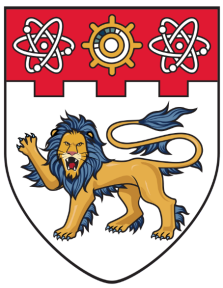

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4041 - Machine Learning

Project Topic: Plant Seedlings Classification

Best Kaggle Rank: 54 out of 833

Team Members:

Chatterjee Pramurta (U1822597G)

Singh Kirath (U1922329J)

Arya Shashwat (U1822436J)

Team Structure and Work Distribution

Team Member Names	Contribution
Chatterjee Pramurta	Exploratory Data Analysis, Data Pre-processing, Machine Learning Models, Video Presentation
Singh Kirath	Deep Learning Models, Data Pre-processing, Video Presentation
Arya Shashwat	Exploratory Data Analysis, Data Augmentation, Report Compilation

Exploratory Data Analysis

Dataset Overview

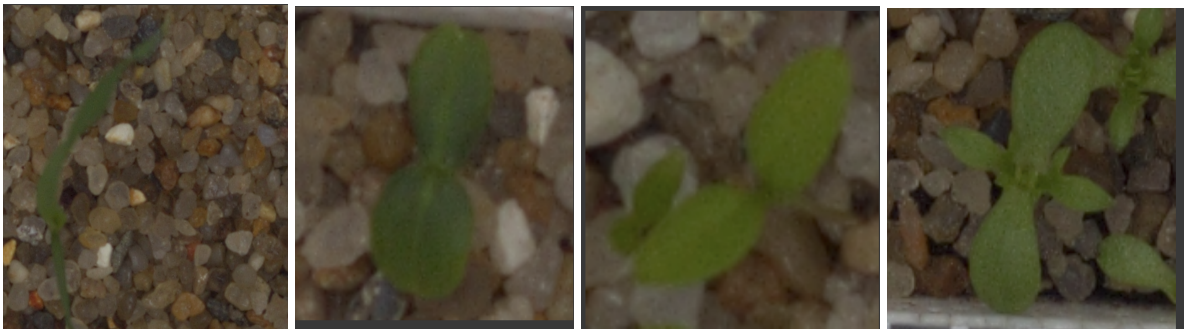

Figure 1: Some sample images from the dataset

The Plant Seedlings dataset has been prepared by The Aarhus University Signal Processing group, in collaboration with University of Southern Denmark which contains images of approximately 960 unique plants belonging to 12 species at several growth stages. There are a total of 4750 RGB images divided into 12 categories available for training our models. Also, Kaggle has provided 794 unlabelled RGB images for model evaluation and leaderboard ranking.

The image shapes aren't consistent across different types of plant seedlings. For example, some images can be as big as 1135*1135 pixels and some as small as 154*154 pixels.

Dataset Distribution

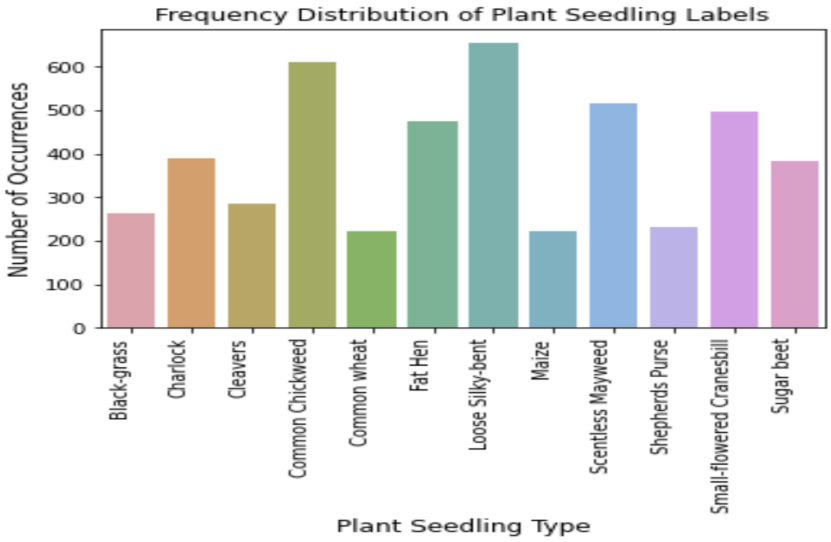

Figure 2: Frequency Distribution of Plant Seedling Labels

The bar plot of the frequency distribution of the various species of plant seedlings clearly depicts non-uniformity in the dataset with some types of plant seedlings like Common Chickweed having more than twice the number of image samples than Shepherds Purse. An imbalanced distribution of data can cause the ML models to give poor results.

RGB Channel Analysis

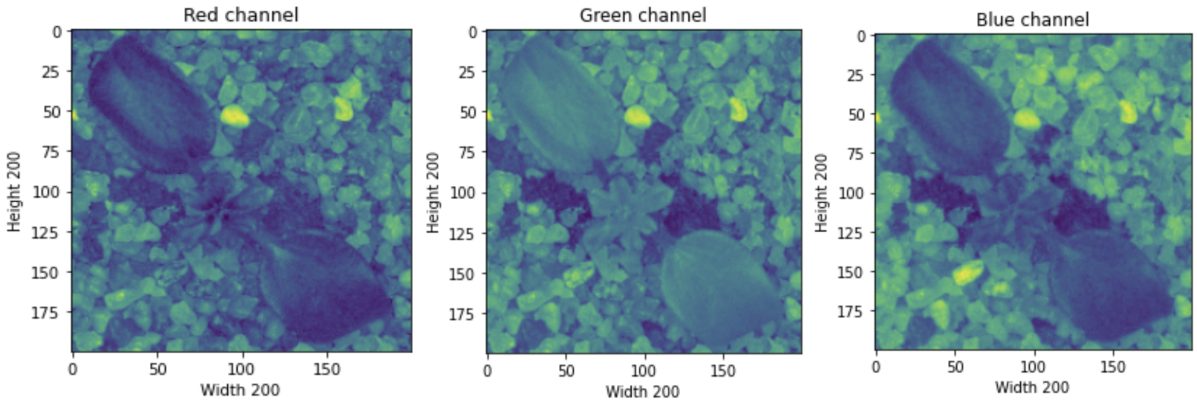

Figure 3: RGB Channel Analysis of a plant seedling

The three figures shown above represent the pixel intensities of the red, green and blue channels of a sample plant seedling of type Black Grass. From the 3 figures, we can clearly see that the green channel gives us the most relevant information about the plant seedling, as it should, since we are dealing with images of plants. In the red and blue channels, we have information about the surroundings like soil and gravel but not much about the plant seedling.

BGR Histograms

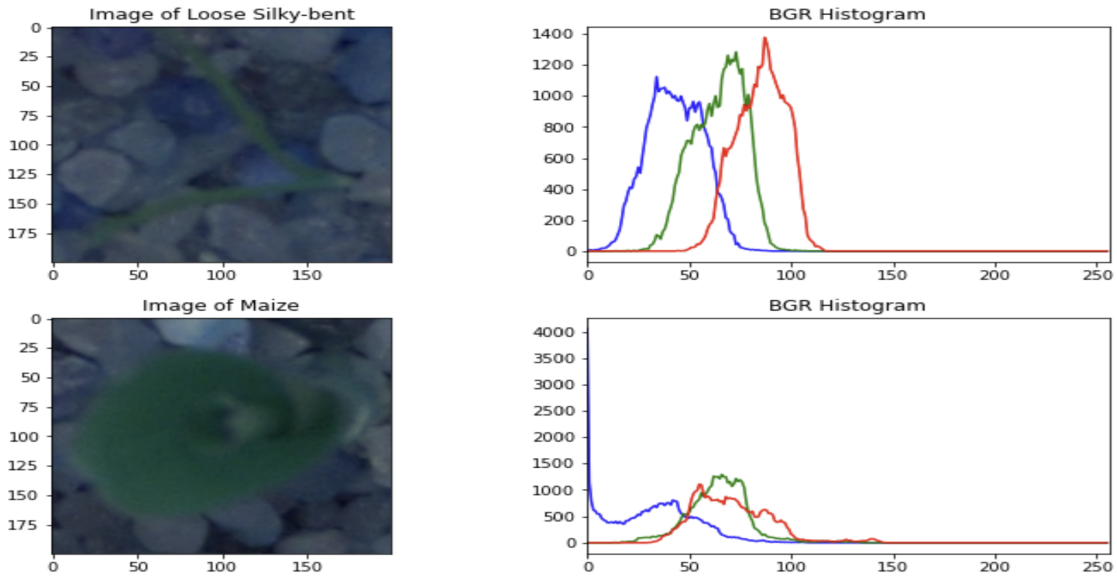

Figure 4: BGR Histograms of Maize and Loose Silky-bent Plant Seedlings

After careful inspection of the BGR histograms of different plant seedlings, many useful insights were drawn from them. It helped us identify and understand some properties of the plant seedling images like tonal range, color distribution, contrast across different channels. The 2 figures shown above are the BGR histograms of 2 plant seedling categories: Loose Silky-bent and Maize.

The images of Maize plant seedlings have overall low BGR intensities which might need us to perform histogram equalization or adjust the brightness of those images. On the other hand, the image samples of the Loose Silky-bent plant seedlings have overall high BGR intensities, where the high red and blue intensity most likely depicts the soil around the seedling and the high green intensity represents the seedling.

Principal Component Analysis (PCA)

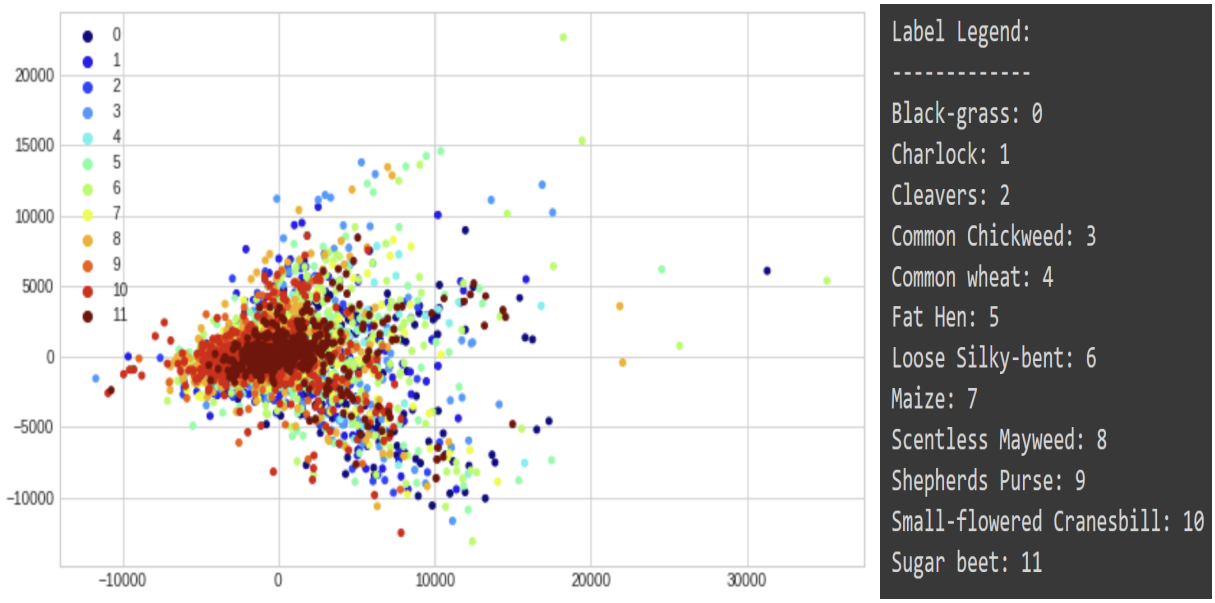

Figure 5: PCA of all the plant seedlings in 2 dimensions

We have used Principal Component Analysis to reduce the dimensionality of our data for better analysis and visualization. All the images have been reduced to 2 dimensions and there are few observations we can make from the plot. Firstly, most of the plant seedlings are grouped closely together along with a few outliers. Secondly, since there is little to no segregation between the plant seedling categories, it'll be difficult for the machine learning and deep learning algorithms to perform classification [1].

Data Pre-Processing

Image Augmentation

For the ML/DL models to be more robust and generalize better to test and validation images, and not overfit, data augmentation was performed where slightly modified copies of existing images were added to the training data. The following image augmentation techniques were followed:

- **Flipping:** The images were flipped both in the horizontal and vertical directions by 180 degrees.

- **Rotation:** The images were rotated by random degrees within the range [0,50] in both clockwise and anti-clockwise directions.
- **Width and Height Shift:** The width and height of the images were shifted by a random amount within the range of [0,20%] of the image width and height respectively, thus creating new randomly cropped images.

Feature Extraction with Color-based Segmentation

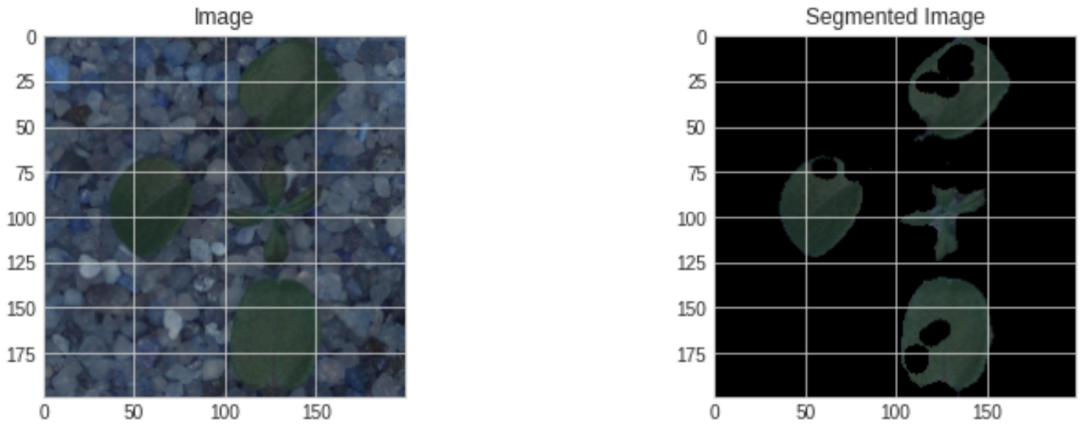

Figure 6: Original image(on the left), Segmented image(on the right)

The purpose of this pre-processing step is to filter out the background noise from the plant seedling images. Upon inspection of the image samples we notice the presence of soil and gravel around the seedling. Therefore we wanted to remove those parts of the images so that our ML/DL models can solely focus on learning the features of the plant seedlings.

The HSV(Hue-Saturation-Value) is used to segment out all the pixels in the green/yellow range. To further reduce noise in the images, we perform image sharpening by applying a segmentation mask, utilising image erosion followed by dilation.

Models Used

Approach 1: K-nearest Neighbors

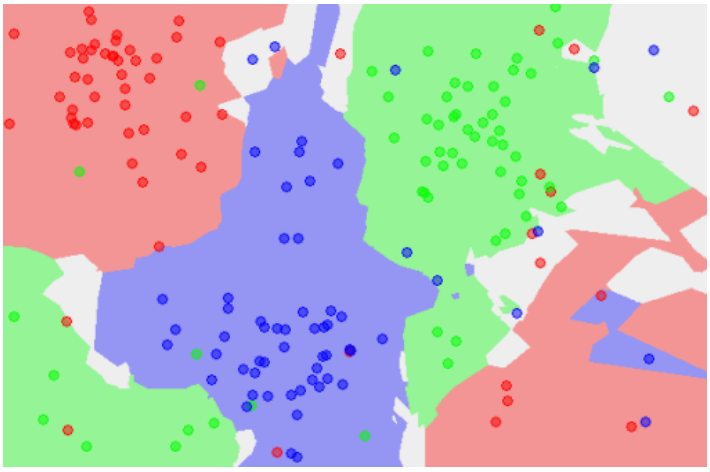

Figure 7: K-Nearest Neighbors Classification

Overview

K-nearest Neighbors is a supervised learning machine learning algorithm which has been employed both in classification and regression tasks. As shown in the figure above, the points similar to each other exist in close proximity which is the fundamental assumption of the KNN classifier. It's a lazy learning algorithm because it postpones computation until the evaluation is performed. This algorithm classifies new data points based on similarity measures (e.g. distance function). In this case, we have used the Euclidean distance measure. The assignment of a data point to a class is done by a majority vote where the point is assigned to the class having the highest count among the k nearest neighbors[2].

Motivation

We started with KNN Classifiers because we wanted to know if the training dataset was well segregated in the n-dimensional where n is the number of features used for prediction, in which case, KNN would do a fantastic job. KNN is also one of the simplest classification algorithms and relatively inexpensive when it comes to utilising computational resources. It's because virtually no training takes place in the training phase, and only information is extracted from the training data and stored in a collection.

Experiments

Before passing the images to the KNN classifier model, they were pre-processed using methods described in the Data Pre-processing section. We did a 9:1 train-validation split for training the model, where out of the 4750 image samples 4275 were used for training and 475 were used for validation. Furthermore, the 794 image samples provided in the Kaggle test set were used for model evaluation. After performing numerous experiments with different sets of hyperparameters, the following gave the best results:

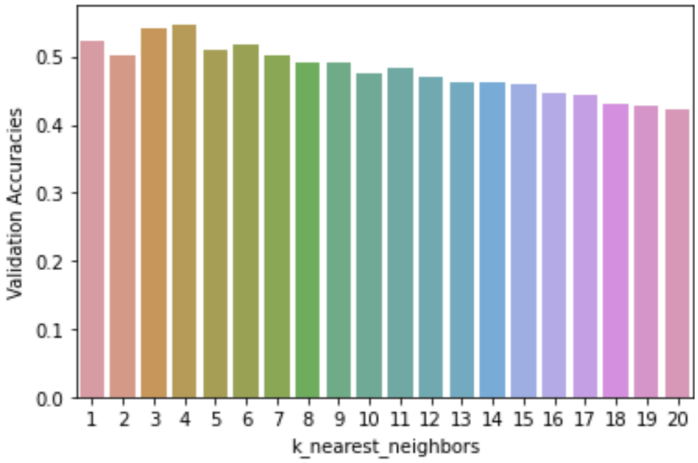

Figure 8: Value of K vs Validation Accuracy

- **Best Value of K:** 4
- **Weight for each point:** same for all
- **Distance metric:** Euclidean

Results

Figure 9: Kaggle Submission for KNN Classifier

- **Validation Accuracy:** 0.54737
- **Private Kaggle F1-Score:** 0.56360
- **Public Kaggle F1-Score:** 0.56360

Approach 2: Support Vector Machine

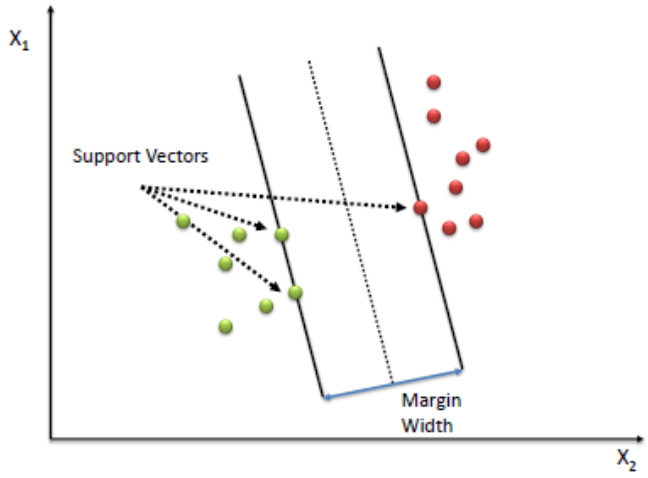

Figure 10: Support Vector Machine Classifier with the support vectors

Overview

Support Vector Machine (SVM) is a supervised machine learning algorithm which is mostly used for classification problems. The figure shown above illustrates the concept of SVM where each sample is plotted in an n-dimensional space and based on the distribution of the data points, the SVM algorithm tries to construct a hyperplane which separates the data points belonging to different classes with the maximum margin possible. The hyperplanes essentially act as the decision boundaries which are used to perform classification.[3]

Motivation

Upon doing research we found out that SVM generally tends to do well with unstructured or semi-structured data like images. Also, SVM is more memory efficient compared to other Machine Learning algorithms as they don't use all the training points but only a subset of them which are referred to as support vectors.

Experiments

Before passing the images to the SVM classifier model, they were pre-processed using methods described in the Data Pre-processing section. Also, we performed a 9:1 train-validation split for training the model, where out of the total 4750 image samples, 4275 were used for training and 475 were used for validation. Furthermore, the 794 image samples provided in the Kaggle test

set were used for model evaluation. After performing numerous experiments with different sets of hyperparameters, the following gave the best results:

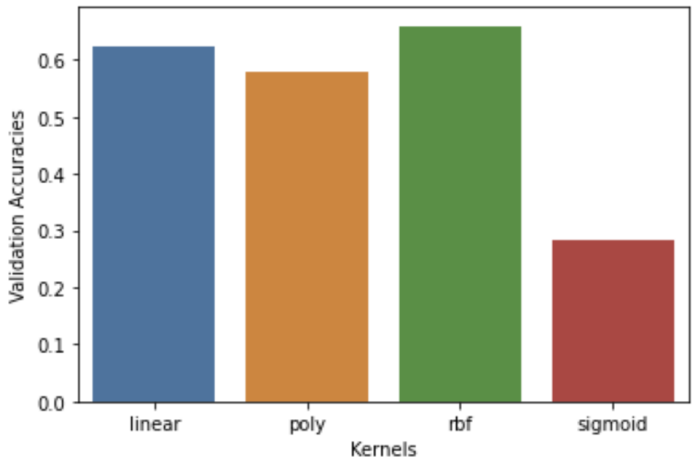

Figure 11: Kernels vs Validation Accuracy

- **Best Kernel:** rbf
- **C(Regularization parameter):** 1.0
- **Gamma:** Scaled

Results

Figure 12: Kaggle Submission for SVC Classifier

- **Validation Accuracy:** 0.65895
- **Private Kaggle F1-Score:** 0.65365
- **Public Kaggle F1-Score:** 0.65365

Approach 3: VGG-16

Overview

VGG16 is a simple and widely used Convolutional Neural Network (CNN) Architecture used for ImageNet, a large visual database project used in visual object recognition software research. The VGG16 Architecture was developed and introduced by Karen Simonyan and Andrew Zisserman from the University of Oxford, in the year 2014, through their paper "Very Deep Convolutional Networks for Large-Scale Image Recognition." [5] 'VGG' is the abbreviation for Visual Geometry Group, which is a group of researchers at the University of Oxford who developed this architecture, and '16' implies that this architecture has 16 layers.

Architecture

The original VGG16 model was trained on ImageNet dataset with an input shape of 224*224*64. The training images are passed through a stack of convolution layers. There are a total of 13 convolutional layers and 3 fully connected layers in the original VGG16 architecture. VGG16 has smaller filters (3*3) with more depth instead of having large filters. This construction consists of a total of 138 million parameters. The thing that makes the vgg16's architecture unique is that instead of having a high number of hyperparameters, it contains convolution layers with a smaller kernel size of (3*3) and a stride of 1, additionally, it maintains a consistent arrangement of the layers present in the architecture with respect to the kernel size, stride as well as pooling layers. The smaller kernel sizes allow the model to have a greater number of weight layers thereby leading to improved performance.

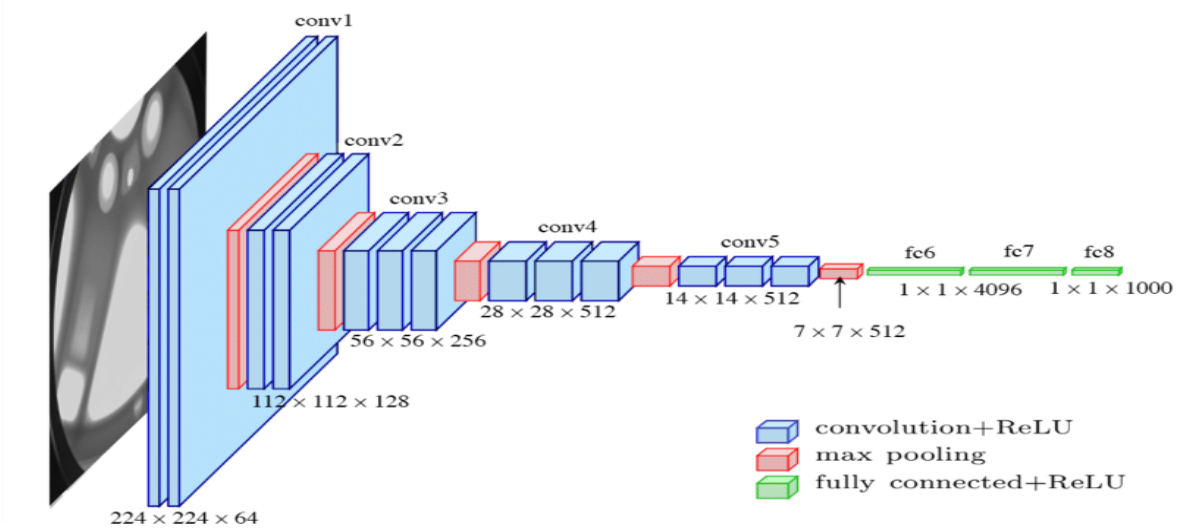

Figure 13: Original VGG-16 architecture

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

Figure 14: Layers in the original vgg 16 architecture

Layer walkthrough of the original architecture

- The first two layers are convolutional layers with 3*3 filters, and first two layers use 64 filters that results in 224*224*64 volume as same convolutions are used. The filters are always 3*3 with stride of 1
- After this, pooling layer is used with max-pool of 2*2 size and stride 2 which reduces height and width of a volume from 224*224*64 to 112*112*64.
- This is followed by 2 more convolution layers with 128 filters. This results in the new dimension of 112*112*128.
- After pooling layer is used, volume is reduced to 56*56*128.
- More convolution layers are added with 256 filters each followed by down sampling layer that reduces the size to 28*28*256.

- Two more stacks each with 3 convolution layers are separated by a max-pool layer.
- After the final max pooling layer, 7*7*512 volume is flattened into Fully Connected (FC) layer with 4096 channels and softmax output of 1000 classes.

Improvements and Modifications

In order to use the VGG16 architecture to fit our problem statement we came up with a set of changes -

1. Added a **batch normalisation** layers as it limits the effect to which updating the parameters of early layers can affect the distribution of values seen by the next layers. Moreover, batch normalisation enables the layers to learn better and more independently. [6]

2. Added dropout layers with varying dropout rates. Dropouts help us in preventing overfitting.

3. Changed the input shape to 250*250*3 as the augmented images generated from the pre-processing had a shape of 250*250*3.

4. Reduced the nodes in the output layer from 1000 to 12 as the model is supposed to make the probability predictions of 12 different classes of objects as opposed to 1000.

5. Added l2 regularisation to all the convolution layers. This helps in significantly reducing the variance of our model with a weight decay of 0.0005.

6. Call-backs added -
    - Early stopping with a patience of 40 epochs by monitoring test loss.
    - Model checkpoint to save the best model during training by monitoring test accuracy..
    - Learning rate decay by a factor of 0.9 with patience 3, the minimum learning rate was set to 0.000001 and the initial learning rate was set to 0.1.

7. SGD optimiser was used with nesterov set to True and a momentum of 0.9. SGD was preferred over adam because, during our experimentation, SGD with the above-mentioned parameters was making our model converge, while on the other hand adam optimiser did not lead to convergence due to the large number of parameters in the VGG network. The reason behind this behaviour is that SGD is more locally unstable and is more likely to converge to the minima at the flat or asymmetric basins/valleys which often have better generalization performance over other type minima.[7] A study was conducted in University of California, Berkeley where similar results were found.[8] In

this study, a set of optimisers were used in the standard vgg network alongside batch normalisation and dropouts , the test model was trained on the cifar-10 dataset. The experiment showed that adaptive methods are not superior to non-adaptive methods and more often than not, finely tuned non-adaptive methods perform better than adaptive methods. The paper however stated that the reason behind this behaviour could not be determined and suggested using finer step-size tuning in order to utilize the full potential of SGD optimiser.

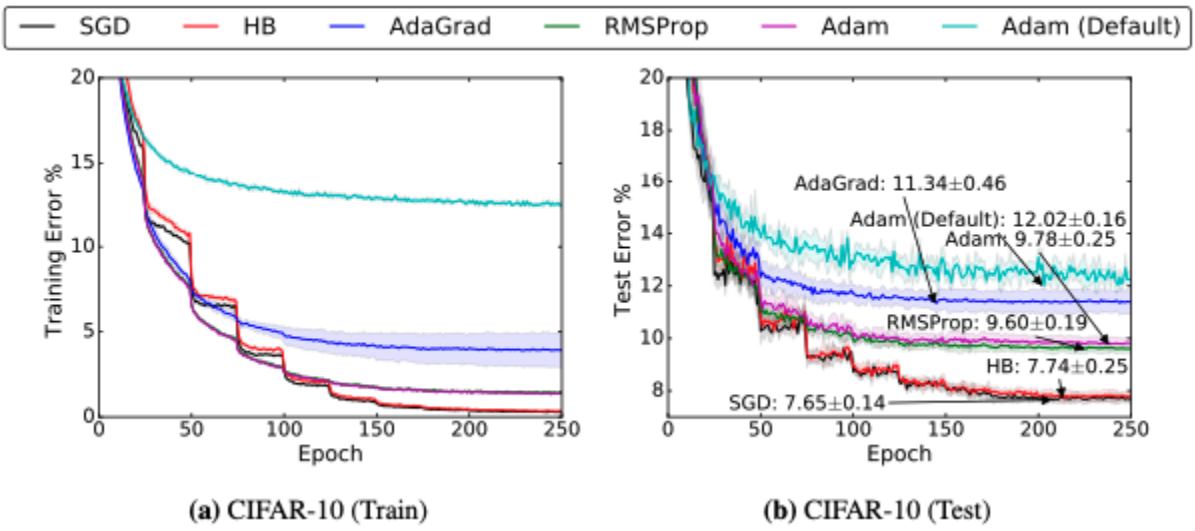

Figure 15: Results of the VGG network with different optimisers when trained on Cifar-10

8. Used a validation split of 0.08(92% training data and 8% test data). We purposefully kept the test data as little as we could in order to supply our model with more training samples for training.

9. Performed an experimentation using batch sizes in the sample space - (16,32,64,96,128,160,192,256). The best results were obtained using a batch size of 128 as the lower batch sizes had an enormous training time, while on the other hand, the validation results were not good enough for higher batch sizes.

Motivations

1. A finely tuned VGG16 model performs as well as other state of the art image classification models like Inception. Similar results were found in the study conducted in IOWA State University. The study showed that a finely tuned VGG16 had similar test

accuracies as compared to finely tuned Inception v3 as well as Inception-Resnet v2 models, when trained on a smaller dataset containing 3000 training samples, 2000 validation samples and 1000 test samples.[8]

2. VGG 16 involves less floating point instructions per seconds as compared to the other image classification models in the market like Inception,Xception, Inception-Resnet,etc. thereby making it less computationally expensive for training.

Training Results

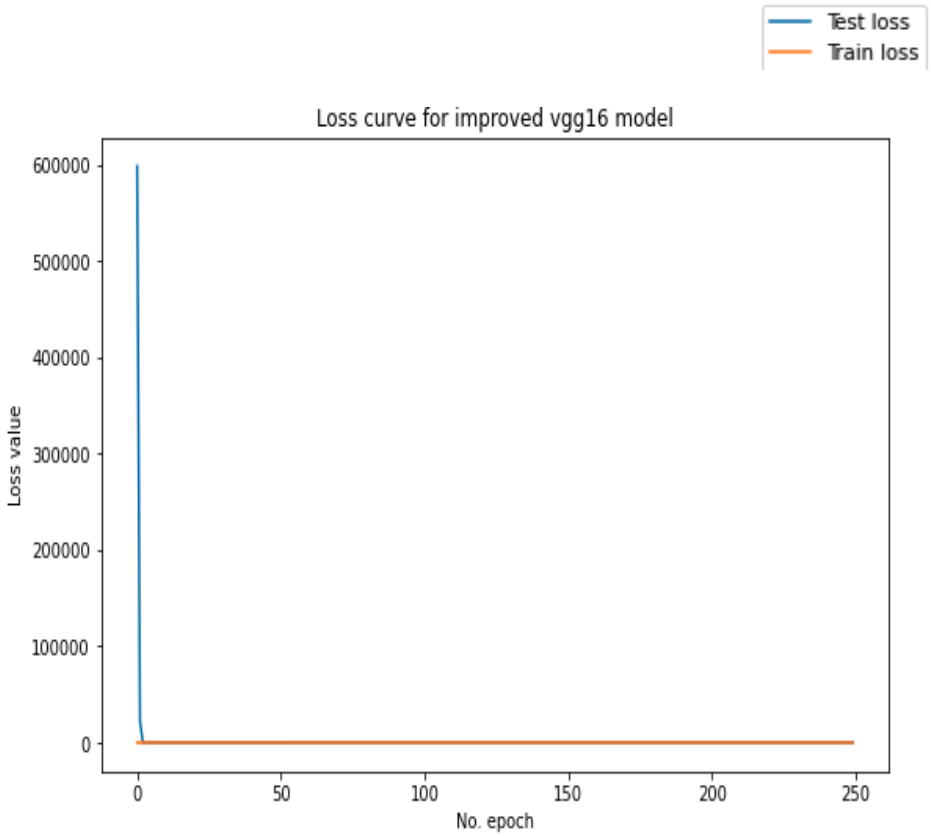

Figure 16: Training and validation loss curves for the improved vgg16 model

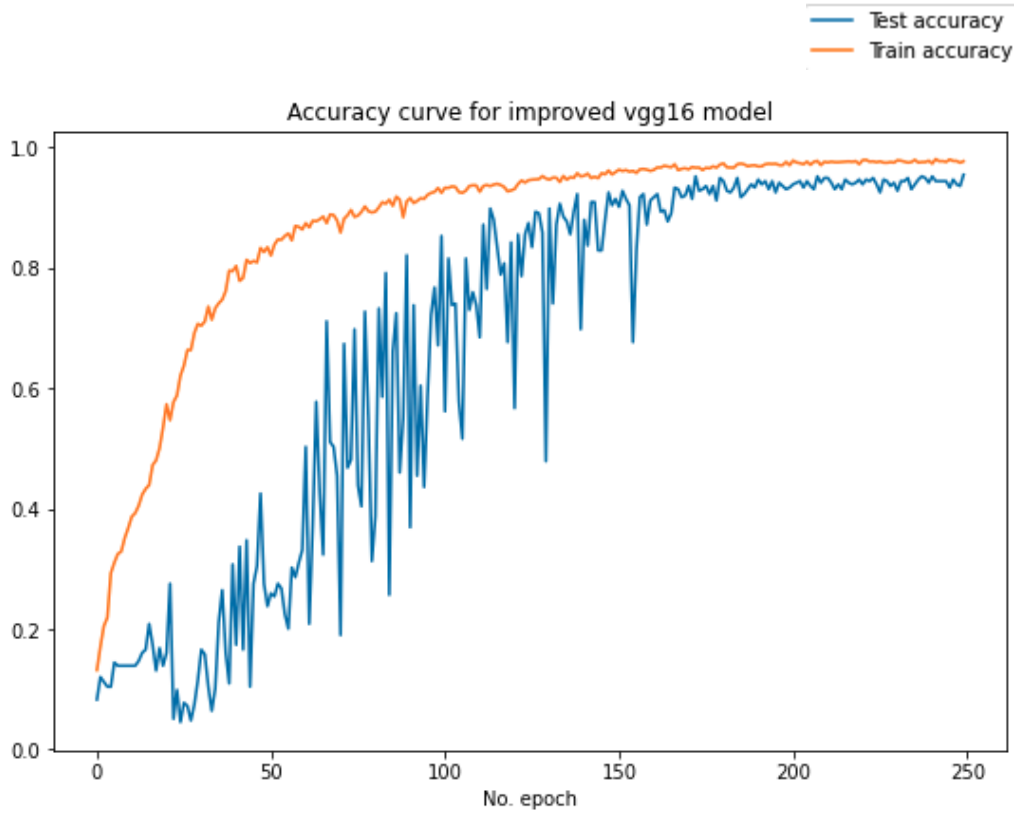

Figure 17: Training and validation accuracy curves for the improved Vgg-16 model

Results

Figure 18: Kaggle Submission for VGG-16

- **Public Kaggle F1-Score**: 97.732%
- **Private Kaggle F1-Score**: 97.732%
- **Rank obtained:** 173

Figure 19: Leaderboard Rank of VGG-16

Approach 4: Inception-Resnet-V2

Overview

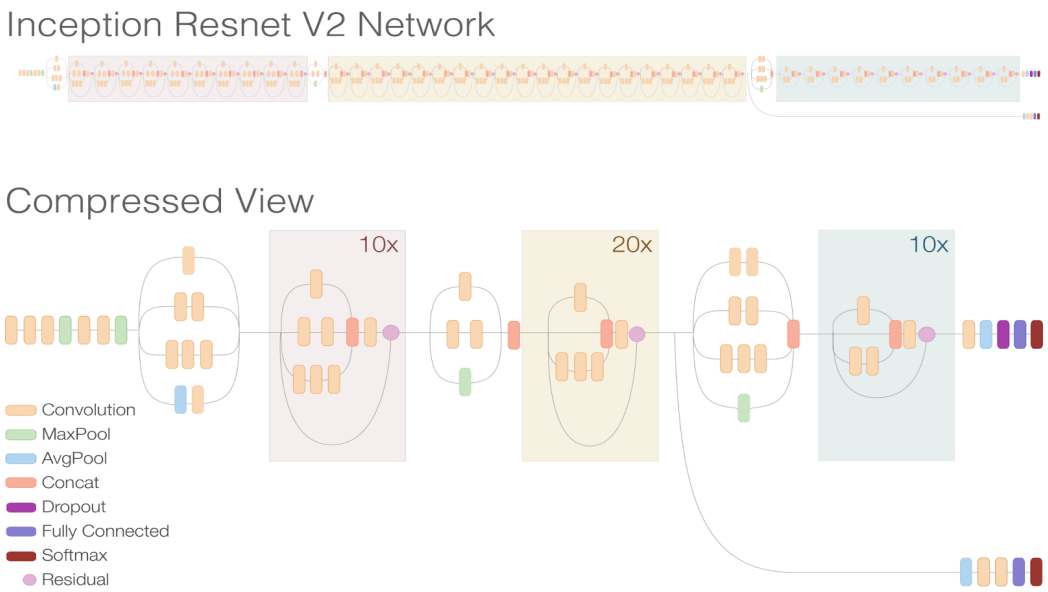

Figure 20: Architecture of the Inception-Resnet-V2 Model

Inception-ResNet-v2 is one of the newest deep learning architectures of today derived from the earlier Inception v3 model released by Google. Apart from having Inception blocks, this model incorporates residual skip connections which allowed the researchers to build deeper neural networks as the skip connections addressed the problem of vanishing gradients. As a result, the model has fewer parallel connections compared to Inception v3 but has more layers[4].

Architecture

The Inception-ResNet-v2 is a hybrid of the Inception and Resnet designs that is trained on more than a million images from the ImageNet database. The model has a total of 164 layers and was originally trained to classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network learned rich feature representations for a wide range of images. The network has an image input size of 299-by-299.

The model comprises multiple Inception-Resnet blocks stacked together along with few convolutional and pooling layers. Also, one of the most important features in the Inception-Resnet-V2 model is the scaling residuals which is explained below

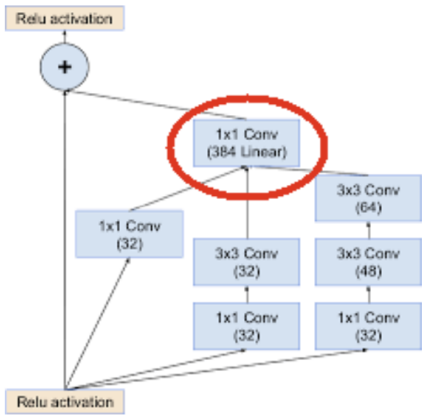

Figure 21: Inception-Resnet-V2 block

- Inception-Resnet-V2 block: It comprises an Inception block which is followed by a kernel filter (1 × 1 convolution without activation). This is done to scale up the dimensionality of the filter bank before the skip connection addition to match the depth of the input. Also, it must be noted that in the case of Inception-Resnet-V2, batch normalization is performed only on top of the traditional layers, and not on top of the summations.
- Scaling Residuals: After performing several experiments, the authors of the Inception-Resnet-V2 model found out that if the number of convolutional/pooling filters surpassed 1000 then the residual variants showed a lot of instabilities and the training accuracy just plateaued after a few epochs. This happened because the last layer before the average pooling layer produced only zeros as its output which affected training. At first, the researchers tried to lower the learning rate, and added an extra batch normalization layer but it was to no avail. Finally, when they scaled down the residuals by a factor of 0.2 before adding them to the previous layer activation, the training accuracy improved.

Motivations

- The presence of residual skip connections allows the model to not suffer from the problem of vanishing gradient and ensures smooth training.
- The presence of pointwise convolution in the Inception block greatly reduces the number of Floating point computation operations.

Modifications

- **Dense Layer:** A dense layer of 256 neurons was added to the Inception-Resnet-V2 model with a ReLU activation

- **Softmax Layer:** A softmax layer of 12 output neurons was added at the end to perform classification of the plant seedling types.

Training Results

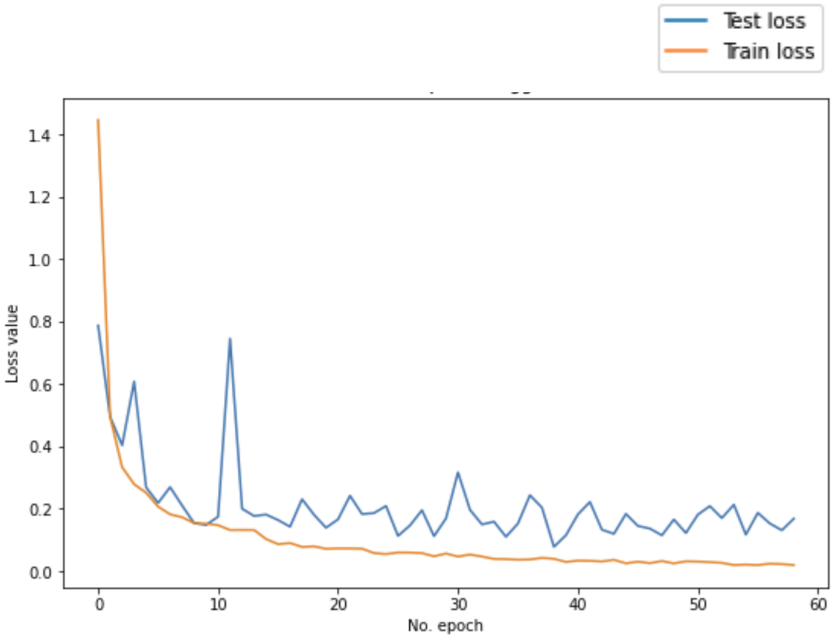

Figure 22: Training and validation loss curves for the Inception-Resnet-V2 Model

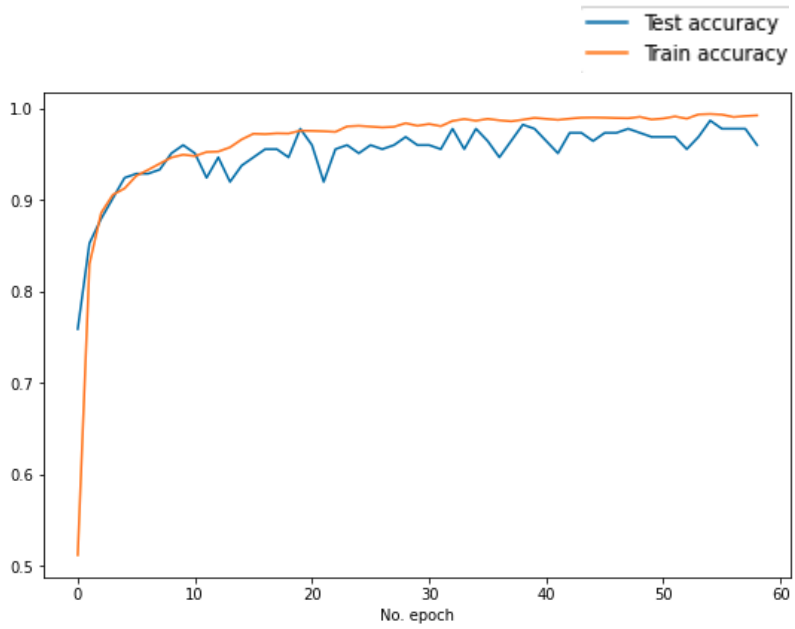

Figure 23: Training and validation accuracy curves for the Inception-Resnet-V2 Model

Results

Figure 24: Kaggle Submission for Inception-Resnet-V2

- **Public Kaggle F1-Score**: 98.488%
- **Private Kaggle F1-Score**: 98.488%
- **Rank obtained:** 54

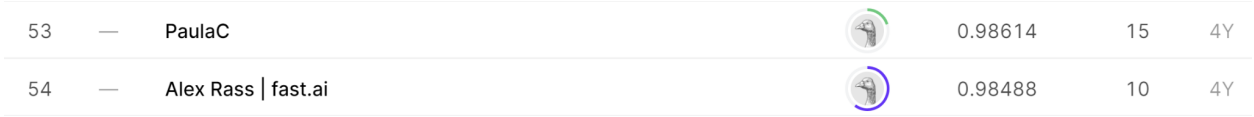

Figure 25: Leaderboard Rank of Inception-Resnet-V2

Final Leaderboard

Approach	Public Score	Private Score	Rank	Screenshot
K Nearest Neighbors	0.5636	0.5636	761	submission.csv 7 hours ago by Pramurta KNN second submission 0.56360 0.56360
Support Vector Machines	0.65365	0.65365	748	submission.csv 7 hours ago by Pramurta SVC second submission 0.65365 0.65365
VGG-16	0.97732	0.97732	173	vgg16.csv a few seconds ago by Kirath Singh 0.97732 0.97732
Inception-resnet	0.98488	0.98488	54	Submission and Description / Private Score / Public Score inception-resnet.csv 15 minutes ago by Kirath Singh add submission details 0.98488 0.98488

Conclusion

We achieved a higher accuracy with CNN based image processing models like vgg-16, Inception-Resnet as compared to the other models like Support Vector Machine and K nearest neighbors.

The accuracies obtained on the test set for were 97.732% for the vgg-16 model and 98.488% for the inception-resnet. In the end, the best results were obtained for the inception-resnet model due to the fact that it is more complex and is able to gather more information during training as compared to vgg-16 which is a slightly older architecture.

Learning points

This project gave us a great opportunity to put to test our skills and knowledge obtained in the deep learning and neural networks lectures and tutorials.

Our key learning points were:-

1. **Convolutional neural networks** – how to implement them in TensorFlow, the different hyperparameters that we can tune, what is max pooling and why we need to do batch normalisation, etc.
2. **Knowledge about State-of-the-Art Models** – VGG-16 and Inception-Resnet are high quality state of the art models used in numerous applications around the globe. It was a great opportunity to put them to test our work and learn about these industry-relevant models, and how they work.
3. **The benefit of GPUs** – Highly efficient and fast computational resources are highly useful for operations involving neural networks. Hence, we found out about online tools, and used Google Collaboratory for our project.
4. **Teamwork and patience** – This project required meetings to convey to one another our ideas and knowledge, and to help one another in times of need. We also required a lot of patience since our models took magnanimous amounts of time to train.

Challenges

In this project, we faced numerous challenges, which are mainly attributed to the way neural networks train and the resources required by CNNs to run efficiently:-

1. **Large training time** – Our models took at least 4-5 hours to train on our dataset, and sometimes even more as layers were added.
2. **Limited GPU usage** – Due to lack of GPUs in our own machines, we relied on Colaboratory by Google, which has a maximum quota of 12 hours for GPU at a time. Hence, we faced issues when our model couldn't train within the stipulated quota time.
3. **Large number of hyperparameters** – For our project we noticed that there are a lot of hyperparameters that can be tuned, and given the training time, it was challenging for selecting the best set of parameters to tune in order to obtain results on time, since we had to judiciously use online GPUs.

References

[1]. B. Ghojogh, F. Karray, and M. Crowley, "Principal component analysis using structural similarity index for images," *arXiv.org*, 25-Aug-2019. [Online]. Available: https://arxiv.org/abs/1908.09287v1. [Accessed: 27-Nov-2021].

[2]. CatalyzeX, "K-nearest neighbor optimization via randomized hyperstructure convex hull: Paper and code," *CatalyzeX*. [Online]. Available: https://www.catalyzex.com/paper/arxiv:1906.04559. [Accessed: 27-Nov-2021].

[3]. M. Claesen, F. De Smet, J. Suykens, and B. De Moor, "ENSEMBLESVM: A library for ensemble learning using support Vector Machines," *arXiv.org*, 04-Mar-2014. [Online]. Available: https://arxiv.org/abs/1403.0745v1. [Accessed: 27-Nov-2021].

[4]. C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-V4, inception-resnet and the impact of residual connections on learning," *arXiv.org*, 23-Aug-2016. [Online]. Available: https://arxiv.org/abs/1602.07261v2. [Accessed: 27-Nov-2021].

[5]. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv.org*, 10-Apr-2015. [Online]. Available: https://arxiv.org/abs/1409.1556. [Accessed: 27-Nov-2021].

[6]. S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 02-Mar-2015. [Online]. Available: https://arxiv.org/pdf/1502.03167.pdf%27. [Accessed: 27-Nov-2021].

[7]. P. Zhou, J. Feng, C. Ma, C. Xiong, S. HOI, and W. E, “Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning,” 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/f3f27a324736617f20abbf2ffd806f6d-Paper.pdf. [Accessed: 27-Nov-2021].

[8]. A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, “The Marginal Value of Adaptive Gradient Methods in Machine Learning,” 22-May-2018. [Online]. Available: https://arxiv.org/pdf/1705.08292.pdf. [Accessed: 27-Nov-2021].