Machine Learning Engineer Nanodegree

Capstone Project

Punita Ojha

April 7th, 2019

I. Definition

Project Overview

The problem statement is shared on kaggle.com as a competition by one of the start ups named **'State Farm - a financial insurance corporation'** .

In this competition you are given driver images, each taken in a car with a driver doing something in the car (texting, eating, talking on the phone, makeup, reaching behind, etc). Our goal is to predict the likelihood of what the driver is doing in each picture.

We need to classify each image into one of these 10 classes :

- **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 make up
- **c9 -** talking to passenger

The domain background of this problem statement is computer vision. Computer Vision is the broad parent name for any computations involving visual content – that means images, videos, icons, and anything else with pixels involved. The various sub-fields of computer vision are - Object Classification, Object Identification, Motion Analysis, Image Classification, Scene Reconstruction, Image Restoration etc. Our project falls into the category of image classification. In image classification, we train a model on a dataset of some training images, and the model classifies new images as belonging to one or more of your training categories. The solution to these kind of problems can be found using deep learning and transfer learning. Also here is the link of research paper from stanford [https://web.stanford.edu/class/cs231a/prev_projects_2016/egpaper_final%20(3).pdf] where deep learning is applied to recognise human activities from videos. Thus this research paper cites the use of deep learning in a similar type of problem .

The personal motivation for picking up problem from the domain of computer vision is before taking this course, I always heard about image recognization and classification, computer vision but how this is actually achieved was mere imagination for me. While undergoing lessons of deep learning in this course, the domain drived me crazy. The choice of project is also pretty close to my heart because if using computer vision we can reduce the number of accidents taking place by tracking or detecting distracted drivers, it will be wonderful contribution to humankind and the impact is we can reduce accidents which can save lot of innocent lives.

The link to the dataset of this problem is here https://www.kaggle.com/c/state-farm-distracted-driver-detection/data . We can download the dataset from this link . I have explained more details about the dataset of this problem in the ReadMe.md file submitted along with this . Also the section in the proposal named 'Dataset and Input' also explains the same .

Problem Statement

We are working on a real world project titled **'State Farm Distracted Driver Detection'**. The problem statement is a part of a kaggle competition where the problem statement is proposed by State Farm . We've all been there: a light turns green and the car in front of you doesn't budge. Or, a previously unremarkable vehicle suddenly slows and starts swerving from side-to-side.

When you pass the offending driver, what do you expect to see? You certainly aren't surprised when you spot a driver who is texting, seemingly enraptured by social media, or in a lively hand-held conversation on their phone.

According to the CDC motor vehicle safety division, one in five car accidents is caused by a distracted driver. Sadly, this translates to 425,000 people injured and 3,000 people killed by distracted driving every year. State Farm hopes to improve these alarming statistics, and better insure their customers, by testing whether dashboard cameras can automatically detect drivers engaging in distracted behaviors. Given a dataset of 2D dashboard camera images, State Farm is challenging Kagglers to classify each driver's behavior. Are they driving attentively, wearing their seatbelt, or taking a selfie with their friends in the backseat?

So we are given with lots of training images of the drivers performing different activity . Each of these training images belong to one class out of the classes we discussed in Problem Overview Section . For the test images, we are not shared the labels so for test images we will predict the likelihood of the image belonging to a specific class. Since this is a problem of the domain of computer vision and more specifically image classification, we will use convolutional neural networks to classify these images. Convolutional neural networks tend to give good results in cases of image classification . We will be using pre-trainned network VGG-16 for image classification with all layers of the pre-trained network freezed and if needed we will use VGG-16 with fine tuning .

Metrics

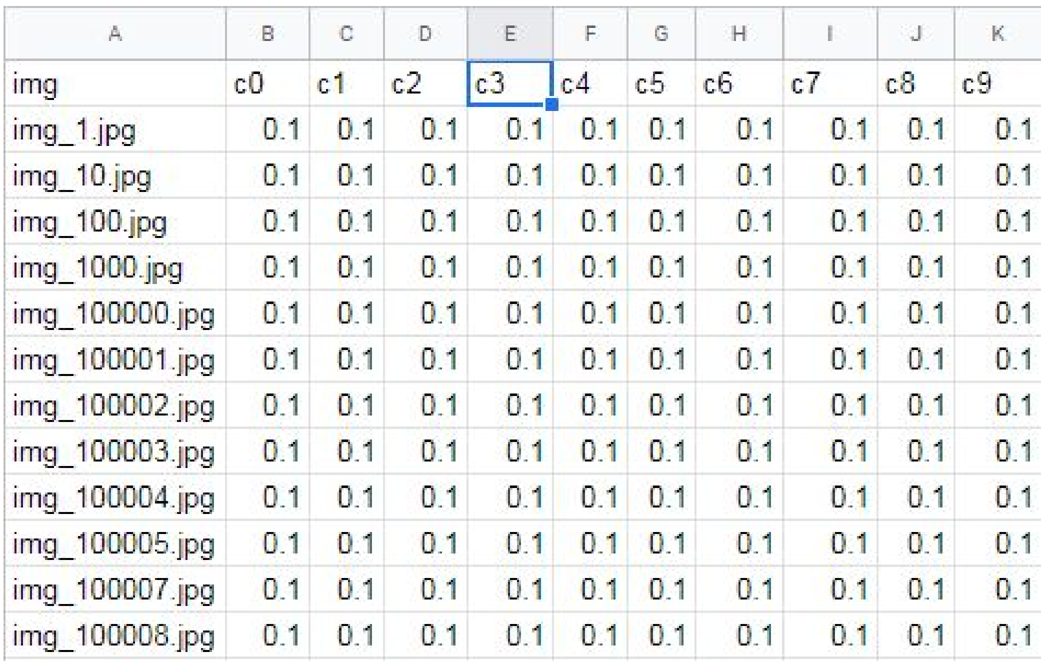
We will evaluate using the multi-class logarithmic loss function . Each image in our test set will be labeled with one true class. For each image, you must submit a set of predicted probabilities (one for every image). The formula is then,

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

where **N** is the number of images in the test set,
M is the number of image class labels,
log is the natural logarithm,
yij is 1 if observation i belongs to class j and 0 otherwise,
pij is the predicted probability that observation i belongs to class j.

The submitted probabilities for a given image are not required to sum to one because they are rescaled prior to being scored (each row is divided by the row sum). In order to avoid the extremes of the log function, predicted probabilities are replaced with max(min(p,1−10−15),10−15).

Thus, we will predict the probabilities of each test image belonging to a specific class in a spreadsheet using our trained solution and then store each of these probabilities in a csv file along with the image names and this final csv will then be uploaded on kaggle.com as a submission to this competition. Here is the format for the sample submission which is shared with us on kaggle, for convenience I have shared the screenshot of the sample submission in figure 1. So, when we minimize the multi-class logarithmic loss, we are maximizing the likelihood of your data getting predicted correctly. The multi-class log will punish large errors in your predictions while giving you smaller gains when you are close to the correct answer. Thus the metric Multi Class Logarithmic Log function chosen for this problem is reasonable and justified to use because it does not give an impression of totally correct or totally incorrect but rather illustrates the idea of giving a measure of being incorrect by how much.



A	B	C	D	E	F	G	H	I	J	K
img	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
img_1.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_10.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_1000.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100000.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100001.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100002.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100003.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100004.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100005.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100007.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100008.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

Figure 1 : Sample Format Of The Submission File To Be Uploaded To Kaggle

II. Analysis

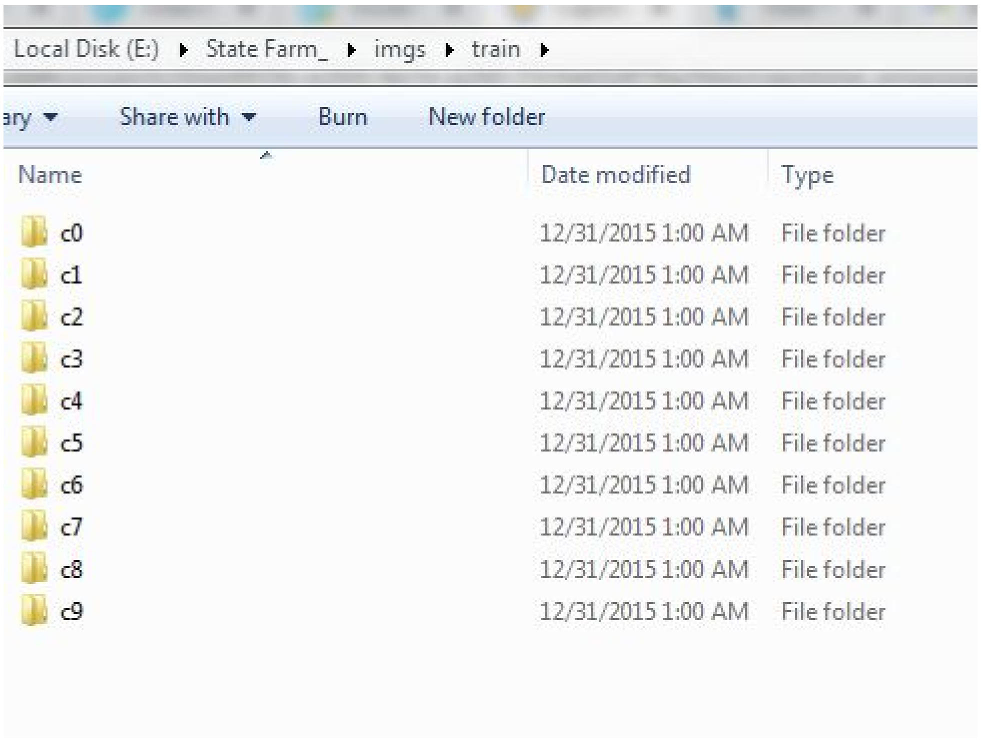
Data Exploration

The dataset for the project can be downloaded from here.

The dataset contains three files :

1. **Imgs.zip -** The zip file 'imgs.zip' contain two folders namely train and test .

- **'train' Folder** contains 10 sub-folders inside it . Each of these folders contains training images belonging to a specific class as the name specified on the folder. The brief description of each of these sub - folders and also an image to explain how these are structured is shown in figure 2.


Local Disk (E:) ▸ State Farm_ ▸ imgs ▸ train ▸

Name	Date modified	Type
c0	12/31/2015 1:00 AM	File folder
c1	12/31/2015 1:00 AM	File folder
c2	12/31/2015 1:00 AM	File folder
c3	12/31/2015 1:00 AM	File folder
c4	12/31/2015 1:00 AM	File folder
c5	12/31/2015 1:00 AM	File folder
c6	12/31/2015 1:00 AM	File folder
c7	12/31/2015 1:00 AM	File folder
c8	12/31/2015 1:00 AM	File folder
c9	12/31/2015 1:00 AM	File folder

Figure 2 : Structure Of Train Folder In Our Dataset

a) 'c0' - contains 2489 training images of drivers driving safely.

b) c1 - contains 2267 training images of drivers who are texting using their right hand.

c) c2 - contains 2317 training images of drivers who are talking on phone using their right hand.

d) c3 - contains 2346 training images of drivers who are texting using their left hand .

e) c4 - contains 2326 training images of drivers who are talking on phone using their left hand.

f) c5 - contains 2312 training images of drivers who are operating radios .

g) c6 - contains 2325 training images of drivers who are drinking .

h) c7- contains 2002 training images of drivers who are reaching behind.

i) c8 - contains 1911 training images of drivers who are engaged in grooming their hairs of doing makeup.

j) c9 - contains 2129 training images of drivers who are talking to a passenger close by.

There are total 22424 training images in our dataset with about 26 unique drivers .

test Folder - The test folder contains 79726 test images of drivers in car . We are going to make our predictions on this test images .

The size of the images in both training set and test set are 640 * 480 pixels and all the images are colour images.

Also another very important point to note about the dataset is the train and test data are split on the drivers, such that one driver can only appear on either train or test set.

2. **sample_submission.csv** - a sample submission file in the correct format.

3. Driver_imgs_list.csv - a list of training images, their subject (driver id) and class.

The sample image belonging to the training dataset are shown below with their class names are shown below in figure 3, figure 4, figure 5 and figure 6.

Since there is no validation set here, we will divide our training images into training set and validation set based on different drivers. This is very important to note that in the problem overview discussion of this kaggle.com, it is shared that the data is divided based on the fact that the drivers in the training and testing set are different. This is done, because we don’t want our model to fall into the trap of recognising drivers instead of the action they are performing in the image and henceforth we will keep this condition intact with our validation set so that the drivers in the train set and the drivers in the validation set are different .

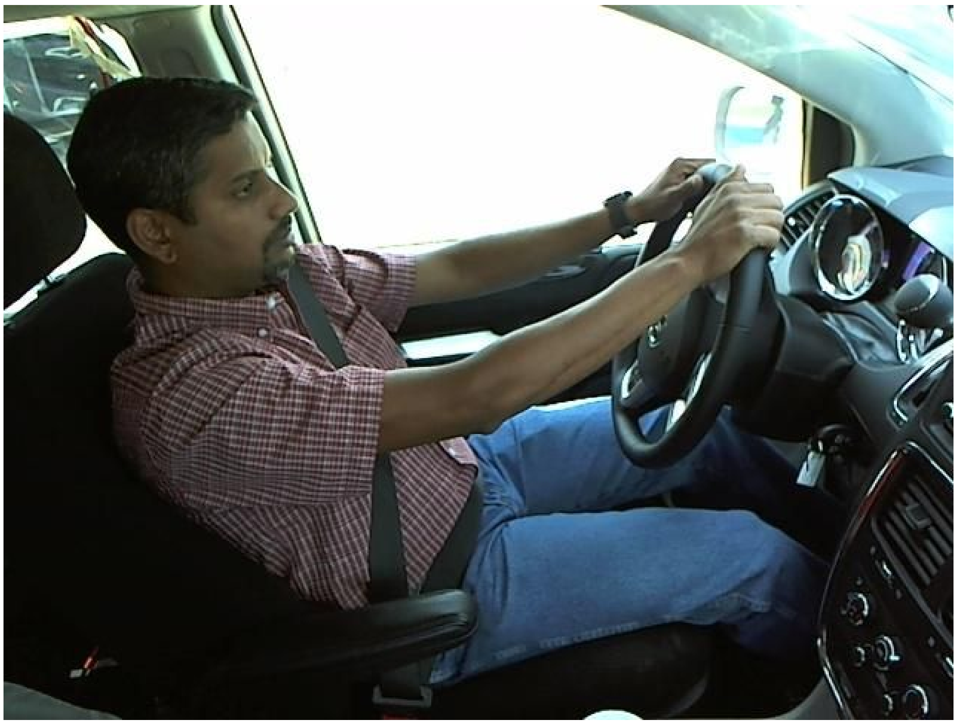


Figure 3: c0 - Image Of Driver Driving Safely

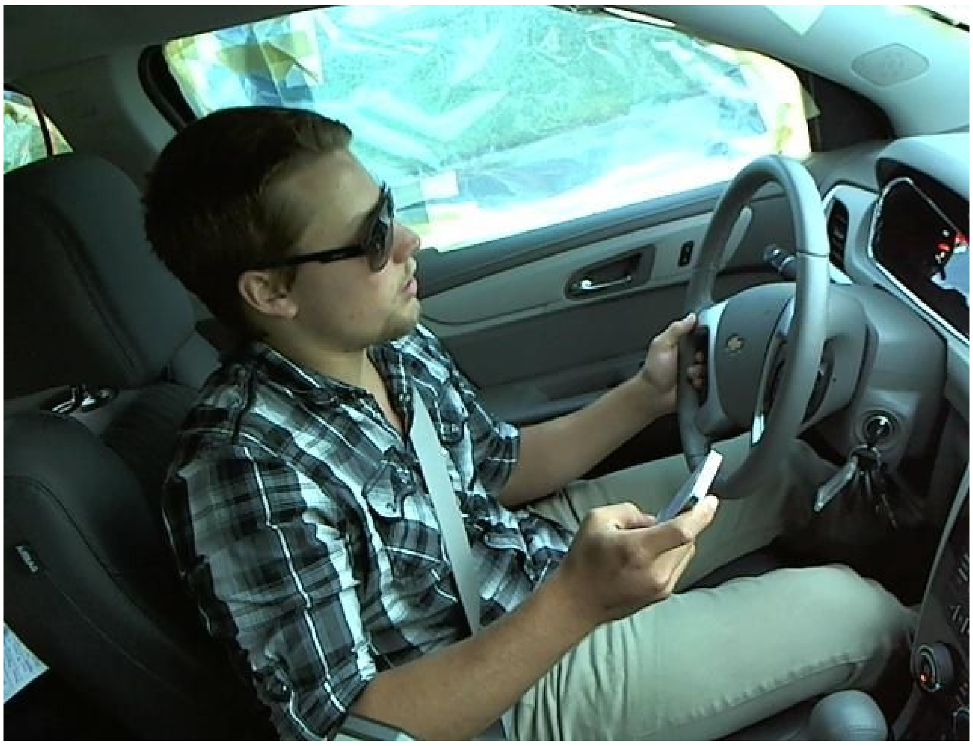


Figure 4 : c1 : Image of driver who is texting using right hand

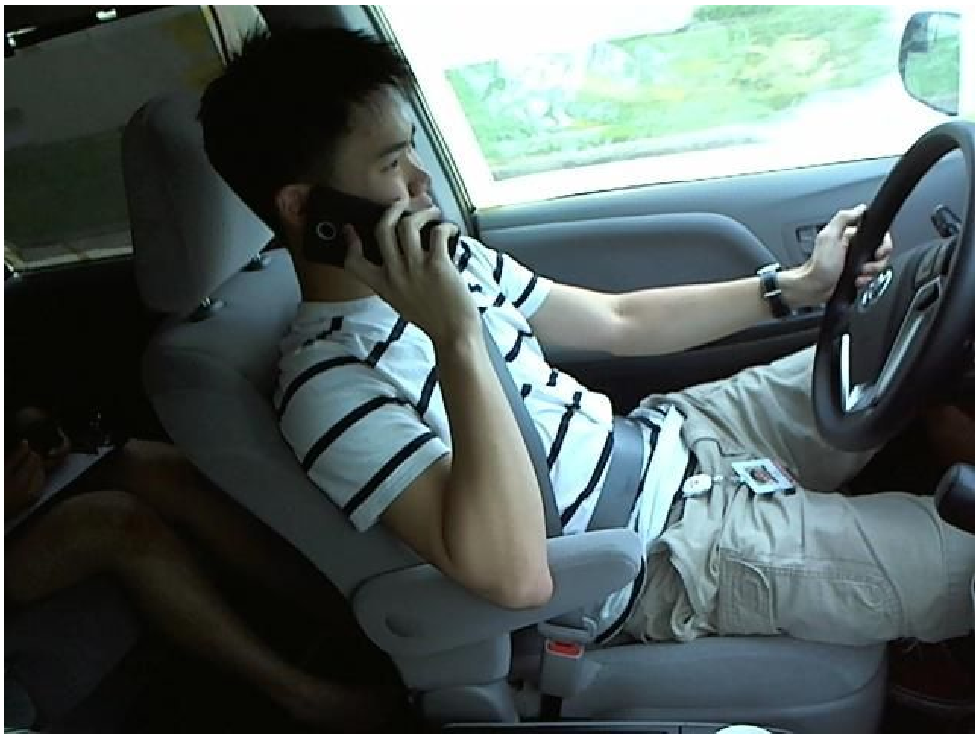


Figure 5 : Image Of Driver Talking On Phone Using Right Hand

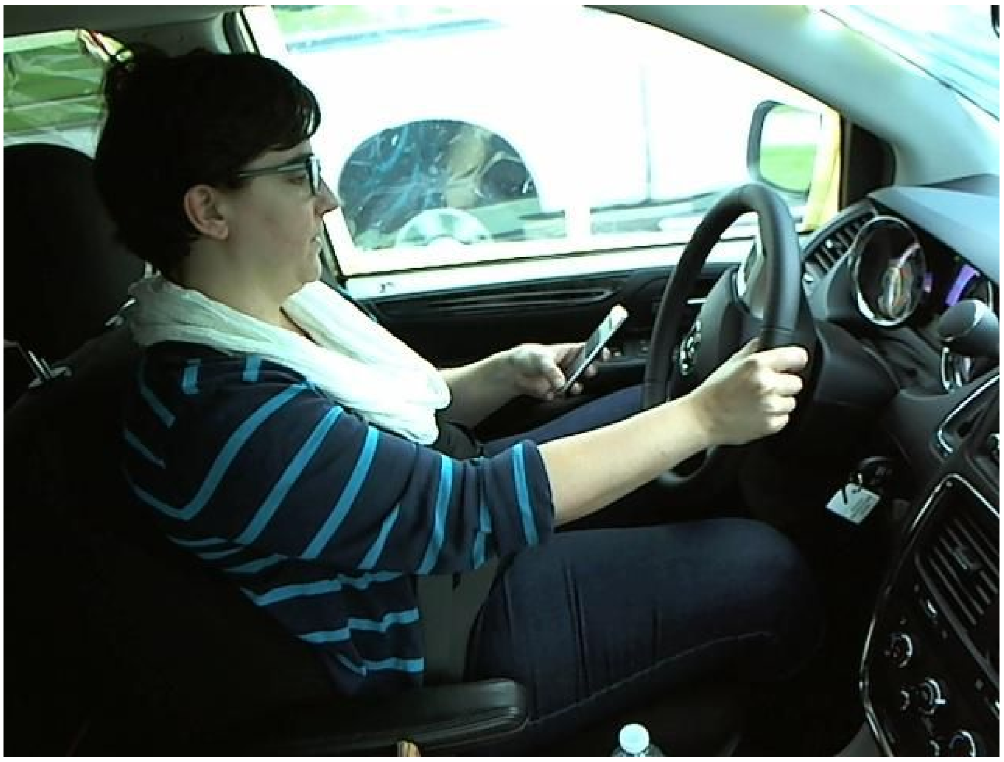


Figure 6 : Image Of Driver Who Is Text Using Left Hand

Type Of Image	**Count Of Image**
Training Images	22424
Test Images	79726

Table 1: Count Of Total Training And Testing Images In The Original Dataset

Class Name	**Number Of Training Images**
c0	2489
c1	2267
c2	2317
c3	2346
c4	2326
c5	2312
c6	2325
c7	2002
c8	1911
c9	2129

Table 2 : Count Of Total Training Images In Each Class

Type of Image	**Count Of Images**
Training Images	18351
Test Images	79726
Validation Images	4073

Table 3 : Count Of Total Train , Test And Validation Images After Creating Validation Set

Exploratory Visualization

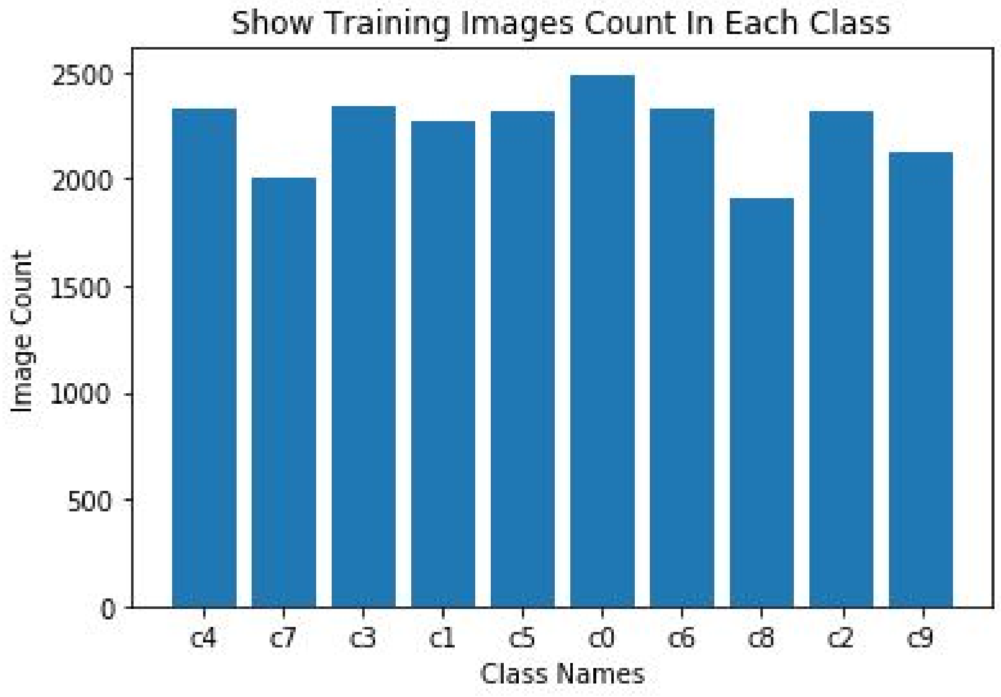


Figure 7 : Bar Graph For Count Of Training Images In Each Class

Thus we can see clearly from the plot shown in the figure 7 illustrating the count of training images in each class. It communicated that classes are not balanced some classes have more training images that means more learning data than others . We can see clearly from the plot that Class c0 have highest training images that means it has largest learning data . And it will be very difficult to predict c8 because has least training images and hence least learning data. Also we can say that it will be difficult to train this model because there are only 26 unique drivers in the training set so it is quite possible that the model aims to identify the person in the image rather than the actions or activities the person is performing.

We have also illustrated using dataset statistics in the tabular format above. Table 1 illustrates the division of training and testing images in the original dataset. Table 2 illustrated the count of training images in each class. Table 3 illustrated total train, test and validation images after creation of validation data.

Algorithms and Techniques

We have made the choice of solving this problem using deep learning which is based on the fundamental concept of convolutional neural networks. Convolutional Neural Network is one of the main categories to do images recognisation and images classifications, object detections etc. So to do image recognisation using Convolutional Neural Networks, we will pass each image as input into the convolutional neural networks, so this image will now pass through a series of convolutional layers with filters, Pooling layers and fully connected layers and then apply a softmax function to classify an object with probabilistic values between 0 and 1 . We have shown an image below in Figure 8 which illustrates a Convolutional Neural Networks with Convolutional Layers, Pooling Layers and Fully Connected Layers.

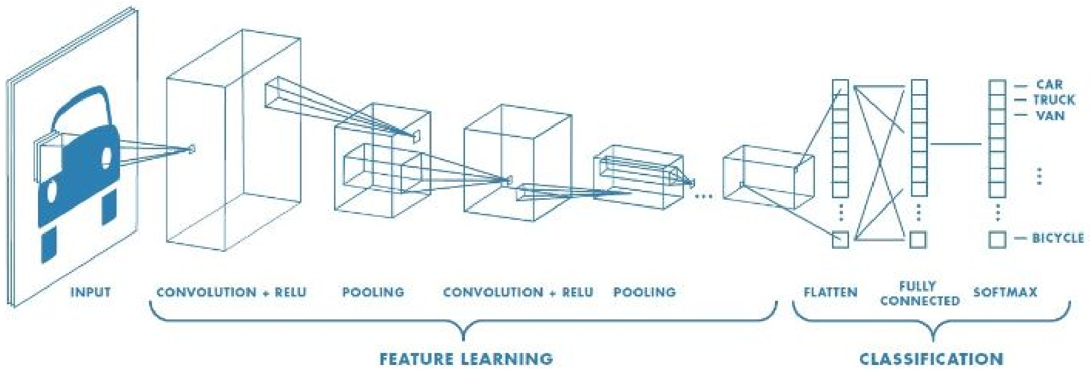


Figure 8 : Convolutional Neural Networks with Convolutional Layers, Pooling Layers and Fully Connected Layers .

Convolutional Layers : Convolutional Layer is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image using small squares of input data. The output generated from a convolutional layer is known as feature map. Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters.

Strides : Stride is the number of pixel shift over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filter to 1 pixel at a time.

Padding : Sometimes the filter does not fit the input image perfectly. We have two options .

- Pad the image with zero padding, so that the image fits. This type of padding is called same padding.

- Drop the part of the image where the filter did not fit. This is called valid padding.

ReLU : ReLU stands for rectified linear unit of a non linear equation. ReLUs purpose is to introduce non linearity in the convolutional networks.

Pooling Layers : Pooling Layers would reduce the number of parameters when the images are too large. Spatial Pooling is also called subsampling or downsampling where we reduce the dimension of each map but we retain the important information. Pooling layers can be of different types :

- **Max Pooling** - it takes the largest element from the rectified feature map.
- **Average Pooling** - it takes the average of all the elements in the rectified feature map.
- **Sum Pooling** - it takes the sum of all the elements in the rectified feature map.

Fully Connected Layers : We first flatten our matrix into vector and feed this vector into fully connected layers of the neural network .

In our Project we have used keras framework to built this network.

Transfer Learning : Transfer Learning is a important method in computer vision because it allows us to build accurate models that too in a time saving way. In transfer learning, instead of learning from scratch we start from patterns that have already been learnt when solving a different problem . Transfer Learning is usually expressed through the use of pre-trained network. A Pre-trained model is a model that was trained on large dataset to solve a problem similar to the one we want to solve. Examples of some pre-trained models are VGG-16, VGG-19, Inception, Xception, Googlenet, Alexnet etc. For the purpose of this project we have chosen VGG-16 as our transfer learning model .

VGG-16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the university of Oxford in the paper ”Very Deep Convolutional Neural Networks for large scale image recognisation ”. The model achieves 92.7% top-5 test accuracy in ImageNet which is dataset of over 14 million images belonging to 1000 classes. It was one of the famous models submitted to ILSVRC-2014. VGG-16 was trained for weeks and was using NVIDIA Titan Black GPUs .

We have shown the architecture of VGG16 model in the figure 9.

There are three types of approaches that we can when dealing with pre-trained models :

1. **Training the entire model :** this approach would need lot of time, lot of database and also lot of computational power . So we will avoid this approach in our project. We won't train VGG16 from scratch .

2 **Freeze the convolutional base:**We will keep all the layers of the pre-trained Network freezed and add some layers at the end of this network and only train those .

3. **Train some layers and freeze some layers :** This kind of approach is called Fine tuning where we unfreeze the last few layers of the pre-trained network And we train them again in our model .

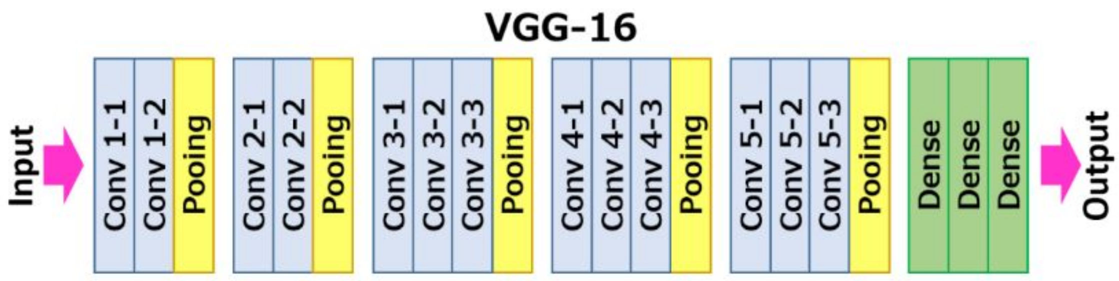


Figure 9 Architecture of VGG-16 Model

Benchmark

Our benchmark model will help us compare how better our model is doing. As suggested by the reviewer of my capstone proposal, we will pick up a simple convolutional neural network with few layers as our benchmark model and our goal will be to provide a solution that can beat our simple convolutional neural network . We will submit the results of our simple convolutional neural network (primary benchmark model) on kaggle.com and compare the score with the solution we provide to this problem state by submitting results of our model which wins over benchmark on kaggle.com. This will be our primary benchmark model.

Our alternate benchmark model will be to compare Since this is a problem statement picked up from kaggle.com , we will make our submissions on kaggle.com and compare our score against other competitors . Our goal would be to make it among top 40% of participants on kaggle.com as of now entry number 576(Top 40% of competitors) on kaggle submissions. The multi- class logarithmic loss for this 576th submission entry on kaggle.com is 1.16598. Our objective will be to perform better than this threshold value there by achieving a multi-class logarithmic loss value less than this using our transfer learning model.

We will submit a .csv file with our predicted outputs against all test images on kaggle.com and check our score against other competitors.

The evaluation criteria to get this score will be based on multi-class logarithmic loss function which is discussed in detail under Metrics section of the report.

III. Methodology

Data Preprocessing

Here are the steps taken for performing data preprocessing for this task :

- We have total 22424 images in the training set and 79726 images in the testing set in the original dataset as shown in table 1. Also the drivers in the testing set and training set are different . So we will first divide our images into training, testing and validation set. We will keep that testing set intact . We will divide our training set into training set and validation set . Since there are 26 unique drivers in the training set . We will divide our training set into training set and validation set based on driver ids . Since we are dividing the data into 80% and 20% criteria of training and validation set. We will select out of 26, 5 drivers for our validation set randomly . Then all the images of these 5 drivers belongs to our validation set . Performing so we got 18351 training images, 4073 validation images and 79726 test images as shown in table 3.

- We will store these images in the new folder named valid and will keep the structure of storing the images same as train folder as shown in figure 2.

- Now before we need these images into neural networks we will first resize them from 640 * 480 pixels to 224 * 224 pixels . We will normalize the images by dividing every pixel with 255 . We will also consider all three channels of the images RGB as these are colour images.

- Another important concern is since we are dealing with a huge dataset of 18351 training images, 4073 validation images and 79726 testing images . It is not possible for us to load all the training images into RAM in one go and then pre-process them as the RAM provided to us in Google Colab GPU environment is only 12 GB. Same is for valid and testing set . So we will use the fundamental concept of progressive loading. So we will take a small subset of images in a batch , pre process them and then train them . Then we will take the next subset, pre-process them and then train them and so on . Fortunately we need not write the code to achieve this but we can use ImageDataGenerator Class of keras to achieve this . It has and inbuilt method flow_from_directory which helps us achieve this . But the folder structure in which the images are stored is very important here . The folder structure needed is shown in figure 10 . So we can see that for train images we have a parent folder train then the classes folders for each class and then respective class images inside the class folders and same is for valid set. For test images since we don't know the classes we will have just one subfolder inside test folder . The name of the sub folder in this case can be anything . So we will now restructure our test folder using this format and then use keras ImageDataGenerator Class, flow_from_directory method to achieve this .

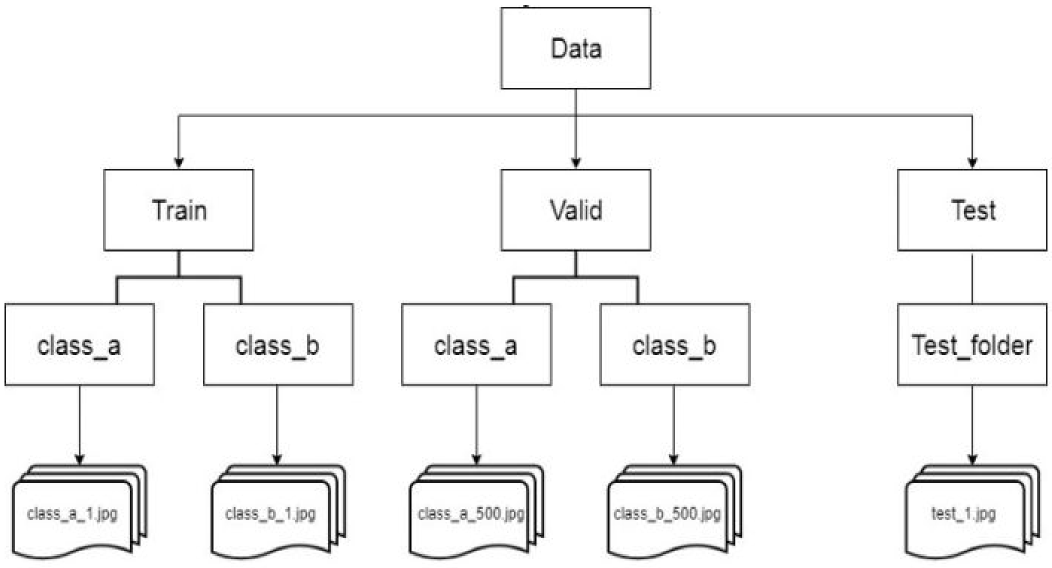
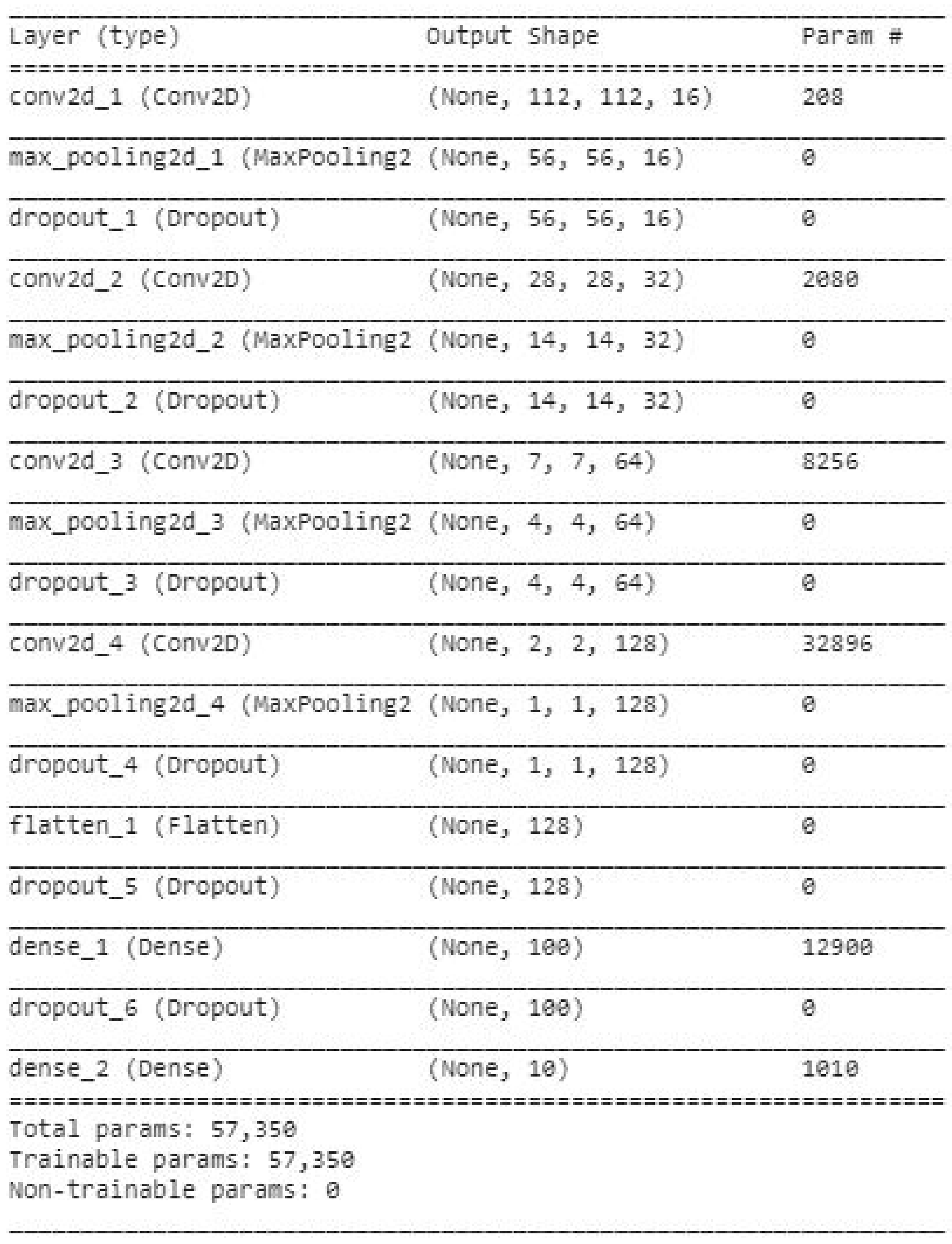


Figure 10 : Folder Structure needed to use keras flow_from_directory Method .

Implementation

Here are the steps of implementation of a Simple Convolutional Neural Network .

1. We divided our images into training and validation set based on driver ids . We also stored our validation images in the folder structure needed by keras ImageDataGenerator Class , flow_from_directory method .
2. We restructured our test folder as per our keras need of ImageDataGenerator Class flow_from_directory method discussed in Data Preprocessing section of this report .
3. We designed our simple convolutional neural network architecture as shown in figure 11.
4. We can see from the figure that the simple convolutional neural network consists of a pair of 4 convolutional layers with max-pooling layers in between . The filters in the convolutional layers are increasing from 16 in the first convolutional layer to 32 in the second convolutional layer, to 64 in the third convolutional layer to 128 in the fourth convolutional layer. Then we added the flatten layer to convert the matrix into a vector and then pass it to a dense layer of 100 nodes and through it to an output dense layer of 10 nodes . We have used ReLU activation function in our convolutional layer and dense layer of 100 nodes . We have used softmax function as activation function in the output layer with 10 dense nodes so that it can give th probabilistic probabilities . So have also added Dropout layers in our architecture to get better results and avoid overfitting of our model .
5. We have then compiled our model using adam optimiser and categorical cross entropy as the loss function .
6. The model is then trained for 80 epochs with a batch size of 32 .
7. Please note that we have used progressive loading of images into neural network to achieve this task but pre-processing all the images in one go and then training them was causing RAM out of memory issues and due to large data and insufficient RAM we were ending up in crashing our RAM . So we found a method of dealing with larger dataset problems in machine learning known as progressive loading .
8. After training the model, we pre-processed our test images and then predicted the probabilities of test images and then submitted it to kaggle.com in the format expected by them . The sample of that format is shared in figure 1.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 112, 112, 16)      208       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 16)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 56, 56, 16)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 32)        2080      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 64)          8256      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 64)          0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 64)          0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 2, 2, 128)         32896     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 1, 1, 128)         0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 1, 1, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0         
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 100)               12900     
_________________________________________________________________
dropout_6 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010      
=================================================================
Total params: 57,350
Trainable params: 57,350
Non-trainable params: 0
_________________________________________________________________
```

Figure 11 : Simple CNN Architecture

Refinement

So with the simple convolutional neural network architecture shown in figure 11, we scored a log loss of 1.53978 on test set in the kaggle public score .

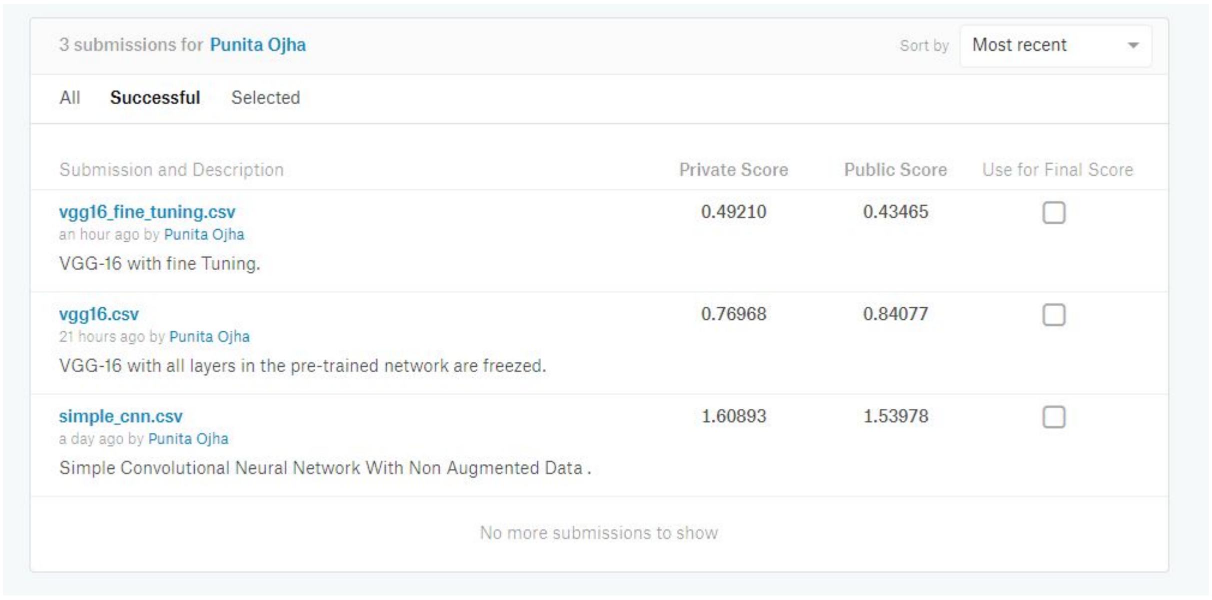
Then to refine the results produced from this we decided to use transfer learning using VGG16 architecture will all layers freezed initally . We removed the last dense layer of the VGG16 architecture and instead added our own dense layer of 10 nodes we also added a dropout layer before dense layer to avoid overfitting . Doing so we could achieve a kaggle score of 0.84077 on test dataset on kaggle public score board . So this refinement technique helped us perform better than our benchmark model - the simple convolutional neural network .

We further decided to reform this architecture and work on fine tuning VGG16 . So we freezed all the layers of the VGG16 architecture except the last 4 layers and then added a dense layer of 518 nodes followed by a BatchNormalization Layer and relu activation function and an output layer of 10 nodes and followed by a Batchnormalization layers and a softmax activation function . Batch normalisation is a technique for improving the performance and stability of neural networks and also makes more sophisticated deep learning architectures work in practice.

The idea is to normalise the input of each layer in such a way that they have a mean output activation of zero and standard deviation of 1. This is how input to networks are standardized . Performing operations this technique helped us score 0.43465 on test set on kaggle public score .

IV. Results

Model Evaluation and Validation



3 submissions for Punita Ojha

Sort by Most recent

All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
vgg16_fine_tuning.csv an hour ago by Punita Ojha VGG-16 with fine Tuning.	0.49210	0.43465	☐
vgg16.csv 21 hours ago by Punita Ojha VGG-16 with all layers in the pre-trained network are freezed.	0.76968	0.84077	☐
simple_cnn.csv a day ago by Punita Ojha Simple Convolutional Neural Network With Non Augmented Data .	1.60893	1.53978	☐

No more submissions to show

Figure 12 : Kaggle Results Obtainned On Simple CNN, VGG16 and VGG16 Fine tuning

Parameter / model	**Simple CNN**	**VGG16**	**VGG16 Fine Tuning**
No. of epochs trained for	***80***	***150***	***85***
Training Time	***2.6 Hours***	***6.4hours***	***3.6 hours***
Training Accuracy On Best Weights	***72.55%***	***91%***	***99%***
Validation Accuracy On Best Weights	***50.98%***	***76%***	***90%***
Validation Loss On Best Score	***1.4241***	***0.7380***	***0.43514***
Kaggle Public Score	***1.53978***	***0.84077***	***0.43465***
Approx Kaggle Rank On Public Score Board (out of 1440 submissions)	***784***	***473***	***280***
Top how much percent on kaggle	***54.4%***	***32.8%***	***19.4%***

Table 4 : comparisons of our three solution models based on various parameters

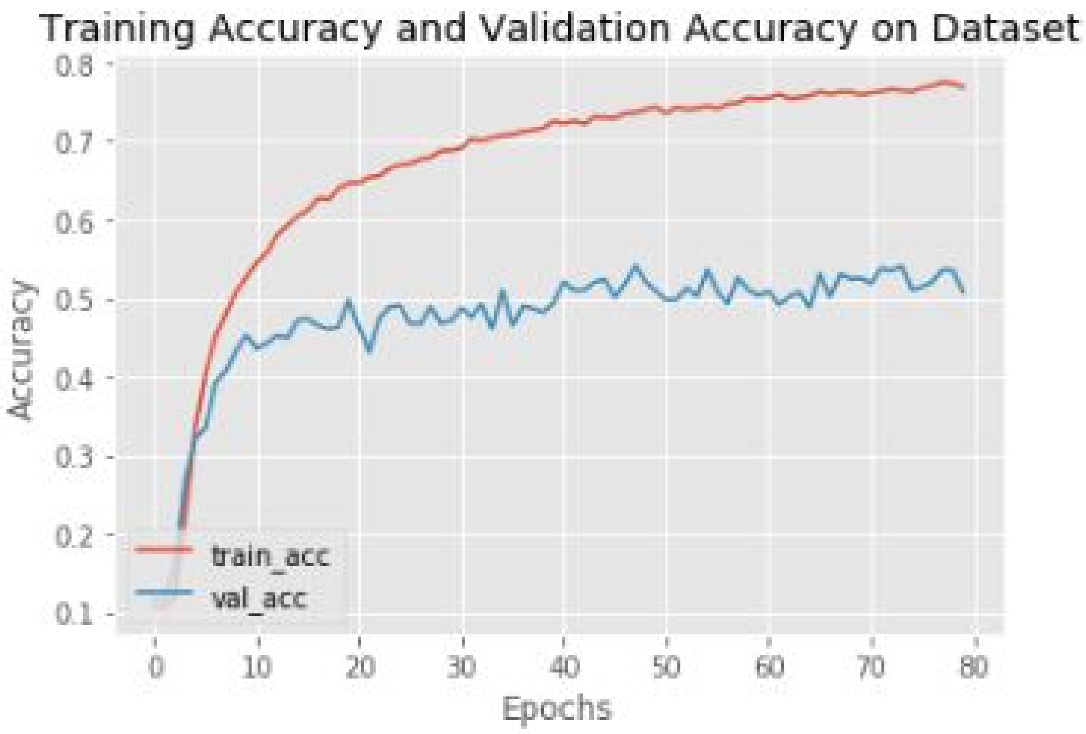


Figure 13 Graph plot of training accuracy and validation accuracy on Simple CNN

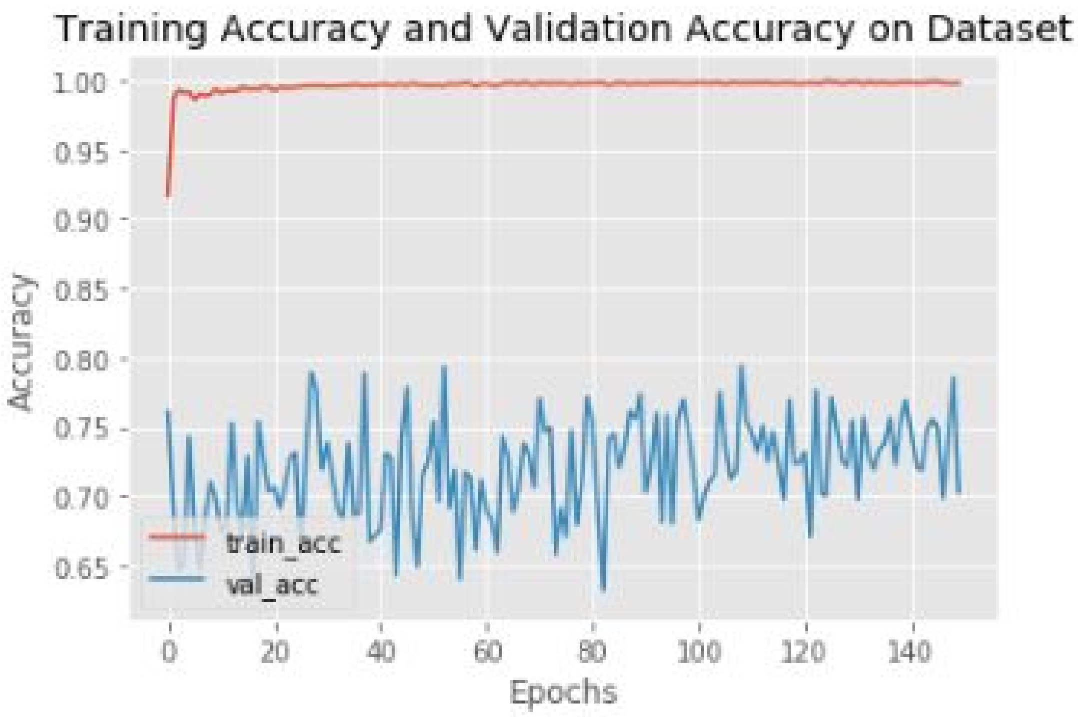


Figure 14 Graph Plot Of VGG16 With All Pre-trained Layers Freezed

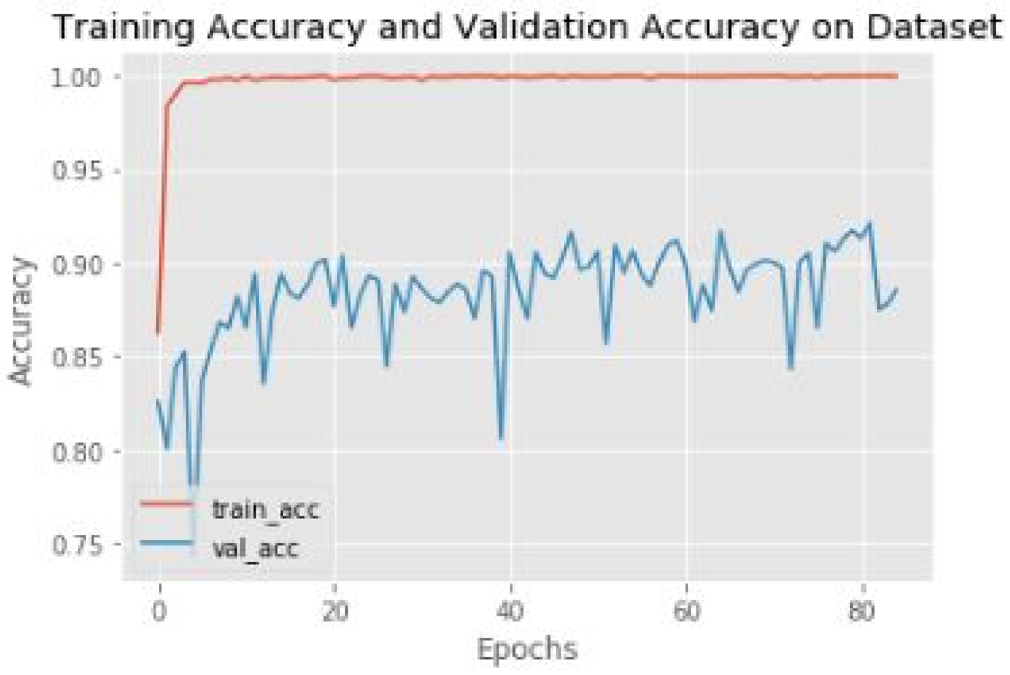


Figure 15 Graph Plot For VGG16 Fine tuning model

The final model which is VGG16 Fine tuning architecture is the best out of all the solution models that we have given here . We can see from the comparisons made in table 4 that VGG16 fine tuning model performing best on kaggle public score by scoring in the top 19.4 percent of the participants on kaggle public score board on the testing data with a rank of 280 out of 1440 and public score of 0.43465 . Also another important thing to observe here is the training and validation accuracy graphs which we have shared for all the three models in figure 13 for simple cnn, figure 14 for VGG16 with all pre-trained layer freezed and figure 15 for VGG16 with fine tuning. We can see that for simple CNN the highest training accuracy we can reach is 80% where as validation accuracy is around 55% and we observe a case for overfitting. In case of transfer learning with VGG16 all pre-trained layers freezed we can achieve training accuracy of 100% but validation accuracy of 79%. But we got a much wonderful observation with VGG16 Fine tuning model achieving training accuracy of 100% and validation accuracy of 91% . Though we could easily overcome and outperform better than our benchmark, but I would not say that this is the best possible solution to achieve but clearly we have made significant improvement in overcoming overfitting observed in simple cnn and vgg16 with all pre-trainned layers freezed and hence the solution satisfies me but as a part of future project we will try to perform better than this by trying out strategies mentioned in the improvements section. .

Justification

Our final model VGG16 fine tuning scored 0.43465 on kaggle public score which would rank at 280 out of 1440 submissions on kaggle public board on this competition. That is top 19.4% of the kaggle public score board . It took 2.6 hours for this model to make prediction on test set of 79726 images of size 340 * 480 pixels . Yes the final solution is sufficient enough to solve the problem because we have managed to successfully overcome both our primary and alternate benchmark model. The goal of our primary benchmark model was our simple CNN to score better than 1.53978 on public score board, the public score of 0.43465 obtainned on VGG16 Fine tuning model has managed to outbeat the primary benchmark model and our alternate benchmark model of scoring in top 40% submissions on kaggle.com public score board is also outbeaten by scoring in top 19.4% by our VGG16 fine tuning model .

V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization

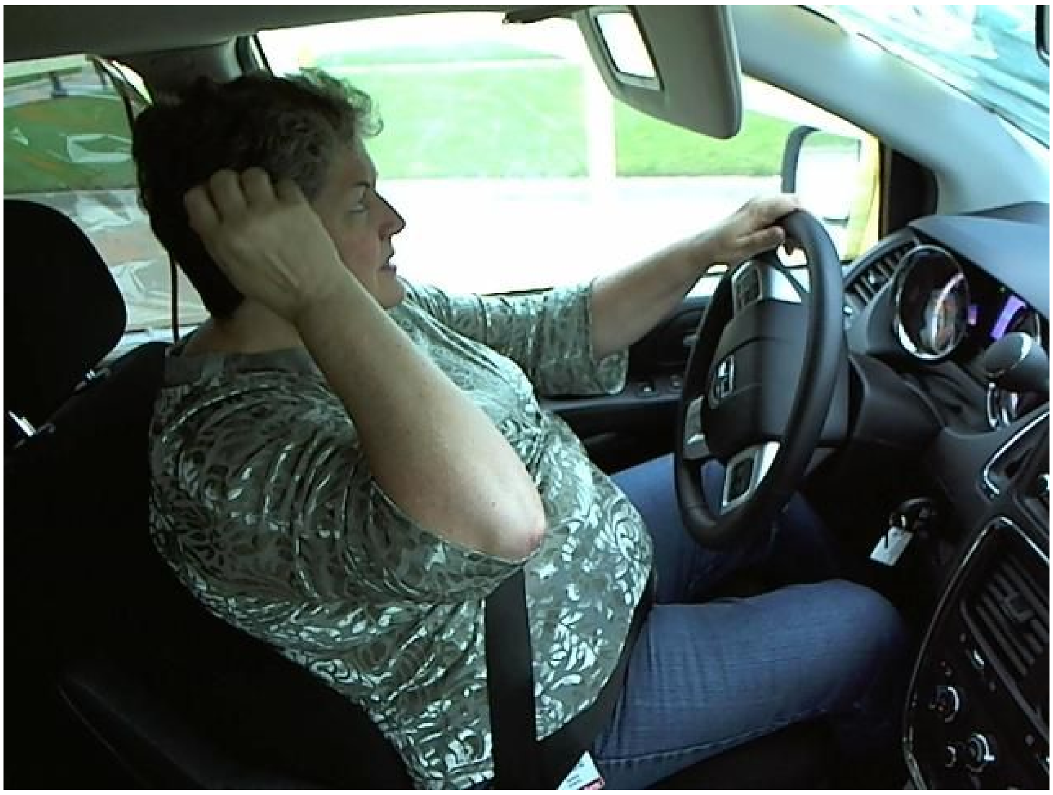
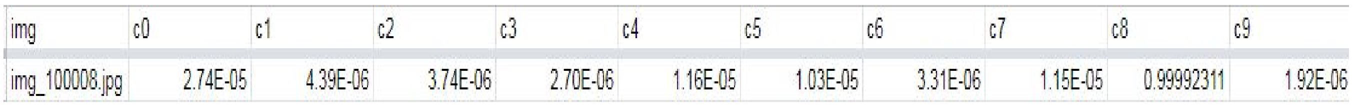


Figure 16 Test Image img_100008.img

Test Image with name img_100008.img is predicted on our VGG16 fine tuning model as belonging to class 8 as driver grooming hairs or doing makeup . and we can make out from the predictions that this is correct . We are drawing these predictions from vgg16_fine_tuning.csv file as we can see for this image c8 got 0.999 probability prediction, we have also added screenshot in figure as solution.



img	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
img_100008.jpg	2.74E-05	4.39E-06	3.74E-06	2.70E-06	1.16E-05	1.03E-05	3.31E-06	1.15E-05	0.99992311	1.92E-06

Figure 17 Prediction for image made by vgg16_fine_tuning.csv file

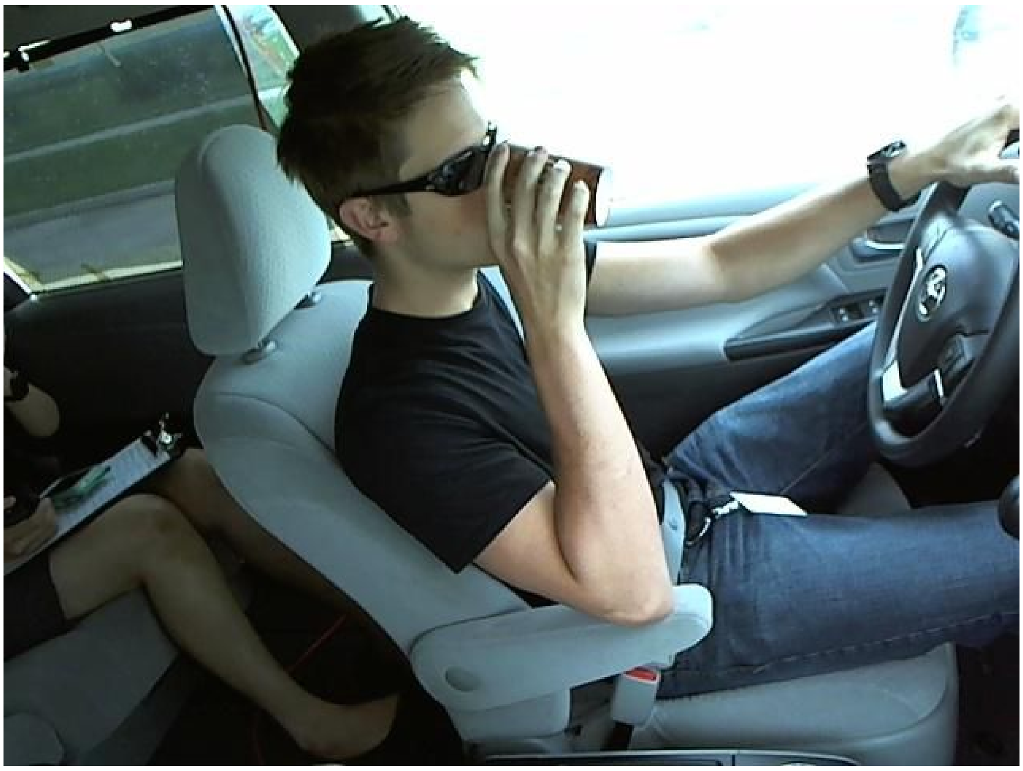


Figure 18 Test Image img_100004.jpg

Test image with name img_100004.jpg is predicted on our VGG-16 fine tuning model as belonging to class c6 and performing activity drinking . And we can see from the visualization that this is correct .

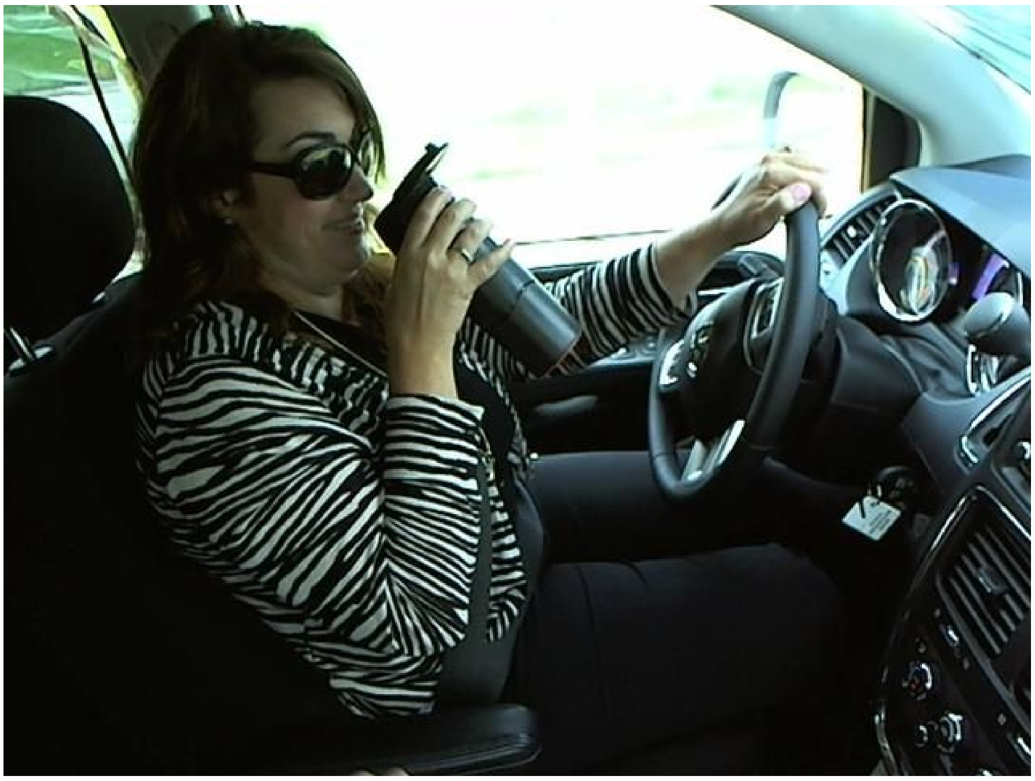


Figure 19 Test Image img_100019.jpg

Test image with name img_100004.jpg is predicted on our VGG-16 fine tuning model as belonging to class c6 and performing activity drinking . And we can see from the visualization that this is correct .

To verify these predictions please visit the submission file vgg16 fine tuning .

Reflection

The summary of the process we followed to reach to the solution of the problem :

- We started with looking for a problem and problem domain we are passionate about Deep Learning was the domain which always attracted me and then I came across this challenge of State Farm Distracted driver detection on kaggle.com . Since the dataset was easily available on kaggle.com and their were no copyright issues which using this dataset for our project we went with this .

- We gathered understanding of the problem by reading about it in the problem overview section on kaggle.com .

- Then we started with understanding the dataset available to us and complexities with the dataset . We planed the preprocessing steps we need to work with this dataset .

- Since the problem was from the domain of deep learning, we prepared a rough draft explaining problem statement, dataset, exploratory visuals of the dataset, benchmark models, designing and planning strategy we will use to solve this problem and submitted the entire planning as a part of our capstone project . This planning helped us in understanding our project and our approach in a better way .

- Then we started with simpler architectures and simpler steps first and then to refine we moved to complex architectures .

- We figured out that we will be GPU based environment to work on this usecase and hence went through some quick start tutorials on Google Colab to leverage the free 12 hour GPU they give .

- We wrote down steps in sequence and troubleshooted any difficulties that we encounter during practical implementation . Like an example is when we tried to preprocess all the training image in one go . We ran into RAM out of memory issues and our RAM crashed . We used the fundamental concept of progressive loading to figure out the results .

- Finally after working with simple convolutional neural networks, we worked on hypertuning some parameters and added some dropout layers to get better results on simple convolutional layer . We trusted this model as our benchmark model and submitted the predictions on kaggle.com .

- Then we worked on refining our simple convolutional neural networks and worked on VGG16 architecture with all layers freezed. We made our submission to kaggle.com and scored better than simple convolutional neural network .

- Lastly we worked on fine-tuning VGG16 architecture and played with adding more layers to the end of this . Using concepts of batch normalization to score higher and submitted results to kaggle.com and scored better than VGG16 architecture with all pre-trained layers freezed .

- We then drafted our conclusions and observations into a capstone report for submission

Improvement

The following improvements are possible in the solution for future purposes : -

- We can use ensemble methods to achieve model accuracy . Like we can use GoogleNet and InceptionV3 and combine the results produced from these two and get our final solution.

- We can try implementing more complex architectures of transfer learning like GoogleNet and AlexNet etc .

- We can try keep the size of the images larger than 224 * 224 pixels like may be 448 * 448 pixels so that we can get better results . This would take us more time to train our network and also

- We can apply techniques like Data Augmentation to avoid overfitting . Also since the number of training images are less compared to that of testing images, Data Augmentation will help us produce oversampling of images and hence produce better results . Data augmentation is the technique of rotation, translation and flipping the images to get more variance in the training data and hence produce better outputs . Please note that we have already written some starter code in our project which can be used in future for data augmentation however we have avoided exploring it for the course of this as it was taking more than 5 minutes to train 1 epoch on VGG16 all layers freezed architecture and since we have only 12 hours of GPU instance avaiable, we choose to try data augmentation in future usecases .

References

- Problem Statement : State Farm Distracted Driver Detection
  https://www.kaggle.com/c/state-farm-distracted-driver-detection

- Brief description of computer vision and its sub-fields
  https://blog.algorithmia.com/introduction-to-computer-vision/

- Better understanding of transfer learning
  http://cs231n.github.io/transfer-learning/

- Convolutional Neural Networks
  https://www.youtube.com/watch?v=bNb2fEVKeEo

- Dealing with the funda of progressive loading in keras
  https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

- Understanding of Convolutional Neural Networks
  https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

- Progressive Loading Of Dataset in Machine Learning .
  https://medium.com/@vijayabhaskar96/tutorial-image-classification-with-keras-flow-from-directory-and-generators-95f75ebe5720