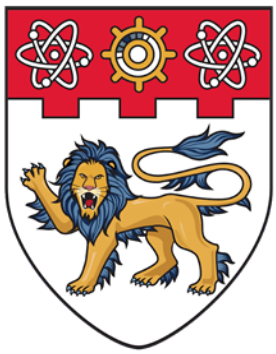

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

AY21/22 SEMESTER 2

CZ4041 MACHINE LEARNING

Project Report

Kaggle - Plant Seedlings Classification

Rank 227 out of 835

GROUP 11

U1922153D Colin Tan G-Hao

U1921779B Justina Quak Lin Ying

U1921135J Ang Jia Ning, Min

U1921075D Venkat Subramanian

Table of Contents

Roles and Contributions

Name	Contributions
Ang Jia Ning, Min	EfficientNet, VGG16 Models and Data Preprocessing
Colin Tan	Exploratory Data Analysis, K-Nearest Neighbours and Video presentation
Justina Quak Lin Ying	EfficientNet, Inception-ResNet, Exploratory Data Analysis and Video presentation
Venkat Subramanian	ResNet50, Inception-ResNet Models and Data Preprocessing

Introduction

Kaggle Plant Seedlings Classification Challenge
The Aarhus University Signal Processing group, in collaboration with University of Southern Denmark, has released a dataset containing images of approximately 960 unique plants belonging to 12 species at several growth stages.

This Kaggle competition is aimed at giving the community an opportunity to experiment with different image classification techniques, as well to provide a place to cross-pollinate (pun!) ideas.

Problem Statement

Knowing how to tell seedlings from weeds is a great skill to have, and even for the most seasoned gardeners, this is a tricky one to learn and master. The ability to do so will effectively increase crop yields and better stewardship of the environment.

Our aim with this project is to help gardeners avoid ruining their vegetable beds before they have barely started, by using machine learning models to differentiate a weed from a crop seedling.

Exploratory Data Analysis (EDA)

Overview

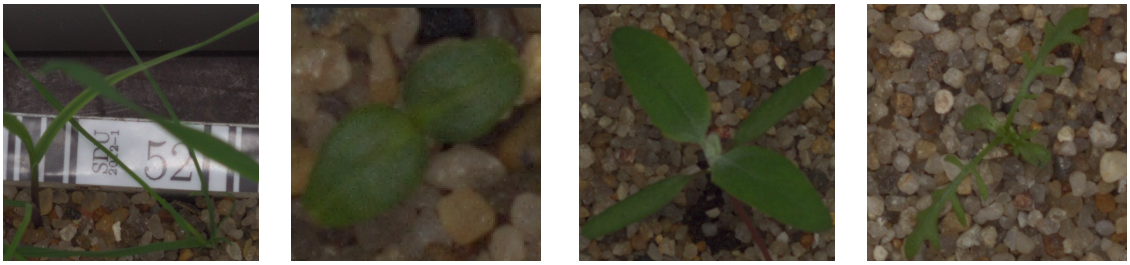

Figure 1: Sample images of different plants from the dataset

Figure 1 shows samples of the images in the dataset. As observed from the sample images, the images have a bit of noise (barcode, pebbles etc), which make image classification slightly difficult. These images would later be processed in the pre-processing stage.

	Species Name	No. Of Images
0	Black-grass	263
1	Charlock	390
2	Cleavers	287
3	Common Chickweed	611
4	Common wheat	221
5	Fat Hen	475
6	Loose Silky-bent	654
7	Maize	221
8	Scentless Mayweed	516
9	Shepherds Purse	231
10	Small-flowered Cranesbill	496
11	Sugar Beet	385

Figure 2: Table of number of images of each species in the dataset

As shown in Figure 2, there were a total of 4750 images of the plant seedlings, which were categorized into the 12 species and labeled. These images were provided for model building, and there was another set of 794 hidden and unlabeled images that were used by Kaggle for our model evaluation.

Data Trends/Distribution

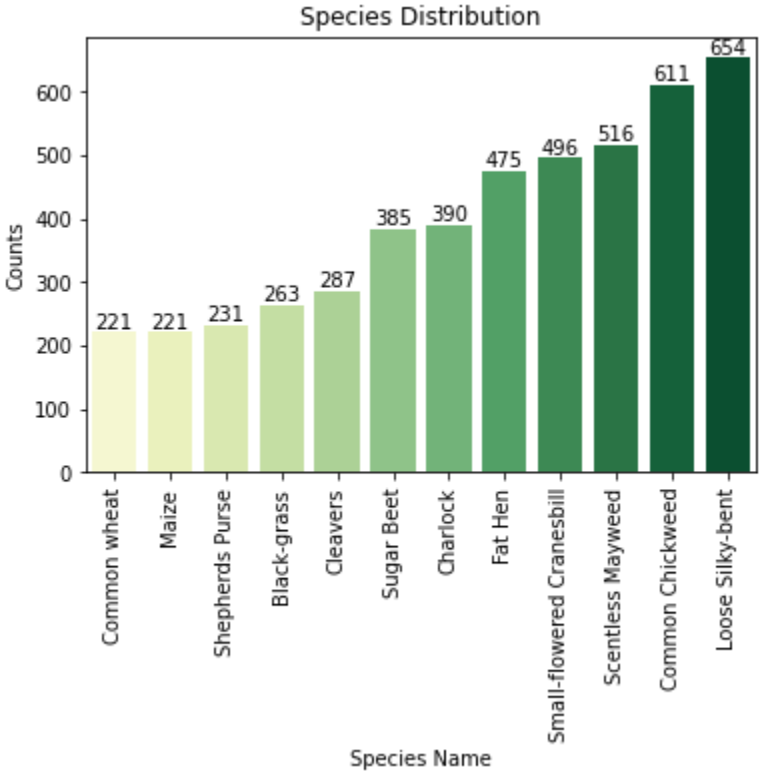

Figure 3: Bar Chart of number of images of each plant species in the dataset

Figure 3 shows the distribution of the various plant types in our data. As observed, there is a large variation in the numbers as Loose Silky-bent and Common Chickweed has more than double the numbers of Common Wheat, Maize and Shepherds Purse. A possible concern would be whether such imbalance distribution would result in lower model accuracy.

Data Preprocessing

In our Data Preprocessing process, we first tried to remove the lines in the image by using OpenCV's Canny Edge Detection function. The image below shows what we got.

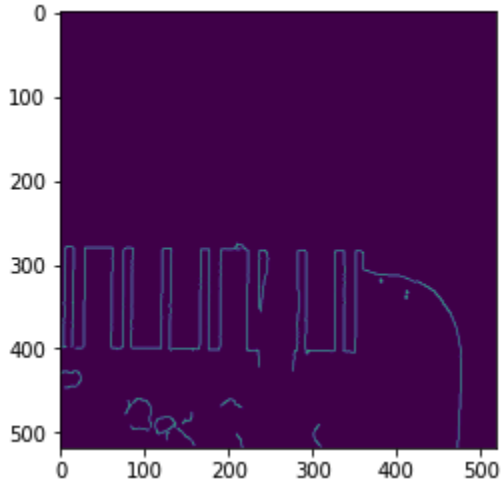

Figure 4: Result of Canny Edge Detection

As it is quite difficult to simply remove the lines from the image (as the lines of the plant did not get detected), we decided to use a histogram of the image to segment it instead.

RGB Splitter

Next, we tried to create an RGB splitter function to separate the image into its specific colour layers.

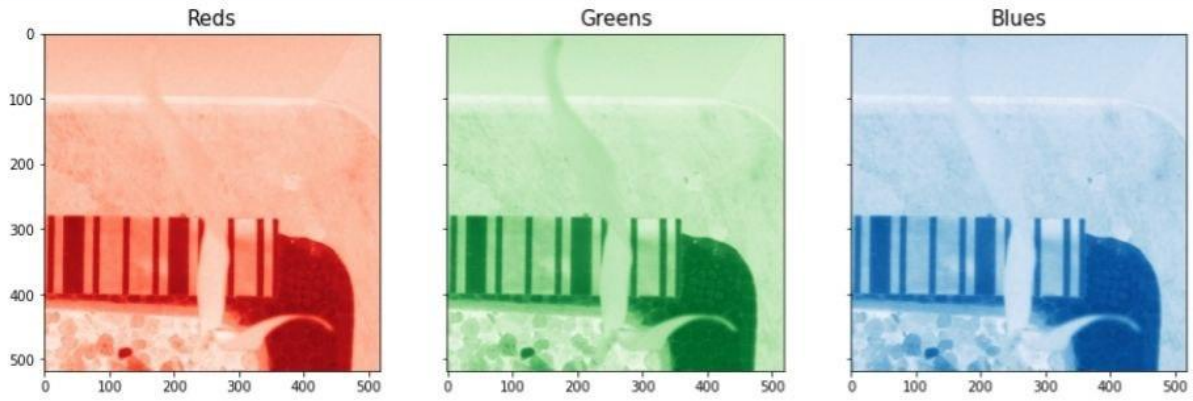

Figure 5: Result of RGB Splitter

However, there seems to be no drastic difference between the layers. Hence, this is not a feasible method for image segmentation.

Image Histogram

We then used OpenCV's calcHist function to create a colour histogram. This histogram represents the distribution of pixel intensities, which provide a high level intuition of the intensity (pixel value) distribution. With that, we get the graph below.

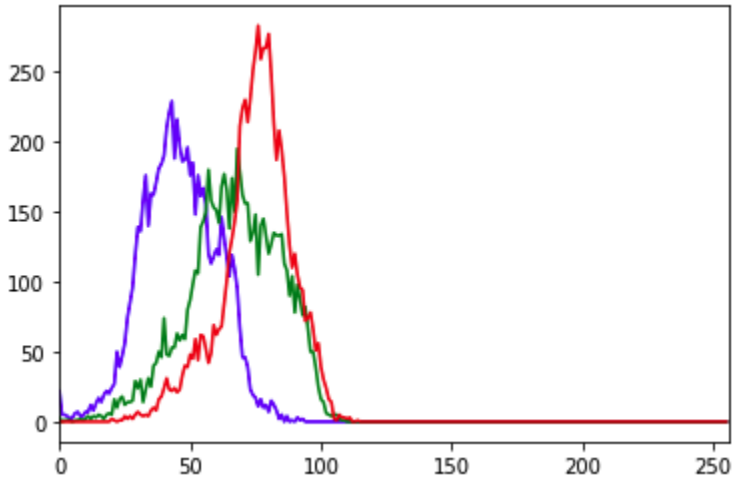

Figure 6: Histogram of colour distribution

The x-values of the graph represent the pixel value and the y-value represents the number of times these values were found in the image.

However, as we can see from the graph, the images do not have a clear pixel distribution, and hence they cannot be segmented purely using pixel intensities.

Image Resizing

After trying out the previous two image preprocessing methods, our next step lied in resizing of the images. A requirement for most machine learning models are that the images should be in fixed sizes. In our seedlings dataset however, the images were of different sizes, hence we had to scale the images to (*224 x 224)* pixels using the method *cv2.resize*.

Image Normalization

As all the images in the dataset are coloured, they contained 3 channels of R, G and B. For every channel, our objective is to normalize the pixel values such that they will have a mean value of 0 and standard deviation of 1. This was done using the method cv2.normalize.

In Machine Learning, data normalization helps the model to converge faster and avoids the oscillations of gradients in a back and forth manner such that it can find the global/local minimum easily.

Image Segmentation using Hue and Saturation

Next, we segmented the images by its hue and saturation.

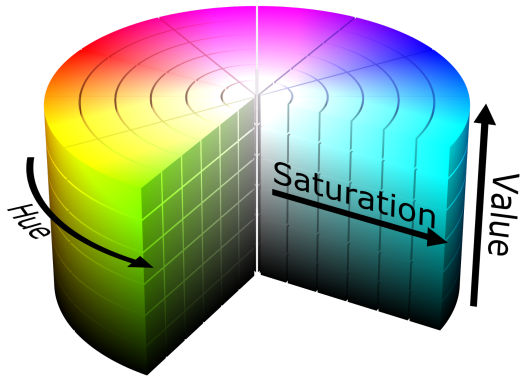

Figure 7: Representation of Hue and Saturation distribution

In our analysis, we found that the best range of values for segmentation are (25, 40, 50) to (75, 255, 255) for hue, saturation and brightness respectively. We used these values to segment the images, and normalized the data as neural networks perform better with normalized data.

Results of Data Preprocessing

After such preprocessing is done, we get images like below.

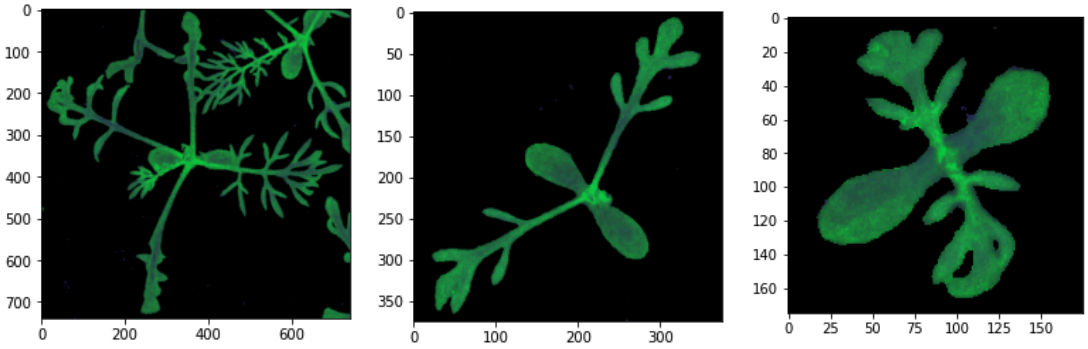

Figure 8: Example images after data preprocessing

Our dataset is now ready to be used for training in our models!

Methodology

Approach 1: K-Nearest Neighbours (K-NN)

Overview

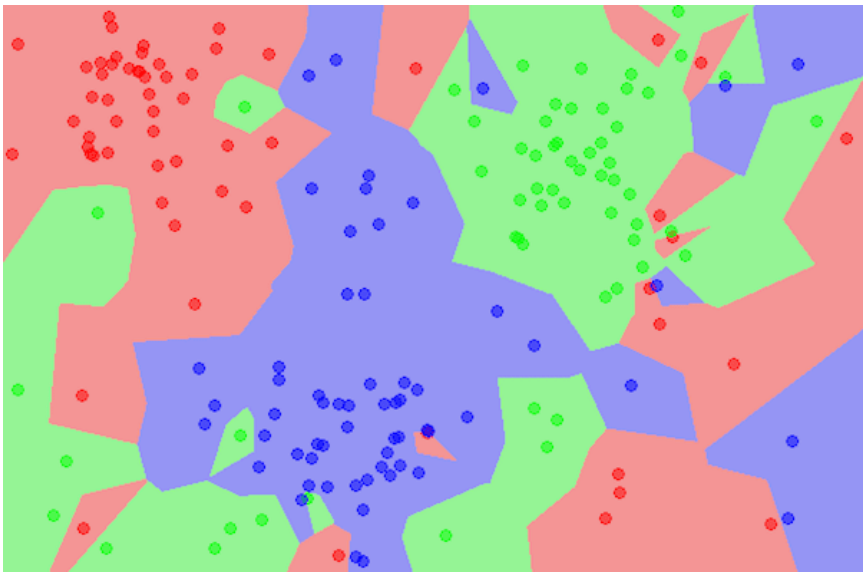

Figure 9: K-Nearest Neighbours Classification Example

K-Nearest Neighbours (K-NN) is a supervised learning algorithm used for both classification and regression. Training examples which are closer in n-dimensions are assigned to be similar to each other, as depicted in the figure above. It is a lazy learning algorithm where the computation is postponed until the evaluation.

Firstly, “k” is initialized to the nearest number of neighbours required. For every entry in the dataset, the Euclidean distance is calculated and stored in a sorted collection. After the iteration, the first “k” entries are picked from the collection for each example and the output it returns.

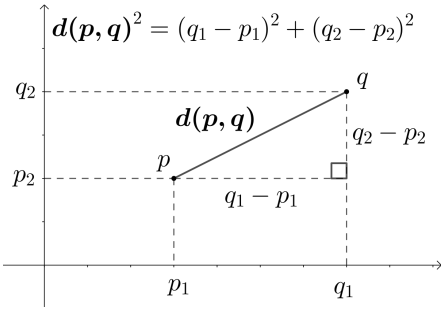

Figure 10 : How Euclidean distance is measured in 2 dimensions

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2}.$$

Equation : Measuring Euclidean distance in n-dimensions

Motivation

We started off with KNN algorithm because it uses supervised learning, which means that it utilizes the labels in our training data. It is also relatively computationally inexpensive. This is because essentially no training happens in the training phase, only information is derived from the dataset and stored in a collection. KNN is easy to implement for multi-class classification datasets such as ours compared to some algorithms and it gives us the flexibility to choose from a variety of distances to measure with: Euclidean distance, Hamming distance and Manhattan distance.

Experiments

The images are pre-processed, and features are extracted using methods described in the “Dataset” section above. All the 4750 examples are used for training and subsequently, the test set containing 794 images is used for model evaluation. The parameters used to train the KNN algorithm using Scikit-Learn are listed in the table below.

Parameters	# of Nearest Neighbours	5
	Weight for each point	Equal
	Distance metric	Euclidean

Limitations

1. Does not work well with a large dataset
   Since KNN is a distance-based algorithm, the cost of calculating distance between a new point and each existing point is very high, in turn degrading its performance.

2. Does not work well with high number of dimensions
   Same as above. In higher dimensional space, the cost to calculate distance becomes expensive and hence impacts the performance.

3. Sensitive to outliers and missing values
   KNN is sensitive to outliers and missing values, thus there is a need to first input the missing values and get rid of outliers before applying the KNN algorithm.

Approach 2: EfficientNet

Overview

EfficientNet is a convolutional neural network architecture and scaling method that uses a compound coefficient to uniformly scale all dimensions of depth/width/resolution. This is unlike conventional practice that arbitrarily scales these factors.

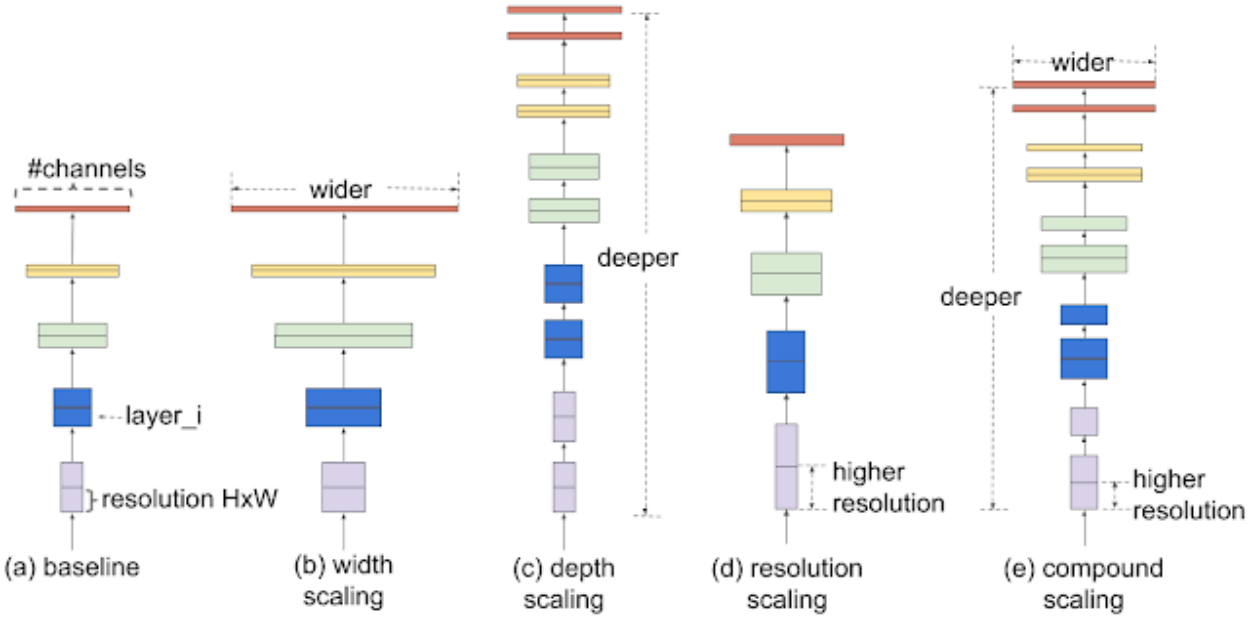

Figure 11: Comparison of different scaling methods

GoogleAI has created this family of models, called EfficientNets, which is powered by this novel scaling method, which surpasses state of the art accuracy with up to 10 times better efficiency, as it is smaller and faster.

Motivation

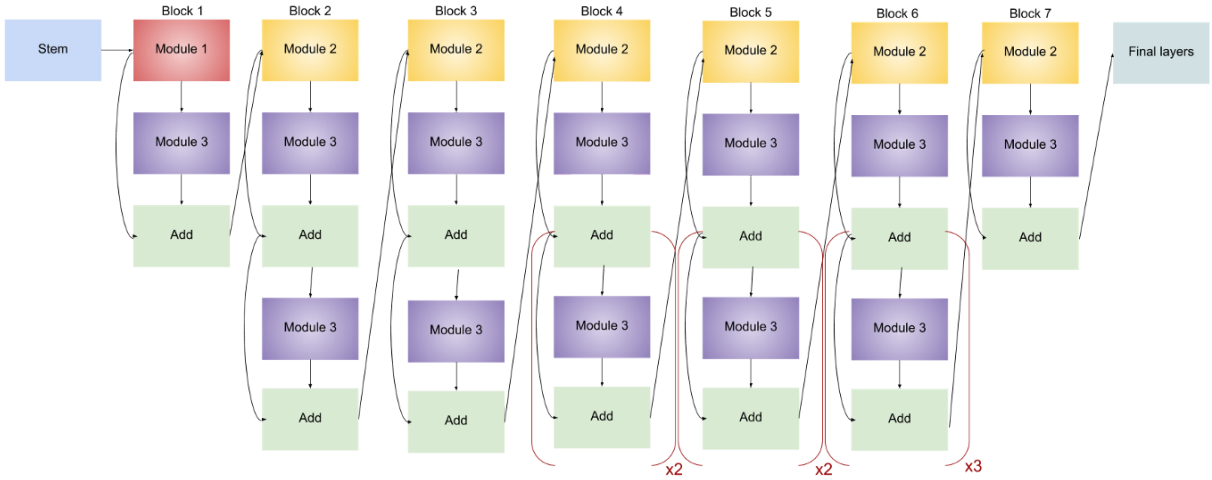

Figure 12: Architecture of network EfficientNet-B1

We have chosen to use EfficientNetB1 as it is simple and clean, which makes it easier to scale and generalize based on our needs. Its architecture is the same as the baseline model EfficientNetB0, the only difference between them is that the number of feature maps (channels) is varied, which increases the number of parameters.

Moreover, EfficientNet has very high accuracy obtained on top of very few parameters and floating points per second (FLOPS), meaning they can run at high speeds.

https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

Experiments

The images are pre-processed, and features are extracted according to the Data Preprocessing section above. Out of the 4750 images in the dataset, 80% is used for training and 20% is used for the validation set. This means that we used 3800 images for training purposes. The python library used to implement the model was Keras by Tensorflow and below are the parameters used.

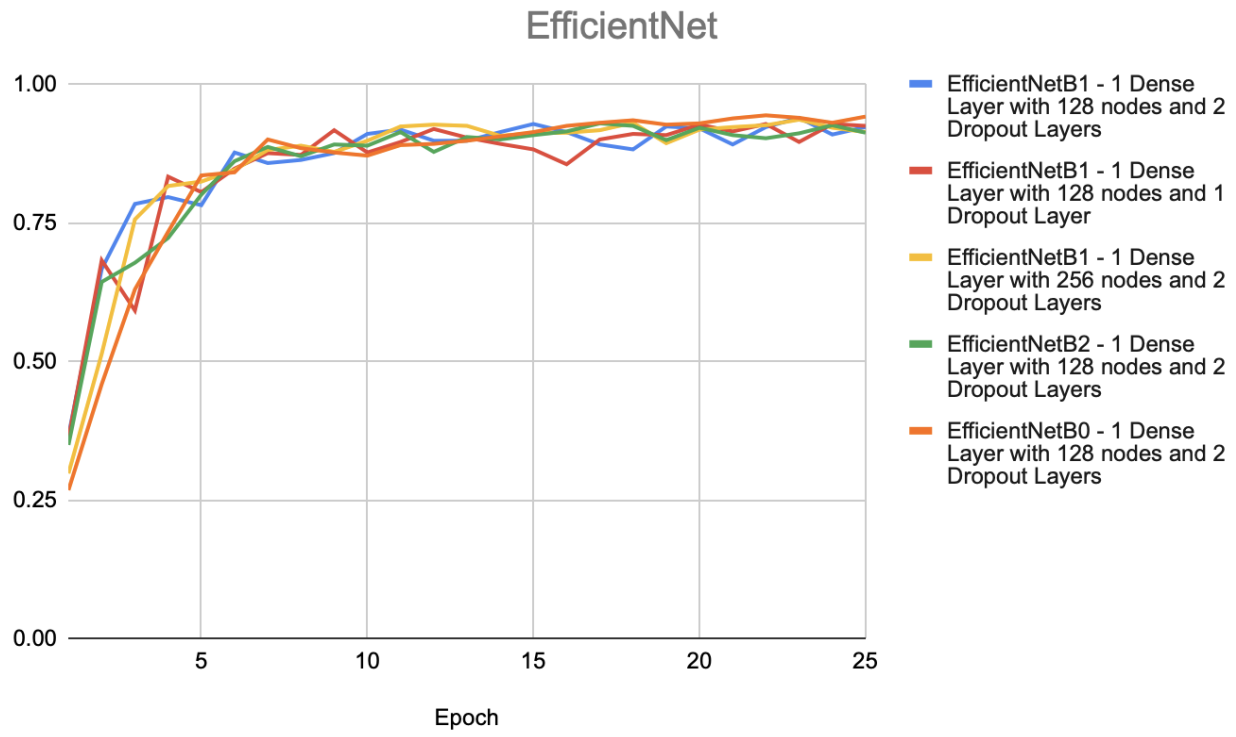

Figure 13: Graph showing different validation accuracy of the different models

We experimented with different numbers of dense layers, nodes and batch normalization and the parameters that yielded the highest validation accuracy are in the table below.

Parameters	Epoch	25
	Batch Size	64
	Learning Rate Range	$1e^{-10}$ to $5e^{-4}$
	Validation Size	20% of Training Data

Limitations

While EfficientNet is able to significantly reduce the FLOPS and parameters required by removing all channel connections in the depthwise layer, in turn reducing the complexity of the solution that the layer can represent, the number of channels is actually increased to boost the overall capacity. The result is more data movement due to the larger number of channels.

Hardware accelerators like GPUs are designed for models with large amounts of compute and where data movement is a relatively small component of overall performance. This means that EfficientNet performs poorly on hardware accelerators.

Approach 3: VGG16

Overview

VGG16 is a convolutional neural network trained on a subset of the ImageNetdataset, a collection of over 14 million images belonging to 22,000 categories. It has 16 Convolutional and Max Pooling layers, 3 Dense layers for the Fully-Connected layer and an output layer of 1,000 nodes.

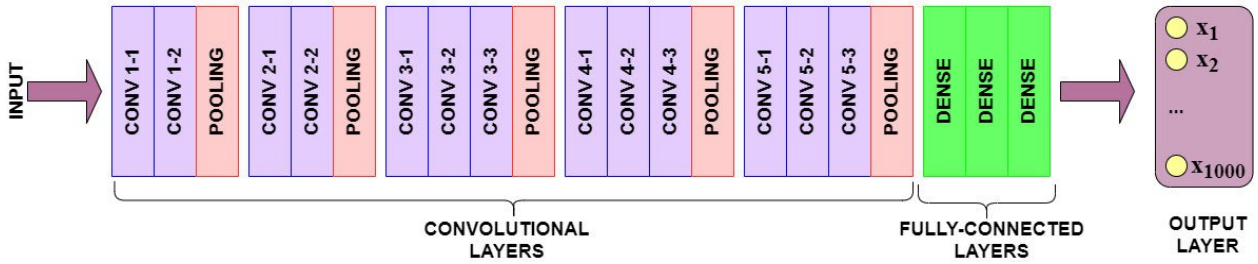

Figure 14: Architecture of VGG16

The most unique thing about VGG16 is that instead of having a large number of hyper-parameters, it is focused on having convolution layers of 3x3 filter with stride 1 and always uses the same padding and maxpool layer of 2x2 filter of stride 2. It has a consistent arrangement of convolution and max pool layers throughout its whole architecture. This network is a pretty large network and it has about 138 million (approx) parameters.

Motivation

VGG16 is a great building block for learning purposes as it is **easy to implement**. Moreover, it achieved 92.7% top-5 test accuracy in ImageNet (a dataset containing more than 14 million images belonging to 1000 classes). It won the Large Scale Visual Recognition Challenge in 2014. It made an improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the 1st and 2nd convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. Till date, it is considered to be **one of the most excellent vision model architectures**.

Experiments

The images are pre-processed, and features are extracted according to the Data Preprocessing section above. Out of the 4750 images in the dataset, 80% is used for training and 20% is used for the validation set. This means that we used 3800 images for training purposes. The python library used to implement the model was Keras by Tensorflow and below are the parameters used.

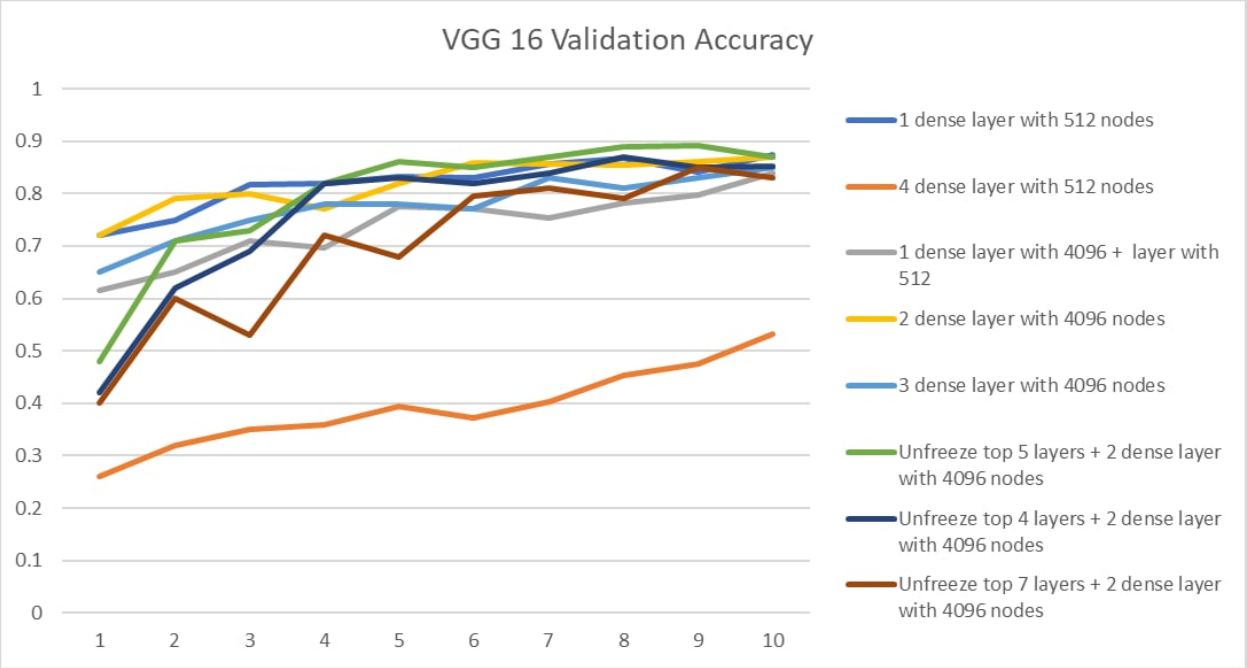

Figure 15: Graph showing different validation accuracy of the different models

We experimented with different numbers of dense layers, nodes and batch normalization and the parameters that yielded the highest validation accuracy are in the table below.

Parameters	Epoch	10
	Batch Size	100
	Learning Rate Range	0.0001
	Validation Size	20% of Training Data

Limitations

It is very slow to train (the original VGG model was trained on the Nvidia Titan GPU for 2–3 weeks).

The size of VGG-16 trained imageNet weights is *528* MB. This takes a substantial amount of disk space and bandwidth which makes it inefficient.

Approach 4: Inception-Resnet-V2

Overview

Inception-ResNet-v2 is a convolutional neural architecture that builds on the Inception family of architectures but incorporates residual connections (replacing the filter concatenation stage of the Inception architecture). It is trained on more than a million images from the ImageNet database. The network is 164 layers deep and can classify images into 1000 object categories. As a result, the network has learned rich feature representations for a wide range of images. With the combination of the Inception model and the ResNet architecture, the model has fewer parallel connections than before, but the number of layers has increased. The intricate architecture is seen in the Figure below.

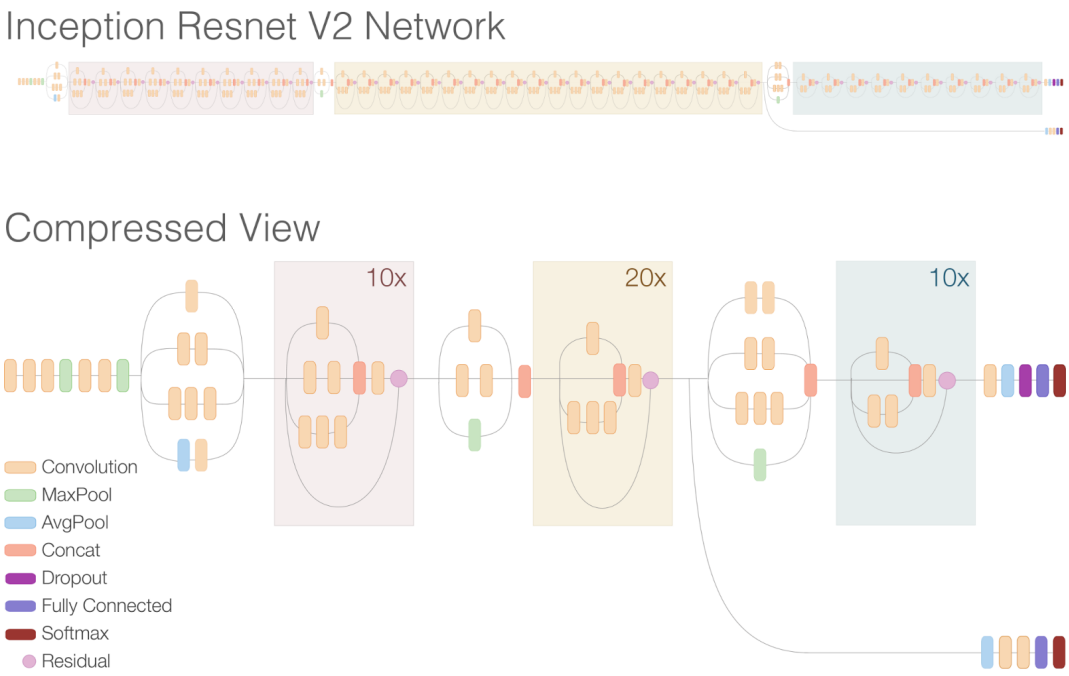

Figure 16: Schematic diagram of Inception-ResNet-v2

Motivation

Transfer Learning: The power of transfer learning could be leveraged in the Inception-ResNet-V2 model, using the weights from the ImageNet trained model in this method. This is because the model trained to classify millions of images from the ImageNet dataset was a task quite similar to our plant seedlings' classification. Each layer in this architecture is a feature extractor that could be trained on our dataset, and finetuned to produce results that are accurates when it is applied to our plants seedlings classification problem.

ResNet performs better with deeper architectures: In general, ResNet showed that it solved the problem of decreasing accuracy when consistently increasing the depth of the neural network. As seen in the figure below, a comparison between plain and ResNet layers, when we observe the plain networks on the left, as we increase the layers there is a decrease in the train error, but after a few layers, the error goes back increasing. When it comes to ResNet however, as the layers are increasing, the error only tends to decrease strictly. This significant boost in performance made ResNet architecture an optimal choice for our seedlings classification task.

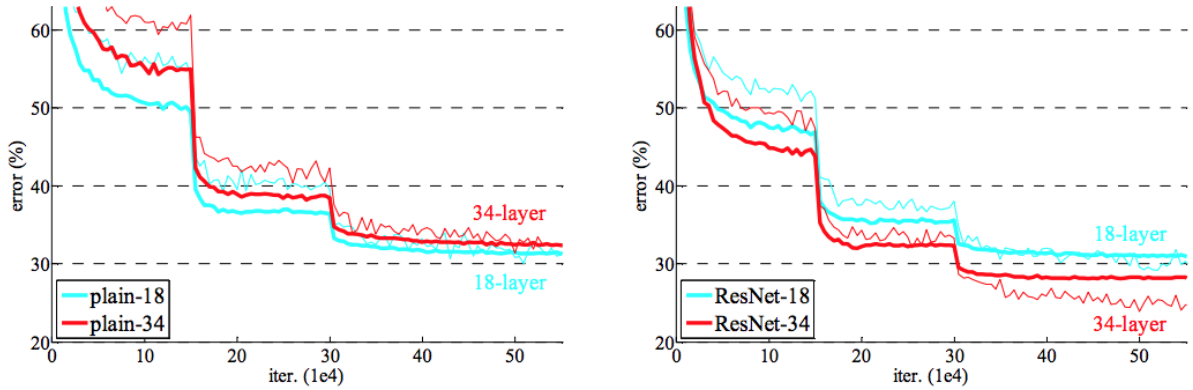

Figure 17: Plain vs ResNet Layers

Experiments

The images are pre-processed, and features are extracted according to the Data Preprocessing section above. Out of the 4750 images in the dataset, 80% is used for training and 20% is used for the validation set. This means that we used 3800 images for training purposes. The python library used to implement the model was Keras from Tensorflow.

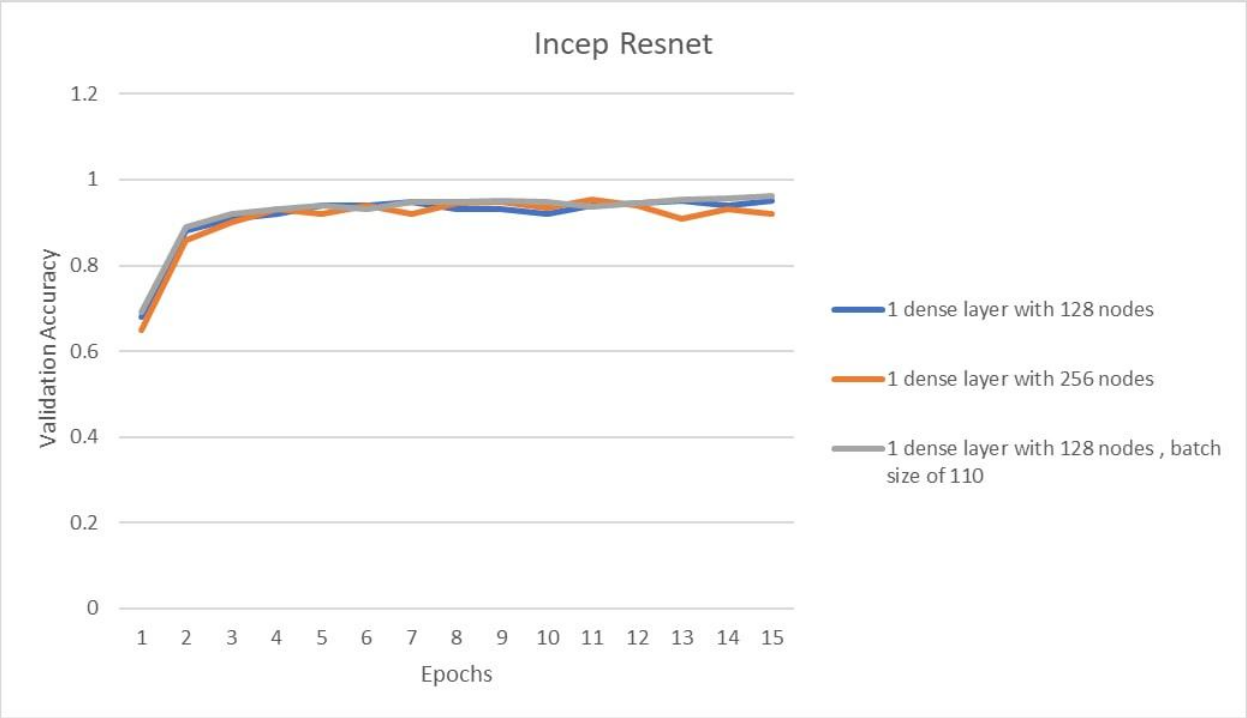

Figure 18: Graph showing different validation accuracy of the different models

We experimented with different numbers of dense layers, nodes and batch normalization and the parameters that yielded the highest validation accuracy are in the table below.

Parameters	Epoch	15
	Batch Size	110
	Learning Rate	$5e^{-4}$
	Validation Size	20% of Training Data

Limitations

The main limitation of the Inception-ResNetV2 model is that it takes a roughly higher training and testing run time compared to other models due to the complex structure of the inside modules.

Approach 5: ResNet50

Overview

Residual Networks (ResNet) is a classic neural network used as a backbone for many computer vision tasks.

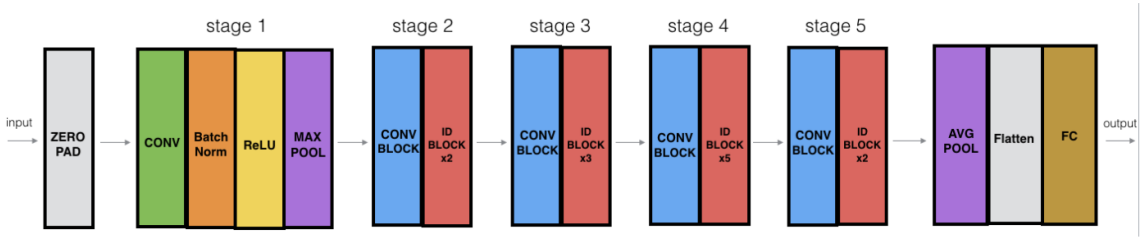

Figure 19: Architecture Diagram of ResNet50

The ResNet-50 model consists of 5 stages each with a convolution and Identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. The ResNet-50 has over 23 million trainable parameters.

ResNet first introduced the concept of skip connection. The diagram below illustrates skip connection. The figure on the left is stacking convolution layers together one after the other. On the right we still stack convolution layers as before but we now also add the original input to the output of the convolution block. This is called skip connection.

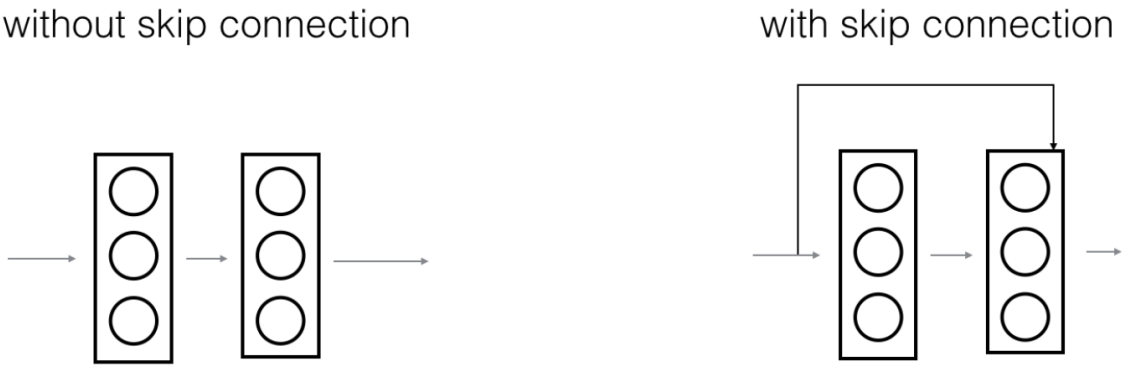

Figure 20: Illustration of Skip Connection in ResNet

With skip connection, it:

- Allows the model to learn an identity function, ensuring that higher layers performs at least as good as or better than lower layers
- Mitigate the problem of vanishing gradient by allowing an alternate shortcut path for gradient to flow through

https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

Motivation

This model was the winner of the ImageNet challenge in 2015. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+layers successfully. Prior to ResNet, training very deep neural networks was difficult due to the problem of vanishing gradients.

Experiments

The images are pre-processed, and features are extracted according to the Data Preprocessing section above. Out of the 4750 images in the dataset, 80% is used for training and 20% is used for the validation set. This means that we used 3800 images for training purposes. The python library used to implement the model was Keras from Tensorflow and below are the parameters used.

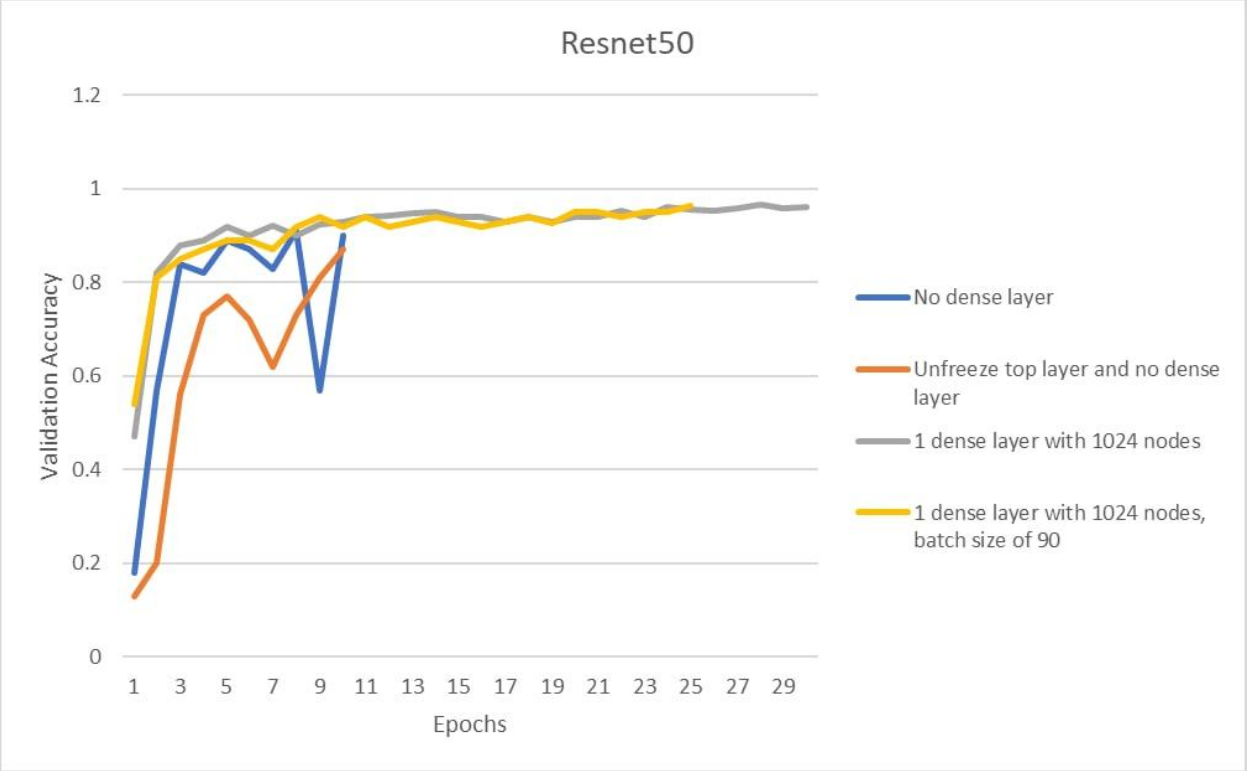

Figure 21: Graph showing different validation accuracy of the different models

We experimented with different numbers of dense layers, nodes and batch normalization and the parameters that yielded the highest validation accuracy are in the table below.

Parameters	Epoch	30
	Batch Size	100
	Learning Rate	0.001
	Validation Size	20% of Training Data

Limitations

One major drawback of ResNet50 is that a deeper network such as this, usually requires weeks for training, making it practically infeasible in real-world applications. To tackle this issue, we introduced a counter-intuitive method of randomly dropping layers during training, and using the full network in testing.

Approach 6: Ensemble Learning - Random Forest

Overview

Ensembling is a machine learning method that combines a few base models in order to produce one optimal model. The necessary conditions for an ensemble classifier to perform better than a single classifier would be that the base classifiers are not perfectly correlated, and that each base classifier model should perform better than random guessing. In our case, this means that each model has to perform better than 8.3% accuracy.

Random Forests are a class of ensemble methods specifically designed for decision tree classifiers, where many trees are grown and generated based on a random subset of features, and the final prediction is based on voting by the trees.

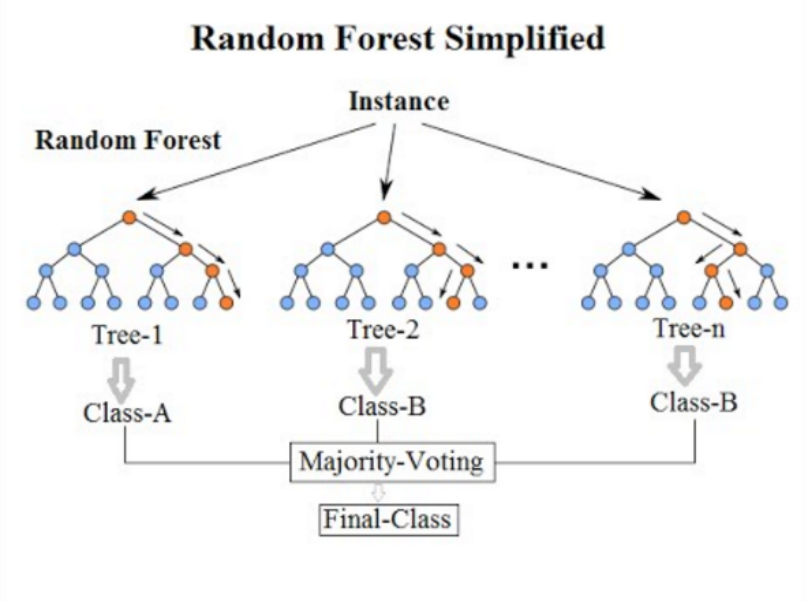

Figure 22: Diagram demonstrating how Random Forest is implemented

Motivation

Having done the 5 models, we wanted a way to combine the predictions, and ensemble learning is the best way to do so. We chose to use Random Forest as it uses the bagging technique, which reduces the variance in the dataset, and the chance of overfitting is also reduced.

Experiments

The library used was RandomForestClassifier by sklearn.

Only models with high validation set accuracy were chosen as input models.

	EfficientNet	Incep Resnet	Resnet50 V1	Resnet50 V2	Random Forest
Validation Set Accuracy	0.922	0.94	0.962	0.967	0.970

Using these input models, we ran the model to find the best parameters.

Parameters	n_estimators	1577
	min_samples_split	5
	min_samples_leaf	1
	max_features	auto

	max_depth	110
	bootstrap	True
	Validation Size	20% of Training Data

Using random forest, we managed to obtain a validation set accuracy of 0.970, which is higher than any of the input values, and hence we can say that the random forest worked.

Limitations

The main limitation of random forest is that a large number of trees can make the algorithm too slow and ineffective for real time predictions.

Solution Novelty

Image Segmentation

Instead of training the model directly on the raw data, we understood the importance of image segmentation, especially in this scenario where the images in the dataset had lots of noise. Image segmentation is the process whereby a digital image is broken down into various subgroups called image segments, which helps in reducing the complexity of the image to make further processing or analysis of the image simpler.

In other words, this is simply like assigning labels to each pixel, and what we did was to decide whether each pixel was relevant or not to our model. By doing so, we effectively reduce the noise and enhance the model's accuracy by training on relevant data.

Test-Time Augmentation

Test-Time Augmentation is a powerful heuristic that takes advantage of data augmentation which was done during testing to produce an averaged output. It is a technique used to improve performance and reduce generalization error when training neural network models for computer vision problems, like what we did in this plant classification project. TTA in particular creates multiple augmented copies of each image in the test set (through various image manipulation techniques such as zooms, crops, shifts), and then having the model make a prediction on each augmented image, before returning an ensemble of those predictions. This is largely effective and makes our model better by seemingly increasing the data inputs available, allowing our model to make the best decision possible.

Ensembling: Random Forest

Ensemble methods are techniques that involve creating multiple models and then combining them to produce improved results. In our case, we chose to use random forest as our ensemble machine learning algorithm, as it performs well across a wide range of classification and regression predictive modeling problems. Random decision forests correct for decision trees' tendency to overfit to the training set. In our case, we also prevented this by having a validation set, on top of the test set provided by Kaggle.

Keras Callbacks

Keras provides strong callbacks tools that can be used during the execution of each epoch. Especially for the training of our models, we have used 2 of the most important functions: *ReduceLROnPlateau* and *EarlyStopping*.

ReduceLROnPleateau serves as a way to reduce the learning rate by a certain factor when the loss has stopped decreasing.
EarlyStopping helps to stop the training early when the loss has converged and stopped decreasing after a certain number of epochs.

These 2 callbacks work together to help ensure that the training loss converges with the minima but does not overfit the training data. As a result, this has helped our team to achieve a much higher testing accuracy.

```
def callbacks():

    # save best model regularly
    save_best_model = tf.keras.callbacks.ModelCheckpoint(filepath = 'incepresnet',
        monitor = 'val_acc', save_best_only = True, verbose = 1)

    # reduce learning rate when it stops decreasing
    reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor = 'loss', factor = 0.4,
                              patience = 3, min_lr = 1e-10, verbose = 1, cooldown = 1)

    # stop training early if no further improvement
    early_stopping = tf.keras.callbacks.EarlyStopping(
        monitor = 'loss', min_delta = 1e-2, patience = 8, verbose = 1,
        mode = 'min', baseline = None, restore_best_weights = True
    )
```

Figure 23: Keras Callbacks - *ReduceLROnPleateau* and *EarlyStopping*

Challenges Faced

Throughout the project, there were many challenges we had encountered that slowed us down in our progress. However, through much hardwork and patience, we were able to overcome them and successfully conquered the seedlings classification task!

1. **Memory Errors**
   We ran the training of our models on Kaggle Notebooks, a cloud computational environment that enables reproducible and collaborative analysis. This serves as a platform for our team to collaborate and work on the code together, helping out each other where necessary. However, the use of such cloud technologies will incur certain limitations, especially in terms of hardware resources.

   The largest challenge we had faced using the Kaggle Notebooks were the memory limitations. In particular, the CPU in the remote servers only had a total of 16GB of RAM that we could use. This resulted in our program having a ton of 'MemoryError' during the processing and training of the data, resulting in delays to our progress.

2. **Time and resource consumption**
   Due to the huge seedlings dataset, the time taken to run the processing and training of the model is extremely long and arduous. This was one of the biggest challenges as it rendered our computer useless as we were unable to perform other tasks during the training process.

   Furthermore, there were certain models that could take up to **11 hours** to complete training. This taught us that each time the model is trained, we must be very careful of the parameters that we input into the model, in order to prevent the large amount of time used to train the model to go to waste.

Leaderboard

Ensembling - Random Forest

Using 4 input variables, we created a random forest that achieved 0.970 validation set accuracy. We submitted this to Kaggle, and below are our results.

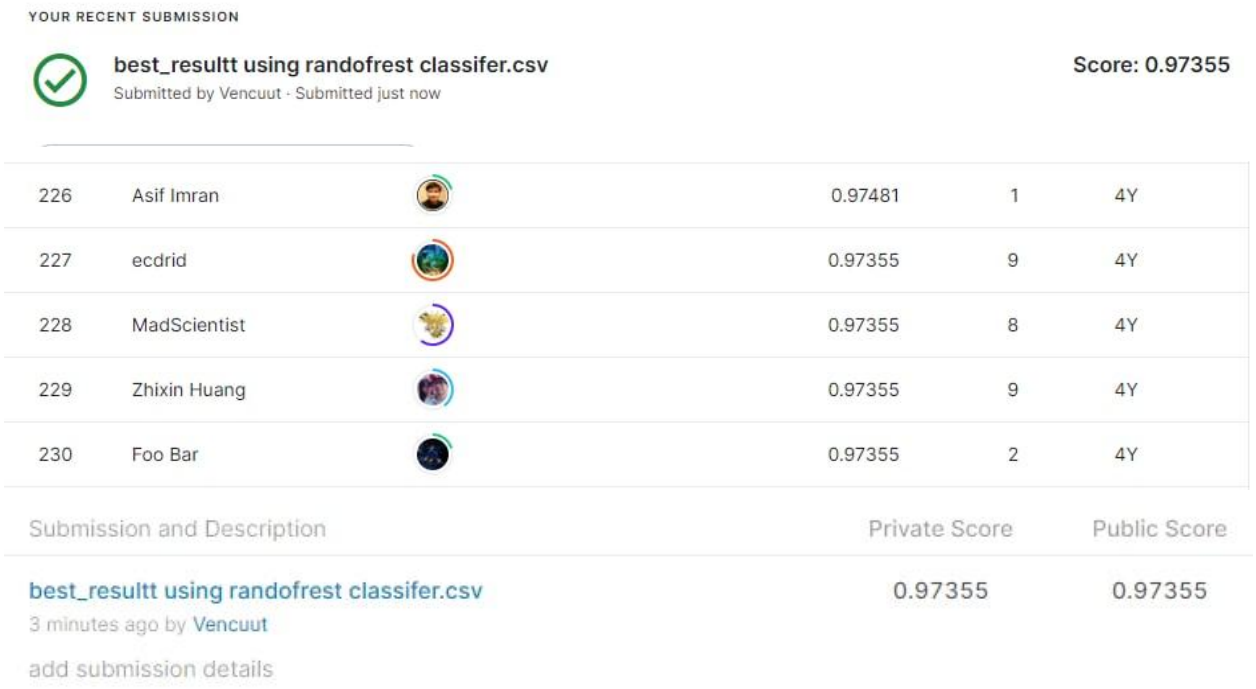

Figure 24: Our final score on Kaggle

We achieved a private and public score accuracy of **0.97355**, putting us at **227th out of 835** teams.

Conclusion

This Machine Learning project has helped us obtain major insights into the complete Machine Learning Workflow. We were exposed to the multiple phases of:

1. Exploratory Data Analysis
2. Data Pre-processing
3. (Different) Model Training, Evaluation and Analysis

These phases allowed us to get a better understanding of the given data, and allowed us to understand the importance of proper data preprocessing. This is particularly interesting as we thought we were able to train the models without doing any major image processing methods, but we were proven wrong by the low model accuracies.

We also did an incremental methodology, by trying different approaches and tuning various hyper parameters before arriving at the Ensemble method.

Throughout this project, we better understood the models that we learnt in class as well as its implementation methods. It was great to be able to apply what we've learnt in the module, and see it come to life through the different models. We were also able to see the beauty of ensemble learning methods, as it is true that no singular model is the best, and it is with the collective effort of various models that we are able to get the best result.

The usage of Kaggle Notebook also taught us valuable lessons with regards to collaborative team effort. With Machine Learning models, the time taken to run the program can easily take up to hours each time. Hence, it is important for our team to have effective communication, set out tasks and roles for each person and ensure that none of our work overlaps nor would it affect the workflow of another person. Overall, we are very proud to say that our team worked cohesively as a unit and managed to conquer all the challenges we faced throughout this project and achieved a result that we are all happy with and proud of.