Comparison of Classification Methodologies using Convolutional Neural Networks in a Dataset of Plant Leaf Diseases

Ruairi O’Donohoe

A Thesis Submitted in Partial Fulfilment

of the requirements for the

Degree of

Master of Science in Data Analytics

Diagram

Description automatically generated with medium confidence

27th September 2024

Supervisor: Vikas Tomer

Acknowledgements

I would like to thank my supervisor, Vikas Tomer, who has provided valuable assistance and guidance throughout this project. I am also grateful to all my lecturers at CCT for providing me with a valuable education over the last 12 months.

Lastly, I would like to personally thank my partner Gina who has been extremely supportive and patient throughout my pursuit of this Master’s degree, and I hope she’ll be happy to have me back and out from the depths of a computer screen.

Abstract

**Aim:** This project investigates the impact of classification methodology selection on the performance of four Convolutional Neural Network models, as applied to a multi-label image dataset.

**Background:** The dataset used in this project’s experiments contains images of plant leaves with one or more diseases. Two classification methodologies, multi-label and multi-class, and their associated parameters, are compared with regards to the resulting model performance metrics.

**Discussion:** It is hypothesised that the multi-label methodology will perform better on this multi-label dataset however previous research has applied different methodological approaches to this topic. Transfer learning is used to train the CNN models, and Hyperparameter tuning is conducted on the best performing model.

**Conclusion:**  The results indicate that the multi-label classification models perform better with regards to Loss and Accuracy metrics, however they perform worse for F1 score, which is suggested to be a more appropriate metric for this type of task. This is a surprising insight which refutes the hypothesis that the multi-label methodology would perform better on this task. The nature of this discrepancy and the appropriateness of each methodology is further discussed.

**Implications for Practice:** It is hoped that this research will assist both in furthering the understanding of classification of a multi-label dataset, and in its application in plant disease detection through machine learning.

**Keywords:** Convolutional neural network, multi-label, multi-class, classification, plant disease

**Relevant Links:**

**GitHub Link:** ProjectRepository

<https://github.com/CCT-Dublin/msc-data-analytics-capstone-thesis-july-2024-ruairi-o-donohoe.git>

**Google Drive Link:** For dataset storage, directory is too big to be stored in GitHub

<https://drive.google.com/drive/folders/1x1-zXPmGUENZ8Ldoq3ELUbIIiWXouuXD?usp=sharing>

Table of Contents

[1. List of Figures 1](#_Toc178154134)

[2. List of Tables 1](#_Toc178154135)

[3. Introduction 2](#_Toc178154136)

[3.1 Background 2](#_Toc178154137)

[3.2 Domain Area, Problem Area and Delimitations 3](#_Toc178154138)

[3.3 Research Aim, Objectives and Questions 3](#_Toc178154139)

[3.3.1 Research Aim 3](#_Toc178154140)

[3.3.2 Research Objectives 3](#_Toc178154141)

[3.3.3 Research Questions 4](#_Toc178154142)

[4. Literature Review 6](#_Toc178154143)

[4.1 Convolutional Neural Networks 6](#_Toc178154144)

[4.1.1 Origins 6](#_Toc178154145)

[4.1.2 Datasets and Competitions 6](#_Toc178154146)

[4.1.3 Key Concepts 8](#_Toc178154147)

[4.2 Notable CNN Models 12](#_Toc178154148)

[4.2.1 “AlexNet” 12](#_Toc178154149)

[4.2.2 VGGNet 13](#_Toc178154150)

[4.2.3 Inception Nets 14](#_Toc178154151)

[4.2.4 MobileNets 17](#_Toc178154152)

[4.3 Transfer Learning and Fine Tuning 18](#_Toc178154153)

[4.4 FieldPlant Dataset and Research 20](#_Toc178154154)

[4.5 Multi-class vs Multi-label Task Selection 21](#_Toc178154155)

[4.5.1 Multi-class Classification 21](#_Toc178154156)

[4.5.2 Multi-label Classification 22](#_Toc178154157)

[4.5.3 Alternative Metrics 23](#_Toc178154158)

[4.6 Conclusion 24](#_Toc178154159)

[5. Methodology 25](#_Toc178154160)

[5.1 Research Methodology 25](#_Toc178154161)

[5.2 Ethics 27](#_Toc178154162)

[5.3 Preliminary Setup 27](#_Toc178154163)

[5.3.1 Dataset and system configuration 28](#_Toc178154164)

[5.3.2 Exploratory Data Analysis 28](#_Toc178154165)

[5.3.3 Dataset Preparation and Preprocessing 30](#_Toc178154166)

[5.4 Model Training and Evaluation 31](#_Toc178154167)

[5.4.1 Model Training Setup 31](#_Toc178154168)

[5.4.2 Training New Top Classifier 32](#_Toc178154169)

[5.4.3 Fine Tuning and Evaluation 33](#_Toc178154170)

[5.4.4 Hyperparameter Tuning 34](#_Toc178154171)

[6. Project Evaluation 36](#_Toc178154172)

[6.1 Selection of Image Dimensions and Batch Size 36](#_Toc178154173)

[6.2 Selection of Classification Methodology 36](#_Toc178154174)

[6.3 Methods to Address Class Imbalance 37](#_Toc178154175)

[6.4 Evaluation Metrics 38](#_Toc178154176)

[6.5 Evaluation of Hyperparameter Tuning Procedure 39](#_Toc178154177)

[6.6 Strengths, Weaknesses and Limitations 40](#_Toc178154178)

[7. Results 42](#_Toc178154179)

[7.1 Experiment 1: Effect of Dataset Stratification 42](#_Toc178154180)

[7.2 Experiment 2: Comparing Classification Methodology 42](#_Toc178154181)

[7.3 Experiment 3: Identify Best Performing Model 44](#_Toc178154182)

[7.4 Experiment 4: Conduct Hyperparameter Tuning 45](#_Toc178154183)

[8. Discussion of Results 48](#_Toc178154184)

[8.1 Experiment 1: Effect of Dataset Stratification 48](#_Toc178154185)

[8.2 Experiment 2: Comparing Classification Methodology 48](#_Toc178154186)

[8.3 Experiment 3: Identify Best Performing Model 49](#_Toc178154187)

[8.4 Experiment 4: Conduct Hyperparameter Tuning 49](#_Toc178154188)

[9. Conclusion and Future Work 52](#_Toc178154189)

[10. Appendix 53](#_Toc178154190)

[11. References 55](#_Toc178154191)

# List of Figures

[Figure 1: Convolutional and Pooling Layers. 9](#_Toc178179666)

[Figure 2: Hierarchical features learned by CNN trained on ImageNet 19](#_Toc178179667)

[Figure 3: Proportions of disease classes and crop classes in the FieldPlant dataset. 28](#_Toc178179668)

[Figure 4: Distribution of dimensions of Images in FieldPlant dataset. 28](#_Toc178179669)

[Figure 5: Training metrics during Hyperparameter tuning searching for best epoch. 42](#_Toc178179670)

[Figure 6: Experiment 4 - Best model training history 43](#_Toc178179671)

[Figure 7: Example of class predictions in multi-label classification. 50](#_Toc178179672)

[Figure 8: Example of class predictions in multi-class classification. 50](#_Toc178179673)

# List of Tables

[Table 1: Parameters for Multi-class and Multi-label classification methodologies 32](#_Toc178154200)

[Table 2: Results of Experiment 1 - Effect of Dataset Stratification 42](#_Toc178154201)

[Table 3: Results of Experiment 2 - Comparing Classification Methodology. 43](#_Toc178154202)

[Table 4: Results of Experiment 3 - Identify Best Performing Model 44](#_Toc178154203)

[Table 5: Experiment 4 - Hyperparameters discovered during RandomSearch. 45](#_Toc178154204)

[Table 6: Results of Experiment 4 - Hyperparameter Tuning 46](#_Toc178154205)

[Table 7: F1 Score per class, evaluating hyperparameter tuned model on test dataset. 47](#_Toc178154206)

# Introduction

## Background

Image classification is a common task in the field of Computer Vision, which involves training a model to predict the label class of an image (Rawat and Wang, 2017). A task which involves selecting a single label from a list of many possible classes is termed multi-class classification, and one that involves selecting multiple labels that are relevant to an image is termed multi-label classification (Tsoumakas and Katakis, 2009). The distinct differences between these methodologies necessitate differing approaches and parameter selections. However, there often appears to be varying use of these methodologies in the research with regards to the specific task, and whether this is intentional or not is not certain.

Convolutional Neural Networks (CNN) have excelled in computer vision and image classification (LeCun et al., 2015) since 2012, when a team using CNN models (Krizhevsky et al., 2012) won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2014) by a substantial margin. Since then, interest in CNN research has increased dramatically, leading to significant improvements in model architecture and performance.

CNNs have been applied to the many problem areas, one of which being plant disease detection (Adi et al., 2021), and a recent dataset collected in 2013 by The FieldPlant team (Moupojou et al., 2023) is the first publicly available dataset consisting of photographs taken entirely in the field as opposed to laboratory conditions. The experiments of this project are based on those of FieldPlant research; however, it appears that they utilise parameters that are more often associated with multi-class classification, despite the dataset containing many images with multiple labels. The primary experiment of this project investigates the effect of the choice of classification methodology, either multi-class or multi-label, on model performance. Using both methodologies, four pre-trained CNN models are trained on the FieldPlant dataset using transfer learning and fine-tuning. Automated hyperparameter tuning is then performed on the best performing model architecture to enhance its performance.

## Domain Area, Problem Area and Delimitations

The primary focus of this research project is the utilization of data analytical methodologies in the domain area of image classification using Convolutional Neural Networks, as applied to the problem area of plant disease detection using the FieldPlant dataset.

The models used in this project are commonly used in research, however their selection is not based on their performance, rather because of their use in the FieldPlant research, and better performing models may exist. This project’s experiments do not seek to optimize model performance, but instead to investigate the research objectives specified below. Several key concepts related to machine learning are explained in the Literature Review, however the mathematical discussion is beyond the scope of this project.

## Research Aim, Objectives and Questions

### Research Aim

This research project aims to compare the effect of applying two different classification methodologies, multi-class and multi-label, and their appropriate parameter selection, on the performance of CNN models, and their suitability for use with a multi-label dataset.

### Research Objectives

These objectives have evolved throughout the project, as iterative data analysis and experimentation have revealed further insights. Objectives 1 and 3, which investigate the use of stratification and identify the best performing model, respectively, serve to support the remaining objectives. Objectives 2 and 4 serve as the primary objectives as they aim to investigate the effect of the specific classification methodologies on model performance.

1. Effect of Dataset Stratification: Evaluate the effect of stratification on model performance

The FieldPlant dataset is heavily imbalanced regarding the disease classes, however the original research paper does not use dataset stratification. In this project, stratification is performed to create a more equally proportioned distribution of classes between the training, validation and test sets. One CNN model architecture is trained and evaluated on both the stratified and unstratified datasets, to investigate the effects of stratification on model performance.

1. Comparing Classification Methodology: Compare the effect of multi-class vs. multi-label classification with regards to model performance.

Experiments in the FieldPlant research use model parameters that are more frequently associated with multi-class classification methodology, despite the dataset containing images with multiple labels. Softmax is used as the classifier function, sparse categorical crossentropy as the loss function, and categorical accuracy as an evaluation metric. Parameters more associated with multi-label classification are a sigmoid classifier function, binary crossentropy loss, and binary accuracy. (Lydia and Francis, 2020). This experiment performs transfer learning and fine-tuning using four pretrained models, as in FieldPlant, to compare the effect of methodological selection on model performance.

1. Identify the Best Model: Identify the best performing model and classification methodology

Following completion of objective two, utilizing four models and two methodologies, a total of eight models are trained and evaluated on the test dataset. The best performing combination of model and methodology is identified.

1. Conduct Hyperparameter Tuning: Conduct hyperparameter tuning on the best performing model

Perform hyperparameter tuning on the best performing model identified in the previous objective. Train the best model using the best identified hyperparameters, and evaluate performance on the test dataset, as compared to the standard models trained in the previous experiments.

### Research Questions

1. *Does data stratification improve the performance of an image classification CNN model?* It is hypothesized that the models will perform better using the stratified dataset.
2. *Does using multi-label classification methodology improve performance on a multi-label dataset as compared to multi-class methodology?* It is hypothesized that the multi-label methodology will perform better due to the dataset also being multi-label.
3. *What is the best combination of model and classification methodology from a selection of four pretrained CNN models?* It is hypothesized that the best combination will include the use of multi-label methodology, as suited to the dataset.
4. *Does hyperparameter tuning improve model performance over standard transfer learning?* It is hypothesized that the hyperparameter tuned model will perform better than the standard models from the initial experiments.

The remainder of this body of this paper is structured in sections as follows:

4. Literature Review: Discuss related works in literature that have guided this project.

5. Methodology: Discuss the steps taken to complete the research objectives.

6. Project Evaluation: A critical evaluation of the steps involved in this project.

7. Results: The results of the experiments are provided.

8. Discussion of Results: The results are discussed and evaluated.

# Literature Review

This review aims to provide a deeper understanding of image classification using Convolutional Neural Networks to guide this project’s experiments, and is organized using thematic and chronological principles. After an initial background is provided, several key concepts are then discussed as these will be relevant thereafter. The evolution of CNNs is then explored through discussion of the four models used in this project’s experiments: VGG16, InceptionV3, InceptionResNetV2, MobileNetV2. This is followed by discussion of the transfer learning approach and dataset used during model training. To conclude, details of the two classification methodologies being investigated in this project are compared.

Materials used in this literature review are mainly sourced from high quality peer-reviewed journals and databases such as IEEE, ACM, Frontiers, ScienceDirect, and from official documentation where relevant, to ensure a high level of quality and validity.

## Convolutional Neural Networks Background

### Origins

A convolutional neural network (CNN) is a type of Artificial Neural Network (ANN) which has become prominent in the field of computer vision. CNNs differ to typical ANNs through their use of convolutional layers, which are used to process data in the form of multiple arrays, such as RGB pixel data (LeCun et al., 2015). CNN models trace back to the 1990’s with the work of Yann LeCun, who designed his “LeNet-5” model for the recognition of hand-written digits (LeCun et al., 1998). Research during this time looked promising, however, it wasn’t until 2012 that CNN models truly gained a foothold in the field of computer vision, when the winning team (Krizhevsky et al., 2012) of ILSVRC competition (Russakovsky et al., 2014) entered a CNN model which won by an impressive margin. The ILSVRC is discussed below before further exploring the evolution of CNNs.

### ImageNet

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual competition established in 2010 for the purpose of creating a standard benchmark competition to aid in advancing algorithms and models for computer vision, and has greatly contributed to the advancement of CNNs. The backbone of the ILSVRC competition is the large dataset named ImageNet (Deng et al., 2009), which is discussed briefly below.

The ImageNet project began in 2009, with the aim of providing a large-scale publicly available dataset of cleanly labelled, full resolution images. The structure of ImageNet is similar to the text-based database named WordNet (Soergel, 1998), which constructs a hierarchical representation of word terms e.g. mammal, dog, German Shephard. The goal of ImageNet was to gather a large quantity of multi-labelled images for each of these terms, or “Synonym Sets” (synsets). Images were gathered from internet search engines using terms from WordNet through multiple languages. By August 2014, the dataset contained almost 15 million images, across 21,841 synsets, each containing an average of 650 labelled images, far surpassing similar datasets such as PASCAL (Everingham et al., 2015) and Caltech-256 (Fei-Fei et al., 2006). Due to the enormity of this project, external assistance was required with image labelling, and the website “Amazon Mechanical Turk” was utilized (Sorokin and Forsyth, 2008), by requesting paid services from people on the internet. For a label to be deemed accurate, agreement between multiple annotators was required. Independent assessment of random images demonstrated that the accuracy of the labels in this dataset was 99.7% (Deng et al., 2009).

The ILSVRC competitions use a subset of 1.2 million images across 1000 classes from the ImageNet dataset as its training data, 50,000 as validation data (Russakovsky et al., 2014), and 100,000 newly sourced images which were not part of the original ImageNet dataset as test data. The ILSVRC began as an image classification competition but has since expanded to include single object localisation, and object detection. The classification task requires the model to classify an image with a list of predicted labels and their degree of probability. The most prominent object in the image is assigned to be the image’s sole label, making this a multi-class dataset and classification task. Two metrics are used to measure performance, the top-1 and top-5 error rate, which measure how often the true label does not appear in the model’s top-1 or top-5 most probable predictions respectively, with a lower error rate indicating better performance. The top-5 error rate is used to select the winner of the competition. The single object localisation and object detection tasks require the model to list all the object classes within the image, and to then locate one or all instances of each class, respectively.

### Key Concepts

An understanding of the following concepts will be of benefit for the subsequent sections. A short explanation for each concept is provided, however the underlying mathematics are beyond the scope of this project.

#### 

#### Activation Functions

The output of a neuron in a CNN is determined by the computation of an activation function. These functions introduce non-linearity into the neural network, which significantly improves discriminative ability. ReLU (Nair and Hinton, 2010) is the most widely used, as it improves the model optimization and training speed as compared to using TanH (Krizhevsky et al., 2012). ReLU, defined as *f(x) = max(x, 0),* returns the maximum value of either the input (x) or 0. Other functions include TanH, Swish, Softmax, and Sigmoid. TanH, commonly used before the development of ReLU, outputs values between -1 and 1. Swish is a more recently developed function which “tends to work better than ReLU on deeper models” (Ramachandran et al., 2017).

Softmax is often used as the classifier function for multi-class classification, as the goal is to predict the single most probable class (TensorFlow, 2024a). Each unit in this layer represents a class. The input to the softmax layer is a vector of values computed by the model, which is converted to a probability distribution where all the probabilities sum to one, and the unit with the highest probability is chosen as the predicted class. For any one probability to increase, at least one other must decrease. Due to this interdependent probability, the Softmax function is less suited to multi-label classification, in which the presence of any one label is not impacted by the probability of any other (Lydia and Francis, 2020).

Sigmoid is the equivalent of a two-class softmax, where the classes are 0 and 1. This is most often used in binary or multi-label classification where the goal is to determine if a specific class is present or not. For input values of approximately -5 or less, the output is very close to 0, and for input values of approximately +5 or greater, the output is very close to 1 (TensorFlow, 2024b). The probability of each class is calculated independently, unlike with Softmax, making Sigmoid an appropriate function for multi-label classification where an image may contain multiple classes independent of each other.

#### CNN Model and Layers

CNN models are best suited to processing data in the form of multidimensional arrays, such as image RGB pixel data. CNNs main feature is the convolution layer, which takes advantage of the inherent patterns contained within the spatial orientation of pixels, to automatically learn hierarchical representations of training data, replacing the need for prior feature engineering (LeCun et al., 1998).

With an image represented as a matrix of pixels, a convolution layer applies matrix filters of smaller window size (e.g. 3x3 or 5x5 pixels) which connect to a local patch of image pixels. These filters slide, or “convolve”, over the pixel matrix, and through calculating the matrix dot product at each step, return a new matrix called a feature map (Figure 1A) which is fed to the activation function to compute the layer’s output. The convolution process results in extracting features from the pixel’s spatial arrangement, such as corners and edges.

A pooling layer then convolves a window over the previous output matrix, commonly of size 2x2 pixels and taking 2 steps each time, which is most often used to extract either the average or maximum value within the window before moving to the next location (Figure 1B). This serves to summarise the values of the previous matrix and extract the important feature information, while also reducing the matrix dimensions. The output of the pooling layer is then fed to the next convolution layer as the process is repeated, and more higher-level features are extracted, such as patterns or facial features. The hierarchical feature extraction process is best described as follows: “In images, local combinations of edges form motifs, motifs assemble into parts, and parts form objects.” (LeCun et al., 2015).

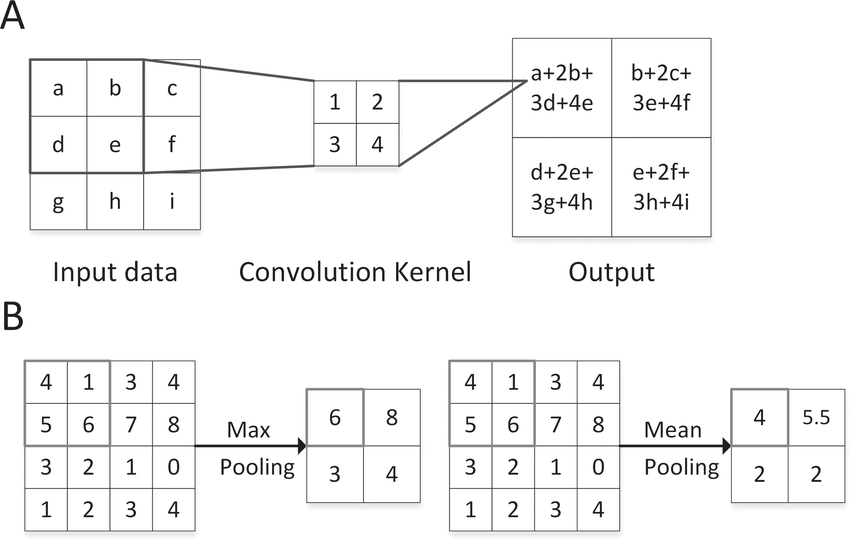


Figure 1: Convolutional and Pooling Layers.

**A. Convolutional Layer**: The convolutional kernel is applied to the input data, and the dot product is calculated for each position, resulting in the output matrix. **B. Pooling Layers**: The pooling layer extracts either the maximum or average value from the window at each position (Zhang et al., 2018).

Classic CNN architecture typically consists of several stacked alternating convolutional and pooling layers, followed by one or several fully connected layers. A fully connected layer differs from a convolutional layer in that all its neurons are connected to all neurons from the previous layer, rather than to local patches of neurons in the previous feature map. Fully connected layers perform high-level decision-making using the features extracted from the preceding layers, ultimately leading to a prediction (EITCA Academy, 2023).

The capacity of a CNN model to learn is influenced by its size, both in terms of depth (the number of layers) and width (the number of filters per layer), as larger models contain more connections between layers, each with a learnable weight parameter. The “LeNet-5” model consisted of five alternating convolutional and average pooling layers, followed by two fully connected layers, the second of which using a 10-way softmax function to predict amongst the ten digits in the dataset.

A substitutable variation on the convolutional layer is called the Depthwise Separable Convolutional Layer (Sifre and Mallat, 2014), which splits the job of the standard convolutional layer into two separate stages, “filtering” then “combining”. A convolutional layer typically applies the filters across all input channels (input “depth”) before combining the outputs. A depthwise convolutional applies an individual filter to each of the input channels, before a pointwise convolution applies a 1x1 convolution to the outputs to recombine them. The depthwise separable convolutional layer reduces the computation cost considerably, with only a minor reduction of accuracy (Howard et al., 2017), and the relative savings increase with even larger filter sizes.

#### Back Propagation and Stochastic Gradient Descent (SGD)

The goal of training a classification model is to maximise its ability to predict the class of a new unseen sample. During training, iterative adjustment are made to the weight values of the model’s connections using the “back propagation” algorithm (LeCun et al., 1989), with the aim of minimizing the “loss”. Loss is a measure of model prediction error with respect to the true labels (Demirkaya et al., 2020; TensorFlow, 2024c). By working to reduce the loss, the model’s predictive power increases. Stochastic gradient descent (SGD) is a commonly employed algorithm which uses small batches of randomly selected data to train a neural network, where the loss calculated over a batch is an estimate of the loss over the entire dataset. The model’s weights are then updated in the direction of the negative gradient to reduce the loss, before the process is repeated with another random batch of data (Nagendram et al., 2023). Several optimized versions of SGD have been developed, such as momentum (Sutskever et al., 2013), Adagrad (Duchi et al., 2011) and RMSProp (Hinton and Tieleman, 2012). The “Adam” (Adaptive Momentum Estimation) optimizer combines the advantages of these algorithms and computes individual adaptive learning rates for each of the model’s parameters. Adam is computationally efficient with little memory requirement, and works well with large numbers of parameters. (Kingma and Ba, 2014).

#### Learning Rate

Learning rate (LR) is often considered to be the most important parameter in training neural networks (Raitoharju, 2022). The LR controls the size of the step against the gradient during SGD. A high LR will update the model’s weights more significantly and move down the gradient faster thus speeding up model training. However, if LR is set too high, this can result in a move up the gradient causing the loss function to worsen. If LR is small, training will be slow but can be more accurate, but if too small, the loss function may get stuck in a local minimum, a local low point of the loss curve, but not the lowest globally. As training progresses, the movements down the loss curve often get smaller, so LR can be adjusted to a smaller value manually, or automatically by using a decay parameter.

#### Overfitting and Regularization

Overfitting describes when a model learns the training data to such a great extent that its ability to generalize to new data is impaired. Regularization is a set of techniques that can reduce overfitting in various ways. Dropout(Srivastava et al., 2014) involves randomly setting the output of a neuron to 0 with a predefined probability, often set to 50%, thus not partaking in the forward pass of information to the next layer, or in back propagation. The dropout neurons are randomly selected for each iteration, and therefore the layers see a changing distribution of input data. This means that a layer cannot rely on the input from any one particular neuron, and is forced to learn a more generalized representation of input data (Srivastava et al., 2014). L2 weight decay is another common form of regularization, which penalizes larger weight values, encouraging them to stay small (Tibshirani, 1996). Data Augmentation is a means of artificially generating additional training data by applying various transformations to the original training data (Mikołajczyk-Bareła and Grochowski, 2018). Common image augmentation techniques include horizontal or vertical, stretching or distorting the shape, and changing pixel colour. The resultant dataset provides more data to the model for training and is conducive to learning more generalized representations. Batch Normalization, as discussed next, is also thought to have a regularization effect.

#### Batch Normalization (BN)

Networks can train faster if their input is normalized to have a mean of 0 and a variance of 1 (LeCun et al., 1998b; Wiesler & Ney, 2011). This concept is also extended to the inputs of layers within the network. During SGD, the model’s layers see a different distribution (mean and variance) of input values each iteration. However, because the distribution changes with each new batch of data, there is an internal covariant shift, defined as “the change in the distributions of internal nodes of a deep network” (Ioffe and Szegedy, 2015). BN aims to improve training speed by reducing covariant shift, by normalising the outputs of each layer prior to the activation function, before being passed to the next layer as input. Additional benefits of BN include its regularisation effect and subsequent reduced reliance on other techniques such as Dropout or L2 weight decay; and the ability to use a higher initial learning rate and faster learning rate decay, again significantly improve training speed.

## Notable CNN Models

The following sections introduce several innovative models which have had a major impact on the development of CNN architecture throughout the years. The first section discusses “AlexNet”, the model which reignited interest in CNNs in 2012. The subsequent sections discuss several other model architectures which are used in this project’s experiments.

### “AlexNet”

CNNs became much more prominent in the field of computer vision thanks to the winning model of the ILSVRC 2012. The team headed by Alex Krizhevsky developed a CNN model based on “LeNet-5”, aptly named “AlexNet” (Krizhevsky et al., 2012). AlexNet consists of five convolutional layers (filter sizes 11x11, 5x5 and three 3x3 sequentially) followed occasionally by max pooling, and ends with three fully connected layers and a 1000-way softmax to classify amongst the classes of the ILSVRC dataset. This model is deeper than LeNet-5, which the researchers demonstrated to be important, as removing any convolutional layer reduced performance significantly. The research team admit that a large part of their success was due to the contemporary availability of large-scale datasets such as ImageNet, and advancements in GPU technology allowing for more efficient computation. Two GPUs, each with 3GB of memory, were used to train this network, taking almost six days. The depth of “AlexNet” was ultimately limited by the amount of GPU memory available, and the researchers speculated that further performance improvements would be possible with faster GPUs and larger datasets.

Some data preprocessing was performed, including resizing of pixel matrices to a uniform shape of 256x256 and centering the values by subtracting their mean value. ReLU was chosen over the more common TanH, as the team’s experiments demonstrated that ReLU reached a 25% error rate six times faster than TanH. AlexNet has approximately 60 million parameters, increasing the risk of overfitting, which was combatted using dropout and data augmentation with image translation and horizontal flipping techniques, as well as pixel colour alterations. Dropout was used at a rate of 0.5, which substantially reduced overfitting, however doubled the required training time to achieve convergence. Pooling layers used overlapping windows, by setting the stride parameter to smaller than the filter size, which further reduced overfitting. Training used SGD with a batch size of 128. The initial learning rate was set to 0.01 and was reduced by a factor of 10 a total of three times throughout the training cycle.

The team’s 2012 ILSRVC entry consisted of an ensemble of seven such “AlexNet” models, where the final predictions were the average across all softmax predictions. This ensemble won with a top-5 error rate of 15.3% (Krizhevsky et al., 2012), an impressive result above the second-place rate of 26.2%.

### VGGNet

Following the success of “AlexNet”, the VGG team began experimenting with further increasing model depth (Simonyan and Zisserman, 2015). They maintained the classic architecture of “LeNet-5” and “AlexNet” using stacked convolutional layers with ReLU, occasional max pooling, and three fully connected layers ending in a 1000-way softmax. Several deviations were made, however, from the classic models. A one-pixel padding was used around the image, to maintain the spatial resolution of the original input. Max pooling was used only after some convolutional layers, with a window size of 2x2 and a stride of 2, halving the dimensionality. Instead of using large filter sizes, stacks of convolutional layers with smaller 3x3 filters were employed, which could effectively replicate the same receptive field as the bigger filters e.g. two 3x3 filters can replace one 5x5, three 3x3 filters can replace a 7x7. This reduces computational cost, as a 7x7 filter would have 81% more parameters than three 3x3 filters, thus allowing for further increases in model depth. The additional non-linear transformation of each layer also improves the model’s discriminative capacity. Weight decay, dropout and data augmentation were implemented to combat overfitting.

Experiments using the ILVRC dataset demonstrate that substituting the 5x5 filters with two 3x3 filters improved top-1 error rate by 7%, demonstrating that deeper models using smaller filters outperform shallower models with larger filters. Experiments also compared five models of increasing depth, from 11 layers to 19, and their results indicate that increasing model depth further reduces error rate, plateauing at 19 layers.

The VGG entry to ILSVRC 2014 used an ensemble of 7 such models, achieving a top-5 error rate of 7.3%, coming in second place behind the winning ensemble of seven “GoogLeNet” models with a top-5 error rate of 6.7%. However, post-competition, an ensemble of VGG’s two best models achieved a top-5 error rate of 6.8%. When evaluating the single best performing model, the VGG model scored a top-5 error rate of 7.0%, outperforming a single “GoogLeNet” model by 0.9%.

### Inception Nets

The Inception networks are named for their use of “Network-in-Network” (NIN) architecture and the similarity of concept to a popular movie (Szegedy et al., 2015). Several versions of Inception networks have been released, with InceptionV1 winning the 2014 ILSVRC, also named “GoogLeNet” in homage to the original LeNet-5. The goal of Inception networks is to enhance the model’s representation capacity by increasing its depth in a computationally efficient manner. The deep VGG architecture is much simpler, however at the cost of greater computation. InceptionV1 had only 7 million parameters, 9 times fewer than AlexNet and 27 times fewer than VGGNet (Szegedy et al., 2015).

Inception Networks deviate from the classical CNN architecture by employing a NIN-type architecture. The original NIN introduced the “MlpConv” layer, which is the insertion of a micro-network of Multilayer Perceptrons between convolutional layers (Lin et al., 2013). Lin’s model consisted of a stack of 3 such MlpConv layers. The typical fully connected layers before a softmax classifier were replaced by Average Pooling, which helped to combat overfitting, even outperforming fully connected layers with dropout. The NIN model achieved new state-of-the-art results on two commonly used image classification datasets. The model scored an error rate of 8.81% on CIFAR-10 dataset (Krizhevsky, 2012) using data augmentation and dropout, outperforming the previous best of 9.32%. Without augmentation, the model scored an error rate of 35.68% on CIFAR-100 dataset (Krizhevsky, 2012), outperforming the previous best of 36.85% (Lin et al., 2013).

InceptionV1 implements the NIN concept using several stacked Inception modules, which consist of parallel convolutional layers of various filter sizes (1x1, 3x3 and 5x5) whose outputs are concatenated before being fed into the next module. A 1x1 convolutional layer is used to reduce dimensionality prior to the larger filters, resulting in a significant reduction in computation, subsequently allowing for greater model depth. Auxiliary classifiers are connected to intermediate layers within the model, enhancing the back propagation signal to earlier layers. Average Pooling and 40% Dropout are used before the final fully connected layer with 1000-way softmax. InceptionV1 consists of 22 trainable layers, not including pooling layers, all using ReLU. The ensemble of seven such models achieved a top-5 error rate of 6.7%, signifying a 56.5% relative reduction in error rate compared to “AlexNet”. This model was tested against a human annotator, who scored an error rate was 5.1% (Russakovsky et al., 2014), indicating that GoogLeNet achieved near human performance. This significant improvement is attributed to advancements in algorithms and model architecture, rather than hardware or data availability.

In InceptionV2 (Ioffe and Szegedy, 2015), Batch Normalization (BN) is introduced which significantly improves training time. The team’s experiments show that adding it to “GoogLeNet” could match its performance while requiring only 7% the number of training steps, and further improve performance with more training (Ioffe and Szegedy, 2015). Due to BN’s regularization effect, experiments also show that Dropout can be removed, further improving training time without sacrificing performance. BN also allows InceptionV2 to increase initial learning rate to 30 times higher (0.045) than InceptionV1, and to decay learning rate six times faster. Implementing the above changes, InceptionV2 evaluated on the ILSVRC dataset scores a top-1 accuracy (inverse of error rate) of 74.8%, outperforming InceptionV1 scoring 72.2%, while also requiring only a fifth the number of training steps.

InceptionV3 introduces several forms of factorization in order to enhance computational efficiency (Szegedy et al., 2016b). The 5x5 and 7x7 layers are factorized into stacks of 3x3 layers, as in VGGNet. The previous pooling method is replaced by two parallel blocks of pooling and convolution each with stride 2, reducing dimensionality with lower computation. The reduction in overall computation allows for expansion to a depth of 42 layers, while costing only 2.5 times the computation of “GoogLeNet”. When evaluated on the ILSVRC dataset, InceptionV3 achieves a Top-1 and Top-5 error rate of 21.2% and 5.6% respectively, setting a new state-of-the-art.

InceptionV4 (Szegedy et al., 2016a) is the most recent version. During the migration of implementation to TensorFlow (Abadi et al., 2016), much of the previous “baggage” that came with the accumulative development of InceptionV3, could be removed. InceptionV4 has a much simpler architecture, and due to TensorFlow’s memory optimization, was able to be made deeper and wider once again, resulting in 14 Inception modules compared to the 10 modules of InceptionV3.

InceptionResNet (Szegedy et al., 2016a) introduces Residual Connections (He et al., 2015) between its Inception Modules. A Residual Connection adds the layer’s unaltered input to its resultant output, subsequently enhancing the propagation of the original signal deeper through the network. In InceptionResNet, the module’s output dimensions are resized to match the input, and the output values are scaled down in magnitude, to prevent the network from network “dying” and producing only 0 values. The scaled and resized outputs are then added to the inputs before being passed to the next module. This summation replaces the filter concatenation of standard Inception modules. He et al. (He et al., 2015) argue that residual connections are necessary for larger models, but the Inception team do not agree, as experiments demonstrate that their deep models are capable of training without them, however their inclusion does greatly speed up training (Szegedy et al., 2016a). Batch Normalization is not applied after the Residual summations, as the memory costs proved too high. InceptionResNetV1 and V2 models have similar computational complexity to InceptionV3 and InceptionV4 respectively. InceptionV4 and InceptionResNetV2 were evaluated on the ILSVRC dataset and achieved a top-5 error rate of 3.8% and 3.7% respectively, outperforming InceptionV3. An ensemble of one InceptionV4 and three InceptionResNetV2 models achieved a top-5 error rate of 3.1%, setting a new state-of-the-art.

### MobileNets

The goal of the MobileNet team is to develop smaller, more efficient models, better suited for use on mobile devices, without compromising performance (Howard et al., 2017). A small model contained fully on a device could enhance user privacy by eliminating the need to send data to a separate server. Other architectures with similar goals include SqueezeNet (Iandola et al., 2016), CondenseNet (Huang et al., 2017) and ShiftNet (Wu et al., 2017).

MobileNetV1 (Howard et al., 2017) is primarily built on Depthwise Separable Convolutional Layers (Sifre and Mallat, 2014) with 3x3 filters, which are calculated to be 9 times more efficient than the standard convolution counterparts. MobileNet experiments demonstrate that a model using depthwise separable convolutional layers has approximately 14% the number of parameters compared to a standard model, while only reducing accuracy by 1%. MobileNetV1 contains 28 layers when counting depthwise and pointwise convolutions separately, all of which use ReLU except for the final softmax classifier. Dimensionality reduction is achieved by strided convolutions, as in InceptionV3 (Szegedy et al., 2016b). Batch normalization is used for each layer, and Average Pooling is used before the final fully connected layer (Lin et al., 2013), both of which helping with regularization. With fewer parameters, there is less reliance on additional regularization techniques as small networks are inherently less prone to overfitting.

The MobileNet team also developed two parameters which are designed to reduce the model size as required to suit the capacity of the device. The “width parameter” influences the number of filters in each layer, and typical settings are 1.0, 0.75, 0.5, and 0.25. This parameter is multiplied by the default number of filters, which reduces their number thus making the model “thinner”. The “resolution multiplier” parameter is applied to the input image resolution, making it smaller and subsequently affecting the dimensionality of layers downstream. These two parameters influence the size of the model and the number of parameters, thus reducing computation even further. However, fewer parameters come with a trade off in learning capacity. Experiments by the MobileNet team show that a “thinner” but deep model is 3% more accurate than a “wider” but shallow model, each containing a similar number of parameters, and that accuracy reduces gradually as the width reduces, until dropping significantly at a value of 0.25. Similarly, accuracy reduces with the resolution parameter, however with no obvious drop-off. Experiments using the ImageNet dataset, demonstrate that MobileNetV1 is nearly as accurate as VGG16, and more accurate than InceptionV1, while being 27 times and 2.5 times less computationally demanding, respectively (Howard et al., 2017).

MobileNetV2 introduces “inverted residuals and linear bottlenecks”, which aim to capture features in a compressed lower dimension “bottleneck” at lower computation (Sandler et al., 2018). This matrix is then expanded and inputted to a depthwise separable convolutional layer, before being compressed again and passed to the following module. Residual connections are applied between the bottleneck layers, which performs better than between expanded layers. When evaluated on the ImageNet dataset, MobileNetV2 achieves a Top-1 accuracy of 72.0% compared to 70.6% of MobileNetV1, with much fewer parameters.

MobileNetV3 comes in two versions, Large and Small, suitable for high and low resource devices respectively (Howard et al., 2019). These models were created with the help of Network Architecture Search (NAS) which discovered a more efficient architecture for these models, discussion of which is beyond the scope of this project (Tan et al., 2019; Yang et al., 2018). ReLU was substituted for the recently developed Swish function (Ramachandran et al., 2017). However, its use of a Sigmoid function comes with additional computational costs, which are alleviated by using a hard-Sigmoid function instead, thus improving latency. Experiments using the ImageNet dataset demonstrate that MobileNetV3-Large is 3.2 % more accurate than MobileNetV2, while having a 20% lower latency.

## Transfer Learning and Fine Tuning

Transfer learning is a technique used to reap the benefits of models that have been pretrained on large datasets such as ImageNet. These models have often learned detailed representations of features, and their learning is hoped to transfer onto a new target dataset in the form of the pre-trained weights (Pan and Yang, 2010). The principle underlying Transfer Learning is that low-level features extracted from data are similar and can be applied to other datasets (Ahmad et al., 2021). Pre-trained models are available from the Keras API (Keras, 2024a), including those that will be used in this project’s. The API has the option to include the model’s original top classifier layer as trained on the ImageNet dataset. However, if it is excluded, a new classifier can be appended to the model and trained on the target dataset with the number of units equaling the number of classes. When training a new top classifier layer, it is important to freeze the base model’s layers, rendering them untrainable. To not do so would cause large gradient updates during back propagation due to the top classifier’s randomly initialized weights, thus essentially destroying the information learned from pre-training. Another important factor is to set Batch Normalization layers to run in inference rather than training mode, as neglecting to do so allows their parameters to update during training and again destroy the previous learned statistics of the large dataset (Chollet, 2020).

After the top classifier layer has been trained on the target dataset, several of the base model’s later layers can be unfrozen and fine-tuned. The base model’s earlier layers contain information regarding low-level universal features such as edges and corners and so can be retained. However, the later layers contain information regarding more high-level features associated with the specific dataset (Figure 2) and so can be fine-tuned to allow their weights to adapt to new classes in the target dataset. A lower learning rate is used during fine-tuning to ensure that any updates to model weights are small so as not to destroy the previously learned information.

A collage of different images

Description automatically generated

Figure 2: Hierarchical features learned by CNN trained on ImageNet

Source: Yann LeCun presentation, image available at (Kalfas, 2018), adapted from (Zeiler and Fergus, 2014).

This project employs the transfer learning protocol described above on the pre-trained models. However, other transfer learning approaches also exist, such as Stepwise Transfer Learning which gradually unfreezes sequential layers when the loss has failed to improve and continues until all layers are unfrozen and trainable. This approach has previously been applied to plant disease datasets with good success (Ahmad et al., 2021),

## FieldPlant Dataset and Research

The FieldPlant dataset was collected to assist in the detection of disease in plant leaves (Moupojou et al., 2023). Research in this area has included CNN models (Adi et al., 2021), however, available datasets have proven unsuitable for various reasons, such as containing images taken under controlled laboratory conditions, with full lighting and uniform background (Hughes and Salathe, 2016) or lacking expert annotation or sufficient image quality (Singh et al., 2020). Models trained on these datasets performed well when tested on laboratory images, but less so on field images (Ahmad et al., 2021, 2021; Wang et al., 2022), likely due to the real-world influences of lighting, background and other variations (Ngugi et al., 2020).

The FieldPlant dataset consists of 5,170 expertly annotated in-field images across 27 disease classes and aims to provide a more suitable dataset for plant disease classifications. The details of data collection are discussed in Section 5.1.The FieldPlant team conducted an image classification experiment using the raw images. Four CNN models, all pretrained on the ImageNet dataset, were sourced using the Keras API. The authors indicate that they use the Multi-label CSV dataset available from RoboFlow (RoboFlow, 2023) with the goal of recognizing the diseases present in the images. However, their selection of parameters doesn’t appear to reflect this goal, as discussed in Section 4.5 below. Details supplied in their paper and sourced through contact with the authors reveal the use of the sparse categorical crossentropy loss function, softmax classifier function, and categorical accuracy, typically used in multi-class classification. This project makes several adjustments to these parameters to transform this into a suitable multi-label classification task. Subsequently, the results from both methodologies will be compared.

## Multi-class vs Multi-label Classification Methodology

The distinct differences between a multi-class and multi-label classification task form the primary basis for the experiments in this project. It is important to identify the correct methodology for the task at hand and select the appropriate parameters accordingly. The differences in how the various metrics are calculated is another important distinction between the two tasks, which may mean that they are not suitable for comparison between methodologies. A brief description of these metrics is included below, but the mathematical explanations are beyond the scope of this project.

### Multi-class

Multi-class classification involves attempting to predict the sample’s single true label. Labels are mutually exclusive, e.g. an image can contain either a cat or a dog, but not both. Labels are often in the form of a one-hot encoded vector, where the true label is assigned a 1 and all others a 0. A softmax function is used as the final classifier, because the resultant probability values are interdependent. Categorical crossentropy is used to calculate the loss at the index of the true label. Sparse categorical crossentropy is used in FieldPlant research, which is used with integer encoded labels rather than one-hot encoded. Categorical accuracy is used to measure the rate at which the true label matches its prediction. The index with the highest predicted probability is converted to a 1, and all others to a 0. Only the index of the true label is compared to its predicted counterpart, and if they match, the accuracy is 100%, and 0% otherwise. The final accuracy score is calculated as the average accuracy across all samples (TensorFlow, 2024d). The error rate metrics used by ILSVRC are the inverse of the accuracy metric, as it monitors when predictions do not match their label. To summarise, the parameters selected for a multi-class classification task are the softmax classification function, categorical crossentropy loss function and categorical accuracy metric (Lydia and Francis, 2020).

### Multi-label

In multi-label classification, an image can have more than one true label. For example, an image can contain a cat, a dog, or both. This distinction necessitates a different set of parameters (Lydia and Francis, 2020). Binary-encoded labels are often used, which is similar to one-hot-encoding, except more than one index can contain a 1 indicating the class’s presence (Sayak and Rakshit, 2020). A sigmoid function is most often used for this task, as the probability of any one class is calculated independent of any other (GlassboxMedicine, 2019). Binary crossentropy is used as the loss function, which is calculated for each of the labels independently, before all values being summed for the sample (TensorFlow, 2024e; V7Labs, 2023). Binary accuracy is used to calculate the rate at which the predicted labels match the true labels, and this too is calculated for each index. To summarise, the parameters most often selected for a multi-label classification task are the sigmoid classification function, binary crossentropy loss function and binary accuracy metric. Despite this commonly used methodology for multi-label classification, some research does still use parameters associated with multi-class classification (Mahajan et al., 2018; Moupojou, 2023), however their suitability is questioned in this project.

The main difference between the categorical metrics and their binary counterparts is that the binary metrics are calculated for each label index independently, rather than just the most probable index. For binary crossentropy, the loss values are summed for all classes in the sample, whereas in categorical crossentropy this is only calculated for the index of the true label. For binary accuracy, a probability threshold is set, often at 0.5, but this depends on the nature of the task. If the predicted probability exceeds this threshold, the index is assigned a 1, otherwise a 0 (TensorFlow, 2024f). As a result, there may be several classes assigned a predicted label of 1, as the probability of any one index when using Sigmoid is independent of any other. The vector of predicted labels is compared to the true labels, and the accuracy at each index is calculated and averaged over the sample. The final accuracy metric of the model is then calculated as the average accuracy over all samples. If a softmax classifier were instead used for a multi-label classification task with binary accuracy, the probabilities of the relevant labels may not exceed the set threshold as they are interdependent, thus all labels could be predicted as a 0. Similarly, if categorical accuracy were instead used, the index of the label with the highest probability would be transformed to a 1, but the remaining indices would be transformed to 0. This could incorrectly transform the second highest probability to a 0 despite having the potential to be a true label, and thus misclassify the label as absent.

As previously mentioned, the sigmoid function is the more appropriate classifier function for a multi-label task, as the presence of one label does not preclude the presence of any other. However, this independent approach also does not consider correlations between labels, e.g. sky and cloud would commonly appear together, however water and car hopefully much less so (Xiangyang Xue et al., 2011). As such, more advanced methods of multi-label classification have been investigated, such as the CNN-RNN framework (Wang et al., 2016), which utilizes a recurrent neural network to model the label dependencies, and combines this information with the CNN classification approach. This project, however, will focus on the comparison between the standard multi-label and multi-class classification methodologies as described above.

### Alternative Metrics

The binary accuracy metric can be misleading when used in for multi-label classification, especially with a significant class imbalance. For example, consider a vector which contains 2 true labels out of a possible 10 classes. If the model were to predict 0 for every class, then binary accuracy would classify 8 of the labels to be correct and 2 incorrect, resulting in a score of 80% accuracy. This, however, is a misleadingly high result considering the model has not predicted the presence of any true label (Thölke et al., 2023).

Due to the potentially misleading nature of the accuracy metric, other metrics may be used instead, such as Precision, Recall, and F1 Score. Precision is defined as the ratio of True Positive predictions to all positive predictions made. Hence, the greater number of False Positive predictions made, the lower this score. Precision indicates the model’s ability to detect only the relevant labels in the dataset. Recall is defined as the ratio of True Positive predictions to all positive labels present in the dataset. Hence, the greater the number of False Negative predictions made, the lower this score. Recall indicates the model’s ability to find all relevant labels in the dataset. F1 score is defined as the harmonic mean of Recall and Precision (Builtin.com, 2023). These three alternative metrics take into account the false negative and false positive predicted labels, which is not the case for the accuracy metric, thus making them a more representative metric for this kind of task, where it is important to correctly identify the presence of a disease (Thölke et al., 2023). F1 score is used as the primary evaluation metric in this project. The “F1 Score weighted-average” version is used, which calculates the average value weighted by label support (the number of samples per label), as it is more suitable for a dataset with a class imbalance.(TensorFlow, 2024g)

## Conclusion

This review reveals that the performance of CNN models has improved throughout their lifespan not only due to larger datasets and advancing technology, but also due to advances in algorithms and architecture. Transfer learning is an effective means of harnessing the information that pretrained models have learned from large datasets and applying it to a new problem area. The selection of image classification methodology and metrics is dependent on the task at hand, however, there appears to be some conflicting use of methodology in the research. The FieldPlant research is a positive step towards advancing datasets in plant disease detection, however their methodological choices seem to be inappropriate. This review has highlighted a gap in the literature in the problem area of plant disease detection regarding an explanation or investigation into the different methodologies of image classification, multi-class and multi-label, and their suitability for a multi-label dataset such as FieldPlant. This project aims to fill this gap by investigating the effect of these two methodologies on the ability of a CNN model to classify plant diseases.

# Methodology

This section discusses the steps implemented to achieve this project’s research objectives. The inherent bias of this project’s author is recognised as ever present, and decisions are made with awareness of such throughout this project with the intention to minimise bias. The areas for discussion in this chapter include: the research methodology employed, the preliminary setup prior to experiments, the findings of Exploratory Data Analysis, the steps involved in dataset preparation, and the experimental procedures.

## Research Methodology

A positivist research philosophy (Ryan, 2018) is adopted in this project, utilising a quantitative research approach and experimental research strategy, with the aim of objectively investigating the research questions as defined in Section 3.3. The conditions for inferring experimental causality are explored as follows (Hunt, 2010):

1. Demonstration of Concomitant Variation is attempted through observation of the change in performance metrics in relation to the change in classification methodology.
2. Temporal Sequence of events is demonstrated by the fact that the resulting metrics are produced after the experiments are conducted using the pre-defined methodology.
3. Theoretical Support for the association between the independent and dependent variables is provided in Section 4.5 of the literature review and in the discussion of each experiment.
4. Non-spurious Association is sought to be demonstrated through controlling extraneous variables as much as possible. The same coding procedures are utilised, only differing with regards to the model and classification methodology used. The data stratification function does not have a random seed option to ensure reproducibility of dataset splits, however the class proportions are reproducible through this function.

The FieldPlant paper provides details of their image collection procedures, however these cannot be verified. The dataset contains a total of 3 types of crops, and 27 disease classes, thus this dataset and this project’s results are only representative of these classes of crops and diseases. All images were taken in Cameroon across two of the five agro-ecological zones, and as such may not be representative of other regions or countries. The sampling strategy is not specified in their paper, however, the authors state that their goal was to collect as many diseases as possible, and this may indicate that they collected as many leaf samples as possible using a non-probabilistic convenience sampling strategy. Certain measures have been taken to reduce the risk of bias during the collection procedures. Images were taken in-field, collected at different times of the year, at different stages of plant growth, with a variety of lighting and complex background conditions, which allows for better representation of plant and disease diversity in the dataset. The image collection stage was supervised by plant pathologists, which strengthens the validity of the data. Image labelling was performed using a two-stage process, initially labelled by a data scientist then verified by a plant pathologist. However, the paper doesn’t specify if there is more than one person involved at each stage. If there is only one person, there is an increased risk of bias from that individual. If there is more than one person, then the paper doesn’t specify if there was discussion between pathologists to corroborate their verification, to further reduce the risk of bias. It is however, recognised that the images have been inspected and labelled by two independent persons, which does reduce the risk of bias somewhat.

A probabilistic stratification sampling strategy is used to randomly split the images of the FieldPlant dataset into training, validation, and testing datasets. Stratification is performed to enhance the representativeness of the datasets to the entire dataset through equally distributed proportions of disease classes. Experiment 1, investigating the effect of stratification, also uses a probabilistic simple random strategy to split the unstratified dataset.

Primary research is conducted by gathering and interpreting data regarding model performance and comparing between models. This primary data is gathered using a non-probabilistic convenience sampling strategy, as the results from each of the model evaluations are used to compare performance. This previously unavailable primary research is performed by the author of this project, for the purpose of this research project. In experiment 1, the independent variable is the choice of stratification. In experiment 2 and 4, the independent variable is the choice of classification methodology, however this requires changing several associated parameters (Table 1). In all experiments, the dependent variable is the performance metrics resulting from model evaluation on the test dataset, including Loss, Accuracy and F1 Score. F1 score is selected as the primary evaluation metric as it is deemed the most valid metric for comparison between methodologies, as described in Section 4.5.3.

Interviews with experts in the fields of both computer vision and plant disease had originally been planned as part of this project, however due to time constraints and interviewee availability these were not performed. This research project is conducted within a limited timeframe from July 1st to September 27th totaling 89 days, due to the constraints of the Master’s programme for which it is performed.

## Ethics

The “Ethics and Data Protection” document of the European Commission (Hayes and Kuyumdzheiva, 2021) is used to inform and guide this project in relation to ethical considerations.

This project does not handle any data that is considered personal or sensitive, or belonging to a vulnerable population. The secondary dataset is used in accordance with its Creative Commons Licence (Creativecommons.org, 2023). The results of this research project have several ethical implications. Developing a model that can impact the treatment of disease in plants can have secondary effects such as improving worldwide food supply (Baldi and La Porta, 2020), reducing economic loss (Agrios, 2005), and mitigating the negative impact of harmful pesticides on the environment (Simhadri and Kondaveeti, 2023). It would be morally imperative to make available this research, as not doing so may have ethical consequences for certain potentially vulnerable populations. Machine learning models are often used to make automated decisions; thus, it must be ensured that the results of any automated decisions have minimal negative repercussions for involved populations. For example, a false positive detection of disease may result in unnecessary use of environment damaging pesticides, or may require a farmer to destroy some of their crop to prevent infection spreading thus impacting their livelihood (Hayes and Kuyumdzheiva, 2021)

After having already written most of the code for this project and experimented with various parameters, the authors of the FieldPlant paper were contacted to verify certain details that were omitted from their paper. They subsequently provided a copy of their code (Moupojou, 2023) which was inspected to verify these parameters. This resulted in some changes being made to this project’s original code, which are discussed where relevant. This communication is being disclosed for reasons of ethical and academic integrity.

## Preliminary Setup

The following sections discuss details regarding the preparatory steps performed prior to model training and evaluation. The following sections discuss dataset sourcing and system configuration, the findings from Exploratory Data Analysis, and the steps involved in preparing the dataset prior to use in machine learning.

### Dataset and system configuration

The FieldPlant dataset (Moupojou et al., 2023), publicly available under the Creative Commons 4.0 license (Creativecommons.org, 2023), is downloaded from the RoboFlow website (RoboFlow, 2023). The experiments of this project closely relate to that of FieldPlant research, and is subsequently further expanded upon. Four models pretrained on ImageNet and available through the TensorFlow Keras API are used in the experiments: MobileNetV2, VGG16, InceptionV3, InceptionResNetV2. All experiments are performed on Windows Subsystem for Linux (WSL2), using a single dedicated CUDA enabled Nvidia GeForce RTX 3050 GPU with 4GB RAM. Mixed precision training is enabled in Keras, which uses lower precision floating point numbers when appropriate, to optimize computation without compromising precision (TensorFlow, 2024h).

### Exploratory Data Analysis

The folder downloaded from RoboFlow contains a folder of 5,156 jpeg images, despite the FieldPlant paper specifying 5,170. Information regarding the missing 14 samples was not discovered. The accompanying CSV file is loaded to a Pandas dataframe, which contains a corresponding 5,156 rows and 28 columns. The first column contains the image filename, and no duplicates were detected. Each filename in the dataframe is checked against the list of filenames in the image directory, to ensure that they match correctly. The remaining 27 columns contain the binary-encoded class labels, where 1 indicates the presence of disease and 0 the absence. The count of images containing one label is 4,967 (96.3% of the dataset), two labels is 188 (3.6%), and only one sample contains three labels (less than 1%). A list of diseases per sample is extracted from the binary-encoded labels, and the specific crop type is extracted from this list of diseases. The value counts of each disease and each crop is displayed in Figure 3, and the full tables are displayed Appendix 1 and Appendix 2, respectively. This disease classes are heavily imbalanced, as the most populated class constitutes 20.8% of the entire dataset. The three most populated classes constitute a combined 58.7%, with the other 24 classes making up the remaining 41.3%. There are 13 classes which each constitute less than 1% of the dataset. The two least populated classes each contain only one sample.

A graph of a disease and a class

Description automatically generated with medium confidence

Figure 3: Proportions of disease classes and crop classes in the FieldPlant dataset.

The image height and width dimensions are investigated and found to not be uniform. The median width and height are both 3,120 pixels, however the aspect ratio ranges from 0.45 to 2.26, indicating that some images are wider than they are long, and vice versa. The image dimensions will need to be transformed to a uniform size before being inputted to the CNN models.

A graph with a bar graph

Description automatically generated with medium confidence

Figure 4: Distribution of dimensions of Images in FieldPlant dataset.

### Dataset Preparation and Preprocessing

The full dataframe is randomly split at a ratio of 80% for the training set, and the remaining 20% is further split equally into validation and testing sets. The final proportions are 80-10-10%. The splits are executed using Scikit-Multilearn’s Iterative Stratification module which splits the datasets with an equal distribution of class proportions using a method more suitable for multi-label datasets as compared to others (Szymański and Kajdanowicz, 2017a), which will be discussed in Section 6.3. Prior to stratification, the two classes each containing only one sample are dropped, as they are not capable of being split. As such, the final stratified dataset contains 5,154 samples and 25 classes, with the least populated class now containing 6 samples. There is no class that is not present in the training dataset, meaning that the model will always see samples from every class. Dataset stratification is not used in the FieldPlant research, and as such, may have affected their results. Experiment 1 compares the use of stratified and unstratified datasets, and the unstratified dataset split is performed using Scikit-Learn’s train-test-split function (Scikit-Learn, 2024a). The remaining experiments use the stratified dataset to allow for fair evaluation of model performance.

A TensorFlow data pipeline (TensorFlow, 2024i) is used to convert the split dataframes into TensorFlow datasets, consisting of one tensor containing filenames, and another containing the binary-encoded labels. The filenames are used to load the relevant jpeg from the directory and convert it to a tensor of pixel values (TensorFlow, 2024j). The pixel tensor’s dimensions are resized to 299x299, chosen due to this being the largest default size of the four pre-trained models. Samples of images and their labels are manually verified to ensure validity, over several random iterations to reduce bias, by checking that the labels match the filename in the CSV, and that the filename matches the image in the directory. After verification, the filenames are dropped and the resulting dataset contains just the image pixel tensor, and the binary-encoded labels tensor. The datasets are configured to optimize performance in the data pipeline, by caching, batching and prefetching (TensorFlow, 2024k). Caching allows the dataset to be stored in memory, so it does not need to be loaded each time it is used. A batch size of 16 was used, as larger sizes resulted in unreliable performances of the computer system, occasionally crashing. Prefetching loads the next batch of data into memory as the current batch is being processed by the model, thus reducing wait times. Data augmentation is not used in this project, similar to the FieldPlant research.

Each of the models has specific requirements for preprocessing the image tensors, and this is available alongside the models as a preprocessing module from Keras. For MobileNetV2, images are resized to 224x224 pixels, and pixel values are scaled between -1 and 1. For VGG16, images are scaled to 224x224 and converted from RGB to BGR format without scaling. For both InceptionV3 and InceptionResNetV2, images are resized to 299x299 and scaled to between -1 and 1.

## Model Training and Evaluation

The following sections discuss the setup for model training, the transfer learning procedures including training a new top classifier and fine-tuning several base layers, and conclude with the procedure for hyperparameter tuning.

### Model Training Setup

An end-to-end code has been created so that each model and methodology can be specified and trained with little required change to the code. Certain parameters are specified at the start of the code, the “filtered”, “binary”, and “model\_name” parameters, which are used to set up the version of experiment to be conducted. The “filtered” parameter indicates whether to use the filtered stratified dataset if set to True, or the unstratified dataset if False. The “binary” parameter indicates whether to utilise multi-label classification methodology if set to True, and multi-class if False. The parameters selected for use in each of these methodologies are specified in Table 1. The multi-class parameters are similar to those used in FieldPlant experiments, except for their use of sparse categorical crossentropy. The F1 score metric is additionally included in this project’s experiments, as this metric gives a more representative indication of the model’s performance in a class-imbalanced classification task as compared to the accuracy metric. A probability threshold is set to 0.5 for multi-label metrics, which means that the predicted probability needs to exceed this threshold to be classified as a 1. There is no such threshold set for multi-class metrics, thus the predicted label is determined by the index with the highest probability. The four selected models will each be trained using both methodologies.

|  |  |  |
| --- | --- | --- |
|  | **Multi-class** | **Multi-label** |
| **Classifier activation function** | Softmax | Sigmoid |
| **Loss function** | Categorical Crossentropy | Binary Crossentropy |
| **Accuracy** | Categorical Accuracy | Binary Accuracy |
| **F1/Acc. probability threshold** | None, chooses argmax | 0.5 |

Table 1: Parameters for Multi-class and Multi-label classification methodologies

The “model\_name” parameter is used to load the base model from the Keras API along with its required pre-processing module. The top classifier layer of the base model is excluded, the pretrained weights are specified as those from “imagenet”, and the default input size is selected. When the top classifier is excluded, this also removes several other top layers. As such, these additional layers were observed in the models which include the top classifier, and a custom Sequential “pre\_classifier\_layers” module is created so that these layers can be appended to the base model once again before the new top classifier. The additional layers specified for each model in their respective research papers appear to differ from what is available on Keras, so the layers used in this project are those available through Keras. After the base model, VGG16 uses a Flattening layer, followed by two Dense layers of 4096 units using ReLU. In this project, however, the Flattening layer is changed to GlobalAveragePooling2D, and the Dense layers’ units changed to 1024, as the system was not able to handle the extra computation reliably. The other three models from Keras only include an additional GlobalAveragePooling2D layer after the base model. The full procedure is similar for each model and is explained in detail below. In summary, for each model, the base model is frozen, and a new top classifier is trained. Following this, the base model is unfrozen from a specified layer and fine tuning is performed. The dropout rate, optimizer learning rate, and the fine-tuning start layer are specified as in both the FieldPlant paper and the code provided by its authors.

### Training New Top Classifier

The base model’s “trainable” parameter is set to False, to ensure that it is frozen while training the new top classifier. The relevant parameters for classification function, loss function and metrics are obtained as determined by the “binary” parameter. The specified dataset is loaded as determined by the “filtered” parameter. The pixels tensor is resized to the default size of the specified model, before being passed to the preprocessing module, and subsequently to the base model. When calling the base model, the “training” parameter is set to False, which ensures that the Batch Normalization layers are in inference mode. The data is next passed to the custom “pre\_classifier\_layers” module specific to each model, followed by dropout with a rate of 0.2 The data is finally passed to the classifier, whose number of units is selected for the number of classes in the dataset, with multi-class classification using 27 units, and multi-label using 25, as 2 classes are dropped prior to stratification. The activation function, loss function and metrics are pre-selected depending on the classification methodology and are specified in Table 1. The model is compiled using the Adam optimizer with a learning rate of 0.001. The EarlyStopping callback is used to stop training when the validation loss has not improved over the previous 10 epochs, returning the model with the previous best weights. The ModelCheckpoint callback saves the model to file only when the validation loss improves beyond its previous best value, so the best model gets saved. Before training, the model’s configuration is printed, to verify the selection of parameters and that the correct layers are frozen. The model is then trained using the training dataset created earlier, for a maximum of 100 epochs, however EarlyStopping may be activated before this. The validation dataset is used to monitor the model’s best metrics. After training, the training history is stored in a Pandas dataframe. The best epoch is identified as the epoch with the lowest validation loss, and all epochs after this point are removed from the dataframe so that the fine-tuning history can be appended later.

### Fine Tuning and Evaluation

Several of the base model’s layers are unfrozen, by setting their “trainable” parameter to True. The specific layer numbers after which the model is unfrozen are as follows: MobileNet from 120, VGG16 from 14, InceptionV3 from 172, and InceptionResNetV2 from 516. The model is compiled again using the same parameters as before, except the learning rate is reduced by a factor of 10 to 0.0001. The EarlyStopping callback is used again, this time with the threshold set to the value of the best epoch’s validation loss from the previous stage. This means that the callback will not reset unless this threshold is crossed, indicating that the model’s performance has improved. The ModelCheckpoint callback is used again, with the threshold set to the best validation loss, meaning that the saved model will not be overwritten unless the performance surpasses its previous best value. The model configuration is again printed to verify the selection of parameters and the now unfrozen trainable layers for fine-tuning. The model is trained again, with the “initial epoch” parameter set to be one greater than the best epoch number from the previous stage. When completed, the fine-tuning history is concatenated to the previous history dataframe as a continuous record of training epochs and saved to a CSV file. The best model saved by ModelCheckpoint is reloaded and evaluated on the test dataset, which returns the test metrics.

The processes described in the above two sections are repeated for each of the four models and for each of the 2 classification methodologies, resulting in eight sets of experimental data. These testing data are saved to a CSV file, along with the model’s name, classification methodology used, and the training and validation metrics from the best epoch.

### Hyperparameter Tuning

Following analysis of the results of the fine-tuning process, the best performing model as defined by the lowest test loss, MobileNetV2, is selected for Hyperparameter Tuning. The stratified dataset was recreated using the same procedures as in Section 5.3.3. The base code for creating the full MobileNetV2 model was recreated in the form of a function, so that it can be included as the hypermodel parameter in a Keras Tuner.

The variables used for hyperparameter tuning are the number of units in the fully connected layer preceding the classifier layer with options for 0, 256, and 512; the probability rate of Dropout with options for 0.2 and 0.5; and the choice of classifier activation function with options for either sigmoid or softmax. If the number of units was not set to 0, then an extra fully connected layer was included before the final classifier layer. The “use\_bias” parameter of this layer is set to False, and it is followed by Batch Normalization and then ReLU, before the output is passed to the final classifier. The choice of classifier activation function determines whether this is a multi-class or multi-label classification task, and the loss function and metrics are subsequently determined by this selection. The learning rate was fixed at 0.0001 rather than included as an optional hyperparameter. This may prolong the training time per model but may improve the final loss achieved, and also reduce the total number of models to be trained as the total number of hyperparameter combinations is less.

The RandomSearch Tuner, available from the Keras-Tuner API (Keras, 2024b), is used which randomly selects a combination of hyperparameters for each trial. A total of 12 trials are performed, which includes all possible combinations of the parameters. A random seed was set for reproducibility, and the overwrite function was set to False, so that in the event of the system crashing, the procedure could be restarted from its previously recorded logs. The EarlyStopping callback is used as previously described. After completing the RandomSearch, the MobileNetV2 model is trained again using the best identified hyperparameters. The best number of epochs is then identified as the epoch with the best validation loss. Having discovered the best hyperparameters and best number of epochs, the training and validation datasets are concatenated, and the model trained on this combination for the best number of epochs. The fully trained best model is then saved to file, to be reloaded and evaluated on the hold out test set.

# Project Evaluation

This section serves as a critical evaluation of the steps undertaken throughout this project. Several aspects have been discussed in the Methodology, and the following sections will further explore the decision-making processes and challenges faced. Each section provides justification for the decisions made, as well as discussing viable alternative options where relevant.

## Selection of Image Dimensions and Batch Size

The choice of image dimensions when initially loading the dataset was partly dependent on the availability of memory. However, the decision to select 299x299 pixels is due to this being the largest default size for any of the pre-trained models (Szegedy et al., 2016a). Initially, the dataset had been loaded with a size of 180x180, however this meant that for all models, the dimensions needed to be upscaled using interpolation (Keras, 2024c). Instead, it was decided to use a larger dimension initially and downsize as needed, to maintain original pixels in the matrix. These dimensions aren’t selected specifically for any model performance reason, however, previous research has shown that lower resolutions may improve performance, as fewer parameters reduce overfitting (Sabottke and Spieler, 2020).

After several iterations of experimenting with various other batch sizes, the value of 16 was chosen primarily for reasons related to system performance, as any greater size resulted in unreliable performance and occasional crashing. Research does indicate an ideal range of batch sizes between 16 to 64. The authors recommend beginning with 32 and adjusting as needed, suggesting to “decrease it for accuracy or increase it for efficiency” (Lin, 2022).

## Selection of Classification Methodology

One of the original objectives of this project was to assess the performance of pre-trained models compared to a custom-built model. On initial examination of the dataset, it was mistakenly assumed to consist of one-hot-encoded rather than binary-encoded labels, hence assumed to be a multi-class dataset. After several rounds of experimentation with unsatisfactory results, further exploration of the dataset was performed. The total sum of all label columns was calculated and found to be greater than the number of samples in the dataset, indicating the presence of more than one label per sample. The sum of labels in each row was then calculated and the value counts inspected, demonstrating that there were samples with 1, 2 and 3 labels. This discovery changed the methodology of this task to be multi-label classification. As the FieldPlant experiments had been performed with parameters more commonly associated with multi-class classification, it was then decided that this project would primarily investigate the difference between these two methodologies and their associated parameters.

## Methods to Address Class Imbalance

Following unsatisfactory validation results during experimentation, it was decided to perform dataset stratification for a more equally proportioned distribution of disease classes. Initially, Scikit-Learn’s train-test-split module (Scikit-Learn, 2024a) was trialed, however it was discovered that traditional single label stratification methods such as this are not suitable for multi-label datasets (Sechidis et al., 2011; Szymański and Kajdanowicz, 2017b) as they “fail to provide balanced data set divisions which prevents classifiers from generalizing information”(Scikit-Multilearn, 2017). The problem that occurs when using Scikit-Learn for this procedure is that new classes are created from the combination of others. For example, if a sample contains class “A” and class “B”, this will be converted into a single class “A, B” rather than be treated as two separate classes. This method is not suitable for multi-label classification, as the model seeks to identify the diseases present in the samples, not just if any disease is present. Instead, Scikit-Multilearn’s Iterative Stratification function is used, as it results in a well-balanced distribution of class samples more suitable for a multi-label classification task (Scikit-Multilearn, 2017).

Experiment 1 compares the results obtained by one model trained and evaluated on the stratified dataset versus on the unstratified version. The remainder of the experiments use the stratified dataset. The Iterative Stratification method does not appear to have a random seed parameter, so when recreating the dataset splits for Hyperparameter Tuning, the datasets may not be strictly equivalent to those of the previous experiments, possibly leading to inconsistent and biased results. However, given that the stratification process should provide equal proportions of class distributions, the datasets should be suitably comparable.

Other options are available to manage dataset class imbalances, however these methods are not employed, to maintain a comparable training procedure to FieldPlant experiments. Over-sampling of the underrepresented classes, or under-sampling of the overrepresented classes are alternative methods that could have been employed to create more favourable proportions. Research comparing several methods to address class imbalances in CNN models found that oversampling emerged as the best method (Buda et al., 2018). Standard data augmentation would have provided more training samples of the under-represented classes, but so too for the overrepresented classes, thus not reliably changing the class imbalance. Expansive Over Sampling (EOS) (Dablain et al., 2023) is a data augmentation technique that creates synthetic training instances that are similar to the minority classes, and is found to improve model accuracy with less computational cost and shorter training times compared to other standard augmentation methods such as Synthetic Minority Oversampling Technique (SMOTE) and General Adversarial Network (GAN) based oversampling (Shorten and Khoshgoftaar, 2019).

## Evaluation Metrics

Having made changes to the classification methodology and dataset stratification as described in the previous two sections, the models began to return more satisfactory results. However, it was noted that the value for Binary Accuracy was a lot higher than expected, at approximately 98%. On further research, it was discovered that Categorical Accuracy and Binary Accuracy differ in their methods of calculation, as explained in Section 4.5.3, which results in Binary Accuracy being very high. As a result, accuracy is considered a misleading metric for this task, and other more appropriate metrics are explored, namely F1 score. This metric is more robust when it comes to multi-label classification, as it punishes incorrect predictions more harshly, likely leading to a lower value, but one that is a better representation of performance. The F1 score used the “weighted average” calculation method which accounts for class imbalance (TensorFlow, 2024g).

An additional method of evaluation that would have provided valuable insight into the performance of this model is the use of a Confusion Matrix. However, due to the nature of the multi-label dataset, this proved difficult. Several options were explored however they each proved inappropriate as they either did not correctly display the desired outcome, or did not accept the appropriate input. For example, the Scikit-Learn “multi-label confusion matrix” function returns an array of 2x2 confusion matrices, each representing the results for only a single class (Scikit-Learn, 2024b). The returned result proved too cumbersome and not sufficiently informative to use for evaluation. The TensorFlow math.confusion\_matrix function (TensorFlow, 2024l) was also trialed, however this function only accepts single integer labels as input, making the binary-encoded labels incompatible. A confusion matrix would provide insight into the classes for which the model performed poorly, as observed through the higher values for False Negative and False Positive results. To circumvent this challenge, the weighted F1 score metric was calculated per class when evaluating the best hyperparameter-tuned model in Experiment 4.

## Evaluation of Hyperparameter Tuning Procedure

The optimizer’s learning rate was initially selected as an optional hyperparameter, however was subsequently excluded in favour of reducing the total number of combinations and search time. The hyperparameter tuning procedure does not perform the fine-tuning steps as used in Experiment 2, due to the time constraints of this project. Also, the identified hyperparameters were discovered to be the best when searching only with the top classifier layer being trainable, and as such the best hyperparameters may not be the same if applied to fine-tuning.

Other hyperparameter tuning algorithms are available, including Bayesian Optimization and Hyperband algorithms. Bayesian Optimization aims to reduce the time taken to find a good combination, and research demonstrates that it outperforms standard algorithms such as RandomSearch (Eggensperger et al., 2013; Snoek et al., 2015; Thornton et al., 2012). This algorithm selects hyperparameters based on the performances of previous selections, in order to probabilistically improve its next selection (Wang, 2022). The Hyperband tuner is described as an “infinite-arm bandit” algorithm which makes a random selection of hyperparameter combinations, and employs EarlyStopping and adaptive resource allocation (Li et al., 2018). This allows the tuner to eliminate certain combinations of hyperparameters that seem to be performing poorly during training, and to subsequently allocate more resources to those performing better. These resources can include the number of training epochs, and the number of training samples and features used. Research demonstrates that Hyperband is 5 to 30 times faster than Bayesian Optimization (Li et al., 2018). The Hyperband tuner had initially been trialed to perform the hyperparameter search procedure in this project. However, this tuner was only comparing the models using the first two training epochs, without extending the training further beyond. As such, it was decided that this approach was not sufficiently representative of the model’s performance and was substituted for RandomSearch.

## Strengths, Weaknesses and Limitations

This project includes several features which provide additional strength beyond the methodology of FieldPlant research on which it is based. The decision to stratify the dataset, the reframing of this task as a multi-label classification task, and the inclusion of alternative metrics are all strengths of this project, as they allow for a more suitable evaluation of model performance on this multi-label dataset.

The primary focus of this project is to investigate the effect of two distinct classification methodologies on model performance. Given the limited time available to investigate a single dataset and a small number of models, the conclusions of this research are preliminary. Had there been more time and resources available, statistical tests could have been conducted to investigate this relationship statistically. As will be seen, the experiments fail to reach all four conditions for inferring causality. Given more available time, discussion with experts in this area would be sought to understand why these experiments failed to achieve the four criteria.

Despite not being the primary focus, there are several areas of strength and weakness for discussion related to improving model performance. The best performing model from Experiment 3 is subsequently selected for hyperparameter tuning, to further improve its performance. The concatenation of the training and validation datasets allows for a larger and more diverse dataset on which to train this best model. Thes two factors are strengths of this project and should assist in training a better performing model. The exclusion of data augmentation and additional methods for addressing class imbalance beyond stratification are weaknesses of this project. Data augmentation, specifically the Expansive Over Sampling (EOS) approach would assist in improving the models’ ability to generalize to new unseen test data. Additionally, the use of class weights during model training would allow underrepresented classes to have a greater influence on the loss function, and thus influence the final weights of the model to better classify these underrepresented classes.

# Results

The experiments conducted as part of this project aim to achieve the research objectives as specified in Section 3.3. The results from the experiments are obtained on the test dataset, unless otherwise specified. Note that if the type of Loss, Accuracy or F1 Score is unspecified, the methods of calculation are that which correspond to the specified classification methodology, as described in Section 4.5.3. Results are compared to FieldPlant multi-class classification results where relevant, and only the categorical accuracy metric is made available. The evaluation of these results is reserved for Section 8.

## Experiment 1: Effect of Dataset Stratification

The multi-class classification methodology is conducted using a single model, MobileNetV2, to compare the effect of dataset stratification on model performance. The model is trained and evaluated on both the unstratified dataset, similar to FieldPlant experiments, and the stratified dataset as described earlier in this project.

|  |  |  |
| --- | --- | --- |
|  | **Categorical Acc.** | **F1 Score** |
| **FieldPlant** | 82.90% | - |
| **Unstratified** | 82.36% | 81.37% |
| **Stratified** | 78.40% | 79.10% |

Table 2: Results of Experiment 1 - Effect of Dataset Stratification

The model using the unstratified dataset performs similarly to that of the FieldPlant experiments, with regards to Categorical Accuracy. The model using the stratified dataset demonstrates a lower accuracy and F1 score as compared to both the unstratified dataset from this project and the FieldPlant experiments.

## Experiment 2: Comparing Classification Methodology

Using the stratified dataset only, both the multi-label and multi-class methodologies are conducted using the four pretrained models, and the results compared within each model. In all cases, both the loss and accuracy metrics of the multi-label models outperform the multi-class counterparts. This observation reverses for F1 score, as in all cases, the multi-class model outperforms the multi-label. All multi-class models in this project, except for MobileNetV2, outperform the results from FieldPlant.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Task** | **Loss** | **Accuracy** | **F1 Score** |
| **InceptionResNetV2** | **Multi-Label** | 0.05 | 98.47% | 79.67% |
|  | **Multi-Class** | 0.63 | 85.58% | 83.49% |
|  | **FieldPlant** | - | 81.81% | - |
| **InceptionV3** | **Multi-Label** | 0.04 | 98.67% | 80.94% |
|  | **Multi-Class** | 0.61 | 86.63% | 85.37% |
|  | **FieldPlant** | - | 82.54% | - |
| **MobileNetV2** | **Multi-Label** | 0.04 | 98.48% | 77.41% |
|  | **Multi-Class** | 0.67 | 78.40% | 78.10% |
|  | **FieldPlant** | - | 82.90% | - |
| **VGG16** | **Multi-Label** | 0.05 | 98.32% | 78.95% |
|  | **Multi-Class** | 0.59 | 85.60% | 83.61% |
|  | **FieldPlant** | - | 80.54% | - |

Table 3: Results of Experiment 2 - Comparing Classification Methodology.

*Results also compared to FieldPlant research, which uses multi-class classification methodology.*

## Experiment 3: Identify Best Performing Model

Table 4 displays the results of the eight model and methodology combinations in order of best to worst performances, across each of the included metrics. For Loss, results are in ascending order, and for Accuracy and F1 score, results are in descending order.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Model Loss** | | **Model Accuracy** | | **Model F1 score** | |
| **1** | Mobilenet  \_multilabel | 0.042 | inceptionv3  \_multilabel | 98.67% | inceptionv3  \_multiclass | 85.37% |
| **2** | inceptionv3  \_multilabel | 0.044 | Mobilenet  \_multilabel | 98.48% | vgg16  \_multiclass | 83.61% |
| **3** | vgg16  \_multilabel | 0.050 | inceptionresnetv2  \_multilabel | 98.47% | inceptionresnetv2  \_multiclass | 83.49% |
| **4** | inceptionresnetv2  \_multilabel | 0.052 | vgg16  \_multilabel | 98.32% | inceptionv3  \_multilabel | 80.94% |
| **5** | vgg16  \_multiclass | 0.594 | inceptionv3  \_multiclass | 86.63% | inceptionresnetv2  \_multilabel | 79.67% |
| **6** | inceptionv3  \_multiclass | 0.607 | vgg16  \_multiclass | 85.6% | vgg16  \_multilabel | 78.95% |
| **7** | inceptionresnetv2  \_multiclass | 0.634 | inceptionresnetv2  \_multiclass | 85.58% | Mobilenet  \_multiclass | 78.1% |
| **8** | Mobilenet  \_multiclass | 0.674 | Mobilenet  \_multiclass | 78.4% | Mobilenet  \_multilabel | 77.41% |

Table 4: Results of Experiment 3 - Identify Best Performing Model

The best performing models according to each metric all differ. The best models for Loss, Accuracy, and F1 Score are MobileNetV2\_multilabel, InceptionV3\_multilabel, and InceptionV3\_multiclass respectively. Despite MobileNetV2 performing the best for Loss, it performed the worst for F1 Score. The best model as determined by F1 score is the InceptionV3 model using multi-class methodology, with a score of 85.37%.

## Experiment 4: Conduct Hyperparameter Tuning

The best performing model from Experiment 3 as determined by the lowest test loss, is selected for Hyperparameter Tuning. MobileNetV2 model is used with the RandomSearch tuner to discover the best combination of hyperparameters and number of training epochs. The available hyperparameter options are specified in Section 5.4.4. Table 5 displays the best hyperparameter combination and best number of epochs returned by RandomSearch. The best methodology is found to be multi-label classification through use of the sigmoid classifier function.

|  |  |
| --- | --- |
| **Activation** | Sigmoid |
| **Dropout Rate** | 0.2 |
| **Units** | 256 |
| **Epochs** | 47 |

Table 5: Experiment 4 - Hyperparameters discovered during RandomSearch.

The graphs of training history when searching for the best epoch number are displayed in Figure 5. The blue and orange lines represent the training and validation metrics respectively, and the green vertical line indicates the best epoch as defined by the lowest validation loss. These graphs demonstrate model overfitting, as the gap between training and validation widens as training continues, however the metrics do not seem to perform significantly worse with further increasing epochs.

A graph of a number of data

Description automatically generated with medium confidence

Figure 5: Training metrics during Hyperparameter tuning searching for best epoch.

The model is subsequently retrained with these best hyperparameters, using the combined training and validation datasets and for the best number of epochs. The results displayed in Table 6 are obtained when evaluating this model on the test dataset. These results demonstrate that the hyperparameter-tuned MobileNetV2 model outperforms the MobileNetV2 model with fine-tuning transfer learning from Experiment 3, with an F1 scores of 81.56% compared to 77.41%.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset** | **Loss** | **Binary Accuracy** | **F1 Score** |
| **Train** | 0.0134 | 99.57% | 94.70% |
| **Test** | 0.0394 | 98.54% | 81.56% |

Table 6: Results of Experiment 4 - Hyperparameter Tuning

The final training history is displayed in Figure 6. After the last training epoch, the model is saved and evaluated on the test dataset, and these results are indicated by the red line and red “X” marker, signifying overtraining as the training data performs significantly better.

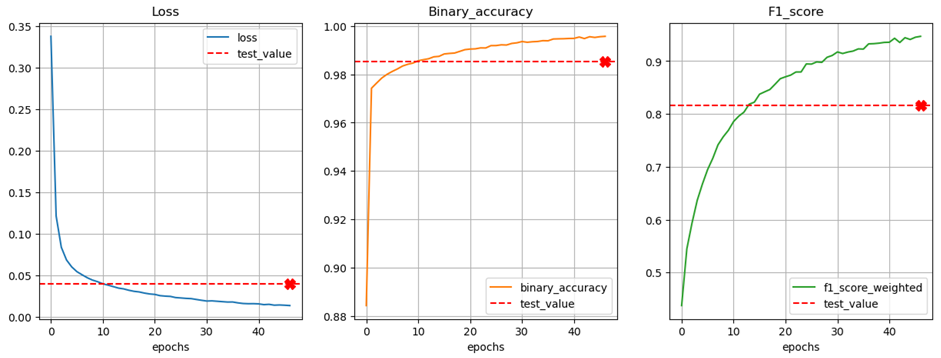


Figure 6: Experiment 4 - Best model training history

The red line and “X” indicate the best value achieved when using the model saved after the final training epoch.

The F1 scores per disease class are displayed in Table 7. The left side of the table contains all classes with ten or more instances of support, and the right side contains the classes with less than 10. Most disease classes on the left side score greater than 50%, however both “Cassava Brown Leaf Spot” and “Tomato Healthy” classes score less than 50%. There are 4 disease classes on the right side of the table with a score of 0%, “Corn Chlorotic Leaf Spot”, “Tomato Leaf Mosaic Virus”, “Cassava Bacterial Blight”, “Corn Violet Decoloration”.

The model made a total of 532 label predictions on the test dataset, which contains a total of 519 samples and 538 true labels. This underprediction rate may also be relevant to the standard models of the previous experiments and will be further discussed in the evaluation of Experiment 4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Support** | **F1** |  | **Support** | **F1** |
| **Corn leaf blight** | 111 | 0.87 | **Tomato leaf yellow virus** | 7 | 0.67 |
| **Cassava Mosaic** | 108 | 0.92 | **Cassava Root Rot** | 7 | 0.75 |
| **Tomato Brown Spots** | 95 | 0.88 | **Corn rust leaf** | 5 | 0.60 |
| **Tomato blight leaf** | 36 | 0.77 | **Corn Mildew** | 4 | 0.67 |
| **Corn Yellowing** | 31 | 0.89 | **Corn Gray leaf spot** | 3 | 1.00 |
| **Cassava Healthy** | 27 | 0.69 | **Corn Chlorotic Leaf Spot** | 3 | 0.00 |
| **Cassa Brown Leaf Spot** | 19 | 0.38 | **Tomato Leaf Mosaic Virus** | 2 | 0.00 |
| **Corn Streak** | 18 | 0.80 | **Cassava Bacterial Blight** | 2 | 0.00 |
| **Corn Brown Spots** | 17 | 0.90 | **Corn Yellow Spots** | 2 | 0.67 |
| **Tomato healthy** | 14 | 0.48 | **Corn Insects Damages** | 2 | 0.67 |
| **Corn Stripe** | 11 | 0.96 | **Corn Smut** | 2 | 0.50 |
| **Corn Healthy** | 10 | 0.82 | **Corn Violet Decoloration** | 1 | 0.00 |
| **-** | - | - | **Corn Purple Discoloration** | 1 | 1.00 |

Table 7: F1 Score per class, evaluating hyperparameter tuned model on test dataset.

# Discussion of Results

The results of each experiment in the previous section are discussed below, along with the extent to which the four conditions for inferring causality can be demonstrated where relevant, Each experiment relates to its numerical counterpart in Section 3.3.2 Research Objectives, and a conclusion is presented for each relevant hypothesis.

## Experiment 1: Effect of Dataset Stratification

Using multi-class methodology, the results obtained on the stratified dataset are worse than on the unstratified dataset, and as compared to the FieldPlant experiment. The better performance on the unstratified datasets may be due to the distribution of classes, as it is possible that all rare disease samples were assigned to the training set, thus there being none present in the testing set to be incorrectly predicted. If this were the case, the stratified datasets would be less biased as the test set is guaranteed to contain a portion of these rare samples, which the model is likely to misclassify having been trained on only a very limited number of samples. This would result in a lower accuracy and F1 score but would provide a more valid representation of model performance.

This experiment demonstrates concomitant variation and temporal sequence as changing the independent variable, dataset stratification, leads to a resulting change of the dependent variable, model performance. However, the direction of this relationship is opposite to what is hypothesized. There is theoretical support in the literature for the benefit of dataset stratification in multi-label classification tasks (Sechidis et al., 2011; Szymański and Kajdanowicz, 2017b). However, it is not possible to demonstrate a non-spurious association between the two variables, as the results may be dependent on the randomization process during splitting of the unstratified dataset as the class proportions may vary with repeated trials. Despite there being a potential relationship between the two variables, albeit in the opposite direction to that hypothesized, the conditions for inferring causality are not met for hypothesis one.

## Experiment 2: Comparing Classification Methodology

In all cases, the multi-label models outperform their multi-class counterparts for loss and accuracy. However, this is likely due to the different methods used to calculate the “categorical” and “binary” metrics, which is more likely to result in higher values for the “binary” metrics. F1 Score is hoped to provide a metric that is applicable to and more comparable across both methodologies. Using this metric, it appears that the multi-class models outperform their multi-label counterparts. This may be influenced by the underprediction rate of the multi-label models, which will be discussed in the evaluation of Experiment 4.

This experiment demonstrates concomitant variation and temporal sequence as changing the independent variable, classification methodology, leads to a resulting change of the dependent variable, model performance. For Loss and Accuracy metrics, the results are better for multi-label classification as hypothesized, however, this relationship reverses for F1 Score. There is theoretical support in the literature for the use of this multi-label methodology for multi-label classification (Lydia and Francis, 2020). However, it is not possible to demonstrate a non-spurious association between the two variables, as the results for binary accuracy and F1 score depend on their probability threshold. This is arbitrarily set to 0.5 in this project, and a different value would likely lead to different results. Despite there being a potential relationship between the independent and dependent variables, albeit with F1 having the opposite direction to that hypothesized, the conditions for inferring causality are not met for hypothesis two.

## Experiment 3: Identify Best Performing Model

Results reveal that the “best performing model” is not consistent across all metrics. Surprisingly, the best performing model for loss is the worst performing model for F1 Score. This is an interesting insight, as it highlights the importance of choosing the correct metrics for the problem at hand. As well as outperforming their multi-label counterparts, as discovered in Experiment 2, all but one multi-class models rank in the top 4 with regards to F1 Score. This discrepancy again will be discussed in the evaluation of Experiment 4. This experiment did not attempt to infer causality, but rather to identify the best performing model, and it is hypothesized that this would be a multi-label classification model. The hypothesis is true if using Loss or Accuracy as the deciding metric, as in this project, but False if using F1 score.

## Experiment 4: Conduct Hyperparameter Tuning

The model returned by the hyperparameter tuning process performs better than its standard non-tuned counterpart. This is interesting, considering that the standard model is trained with additional base model fine-tuning. However, the hyperparameter-tuned model has been trained on a larger combined dataset, allowing for a greater number of rare samples to be seen, indicating the preference for a larger dataset over enhanced model architecture. Some overfitting occurs during this training process, which could be remedied with the inclusion of data augmentation.

This experiment demonstrates concomitant variation and temporal sequence as selection of the independent variable, the application of hyperparameter tuning, leads to a resulting change of the dependent variable, model performance. The hyperparameter tuned model performs better for all metrics, as hypothesized. There is theoretical support in the literature for the application of hyperparameter tuning and its additional benefit to model performance (Li et al., 2018; Wang, 2022). However, it is not possible to demonstrate a non-spurious association between the two variables, as the standard model uses fine-tuning of several layers of the base model, whereas the hyperparameter-tuned model did not, only training the top classifier. In addition to this, the independent variable was whether hyperparameter-tuning was used or not, however the decision to use it is accompanied by changes in several other associated parameters which also differ to the standard model. Despite there being a potential relationship between the independent and dependent variables, the conditions for inferring causality are not met for hypothesis four.

The hyperparameter tuning procedure selected the multi-label methodology. However, the algorithm was set to monitor the validation loss metric, which is most often significantly lower when using binary crossentropy compared to categorical, so this selection is not surprising. It may be the case that if the best performing model as determined by F1 Score was used instead (InceptionV3 using multi-class methodology), that an even better performance could be achieved.

This model made fewer predictions than there are labels in the test dataset. This underprediction rate would mean that the undiscovered labels would always be considered incorrect, and hence contribute to a lower F1 score. This may also be the case for the standard models as alluded to in the discussions of the previous experiments however it was not inspected. It would be interesting to investigate the effect of a lower probability threshold for Binary Accuracy and F1 Score metrics. This would likely allow for more predictions to be made, however potentially at the expense of more False Positives. A likely reason for the multi-class models to outperform their multi-label counterparts for accuracy and F1 score, is that they are guaranteed to return a label for each sample, because the predicted label is determined by the argmax of the probability vector. This is not the case in the binary equivalents, as the predicted probability instead needs to exceed a predefined threshold. If the model is not confident so that the highest probability is below threshold, it will be labelled as a 0. The difference between the binary and classification accuracies and their true labels is demonstrated Appendix 1.

The F1 scores per class (Table 7) reveal the diseases that the model had difficulty classifying. Disease classes with a greater number of support samples tend to score higher. However, this is not the case for all, as both “Cassava Brown Leaf Spot” and “Tomato Healthy” classes score less than 50% and given that other diseases with less support have a higher F1 score, this may indicate that these two classes are inherently more challenging to classify and may require more training samples. The 4 classes on the right side of the table with a score of 0% all have a significantly low number of support samples. The poor result for these classes is likely indicative of a sparsity of training data, rather than these diseases being inherently more challenging.

# Conclusion and Future Work

This project aims to achieve several research objectives, as described in Section 3.3. The primary focus of this research is to compare the results of two classification methodologies, multi-class and multi-label. The initial hypothesis is that the multi-label methodology would perform better on the FieldPlant dataset, as it is a multi-label dataset. However, the results of Experiment 2 demonstrate that the multi-label models outperform their multi-class counterparts, and Experiment 3 demonstrates that three of the four multi-class models outperform all the multi-label models. These results would appear to refute the hypothesis. However, it may be that the metrics used to compare these methodologies are not suitable. It would still appear that the multi-class methodology, despite performing better, is fundamentally inappropriate for this multi-label dataset. The main concerns being the choice of the softmax classifier function, and the methods of calculating the chosen loss and label for accuracy. It is usual in the case of softmax, that the index with the highest probability is transformed to 1 and all others are transformed to 0. This method would not be suitable for a multi-label dataset, as a second label present in any sample would always be classified as a 0. There may be other reasons as to why the multi-class methods performed better, such as the guarantee to always choose one label per sample, however the fundamental methodology seems inappropriate for a multi-label dataset.

Future research in this area should investigate the use of alternative metrics that would allow for a more suitable comparison across methodologies. However, the poor performance by the multi-label methodologies may just fundamentally be a lack of sufficient data, especially in the rare disease classes, which may be alleviated by gathering more data or using data augmentation. The conclusions of the FieldPlant paper were that “the existing models are not sufficiently accurate”, however this may be due to the methodology used rather than the model architecture, and this research project argues that the application of a more appropriate methodology should provide a better insight into the models’ performance.

# Appendix

1. Predicted labels compared to the true labels.

Below is a demonstration of the difference between the true labels and label predictions when using binary accuracy and categorical accuracy methodologies. There are three examples provided for each methodology. The predicted labels are on the top row, and the true labels are on the bottom. In all six examples, the vector of true labels contains more than one class.

Figure 7 displays examples of predicted labels when using multi-label classification methodology. In the top example, the model predicts the presence of one true label and the absence of the other. In example 2, the model correctly predicts the presence of both true labels. In example 3, again, the model predicts the presence of one true label and the absence of the other. This demonstrates how the multi-label classification methodology could result in fewer predicted labels than there are true labels.

A row of black circles with white text

Description automatically generated

Figure 7: Example of class predictions in multi-label classification.

Figure 8 displays examples of predicted labels when using multi-class classification methodology. In all examples, the model predicts the presence of one true label but does not predict the other. This demonstrates how the multi-class classification method is guaranteed to predict a label for every sample; however, it cannot predict more than one label per sample.

A number of labels on a white background

Description automatically generated with medium confidence

Figure 8: Example of class predictions in multi-class classification.

2. Value Counts of Diseases and Percent of Total Dataset.

|  |  |  |
| --- | --- | --- |
| **Disease Class** | **Count** | **Percent** |
| Tomato bacterial wilt | 1 | 0.02% |
| Corn Charcoal | 1 | 0.02% |
| Corn Violet Decoloration | 6 | 0.11% |
| Corn Purple Discoloration | 8 | 0.15% |
| Corn Smut | 13 | 0.24% |
| Tomato leaf mosaic virus | 13 | 0.24% |
| Corn Yellow Spots | 16 | 0.30% |
| Corn Insects Damages | 17 | 0.32% |
| Cassava Bacterial Blight | 22 | 0.41% |
| Corn Chlorotic Leaf Spot | 26 | 0.49% |
| Corn Gray leaf spot | 30 | 0.56% |
| Corn Mildew | 41 | 0.77% |
| Corn rust leaf | 46 | 0.86% |
| Tomato leaf yellow virus | 66 | 1.24% |
| Cassava Root Rot | 74 | 1.38% |
| Corn Stripe | 104 | 1.95% |
| Corn Healthy | 105 | 1.96% |
| Tomato healthy | 141 | 2.64% |
| Corn Brown Spots | 172 | 3.22% |
| Corn Streak | 182 | 3.40% |
| Cassava Brown Leaf Spot | 192 | 3.59% |
| Cassava Healthy | 264 | 4.94% |
| Corn Yellowing | 307 | 5.74% |
| Tomato blight leaf | 359 | 6.72% |
| Tomato Brown Spots | 952 | 17.81% |
| Cassava Mosaic | 1077 | 20.15% |
| Corn leaf blight | 1111 | 20.78% |

3. Value counts of Crops and Percentage of Total Dataset

|  |  |  |
| --- | --- | --- |
|  | **Count** | **Percent** |
| **Tomato** | 1403 | 27.21% |
| **Cassava** | 1603 | 31.08% |
| **Corn** | 2151 | 41.71% |

# References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. s, Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Zheng, X., 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. https://doi.org/10.48550/arXiv.1603.04467

Adi, M., Singh, A.K., Reddy A, H., Kumar, Y., Challa, V.R., Rana, P., Mittal, U., 2021. An Overview on Plant Disease Detection Algorithm Using Deep Learning, in: 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM). Presented at the 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), pp. 305–309. https://doi.org/10.1109/ICIEM51511.2021.9445336

Agrios, G., 2005. Plant pathology. Elsevier Academic Press, Burlington, Ma. USA, pp. 79–103.

Ahmad, M., Abdullah, M., Moon, H., Han, D., 2021. Plant Disease Detection in Imbalanced Datasets Using Efficient Convolutional Neural Networks With Stepwise Transfer Learning. IEEE Access 9, 140565–140580. https://doi.org/10.1109/ACCESS.2021.3119655

Baldi, P., La Porta, N., 2020. Molecular Approaches for Low-Cost Point-of-Care Pathogen Detection in Agriculture and Forestry. Front. Plant Sci. 11, 570862. https://doi.org/10.3389/fpls.2020.570862

Buda, M., Maki, A., Mazurowski, M.A., 2018. A systematic study of the class imbalance problem in convolutional neural networks. Neural Netw. 106, 249–259. https://doi.org/10.1016/j.neunet.2018.07.011

Builtin.com, 2023. Precision and Recall in Classification Models [WWW Document]. Built In. URL https://builtin.com/data-science/precision-and-recall (accessed 9.18.24).

Chollet, F., 2020. Keras documentation: Transfer learning & fine-tuning [WWW Document]. URL https://keras.io/guides/transfer\_learning/ (accessed 9.11.24).

Creativecommons.org, 2023. Deed - Attribution 4.0 International - Creative Commons [WWW Document]. URL https://creativecommons.org/licenses/by/4.0/ (accessed 9.9.24).

Dablain, D., Bellinger, C., Krawczyk, B., Chawla, N., 2023. Efficient Augmentation for Imbalanced Deep Learning. https://doi.org/10.1109/ICDE55515.2023.00114

Demirkaya, A., Chen, J., Oymak, S., 2020. Exploring the Role of Loss Functions in Multiclass Classification, in: 2020 54th Annual Conference on Information Sciences and Systems (CISS). Presented at the 2020 54th Annual Conference on Information Sciences and Systems (CISS), pp. 1–5. https://doi.org/10.1109/CISS48834.2020.1570627167

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Li, F.-F., 2009. ImageNet: a Large-Scale Hierarchical Image Database, IEEE Conference on Computer Vision and Pattern Recognition. https://doi.org/10.1109/CVPR.2009.5206848

Duchi, J., Hazan, E., Singer, Y., 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. J Mach Learn Res 12, 2121–2159.

Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H.H., Leyton-Brown, K., 2013. Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters.

EITCA Academy, 2023. What is the role of the fully connected layer in a CNN? URL https://eitca.org/artificial-intelligence/eitc-ai-dlptfk-deep-learning-with-python-tensorflow-and-keras/convolutional-neural-networks-cnn/introduction-to-convolutional-neural-networks-cnn/examination-review-introduction-to-convolutional-neural-networks-cnn/what-is-the-role-of-the-fully-connected-layer-in-a-cnn/ (accessed 9.10.24).

Everingham, M., Eslami, S.M.A., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A., 2015. The Pascal Visual Object Classes Challenge: A Retrospective. Int. J. Comput. Vis. 111, 98–136. https://doi.org/10.1007/s11263-014-0733-5

Fei-Fei, L., Fergus, R., Perona, P., 2006. One-shot learning of object categories. IEEE Trans. Pattern Anal. Mach. Intell. 28, 594–611. https://doi.org/10.1109/TPAMI.2006.79

GlassboxMedicine, 2019. Multi-label vs. Multi-class Classification: Sigmoid vs. Softmax. Glass Box. URL https://glassboxmedicine.com/2019/05/26/classification-sigmoid-vs-softmax/ (accessed 9.11.24).

Google Colab, 2024. colab.google [WWW Document]. colab.google. URL http://0.0.0.0:8080/ (accessed 9.17.24).

Hayes, B., Kuyumdzheiva, A., 2021. Ethics and data protection.

He, K., Zhang, X., Ren, S., Sun, J., 2015. Deep Residual Learning for Image Recognition. https://doi.org/10.48550/arXiv.1512.03385

Hinton, G., Tieleman, T., 2012. Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude. COURSERA Neural Netw. Mach. Learn. 26–31.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q., Adam, H., 2019. Searching for MobileNetV3. https://doi.org/10.48550/arXiv.1905.02244

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. https://doi.org/10.48550/arXiv.1704.04861

Huang, G., Liu, S., van der Maaten, L., Weinberger, K., 2017. CondenseNet: An Efficient DenseNet using Learned Group Convolutions. https://doi.org/10.48550/arXiv.1711.09224

Hughes, D.P., Salathe, M., 2016. An open access repository of images on plant health to enable the development of mobile disease diagnostics. https://doi.org/10.48550/arXiv.1511.08060

Hunt, S.D., 2010. Marketing theory: foundations, controversy, strategy, resource-advantage theory. M.E. Sharpe, Armonk, N.Y.

Iandola, F., Moskewicz, M., Ashraf, K., Han, S., Dally, W., Keutzer, K., 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and textless1MB model size.

Ioffe, S., Szegedy, C., 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://doi.org/10.48550/arXiv.1502.03167

Kalfas, I., 2018. A Connectionist approach to Deep Learning. Medium. URL https://medium.com/@kalfasyan/my-connectionist-approach-to-deep-learning-part-1-a9b190356295 (accessed 9.16.24).

Keras, 2024a. Keras documentation: Keras Applications [WWW Document]. URL https://keras.io/api/applications/ (accessed 9.11.24).

Keras, 2024b. Keras documentation: RandomSearch Tuner [WWW Document]. URL https://keras.io/api/keras\_tuner/tuners/random/ (accessed 9.18.24).

Keras, 2024c. Keras documentation: Resizing layer [WWW Document]. URL https://keras.io/api/layers/preprocessing\_layers/image\_preprocessing/resizing/ (accessed 9.19.24).

Kingma, D.P., Ba, J., 2014. Adam: A Method for Stochastic Optimization [WWW Document]. arXiv.org. URL https://arxiv.org/abs/1412.6980v9 (accessed 9.10.24).

Krizhevsky, A., 2012. Learning Multiple Layers of Features from Tiny Images. Univ. Tor.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 84–90. https://doi.org/10.1145/3065386

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436–444. https://doi.org/10.1038/nature14539

LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D., 1989. Backpropagation Applied to Handwritten Zip Code Recognition. Neural Comput. 1, 541–551. https://doi.org/10.1162/neco.1989.1.4.541

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86, 2278–2324. https://doi.org/10.1109/5.726791

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A., 2018. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. J. Mach. Learn. Res. 18, 1–52.

Lin, M., Chen, Q., Yan, S., 2013. Network In Network.

Lin, R., 2022. Analysis on the Selection of the Appropriate Batch Size in CNN Neural Network. https://doi.org/10.1109/MLKE55170.2022.00026

Lydia, A.A., Francis, F.S., 2020. Multi-Label Classification using Deep Convolutional Neural Network. 2020 Int. Conf. Innov. Trends Inf. Technol. ICITIIT 1–6. https://doi.org/10.1109/ICITIIT49094.2020.9071539

Mahajan, D., Girshick, R., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., van der Maaten, L., 2018. Exploring the Limits of Weakly Supervised Pretraining, in: Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part II. Springer-Verlag, Berlin, Heidelberg, pp. 185–201. https://doi.org/10.1007/978-3-030-01216-8\_12

Mikołajczyk-Bareła, A., Grochowski, M., 2018. Data augmentation for improving deep learning in image classification problem. https://doi.org/10.1109/IIPHDW.2018.8388338

Moupojou, E., 2023. Python code for FieldPlant experiments.

Moupojou, E., Tagne, A., Retraint, F., Tadonkemwa, A., Wilfried, D., Tapamo, H., Nkenlifack, M., 2023. FieldPlant: A Dataset of Field Plant Images for Plant Disease Detection and Classification With Deep Learning. IEEE Access PP, 1–1. https://doi.org/10.1109/ACCESS.2023.3263042

Nagendram, S., Singh, A., Harish Babu, G., Joshi, R., Pande, S.D., Ahammad, S.K.H., Dhabliya, D., Bisht, A., 2023. Stochastic gradient descent optimisation for convolutional neural network for medical image segmentation. Open Life Sci. 18, 20220665. https://doi.org/10.1515/biol-2022-0665

Nair, V., Hinton, G., 2010. Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair, Proceedings of ICML.

Ngugi, L., Abdelwahab, M., Abo-Zahhad, M., 2020. Tomato leaf segmentation algorithms for mobile phone applications using deep learning. Comput. Electron. Agric. 178, 105788. https://doi.org/10.1016/j.compag.2020.105788

Pan, S., Yang, Q., 2010. A Survey on Transfer Learning. Knowl. Data Eng. IEEE Trans. On 22, 1345–1359. https://doi.org/10.1109/TKDE.2009.191

Raitoharju, J., 2022. Chapter 3 - Convolutional neural networks, in: Iosifidis, A., Tefas, A. (Eds.), Deep Learning for Robot Perception and Cognition. Academic Press, pp. 35–69. https://doi.org/10.1016/B978-0-32-385787-1.00008-7

Ramachandran, P., Zoph, B., Le, Q.V., 2017. Searching for Activation Functions.

Rawat, W., Wang, Z., 2017. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. Neural Comput. 29, 2352–2449. https://doi.org/10.1162/neco\_a\_00990

RoboFlow, 2023. FieldPlant Object Detection Dataset by Plant Disease Detection [WWW Document]. Roboflow. URL https://universe.roboflow.com/plant-disease-detection/fieldplant (accessed 9.11.24).

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A., Fei-Fei, L., 2014. ImageNet Large Scale Visual Recognition Challenge. Int. J. Comput. Vis. 115. https://doi.org/10.1007/s11263-015-0816-y

Ryan, G., 2018. Introduction to positivism, interpretivism and critical theory. Nurse Res. 25, 14–20. https://doi.org/10.7748/nr.2018.e1466

Sabottke, C.F., Spieler, B.M., 2020. The Effect of Image Resolution on Deep Learning in Radiography. Radiol. Artif. Intell. 2, e190015. https://doi.org/10.1148/ryai.2019190015

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C., 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. 2018 IEEECVF Conf. Comput. Vis. Pattern Recognit. 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

Sayak, P., Rakshit, S., 2020. Keras documentation: Large-scale multi-label text classification [WWW Document]. URL https://keras.io/examples/nlp/multi\_label\_classification/ (accessed 9.11.24).

Scikit-Learn, 2024a. train\_test\_split [WWW Document]. Scikit-Learn. URL https://scikit-learn/stable/modules/generated/sklearn.model\_selection.train\_test\_split.html (accessed 9.12.24).

Scikit-Learn, 2024b. multilabel\_confusion\_matrix [WWW Document]. Scikit-Learn. URL https://scikit-learn/stable/modules/generated/sklearn.metrics.multilabel\_confusion\_matrix.html (accessed 9.18.24).

Scikit-Multilearn, 2017. scikit-multilearn | Multi-label classification package for python [WWW Document]. URL http://scikit.ml/stratification.html (accessed 9.12.24).

Sechidis, K., Tsoumakas, G., Vlahavas, I., 2011. On the Stratification of Multi-label Data. https://doi.org/10.1007/978-3-642-23808-6\_10

Shorten, C., Khoshgoftaar, T.M., 2019. A survey on Image Data Augmentation for Deep Learning. J. Big Data 6, 60. https://doi.org/10.1186/s40537-019-0197-0

Sifre, L., Mallat, S., 2014. Rigid-Motion Scattering for Texture Classification.

Simhadri, C.G., Kondaveeti, H.K., 2023. Automatic Recognition of Rice Leaf Diseases Using Transfer Learning. Agronomy 13, 961. https://doi.org/10.3390/agronomy13040961

Simonyan, K., Zisserman, A., 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. https://doi.org/10.48550/arXiv.1409.1556

Singh, D., Jain, N., Jain, P., Kayal, P., Kumawat, S., Batra, N., 2020. PlantDoc: A Dataset for Visual Plant Disease Detection. https://doi.org/10.1145/3371158.3371196

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, Md.M.A., Prabhat, M., Adams, R., 2015. Scalable Bayesian Optimization Using Deep Neural Networks. Statistics.

Soergel, D., 1998. WordNet. An Electronic Lexical Database.

Sorokin, A., Forsyth, D., 2008. Utility data annotation with Amazon Mechanical Turk, Urbana. https://doi.org/10.1109/CVPRW.2008.4562953

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. J. Mach. Learn. Res. 15, 1929–1958.

Sutskever, I., Martens, J., Dahl, G., Hinton, G., 2013. On the importance of initialization and momentum in deep learning. 30th Int. Conf. Mach. Learn. ICML 2013 1139–1147.

Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A., 2016a. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. AAAI Conf. Artif. Intell. 31. https://doi.org/10.1609/aaai.v31i1.11231

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2015. Going deeper with convolutions, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/CVPR.2015.7298594

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z., 2016b. Rethinking the Inception Architecture for Computer Vision. https://doi.org/10.1109/CVPR.2016.308

Szymański, P., Kajdanowicz, T., 2017a. A scikit-based Python environment for performing multi-label classification. J. Mach. Learn. Res. 20.

Szymański, P., Kajdanowicz, T., 2017b. A Network Perspective on Stratification of Multi-Label Data. https://doi.org/10.48550/arXiv.1704.08756

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q., 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. https://doi.org/10.1109/CVPR.2019.00293

TensorFlow, 2024a. tf.keras.activations.softmax | TensorFlow v2.16.1 [WWW Document]. TensorFlow. URL https://www.tensorflow.org/api\_docs/python/tf/keras/activations/softmax (accessed 9.10.24).

TensorFlow, 2024b. tf.keras.activations.sigmoid | TensorFlow v2.16.1 [WWW Document]. TensorFlow. URL https://www.tensorflow.org/api\_docs/python/tf/keras/activations/sigmoid (accessed 9.10.24).

TensorFlow, 2024c. tf.keras.losses.CategoricalCrossentropy | TensorFlow v2.16.1 [WWW Document]. TensorFlow. URL https://www.tensorflow.org/api\_docs/python/tf/keras/losses/CategoricalCrossentropy (accessed 9.14.24).

TensorFlow, 2024d. tf.keras.metrics.CategoricalAccuracy | TensorFlow v2.16.1 [WWW Document]. TensorFlow. URL https://www.tensorflow.org/api\_docs/python/tf/keras/metrics/CategoricalAccuracy (accessed 9.11.24).

TensorFlow, 2024e. tf.keras.losses.BinaryCrossentropy | TensorFlow v2.16.1 [WWW Document]. TensorFlow. URL https://www.tensorflow.org/api\_docs/python/tf/keras/losses/BinaryCrossentropy (accessed 9.14.24).

TensorFlow, 2024f. tf.keras.metrics.BinaryAccuracy | TensorFlow v2.16.1 [WWW Document]. TensorFlow. URL https://www.tensorflow.org/api\_docs/python/tf/keras/metrics/BinaryAccuracy (accessed 9.11.24).

TensorFlow, 2024g. tf.keras.metrics.F1Score | TensorFlow v2.16.1 [WWW Document]. TensorFlow. URL https://www.tensorflow.org/api\_docs/python/tf/keras/metrics/F1Score (accessed 9.14.24).

TensorFlow, 2024h. Mixed precision | TensorFlow Core [WWW Document]. TensorFlow. URL https://www.tensorflow.org/guide/mixed\_precision (accessed 9.12.24).

TensorFlow, 2024i. tf.data: Build TensorFlow input pipelines  |  TensorFlow Core [WWW Document]. URL https://www.tensorflow.org/guide/data (accessed 9.12.24).

TensorFlow, 2024j. Load and preprocess images | TensorFlow Core [WWW Document]. URL https://www.tensorflow.org/tutorials/load\_data/images (accessed 9.12.24).

TensorFlow, 2024k. Better performance with the tf.data API | TensorFlow Core [WWW Document]. TensorFlow. URL https://www.tensorflow.org/guide/data\_performance (accessed 9.12.24).

TensorFlow, 2024l. tf.math.confusion\_matrix | TensorFlow v2.16.1 [WWW Document]. TensorFlow. URL https://www.tensorflow.org/api\_docs/python/tf/math/confusion\_matrix (accessed 9.18.24).

Thölke, P., Mantilla-Ramos, Y.-J., Abdelhedi, H., Maschke, C., Dehgan, A., Harel, Y., Kemtur, A., Mekki Berrada, L., Sahraoui, M., Young, T., Bellemare Pépin, A., El Khantour, C., Landry, M., Pascarella, A., Hadid, V., Combrisson, E., O’Byrne, J., Jerbi, K., 2023. Class imbalance should not throw you off balance: Choosing the right classifiers and performance metrics for brain decoding with imbalanced data. NeuroImage 277, 120253. https://doi.org/10.1016/j.neuroimage.2023.120253

Thornton, C., Hutter, F., Hoos, H., Leyton-Brown, K., 2012. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. KDD. https://doi.org/10.1145/2487575.2487629

Tibshirani, R., 1996. Regression Shrinkage and Selection via the Lasso. J. R. Stat. Soc. Ser. B Methodol. 58, 267–288.

Tsoumakas, G., Katakis, I., 2009. Multi-Label Classification: An Overview. Int. J. Data Warehous. Min. 3, 1–13. https://doi.org/10.4018/jdwm.2007070101

V7Labs, 2023. Cross Entropy Loss: Intro, Applications, Code [WWW Document]. URL https://www.v7labs.com/blog/cross-entropy-loss-guide, https://www.v7labs.com/blog/cross-entropy-loss-guide (accessed 9.14.24).

Wang, D., Wang, J., Ren, Z., Li, W., 2022. DHBP: A dual-stream hierarchical bilinear pooling model for plant disease multi-task classification. Comput. Electron. Agric. 195, 106788. https://doi.org/10.1016/j.compag.2022.106788

Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., Xu, W., 2016. CNN-RNN: A Unified Framework for Multi-label Image Classification, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Las Vegas, NV, USA, pp. 2285–2294. https://doi.org/10.1109/CVPR.2016.251

Wang, W., 2022. Bayesian Optimization Concept Explained in Layman Terms [WWW Document]. Medium. URL https://towardsdatascience.com/bayesian-optimization-concept-explained-in-layman-terms-1d2bcdeaf12f (accessed 9.18.24).

Wu, B., Wan, A., Yue, X., Jin, P., Zhao, S., Golmant, N., Gholaminejad, A., Gonzalez, J., Keutzer, K., 2017. Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions.

Xiangyang Xue, Wei Zhang, Jie Zhang, Bin Wu, Jianping Fan, Yao Lu, 2011. Correlative multi-label multi-instance image annotation. 2011 Int. Conf. Comput. Vis. 651–658. https://doi.org/10.1109/ICCV.2011.6126300

Yang, T.-J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., Adam, H., 2018. NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part X. pp. 289–304. https://doi.org/10.1007/978-3-030-01249-6\_18

Zeiler, M.D., Fergus, R., 2014. Visualizing and Understanding Convolutional Networks, in: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (Eds.), Computer Vision – ECCV 2014. Springer International Publishing, Cham, pp. 818–833. https://doi.org/10.1007/978-3-319-10590-1\_53