by combining the ResNeXt feature extraction network with the FPN network.
Assunção *et al.* (2022)	Weed detection.	Datasets.	TMG. DeepLabV3. MobilenetV2.	mIoU	This result illustrates the potential for advancements in the development of small models with high projected accuracy. Given that the hyperparameters OS and DM may be used to control the trade-off between segmentation accuracy and inference time, DeepLabV3 has shown to be an extraordinarily adaptable model for segmentation tasks.
Peña *et al.* (2015)	Weed seedling in sunflowers field detection.	Visible-light and multispectral cameras in a UAV.	OBIA.	Accuracy	At lower flight altitudes, the visible light camera performed better, but at higher altitudes, the multispectral camera showed to be more accurate. In the higher elevations, several errors were brought on by the spectrum mixing of the floral and bare soil components.
Huang *et al.* (2018)	Weed cover maps to detect weeds and crops in rice fields.	UAV with a digital camera.	FCN.	Accuracy IoU	In terms of efficiency and accuracy for weed detection, the FCN technology performed well. Being a supervised algorithm, FCN requires a significant amount of manual labelling work since it requires a large number of tagged images for training and updating.

Bah *et al.* (2018)	Weed detection in bean and spinach fields.	Drone with a digital camera.	CNN: Resnet18.	Accuracy	Unsupervised labelling may be a superior option for weed detection given the variations in labelling accuracy between supervised and unsupervised approaches, especially when crop rows are spaced widely apart.
Osorio *et al.* (2020)	Weed detection in lettuce crops (VIs added to the input).	Mavic Pro with a multispectral camera.	SVM+ HOG; Mask R-CNN; YOLOv3.	F1-Score Accuracy Precision Recall Specificity	Given that it needs less processing power and has been demonstrated to operate effectively, the HOG-SVM technique is a fantastic option for IoT devices. The YOLO technique overestimates the high values of marijuana coverage in comparison to the other two.
Islam *et al.* (2022)	Weed detection in chilli crops.	RGB camera coupled in a UAV	RF. KNN. SVM.	Accuracy Recall Precision FPR k	According to the experimental findings, RF performed better than the other classifiers. RF and SVM are effective classifiers for weed detection from UAV images.

2.8.3. Weed Classification

A crucial component of agricultural management is the categorization of species, including those of insects, birds, and plants. The conventional human method of species identification takes time and specific knowledge. Real-world data may be analysed using deep learning to produce speedier, more precise answers (Alibabaei *et al.,* 2022).

Weed classification in individual plants

To identify plant species in coloured photos, a convolutional neural network was built, according to Dyrmann *et al.* (2016). The photos are from six distinct datasets: Dyrmann and Christiansen (2014), Robo Weed Support (2015), From Kim Andersen and Henrik Midtiby, Sgaard (2005), Scharr, Minervini, Fischbach, and Tsaftaris (2014), and Aarhus University - Department of Agroecology and SEGES (2015). The six datasets comprise all images captured during lightning-controlled events as well as mobile-device images captured in the field under various lightning conditions. The network was built from scratch and trained and tested on a total of 10,413 samples of 22 weed and crop species in the early stages of growth.

The training set, which was divided into two sets—one for training and the other for testing—contained more than 60% of the total number of images. The training set was then multiplied by eight and the images were then mirrored and rotated 90 degrees. A simple excessive green segmentation was used to locate green pixels. Any non-green pixels were then removed before the images were uploaded to the network. The network used 128 x 128 RGB pictures as inputs. Following that, batch normalisation was used to ensure that the inputs to layers always fell within the same range, even when the preceding layers changed. Non-linear decision boundaries are added by the network's activation function, ReLU. The next step is max pooling, a technique that reduces the spatial length of a feature map while granting the network translation invariance. In this study, network stacking was chosen after considering the filtering and coverage capabilities of the network.

The training was ended after 18 epochs to get the maximum accuracy feasible without over-fitting the network. The categorization accuracy of the network ranged from 33% to 98%, with an average accuracy of 86.2%. Thale Cress (*A. thaliana*), Sugar Beet (*B. vulgaris*), and Barley (*H. vulgare L.*) were commonly correctly identified, with accuracy rates of 98%, 98%, and 97%, respectively. While *Veronica*, Field Pancy (*Viola arvensis*), and Broadleaved Grasses (*Poaceae*) were regularly misclassified. Of these three species, only 46%, 33%, and 50% were properly classified. Overall, the classes with the most species had the best categorization accuracy. As a result, classes with fewer photo samples suffered a smaller total loss.

Andrea *et al.* (2017), presented the development of an algorithm capable of image segmentation and classification. It applies a CNN to distinguish maize plants from weeds in real time. This discrimination was performed using four types of CNN. Models namely, LeNet, AlexNet, cNet, and sNet. A multispectral camera was used to acquire RGB and NIR images for the purpose of segmentation and classification. The photos were initially processed digitally, and the

green channel of each one was normalised. By removing light and shadow from the photos, was done to enhance the detection of green colours. Then, to distinguish the plants from the soil and other non-plant features, the green colour was removed from the photos to create a grayscale equivalent. The CNN was trained using a dataset produced during the segmentation step. While classification strives to determine which photos fall into the two stated groups, segmentation aims to separate the target plant from the original image. After being chosen, the identical dataset and solver of type Adam were used to train each of the four CNN models.

The most effective algorithms have a lot of potential for weed and plant classification in real-time autonomous systems. The cNET with 16 filters was the network that delivered the best outcomes. It utilised a dataset of 44580 segmented images from both classes and obtained a training accuracy of 97.23%.

A hyperspectral NIR snapshot camera was suggested by Gao *et al.* (2018) for classifying weeds and maize by measuring the spectral reflectance of an area of interest (ROI). This study set out to determine the practicality of classifying weeds and maize using a NIR snapshot mosaic hyperspectral camera, pinpoint the critical spectral wavelengths and characteristics for classification, and offer the most effective inputs for building RF models. a machine learning method that recognises weeds and crops using the best hyperparameters. Approaches from the disciplines of feature engineering, machine learning, and image processing were investigated in order to construct the best classification model for the three different types of weeds and maize. In this study, 185 characteristics were found using the NDVI and RVI vegetation indices.

The first step of this study was the image acquisition. Corn and weed seeds were planted in pots filled with sandy soil. The images were recorded in a laboratory. Then, the images were pre-processed, and the calibration formula was applied, the reflectance was determined as band features. Following the creation of features, algorithms for feature selection were used to extract distinguishing characteristics. Finally, different types of features were combined as model input for RF.

The findings demonstrated that for the crop Zea mays, the weeds *Convolvulus arvensis, Rumex,* and *Cirsium arvense*, the best random forest model with 30 significant spectral characteristics can reach recalls of 100%, 78.9%, 69.1%, and 75.2%, respectively. The ability to identify Z. mays with 100% recall (sensitivity) and 94% accuracy (positive predictive values) has been proven. For the crop *Zea mays* and the weeds *Convolvulus arvensis*, *Rumex*, and *Cirsium arvense*, the model achieved accuracy and F1-Score of 94%, 96.9%, 95.9%, 86.6%, 70.3%, and 69.7%, respectively.

Bakhshipour & Jafari (2018), classified sugar beet crop and four types of weeds were performed using an SVM and an ANN classifier using shape features. Pictures were captured by using a weed robot with a camera, providing RGB images. The detection of the green areas inside the pictures and the creation of primary binary images were accomplished by using appropriate thresholds on the RGB colour spaces of ExG and ExGR, as well as the green colour difference. Fourier descriptors were retrieved from pictures and assessed as boundary-based features, together with shape factors and moment invariants as region-based features. Feed-forward

perceptron with several layers, two hidden layers and the LMA back-propagation learning technique were used to generate the ANN. As a feature selection technique, PCA was used to condense the initial 31 feature expressions into four components. The SVM program then used the PCA values.

After utilizing the ExGR approach to segment the pictures, area thresholding was used to eliminate very minute things like noise and microscopic plants. These overlapping sections were removed, and the plants were separated by performing a series of erosion and dilation techniques to the binary pictures. The form features of the remaining items were retrieved once they were labelled. The stored trained classifier was then given the extracted shape data to determine if the item was a sugar beet or weed. This process is designed for a shape-based weed detection (SBWD) algorithm.

The accuracy of classification for the sugar plants was 93.33% for ANN and 96.67% for SVM, respectively. The weeds were successfully recognised by ANN and SVM 92.50% and 93.33% of the time, respectively, in comparison to the sugar beetroot crop. With an overall accuracy of 92.92% and 95%, respectively, both ANN and SVM were able to recognise the shape-based patterns and categorise the weeds quite well.

The results of the SBWD algorithm at various levels are illustrated in Figure 21. In (a), the initial RGB image is shown; in (b), plants are segmented using the ExG method; in (c), the image created using morphological techniques (noise removal, area thresholding for removing small plants, and edge erosion for removing touching overlaps) is shown; in (d), sugar beets are segmented using the SBWD algorithm; in (e), the result of subtracting the greenness from image (c) from image (b) showing the weeds; finally, the SBWD algorithm's output in (f) depicts weeds, sugar beet plants, and false negatives. Red pixels denote weeds, green pixels reveal sugar beet plants, and yellow pixels depict areas that were mistakenly classified as undesirable things.

Sa *et al.* (2018) classified sugar beetroot and weed species using a CNN and multispectral data gathered by an MAV. SegNet format conversion was performed on these photos. The data collected from this field was separated into images showing only crops, solely weeds, or both crops and weeds. After that, vegetation in the homogenous imaging set is automatically differentiated from other objects by eliminating NDVI from multispectral pictures and utilising image processing for model training. After that, SegNet received the tagged photos. Depending on the training dataset, the frequency of appearance (FoA) for each class is changed to improve class balance. The authors trained six different models with different input channel sizes and training conditions, evaluated them quantitatively with F1-scores and AUC as measures, and compared the outcomes.

The maximum number of iterations was 640 epochs, with the learning rate for the training model set to 0.001, the batch size being 6, the weight delay rate being 0.005, and the batch size being 6. Using the test data, this model was able to provide an average F1-score of 80% and an average accuracy of 80%. However, due to restrictions in the dataset our model was trained on, spatiotemporal discrepancies were discovered.

A sample of a seven-frame result is shown in Figure 22. The NIR, Red, and NDVI pictures are the first three columns' input data to CNN in that sequence. The fifth column contains our probability output for the proposed model, where red denotes a crop, green denotes weeds, and blue is the background. The fourth column contains annotated ground truth.

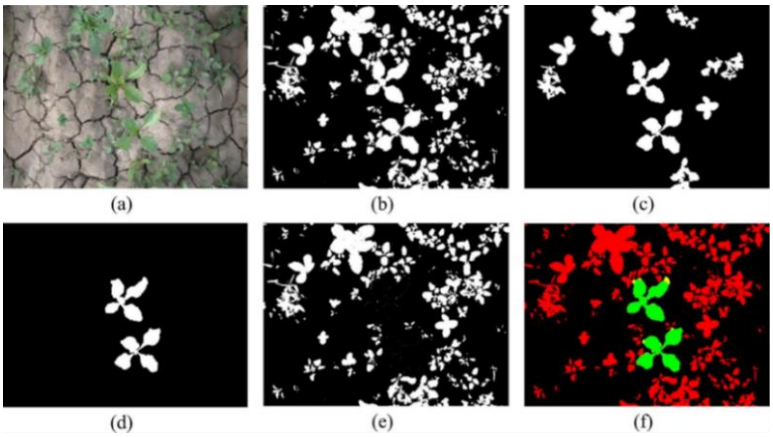


Figure 21. Results of the SBWD algorithm's : (a) The initial RGB image; (b) The segmented plants obtained using the ExG method; (c) The outcome of the morphological filtering of small objects; (d) The segmented sugar beets obtained using the SBWD algorithm; (e) The weeds removed from the segmented sugar beets obtained using the SBWD algorithm; and (f) The result of the SBWD algorithm displaying the false negatives, weeds, and sugar beets (Bakhshipour & Jafari, 2018).

Yang *et al.* (2018) investigated DL methods for classifying hyperspectral images. Particularly, the authors created and improved four DL models. a CNN in two dimensions (2-D-CNN). a CNN in three dimensions (3-D-CNN). a 2-dimensional CNN based on regions (R-2-D-CNN). And a 3-D CNN with a regional focus (R-3-D-CNN). Regarding the hyperspectral images retrieved from six datasets—Indian Pines Scene, Botswana Scene, Salinas Scene, Pavia Centre Scene, Pavia University Scene, and Kennedy Space Center—the goal was to demonstrate that a 2-D-CNN operated in the spatial context and a 3-D-CNN operated in both spectral and spatial factors.

The steps of the 2-D-CNN model include patch extraction, feature extraction, and label identification. The main difference is that the 3-D-CNN model includes an extra phase for reordering. In this step, the D hyperspectral bands are reorganised in ascending order. The R-2-D-CNN model fuses a large number of shrinking patches into multilevel instances, which are subsequently utilised to generate predictions. The main difference between the two is that the former employs 3-D convolution operators while the earlier employs their 2-D equivalents.

Since both have an effect on the class label prediction of a pixel, an efficient hyperspectral image classification algorithm should take both the spectral factor and the spatial component into account. With this knowledge, the R-2-D-CNN and the R-3-D-CNN, two suggested deep learning models, produced improved outcomes. The first network produced the best results in one of the

datasets, scoring 99.67% and 99.89%, respectively, for the average accuracy of each class and the total accuracy of all classes. For the same measure, the top outcomes in the second model were 99.87% and 99.97%.

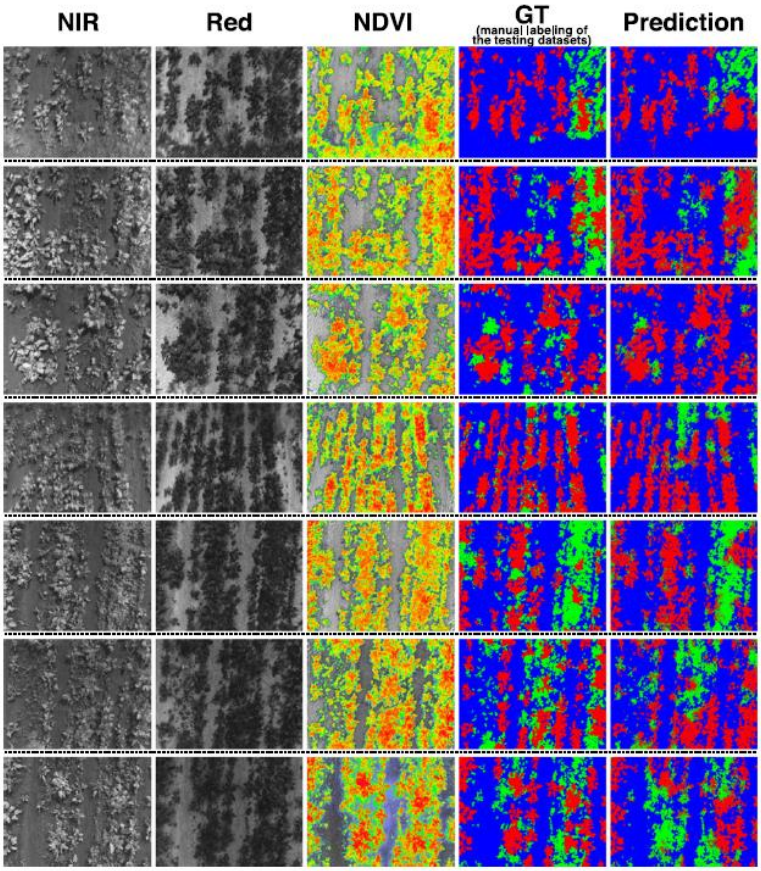


Figure 22. The seven frames' qualitative outcomes (row-wise). The first three columns contain input data for CNN, whereas the fourth and fifth columns display ground facts and probability forecasts of the suggested model (Sa *et al.*, 2018).

Yashwanth *et al.* (2020) used the Deep Learning function to develop the picture classification technique. Python has made use of the KERAS API in conjunction with the Tensorflow backend. Images of 9 different crops and their respective weeds have been collected (wheat-*Parthenium; Soybean-Amaranthus Spinosus; Maize-Dactyloctenium Aegyptium; Brinjal-Datura Fatuosa; Castor-Portulaca Oleracea; Sunflower-Cyperus Rotundus; Sugarcane-Convolvulus Arvensis; Paddy-Chloris Barbata; Paddy-Echinochloa colona*. The initial stage involves pre-processing the pictures that will be used to train the neural network. The next step is to acquire the image's rectified feature map using the ReLU activation function. Edge detection is accomplished via pooling. After applying this flatten function, the matrix is flattened. This feeding goes to the thick layer. A fully connected layer can identify the item in the picture.

The model was tested using nine different types of crops and the corresponding weeds, and the highest accuracy was found to be 96.3%. It was used 250 images of each plant, acquired in the field. All the provided photos were correctly categorized as either plants or weeds.

Jin *et al.* (2021) developed a weed identification algorithm based on deep learning and image processing for robotic weed removal in the vegetable plantation. Images were captured in the field using a digital camera. The dataset was expanded after the gathered photographs underwent pre-processing for colour, brightness, rotation, and image definition. Bounding boxes were drawn on the veggie in the input photos a manual annotation. In CenterNet, each item is represented by a single point, and object centres are predicted using a heatmap. Estimated centres are obtained from the heatmap's peak values using a Gaussian kernel and an FCN. Each ground truth key point is converted to a smaller key-point heatmap using a Gaussian kernel and focal loss to train the network. The remaining green items that continued to fall out of the surrounding boxes were then classified as weeds. To extract weeds from the background, a colour index was determined and evaluated through Genetic algorithms (GA) according to Bayesian classification error.

The trained CenterNet achieved a precision of 95.6%, a recall of 95.0 and a *F*1-score of 95.3%.

Figure 23 shows the results of the detection of vegetables under different conditions using the CenterNet, and results with the final segmentation with vegetable regions marked with a red box.

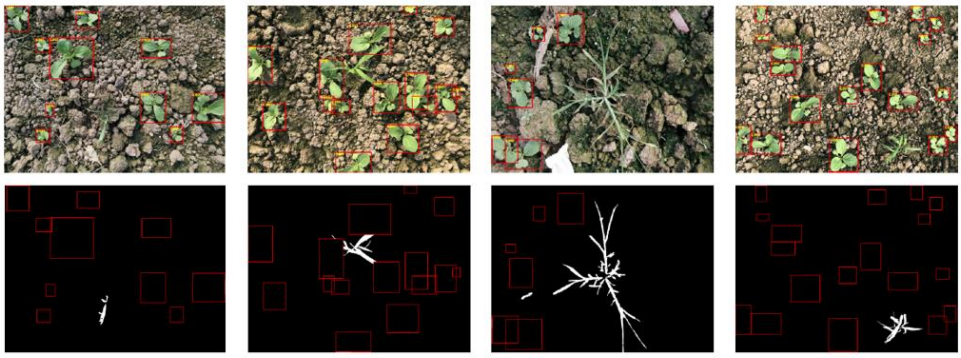


Figure 23. The results of vegetable detection using the CenterNet detection model under varied situations are shown in the first line. The final segmentation findings with vegetable sections highlighted in red boxes are shown in the second line. (Jin *et al.,* 2021).

A new approach based on metaheuristic optimisation and machine learning was put out by El-Kenawy *et al.* (2022) with the goal of classifying weeds using wheat photos taken by a drone. Three models—namely, ANN, SVM, and KNN—were suggested. Through feature extraction and transfer learning, a publically available dataset was used to train the ANN. A binary optimizer is further recommended by AlexNet to enhance the feature selection process and find the ideal combination of characteristics. To analyse the performance of the recommended approach, a set of assessment criteria is employed to gauge the effectiveness of the feature selection algorithm. Two further distinct types of machine learning models, namely SVM and KNN, were used in the suggested model to enhance the parameters. A novel optimisation strategy that combines grey wolf (GWO) and sine cosine optimizers (SCA) enhances this classifier. These proposed classifiers assist in the development of a hybrid algorithm.

With a detection accuracy of 97.70%, an F1-score of 98.60%, a specificity of 95.20 %, and a sensitivity of 98.40 %, the findings showed that the suggested technique outperforms other options and increases classification accuracy.

G C *et al.* (2022) studied the performance of a deep learning model for weed detection in photos with potting mix (non-uniform) and black pebbles (uniform) backgrounds. Four canon digital cameras were used to capture the weed and crop shots, namely, Horseweed, Palmer Amaranth, Redroot Pigweed, Ragweed, Waterhemp, and Kochia and crop species of Canola and Sugar beets. Weed classification models were developed using deep learning architectures, namely, CNN based in ResNet50, and VGG16. The process of weed and crop species extraction from a single picture followed the acquisition of the photos. By creating a directory for each species and moving the cropped photographs there using a Python script, the annotated images were created. For the 3488 photographs with uniform background datasets and the 2868 images with non-uniform background datasets, two folders were made. The non-uniform background scenario data, uniform background scenario data, and combined datasets scenarios created after combining both scenarios' data were trained using the ResNet50, and VGG16.

The VGG16 and ResNet50 models, created from non-uniform background photos performed well on the uniform background, with an average f1-score of 82.75% and 75%, respectively. On the other hand, the performance of the VGG16 and ResNet50 models, which were created from uniform backdrop pictures was performed not as well on non-uniform background images, with average f1-scores of 77.5% and 68.4%, respectively. A model trained with fused datasets from two background situations was able to obtain the f1-score value of 92% to 99%.

Zhang *et al.* (2022) compared the classification models of SVM and deep learning-based VGG16 utilizing RGB picture texture information to categorize weeds and crop species. The SVM and VGG16 deep learning classifiers were used to classify four weeds (horseweed, kochia, ragweed, and waterhemp) and six crop species (black bean, canola, corn, flax, soybean, and sugar beets). The greenhouse yielded a total of 3792 RGB photos of crop and weed samples, including 1521 images of crops and 2271 images of weeds. Therefore, the green portion of the picture was extracted using image processing techniques. Using pre-processing techniques, a picture was turned into a grayscale image. Local binary pattern (LBP) features and Gray-level co-occurrence matrix (GLCM) features are two different categories of texture characteristics that were retrieved from the grayscale picture. The next phase is the feature selection using the ReliefF algorithm. After this, a machine learning classifier was built by using an SVM and VGG16.

The VGG16 model classifiers had outperformed all the SVM model classifiers. The results demonstrated that the average F1 results of the VGG16 model classifier ranged from 93% to 97.5%. While the average F1-score results of SVM ranged from 83% to 94%. In the VGG16 Weeds-Corn classifier, the corn class achieved a F1-score value of 100%.

Table 4 gathers an overview of research works published in the "Weed Classification" field. The table includes the reference, the application, the data, model and metric applied. The table also includes a column describing the model performance.

Table 4. Overview of research works in the "Weed Classification" field.

REFERENCES	Application	Data Used	Model Used	Metric Used	Model Performance
Dyrmann *et al.* (2016)	Weed classification in 22 different crops.	Dataset.	CNN.	Accuracy	Overall, the classification accuracy was highest in the classes with the most species. Classes with fewer photo samples had a lesser overall loss as a result. The classification accuracy of the network ranged from 33% up to 98% with an average accuracy of 86.2%.
Andrea *et al.* (2017)	Image segmentation and classification of weeds in maize fields.	Multispectral camera to acquire RGB and NIR images.	CNN: LeNet, AlexNet, cNet, and sNet.	Accuracy	The network cNET offered the best training results due to its accuracy and processing speed.
Gao *et al.* (2018)	Weed and maize classification (VIs added to the input).	Hyperspectral snapshot camera sensor.	RF.	Precision Recall F1-score	The RF model worked effectively for building classifiers with different spectral feature combinations. Vegetation indices are helpful tools for creating crucial characteristics for classifying weeds and crops.
Bakhshipour & Jafari. (2018)	Classification of sugar beet crop and weeds.	RGB camera.	SVM. ANN.	Accuracy	Both ANN and SVM properly identified the effectiveness of sugar plants and weeds.

Sa *et al.* (2018)	Classification of sugar beet and weeds.	Multispectral images collected by a MAV.	CNN.	Accuracy F1-score	Most of the weeds had accurate classifications. Due to restrictions in the dataset the model was trained on, certain spatiotemporal discrepancies were identified.
Yang *et al.* (2018)	Hyperspectral image classification of weeds in crops and landscapes.	Datasets.	CNN: 2-D-CNN, 3-D-CNN, R-2-D-CNN, R-3-D-CNN.	Accuracy	For the majority of the data sets, the suggested R-3-D-CNN model performs better than the majority of the current models and can converge more quickly. However, compared to conventional machine learning techniques, these models need more training samples.
Yashwanth *et al.* (2020)	Image classification of weeds in nine different crops.	Digital camera.	Keras API; Tensorflow.	Accuracy	Nine different types of crops and their matching weeds were used to evaluate the model, and 96.3% accuracy was found to be the highest. Every image offered was appropriately identified as either a plant or a weed.
Jin *et al.* (2021)	Weed identification in cabbage fields.	Digital camera.	GA. CenterNet.	Precision Recall F1-score	The recommended approach has application value for the sustainable development of the vegetable sector and is suited for ground-based weed detection in vegetable agricultural land under diverse circumstances, lighting, complicated backdrops, as well as various growth phases.
El-Kenawy *et al.* (2022)	Weed classification in wheat crops.	Images captured by a drone.	NN. SVM. KNN. GWO. SCA.	Accuracy F1-score Recall Specificity	The study shown that the suggested strategy outperforms existing methods and improves classification accuracy, with a detection accuracy of 97.70%, an F1-score of 98.60%, a specificity of 95.20%, and a sensitivity of 98.40%.

G C *et al.* (2022)	Weed classification in sugar beets.	Four canon digital cameras.	CNN: VGG16, ResNet50.	F1-Score	The VGG16 and ResNet50 models, which were created using non-uniform backdrop photos, performed well on the uniform background, with average f1-scores of 82.75% and 75%, respectively. While using a non-uniform backdrop didn't work well. The model that outperformed all others was the one that was trained utilising merged datasets from two background situations.
Zhang *et al.* (2022)	Weed classification in black bean, canola, corn, flax, soybean, and sugar beets.	RGB camera.	SVM; VGG16.	F1-score	All SVM model classifiers have fallen short in comparison to the VGG16 model classifiers. The results demonstrated that the range of the VGG16 model classifier's average F1-scores was between 93% and 97.5%. The range of SVM average F1-scores was 83 to 94 percent. In the VGG16 Weeds-Corn classifier, the corn class scored 100% F1.

2.8.4. Fruit Detection

Fruit quality detection is a method for automatically assessing the quality of fruits based on several features of a photograph, including colour, size, texture, and shape, among others. This approach works in conjunction with weed detection. Fruit quality is the major factor preventing health problems in individuals. Automatic detection is essential in the food industry, and in agriculture.

The prediction of an item's location in a picture may be done quickly and precisely using deep learning-based object identification. The object detector uses deep learning, a powerful machine learning technique, to automatically learn the visual properties required for detecting jobs. With fewer costs, side effects, and environmental issues, DL algorithms can enhance fruit detection, as well as its quality. This section focuses on the overall structure of an agricultural sector automatic inspection system for classifying fruits according to quality (Alibabaei *et al.,* 2022).

Fruit detection in individual plants

According to Pereira *et al.* (2019), a methodology based on the AlexNet architecture and transfer learning scheme was used to automatically identify and classify the six grape varietals that are most common in the Douro Region. The Douro Red Grape Variety (DRGV) and GRGV_2018 image files are two natural vineyard picture datasets that were captured in diverse Douro locations. Different image processing techniques, including independent components filters (ICF), leaf segmentation algorithm (LSA) with four-corners-in-one, leaf patch extraction (LPE), LPE with ICF, LPE with canny edge detector (CED), and LPE with Gray-scale morphology processing (GMP), were used for picture management. For training/validation, the authors set the data augmentation in three processes. The first data augmentation method was the one-pixel image translation with a given factor. The second, image was rotated by an angle of 180°. And, finally, the third augmentation method was image rotation. These new datasets, with pre-processed and augmentation pictures, were then trained in the AlexNet CNN.

In the set of experiments, the four-corners-in-one recommended approach with LSA added to it showed success in achieving the best classification accuracy. The experimental findings demonstrated the recommended classifier's reliability with a testing accuracy of 77.30%. The program needed about 6.1 milliseconds to recognise the type of grapes in a photo.

For edge Central Process Unit (CPU) devices, Mao *et al.* (2022) suggested a Real-Time Fruit Detection model (RTFD), a simple method that can identify fruit, namely tomatoes and strawberries. The PicoDet-S model is the foundation for RTFD, which maximises the efficiency of real-time detection for edge CPU computing devices by enhancing the model's structure, loss function, and activation function. The tomato dataset was compiled from the publicly accessible dataset Laboro Tomato, while the strawberry dataset was generated from the publicly available dataset StrawDI. The images used in each dataset were shot in various environments.

Two goals were set for the technological path: model training and model quantization and deployment. In the first, the RTFD model's performance was improved using the IoU bounding

box loss function, the ACON-C activation function, and the three-layer LC-PAN architecture. The tomato and strawberry datasets were used to train the RTFD model. The PicoDet-S model's performance and flexibility were enhanced by modifying its structure and using more sophisticated activation and loss functions to create the RTFD model. A three-layer LC-PAN with three detector head scales was utilised in the PicoDet-S model.

In the second, to evaluate the model's performance in actual deployment, the RTFD model was quantitatively trained using an online quantization technique. After being transformed into a Paddle Lite model and integrated into a testing Android smartphone app, the RTFD model performed extremely accurately in terms of real-time detection. The training set, test set, and validation set of photos were each given a random number. The modelling and training in this experiment are done using the PaddlePaddle 2.2.2 framework.

It is anticipated that edge computing will successfully implement the concept of redesigning the model structure, loss function, and activation function, as well as training by quantization, to expedite the detection of deep neural networks. The proposed RTFD has enormous potential for intelligent picking machines. The accuracy of the dataset's average was employed by the authors as a performance parameter. For the strawberry and tomato datasets, PicoDet-S has an accuracy of 94.2% and 76.8%, respectively. It is anticipated that edge computing will successfully implement the concept of redesigning the model structure, loss function, and activation function, as well as training by quantization, to expedite the detection of deep neural networks. The proposed RTFD has enormous potential for intelligent picking machines.

The findings of the strawberry and tomato detection are shown in Figure 24. The image has many coloured borders that represent various categories. The blue circles indicate areas of inaccurate or missing detections, while the blue arrows act as suggested indicator symbols. Strawberries that are fully grown, partially grown, and immature are represented by the colours red, orange/yellow, and light blue, respectively.

Figure 24. The impact of RTFD model detection on the tomato and strawberry datasets. The blue circles denote locations of inaccurate or missing detections, while the blue arrows act as suggestive indicator symbols. Red, orange/yellow, and light blue colours represented full, half-mature, and immature strawberries, respectively. (Mao *et al.,* 2022).

Fruit detection in areas of crops

Santos *et al.* (2017) estimated grape wine production from RGB photos including deep learning algorithms and computer vision models. Researchers demonstrate how grape clusters may be effectively recognised, segmented, and tracked using cutting-edge CNNs for wine grapes, a crop that exhibits high levels of variability in terms of shape, colour, size, and compactness. From five distinct grape varietals, pictures were taken using a Canon camera and a smartphone. The Embrapa Wine Grape Instance Segmentation Dataset (WGISD) contains 300 RGB pictures.

Mask R-CNN, YOLOv2, and YOLOv3 models from deep learning were trained to recognise and separate grapes in the photos. After that, spatial registration was carried out using the Structure from Motion (SfM) image processing technique, incorporating the information produced by the CNN-based stage. In order to prevent counting the same clusters across many photos, the clusters found in distinct images were removed using the CNN model's outputs in the final phase.

While the Mask R-CNN outperformed YOLOv2 and YOLOv3 in terms of object detection, the YOLO model outperformed it in terms of detection time. The worst performance was obtained with YOLOv3. Mask R-CNN obtained average accuracy, precision, recall, and F1-score of 87.3%, 89.0%, and 90.7% for an IoU of 3%. The average accuracy, precision, recall, and F1-score for YOLOv2 were 67.5%, 89.3%, 72.8%, and 80.2%, respectively. YOLOv3 came in last with average accuracy of 56.6%, precision of 90.1%, recall of 59.7%, and F1-score of 71.8%.

Figure 25 shows the three neural networks, Mask R-CNN, YOLOv2 and YOLOv3, gave certain object detection results, one for each grape type, where the colour does not always indicate correlation.

Figure 25. An example of an object detection result from each of the three neural networks—Mask R-CNN, YOLOv2 and YOLOv3—for each type of grape. same colour does not imply connect ion. (Adapted from Santos *et al.,* 2017).

For a real-time peach fruit identification application, Assunção *et al.* (2022) created a tensor processing unit (TPU) accelerator with a Raspberry Pi target device, lightweight MobileDet detector model, and hardware awareness. Sweet Dream, Royal Time, and Catherine, three fruit peach varieties, were integrated into a single image dataset. The photographs were taken at peach orchards in the Beira Interior area of Portugal, which is where most of the country's peaches are grown. The images were taken with an RGB Sony DSC-RX100M2 camera. The Raspberry Pi 4 microcontroller development kit, a Coral TPU accelerator, a Raspberry Pi Camera Module 2, a DC-to-DC converter, and three Li-ion batteries make up the hardware platform (edge device) used to perform inferences. The single-shot detector (SSD) model was used as a detector. SSD adjustments were made to the backbones. In this study, experiments were conducted using a MobileNet CNN as the basis for the SSD model to investigate the trade-off between detection accuracy and inference time. The backbones used were MobileNetV1, MobileNetV2, MobileNet EdgeTPU, and MobileDet.

The research showed that SSD MobileDet outperformed the other models, obtaining an AP of 88.2% on the target TPU device. SSD MobileNet Edge TPU had the least amount of performance degradation (drop), with a loss of 0.5%, while SSD MobileNetV2 had the worst impact, with a drop of 1.5%. The lowest average latency is 47.6 ms with SSD MobileNetV1.

Authors have contributed by expanding the applications of accelerators (the TPU) for edge devices in precision agriculture. A fresh dataset of peaches with three cultivars was used to assess the suggested strategy; this dataset will be made accessible for further research. The model performed at 19.84 frames per second (FPS) with an average precision of 88.2% and a 640 × 480 picture size. According to the results, the TPU accelerator can be a great replacement for processing at the cutting edge in precision agriculture.

Figure 26 shown an example of detection samples of the three cultivars, Catherine at left, Sweet Dream in the middle, and Royal Time at right.

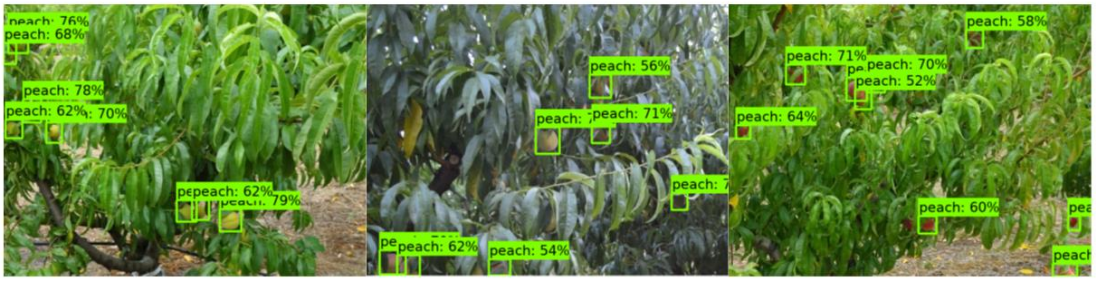


Figure 26. Detection samples for the Royal Time, Sweet Dream, and Catherine varieties of peaches are shown on the left, middle, and right, respectively. (Adapted from Assunção *et al.* 2022).

Table 5 gathers an overview of research works published in the "Fruit Detection" field. The table includes the reference, the application, the data, model and metric applied. The table also includes a column describing the model performance.

Table 5. Overview of research works in the "Fruit Detection" field.

REFERENCES	Application	Data Used	Model Used	Metric Used	Model Performance
Pereira *et al.* (2019)	Identify and classify grapes.	Dataset.	LSA; CED; GMP; LPE; ICF; CNN: AlexNet.	Accuracy	The experimental results demonstrated the reliability of the recommended classifier with a testing accuracy of 77.30%. The programme needed about 6.1 milliseconds to recognise the type of grapes in a photo.
Santos *et al.* (2020)	Identify grapes and estimate grape wine yield.	RGB camera.	Mask R-CNN; YOLOv2; YOLOv3.	Precision Recall F1-score	While the Mask R-CNN outperformed YOLOv2 and YOLOv3 in terms of object detection, the YOLO model outperformed it in terms of detection time. The worst performance was obtained with YOLOv3.
Mao *et al.* (2022)	Identify tomatoes and strawberrys.	Dataset.	PicoDet-S	Accuracy	The proposed RTFD holds great promise for intelligent picking machines, and edge computing is expected to successfully implement the concept of redesigning the model structure, loss function, and activation function, as well as training by quantization, to speed up the detection of deep neural networks.
Assunção *et al.* (2022)	Indentify peaches.	RGB camera.	MobileDet; MobileNet Edge TPU; MobileNetV2; MobileNetV1.	Precision	The model operated at a frame rate of 19.84 fps, an average precision of 88.2%, and an image size of 640 by 480. The TPU accelerator can be an excellent substitute for processing at the leading edge in precision agriculture, according to the findings.

2.9. Discussion

According to the studies presented above, the following conclusions and challenges can be pointed.

Afonso *et al.* (2019) used a deep network with performance parameters for recall and accuracy and photos from an RGB camera. According to the study's findings, ResNet18 may be a reliable CNN for detecting potatoes infected with blackleg disease in the field. However, with larger datasets and data augmentation, the performance may be enhanced.

In Assunção *et al.* (2020), images from five distinct datasets were utilised to train a CNN, and the performance was measured using an F1-Score. The findings of this analysis demonstrate that no illness is incorrectly categorised by CNN.

A CNN was used by Kerkech *et al.* (2020), with accuracy used as a performance parameter. A RGB sensor was used to take the pictures. The study's findings showed that the model that was taught using RGB photos performed better than the model that was taught using infrared images. The tiny size of the training sample is one of the research's shortcomings.

In Azgomi *et al.* (2022), images were collected using a digital camera, the accuracy of the MLP with the ANN model was evaluated, and the model was then built. The two-layer layout with eight neurons in the first layer and eight neurons in the second layer produced a maximum accuracy of 73.7%.

An OBIA model using pictures from a visible-light and multispectral camera were utilised in Peña *et al.* (2015). The system was evaluated with accuracy. The investigation's findings indicate that the multispectral camera performed better at higher flight altitudes than the visible light camera did at lower flight altitudes. The spectrum mixing of the flower and bare soil components caused some mistakes at the higher elevations, too.

In Sujaritha *et al.* (2017), two digital cameras were employed to develop a fuzzy real-time classifier, as well as accuracy for classifying the model. With better accuracy and in less time, the prototype identified weed species. The robot did, however, veer off the intended course while it was going because of the field impediments.

Huang *et al.* (2018) utilised an FCN, photos captured with a digital camera, as well as measures of precision and IoU performance. The findings of this study demonstrate that, in terms of accuracy and effectiveness, FCN technology performed well for weed identification. On the other hand, it necessitates a sizable amount of manual labelling labour because it needs several annotated photos for training and updating.

In Milioto *et al.* (2018), accuracy, IoU, precision, and recall of the model were evaluated using RGB and NIR pictures and a Mask R-CNN. The model was enhanced by adding a second channel to its input.

Bah *et al.* (2018) built a CNN from images taken with a digital camera and used accuracy as a metric. The versatility and flexibility of this strategy are interesting since a model may be quickly trained on a dataset. However, there is a lot of manual labelling needed.

Lottes *et al.* (2018) assessed an encoder-decoder CNN using RGB and NIR pictures and the F1-Score as a classification measure. Encoder-decoder FCN was surpassed by the recommended model, which was able to accurately recognise crops in all growth phases.

In Ma *et al.* (2019), an FCN with RGB pictures was created. Accuracy was employed once again. The recommended method identified the pixels in weed photos with success.

Images were derived from two datasets in this work using a different model, JULE and DeepCluster, and the authors chose accuracy and normalised mutual information. Using DeepCluster, the model performed better. Additionally, the results from these datasets suggest that clustering and unsupervised learning may be successfully applied to problems in agriculture.

Using images from two datasets and working with accuracy and IoU, an encoder-decoder CNN was used in Wang *et al.* (2020) to test the model. The findings demonstrate that, in contrast to VIs lacking NIR information, which did not improve segmentation outcomes, NIR information is beneficial for precise segmentation in low-light conditions.

Three models, namely SVM-HOG, Mask R-CNN, and YOLOv3 were used by Osorio *et al.* (2020). A multispectral camera was utilised to take the pictures, and the following performance measures were employed: F1-Score, accuracy, specificity, precision, and recall. The results of this study suggest that the HOG-SVM strategy was found to function well and is a great option for IoT devices since it uses fewer processing resources.

Islam *et al.* (2022) employed the RF, KNN, and SVM models. Accuracy, kappa, FPR, precision, and recall were utilised as performance indicators when taking pictures with an RGB camera. According to the experimental data, the findings of this study demonstrate that RF performed better than the other classifiers. Furthermore, it is clear how effective RF and SVM are in classifying weeds from UAV photos.

In Kamath *et al.* (2022), the model of the digital camera that was utilised was examined by IoU. There were four different CNN models used. The results showed that PSPNet functioned more efficiently than SegNet and UNet.

Mu *et al.* (2022) evaluated two models, FPN and Faster R-CNN, using pictures from a dataset using the metrics accuracy, recall, F1-Score, and IoU. The experimental findings show that the Faster R-CNN-FPN deep network model obtains a greater recognition accuracy by combining the ResNeXt feature extraction network with the FPN network.

The following models were employed in Assunção *et al.* (2022): TMG, DeepLabv3, and MobileNetv2. The results show that the trade-off between segmentation accuracy and inference time can be modified using the hyperparameters OS and DM, making DeepLabV3 a highly flexible model for segmentation tasks.

The accuracy of the model was tested by Dyrmann *et al.* (2016) which used a CNN with six datasets in that study. The model achieved an accuracy of 86.2%. Less picture examples were used in the classes, but the overall loss was reduced.

In Andrea *et al.* (2017), various CNN models were employed, pictures were captured using RGB and NIR cameras, and accuracy was once more applied. The results showed that the network cNET offered the best training results based on its accuracy and processing speed.

Gao *et al.* (2018) created an RF model, tested it using hyperspectral pictures, and calculated its accuracy, recall, and F1-Score. The study revealed that the RF model functioned admirably and that vegetation indices are also helpful strategies for defining critical criteria for classifying weeds and crops.

Using RGB pictures and accuracy as a performance indicator, an SVM and an ANN were developed in Bakhshipour & Jafari. (2018). The findings indicated that both models correctly distinguished between weeds and sugar plants.

In Sa *et al.* (2018), multispectral images were utilised to build a CNN, and the model's accuracy and F1-Score were used to assess its performance. Most of the weeds were accurately classified, according to the research. However, due to limitations in the dataset it was trained on, the model's spatiotemporal inconsistencies were found.

In Yang *et al.* (2018), four CNN models were applied, in addition to six datasets and accuracy as a performance criterion. The results show that for the majority of the data sets, the proposed R-3-D-CNN model can converge more quickly and consistently outperform the existing methods. More training data is, however, required for these models.

A model using the Keras API and Tensorflow was constructed by Yashwanth *et al.* (2020), and photos were taken using a digital camera. The model's accuracy was put to the test. The maximum accuracy was achieved by correctly identifying the weeds and crops.

The CenterNet network was used to create genetic algorithms in Jin *et al.* (2021). A digital camera was used together with accuracy, recall, and F1-Score to evaluate the model. Under various illumination, background complexity, and growth stage conditions, the suggested method for ground-based weed identification in vegetable agricultural land is successful.

Three models, NN, SVM, and KNN, were created by El-Kenawy *et al.* (2022), and their various iterations were assessed using GWO and SCA genetic algorithms. According to the research, the suggested strategy outperforms alternative approaches and improves classification accuracy.

In G C *et al.* (2022), CGG16 and ResNet50 were used to create a CNN model. The pictures were taken using four digital cameras, and F1-score was utilised to assess the model. According to the findings, the model that was developed utilising pooled information from two background situations outperformed all others. The models that were created utilising non-uniform backdrop images also performed well on a consistent background. Although using a non-uniform backdrop has poor performance.

Zhang *et al.* (2022) developed a model using SVM and VGG16 that utilised RGB pictures. The findings demonstrated that all SVM model classifiers were outperformed by the VGG16 model classifiers.

In Santos *et al.* (2020) was used as Mask R-CNN with a YOLOv2 and YOLOv3. The model's developers used RGB images and assessed them with accuracy, recall, and F1-Score. The results demonstrated that the YOLO model outperformed the Mask R-CNN in terms of detection time while the Mask R-CNN outperformed YOLOv2 and YOLOv3 in terms of object detection.

Mao *et al.* (2022) created a picoDet-S CNN model. Accuracy was utilised as a performance metric by the researchers, who used two datasets. It is projected that the idea of revamping the

model structure, loss function, and activation function, as well as training by quantization, would be successfully implemented by edge computing to hasten the detection of deep neural networks. There is a lot of promise for intelligent picking machines with the suggested RTFD.

A CNN model with optimising algorithms was created by Pereira *et al.* (2019). The accuracy was employed as a performance parameter across two datasets. The outcomes show that, with an accuracy of 77.30%, the model classifiers are reliable.

In Assunção *et al.* (2022), the research used RGB images, a MobileNet with TPU model was created, and the model's effectiveness was assessed using accuracy. The results show that the TPU accelerator can be a great replacement for processing at the cutting edge in precision agriculture.

The articles up for discussion draw attention to the following difficulties. The datasets are the most crucial and necessary component. The model may still require a sizable quantity of data to be trained, even with transfer learning and data augmentation; otherwise, the model may have substantial failures (Yang *et al.,* 2018; Afonso *et al.,* 2019; Kerkech *et al.,* 2020). The performance will be better with more sampled datasets. Additionally, the dataset quality is problematic since low-quality images could produce poor results (Sa *et al.,* 2018). As a result, acquiring actual field data and images in a variety of settings is the first and most important step.

Agricultural image datasets are more complex because of several factors, such as outdoor conditions, the object of interest typically occupying a very small and off-centre portion of the image, similarity between objects and background, the object being obscured by leaves and branches, multiple objects in one image, and a few others. For the dataset to be relevant in the real world, the environment's state must be appropriately reflected (Pereira *et al.,* 2019; Assunção *et al.,* 2022; Santos *et al.,* 2020; Mao *et al.,* 2022). In some situations, such as those with variable illumination, data augmentation may also be helpful.

The quantity of data that must be labelled is a significant barrier as well. This process costs money and takes time. Furthermore, many tasks, such as those concerning plant diseases, can only be done by industry professionals. Many tagged pictures are required for supervised learning in order to train and update the models (Huang *et al.,* 2018; Bah *et al.,* 2018).

Data augmentation and transfer learning are techniques to avoid labelling a big dataset, as observed in multiple studies, but labelling a small dataset still needs effort. Although techniques for unsupervised and semi-supervised learning offer considerable potential, more research is still required (Ma *et al.,* 2019; Bah *et al.,* 2018; Ferreira *et al.,* 2019).

The type of input affects how well the model performs (Assunção *et al.,* 2020; Azgomi *et al.,* 2022; Peña *et al.,* 2015; Milioto *et al.,* 2018; Lottes *et al.,* 2018). The model's performance is impacted by background removal from photos (El-Kenawy *et al.,* 2022), implementing various input colour spaces and vegetation indices (Milioto *et al.,* 2018), crop identification or detection at different growth stages (Jin *et al.,* 2021). For the input, the height from which the photos were obtained is also crucial (Peña *et al.,* 2015). So, it might be challenging to choose the optimum input set for a particular activity.

Field obstructions may provide a challenge for the utilisation of field robots. Robots deviate from their intended course when they encounter obstacles (Sujaritha *et al.,* 2017). Instead of using

robots to handle this issue, drones might be used. Farmers can also have the field cleared and levelled as an alternative.

It is necessary to trade between accuracy and inference time when selecting a model for a particular task. The model can be selected based on use. The DL model cannot be used in every situation since no two agricultural settings are similar to one another and because every environment and problem has a different dataset. The model performance could deteriorate because the photos in the training and test datasets have different visual qualities (Dyrmann *et al.,* 2016). Retraining the previously learned model with a tiny dataset from the new environment is one way to get around this (Alibabaei *et al.,* 2022).

Also, the selection of hyperparameters, loss functions, and optimisation algorithms affects how well these models work. Finding the appropriate hyperparameters can be aided by algorithms like Bayesian optimisation (Alibabaei *et al.,* 2022; Assunção *et al.,* 2020).

The models' real-time applicability presents another challenge. Most deep learning models require significant parameter training and once learned, they cannot make real-time conclusions. Time inference is crucial, for instance, when utilising a robot for harvesting. Only two of the issues with implementation on devices like smartphones that must be considered are memory utilisation and performance. Deep learning models may now be used in practical and real-world applications as a result of the development of edge devices like the Raspberry Pi and Jetson Nano, lightweight categorization models like MobileNet, and cloud computing. The model size may be decreased and the detection speed increased by using the quantization approach (Alibabaei *et al.,* 2022; Assunção *et al.,* 2020).

2.10. Conclusive Remarks

This state-of-the-art reviewed current literature needed to develop this work. More specifically, plant agrobiodiversity was studied, as well as data acquisition, vegetation indices that can be employed, and the performance metrics handled to evaluate the effectiveness or quality of the model. In the last subchapter, some studies that used deep learning in agriculture were examined, this contributes to the development of new algorithms for flora, that can become weeds, classification.

It is feasible to promote agricultural sustainability and biodiversity by using fewer pesticides, fewer herbicides, organic farming, enough crop rotations, small-scale fields, and keeping natural gaps between agroecosystems (Alibabaei *et al.,* 2022). As said earlier, this may be accomplished by fusing the most advanced IoT technologies with the latest AI models and biodiversity algorithms. They can be used to find, classify, and eliminate weed species, locate and identify fruits and vegetables, find illnesses, and boost ecosystem productivity—all without using practices that are damaging to the environment.

Despite being employed in several agricultural applications before, deep learning is still only sometimes applied in this field. There is a need for more study as evidenced by the characteristics of DL model use and the difficulties in agriculture.

When DL is applied in agriculture, it is important to understand which type of plants people are dealing with. Since some vegetation has advantages to the soil and crops. Furthermore, knowing the characteristics of the plant as well as its habitat and origin results in more sustainable maintenance.

In terms of performance metrics, the most metrics employed are accuracy, precision, recall, and F1-Score. Usually, precision, recall and F1-Score are utilized together. However, both accuracies and F1-Score are shown on your own.

In addition, the type of data that researchers employed the most are prevenient for datasets, camera RGB, and digital cameras.

The most often used deep learning model in agriculture is CNN and related models. Adopting innovative techniques like attention mechanisms, new lightweight models, and single-stage detection models might enhance the model's performance because even a little increase in runtime and accuracy can lead to noticeably better outcomes.

In the future, the objective is to create or enhance crop management decision-support models for farmers. These models may be included in decision support systems, which can lead users through certain stages and recommend the best course of action.

3.Materials and Methods

In the pursuit of advancing scientific understanding and technological innovation, the careful selection and rigorous application of materials and methods are predominant. This chapter dives into the foundational framework that supports our research, detailing the critical components that facilitate the achievement of the objectives. This section explores the architectures of PyTorch models, which form the computational core of our experiments, alongside an analysis of the selected data and insights into the development and simulation of the algorithms.

3.1. PyTorch Models

With their unique architecture and sophisticated algorithms, CNN models have propelled the accuracy and efficiency of image analysis to unprecedented levels. A strong framework for creating, developing, and deploying CNN models is offered by PyTorch (Paszke *et al.,* 2019).

PyTorch is a flexible and intuitive DL platform for building and training neural networks. It is an open-source framework that gives model development, training, and deployment a flexible and dynamic approach. Because of its user-friendly interface and support for dynamic computing graphs, it is widely utilized for research and production. Furthermore, is an optimized tensor library for DL using GPUs and CPUs (Paszke *et al.,* 2019).

According to the available models in PyTorch, twenty models were used. However, some of these models contain several model builders (other architectures). Thus, eighty distinct models were simulated as a result.

Table 6 shows the models and their model builders for image classification tasks available in PyTorch.

Table 6. Classification models and model builders available in PyTorch that were used for the simulations.

Models	Model builders
MobileNetV2	mobilenet_v2
VisionTransformer	vit_b_16/ vit_b_32/ vit_l_16/ vit_l_32/ vit_h_14
SwinTransformer	swin_t/ swin_s/ swin_b/ swin_v2_t/ swin_v2_s/ swin_v2_b
SqueezeNet	squeezenet1_0/ squeezenet1_1
MobileNet V3	mobilenet_v3_large/ mobilenet_v3_small
EfficientNetV2	efficientnet_v2_s/ efficientnet_v2_m/ efficientnet_v2_l
MaxVit	maxvit_t
MNASNet	mnasnet0_5/ mnasnet0_75/ mnasnet1_0/ mnasnet1_3
AlexNet	alexnet
ConvNeXt	convnext_tiny/ convnext_small/ convnext_base/ convnext_large
DenseNet	densenet121/ densenet161/ densenet169/ densene201
EfficientNet	efficientnet_b0/ efficientnet_b1/ efficientnet_b2/ efficientnet_b3/ efficientnet_b4/ efficientnet_b5/ efficientnet_b6/ efficientnet_b7
GoogLeNet	googlenet
RegNet	regnet_y_400mf/ regnet_y_800mf/ regnet_y_1_6gf/ regnet_y_3_2gf/ regnet_y_8gf/ regnet_y_16gf/ regnet_y_32gf/ regnet_y_128gf/ regnet_x_400mf/ regnet_x_800mf/ regnet_x_1_6gf/ regnet_x_3_2gf/ regnet_x_8gf/ regnet_x_16gf/ regnet_x_32gf
Inception V3	inception_v3
ResNet	resnet18/ resnet34/ resnet50/ resnet101/resnet152
ResNeXt	resnext50_32x4d/ resnext101_32x8d/ resnext101_64x4d
ShuffleNet V2	shufflenet_v2_x0_5/ shufflenet_v2_x1_0/ shufflenet_v2_x1_5/ shufflenet_v2_x2_0
VGG	vgg11/ vgg11_bn/ vgg13/ vgg13_bn/ vgg16/ vgg16_bn/ vgg19/ vgg19_bn
Wide ResNet	wide_resnet50_2/ wide_resnet101_2
20 Models	**80 Model builders**

3.2. Data Collection and Sample Preparation

To develop this work, a dataset was used containing images of the vineyard's flora. All the images were obtained through a Sony Camera, in 2022, in two different seasons, specifically, spring and autumn. The vineyard's flora in research are from Douro Natural Park.

The dataset contains 172 images divided by five classes of weeds. The species used for classification were *Centaurea melitensis*, *Echium plantagineum*, *Erodium moschatum*, *Lolium rigidum*, and *Ornithopus compressus*.

These species are represented by order in Figure 27.

After the collection of images, they were divided for further utilization in the algorithms, for training and validation of the models. According to the literature, the 172 shots were split into 85% and 15 %, respectively (SEP 4 & Read, 2020). The division of the pictures was random

through RoboFlow. Thus, 145 images were employed for training, and 27 photos were applied for validation of the model.

The training set is the biggest part of the dataset and refers to the procedure of teaching a deep learning or machine learning model to identify and classify various items or groups of objects inside photographs. The purpose of training is to provide the model with the ability to recognise patterns, features, and representations from labelled samples in order to accurately forecast the appearance of new pictures (SEP 4 & Read, 2020).

The validation set refers to the practice of assessing how well a trained model performed on a different dataset that wasn't utilized during training (SEP 4 & Read, 2020).

Figure 28 shows this division of the pictures, specifically, how many photos of each species are in the train and validation dataset.

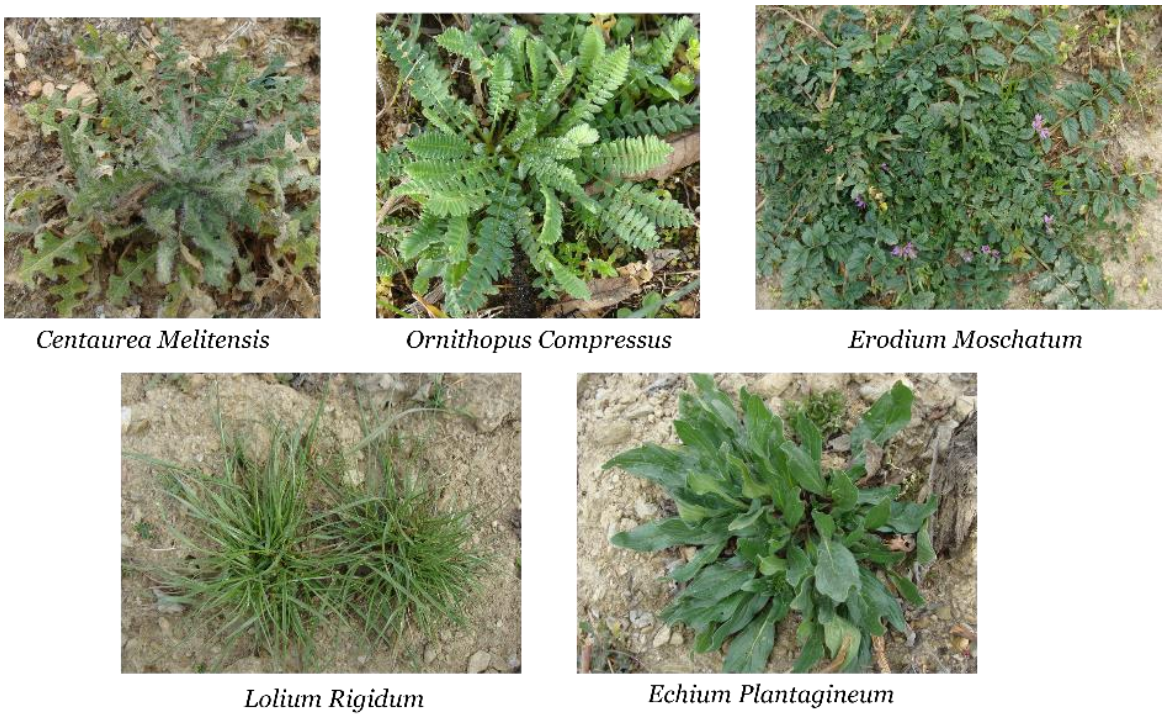


Figure 27. Illustration of the five species used for image classification task.

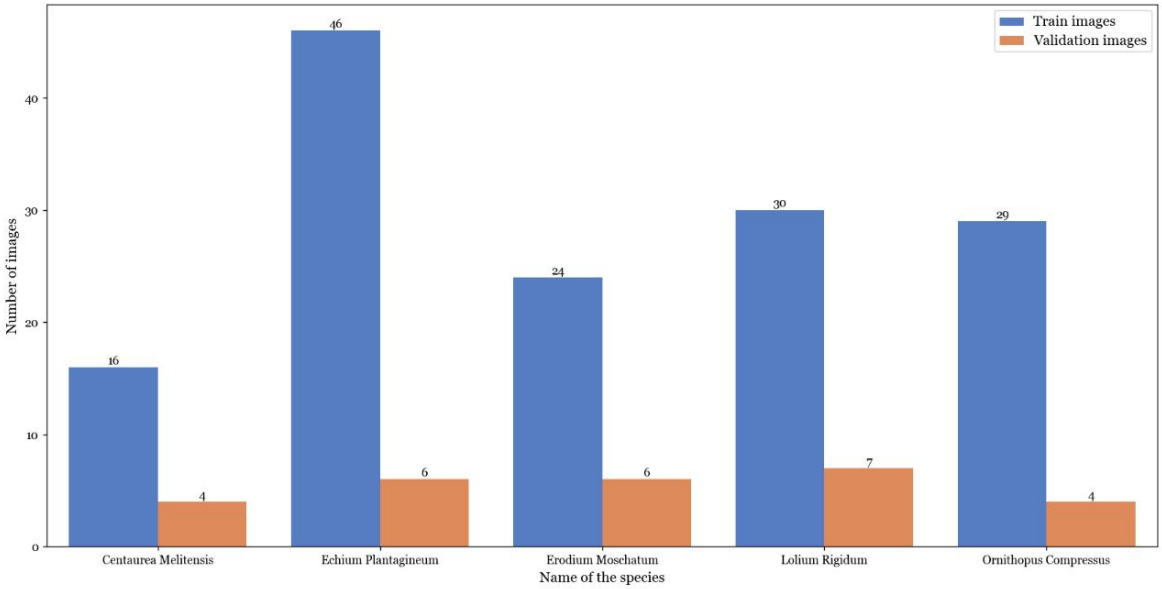


Figure 28. Dataset split for model training and validation: visualizing species-specific distribution. The blue line represents the number of images for train of each specie, and the orange one, the number of images for validation of each specie.

3.3. Algorithm Development

The proposed algorithm, developed within the PyTorch framework, serves the primary objective of evaluating all available classification models in PyTorch to identify the most suitable model for our dataset. In the initial phase, several libraries were imported to facilitate various tasks throughout the code, including PyTorch integration, data handling, and machine learning operations.

Subsequently, it becomes imperative to specify the computational device for executing calculations. This involves selecting the computing resource, where, in the presence of a CUDA-enabled GPU, CUDA (GPU) acceleration is utilized; otherwise, CPU processing is employed. The type of device utilized is then reported. Using a GPU is advantageous over a CPU for specific tasks due to its parallel processing capabilities, specialized hardware for mathematical operations, and speed. GPUs are highly efficient in parallelizable workloads like deep learning and scientific simulations, often outperforming CPUs (Caulfield, 2009).

Following this, a sequence of data transformations is established to be applied to the dataset images. This sequence encompasses actions such as resizing images to match the default model size, conversion into PyTorch tensors, and subsequent normalization.

Further, we defined the needed hyperparameters, encompassing the number of classes in the classification problem, the learning rate for optimization, the batch size dictating the processing of samples in each iteration, and the number of epochs, signifying the dataset's complete passage through the model.

Afterwards, the paths to directories containing training and validation images are retrieved, and lists are created to store training and validation loss and accuracy values. Dataloaders are initialized to effectively manage and load training and validation datasets. These dataloaders, derived from the DataLoader class, facilitate batch loading with shuffling to prevent the model from memorizing the training data order.

Then, calculations are performed, and results are displayed to ascertain the total count of training and validation images within each respective directory. This computation leverages Equation (22), where 'images' corresponds to the directory housing the relevant image files. The 'glob.glob()' function generates a list of file paths adhering to a specified pattern, '/**/./.jpg', matching all .jpg files within the directory and its subdirectories. The 'len()' function subsequently returns the length of the file path list, indicating the count of image files.

$$\text{Count} = \text{len}(\text{glob.glob}(\text{images} + '/**/*.\text{jpg}')) \tag{22}$$

A custom neural network model is continuously defined by inheriting from the nn.Module class. This model leverages the "PyTorch_Model" architecture from the torchvision library as a pre-trained base, with further adjustments made to the fully connected layers responsible for classification. The last layers of the architecture are substituted with a sequence of linear (fully connected) or sequential layers to align with the specific number of classes in the problem.

Subsequently, the total count of model parameters is calculated and reported using Equation (23). In this context, 'model' denotes the instance of the neural network model. The '.parameters()' method is employed to obtain an iterator encompassing all the model's learnable parameters, comprising the weights and biases of all layers. The 'sum' function is utilized to aggregate the values generated by the generator expression. Lastly, 'p.numel() for p in model.parameters():' signifies a generator expression that iterates through each parameter 'p' within the model's parameters and returns the number of elements within each parameter 'p'.

$$\text{Parameters} = \text{sum}\left(\text{p.numel}(\,)\text{for p in model.parameters}(\,)\right) \tag{23}$$

A loss function, specifically CrossEntropyLoss, is defined alongside an optimizer known as Adam, which serves as an optimization algorithm for updating the model parameters. The role of the loss function is to compute the classification loss, while Adam facilitates the iterative refinement of model parameters.

Subsequently, an iterative process spanning the specified number of epochs commences. Each epoch entails the training and validation of the model, with the best-performing model based on validation accuracy being saved.

To initiate this process, a variable denoted as 'best_accuracy' is initialized to a value of 0.0. This variable serves to monitor and record the highest validation accuracy achieved during the training phase.

The epoch iteration begins by setting the model to training mode using 'model.train()' and initializing several variables designed to capture training and validation metrics for the current epoch.

Within the training loop, an inner loop iterates through the training dataset using the 'train_loader'. This loop retrieves batches of images along with their corresponding labels. The images and labels are then relocated to the designated device, whether it be GPU or CPU.

To ensure the avoidance of gradient accumulation from previous iterations, 'optimizer.zero_grad()' resets the optimizer's gradient. The model is employed to predict outputs based on the input images, and the loss is subsequently computed using the specified loss function, quantifying the disparity between the predicted outputs and actual labels.

Backpropagation is employed to calculate gradients of the loss with respect to the model's parameters, facilitated by 'loss.backward()', and model parameters are updated via 'optimizer.step()'.

Following this, the training loss is updated by incorporating the loss value scaled by the batch size, as detailed in Equation (24). The predictions with the highest associated probability are obtained using 'torch.max(outputs.data, 1)'. This line contributes to the accumulation of the training loss by multiplying the current batch's loss by the batch size (images.size(0)). The '.cpu()' function serves to transfer the loss tensor to the CPU, and '.data' is used to extract the scalar value.

$$\text{train\_loss} = += \ \text{loss.cpu().data} * \text{images.size(0)} \tag{24}$$

The training accuracy is determined by aggregating the number of correct predictions within the batch. This computation is facilitated by employing 'torch.max()' to identify the class with the highest predicted probability for each sample within the batch. Subsequently, a comparison is made between these predictions and the actual labels to ascertain the count of accurately predicted instances, as exemplified in Equation (25).

$$\text{train\_accuracy} \mathrel{+}= \text{int}(\text{torch.sum}(\text{prediction} == \text{labels.data})) \tag{25}$$

The training accuracy and loss are computed by dividing the cumulative values by the total count of training samples, as indicated in Equations (26) and Equation (27). Following this computation, the resulting values are rounded and appended to their respective lists, preserving them for subsequent analysis.

$$\text{train\_accuracy} = \text{np.round}((\text{train\_accuracy} / \text{train\_count}), 4) \tag{26}$$

$$\text{train\_loss} = \text{np.round}((\text{train\_loss} / \text{train\_count}), 4) \tag{27}$$

Before evaluating the model on the validation dataset, a crucial step involves setting the model to evaluation mode through the use of 'model.eval()'. This ensures consistent evaluation

procedures. The validation process mirrors that of the training loop, encompassing the calculation of validation accuracy and loss.

During validation, 'torch.no_grad()' is employed to temporarily deactivate gradient computation. This measure proves essential, as gradient computation is unnecessary during validation, where parameters remain static. This not only conserves memory but also accelerates computations by obviating the need for parameter updates.

Similarly to the training loop, the validation loop involves an inner iteration through the training dataset, utilizing the 'validation_loader' to fetch batches of images and their corresponding labels. These images and labels are appropriately transferred to the chosen computing device, be it GPU or CPU.

Validation accuracy is computed by summing the number of correct predictions within the batch. This calculation leverages 'torch.max()' to identify the class with the highest predicted probability for each sample within the batch. Subsequently, these predictions are juxtaposed with the actual labels, enabling the count of accurately predicted instances, a process elucidated in Equation (28).

$$\text{validation\_accuracy} \mathrel{+}= \text{int}(\text{torch.sum}(\text{prediction} == \text{labels.data})) \tag{28}$$

Mirroring the process employed in the training loop, the validation loss undergoes updates through the addition of the loss value, duly scaled by the batch size, as exemplified in Equation (29). Furthermore, predictions with the highest associated probability are extracted through the utilization of 'torch.max(outputs.data, 1)'.

$$\text{validation\_loss} = \mathrel{+}= \text{loss.cpu().data} * \text{images.size(0)} \tag{29}$$

Validation accuracy and loss are computed through the division of the cumulative values by the total count of training samples, as elucidated in Equations (30) and Equation (31).

Subsequently, these computed values are subjected to rounding and subsequently appended to their respective lists, thereby preserving them for subsequent analytical purposes.

$$\text{validation\_accuracy} = \text{np.round}((\text{validation\_accuracy} / \text{validation\_count}), 4) \tag{30}$$

$$\text{validation\_loss} = \text{np.round}((\text{validation\_loss} / \text{validation\_count}), 4) \tag{31}$$

Preceding the evaluation of the validation dataset, the model undergoes a transition into evaluation mode through the utilization of 'model.eval()'. This phase mirrors the procedures employed in the training loop, encompassing the calculation of validation accuracy and loss.

Upon conclusion of each epoch, a crucial check is performed: if the prevailing validation accuracy surpasses the previous best accuracy recorded, the state dictionary of the model is diligently preserved under the appellation 'best_checkpoint.pth'. Simultaneously, the 'best_accuracy' variable is updated to reflect this newfound peak accuracy. This iterative process persists for the designated number of epochs.

In the culmination of the iterative process, encompassing all epochs, the training and validation accuracies, along with their respective losses, are disseminated in both printed form and saved as CSV files for future reference.

In Figure 29, a schematic representation outlines the sequential steps indispensable for the development of the algorithm for simulation.

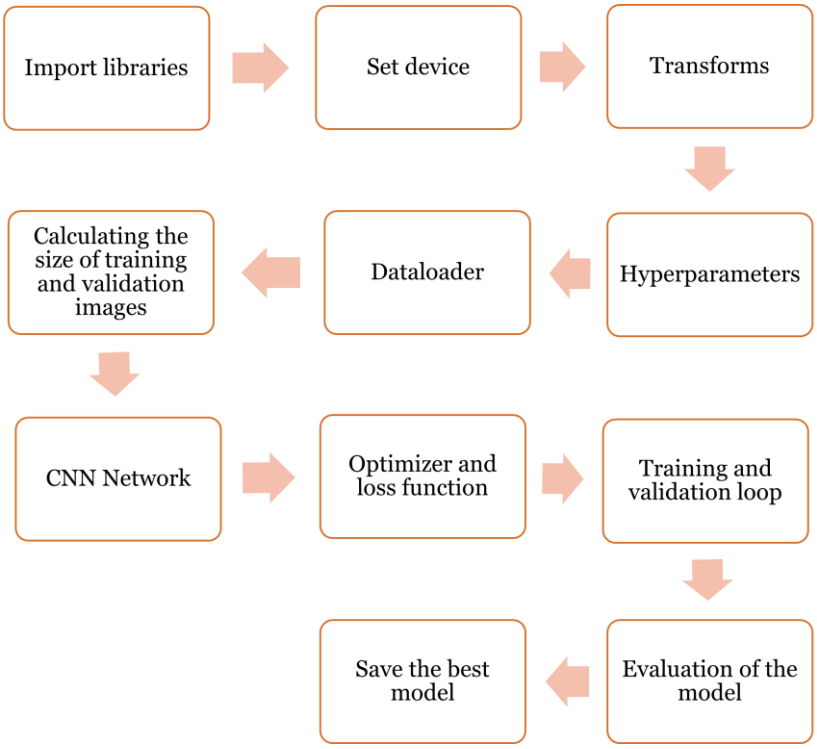


Figure 29. Description of the steps for algorithm development.

An inference code is designed as a distinct algorithm for evaluating models within a web application.

The initial step entails the importation of essential libraries. These libraries serve as integral components throughout the code, enabling various functions such as interaction with PyTorch, data management, and execution of machine learning operations.

Subsequently, meticulous configuration of the computing device is imperative. This pivotal action designates the computing environment for ensuing computations. In the presence of a CUDA-enabled GPU, computation is accelerated through CUDA; otherwise, the CPU is utilized. The type of device employed is duly reported.

In parallel with the prior description of the simulation algorithm, this section constructs a distinctive neural network class employing PyTorch models.

Following this, the model's weights are sourced from a checkpoint file, and it is subsequently relocated to the designated device (be it GPU or CPU) for computation.

Using the PIL library, the ensuing code segments read a PNG image and convert it to the RGB colour mode.

Subsequent lines of code execute a sequence of data transformations, like those employed in the previous algorithm, to be applied to the input image.

The model is then transitioned into evaluation mode, and the image tensor is migrated to the specified device.

The following tasks delineate the path to the directory housing the testing images, instantiate a path object, and organize the class names from the subdirectories within the training directory.

Following these preparations, predictions are generated for the pre-processed image leveraging the loaded model. The script orchestrates the conversion of the prediction tensor into a NumPy array on the CPU and proceeds to extract the predicted class index. This segment of code involves the following steps: it accepts the pre-processed image as input, processes it through the model to procure the predicted scores for each class, designates the class with the highest score as the predicted class label, and subsequently extracts the numerical index corresponding to that class for subsequent processing.

After this, the predicted class index (labelID) undergoes further processing to extract the corresponding class name. This procedure presumes that the class names adhere to the format "firstname_lastname." Accordingly, the initial letters of both the first name and last name are capitalized, and they are concatenated with a space to form the complete name. Additionally, processing is conducted to extract the first and last names from the predicted class label.

Ultimately, the predicted class label is presented in the following format: "Predicted class: Firstname Lastname."

Figure 30 encapsulates a schematic representation outlining the essential steps requisite for the development of the inference algorithm.

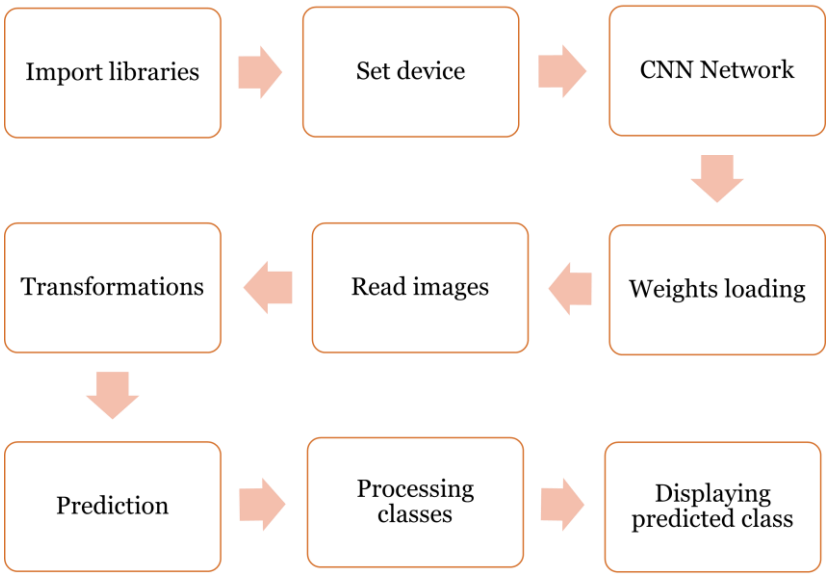


Figure 30. Description of the steps for inference algorithm development.

The web application relies on the Gradio library to develop an image classification interface.

Gradio is an open-source Python library that is used to simplify the process of creating and deploying interactive ML models and web applications. It offers a straightforward API that allows to define input and output components of ML models. It is compatible with different ML frameworks, including PyTorch, making it easy to integrate pre-trained models into web applications. It provides a user-friendly interface for building web-based user interfaces around ML models, allowing users to interact with the models through a web browser without needing to write complex front-end code. Users can provide input through the interactive interface and see the model's predictions. Furthermore, it supports various input types, including text, images, videos, audio, and even custom components. This flexibility makes it suitable for a wide range of ML applications (Team, n.d.).

To start this process, the first step involves importing essential libraries. These libraries play pivotal roles in constructing the web interface, carrying out numerical operations, and facilitating theme customization. Additionally, the code includes the crucial step of loading the model's weights, which are utilized in the web application.

Subsequently, a function named `DLinferenceMethod` is created. This function accepts an input image and employs the `InferenceModel` function to execute the model and receive its predictions. These predictions rely on the model loaded earlier, along with any necessary transformations.

The deep learning model and its associated transformations are loaded using the `loadModel` function. This step ensures that the model is ready for making predictions.

Moreover, the layout of the web application is defined using `gr.Row` and `gr.Column`. These layout components structure the different sections of the web app, such as text descriptions and input areas for images, this also enables the web page to be responsive.

To design the input components, the 'gr.Image' block enables users to upload images. The 'gr.Button' component facilitates the submission of the image for classification. Finally, the 'gr.Columns' element is responsible for displaying the classification result.

An event is defined for the 'submit_btn' button. When users click this button, it activates the 'DLinferenceMethod' function. This function takes the input image, processes it, and then presents the prediction within the 'final_prediction' textbox.

After setting up the interface and functionality, the Gradio web application is launched on a configured server address and port, showing the web interface for users.

3.4. Simulation of the Algorithms

After the division of the images for train and validation, and the redaction of the algorithms the third part of the work was carried out. Thus, the simulations started to determine which model

produced superior results, the tests of each of the models mentioned earlier were run along with modifications in the hyperparameters.

The conducted simulations were developed with different combinations of hyperparameters. These fusions were implemented to determine which association of each value of parameters present a better result.

Hyperparameters are variables that give exact values to control how algorithms learn and modify the performance of the model.

Thus, the chosen parameters were batch size, number of epochs, learning rate, decay, and number of layers.

The batch size is the number of samples that are processed before the model is updated. A batch must have a minimum size of one and a maximum size that is less than or equal to the number of samples in the training dataset (Brownlee, 2022).

The number of epochs controls how many times the learning algorithm will run over the full training dataset. There are one or more batches within an epoch (Brownlee, 2022).

The hyperparameter learning rate (α) controls how frequently the model changes its parameters while being trained. In further detail, it regulates the step size during each iteration as it advances towards the loss function minimum. It is a scalar number that normally ranges from 0 to 1 (Nabi, 2019).

Weight decay is a type of regularisation that penalises heavy weights. This is accomplished by including a term in the loss function that is proportional to the squared weight sum. The term decreases the weights' size and stops them from expanding too much (Nabi, 2019).

The number of layers is an important element of the network design. Feature extraction is possible via layers. In most machine learning problems, a linear connection is insufficient to adequately represent the task complexity. Here, it becomes crucial to test fewer or more layers (Zhao *et al.,* 2019).

The simulations that were run always used a batch size of four, due to the memory of one of the computers. The computer has a memory of 4096MiB, so it only accepts the maximum batch size of four.

Also, a number of epochs of 200 was defined in all the tests, in order to study the learning of the model.

Additionally, the simulated values of learning rate were 0.01, 0.001, and 0.0001. As well as weight decay, it was used a default value of zero incorporated in the Adam optimizer and the value of 0.0001.

To test how the number of layers would influence the model, we applied two different combinations. These combinations consist in a linear and a sequential number of layers configuration. Fully connected layers, in neural network terminology, are components in a neural network architecture where each neuron in a layer is connected to every neuron in the previous and subsequent layers. These connections enable each neuron to receive input from all neurons in the previous layer and send its output to all neurons in the next layer.

A linear number of layers refers to a neural network architecture where each layer is connected linearly to the next. In other words, the features that came out of the convolutional layers are linearly connected to the number of classes (Zhao *et al.,* 2019).

A sequential number of layers refers to a sequence of linear layers where each layer is composed one after the other. The sequence of layers is decreased impacted by the number of bits, and the last layer represents the number of classes.

In Table 7, all the twelve combinations are presented. This combinations of hyperparameters will be applied in the simulations for each model.

Table 7. Hyperparameter combinations for model simulation for exploring the configuration.

Learning rate	**Weight Decay**	**Layers**
0.01	Default	Sequential
0.01	Default	Linear
0.01	0.0001	Linear
0.01	0.0001	Sequential
0.001	Default	Sequential
0.001	Default	Linear
0.001	0.0001	Linear
0.001	0.0001	Sequential
0.0001	Default	Sequential
0.0001	0.0001	Linear
0.0001	0.0001	Sequential
0.0001	Default	Linear

The simulated algorithms were 624. PyTorch has available 20 models, with the model builders of each one, it gives a total of 80 models. As the combinations of hyperparameters provides 12 different simulations for each model, it gives a total of 960 algorithms to run. However, some models were impossible to run due to the graphics of the computers because they don't have enough memory. Thus, 924 algorithms with different combinations of hyperparameters were simulated and evaluated. Figure 31 shows a visual explanation of the number of models and algorithms to simulate.

The simulations were realized in two different computers, the first one the graphics was a NVIDIA GeForce GTX 1650 Ti, the CPU was an AMD Ryzen 5 4600H, and a RAM of 8GB. The second computer has a NVIDIA GeForce RTX 2080 SUPER, the CPU was an Intel i7-4790 (8), 4.000GHz, and contain a RAM of 32GB.

4. Experimental Results

Simulation serves as the proving ground for the algorithms, enabling us to test hypotheses, assess performance, and refine the approaches in a controlled environment. In this final subchapter, we unveil the intricate simulations that validate the efficacy and robustness of our algorithms, providing insights into their real-world applicability. The results obtained from the different developed experiments are presented, analysed, and discussed.

4.1. Experiment 1: Simulating PyTorch Classification Models

For the first experiment, all the models available in PyTorch were simulated with different combinations of hyperparameters. The combinations represented in Table 9 were used in all the models and their architectures. As said before, all the algorithms were simulated with a batch size of four, 200 epochs, and with 12 different combinations of parameters each, presented in Table 8.

The objective of this part of the work was to determine which one of the models was the best for vineyard flora classification.

To this purpose, many models achieved high accuracy. The results of all the simulations can be seen in Appendix A. In that table is possible to see the results from 0, which is equal to 0%, to 1, equivalent to 100%, with four decimals, as they were recorded during the simulation.

Table 8 is presented the best results in accuracy and loss for each model. The table is organized in the following order: name of the model, value of the learning rate, the value of the weight decay, the number of layers, the best accuracy achieved by that model in the train, the best accuracy achieved by that model in the validation, the best loss achieved by that model in the train, and, finally, the best loss achieved by that model in the validation.

Figure 32 shows the best accuracies achieved by each model, simulated with the small and first dataset.

Table 8. Loss and accuracy performance across PyTorch classification models.

Models	Learning rate	Weight decay	Number of Layers	Best acc train [%]	Best acc validation [%]	Best loss train	Best loss validation	Number of parameters
MobileNetV2	0.0001	0.0001	Sequential	33	44	1.6700	1.5600	3044742
MobileNetV3_Large	0.0001	Default	Linear	33.57	44	1.6821	1.1900	4209718
MobilenetV3_Small	0.001	0.0001	Sequential	26.57	44	1.7640	1.3600	2207654
MaxVit	0.001	0.0001	Linear	100	96	0.0004	0.0003	30410702
AlexNet	0.0001	Default	Sequential	100	96	0.0001	0.0001	68182470
GoogLeNet	0.01	Default	Linear	100	96	0.0300	0.0006	5606054
Vit_b_24	0.0001	Default	Sequential	100	96	0.0001	0.0001	87126382
Vit_b_32	0.0001	0.0001	Linear	100	96	0.0001	0.0001	88230382
ResNeXt50_32x4d	0.0001	0.0001	Linear	100	80	0.0005	0.0004	23669702
ResNeXt101_32x8d	0.01	0.0001	Sequential	42	48	1.6000	0.1870	87956422
ResNeXt101_64x4d	0.0001	0.0001	Sequential	100	76	0.0005	0.0002	82620358
ResNet18	0.0001	Default	Linear	100	84	0.0001	0.0001	11179590
ResNet34	0.0001	Default	Sequential	100	84	0.0001	0.0001	21962670
ConvNeXt_Tiny	0.0001	Default	Sequential	100	99.30	0.0001	0.0001	28378854
ResNet50	0.0001	Default	Linear	100	80	0.0002	0.0002	23520326
Convnext_small	0.001	0.0001	Linear	88	99.30	0.0400	0.0006	49459302
Wide_ResNet50_2	0.0001	0.0001	Linear	100	64	0.0020	0.0010	66846534
Wide_ResNet101_2	0.0001	Default	Sequential	100	96	0.0020	0.0001	127625670
Convnext_base	0.0001	Default	Sequential	100	96	0.0000	0.0000	88256262
Convnext_large	0.0001	Default	Sequential	100	96	0.0001	0.0001	197969478
EfficientNet_v2_s	0.0001	0.0001	Sequential	51.05	65	1.3300	0.7300	20998358
EfficientNet_v2_m	0.0001	0.0001	Linear	37.06	48.95	1.5900	0.9800	52866042
EfficientNet_v2_l	0.0001	Default	Sequential	44.76	48	1.4600	1.0100	118055142
Swin_t	0.0001	Default	Sequential	100	68	0.0001	0.0001	27749888

Swin_s	0.0001	Default	Sequential	100	72	0.0002	0.0001	49067792
Swin_b	0.0001	Default	Sequential	100	76	0.0002	0.0001	87433022
Swin_v2_t	0.0001	Default	Sequential	100	99.30	0.0004	0.0001	27813104
Swin_v2_s	0.0001	Default	Linear	100	64	0.0001	0.0001	48973056
Swin_v2_b	0.0001	Default	Sequential	100	72	0.0001	0.0001	87930848
DenseNet201	0.001	Default	Sequential	100	96	0.0052	0.0001	19241478
DenseNet161	0.01	Default	Linear	100	96	0.0105	0.0001	26485254
DenseNet169	0.01	Default	Sequential	97.20	99.30	0.1187	0.0001	15436294
DenseNet121	0.01	Default	Linear	100	99.30	0.0056	0.0001	6960006
MNASNet0_5	0.001	Default	Sequential	100	96	0.0001	0.0001	2939054
MNASNet0_75	0.0001	Default	Sequential	100	99.30	0.0001	0.0001	3890750
MNASNet1_0	0.0001	0.0001	Linear	100	99.30	0.0024	0.0002	3890750
MNASNet1_3	0.0001	Default	Sequential	100	96	0.0001	0.0001	7002798
ShuffleNet_v2_x0_5	0.0001	Default	Linear	100	99.30	0.0003	0.0001	1372942
ShuffleNet_v2_x1_0	0.001	Default	Linear	100	99.30	0.0001	0.0001	2284754
ShuffleNet_v2_x1_5	0.001	0.0001	Linear	100	99.30	0.0001	0.0001	3509774
ShuffleNet_v2_x2_0	0.001	0.0001	Linear	100	96	0.0003	0.0001	7406290
EfficientNet_B0	0.001	Default	Linear	100	99.30	0.0001	0.0001	4015234
EfficientNet_B1	0.0001	Default	Linear	100	99.30	0.0001	0.0001	6520870
EfficientNet_B2	0.001	0.0001	Sequential	100	99.30	0.0001	0.0001	8587400
EfficientNet_B3	0.001	Default	Linear	100	99.30	0.0004	0.0001	12959918
EfficientNet_B4	0.001	0.0001	Linear	100	99. 30	0.0001	0.0001	17559374
EfficientNet_B5	0.0001	Default	Sequential	100	99.30	0.0307	0.0001	28353078
EfficientNet_B6	0.001	Default	Linear	92.31	99.30	0.2048	0.0001	40749534
EfficientNet_B7	0.001	Default	Sequential	100	96	0.0001	0.0001	67099222
SqueezeNet1_0	0.0001	Default	Linear	100	96	0.0001	0.0001	738502
SqueezeNet1_1	0.0001	Default	Linear	100	96	0.0001	0.0001	725574
VGG11	0.0001	Default	Sequential	100	96	0.0001	0.0001	139944966

RegNet_y_400mf	0.001	Default	Sequential	100	96	0.0001	0.0001	4049710
VGG11_bn	0.001	Default	Linear	100	96	0.0001	0.0001	128796422
RegNet_y_800mf	0.001	Default	Sequential	100	96	0.0001	0.0001	5882142
VGG_13	0.0001	Default	Sequential	100	96	0.0001	0.0001	140129478
RegNet_Y_1_6GF	0.001	Default	Sequential	100	96	0.0001	0.0001	10574684
VGG13_bn	0.0001	Default	Sequential	100	96	0.0001	0.0001	140135366
RegNet_y_3_2gf	0.001	Default	Sequential	100	96	0.0001	0.0001	19344160
RegNet_y_8gf	0.001	Default	Linear	100	96	0.0001	0.0001	37376574
VGG16	0.0001	Default	Sequential	100	96	0.0001	0.0001	145439174
VGG16_bn	0.0001	Default	Sequential	100	96	0.0001	0.0001	145447622
VGG19	0.0001	0.0001	Linear	100	96	0.0001	0.0001	139594822
VGG19_bn	0.0001	Default	Sequential	100	96	0.0001	0.0001	150759878

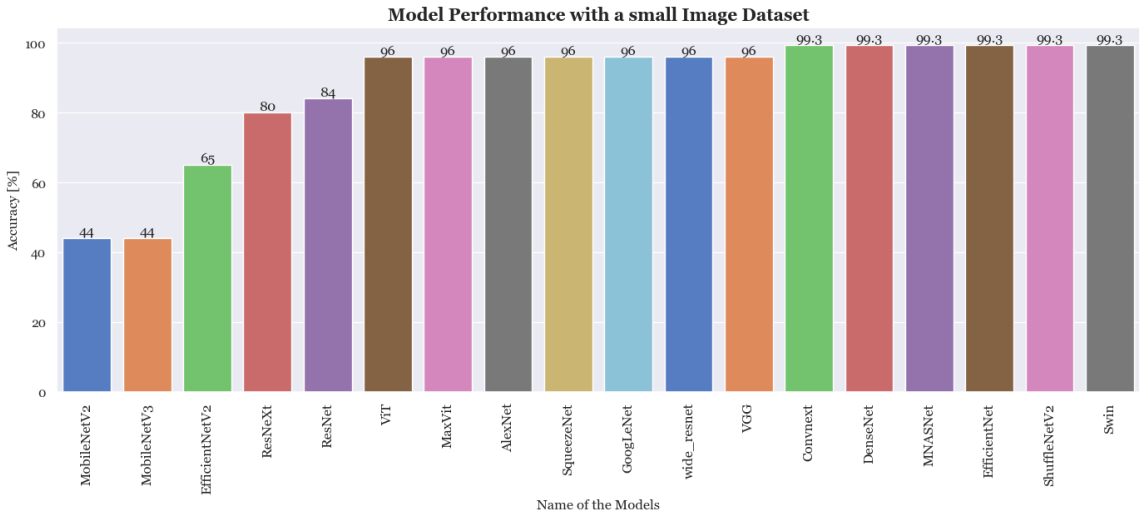
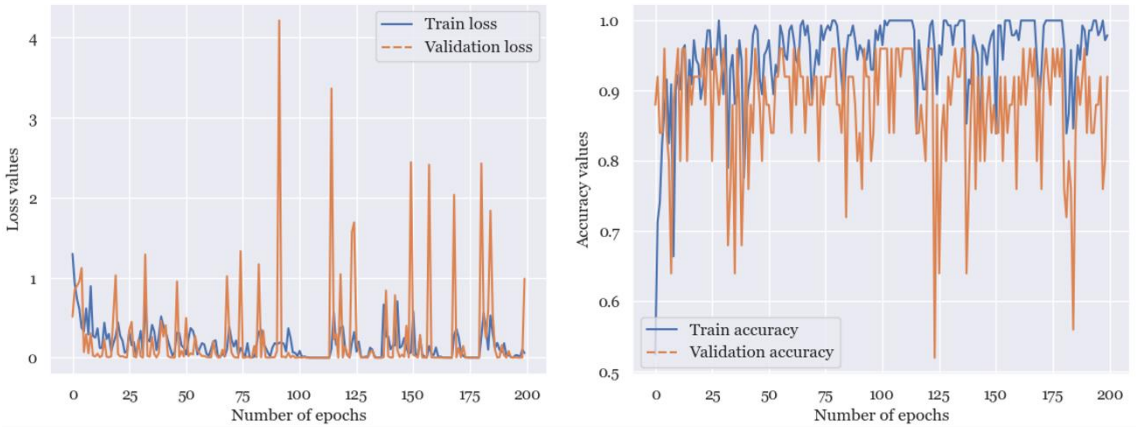


Figure 31. Highest accuracy achieved of the all the models simulated with the first and small dataset.

However, according to the results and the analysis of their accuracy and loss charts, three different models stood out. The three models with the best performance were the MaxVit, ShuffleNetV2, and EfficientNet.

The MaxVit model achieved a maximum accuracy of 96% in validation. This value was obtained in different epochs during the simulation. Figure 33 presents the results from loss (a) and accuracy (b) for the train (blue) and validation (orange) from the MaxVit model with 200 epochs, a learning rate of 0.001, variable weight decay, and a linear number of layers, as demonstrated in Table 8.

In neural networks, fully connected layers establish connections between every neuron in a layer to those in the previous and subsequent layers, facilitating comprehensive information exchange. Linear architecture involves linear connections between consecutive layers, particularly between the features generated by convolutional layers and the final output, which represents the number of classes.

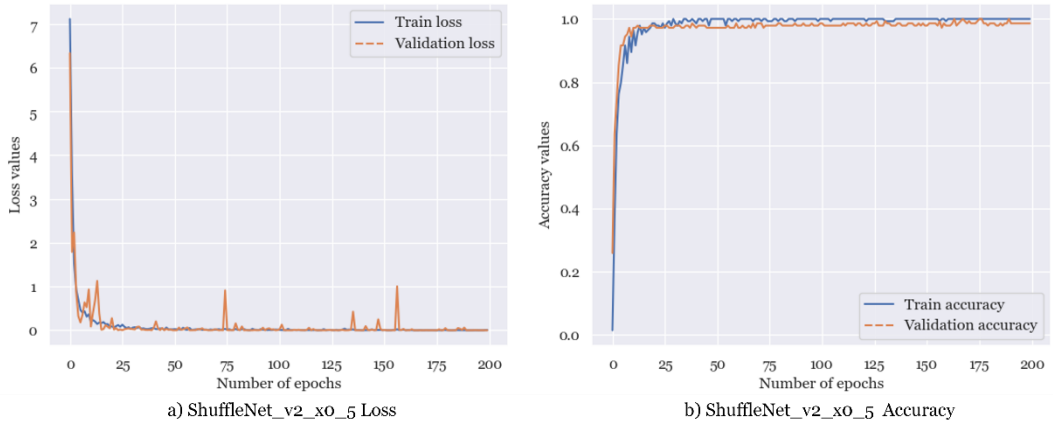


a) MaxVit Loss b) MaxVit Accuracy

Figure 32. Loss (a) and accuracy (b) results from training (blue) and validation (orange) using MaxVit model. The simulation was set with the following parameters: 200 epochs, learning rate of 0.001, weight decay variable and linear number of layers.

The ShuffleNetV2 model and its architectures, ShuffleNet_v2_x0_5, ShuffleNet_v2_x1_0, ShuffleNet_v2_x1_5, and ShuffleNet_v2_x2_0, also accomplished very good results.

The ShuffleNet_v2_x0_5 attained a maximum accuracy of 99.3% during validation. Figure 34 shows the loss (a) and accuracy (b) results during train (blue) and validation (orange) for the ShuffleNet_v2_x0_5 model with 200 epochs, learning rate of 0.0001, default weight decay, and a linear number of layers, as Table 8 shows.

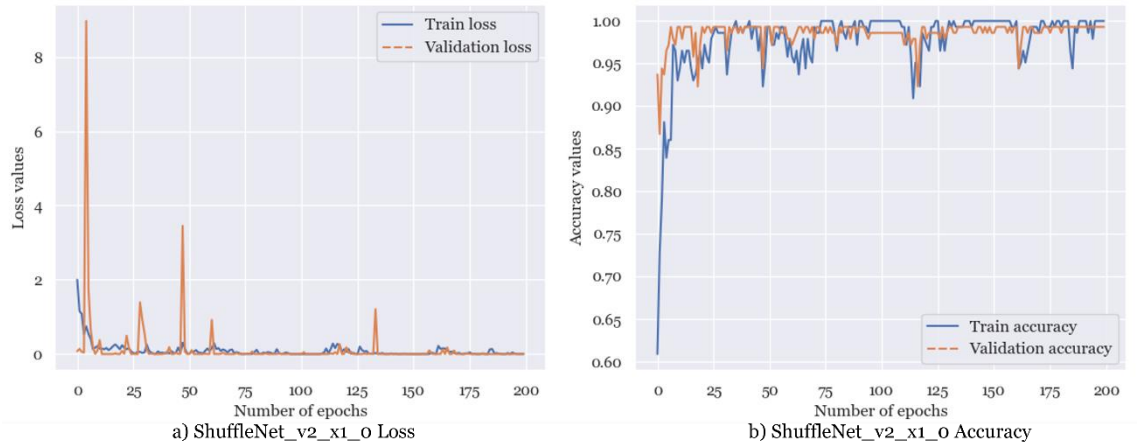


a) ShuffleNet_v2_x0_5 Loss b) ShuffleNet_v2_x0_5 Accuracy

Figure 33. Loss (a) and accuracy (b) results during train (blue) and validation (orange) from ShuffleNet_v2_x0_5, with 200 epochs, a learning rate of 0.0001, a default weight decay, and a linear number of layers.

The ShuffleNet_v2_x1_0 model obtained in validation a maximum accuracy of 99.3%. In Figure 35 are demonstrated the results from loss (a) and accuracy (b) obtained in train (blue) and

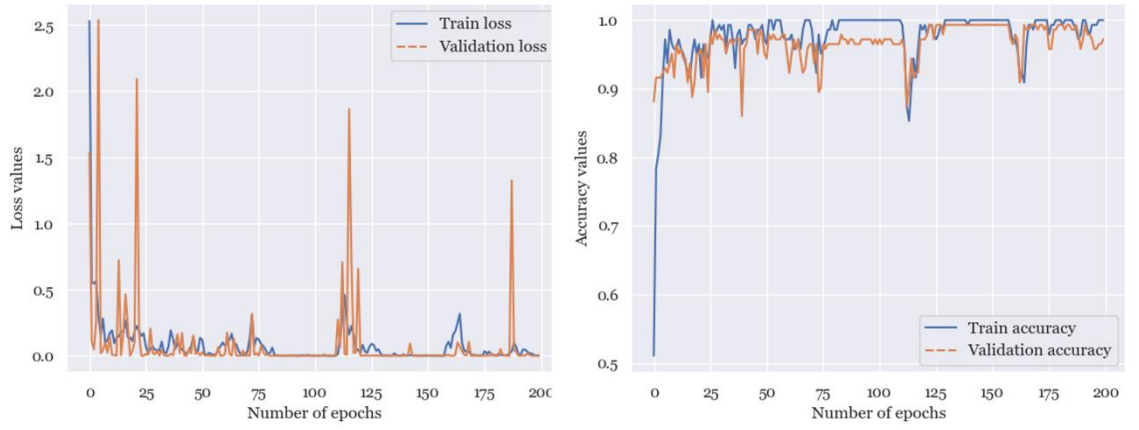
validation (orange) from ShuffleNet_v2_x1_0 model with 200 epochs, learning rate of 0.001, with default weight decay, and a linear number of layers, as represented in Table 8.



a) ShuffleNet_v2_x1_0 Loss b) ShuffleNet_v2_x1_0 Accuracy

Figure 34. Loss (a) and accuracy (b) results obtained in train (blue) and validation (orange) from ShuffleNet_v2_x1_0 model. The simulation ran with 200 epochs, a learning rate of 0.001, default weight decay, and a linear number of layers.

The ShuffleNet_v2_x1_5 model acquired a maximum accuracy of 99.3% in validation. In Figure 36 are demonstrated the results from loss (a) and accuracy (b) achieved in train (blue) and validation (orange), with 200 epochs, learning rate of 0.001, with variable weight decay, and a linear number of layers, as depicted in Table 8.

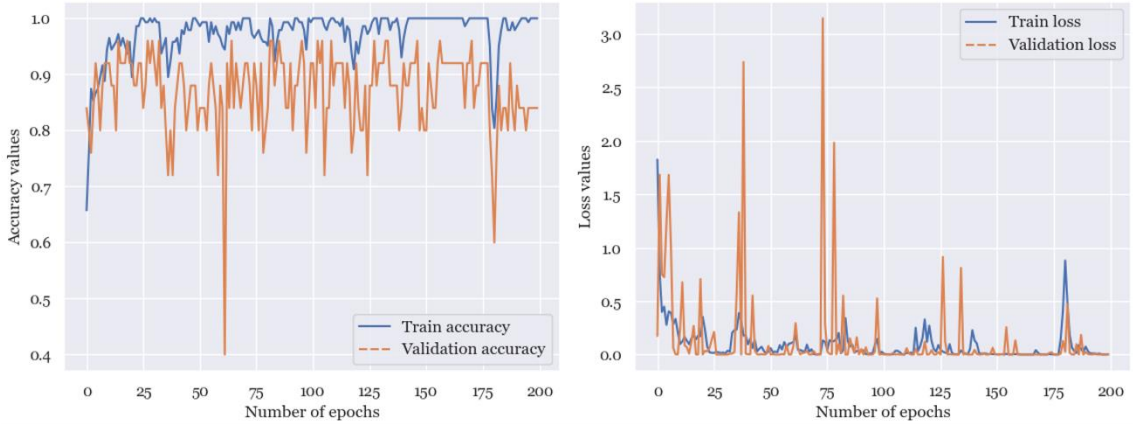


a) ShuffleNet_v2_x1_5 Loss b) ShuffleNet_v2_x1_5 Accuracy

Figure 35. Loss (a) and accuracy (b) results from train (blue) and validation (orange) ShuffleNet_v2_x1_5. The simulation was executed, achieving success with a configuration of 200 epochs, a learning rate set at 0.001, variable weight decay, and a linearly number of layers.

In the ShuffleNet_v2_x2_0 attained a maximum accuracy of 96% in validation. In Figure 37 are presented the results from loss (a) and accuracy (b) from train (blue) and validation (orange)

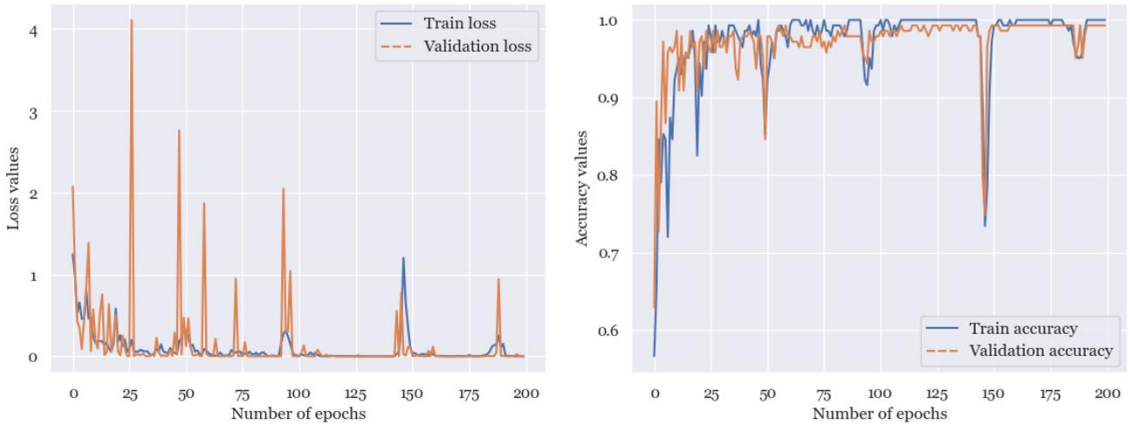
with 200 epochs, learning rate of 0.001, variable weight decay, and a linear number of layers, as indicated in Table 8.



a) ShuffleNet_v2_x2_0 Loss b) ShuffleNet_v2_x2_0 Accuracy

Figure 36. Loss (a) and accuracy (b) results from train (blue) and validation (orange) in ShuffleNet_v2_x2_0 model. The simulation was carried out with a configuration of 200 epochs, a learning rate of 0.001, variable weight decay, and a linear number of layers.

The EfficientNet_B0 model achieved the highest validation accuracy of 96%. Figure 38 displays the loss (a) and accuracy (b) results for train (blue) and validation (orange) in the model trained over 200 epochs with a learning rate of 0.001, incorporating default weight decay, and featuring a linear number of layers, as illustrated in Table 8.

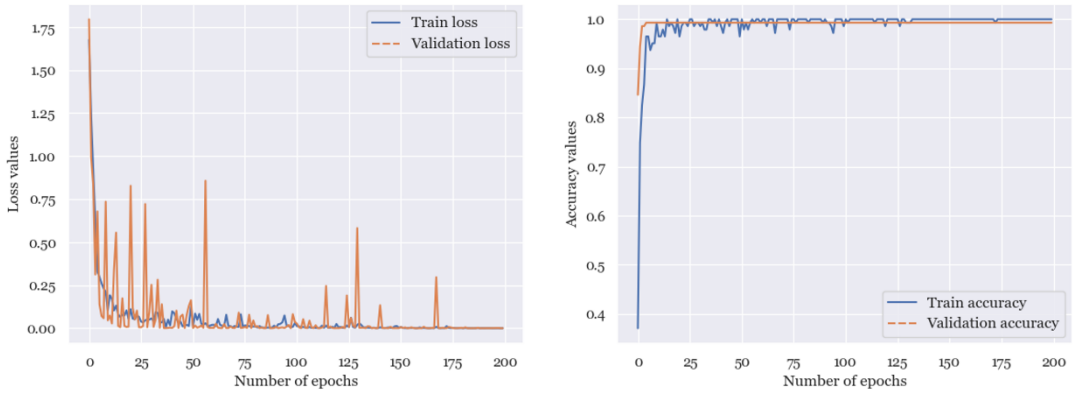


a) EfficientNet_B0 Loss b) EfficientNet_B0 Accuracy

Figure 37. Loss (a) and accuracy (b) results, depicted for both training (blue) and validation (orange), for the EfficientNet_B0 model. The simulation was executed with 200 epochs, a learning rate of 0.001, default weight decay, and a linear number of layers.

The EfficientNet_B1 model has reached the greatest validation accuracy of 96%. Figure 39 illustrates the loss (a) and accuracy (b) results for both training (blue) and validation (orange) in

the model trained across 200 epochs, utilizing a learning rate of 0.0001, incorporating default weight decay, and employing a linear number of layers, as showcased in Table 8.



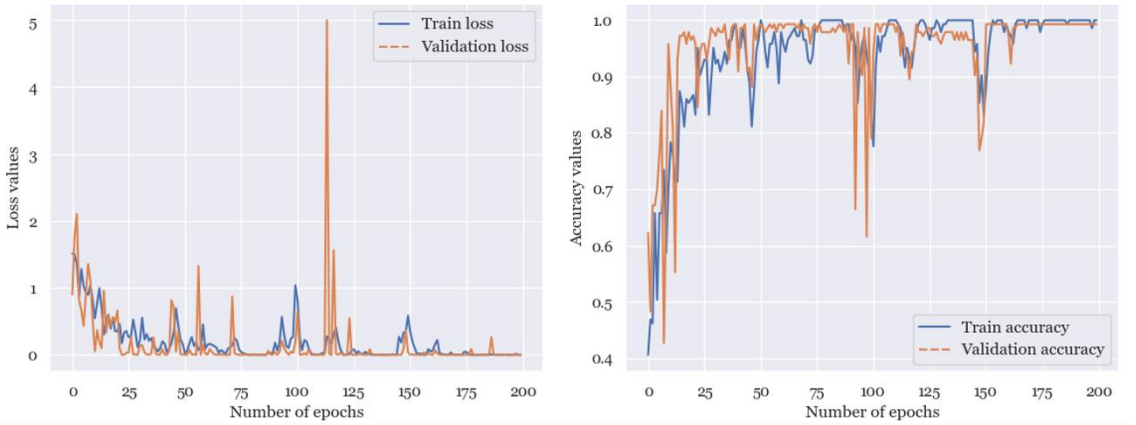
a) EfficientNet_B1 Loss

b) EfficientNet_B1 Accuracy

Figure 38. Loss (a) and accuracy (b) results, for both training (blue) and validation (orange), in the EfficientNet_B1 model. The hyperparameters used were 200 epochs, a learning rate of 0.0001, a default weight decay, and a linear number of layers.

The EfficientNet_B2 model has achieved the highest validation accuracy of 96%. Figure 40 displays the loss (a) and accuracy (b) results for both training (blue) and validation (orange) in the model trained over 200 epochs. It utilizes a learning rate of 0.001, incorporates variable weight decay, and a sequential number of layers, as demonstrated in Table 8.

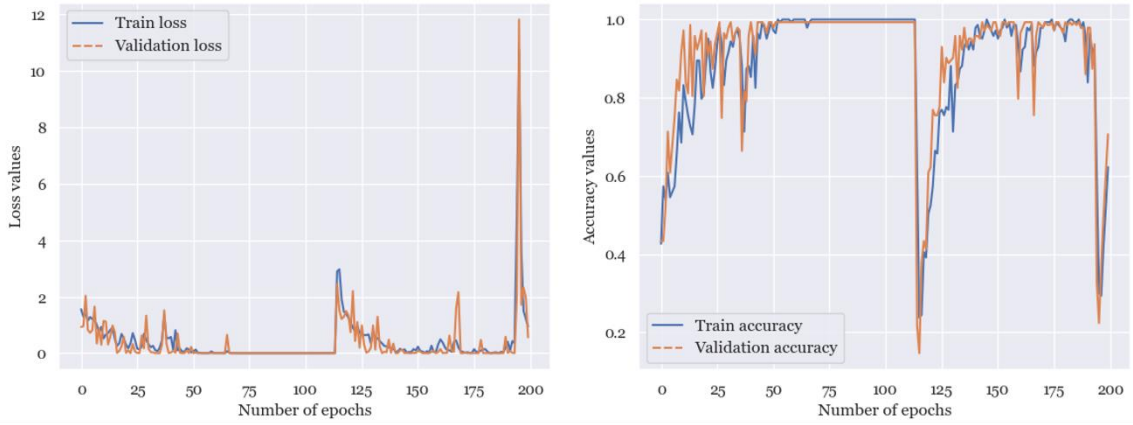
In neural networks, fully connected layers link every neuron in a layer to those in the preceding and succeeding layers, enabling thorough information exchange. The term sequential layers signify a linear sequence of layers, where the results generated by previous convolutional layers are linearly connected to subsequent layers. This process involves a gradual value reduction, influenced by factors like the number of bits, until reaching the final output representing the number of classes.



a) EfficientNet_B2 Loss b) EfficientNet_B2 Accuracy

Figure 39. For the EfficientNet_B2 model, the loss (a) and accuracy (b) results are presented, with training data displayed in blue and validation data in orange. The simulation was conducted with a setup that included 200 epochs, a learning rate of 0.001, variable weight decay, and a sequential number of layers.

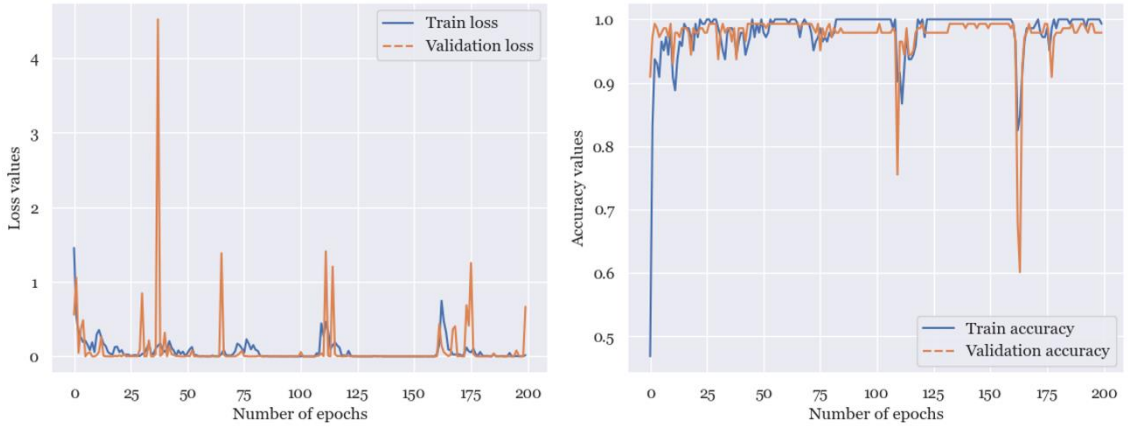
The EfficientNet_B3 model has reached an outstanding validation accuracy of 96%. Figure 41 showcases the loss (a) and accuracy (b) outcomes for both the training (blue) and validation (orange) of the model trained over 200 epochs. This training employs a learning rate of 0.001, default weight decay, and employs a linear number of layers, as shown in Table 8.



a) EfficientNet_B3 Loss b) EfficientNet_B3 Accuracy

Figure 40. Regarding the EfficientNet_B3 model, the results for loss (a) and accuracy (b), for training data (blue) and validation (orange). The simulation was carried out with a setup comprising 200 epochs, a learning rate of 0.001, default weight decay, and a linear number of layers.

The EfficientNet_B4 model has achieved a validation accuracy of 96%. Figure 42 presents the results for both training (blue) and validation (orange) in terms of loss (a) and accuracy (b) for the model trained over a span of 200 epochs. This training regimen utilizes a learning rate of 0.001, incorporates variable weight decay, and follows a linear number of layers, as outlined in Table 8.

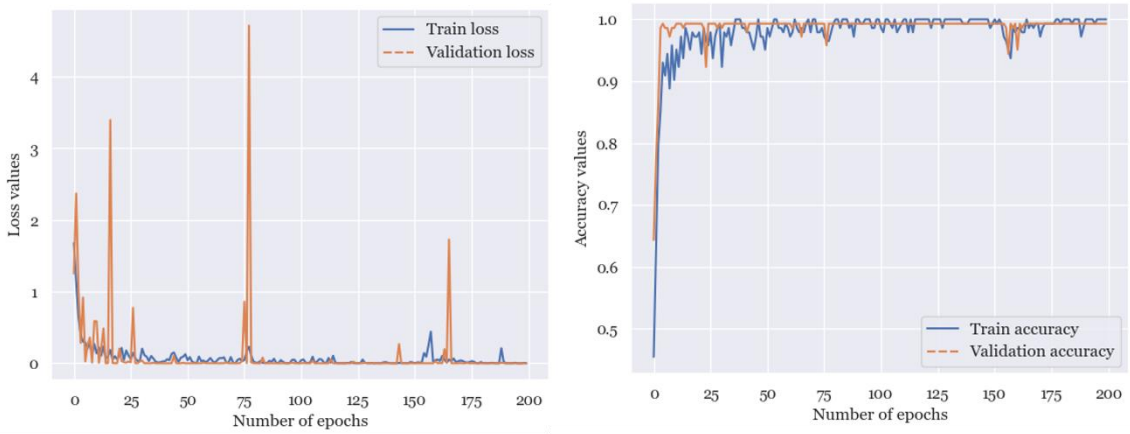


a) EfficientNet_B4 Loss

b) EfficientNet_B4 Accuracy

Figure 41. Results of the EfficientNet_B4 model, for loss (a) and accuracy (b) concerning both the training data (blue) and validation data (orange). The simulation was performed with a configuration that included 200 epochs, a learning rate of 0.001, variable weight decay, and a linear number of layers.

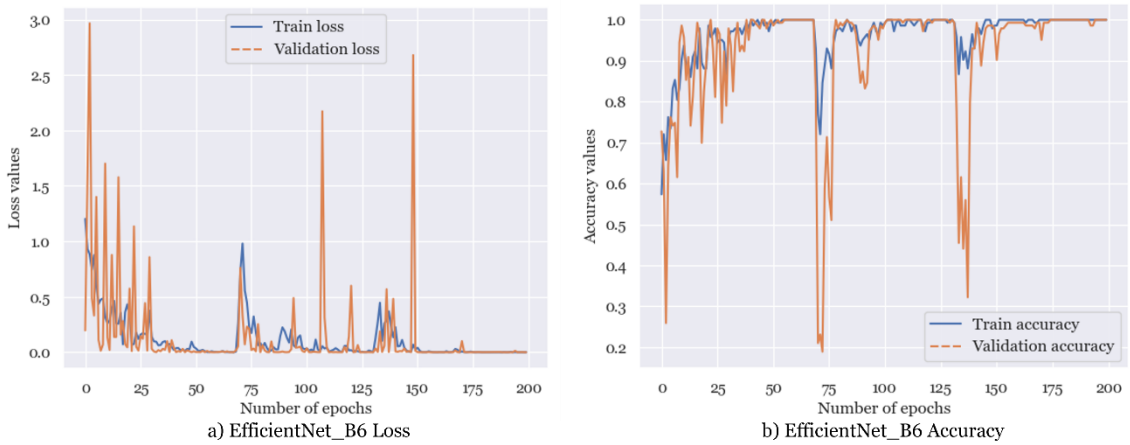
The EfficientNet_B5 model has successfully attained a validation accuracy of 96%. Figure 43 displays the outcomes for both training (blue) and validation (orange) in relation to loss (a) and accuracy (b) for the model trained across a duration of 200 epochs. This training approach employs a learning rate of 0.0001, integrates default weight decay, and employs a sequential number of layers, as detailed in Table 8.



a) EfficientNet_B5 Loss b) EfficientNet_B5 Accuracy

Figure 42. The outcomes of the EfficientNet_B5 model presented for both loss (a) and accuracy (b), encompassing both the training data (blue) and validation data (orange). The simulation was executed with a setup that consisted of 200 epochs, a learning rate of 0.0001, default weight decay, and a sequential number of layers.

The validation accuracy of the EfficientNet_B6 model has successfully reached 96%. Figure 44 provides a visualization of the outcomes for both training (blue) and validation (orange) with regard to loss (a) and accuracy (b). These results are derived from the model's training spanning 200 epochs, incorporating a learning rate of 0.001, applying default weight decay, and featuring a linear number of layers, as Table 8 shows.



a) EfficientNet_B6 Loss b) EfficientNet_B6 Accuracy

Figure 43. The results for both loss (a) and accuracy (b) of the EfficientNet_B6 model are displayed, covering both the training data (blue) and validation data (orange). The simulation was conducted with 200 epochs, a learning rate set at 0.001, default weight decay, and a linear number of layers.

The results presented above do not pass for a smoothing process, so the values are the real results at the end of the simulations. Because of this, the graphs show several peaks. In the first 25 epochs, the models had already reached their maximum accuracy. From this point onwards, the network learned some classes better and ended up unlearning others, which can be seen in the upward and downward spikes, some of which were drastic. For example, the models EfficientNet_B1, EfficientNet_B2, and ShuffleNet_V2_X0_5 were the models where accuracy was more consistent. Of all the models, these three achieved good accuracies. This means that these models did not unlearn during training. However, the other models also performed well, but their unlearning was noticeable.

4.1.1. MaxVit, ShuffleNet and EfficientNet Models in depth

Convolutional layers, pooling layers, and fully linked layers are components of conventional CNN models. The model is computationally complex since it uses big convolution kernels and pooling layers. Due to the performance limitations of some application contexts, such as mobile devices, the model needs to be highly precise and compact (Ma *et al.,* 2018).

ShuffleNet V2 solves all these issues without utilizing big convolution kernels or pooling layers. In place of the conventional convolutional layer, a depth-wise convolution and a 1x1 small convolution kernel are used (Ma *et al.,* 2018).

The depth-wise convolution kernel size is 3x3, and since each convolution kernel handles one input channel, there are exactly as many convolution kernels as input channels. The features of the output of depth-wise convolution are combined using 1x1 convolution. Without changing the size of the output feature graph, this increases the network's nonlinearity and expressiveness. ShuffleNet V2 down-samples the feature by altering the depth-wise convolutional step in place of the conventional pooling layer (Liu *et al.,* 2019).

The ShuffleNet V2 network architecture may be seen in Figure 45, and it consists mostly of two stacked basic units. The component in Figure 45 (a) corresponds to the ShuffleNet V2 basic unit and uses "Channel Split", "Concat", and "Channel Shuffle" to make the transfer of feature information between several channels easier. Beginning with the "Channel Spilt" procedure, the input feature map's channel dimension is split into two equal branches, one branch is left unchanged, while the other branch has three convolution layers. The two branches' outputs are then combined using the "Concat" method following convolution. Concatenating the two branches results in the same number of channels. Following that, the same "channel shuffle" mechanism is used to enable communication between the two branches (Liu *et al.,* 2019; Ma *et al.,* 2018).

The unit is slightly modified for spatial down sampling. Figure 45 (b) shows the modifications. There is no longer a channel splitter. Consequently, there are double output channels. Furthermore, "Concat", "Channel Shuffle", and "Channel Split," three subsequent element-wise operations, are combined into one (Liu *et al.,* 2019 ;Ma *et al.,* 2018).

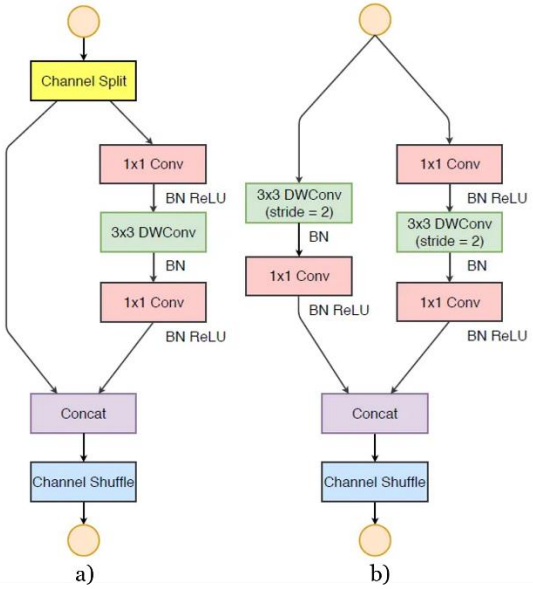


Figure 44. ShuffleNet V2 architecture: (a) basic unit of ShuffleNetV2; (b) Basic unit for spatial down-sampling (Liu *et al.*, 2019).

A significant innovation made by ShuffleNet is the channel shuffle technique, which may efficiently exchange channel information during the feature extraction process. Figure 46 illustrates the channel shuffle's fundamental idea. The features following group convolution are connected in the input and output dimensions because the reorganized feature set contains the channel features of each grouping (Chen *et al.*, 2022).

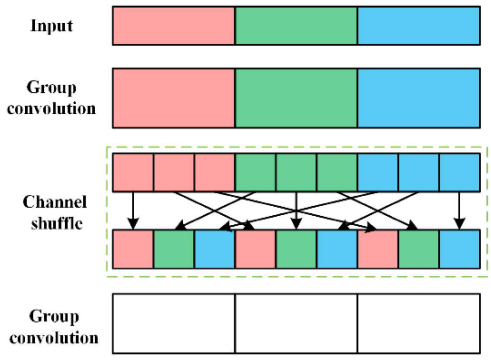


Figure 45. The fundamental idea behind channel shuffle (Chen *et al.*, 2022).

The ShuffleNetV2 versions represented by the x0.5, x1.0, x1.5, and x2.0 have a distinct model complexity.

ShuffleNetV2 x0.5: This variant has approximately half the number of channels and model parameters compared to the baseline ShuffleNetV2. It's suitable for resource-constrained

environments, such as mobile devices or embedded systems, where memory and computation power are limited. It sacrifices some accuracy for improved efficiency (Chen *et al.,* 2022).

ShuffleNetV2 x1.0: With the "x1.0" designation, this variant maintains a similar number of channels and model parameters as the original ShuffleNetV2. It strikes a good balance between model size and accuracy and is commonly used in various computer vision tasks (Chen *et al.,* 2022).

ShuffleNetV2 x1.5: The "x1.5" variant increases the number of channels and model parameters by 1.5 times compared to the baseline ShuffleNetV2. It can achieve better accuracy but requires more computational resources than the x1.0 variant (Chen *et al.,* 2022).

ShuffleNetV2 x2.0: With "x2.0," these variant doubles the number of channels and model parameters compared to ShuffleNetV2 x1.0. It offers the highest accuracy among the ShuffleNetV2 variants but demands significant computational power and memory (Chen *et al.,* 2022).

The computer vision community has recently paid a lot of attention to Transformers. ViT is made to split up images into patches and feed those patches into a transformer model to learn how to represent those image patches globally. In contrast, the MaxViT model's multi-axis attention allows it to capture both local and global interactions between image patches. Additionally, unlike typical ViTs, MaxVit employs a hierarchical patch partitioning strategy to divide the input image into patches of varying sizes, allowing the model to capture both fine- and coarse-grained properties in the input data. Depending on how complex the input photos are, the size of the patches is dynamically modified. MaxViT is additionally made to be extremely scalable, allowing it to be used with very big images (Hossain *et al.,* 2023).

Thus, MaxVit is a model of effective and scalable attention. Blocked local attention and dilated global attention are the two halves of this model. At every input resolution, these design choices permit linearly complex global-local spatial interactions (Tu *et al.,* 2022).

The MaxViT Architecture combines multi-axis attention with convolution. First, a convolutional stem is first used to process the input image. The output of the stem then passes through a series of Max-SA, which is a MBConv block with a SE module followed by a Max-SA module (Tu *et al.,* 2022). MaxVit architecture is shown in Figure 47.

The Max-SA block applies self-attention along a single axis of the feature map at a time and then combines the attention maps of multiple axes to achieve a global receptive field (Tu *et al.,* 2022).

The first part of Max-SA is the MBConv block that incorporates a number of methods to lower the computational expense while preserving or enhancing model performance. A depthwise separable convolution, a simplified variation of the normal convolution process, is the first step in the MBConv block (Tu *et al.,* 2022).

The inverted bottleneck concept calls for increasing the input features' number of channels before conducting depthwise convolution. The model can capture more complicated patterns because of this expansion factor without considerably raising the computational cost (Tu *et al.,* 2022).

After the depthwise convolution, a squeeze-and-excitation block may be included in some MBConv block versions. By learning channel-specific scaling factors, the SE block recalibrates the channel-wise feature responses, enabling the model to concentrate on more informative channels (Tu *et al.,* 2022).

The block attention involves using a fixed-size local attention window to promote close-proximity interactions. These methods aid the model's ability to concentrate on particular spatial areas of the input, improving its capacity to identify objects or patterns within those areas (Tu *et al.,* 2022).

Global attention is performed sparsely by the grid. Grid attention begins with a data structure that resembles a grid as input. Grid attention utilizes query, key, and value vectors for each position in the grid, just like conventional attention techniques. To calculate attention scores, these vectors are often derived from the input data. These ratings show how relevant or comparable various grid positions are to one another. These scores indicate the degree to which each position in the grid should "pay attention" to other positions (Tu *et al.,* 2022).

The values at each position in the grid are aggregated or merged to create the output for that position using the attention weights. The outcome is an output grid with values that have been changed to reflect the connections and dependencies found in the input grid (Tu *et al.,* 2022).

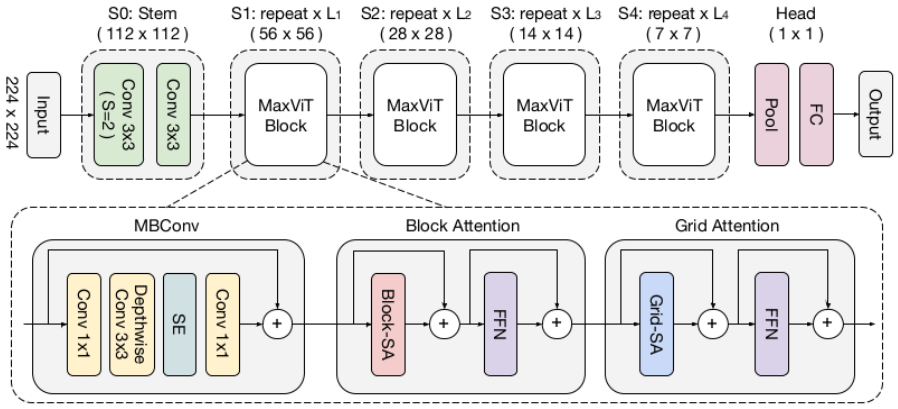


Figure 46. MaxVit architecture (Tu *et al.,* 2022).

EfficientNet is a convolutional neural network that employs the concept of "compound scaling" to address the trade-off between model size, accuracy, and computational efficiency. Compound scaling involves adjusting three crucial dimensions of the neural network: width, depth, and resolution (Tan & Le, 2020).

Width scaling involves modifying the number of channels in each layer of the network. Increasing the width enhances the model's ability to capture complex patterns and features, thereby improving accuracy. Conversely, reducing width yields a more lightweight model suitable for resource-constrained environments (Tan & Le, 2020).

Depth scaling concerns the total number of layers in the network. Deeper models can capture intricate data representations but demand greater computational resources. In contrast, shallower models are computationally efficient but may sacrifice accuracy (Tan & Le, 2020).

Resolution scaling entails altering the input image's size (width x height). Higher-resolution images provide richer information, potentially leading to improved performance. However, they also require more memory and computational power. Lower-resolution images consume fewer resources but may result in a loss of fine-grained details (Tan & Le, 2020).

The compound scaling process starts with a baseline model, initially a reasonably sized neural network effective for a given task but not necessarily optimized for efficiency. A compound coefficient (φ) is introduced as a user-defined parameter to determine how the neural network's dimensions are scaled uniformly – width, depth, and resolution. Adjusting this φ value allows control over the model's overall complexity and resource requirements. The scaling factors for each dimension are derived from the φ (Tan & Le, 2020).

The key concept behind compound scaling is to balance and coordinate the scaling of the baseline model's dimensions. The width scaling increases the network's width proportionally by raising φ to the power of a specific exponent (θ). Depth scaling similar to width scaling, it adjusts the network's depth by raising φ to another exponent (typically β). Last, resolution scaling scales the input image size by multiplying the original resolution by φ raised to a different exponent (commonly γ). Thus, θ, β and γ are constant coefficients determined by a small grid search on the original small model (Tan & Le, 2020).

The next step involves determining the optimal exponents (α, β, γ) to achieve the best balance between model accuracy and computational efficiency. This is typically done through empirical grid searches or optimization processes (Tan & Le, 2020).

Once these scaling factors for width, depth, and resolution are established, they are applied to the baseline model. This transformed model is now an EfficientNet with a specific φ value. Smaller φ values yield lightweight and efficient models, while larger φ values result in more powerful but computationally demanding models (Tan & Le, 2020).

Compound scaling primarily focuses on adjusting the width, depth, and resolution of the neural network but does not alter the specific operations used within a layer of the network. It is a strategy for enhancing the overall architecture's efficiency and computational characteristics while keeping the fundamental operations, such as convolution and pooling, consistent across the layers (Tan & Le, 2020).

EfficientNet utilizes a combination of MBConv layers and SE optimization to boost its performance.

The MBConv layer, a foundational component of the EfficientNet architecture, draws inspiration from the inverted residual blocks found in MobileNetV2. However, it introduces some modifications to enhance efficiency. The MBConv layer starts with a depth-wise convolution, followed by a point-wise convolution (1x1 convolution) to expand the number of channels. Finally, another 1x1 convolution is applied to reduce the channels back to their original count. This

bottleneck design enables the model to learn effectively while retaining a robust representational capacity (Tan & Le, 2020).

In addition to MBConv layers, EfficientNet incorporates SE blocks, which aid the model in concentrating on crucial features while suppressing less important ones. The SE block employs global average pooling to condense the spatial dimensions of the feature map to a single channel. This is followed by two fully connected layers to refine feature importance. These layers allow the model to learn channel-wise feature dependencies and create attention weights that are multiplied by the original feature map, emphasizing important information (Tan & Le, 2020).

Figure 48 shows an illustration of the EfficientNet architecture.

EfficientNet is a family of CNN architectures designed to balance model size and accuracy effectively. The variants of EfficientNet, B0, B1, B2, B3, B4, B5, and B6, represent different levels of model complexity, with B0 being the smallest and B6 being the largest (Tan & Le, 2020).

The EfficientNet B0 is the smallest variant with fewer parameters, making it suitable for resource-constrained environments. It provides a good balance between model size and accuracy.

EfficientNet B1 is slightly larger than B0, offering improved accuracy due to more parameters and computational complexity while remaining relatively efficient (Tan & Le, 2020).

EfficientNet B2: Larger than B1 and B0, suitable for tasks where higher accuracy is desired, and more computational resources are available (Tan & Le, 2020).

EfficientNet B3 is larger than B2, and continues to increase model complexity, offering even better accuracy but requiring more computational resources and memory (Tan & Le, 2020).

EfficientNet B4 has a significant step up in complexity compared to B3, making it suitable for tasks demanding high accuracy and having substantial computational resources (Tan & Le, 2020).

The EfficientNet B5 was designed for applications prioritizing extremely high accuracy, it has a more substantial number of parameters and computational demands (Tan & Le, 2020).

The EfficientNet B6 is the largest and most complex variant in the family, offering good accuracy but coming with the highest computational and memory requirements (Tan & Le, 2020).

By following the compound scaling method, EfficientNet can efficiently explore a wide range of model architectures that strike the perfect balance between accuracy and resource consumption (Tan & Le, 2020).

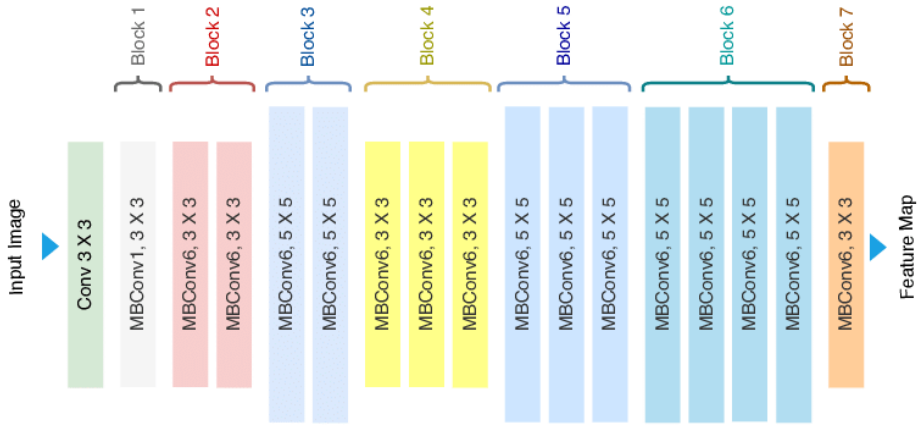


Figure 47. EfficientNet architecture (Tan & Le, 2020).

4.2. Experiment 2: Evaluating Model Performance Across Epochs and Establishing Baselines

With the best models chosen and data analysed in this part of the work, only the superior results were simulated. Thus, the models MaxVit, ShuffleNetV2 and EfficientNet were used.

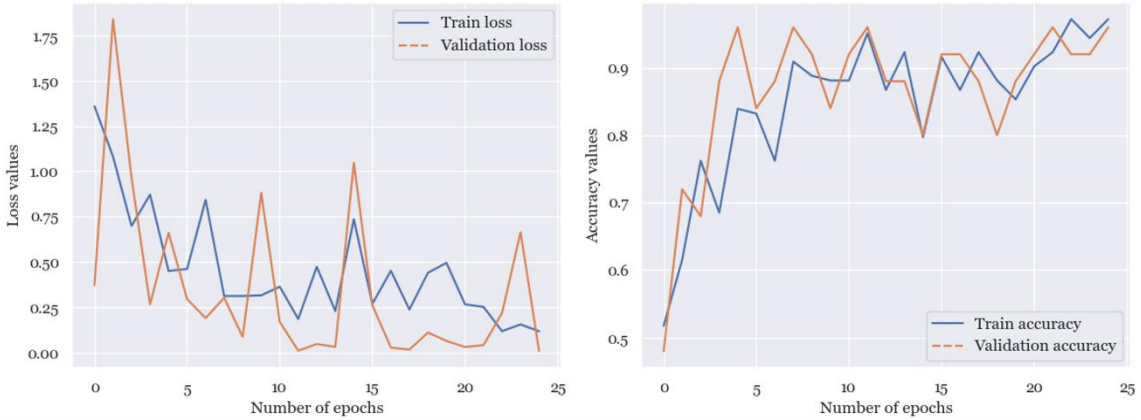
According to most of the results, 25 epochs were enough to achieve the maximum result. So, in this part of the study, the twelve models were simulated with only 25 epochs and the combination of hyperparameters was used to accomplish the best outcome.

The first simulations occurred with the same dataset, same hyperparameters and 25 epochs instead of 200. Table 9 shows the results that each model achieved with 25 epochs.

Table 9. Performance evaluation of the top twelve models: 25 epochs simulation results.

Models	Learning rate	Weight decay	Number of layers	Best acc train [%]	Best acc validation [%]	Best loss train	Best loss validation	Number of parameters
MaxVit	0.001	0.0001	Linear	97,2	96	0.1167	0.0670	30410189
ShuffleNet_v2_x0_5	0.0001	Default	Linear	100	84	0.0467	0.0180	1371917
ShuffleNet_v2_x1_0	0.001	Default	Linear	100	96	0.0001	0.0001	2283729
ShuffleNet_v2_x1_5	0.001	0.0001	Linear	100	96	0.0123	0.0003	3508749
ShuffleNet_v2_x2_0	0.001	0.0001	Linear	97,2	96	0.0832	0.0002	7404241
EfficientNet_b0	0.001	Default	Linear	100	96	0.0308	0.0138	4013953
EfficientNet_b1	0.0001	Default	Linear	99,31	96	0.0623	0.6131	6519589
EfficientNet_b2	0.001	0.0001	Sequential	93.79	92	0.1983	0.0041	8587271
EfficientNet_b3	0.001	Default	Linear	99.31	96	0.0300	0.0180	10703917
EfficientNet_b4	0.001	0.0001	Linear	100	96	0.150	0.0005	17557581
EfficientNet_b5	0.0001	Default	Sequential	98.62	96	0.0502	0.0523	31128629
EfficientNet_b6	0.001	Default	Linear	100	92	0.0134	0.0409	40747229

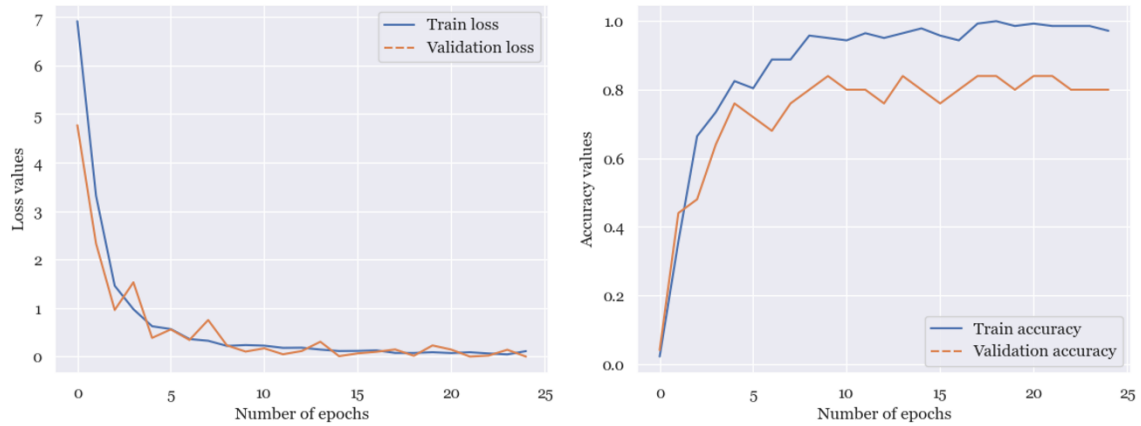
The MaxVit model achieved a maximum validation accuracy of 96% through multiple epochs during the simulation. Figure 49 illustrates the corresponding loss (a) and accuracy (b) results for both training (in blue) and validation (in orange) phases of the MaxVit model, trained over 25 epochs. This training utilized a learning rate of 0.001, incorporated variable weight decay, and employed a linear sequence of layers, as Table 9 shows.



a) MaxVit Loss b) MaxVit Accuracy

Figure 48. Loss (a) and accuracy (b) results, displayed for both training (blue) and validation (orange) for the MaxVit model. The simulation was conducted with the following parameters: 25 epochs, a learning rate of 0.001, variable weight decay, and a linear progression in the number of layers.

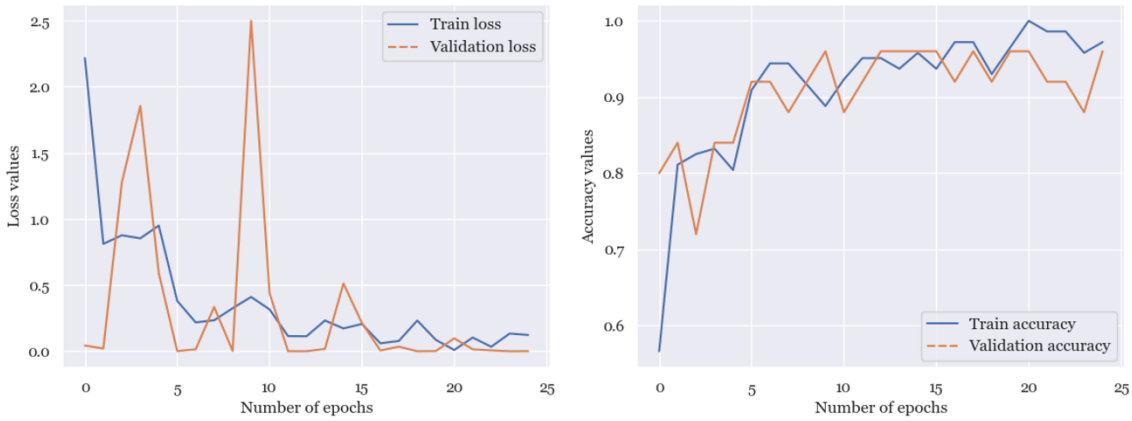
The ShuffleNet_v2_x0_5 model has achieved a remarkable 84% accuracy in validation. Figure 50 provides a visual representation of the loss and accuracy results for the ShuffleNet_v2_x0.5 model over 25 epochs. This model was trained over 200 epochs, utilizing a learning rate of 0.0001, default weight decay, and a linear sequence of layers. As Table 9 shows.



a) ShuffleNet_v2_x0_5 Loss b) ShuffleNet_v2_x0_5 Accuracy

Figure 49. Loss (a) and accuracy (b) outcomes for training (blue) and validation (orange) are depicted for the ShuffleNet_v2_x0_5 model. This model was trained over 25 epochs, with a learning rate of 0.0001, default weight decay, and a linear progression in the number of layers.

The ShuffleNet_v2_x1_0 model achieved a validation accuracy of 96%. Figure 51 illustrates the training and validation results for loss (a) and accuracy (b) using the ShuffleNet_v2_x1_0 architecture with 25 epochs, a learning rate of 0.001, default weight decay, and a linear number of layers. In the Figure, the blue curve represents the training performance, while the orange curve represents the validation performance. As illustrated in Table 9.



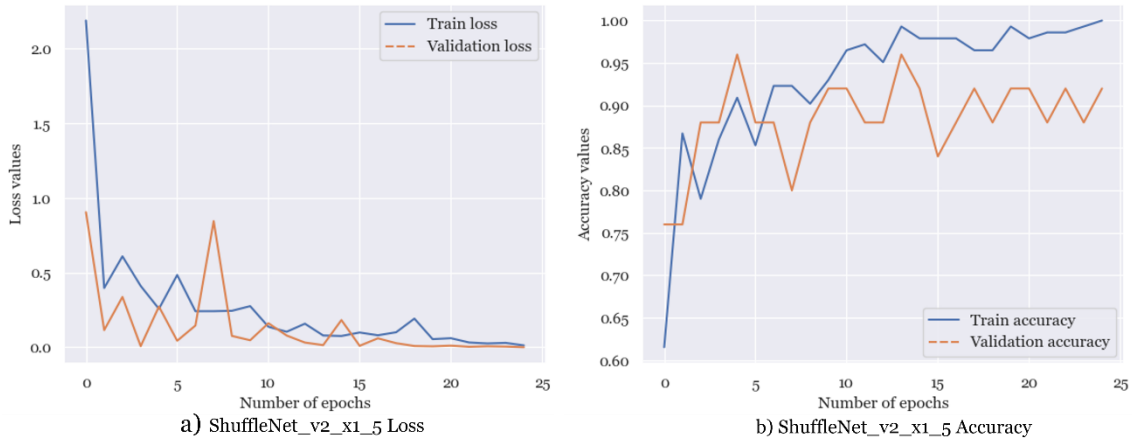
a) ShuffleNet_v2_x1_0 Loss

b) ShuffleNet_v2_x1_0 Accuracy

Figure 50. Loss (a) and accuracy (b) results for training (blue) and validation (orange) from ShuffleNet_v2_x1_0. The simulation ran with 25 epochs, a learning rate of 0.0001, default weight decay, and a linear progression in the number of layers.

The ShuffleNet_v2_x1_5 acquired a maximum accuracy of 96% in validation.

In Figure 52 are shown the results from loss (a) and accuracy (b) for train (blue) and validation (orange) from ShuffleNet_v2_x1_5 model with 25 epochs, learning rate of 0.001, variable weight decay and a linear layer architecture. As depicted in Table 9.



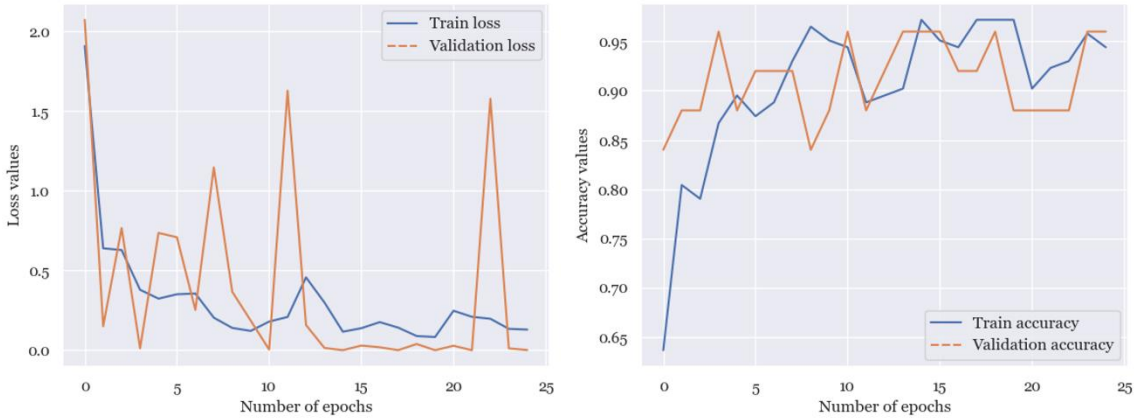
a) ShuffleNet_v2_x1_5 Loss

b) ShuffleNet_v2_x1_5 Accuracy

Figure 51. Loss (a) and accuracy (b) results from train (blue) and validation (orange) ShuffleNet_v2_x1_5 model with 25 epochs, learning rate of 0.001, variable weight decay and a linear layer architecture.

In the ShuffleNet_v2_x2_0 a maximum accuracy of 96% in validation was obtained.

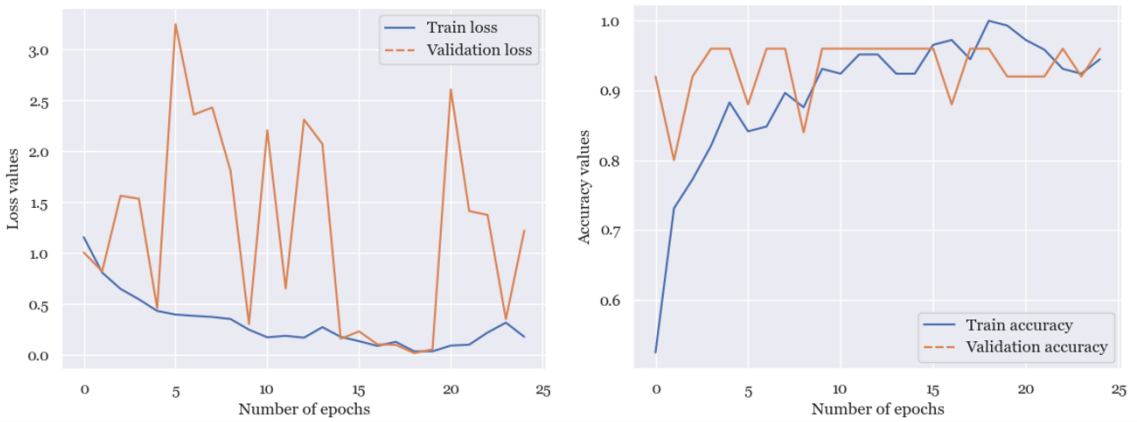
In Figure 53 are presented the results from loss (a) and accuracy (b) for train (blue) and validation (orange) from ShuffleNet_v2_x2_0 model with 25 epochs, learning rate of 0.001, variable weight decay and a liner architecture of layers. As Table 9 shows.



a) ShuffleNet_v2_x2_0 Loss b) ShuffleNet_v2_x2_0 Accuracy

Figure 52. Loss (a) and accuracy (b) results from ShuffleNet_v2_x2_0 during training (blue) and validation (orange), using 25 epochs, a learning rate of 0.001, flexible weight decay, and a linear layer architecture.

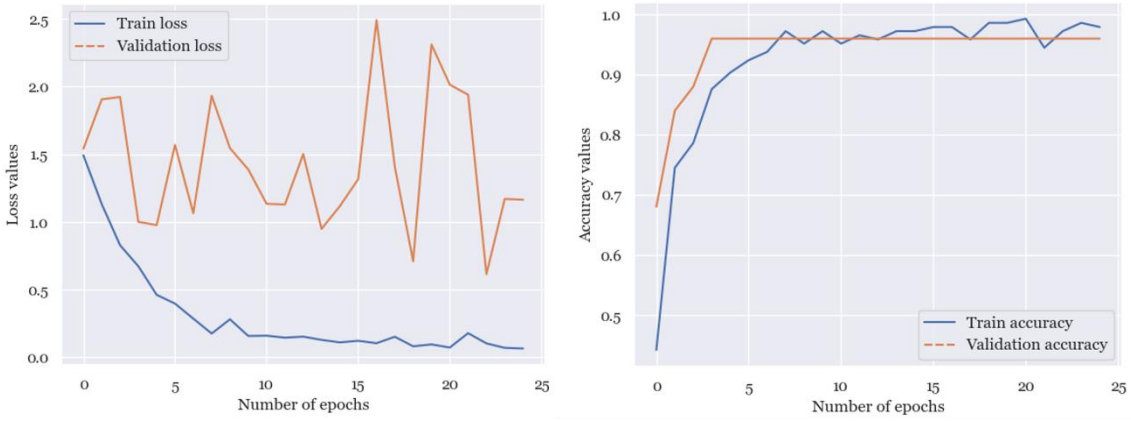
The EfficientNet_B0 model achieved the highest validation accuracy of 96%. Figure 54 displays the loss (a) and accuracy (b) results for train (blue) and validation (orange) in the model trained over 25 epochs with a learning rate of 0.001, incorporating default weight decay, and featuring a linear number of layers, as illustrated in Table 9.



a) EfficientNet_B0 Loss a) EfficientNet_B0 Accuracy

Figure 53. Results from EfficientNet_B0 training (blue) and validation (orange) phases utilizing 25 epochs, a learning rate of 0.001, default weight decay, and a linear layer architecture are shown in loss (a) and accuracy (b).

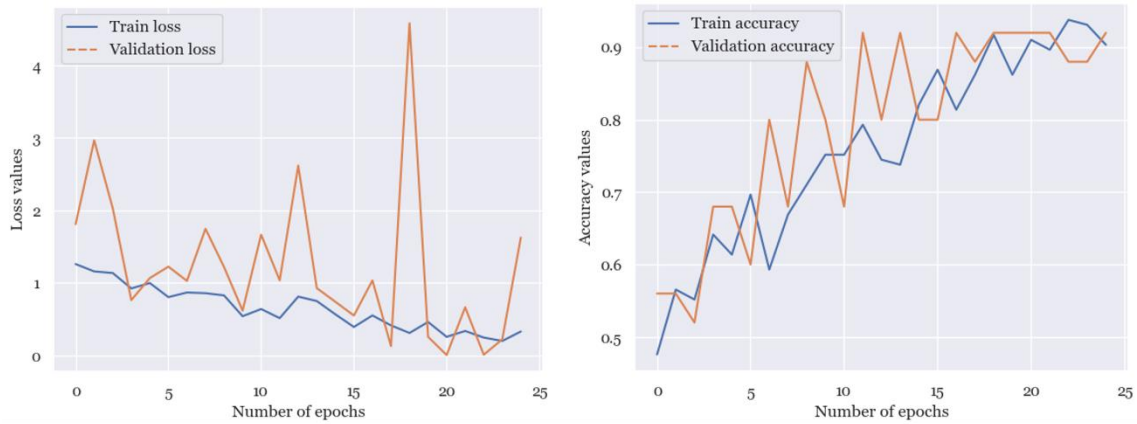
The model EfficientNet_B1 has a 96% accuracy, as highest validation accuracy. As shown in Table 9, the model was trained across 25 epochs using a learning rate of 0.0001, default weight decay, and a linear number of layers. Figure 55 shows the results for loss (a) and accuracy (b) for both training (blue) and validation (orange).



a) EfficientNet_B1 Loss a) EfficientNet_B1 Accuracy

Figure 54. Loss (a) and accuracy (b) results of the EfficientNet_B1 training (blue) and validation (orange) phases using 25 epochs, a learning rate of 0.0001, default weight decay, and a linear layer architecture.

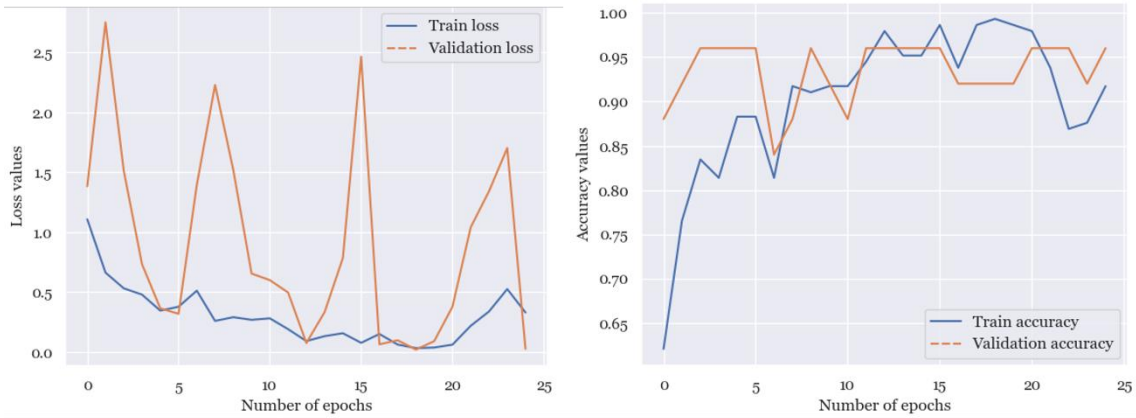
The model EfficientNet_B2, achieved a 96% accuracy. The model was trained across 25 epochs, and Figure 56 shows the accuracy (b) and loss (a) results for both training (blue) and validation (orange). As shown in Table 9, it employs a learning rate of 0.001, variable weight decay, and a sequential number of layers.



a) EfficientNet_B2 Loss a) EfficientNet_B2 Accuracy

Figure 55. The results for both loss (a) and accuracy (b) of the EfficientNet_B2 model are visualized, encompassing both the training data (blue) and validation data (orange). The simulation was carried out using 25 epochs, a learning rate of 0.001, variable weight decay, and a sequential layer structure.

The validation accuracy achieved by the EfficientNet_B3 model is 96%. Figure 57 illustrates the outcomes for loss (a) and accuracy (b) in both the training (blue) and validation (orange) datasets during the 25-epoch training process. This simulation utilizes a learning rate of 0.001, default weight decay, and employs a linear layer structure, as outlined in Table 9.

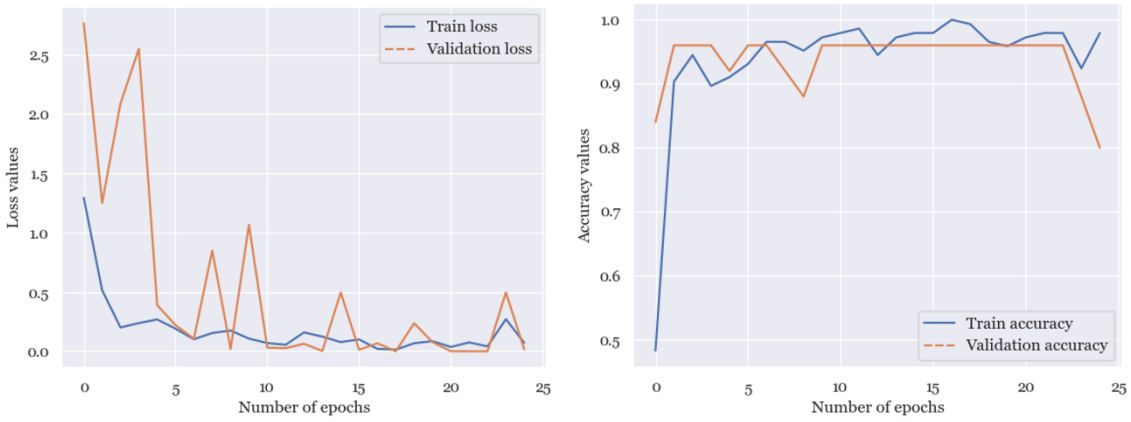


a) EfficientNet_B3 Loss

a) EfficientNet_B3 Accuracy

Figure 56. The loss (a) and accuracy (b) results for the EfficientNet_B3 model are displayed, covering both the training data (blue) and validation data (orange). The simulation was conducted with 25 epochs, a learning rate of 0.001, default weight decay, and a linear layer structure.

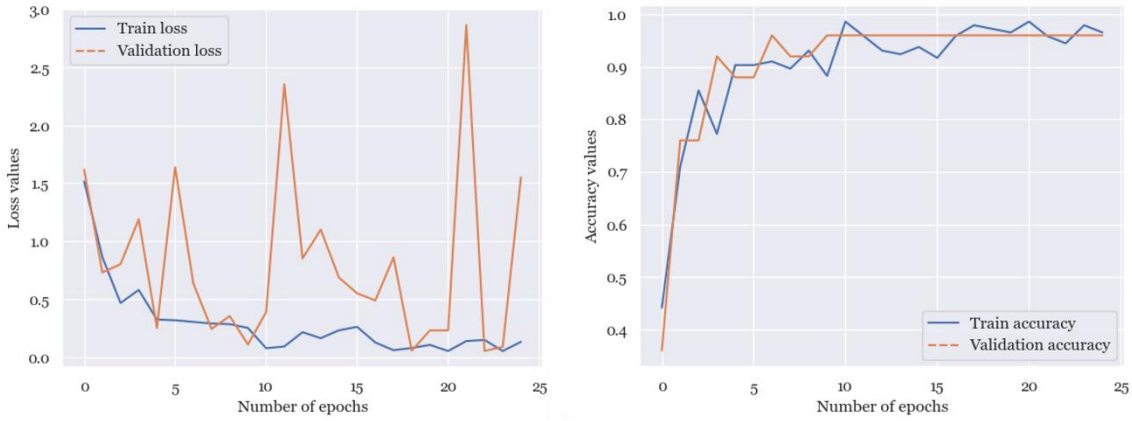
The EfficientNet_B4 model has attained a validation accuracy of 96%. Figure 58 displays the results for both training (blue) and validation (orange) in terms of loss (a) and accuracy (b) for the model trained over a span of 25 epochs. This experiment employs a learning rate of 0.001, incorporates variable weight decay, and follows a linear layer structure, as outlined in Table 9.



a) EfficientNet_B4 Loss a) EfficientNet_B4 Accuracy

Figure 57. The EfficientNet_B4 model's loss (a) and accuracy (b) results are shown, covering both the training data (blue) and validation data (orange). With 25 iterations, a learning rate of 0.001, variable weight decay, and a linear layer structure, the simulation was run.

The EfficientNet_B5 model has achieved a validation accuracy of 96%. Figure 59 presents the results for both training (blue) and validation (orange) regarding loss (a) and accuracy (b) during the model's 25 epoch training. This training strategy utilizes a learning rate of 0.0001, incorporates default weight decay, and employs a sequential layer structure, as outlined in Table 9.

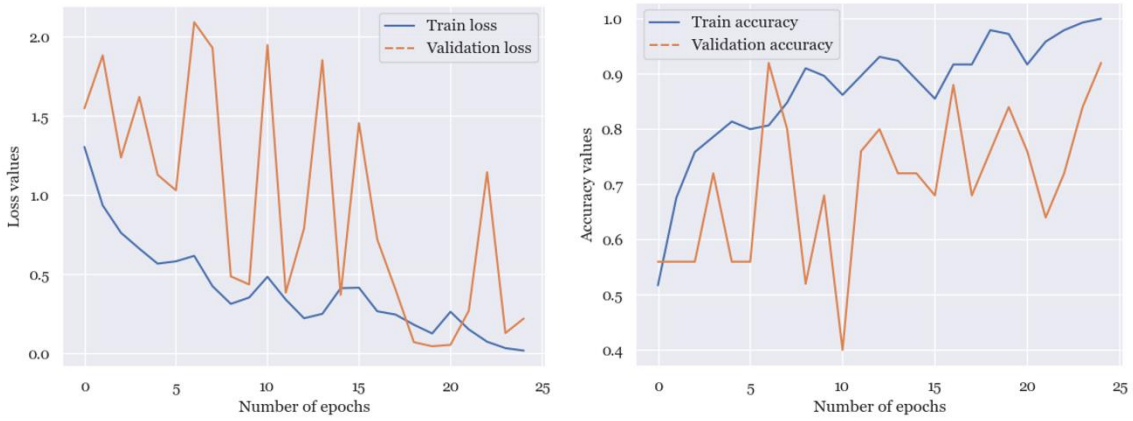


a) EfficientNet_B5 Loss a) EfficientNet_B5 Accuracy

Figure 58. The EfficientNet_B5 model's accuracy (b) and loss (a) results are shown, taking into account both the training data (blue) and validation data (orange). A sequential layer structure was used, a learning rate of 0.0001, default weight decay, and 25 iterations of the simulation were run.

The EfficientNet_B6 model has achieved a validation accuracy of 96%. Figure 60 visually displays the results for both training (blue) and validation (orange) in terms of loss (a) and accuracy (b) from the model's 25 epoch training process. This training regimen utilizes a learning

rate of 0.001, incorporates default weight decay, and follows a linear layer structure, as detailed in Table 9.



a) EfficientNet_B6 Loss

a) EfficientNet_B6 Accuracy

Figure 59. The accuracy (b) and loss (a) results for the EfficientNet_B6 model are presented, considering both the training data (blue) and validation data (orange). The simulation was conducted with 25 iterations, utilizing a learning rate of 0.001, incorporating default weight decay, and employing a linear layer structure.

The results decreased slightly due to the small size of the data set and the reduction in simulation time.

Even so, in this experiment, as in the previous one, the EfficientNet_B1, EfficientNet_B2 and ShuffleNet_V2_X0_5 models had the most consistent accuracy. The two EfficientNet models both obtained a maximum accuracy of 96 %, but ShuffleNet_V2_X0_5 obtained 84 %, which was the lowest accuracy. Thus, we continue to have three architectures with good performance over 25 epochs.

4.3. Experiment 3: Exploring Model Performance with an Expanded Image Dataset

For the new dataset, the new images were acquired from Flora-On (Flora-On | Flora de Portugal, 2022); INaturalist (Uma Comunidade Para Naturalistas · INaturalist, n.d.); Jardim Botânico da UTAD (UTAD, 2023); and Global Biodiversity Information Facility (GBIF) (GBIF, n.d.).

The new dataset contains 6730 images divided by five classes of weeds. The division of the pictures was random through RoboFlow. According to the literature, the images were split into 85% and 15 %, respectively (SEP 4 & Read, 2020). Thus, 5724 images were employed for training, and 1006 photos were applied for validation of the model. Figure 61 shows this division of the images, specifically, how many photos of each species are in the train and validation dataset.

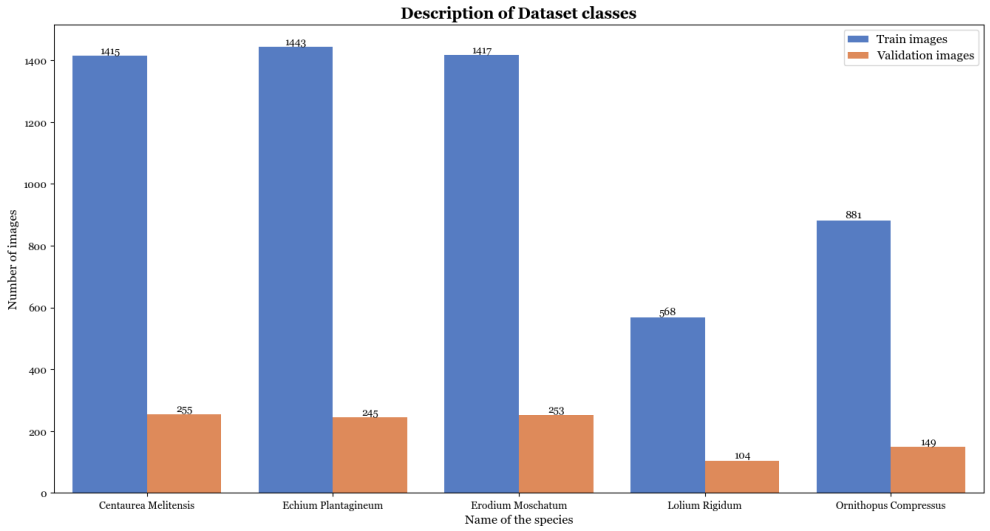


Figure 60. Description of the division of the new dataset for train and validation of the models.

In this part of the study, the twelve models were once again simulated with 200 epochs, as initial, with the combination of hyperparameters used for accomplishing the best result, as Table 10 shows, and with the new dataset. The objective of this test was to understand if the same model will be able to perform well with less or more quantity of images for training and simulation.

Table 10 shows the results that each model achieved with 200 epochs and with the new dataset. Figure 62 shows the model performance with an extended image dataset of all twelve models.

Table 10. Performance evaluation of the top twelve models: 200 epochs training with the new dataset.

Models	Learning rate	Weight decay	Number of layers	Best acc train [%]	Best acc validation [%]	Best loss train	Best loss validation	Number of parameters
MaxVit	0.001	0.0001	Linear	98.37	87.77	0.0562	0.0001	30410189
ShuffleNet_v2_x0_5	0.0001	Default	Linear	99,97	90,56	0,0023	0.0001	1371917
ShuffleNet_v2_x1_0	0.001	Default	Linear	99,81	88,87	0,0082	0.0001	2283729
ShuffleNet_v2_x1_5	0.001	0.0001	Linear	98,39	88,37	0,0517	0.0001	3508749
ShuffleNet_v2_x2_0	0.001	0.0001	Linear	98,95	87,18	0,0493	0.0001	7404241
EfficientNet_B0	0.001	Default	Linear	99.81	91.35	0.0069	0.0001	4013953
EfficientNet_B1	0.0001	Default	Linear	99.97	95.13	0.0011	0.0001	6519589
EfficientNet_B2	0.001	0.0001	Sequential	98.48	89.36	0.0499	0.0001	8587271
EfficientNet_B3	0.001	Default	Linear	99.88	92.05	0.0044	0.0001	10703917
EfficientNet_B4	0.001	0.0001	Linear	98.37	92.15	0.0504	0.0001	17557581
EfficientNet_B5	0.0001	Default	Sequential	99.90	94.83	0.0054	0.0001	17557581
EfficientNet_B6	0.001	Default	Linear	99.77	92.54	0.0065	0.0001	40747229

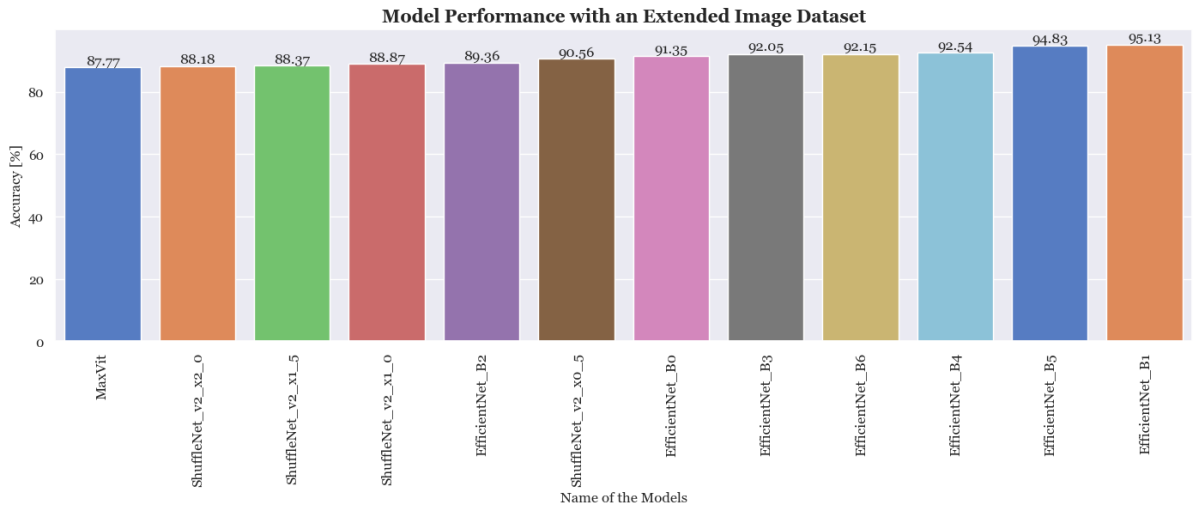
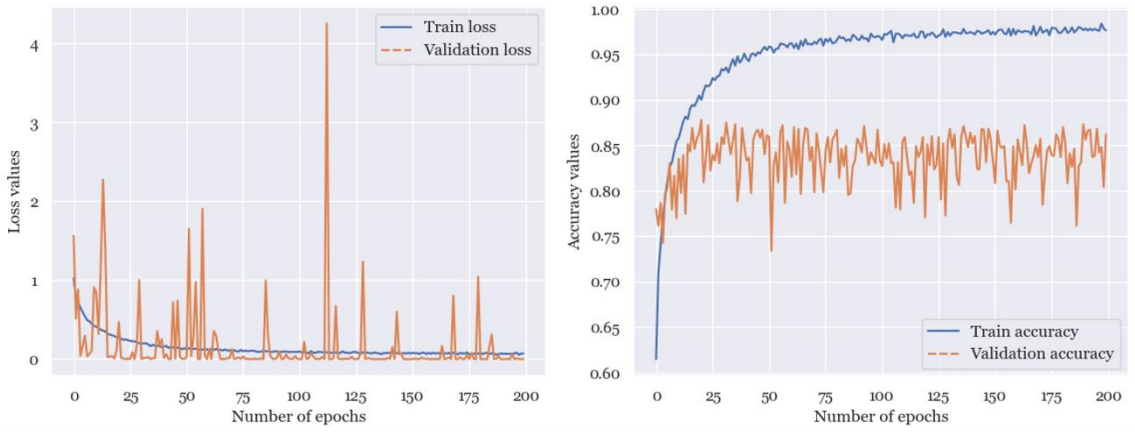


Figure 61. Model performance using the accuracy metric of all the best twelve models with the new expanded Dataset.

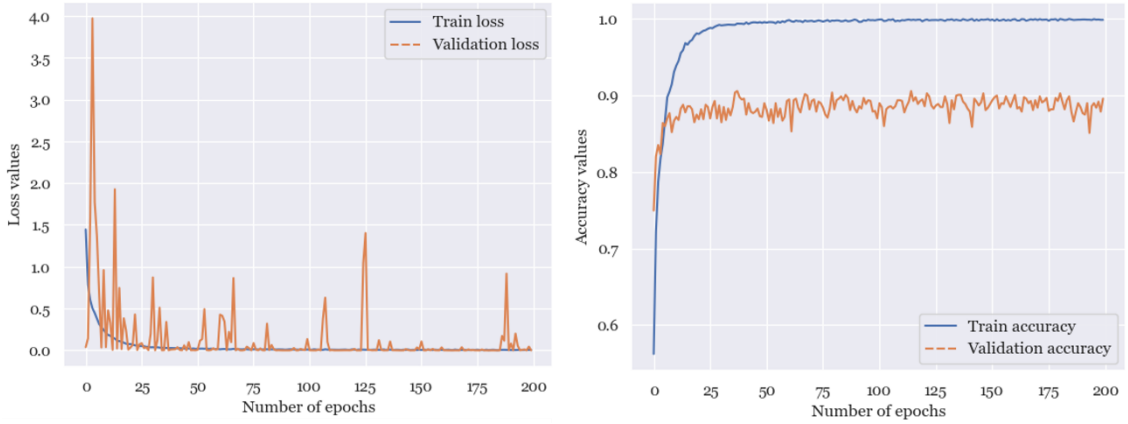
The MaxVit model achieved a maximum accuracy of 87.77% in validation. In Figure 63 is presented the results from loss (a) and accuracy (b) from MaxVit during training (blue) and validation (orange) with 200 epochs, and the new dataset. The hyperparameters used a learning rate of 0.001, variable weight decay, and a linear layer architecture. As shows Table 10.



a) MaxVit Loss b) MaxVit Accuracy

Figure 62. Loss (a) and accuracy (b) results from MaxVit model during training (blue) and validation (orange). The setup employed the new dataset, 200 epochs, learning rate of 0.001, variable weight decay and a liner architecture.

During validation, the ShuffleNet_v2_x0_5 achieved a maximum accuracy of 90.56%. As shown in Table 10, the ShuffleNet_v2_x0_5 model has 200 epochs, the new dataset, a learning rate of 0.0001, default weight decay, and a linear number of layers. Figure 64 displays the results for loss (a) and accuracy (b) during train (blue) and validation (orange).

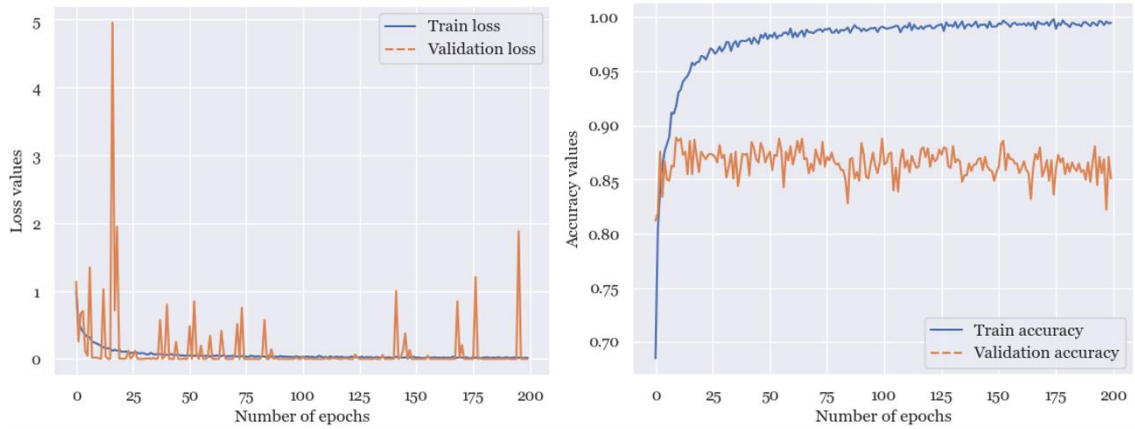


a) ShuffleNet_v2_x0_5 Loss

b) ShuffleNet_v2_x0_5 Accuracy

Figure 63. Results of the ShuffleNet_v2_x0_5 network using 200 epochs, the new dataset, a learning rate of 0.0001, a default weight decay, and a linear number of layers for loss (a) and accuracy (b) during train (blue) and validation (orange).

The ShuffleNet_v2_x1_0 model achieved a validation accuracy of 88.87%. Figure 65 illustrates the training and validation results for loss (a) and accuracy (b) in blue and orange, respectively, using the ShuffleNet_v2_x1_0 model with a new dataset. The training was conducted over 200 epochs with a learning rate of 0.001, default weight decay, and a linear layer configuration, as detailed in Table 10.

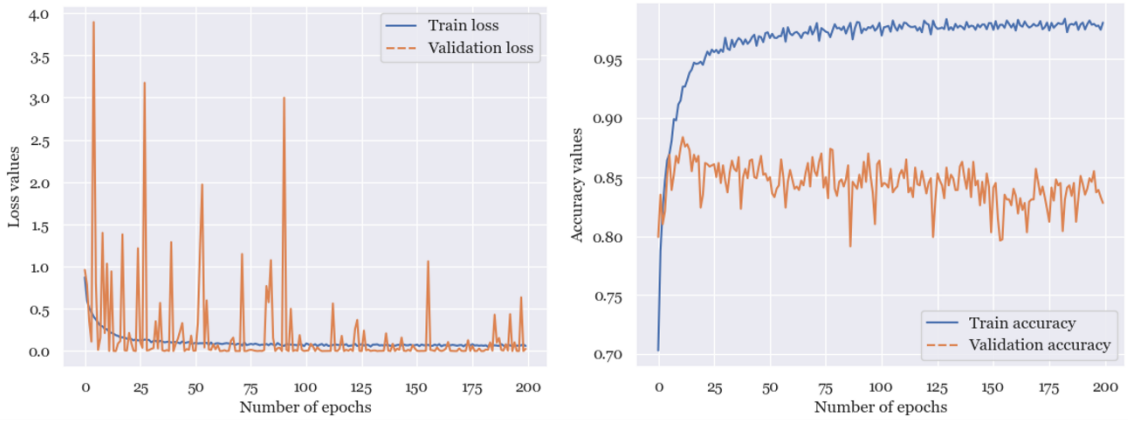


a) ShuffleNet_v2_x1_0 Loss

b) ShuffleNet_v2_x1_0 Accuracy

Figure 64. Loss (a) and accuracy (b) results for the ShuffleNet_v2_x1_0 model during training (in blue) and validation (in orange). The simulation was carried out with the new dataset, over 200 epochs with a learning rate of 0.001, default weight decay, and a linear layer configuration.

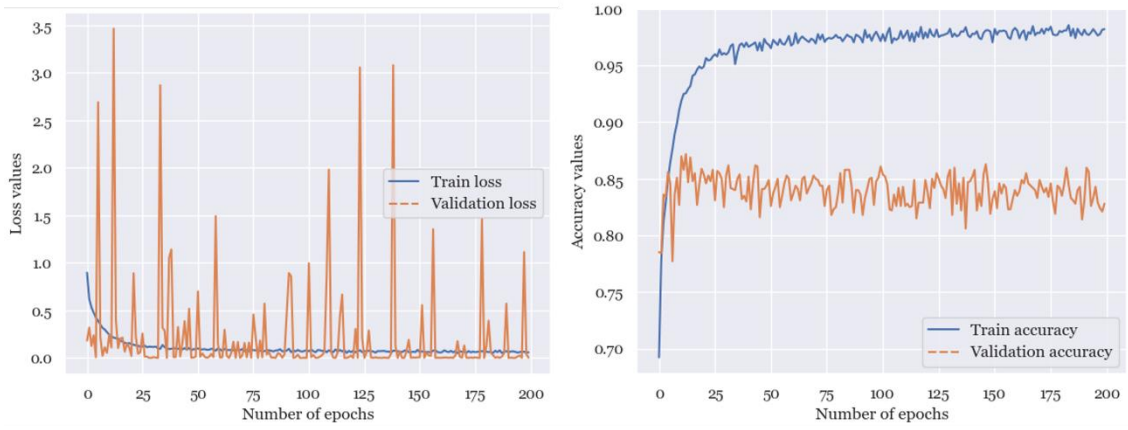
The ShuffleNet_v2_x1_5 model reached its highest validation accuracy at 88.37%. Figure 66 illustrates the results for loss (a) and accuracy (b) obtained using the new dataset during training (in blue) and validation (in orange). This was achieved over 200 epochs with a learning rate of 0.001, variable weight decay, as specified in Table 10, and a linear layer configuration.



a) ShuffleNet_v2_x1_5 Loss b) ShuffleNet_v2_x1_5 Accuracy

Figure 65. Results for the ShuffleNet_v2_x1_5 model's loss (a) and accuracy (b) during training (blue) and validation (orange). With the new dataset, over 200 iterations, a learning rate of 0.001, variable weight decay, and a linear layer configuration, the simulation was run.

The ShuffleNet_v2_x2_0 model achieved its highest validation accuracy at 87.18%. Figure 67 displays the results for loss (a) and accuracy (b) during training (in blue) and validation (in orange). This training process encompassed 200 epochs and utilized a learning rate of 0.001, variable weight decay, as specified in Table 10, a linear layer configuration, and the new dataset.

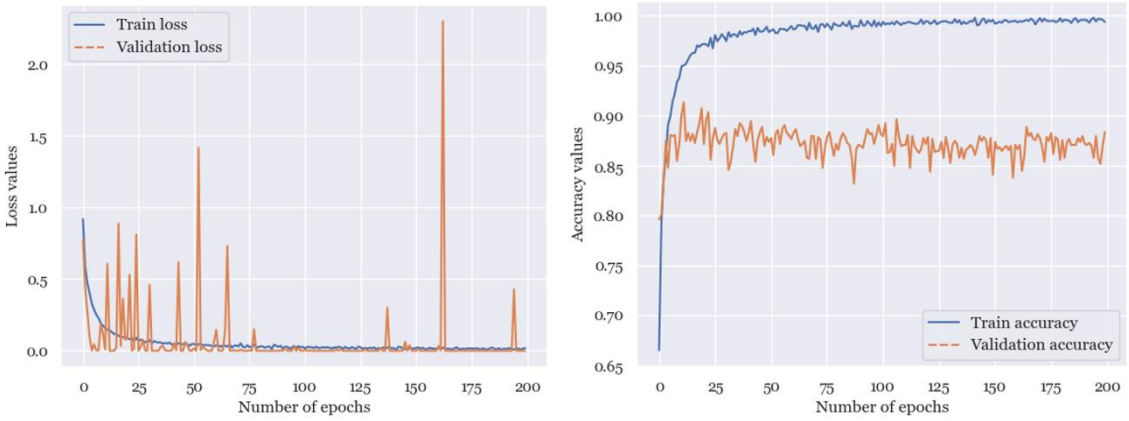


a) ShuffleNet_v2_x2_0 Loss b) ShuffleNet_v2_x2_0 Accuracy

Figure 66. Results of the ShuffleNet_v2_x2_0 model's train (blue) and validation (orange) phases for loss (a) and accuracy (b). A linear number of layers, 200 epochs, a learning rate of 0.001, a variable weight decay, and the new dataset were used to run the simulation.

The EfficientNet_B0 model attained its peak validation accuracy at 91.35%. In Figure 68, you can observe the results for loss (a) and accuracy (b) during both training (in blue) and validation

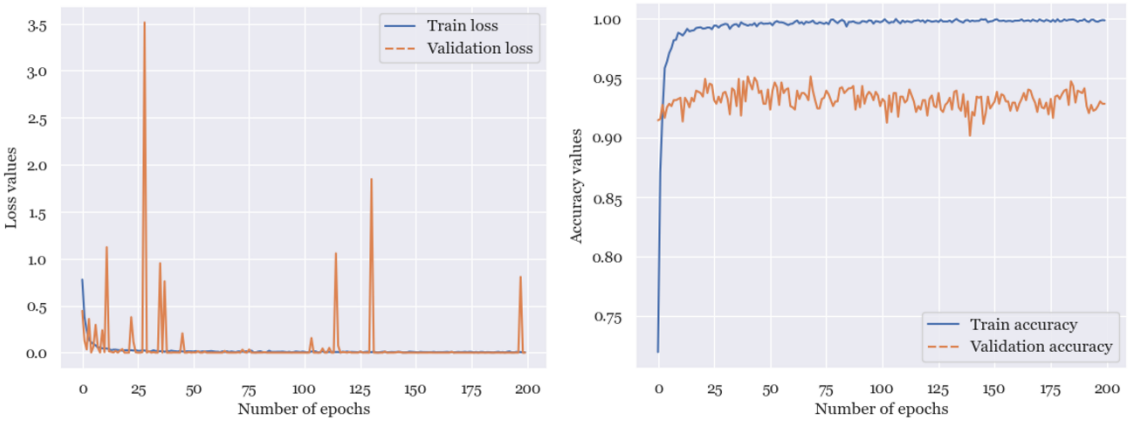
(in orange). The model underwent training for 200 epochs on a new dataset, utilizing a learning rate of 0.001, default weight decay, and a linear layer configuration, as outlined in Table 10.



a) EfficientNet_B0 Loss b) EfficientNet_B0 Loss Accuracy

Figure 67. The training and validation outcomes for the EfficientNet_B0 model, depicted in blue and orange respectively, for loss (a) and accuracy (b). This simulation was conducted with a linear layer configuration, 200 epochs, utilizing a learning rate of 0.001, incorporating variable weight decay, and employing the new dataset.

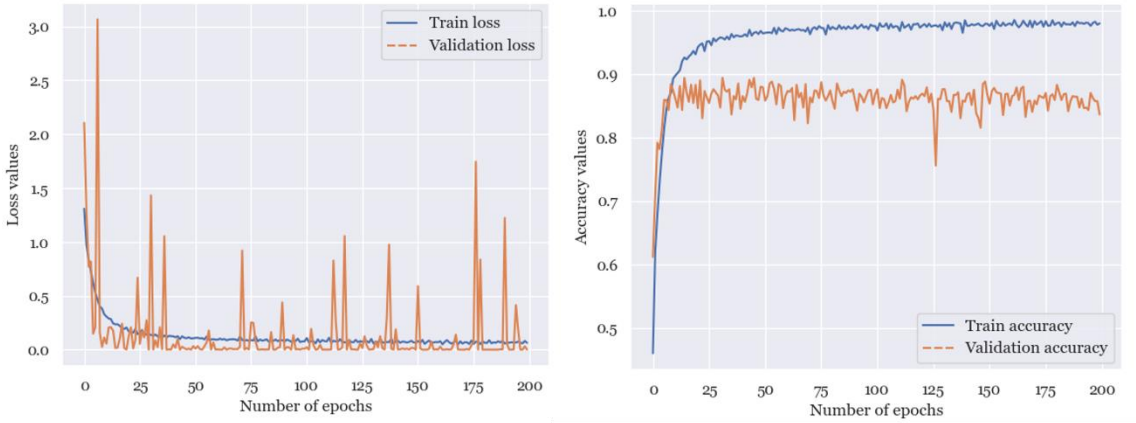
The EfficientNet_B1 model has reached the greatest validation accuracy of 95.13%. Figure 69 illustrates the loss (a) and accuracy (b) results for both training (blue) and validation (orange) in the model trained across 200 epochs, utilizing a learning rate of 0.0001, incorporating default weight decay, and employing a linear number of layers, in the new dataset, as showcased in Table 10.



a) EfficientNet_B1 Loss b) EfficientNet_B1 Loss Accuracy

Figure 68. The results of both training and validation for the EfficientNet_B1 model, represented as blue and orange lines for loss (a) and accuracy (b) respectively, were obtained. This experiment used the new dataset, a linear layer configuration, 200 epochs, with a learning rate of 0.0001, and incorporated default weight decay.

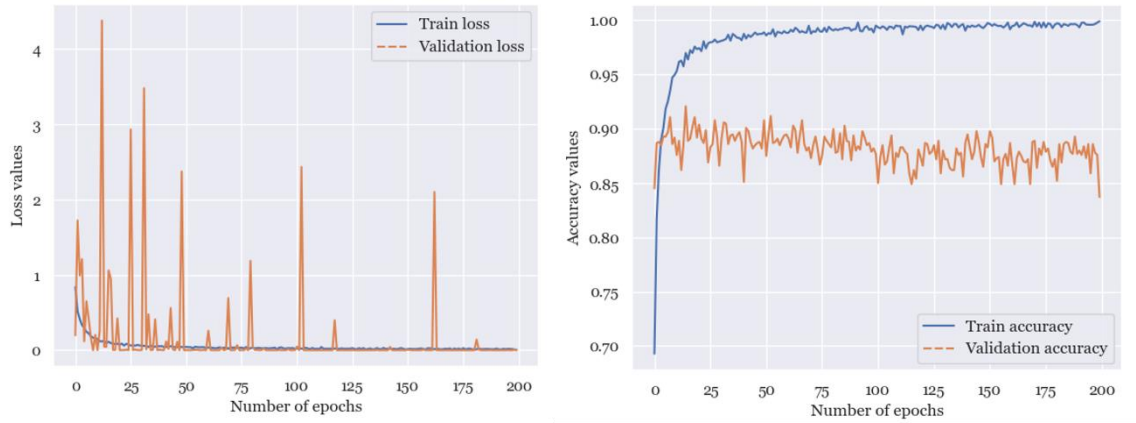
The EfficientNet_B2 model has attained a validation accuracy of 89.36%. Figure 70 illustrates the results for loss (a) and accuracy (b) during both training (in blue) and validation (in orange) across 200 epochs. The training utilized a learning rate of 0.001, variable weight decay, and employed a sequential layer configuration, as outlined in Table 10, along with the new dataset.



a) EfficientNet_B2 Loss b) EfficientNet_B2 Accuracy

Figure 69. The results of both the training and validation phases for the EfficientNet_B2 model, depicted as blue and orange lines representing loss (a) and accuracy (b) respectively, have been acquired. This experiment employed the new dataset, featured a sequential layer setup, 200 epochs, utilized a learning rate of 0.001, and included variable weight decay.

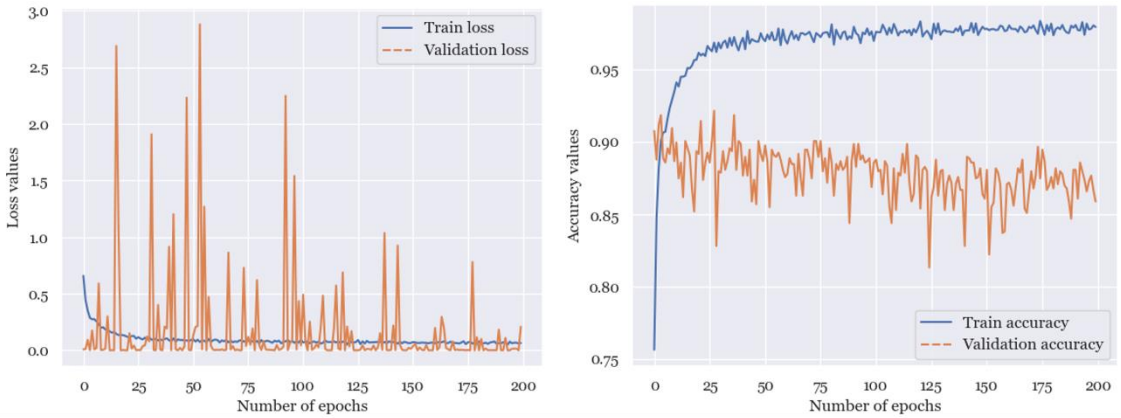
The validation accuracy achieved by the EfficientNet_B3 model stands at 92.05%. Figure 71 highlights the results for loss (a) and accuracy (b) during both training (blue) and validation (orange) across 200 epochs, in the new dataset. This training approach incorporates a learning rate of 0.001, default weight decay, and employs a linear layer configuration, as detailed in Table 10.



a) EfficientNet_B3 Loss b) EfficientNet_B3 Accuracy

Figure 70. The EfficientNet_B3 model's training and validation results are represented by blue and orange lines, which stand for loss (a) and accuracy (b), respectively. This experiment made use of the new dataset, had 200 epochs, linear layers, a learning rate of 0.001, and default weight decay.

The validation accuracy for the EfficientNet_B4 model is 92.15%. Figure 72 displays the results for the model trained over a period of 200 epochs in terms of loss (a) and accuracy (b) for both training (blue) and validation (orange). According to Table 10, this training program has a learning rate of 0.001, includes variable weight decay, employs a linear number of layers, and it was simulated with the new dataset.

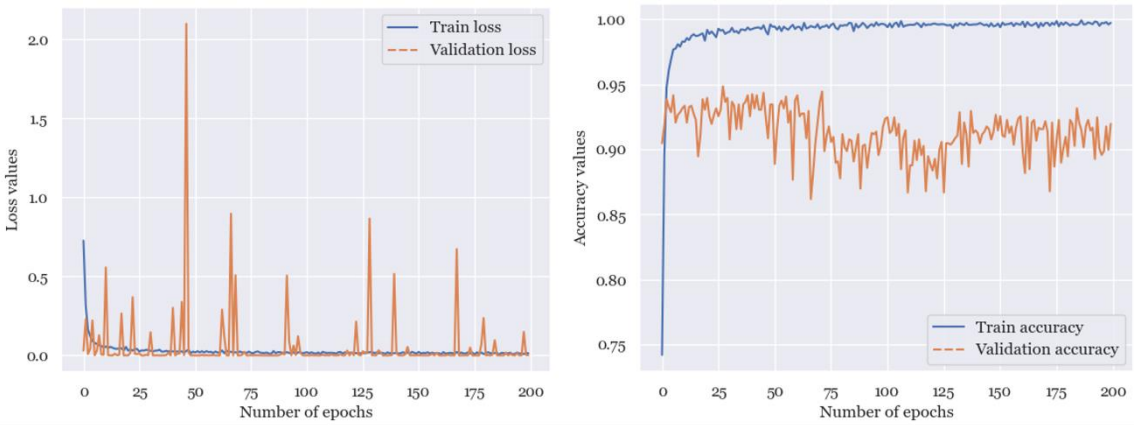


a) EfficientNet_B4 Loss

b) EfficientNet_B4 Accuracy

Figure 71. The training and validation results for the EfficientNet_B4 model are shown as blue and orange lines, which stand for loss (a) and accuracy (b), respectively. Utilizing the new dataset, 200 epochs, linear layers, a learning rate of 0.001, and variable weight decay were all used in this experiment.

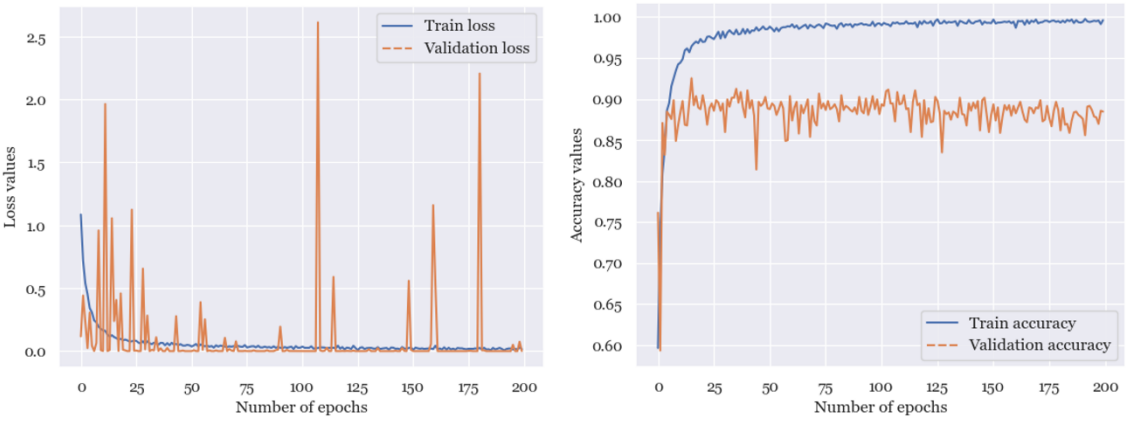
The EfficientNet_B5 model has achieved an impressive validation accuracy of 94.83%. Figure 73 showcases the results for both training (blue) and validation (orange) regarding loss (a) and accuracy (b) across 200 epochs. This training strategy incorporates a new dataset, a learning rate of 0.0001, incorporates default weight decay, and adopts a sequential layer configuration as outlined in Table 10.



a) EfficientNet_B5 Loss b) EfficientNet_B5 Accuracy

Figure 72. The training and validation results for the EfficientNet_B5 model are presented graphically, with loss (a) shown in blue and accuracy (b) in orange. These results were obtained using the new dataset and involved a 200 epoch training. The model configuration included sequential layers, a learning rate of 0.0001, and a default weight decay.

The EfficientNet_B6 model's validation accuracy has successfully achieved 92.54% of accuracy. Figure 74 shows the results in terms of loss (a) and accuracy (b) for both training (blue) and validation (orange). As shown in Table 10, these results came from training the model with the new dataset over 200 epochs using a learning rate of 0.001, default weight decay, and a linear number of layers.



a) EfficientNet_B6 Loss b) EfficientNet_B6 Accuracy

Figure 73. The training and validation results for the EfficientNet_B6 model are visually represented in the following way: loss (a) is represented by the blue curve, while accuracy (b) is depicted by the orange curve. These results were derived from a training process involving 200 epochs, using a new dataset. The model architecture consisted of linear layers, employing a learning rate of 0.001, and employing default weight decay.

Once again, the results have not been smoothed, so the values are the real results at the end of the simulations. For this reason, the graphs show several peaks.

Looking at the graphs, we can see once again that in the first 25 epochs, the models had already reached their maximum accuracy. From then on, the network learned some classes better and ended up unlearning others, which can be seen in the upward and downward spikes. All the models achieved a high final accuracy, but they all have several peaks, showing less consistency. This means that these models are unlearned and learned during training. This may have been due to the new dataset as the network in this experiment had more images to analyse with new orientations and colour sets.

4.4. Experiment 4: Testing Models in a Web Application-Real-world Performance Evaluation

For testing the models, a web application was developed in Gradio. The testing images were captured after all the training and all the tests were carried out. Thus, an average of five images per species were taken with a digital camera for further testing.

During this phase of the work, the images of the five plant species were sequentially uploaded to the web application's interface and their classifications were annotated. These annotations were used to determine the accuracy models.

In the testing phase, the MaxVit and the versions of ShuffleNet and EfficientNet which were previously employed in the preceding chapters, were used. For each of these models, the conducted tests worked with the same dataset.

Therefore, the web application conducted tests on a total of 26 images across the twelve different models.

The web application allows users to upload an image of a species, request its classification, and in the end, be able to see the name of the species.

The web application allows users to upload an image of a plant, request its classification, and ultimately access the species' name. Figure 75 showcases the user interface of the web application, providing the capability to upload images and receive their respective classifications. It describes the interface and objectives to the user, as well as some examples that operators can test.

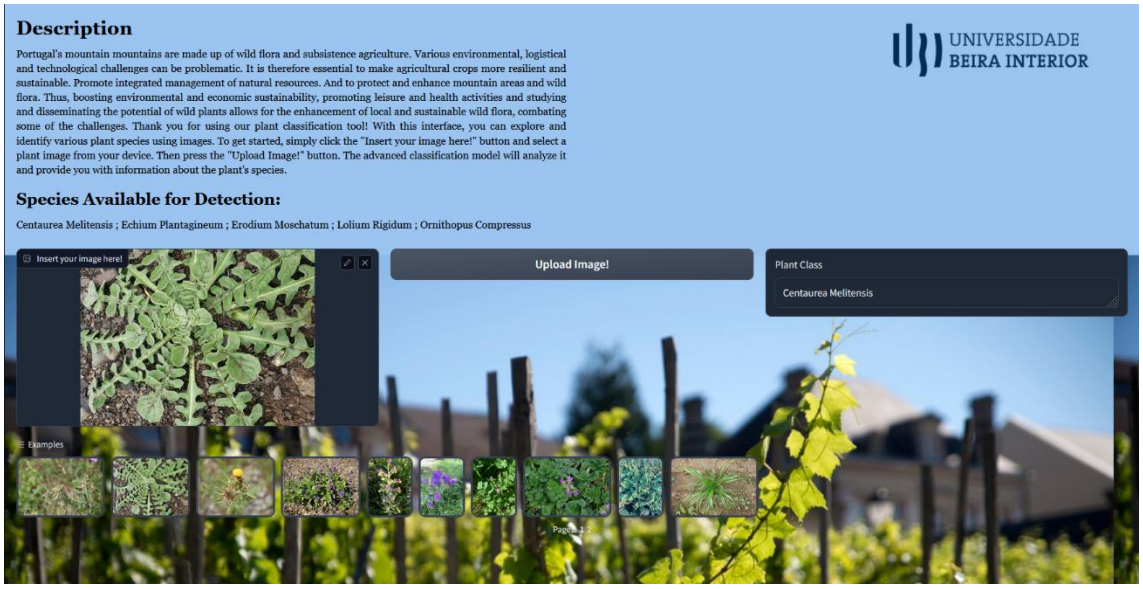


Figure 74. Web interface for image classification.

The accuracy was calculated by dividing the number of correct predictions by the total number of predictions made by the web application. This is demonstrated in Equation (32).

Thus, after conducting all the tests, the accuracy was calculated. Table 11 shows the results in percentage as well as the number of images that were correct and incorrect classified.

$$acc = \frac{Number\ of\ correct\ classifications}{Total\ number\ of\ images} * 100 \tag{32}$$

Table 11. Accuracy results following the testing phase, including the count of correctly classified images and the count of incorrectly classified images.

Models	Correct Images	Incorrect Images	Accuracy [%]
MaxVit	24	2	92.30
ShuffleNet_v2_x0_5	20	6	76.92
ShuffleNet_v2_x1_0	22	4	84.61
ShuffleNet_v2_x1_5	24	2	92.30
ShuffleNet_v2_x2_0	20	6	76.92
EfficientNet_B0	23	6	88.46
EfficientNet_B1	25	1	96.15
EfficientNet_B2	20	6	76.92
EfficientNet_B3	24	2	92.30
EfficientNet_B4	23	3	88.46
EfficientNet_B5	25	1	96.15
EfficientNet_B6	21	5	80.76

To show the effectiveness of the models, a confusion matrix was developed for each model.

The names in the row correspond to the real results, ground truth, on the other side, the column is correlated with the predictions that the network have done. In the tables when the value is written in the cell that has the name of the species equal the column and row means that value is the correct prediction that the model has made for that species. However, when the number that is in the cell and the names does not correspond means that the model should have predicted the name that is in the row, but projected another name that corresponds to the name in the column represented by the predicted values.

Table 12 provides the confusion matrix of the MaxVit model. In this table is possible to observe that the model correctly classified the species *Ornithopus Compressus, Lolium Rigidum,* and *Echium Plantagineum*. In the five simulated images for each of the species, this model was able to correctly identify every single one of them.

However, for *Centaurea Melitensis* and *Erodium Moschatum,* the model only classified well 5 and 4, respectively. For the *Centaurea Melitensis* an incorrect prediction assumed that one image was an *Erodium Moschatum*. For the *Erodium Moschatum* the incorrect prediction assumed that one image was an *Lolium Rigidum*.

Table 12.Confusion matrix of the MaxVit model.

MaxVit		**Grount truth values**				
		Centaurea Melitensis	*Echium Plantagineum*	*Erodium Moschatum*	*Lolium Rigidum*	*Ornithopus Compressus*
Predicted Values	*Centaurea Melitensis*	5	0	0	0	0
	Echium Plantagineum	0	5	0	0	0
	Erodium Moschatum	1	0	4	0	0
	Lolium Rigidum	0	0	1	5	0
	Ornithopus Compressus	0	0	0	0	5

Table 13 represents the confusion matrix of the ShuffleNet_v2_x0_5 model. The model correctly classified the species *Echium Plantagineum and Ornithopus Compressus*. This model was able to correctly identify every single one of them.

However, for *Centaurea Melitensis* the incorrect prediction assumed that one image was an *Ornithopus Compressus a*nd two were *Erodium Moschatum*. In *Erodium Moschatum,* the model only classified well three images, and the others were incorrect assuming that one was *Lolium Rigidum* and another one was *Ornithopus Compressus*. The specie *Lolium Rigidum,* got it right four images, and one was predicted as *Centaurea Melitensis*.

Table 13. Confusion matrix of the ShuffleNet_v2_x0_5 model.

ShuffleNet_v2_x0_5		**Grount truth values**				
		Centaurea Melitensis	*Echium Plantagineum*	*Erodium Moschatum*	*Lolium Rigidum*	*Ornithopus Compressus*
Predicted Values	*Centaurea Melitensis*	3	0	0	1	0
	Echium Plantagineum	0	5	0	0	0
	Erodium Moschatum	2	0	3	0	0
	Lolium Rigidum	0	0	1	4	0
	Ornithopus Compressus	1	0	1	0	5

Table 14 represents the confusion matrix of the ShuffleNet_v2_x1_0 model. The model correctly classified all the images of the species *Centaurea Melitensis, Echium Plantagineum* and *Lolium Rigidum*.

Nonetheless, for *Erodium Moschatum* this model predicted incorrectly that one image was a *Lolium Rigidum* and two were *Ornithopus Compressus*. In the last specie, *Ornithopus Compressus* only one image was classified incorrectly with the name of *Centaurea Melitensis*.

Table 14. Confusion matrix of the ShuffleNet_v2_x1_0 model.

ShuffleNet_v2_x1_0		**Grount truth values**				
		Centaurea Melitensis	*Echium Plantagineum*	*Erodium Moschatum*	*Lolium Rigidum*	*Ornithopus Compressus*
Predicted Values	*Centaurea Melitensis*	6	0	0	0	1
	Echium Plantagineum	0	5	0	0	0
	Erodium Moschatum	0	0	3	0	0
	Lolium Rigidum	0	0	1	5	0
	Ornithopus Compressus	0	0	2	0	4

Table 15 represents the confusion matrix of the ShuffleNet_v2_x1_5 model. The model correctly classified all the images of the species *Erodium Moschatum, Lolium Rigidum* and *Ornithopus Compressus* correctly classified all the images.

Even so, for *Centaurea Melitensis* this model inaccurately previews one image as *Ornithopus Compressus*. For *Echium Plantagineum* one image was categorized wrongly with the name of *Erodium Moschatum*.

Table 15. Confusion matrix of the ShuffleNet_v2_x1_5 model.

ShuffleNet_v2_x1_5		**Grount truth values**				
		Centaurea Melitensis	*Echium Plantagineum*	*Erodium Moschatum*	*Lolium Rigidum*	*Ornithopus Compressus*
Predicted Values	*Centaurea Melitensis*	5	0	0	0	0
	Echium Plantagineum	0	4	0	0	0
	Erodium Moschatum	0	1	5	0	0
	Lolium Rigidum	0	0	0	5	0
	Ornithopus Compressus	1	0	0	0	5

Table 16 represents the confusion matrix of the ShuffleNet_v2_x2_0 model. The model precisely categorized the species *Echium Plantagineum and Ornithopus Compressus.*

Conversely, for *Centaurea Melitensis* the mistaken expectation presumed that one image was an *Erodium Moschatum.* In the *Erodium Moschatum* classification, the system wrongly preview that two images were *Ornithopus Compressus* and one was *Echium Plantagineum*. In the *Lolium Rigidum* classified wrongly one *Centaurea Melitensis* and one *Ornithopus Compressus.*

Table 16. Confusion matrix of the ShuffleNet_v2_x2_0 model.

ShuffleNet_v2_x2_0		**Grount truth values**				
		Centaurea Melitensis	*Echium Plantagineum*	*Erodium Moschatum*	*Lolium Rigidum*	*Ornithopus Compressus*
Predicted Values	*Centaurea Melitensis*	5	0	0	1	0
	Echium Plantagineum	0	5	1	0	0
	Erodium Moschatum	1	0	2	0	0
	Lolium Rigidum	0	0	0	3	0
	Ornithopus Compressus	0	0	2	1	5

Table 17 represents the confusion matrix of the EfficientNet_B0 model. The model accurately classified the species *Echium Plantagineum, Lolium Rigidum* and *Ornithopus Compressus.*

In opposition, for *Centaurea Melitensis* the inaccurate prediction recognized one image as an *Erodium Moschatum,* and ontoher one as *Ornithopus Compressus.* In the *Erodium Moschatum* arrangement, the procedure incorrectly preview that one image were *Ornithopus Compressus.*

Table 17. Confusion matrix of the EfficientNet_B0 model.

EfficientNet_B0		**Grount truth values**				
		Centaurea Melitensis	*Echium Plantagineum*	*Erodium Moschatum*	*Lolium Rigidum*	*Ornithopus Compressus*
Predicted Values	*Centaurea Melitensis*	4	0	0	0	0
	Echium Plantagineum	0	5	0	0	0
	Erodium Moschatum	1	0	4	0	0
	Lolium Rigidum	0	0	0	5	0
	Ornithopus Compressus	1	0	1	0	5

Table 18 represents the confusion matrix of the EfficientNet_B1 model. The model accurately classified the species *Echium Plantagineum, Erodium Moschatum, Lolium Rigidum* and *Ornithopus Compressus*.

In opposition, for *Centaurea Melitensis* the model incorrectly predicted that one image was an *Erodium Moschatum*.

Table 18. Confusion matrix of the EfficientNet_B1 model.

EfficientNet_B1		**Grount truth values**				
		Centaurea Melitensis	*Echium Plantagineum*	*Erodium Moschatum*	*Lolium Rigidum*	*Ornithopus Compressus*
Predicted Values	*Centaurea Melitensis*	5	0	0	0	0
	Echium Plantagineum	0	5	0	0	0
	Erodium Moschatum	1	0	5	0	0
	Lolium Rigidum	0	0	0	5	0
	Ornithopus Compressus	0	0	0	0	5

Table 19 represents the confusion matrix of the EfficientNet_B2 model. The model accurately classified the species *Echium Plantagineum,* and *Ornithopus Compressus*.

In opposition, for *Centaurea Melitensis* the model incorrectly predicted that one image was an *Erodium Moschatum*. The specie *Erodium Moschatum wrongly* predicted that one image belongs to the *Echium Plantagineum,* and two to the *Ornithopus Compressus*. Finally, the *Lolium Rigidum* simulation incorrectly preview that two images were an *Erodium Moschatum,* and an *Ornithopus Compressus*.

Table 19. Confusion matrix of the EfficientNet_B2 model.

EfficientNet_B2		**Grount truth values**				
		Centaurea Melitensis	*Echium Plantagineum*	*Erodium Moschatum*	*Lolium Rigidum*	*Ornithopus Compressus*
Predicted Values	*Centaurea Melitensis*	4	0	0	0	0
	Echium Plantagineum	0	5	1	0	0
	Erodium Moschatum	1	0	2	1	0
	Lolium Rigidum	0	0	0	3	0
	Ornithopus Compressus	0	0	2	1	5

Table 20 represents the confusion matrix of the EfficientNet_B3 model. The model perfectly classified the species *Erodium Moschatum, Lolium Rigidum* and *Ornithopus Compressus*.

However, for *Centaurea Melitensis* the model incorrectly predicted that one image was an *Erodium Moschatum*. And, the specie *Echium Plantagineum incorrectly* predicted that one image belongs to the *Erodium Moschatum*.

Table 20. Confusion matrix of the EfficientNet_B3 model.

EfficientNet_B3		**Grount truth values**				
		Centaurea Melitensis	*Echium Plantagineum*	*Erodium Moschatum*	*Lolium Rigidum*	*Ornithopus Compressus*
Predicted Values	*Centaurea Melitensis*	5	0	0	0	0
	Echium Plantagineum	0	4	0	0	0
	Erodium Moschatum	1	1	5	0	0
	Lolium Rigidum	0	0	0	5	0
	Ornithopus Compressus	0	0	0	0	5

Table 21 corresponds to the confusion matrix of the EfficientNet_B4 model. The model correctly categorized the species *Echium Plantagineum, Lolium Rigidum* and *Ornithopus Compressus*.

Nonetheless, for *Centaurea Melitensis* the simulation inaccurately anticipated that two images were an *Erodium Moschatum* and *Echium Plantagineum,* respectively. The specie *Erodium Moschatum wrongly* predicted that one image belongs to the *Echium Plantagineum*.

Table 21. Confusion matrix of the EfficientNet_B4 model.

EfficientNet_B4		**Grount truth values**				
		Centaurea Melitensis	*Echium Plantagineum*	*Erodium Moschatum*	*Lolium Rigidum*	*Ornithopus Compressus*
Predicted Values	*Centaurea Melitensis*	4	0	0	0	0
	Echium Plantagineum	1	5	1	0	0
	Erodium Moschatum	1	0	4	0	0
	Lolium Rigidum	0	0	0	5	0
	Ornithopus Compressus	0	0	0	0	5

Table 22 corresponds to the confusion matrix of the EfficientNet_B5 model. The model correctly categorized the species *Echium Plantagineum, Erodium Moschatum, Lolium Rigidum* and *Ornithopus Compressus*.

However, for *Centaurea Melitensis* the simulation inaccurately anticipated one image as being an *Erodium Moschatum*.

Table 22. Confusion matrix of the EfficientNet_B5 model.

EfficientNet_B5		**Grount truth values**				
		Centaurea Melitensis	*Echium Plantagineum*	*Erodium Moschatum*	*Lolium Rigidum*	*Ornithopus Compressus*
Predicted Values	*Centaurea Melitensis*	5	0	0	0	0
	Echium Plantagineum	0	5	0	0	0
	Erodium Moschatum	1	0	5	0	0
	Lolium Rigidum	0	0	0	5	0
	Ornithopus Compressus	0	0	0	0	5

Table 22 corresponds to the confusion matrix of the EfficientNet_B6 model. The model correctly categorized all the images only in the specie *Ornithopus Compressus*.

However, for *Centaurea Melitensis* the simulation inaccurately anticipated one image as being an *Erodium Moschatum,* and onother as *Ornithopus Compressus*. Furthermore, the *Erodium Moschatum* wrongly classified one image as being an *Erodium Moschatum*. *The Erodium Moschatum* has classified wrongly one image as an *Ornithopus Compressus*. And the last incorrectly classification was in the *Lolium Rigidum,* the model predicted one image as an *Echium Plantagineum*.

Table 23. Confusion matrix of the EfficientNet_B6 model.

EfficientNet_B5		Grount truth values				
		Centaurea Melitensis	*Echium Plantagineum*	*Erodium Moschatum*	*Lolium Rigidum*	*Ornithopus Compressus*
Predicted Values	*Centaurea Melitensis*	4	0	0	0	0
	Echium Plantagineum	0	4	0	1	0
	Erodium Moschatum	1	1	4	0	0
	Lolium Rigidum	0	0	0	4	0
	Ornithopus Compressus	1	0	1	0	5

The values that were incorrectly predicted are related to the fact that some of the images are very similar, in terms of the colour of the plants and the soil, as well as the similarity between leaves in some of the species, for example between *Centaurea Melitensis, Erodium Moschatum* and *Ornithopus Compressus,* as can be seen in Figure 27 shown in Chapter 3.

According to the results of this test, specially, the confusion matrix and the accuracy that each model achieved, it is possible to conclude that the best models for image classification, for the datasets used, were the EfficientNet_B1 and EfficientNet_B5, because these two architectures achieved a good and high accuracy in all the experiments realized, including in this test experiment.

4.5. Discussion of results

According to the results of the realized tests, the following conclusions and challenges can be aimed.

The first experiment successfully identified both the optimal combination of hyperparameters and the top-performing simulated models. According to the results, a learning rate of 0.01 was observed in 6 instances, while 0.001 was employed in 21 simulations, and 0.0001 was utilized in 37 occurrences.

Furthermore, a default weight decay setting was employed in 48 instances, while a variable weight decay was selected in 16 occurrences.

Additionally, the type of fully connected layers was observed in a linear configuration during 28 tests, whereas the sequential arrangement was chosen in 36 instances.

Nonetheless, it's crucial to emphasize that the selection of hyperparameters like learning rate, weight decay, and layer configuration can vary based on the specific problem, dataset, and model architecture.

In this case, it’s possible to conclude, according to the architectures and the dataset used, a smaller value of learning rate is better for achieving a higher accuracy. A reduced learning rate can contribute to a more stable convergence of the optimization algorithm. It enables the model

to make smaller adjustments to its parameters during each iteration, mitigating the risk of overshooting the optimal solution (Brownlee, 2019).

Also, a default weight decay with the value of zero performed better in these conditions. This happens due to the fact that a default weight decay serves as a regularization method, mitigating overfitting by introducing a penalty term into the loss function. This regularization encourages the model's weights to remain modest, thereby potentially decreasing the model's complexity and enhancing its capacity to generalize effectively to previously unseen data (Vasani, 2019).

In addition, the configuration of the fully connected layers that performed better was sequential. Fully connected layers are effective at learning complex patterns and relationships in data. They can combine information from all input neurons, allowing the model to capture high-level representations and abstract features from the data (Unzueta, 2022).

Furthermore, according to the results, the models that achieved a superior performance were MaxVit, ShuffleNet, and EfficientNet. After analysis of the charts, MaxVit and ShuffleNet_v2_x2_0 achieved a good accuracy of 96%, while the others accomplished an accuracy of 99.3%.

MaxVit, ShuffleNet, and EfficientNet distinguish themselves with their innovative and resource-efficient architectural designs. These models adeptly leverage available resources, including parameters and computational power, to achieve an optimal equilibrium between model depth and complexity. Their architecture which may encompass pioneering building blocks, strategic utilization of skip connections, or the adoption of depth-wise convolutions, substantially reinforces their capability to acquire intricate data representations.

Additionally, models attaining high accuracy owe a substantial portion of their success to the meticulous fine-tuning of hyperparameters during the training process (Ma *et al.,* 2018), (Tu *et al.,* 2022), (Tan & Le, 2020).

The quality of the data, including meticulous labelling and preprocessing, exerts a profound influence on the overall performance of these models.

MaxVit likely adopts an innovative and resource-efficient architecture that maximizes the utilization of available resources, such as parameters and computation, all while maintaining an optimal balance of depth and complexity. This architectural approach incorporates elements such as novel building blocks, strategic utilization of skip connections, or the integration of depth-wise convolutions, all of which augment its ability to acquire intricate data representations (Tu *et al.,* 2022).

ShuffleNet distinguishes itself with a unique channel shuffling operation, facilitating the exchange of information among distinct groups of convolutional channels. This operation elevates feature learning and fosters cross-channel interactions, ultimately contributing to enhanced accuracy (Ma *et al.,* 2018).

EfficientNet employs the concept of compound scaling, an approach that concurrently optimizes the model's depth, width, and resolution. This harmonious scaling strategy strikes a well-calibrated balance among these dimensions, ensuring effective resource utilization and a well-rounded model expansion, ultimately resulting in improved accuracy (Tan & Le, 2020).

The dataset consisting of only 172 images is notably small when it comes to training DL models. DL models, particularly those of substantial complexity demand a substantial volume of data to generalize effectively. In the context of a limited dataset, there is a heightened risk of overfitting, where models essentially memorize the training data instead of learning broader underlying patterns (Alibabaei *et al.,* 2022).

For smaller-sized datasets, very deep and intricate architectural designs might not be the most appropriate choice. More intricate models usually necessitate more extensive data to yield effective training outcomes (Alibabaei *et al.,* 2022).

The performance of these models is markedly sensitive to hyperparameters such as learning rate, batch size, and weight initialization. It is plausible that the hyperparameters selected for these models were not suitably fine-tuned for the characteristics of the dataset (Alibabaei *et al.,* 2022).

Moreover, if a substantial class imbalance exists within the dataset, with certain classes being represented by very few examples, it can negatively impact model performance. Classes with limited data may face challenges in achieving satisfactory results (Alibabaei *et al.,* 2022).

In the second experiment, the results slightly decreased due to the fact that the dataset is small, and the time of simulation was reduced as well. The smallest accuracy was 84% in the ShuffleNet_v2_x0_5 model, followed by EfficientNet_B2 and EfficientNet_B6 with an accuracy of 92%. The remainder models achieved a maximum accuracy of 96%.

Complex deep learning models, particularly the large and complex ones such as MaxVit, ShuffleNet, and EfficientNet, frequently demand extended training periods to unleash their complete capabilities. In the case of a mere 25 epochs, these models might not have had ample time to converge towards an optimal solution. It is advisable to consider prolonging the training duration by increasing the number of epochs, thus affording them a more generous timeframe to converge and extract valuable insights from the dataset (Unzueta, 2022).

The third experiment achieved a minimum accuracy of 87.18% in ShuffleNet_V2_X2_0 and a maximum of 95.13% in EfficientNet_B1. It is possible to conclude that with the biggest dataset these 3 models achieved very good accuracy as well. However, it reached values of accuracy slightly smaller than with the minor dataset.

The larger dataset comprising 6,000 images may exhibit distinct characteristics and variations when compared to the smaller dataset of 172 images. These disparities can introduce different complexities for the models, potentially posing a slightly greater challenge in achieving the same level of accuracy (Kapoor *et al.,* 2021).

In the case of the smaller dataset, a heightened risk of overfitting was present, wherein the models could have effectively memorized the training data. This scenario might have yielded seemingly higher accuracy on the smaller dataset. In contrast, the larger dataset offers a more extensive array of examples, introducing greater diversity and making it more demanding for the models to overfit (Kapoor *et al.,* 2021).

The hyperparameters that proved effective for the smaller dataset may not be optimally suited for the larger dataset. Models often necessitate adjustments to their hyperparameters when

transitioning to datasets of varying sizes. Thoughtful hyperparameter tuning tailored to the new dataset has the potential to enhance accuracy (Kapoor *et al.,* 2021).

Furthermore, the distribution of classes or patterns within the new dataset could differ from that of the smaller dataset. If the new dataset encompasses more intricate or diverse instances, this divergence may elucidate the marginally lower accuracy (Kapoor *et al.,* 2021).

The presence of data quality issues within the new dataset can indeed lead to diminished accuracy.

Nevertheless, certain models succeed when trained on extensive distributed computing systems equipped with abundant data and substantial computational resources. Bigger diversity in the training data for a neural network can improve its resilience and adaptability to generalize. Exposing the network to a wide array of data enables it to learn and adjust to various patterns, variations, and situations. This diversity not only helps in reducing overfitting but also prepares the model to deal with the substantial variability often encountered in real-world applications (Taye, 2023).

Finally, in the fourth experiment, the models trained with the new dataset were tested in real time, achieving very good results. The ShuffleNet_V2_X0_5, ShuffleNet_V2_X2_0 and EfficienttNet_B2 achieved the lowest accuracy, with 76.92%, misclassifying 6 images out of 26. However, the maximum accuracy was obtained for EfficienttNet_B1 and EfficienttNet_B5, misclassifying 1 image out of 26, thus having an accuracy of 96.15%.

In summary, the study conducted a comprehensive evaluation of various PyTorch models for a classification task and found EfficientNet_B1 and EfficientNet_B5 to be the top performers. Both models consistently achieved high accuracy rates exceeding 94% across all experiments and outperformed the other twelve models, especially in real-time testing on an online application.

EfficientNet_B5, being deeper and more complex with a larger parameter count, demonstrated superior capability in capturing intricate patterns within the dataset. This depth and complexity proved advantageous, especially when dealing with complex and diverse datasets. On the other hand, EfficientNet_B1, while less complex, showcased resource efficiency in terms of both memory usage and computation power when compared to EfficientNet_B5 (Tan & Le, 2020).

The dataset's specific characteristics, such as its size, diversity, and complexity, play a pivotal role in determining which model performs optimally. EfficientNet_B1 and EfficientNet_B5 exhibited adaptability to the dataset by effectively capturing and generalizing complex patterns and information.

Researchers and practitioners should consider these findings when selecting models for similar classification tasks, taking into account the trade-off between model complexity and computational resources based on the specific dataset at hand.

5. Conclusions

5.1. General Conclusions

In the agricultural sector, rapid advancements in AI, particularly CNNs, have emerged as a predominant solution against unwanted plants. Within the realm of PA, a valuable strategy involves utilizing remote sensing technology to map not needed plant patches in agricultural areas. ML algorithms offer excellent capabilities of image classification to distinguish cultivated or unnecessary plants.

Plants with an infestation character not only compete with cultivated crops for essential resources but also pose a significant economic and environmental threat. In the context of comprehensive vineyard management, vegetation control takes on particular significance. Plants near the ground within vineyards wield a substantial influence over the prevalence of various pests, including nematodes, animals, mites, insects, and diseases. Furthermore, is imperative that plant control methods prioritize environmental sustainability.

ML algorithms, equipped with the ability to analyse vast datasets and make intelligent decisions, have ushered in a new era of plant classification. By harnessing the power of computer vision, they are revolutionizing the way plants are detected, identified, and managed in agriculture.

The present work aims to help to improve the concept of a more sustainable agriculture sector as well as the image classification field.

The main goals of this dissertation involve the creation of CNN models for flora classification using the framework PyTorch. This research aims to investigate the ideal architecture, hyperparameters, and training methods to attain exceptional accuracy while also gaining insights into their respective advantages and limitations.

The research conducted in this dissertation contributes to reduce the environmental impact associated with pesticide use by facilitating more precise and targeted weed management practices, aligning with the overarching goal of promoting sustainable agriculture. Moreover, the automation of weed classification tasks reduces the reliance on manual labour for field scouting and monitoring, offering both cost savings and environmental benefits as a result.

Furthermore, a comprehensive review of the current state of the art in this field was conducted, with a specific emphasis on the fundamental principles and theoretical aspects related to this subject. This study investigated essential concepts associated with plant agrobiodiversity, underscoring the notion that not all plants can be categorized as weeds, each possessing its unique attributes and advantages. Subsequently, an analysis was undertaken to explore data acquisition methods, vegetation indices, and metrics applicable for model evaluation. Equally important, an overview of PyTorch models and CNN principles were developed, as well as loss functions and optimizers that can be employed. To conclude the state of the art, an analysis of literature was employed, where different projects for image classification, and detection were developed.

Moreover, in the following chapter, the development of this work was elucidated step by step. This included the models that were simulated, an explanation of the data acquisition and preparation processes, as well as the development and simulation of the algorithms. The conduct simulations were evaluated for measuring train and validation accuracy, and loss.

To validate the effectiveness of the simulated classification algorithms, various testing methodologies were employed.

According to the results, three different models stand out. The models MaxVit, ShuffleNet, and EfficientNet and their architectures have achieved in all the tests a high accuracy, near one, as well as a good performance in all the tests. In the first and second experiments, the most critical point involves the dataset. The models contained very few images for each species for train and validation. Training the model with an insufficient dataset could lead to significant failures. Additionally, the number of epochs plays a crucial role. When simulated with a limited number of epochs, even though the models achieved good accuracy, they exhibited a slight decrease.

In these experiments was also possible to conclude that the best combination of hyperparameters for achieving a higher accuracy was with a learning rate equal to 0.0001, a weight decay of zero, and with a sequential configuration of fully connected layers.

When the new large dataset, was noticed a difference in the results, the twelve architectures, from the 3 models, appeared to perform well. They all achieved high results. Greater diversity in a neural network's training data can enhance its ability to generalize and adapt, allowing it to learn and adjust to a wide range of patterns, variations, and real-world scenarios.

This was proved when the models were tested in the web application. All the models trained and validated by the new dataset achieved excellent results, standing out the EfficientNet_B1 and EfficientNet_B5 with an accuracy of 96.15%.

MaxVit, ShuffleNet, and EfficientNet stand out due to their inventive and resource-efficient architectural designs, compared to the other PyTorch models.

The proposed algorithm was able to achieve a remarkable accuracy in image classification accomplished by the EfficientNet_B1 and EfficientNet_B5. These two models were able to have a similar and high performance in all the tests, beating all the other models in the testing phase.

In conclusion, this image classification project has demonstrated the effectiveness of the developed algorithms in achieving outstanding performance.

As we move forward, the continuous refinement and evolution of these algorithms hold great promise for further improving their capabilities and expanding their applicability. This project represents a significant step towards harnessing the potential of AI and ML to solve complex real-world challenges.

5.2. Suggestions of Future Work

With the completion of this dissertation some of the proposed challenges was not able to be fulfilled. So, as future work, the main objective is to complete the simulations of the remaining models. And made another analysis and study of the new models simulated and compared them to the results that were achieved in this dissertation. Also, develop a hyperparameter tuning to search for the best combination of hyperparameters.

Furthermore, one of the purposes is to expand our dataset with more new images and improve the number of classes that we have in this moment, adding more species to this image classification task.

Finally, the last goal is to develop farther the web application so that any user can carry out a task of classification.

References

Afonso, M., Blok, P. M., Polder, G., van der Wolf, J. M., & Kamp, J. (2019). Blackleg Detection in Potato Plants using Convolutional Neural Networks. IFAC-PapersOnLine, 52(30), 6–11. https://doi.org/10.1016/j.ifacol.2019.12.481

Alibabaei, K., Gaspar, P. D., Lima, T. M., Campos, R. M., Girão, I., Monteiro, J., & Lopes, C. M. (2022). A Review of the Challenges of Using Deep Learning Algorithms to Support Decision-Making in Agricultural Activities. Remote Sensing, 14(3), 638. https://doi.org/10.3390/rs14030638

Andrea, C.-C., Mauricio Daniel, B. B., & Jose Misael, J. B. (2017). Precise weed and maize classification through convolutional neuronal networks. 2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM), 1–6. https://doi.org/10.1109/ETCM.2017.8247469

Assuncao, E., Diniz, C., Gaspar, P. D., & Proenca, H. (2020). Decision-making support system for fruit diseases classification using Deep Learning. 2020 International Conference on Decision Aid Sciences and Application (DASA), 652–656. https://doi.org/10.1109/DASA51403.2020.9317219

Assunção, E., Gaspar, P. D., Alibabaei, K., Simões, M. P., Proença, H., Soares, V. N. G. J., & Caldeira, J. M. L. P. (2022). Real-Time Image Detection for Edge Devices: A Peach Fruit Detection Application. Future Internet, 14(11), 323. https://doi.org/10.3390/fi14110323

Assunção, E., Gaspar, P. D., Mesquita, R., Simões, M. P., Alibabaei, K., Veiros, A., & Proença, H. (2022). Real-Time Weed Control Application Using a Jetson Nano Edge Device and a Spray Mechanism. Remote Sensing, 14(17), 4217. https://doi.org/10.3390/rs14174217

Azgomi, H., Haredasht, F. R., & Safari Motlagh, M. R. (2022). Diagnosis of some apple fruit diseases by using image processing and artificial neural network. Food Control, 145, 109484. https://doi.org/10.1016/j.foodcont.2022.109484

Bah, M., Hafiane, A., & Canals, R. (2018). Deep Learning with Unsupervised Data Labeling for Weed Detection in Line Crops in UAV Images. Remote Sensing, 10(11), 1690. https://doi.org/10.3390/rs10111690

Bakhshipour, A., & Jafari, A. (2018). Evaluation of support vector machine and artificial neural networks in weed detection using shape features. Computers and Electronics in Agriculture, 145, 153–160. https://doi.org/10.1016/j.compag.2017.12.032

Boesch, G. (2023, January 1). Deep Residual Networks (ResNet, ResNet50)—2023 Guide. Viso.Ai. https://viso.ai/deep-learning/resnet-residual-neural-network/

Britannica, T. Editors of Encyclopaedia (2022). Herbicide. Encyclopedia Britannica. https://www.britannica.com/science/herbicide

Brownlee, J. (2019, January 24). Understand the Impact of Learning Rate on Neural Network Performance. MachineLearningMastery.Com.

Brownlee, J. (2022, August 9). Difference Between a Batch and an Epoch in a Neural Network. MachineLearningMastery.Com. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

Carvalho, L. B. de. (2013). Plantas Daninhas (1st ed., Vol. vi). Lages – SC.Gaião, D., Nave, A., Teixeira, D., Nunes, L., & Costa, C. A. D. (2019). A biodiversidade da flora associada a ecossistemas agrários com enrelvamento natural. Revista de Ciências Agrárias, 24-31 Páginas. https://doi.org/10.19084/RCA16175

Caulfield, B. (2009, December 17). CPU vs GPU: What's the difference? NVIDIA Blog. https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/

Charlton, M., Sims, M., Coats, T., & Thompson, J. P. (2017). The microcirculation and its measurement in sepsis. Journal of the Intensive Care Society, 18(3), 221–227. https://doi.org/10.1177/1751143716678638

Chen, Z., Yang, J., Feng, Z., & Chen, L. (2022). RSCNet: An Efficient Remote Sensing Scene Classification Model Based on Lightweight Convolution Neural Networks. Electronics, 11(22), 3727. https://doi.org/10.3390/electronics11223727

Corceiro, A., Alibabaei, K., Assunção, E., Gaspar, P. D., & Pereira, N. (2023). Methods for Detecting and Classifying Weeds, Diseases and Fruits Using AI to Improve the Sustainability of Agricultural Crops: A Review. Processes, 11(4), 1263. https://doi.org/10.3390/pr11041263

Deep Learning. (n.d.). Retrieved 1 September 2023, from https://www.deeplearningbook.org/

Diego, A., Simone, C., Vittorio, M., & Marcello, C. (2021). Semantic Segmentation Vineyard Rows [Data set]. Zenodo. https://doi.org/10.5281/zenodo.4601472

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (arXiv:2010.11929). arXiv. http://arxiv.org/abs/2010.11929

Dyrmann, M., Karstoft, H., & Midtiby, H. S. (2016). Plant species classification using deep convolutional neural network. Biosystems Engineering, 151. 72–80. https://doi.org/10.1016/j.biosystemseng.2016.08.024

El-Kenawy, E.-S. M., Khodadadi, N., Mirjalili, S., Makarovskikh, T., Abotaleb, M., Karim, F. K., Alkahtani, H. K., Abdelhamid, A. A., Eid, M. M., Horiuchi, T., Ibrahim, A., & Khafaga, D. S. (2022). Metaheuristic Optimization for Improving Weed Detection in Wheat Images Captured by Drones. Mathematics, 10(23), 4421. https://doi.org/10.3390/math10234421

Espejo-Garcia, B., Mylonas, N., Athanasakos, L., & Fountas, S. (2020). Improving weeds identification with a repository of agricultural pre-trained deep neural networks. Computers and Electronics in Agriculture, 175, 105593. https://doi.org/10.1016/j.compag.2020.105593

European Food Safety Authority. (2022). Pesticides in food: latest report published. https://www.efsa.europa.eu/en/news/pesticides-food-latest-report-published

FAOSTAT. (2022). Food and Agriculture Organization of the United Nations. Crops and livestock products. Retrieved 8 November 2022, from https://www.fao.org/faostat/en/#country/174

FAOSTAT. (2022). Food and Agriculture Organization of the United Nations. Pesticides use. Retrieved 6 November 2022, from https://www.fao.org/faostat/en/#data/QCL/visualize.

FAOSTAT. (2022). Food and Agriculture Organization of the United Nations. Pesticides use. Retrieved 6 November 2022, from https://www.fao.org/faostat/en/#data/RP/visualize

FAOSTAT. (2022). Food and Agriculture Organization of the United Nations. Portugal. Retrieved 8 November 2022, from https://www.fao.org/faostat/en/#country/174

Ferlic, N. (2019). Forward scattering meter for visibility measurements. https://doi.org/10.13140/RG.2.2.20196.12168

Fernández-Aparicio, M., Delavault, P., & Timko, M. P. (2020). Management of Infection by Parasitic Weeds: A Review. Plants, 9(9), 1184. https://doi.org/10.3390/plants9091184

Ferreira, A. D., Freitas, D. M., da Silva, G. G., Pistori, H., & Folhes, M. T. (2019). Unsupervised deep learning and semi-automatic data labeling in weed discrimination. Computers and Electronics in Agriculture, 165, 104963. https://doi.org/10.1016/j.compag.2019.104963

Flora-On | Flora de Portugal. (2022). Retrieved 19 April 2023, from https://flora-on.pt/

Fu, S., Tian, Y., & Tang, L. (2023). Robust regression under the general framework of bounded loss functions. European Journal of Operational Research, 310(3), 1325–1339. https://doi.org/10.1016/j.ejor.2023.04.025

G C, S., Koparan, C., Ahmed, M. R., Zhang, Y., Howatt, K., & Sun, X. (2022). A study on deep learning algorithm performance on weed and crop species identification under different image

background. Artificial Intelligence in Agriculture, 6, 242–256. https://doi.org/10.1016/j.aiia.2022.11.001

G C, S., Zhang, Y., Koparan, C., Ahmed, M. R., Howatt, K., & Sun, X. (2022). Weed and crop species classification using computer vision and deep learning technologies in greenhouse conditions. Journal of Agriculture and Food Research, 9, 100325. https://doi.org/10.1016/j.jafr.2022.100325

Gao, J., Nuyttens, D., Lootens, P., He, Y., & Pieters, J. G. (2018). Recognising weeds in a maize crop using a random forest machine-learning algorithm and near-infrared snapshot mosaic hyperspectral imagery. Biosystems Engineering, 170. 39–50. https://doi.org/10.1016/j.biosystemseng.2018.03.006

GBIF. (n.d.). Retrieved 9 September 2023, from https://www.gbif.org/Uma comunidade para naturalistas · iNaturalist. (n.d.). Retrieved 9 September 2023, from https://www.inaturalist.org/

Guzmán, G., Cabezas, J. M., Sánchez-Cuesta, R., Lora, Á., Bauer, T., Strauss, P., Winter, S., Zaller, J. G., & Gómez, J. A. (2019). A field evaluation of the impact of temporary cover crops on soil properties and vegetation communities in southern Spain vineyards. Agriculture, Ecosystems & Environment, 272, 135–145. https://doi.org/10.1016/j.agee.2018.11.010

Habib, G., & Qureshi, S. (2022). Optimization and acceleration of convolutional neural networks: A survey. Journal of King Saud University - Computer and Information Sciences, 34(7), 4244–4268. https://doi.org/10.1016/j.jksuci.2020.10.004

Hasan, A. S. M. M., Sohel, F., Diepeveen, D., Laga, H., & Jones, M. G. K. (2021). A survey of deep learning techniques for weed detection from images. Computers and Electronics in Agriculture, 184, 106067. https://doi.org/10.1016/j.compag.2021.106067

Hossain, S., Tanzim Reza, M., Chakrabarty, A., & Jung, Y. J. (2023). Aggregating Different Scales of Attention on Feature Variants for Tomato Leaf Disease Diagnosis from Image Data: A Transformer Driven Study. Sensors, 23(7), 3751. https://doi.org/10.3390/s23073751

Huang, H., Deng, J., Lan, Y., Yang, A., Deng, X., & Zhang, L. (2018). A fully convolutional network for weed mapping of unmanned aerial vehicle (UAV) imagery. PLOS ONE, 13(4), e0196302. https://doi.org/10.1371/journal.pone.0196302

Integrated Weed Management / Grape / Agriculture: Pest Management Guidelines / UC Statewide IPM Program (UC IPM). (2015). Retrieved 19 December 2022, from https://www.ipm.ucanr.edu/agriculture/grape/integrated-weed-management/

Islam, N., Rashid, M. M., Wibowo, S., Xu, C.-Y., Morshed, A., Wasimi, S. A., Moore, S., & Rahman, S. M. (2021). Early Weed Detection Using Image Processing and Machine Learning Techniques

in an Australian Chilli Farm. Agriculture, 11(5), 387. https://doi.org/10.3390/agriculture11050387

Jin, X., Che, J., & Chen, Y. (2021). Weed Identification Using Deep Learning and Image Processing in Vegetable Plantation. IEEE Access, 9, 10940–10950. https://doi.org/10.1109/ACCESS.2021.3050296

Jin, X., Che, J., & Chen, Y. (2021). Weed Identification Using Deep Learning and Image Processing in Vegetable Plantation. IEEE Access, 9, 10940–10950. https://doi.org/10.1109/ACCESS.2021.3050296

Kamath, R., Balachandra, M., Vardhan, A., & Maheshwari, U. (2022). Classification of paddy crop and weeds using semantic segmentation. Cogent Engineering, 9(1), 2018791. https://doi.org/10.1080/23311916.2021.2018791

Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. Computers and Electronics in Agriculture, 147, 70–90. https://doi.org/10.1016/j.compag.2018.02.016

Kapoor, R., Sharma, D., & Gulati, T. (2021). State of the art content based image retrieval techniques using deep learning: A survey. Multimedia Tools and Applications, 80(19), 29561–29583. https://doi.org/10.1007/s11042-021-11045-1

Kerkech, M., Hafiane, A., & Canals, R. (2020). Vine disease detection in UAV multispectral images using optimized image registration and deep learning segmentation approach. Computers and Electronics in Agriculture, 174, 105446. https://doi.org/10.1016/j.compag.2020.105446

Kim, J., Suh, Y., Lee, J., Chae, H., Ahn, H., Chung, Y., & Park, D. (2022). EmbeddedPigCount: Pig Counting with Video Object Detection and Tracking on an Embedded Board. Sensors, 22(7), 2689. https://doi.org/10.3390/s22072689

Learn And Code Confusion Matrix With Python. (2019). Retrieved 6 December 2022, from https://www.nbshare.io/notebook/626706996/Learn-And-Code-Confusion-Matrix-With-Python/

Littman L. M., Ajunwa I., Berger G., Boutilier C., Currie M., Doshi-Velez F., Hadfield G., Horowitz C. M., Isbell C., Kitano H., Levy K., Lyons T., Mitchell M., Shah J., Sloman S., Vallor S., and Walsh T. (2021. September) Gathering Strength, Gathering Storms: The One Hundred Year Study on Artificial Intelligence (AI100) 2021 Study Panel Report. Stanford University. Retrieved 6 November 2022, from http://ai100.stanford.edu/2021-report.

Liu, H., Yao, D., Yang, J., & Li, X. (2019). Lightweight Convolutional Neural Network and Its Application in Rolling Bearing Fault Diagnosis under Variable Working Conditions. Sensors, 19(22), 4827. https://doi.org/10.3390/s19224827

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., & Guo, B. (2021). Swin Transformer: Hierarchical Vision Transformer using Shifted Windows (arXiv:2103.14030). arXiv. http://arxiv.org/abs/2103.14030

Lottes, P., Behley, J., Milioto, A., & Stachniss, C. (2018). Fully Convolutional Networks With Sequential Information for Robust Crop and Weed Detection in Precision Farming. IEEE Robotics and Automation Letters, 3(4), 2870–2877. https://doi.org/10.1109/LRA.2018.2846289

Luis Quintanilla. (2022, March 18). Machine learning tasks—ML.NET. https://learn.microsoft.com/en-us/dotnet/machine-learning/resources/tasks

Lv, Q., Zhang, S., & Wang, Y. (2022). Deep Learning Model of Image Classification Using Machine Learning. Advances in Multimedia, 2022, 1–12. https://doi.org/10.1155/2022/3351256

Ma, N., Zhang, X., Zheng, H.-T., & Sun, J. (2018). ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design (arXiv:1807.11164). arXiv. http://arxiv.org/abs/1807.11164

Ma, X., Deng, X., Qi, L., Jiang, Y., Li, H., Wang, Y., & Xing, X. (2019). Fully convolutional network for rice seedling and weed image segmentation at the seedling stage in paddy fields. PLOS ONE, 14(4), e0215676. https://doi.org/10.1371/journal.pone.0215676

MacLaren, C., Storkey, J., Menegat, A., Metcalfe, H., & Dehnen-Schmutz, K. (2020). An ecological future for weed science to sustain crop production and the environment. A review. Agronomy for Sustainable Development, 40(4), 24. https://doi.org/10.1007/s13593-020-00631-6

Mao, D., Sun, H., Li, X., Yu, X., Wu, J., & Zhang, Q. (2022). Real-time fruit detection using deep neural networks on CPU (RTFD): An edge AI application. Computers and Electronics in Agriculture, 204, 107517. https://doi.org/10.1016/j.compag.2022.107517

Milioto, A., Lottes, P., & Stachniss, C. (2018). Real-Time Semantic Segmentation of Crop and Weed for Precision Agriculture Robots Leveraging Background Knowledge in CNNs. 2018 IEEE International Conference on Robotics and Automation (ICRA), 2229–2235. https://doi.org/10.1109/ICRA.2018.8460962

Mishra, M. (2020. September 2). Convolutional Neural Networks, Explained. Medium. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

Moazzam, S. I., Khan, U. S., Tiwana, M. I., Iqbal, J., Qureshi, W. S., & Shah, S. I. (2019). A Review of Application of Deep Learning for Weeds and Crops Classification in Agriculture. 2019 International Conference on Robotics and Automation in Industry (ICRAI), 1–6. https://doi.org/10.1109/ICRAI47710.2019.8967350

Mu, Y., Feng, R., Ni, R., Li, J., Luo, T., Liu, T., Li, X., Gong, H., Guo, Y., Sun, Y., Bao, Y., Li, S., Wang, Y., & Hu, T. (2022). A Faster R-CNN-Based Model for the Identification of Weed Seedling. Agronomy, 12(11), 2867. https://doi.org/10.3390/agronomy12112867

Nabi, J. (2019, March 16). Hyper-parameter Tuning Techniques in Deep Learning. Medium. https://towardsdatascience.com/hyper-parameter-tuning-techniques-in-deep-learning-4dad592c63c8

Nations, U. (n.d.). Population. United Nations; United Nations. Retrieved 8 November 2022, from https://www.un.org/en/global-issues/population

Nations, U. (n.d.). Water. United Nations; United Nations. Retrieved 8 November 2022, from https://www.un.org/en/global-issues/water

Osorio, K., Puerto, A., Pedraza, C., Jamaica, D., & Rodríguez, L. (2020). A Deep Learning Approach for Weed Detection in Lettuce Crops Using Multispectral Images. AgriEngineering, 2(3), 471–488. https://doi.org/10.3390/agriengineering2030032

O'Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks (arXiv:1511.08458). arXiv. http://arxiv.org/abs/1511.08458

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library (arXiv:1912.01703). arXiv. http://arxiv.org/abs/1912.01703

Pereira, C. S., Morais, R., & Reis, M. J. C. S. (2019). Deep Learning Techniques for Grape Plant Species Identification in Natural Images. Sensors, 19(22), 4850. https://doi.org/10.3390/s19224850

Performance Metrics in Machine Learning—Javatpoint. (2021). Www.Javatpoint.Com. Retrieved 6 December 2022, from https://www.javatpoint.com/performance-metrics-in-machine-learning

Peña, J., Torres-Sánchez, J., Serrano-Pérez, A., de Castro, A., & López-Granados, F. (2015). Quantifying Efficacy and Limits of Unmanned Aerial Vehicle (UAV) Technology for Weed Seedling Detection as Affected by Sensor Resolution. Sensors, 15(3), 5609–5626. https://doi.org/10.3390/s150305609

Radosavovic, I., Kosaraju, R. P., Girshick, R., He, K., & Dollár, P. (2020). Designing Network Design Spaces (arXiv:2003.13678). arXiv. http://arxiv.org/abs/2003.13678

Rushikanjaria. (2021. July 6). Classification Model Performance Evaluation using AUC-ROC and CAP Curves. Geek Culture. https://medium.com/geekculture/classification-model-performance-evaluation-using-auc-roc-and-cap-curves-66a1b3fc0480

Sa, I., Chen, Z., Popovic, M., Khanna, R., Liebisch, F., Nieto, J., & Siegwart, R. (2018). weedNet: Dense Semantic Weed Classification Using Multispectral Images and MAV for Smart Farming. IEEE Robotics and Automation Letters, 3(1), 588–595. https://doi.org/10.1109/LRA.2017.2774979

Santos, T. T., de Souza, L. L., dos Santos, A. A., & Avila, S. (2020). Grape detection, segmentation, and tracking using deep neural networks and three-dimensional association. Computers and Electronics in Agriculture, 170. 105247. https://doi.org/10.1016/j.compag.2020.105247

SEP 4, J. S., & Read, 2020 6 Min. (2020. September 4). Train, Validation, Test Split and Why You Need It. Roboflow Blog. https://blog.roboflow.com/train-test-split/

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition (arXiv:1409.1556). arXiv. http://arxiv.org/abs/1409.1556

Sishodia, R. P., Ray, R. L., & Singh, S. K. (2020). Applications of Remote Sensing in Precision Agriculture: A Review. Remote Sensing, 12(19), 3136. https://doi.org/10.3390/rs12193136

Sujaritha, M., Annadurai, S., Satheeshkumar, J., Kowshik Sharan, S., & Mahesh, L. (2017). Weed detecting robot in sugarcane fields using fuzzy real time classifier. Computers and Electronics in Agriculture, 134, 160–171. https://doi.org/10.1016/j.compag.2017.01.008

Swift, A., Heale, R., & Twycross, A. (2020). What are sensitivity and specificity? Evidence-Based Nursing, 23(1), 2–4. https://doi.org/10.1136/ebnurs-2019-103225

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision (arXiv:1512.00567). arXiv. http://arxiv.org/abs/1512.00567

Tamirat Wato, M. A. (2020). The Agricultural Water Pollution and Its Minimization Strategies – A Review. Journal of Resources Development and Management. https://doi.org/10.7176/JRDM/64-02

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). MnasNet: Platform-Aware Neural Architecture Search for Mobile (arXiv:1807.11626). arXiv. http://arxiv.org/abs/1807.11626

Taye, M. M. (2023). Understanding of Machine Learning with Deep Learning: Architectures, Workflow, Applications and Future Directions. Computers, 12(5), 91. https://doi.org/10.3390/computers12050091

Team, G. (n.d.). Quickstart. https://www.gradio.app/guides/quickstart

Tripathi, A. D., Mishra, R., Maurya, K. K., Singh, R. B., & Wilson, D. W. (2019). Estimates for World Population and Global Food Availability for Global Health. In The Role of Functional Food Security in Global Health (pp. 3–24). Elsevier. https://doi.org/10.1016/B978-0-12-813148-0.00001-3

Tsouros, D. C., Bibi, S., & Sarigiannidis, P. G. (2019). A Review on UAV-Based Applications for Precision Agriculture. Information, 10(11), 349. https://doi.org/10.3390/info10110349

Tu, Z., Talebi, H., Zhang, H., Yang, F., Milanfar, P., Bovik, A., & Li, Y. (2022). MaxViT: Multi-Axis Vision Transformer (arXiv:2204.01697). arXiv. http://arxiv.org/abs/2204.01697

Unzueta, D. (2022, March 15). Convolutional Layers vs Fully Connected Layers. Medium. https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b

Using Deep Learning Models / Convolutional Neural Networks. (n.d.). Retrieved 1 September 2023, from https://docs.ecognition.com/eCognition_documentation/User%20Guide%20Developer/8%20Classification%20-%20Deep%20Learning.htm

UTAD, J. B. (2023). Jardim Botânico UTAD | Espécie Lemna minor. Jardim Botânico UTAD. Retrieved 15 April 2023, from https://jb.utad.pt/especie/Lemna_minor

Vasani, D. (2019, November 18). This thing called Weight Decay. Medium. https://towardsdatascience.com/this-thing-called-weight-decay-a7cd4bcfccab

Veiros, A., Mesquita, R., Gaspar, P., & Simões, M. P. (2022). Multitask robotic rover for agricultural activities (R2A2): A robotic platform for peach orchards. Acta Horticulturae, 1352, 89–96. https://doi.org/10.17660/ActaHortic.2022.1352.12

Wang, A., Xu, Y., Wei, X., & Cui, B. (2020). Semantic Segmentation of Crop and Weed using an Encoder-Decoder Network and Image Enhancement Method under Uncontrolled Outdoor Illumination. IEEE Access, 8, 81724–81734. https://doi.org/10.1109/ACCESS.2020.2991354

Wang, A., Zhang, W., & Wei, X. (2019). A review on weed detection using ground-based machine vision and image processing techniques. Computers and Electronics in Agriculture, 158, 226–240. https://doi.org/10.1016/j.compag.2019.02.005

Wang, S.-C. (2003). Artificial Neural Network. In S.-C. Wang (Ed.), Interdisciplinary Computing in Java Programming (pp. 81–100). Springer US. https://doi.org/10.1007/978-1-4615-0377-4_5

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). *Aggregated Residual Transformations for Deep Neural Networks*(arXiv:1611.05431)arXiv. http://arxiv.org/abs/1611.05431

Yang, X., Ye, Y., Li, X., Lau, R. Y. K., Zhang, X., & Huang, X. (2018). Hyperspectral Image Classification With Deep Learning Models. IEEE Transactions on Geoscience and Remote Sensing, 56(9), 5408–5423. https://doi.org/10.1109/TGRS.2018.2815613

Yashwanth, M., Chandra, M. L., Pallavi, K., Showkat, D., & Kumar, P. S. (2020). Agriculture Automation using Deep Learning Methods Implemented using Keras. 2020 IEEE International Conference for Innovation in Technology (INOCON), 1–6. https://doi.org/10.1109/INOCON50539.2020.9298415

Zagoruyko, S., & Komodakis, N. (2017). *Wide Residual Networks* (arXiv:1605.07146). arXiv. http://arxiv.org/abs/1605.07146

Zhang, Z. (2018). Artificial Neural Network. In Z. Zhang (Ed.), Multivariate Time Series Analysis in Climate and Environmental Research (pp. 1–35). Springer International Publishing. https://doi.org/10.1007/978-3-319-67340-0_1

Zhao, G., Liu, G., Fang, L., Tu, B., & Ghamisi, P. (2019). Multiple convolutional layers fusion framework for hyperspectral image classification. Neurocomputing, 339, 149–160. https://doi.org/10.1016/j.neucom.2019.02.019

Appendix A

A.1. Loss and accuracy performance across all the simulated PyTorch classification models.

Models	Learning rate	Weight decay	Number of layers	Best acc train	Best acc validation	Best loss train	Best loss validation	Number of parameters
MobileNetV2	0.01	Default	Sequential	0.2874	0.2803	1.7600	1.4300	3044742
MobileNetV2	0.01	Default	Sequential	0.2580	0.2796	2,1940	1.9450	3044742
MobileNetV2	0.01	Default	Sequential	0.2586	0.2802	1.8310	1.9310	3044742
MobileNetV2	0.01	Default	Linear	0.2935	0.3600	1.7289	1.3300	2231558
MobileNetV2	0.01	0.0001	Linear	0.2797	0.3200	1.7390	1.2329	2231558
MobileNetV2	0.01	0.0001	Sequential	0.2790	0.2800	1.7400	0.5400	3044742
MobileNetV2	0.001	Default	Sequential	0.2797	0.4000	1.7563	1.5690	3044742
MobileNetV2	0.001	Default	Linear	0.3300	0.3200	1.7100	1.5000	2231558
MobileNetV2	0.001	0.0001	Linear	0.3000	0.4000	1.7200	1.2400	2231558
MobileNetV2	0.001	0.0001	Sequential	0.2600	0.4000	1.7500	1.5700	3044742
MobileNetV2	0.0001	Default	Sequential	0.3300	0.4000	1.6760	1.5390	3044742
MobileNetV2	0.0001	0.0001	Linear	0.3300	0.4000	1.6400	1.5400	2231558
MobileNetV2	0.0001	0.0001	Sequential	0.3300	0.4400	1.6700	1.5600	3044742
MobileNetV2	0.0001	Default	Linear	0.3040	0.3600	1.6700	1.1800	2231558
MobileNetV3_Large	0.01	Default	Sequential	0.3000	0.3600	1.7900	1.6100	5022902
MobileNetV3_Large	0.01	Default	Linear	0.2500	0.3600	1.7737	1.0300	4209718
MobileNetV3_Large	0.01	0.0001	Linear	0.2587	0.4400	1.6300	1.1280	4209718
MobileNetV3_Large	0.01	0.0001	Sequential	0.2517	0.4000	1.7430	0.7491	5022902
MobileNetV3_Large	0.001	Default	Sequential	0.2800	0.3600	1.7500	1.4770	5022902
MobileNetV3_Large	0.001	Default	Linear	0.3712	0.4400	1.6400	1.1600	4209718
MobileNetV3_Large	0.001	0.0001	Linear	0.3357	0.4000	1.6700	0.0001	4209718
MobileNetV3_Large	0.001	0.0001	Sequential	0.2867	0.4000	1.6800	0.0005	5022902

MobileNetV3_Large	0.0001	Default	Sequential	0.3625	0.4400	1.6600	1.4500	5022902
MobileNetV3_Large	0.0001	Default	Linear	0.3357	0.4400	1.6821	1.1900	4209718
MobileNetV3_Large	0.0001	0.0001	Linear	0.3427	0.3600	1.6589	1.2662	4209718
MobileNetV3_Large	0.0001	0.0001	Sequential	0.3357	0.4000	1.6800	1.3911	5022902
MobilenetV3_Small	0.01	Default	Sequential	0.2700	0.3600	1.7400	0.8200	2207654
MobilenetV3_Small	0.01	Default	Linear	0.2378	0.3200	1.7700	1.2091	1524006
MobilenetV3_Small	0.01	0.0001	Linear	0.2500	0.4000	1.7500	0.7520	1524006
MobilenetV3_Small	0.01	0.0001	Sequential	0.2650	0.3200	1.7700	0.1700	2207654
MobilenetV3_Small	0.001	Default	Sequential	0.2800	0.3600	1.7500	1.3700	2207654
MobilenetV3_Small	0.001	Default	Linear	0.3070	0.3600	1.7200	1.1090	1524006
MobilenetV3_Small	0.001	0.0001	Linear	0.2867	0.4000	1.7400	1.2100	1524006
MobilenetV3_Small	0.001	0.0001	Sequential	0.2657	0.4400	1.7640	1.3600	2207654
MobilenetV3_Small	0.0001	Default	Sequential	0.3300	0.4000	1.7100	1.3100	2207654
MobilenetV3_Small	0.0001	Default	Linear	0.3000	0.3600	1.5100	0.9900	1524006
MobilenetV3_Small	0.0001	0.0001	Linear	0.3200	0.4000	1.7100	1.2680	1524006
MobilenetV3_Small	0.0001	0.0001	Sequential	0.3200	0.4000	1.7000	1.3400	2207654
MaxVit	0.01	Default	Sequential	0.3210	0.5600	1.6700	0.7400	30572622
MaxVit	0.01	Default	Linear	0.2500	0.2800	2,4200	0.9100	30410702
MaxVit	0.01	0.0001	Linear	0.2400	0.2000	1.7700	0.6900	30410702
MaxVit	0.01	0.0001	Sequential	0.2500	0.2800	1.7700	1.1500	30572622
MaxVit	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0000	30572622
MaxVit	0.001	Default	Linear	1.0000	0.9600	0.0005	0.0002	30410702
MaxVit	0.001	0.0001	Linear	1.0000	0.9600	0.0004	0.0003	30410702
MaxVit	0.001	0.0001	Sequential	1.0000	0.9600	0.0007	0.0000	30572622
MaxVit	0.0001	Default	Sequential	1.0000	0.9600	0.0010	0.0001	30572622
MaxVit	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	30410702
MaxVit	0.0001	0.0001	Linear	1.0000	0.9600	0.0003	0.0001	30410702
MaxVit	0.0001	0.0001	Sequential	1.0000	0.9600	0.0010	0.0001	30572622

AlexNet	0.01	Default	Sequential	0.2300	0.2800	1.9100	0.0001	68182470
AlexNet	0.01	Default	Linear	0.2400	0.2800	1.7700	1.5600	57028422
AlexNet	0.01	0.0001	Linear	0.2600	0.2800	1.7600	0.9900	57028422
AlexNet	0.01	0.0001	Sequential	0.2238	0.2800	87487,0000	3429,0000	68182470
AlexNet	0.001	Default	Sequential	0.2238	0.2800	1.7700	1.5600	68182470
AlexNet	0.001	Default	Linear	0.2300	0.2800	1.7700	1.5900	57028422
AlexNet	0.001	0.0001	Linear	0.2300	0.2800	1.7600	1.5100	57028422
AlexNet	0.001	0.0001	Sequential	0.2900	0.2800	1.7700	0.4200	68182470
AlexNet	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	68182470
AlexNet	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	57028422
AlexNet	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	57028422
AlexNet	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	68182470
GoogLeNet	0.01	Default	Sequential	0.5600	0.6800	1.0500	0.5160	6289702
GoogLeNet	0.01	Default	Linear	1.0000	0.9600	0.0300	0.0006	5606054
GoogLeNet	0.01	0.0001	Linear	0.7700	0.6800	0.6800	0.1322	5606054
GoogLeNet	0.01	0.0001	Sequential	0.2900	0.4000	1.7100	0.6850	6289702
GoogLeNet	0.001	Default	Sequential	1.0000	0.8400	0.0008	0.0001	6289702
GoogLeNet	0.001	Default	Linear	1.0000	0.9600	0.0003	0.0001	5606054
GoogLeNet	0.001	0.0001	Linear	1.0000	0.9600	0.0008	0.0007	5606054
GoogLeNet	0.001	0.0001	Sequential	1.0000	0.8400	0.0200	0.0001	6289702
GoogLeNet	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	6289702
GoogLeNet	0.0001	Default	Linear	1.0000	0.9600	0.0030	0.0002	5606054
GoogLeNet	0.0001	0.0001	Linear	1.0000	0.9600	0.0010	0.0020	5606054
GoogLeNet	0.0001	0.0001	Sequential	1.0000	0.9600	0.0002	0.0001	6289702
Vit_b_16	0.01	Default	Sequential	0.9800	0.6800	0.1420	0.0007	87126382
Vit_b_16	0.01	Default	Linear	0.7552	0.6800	0.6800	0.0912	8657227
Vit_b_16	0.01	0.0001	Linear	0.7800	0.5600	0.5910	0.1778	8657227
Vit_b_16	0.01	0.0001	Sequential	0.7400	0.6400	0.7470	0.1179	87126382

Vit_b_16	0.001	Default	Sequential	0.8300	0.5200	0.4422	0.0225	87126382
Vit_b_16	0.001	Default	Linear	1.0000	0.8400	0.0001	0.0001	8657227
Vit_b_16	0.001	0.0001	Linear	1.0000	0.8400	0.0008	0.0002	8657227
Vit_b_16	0.001	0.0001	Sequential	1.0000	0.8000	0.0010	0.0001	87126382
Vit_b_16	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	87126382
Vit_b_16	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	86572270
Vit_b_16	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	86572270
Vit_b_16	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	87126382
Vit_b_32	0.01	Default	Sequential	0.9650	0.6000	0.1010	0.0001	88914030
Vit_b_32	0.01	Default	Linear	0.8400	0.5200	0.3500	0.0270	88230382
Vit_b_32	0.01	0.0001	Linear	0.2000	0.2800	1.9000	1.0000	88230382
Vit_b_32	0.01	0.0001	Sequential	1.0000	0.5600	0.0270	0.0003	88914030
Vit_b_32	0.001	Default	Sequential	0.8100	0.6400	0.5400	0.0290	88914030
Vit_b_32	0.001	Default	Linear	1.0000	0.4400	0.0001	0.0001	88230382
Vit_b_32	0.001	0.0001	Linear	1.0000	0.8900	0.0010	0.0002	88230382
Vit_b_32	0.001	0.0001	Sequential	1.0000	0.6400	0.0001	0.0001	88914030
Vit_b_32	0.0001	Default	Sequential	1.0000	0.8400	0.0001	0.0001	88914030
Vit_b_32	0.0001	Default	Linear	1.0000	0.8000	0.0001	0.0001	88230382
Vit_b_32	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	88230382
Vit_b_32	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	88914030
Vit_l_16	Without data due to the computer graphics.							
Vit_l_32	Without data due to the computer graphics.							
Vit_h_14	Without data due to the computer graphics.							
ResNeXt50_32x4d	0.01	Default	Sequential	0.2500	0.3600	1.7800	0.6900	23669702
ResNeXt50_32x4d	0.01	Default	Linear	1.0000	0.4400	0.0001	0.0001	23669702
ResNeXt50_32x4d	0.01	0.0001	Linear	0.2700	0.3600	1.7700	1.0500	23669702
ResNeXt50_32x4d	0.01	0.0001	Sequential	0.2500	0.4400	1.7700	0.2800	23669702
ResNeXt50_32x4d	0.001	Default	Sequential	1.0000	0.5200	0.0001	0.0001	23669702

ResNeXt50_32x4d	0.001	Default	Linear	1.0000	0.4000	0.0001	0.0001	23669702
ResNeXt50_32x4d	0.001	0.0001	Linear	1.0000	0.4000	0.0030	0.0006	23669702
ResNeXt50_32x4d	0.001	0.0001	Sequential	0.2400	0.4000	1.7700	1.5500	23669702
ResNeXt50_32x4d	0.0001	Default	Sequential	1.0000	0.5200	0.0001	0.0001	23669702
ResNeXt50_32x4d	0.0001	Default	Linear	1.0000	0.7600	0.0001	0.0001	23669702
ResNeXt50_32x4d	0.0001	0.0001	Linear	1.0000	0.8000	0.0005	0.0004	23669702
ResNeXt50_32x4d	0.0001	0.0001	Sequential	1.0000	0.4800	0.0002	0.0001	23669702
ResNeXt101_32x8d	0.01	Default	Sequential	0.3400	0.4400	1.6800	0.5200	87956422
ResNeXt101_32x8d	0.01	Default	Linear	0.2300	0.3600	1.7700	1.4200	86754630
ResNeXt101_32x8d	0.01	0.0001	Linear	0.2500	0.3200	1.7700	1.0700	86754630
ResNeXt101_32x8d	0.01	0.0001	Sequential	0.4200	0.4800	1.6000	0.1870	87956422
ResNeXt101_32x8d	0.001	Default	Sequential	0.2700	0.3600	1.7600	1.2200	87956422
ResNeXt101_32x8d	0.001	Default	Linear	1.0000	0.4000	0.0001	0.0001	86754630
ResNeXt101_32x8d	0.001	0.0001	Linear	1.0000	0.4800	0.0040	0.0020	86754630
ResNeXt101_32x8d	0.001	0.0001	Sequential	0.2500	0.2800	1.7700	1.4800	87956422
ResNeXt101_32x8d	0.0001	Default	Sequential	0.2797	0.4000	1.7000	1.2100	87956422
ResNeXt101_32x8d	0.0001	Default	Linear	0.2168	0.3600	1.7700	1.1300	86754630
ResNeXt101_32x8d	0.0001	0.0001	Linear	0.2378	0.3200	1.7700	1.3000	86754630
ResNeXt101_32x8d	0.0001	0.0001	Sequential	0.2587	0.2800	1.7800	0.0100	87956422
ResNeXt101_64x4d	0.01	Default	Sequential	0.9200	0.4800	0.2500	0.0010	82620358
ResNeXt101_64x4d	0.01	Default	Linear	1.0000	0.3600	0.0001	0.0001	81418566
ResNeXt101_64x4d	0.01	0.0001	Linear	0.2517	0.2600	1.7700	0.5700	81418566
ResNeXt101_64x4d	0.01	0.0001	Sequential	0.3000	0.4400	1.7000	0.1200	82620358
ResNeXt101_64x4d	0.001	Default	Sequential	1.0000	0.4000	0.0001	0.0001	82620358
ResNeXt101_64x4d	0.001	Default	Linear	1.0000	0.4000	0.0001	0.0001	81418566
ResNeXt101_64x4d	0.001	0.0001	Linear	1.0000	0.4800	0.0010	0.0001	81418566
ResNeXt101_64x4d	0.001	0.0001	Sequential	0.2500	0.3600	1.7700	1.3200	82620358
ResNeXt101_64x4d	0.0001	Default	Sequential	1.0000	0.7200	0.0001	0.0001	82620358

ResNeXt101_64x4d	0.0001	Default	Linear	1.0000	0.5600	0.0001	0.0001	81418566
ResNeXt101_64x4d	0.0001	0.0001	Linear	1.0000	0.6400	0.0002	0.0001	81418566
ResNeXt101_64x4d	0.0001	0.0001	Sequential	1.0000	0.7600	0.0005	0.0002	82620358
ResNet18	0.01	Default	Sequential	1.0000	0.3600	0.0001	0.0001	12379310
ResNet18	0.01	Default	Linear	1.0000	0.3600	0.0001	0.0001	11179590
ResNet18	0.01	0.0001	Linear	0.2500	0.4700	1.7700	1.1600	11179590
ResNet18	0.01	0.0001	Sequential	0.2500	0.3600	1.8000	1.0900	12379310
ResNet18	0.001	Default	Sequential	1.0000	0.3200	0.0001	0.0001	12379310
ResNet18	0.001	Default	Linear	1.0000	0.3600	0.0001	0.0001	11179590
ResNet18	0.001	0.0001	Linear	1.0000	0.3200	0.0010	0.0005	11179590
ResNet18	0.001	0.0001	Sequential	1.0000	0.4400	0.0007	0.0003	12379310
ResNet18	0.0001	Default	Sequential	1.0000	0.4800	0.0001	0.0001	12379310
ResNet18	0.0001	Default	Linear	1.0000	0.8400	0.0001	0.0001	11179590
ResNet18	0.0001	0.0001	Linear	1.0000	0.5200	0.0001	0.0001	11179590
ResNet18	0.0001	0.0001	Sequential	1.0000	0.5600	0.0001	0.0001	12379310
ResNet34	0.01	Default	Sequential	1.0000	0.4000	0.0001	0.0001	21962670
ResNet34	0.01	Default	Linear	1.0000	0.4000	0.0001	0.0001	21287750
ResNet34	0.01	0.0001	Linear	0.2500	0.3200	1.7600	1.2100	21287750
ResNet34	0.01	0.0001	Sequential	0.2500	0.3200	1.7900	0.7377	21962670
ResNet34	0.001	Default	Sequential	1.0000	0.4800	0.0001	0.0001	21962670
ResNet34	0.001	Default	Linear	1.0000	0.4000	0.0001	0.0001	21287750
ResNet34	0.001	0.0001	Linear	1.0000	0.4800	0.0010	0.0003	21287750
ResNet34	0.001	0.0001	Sequential	1.0000	0.5200	0.0005	0.0001	21962670
ResNet34	0.0001	Default	Sequential	1.0000	0.8400	0.0001	0.0001	21962670
ResNet34	0.0001	Default	Linear	1.0000	0.5600	0.0001	0.0001	21287750
ResNet34	0.0001	0.0001	Linear	1.0000	0.7200	0.0006	0.0004	21287750
ResNet34	0.0001	0.0001	Sequential	1.0000	0.6400	0.0003	0.0002	21962670
ResNet50	0.01	Default	Sequential	1.0000	0.3600	0.0001	0.0001	28345006

ResNet50	0.01	Default	Linear	1.0000	0.4400	0.0001	0.0001	23520326
ResNet50	0.01	0.0001	Linear	0.2300	0.3200	1.7700	0.7200	23520326
ResNet50	0.01	0.0001	Sequential	0.2500	0.3200	1.8100	0.9600	28345006
ResNet50	0.001	Default	Sequential	1.0000	0.4000	0.0001	0.0001	28345006
ResNet50	0.001	Default	Linear	1.0000	0.3600	0.0001	0.0001	23520326
ResNet50	0.001	0.0001	Linear	0.9930	0.4400	0.0609	0.0086	23520326
ResNet50	0.001	0.0001	Sequential	1.0000	0.4400	0.0026	0.0003	28345006
ResNet50	0.0001	Default	Sequential	1.0000	0.4400	0.0001	0.0001	28345006
ResNet50	0.0001	Default	Linear	1.0000	0.8000	0.0002	0.0002	23520326
ResNet50	0.0001	0.0001	Linear	0.4615	0.3600	1.2485	0.1070	23520326
ResNet50	0.0001	0.0001	Sequential	1.0000	0.5200	0.0010	0.0003	28345006
ConvNeXt_Tiny	0.01	Default	Sequential	0.2448	0.2800	1.7700	1.0490	28378854
ConvNeXt_Tiny	0.01	Default	Linear	0.2400	0.2100	1.7700	1.4900	27824742
ConvNeXt_Tiny	0.01	0.0001	Linear	0.9700	0.6400	0.1000	0.0060	27824742
ConvNeXt_Tiny	0.01	0.0001	Sequential	0.2700	0.4000	1.7200	1.1400	28378854
ConvNeXt_Tiny	0.001	Default	Sequential	0.2600	0.2800	1.7700	1.5600	28378854
ConvNeXt_Tiny	0.001	Default	Linear	0.9800	0.9900	0.0500	0.0002	27824742
ConvNeXt_Tiny	0.001	0.0001	Linear	1.0000	0.8800	0.0060	0.0003	27824742
ConvNeXt_Tiny	0.001	0.0001	Sequential	0.4100	0.5200	1.3500	0.6700	28378854
ConvNeXt_Tiny	0.0001	Default	Sequential	1.0000	0.9930	0.0001	0.0001	28378854
ConvNeXt_Tiny	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	27824742
ConvNeXt_Tiny	0.0001	0.0001	Linear	1.0000	0.9930	0.0001	0.0001	27824742
ConvNeXt_Tiny	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	28378854
Convnext_small	0.01	Default	Sequential	0.2600	0.2800	1.7700	1.0200	50013414
Convnext_small	0.01	Default	Linear	0.2600	0.2800	1.7700	1.3200	49459302
Convnext_small	0.01	0.0001	Linear	0.2500	0.3600	1.7400	1.3700	49459302
Convnext_small	0.01	0.0001	Sequential	0.2378	0.2800	1.7700	1.0800	50013414
Convnext_small	0.001	Default	Sequential	0.2600	0.2800	1.7700	1.2900	50013414

Convnext_small	0.001	Default	Linear	0.9300	0.5600	0.1600	0.0005	49459302
Convnext_small	0.001	0.0001	Linear	0.8800	0.9930	0.0400	0.0006	49459302
Convnext_small	0.001	0.0001	Sequential	0.2100	0.2800	1.7700	1.5200	50013414
Convnext_small	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	50013414
Convnext_small	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	49459302
Convnext_small	0.0001	0.0001	Linear	1.0000	0.9600	0.0002	0.0001	49459302
Convnext_small	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	50013414
Convnext_base	0.01	Default	Sequential	0.2657	0.2800	1.7700	1.2800	88256262
Convnext_base	0.01	Default	Linear	0.2500	0.2800	1.7800	1.3700	87572614
Convnext_base	0.01	0.0001	Linear	0.2700	0.2800	1.7800	1.4023	87572614
Convnext_base	0.01	0.0001	Sequential	0.2587	0.2800	1.7700	0.9641	88256262
Convnext_base	0.001	Default	Sequential	0.2650	0.2800	1.7600	1.4100	88256262
Convnext_base	0.001	Default	Linear	0.9720	0.6000	0.0030	0.1300	87572614
Convnext_base	0.001	0.0001	Linear	0.7692	0.6800	0.6308	0.2645	87572614
Convnext_base	0.001	0.0001	Sequential	0.3497	0.4000	1.5321	1.2337	88256262
Convnext_base	0.0001	Default	Sequential	1.0000	0.9600	0.0000	0.0000	88256262
Convnext_base	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	87572614
Convnext_base	0.0001	0.0001	Linear	1.0000	0.9600	0.0002	0.0001	87572614
Convnext_base	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	88256262
Convnext_large	0.01	Default	Sequential	0.2657	0.2098	1.7700	1.4600	197969478
Convnext_large	0.01	Default	Linear	0.2098	0.2657	1.7800	1.3100	196239558
Convnext_large	0.01	0.0001	Linear	0.2587	0.2727	1.7700	1.1100	196239558
Convnext_large	0.01	0.0001	Sequential	0.2449	0.2098	1.7700	1.3200	197969478
Convnext_large	0.001	Default	Sequential	0.2517	0.2098	1.7700	1.5700	197969478
Convnext_large	0.001	Default	Linear	0.2517	0.2800	177,0000	1.5788	196239558
Convnext_large	0.001	0.0001	Linear	0.4965	0.6923	1.6187	1.0236	196239558
Convnext_large	0.001	0.0001	Sequential	0.3100	0.5600	1.6300	1.1250	197969478
Convnext_large	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	197969478

Convnext_large	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	196239558
Convnext_large	0.0001	0.0001	Linear	1.0000	0.9600	0.0002	0.0001	196239558
Convnext_large	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	197969478
Wide_ResNet50_2	0.01	Default	Sequential	0.4200	0.4000	1.5400	0.0005	69622214
Wide_ResNet50_2	0.01	Default	Linear	1.0000	0.4000	0.0001	0.0001	66846534
Wide_ResNet50_2	0.01	0.0001	Linear	0.2448	0.4000	1.7800	1.0400	66846534
Wide_ResNet50_2	0.01	0.0001	Sequential	0.5523	0.4000	1.5300	0.0003	69622214
Wide_ResNet50_2	0.001	Default	Sequential	0.4300	0.4400	1.2193	0.8000	69622214
Wide_ResNet50_2	0.001	Default	Linear	1.0000	0.4400	0.0047	0.0018	66846534
Wide_ResNet50_2	0.001	0.0001	Linear	0.2700	0.4400	1.7000	0.9900	66846534
Wide_ResNet50_2	0.001	0.0001	Sequential	0.2300	0.3200	1.7700	1.4000	69622214
Wide_ResNet50_2	0.0001	Default	Sequential	0.2727	0.4000	1.7400	1.4100	69622214
Wide_ResNet50_2	0.0001	Default	Linear	1.0000	0.4000	0.0001	0.0001	66846534
Wide_ResNet50_2	0.0001	0.0001	Linear	1.0000	0.6400	0.0020	0.0010	66846534
Wide_ResNet50_2	0.0001	0.0001	Sequential	0.4965	0.4400	1.0859	0.6900	69622214
Wide_ResNet101_2	0.01	Default	Sequential	0.2867	0.5200	1.7700	0.0006	127625670
Wide_ResNet101_2	0.01	Default	Linear	0.2300	0.3200	1.7700	1.4200	124849990
Wide_ResNet101_2	0.01	0.0001	Linear	0.2400	0.4000	1.7800	0.9100	124849990
Wide_ResNet101_2	0.01	0.0001	Sequential	0.4000	0.4400	1.7400	0.0001	127625670
Wide_ResNet101_2	0.001	Default	Sequential	0.8000	0.5600	0.4591	0.2329	127625670
Wide_ResNet101_2	0.001	Default	Linear	1.0000	0.8800	0.0040	0.0001	124849990
Wide_ResNet101_2	0.001	0.0001	Linear	1.0000	0.8800	0.0100	0.0003	124849990
Wide_ResNet101_2	0.001	0.0001	Sequential	0.8112	0.7600	0.4239	0.0990	127625670
Wide_ResNet101_2	0.0001	Default	Sequential	1.0000	0.9600	0.0020	0.0001	127625670
Wide_ResNet101_2	0.0001	Default	Linear	1.0000	0.9600	0.0008	0.0002	124849990
Wide_ResNet101_2	0.0001	0.0001	Linear	1.0000	0.9600	0.0005	0.0001	124849990
Wide_ResNet101_2	0.0001	0.0001	Sequential	1.0000	0.9600	0.0010	0.0001	127625670
EfficientNet_v2_s	0.01	Default	Sequential	0.3200	0.2800	1.7800	0.6600	20998358

EfficientNet_v2_s	0.01	Default	Linear	0.2800	0.4000	1.7900	0.1300	20185174
EfficientNet_v2_s	0.01	0.0001	Linear	0.2657	0.3200	1.8300	0.6483	20185174
EfficientNet_v2_s	0.01	0.0001	Sequential	0.2657	0.3200	1.7600	0.8500	20998358
EfficientNet_v2_s	0.001	Default	Sequential	0.3000	0.4000	1.6727	1.1000	20998358
EfficientNet_v2_s	0.001	Default	Linear	0.3427	0.3776	1.6300	1.1700	20185174
EfficientNet_v2_s	0.001	0.0001	Linear	0.4196	0.5200	1.4244	0.6773	20185174
EfficientNet_v2_s	0.001	0.0001	Sequential	0.3636	0.4400	1.4700	1.0700	20998358
EfficientNet_v2_s	0.0001	Default	Sequential	0.4755	0.6014	1.3700	0.9400	20998358
EfficientNet_v2_s	0.0001	Default	Linear	0.4895	0.4800	1.3400	0.8306	20185174
EfficientNet_v2_s	0.0001	0.0001	Linear	0.5035	0.6000	1.3300	0.7800	20185174
EfficientNet_v2_s	0.0001	0.0001	Sequential	0.5105	0.6500	1.3300	0.7300	20998358
EfficientNet_v2_m	0.01	Default	Sequential	0.2657	0.2800	1.7800	1.0070	53679226
EfficientNet_v2_m	0.01	Default	Linear	0.2867	0.2800	1.8578	0.5038	52866042
EfficientNet_v2_m	0.01	0.0001	Linear	0.2500	0.2587	1.9200	0.4943	52866042
EfficientNet_v2_m	0.01	0.0001	Sequential	0.3000	0.4000	1.7800	0.4636	53679226
EfficientNet_v2_m	0.001	Default	Sequential	0.2657	0.3600	1.7500	1.3416	53679226
EfficientNet_v2_m	0.001	Default	Linear	0.3007	0.4000	1.7300	0.9500	52866042
EfficientNet_v2_m	0.001	0.0001	Linear	0.3217	0.4196	1.6600	0.8440	52866042
EfficientNet_v2_m	0.001	0.0001	Sequential	0.3287	0.4000	1.6066	1.2298	53679226
EfficientNet_v2_m	0.0001	Default	Sequential	0.3845	0.3600	1.5400	1.1300	53679226
EfficientNet_v2_m	0.0001	Default	Linear	0.3566	0.3600	1.6100	0.8585	52866042
EfficientNet_v2_m	0.0001	0.0001	Linear	0.3706	0.4895	1.5900	0.9800	52866042
EfficientNet_v2_m	0.0001	0.0001	Sequential	0.3600	0.4266	1.4900	1.0770	53679226
EfficientNet_v2_l	0.01	Default	Sequential	0.2517	0.3600	1.7500	0.0058	118055142
EfficientNet_v2_l	0.01	Default	Linear	0.2936	0.3600	1.8900	0.7800	117241958
EfficientNet_v2_l	0.01	0.0001	Linear	0.2448	0.3600	1.9700	0.2489	117241958
EfficientNet_v2_l	0.01	0.0001	Sequential	0.2936	0.2517	1.7700	0.7549	118055142
EfficientNet_v2_l	0.001	Default	Sequential	0.3070	0.3200	1.7500	1.0700	118055142

EfficientNet_v2_l	0.001	Default	Linear	0.3077	0.2727	1.7271	0.8469	117241958
EfficientNet_v2_l	0.001	0.0001	Linear	0.3497	0.3217	1.7109	1.0430	117241958
EfficientNet_v2_l	0.001	0.0001	Sequential	0.2937	0.3000	1.7300	1.4500	118055142
EfficientNet_v2_l	0.0001	Default	Sequential	0.4476	0.4800	1.4600	1.0100	118055142
EfficientNet_v2_l	0.0001	Default	Linear	0.3986	0.4800	1.5200	0.8071	117241958
EfficientNet_v2_l	0.0001	0.0001	Linear	0.4200	0.4800	1.5200	0.8544	117241958
EfficientNet_v2_l	0.0001	0.0001	Sequential	0.4300	0.4000	1.4924	1.1400	118055142
Swin_t	0.01	Default	Sequential	0.2726	0.2800	1.7700	0.8490	27749888
Swin_t	0.01	Default	Linear	0.2517	0.2800	1.7700	1.2801	27523968
Swin_t	0.01	0.0001	Linear	0.2517	0.3200	1.7700	1.1400	27523968
Swin_t	0.01	0.0001	Sequential	0.2657	0.2800	1.7631	1.3790	27749888
Swin_t	0.001	Default	Sequential	0.2517	0.2800	1.7622	1.1100	27749888
Swin_t	0.001	Default	Linear	0.2517	0.2800	1.7600	1.3072	27523968
Swin_t	0.001	0.0001	Linear	0.2448	0.3200	1.7400	1.2900	27523968
Swin_t	0.001	0.0001	Sequential	0.2657	0.3200	1.6500	1.2000	27749888
Swin_t	0.0001	Default	Sequential	1.0000	0.6800	0.0001	0.0001	27749888
Swin_t	0.0001	Default	Linear	1.0000	0.6000	0.0001	0.0001	27523968
Swin_t	0.0001	0.0001	Linear	1.0000	0.6400	0.0003	0.0001	27523968
Swin_t	0.0001	0.0001	Sequential	1.0000	0.6000	0.0001	0.0001	27749888
Swin_s	0.01	Default	Sequential	0.2517	0.2800	1.7700	1.4600	49067792
Swin_s	0.01	Default	Linear	0.2657	0.2800	1.7700	1.2600	48841872
Swin_s	0.01	0.0001	Linear	0.3100	0.2800	1.7700	1.3300	48841872
Swin_s	0.01	0.0001	Sequential	0.2797	0.3200	1.7500	1.4000	49067792
Swin_s	0.001	Default	Sequential	0.2500	0.2800	1.7600	1.5680	49067792
Swin_s	0.001	Default	Linear	0.2500	0.3200	1.7400	1.2800	48841872
Swin_s	0.001	0.0001	Linear	0.2517	0.3200	1.7200	1.4400	48841872
Swin_s	0.001	0.0001	Sequential	0.2727	0.3200	1.7400	1.5000	49067792
Swin_s	0.0001	Default	Sequential	1.0000	0.7200	0.0002	0.0001	49067792

Swin_s	0.0001	Default	Linear	1.0000	0.6800	0.0001	0.0001	48841872
Swin_s	0.0001	0.0001	Linear	1.0000	0.6400	0.0001	0.0001	48841872
Swin_s	0.0001	0.0001	Sequential	1.0000	0.5600	0.0002	0.0001	49067792
Swin_b	0.01	Default	Sequential	0.2600	0.2800	1.7700	0.9700	87433022
Swin_b	0.01	Default	Linear	0.2500	0.2800	1.3100	1.7800	86749374
Swin_b	0.01	0.0001	Linear	0.2657	0.2378	1.7800	1.3400	86749374
Swin_b	0.01	0.0001	Sequential	0.2500	0.2800	1.7700	1.1000	87433022
Swin_b	0.001	Default	Sequential	0.2797	0.2800	1.7700	1.4300	87433022
Swin_b	0.001	Default	Linear	0.2500	0.2800	1.7600	1.5300	86749374
Swin_b	0.001	0.0001	Linear	0.2500	0.2657	1.7279	1.5400	86749374
Swin_b	0.001	0.0001	Sequential	0.2098	0.2800	1.7700	1.5825	87433022
Swin_b	0.0001	Default	Sequential	1.0000	0.7600	0.0002	0.0001	87433022
Swin_b	0.0001	Default	Linear	1.0000	0.6800	0.0001	0.0001	86749374
Swin_b	0.0001	0.0001	Linear	1.0000	0.6000	0.0001	0.0001	86749374
Swin_b	0.0001	0.0001	Sequential	1.0000	0.6400	0.0002	0.0001	87433022
Swin_v2_t	0.01	Default	Sequential	0.2657	0.2098	1.7600	1.3600	27813104
Swin_v2_t	0.01	Default	Linear	0.2517	0.2800	1.7700	1.3900	27587184
Swin_v2_t	0.01	0.0001	Linear	0.2700	0.2800	1.7700	1.3872	27587184
Swin_v2_t	0.01	0.0001	Sequential	0.2400	0.2098	1.7700	1.4500	27813104
Swin_v2_t	0.001	Default	Sequential	0.2378	0.2800	1.7700	1.5700	27813104
Swin_v2_t	0.001	Default	Linear	0.2937	0.3200	1.6900	1.3200	27587184
Swin_v2_t	0.001	0.0001	Linear	0.2600	0.3600	1.7100	1.1400	27587184
Swin_v2_t	0.001	0.0001	Sequential	0.3200	0.3600	1.6789	1.4200	27813104
Swin_v2_t	0.0001	Default	Sequential	1.0000	0.9930	0.0004	0.0001	27813104
Swin_v2_t	0.0001	Default	Linear	1.0000	0.6000	0.0001	0.0001	27587184
Swin_v2_t	0.0001	0.0001	Linear	1.0000	0.9930	0.0001	0.0001	27587184
Swin_v2_t	0.0001	0.0001	Sequential	1.0000	0.993	0.0001	0.0001	27813104
Swin_v2_s	0.01	Default	Sequential	0.2700	0.2800	1.7600	1.0300	49198976

Swin_v2_s	0.01	Default	Linear	0.2581	0.2800	1.7600	1.2986	48973056
Swin_v2_s	0.01	0.0001	Linear	0.2600	0.2800	1.7700	1.1500	48973056
Swin_v2_s	0.01	0.0001	Sequential	0.2657	0.2800	1.7721	1.5280	49198976
Swin_v2_s	0.001	Default	Sequential	0.2587	0.2800	1.7623	1.5799	49198976
Swin_v2_s	0.001	Default	Linear	0.2587	0.2800	1.7672	1.5440	48973056
Swin_v2_s	0.001	0.0001	Linear	0.2937	0.3600	1.7219	1.4215	48973056
Swin_v2_s	0.001	0.0001	Sequential	0.3077	0.3600	1.6332	1.0733	49198976
Swin_v2_s	0.0001	Default	Sequential	1.0000	0.5600	0.0001	0.0001	49198976
Swin_v2_s	0.0001	Default	Linear	1.0000	0.6400	0.0001	0.0001	48973056
Swin_v2_s	0.0001	0.0001	Linear	1.0000	0.6400	0.0001	0.0001	48973056
Swin_v2_s	0.0001	0.0001	Sequential	1.0000	0.6000	0.0004	0.0001	49198976
Swin_v2_b	0.01	Default	Sequential	0.2517	0.2800	1.7700	0.0007	87930848
Swin_v2_b	0.01	Default	Linear	0.2378	0.2800	1.7700	1.2200	86911998
Swin_v2_b	0.01	0.0001	Linear	0.2308	0.2800	1.7800	1.4752	86911998
Swin_v2_b	0.01	0.0001	Sequential	0.2500	0.2800	1.7600	0.7777	87930848
Swin_v2_b	0.001	Default	Sequential	0.2517	0.2800	1.7707	1.5800	87930848
Swin_v2_b	0.001	Default	Linear	0.2378	0.2800	1.7696	1.5800	86911998
Swin_v2_b	0.001	0.0001	Linear	0.3077	0.3200	1.7100	1.3486	86911998
Swin_v2_b	0.001	0.0001	Sequential	0.2378	0.3200	1.7675	1.4918	87930848
Swin_v2_b	0.0001	Default	Sequential	1.0000	0.7200	0.0001	0.0001	87930848
Swin_v2_b	0.0001	Default	Linear	1.0000	0.6000	0.0001	0.0001	86911998
Swin_v2_b	0.0001	0.0001	Linear	1.0000	0.6400	0.0001	0.0001	86911998
Swin_v2_b	0.0001	0.0001	Sequential	1.0000	0.6400	0.0002	0.0001	87930848
DenseNet201	0.01	Default	Sequential	0.8322	0.7200	0.4054	0.0370	19241478
DenseNet201	0.01	Default	Linear	1.0000	0.9200	0.0001	0.0042	18104454
DenseNet201	0.01	0.0001	Linear	0.9720	0.9200	0.1095	0.0010	18104454
DenseNet201	0.01	0.0001	Sequential	0.6503	0.8000	0.8053	0.1263	19241478
DenseNet201	0.001	Default	Sequential	1.0000	0.9600	0.0052	0.0001	19241478

DenseNet201	0.001	Default	Linear	1.0000	0.9600	0.0022	0.0001	18104454
DenseNet201	0.001	0.0001	Linear	1.0000	0.9600	0.0013	0.0001	18104454
DenseNet201	0.001	0.0001	Sequential	0.9930	0.9600	0.0292	0.0001	19241478
DenseNet201	0.0001	Default	Sequential	1.0000	0.9600	0.0003	0.0001	19241478
DenseNet201	0.0001	Default	Linear	1.0000	0.9600	0.0011	0.0001	18104454
DenseNet201	0.0001	0.0001	Linear	1.0000	0.9600	0.0015	0.0002	18104454
DenseNet201	0.0001	0.0001	Sequential	1.0000	0.9600	0.0004	0.0001	19241478
DenseNet161	0.01	Default	Sequential	0.5944	0.7200	0.2034	0.9694	29423814
DenseNet161	0.01	Default	Linear	1.0000	0.9600	0.0105	0.0001	26485254
DenseNet161	0.01	0.0001	Linear	0.0965	0.9200	0.1318	0.0006	26485254
DenseNet161	0.01	0.0001	Sequential	0.6154	0.8000	0.9971	0.1674	29423814
DenseNet161	0.001	Default	Sequential	1.0000	0.9600	0.0296	0.0001	29423814
DenseNet161	0.001	Default	Linear	1.0000	0.9600	0.0016	0.0001	26485254
DenseNet161	0.001	0.0001	Linear	1.0000	0.9600	0.0026	0.0001	26485254
DenseNet161	0.001	0.0001	Sequential	0.9860	0.8800	0.0559	0.0019	29423814
DenseNet161	0.0001	Default	Sequential	1.0000	0.9600	0.0002	0.0001	29423814
DenseNet161	0.0001	Default	Linear	1.0000	0.9600	0.0010	0.0002	26485254
DenseNet161	0.0001	0.0001	Linear	1.0000	0.9600	0.0010	0.0002	26485254
DenseNet161	0.0001	0.0001	Sequential	0.7832	0.8000	0.6153	0.0382	29423814
DenseNet169	0.01	Default	Sequential	0.9720	0.9930	0.1187	0.0001	15436294
DenseNet169	0.01	Default	Linear	1.0000	0.9930	0.0003	0.0210	12497734
DenseNet169	0.01	0.0001	Linear	0.9860	0.9600	0.0641	0.0035	12497734
DenseNet169	0.01	0.0001	Sequential	0.8112	0.9161	0.8530	0.4897	15436294
DenseNet169	0.001	Default	Sequential	1.0000	0.9200	0.0105	0.0001	15436294
DenseNet169	0.001	Default	Linear	1.0000	0.9600	0.0003	0.0001	12497734
DenseNet169	0.001	0.0001	Linear	1.0000	0.9930	0.0039	0.0001	12497734
DenseNet169	0.001	0.0001	Sequential	1.0000	0.9600	0.0128	0.0002	15436294
DenseNet169	0.0001	Default	Sequential	1.0000	0.9930	0.0005	0.0001	15436294

DenseNet169	0.0001	Default	Linear	1.0000	0.9600	0.0018	0.0001	12497734
DenseNet169	0.0001	0.0001	Linear	1.0000	0.9600	0.0011	0.0002	12497734
DenseNet169	0.0001	0.0001	Sequential	1.0000	0.9600	0.0002	0.0001	15436294
DenseNet121	0.01	Default	Sequential	0.9231	0.8400	0.2907	0.0138	7643654
DenseNet121	0.01	Default	Linear	1.0000	0.9930	0.0056	0.0001	6960006
DenseNet121	0.01	0.0001	Linear	0.9860	0.9200	0.0741	0.0014	6960006
DenseNet121	0.01	0.0001	Sequential	0.7344	0.7600	0.5947	0.0653	7643654
DenseNet121	0.001	Default	Sequential	1.0000	0.9600	0.0011	0.0000	7643654
DenseNet121	0.001	Default	Linear	1.0000	0.9930	0.0006	0.0001	6960006
DenseNet121	0.001	0.0001	Linear	1.0000	0.9600	0.0008	0.0001	6960006
DenseNet121	0.001	0.0001	Sequential	1.0000	0.9600	0.0163	0.0001	7643654
DenseNet121	0.0001	Default	Sequential	1.0000	0.9600	0.0004	0.0001	7643654
DenseNet121	0.0001	Default	Linear	1.0000	0.9600	0.0017	0.0002	6960006
DenseNet121	0.0001	0.0001	Linear	1.0000	0.9600	0.0014	0.0002	6960006
DenseNet121	0.0001	0.0001	Sequential	1.0000	0.9930	0.0003	0.0001	7643654
MNASNet0_5	0.01	Default	Sequential	0.6643	0.3600	0.9090	0.5818	2939054
MNASNet0_5	0.01	Default	Linear	1.0000	0.8000	0.0001	0.0001	945198
MNASNet0_5	0.01	0.0001	Linear	1.0000	0.8800	0.0001	0.0001	945198
MNASNet0_5	0.01	0.0001	Sequential	0.5874	0.2800	1.0712	0.4014	2939054
MNASNet0_5	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	2939054
MNASNet0_5	0.001	Default	Linear	1.0000	0.9600	0.0001	0.0001	945198
MNASNet0_5	0.001	0.0001	Linear	1.0000	0.3600	0.0015	0.0001	945198
MNASNet0_5	0.001	0.0001	Sequential	1.0000	0.6000	0.0001	0.0001	2939054
MNASNet0_5	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	2939054
MNASNet0_5	0.0001	Default	Linear	1.0000	0.9600	0.0153	0.0004	945198
MNASNet0_5	0.0001	0.0001	Linear	1.0000	0.6000	0.0074	0.0006	945198
MNASNet0_5	0.0001	0.0001	Sequential	1.0000	0.8700	0.0001	0.0001	2939054
MNASNet0_75	0.01	Default	Sequential	0.2587	0.5200	1.7737	1.2804	3890750

MNASNet0_75	0.01	Default	Linear	1.0000	0.2098	0.0010	0.0001	1896894
MNASNet0_75	0.01	0.0001	Linear	1.0000	0.5600	0.0237	0.0007	1896894
MNASNet0_75	0.01	0.0001	Sequential	0.4336	0.3600	1.4730	0.5863	3890750
MNASNet0_75	0.001	Default	Sequential	1.0000	0.4800	0.0001	0.0001	3890750
MNASNet0_75	0.001	Default	Linear	1.0000	0.6800	0.0001	0.0001	1896894
MNASNet0_75	0.001	0.0001	Linear	1.0000	0.7600	0.0001	0.0001	1896894
MNASNet0_75	0.001	0.0001	Sequential	0.5105	0.2800	0.1632	0.5757	3890750
MNASNet0_75	0.0001	Default	Sequential	1.0000	0.9930	0.0001	0.0001	3890750
MNASNet0_75	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	1896894
MNASNet0_75	0.0001	0.0001	Linear	1.0000	0.9600	0.0003	0.0001	1896894
MNASNet0_75	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	3890750
MNASNet1_0	0.01	Default	Sequential	0.7483	0.3077	0.6015	0.0156	5103854
MNASNet1_0	0.01	Default	Linear	1.0000	0.3600	0.0029	0.0001	3890750
MNASNet1_0	0.01	0.0001	Linear	0.9720	0.2800	0.0747	0.0012	3890750
MNASNet1_0	0.01	0.0001	Sequential	0.4895	0.3636	1.1581	0.7238	5103854
MNASNet1_0	0.001	Default	Sequential	1.0000	0.6000	0.0001	0.0001	5103854
MNASNet1_0	0.001	Default	Linear	1.0000	0.9860	0.0001	0.0001	3890750
MNASNet1_0	0.001	0.0001	Linear	1.0000	0.8400	0.0002	0.0001	3890750
MNASNet1_0	0.001	0.0001	Sequential	1.0000	0.8800	0.0003	0.0001	5103854
MNASNet1_0	0.0001	Default	Sequential	1.0000	0.9600	0.0012	0.0001	5103854
MNASNet1_0	0.0001	Default	Linear	1.0000	0.9600	0.0026	0.0003	3890750
MNASNet1_0	0.0001	0.0001	Linear	1.0000	0.9930	0.0024	0.0002	3890750
MNASNet1_0	0.0001	0.0001	Sequential	1.0000	0.9600	0.0003	0.0000	5103854
MNASNet1_3	0.01	Default	Sequential	0.3846	0.2800	1.5626	1.0990	7002798
MNASNet1_3	0.01	Default	Linear	1.0000	0.3600	0.0028	0.0001	5008942
MNASNet1_3	0.01	0.0001	Linear	0.9930	0.5200	0.0525	0.0019	5008942
MNASNet1_3	0.01	0.0001	Sequential	0.4900	0.2800	1.1396	0.6808	7002798
MNASNet1_3	0.001	Default	Sequential	1.0000	0.4800	0.0001	0.0001	7002798

MNASNet1_3	0.001	Default	Linear	1.0000	0.2800	0.0001	0.0017	5008942
MNASNet1_3	0.001	0.0001	Linear	1.0000	0.5200	0.0002	0.0001	5008942
MNASNet1_3	0.001	0.0001	Sequential	1.0000	0.5455	0.0008	0.0001	7002798
MNASNet1_3	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	7002798
MNASNet1_3	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	5008942
MNASNet1_3	0.0001	0.0001	Linear	1.0000	0.9600	0.0003	0.0001	5008942
MNASNet1_3	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	7002798
ShuffleNet_v2_x0_5	0.01	Default	Sequential	1.0000	0.9600	0.0012	0.0001	2056590
ShuffleNet_v2_x0_5	0.01	Default	Linear	1.0000	0.9600	0.0012	0.0001	1372942
ShuffleNet_v2_x0_5	0.01	0.0001	Linear	1.0000	0.9600	0.0054	0.0003	1372942
ShuffleNet_v2_x0_5	0.01	0.0001	Sequential	1.0000	0.9600	0.0062	0.0005	2056590
ShuffleNet_v2_x0_5	0.001	Default	Sequential	1.0000	0.9930	0.0001	0.0001	2056590
ShuffleNet_v2_x0_5	0.001	Default	Linear	1.0000	0.9600	0.0001	0.0001	1372942
ShuffleNet_v2_x0_5	0.001	0.0001	Linear	1.0000	0.9600	0.0003	0.0001	1372942
ShuffleNet_v2_x0_5	0.001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	2056590
ShuffleNet_v2_x0_5	0.0001	Default	Sequential	1.0000	0.9600	0.0006	0.0001	2056590
ShuffleNet_v2_x0_5	0.0001	Default	Linear	1.0000	0.9930	0.0003	0.0001	1372942
ShuffleNet_v2_x0_5	0.0001	0.0001	Linear	1.0000	0.9600	0.0005	0.0001	1372942
ShuffleNet_v2_x0_5	0.0001	0.0001	Sequential	1.0000	0.9860	0.0005	0.0001	2056590
ShuffleNet_v2_x1_0	0.01	Default	Sequential	1.0000	0.9600	0.0057	0.0001	2968402
ShuffleNet_v2_x1_0	0.01	Default	Linear	1.0000	0.9600	0.0067	0.0001	2284754
ShuffleNet_v2_x1_0	0.01	0.0001	Linear	1.0000	0.9600	0.0053	0.0005	2284754
ShuffleNet_v2_x1_0	0.01	0.0001	Sequential	1.0000	0.9600	0.0230	0.0010	2968402
ShuffleNet_v2_x1_0	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	2968402
ShuffleNet_v2_x1_0	0.001	Default	Linear	1.0000	0.9930	0.0001	0.0001	2284754
ShuffleNet_v2_x1_0	0.001	0.0001	Linear	1.0000	0.9600	0.0013	0.0001	2284754
ShuffleNet_v2_x1_0	0.001	0.0001	Sequential	1.0000	0.9930	0.0013	0.0001	2968402
ShuffleNet_v2_x1_0	0.0001	Default	Sequential	1.0000	0.9600	0.0002	0.0001	2968402

ShuffleNet_v2_x1_0	0.0001	Default	Linear	1.0000	0.9600	0.0006	0.0001	2284754
ShuffleNet_v2_x1_0	0.0001	0.0001	Linear	1.0000	0.9600	0.0004	0.0001	2284754
ShuffleNet_v2_x1_0	0.0001	0.0001	Sequential	1.0000	0.9600	0.0002	0.0001	2968402
ShuffleNet_v2_x1_5	0.01	Default	Sequential	1.0000	0.9600	0.0005	0.0001	4193422
ShuffleNet_v2_x1_5	0.01	Default	Linear	1.0000	0.9930	0.0002	0.0001	3509774
ShuffleNet_v2_x1_5	0.01	0.0001	Linear	1.0000	0.8800	0.0066	0.0001	3509774
ShuffleNet_v2_x1_5	0.01	0.0001	Sequential	1.0000	0.9200	0.0100	0.0001	4193422
ShuffleNet_v2_x1_5	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	4193422
ShuffleNet_v2_x1_5	0.001	Default	Linear	1.0000	0.9600	0.0001	0.0001	3509774
ShuffleNet_v2_x1_5	0.001	0.0001	Linear	1.0000	0.9930	0.0001	0.0001	3509774
ShuffleNet_v2_x1_5	0.001	0.0001	Sequential	1.0000	0.9600	0.0002	0.0001	4193422
ShuffleNet_v2_x1_5	0.0001	Default	Sequential	1.0000	0.9600	0.0005	0.0001	4193422
ShuffleNet_v2_x1_5	0.0001	Default	Linear	1.0000	0.9200	0.0004	0.0001	3509774
ShuffleNet_v2_x1_5	0.0001	0.0001	Linear	1.0000	0.9650	0.0008	0.0001	3509774
ShuffleNet_v2_x1_5	0.0001	0.0001	Sequential	1.0000	0.9600	0.0006	0.0001	4193422
ShuffleNet_v2_x2_0	0.01	Default	Sequential	1.0000	0.9200	0.0002	0.0001	10181970
ShuffleNet_v2_x2_0	0.01	Default	Linear	1.0000	0.9200	0.0001	0.0001	7406290
ShuffleNet_v2_x2_0	0.01	0.0001	Linear	1.0000	0.8400	0.0036	0.0001	7406290
ShuffleNet_v2_x2_0	0.01	0.0001	Sequential	1.0000	0.8000	0.0223	0.0001	10181970
ShuffleNet_v2_x2_0	0.001	0.0001	Linear	1.0000	0.9600	0.0003	0.0001	7406290
ShuffleNet_v2_x2_0	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	10181970
ShuffleNet_v2_x2_0	0.001	Default	Linear	1.0000	0.9600	0.0001	0.0001	7406290
ShuffleNet_v2_x2_0	0.001	0.0001	Sequential	1.0000	0.9600	0.0003	0.0001	10181970
ShuffleNet_v2_x2_0	0.0001	Default	Sequential	1.0000	0.9510	0.0006	0.0001	10181970
ShuffleNet_v2_x2_0	0.0001	Default	Linear	1.0000	0.9200	0.0001	0.0001	7406290
ShuffleNet_v2_x2_0	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	7406290
ShuffleNet_v2_x2_0	0.0001	0.0001	Sequential	1.0000	0.9200	0.0002	0.0001	10181970
EfficientNet_b0	0.01	Default	Sequential	0.4755	0.6000	1.3192	0.7840	4828418

EfficientNet_b0	0.01	Default	Linear	1.0000	0.8800	0.0003	0.0001	4015234
EfficientNet_b0	0.01	0.0001	Linear	1.0000	0.8400	0.0214	0.0001	4015234
EfficientNet_b0	0.01	0.0001	Sequential	0.4400	0.6000	1.3521	0.6292	4828418
EfficientNet_b0	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	4828418
EfficientNet_b0	0.001	Default	Linear	1.0000	0.9930	0.0001	0.0001	4015234
EfficientNet_b0	0.001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	4015234
EfficientNet_b0	0.001	0.0001	Sequential	1.0000	0.9600	0.0003	0.0001	4828418
EfficientNet_b0	0.0001	Default	Sequential	1.0000	0.9930	0.0002	0.0001	4828418
EfficientNet_b0	0.0001	Default	Linear	1.0000	0.9600	0.0004	0.0001	4015234
EfficientNet_b0	0.0001	0.0001	Linear	1.0000	0.9600	0.0011	0.0001	4015234
EfficientNet_b0	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	4828418
EfficientNet_b1	0.01	Default	Sequential	0.4196	0.6000	1.5442	1.0244	7334054
EfficientNet_b1	0.01	Default	Linear	1.0000	0.9930	0.0003	0.0001	6520870
EfficientNet_b1	0.01	0.0001	Linear	1.0000	0.7200	0.0110	0.0001	6520870
EfficientNet_b1	0.01	0.0001	Sequential	0.4965	0.6400	0.9203	0.3723	7334054
EfficientNet_b1	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	7334054
EfficientNet_b1	0.001	Default	Linear	1.0000	0.9300	0.0001	0.0001	6520870
EfficientNet_b1	0.001	0.0001	Linear	1.0000	0.9600	0.0002	0.0001	6520870
EfficientNet_b1	0.001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	7334054
EfficientNet_b1	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	7334054
EfficientNet_b1	0.0001	Default	Linear	1.0000	0.9930	0.0002	0.0001	6520870
EfficientNet_b1	0.0001	0.0001	Linear	1.0000	0.9600	0.0006	0.0001	6520870
EfficientNet_b1	0.0001	0.0001	Sequential	1.0000	0.9600	0.0002	0.0001	7334054
EfficientNet_b2	0.01	Default	Sequential	0.3916	0.5600	1.3611	0.6248	8587400
EfficientNet_b2	0.01	Default	Linear	1.0000	0.7600	0.0068	0.0001	7709448
EfficientNet_b2	0.01	0.0001	Linear	0.9790	0.6400	0.2587	0.0002	7709448
EfficientNet_b2	0.01	0.0001	Sequential	0.4895	0.6643	1.3200	0.5658	8587400
EfficientNet_b2	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	8587400

EfficientNet_b2	0.001	Default	Linear	1.0000	0.9600	0.0001	0.0001	7709448
EfficientNet_b2	0.001	0.0001	Linear	1.0000	0.9600	0.0005	0.0001	7709448
EfficientNet_b2	0.001	0.0001	Sequential	1.0000	0.9930	0.0001	0.0001	8587400
EfficientNet_b2	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	8587400
EfficientNet_b2	0.0001	Default	Linear	1.0000	0.9600	0.0005	0.0001	7709448
EfficientNet_b2	0.0001	0.0001	Linear	1.0000	0.9600	0.0006	0.0001	7709448
EfficientNet_b2	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	8587400
EfficientNet_b3	0.01	Default	Sequential	0.5455	0.7200	0.9756	0.4941	12959918
EfficientNet_b3	0.01	Default	Linear	1.0000	0.7200	0.0068	0.0001	10705454
EfficientNet_b3	0.01	0.0001	Linear	1.0000	0.7200	0.0775	0.0001	10705454
EfficientNet_b3	0.01	0.0001	Sequential	0.6154	0.7600	0.8703	0.0855	12959918
EfficientNet_b3	0.001	Default	Sequential	1.0000	0.9930	0.0004	0.0001	12959918
EfficientNet_b3	0.001	Default	Linear	1.0000	0.9930	0.0001	0.0001	10705454
EfficientNet_b3	0.001	0.0001	Linear	1.0000	0.9600	0.0002	0.0001	10705454
EfficientNet_b3	0.001	0.0001	Sequential	1.0000	0.9930	0.0001	0.0001	12959918
EfficientNet_b3	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	12959918
EfficientNet_b3	0.0001	Default	Linear	1.0000	0.9600	0.0003	0.0001	10705454
EfficientNet_b3	0.0001	0.0001	Linear	1.0000	0.9600	0.0004	0.0001	10705454
EfficientNet_b3	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	12959918
EfficientNet_b4	0.01	Default	Sequential	0.7832	0.9600	0.6634	0.0884	20074446
EfficientNet_b4	0.01	Default	Linear	1.0000	0.9200	0.0015	0.0001	17559374
EfficientNet_b4	0.01	0.0001	Linear	0.9860	0.9600	0.0432	0.0001	17559374
EfficientNet_b4	0.01	0.0001	Sequential	0.8671	0.8800	0.3501	0.0107	20074446
EfficientNet_b4	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	20074446
EfficientNet_b4	0.001	Default	Linear	1.0000	0.9600	0.0001	0.0001	17559374
EfficientNet_b4	0.001	0.0001	Linear	1.0000	0.9930	0.0001	0.0001	17559374
EfficientNet_b4	0.001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	20074446
EfficientNet_b4	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	20074446

EfficientNet_b4	0.0001	Default	Linear	1.0000	0.9930	0.0006	0.0001	17559374
EfficientNet_b4	0.0001	0.0001	Linear	1.0000	0.9600	0.0003	0.0001	17559374
EfficientNet_b4	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	20074446
EfficientNet_b5	0.01	Default	Sequential	0.5734	0.6800	1.0614	0.4009	31128758
EfficientNet_b5	0.01	Default	Linear	1.0000	0.9930	0.0307	0.0001	28353078
EfficientNet_b5	0.01	0.0001	Linear	0.9650	0.6800	0.1200	0.0003	28353078
EfficientNet_b5	0.01	0.0001	Sequential	0.6923	0.5944	1.0370	0.1000	31128758
EfficientNet_b5	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	31128758
EfficientNet_b5	0.001	Default	Linear	1.0000	0.9600	0.0001	0.0001	28353078
EfficientNet_b5	0.001	0.0001	Linear	1.0000	0.9600	0.0006	0.0001	28353078
EfficientNet_b5	0.001	0.0001	Sequential	1.0000	0.9600	0.0004	0.0001	31128758
EfficientNet_b5	0.0001	Default	Sequential	1.0000	0.9930	0.0002	0.0001	31128758
EfficientNet_b5	0.0001	Default	Linear	1.0000	0.9600	0.0002	0.0001	28353078
EfficientNet_b5	0.0001	0.0001	Linear	1.0000	0.9600	0.0005	0.0001	28353078
EfficientNet_b5	0.0001	0.0001	Sequential	1.0000	0.9930	0.0001	0.0001	31128758
EfficientNet_b6	0.01	Default	Sequential	0.6643	0.6800	0.9485	0.0001	43785822
EfficientNet_b6	0.01	Default	Linear	0.9790	0.7600	0.0576	0.0001	40749534
EfficientNet_b6	0.01	0.0001	Linear	0.9231	0.9930	0.2048	0.0001	40749534
EfficientNet_b6	0.01	0.0001	Sequential	0.5035	0.5600	1.2590	0.8046	43785822
EfficientNet_b6	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	43785822
EfficientNet_b6	0.001	Default	Linear	1.0000	0.9930	0.0001	0.0001	40749534
EfficientNet_b6	0.001	0.0001	Linear	1.0000	0.9600	0.0002	0.0001	40749534
EfficientNet_b6	0.001	0.0001	Sequential	1.0000	0.9930	0.0003	0.0001	43785822
EfficientNet_b6	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	43785822
EfficientNet_b6	0.0001	Default	Linear	1.0000	0.9600	0.0003	0.0001	40749534
EfficientNet_b6	0.0001	0.0001	Linear	1.0000	0.9930	0.0006	0.0001	40749534
EfficientNet_b6	0.0001	0.0001	Sequential	1.0000	0.9600	0.0003	0.0001	43785822
EfficientNet_b7	0.01	Default	Sequential	0.3566	0.5200	1.6070	0.0001	67099222

EfficientNet_b7	0.01	Default	Linear	0.9790	0.6800	0.0600	0.0001	63802326
EfficientNet_b7	0.01	0.0001	Linear	0.8600	0.7200	0.4739	0.0001	63802326
EfficientNet_b7	0.01	0.0001	Sequential	0.4266	0.6000	1.2250	0.7675	67099222
EfficientNet_b7	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	67099222
EfficientNet_b7	0.001	Default	Linear	1.0000	0.9600	0.0001	0.0001	63802326
EfficientNet_b7	0.001	0.0001	Linear	1.0000	0.9600	0.0007	0.0001	63802326
EfficientNet_b7	0.001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	67099222
EfficientNet_b7	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	67099222
EfficientNet_b7	0.0001	Default	Linear	1.0000	0.9600	0.0002	0.0001	63802326
EfficientNet_b7	0.0001	0.0001	Linear	1.0000	0.9600	0.0005	0.0001	63802326
EfficientNet_b7	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	67099222
SqueezeNet1_0	0.01	Default	Sequential	0.1608	0.2800	2,5150	2,0200	900422
SqueezeNet1_0	0.01	Default	Linear	0.1180	0.1600	1.7907	1.7917	738502
SqueezeNet1_0	0.01	0.0001	Linear	0.1958	0.1600	1.7914	1.7839	738502
SqueezeNet1_0	0.01	0.0001	Sequential	0.1748	0.2800	2,6085	2,3398	900422
SqueezeNet1_0	0.001	Default	Sequential	1.0000	0.2000	0.0001	0.0001	900422
SqueezeNet1_0	0.001	Default	Linear	1.0000	0.7200	0.0001	0.0001	738502
SqueezeNet1_0	0.001	0.0001	Linear	1.0000	0.7200	0.0003	0.0001	738502
SqueezeNet1_0	0.001	0.0001	Sequential	1.0000	0.2400	0.0001	0.0001	900422
SqueezeNet1_0	0.0001	Default	Sequential	1.0000	0.1600	0.0001	0.0001	900422
SqueezeNet1_0	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	738502
SqueezeNet1_0	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	738502
SqueezeNet1_0	0.0001	0.0001	Sequential	1.0000	0.2000	0.0001	0.0001	900422
SqueezeNet1_1	0.01	Default	Sequential	0.1818	0.1600	2,5100	2,1026	887494
SqueezeNet1_1	0.01	Default	Linear	0.1189	0.1600	1.7916	1.7918	725574
SqueezeNet1_1	0.01	0.0001	Linear	0.1329	0.1600	1.7918	1.7918	725574
SqueezeNet1_1	0.01	0.0001	Sequential	0.1678	0.2000	2,3938	2,2272	887494
SqueezeNet1_1	0.001	Default	Sequential	1.0000	0.2400	0.0001	0.0001	887494

SqueezeNet1_1	0.001	Default	Linear	1.0000	0.6000	0.0001	0.0001	725574
SqueezeNet1_1	0.001	0.0001	Linear	0.2517	0.3600	1.7619	1.5616	725574
SqueezeNet1_1	0.001	0.0001	Sequential	1.0000	0.3600	0.0001	0.0001	887494
SqueezeNet1_1	0.0001	Default	Sequential	1.0000	0.1200	0.0001	0.0001	887494
SqueezeNet1_1	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	725574
SqueezeNet1_1	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	725574
SqueezeNet1_1	0.0001	0.0001	Sequential	1.0000	0.2800	0.0001	0.0001	887494
VGG11	0.01	Default	Sequential	0.2448	0.2800	47,5300	0.0950	139944966
VGG11	0.01	Default	Linear	0.2867	0.2800	1.7664	1.4900	139950470
VGG11	0.01	0.0001	Linear	0.2517	0.2800	1.7665	1.2474	139950470
VGG11	0.01	0.0001	Sequential	0.2657	0.2800	54502,6600	1.0000	139944966
VGG11	0.001	Default	Sequential	0.2378	0.2800	1.7708	1.4512	139944966
VGG11	0.001	Default	Linear	0.2448	0.2800	1.7665	1.4555	139950470
VGG11	0.001	0.0001	Linear	0.2587	0.2800	1.7663	1.5232	139950470
VGG11	0.001	0.0001	Sequential	0.2517	0.3200	1.7623	0.0140	139944966
VGG11	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	139944966
VGG11	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	139950470
VGG11	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	139950470
VGG11	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	139944966
VGG11_bn	0.01	Default	Sequential	0.2587	0.3200	0.9100	0.0002	139950470
VGG11_bn	0.01	Default	Linear	0.2657	0.2800	1.7716	1.2532	128796422
VGG11_bn	0.01	0.0001	Linear	0.2797	0.3600	1.7707	1.0370	128796422
VGG11_bn	0.01	0.0001	Sequential	0.2517	0.2800	2,1020	0.4506	139950470
VGG11_bn	0.001	Default	Sequential	0.3007	0.5200	1.7236	1.4234	139950470
VGG11_bn	0.001	Default	Linear	1.0000	0.9600	0.0001	0.0001	128796422
VGG11_bn	0.001	0.0001	Linear	1.0000	0.9200	0.0001	0.0001	128796422
VGG11_bn	0.001	0.0001	Sequential	0.5105	0.6000	1.1900	0.5874	139950470
VGG11_bn	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	139950470

VGG11_bn	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	128796422
VGG11_bn	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	128796422
VGG11_bn	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	139950470
VGG_13	0.01	Default	Sequential	0.2587	0.2800	2,8100	0.7180	140129478
VGG_13	0.01	Default	Linear	0.2867	0.2800	1.7639	1.5456	128975430
VGG_13	0.01	0.0001	Linear	0.2587	0.2800	1.8000	1.8100	128975430
VGG_13	0.01	0.0001	Sequential	0.2308	0.2800	1.0300	1.6800	140129478
VGG_13	0.001	Default	Sequential	0.2657	0.2800	1.7641	1.4610	140129478
VGG_13	0.001	Default	Linear	0.2657	0.3600	1.7663	1.4370	128975430
VGG_13	0.001	0.0001	Linear	0.2517	0.2800	1.7670	1.5400	128975430
VGG_13	0.001	0.0001	Sequential	0.2517	0.4000	1.7634	0.6238	140129478
VGG_13	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	140129478
VGG_13	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	128975430
VGG_13	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	128975430
VGG_13	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	140129478
VGG13_bn	0.01	Default	Sequential	0.2657	0.2800	1.7965	1.0150	140135366
VGG13_bn	0.01	Default	Linear	0.2238	0.2800	0.0001	1.5627	128981318
VGG13_bn	0.01	0.0001	Linear	0.2587	0.3600	0.0003	1.0150	128981318
VGG13_bn	0.01	0.0001	Sequential	0.2448	0.3600	1.8392	1.0290	140135366
VGG13_bn	0.001	Default	Sequential	0.4406	0.4800	1.3956	0.6856	140135366
VGG13_bn	0.001	Default	Linear	1.0000	0.9200	0.0001	0.0001	128981318
VGG13_bn	0.001	0.0001	Linear	1.0000	0.9200	0.0001	0.0001	128981318
VGG13_bn	0.001	0.0001	Sequential	0.5384	0.5400	1.0770	0.6147	140135366
VGG13_bn	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	140135366
VGG13_bn	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	128981318
VGG13_bn	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	128981318
VGG13_bn	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	140135366
VGG16	0.01	Default	Sequential	0.2657	0.2800	24,5200	0.9740	145439174

VGG16	0.01	Default	Linear	0.2517	0.2800	1.7620	1.4800	134285126
VGG16	0.01	0.0001	Linear	0.2587	0.2800	1.7717	0.0010	134285126
VGG16	0.01	0.0001	Sequential	0.2727	0.2800	1.090	1.3562	145439174
VGG16	0.001	Default	Sequential	0.2448	0.2800	1.7717	1.4770	145439174
VGG16	0.001	Default	Linear	0.2657	0.2800	1.7675	1.4997	134285126
VGG16	0.001	0.0001	Linear	0.2727	0.2800	1.7677	1.4916	134285126
VGG16	0.001	0.0001	Sequential	0.2517	0.2800	1.7739	0.7477	145439174
VGG16	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	145439174
VGG16	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	134285126
VGG16	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	134285126
VGG16	0.0001	0.0001	Sequential	0.9441	0.8800	0.1520	0.0001	145439174
VGG16_bn	0.01	Default	Sequential	0.2517	0.2800	1.8024	1.0656	145447622
VGG16_bn	0.01	Default	Linear	0.2517	0.2800	1.7641	1.0715	134293574
VGG16_bn	0.01	0.0001	Linear	0.3147	0.3200	1.6943	1.0280	134293574
VGG16_bn	0.01	0.0001	Sequential	0.2517	0.2800	1.7804	0.7027	145447622
VGG16_bn	0.001	Default	Sequential	0.4825	0.6000	1.1369	0.4847	145447622
VGG16_bn	0.001	Default	Linear	1.0000	0.8000	0.0001	0.0001	134293574
VGG16_bn	0.001	0.0001	Linear	1.0000	0.4800	0.0016	0.0001	134293574
VGG16_bn	0.001	0.0001	Sequential	0.8811	0.4800	0.3166	0.0009	145447622
VGG16_bn	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	145447622
VGG16_bn	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	134293574
VGG16_bn	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	134293574
VGG16_bn	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	145447622
VGG19	0.01	Default	Sequential	0.2587	0.2800	1.526	1.0415	150748870
VGG19	0.01	Default	Linear	0.2797	0.2800	1.0546	1.375400	139594822
VGG19	0.01	0.0001	Linear	0.2797	0.2800	1.6434	1.5654	139594822
VGG19	0.01	0.0001	Sequential	0.2517	0.2800	2.057	1.7216	150748870
VGG19	0.001	Default	Sequential	0.2657	0.2800	1.7686	1.5590	150748870

VGG19	0.001	Default	Linear	0.2657	0.2800	1.7636	1.5226	139594822
VGG19	0.001	0.0001	Linear	0.2657	0.2800	1.7666	1.3971	139594822
VGG19	0.001	0.0001	Sequential	0.2378	0.3200	1.8348	0.0001	150748870
VGG19	0.0001	Default	Sequential	0.5875	0.6400	0.8203	0.2067	150748870
VGG19	0.0001	Default	Linear	1.0000	0.7600	0.0001	0.0001	139594822
VGG19	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	139594822
VGG19	0.0001	0.0001	Sequential	1.0000	0.8400	0.0001	0.0001	150748870
VGG19_bn	0.01	Default	Sequential	0.2517	0.3600	1.7813	1.1745	150759878
VGG19_bn	0.01	Default	Linear	0.2727	0.4400	1.7709	1.1805	139605830
VGG19_bn	0.01	0.0001	Linear	0.2587	0.3200	1.7503	1.0514	139605830
VGG19_bn	0.01	0.0001	Sequential	0.2937	0.4400	1.8339	0.5404	150759878
VGG19_bn	0.001	Default	Sequential	0.5524	0.5600	1.1613	0.4400	150759878
VGG19_bn	0.001	Default	Linear	1.0000	0.7600	0.0016	0.0001	139605830
VGG19_bn	0.001	0.0001	Linear	0.9930	0.8400	0.0128	0.0001	139605830
VGG19_bn	0.001	0.0001	Sequential	0.8951	0.6800	0.3316	0.0069	150759878
VGG19_bn	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	150759878
VGG19_bn	0.0001	Default	Linear	1.0000	0.9600	0.0001	0.0001	139605830
VGG19_bn	0.0001	0.0001	Linear	1.0000	0.9600	0.0001	0.0001	139605830
VGG19_bn	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	150759878
RegNet_y_400mf	0.01	Default	Sequential	1.0000	0.8000	0.0008	0.0001	4049710
RegNet_y_400mf	0.01	Default	Linear	1.0000	0.8400	0.0006	0.0001	3905790
RegNet_y_400mf	0.01	0.0001	Linear	1.0000	0.8400	0.0039	0.0001	3905790
RegNet_y_400mf	0.01	0.0001	Sequential	1.0000	0.8800	0.0081	0.0002	4049710
RegNet_y_400mf	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	4049710
RegNet_y_400mf	0.001	Default	Linear	1.0000	0.8400	0.0001	0.0001	3905790
RegNet_y_400mf	0.001	0.0001	Linear	1.0000	0.9600	0.0003	0.0001	3905790
RegNet_y_400mf	0.001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	4049710
RegNet_y_400mf	0.0001	Default	Sequential	1.0000	0.9600	0.0006	0.0001	4049710

RegNet_y_400mf	0.0001	Default	Linear	1.0000	0.9600	0.0002	0.0001	3905790
RegNet_y_400mf	0.0001	0.0001	Linear	1.0000	0.9600	0.0180	0.0002	3905790
RegNet_y_400mf	0.0001	0.0001	Sequential	1.0000	0.9600	0.0008	1.0000	4049710
RegNet_y_800mf	0.01	Default	Sequential	1.0000	0.6400	0.0098	0.0001	5882142
RegNet_y_800mf	0.01	Default	Linear	1.0000	0.8800	0.0001	0.0001	5652222
RegNet_y_800mf	0.01	0.0001	Linear	1.0000	0.7600	0.0158	0.0003	5652222
RegNet_y_800mf	0.01	0.0001	Sequential	0.9371	0.7600	0.2914	0.0352	5882142
RegNet_y_800mf	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	5882142
RegNet_y_800mf	0.001	Default	Linear	1.0000	0.9600	0.0005	0.0001	5652222
RegNet_y_800mf	0.001	0.0001	Linear	1.0000	0.9600	0.0021	0.0001	5652222
RegNet_y_800mf	0.001	0.0001	Sequential	1.0000	0.9600	0.0021	0.0001	5882142
RegNet_y_800mf	0.0001	Default	Sequential	1.0000	0.9600	0.0013	0.0001	5882142
RegNet_y_800mf	0.0001	Default	Linear	1.0000	0.9600	0.0011	0.0001	5652222
RegNet_y_800mf	0.0001	0.0001	Linear	1.0000	0.9600	0.0010	0.0003	5652222
RegNet_y_800mf	0.0001	0.0001	Sequential	1.0000	0.9600	0.0009	0.0001	5882142
RegNet_Y_1_6GF	0.01	Default	Sequential	1.0000	0.6800	0.0310	0.0002	10574684
RegNet_Y_1_6GF	0.01	Default	Linear	1.0000	0.8400	0.0015	0.0001	10318764
RegNet_Y_1_6GF	0.01	0.0001	Linear	1.0000	0.9200	0.0145	0.0003	10318764
RegNet_Y_1_6GF	0.01	0.0001	Sequential	1.0000	0.8000	0.0216	0.0005	10574684
RegNet_Y_1_6GF	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	10574684
RegNet_Y_1_6GF	0.001	Default	Linear	1.0000	0.9600	0.0018	0.0001	10318764
RegNet_Y_1_6GF	0.001	0.0001	Linear	1.0000	0.9600	0.0023	0.0001	10318764
RegNet_Y_1_6GF	0.001	0.0001	Sequential	1.0000	0.9600	0.0004	0.0001	10574684
RegNet_Y_1_6GF	0.0001	Default	Sequential	1.0000	0.9600	0.0010	0.0001	10574684
RegNet_Y_1_6GF	0.0001	Default	Linear	1.0000	0.9600	0.0010	0.0001	10318764
RegNet_Y_1_6GF	0.0001	0.0001	Linear	1.0000	0.9600	0.0027	0.0002	10318764
RegNet_Y_1_6GF	0.0001	0.0001	Sequential	1.0000	0.9600	0.0003	0.0001	10574684
RegNet_y_3_2gf	0.01	Default	Sequential	0.8462	0.6800	0.4218	0.0451	19344160

RegNet_y_3_2gf	0.01	Default	Linear	1.0000	0.8800	0.0027	0.0001	17932416
RegNet_y_3_2gf	0.01	0.0001	Linear	1.0000	0.7600	0.0096	0.0002	17932416
RegNet_y_3_2gf	0.01	0.0001	Sequential	0.6364	0.6000	0.7915	0.4625	19344160
RegNet_y_3_2gf	0.001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	19344160
RegNet_y_3_2gf	0.001	Default	Linear	1.0000	0.9600	0.0002	0.0001	17932416
RegNet_y_3_2gf	0.001	0.0001	Linear	1.0000	0.9600	0.0005	0.0001	17932416
RegNet_y_3_2gf	0.001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	19344160
RegNet_y_3_2gf	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	19344160
RegNet_y_3_2gf	0.0001	Default	Linear	1.0000	0.9600	0.0005	0.0001	17932416
RegNet_y_3_2gf	0.0001	0.0001	Linear	1.0000	0.9600	0.0004	0.0001	17932416
RegNet_y_3_2gf	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	19344160
RegNet_y_8gf	0.01	Default	Sequential	0.9650	0.7200	0.1024	0.0032	37914494
RegNet_y_8gf	0.01	Default	Linear	1.0000	0.8000	0.0005	0.0001	37376574
RegNet_y_8gf	0.01	0.0001	Linear	0.9930	0.8400	0.0690	0.0033	37376574
RegNet_y_8gf	0.01	0.0001	Sequential	0.9441	0.8000	0.1974	0.0049	37914494
RegNet_y_8gf	0.001	Default	Sequential	1.0000	0.8800	0.0003	0.0001	37914494
RegNet_y_8gf	0.001	Default	Linear	1.0000	0.9600	0.0001	0.0001	37376574
RegNet_y_8gf	0.001	0.0001	Linear	1.0000	0.9600	0.0002	0.0001	37376574
RegNet_y_8gf	0.001	0.0001	Sequential	1.0000	0.9600	0.0003	0.0001	37914494
RegNet_y_8gf	0.0001	Default	Sequential	1.0000	0.9600	0.0001	0.0001	37914494
RegNet_y_8gf	0.0001	Default	Linear	1.0000	0.9600	0.0002	0.0001	37376574
RegNet_y_8gf	0.0001	0.0001	Linear	1.0000	0.9600	0.0003	0.0001	37376574
RegNet_y_8gf	0.0001	0.0001	Sequential	1.0000	0.9600	0.0001	0.0001	37914494