Distracted Driver Detection

Pavan Gurudath

May 1, 2018

Abstract

In this project, a classification of drivers driving safely or in an unsafe manner is done based on the dataset so obtained from Kaggle. It looks at several images of distracted drivers that were taken from a dashboard camera looking at the driver performing different actions. These actions consists of activities that could be deemed as distracting whilst behind the wheel of a car. A Convolutional Neural Network is used in order to more accurately predict what activity a driver is being distracted by.

Contents

1 Introduction

Convolutional Neural Networks (ConvNets or CNNs) is a category of Neural Networks that are used in areas such as image recognition and classification. It is a modular sort of classifier. One of the eye-opening developments in Machine Learning/Computer vision is that a cascade or chain of very simple image processing operations can outperform more complicated state-of-the-art object recognition algorithms. Of course, the specific image processing operations used have to be well-chosen. The real magic in a CNN is learning the underlying convolution filters. A certain number of layers are chained together in order to form what is known as deep layer. These long chains are formed by building small and simple computational building blocks. Much of the heavy computational and big-data requirements for training CNNs (so-called deep learning) are concentrated on this learning process. Among various networks, Convolutional neural networks has demonstrated high performance on image classification. CNNs are successfully used in various situations such as identifying faces, objects and traffic signs for autonomous driving, powering vision in robots.

Distracted driver, something that most of us are either while driving after a long day's work or while driving for a long duration of time. Distracted driving is any activity that diverts attention from driving, including talking or texting on your phone, eating and drinking, talking to people in your vehicle, fiddling with the stereo, entertainment or navigation systemanything that takes your attention away from the task of "safe driving". We've always been in a situation when the light turns green and the car in front doesn't budge and there's unnecessary disrupt in traffic. This might seem innocuous but is much more dangerous than people realize.

According to the World Health Organization(WHO)[1], it was estimated that over 1.25 million people were killed on the roads worldwide, making it the leading cause of death globally. One study found that 69% of drivers in the United States of America (USA) had used their mobile phone while driving within the previous 30 days a percentage higher than in Europe, where it ranged from 21% in the United Kingdom to 59% in Portugal. [1] According to the National Highway Traffic Safety Administration (NHTSA), each day in the United States, approximately 9 people are killed and more than 1,000 injured in crashes that are reported to involve a distracted driver. And in 2015, there were 3,477 people killed, an estimated additional 391,000 injured in motor vehicle crashes involving distracted drivers and approximately 660,000 drivers who use cell phones while driving. [2]

This growing number in injuries is one of the main motivation for this project. A system could be developed into cars which warns the drivers whenever they are in one of the distracted modes. This statistics could also be used by insurance companies to better insure their customers, by testing whether dashboard cameras can automatically detect drivers engaging in distracted behaviours.

The problem statement that was stated for this project is **to implement a convolutional neural network to classify if the driver is alert or distracted with miscellaneous activity such as using the phone, drinking, turning back, etc.** The list of classification is as shown in Table 1. Using the dataset provided by Kaggle [3] the goal is to classify the input images into the categories as shown in table 1.

2 Related Work

The dataset for this project is obtained from a Kaggle competition[3]. There have been 1440 submissions in total on the public leader board of the competition. However, most of them haven't reported the methodology or the parameters that has been used. Amongst the ones that have been submitted for conferences, the methodology that have been used include transfer learning on Alexnet and VGG-16, VGG-GAP amongst several other non deep learning models. Some of the papers have used a methodology named DarNet. Besides these models, some of the other solutions used are PCA, SVM using one vs one scheme in [4]. As spoken about in 3, this dataset contains atleast very little training dataset and has four times its test dataset. Therefore to tackle overfitting, a few implementations used aggressive data augmentation in order to tackle this issue.

c0	safe driving
c1	texting - right
c2	talking on the phone - right
c3	texting - left
c4	talking on the phone - left
c5	operating the radio
c6	drinking
c7	reaching behind
c8	hair and makeup
c9	talking to passenger

Table 1: Classfication labels

3 Dataset

In this project, a dataset of driver images is used. The dataset was obtained from a kaggle competition [3] in 2016. This dataset consists of images that are captured from a camera that is placed on the dashboard of the car *i.e.* next to the driver. This camera is used to capture the driver's action. The dataset was obtained by making 26 distinct drivers to drive for about 5 minutes performing all the different actions. These actions include driving normally to driving with distractions. The distracted activities include driving by using the cell phone to make a call using either of their hands, drinking a cup of coffee, turning around and talking to a co-passsenger or texting and driving. A video of these actions were taken and converted into images by taking different frames as inputs. Each of the images that are taken in the car showcases a driver doing a certain activity in the car. These activities include texting, eating, talking on the phone, applying makeup, reaching behind or turning back. The data has been collected by State Farm, an insurance company, in order to better insure their drivers based on inputs such as these. The dataset consists of the following characteristics:

Size	640x480 (RGB)
Training images	22424 images approximately 2200 images per class
Testing images	79726

This dataset does not contain a constant number of images amongst different classes. The classes are as shown in Table 1. The histogram of the dataset is as shown in Figure 1. The total number of images in each class is shown at the head of each bar.

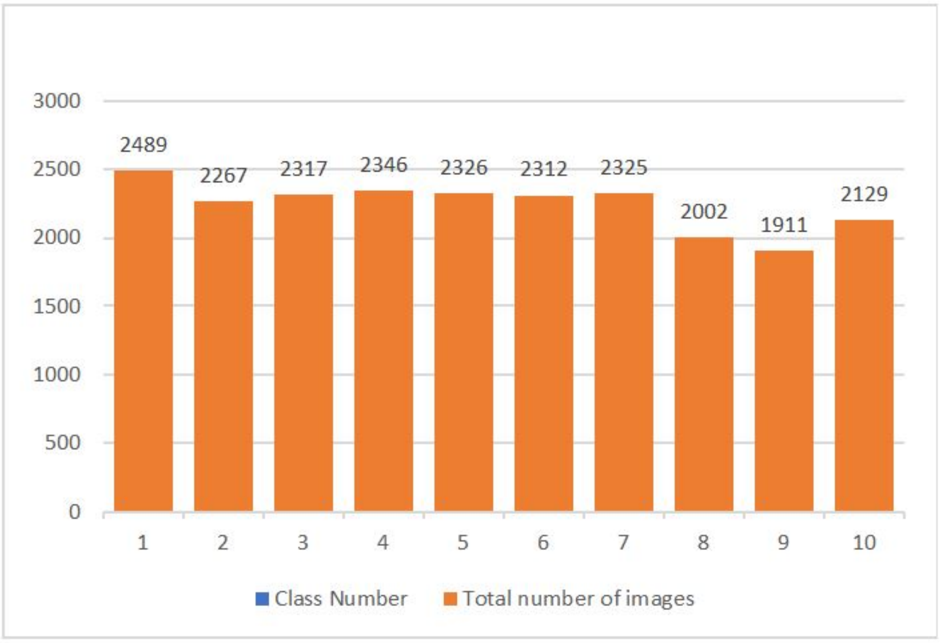


Figure 1: Training data histogram

To show an example of the type of dataset that is present, the following figures give one sample each from each class. They are as shown from Figure 2 through 11. Each of the images are of the size 640×480 dimension.

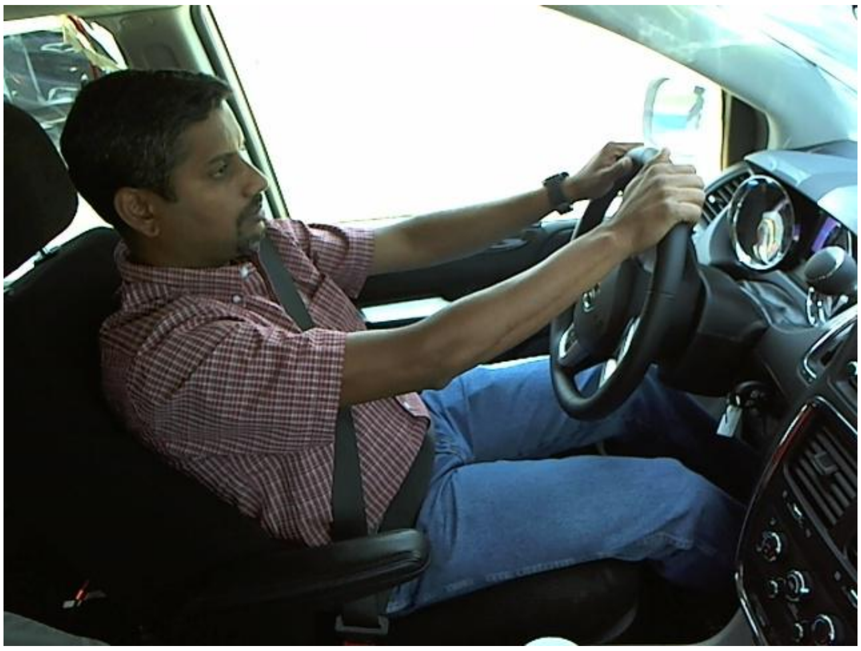


Figure 2: c0: Safe driving

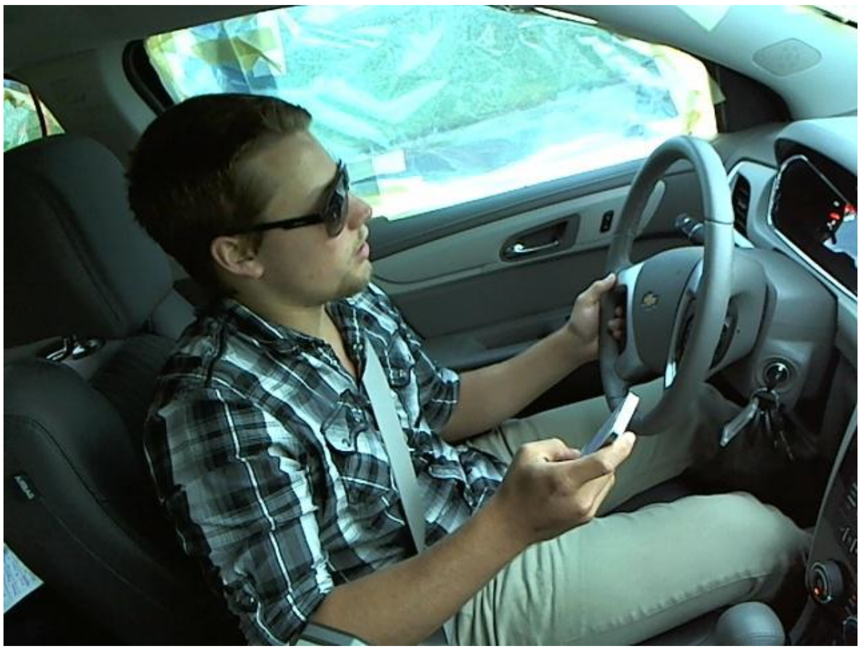


Figure 3: c1: Texting using right hand

The main difference between the training and test dataset is that the training data contains the ground truth labels of each image, while the testing images did not. This has been done deliberately by kaggle since it was a competition and like such, it allowed for some measure of success between different submissions. In this work, however, in order

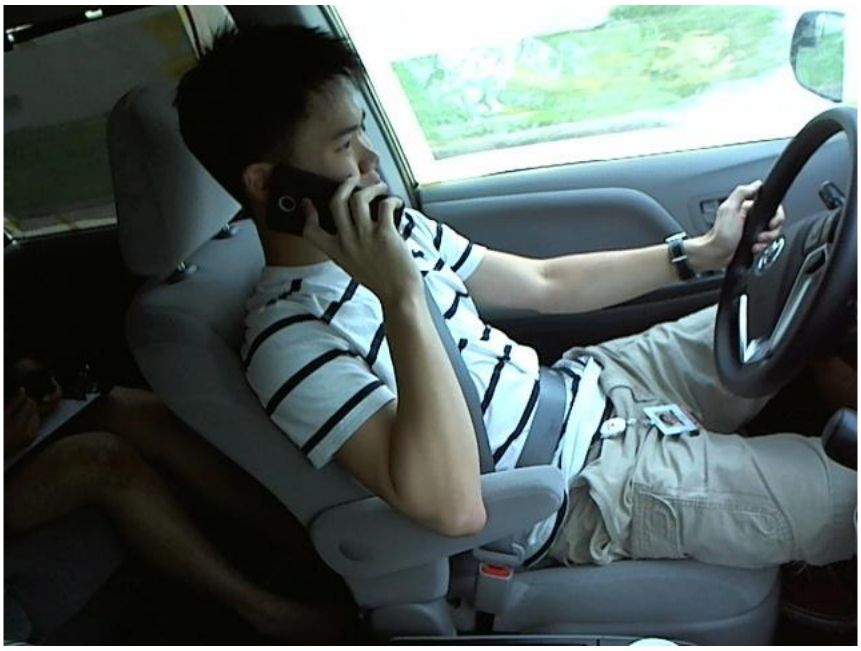


Figure 4: c2: Talking on the phone using right hand

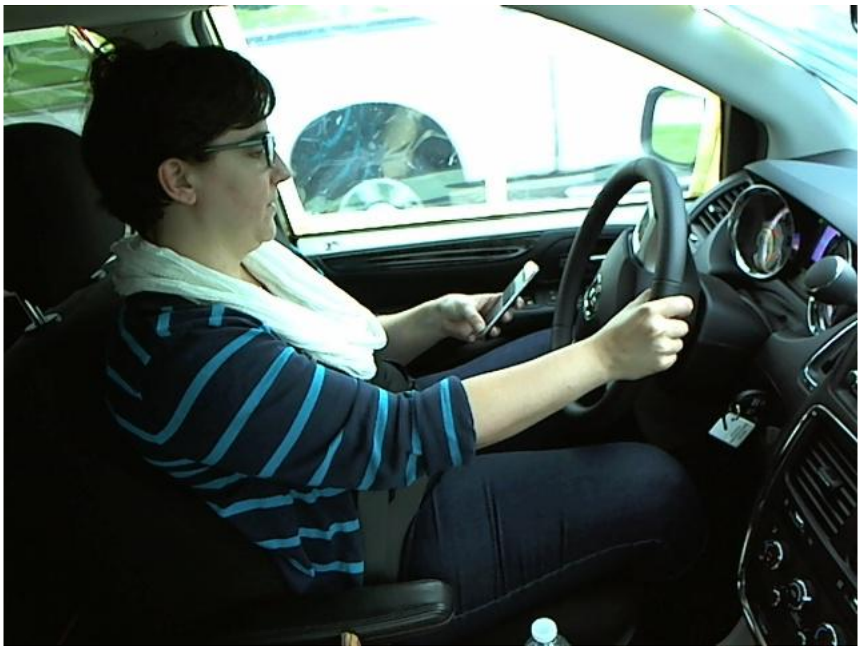


Figure 5: c3: Texting using left hand

to train and evaluate the performance of the models, the dataset were split in different percentages for different experiments. For the methodology carried out in Section 4.2.1, the 22,424 training labeled images were split up further into 60% actual training data, 20% validation, and 20% testing data and for the methodology carried out in Section 4.2.2 and for further work, the 22,424 training labeled images were split up further into 70% actual training data, 10% validation, and 20% testing data. This allows me to evaluate the model offline without having to submit to Kaggle, but also test the model

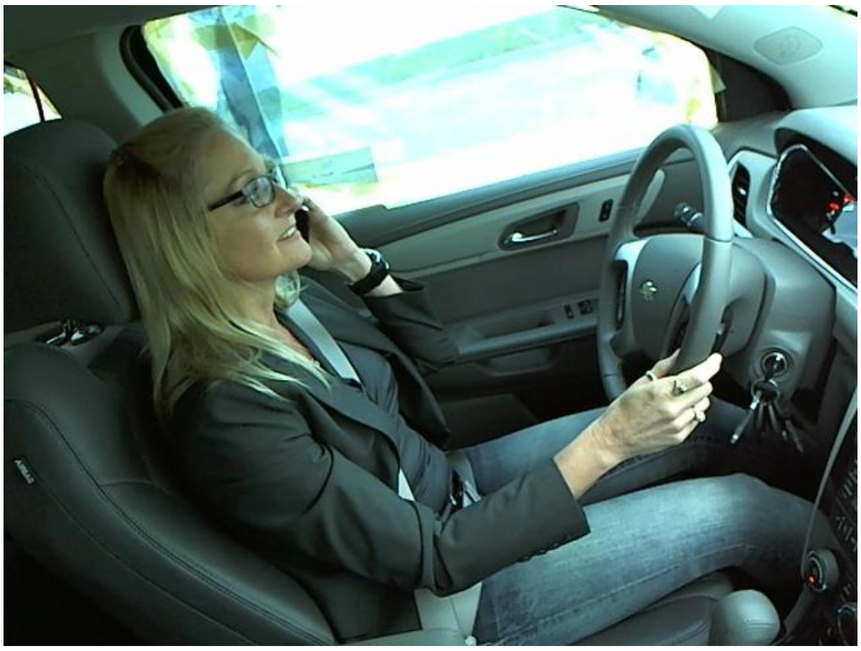


Figure 6: c4: Talking on the phone using left hand

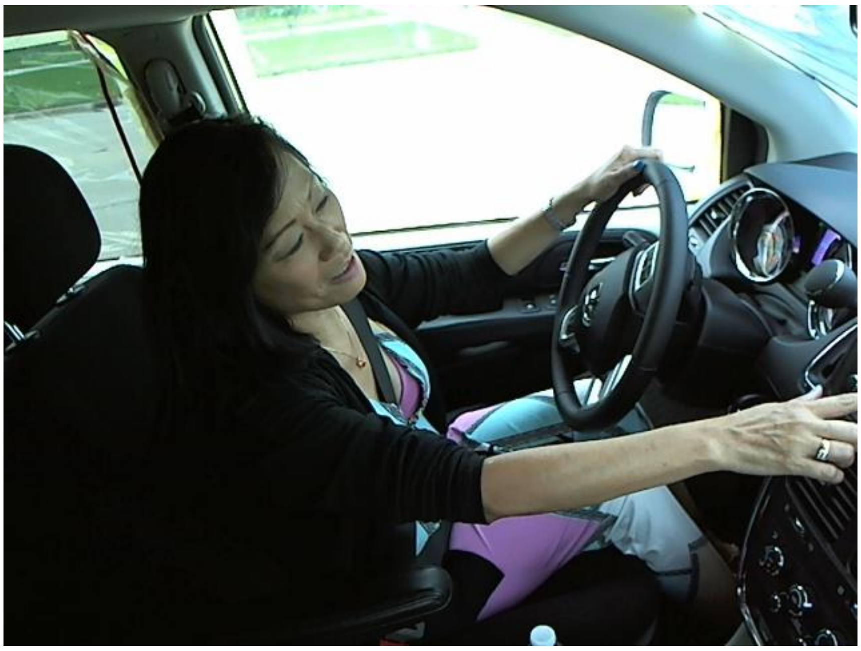


Figure 7: c5: Operating the radio

against unseen data and apply certain testing measures. The initial concern is that it might not be enough data to train the model; however this will be supplemented by data augmentation using the keras library on Python.

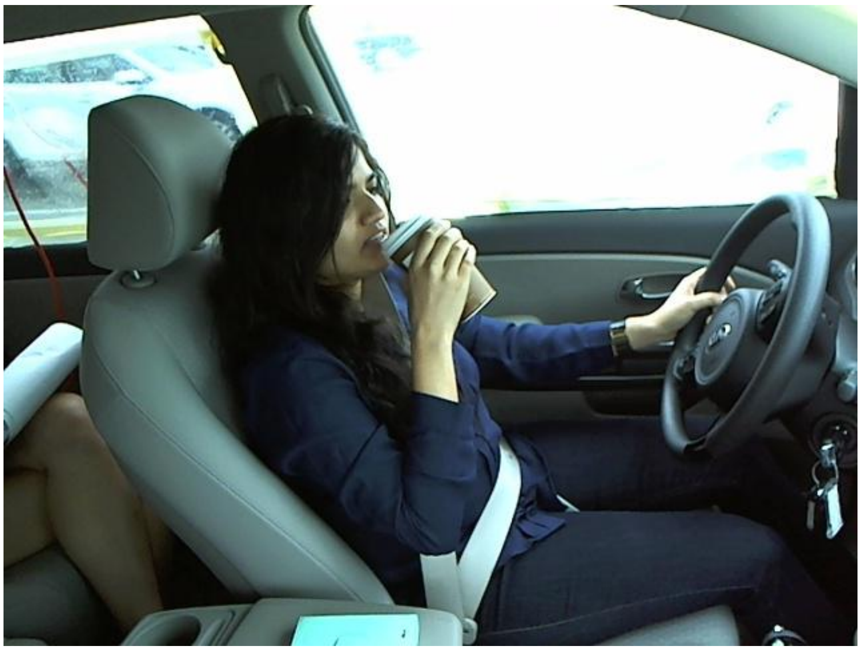


Figure 8: c6: Drinking while driving

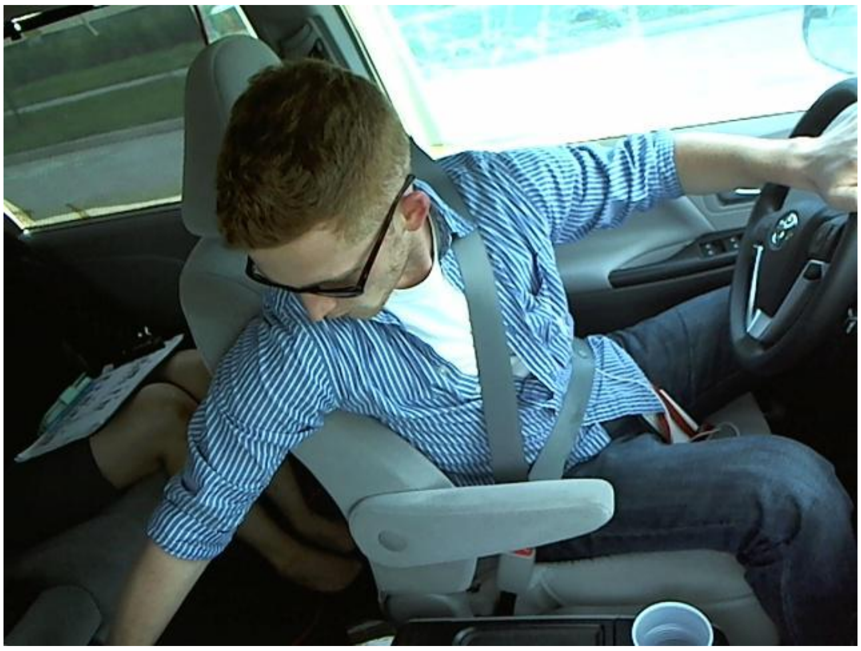


Figure 9: c7: Reaching for something behind

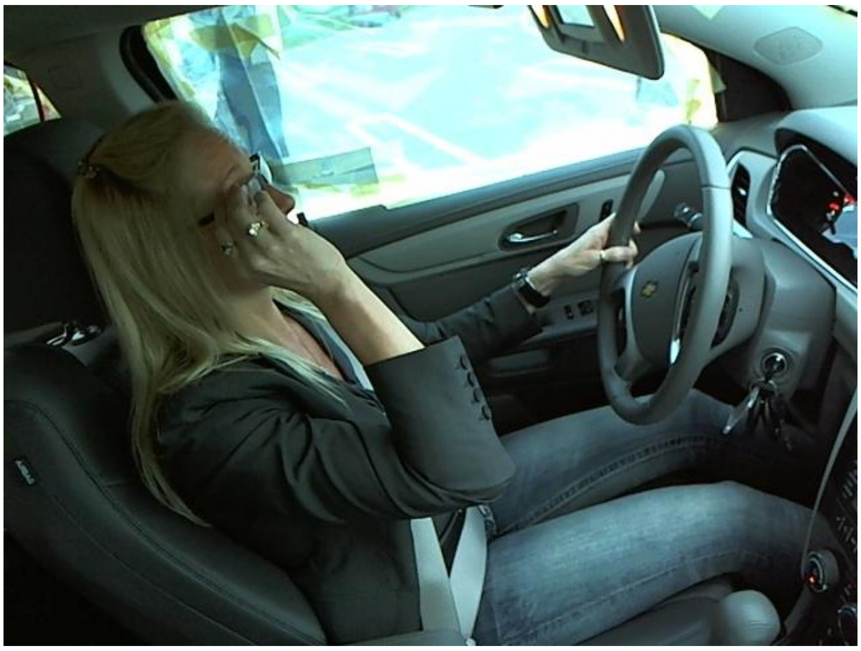


Figure 10: c8: Hair and Makeup

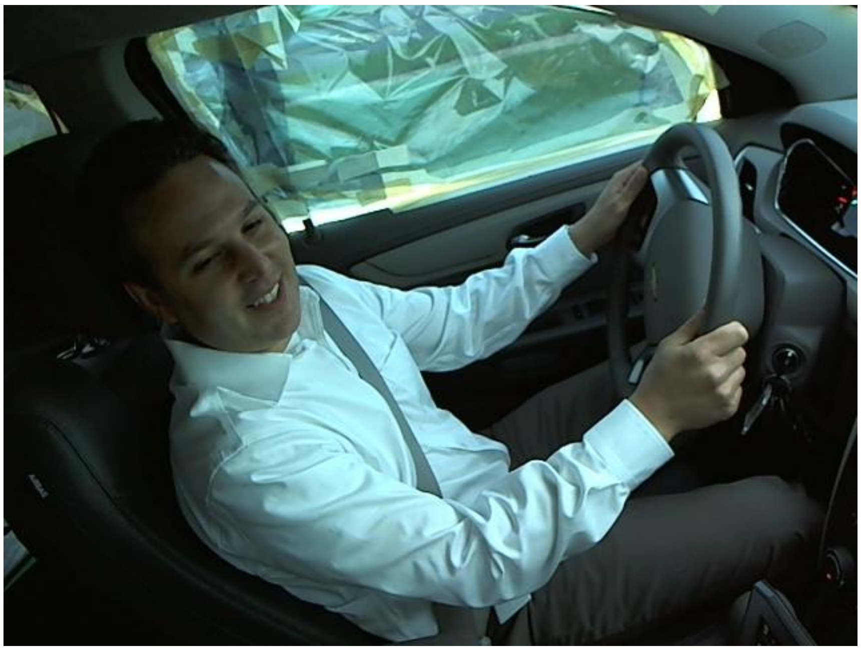


Figure 11: c9: Talking to passenger

4 Methods

4.1 CNN Architecture

A Convolutional Network is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. There are three main types of layers to build ConvNet architectures: **Convolutional Layer**,**Pooling Layer**, and **Fully-Connected Layer**. These layers are stacked to form a full Convolutional Network architecture.

In order to tackle the issue of distracted driver detection, Convolutional Neural Network (CNN) models are utilized. CNNs have proven to perform well on the subject of classification of images, and as such, are a great fit for the current problem statement. We have used CNN to classify different Wallpaper patterns during the third course project and this problem is somewhat of a similar category. It is worth noting that CNNs generally suffer from overfitting, which occurs when a model adapts too well on trained data, but performs poorly on new data, and is thus said to generalize poorly. This is an issue that people who work on deep learning algorithms try to mitigate as much as possible throughout their work. Thereby I wish to solve this problem by comparing myself to the leaderboard on Kaggle using the Kaggle loss score as discussed in section **??**.

Rather than have a fully connected layer for every pixel, CNNs only have enough weights to look at small parts of the image at a time. The process typically involves a convolution layer, followed by a pooling and an activation function, but not necessarily in that exact order. These three separate operations can be applied as separate layers to the original image, typically multiple times. Finally, a fully connected layer (or several) are added at the end in order to classify an image accordingly. With the highly variable number of combinations and permutations, it can be difficult to find the exact one that gives the optimal performance. CNN designs are driven by the community and research who thankfully have made some successful results, and made their CNNs publicly available for others to use and improve upon. Every convolutional network would most likely consist of the following:

- **Input:** This layer holds the raw pixel values of the image. The image could be of any size and is of the form $N \times M \times D$, where N and M are the size of the image and D is the number of channels. Channels is just a word to describe the number of values associated with each 2D spatial pixel location in an image, so, for example, a grayscale image has 1 channel whereas a color image has 3 channels (red, green, blue).

- **Conv:** This layer is the first layer after feeding in the input for any CNN. It computes the output of neurons, each computing a dot product between the weights and a small region that are connected to in the input volume. These small regions are known as *filters*. After sliding the filter over all the locations, we obtain an output matrix of numbers, which is referred to as an activation map or feature map.

- **ReLU:** ReLU stands for Rectified Linear Unit, but despite the fancy name it is just a thresholding operation where any negative numbers in the input become 0 in the output. If the input is an array of size $N \times M \times D$, then the output is an array of the same size with each value in the output array is defined as

$$output(i, j, k) = max(input(i, j, k), 0)$$

- **Pool:** This layer is responsible for the downsizing operation along the spatial dimensions $i.e.$width and height. Reducing the dimensionality of the images is a necessity in convnet since were dealing with high dimensional data and a lot of filters, which implies that we will have huge amount of parameters in our convnet. In this layer, we define the stride $i.e.$the number of pixels by which the filter should move across in order to obtain the summary of that filter. This summarization of the filter could be the average, maximum or minimum value of that patch. In our network, we use the maxPool which simply takes the maximum of the patch from the matrix.

- **Fully-Connected:** If the input is an array of size $N \times M \times D1$ and output is an array of size $1 \times 1 \times D2$ then the fully connected layer applies a filter bank of D2 linear filters and D2 scalar bias values to compute output values. However, unlike convolution layers, each filter is of the same size as that of an input image, and is applied in a way that computes a single, scalar valued output. Because there are D2 filters and bias values, the output image will contain D2 scalar values. These D2 values corrosponds to the class scores.

- **Softmax:** The softmax takes a vector of arbitrary real numbers (here class scores) and converts them into numbers that can be viewed as probabilities $i.e.$ they will all lie between 0 and 1 and sum up to 1. If the input is an array of size $1 \times 1 \times D$, then the output is of the same size and is calculated as follows:

$$output(1, 1, k) = \frac{exp(\ln(1,1,k)-\alpha)}{\sum_{k=1}^{D} exp(\ln(1,1,k)-\alpha)}$$

The figure 12 depicts the various layers for an example case.

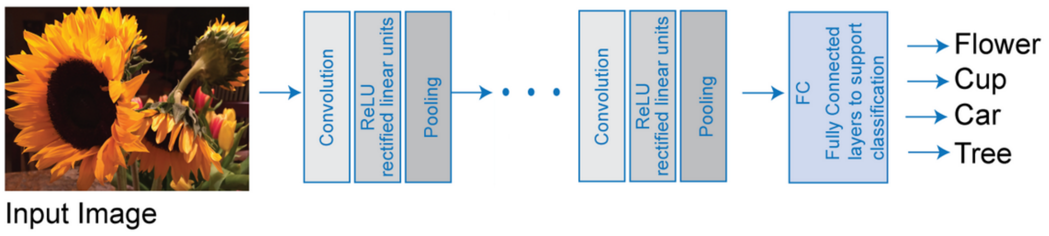

Input Image
Convolution
ReLU rectified linear units
Pooling
Convolution
ReLU rectified linear units
Pooling
FC Fully Connected layers to support classification
Flower
Cup
Car
Tree

Figure 12: Convolutional Neural Network Layer representation

4.1.1 Transfer Learning

Transfer learning exemplifies the idea that knowledge gained whilst solving one problem, can be applied in solving a different, but related problem. Typically, transfer learning is applied onto CNNs. This is usually most appropriate when there is insuffi- cient data or computational power, in which case a model with pre-trained weights on a much larger database to solve one problem can be applied. One strategy typically employed is to use the pre-trained convolutional network as a fixed feature extractor. This is done by completely removing the last fully-connected layers, and then using the rest of the CNN as a feature extractor for a different problem. A linear classifier can then be trained using those features, as well as the new dataset to solve a different problem. Due to the nature of convolutional networks, the deeper the layers, the more specific they become based on the data they were trained on. This can be problematic when using Transfer Learning to solve another problem, since some layers will be oriented towards the original dataset.

For example, the pre-trained ImageNet VGG16 CNN, might have some generic first layers that detect things like lines and edges; however, in the deeper layers, other features are learned, which are only specific to classes in the original data such as cats, dogs,etc. If the more generic layers of the pre-trained CNN could be frozen, then the more specific layers could be re-trained on the new dataset and eventually build a model specific to the new data. This process is known as Fine-Tuning[5] and is a commonly used strategy in Transfer Learning. The aforementioned Transfer Learning strategies are also something that will be utilized throughout in my work. The primary reason for this is to overcome the hardware limitations and to more importantly achieve a higher accuracy.

4.2 Models

4.2.1 Small CNN

In the first attempt at solving the distracted driver problem, several simple Convolutional Networks were implemented. Since coding in Python and using keras was new to me, I tried several different small networks. Due to limited hardware at my disposal, I reduced the images from 640 x 480 RGB to 24 x 24 grayscale images. This was a feasible solution since I was trying out different networks with different parameters and it allowed me to run in a reasonable amount of time. However, the reduction in quality greatly reduced the amount of information available to the model, and had a significant impact on the results.
The Convolutional Neural Network architecture implemented consists of 2 convolutional layers and 2 fully connected ones. The architecture of the model can be seen in Figure 13. The conv layers represent a convolution followed by a Rectified Linear Unit (ReLU) activation. The pool layers represent pooling layers, and norm layers depicts local response normalization. Finally, the fc layers represent a fully connected layer with a ReLU activation. After training the model on a CPU overnight for several hours and 25 epochs, it achieved an all-time-high accuracy of 10.418%. The results were pretty bad and not up to par with my expectations since the model performed only slightly better than a random model, which would have a 10% accuracy.
In order to improve the accuracy, a Keras open source neural network library, with TensorFlow as the backend was used to build a simple CNN by using Keras documentation [6]. The architecture of the simple convolutional network is as shown in Figure 14.
Similar to the initial network, the conv layers represent a convolution followed by a ReLU activation layer, pool layers represent pooling layers, and fc layers represent fully connected layers followed by a ReLU activation. The new addition to the model is the dropout layer which has replaced the previous norm layers. Dropout is mainly introduced to deal with regularization by introducing noise into the model and consequently reducing the amount of overfitting. The noise introduced by dropout is done by, with some probability, turning off some perceptrons to prevent the model from memorizing patterns that are specific only to the training data. Lastly another convolutional and pooling layer were added to add some complexity to the model. A significant change worth noting is that the image size was resized to 150 x 150 as opposed to the previous 24 x 24. This was significant because a lot more information was retained in the image. Additionally a random rotation of 10° and zoom of 10% was applied at random on the training images to help alleviate the problem of overfitting.

This model depicted in Figure 14 was run with 13,447 training, 4,490 testing and 4,487 validation samples on the CPU. The model was set to run with 50 epochs and a dropout rate of 50%. The main challenge in this step was to not overfit the model, which is why I opted to use the validation set. At every epoch, the model was trained and tested against the validation data that it had not encountered during training. If the accuracy against the validation set increased, that would indicate the model is improving; however, if the validation accuracy began to drop, this meant that the model was starting to overfit and it should be stopped.

As soon as the validation accuracy stopped improving, or got worse, the model would terminate training after 3 epochs of constant outcome. This was set using the patience

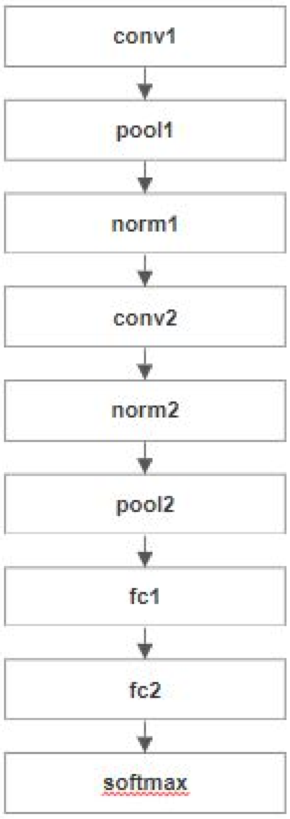


Figure 13: Small CNN architecture - 1

command. Finally, the best recorded weights up to that point were saved. After training for about 8 hours, the model finished after 11 epochs and recorded the best weights with a 96.808% accuracy. This was significantly higher than the initial few attempts, which could be largely due to increased image size, data augmentation such as random rotations and zooming, dropout layer, and the model complexity. This was a significant result and thus I moved onto the next model knowing that using transfer learning on a good network would improve its accuracy further. Figure 15 shows the model summary of the network that gives us a high accuracy on this dataset.

The next method was a transfer learning implemented on a VGG-16 network and is as explained in Section 4.2.2.

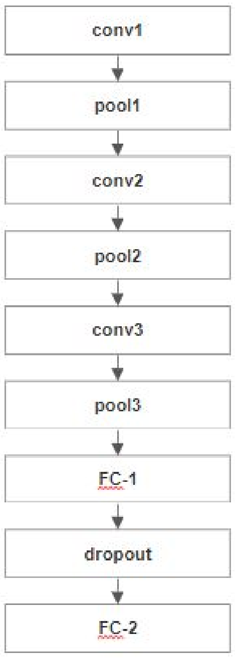
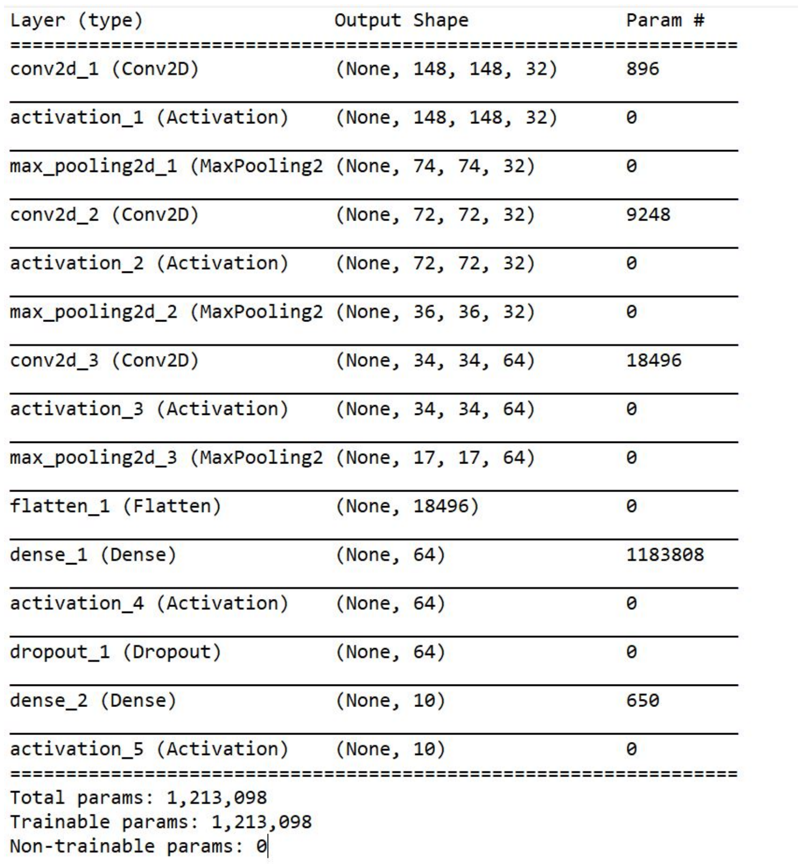


Figure 14: Small CNN architecture - 2



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 148, 148, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 72, 72, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 34, 34, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                1183808
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 1,213,098
Trainable params: 1,213,098
Non-trainable params: 0
```

Figure 15: Model summary of the second small architecture

4.2.2 VGG-16

While the proposal listed VGG-19, I first tried out VGG-16 [7] network just to see its performance. However, after performing fine tuning and training on the dataset, a good performance on this model was achieved which is discussed below. Therefore, instead of further implementing with a VGG-19 model, I would use this model in place of it and continue the remainder of my project.

The VGG16 network was pre-trained on the ImageNet dataset and is easily accessible through the Keras library. The idea is that the pre-trained network would have learned features from previous data that might prove useful in predicting images of distracted drivers. This wont work out-of-the-box because the network was trained to predict things like cats and dogs, and doesnt directly transfer to the distracted driver images. The strategy is to disconnect the fully connected layers from VGG16, since they contain weights very specific to the original ImageNet dataset. In order to augment the model to fit the distracted driver dataset, a new sequential model was created with a similar fully-connected classifier as the Simple CNN in section 4.2.1. The model was built with the help of Keras documentation [6] and its architecture is as shown in Figure 16.
The training and validation images were propagated through the VGG16 network a single time, the output bottleneck features were saved into files and were then fed into the newly created full-connected classifier top model. It is important to note that only the top model was trained on the new dataset, and as such, it learned features only prevalent in the distracted driver dataset. The key takeaway is the retrained fully-connected classifier that was stitched to the top of the VGG16 model to be trained on the distracted driver images. The images used were of the size 224 x 224 since that is the input size that the network was built upon, and this enabled the model to retain even more of the information than our previous implementations.

The training of the top model was done similarly to the training method applied in the Simple CNN model. Two models were tried, whose parameters are shown in Table 2. The validation set was used with a patience of 5 and 2 and ran with 50 epochs. The best model amongst the two stopped after 11 epochs, while the other stopped after 13 where there were no more positive changes to the accuracy of the validation set, so the training process terminated and the best weights were saved.

Description	**VGG16-model 1**	**VGG16-model 2**
Batch size	64	32
Number of epochs	50(patience=5)	50(patience=2)
Number of epochs run	11	13
Accuracy	99.354%	99.1%
Loss (crossentropy)	0.02322	0.04889

Table 2: VGG16 parameters of the two models

Running the model on the test set achieved an accuracy of 99.354% and 99.1% respectively for the two models. The model took relatively much lesser time to train and achieved an even higher score than the previous implementation. These results are as shown in section 5

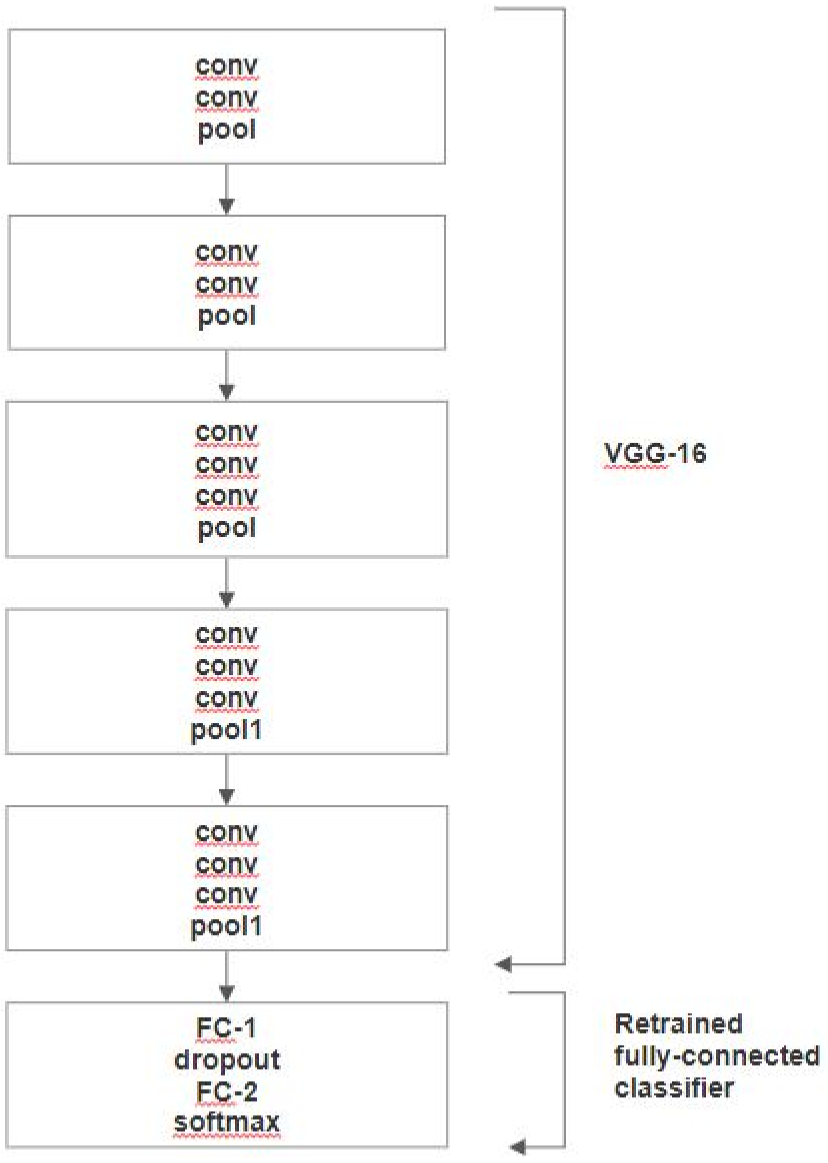
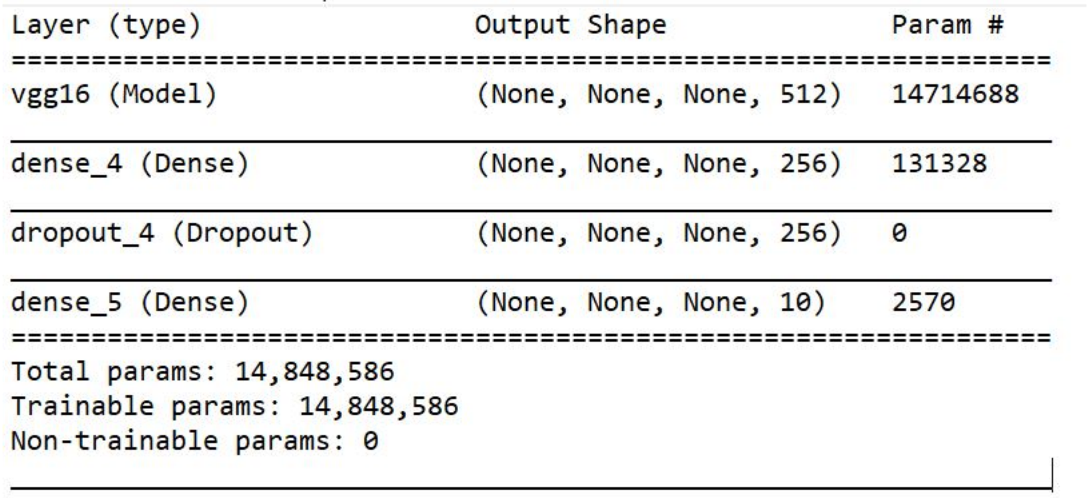


Figure 16: VGG-16

Figure 17 shows the summary of the model that was used in order to train the network.



```
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, None, None, 512)   14714688
_________________________________________________________________
dense_4 (Dense)              (None, None, None, 256)   131328
_________________________________________________________________
dropout_4 (Dropout)          (None, None, None, 256)   0
_________________________________________________________________
dense_5 (Dense)              (None, None, None, 10)    2570
=================================================================
Total params: 14,848,586
Trainable params: 14,848,586
Non-trainable params: 0
_________________________________________________________________
```

Figure 17: Model summary of the VGG-16 architecture

4.2.3 ResNet50

Initially. I thought of trying the GoogLeNet model, known by its next version InceptionV3 which is avialble on keras. But having tried to run a model, due to its large size of the network, the model was taking a lot of time to run even on the GPU. Due to less availbility in time, I made the decision to run ResNet50[8] model.

The ResNet50 network was pre-trained on the ImageNet dataset and is easily accessible through the Keras library. The idea is that the pre-trained network would have learned features from previous data that might prove useful in predicting images of distracted drivers. This wont work out-of-the-box because the network was trained to predict things like cats and dogs, and doesnt directly transfer to the distracted driver images. The strategy is to disconnect the fully connected layers from ResNet50, since they contain weights very specific to the original ImageNet dataset. In order to augment the model to fit the distracted driver dataset, a new sequential model was created with a similar fully-connected classifier. The model was built with the help of Keras documentation [6] and its architecture is as shown in Figure 18.
The training and validation images were propagated through the ResNet a single time, the output bottleneck features were saved into files and were then fed into the newly created full-connected classifier top model. It is important to note that two models were trained. One where only the top model was trained on the new dataset, and as such, it learned features only prevalent in the distracted driver dataset. The other network was trained for 5epochs by freezing the last 5 layers. The images used were of the size 224 x 224 since that is the input size that the network was built upon. The training of the top model was done similarly to the training method applied in the other models.

4.2.4 Ensemble Learning

Ensemble learning is the method of combining two or more models and using their weights in order to determine the output predictions. You combine two or more methods in order to improvise on the stability and predictive power of each of the model. In my project, I've adopted a simple ensemble learning method, where based on the log loss error of the two best models from VGG and ResNet50, a weighted average is done. The weights are given randomly based on how the models have performed based on the public and private leaderboard of the Kaggle competition. The combined result is as shown in Section 5.3

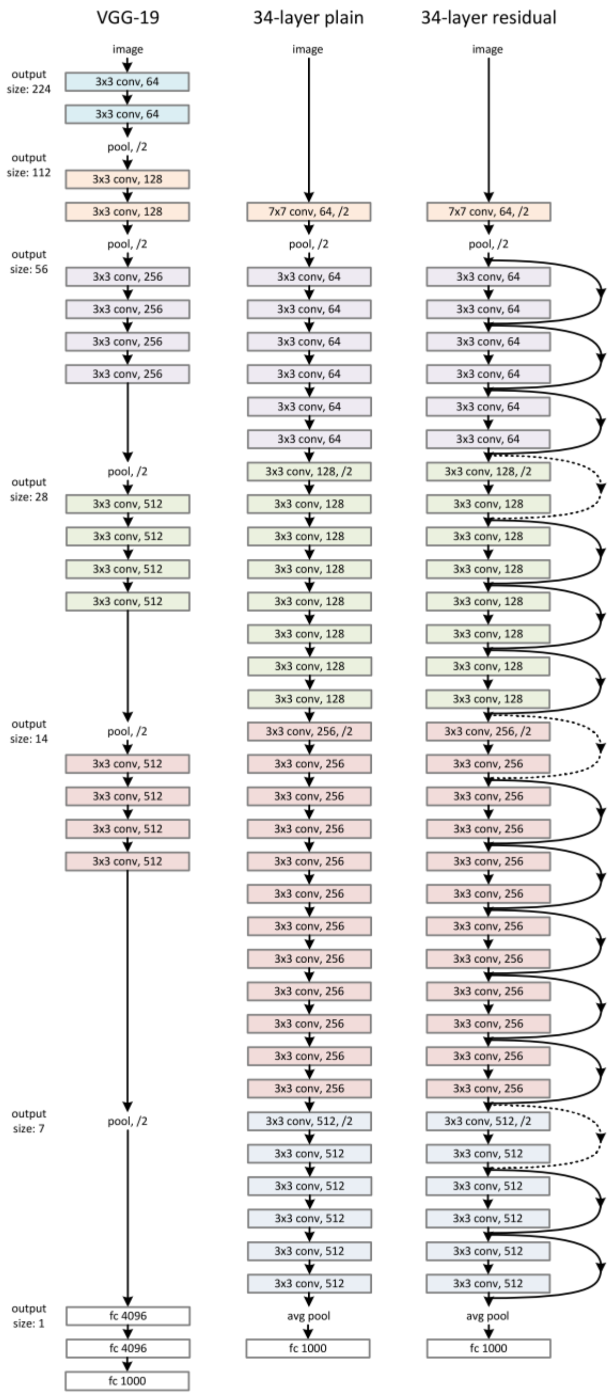


Figure 18: ResNet50 architecture (extreme right)

5 Results

5.1 Summary of the small CNN and VGG-16 networks

To summarize the results, Table 3 shows that the VGG16 model-1 with a retrained fully-connected classifier to be the best performing model out of the four. The VGG Top Retrained model seems to have some time cost in creating the bottleneck features, but since they can be stored offline and loaded at any time, the subsequent training of the fully connected layers takes significantly less time. It seems as though this is the best overall model thus far, since it not only results in the highest accuracy, but also trains in the least amount of time.

CNN	**Accuracy**
small CNN architecture-1 13	10.418%
small CNN architecture-2 14	96.808%
VGG16-model 1 (best)	99.354%
VGG16-model 2	99.1%

Table 3: Accuracy of different networks

In the best small Convolutional neural network model, *i.e.* the small CNN architecture 2 14, the graph of accuracy and the loss of training and validation are as shown here in Figure 19 and 20. It can be seen that the accuracy of the validation is almost close to 100 and is really high!

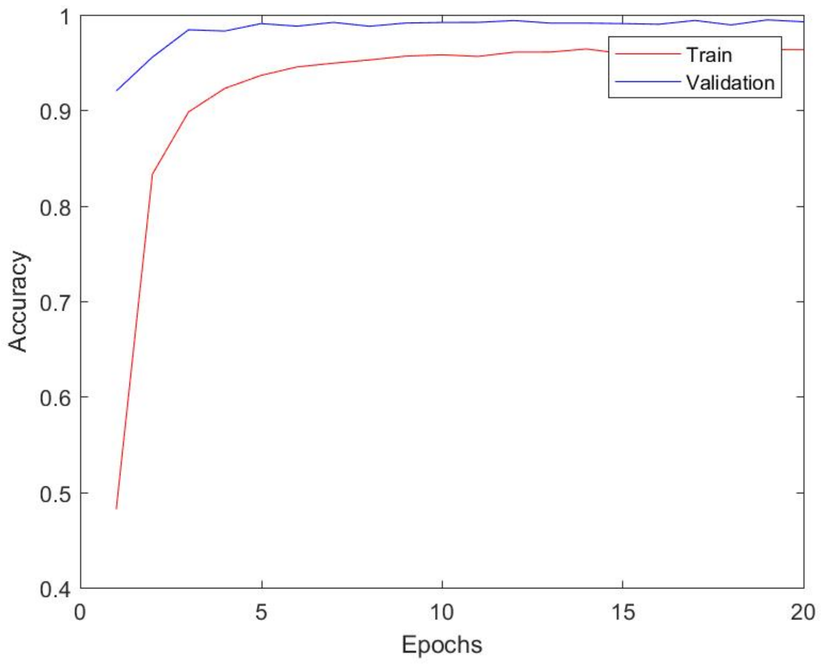


Figure 19: Small CNN2: Accuracy for training and validation

The model is further analyzed by looking at their accuracy and loss graphs which are shown in Figure 21 and Figure 22. The Confusion Matrix are also shown in Figure 23

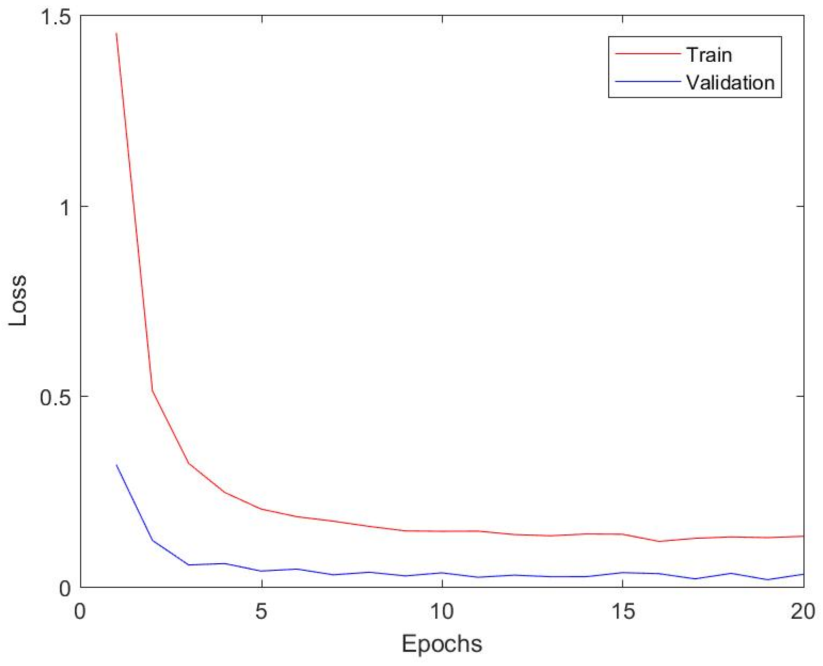


Figure 20: Small CNN2: Loss for training and validation

and 24. It can be observed that most of the predicted labels are close to 100% accuracy. Here as well, it can be seen that the model performs really well. The Receiver Operating Characteristic graph is as shown in Figure 25 and Figure 26 for the vgg-16 model 1 and model 2 respectively.

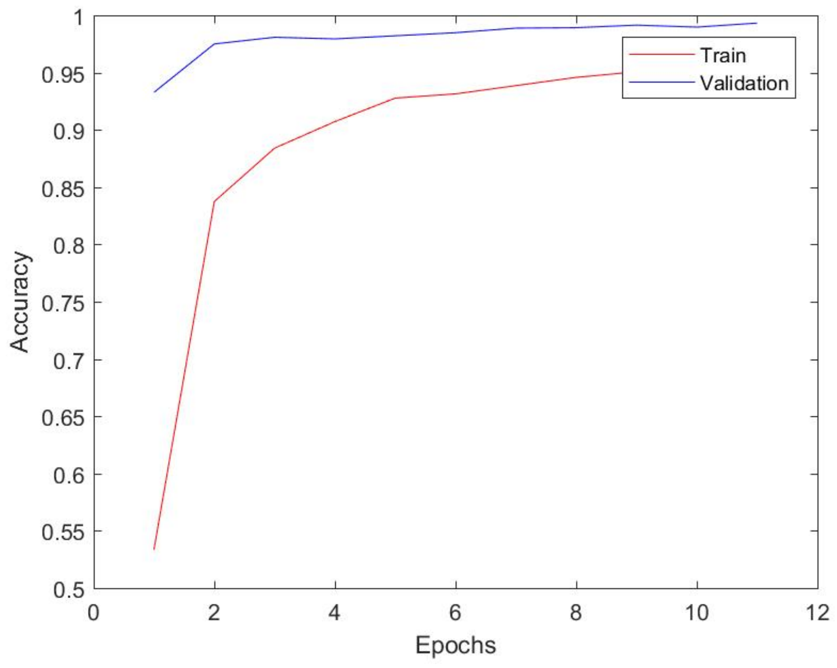


Figure 21: VGG=16 model 1: Accuracy for training and validation

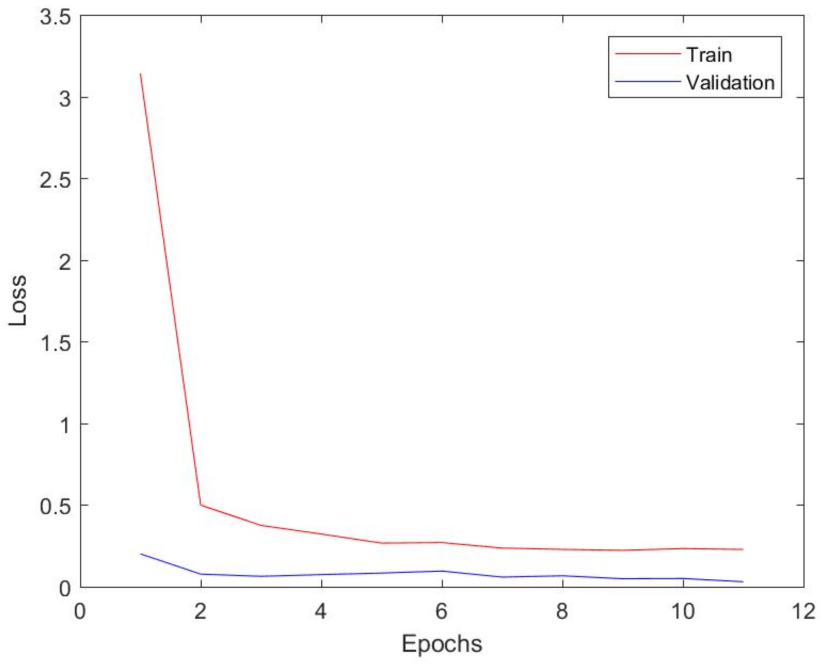
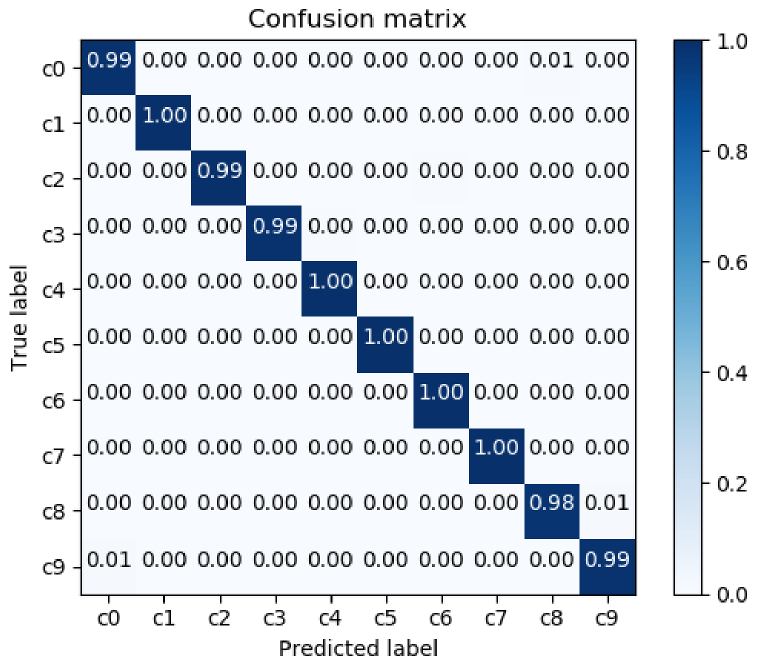


Figure 22: VGG-16 model 1: Loss for training and validation



	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
c1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
c2	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00
c3	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00
c4	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
c5	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
c6	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
c7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
c8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.01
c9	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99

Figure 23: Confusion matrix of VGG16 model-1

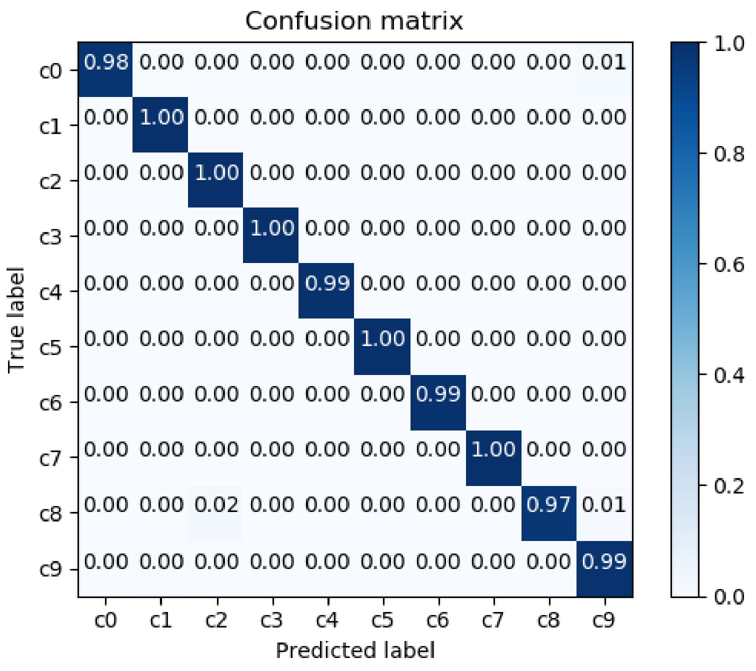


Figure 24: Confusion matrix of VGG16 model-2

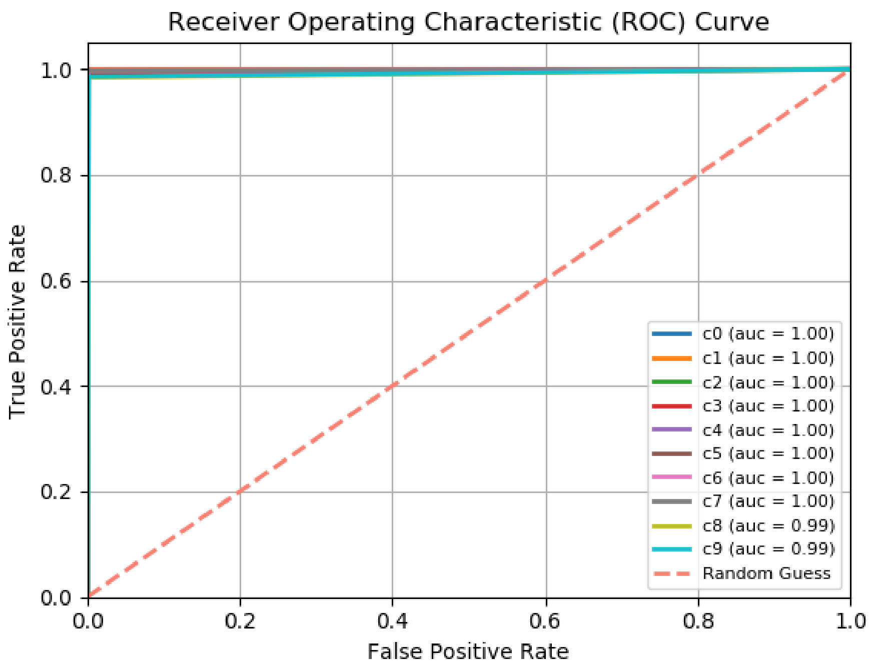


Figure 25: VGG-16 model-1: Reciever Operating Curve

The final results are tested on the 79,276 test dataset which are unlabelled. A script for generating the .csv file for these 80k files is written and the file is submitted onto the

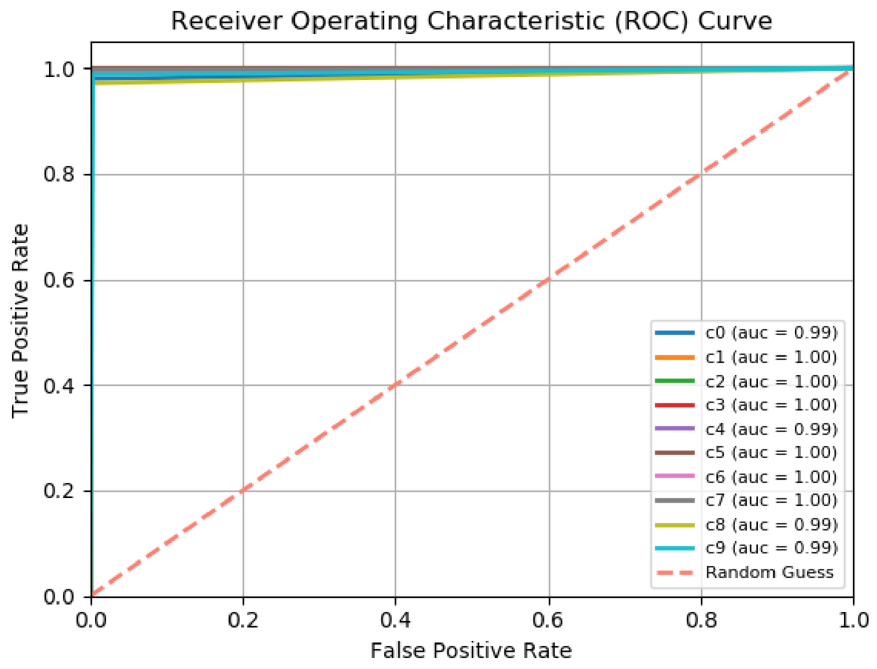
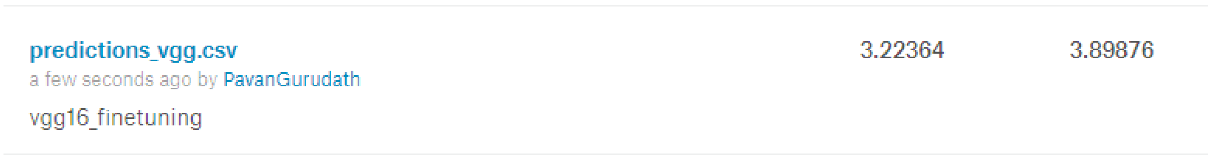


Figure 26: VGG-16 model-2: Reciever Operating Curve

leaderboard. The log loss error is calculated using the formulae given in Equation 1

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij}) \tag{1}$$

Here N is the number of images in the test set, M is the number of image class labels, log is the natural logarithm, y_{ij} is 1 if observation i belongs to class j and 0 otherwise, and p_{ij} is the predicted probability that observation i belongs to class j.
While having to do this portion, I had run into a problem whereby the GPU as well as the CPU was running into an error. After a few days of debugging, the final score on the leaderboard was generated and is shown in Figure 27.



predictions_vgg.csv 3.22364 3.89876
a few seconds ago by PavanGurudath
vgg16_finetuning

Figure 27: Kaggle score on VGG16 best

5.2 Summary of ResNet model

After running the ResNet-50 model, the model was observed to have not performed well. From the Figure 28 and 29, it can be seen that this model is delineating from the typical model behaviour. It can be seen that the model is being overfit onto the training dataset. A second model was trained where the last 5 layers were unfreezed and the network was trained for 5epochs due to time constraint. However, useful observations can be made after running it only for 5 epochs. Figure 30 and 31 shows the graph of accuracy and loss for 5epochs. From these graphs it can be concluded that this model requires more training, and more layers needs to be unfreezed in order to train it better. This observation is made due to the dip in the way the validation loss is increasing and that it is tending towards decreasing. It can also be seen in the way that the loss is not as high as it was in the first model as seen in Figure 29. Since the loss is continuously increasing and there is no change in the validation loss at all, we can assume that this is a bad network. This is evidenced by the kaggle score obtained for this network as shown in Figure 32

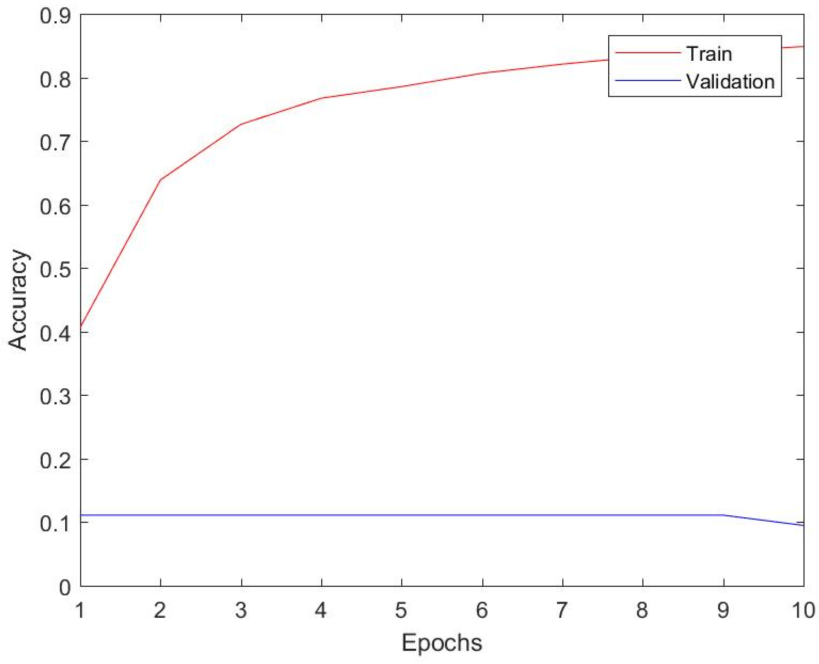


Figure 28: ResNet50: Accuracy for training and validation

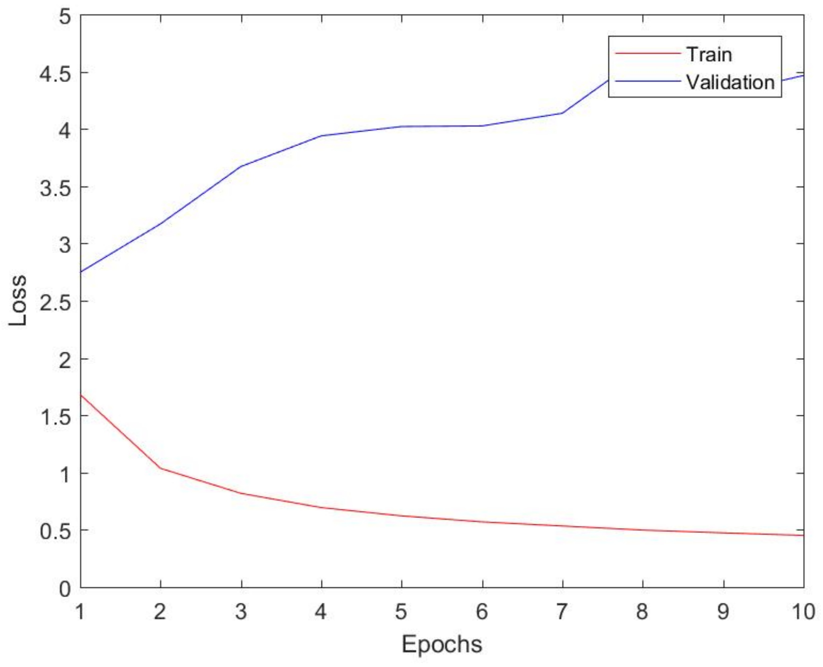


Figure 29: ResNet50: Loss for training and validation

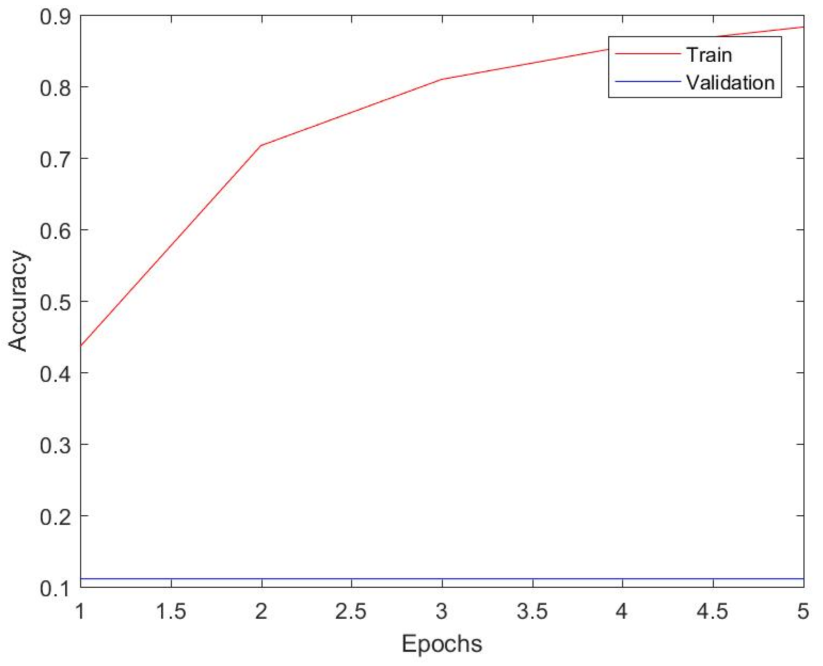


Figure 30: ResNet50 Model 2: Accuracy for training and validation

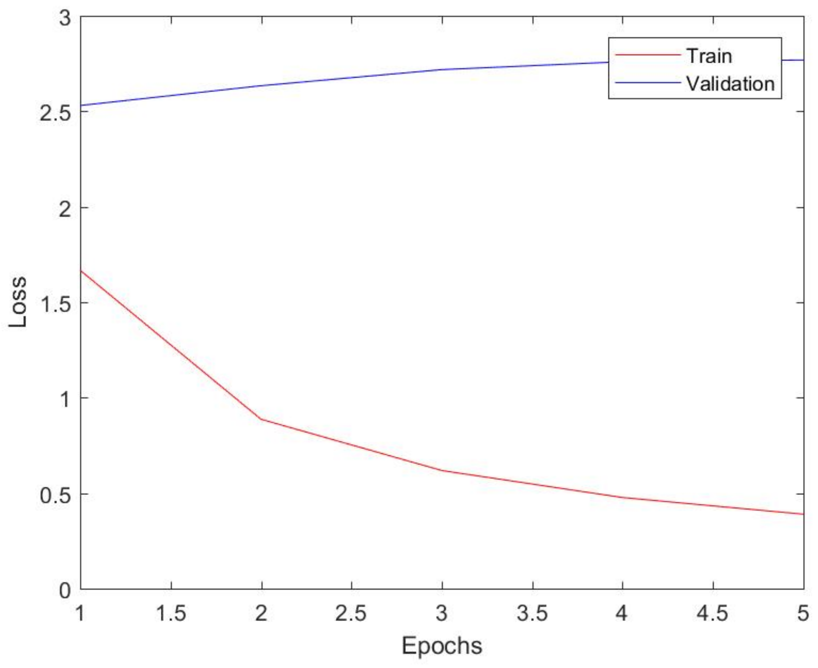
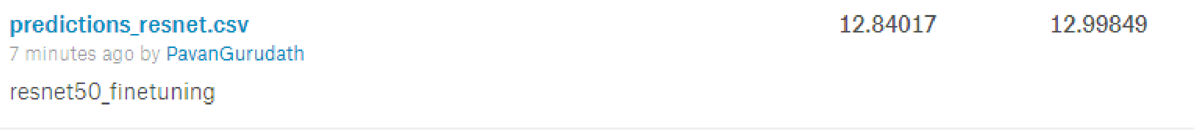


Figure 31: ResNet50 Model2: Loss for training and validation



predictions_resnet.csv 12.84017 12.99849
7 minutes ago by PavanGurudath
resnet50_finetuning

Figure 32: Kaggle score on ResNet50

5.3 Ensemble Learning

After using the two models, *i.e.* VGG-16 model 1 whose output of accuracy is shown in Figure 21 and the Resnet50 model whose graph of accuracy is shown in 30, a weighted average was done based on the Kaggle score shown in Figure 27 and Figure 32. The final log loss error that is obtained after using weights of 0.9 and 0.1 respectively for the two models, we obtain a log loss error of 2.74 on the private leaderboard and 2.845 on the public leaderboard which is shown in Figure 33. This shows that even though the second network was much worse than the first, performing an ensemble learning on them improves its accuracy.

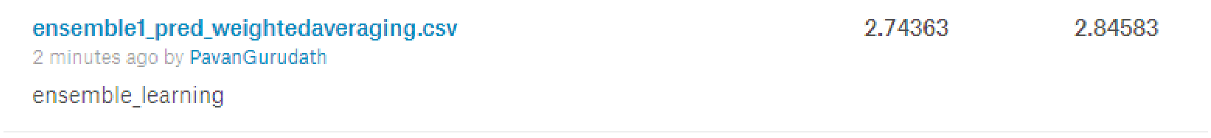

ensemble1_pred_weightedaveraging.csv 2.74363 2.84583
2 minutes ago by PavanGurudath
ensemble_learning

Figure 33: Kaggle score of Ensemble Learning

References

[1] W. H. Organization *et al.*, "Global status report on road safety, who librar. ed. doi: 978 92 4 156506 6," WHO/NMH/NVI/15.6, Tech. Rep., 2015. 3

[2] "National center for statistics and analysis distracted driving 2015. (traffic safety facts research note.report no. dot hs 812 381)," March 2017. 3

[3] "Kaggle, state farm distracted driver," Aug, 2016. [Online]. Available: https://www.kaggle.com/c/state-farm-distracted-driver-detection/data 3, 4, 5

[4] Y. Zhang, "Apply and compare different classical image classification method: Detect distracted driver," December, 2016. 4

[5] F. Chollet, "Building powerful image classification models using very little data," June, 2016. 13

[6] F. Chollet *et al.*, "Keras," https://keras.io, 2015. 14, 17, 19

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556 17

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. 19