ANTALYA BİLİM ÜNİVERSİTESİ

Graduation Project

Project Title: Distracted Driver and Drowsiness Detection

Aissa HOUDJEDJ

Ahmed Amine Taleb Bahmed

Supervisor

Prof. Hilal Kazan

08-01-2018

Table of Contents

Abstract

Every year, many car accidents due to driver fatigue and distraction occur around the world and cause many casualties and injuries. Engineers and researchers in the automobile industry have tried to design and build safer automobiles, but traffic accidents are unavoidable. As Computer Engineers, if we deeply investigated the causes of these accidents, we will find that these behavioral and roadway accident patterns can be useful to develop traffic safety control policies.

We believe that to obtain the greatest possible accident reduction effects with limited budgetary resources, it is important that measures be based on scientific and objective surveys of the causes of accidents and severity of injuries.

This paper is about how to address this problem by Internet of Things and machine learning, two technologies that are taking the world by storm and will someday become an inherent part of every aspect of our lives.

1.Introduction

The transportation industry is associated with high maintenance costs, disasters, accidents, injuries and loss of life. Hundreds of thousands of people across the world are losing their lives to car accidents and road disasters every year. According to the World Health Organization, there were 1.25 million road traffic deaths globally in 2013. [1]

The related costs — including medical expenses, wage and productivity losses and property damage — were estimated at $152 billion. And this doesn't account for general maintenance and repairs costs of the road and highway systems, which earmark billions of dollars of public funds every year — and are still underfunded.

These statistics motivated us to work on developing a System that helps in reducing the amount of car accidents around the world, where There are several approaches that researchers have employed to study this problem. These include neural network and so on. Thus, detecting the driver's state is the beginning of our journey, where more than a quarter of all car crashes in America are likely caused by cell phone use [2].

This report presents our project's idea, where there is one promising strategy involves classifying the driver state in real time and then using this classification to adapt the in-vehicle technologies to mitigate the effects of distraction [3]. So we are going to work on creating a system that detects the Driver behaviors while driving and acts according to his behavior, if he/she uses phone, making-up, eating..etc, using multiple Machine Learning classification models.

2.Statistics

- Every 12 minutes, one person dies because of a car accident. Every 14 seconds, a car accident results in an injured victim.
- 25-50 percent of all motor vehicle crashes in the U.S. are directly related to driver distraction.
    - Driver fatigue (12 percent)
    - Looking at scenery (10 percent)
    - Other passengers or children (9 percent)
    - Adjusting the radio or CD player (7 percent)
    - Reading the newspaper, books, maps, or other documents (less than 2 percent)

  The number of people killed in auto-related accidents is expected to rise by 65 percent by the year 2020.

3.Distracted Driver Detection

3.1 Dataset

The used dataset in our project was supplied by a challenge on the Kaggle platform, the name of the challenge is “State Farm Distracted Driver Detection” , the given Dataset is about 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).

The dataset consists of 22400 training and 79727 testing images (640 × 480 full color) of people either driving safely or doing one of eight kinds of distracted behaviors [4].

Figure-1 *- A sample of Dataset*

The training images come with correct labels and the challenge is to make the best multi-class classifications possible. There are nine classes in total:

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

3.2 Splitting the Dataset

At the beginning we splitted the data into 70% for training and 30% for Testing, because the testıng dataset from kaggle platform was not labeled, and the 70% of training set was used To evaluate the success of models, so we split it into 70% for training and 30% for validation. as it is shown in the **Figure-2**.

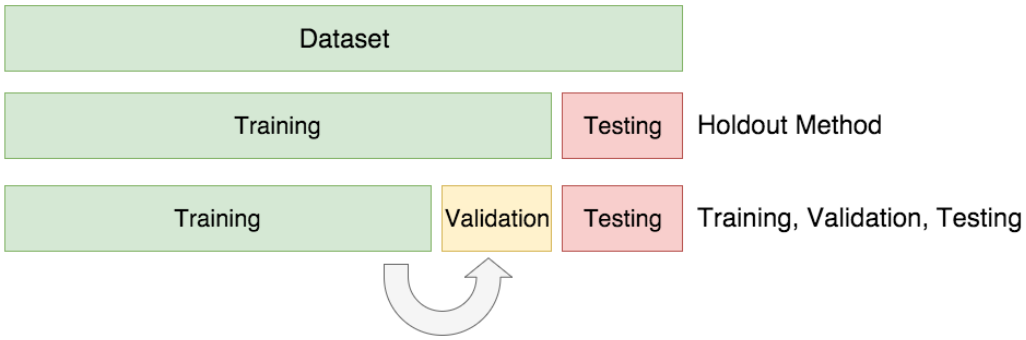


Figure-2 *- Splitted Dataset*

Then we changed the idea of the splitted dataset, as following, we splitted the dataset into different 26 drivers, then we splitted the dataset into 20 drivers for Training and 6 for Testing. Then we used the cross-validation in while training as shown in the **Figure-3** below.

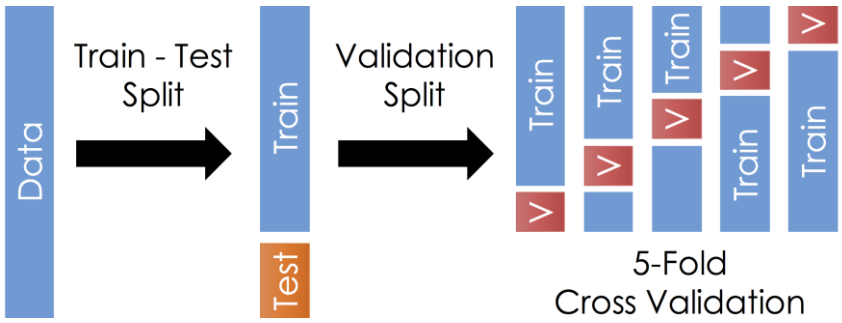


Figure-3 *- Splitted Dataset with Cross Validation*

3.3 Methods and Results

3.3.1 Transfer Learning

Transfer learning is the idea of using a CNN model pre-trained on a large dataset as an initialization [5]. It gave us a significant boost in terms of speed and performance. For each model we used in this project, we only modified the last FC layer to output 9 class predictions instead of 1000 or more. And we added some FC layers before the output layer, We then use our own training set as the input images to train the whole neural network.

3.3.2 Fine Tuning

1. Feature extraction – We can use a pre-trained model as a feature extraction mechanism. What we can do is that we can remove the output layer (the one which gives the probabilities for being in each of the 1000 classes) and then use the entire network as a fixed feature extractor for the new dataset.
2. Use the Architecture of the pre-trained model – What we can do is that we use architecture of the model while we initialize all the weights randomly and train the model according to our dataset again.
3. Train some layers while freeze others – Another way to use a pre-trained model is to train is partially. What we can do is we keep the weights of initial layers of the model frozen while we retrain only the higher layers. We can try and test as to how many layers to be frozen and how many to be trained.

3.3.3 Convolutional Neural Network

Convolutional neural network (CNN) is an application of neural networks (NN), the difference is that they are customized to have images as inputs [5]. This means that the neurons are now structured as a 3D volume. Layers of a CNN transforms one volume to another.

3.3.2.1 Input Layer

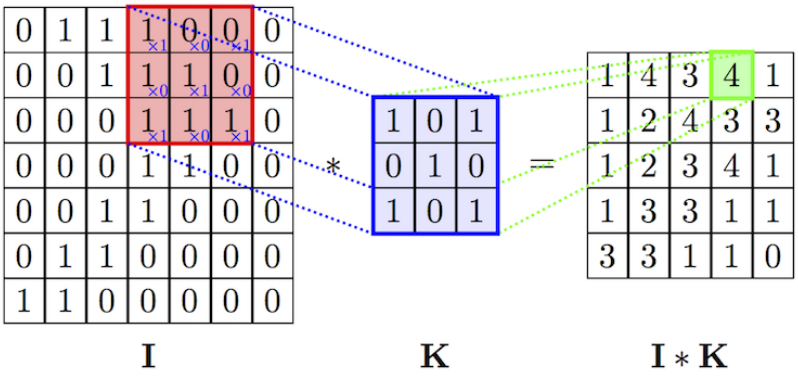
The input layer is the first layer the Convolution Neural Networks, It holds the pixel values of the input image. The input layer in our project is 640 × 480 × 3 (for R, G, B channels).

3.3.2.2 Convolutional (CONV) Layer

The CONV layer's parameters are a set of learnable filters of small dimensions. The filter convolves with the input volume (across width and height in 2D) to select small areas (e.g. 3 × 3) as it is shown in Figure below, and use these small local areas to compute dot products with weights/parameters.

Each filter corresponds to a slice or a depth of one in the output volume (i.e. the depth of the output volume of a CONV layer is determined by the number of filters).

In the figure below, the input is (I) and the filters is (K), and the output of the Conv is I*K.



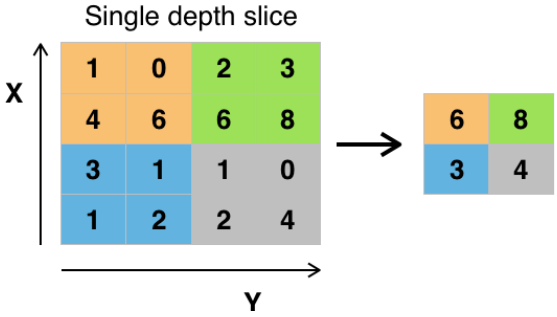
3.3.2.3 Rectified Linear Units (ReLU) Layer

The rectifier function is an activation function f(x) = Max(0, x) which can be used by neurons just like any other activation function, a node using the rectifier activation function is called a ReLU node, It is used to increase nonlinearity in the model.

3.3.2.4 Pooling Layer

A pooling layer reduces the 2D dimensions of the input volume (leaving the depth unchanged) to prevent the model from overfitting and getting too large (too many weights) to compute. It is done independently for each depth slice of the input volume by applying a small filter (e.g. 2× 2). The most popular pooling function is the max function.

Using the Max Pooling. Basically, it takes a filter (normally of size 2x2) and a stride of the same length. It then applies it to the input volume and outputs the maximum number in every sub region that the filter convolves around.

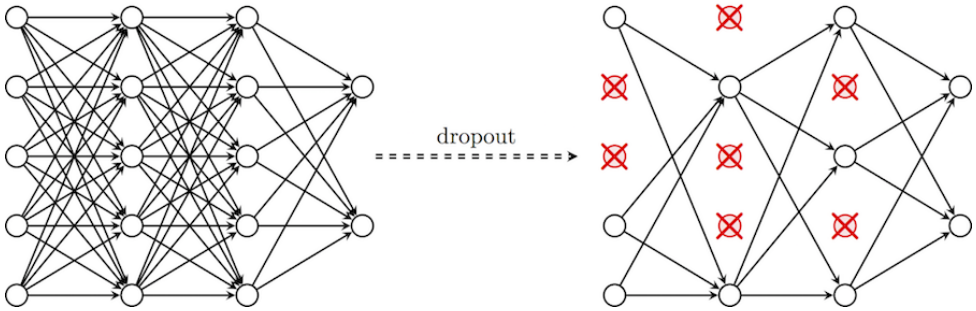


3.3.2.5 Dropout Layer

Adding dropout layers is a regularization method to prevent overfitting. A dropout layer randomly sets some unit activations to zero and thus removes some feature detectors [6].

Basically during training, a set of neurons on a particular layer will be deactivated. This improve generalization because it forces the layer to learn with different neurons the same "concept".

During the prediction phase the dropout is deactivated.

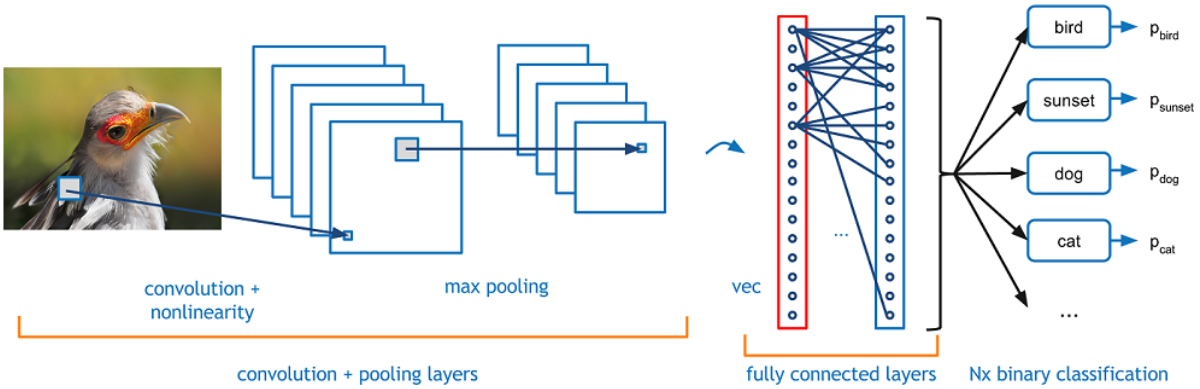


3.3.2.6 Fully-connected (FC) Layer

Fully-connected layers, as the name suggests, is like ordinary NN where each neuron is connected to all the outputs from the previous layer. The last FC layer computes probabilities for each class. For multi-class classification, softmax is a popular choice. Softmax regression has the following log-likelihood function:

$$l(\theta) = \sum_{i=1}^{m} log \prod_{l=1}^{k} \left(\frac{exp(\theta_l^T x^{(i)})}{\sum_{j=1}^{k} exp(\theta_j^T x^{(i)})} \right)^{1\{y^{(i)}=l\}}$$

That is, Softmax trains the final layer to correctly predicts with maximal confidence for each image.



3.4 VGG-16

VGG-16 is a 16-layer CNN developed by the University of Oxford Visual Geometry Group (VGG). We used the pre-trained VGG-16 model in Keras with Tensorflow, and trained our network with a batch size of 64 by gradient descent [7][8].

The architecture of VGG16 model is shown in the **Figure** below.

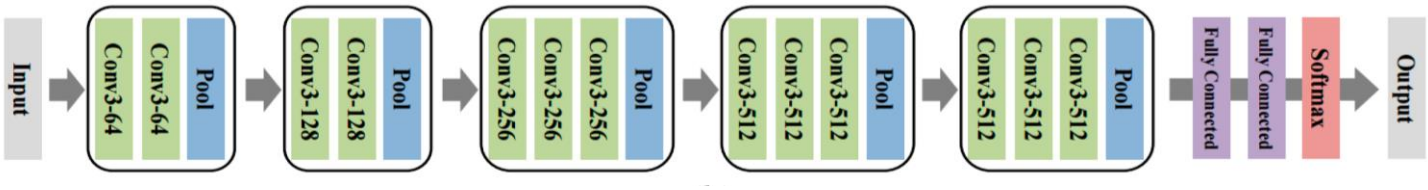


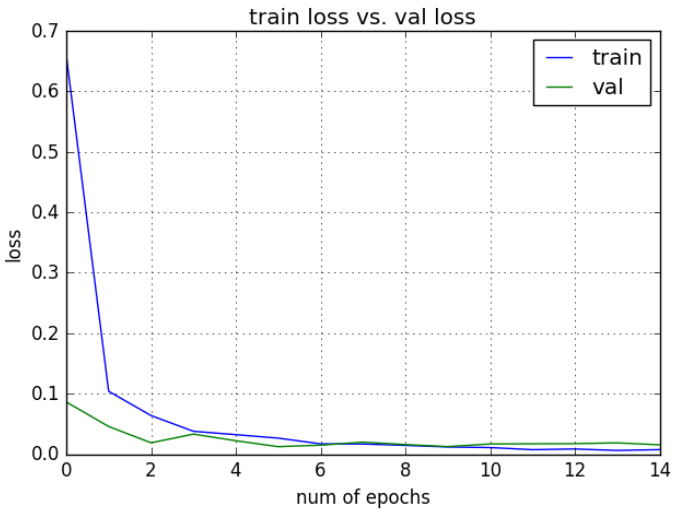
Figure-4 *– VGG16 Architecture*

We used the Stochastic gradient descent optimizer (SGD) from Keras with a value of 0.001, and we used the Early Stopping method to stop training when a validation loss quantity has stopped improving.

Firstly, we modified the VGG16 architecture by deleting the last Fully connected layer, because of the lack of GPU memory. But after having a high GPU performance, we used the Original VGG16 with the Original Architecture.

But we faced a problem of overfitting, we reached 99.95% Training Accuracy, and 99,98% in Validation accuracy, with 15 epochs, and Train loss of 0.0074 and 0.0154 in Validation Loss, but 92% in test Accuracy.

The training graphs are shown in the ***Figure-5,*** and the confusion matrix is shown in ***Figure-6.*** the testing result was not good because of overfitting.



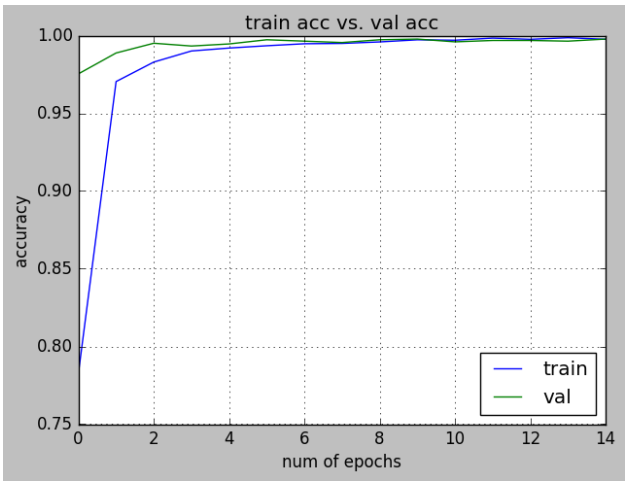
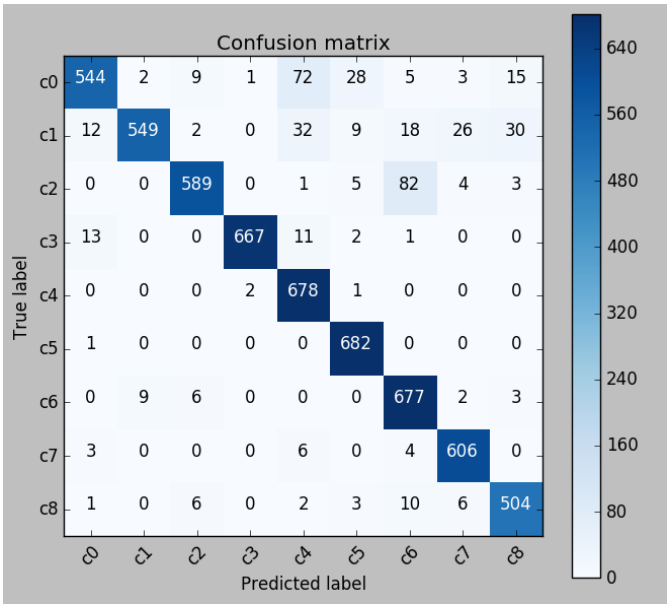


Figure-5 *- Training Graphs of VGG16*



	precision	recall	f1-score	support
c2	0.95	0.80	0.87	679
c8	0.98	0.81	0.89	678
c6	0.96	0.86	0.91	684
c0	1.00	0.96	0.98	694
c4	0.85	1.00	0.91	681
c3	0.93	1.00	0.97	683
c1	0.85	0.97	0.91	697
c7	0.94	0.98	0.96	619
c5	0.91	0.95	0.93	532
avg / total	0.93	0.92	0.92	5947

Figure-6 *- Confusion Matrix and the Classification report of VGG16*

The weak results are because of overfitting. We started searching about how to prevent it, according to Andrew NG's course in Coursera [9], he mentioned that instead of using the Early Stopping method, the best way is to reduce the learning rate when there is no improvement in the validation loss, so we used the ReduceLROnPlateau method in keras Callbacks [10], to reduce the learning rate by a factor of 0.1, and the minimum learning rate to be reached is 0.0001.

The VGG16 model was trained with Cross Validation of 5 folds, with 20 Epochs, we got a Test Accuracy of 92% and Test loss of 0,7. Then Finally we changed the Dropout Values and we trained the Model with 20 Epochs. The results are shown in the Figures below.

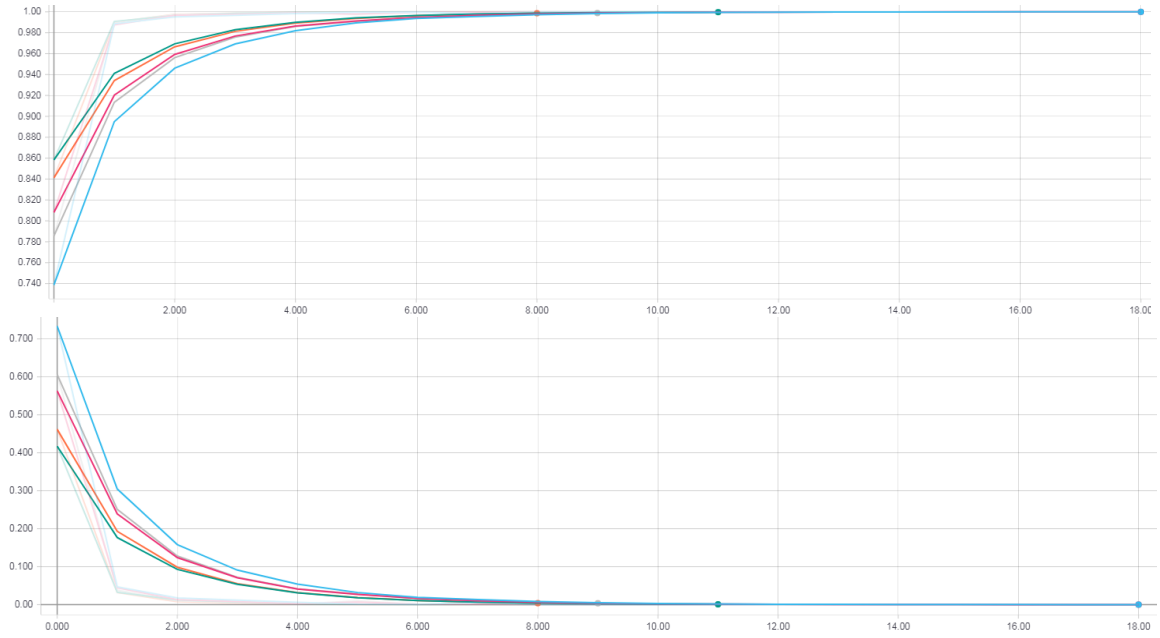


Figure-6 *- Training Accuracy and Training Loss of VGG16 with Cross Validation of 5 Folds*

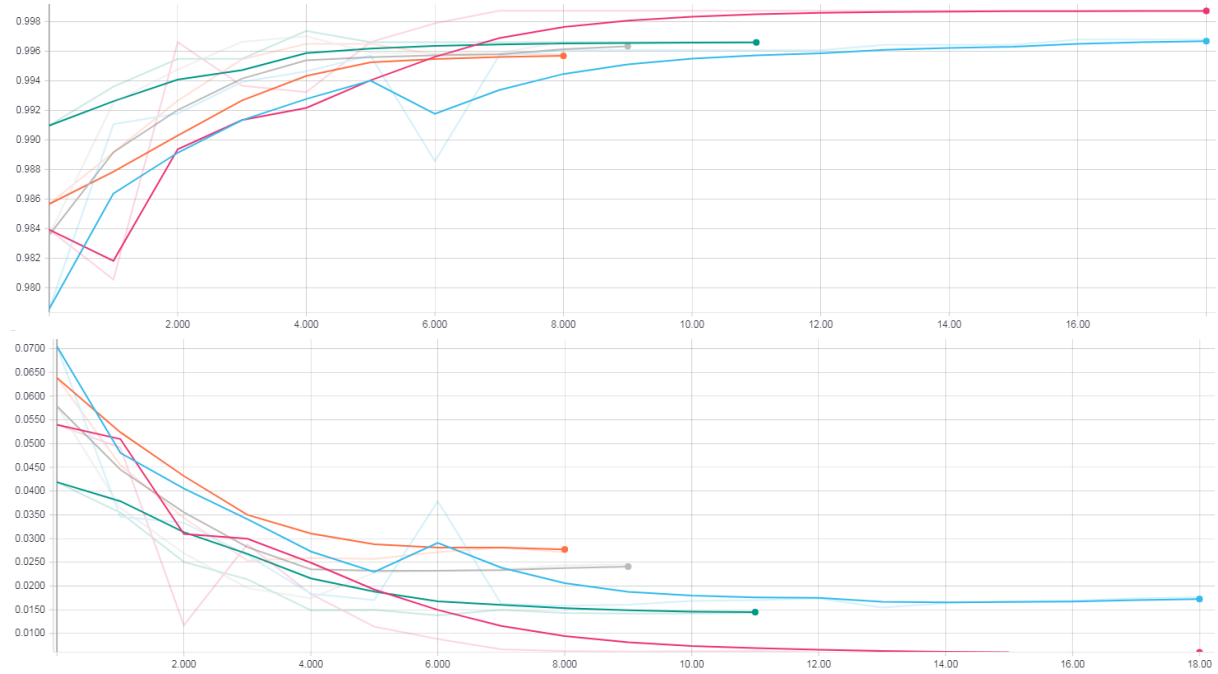


Figure-7 *- Validation Accuracy and Training Loss of VGG16 with Cross Validation of 5 Folds*

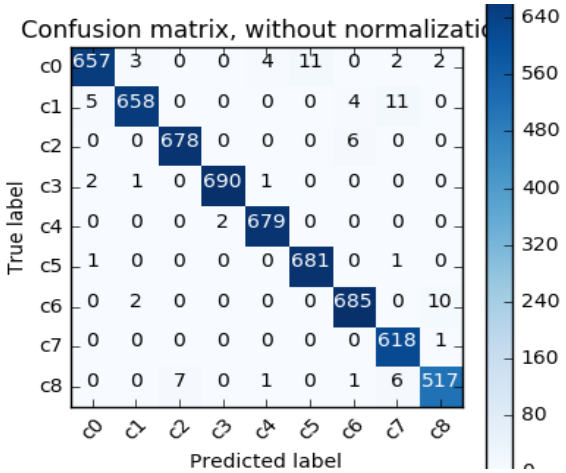
As it is shown in the figures above that some of folds stopped in an Earlier epochs, because of reducing learning rate with a factor of 0.1 ,when there is no improvement in the validation Accuracy,

NewLearningRate = CurrentLearnignRate * 0.1

If the reduced learning rate value reaches the minimum value of 0,00001, an Early stopping occurs, the implementation of the Reduce of LR is shown below:

```
1. reduce_lr = ReduceLROnPlateau(monitor = 'val_acc', patience = 4, verbose = 1, factor = 0.1,
                cooldown = 10, min_lr = 0.00001factor) )
2. #  reduce learning rate(new_lr = lr * 0.1
```

The test Loss is 0.14280589 and test accuracy is 96.22%. The test result of VGG16 is shown below:



	precision	recall	f1-score	support
c0	0.99	0.97	0.98	679
c1	0.99	0.95	0.97	678
c2	0.99	0.99	0.99	684
c3	1.00	0.99	1.00	694
c4	0.99	1.00	0.99	681
c5	0.98	1.00	0.99	683
c6	0.98	0.98	0.98	697
c7	0.95	1.00	0.97	619
c8	0.98	0.97	0.97	532
avg / total	0.98	0.98	0.98	5947

According to the Results we got, we conclude that the Vgg16 model has been well trained, and by using the Reducing Learning Rate method and Dropout, we prevented our training from overfitting.

3.6 Residual Network 50

Deep convolutional neural networks have led to a series of breakthroughs for image classification. Many other visual recognition tasks have also greatly benefited from very deep models. So, over the years there is a trend to go deeper, to solve more complex tasks and to also increase /improve the classification/recognition accuracy. But, as we go deeper, the training of neural network becomes difficult and the accuracy starts saturating and then degrades also. Residual Learning tries to solve both these problems.

What is Residual Learning?

In general, in a deep convolutional neural network, several layers are stacked and are trained to the task at hand. The network learns several low/mid/high level features at the end of its layers. In residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of feature learned from input of that layer. ResNet does this using shortcut connections (directly connecting input of nth layer to some (n+x)th layer. It has proved that training this form of networks is easier than training simple deep convolutional neural networks and also the problem of degrading accuracy is resolved. ResNet50 is a 50 layer Residual Network. There are other variants like ResNet101 and ResNet152 also. [x]

The architecture of ResNet50 is shown in the **Figure-9**.

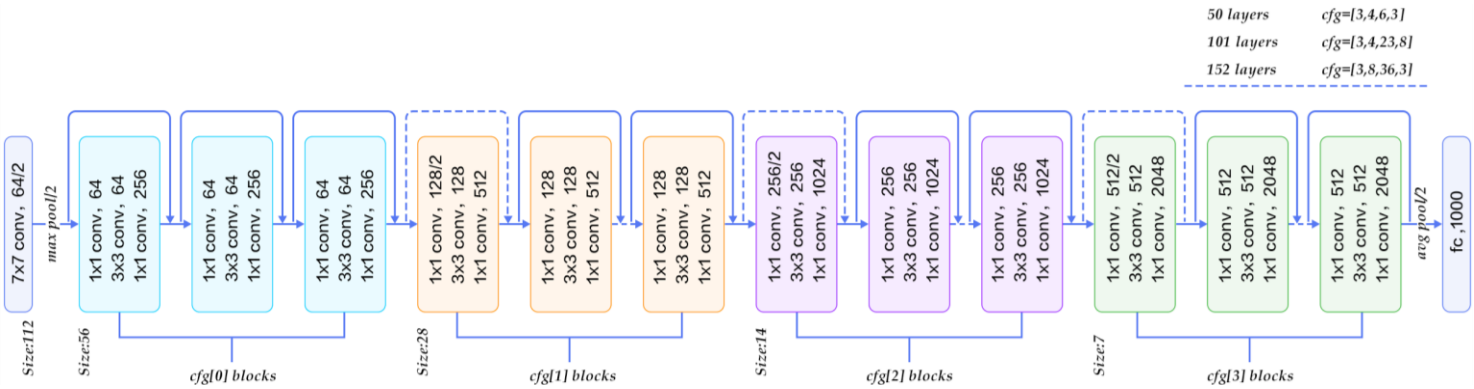


Figure-9 *– The ResNet50 Architecture*

We created the ResNet50 base-model, then we added the top layers to it as its shown in the **Figure-10.**

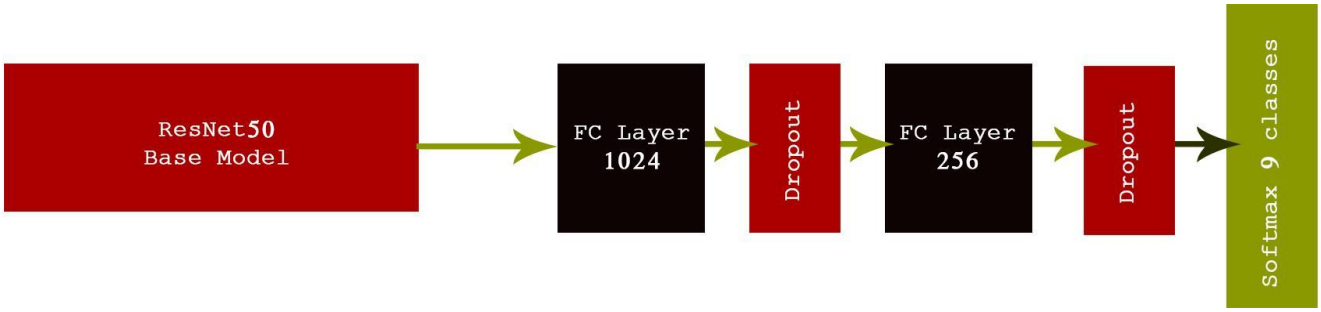


Figure-10 *– The ResNet50 with our Top layers*

First, we freeze all the base model layers, and train just the top model for few epochs, then we saved the weights and train the model again with cross validation of 5 folds, but we got a maximum Training and Val accuracy of 98% and the F1-Score of the test was just 45%. Then we tried another method which is the following:

- Using the same top layers.
- Training the ResNet50 model without freezing any layer.
- changing the dropout values and using the Adam Optimizer instead of SGD.
- we trained the model with cross validation of 5 folds as before, with 15 epochs.

The results are shown in the figures below.

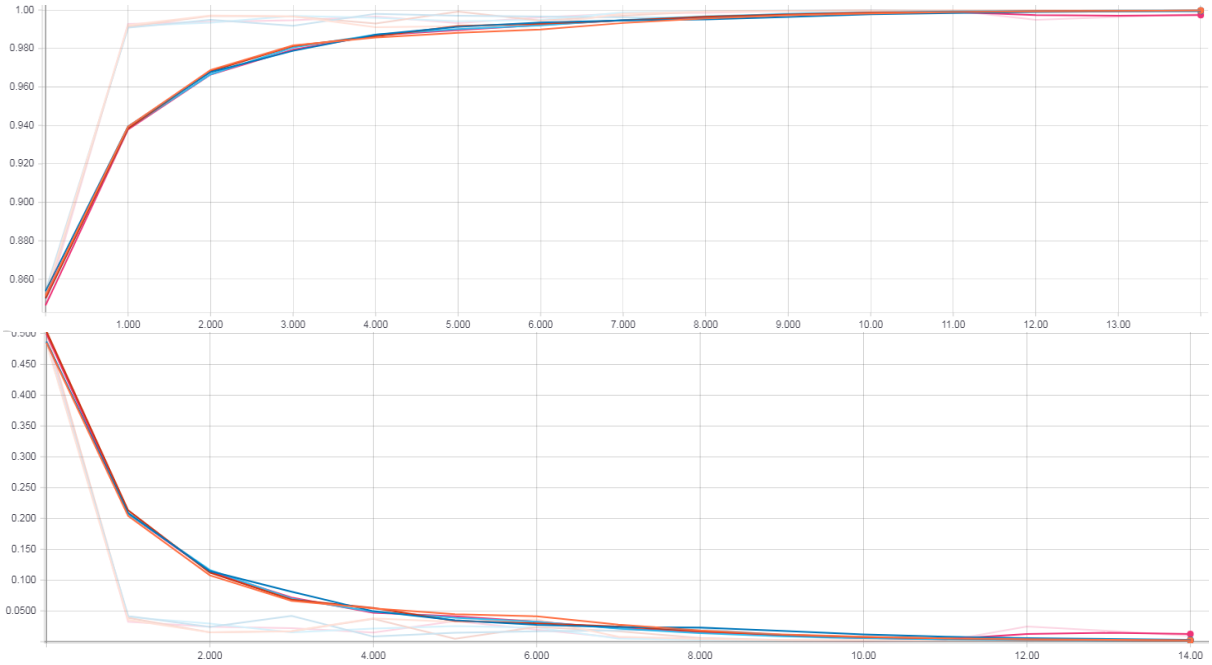


Figure-11 *- Training Accuracy and Training Loss of ResNet50 with Cross Validation of 5 Folds*

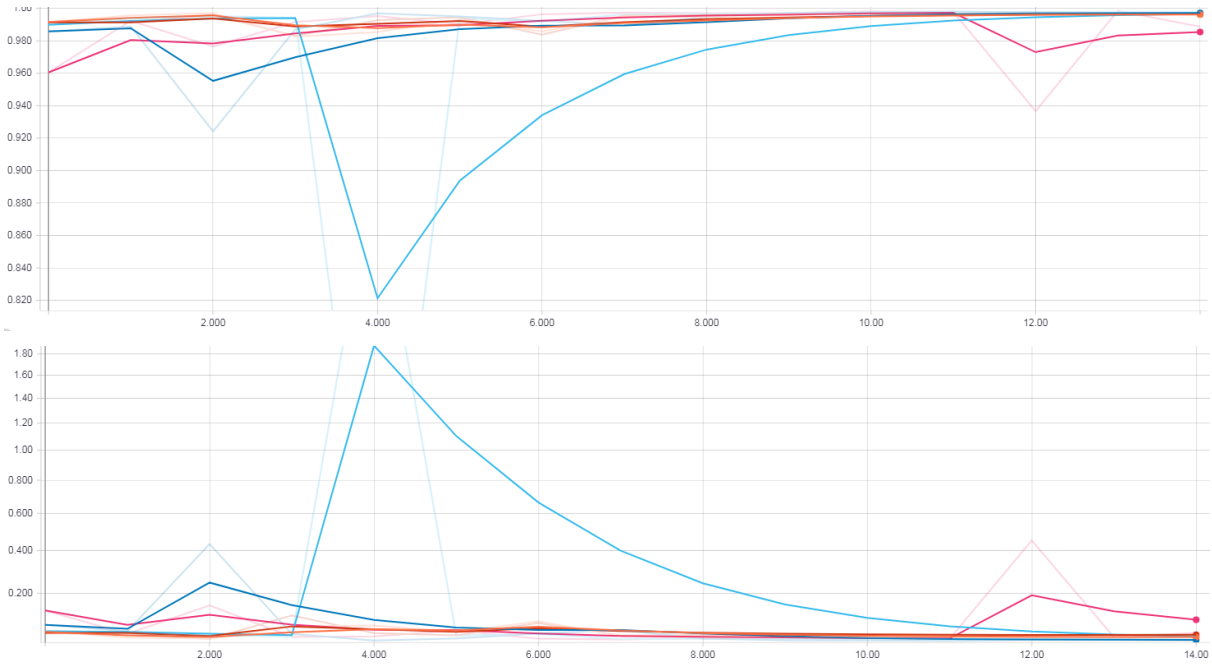
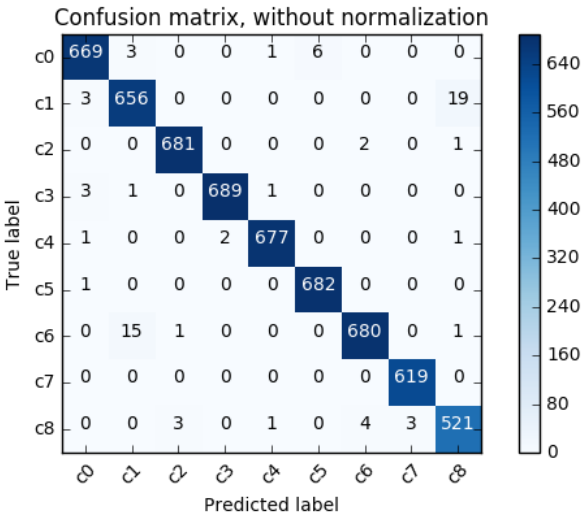


Figure-12 *– Validation Accuracy and Validation Loss of ResNet50 with Cross Validation of 5 Folds*



	precision	recall	f1-score	support
c0	0.99	0.99	0.99	679
c1	0.97	0.97	0.97	678
c2	0.99	1.00	0.99	684
c3	1.00	0.99	0.99	694
c4	1.00	0.99	0.99	681
c5	0.99	1.00	0.99	683
c6	0.99	0.98	0.98	697
c7	1.00	1.00	1.00	619
c8	0.96	0.98	0.97	532
avg / total	0.99	0.99	0.99	5947

Figure-13 *- Test Confusion Matrix and Classification Report*

According to final results, the ResNet50 model has been well trained, and weak results we got while using the Transfer Learning and the Fine Tuning, was because of the base model was not well trained in our dataset, so when training all the layers, even it takes time with our Dataset, with an input of 229 x 229 x3, but it gives a good accuracy in training and in the test with a F1-Score of 99%.

3.7 Inception V3

The idea of inspiration comes from the idea that we need to decide which type of convolution we want to make at each layer: a 3×3? Or a 5×5? And this can go on for a while. as it is introduced in Coursera's course, as in VGG16 we are using just 3x3, but why we use all of them and let the model decide?

So, we are doing each convolution in parallel and concatenating the resulting feature maps before going to the next layer. Now let's say the next layer is also an Inception module. Then each of the convolution's feature maps will be passes through the mixture of convolutions of the current layer. The idea is that we don't need to know ahead of time if it was better to do, for example, a 3×3 then a 5×5. Instead, just we do all the convolutions and let the model pick what's best. Additionally, this architecture allows the model to recover both local feature via smaller convolutions and high abstracted features with larger convolutions.

The architecture of Inception model is shown in Figure-14

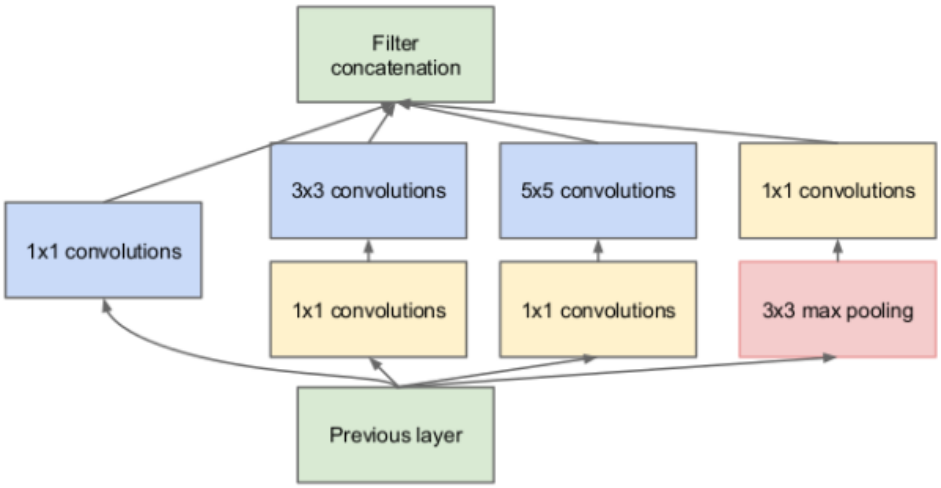


Figure-14 *– Inception Model Architecture*

The authors say that the use of 1×1 convolutions reduces the dimensionality of the input to large convolutions, thus keeping the computations reasonable.

To understand what they are talking about, let's first see why we are in some computational trouble without the reductions.

Let's say we use, we the authors call, the naive implementation of an Inception module.

Figure-15 shows an Inception module that's like the one in *Figure-14*, but it doesn't have the additional 1×1 convolutional layers before the large convolutions (3×3 and 5×5 convolutions are

considered large).

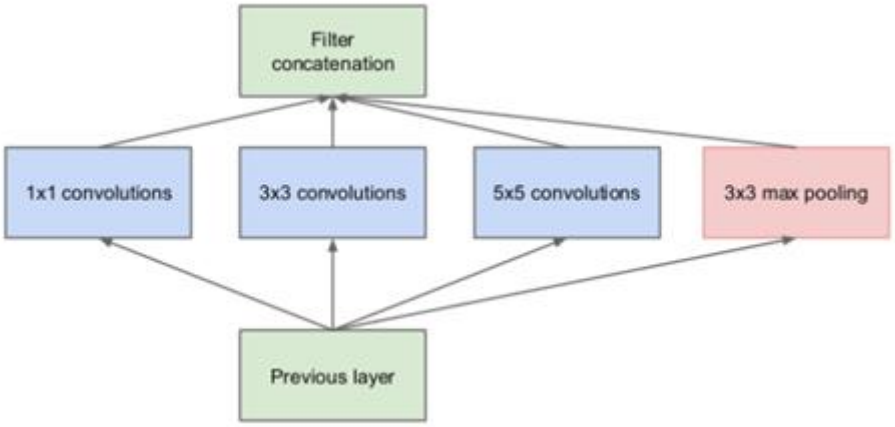


Figure-15 *– Inception Model Architecture*

The figure below shows the added top layers:

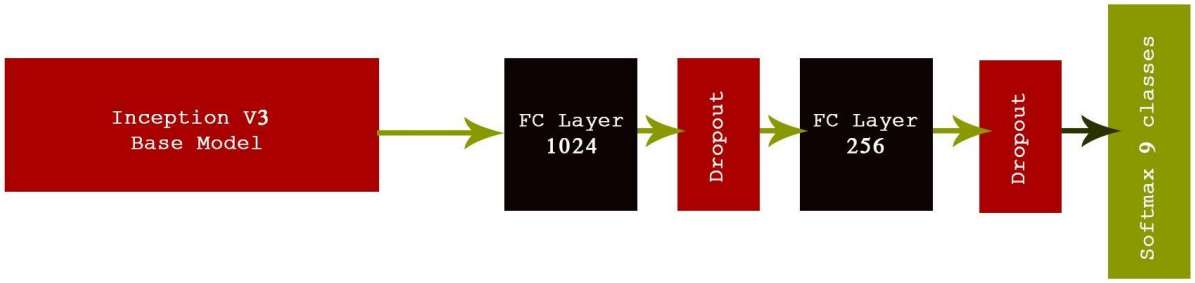


Figure-16 *– Inception Model Architecture with Top Layers*

First, we create the base pre-trained model of InceptionV3 model from Keras, then we add our top model to the base model, then, we train only the top layers (which were randomly initialized) by freezing all convolutional InceptionV3 layers, and train just the top model with 25 epochs, then we saved the weights.

At this point, the top layers are well trained and we can start fine-tuning, we chose to train the top 2 inception blocks, we freeze the first 249 layers and unfreeze the rest, then we trained the model using the Cross Validation as we did before with 5 folds.

But we faced the same problem as with Residual network, then we used the same method to train the Inception Model again, without freezing any layer, then the we got the following results.

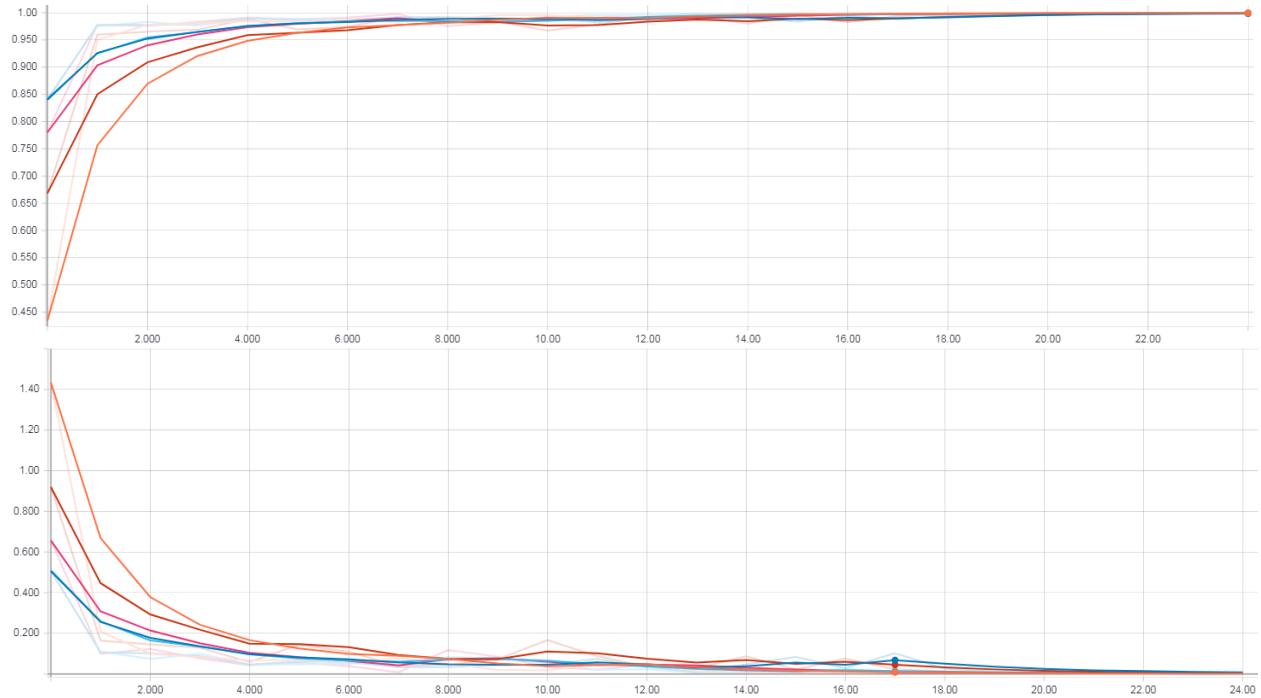


Figure-17 *- Training Accuracy and Training Loss of InceptionV3 with Cross Validation of 5 Folds*

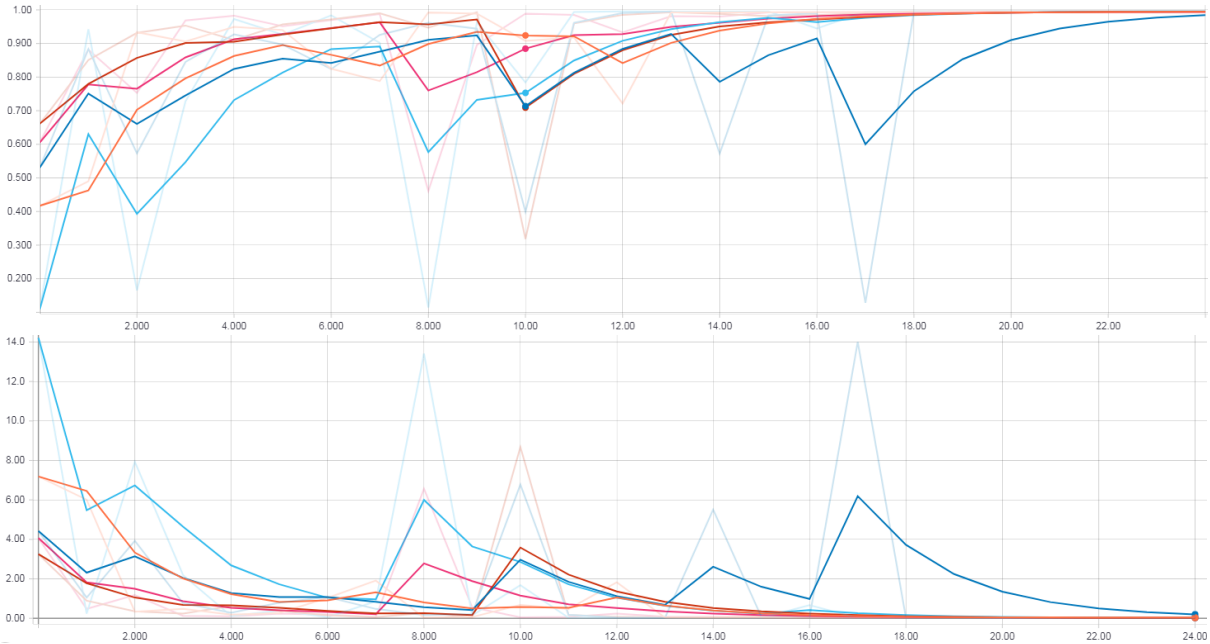
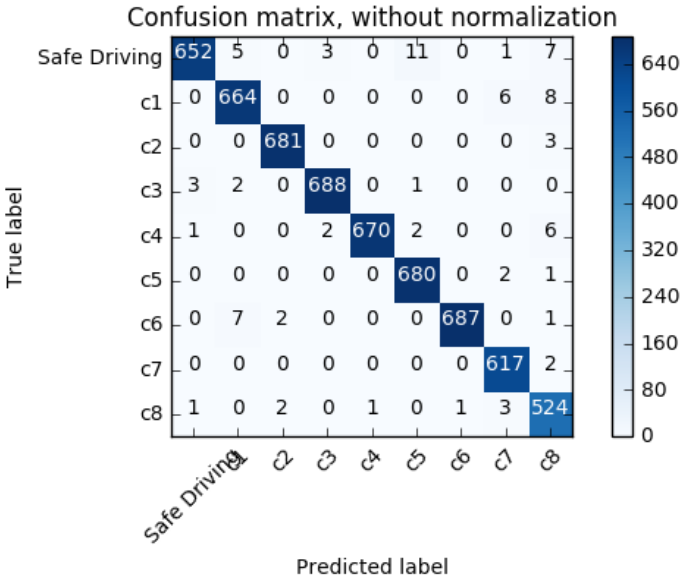


Figure-18 *– Validation Accuracy and Validation Loss of InceptionV3 with Cross Validation of 5 Folds*



	precision	recall	f1-score	support
Safe Driving	0.99	0.96	0.98	679
c1	0.98	0.98	0.98	678
c2	0.99	1.00	0.99	684
c3	0.99	0.99	0.99	694
c4	1.00	0.98	0.99	681
c5	0.98	1.00	0.99	683
c6	1.00	0.99	0.99	697
c7	0.98	1.00	0.99	619
c8	0.95	0.98	0.97	532
avg / total	0.99	0.99	0.99	5947

Figure-19 *- Test Confusion Matrix and Classification Report*

3.8 Xception

Xception was proposed by François Chollet himself, the creator and chief maintainer of the Keras library.

Xception is an extension of the Inception architecture which replaces the standard Inception modules with depthwise separable convolutions.

What is a Depthwise Separable Convolution?

A depthwise separable convolution, commonly called “separable convolution” in deep learning frameworks such as TensorFlow and Keras, consists in a sequence of two operations:

1. A depthwise convolution, i.e. a spatial convolution performed independently over each channel of an input, and
2. a pointwise convolution, i.e. a 1x1 convolution, projecting the channels output by the depthwise convolution onto a new channel space.

Why Separable Convolutions ?

The author postulates that “the fundamental hypothesis behind Inception is that cross-channel correlations and spatial correlations are sufficiently decoupled that it is preferable not to map them jointly”. This decoupling is implemented in the Inception module by a 1x1 convolution followed by a 3x3 convolution (see fig. 1 and 2).

Furthermore, the author makes the following proposition: the Inception module lies in a discrete spectrum of formulations. At one extreme of this spectrum we find the regular convolution, which spans the entire channel space (there is a single segment). The reformulation of the simplified Inception module in Figure 20 uses three segments. Figure 21 shows a module that uses a separate convolution for each channel, which is almost a depthwise separable convolution (more on this below).

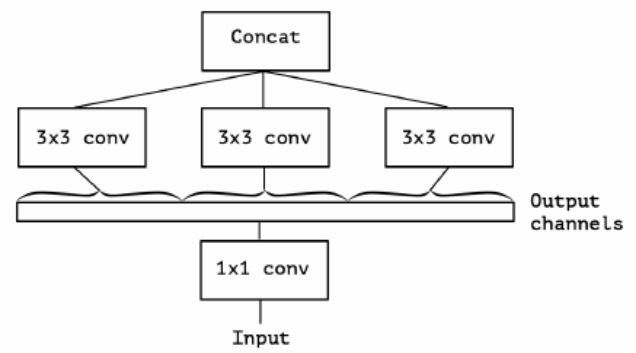


Figure-20 *– A strictly equivalent reformulation of the simplified Inception module*

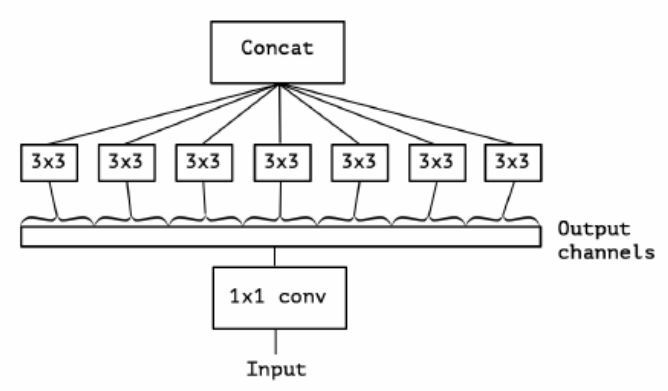


Figure-21 *– An extreme version of Inception Module with one spatial convolution per output chanel of the 1x1 convolution*

There is two minor differences between a depthwise separable convolution and the module shown in Figure 21 : (1) the order of the two operations, and (2) the presence or absence of a non-linearity between those operations. The author argues that the first does not matter. The second difference is important, and the absence of non-linearity improves accuracy.

The Xception Architecture

In short, the Xception architecture is a linear stack of depthwise separable convolution layers with residual connections. “Xception” means “Extreme Inception”, as this new model uses depthwise separable convolutions, which are at one extreme of the spectrum described above.

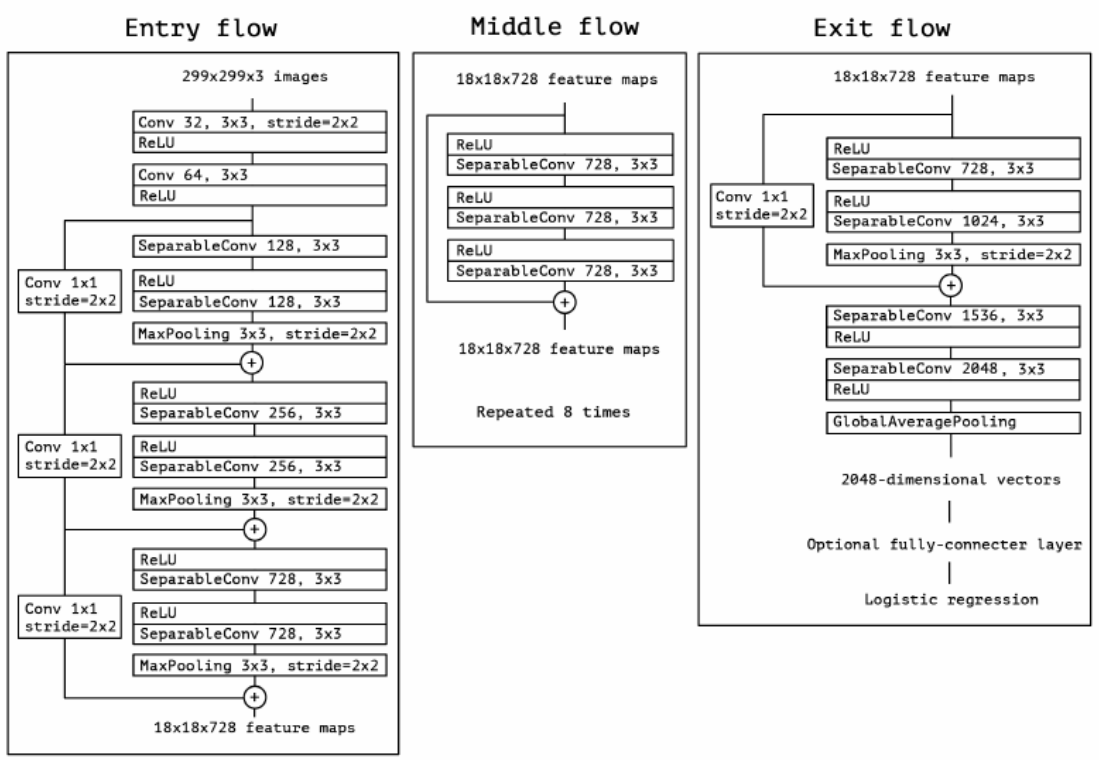


Figure-22 *– The Xception Model Architecture*

Our Xception Model is shown in the figure below.

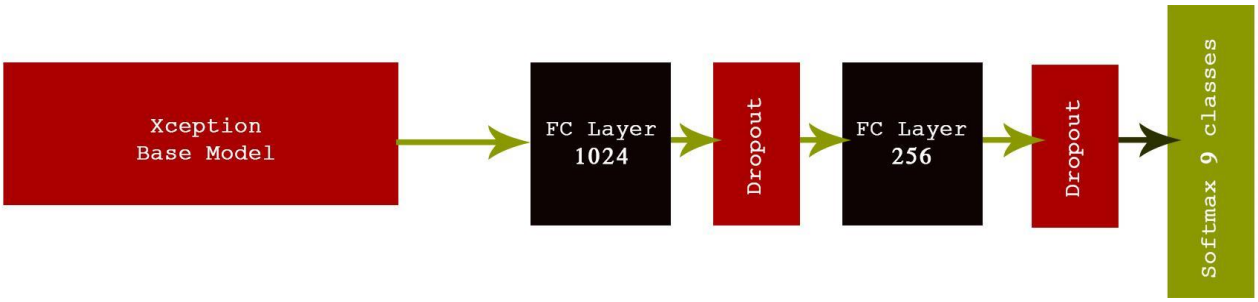


Figure-23 *– The Xception Model Architecture with the Top layers*

We trained our model the same way as we did with the Inception model, with cross validation of 5 folds with 25 epochs, the first time, we Freeze all the layers on the Xception base model, then we saved the weights.

Now, we want to train the last Two blocks of the Xception model, it should be 172 layers before this number will used the pre-trained weights, layers above and including this number will be re-trained based on our data with cross validation of 5 folds.

But we got weak results, an Accuracy of 84%, then we trained the whole model without any

freeze in the layers, with cross validation, with a batch-size of 16, we couldn't use a higher batch size value with GTX 1070 of 8Gb in memory, it takes 2 days to train all the layers.

The results are shown in the Figures below.

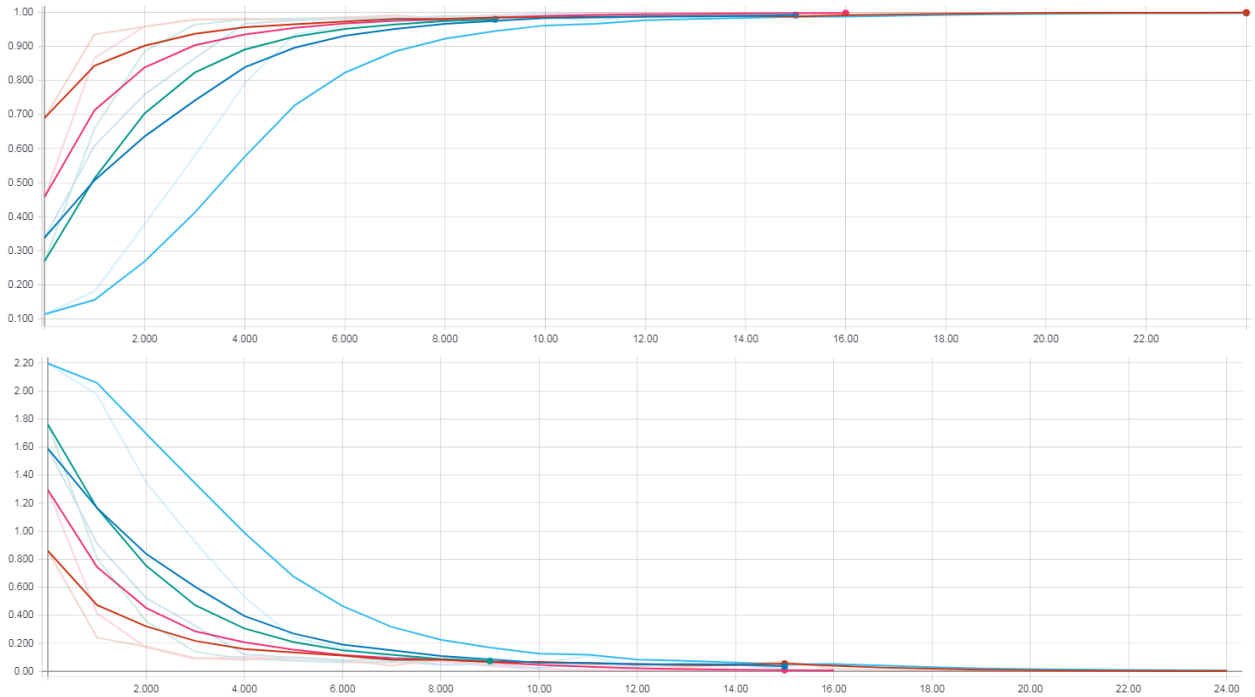


Figure-24 *- Training Accuracy and Training Loss of Xception with Cross Validation of 5 Folds*

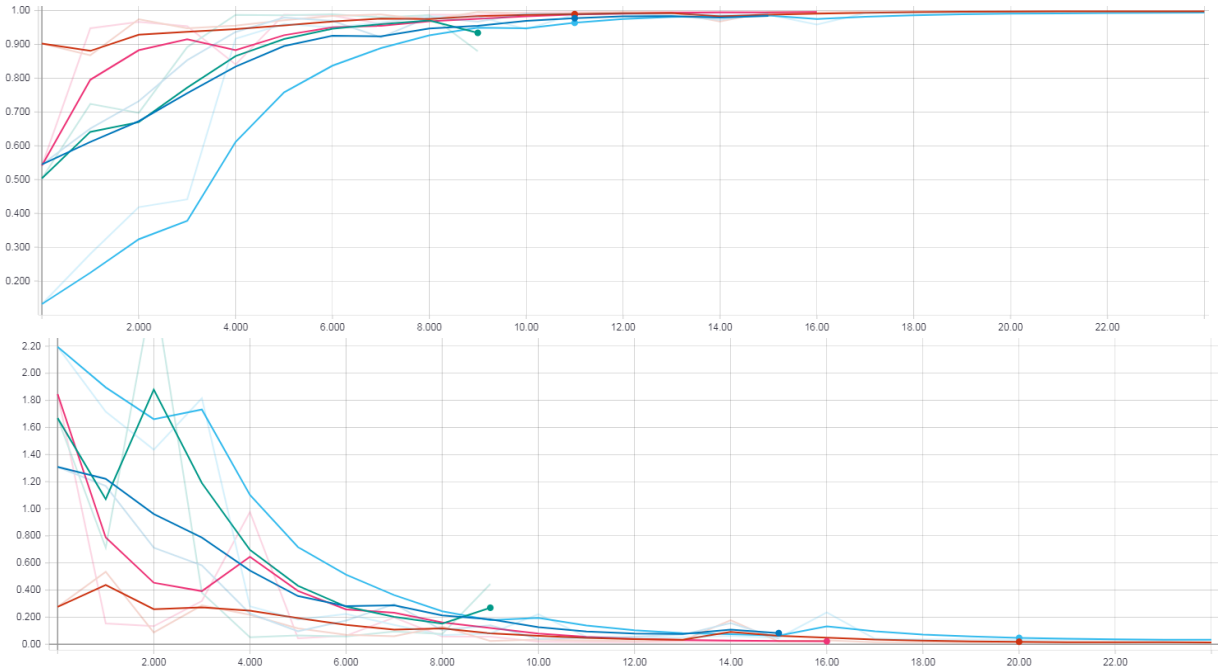
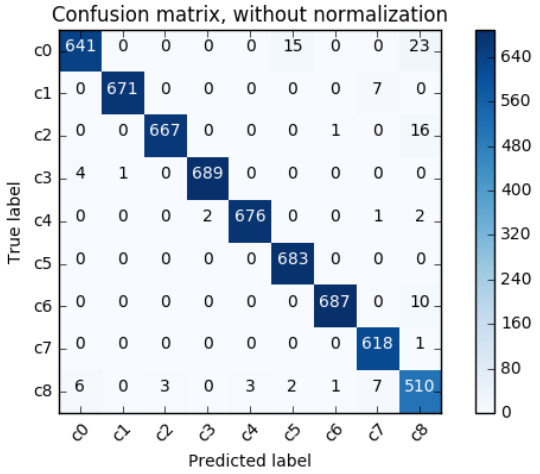


Figure-25 *- Validation Accuracy and Validation Loss of Xception with Cross Validation of 5 Folds*



	precision	recall	f1-score	support
c0	0.98	0.94	0.96	679
c1	1.00	0.99	0.99	678
c2	1.00	0.98	0.99	684
c3	1.00	0.99	0.99	694
c4	1.00	0.99	0.99	681
c5	0.98	1.00	0.99	683
c6	1.00	0.99	0.99	697
c7	0.98	1.00	0.99	619
c8	0.91	0.96	0.93	532
avg / total	0.98	0.98	0.98	5947

Figure-26 *- Test Confusion Matrix and Classification Report*

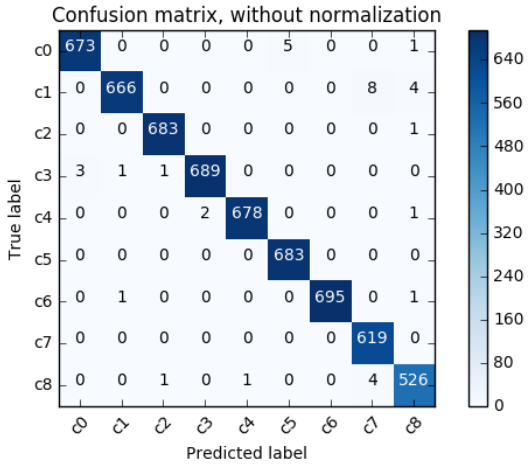
3.9 Network Ensemble

An ensemble is the idea of combining (averaging) the results of several classifiers. By averaging different models, the variance of the trained model can be reduced and a lower loss can be achieved.

The Results of VGG16, Inception, Xception, ResNet50:

Model	F1-Score	Top-1 Accuracy	Top-2 Accuracy	Loss
VGG16 + CrossValidation	99%	96.2226%	98.8833%	0.14280
InceptionV3 + CrossValidation	99%	95.4531%	97.9350%	0.20000
ResNet50 + CrossValidation	98%	96.6773%	98.7186%	0.12851
Xception + CrossValidation	98%	94.1785%	96.8790%	0.30539
ResNet50 + VGG16	99%	96,4499%	98,8008%	0.13565
VGG16 + inceptionV3+ResNet50+Xception	99%	96.3080%	98.1039%	0.23126

And the results are shown in figure below



	precision	recall	f1-score	support
c0	1.00	0.98	0.99	679
c1	0.99	0.98	0.99	678
c2	1.00	1.00	1.00	684
c3	1.00	0.99	0.99	694
c4	1.00	1.00	1.00	681
c5	0.99	1.00	0.99	683
c6	1.00	1.00	1.00	697
c7	0.98	1.00	0.99	619
c8	0.99	0.99	0.99	532
avg / total	0.99	0.99	0.99	5947

Figure-27 *- Test Confusion Matrix and Classification Report*

3.10 Results

In this section, we will mention about some results of other candidates in Kaggle challenge "The State Farm Distracted Driver Detection" .

#	△pub	Team Name	Kernel	Team Members	Score	Entries	Last
1	▲1	jacobkie			0.08739	100	1y
2	▲2	Z_B_C			0.09057	106	1y
3	—	BR BRAZIL POWER BR			0.09058	196	1y
4	▼3	MakeAmericaGreatAgain			0.10065	198	1y
5	▲1	DZS			0.12144	302	1y
6	▼1	TitanX && 1080			0.12673	185	1y
7	▲2	Vinh Nguyen			0.13672	81	1y

Table 1 *– Leaderboards from Kaggle platform*

The Results bellow are taken from some published papers.

	Network	Loss	Accuracy	LB Score
Yehya Abouelnaga, Hesham M. Eraqi, and Mohamed N. Moustafa [19]	AlexNet	0.3909	93.65%	-
	InceptionV3	0.2654	95.17%	-
	Majority Voting Ensemble	0.1661	95.77%	-
Samuel Colbran, Kaiqi Cen, Danni Luo [20]	AlexNet	-	-	0,69126
	CaffeNet + Vgg2	-	-	0.44084
	CaffeNet + Vgg1 + Vgg2 + Vgg3 +Vgg4	-	-	0.34354
	Vgg1 + Vgg2 + Vgg3 + Vgg4	-	-	0.32706
Murtadha D Hssayeni, Sagar Saxena, Raymond Ptucha, Andreas Savakis [21]	AlexNet	-	72.60%	-
	VGG-16	-	82.50%	-
	ResNet -152	-	85.00%	-
CESAR HIERSEMANN [22]	VGG16 + Cross Validation	0.60	80.00%	-

Mateusz Buda and Jokull Johannsson [23]	VGG16	-	82.00%	-
Nishaat Farheen, Mohit Ravi, Lanyin Zhang Instructor: German Creamer [24]	Logistic Regression	1,234	58.80%	-
	Linear Kernel SVM	1,32	54.90%	-
	Random Forest	1,83	49.20%	-
	Gradient Boosting	1,53	57.00%	-

Table 2 *– Results of candidates*

We Could not submit our solution to Kaggle platform, because of the deadline of the competition, but according to the results above we see that our solution is better than the results in the Table 2, the maximum accuracy they achieved was 95.77, and a loss of 0.1661.

According to the loss we got, our solution will be approximately in the top first 8 results.

3.11 Conclusion

Distracted driving is a major problem leading to a striking number of accidents worldwide. Besides, its detection is an important system component to reduce the number of accidents.

After Exploring Various CNN models, we achieved good results with all the network, and the best performing results were achieved by calculating the average of our models (VGG16, ResNet50, Inception, and Xception).

We really leant from the problem we faced, also how to train these models.

In our project, VGG16 was the easiest model in training, but it is hard for Inception, Xception and ResNet50, but the best way we found is to train all the layers instead of freezing some layers and blocks, that what gives the chance to the model to be well trained, and also the best techniques to be used to prevent the overfitting are:

- Use high learning rate values. Then, Reducing the learning rate while no improvement occurs in the validation accuracy. Instead of the Early Stopping.
- Add more data
- Use data augmentation
- Use architectures that generalize well
- Reduce architecture complexity.
- Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

3.12 Visualizing the Intermediate layer output of CNN

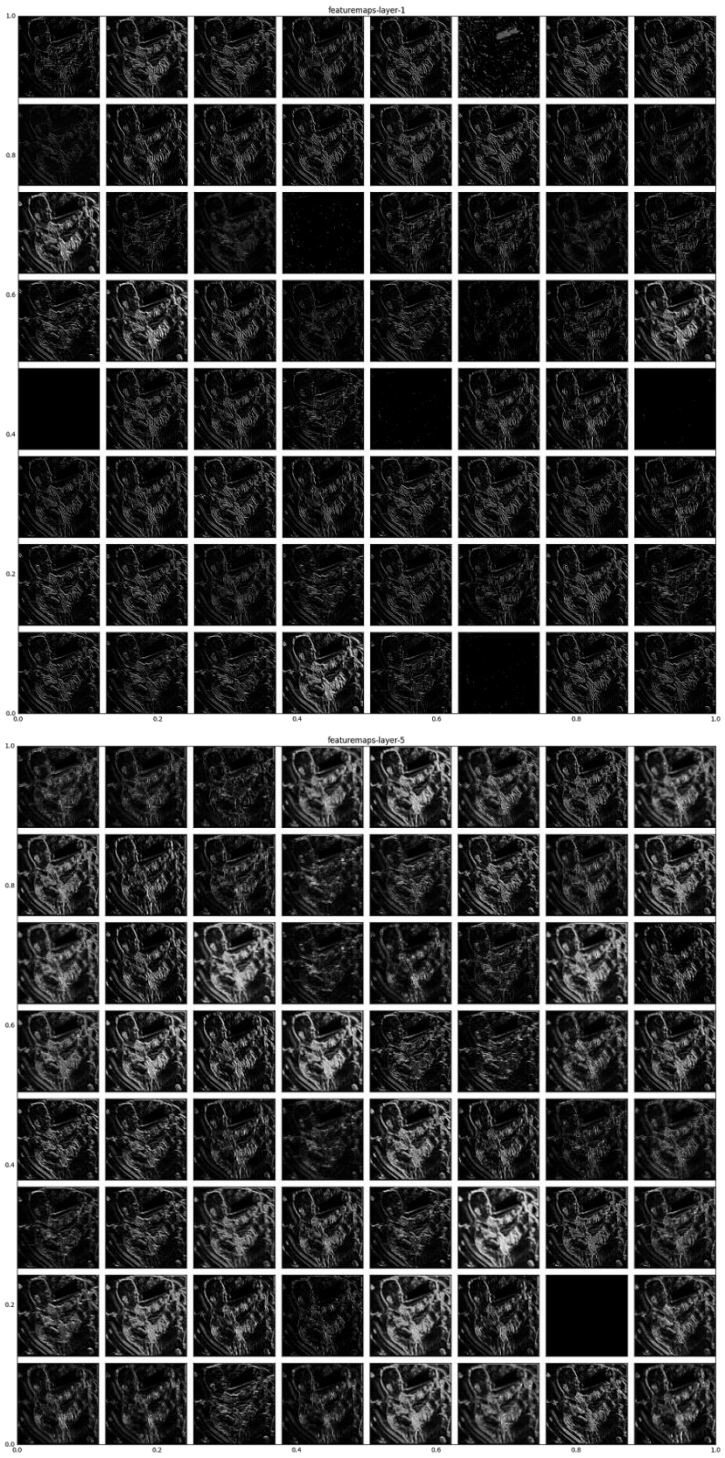
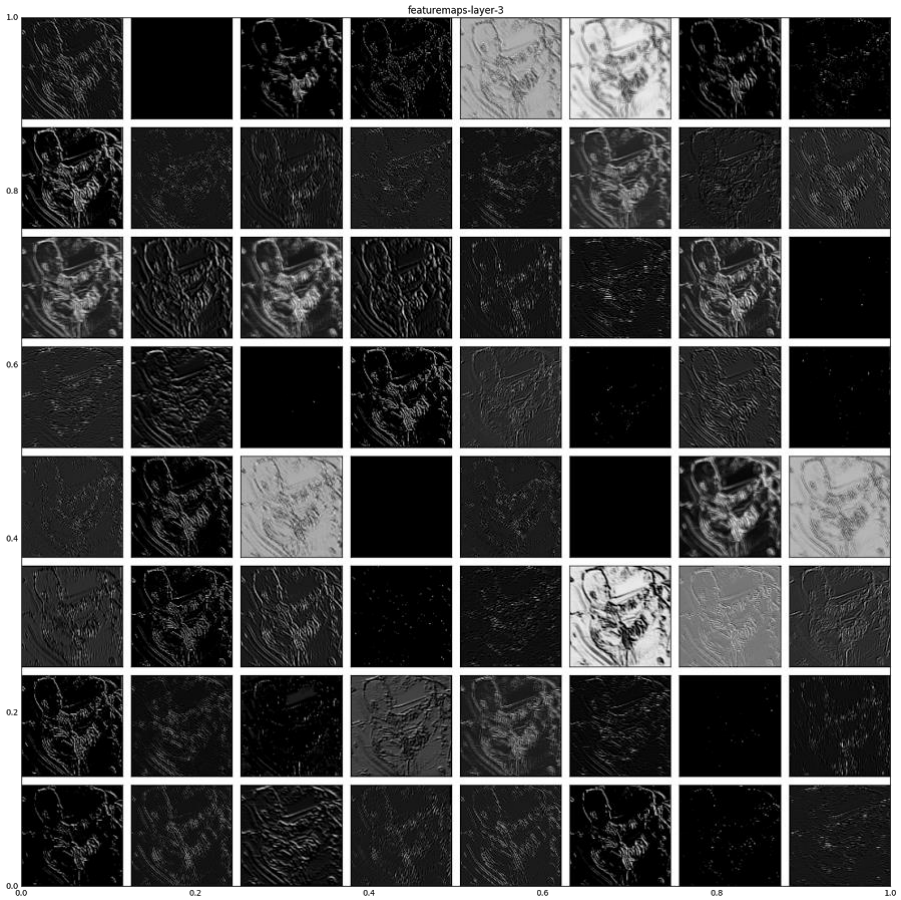


Figure-28 *– Output of VGG16, Layer 2 and 3*



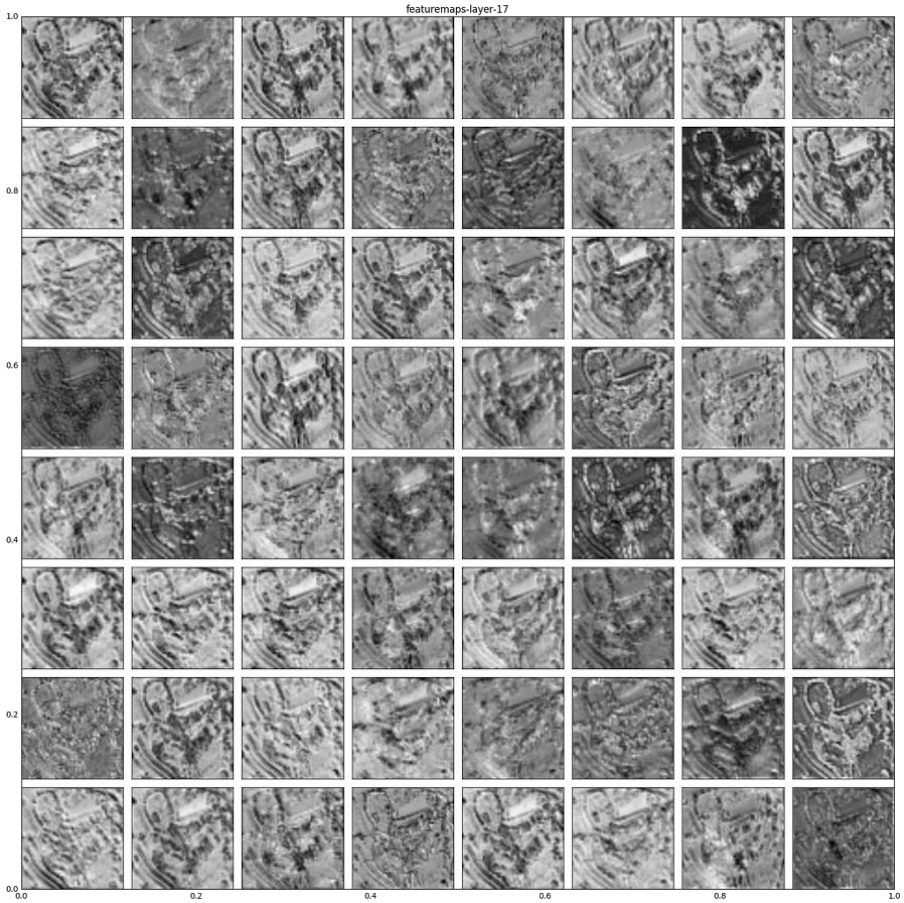


Figure-29 *– Output of InceptionV3, Layer 3 and 17*

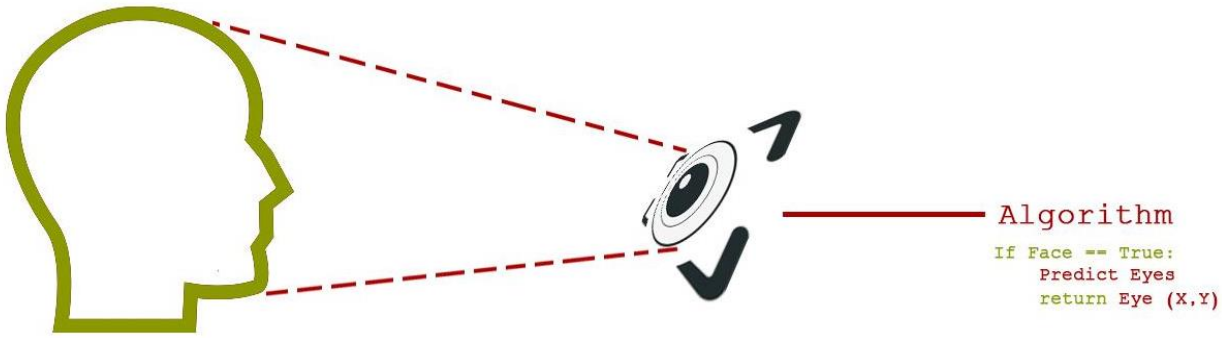
4.Drowsiness Detection

4.1 Introduction

In recent years, there has been growing interest in intelligent vehicles. A notable initiative on intelligent vehicles was created by the U.S. Department of Transportation with the mission of prevention of highway crashes [12]. The ongoing intelligent vehicle research will revolutionize the way vehicles and drivers interact in the future.

The US National Highway Traffic Safety Administration estimates that in the US alone approximately 100,000 crashes each year are caused primarily by driver drowsiness or fatigue [13]. Moreover regardless of the state's reporting format drowsiness maybe underreported due to a lack of firm evidence upon which to base a police finding [14]. Thus incorporating automatic driver fatigue detection mechanism into vehicles may help prevent many accidents. Here you will find some background on the Fatigue Detection and Prediction technologies and the motivation for our approach.

4.2 Our Solution



Our Solution is about using the camera in the front side of the driver to capture pictures continuously, and then send the frames to the Algorithm for prediction and detection of the driver's eyes landmarks.

Our solution consist of computing the Eye Aspect Ratio (EAR), introduced by Soukupová and Čech in their paper[15].

Unlike traditional image processing methods for computing blinks which typically involve some combination of:

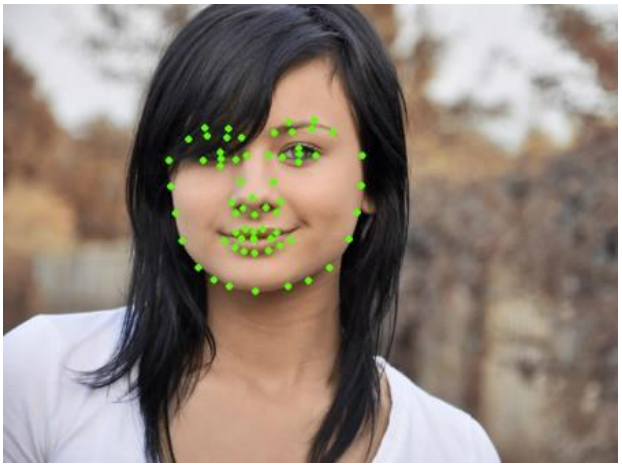
1. Eye localization.
2. Thresholding to find the whites of the eyes.

3. Determining if the “white” region of the eyes disappears for a period of time .
4. The eye aspect ratio is instead a much more elegant solution that involves a very simple calculation based on the ratio of distances between facial landmarks of the eyes.

This method for eye blink detection is fast, efficient, and easy to implement.

4.3 Facial landmarks with dlib

Facial landmarks is a used technique to localize and represent salient regions of the face, such as: Eyes, Eyebrows, Nose, Mouth, Jawline.



What are facial landmarks?

The facial landmarks detection is an application of the shape prediction problem. Given an input image (and normally an ROI that specifies the object of interest), a shape predictor attempts to localize key points of interest along the shape.

In our project, our goal is detecting the Eye on the face using shape prediction methods.

Detecting facial landmarks is therefore a two step process:

- Step 1: Localize the face in the image.
- Step 2: Detect the Eye on it.

We use OpenCV’s built-in Haar cascades [16] to detect the face on the given picture.

We might apply a pre-trained HOG + Linear SVM object detector specifically for the task of face detection.

Or we might even use deep learning-based algorithms for face localization.

In either case, the actual algorithm used to detect the face in the image doesn’t matter. Instead, what’s important is that through some method we obtain the face bounding box (the (x, y)-coordinates of the face in the image).

Given the face region we can then apply Step #2: detecting key facial structures in the face region. In our case we need just the Right eye, and Left eye

The facial landmark detector included in the dlib library is an implementation of the One Millisecond Face Alignment with an Ensemble of Regression Trees paper by Kazemi and Sullivan [17].

This method starts by using:

A training set of labeled facial landmarks on an image. These images are manually labeled, specifying specific (x, y)-coordinates of regions surrounding each facial structure.

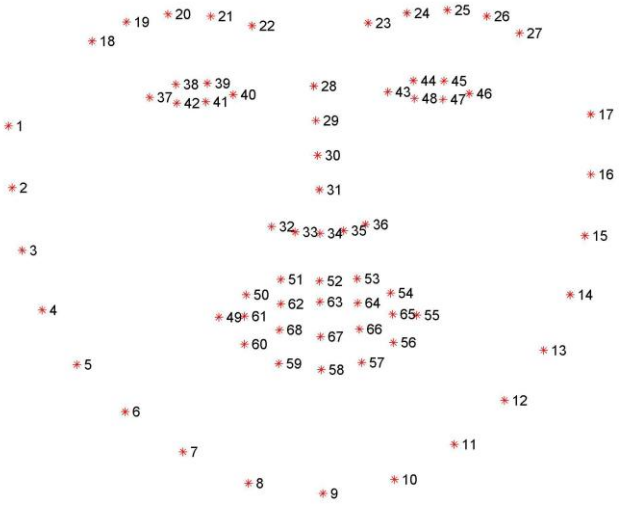
Priors, of more specifically, the probability on distance between pairs of input pixels.

Given this training data, an ensemble of regression trees are trained to estimate the facial landmark positions directly from the pixel intensities themselves (i.e., no “feature extraction” is taking place).

The end result is a facial landmark detector that can be used to detect facial landmarks in real-time with high quality predictions.

For more information and details on this specific technique, be sure to read the paper by Kazemi and Sullivan linked to above, along with the official dlib announcement.

The pre-trained facial landmark detector inside the dlib library is used to estimate the location of 68 (x, y)-coordinates that map to facial structures on the face.



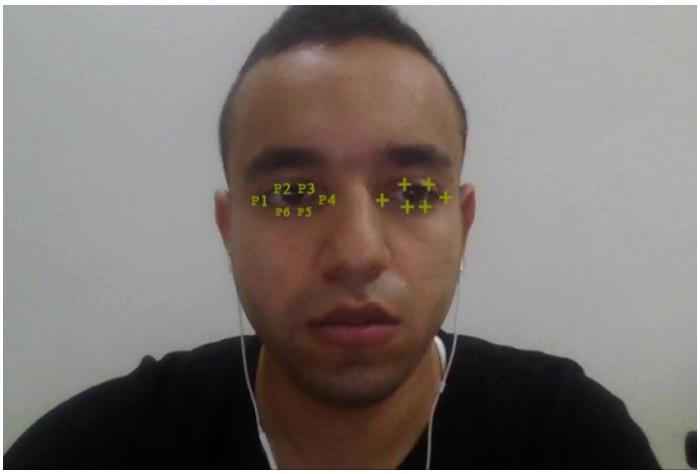
The indexes of the 68 coordinates can be visualized on the right figure:

These annotations are part of the 68 points iBUG 300-W dataset [18] which the dlib facial landmark predictor was trained on.

4.4 Extracting the Facial Landmarks

After launching the camera using the OpenCV, then we capture a frame and run the face detector on it, if a face is detected in the given frame, then we run a landmark predictor, that extracts the left and right eye landmarks.

Each eye is represented by 6 (x, y)-coordinates, starting at the left-corner of the eye. as it is



shown in the Figure below.

Then now, we calculate the Eye Aspect Ratio as follow:

$$\mathrm{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

Then we created a Condition in order to say whether the driver is sleeping or not, which is:

If the eye aspect ratio falls below a certain threshold and then rises above the threshold, then we'll register a "Closed Eye", the EYE Aspect Ratio Threshold (ÊEART), is this threshold value. We default it to a value of 0.23 as this is what has worked best for our applications.

We then have an important constant, Consecutive Frames, this value is set to 10 to indicate that 10 successive frames with an eye aspect ratio less than "EART" must happen in order to sound an alarm (Drowsiness detected).

4.5 Conclusion

In this part of our senior project we presented a system for automatic detection of driver drowsiness from video. Our drowsiness detector hinged on an important computer vision technique, which is the facial landmark detection.

The idea of our solution is use the facial landmarks for the drowsiness detection by calculating the Eye Aspect Ratio by checking if the EAR is low for a successive 10 frames, rather than checking if the “white” region of the eyes disappears for a period of time. Also, the implementation is easy by using OpenCV and Dlib Predictors.

So, if the eyes have been closed for a sufficiently long enough period of time, we can assume the driver is at risk of falling asleep and sound an alarm to grab their attention.

5. Actions

After predicting the driver's state, we would like to take some actions, at the end that's one of the goals of our project.

So we first created a mobile application (running on android), which connect to another device (raspberry pi) via Bluetooth, and via this application the mobile phone would lock the screen in case the driver is detected (by the raspberry pi) to be manipulating the phone, texting or so.

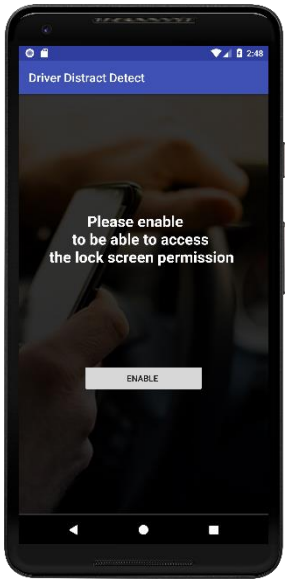
And the second action is to end the calls of the user, while detected talking on the phone. And our last and not least is making some noises, or giving some motivation to the driver, like "Be careful your family is waiting for you", when detected closing his/her eyes caused by fatigue, as discussed above.

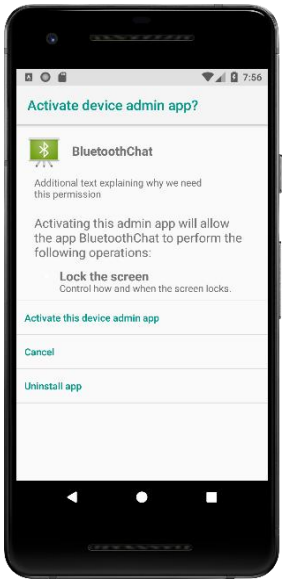
5.1 Android App

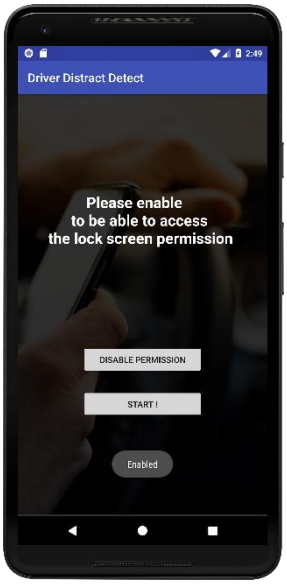
Using Google's sample, BluetoothChatAppAndroid[25], we modified the application as it suits us. We used the chat application, because we would need some signals from the raspberry pi, to know in which state is the driver, from those signals the application would take actions, those signals are strings since it's a chat application it is the only thing send and received.

For example, if the 'pi' sends a string as "1", the application would translate that "1" into locking the screen of the driver's phone.

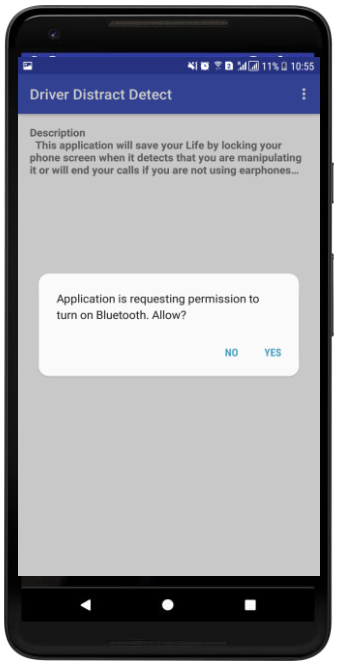
How would the application lock the phone, when launching our application, we first ask the user to get permission to lock the screen, after activation the user can access the application otherwise he can't. [27]

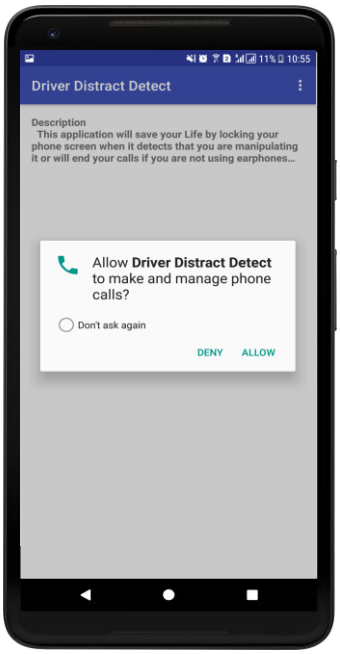






After accessing the permission of lock screen, we can go and allow our application to enable Bluetooth, and manage phone calls.





But we found some difficulties with raspberry's Bluetooth drivers and packages, so we looked for other ways as WiFi-Direct (P2P), and we ended up by using BLE (Bluetooth Low Energy) which fitted our project [29].

5.2 Raspberry pi

Using the raspberry pi, we can deploy our trained model with it and predict the driver states, after detecting the driver state it will send messages to the driver's phone which are connected via Bluetooth.

To detect the driver state, we are using a camera that takes video streaming of the driver (using OpenCV and Keras), first we take a frame, and process it by send it through our neural network model. After process is done we'll get our prediction, from that prediction it will take an action if the driver is not at the 'safe state'.

Firstly, we started by burning the OS (Raspbian) into the SD card, and then we installed all the programs and packages that raspberry needs, to communicate by BLE. Since our server is written in Nodejs using 'bleno' which is a Node.js module for implementing BLE peripherals, we installed that too. The results were good, we had a good and easy connection between the server which is the pi, and our client which is an Android device [30].

5.3Model Deplooyment

Now jumping to the deployment of our model, we started by installing OpenCV, which took a lot of time and many tries. And then installing 'theano' library, 'Keras' API and all what we needed from API's and libraries. We tried our model without a real-time prediction, since our 'pi camera' was defective, the pi was no able to have data from the sensor.

So, we tried the model, and it was predicting fine, but it was very slow. The problem was that our model was too complex that it takes too much time to build the model, and the to predict the classes by going through it. To build the model it took around 73 seconds, and to predict one picture it took 96 secondes.

After this installation, we encounter another problem with our first work, which was connecting the devices, since Keras is using json files, after its installation this last, affected it somehow. The server was not able to work properly, causing many errors. We looked for a solution for a while, and ended up by deciding to let up the deployment of the model, since it was too slow (in real life the driver would already had an accident when the prediction ends). So, we had set up the connection one more time, hoping to ameliorate the deployment using raspberry pi or a more powerful device.

6. Alternatives

For much powerful hardware to predict faster, an alternative to Raspberry pi 3, we can use **Nvidia Jetson TX2,** or **Nvidia Jetson TK1** that are powerful devices for embedded computer vision, they have a GPU and more cores in CPU, that make the image processing much easier, and faster to classify.

7. References

[1] - *World Health Organization*

http://www.who.int/gho/road_safety/mortality/number_text/en/

[2] - National Safety Council, *Annual Estimate of Cell Phone Crashes 2013*

http://www.nsc.org/DistractedDrivingDocuments/Cell-Phone-Estimate-Summary-2013.pdf

[3] - B. Donmez, L. Boyle, and J. D. Lee, "*Taxonomy of mitigation strategies for driver distraction," in Proc. Hum. Factors Ergonom. Soc. 47th Annu. Meeting*, Denver, CO, 2003, pp. 1865–1869.

[4]. Kaggle**.** State Farm Distracted Driver Detection - Data. https://www.kaggle.com/c/state-farm-distracted-driver-detection/data

[5] - Github. CS231n: Convolutional Neural Networks for Visual Recognition. http://cs231n.github.io/

[6] - Hinton, Geoffrey E., et al. "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580 (2012).

[7] - BVLC. Model Zoo. https://github.com/BVLC/caffe/wiki/Model-Zoo

[8] - GitHub. kaggle-statefarm. https://github.com/alireza-a/kaggle-statefarm/blob/master/sr**c/**fine_tune_caffe_net.ipynb

[9] - Andrew Ng**,** Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization, https://www.coursera.org/learn/deep-neural-network.

[10] - Keas Callbacks, ReduceLROnPlateau, https://keras.io/callbacks/#reducelronplateau.

[11] - Keas Callbacks, ReduceLROnPlateau, https://keras.io/callbacks/#reducelronplateau.

[12] - DOT: Intelligent vehicle initiative. United States Department of Transportation. http://www.its.dot.gov/ivi/ivi.htm.

[13] - DOT: Saving lives through advanced vehicle safety technology. USA Department of Transportation. http://www.its.dot.gov/ivi/docs/AR2001.pdf.

[14] - Knipling, R., Wang: Js: Revised estimates of the us drowsy driver crash problem size based on general estimates system case reviews. In: In 39th Annual Proceedings, Association for the Advancement of Automotive Medicine. (1995).

[15] - http://vision.fe.uni-lj.si/cvww2016/proceedings/papers/05.pdf

[16] - https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/

[17] - https://pdfs.semanticscholar.org/d78b/6a5b0dcaa81b1faea5fb0000045a62513567.pdf

[18] - https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/

[19] - Real-time Distracted Driver Posture Classification, Yehya Abouelnaga, Hesham M. Eraqi, and Mohamed N. Moustafa, https://arxiv.org/pdf/1706.09498.pdf

[20] - Classification of Driver Distraction , Samuel Colbran, Kaiqi Cen, Danni Luo, http://cs229.stanford.edu/proj2016/poster/SamCenLuo-ClassificationOfDriverDistraction-poster.pdf

[21] – Distracted Driver Detection: Deep Learning vs Handcrafted Features, Murtadha D Hssayeni, Sagar Saxena, Raymond Ptucha, Andreas Savakis; http://www.ingentaconnect.com/contentone/ist/ei/2017/00002017/00000010/art00004?crawler=true &mimetype=application/pdf

[22] – Distracted Driver Detection, BY: CESAR HIERSEMANN, http://fileadmin.cs.lth.se/cs/Education/edan70/AIProjects/2016/slides/Hiersemann.pdf

[23] – State Farm Distracted Driver Detection, Mateusz Buda and Jokull Johannsson, https://github.com/mateuszbuda/StateFarm/blob/master/DD2427_buda_jokull_final.pdf

[24] – State Farm Distracted Driver Detection, Nishaat Farheen, Mohit Ravi, Lanyin Zhang, https://www.stevens.edu/sites/stevens_edu/files/2016-bia-corporate.pdf

[25] Make a simple chat application through bluetooth in Android.
https://github.com/DevExchanges/BluetoothChatAppAndroid

[26] Simple Bluetooth chat application in Android
http://www.devexchanges.info/2016/10/simple-bluetooth-chat-application-in.html

[27] Create a Lock Screen Device App with Android Studio
https://www.youtube.com/watch?v=JJcbB4FwtLY

[28] Android reject call **http://www.worldbestlearningcenter.com/tips/Android-reject-call.htm**

[29] BLE_Android
https://github.com/qbalsdon/BLE_Android

[30] BLE_Rpi_Peripheral
https://github.com/qbalsdon/BLE_Rpi_Peripheral