Distracted Driver Detection

Charith Musku, Bhargav Kandlagunta, Bhagya Reddy

Abstract
The aim of the project is to classify images of people driving a car, taken by a side view dashboard camera, into different classes of distraction (texting, drinking, talking on the phone, makeup, reaching behind, etc) using Convolutional Neural Networks. The dataset for the problem is taken from Kaggle State Farm competition[3]. Training data is improved by the best possible data augmentation techniques for this data that are width shift and zoom. A basic CNN was used to train the model initially, which resulted in a log loss of 1.4. Later on we performed transfer learning, using pre-trained models such as VGG16 and VGG19, each of them tuned with different parameters, and three fully connected layers added on top of each model. Out of these VGG16 performed better which gave an validation accuracy of 64% and a log loss of 0.97

Keywords
CNN — VGG16 — Transfer Learning

[1] *Data Science, School of Informatics, Computing, and Engineering, Indiana University, Bloomington, IN, USA*

Contents

Introduction

According to the CDC motor vehicle safety division[1], 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 insure their customers better, by testing whether dashboard cameras can automatically detect drivers engaging in distracted behaviors. Given a dataset of 2D dashboard camera images, This competition challenges participants to classify each driver's behavior. Are they driving attentively, wearing their seatbelt, or taking a selfie with their friends in the backseat?

We have initially started with a basic CNN using the very popular MNIST digit recognition architecture, which resulted in a log loss of 1.47. This model has already outperformed the best of other non-CNN models' results without any optimization. Later on we performed transfer learning, using pre-trained models VGG16 and VGG19, each of them tuned with different parameters, with three fully connected layers on top of basic architecture. VGG16 gave better results compared to other models in the end with a log loss value of 0.97.

1. Exploratory Data Analysis

1.1 Data Analysis

The data-set being used is the one provided in the competition[2]. It consists of images captured by a side view dashboard camera of the car. The training folder has sub folders of 22424 images from 10 different classes and test folder with 79276 images. Following are the list of 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 makeup
- c9: talking to passenger

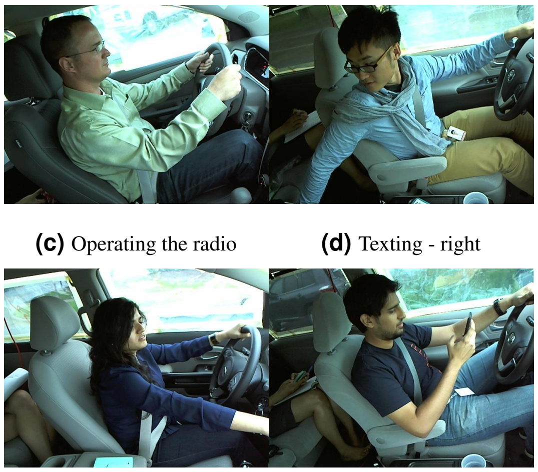

Figure 1. Sample driver images
(a) Safe driving **(b)** Reaching behind
(c) Operating the radio **(d)** Texting - right

A file with information about all the images mapped to corresponding driver id and class was provided. There are 26 unique drivers overall, with number of images being distributed almost equally. But the number of images in each of the class are not equal. Number of images in each class in the training set are distributed in the range of 1911 to 2489.

1.2 Visual Analysis

Initially we thought of cropping and using only the area in the image surrounding the driver, by eliminating unnecessary background and the back seat etc. But we have identified few interesting characteristics of the data, like, consider the below example:

Although both the images belong to the same class they are unique in some ways. For example:

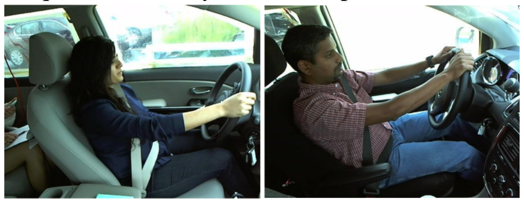

Figure 2

- First image has captured part of a person sitting in the backseat whereas the second image is strictly confined to the driver's seating area.
- The first person uses a single hand to drive by resting the other hand, whereas other second driver uses both hands to drive.

These kind of differences are necessary in classification problems like these. They help to generalize our model much better without limiting to few scenarios. This is just one of the examples, there are many such features of the data such as, Clothing, Lighting, Camera angle, Hand placement etc.

2. Methodologies

2.1 Data Pre-processing

- Images are converted into 3 channels (R, G, B) which would be appropriate for the given data set.
- All the original images which are of size 640 x 480 have been converted to 80 x 80. By doing this we may lose some details of the image, but saves a lot of time in computation and memory.
- Shuffling the training data randomly. Although this might not be much helpful, but we could do this by just passing an extra parameter to the keras[6] function.
- Target labels have been encoded using the one hot encoding technique into a matrix of size M x N where , N are number of samples, and number of classes respectively. This is required since there are 10 classes to be predicted.
- Tried two approaches for data normalization:
  - Normalized the data by calculating the mean pixel values R,G,B and subtracting it with the values at individual pixel.
  - Divide each of the R, G, B pixel values by 255

  Sticked to the first approach as it worked better.
- For the data augmentation, as mentioned earlier in the above section, the given data is already very helpful in generalizing our model. So, we have only tried few additional techniques:
  - Zoom
  - Width Shift
  - Height Shift
  - Horizontal Flip

The performance of the model for each data augmentation technique will be discussed in the later sections.

2.2 Cross Validation

We have tried two different techniques of cross validation here:

Random Split

- We have initially started with dividing the training and validation sets randomly with 1:10 ratio. But that didn't work quite well.

Split by Drivers

- So, later in order to make the validation set totally different from the training set, we divided the train, validation sets based on the driver. The driver data for the same has been provided in the competition. By doing this we can be more reliable on the validation results that we get, as we are validating our model on new data.

2.3 Averaging over K-Folds

- For training, We have used the average of all folds validation accuracy and log loss as metrics to measure the performance and better tune the parameters of the model.
- We saved the resulting weights of CNN model for each fold of validation during training. While predicting test dataset, all test images are subjected to each of the saved weights corresponding to a fold and mean of all the predictions are taken as final result, so that the predictions are not biased.

2.4 Call Backs in Keras

2.4.1 Early Stopping

- We have used the early stopping technique, which is, when we are validating our model over each fold, if there is no significant improvement in the validation log loss for 3 continuous epochs, we discard it so that we can at least save some computation time and figure out what's not working. The usual trend is that the model converges within 4 to 5 epochs, so it doesn't make sense to keep continuing even if there is no significant improvement for consecutive epochs.
- Keras has an inbuilt model to perform this. We can make use of this by passing relevant parameters like what metric to monitor and by what value and patience period, which is the wait period before discarding training process.

2.4.2 Model Checkpoint

- We have used the Model checkpoint in keras to save weights after performing each fold of validation.
- We can save the best weights out of all epochs in each fold by just passing an extra parameter to the Model Checkpoint

3. Models

Since the emergence of Convolutional Neural Networks(CNN), it has become the state of the art technique for Image classification problems. CNN is a Deep Learning architecture comprising of basic units called neurons functioning similar to that of human neurons. CNN is a special case of Artificial Neural Networks. For huge and complex data sets like images, to process and learn patterns lying behind, the type of architecture CNN follows is the most suitable one. With multiple layers of neurons and back propagation the most complex of the features are learned from large number of images. Image classification can also be carried out using other machine learning algorithms but CNN is the state-of-the-art technique.
So, we started with basic CNN model making results of existing non CNN models as a benchmark.

3.1 Why CNN ?

To start with, we have used a very basic CNN model using the very popular MNIST architechture, except changing the color coding from gray-scale to RGB, without any data augmentation or optimizing parameters.
Suprisingly, this model has already outperformed the best of the non CNN models developed for this competition. We came across few SVC models implemented with PCA, Grid search[7] with 5-fold etc., but even best of those models could not fetch better results than CNN. Following are the results:

- SVC model from Kaggle - a reported log loss of 1.53
- Our basic CNN model - Log loss 1.42

If the very basic CNN had such good results. With advanced techniques and optimizing parameters CNNs can result in much better results

3.2 Basic CNN

This is a simple CNN based on MNIST digit classification architecture. Following the layers involved in this:

- 3 Convolutional Layers with 32 to 128 features
- 3 Max-Pooling layers with strides (2,2), alternating after each Convolutional Layer
- And a fully connected Dense layer with 10 nodes and softmax activation function

The architecture is as shown in the figure below:

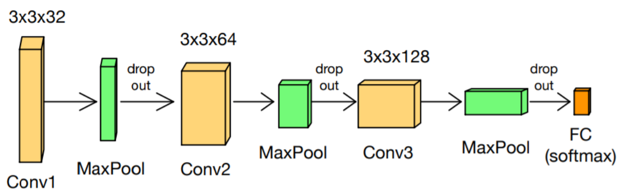

Figure 3. Basic CNN Model

This is a non optimized CNN implementation with all the default configurations and parameters taken as is. All the layers are involved in the training process.

3.3 Transfer Learning

For the transfer learning approach, after understanding few of the popular pre-trained models, we have decided to use the below:

- VGG16
- VGG19

There is a reason why we chose these in particular. The architectures of these were designed to focus more on objects and background, rather than human face or other foreground objects. And that is exactly what we need for these kind of classification problems.

3.3.1 VGG16

VGG is very a deep convolutional network for object recognition developed and trained by Oxford's renowned Visual Geometry Group (VGG) [4], which achieved very good performance on the ImageNet dataset. They have published their network architecture along with it's weights on their website. Below is the Architecture of VGG16:

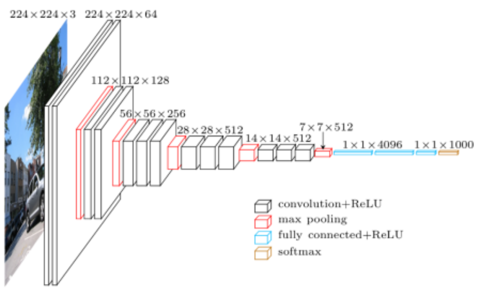

Figure 4. VGG 16 Architecture

Using the above architecture along with it's weights, we have added few additional layers on top of it and trained on our data.

- 2 Dense Layers with 4096 neurons in each layer.
- 2 alternating drop out layers.
- 1 final softmax layer.

This resulted in much better results, we were able to get a log loss value of 0.97.

3.3.2 VGG19

VGG19 is an upgraded version of VGG16, which was develoepd by the same team. The architecture is similar to VGG16 but with extra Convolutional layers added before each of the Max Pooling layers.
Below is the architecture of VGG19:

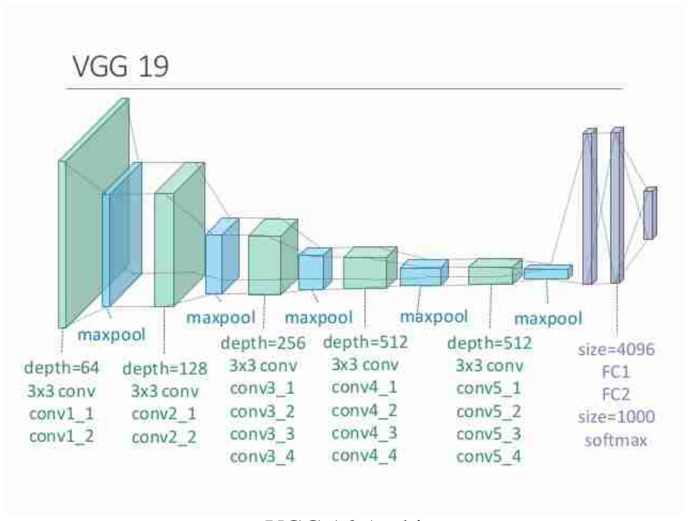

Figure 5. VGG 16 Architecture

Surprisingly, this model did not perform better than VGG16, although it is supposed to be an upgraded version of it. This might be bacause of the data we are dealing with and the patterns that are being learnt from are different for VGG16 and VGG19.

4. Experiments and Results

4.1 Evaluation Metric

Log loss metric is used to check the performance of the models. This is a good evaluation metric as it severely punishes when model predicts an image as a different class from the actual one. Kaggle uses this metric for submissions in the competition. Here is the formula for log loss.

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

4.2 Parameter Tuning

Following are the parameters that we have tried to hand tune:

- Epochs: The usual trend that followed was that the model converged within 6-7 epochs. So we have finalized 10 epochs.
- Batch Size: We have tried with batch size as 32 and 64. Batch size 16 could have produced slightly better results, but at the cost of extra computation.

Grid search for Optimization algorithm and Learning Rate:

- Since grid search involves heavy computation time, especially for models like these. I have used it only for these two parameters, with 3 values each for just 1 fold and 5 epochs.
- Have finally used SGD as optimizer and 0.001 as learning rate.

4.3 Basic CNN

We implemented the simple CNN architecture mentioned in the second section to stat with. For data preprocessing images were resized to 80 x 80 pixels and normalized. Since we have 10 classes the target label classes for all the train images are one hot encoded to comply with the architecture. The following are the parameters considered for implementation:

- Learning rate : 0.001
- Number of epochs : 10
- Batch size : 32

Categorical cross entropy is used as loss metric for the model. The model is trained with the above configurations and resulted in a log loss of 1.42.

4.4 VGG 16

For VGG-16 we used 'Imagenet'[5] pretrained weights. As part of data pre-processing, images are reshaped to 80x80 pixels and then normalized. Since the number of training data images for this problem is relatively low compared to test images, we would not have a great deal of variations in each of training class images. So data augmentation can make up for this by providing augmented images as required. With keras there are a handful of augmentation techniques that can be applied to the training data on the go without utilizing any additional memory for the augmented images. Each of the image is randomly applied with one of the augmentation techniques, thus giving the model a wide variety of images in a class to learn from. The selection of what augmentation techniques to use depends on type of data that is being dealt with. Typically we need to consider the techniques that benefit the model to learn the patterns more efficiently. So here in our case we have implemented few augmentation techniques and evaluated the results to figure out which would be a better choice. Augmentations tried were - zoom, width shift, height shift, horizontal flip and rotation. As the images after processing must not mislead the model, the mentioned ones seemed reasonable for the data set. Better approach to check performance of different techniques is to take a train data sample and look for accuracy of models. Here in this case we have run the models for 5-fold validation for 5 epochs. As seen in the Table 1, results for augmentations zoom and width shift were better. This makes sense as the most significant part of images is the center portion where in the actions of driver are captured where as the remaining part does not have an impact on the actual classification. Also width range, which shifts the image along width in either directions(by up-to 20%) was better because this might make up for different angles in which pictures of driver were taken by camera.
Initial learning rate taken was varied and observed that any values higher than 0.001 or values lower than 0.00001 were not giving any good results. So optimized rate of 0.001 was chosen for further modelling. We used SGD optimizer for the VGG models.

Table 1. Data Augmentation log loss values

Augmentation	Log Loss
Width shift (0.2)	1.289
Height shift (0.2)	1.423
Horizontal flip	1.74
Zoom (0.2)	1.32

Number of epochs were chosen based on the train and validation loss value curves after each epoch by running the model for 15 epochs and looking for the converging point. In most of the cases, the loss values did not have significant change after 10 epochs. The log loss plots for train and valid datasets:

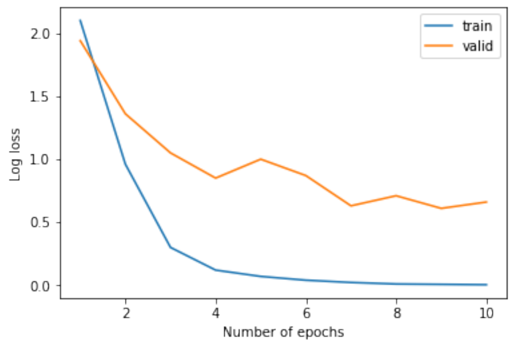

Figure 6. VGG-16 Loss curves

The accuracy plots for train and valid datasets:

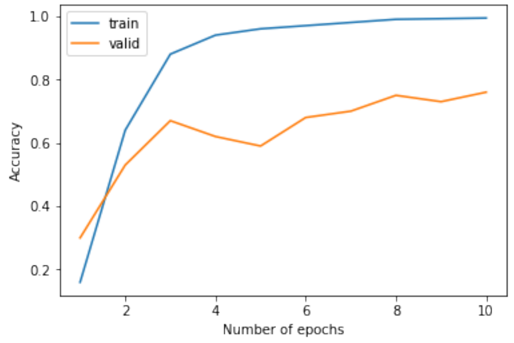

Figure 7. VGG-16 Accuracy curves

To improve the performance of the model we implemented 10-fold cross validation split based on the drivers instead of randomly selecting validation set. VGG Model is trained each time on training data using the parameters given above and corresponding trained weights are saved. During testing phase for each of the fold predictions are made based on the model weights saved and the final predictions were given by averaging the 10 different sets of predictions for each image. This approach improved model's performance by 0.1(log loss). The final log loss(lesser the log loss, better the accuracy) was 1.9

5. Summary and Conclusions

Followed the same training process discussed in earlier sections. To summarize, following are the steps involved in model building and prediction:

Table 2. Final model performance

Model	Log Loss
Basic CNN	1.42
VGG-16(10-fold)	0.97
VGG-19(10-fold)	1.13

- Data Augmentation - Tried different augmentation techniques and chose the best
- Data Normalization
- Using Early Stopping in case of no significant improvement, Using validation loss as a metric to monitor
- K-Fold cross validation, splitting by drivers
- We split the 26 unique drivers as 24, 2 for training and validation sets. By doing this, each of the validation sets will have completely new and unseen data compared to their corresponding train sets.
- Performed a 10-fold cross validation and saved the weights of models after each fold.
- Later on using all of those saved weights to perform prediction on test data, and taking mean results of all models so that it wouldn't be biased towards one model.
- Hand tuning of few parameters and used grid search for parameters like Optimizer and Learning rate

6. Further Improvements

Following are our thoughts on possible improvements:

- We were not able to use an image size greater than 80 x 80, because of some memory issues. Because of this the model, might have missed some details of the image because of small size.
- Because of the same reason, we were not able to try varieties of pre-trained models. While using models with more dense layers, the image being only 80 x 80, got reduced to negative dimensions because of additional convolutional and max pooling layers.
- Had we performed more pre-trained models, we had an idea of using an ensemble of those multiple models, which could have been a huge improvement.

References

[1] Centers for Disease Control and Prevention. https://www.cdc.gov/motorvehiclesafety/distracted_driving/

[2] Kaggle State Farm Distracted Driver Detection - Data. https://www.kaggle.com/c/state-farm-distracted-driver-detec tion/data

[3] Kaggle competition https://www.kaggle.com/c/state-farm-distracted-driver-detection

[4] VGG http://www.robots.ox.ac.uk/ vgg/research/very_deep/

[5] ImageNet http://www.image-net.org/

[6] Keras.io Keras Documentation. https://keras.io/

[7] https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

[8] Convolutional Neural Network for Visual Recognition http://cs231n.github.io/

[9] Using Convolutional Neural Networks to Perform Classification on State Farm Insurance Driver Images. http://cs229.stanford.edu/proj2016spr/report/004.pdf

[10] Using Kaggle environment for Deep Learning https://towardsdatascience.com/how-i-tackled-my-first-kaggle-challenge-using-deep-learning-part-1-b0da29e1351b

[11] Kaggle discussion thread https://www.kaggle.com/c/state-farm-distracted-driver-detection/discussion/22906