Detecting Distracted Drivers – Final Report

Contents

[Summary of problem statement, Data and findings 2](#_Toc522382338)

[Dataset: 2](#_Toc522382339)

[Learning 3](#_Toc522382340)

[Overview of the final process 4](#_Toc522382341)

[Approach -1: 4](#_Toc522382342)

[Approach -2: 5](#_Toc522382343)

[Approach – 3 (Value Add): 6](#_Toc522382344)

[Salient Features of Data: 6](#_Toc522382345)

[Data Pre-processing 8](#_Toc522382346)

[Step by step walkthrough of solution 9](#_Toc522382347)

[Approach -1 9](#_Toc522382348)

[Approach-2: 11](#_Toc522382349)

[Approach – 3: 13](#_Toc522382350)

[Model evaluation 17](#_Toc522382351)

[Confusion matrix 18](#_Toc522382352)

[Precision- Recall curve 19](#_Toc522382353)

[Comparison to benchmark 20](#_Toc522382354)

[Visualization (s) 21](#_Toc522382355)

[Implications 25](#_Toc522382356)

[Limitations 25](#_Toc522382357)

[Closing Reflections 25](#_Toc522382358)

# Summary of problem statement, Data and findings

Number of road disasters is continuously increasing in last few years worldwide. This makes it imperative to take measures to curb the number of road fatalities. The major cause of these accidents is due to driver’s error. We attempt to develop an accurate and robust system for detecting distracted driver and warn them against it. We focus on detecting manual distractions where driver is engaged in other activities than safe driving and also classify the cause of distraction.

We input images of the driver to our model. Each image belongs to one of the 10 classes mentioned in the dataset section. The model then predicts the class of an image by giving as an output a probability for each class.

## Dataset:

State Farm is a large group of insurance and financial services companies throughout the United States. They released their dataset of 2D dashboard camera images for a Kaggle challenge. The dataset had 22400 training images and 79727 testing images. Resolution was 640 x 480 pixels

The training images had corresponding labels attached. Labels belonged to one of the ten classes as mentioned below:

c0: normal driving

c1: texting – right

c2: talking on the phone – right

c3: texting - left

c4: talking on the phone - left

c5: operating the radio

c6: drinking

c7: reaching behind

c8: hair and makeup

c9: talking to passenger

A sample input is shown:



We propose a reliable model that achieves a 95% driving posture classification accuracy. We build a custom code model from scratch and also implemented pre-trained models to compare the accuracy of the test data. Finally, we present the best version of our model that could achieve a 95 % classification accuracy and operate in a real-time environment.

## Learning

Our important learning while executing the project was

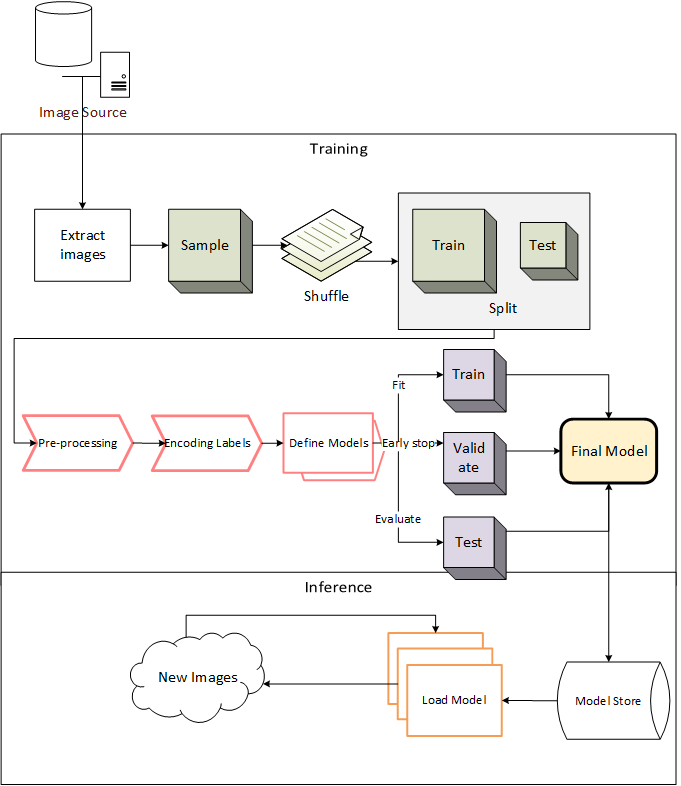
1. We had to use optimal sample size to evaluate different types of models. Using all of the input data was not possible.
2. Applying pre-processing techniques, especially converting to gray scale helped improve performance a lot. Also, helped attain accuracy at a faster pace.
3. Visualizing the images in between the layers, helped understand the emphasis on the section of images being learnt during the course of training.
4. Regular machine learning models should be analyzed for initial understanding before using neural network models and also can be ensembled for better accuracy.

# Overview of the final process

We attempted to solve the problem by using two different approaches.

1. By developing our own custom model
2. By using multiple pre-trained models and choosing the best from those

The goal is to arrive at a model to achieve lowest log loss and best accuracy possible, with the selected samples. The actual result could be visually observed by looking at the predictions given by the chosen models for few images that are unseen by the model.



### Approach -1:

For feature extraction, convolution layers are being used for image recognition and then functional approach of using dense layers and max pool layers to create and train a model which will help us classify into one of the ten types.

We developed custom code to try multiple combinations of convolution and dense layers. 9 models were generated and trained for varying number of samples.

The best model resulted in test accuracy of approximately 96%, which was achieved after applying image pre-processing and regularization techniques.

1. Images were converted to Gray scale for faster execution. This also enabled including more number of images for better and faster prediction. For our problem statement, there is less emphasis on information from color, hence converting to Gray scale seemed to be ideal.
2. DropOut - Dropout is an efficient way of reducing overfitting by randomly dropping out / ignoring some neurons. We have applied standard dropouts across all convolution layers.
3. Batch Normalization - Batch normalization helps to improve the performance and stability of neural networks by explicitly forcing the activations through a layer of network to follow a unit Gaussian distribution [6]. It reduces strong dependence on weight initialization, improves gradient flow through the network as well as allows higher learning rates. In our work, activations of all convolutional layers are normalized.

### Approach -2:

Transfer learning is the idea of using a CNN model pre-trained on a large dataset as an initialization. It gave us a significant boost in terms of speed and performance. For each model we tried, we only modified the last FC layer to output 10 class predictions instead of 1000 or more. We then use our own training set as the input images to train the whole neural network. The below pre-trained models have been trained for the distracted driver image set. The models load a set of weights pre-trained on ImageNet.

1. VGG 16 and VGG 19
2. AlexNet
3. ResNet50
4. MobileNet

Following are the observations from each model.

VGG 16 and 19:

The architecture of VGG is characterized by its simplicity using 3X3 convolution layers stacked on top of each other in increasing depths.

The major drawbacks observed with VGGNet:

1. It is very slow to train.
2. The network architecture weights themselves are quite large in bandwidth.

Due to its depth and number of fully-connected nodes, VGG is over 533MB for VGG16 and 574MB for VGG19. Also, we could observe that for the image sample provided, the networks either could not be adequately trained or did not give a good accuracy for each class. This could be because of the sample size we had chosen.

These models resulted in varying accuracies and predictions for every run and hence were not reliable for our case.

ResNet50:

ResNet50 is a 50 layer Residual Network. ResNet does subtraction of feature learned from input of that layer 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.

For our image set, we could observe the best accuracy and prediction using ResNet50 pretrained model. The model training was faster in comparison to VGG.

MobileNet:

MobileNet is an architecture which is more suitable for mobile and embedded based vision applications where there is lack of compute power. This architecture uses depthwise separable convolutions which significantly reduces the number of parameters when compared to the network with normal convolutions with the same depth in the networks. This results in light weight deep neural networks.

By using depthwise separable convolutions, there is some sacrifice of accuracy for low complexity deep neural network

For our set of images, the accuracy was average using this model.

AlexNet:

AlexNet is relatively simpler model with less number of convolution and fully connected layers. This helps to quickly train the model. AlexNet is designed to be used to execute in parallel in multiple GPUs. Since it uses Relu instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy theoretically. Use dropout instead of regularization to deal with overfitting

For our image set, AlexNet model seemed to train the fastest. However, the test accuracy was bit jittery for different runs.

### Approach – 3 (Value Add):

In addition, we had also tried using machine learning and ensembling techniques, after extracting features. For classifier boosting, SVM is used as an alternative to softmax to enhance generalization ability CNN works as a trainable feature extractor and SVM performs as a recognizer.

## Salient Features of Data:

The distribution of images in various classes is mentioned in the above section. On analyzing random images, the visibility and brightness seems optimal (though bit on the dark side) for images in different classes. Following are histograms of sample images from each category.

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

## Data Pre-processing

Two main Techniques Normalization and Augmentation (mirroring, rotation, scaling, and Color modifications) have been considered for pre-processing.

The purpose is to correct the deficiencies that may damage the learning process, such as omissions, noise and outliers and adapting the data to simplify and optimize the training of the learning model. However, high abstraction capacity of CNNs allows them to work on the original high dimensional space, which reduces the need for manually preparing the input

Data augmentation increases the volume of the training dataset by applying several transformations to the original input. However since we had adequate samples in acceptable quality for the sample size used for training. This technique could be used when higher computing capacity is available.

The major pre-processing applied to the samples were to resize, normalize and convert to gray scale in case of custom model.

# Step by step walkthrough of solution

## Approach -1

Custom Models have been used for the prediction. Both the Convolution and Max\_Pooling Layers have been listed on our own.

Grid Search for the Convolution and Denser Layers combinations are not available in Keras. Hence, we have developed a custom code that could run multiple variations and combinations of the Convolution and Dense Layers.

Initially the channels used was very low and as the number of channels were increased, we could see an improvement in accuracy. Below are couple of the models tried.

Convolution Layer – 1:



Dense Layer -1:



The above combination did not provide satisfactory accuracy.

Convolution Layer -2:



Dense Layer- 2:



This combination provided an overfit on training and 97% accuracy on test.

Convolution Layer- 3:



Dense Layer- 3:



For the above combination, learning Rate was quite low. The train accuracy was 95% and Test Accuracy was 94%.

From the observations, the second model is the best.

## Approach-2:

Similar sample and hyperparameter were applied for various pre-trained models and the observations are as below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **No. of Samples** | **Batch Size** | **No. Of Epochs** | **Optimizer** | **Results** | |
| VGG16 | 3000 | 32 | 25 | SGD | Accuracy Score  Train – 1.000  Test - 0.9622 | Loss  Train - 0.0011  Test - 0.1651 |
| VGG19 | 3000 | 32 | 25 | SGD | Accuracy Score  Train – 0.3100  Test - 0.3377 | Loss  Train - 2.0878  Test - 2.0914 |
| ResNet50 | 3000 | 32 | 50 | adam | Accuracy Score  Train – 1.0000  Test - 0.9677 | Loss  Train - 5.0345e-05  Test - 0.1149 |
| MobileNet | 3000 | 32 | 50 | adam | Accuracy Score  Train – 0.6214  Test - 0.5822 | Loss  Train - 0.8248  Test - 1.0023 |
| AlexNet | 3000 | 32 | 50 | adam | Accuracy Score  Train – 0.9833  Test - 0.8922 | Loss  Train - 0.0596  Test - 0.4927 |

Accuracy and Loss:

|  |  |
| --- | --- |
| VGG | |
|  |  |

|  |  |
| --- | --- |
| AlexNet | |
|  |  |

|  |  |
| --- | --- |
| ResNet50 | |
|  |  |

|  |  |
| --- | --- |
| MobileNet | |
|  |  |
|  |  |

## Approach – 3:

Following are the observations on using machine learning models and ensembling techniques.

Method – 1: By directly feeding images in the form arrays, without using CNN layers.

Method – 2: By using CNN for feature extraction

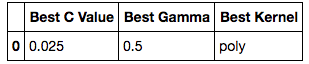


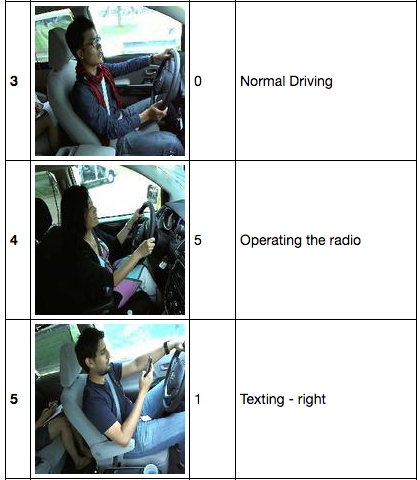
|  |  |
| --- | --- |
| **Logistic Regression** | |
| /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.23.11 AM.png | /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.24.57 AM.png |
| **Naive Bayes Classifier** | |
| /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.30.05 AM.png | /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.24.57 AM.png |
| **Decision Tree Classifier** | |
| /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.32.48 AM.png | /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.32.59 AM.png |
| **KNN Classifier** | |
| /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.42.10 AM.png | /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.42.22 AM.png |
| **Random Forest Classifier** | |
| /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.47.27 AM.png | /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.48.04 AM.png |
| **Ada Boosting Classifier** | |
| /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.51.54 AM.png | /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.52.02 AM.png |
| **Bagging Classifier** | |
| /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.52.13 AM.png | /Users/deepa/Desktop/Screen Shot 2018-08-18 at 11.52.26 AM.png |

Sample Predictions using Logistic Regression Model:



Method – 2: Results





# Model evaluation

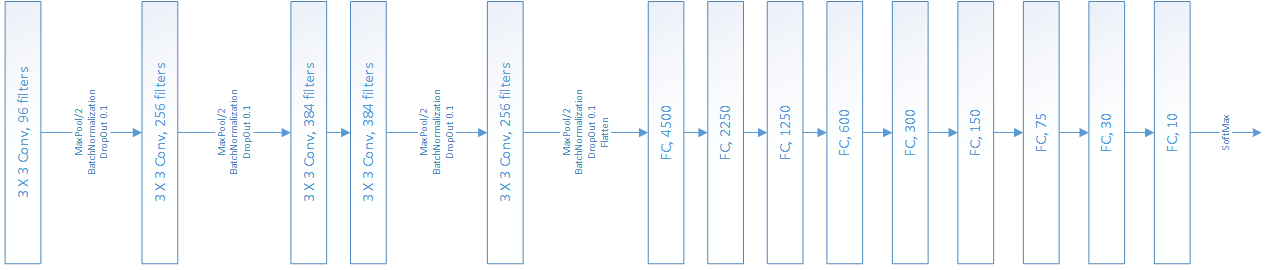
The best model we have arrived at from various trials is below

Convolution Layers:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Layer Size** | **No. of Filters** | **Filter Dimension** | **Stride Dimension** | **Activation** | **MaxPool Dimension** | **MaxPool Stride** |
| 224 | 96 | 3 | 2 | relu | 2 | 2 |
| 56 | 188 | 3 | 2 | relu | 2 | 2 |
| 14 | 256 | 3 | 1 | relu | 2 | 1 |
| 11 | 384 | 3 | 1 | relu | 2 | 1 |
| 8 | 768 | 3 | 1 | relu | 2 | 1 |

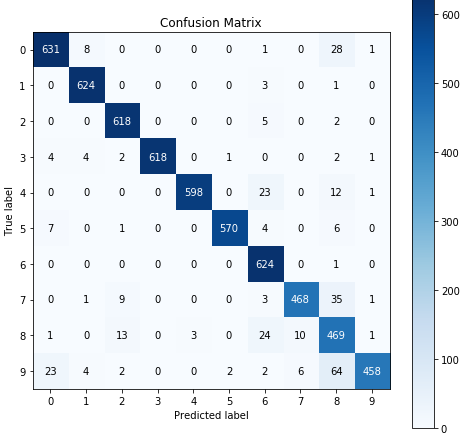
Dense Layers:

|  |  |  |
| --- | --- | --- |
| **Number of Neurons** | **Activation** | **Drop Probability** |
| 4500 | relu | 0.1 |
| 2250 | relu | 0.1 |
| 1250 | relu | 0.1 |
| 600 | relu | 0.1 |
| 300 | relu | 0.1 |
| 150 | relu | 0.1 |
| 75 | relu | 0.1 |
| 30 | relu | 0.1 |
| 10 | softmax | 0.1 |

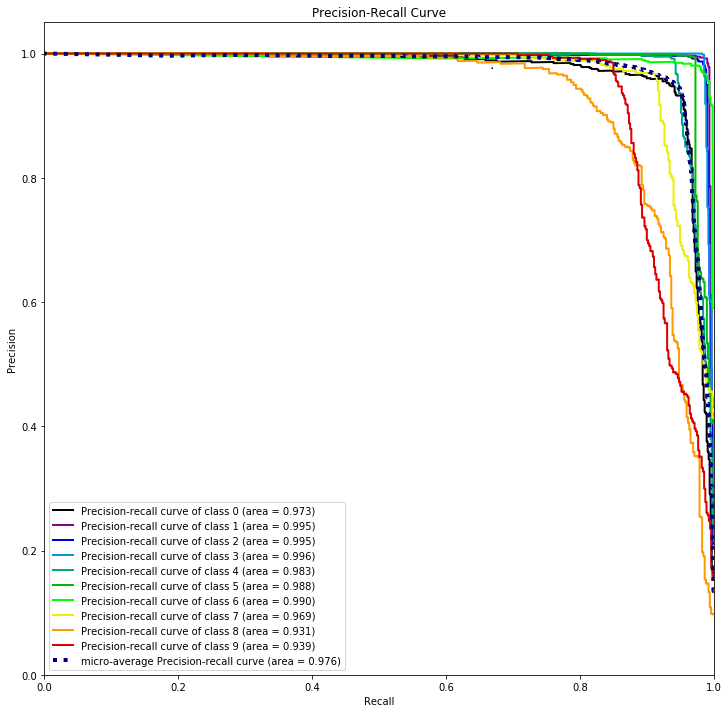


The results from this model are

### Confusion matrix



### Precision- Recall curve



# Comparison to benchmark

This problem was a public challenge hosted on Kaggle by State Farm insurance company two years ago and many approaches were followed by the contributors to achieve better accuracy.

Some of the solutions were based on SVM, face and hand segmentation using RCNN, handcrafted features (HOG and BoWs), and quite a few approaches based on Deep CNN models which are pre-trained on ImageNet such as AlexNet, ResNet152, VGG-16.

The American University in Cairo, Egypt presented a model using a similar dataset that follows the same postures as kaggle. They used weighted ensemble of classifiers using a genetic algorithm to yield 96.4% classification accuracy. Considering this as our benchmark we ensued our try-outs implementing various stratagems. We accomplished 96% accuracy by using custom model which is little less than our benchmark.

We only used CPU systems for this project. If we would have had access to more computing resources, we could have tried to improve our results by implementing L2 regularization and ensemble techniques.

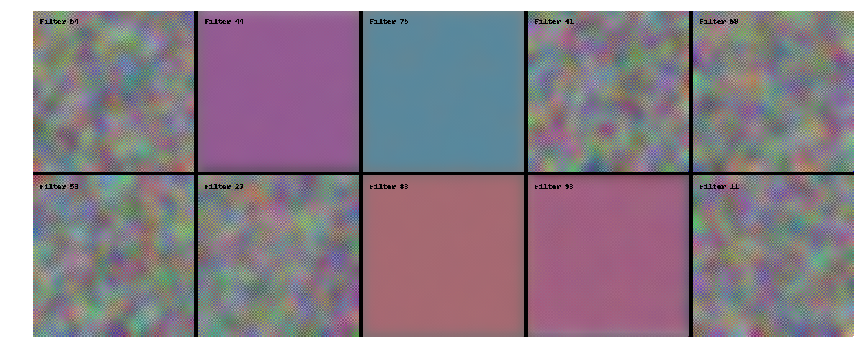
# Visualization (s)

This Keras-Vis helps to visualize the activations of each unit of a neural network based image classifier as a graphical plot. The hidden layers use the ReLU activation function and the output layer uses the softmax activation function. Although the custom model trained works with about 97% accuracy in grayscale format and 87% accuracy in RGB and the primary concern of this experiment is to visualize the activations in each unit of a trained model.

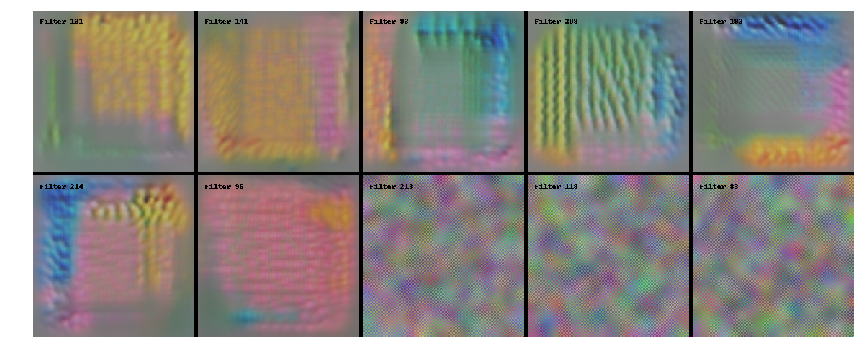
The below are the graphical plots of activations in each unit in each layer for every pixel component (i.e. R, G and B format). Each image is a visualization of what the activations in a specific unit looks like. For example, the first image for layer 1 is the visualization of the activations of the first unit in the first hidden layer.

Each pixel in an image below represents the activation in a specific unit for the corresponding pixel in the input image. The activation for each component (red, green and blue) for each pixel in each unit is computed separately. Then the activations of red, green and blue components in each pixel is combined and shown as a single pixel in an image below.

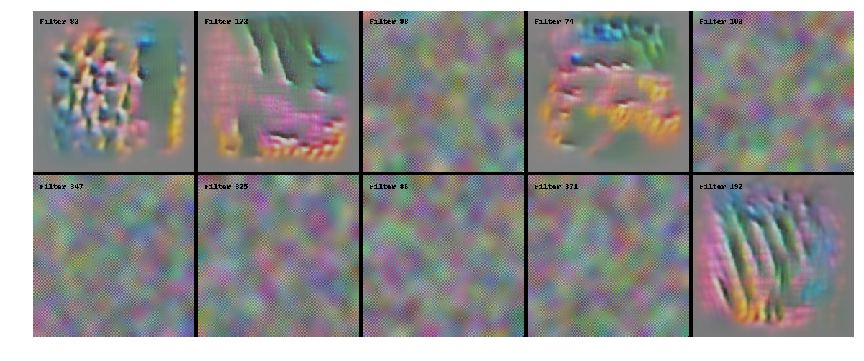
Layer 1 Activations - conv2d\_1



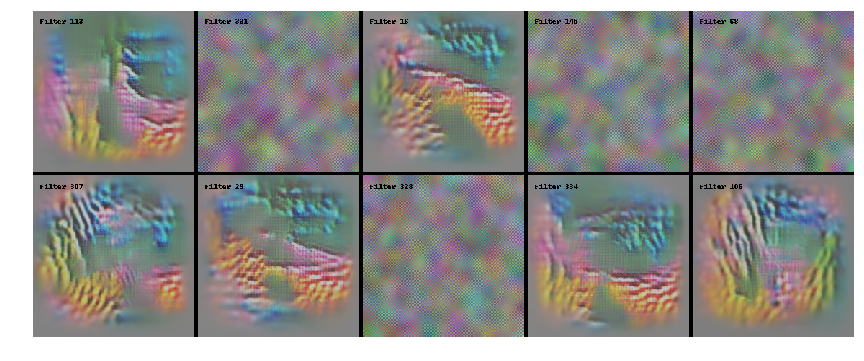
Layer 2 Activations - conv2d\_2



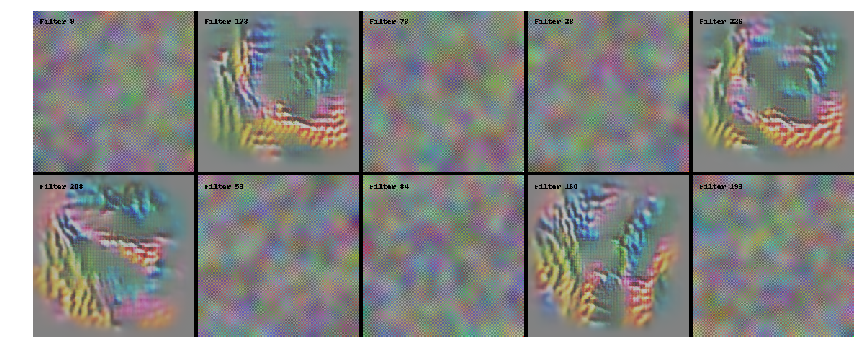
Layer 3 Activations - conv2d\_3



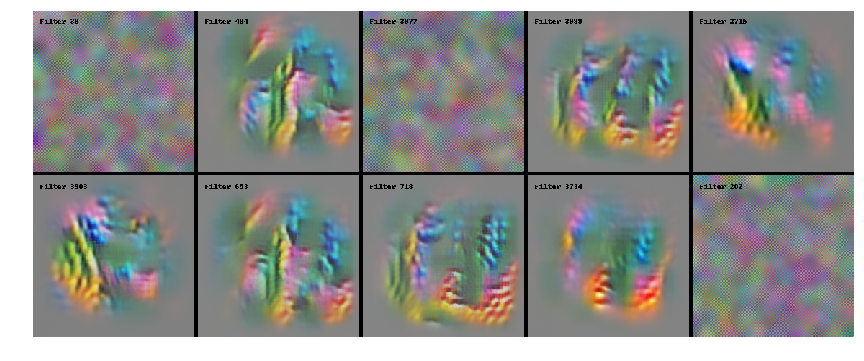
Layer 4 Activations - conv2d\_4



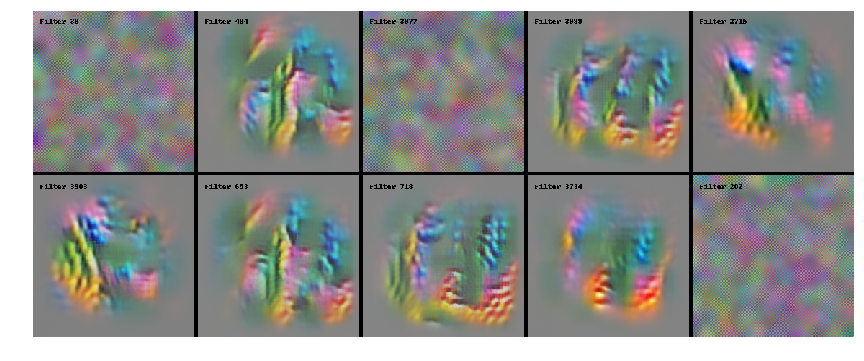
Layer 5 Activations - conv2d\_5



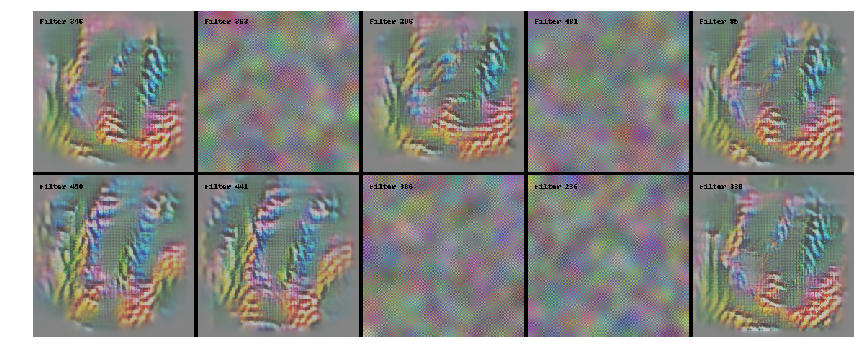
Layer 6 Activations - dense\_1



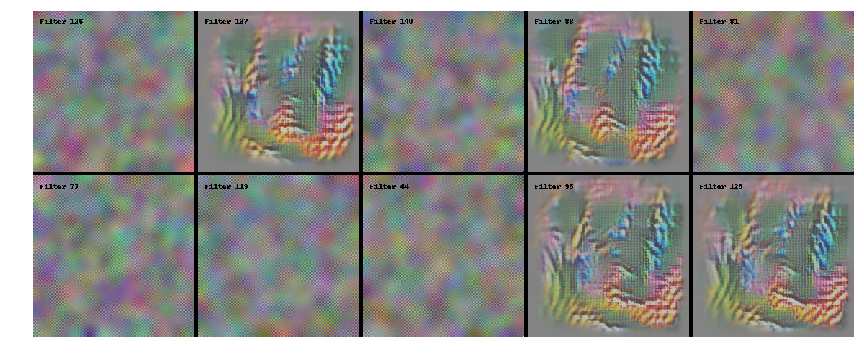
Layer 7 Activations - dense\_4



Layer 8 Activations - dense\_6



Layer 9 Activations - dense\_9



# Implications

The model we proposed can be used as real-time distracted driver posture estimation system and also can be used as an important system component in self-driving vehicles. Since, the custom model yields a very good accuracy of 97% with given limited number of train and test, we expect it to work better in production environment with 90-95% level of confidence when adequately trained. Further we recommend to use wide spectra of images from different countries to train and test the model.

# Limitations

The challenging part of our project is that it is very easy to overfit our training model due to limited number of drivers in our dataset and also due to trivial dataset. Because of high similarities between two images with the same driver, it is more likely for the training model to over-emphasize on the drivers’ features instead of their driving behaviors.

Since the test images are highly correlated and our training data set is small, it is beneficial to try data augmentation including color shifting and rotating to avoid overfitting problem, in real time systems.

# Closing Reflections

**Learning:**

Visualizing the output is also important, once you can see the image, you have a better understanding whether the model is doing a reasonable job or not. Splitting of train and validation set such that images of drivers in the validation set are not in the training set.

Confusion matrix is a really nice visualization for small number of class classifications. Instead of running the entire 20k images, it’s better to kick start with subset of data (around 20%).Starting with barebones model (without any convolutional layer) and then adding layers gradually and fine tuning will help to improve the model better in CNN.

**Future Enhancement:**

Another step that we would like to try in the future for better performance is to devise a better face and hands detector, which manually label hand and face proposals and use them to train a Fast RCNN to localize both faces and hands in one shot. The same can be used to evaluate against our existing custom CNN-based model.