Group 5: Project Report

Kaggle – Plant Seedlings Classification

Rank **54** out of **833** (Top 6.5% Rank)

Bhatia Ritik (U1822529C)

Gupta Jay (U1822549K)

Dwivedee Lakshyajeet (U1822289L)

Mantri Raghav (U1822309B)

Bansal Aditya (U1822156C)

CZ/CE 4041 Machine Learning
School of Computer Science & Engineering
Nanyang Technological University, Singapore

Submitted to—
Sinno Jialin PAN
School of Computer Science & Engineering
Nanyang Technological University, Singapore

Contents

Team Structure

Name	Contribution
Mantri Raghav	Support Vector Machine, Data Augmentation
Dwivedee Lakshyajeet	Deep Neural Network (Xception Net), Image Segmentation, Data Augmentation, Majority Voting (Ensemble Learning)
Bhatia Ritik	Deep Neural Network (EfficientNet), Deep Neural Network (Resnet – Inception – v2) Data Augmentation
Bansal Aditya	Convolutional Neural Network
Gupta Jay	Exploratory Data Analysis, Feature Extraction, K-Means Clustering, K-Nearest Neighbours

Statement of Problem

Can we identify the differences between a weed and a crop seedling? In this Kaggle competition [1], we need to predict the category (species) of a plant seedling with an RGB image of the plant.

Motivation – "The ability to do so effectively can mean better crop yields and better stewardship of the environment." [1]

Evaluation on Kaggle

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}$$

Equation 1: Balanced F_1 Score used for Kaggle Evaluation

Submissions on Kaggle for this competition are evaluated using the F_1 score, as mentioned above, where *tp* is the number of true positives, *fp* is the number of false positives, and *fn* is the number of false negatives.

Exploratory Data Analysis (EDA)

Overview

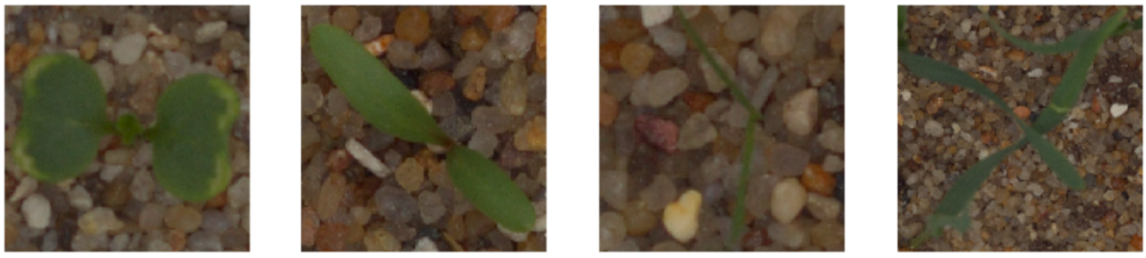


Figure 1: Some samples from the dataset [1]

The Aarhus University Signal Processing group, in collaboration with University of Southern Denmark, has recently released a dataset containing images of approximately 960 unique plants belonging to 12 species at several growth stages [1]. There are in total 4750 RGB labelled images of plant seedlings categorized into 12 species, given for model building. Another set of 794 unlabelled images are used by Kaggle for model evaluation and leader board ranking.

Species Name	# Images
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: Table of species in the dataset

All images are square in shape with different dimensions even within the same species. Some images are larger with sizes `1135 x 1135`, `1285 x 1285` pixels, whereas some images are smaller with sizes `154 x 154` pixels.

Data Distribution

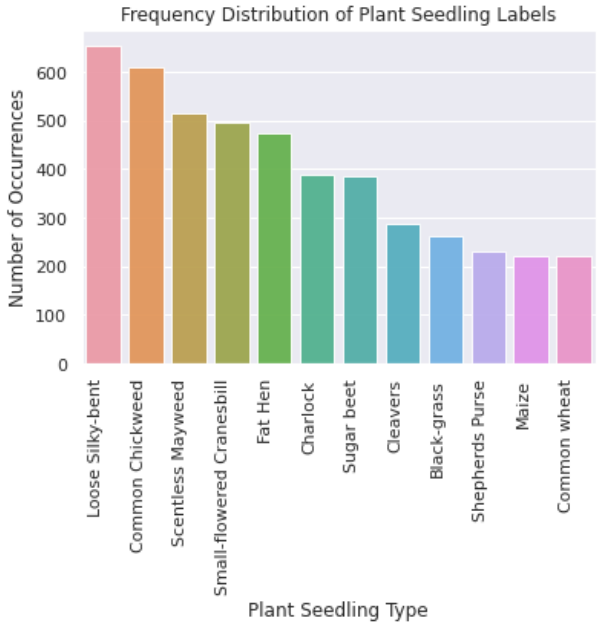


Figure 2: Distribution of Plant Seedling species in the dataset

The figure demonstrates the distribution of data, i.e., the various types of plant seedlings in the Kaggle dataset. We can observe that there is non-uniformity in the distribution where the number of training examples of some type of seedlings such as *Loose Silky bent* are more than double than some other classes such as *Common Wheat* or *Maize*. Such imbalanced distribution may result in slightly low model accuracy.

RGB Channel Analysis

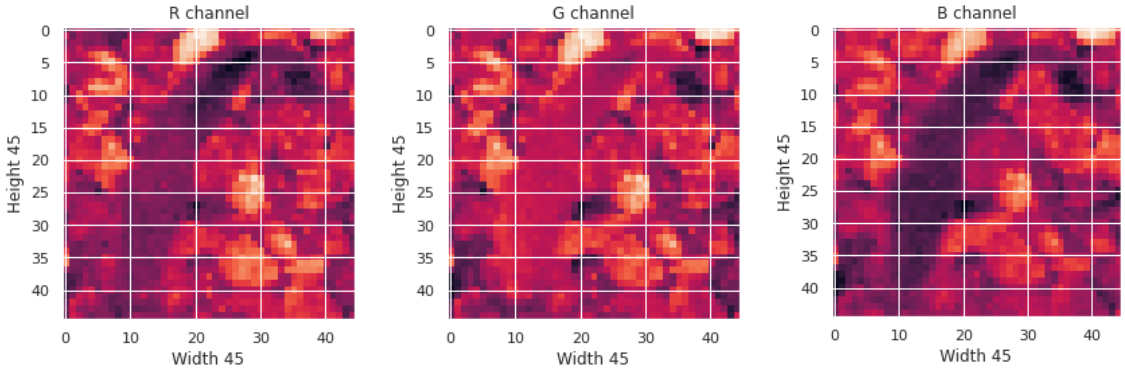


Figure 3: Red (left), Green (centre), and Blue (right) channels of a sample image

In general, machine learning and deep learning model accuracy and performance depends on the dimensionality of the data. In our case, all the images are coloured, therefore we have three channels: red, blue, and green. As our images contain plants, we can observe that the green channel encodes the most amount of relevant information for us to process. In other channels, the location of the plant is comparatively darker.

BGR Histograms

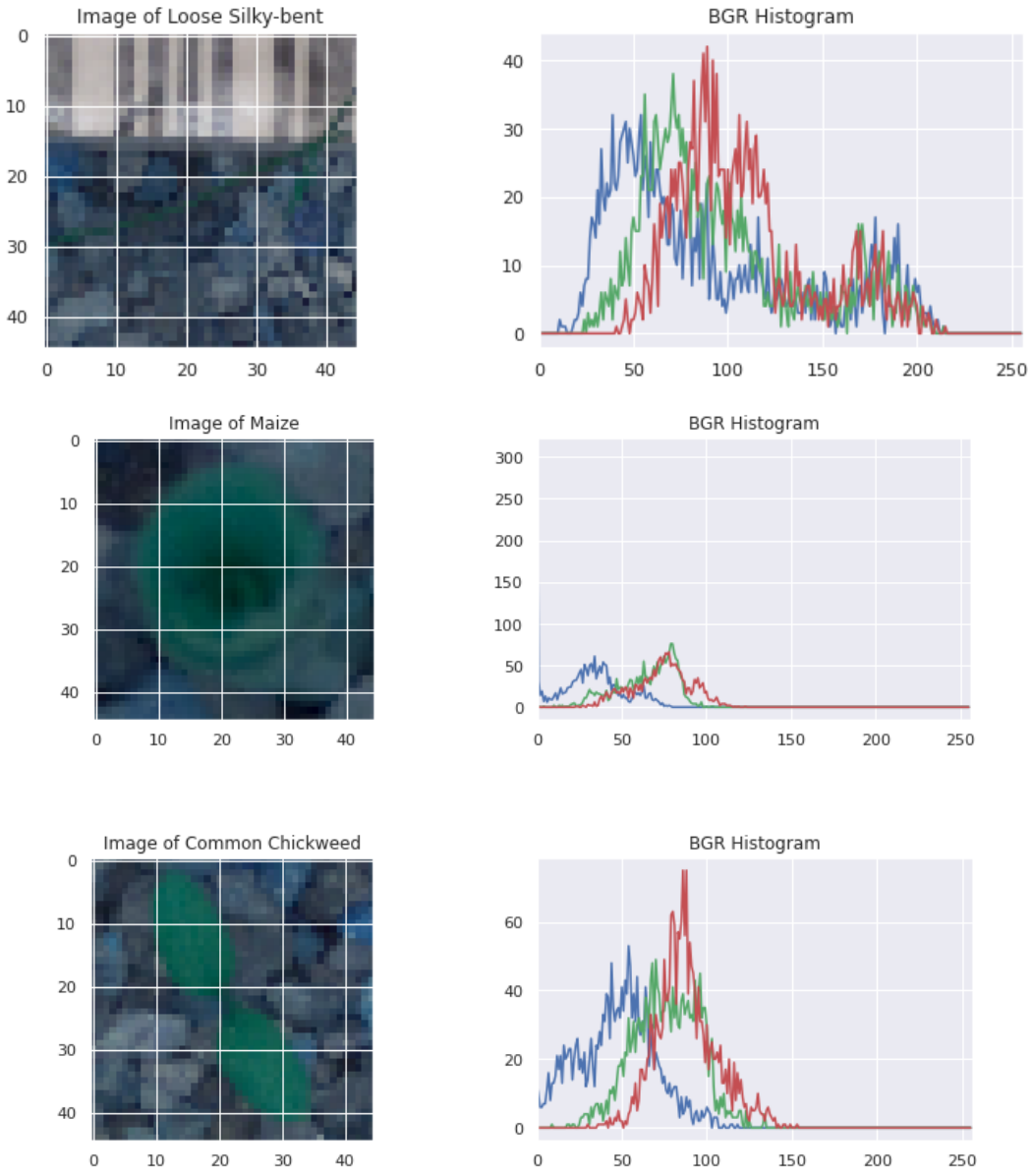


Figure 4: BGR Histograms of three types of plant seedlings

Analysing the BGR histograms provide us many useful insights about the data. It helps to identify some basic properties of an image such as tonal range, colour distribution, brightness, and contrast across different channels. In our case, we plotted BGR histograms for every plant seedling category, and picked the most peculiar categories in the figure above: *Loose Silky bent*, *Maize*, and *Common Chickweed*.

Sample images in *Maize* have an overall low BGR intensity, with values diminishing among all three channels. We will need to perform histogram equalization or change the brightness of the image to highlight features. On the contrary, sample images of *Loose Silky bent* have a spread out BGR channel with high intensities. A high red and blue channel

is a likely representative of the soil around the seedling, whereas the high greens represent the seedling itself.

Principal Component Analysis (PCA)

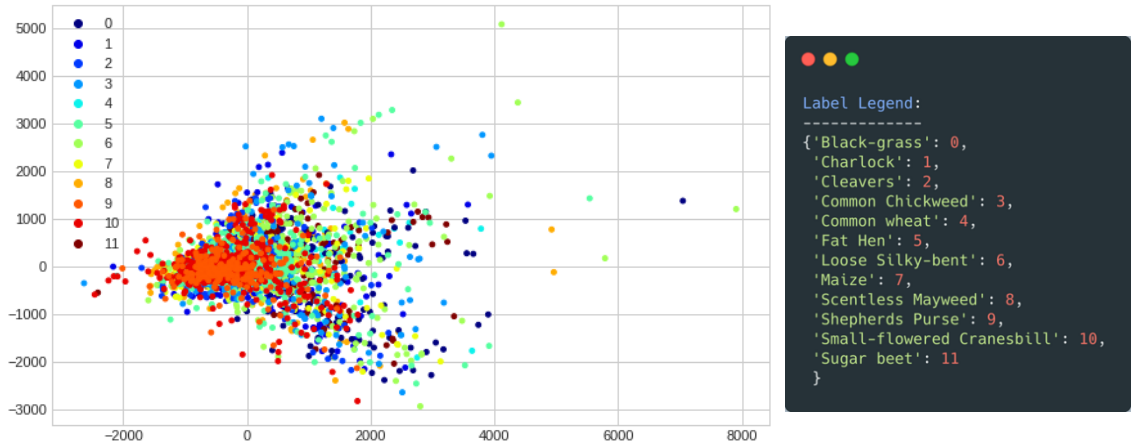


Figure 5: Usage of PCA to reduce the data to 2 dimensions for visualization

Principal Component Analysis is a good technique that helps to reduce the dimensionality of data, used here for easier visualization and analysis. The primary reason we did PCA because, our `45 x 45` pixel size images (after resizing) have 2025 dimensions. Such a high dimensionality cannot be visualized; therefore, we reduced the data to 2 dimensions.

Each colour in the graph represents a plant seedling category with the mapping shown on the right side. Firstly, all the categories are grouped closer within themselves, with a few anomalies. Secondly, we see very little segregation between the different categories, which makes this problem harder to solve for machine learning and deep learning algorithms, as the difference between the species is subtle.

Challenges of Problem

After preliminary data analysis, we conclude two main observations. Firstly, the number of data points, i.e., the number of training example sets are not large (4750 images) which presents the limitation of training very deep neural networks, since they will be prone to overfitting. Secondly, after the PCA analysis, the segregation between the different plant seedling species is minute, which may present a difficulty for learning algorithms to differentiate between the classes.

1. Our first approach to this problem was using k-Means Clustering algorithm, which is an unsupervised learning algorithm. The primary challenge is that due to its unsupervised nature, it is difficult to determine the model accuracy since the algorithm output clusters have no correspondence to the actual labels. Therefore,

while testing, the majority of labels from each cluster were taken and assigned to that cluster for model accuracy measurement.

2. Convolutional Neural Network, for good performance requires significantly larger dataset size, and hence data augmentation was critical int the task to increase the number of training images and introduce variations int the data. Further training CNN requires high computational power, hence GPUs available on Kaggle were used for training the model.
3. Given that the image data is very complex, it was difficult to create a good architecture for stacking the Xception model using a metalearner.
4. Inception – ResNet – v2 is a very recently released state of the art model, which trains very quickly on training data (reaching 99% accuracy on train data within 9 epochs). As such, the same results would not always translate into that for test data. Hence, we had to try several regularization and penalization techniques to force the model to learn specific patterns across the images.

Pre-Processing

Image Augmentation

Rotation **Height Shift** **Zoom**

Figure 6: Original images (top) and their augmented versions (bottom)

A popular technique used to enhance small image datasets is image augmentation which involves performing operations (such as rotation, dimension changing, zoom changing, and image flipping) on existing images and adding these modified images to the existing dataset. This ensures the model can adapt to a larger variation of images. This is important for our dataset since it contains organic matter which can have a wide variety of shapes, some of which may not exist in the original dataset. For the Xception model, test-time augmentation is also conducted which augments each test image using multiple operations and finds the most common prediction out of all the augmented images.

This method was used in K-Means clustering, K-Nearest Neighbours, and Xception Net, Convolutional Neural Network

Resizing and Flattening

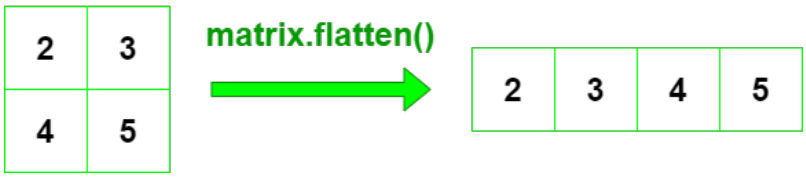


Figure 7: The numbers on the matrix on the left represent the corresponding pixel values of the image, flattened to a single row matrix on the right **[2]**

Another necessary image pre-processing step is image resizing and flattening. Almost all machine learning models expect the images to be in a fixed size. In our dataset, the images are of different sizes, so we have scaled the images to `150 x 150` pixels using the method `cv2.resize(image, (150, 150))`[1] for some of our models.

Furthermore, for some models, data usually cannot be read multi-dimensionally, so image flattening is performed to convert all the images into a 1-dimensional array using the method `image.flatten()`[2]. Such algorithms do not read the spatial features of images. An example illustrating the change in dimensionality after flattening is shown below:

$$\boldsymbol{original\ image\ dimensions}\ (\mathbf{2\ dims}): (n \times n)\ px$$

$$\boldsymbol{flattened\ image\ dimensions}\ (\mathbf{1\ dim}): (n^2)\ px$$

Equation 2: Effect of flattening on a 2D vector

Image Resizing is used in all our models whereas image flattening is used in our implementations using K-Means clustering, K-Nearest Neighbours and Support Vector Machines.

[1] `cv2.resize()` is a method of Python's OpenCV library commonly used for Computer Vision tasks.
[2] `array.flatten()` is a method of Python's Numpy library supporting high-level compute functions.

Image Normalization

$$z = \frac{x - mean\_of\_distribution}{standard\_deviation\_of\_distribution}$$

Equation 3: Standardize features by removing the mean and scaling to unit variance

All the images in the dataset are coloured, i.e., they contain three channels – R,G, & B. For every channel, we normalize the pixel values such that they have a mean value of 0 and standard deviation of 1. It is done using the `StandardScalar()` method in the `scikit-learn` library with the above-mentioned equation.

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

Feature Extraction

Colour-based Segmentation

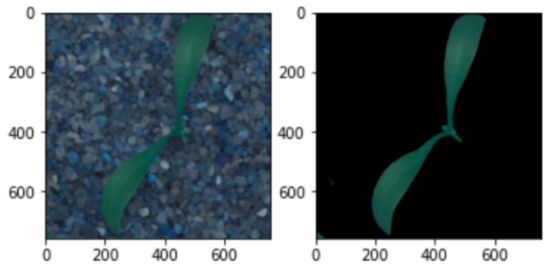


Figure 8: Original Image (left), Segmented Image (right)

We begin by performing a pre-processing step which aims at removing background noise from the image. We can see that the original images contain the soil and gravel alongside the seedling. We only want our model to learn the features present in the seedlings themselves and not to be influenced by the surrounding environment which is highly variable.

Therefore, we use the HSV (Hue-Saturation-Value) to segment all pixels in the green/yellow range. To ensure that no noise is present in the segmented image, we also sharpen the segmentation mask and make use of erosion followed by dilation. This pre-processing operation is performed for all training images as well as any testing images.

This method was used in K-Means clustering, K-Nearest Neighbours, SVM, Xception Net, and Convolutional Neural Network.

Methodology

Approach 1: K-Means Clustering

Overview

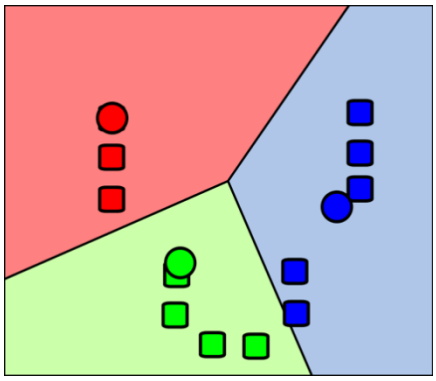


Figure 9: K-Means Clustering **[3]**

K-Means Clustering is a simple and intuitive unsupervised learning algorithm. It only takes the features of the image as an input without the labels and groups ("clusters") the images based on some metrics of similarity. The basic idea is to minimize the distance between the points in a cluster for "k" clusters (taken as a parameter).

Firstly, we choose "k", i.e., the number of clusters. In our case, "k" is 12, for 12 types of plant seedlings. Then, "k" random points are taken from the dataset as centroids and every point is assigned to one of the centroids. The centroid is then re-computed, and the two above-mentioned steps repeat until convergence.

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2.$$

Equation 4: Minimize the sum of squares where x is the data point and μ is the i^{th} cluster.

Motivation

There is an array of reasons why our team decided to start with k-means clustering algorithm. Firstly, due to its simplicity, we were able to establish a fine baseline benchmark for all our future models. Most other algorithms are harder to implement at first. Secondly, the number of clusters for plant seedling classification is deterministic which is required by the algorithm. K-Means scales up to large datasets easily and always guarantees convergence, irrespective of the dataset size.

Experiments

The images are pre-processed, and features are extracted using methods describe in the "Dataset" section. All the 4750 examples are used for training. The number of clusters is set to 12. Due to the unsupervised nature of the algorithm, once the model is trained, the cluster labels do not correspond to the real labels. Therefore, we use a helper function to

approximate the labels by firstly grouping all the images of a same cluster and then taking the *arg(max)* of their labels. We have not used test data provided by Kaggle since the cluster labels will mismatch with the real labels. The accuracy presented below corresponds to the training data.

Results

- **Train Accuracy:** 29.116%

Approach 2: K-Nearest Neighbours (k-NN)

Overview

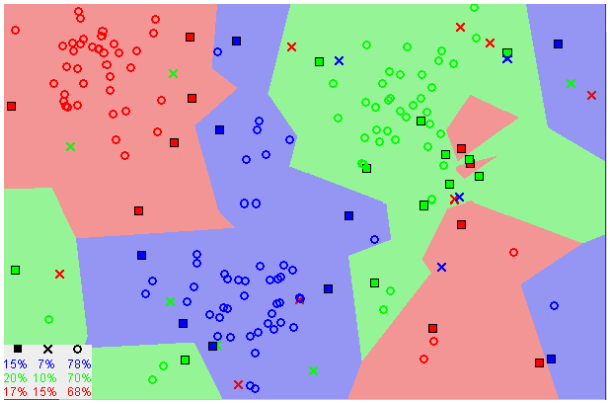


Figure 10: K-Nearest Neighbours Classification **[4]**

K-Nearest Neighbours 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 leaning 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 Euclidian 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 returned.

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

Equation 5: Measuring Euclidian distance in n-dimensions

Motivation

K-Nearest Neighbours algorithm is a natural choice to start with for our classification problem because it uses supervised learning, i.e., utilizes the labels in our training data, as well as is relatively computationally inexpensive. It is so because virtually no training happens in the training phase, only information is derived from the dataset and stored in a collection. K-NN is easy to implement for multi-class classification datasets such as ours as compared to some algorithms and it gives us the flexibility to choose from

a variety of distances to measure from: Euclidian distance, Hamming distance, Manhattan distance, and Minkowski distance.

Experiments

The images are pre-processed, and features are extracted using methods describe in the "Dataset" section above. All the 4750 examples are used for training and subsequently, the Kaggle test set (794 examples) is used for model evaluation. The parameters used to train the K-Nearest Neighbours algorithm using `Scikit-Learn` are listed in the table below.

Parameters	**# Nearest Neighbours**	5
	Weight for each point	Uniform (Same)
	Distance metric	Euclidian

Table 2: Parameters used to train the KNN algorithm

Results

- **Private Kaggle F-Score:** 0.52959
- **Public Kaggle F-Score:** 0.52959

Approach 3: Support Vector Machine

Overview

Support Vector Machine (SVM) is a machine learning algorithm which is used for both regression and classification-based problems, but mostly preferred for the latter. It is a supervised learning algorithm.

In SVM, each data item is plotted in an n-dimensional space, where n denotes the number of features one has. Each feature's value is the value of its coordinate in that space. Based on plotting these features, classification is performed by computing a hyperplane that differentiates two classes very well. The figure below helps in visualising the same.

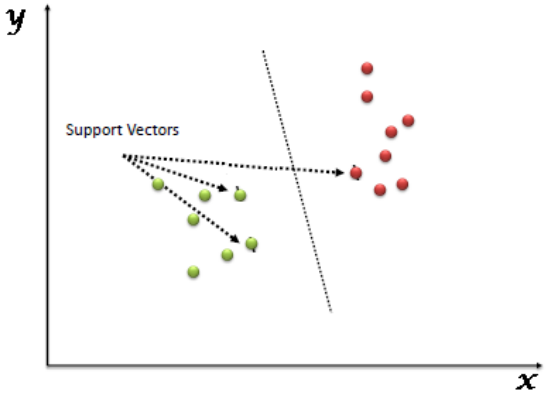


Figure 11: Hyperplane division in an n-dimensional space **[5]**

Here, hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of this hyperplane attribute to different classes.

Motivation

SVM works well with unstructured and semi-structured data like images. It is a supervised learning algorithm like K-Means algorithm, so utilising labels in our training data becomes possible. It is also relatively memory efficient as compared to other classifiers.

SVM is computationally cheaper $O(N^{2K})$ where K is no of support vectors (support vectors are those points that lie on the class margin) whereas logistic regression is $O(N^3)$. It is also easier to implement as compared to other algorithms. Kaggle's online resources were used to pre-process and run the model to obtain results

Experiments

The images are first pre-processed by segmentation. 4750 images were used for training the model, and 794 images were utilised as the test set for evaluating the model. The python library used to implement the SVM algorithm was `scikit-learn`.

Parameters	**Kernel**	Linear
	Gamma	Auto

Table 3: Parameters used to train the SVM algorithm

Results

- **Private Kaggle F-Score:** 0.63979
- **Public Kaggle F-Score:** 0.63979

Approach 4: Convolutional Neural Network

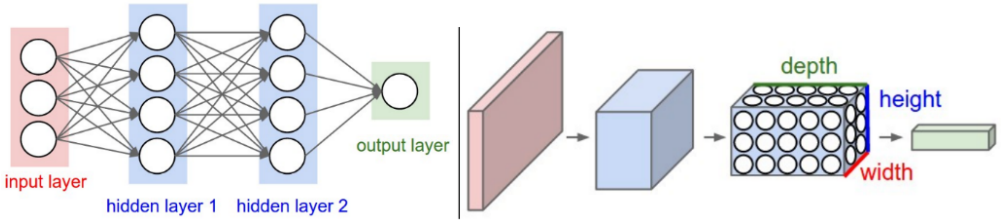


Figure 12: Vanilla 3-layer neural network (left), CNN (right) **[6]**

Overview

While deep neural networks can learn some patterns present in the input data, when the data presented is in the form of images, a lot of spatial information is lost while flattening the image into a single vector. Since we are using images of plant seedlings, it is more appropriate to use Convolutional Neural Networks (CNNs) which use convolution operations on the original images (represented as tensors) to learn both the pixel-level as well as spatial information.

A convolutional neural network is a class of deep neural networks, most applied to analysing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They are used for a variety of domains ranging

from image and video recognition, recommender systems , natural language processing, etc. [7]

Motivation

One of the primary advantages of the convolutional neural networks is that it requires minimal pre-processing compared to other image classification algorithms. This implies that the model can learn the filters from raw data, as opposed to manual feature extraction in traditional algorithms. This makes this method very attractive for the current task of image classification.

Though training deep learning models require high computational cost, and training time, with the availability of enough GPU resources in Kaggle and Google Colab, a convolutional neural network was trained for the classification task.

Experiments

The images are firstly pre-processed using segmentation. To increase the image dataset size, data augmentation was performed, with `RandomFlip()`, `RandomRotation()`, `RandomZoom()` of the `TensorFlow layers.experimental.preprocessing` library. Then the data was split into train and validation set with a with a 80:20 ratio. Before feeding into the model the data was normalised to bring all the values `int` the range [0, 1]. The model architecture is described as below. After training the model for 15 epochs, it was then tested to get predictions on the Kaggle test set. [8]

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (None, 180, 180, 3)       0
_________________________________________________________________
rescaling_1 (Rescaling)      (None, 180, 180, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 180, 180, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 90, 90, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 90, 90, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 22, 22, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 30976)             0
_________________________________________________________________
dense (Dense)                (None, 128)               3965056
_________________________________________________________________
dense_1 (Dense)              (None, 12)                1548
=================================================================
Total params: 3,990,188
Trainable params: 3,990,188
Non-trainable params: 0
_________________________________________________________________
```

Figure 13: Our Convolutional Neural Network Model

The model consists of three 2D convolutional layers, each followed by a Max pooling Layer, to reduce the number of dimension and down sample the inputs and provide basic translation invariance to the network. The RELU activation function was used to

introduce non-linearity to the model. Lastly a dropout layer was added to prevent over-fitting in the model.

Training

The model was trained with for a total of 15 epochs. The Adam optimiser was used for adjusting the weights with the Cross-Entropy function as the loss function, and the default learning rate of 0.001.

Parameters	**Epochs**	15
	Batch Size	32
	Optimizer	Adam
	Loss Function	Sparse Categorical Cross-Entropy
	Learning Rate	0.001
	Validation Data Size	20% of train set

Table 4: Parameters used to train the CNN

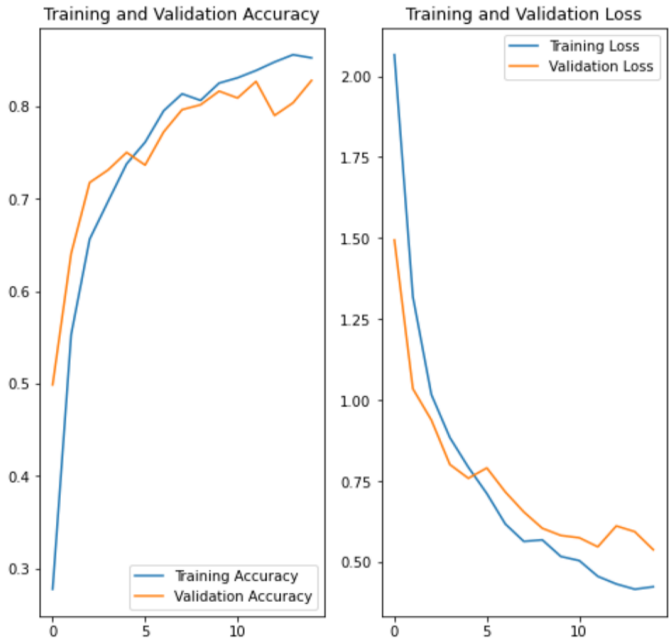


Figure 14: Training & Validation Accuracy & Loss

Results

- **Private Kaggle F-Score:** 0.60831
- **Public Kaggle F-Score:** 0.60831

Approach 5: Deep Neural Network – Xception Net

Overview

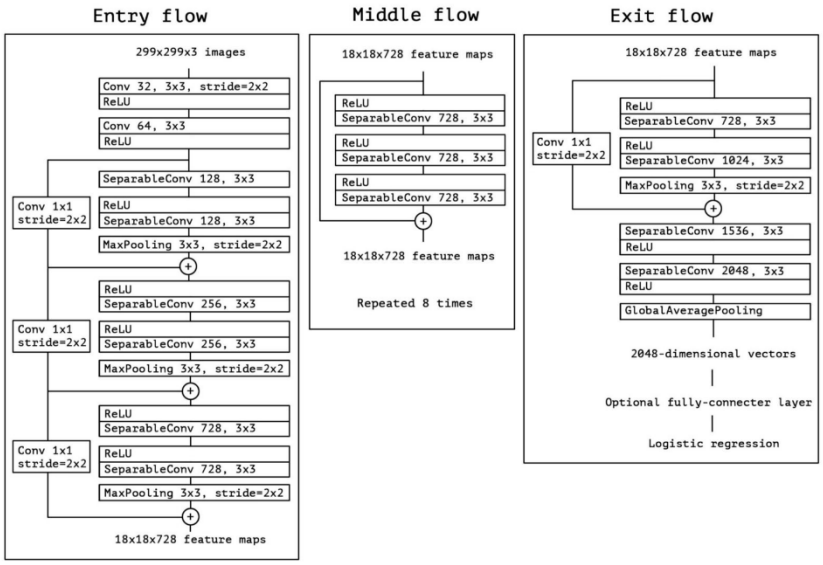


Figure 15: Architecture of the Xception model **[9]**

Xception [6] is a popular deep Convolutional Neural Network (CNN) developed by Google. The Xception model has been shown to outperform VGG-16, ResNet-152, as well as Inception V3 on the ImageNet dataset (consisting of around 14 million images). We use a pre-trained Xception model trained on the ImageNet dataset [7] and perform transfer learning on our plant seedlings training dataset.

Motivation

Transfer Learning: Transfer learning is a technique where a model trained for a specific task is reused as the starting point for another task. In our case, the Xception model has been trained to classify an image into categories, a task like seedling classification.

This is possible since a neural network essentially behaves as a feature extractor. Each layer of a neural network learns to give certain values when certain aspects are present in the given image. If we use a model which has already learnt how to extract features for hundreds of categories, we can utilize these feature extractors for our own classification task. This leads to a higher accuracy than that of a model which has been trained on just our dataset since the model is used to seeing a wide variety of data.

Extreme Inception: Xception stands for "extreme inception". In traditional CNNs, convolutional layers learn correlations across both space (across a channel) as well as depth (between the multiple channels e.g., RGB). The Xception model challenges the concept that we need to consider both at the same time.

The Inception model [10] introduced a slight separation between the two. It used 1x1 convolutions to project the original image into separate smaller input spaces. Then, a different type of filter was used to transform those input spaces into 3D blocks of data.

Xception improves upon this method. Instead of separating the data into several chunks, it transforms the spatial correlation for each output channel separately. It then does a 1x1 depth-wise convolution to capture cross-channel correlation. This new technique is known as ***depth-wise separable convolution*** which consists of a ***depth-wise convolution*** followed by a ***point-wise convolution***.

The result is that the Xception model slightly outperforms VGG-16, ResNet-152, and Inception V3. It also has the same number of parameters as Inception V3 which means it is computationally more efficient.

Experiments

Model additions: We append some layers to the Xception model in order to output our desired labels. We use Global Average Pooling instead of a fully connected layer since it spatially averages each feature map and ensures each feature map can be considered as a category confidence map. We additionally add dense layers of size 1024 (*ReLu*) and then 12 (output *SoftMax*). These are interspersed with dropout layers of probability 0.5 to prevent overfitting since each neuron has a 50% probability of not changing weights each step.

Model training: We use a batch size of 16 images since it provides the best balance between training speed and accuracy. Adam is used as the optimizer during training since it is an adaptive learning algorithm which can escape local minima and can handle sparse gradients on noisy problems. Our learning rate scheduler leads to fast convergence to the global optimum after which only minor changes are made to the weights to fine-tune the parameters for the best accuracy.

Model testing: During testing, test-time augmentation is used to generate multiple variations of the image which are used to generate outputs from different perspectives.

Parameters	**Epochs**	30
	Batch Size	16
	Optimizer	Adam
	Initial Learning Rate	0.001 (for first 6 epochs)
	Learning Rate Decay	0.9

Table 5: Parameters used to train the Xception Net model

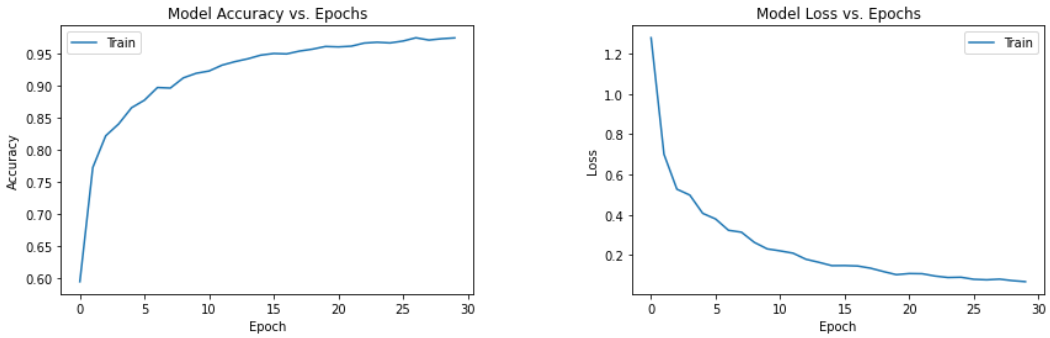


Figure 16: Xception model instance accuracy and loss vs. epochs

Result

- **Private Kaggle F-Score:** 0.97607
- **Public Kaggle F-Score:** 0.97607

Approach 6: Deep Neural Network – Inception-ResNet-v2

Overview

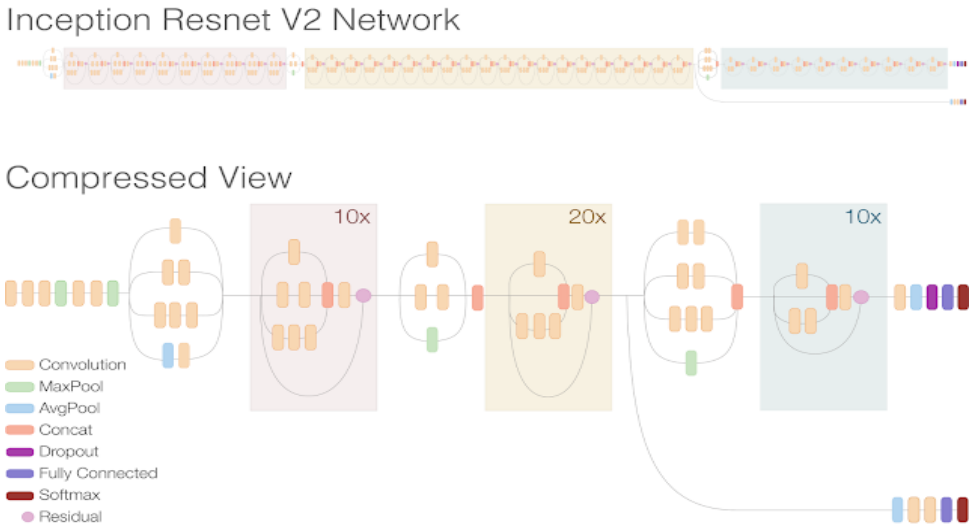


Figure 17: Architecture of the Inception-ResNet-v2 model **[11]**

Inception-ResNet-v2 borrows some of its architecture from the well-known Inception v3 model, released by Google. As the name suggests, the model encapsulates residual connections which skip immediate neurons and directly connect to the neurons of the next layers. ResNets allow for both building deeper neural networks and faster training. With the combination of the Inception model and the ResNet architecture, the model contains fewer parallel connections as before, however, the number of layers is more, i.e., the model is deeper than before. The detailed architecture can be seen from the figure above. On top of the model, with transfer learning, we have added a few dense layers with dropouts and regularization to perform the final plant seedling classification.

Motivation

Transfer Learning: Just like the Xception model explained earlier, we have leveraged the power of Transfer Learning in this method. Transfer Learning is applicable here as the ResNet model has also been trained to classify millions of images from the ImageNet dataset, a task quite similar to seedlings classification. Each layer in this architecture is a feature extractor that can be trained on the given dataset, and finetuned to output results that are accurate when it comes to plant seedling classification.

ResNet as a sophisticated architecture: In general, ResNet showed that it solved the problem of degrading accuracy when consistently increasing the depth of the neural network. It was also much easier to train, owing to the fact that it used shortcuts in the overall architecture. Through several experiments, it has been proved that ResNet has significantly enhanced the performance of neural networks with more layers. Below is a graph that shows a plot of *error%* when comparing neural networks with plain layers

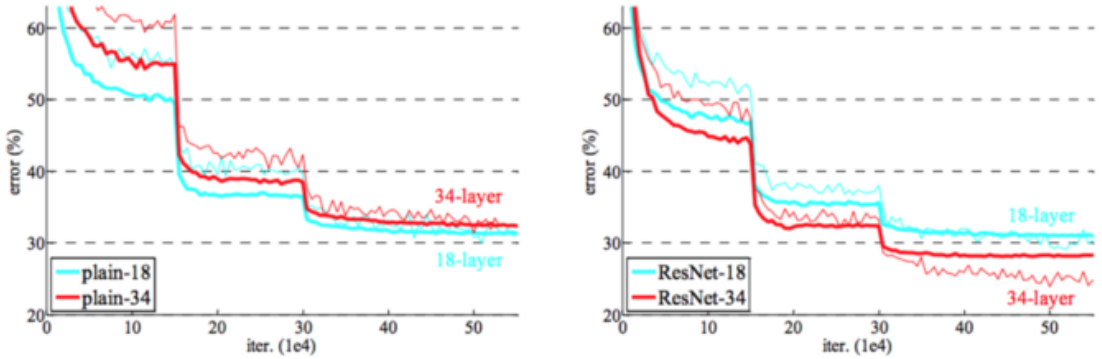


Figure 18: Comparison between plain and ResNet layers **[12]**

As is clearly evident from the above graph, the difference is huge for neural networks with 34 layers as ResNet-34 has a much lower *error%* as compared to plain-34. This significant boost in performance made ResNet architecture a natural choice for the plant seedlings classification task.

Experiments

We first apply several transformations to all the images (as discussed earlier). These image transformations are required to generalize the model as much as possible and overcome the problem of overfitting. This was followed by using the **Inception – ResNet – v2** architecture, to which we add a few custom layers, namely Dense and Dropout layers with L2 regularization. Then, we created the train, validation and test data generators, and defined 3 Keras callbacks: *ModelCheckpoint* (save the best model regularly), *ReduceLROnPlateau* (reduce learning rate when loss stops decreasing) and *EarlyStopping* (break out of training early if no further improvement).

Finally, we trained the model on the data for **100** epochs, with the callbacks being called in every epoch, to **prevent overfitting**. The training data is fed into the model in batches of size **16**, to speed by the training process. Finally, the trained model is used to make the prediction on the test data, and the output *csv* generated is used for submission.

Parameters	**Epochs**	100
	Batch Size	16
	Learning Rate Range	1e-10 to 4e-5
	Validation Data Size	1% of training data

Table 6: Parameters for Inception-ResNet-v2

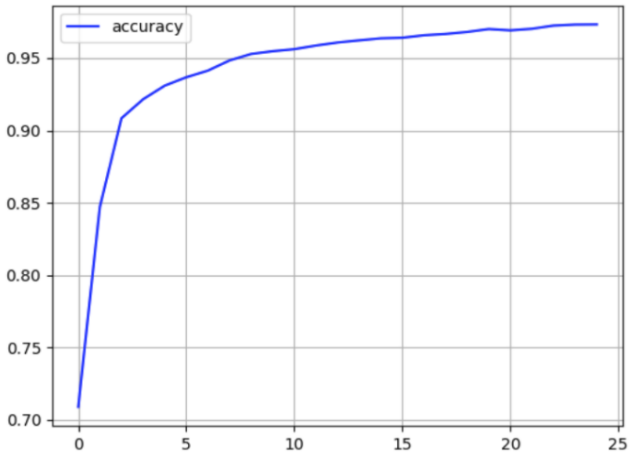


Figure 19: Inception – ResNet training accuracy vs number of epochs

Result

- **Private Kaggle F-Score:** 0.98488
- **Public Kaggle F-Score:** 0.98488

Solution Novelty

To obtain the best results in terms of classification accuracy, we tried out several models as stated earlier. Out of them, the best results were obtained through **Xception** and **Inception-ResNet-v2**, thereby showing that transfer learning from well-tested architectures which have been trained on huge datasets is one of the best options for computer vision tasks. However, plain transfer learning is not enough, as the dataset on which these architectures are trained on can have many differences from our existing dataset, making it imperative to tweak the training process to the custom dataset. We have made several changes to generate a novel, high-performance solution, which includes:

1. **Exploratory Data Analysis**: Instead of jumping straight into training a state-of-the-art model on our dataset, we carried out an initial data exploration to determine the best steps to take in the data pre-processing and augmentation step. For example, visualizing the **BGR Histograms** and **Principal Component Analysis (PCA)** helped us visualize the data and pointed us in the direction of carrying out image segmentation, to focus more on the seedlings and remove the noise due to the background. This shows that it is imperative to have a thorough understanding of the data before carrying out any data processing and training in the subsequent steps.

2. **Test time augmentation**: An application of data augmentation, this method creates augmented copies of each image in the test set, have the model make a prediction for each and finally return an ensemble of each those predictions.

```
for augmentation_count, img_augmented in enumerate(datagen.flow(input)): # Iterate after test set augmentation
    probs += model.predict(img_augmented)[0] # Add probability for each label to probs placeholder
    if augmentation_count > 100: # Quit prediction for this image if more than 100 augmentation instances
        break
model_pred[i] =  int(np.where(probs == np.max(probs))[0][0]) # Get the prediction for current model
```

Figure 20: Test time augmentation using DataGenerator

3. **Ensemble learning:** An example of ensemble learning is voting for classification tasks. We tried to use this to increase the accuracy by using outputs from multiple trained instances of the model to decide on the category of the image.

```
model_pred = np.zeros([MODEL_COUNT], dtype="int64") # Create placeholder for each model's predictions
for i in range(MODEL_COUNT): # Iterate through each of our trained models
  model = models[i]
  probs = np.zeros([12,]) # Create placeholder for predictions (size = [12,1])
  for augmentation_count, img_augmented in enumerate(datagen.flow(input)): # Iterate through each version of image after test set augmentation
    probs += model.predict(img_augmented)[0] # Add probability for each label to probs placeholder
    if augmentation_count > 100: # Quit prediction for this image if more than 100 augmentation instances
      break
  model_pred[i] =  int(np.where(probs == np.max(probs))[0][0]) # Get the prediction for current model

try:
  f.write('{},{}\n'.format(img_name, labels[np.bincount(model_pred).argmax()]) ) # Use voting to select label which was most predicted
except: # If all models give a different output
  f.write('{},{}\n'.format(img_name, labels[model_pred[0]]) ) # Use first model's output
```

Figure 21: Ensemble learning using max voting

4. ***ReduceLROnPlateau* and *EarlyStopping* Keras callbacks**: Keras provides powerful callbacks, which are functions called during the execution of each epoch. For the purposes of our training, we have used 2 important callbacks: *ReduceLROnPlateau*, which reduces the learning rate by a certain factor, when the loss stops decreasing (a larger learning rate makes the loss oscillate about the minima) and *EarlyStopping*, which stops the training early when the loss has converged and stops decreasing after a certain number of epochs. These 2 callbacks help ensure that the training loss converges with the minima but does not overfit the training data. This helped us achieve a much higher testing accuracy.

```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor = 'loss', factor = 0.4,
                          patience = 3, min_lr = 1e-10, verbose = 1, cooldown = 1)

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 22: Keras callbacks - *ReduceLROnPlateau* and *EarlyStopping*

5. **Dropouts and L2 regularization**: To prevent overfitting, wherein, the model learns patterns specific to the training data very quickly, but which do not apply on

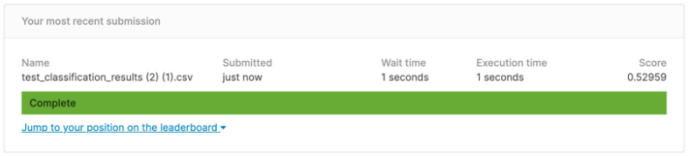
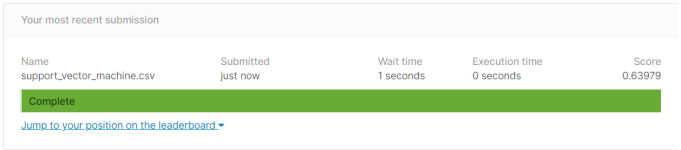
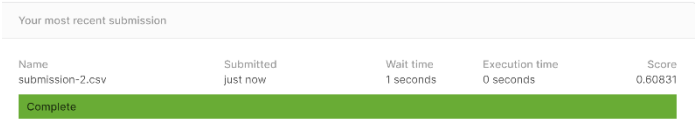
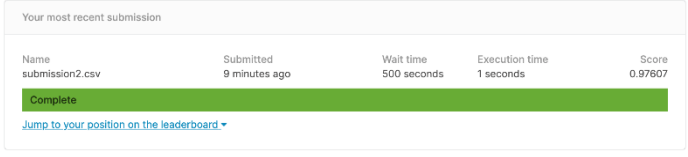
general data, we used several regularization techniques, the two most important being **Dropout** layers and **L2 regularization**. For dropout, we have kept the dropout probability as **0.5**, which means that in each epoch, the mode randomly chooses half nodes to remove from the fully connected Dense layer, for that epoch. L2 regularization on the other hand applies a penalty on the output of a layer, so that the model is forced to learn more intricate details and patterns about an image.

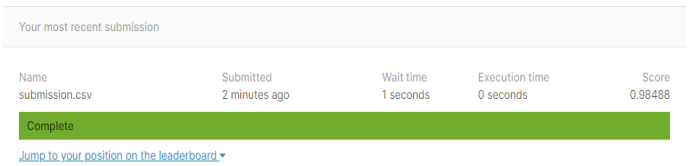
```
# add custom dropout and dense layers
dropout_1 = tf.keras.layers.Dropout(0.5)(flattened_model)
dense_1 = tf.keras.layers.Dense(128, activation = 'relu', activity_regularizer=tf.keras.regularizers.l2(1e-5))
(dropout_1)
dropout_2 = tf.keras.layers.Dropout(0.5)(dense_1)

# output of model
output_model = tf.keras.layers.Dense(12, activation = "softmax", activity_regularizer=tf.keras.regularizers.l2(1e-
5))(dropout_2)
```

Figure 23: Dropout Layers and L2 regularization

Leaderboard

Method	Private Score	Public Score	Screenshot	Public Rank (out of 833)
K-NN	0.52959	0.52959		764
SVM	0.63979	0.63979		752
CNN	0.60831	0.60831		757
Xception Net	0.97607	0.97607		193

Inception-ResNet-v2	0.98488	0.98488		54

Observations

Some of our technical observations were as follows:

- **Exploratory Data Analysis Helps –** EDA helps to get a gist of the dataset and defines the way to approach the Machine Learning problem. In our case, it helped to identify the dataset imbalance between the different classes and discover that due to the variation of RGB pixel value spread among the classes, image normalization will help. Similarly, with PCA, we visualized the corelation between the different classes which otherwise would not have been possible in an n-dimensional space.
- **Cases where Ensemble Learning doesn't help –** Both stacking and majority voting were unable to improve accuracy when multiple instances of the same architecture (Xception Net) with the same hyperparameters were used.
- **Neural networks are needed for images** – Image classification is an area where Neural Networks excel, performing much better than other Machine Learning methods like SVM, K-NN classifiers etc. This is because they very easily identify and learn unique patterns across a class of images. Such patterns work very well when identifying the images from an unlabelled dataset, as the patterns in many cases hold true. Other techniques fail to generalize the patterns for a class of images, leading to much lower accuracies. Deeper neural network architectures and associated helpful concepts of regularization, pooling etc. help it better analyse individual images and use the training knowledge on new images for more accurate classifications.
- **Overfitting on training data reduces test accuracy** – Overfitting is when a neural network model learns weights and biases that are extremely specific to the training data, but do not generalize well for other images. This means that the model achieves high accuracy while training, but this accuracy drops a lot when used on the testing data (which is different from the training data). While training deep neural networks on the provided dataset, like Xception Net and Inception – ResNet, the training accuracy very quickly reached around 98% (within 8 – 9 epochs). However, when used on the testing data, such a trained model would yield a test accuracy of only 60% - 70%, indicating overfitting. Hence, we had to introduce methods to prevent the model from learning and fitting on the provided data at an early stage, such as L2 – regularization, Dropout layers, random transformations on images etc.

Conclusion

This Machine Learning project helped us gain significant insights into the complete **Machine Learning Workflow**. We were exposed to the phases of exploratory data analysis, pre-processing of data, model training, model evaluation, and analysis. The salient features of these phases have been summarised as follows:

Exploratory Data Analysis, one of the key steps to gain a better understanding of the given data, enabled us to identify the type, distribution across different species, and dimensions of the data. The RGB channel analysis revealed the prominence of green channel in the images (being plant seedling data). Secondly, we discovered the need to perform histogram equalization due to the spread out BGR channel intensities with low values in seedlings like *Maize* and high values in *Common Chickweed*. Finally, we saw very little segregation between the different categories using PCA, which makes this problem harder to solve for machine learning algorithms.

We realised the paramount importance of **data pre-processing**, by observing stark difference in performance accuracies on data with and without pre-processing. We performed the steps of data augmentation to increase the size of the image dataset as well as variation in the plant images. Further, we resized and flattened the images to ensure a consistent and fixed size of all input images. Finally, through feature extraction, we aimed to remove all the background noise and focus only on the plant features.

For **SVM**, we noticed that there are different kinds of kernels which can be utilised and checked for regarding accuracy. For image classification purposes and in this case, the linear kernel gives best results - 0.63979 score on Kaggle, compared to sigmoid and RBF which yield 0.22166 and 0.35768 respectively.

A standard **Convolutional Neural Networks** resulted in better classification accuracy than traditional machine learning models. We experimented with different number of layers, learning rate and epochs to see the effects on classification accuracy and training time. The CNN achieved a validation accuracy of 0.8284 and a test accuracy of 0.6083.

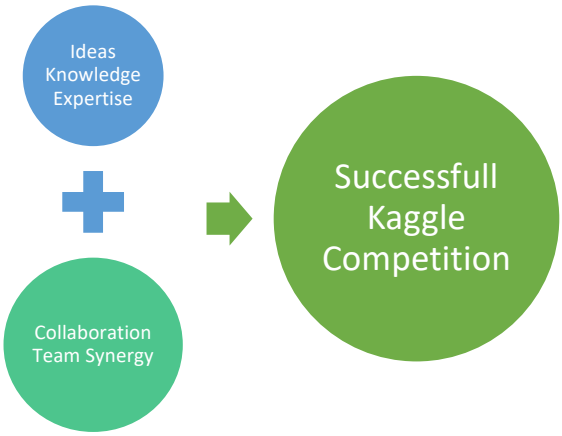
We then saw how **Transfer Learning** using existing architectures which have been trained on large datasets can provide improved results using the Xception Net and Inception-ResNet-v2 architectures. We also saw the benefits of learning rate scheduling, and test-time augmentation which helped increase the final test-set accuracy.

Lessons Learned

Throughout the semester, our team tried a row of approaches to get the highest accuracy for Plant Seedling Classification using the given dataset on Kaggle, and there were many mistakes made and lessons learned throughout this journey.

Firstly, we realised the importance of having a baseline accuracy to set as a benchmark for all other models as well the fact that one or two approaches/models cannot provide the optimal results. In our case, we tried six different approaches, ranging from kNN, SVM, to Deep Neural Networks, and tuned various hyperparameters to identify the best model.

Secondly, one enlightenment was patience with time. The Plant Seedling dataset is of 1.69GB, and with such a size, it took a lot of time to train our models, especially deep neural networks, even with Kaggle Notebooks which support both GPUs and TPUs with up to 15GB of memory each. During our experiments, some training instances took more than 4 hours at a time.



Thirdly, apart from the technical side of things, we realised the importance of team synergy, and respect for each other for a healthy collaboration. Due to the vast nature of this field of Machine Learning, it is important to recognize that every member will have a different level of understanding and knowledge of algorithms, and only via active collaboration, good results can be achieved. Such instances include sharing of optimization techniques in Deep Learning such as Dropouts, Regularisation, Normalisation, Variable Learning Rates, etc. these among teammates.

References

[1] Kaggle, "Plant Seedlings Classification," 2018. [Online]. Available: https://www.kaggle.com/c/plant-seedlings-classification.

[2] GeeksForGeeks, "Flatten a Matrix in Python using NumPy," 29 August 2020. [Online]. Available: https://www.geeksforgeeks.org/flatten-a-matrix-in-python-using-numpy/.

[3] Weston.pace, "File:K Means Example Step 4.svg," 26 July 2007. [Online]. Available: https://commons.wikimedia.org/wiki/File:K_Means_Example_Step_4.svg.

[4] Agor153, "File:Map1NNReducedDataSet.png," 30 January 2013. [Online]. Available: https://commons.wikimedia.org/wiki/File:Map1NNReducedDataSet.png.

[5] S. Ray, "Analytics Vidya," [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/.

[6] F.-F. Li, "CS231n Convolutional Neural Networks for Visual Recognition," [Online]. Available: https://cs231n.github.io/convolutional-networks.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and F.-F. Li, ImageNet: A Large-Scale Hierarchical Image Database, CVPR09, 2009.

[8] A. Krizhevsky, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in neural information processing systems,* 2013.

[9] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," 4 April 2017. [Online]. Available: https://arxiv.org/pdf/1610.02357.pdf.

[10] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *CoRR,* vol. abs/1512.00567, 2015.

[11] J. Peng, S. Kang, Z. Ning, H. Deng, J. Shen, Y. Xu, J. Zhang, W. Zhao, X. Li, W. Gong, J. Huang and L. Liu, "Residual convolutional neural network for predicting response of transarterial chemoembolization in hepatocellular carcinoma from CT imaging," *European Radiology,* pp. 1-12, 2019.

[12] H. Mujtaba, "Introduction to Resnet or Residual Network," 28 September 2020.

[13] S. Ray, “a,” [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/.

END OF PROJECT REPORT