

**CX4041: Machine Learning Group Project Report**

**Topic: Plant Seedlings Classification**

**Ranking:**

**Members:**

Aaron Tay

Leow Guan Wei

Lee Wei Zheng Benedict

Ray Myat Theingar Cho

**Submission Date:**

24 April 2022

**Table of Contents**

[**Team Structure**](#_heading=h.phbtaa1om64j) **4**

[**Problem**](#_heading=h.sk8t8zmhtjxh) **4**

[Introduction](#_heading=h.ldkgxhfghaia) 4

[Objective](#_heading=h.1gwimffwjkkg) 4

[Kaggle Evaluation Metrics](#_heading=h.wd5nomwu0krv) 5

[**Exploratory Data Analysis**](#_heading=h.ow4b5ifmzdeb) **5**

[Overview](#_heading=h.yaunr6j6ai8o) 5

[Data Distribution](#_heading=h.5xh8c74a24zk) 6

[Size distribution](#_heading=h.6gaqfuqgq2jv) 7

[Histogram Properties](#_heading=h.qwn0jwtrpzxg) 8

[**Data Preprocessing**](#_heading=h.4pw45464kjw3) **8**

[Data Augmentation](#_heading=h.u1aixhflgszr) 9

[Image Resizing](#_heading=h.8y9s6vycopw4) 10

[Image Masking](#_heading=h.14c23jic3rjv) 10

[**Methodologies: Traditional**](#_heading=h.nk8ygo47uzoe) **11**

[**Method 1: K-Nearest Neighbours(k-NN)**](#_heading=h.8p9j67y2jb5g) 11

[Overview](#_heading=h.iqb7l3aucfap) 11

[Experiments](#_heading=h.1h33pzebvki) 11

[Results](#_heading=h.4bnt9yntk2i3) 11

[**Method 2: Support Vector Machines (SVM)**](#_heading=h.utg4f828q4id) 12

[Overview](#_heading=h.jhe8qq26uiv) 12

[Experiments](#_heading=h.iy4acq3ddgbr) 12

[Results](#_heading=h.av0g41791dyk) 12

[**Methodologies: Convolutional Neural Network (CNN)**](#_heading=h.m4c79j4qe3ww) **13**

[Overview of CNN](#_heading=h.vjqwr1p0c5rw) 13

[**Method 1: Xception**](#_heading=h.lncgrrsanrrl) 13

[Overview](#_heading=h.q0imiwa1mc7u) 13

[Experiments](#_heading=h.1ufkmt4xlpld) 14

[Results](#_heading=h.ghjry65uu15m) 15

[**Method 2: EfficientNet**](#_heading=h.a5erjh15kj5f) **16**

[Overview](#_heading=h.kz580yjpcta1) 16

[Experiments](#_heading=h.j33wayvnlhtp) 17

[Results](#_heading=h.7fnlbl13crn1) 18

[**Method 3: InceptionResNetV2**](#_heading=h.lf2d26qy56wf) 18

[Overview](#_heading=h.fyj51he6icd9) 18

[Experiments](#_heading=h.uvizcv9p7snz) 19

[Results](#_heading=h.ce3pmzta7pq) 20

[**Method 4: Ensemble Learning Using CNN models**](#_heading=h.45qjvfoal4ne) 20

[Overview](#_heading=h.oj4317jhw964) 20

[Experiments](#_heading=h.188hlpxlv15) 21

[Results](#_heading=h.9i253jy0nep8) 21

[**Summary**](#_heading=h.h6sw8z7zqtck) **21**

[Leaderboard Scores](#_heading=h.7i1xf1d3g45p) 21

[**Solution Novelty**](#_heading=h.ss65z0buk9q2) **21**

[Transfer Learning](#_heading=h.vtjbx0swn7qz) 22

[Implementing our own Fully Connected Layers](#_heading=h.lje8ixbcr9fa) 22

[Ensemble Learning](#_heading=h.wh66bcc2bfph) 22

[Early Stopping Callback](#_heading=h.sz6cfxik55j3) 23

[Model Checkpoint Callback](#_heading=h.79qjtr8mkmn6) 23

[Learning Rate Scheduler Callback](#_heading=h.29wjfl737qvo) 23

[Dropouts](#_heading=h.rfkhb5i4nyw9) 24

[Level 2 Regularisation](#_heading=h.t9iv386875ui) 24

[**Conclusion**](#_heading=h.7ki8sxqagmnm) **24**

[**References**](#_heading=h.7d2s6qqcixy7) **26**

# 

# Team Structure

The table below summarises the contributions made by the members of this group.

| **Name** | **Contribution** |
| --- | --- |
| Aaron Tay  U1921247H | EfficientNetB7, Xception |
| Leow Guan Wei  U1921951F | Exploratory Data Analysis, K Nearest Neighbours, Video Presentation Editing |
| Lee Wei Zheng Benedict  U1920560L | Data Augmentation, Xception, InceptionResNetV2 |
| Ray Myat Theingar Cho  U1920868G | Image Masking, Support Vector Machine |

# Problem

## Introduction

The presence of weeds amongst growing crops poses a significant challenge for agricultural industries and farmers. These weeds compete with the growing crops for water, sunlight and soil nutrients, which may result in a poorly grown harvest. The roots of stems of mature weeds may become entangled with that of the crops, and they risk being uprooted together or damaged if the weeds are removed late.

Early detection and removal of these weeds when they are still young seedlings is therefore essential. Due to the large amounts of agricultural farmland in use, it is not practical to manually check for and remove weeds, and this task is often assigned to automated robots. As such, there is a need to develop and train neural network models capable of classifying the images captured by the robot’s camera during its course of operation in the field. The predicted plant species based on this outcome could then be used by the robot to determine exactly where the weeds are to uproot.

## Objective

The aim of this project is to determine an optimal machine learning strategy for classifying different categories of plant seedling images, with the goal of being able to detect and categorise plant and weed seedlings to a high level of accuracy. The training data should also be sufficiently prepared to remove noise or features that are not of interest. Different approaches to tackle this problem are attempted, ranging from simple classification algorithms to deep or convolutional neural networks.

## Kaggle Evaluation Metrics

The evaluation score on the Kaggle website ranks a submission of test entries with an accuracy value ranging from 0.0 to 1.0, where a score of 1.0 indicates a perfect classification accuracy of 100%.

Precision can be defined as a measure of the exactness of the classifier, referring to the percentage of the classification results which are relevant. It is defined as:

***no.\_of\_true\_positives / (no.\_of\_true\_positives + no.\_of\_false\_positives)***

Recall can be defined as a measure of the completeness of the classifier, referring to the percentage of relevant results which were correctly classified. It is defined as:

***no.\_of\_true\_positives / (no.\_of\_true\_positives + no.\_of\_false\_negatives)***

The F1 score is a harmonic average of the precision and recall of a classifier. It is used to evaluate the classification accuracy of the test dataset class predictions submitted to Kaggle. [1] The equation for obtaining the F1 score is given as:

***2 \* ( (precision \* recall) / (precision + recall) )***

# Exploratory Data Analysis

## Overview

The plant species datasets provided by Kaggle consisted of a training dataset of labelled images, and a test dataset of unlabelled images.

Table 1 lists the classes of plant species in the training dataset provided by Kaggle, as well as the original number of image samples per class. There are a total of 4750 images across 12 categories of plant seedlings provided by Kaggle as training dataset. An additional 794 unlabelled images were also provided as the test dataset. Table 1 lists the categories and the number of images per category for the training dataset.

| **Seedling Type** | **Image Quantity** |
| --- | --- |
| Black Grass | 263 |
| Charlock | 390 |
| Cleavers | 287 |
| Common Chickweed | 611 |
| Common Wheat | 221 |
| Fat Hen | 475 |
| Loose Silky Bent | 654 |
| Maize | 221 |
| Scentless Mayweed | 516 |
| Shepherds Purse | 231 |
| Small Flowered Cranesbill | 496 |
| Sugar Beet | 385 |

Table 1: Categories of plant seedlings and number of training images.

## Data Distribution

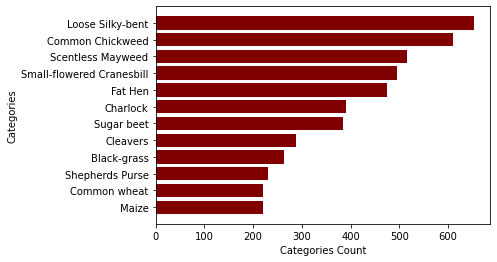


Figure 1: Data Distribution of Plant Seedlings Categories

The distribution of the category of plant seedling species dataset can be seen in Figure 1, where the different categories have a different distribution. It is non-uniform, with the category ‘Loose Silky-bent’ having the highest image count, and ‘Maize’ having a lower image count. We can also observe that the category with the highest count has a count almost thrice of the lowest. This non-uniformity may result in poor model performance due to an imbalance in training for the different classes.

## Size distribution

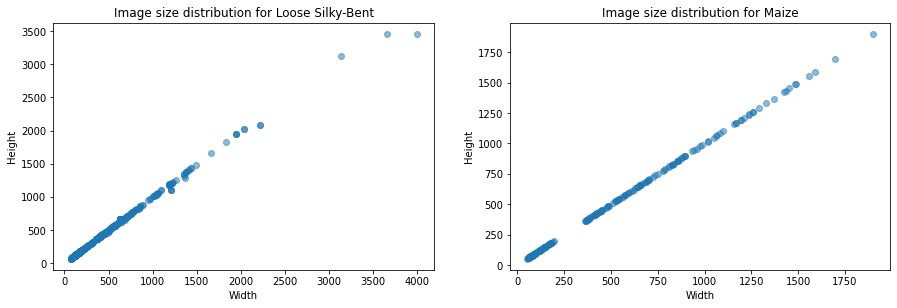


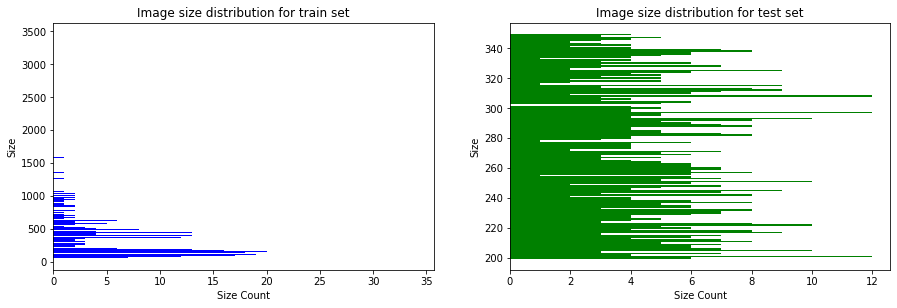
Figure 2: Image size distribution for 2 different categories 

Figure 3: Image size distribution for train and test sets

The size distribution of the images are of varying sizes, even in each category. In both the train and test set, the images also vary in sizes. This can be observed in figures 2 and 3. The sizes range from a size of 49x49 pixels to a size of 3457x3457 pixels. Thus, we will need to resize all images as uniform sizes for training.

## Histogram Properties

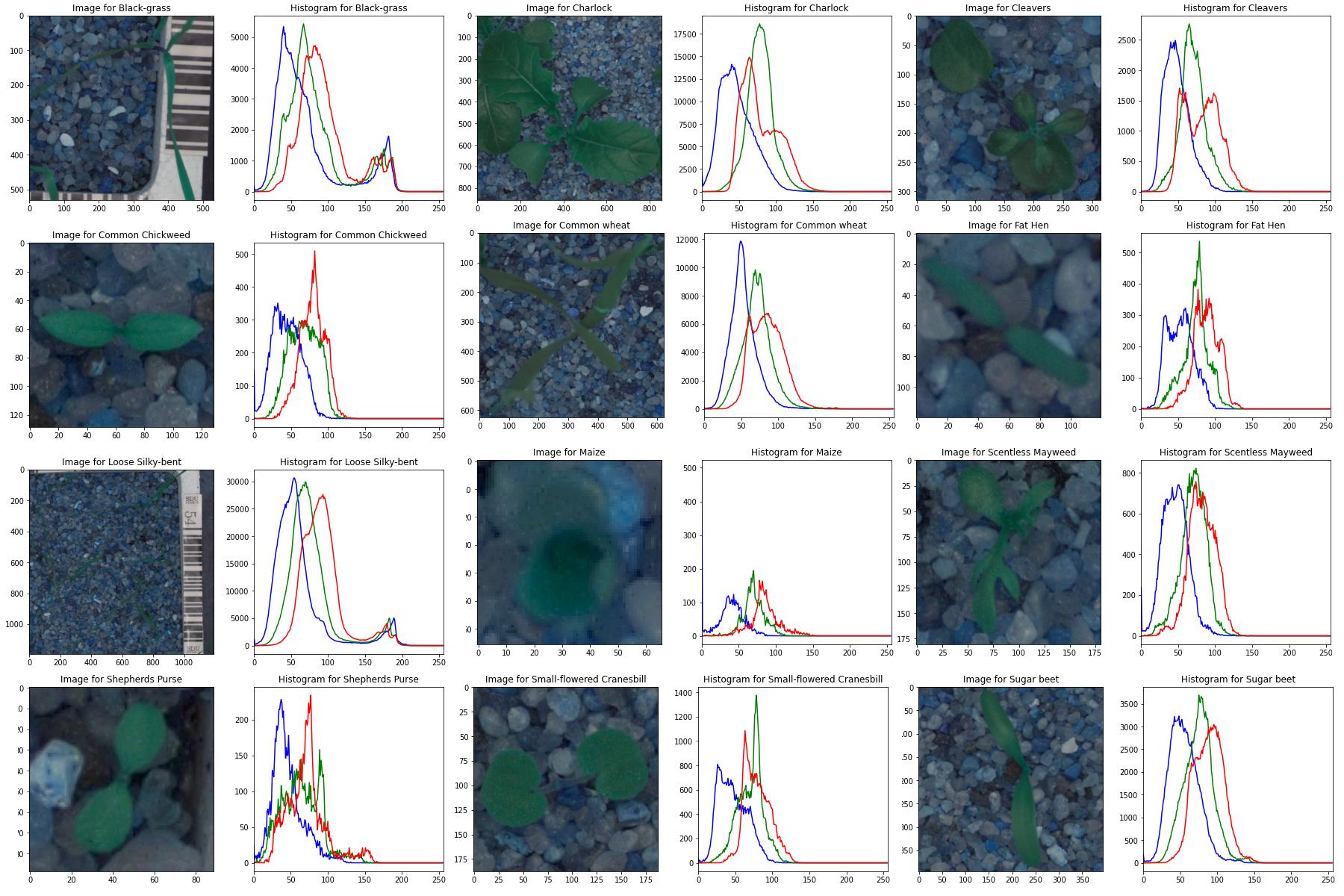


Figure 4: BGR Histograms of sample image in each category

Analysis on the colour histograms for the various categories gives us insights of the colour distribution across various categories in the dataset. It can be observed that the intensity levels vary across the categories. The category of Maize has low intensity across the different channels, whereas the category of Scentless Mayweed has high intensity across the channels. We can observe that various categories will have different distributions and different intensities across the BGR channels.

We can also observe that green is a dominant colour in the colour space when the category has plant seedlings with bigger leaves. Whereas when blue or red is the dominant colour in the colour space, the category has plant seedlings with smaller leaves. This is when the soil or text labels in the background are more dominant.

# Data Preprocessing

The training dataset provided by Kaggle contained a different number of training images in each plant seedling category. This imbalance may result in the model being unevenly trained and biassed towards the classes with a higher number of training samples. To overcome this, data augmentation was used to ensure that there are an equal number of training samples in each category.

There was a large variance in the sizes of the training images within each category and between categories. This vast difference in image size would cause problems during training. It would cause the models to fail as they would not be able to properly identify the image features. To overcome this, all the images were resized to the same dimensions either during preprocessing.

Lastly, there was a large amount of noise present in the training images. It was in the form of the soil surrounding the seedlings, as well as the barcodes and text labels present in some of the images. This noise would significantly hinder the models from identifying the correct features of each category of plant seedlings. To overcome this, masking is required for the images as a preprocessing step to remove unwanted pixels that constitute as noise.

## Data Augmentation

Data augmentation was implemented for all categories, this is so that all categories ended up with the same number of images per class. This ensures that the models we use can train evenly for each category. Within each category, existing images were chosen in sequence and augmented in one of a few different ways.

The Python package ***cv2*** was used to perform one of 5 possible augmentations to the input images for each category respectively. Rotations were performed using ***cv2.rotate***, while flips were performed using ***cv2.flip***. The possible augmentation options that could be randomly selected were:

1. Rotate left 90 degrees.
2. Rotate right 90 degrees.
3. Flip horizontally.
4. Flip vertically.
5. Flip horizontally and vertically, equivalent to 180 degree rotation.

The selection for the type of augmentation performed was determined randomly. This process was repeated until each category had 1000 images. In the end, there were a total of 12,000 training images across the 12 categories. The figure below shows an example of the outcome for each of the 5 possible augmentation options.

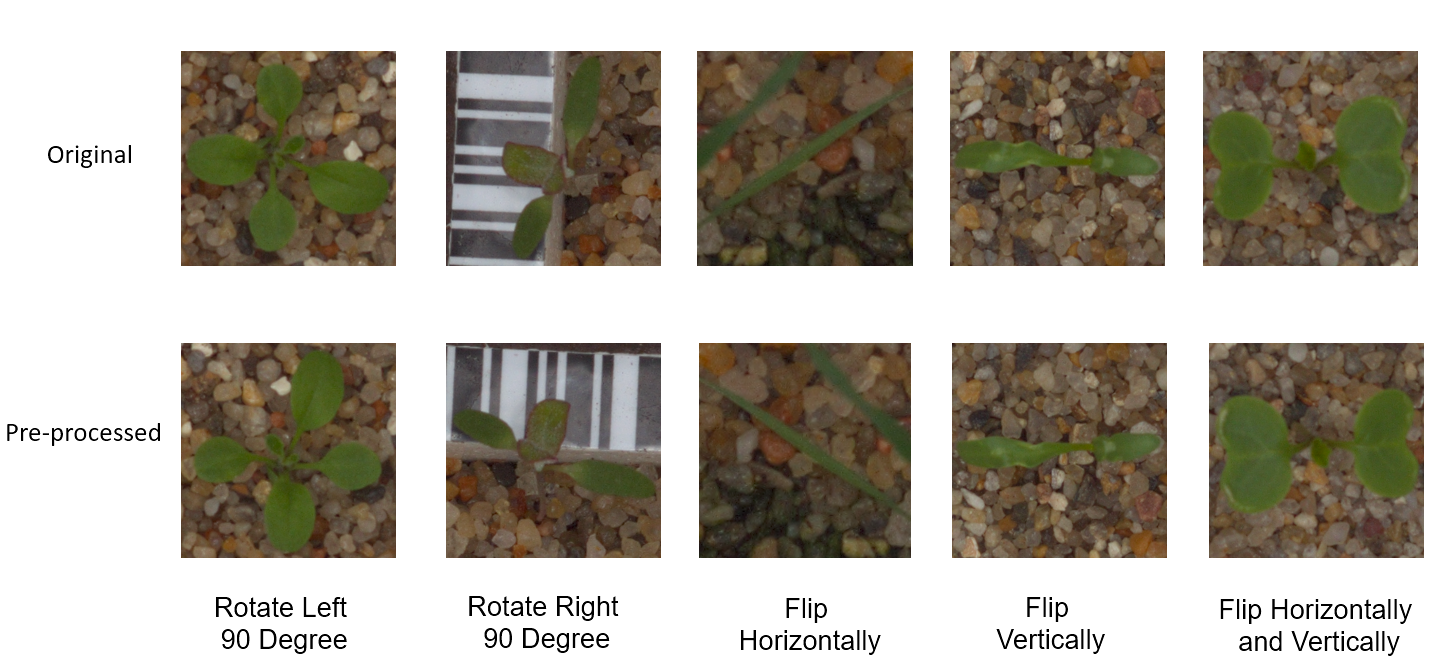


Figure 5: Possible Augmentation Options

## Image Resizing

The convolutional and deep neural networks used in some of our approaches to solving this classification problem have a fixed input layer shape which only accepts images of a specific height and width. Hence, there was a need to resize all the images used to the correct dimensions to feed into those networks.

The final dimensions of all the resized images are (256 x 256). These dimensions were chosen to ensure that sufficient details and features were retained in the images. It also ensures that we would not spend excessive time or computational resources training our chosen models using them [2].

## Image Masking

The aim of masking is to remove pixels that do not belong to the objects of interest in the input images, as those pixels would introduce undesired noise. A key feature of these plants is the green leafy colour of the various plant species and hence it would be beneficial to focus on the pixels that make up each plant’s leaves and stems. Hence, there is a need to remove the surrounding background noise, for example, the soil or the black and white barcode labels within the images.

We first use Gaussian blur to minimise noise by suppressing high frequency components of each image. It is then converted to the HSV format, a colour model that is an alternative to the RGB format. HSV is preferred for masking as it is less noisy than RGB. Afterwhich, a selected range of green colours are used to create a mask and transformed into a boolean mask. Finally, we applied each generated boolean mask on its respective image, achieving an isolated seedling image consisting of only its green leaves and stem, with no background. This can be seen in Figure 6.

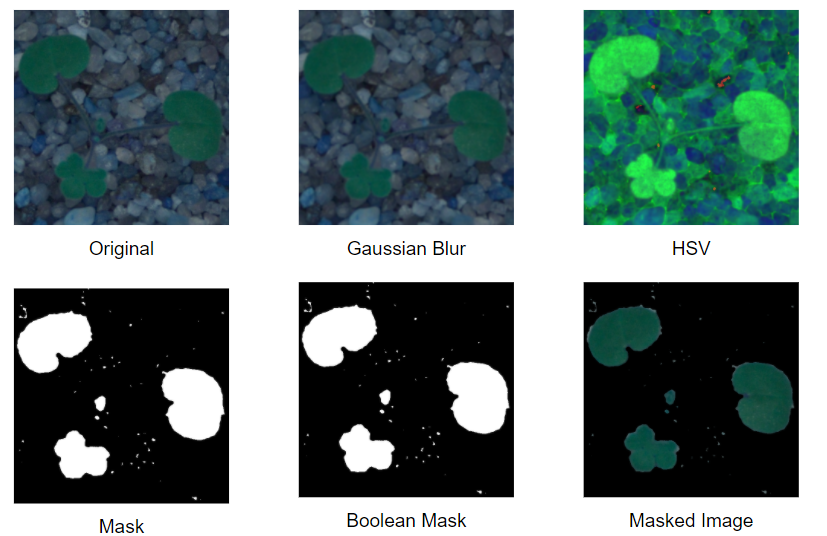


Figure 6: Process of Masking

## Train Test Validation Split

After the methods mentioned earlier, the training image dataset was further split into training and validation datasets in the ratio of 9:1. The training dataset after the validation split was used purely for training the chosen models. On the other hand, the validation dataset serves to simulate a test dataset, but whose labels are known. During training, the class predictions made by the model on the validation dataset images can be compared against their actual class labels to update the model’s weights and biases.

The unlabelled images represent the unseen data on which the trained model must make predictions. This means that the test dataset images must not be used for any part of the initial training or validation. Hence, only after the chosen models were fully trained, were they used to make predictions for the classes of these test images.

# [Methodologies: Traditional](#_heading=h.nk8ygo47uzoe)

## Method 1: K-Nearest Neighbours(k-NN)

### Overview

K-Nearest Neighbours classification is one of the most fundamental and straightforward classification methods, it is a non-parametric supervised learning method. It is also a lazy learning algorithm that postpones computation till after the evaluation [3].

In the training examples, points which are closer in distance, are normally grouped together. The “k” nearest neighbours is initialised to the desired number of nearest neighbours. Then, the distance is calculated for each point in the input data and stored. Then, the first “k” entries, with the closest distance, are chosen for each point and returned.

The K-Nearest Neighbours(k-NN) classification method was chosen as the first approach due to various reasons. Firstly, training is very efficient and computationally inexpensive. Because it is a lazy learning algorithm, no computation is done during the training phase, only information is stored in a collection. Secondly, since the k-NN classification method is simple to implement, we are able to use its results as a baseline for other models.

### Experiments

The k-NN model was imported from the Scikit-Learn library. It was implemented with the default parameters as shown in Table 2.

| **Parameters** | **Nearest neighbours** | 5 |
| --- | --- | --- |
| **Distance Metric** | Minkowski: Euclidean Distance |
| **Weights** | Uniform |
| **Algorithm** | Auto |

Table 2: Parameters for k-NN model

### Results

Kaggle F1 score for k-NN model: **59.445%**

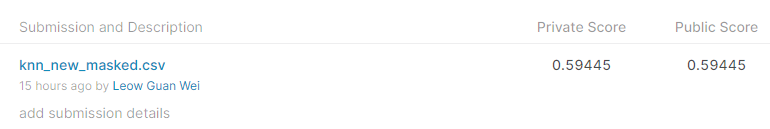


Figure 7: Results of k-NN model

## Method 2: Support Vector Machines (SVM)

### Overview

Support Vector Machine is a widely used supervised machine learning algorithm for classification tasks. The main objective of SVM is to find an ideal hyperplane in an n-dimensional space that maximises the margin between different classes, where n denotes the number of features [4].

Hyperplanes are decision boundaries that help to segregate data points. Data points near to the hyperplanes are known as support vectors. Support vectors have a heavy influence on hyperplane's optimal placement since they help to maximise the hyperplane's margin.

Although SVM is best for linear data, a kernel trick can be used to classify non-linear data. A Kernel Trick is an efficient method to transform non-linear data onto a higher dimension space. Based on the transformation, SVM finds the optimal hyperplane between possible outputs.

### Experiments

The SVM model was imported from the Scikit-Learn library. We implemented the Support Vector Classifier (SVC) of the SVM model as this is a multi-classification problem. The C-value, Gamma, and Kernel are the three main parameters for SVC. We used GridSearchCV to find ideal settings for each parameter. Table 3 shows the final parameters that we implemented in the SVM.

| **Parameters** | **C (control cost of miscalculation)** | 0.1 |
| --- | --- | --- |
| **Kernel** | Linear |
| **Gamma** | Auto |

Table 3: Parameters for SVM model

### Results

Kaggle F1 score for SVM model: **69.017%**



Figure 8: Results of SVM model

# [Methodologies: Convolutional Neural Network (CNN)](#_heading=h.m4c79j4qe3ww)

## [Overview of CNN](#_heading=h.vjqwr1p0c5rw)

CNNs are a form of Deep Neural Networks (DNN) capable of learning and identifying rich features and representations of the contents of images. They contain numerous convolutional and pooling layers for performing convolutional operations and feature extractions respectively. This allows them to successfully capture both spatial and temporal dependencies through the application of various kernel filters to extract features from the input images.

Some advantages of CNN models include being able to identify and distinguish features from images in an unsupervised manner, the ability to work with different image sizes, having to perform minimal preprocessing, and being able to reach decent accuracy rates with minimal tuning.

## Method 1: Xception

### Overview

Xception is a CNN developed by Google and built upon the concept of the inception module, but extends it further by having convolutional layers spanning both space and depth to a greater degree [5][6]. It consists of 71 depthwise layers, which makes more efficient use of model parameters, and is pre-trained using over a million images from the ImageNet database. It can classify objects in images from across 1000 categories.

Xception is constructed using repeating inception modules, with each module containing multiple convolution and pooling operations running in parallel within a single layer. The parallelized nature of this module reduces the depth of the CNN so that it is less prone to overfitting and less computationally expensive. Subsequent layers in Xception make use of the output from these Inception modules to decide the extent to which each feature is fed forward, improving its ability to distinguish the more important ones.

However, unlike traditional Inception based CNNs, Xception performs a few operations differently. It applies depth filters to the input prior to convolutional operations on them. It does not perform any batch normalisation after each summation block, and does not use any non-linear activation functions after each convolutional layer. These changes were observed to have improved the performance of Xception to varying degrees over other CNN models. The figure below shows a summary of the flow of the Xception model.

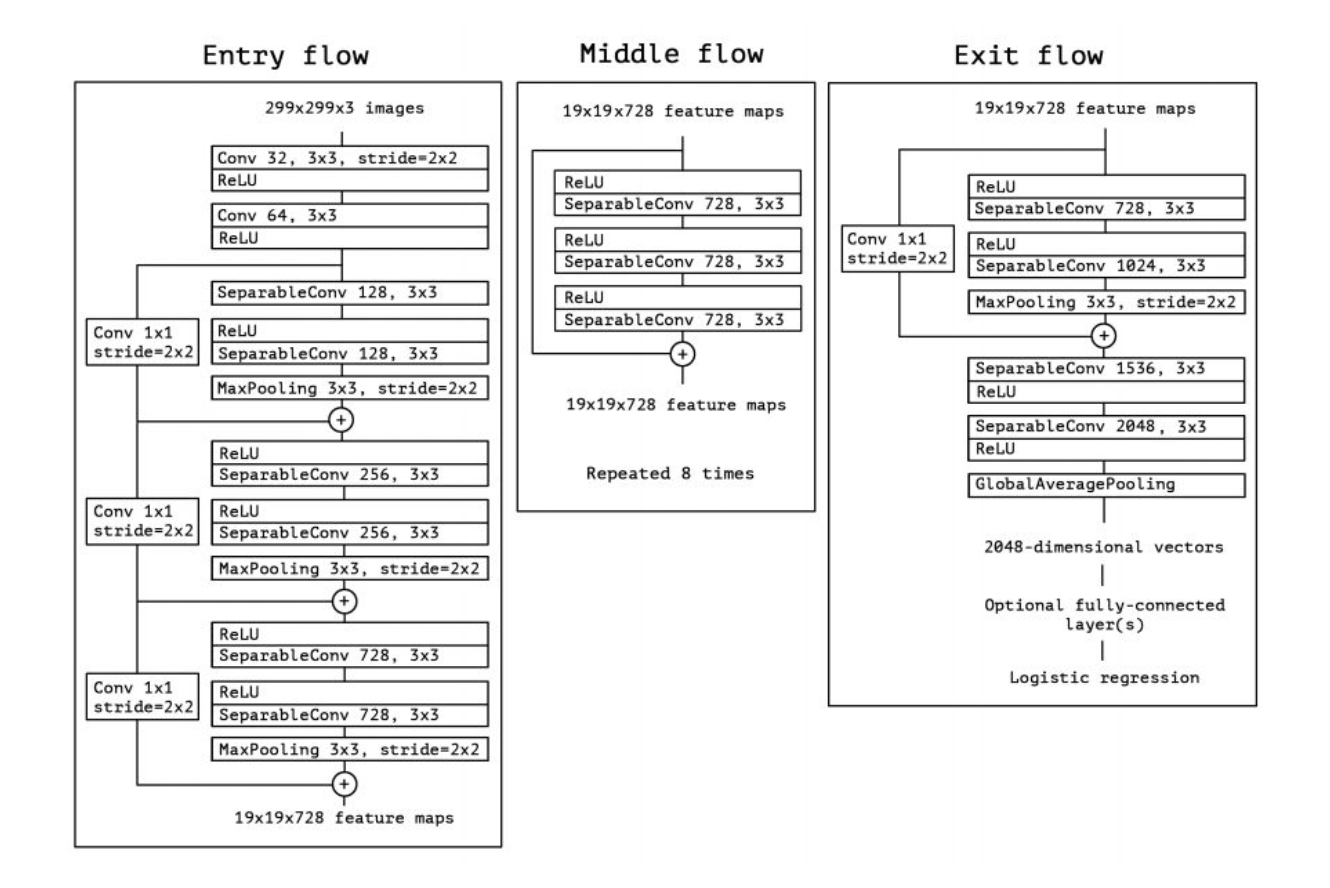


Figure 9: Summary flow of Xception model

### Experiments

Transfer learning was performed on the Xception model, imported from Keras Tensorflow without its original fully connected layers. A similar approach was used in the EfficientNetB7 model; it was adopted by creating a new fully connected layer of 1024 neurons.

The Adam optimizer was used for training the Xception model, with a very small initial learning rate of 0.0001 to allow for better convergence. The learning rate was gradually reduced by 5% in each epoch after the 5th one as the model converged. A larger initial learning rate was observed to cause significant fluctuations in the model’s validation accuracy and loss during training.

The Adam optimiser uses a combination of gradient descent with momentum and root mean square propagation to adaptively update a model’s weights and biases. It is capable of handling the sparse gradients of the data from the masked images as well as tolerating small amounts of noise [7][8].

A batch size of 16 was used for mini batch gradient descent. This larger batch size allowed for computational speedup via parallelism using the GPU without compromising on the model’s accuracy. A summary of the parameters used for training the Xception model are as listed in the table below.

| **Parameters** | **Input Image Size** | (299, 299, 3) |
| --- | --- | --- |
| **CNN Pooling Mode** | Average |
| **Dense Layers** | 1 |
| **Neurons per Dense Layer** | 1024 |
| **Dropout Rate** | 0.5 |
| **Validation Size** | 10% |
| **Max Epochs** | 100 |
| **Batch Size** | 16 |
| **Optimizer** | Adam |
| **Initial Learning Rate (up to epoch 5)** | 1e-04 |
| **Learning Rate Decay (after epoch 5)** | 0.05 |
| **L2 Regularisation** | 1e-05 |
| **Model Loss** | Sparse Categorical Crossentropy |
| **Model Checkpoint Metric** | Validation Loss |
| **Early Stopping Metric** | Validation Loss |
| **Early Stopping Patience** | 5 |

Table 4: Parameters for Xception

### Results

The training and validation accuracies and losses during the training of the Xception model are as shown in the figure below. Significant fluctuations were observed for both the validation accuracies and losses despite the very small initial learning rate and its subsequent decay. This may suggest the presence of some instability of this model for this particular problem.

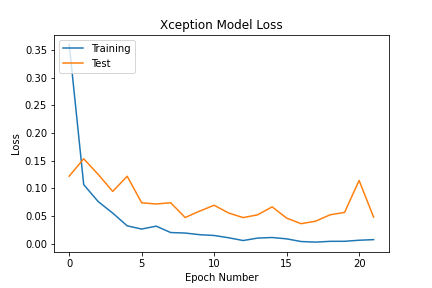
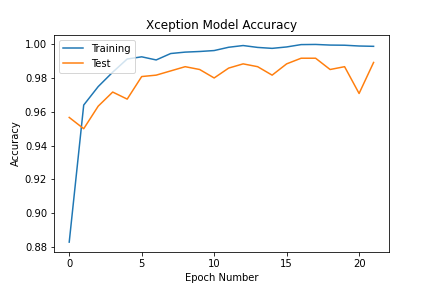


Figure 10: Xception Model Accuracy & Loss

Kaggle F1 score for Xception: **96.095%.**

****

Figure 11: Xception Model Kaggle Result

## [Method 2: EfficientNet](#_heading=h.a5erjh15kj5f)

### Overview

EfficientNet is a type of Convolutional Neural Network(CNN), and a scaling method that uses three scaling dimensions: depth, width, and resolution where each scaling dimension is scaled with a fixed set of scaling coefficients. To increase the computation, simply increase the coefficient αᶰ network depth, βᶰ width, the number of channel in a conv layer, and γᶰ , the image resolution that is passed to a CNN. These constant coefficients are determined by a small grid on the original model. EfficientNet uses a user specific compound coefficient ɸ to uniformly scale network width, depth, and resolution in a principled way. This way it allows the user to control how many resources are available, and α, β, and γ specific how these resources are allocated [9].

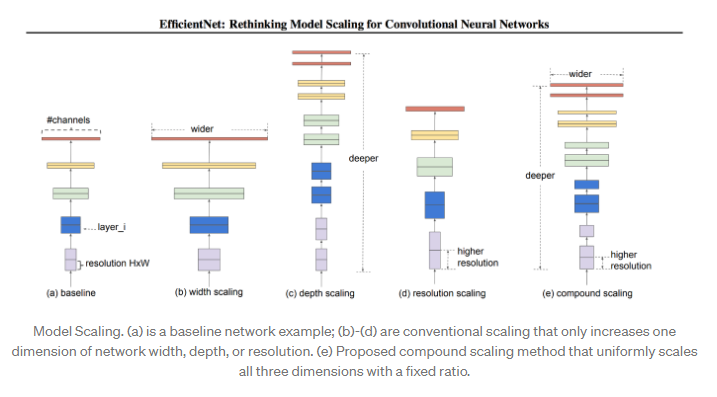


Figure 12: Model Scaling[5]

Scaling up the dimension of network d, width w, resolution r improves the accuracy, but accuracy gain is diminished for bigger models.

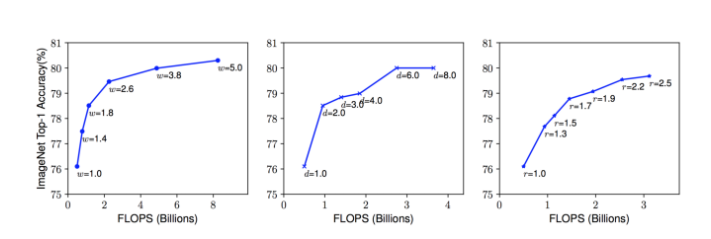


Figure 13: Computational Chart[5]

EfficientNet architecture base network is done by doing a Neural Architecture Search (MAS) using AutoML MAS framework that optimises for both accuracy and efficiency (FLOPS) to reduce computation. The resulting architecture uses mobile inverted bottleneck convolution (MBConv block) which is also used in MobileNetV2 and MnasNet. Hence, it is optimised for accuracy but penalised for if it is being too computational.

### Experiments

As the EfficientNetB7 model has already been pre trained on a large image dataset, its convolutional layers already serve as feature extractors. These feature extractors only require a smaller amount of retraining to adapt them for classifying the plant seedling images used in this project to a sufficiently high degree of accuracy.

Transfer learning was performed on the EfficientNetB7 model, imported from Keras Tensorflow without its original fully connected layers. Instead, a new set of fully connected and output layers were created and appended to the last layer of the EfficientNetB7 model for interpreting the features of the CNN. Average pooling was used for the pooling layers of the model for feature extraction [10]. A dropout layer was also used after the fully connected layer to avoid overfitting [11].

The fully connected layer consists of 1024 neurons [12], while the output layer consists of 12 neurons using the softmax activation function [13], representing the 12 categories of plant species for classification. The softmax activation function generates a vector of probability values across the 12 output neurons, representing the probability that an input image belongs to a specific class.

The Stochastic Gradient Descent (SGD) optimizer was used in the training of this model. SGD introduces randomness by using only one data point from the dataset in each epoch for faster convergence when working with large datasets [7][8].

A small initial learning rate of 0.005 was used for the first 5 epochs, and afterwards reduced by 5% in every subsequent epoch as the model converged so it could reach the global minima [14]. A batch size of 8 was used for a mini batch gradient descent strategy, allowing the model to attain decent precision and computational speedup via parallelism using the GPU [15].

The loss function used for the model was sparse categorical cross entropy, which uses only a single integer to represent a class instead of a one hot vector.

A summary of the parameters used for training the EfficientNetB7 model are as listed in the table below.

| **Parameters** | **Input Image Size** | (299, 299, 3) |
| --- | --- | --- |
| **CNN Pooling Mode** | Average |
| **Dense Layers** | 1 |
| **Neurons per Dense Layer** | 2048 |
| **Dropout Rate** | 0.5 |
| **Validation Size** | 10% |
| **Max Epochs** | 100 |
| **Batch Size** | 8 |
| **Optimizer** | Mini-batch Gradient Descent |
| **Initial Learning Rate (up to epoch 5)** | 5e-03 |
| **Learning Rate Decay (after epoch 5)** | 0.05 |
| **L2 Regularisation** | 1e-05 |
| **Model Loss** | Sparse Categorical Crossentropy |
| **Model Checkpoint Metric** | Validation Loss |
| **Early Stopping Metric** | Validation Loss |
| **Early Stopping Patience** | 5 |

Table 5: Parameters for EfficientNet

### Results

The training and validation accuracies and losses during the training of the EfficientNetB7 model are as shown in the figure below. Deeper variants of EfficientNet, such as the EfficientNetB7 used in this project, are known to encounter GPU memory constraints when training with a batch size larger than 8, hence a value of 8 was used instead of 16.

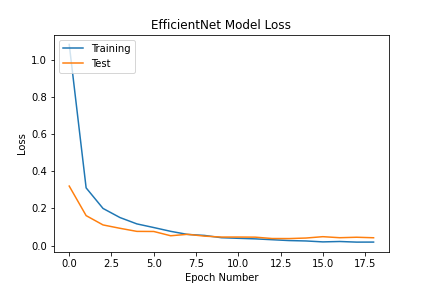
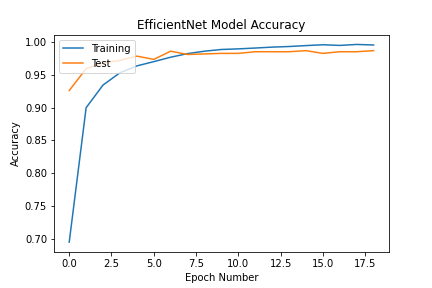


Figure 14: EfficientNet Model Accuracy & Loss

The Kaggle F1 score for EfficientNetB7 is **97.103%**.



Figure 15: EfficientNet Model Kaggle Result

## Method 3: InceptionResNetV2

### Overview

Inception-ResNet-V2 is an adaptation of an Inception based CNN, but which uses the scaled residual connections from a ResNet based CNN [16][17]. It consists of 164 depthwise layers, was similarly pre-trained on over a million images from the ImageNet database and can classify objects in images from over 1000 categories.

Inception-ResNet-V2 uses scaled residual connections between some of its convolutional layers to add shortcuts in its architecture between them. These shortcuts skip over the neurons in those layers, directly connecting the output of the convolution filters in each Inception module to its input.These bypassing connections help to avoid the vanishing gradient problem and reduce training time, while its activation scaling helps to improve training stability. Some of the pooling layers originally present in the Inception modules were also replaced with these residual connections.

The figure below shows a summary of the flow of the InceptionResNetV2 model. Note that the residual blocks in the figure below have been compressed for a summarised representation.

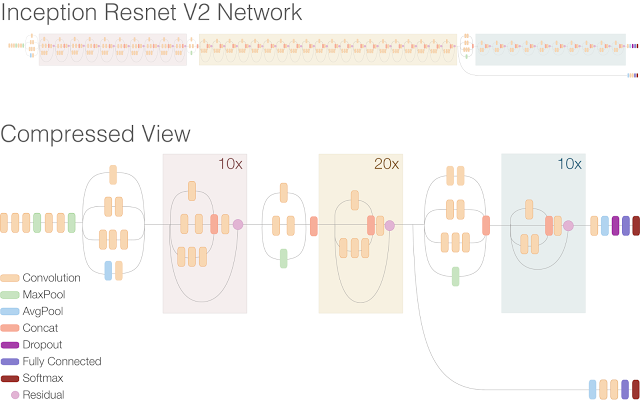


Figure 16: Inception Resnet V2 expanded & narrow network

### Experiments

Transfer learning was performed on the InceptionResNetV2 model, imported from Keras Tensorflow without its original fully connected layers. A similar approach as the Xception model was adopted by creating a new fully connected layer with the same parameters.

The Adam optimizer was used for training the InceptionResNetV2 model, with the same parameters as the Xception model. A summary of the parameters used for training the InceptionResNetV2 model are as listed in the table below.

| **Parameters** | **Input Image Size** | (299, 299, 3) |
| --- | --- | --- |
| **CNN Pooling Mode** | Average |
| **Dense Layers** | 1 |
| **Neurons per Dense Layer** | 1024 |
| **Dropout Rate** | 0.5 |
| **Validation Size** | 10% |
| **Max Epochs** | 100 |
| **Batch Size** | 16 |
| **Optimizer** | Adam |
| **Initial Learning Rate (up to epoch 5)** | 1e-04 |
| **Learning Rate Decay (after epoch 5)** | 0.05 |
| **L2 Regularisation** | 1e-05 |
| **Model Loss** | Sparse Categorical Crossentropy |
| **Model Checkpoint Metric** | Validation Loss |
| **Early Stopping Metric** | Validation Loss |
| **Early Stopping Patience** | 5 |

Table 6: Parameters for InceptionResNetV2

### Results

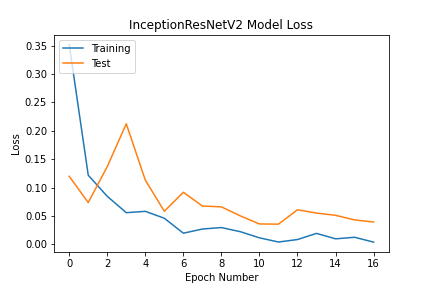
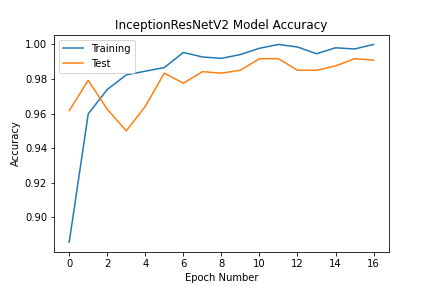
The training and validation accuracies and losses during the training of the Xception model are as shown in the figure below. Significant fluctuations were observed for both the validation accuracies and losses despite the very small initial learning rate and its subsequent decay. This may suggest the presence of some instability of this model for this particular problem.

Figure 17: InceptionResNetV2 Model Accuracy & Loss

Kaggle F1 score for InceptionResNetV2: **97.103%.**

****

Figure 18: InceptionResNetV2 Model Kaggle Result

## Method 4: Ensemble Learning Using CNN models

### Overview

Ensemble learning involves combining the predictions made by two or more models in an attempt to improve the final test accuracy. It helps to improve the average prediction performance across all the members over a single member, where each member should have an accuracy of greater than 50% [18][19]. In this project, the three CNNs, EfficientNet, InceptionResNetV2 and Xception, were combined in an ensemble using a majority voting strategy for improving the prediction accuracy of the test plant seedling images.

### Experiments

The saved model checkpoints from the best training epoch of the EfficientNet, InceptionResNetV2 and Xception models were first loaded. It was then evaluated by each of the above 3 CNNs.

The last layer of each CNN outputs a vector of probability values for each of the 12 possible classes that an input sample image could belong to. This ensemble performs majority voting by first summing the corresponding probability values from each model’s vector into a single vector of probabilities, with each element corresponding to the combined probability for a specific class.

The argmax function from Python’s ***numpy*** package is then used to determine and return the index in the summed vector with the highest probability value. That index will be matched to its respective class name and saved as the class prediction for that input test image. A list of all the predicted classes for all the test images are then saved into a spreadsheet for evaluation using Kaggle’s.

### Results

Kaggle F1 score for ensemble of EfficientNet, InceptionResNetV2 and Xception models: **98.362%**



Figure 19: Results of ensemble learning model

## [Summary](#_heading=h.h6sw8z7zqtck)

### Leaderboard Scores

Public rank is based on 833 submissions.

| **Method** | **Private Score** | **Public Score** | **Public Rank** |
| --- | --- | --- | --- |
| **k-Nearest Neighbours** | 0.59445 | 0.59445 | 758 |
| **Support Vector Machine** | 0.69017 | 0.69017 | 737 |
| **Xception** | 0.96095 | 0.96095 | 383 |
| **EfficientNetB7** | 0.97103 | 0.97103 | 262 |
| **InceptionResNetV2** | 0.97103 | 0.97103 | 262 |
| **Ensemble Learning** | **0.98362** | **0.98362** | **77** |

Table 7: Leaderboard Scores, note that the ensemble result is a combination of the EfficientNetB7. Xception and InceptionResNetV2 models.

# Solution Novelty

The following strategies listed below were employed in the development and training of our models for the classification of the plant seedling images in this project.

## Transfer Learning

Training a CNN from scratch would have required too much time and computational resources. Instead, we imported pre-existing CNN models whose weights and biases have already been pre-trained to identify the features of the datasets they were originally trained on. These learned features are also transferable to different or new data for image classification, requiring only a lesser extent of retraining for it to perform the task of classifying the plant seedling images.

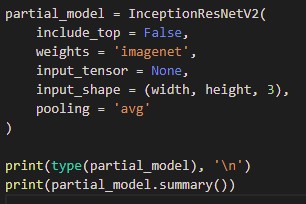


Figure 20: Screenshot of CNN model imported from Keras TensorFlow  
without its fully connected layers and using average pooling.

## Implementing our own Fully Connected Layers

Instead of importing the fully connected (dense) layers that came with each imported CNN, we chose to create and train our own fully connected and output layers. This has the added advantage in that those layers will not have been influenced by previous training on other different types of objects, but instead solely focus on the features of the plant seedling images that the models were to classify.

The fully connected layer used 1024 or 2048 neurons in order to capture the rich and diverse features. Additionally, we were able to customise our implementation of additional layers such as dropouts which would prove to be helpful in preventing overfitting.

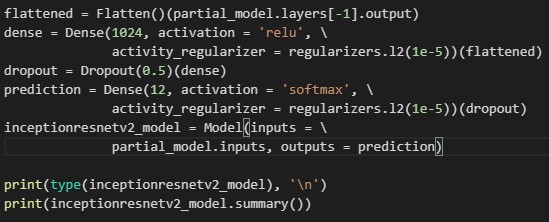


Figure 21: Screenshot of the new fully connected layer and output layer at the top of the CNN model, including the use of dropout and L2 regulariser, and the compilation of the adapted model.

## Ensemble Learning

The bagging ensemble learning method was used by combining three CNN models of different architectures: EfficientNetB7, InceptionResNetV2 and Xception. Each CNN had formerly been pre-trained using transfer learning and had their individual parameters tuned. The combined performance of all 3 CNNs can result in more accurate class predictions. The predicted class of each test image is determined using a majority voting strategy.

For each test image, every individual model in the ensemble would produce an array of 12 normalised values, representing the probabilities that the test image belongs to a particular class. The corresponding probability values from the arrays of all the models were then summed together. We then used the argmax function to obtain the class index with the highest probability value and treat that as the predicted class.

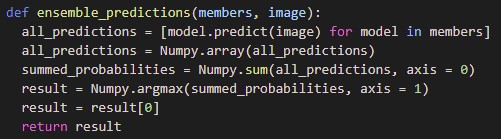


Figure 22: Screenshot of the function using majority voting on all  
the predictions from each CNN model in the ensemble.

## Early Stopping Callback

This callback measures a specific metric and stops the training of the model when that metric value fails to improve beyond a specified number of epochs. Validation loss was used as the monitored metric with a patience value of 5. This prevents the model from running additional epochs unnecessarily once it has converged to reduce the required computation time and to prevent overfitting due to excessive training.

## Model Checkpoint Callback

This callback measures a specific metric and saves the best weights and bias values of the model to a checkpoint file on the disk when that metric improves. Validation loss was also used as the monitored metric. This means that whenever an epoch produces a lower validation loss than any epoch before it, the weights and biases of the model at that point in training are saved to disk. This allows the optimal weights and biases of the model to be loaded in future without having to retrain the model.

## Learning Rate Scheduler Callback

This callback reduces the learning rate of the model gradually and linearly after a specified number of epochs. This helps the model to move towards the local or global minima faster in the beginning, while helping it to converge better towards the end of the training by taking smaller update steps. This helps the model to generalise better, and was observed to improve the final validation accuracies and reduce the final validation loss significantly.

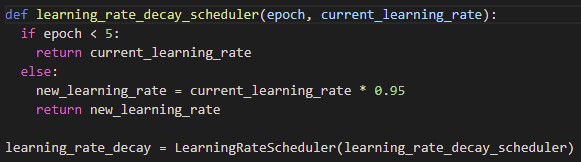


Figure 23: Screenshot of the function of learning rate scheduler

## Dropouts

While the large number of neurons in each of our own implemented fully connected layers help to improve the interpretation and classification of features, they can also make the model prone to underfitting or overfitting as some neurons may end up training to a greater degree than others.

To address this problem, dropouts of 0.5 were used after each dense layer. At every backpropagation update during training, approximately 50% of the neurons will have their output forced to zero, causing the remaining 50% of neurons to have their updates scaled up to learn more effectively from the training data. In this manner, all the neurons will train at approximately the same rate, preventing the underfitting or overfitting of specific neurons in the model.

## Level 2 Regularisation

Level 2 regularisation is used to minimise the complexity of the model and prevent overfitting by regulating the model’s weights. During each backpropagation update, this regulariser penalises the square value of each neuron’s weight by reducing it by a small percentage so that it does not fluctuate excessively. This also prevents the model from learning overly complex concepts with reference to any particular features in the training data to reduce the likelihood of overfitting.

# Challenges

The main challenge faced during this project was the limitations of computing resources. As we did not have enough computing resources on our personal computers, we turned to Google Colab and Kaggle Kernel. Both Google Colab and the Kaggle Kernel offer free GPUs for training, however, it was limited in free resources. Thus, we had to circumvent these limitations with various methods.

First, instead of doing the data preprocessing in the same run time as model training, we preprocessed and saved the images to be loaded later on. Then, we also made sure our image size was not too big, otherwise it will take up more resources to train the models. We also made sure to save a checkpoint whenever the results improve, this is so that we are able to continue the training from that checkpoint if the free resources have been used up. Lastly, we also created multiple accounts so that when we used up the free resources on one account, we could still continue training.

# Conclusion

The ability for computers to perform image classification automatically and accurately can greatly aid in numerous day to day tasks that would otherwise require large amounts of manpower and time. Generally, CNN based models have been shown to be better performing with the highest classification accuracies. This is made possible and easier with the added application of transfer learning to adapt existing CNNs for the specific image classification task.

Performing exploratory data analysis and image augmentation has also proven to be helpful in preprocessing the training and test images. This helped us to identify the need to ensure that there were sufficient unique training samples per class, and unwanted noise was removed. It also helped us identify the need to resize training images to suitable dimensions. This is to ensure that sufficient details and features were retained in the images, as well as using as little computing resources as possible.

Transfer learning not only gives us satisfactory results, it also saves computational resources. We also saw the benefits of learning rate scheduler, dropouts and level 2 regularisation, these helped to greatly improve our accuracies. As training an image model is a resource intensive task, it is highly recommended to use a GPU, or a cloud computing service which provides access to GPUs. This is especially so, for training of models that are CNNs or other forms of DNNs. This helps to reduce the amount of time spent training and testing each model.

In conclusion, many machine learning problems and their applications in this day and age are beginning to require and rely on near perfect accuracy in order to achieve their goals effectively. This is even more crucial when these goals are mission critical, with lives or valuable assets at stake. To achieve such high prediction or classification accuracies, it is likely that two or more neural networks will need to be adequately trained and combined in an ensemble for deployment.

# 

# References

[1] Jason Brownlee, Classification Accuracy is Not Enough: More Performance Measures You Can Use, March 2014. [Online], available: <https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/>

[2] Aravind Ramalingam, How to Pick the Optimal Image Size for Training Convolutional Neural Networks? June 2021. [Online], available: <https://medium.com/analytics-vidhya/how-to-pick-the-optimal-image-size-for-training-convolution-neural-network-65702b880f05>

[3] Scikit Learn, KNeighborsClassififer. [Online], available: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

[4] Vegi Shanmukh, “Image classification using machine learning-support vector machine(SVM),” Medium, 05-Mar-2021. [Online]. Available: <https://medium.com/analytics-vidhya/image-classification-using-machine-learning-support-vector-machine-svm-dc7a0ec92e01>

[5] Aakash Nain, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, June 2019. [Online], available: <https://medium.com/@nainaakash012/efficientnet-rethinking-model-scaling-for-convolutional-neural-networks-92941c5bfb95>

[6] Madhushree Basavarajaiah, Maxpooling vs Minpooling vs Average Pooling, February 2019. [Online], available: <https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9>

[7] Jason Brownlee, A Gentle Introduction to Dropout for Regularizing Deep Neural Networks, December 2018. [Online], available: <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>

[8] Ahmad Gad, Beginners Ask “How Many Hidden Layers/Neurons too Use in Artificial Neural Networks?”, June 2018. [Online], available: <https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e>

[9] Jason Brownlee, How to Choose an Activation Function for Deep Learning, January 2018. [Online], available: <https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/>

[10] Debjeet Asitkumar Das, Optimizers in Deep Learning, July 2020. [Online], available: <https://medium.com/analytics-vidhya/this-blog-post-aims-at-explaining-the-behavior-of-different-algorithms-for-optimizing-gradient-46159a97a8c1>

[11] Aishwarya V Srinivasan, Stochastic Gradient Descent - Clearly Explained, September 2019. [Online], available: <https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31>

[12] Suki Lau, Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning, July 2019. [Online], available: <https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1>

[13] Kevin Shen, Effect of Batch Size on Training Dynamics, June 2018. [Online], available: <https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e>

[14] Adrian Rosebrock, ImageNet: VGGNet, ResNet, Inception, and Xception with Keras, March 2017. [Online], available: <https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/>

[15] OpenGenus IQ: Computing Expertise and Legacy - Xception - Deep Learning with Depth Wise Separable Convolutions. [Online], available: <https://iq.opengenus.org/xception-model/> [

[16] Zahra Elhamraoui, InceptionResNetV2 Simple Introduction, May 2020. [Online], available: <https://medium.com/@zahraelhamraoui1997/inceptionresnetv2-simple-introduction-9a2000edcdb6>

[17] Christian Szegedy, Sergey Loffe, Vincent Vanhoucke, Alex Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. ArXiv, February 2016. [Online], available: <https://arxiv.org/abs/1602.07261v2>

[18] Aishwarya Singh, A Comprehensive guide to Ensemble Learning (with Python codes), June 2018. [Online], available: <https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/>

[19] Jason Brownlee, A Gentle Introduction to Ensemble Learning Algorithms, April 2021. [Online], available: <https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/>