Advancing Social Media Accessibility through Deep Learning

Overview:

In today's highly connected online world, our social lives are increasingly moving online. With this move comes the increased use and popularity of social media platforms such Instagram, FaceBook, YouTube, etc. These platforms are centered on the sharing of photos and videos depicting the human experience. However, these videos, and particularly photos, are largely inaccessible to those with visual disabilities. But, through the use of image recognition deep learning models, we can begin to classify these shared images and automatically provide users with a brief description. This description could then be read aloud, allowing users to more deeply participate in these online interactions.

For this project, we will be performing image classification, the first step in providing visually impaired users a comparable online experience. Therefore, we have chosen a dataset (Imagenette) that contains images one would typically see posted online. For example, many of the images contain people holding or interacting with the objects we are attempting to classify. Therefore, our algorithms must be able to focus their attention on the object, and not the person.

Dataset:

Imagenette is a subset of the larger Imagenet dataset, created by Jeremy Howard. Its purpose was to provide a smaller dataset that was suitable for rapid testing of different computer vision algorithms. It consists of the following ten categories:

1. Tench,
2. English Springer
3. Cassette player
4. Chain saw
5. Church
6. French horn
7. Garbage truck
8. Gas pump
9. Golf ball
10. Parachute

The dataset comes in three different versions (in terms of image size), 320px, 160px, and full size images. For this project, we have selected the 320px dataset in which the shortest side is resized to 320 pixels while preserving the aspect ratio.

Again, this dataset is commonly used among researchers and students as it is not as resource-intensive as a full Imagenet dataset, and can therefore be used to experiment and fine-tune algorithms.

Pre-processing:

The dataset comes in two sets, training and validation. There are 9469 images in the training set and 3925 images in the validation set. We have further split the validation set into two equal parts for validation and for testing using the “random_split” function from PyTorch. This results in a 70, 15, 15 split for training, validation, and testing. Following this, the images are resized to 256 by 256 pixel images, before converting them into tensors to be inputted into the data loaders.

Deep Learning Models:

VGG-16

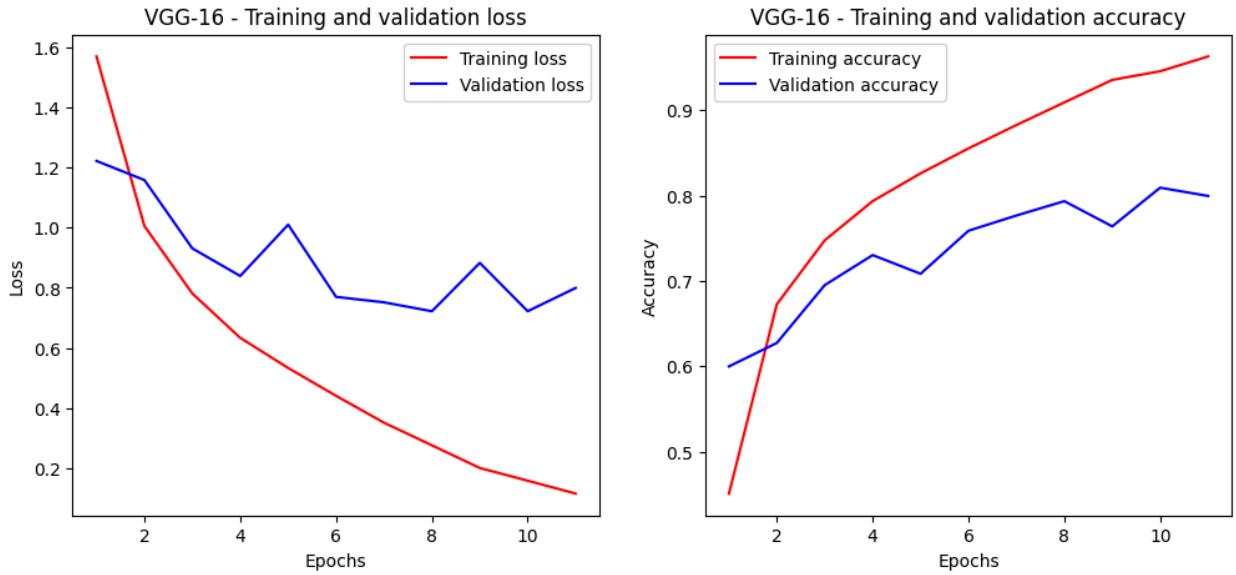
Visual Geometry Group (VGG) is a deep convolutional neural network known for its uniform architecture and simplicity. It stacks small filters in multiple layers resulting in deep networks. This enables it to capture fine details and spatial hierarchies, which makes this model highly accurate. However, it is computationally very expensive and requires substantial time and memory.

Architecture

The network consists of many convolutional layers each followed by batch normalization and ReLU activation, organized into five blocks. Each block ends with a max-pooling layer, which reduces spatial dimensions. Then, the feature maps are flattened and passed through three fully connected layers.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Performance



The network performs well on the dataset for the selected hyperparameters and effectively learns, which can be seen in the plot above. The training loss decreases and accuracy increases, with training accuracy at 96.31%. The same trend can be observed in validation loss and accuracy with minor fluctuations. The validation accuracy stands at 80.94%.

Resnet-34

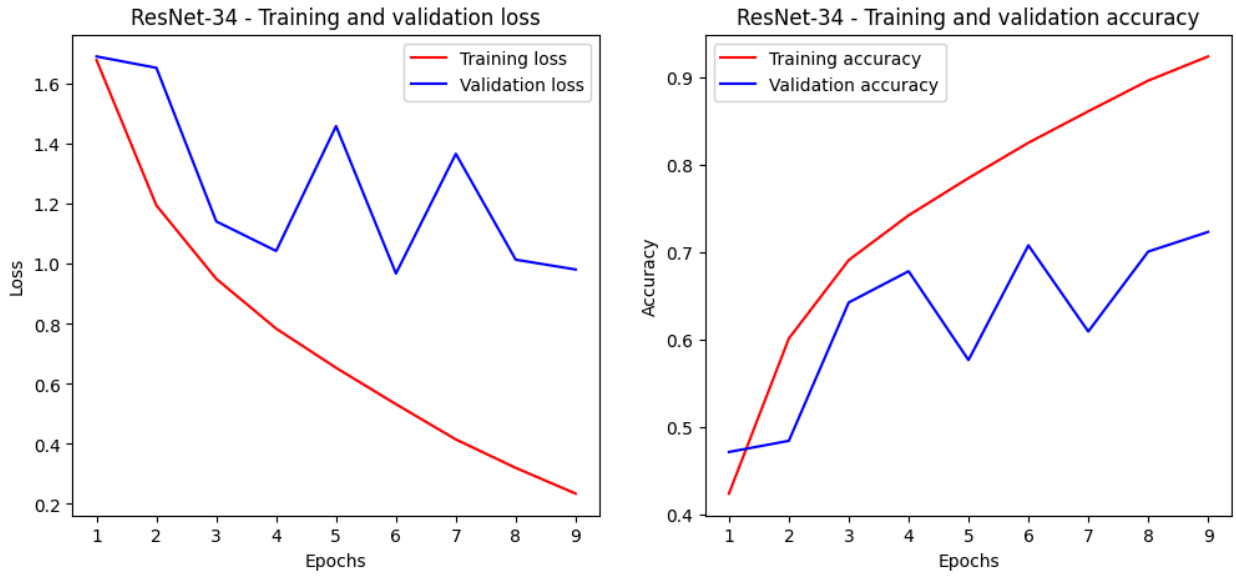
Residual Network (ResNet) is a revolutionary deep learning model that introduces residual connections, or shortcuts, that skip one or more layers, allowing it to address the vanishing gradient problem, and to train the network effectively with fewer layers. It has fewer parameters as compared to VGG. However, it is more resource intensive than MobileNet. The architecture of this model makes it extremely suitable for classification tasks with efficient training and high accuracy.

Architecture

ResNet-34 is composed of several building blocks called residual blocks. Each residual block consists of two convolutional layers, batch normalization layers, and ReLU activations. It also includes shortcut connections to skip layers. It is followed by adaptive average pooling and a fully connected layer. The following image shows different versions of ResNet, and we have referenced it to build the ResNet-34 model.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Performance:



The above graph shows the performance of the ResNet-34 model over nine epochs before it meets the early stopping condition. The training loss consistently decreases indicating effective learning. Also the training accuracy goes up to 92.36%. The same is confirmed through the model's performance on the validation set with the accuracy reaching 72.32%.

MobileNet

MobileNet is an efficient convolution neural network designed for mobile and embedded vision applications. Unlike traditional VGG, DenseNet, and ResNet, MobileNet uses depth wise separable convolutions and pointwise convolutions to decrease the number of parameters and operations. This significantly reduces the computational load and memory requirements while maintaining high accuracy, making it highly efficient.

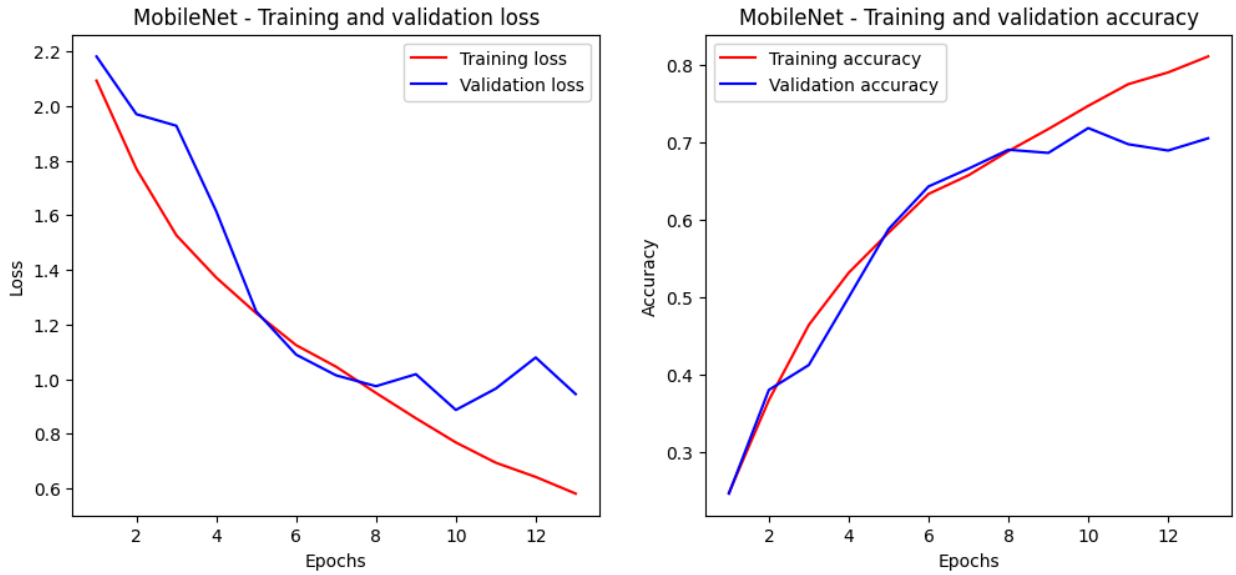
Architecture

The architecture of MobileNet we have implemented is provided in the table below. Instead of using standard convolution, MobileNet divides it into depthwise and pointwise convolutions as mentioned above. The network starts with a normal convolutional layer followed by multiple depthwise separable convolution blocks. Each block consists of a depthwise convolution followed by pointwise convolution. This is then followed by batch normalization with the ReLU activation function before passing it down to average pooling layer and fully connected layer.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Performance



This runs for thirteen epochs before meeting the early stopping condition. It can be inferred from the graph that the model is learning effectively as training loss consistently decreases and training accuracy consistently increases. The same is validated by the model's performance on the validation set with its accuracy reaching 71.87%.

DenseNet-121

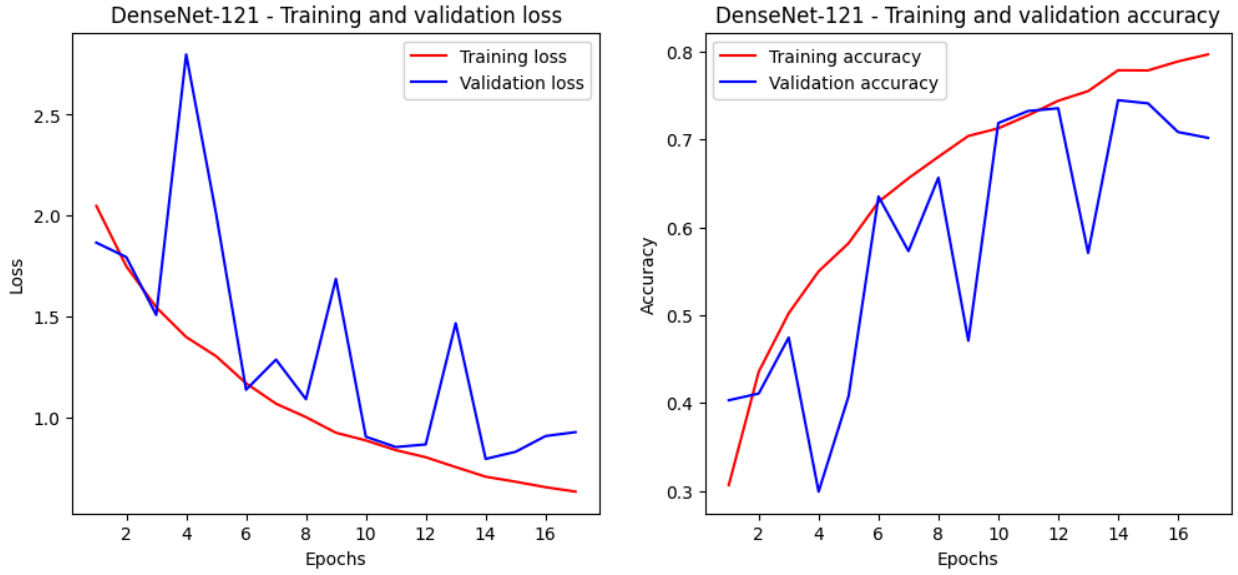
Dense Convolutional Network (DenseNet) introduces dense connections between layers, where each layer receives inputs from all preceding layers enabling feature reuse. DenseNet is more parameter-efficient than VGG and ResNet because of these connections, but is still more computationally expensive than MobileNet.

Architecture

DenseNet-121 is composed of several dense blocks, where each dense block includes multiple layers with dense connections. These connections allow each layer to receive input from all preceding layers. The network also has transition layers in between, which downsamples and reduces the spatial dimensions of the data. It starts with a convolutional layer followed by max-pooling, dense blocks with transition layers, batch normalization, and a fully connected layer. The figure below shows different versions of DenseNet. We have implemented the 121 layer version with a bn_size of 4 and growth rate of 4. We experimented with a larger growth rate of 8, however, this resulted in lower accuracies.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Performance



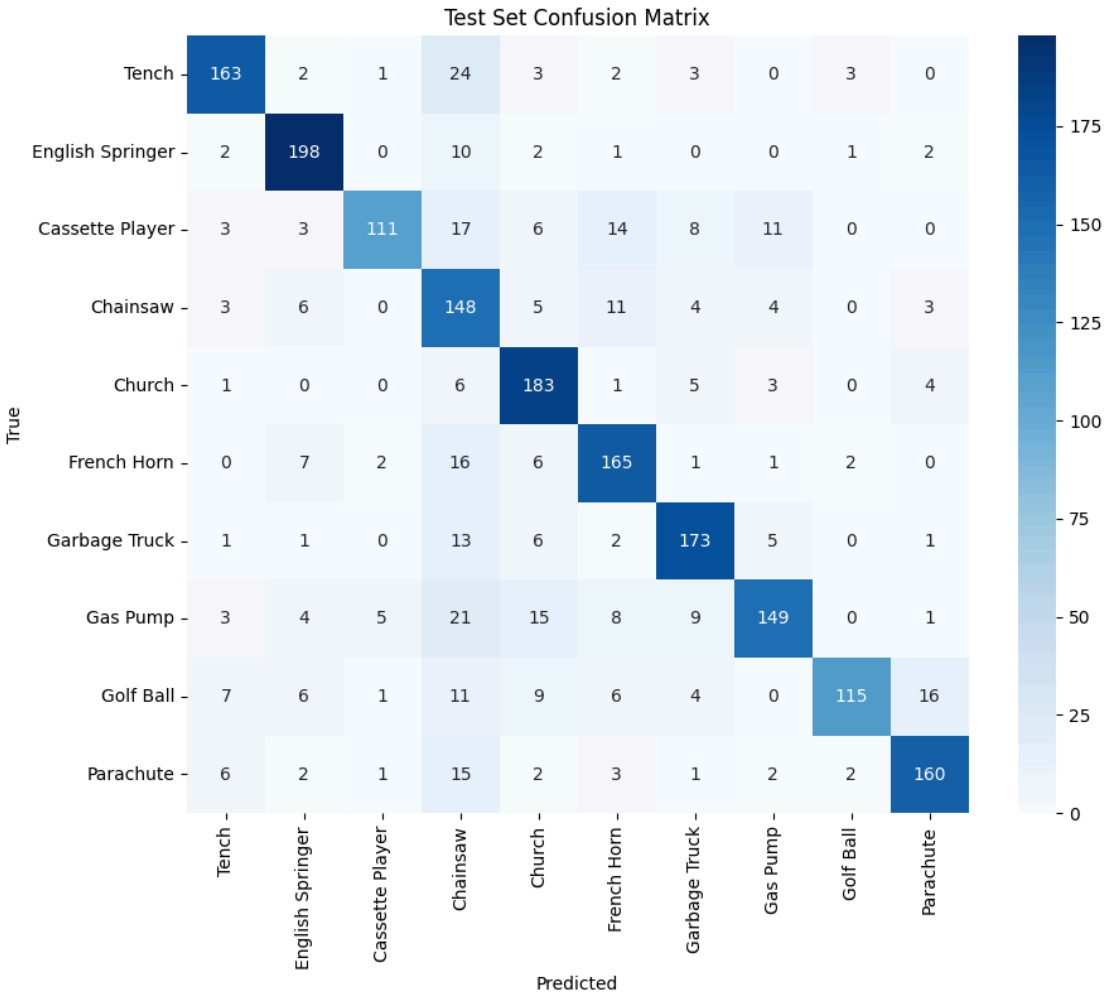
The above plot shows performance of DenseNet over 17 epochs before meeting the early stopping condition. It shows effective learning with increasing accuracy and decreasing loss, with training accuracy at approximately 80%. The validation loss and accuracy fluctuates, but in general there is an upward trend with validation accuracy reaching its peak of 74.46%.

Performance on Test Set

This section includes different performance metrics calculated for each model that was implemented. These metrics include:

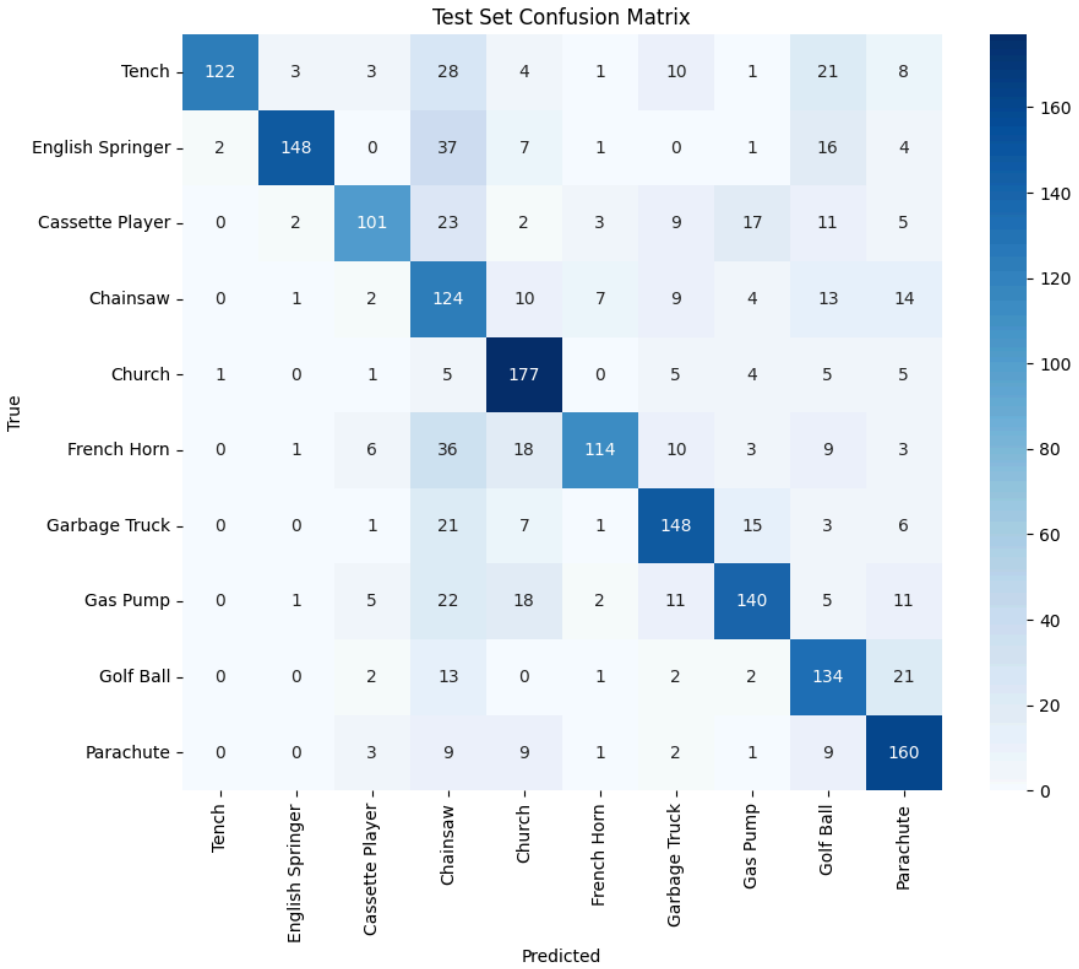
- **Precision:** The number of true positive predictions divided by all positive predictions (correct and incorrect). Therefore, it measures the accuracy of the positive predictions.
- **Recall:** The number of true positive predictions divided by all actual positive instances (true positives and false negatives). It is a measure of a model's ability to capture all positive instances.
- **F1-score:** The harmonic mean of "Precision" and "Recall".
- **Support:** The number of samples of each class
- **Confusion Matrix:** This table provides a detailed breakdown of model's performance by different classes present in the dataset. It indicates the count of true positives.

VGG-16



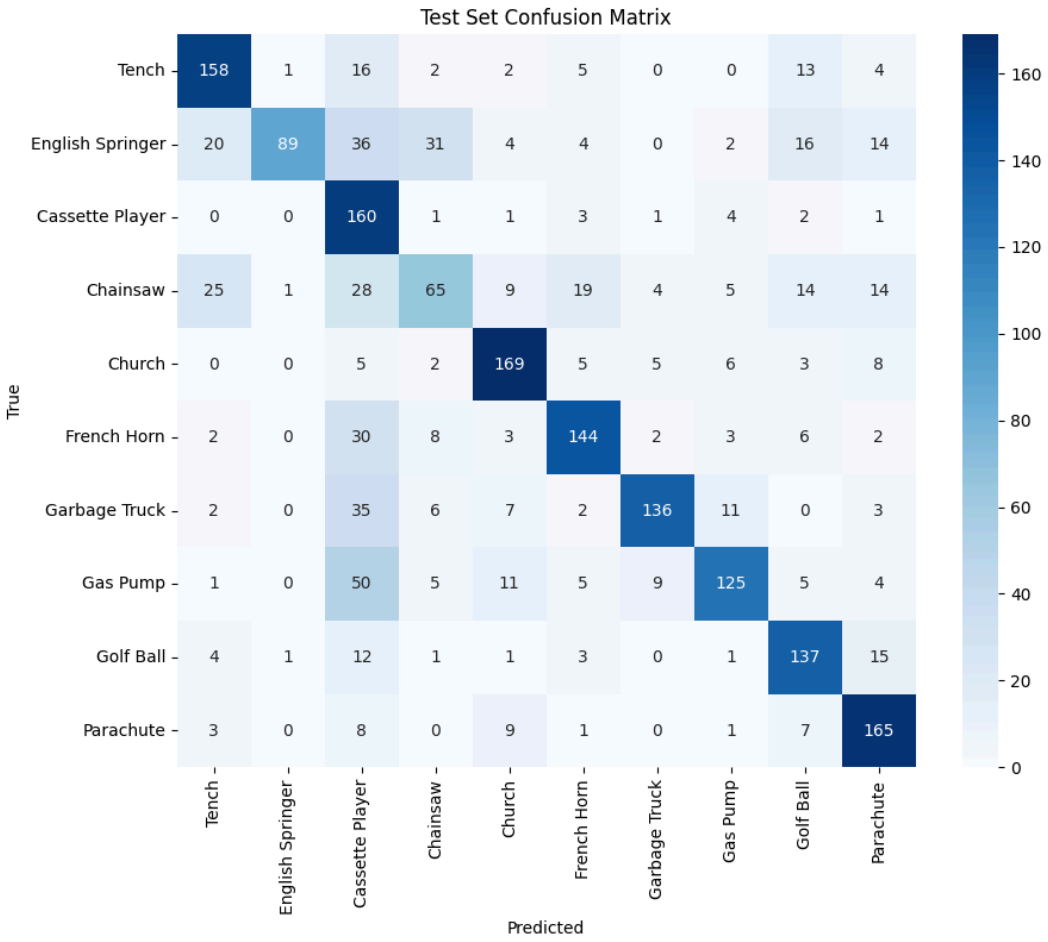
	precision	recall	f1-score	support
0	0.86	0.81	0.84	201
1	0.86	0.92	0.89	216
2	0.92	0.64	0.76	173
3	0.53	0.80	0.64	184
4	0.77	0.90	0.83	203
5	0.77	0.82	0.80	200
6	0.83	0.86	0.84	202
7	0.85	0.69	0.76	215
8	0.93	0.66	0.77	175
9	0.86	0.82	0.84	194
accuracy			0.80	1963
macro avg	0.82	0.79	0.80	1963
weighted avg	0.82	0.80	0.80	1963

ResNet-34



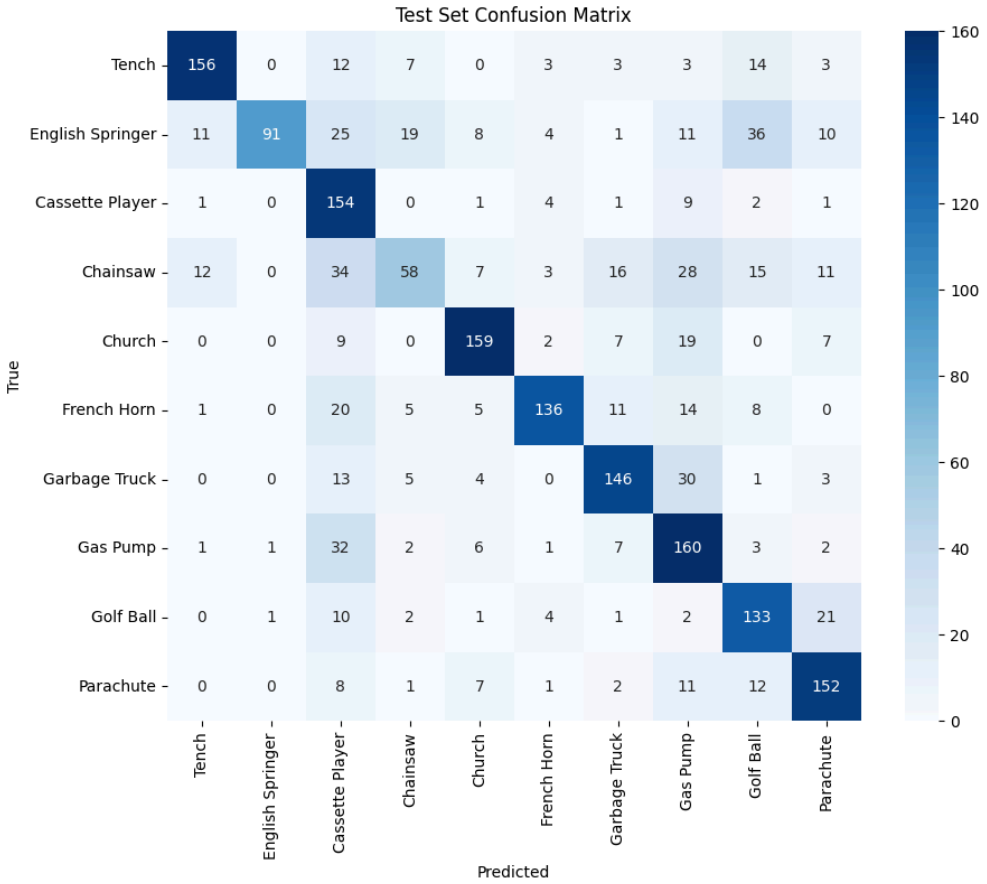
	precision	recall	f1-score	support
0	0.98	0.61	0.75	201
1	0.95	0.69	0.80	216
2	0.81	0.58	0.68	173
3	0.39	0.67	0.49	184
4	0.70	0.87	0.78	203
5	0.87	0.57	0.69	200
6	0.72	0.73	0.73	202
7	0.74	0.65	0.69	215
8	0.59	0.77	0.67	175
9	0.68	0.82	0.74	194
accuracy			0.70	1963
macro avg	0.74	0.70	0.70	1963
weighted avg	0.75	0.70	0.70	1963

MobileNet



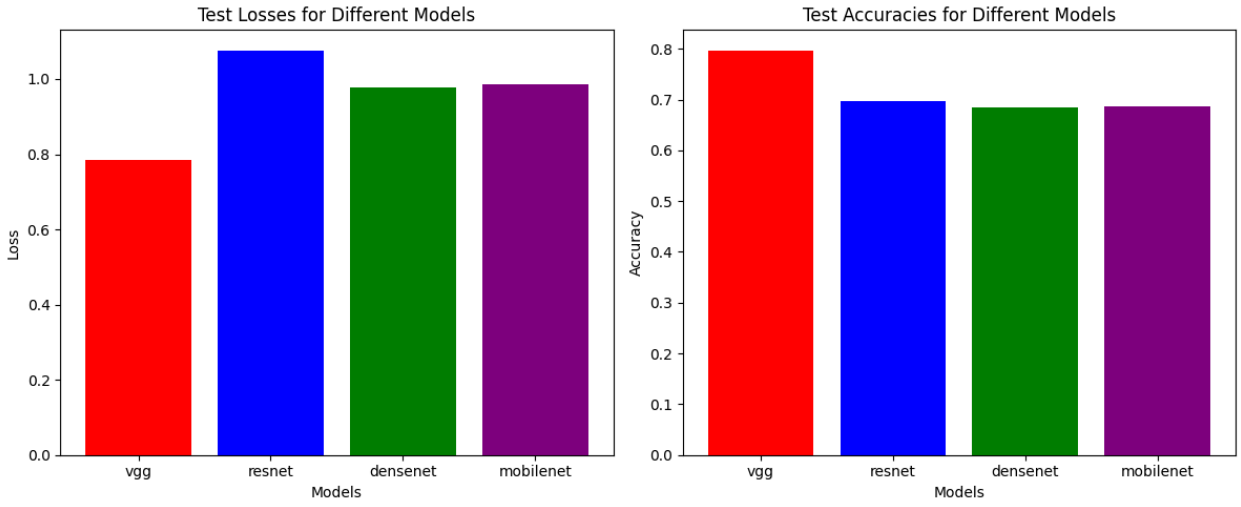
	precision	recall	f1-score	support
0	0.73	0.79	0.76	201
1	0.97	0.41	0.58	216
2	0.42	0.92	0.58	173
3	0.54	0.35	0.43	184
4	0.78	0.83	0.81	203
5	0.75	0.72	0.74	200
6	0.87	0.67	0.76	202
7	0.79	0.58	0.67	215
8	0.67	0.78	0.72	175
9	0.72	0.85	0.78	194
accuracy			0.69	1963
macro avg	0.72	0.69	0.68	1963
weighted avg	0.73	0.69	0.68	1963

DenseNet-121



	precision	recall	f1-score	support
0	0.86	0.78	0.81	201
1	0.98	0.42	0.59	216
2	0.49	0.89	0.63	173
3	0.59	0.32	0.41	184
4	0.80	0.78	0.79	203
5	0.86	0.68	0.76	200
6	0.75	0.72	0.74	202
7	0.56	0.74	0.64	215
8	0.59	0.76	0.67	175
9	0.72	0.78	0.75	194
accuracy			0.69	1963
macro avg	0.72	0.69	0.68	1963
weighted avg	0.73	0.69	0.68	1963

Final Results on Test Set



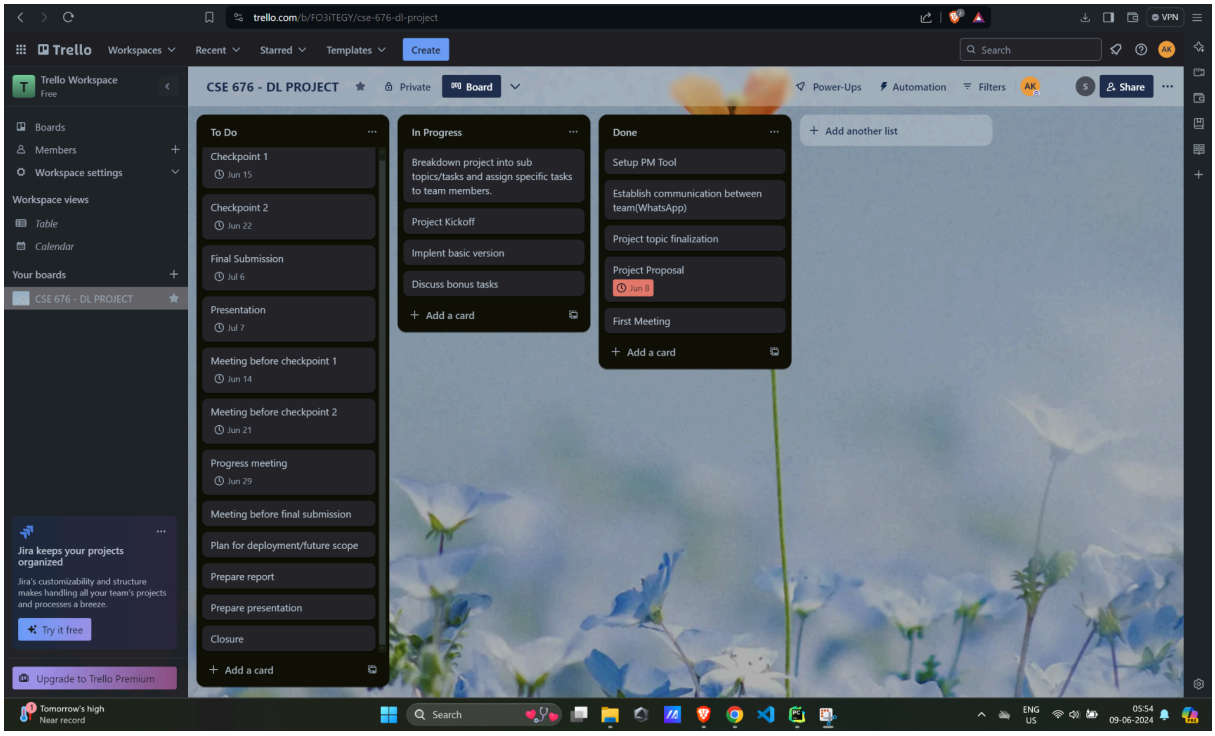
The above graph shows performance of the four architectures we have implemented on the test set in terms of loss and accuracy. VGG has the lowest loss and highest accuracy indicating strong generalization capability.

Ranking on basis of accuracy:

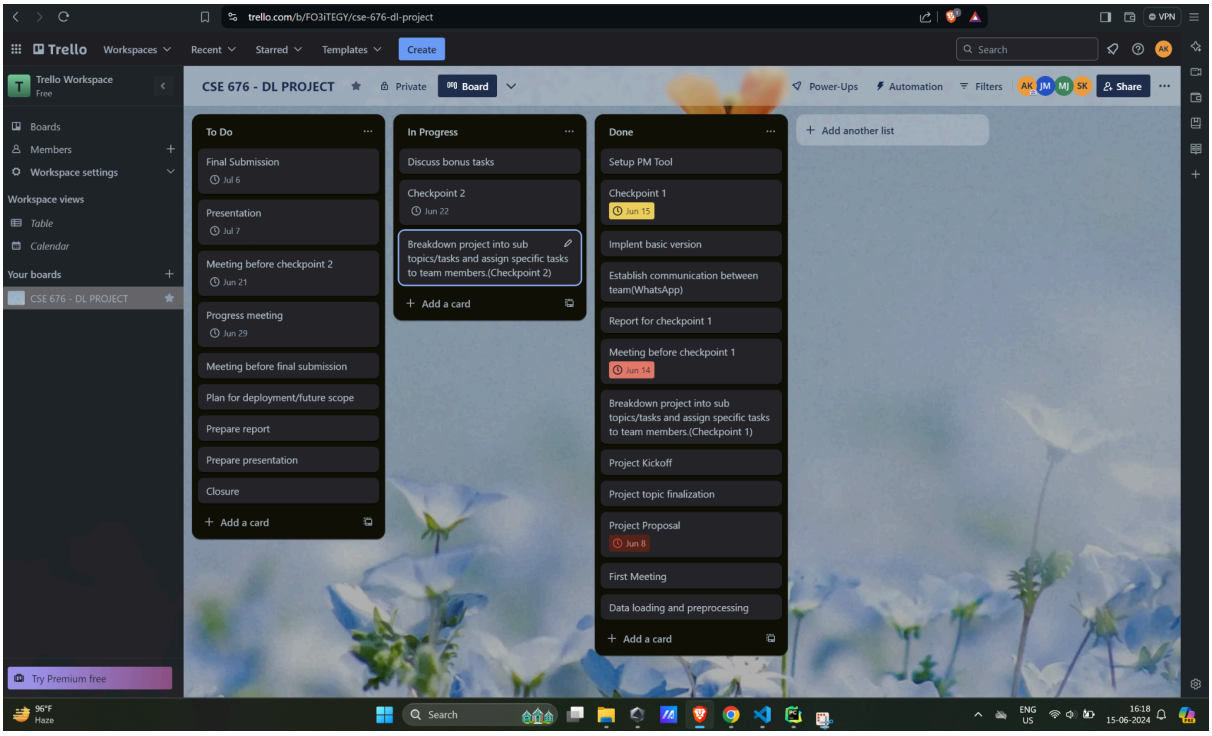
Architectures/Models	Accuracy (%)	Loss
VGG	79.72	0.7834
ResNet	69.69	1.0765
MobileNet	68.67	0.9865
DenseNet	68.52	0.9772

PM Tool Screenshots

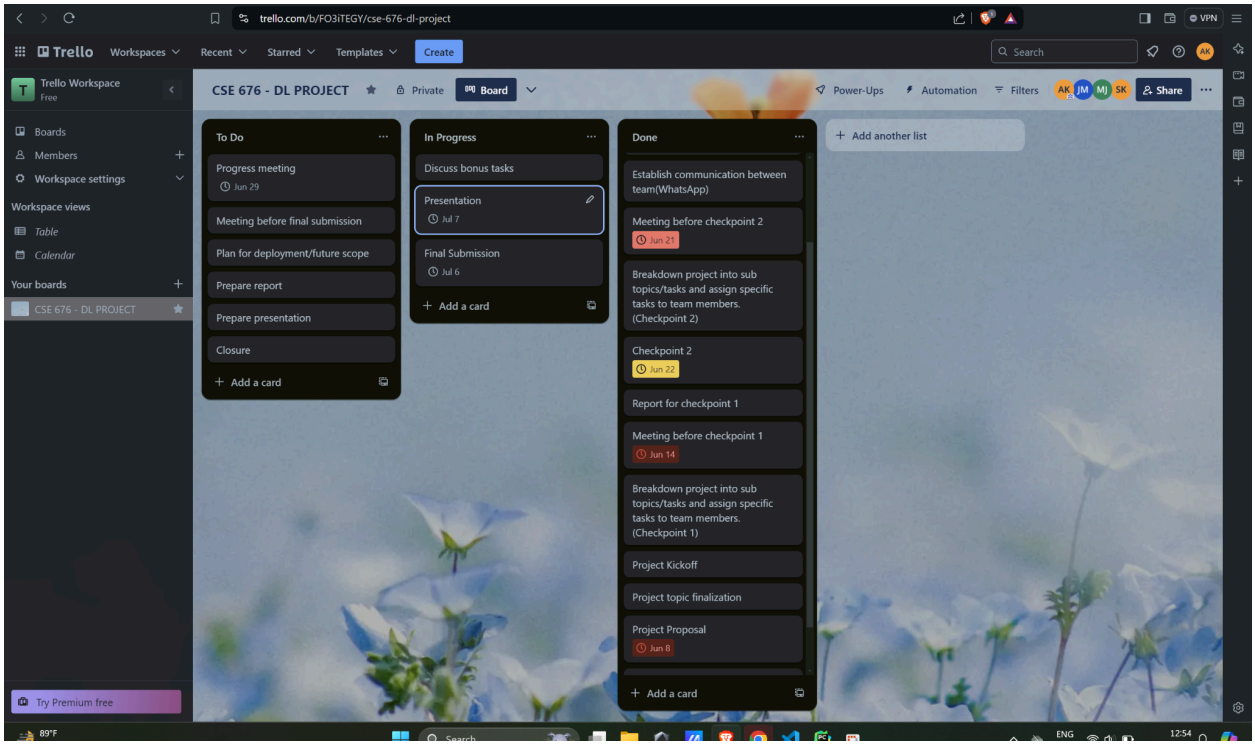
9th June, 2024



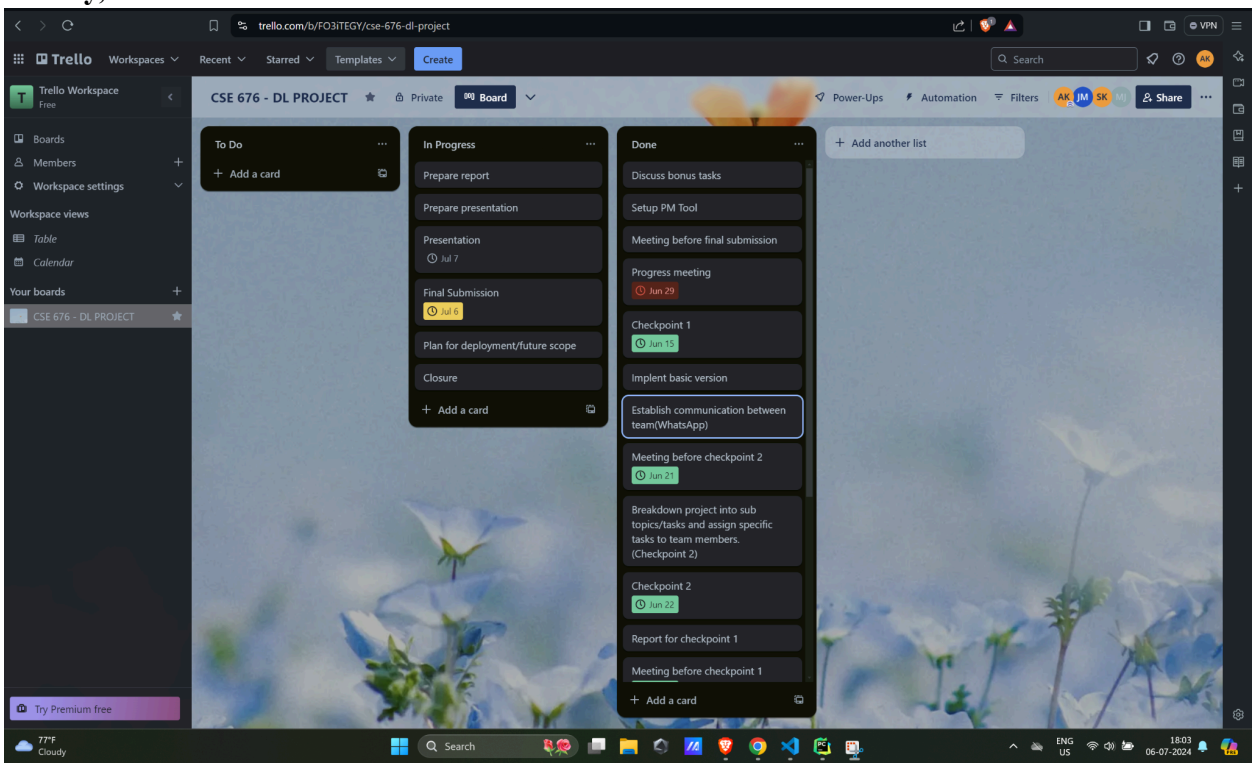
15th June, 2024



22nd June, 2024



6th July, 2024



Team Contribution Summary		
Suman Krupa	**Johann Matzal**	**Arjun Kapoor**
33.3%	**33.3%**	**33.3%**

References:

- https://pytorch.org/tutorials/beginner/basics/data_tutorial.html
- https://pytorch.org/vision/0.9/transforms.html
- https://github.com/murphypei/create-pascal-voc-dataset
- https://medium.com/@godeep48/an-overview-on-mobilenet-an-efficient-mobile-vision-cnn-f301141db94d
- https://arxiv.org/pdf/1409.1556v6
- https://www.geeksforgeeks.org/vgg-16-cnn-model/
- https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918
- https://pytorch.org/vision/main/models/generated/torchvision.models.vgg16.html
- https://medium.com/@alejandro.itoaramendia/densenet-a-complete-guide-84fedef21dcc#:~:text=Summary-,Introduction,by%20reusing%20preceding%20feature%20maps.

- https://www.youtube.com/watch?v=y81RrUHMRSA
- https://pytorch.org/hub/pytorch_vision_densenet/
- https://pytorch.org/hub/pytorch_vision_resnet/
- https://pytorch.org/hub/pytorch_vision_vgg/
- https://pytorch.org/hub/pytorch_vision_mobilenet_v2/