Classification of Driver Distraction

Samuel Colbran (06118942), Kaiqi Cen (06163371), Danni Luo (06116105)

Abstract *- This paper describes the process of using the caffe deep learning framework to train and test two convolutional neural network models (VGG-16 and GoogleNet) to classify distracted drivers for the State Farm challenge on Kaggle. To improve the results, we created an ensemble and also averaged over K nearest neighbors. Our best result is a log loss of 0.28554 using the combined ensemble of VGG-16 and GoogleNet with K = 10. This score puts us within the top 12% among all 1440 participants on Kaggle.*

1. Introduction

The Center for Disease Control and Prevention (CDC) found that nearly one in five motor vehicle accidents were caused by distracted driving [1]. This equates to more than 3,000 deaths and 424,000 injuries every year in the USA. Motivated to reduce these statistics, the purpose of this project is to accurately classify what drivers are doing and whether they are distracted.

The input to our models are images of people driving. Each image belongs to one of the ten classes described in Section 3. We then use two different types of convolutional neural networks (CNN): VGG-16 and GoogleNet to predict to which class the given images belong. The output is a list of predicted class labels and the corresponding probabilities (confidence) for all images.

2. Related Work

Many solutions already exist because this problem was a public challenge hosted by State Farm, an insurance company, on Kaggle. Many top solutions used pre-trained CNN models and the most popular are VGG-16 and ResNet, which were state-of-the-art one or two years ago and had been improving [2][3]. Besides these models, two of the top performing solutions we consulted for our project also used several good ideas: ensemble, K nearest neighbors (KNN) and data augmentation [4][5]. First, because the test set size is about four times as large as the training set, it is easy to overfit. Creating an ensemble of models can reduce variance and alleviate this problem. Secondly, because the images are taken from a video clip, many images can be very similar or almost identical and they should be classified to the same class. Applying KNN can yield more stable results. We have seen from previous solutions that KNN was used for either test or the training test (data augmentation) with different ideology and the results were good in both cases.

The same problem was also attempted in a prior CS 229 project [6]. The report describes two approaches: training a small network from scratch and also use pre-trained VGG-19 model. In this attempt, all images were reduced to 224×224 from 640×480. This could detract from performance by throwing away some information, but shorten the computing time. The weakness of the solution is that the student tried only single models (i.e. no ensemble) and there does not seem to have a significant amount of fine tuning done on the pre-trained VGG-19 model. Although very time consuming, trying to fine tune pre-trained models a few times is a good investment as it can make the pre-trained models more customised towards the specific problem.

3. Dataset and Features

The data was supplied by State Farm for a public Kaggle challenge. Kaggle hosts numerous machine learning challenges for the research community with large prize pools. The dataset consists of 22400 training and 79727 testing images (640×480 full color) of people either driving safely or doing one of nine kinds of distracted behaviors [7]. An example input image is shown in Fig. 1.

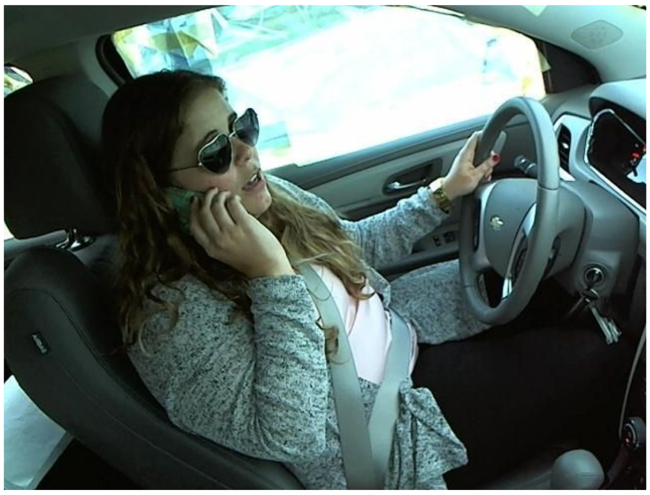


Fig. 1 - Example image input

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

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

To evaluate the success of models, the training images were split into train and validation sets. As the images are highly correlated and each driver only appears in either the training or test set (i.e. none of the drivers that appear in the provided training set appear in the large test set), it was important to choose the images of a certain driver in the training set to be the validation images. Only then can these validation images be independent from the remaining training images of other drivers and thus avoid a falsely high test accuracy. Three drivers were chosen at random and all of their images were separated as validation set, which equated to being 10% of the whole training set.

The provided test images were used only when making a submission to Kaggle, which provided a ranking among all other competitors based on the following metrics [8]:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{10} y_{ij} log(p_{ij})$$

Where N is the number of test images, y_{ij} is 1 if image i belongs to class j with probability p_{ij}.

4. Methods and Results

4.1 Transfer Learning

Transfer learning is the idea of using a CNN model pre-trained on a large dataset as an initialization [9]. 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.

4.2 Convolutional Neural Network

Convolutional neural networks (CNN) are similar to ordinary neural networks (NN) except that they are customized to have images as inputs [9]. This means that the neurons are now structured as a 3D volume. Layers of a CNN transforms one volume to another. The following subsections describe some common types of layers of a CNN. Different CNN models are essentially a permutation of these layers with different filter types, functions, parameters, etc.

4.2.1 Input Layer

The input layer holds the pixel values of the input image. Our project uses full color images, so the input layer is $640 \times 480 \times 3$ (for R, G, B channels).

4.2.2 Convolutional (CONV) Layer

The CONV layer's parameters are a set of learnable filters of small dimensions (e.g. $5 \times 5 \times 3$). The filter convolves with the input volume (across width and height in 2D) to select small areas (e.g. 5×5) 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).

4.2.3 Rectified Linear Units (ReLU) Layer

The ReLU layer applies an element-wise non-linear activation function to increase nonlinearity in the model.

4.2.4 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 (e.g. outputting the maximum values of 2×2 regions, thus reducing the dimension by 75%).

4.2.5 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 [10]. Because some feature detectors make the model adapt to very complex functions in the context of some other very specific feature detectors. By dropping out some activations, some of the high variance due to this is removed.

4.2.6 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.

4.3 VGG-16

VGG-16 is a 16-layer CNN developed by the University of Oxford Visual Geometry Group (VGG). We set up the pre-trained VGG-16 model in Caffe and Jupyter notebooks and trained our network with a batch size of 14 by gradient descent [11][12]. We wanted to set the batch size as high as possible to save training time. However, due to the memory limitation of our hardware, a batch size of 14 is the highest we can set. For the learning rate, we followed Caffe's suggestion: lower the learning rate by a factor of 10 each time and see how the network converges [13]. By doing so we found that our network could not converge well with a learning rate larger than 0.001, or took too long to converge with learning rate smaller than 0.00001. In other words, by setting the learning rate to 0.0001, our VGG network converges quickly. The weight decay (regularization) is set to 5×10^{-4} as recommended by the model. For training iterations, we tested the trained network using our validation set and reached 80% ~ 86% validation accuracy when training iterations were around 1500. With 1500 iterations, VGG achieved a leaderboard (LB) score (log loss) of 0.42, which was the best result we could get with just a single model. The confusion matrix of this best model is shown in Fig. 2 below. It shows that several classes were often mislabelled as Class 8 (hair and makeup). Class 9 (talking to passenger) was often mislabelled as safe driving.

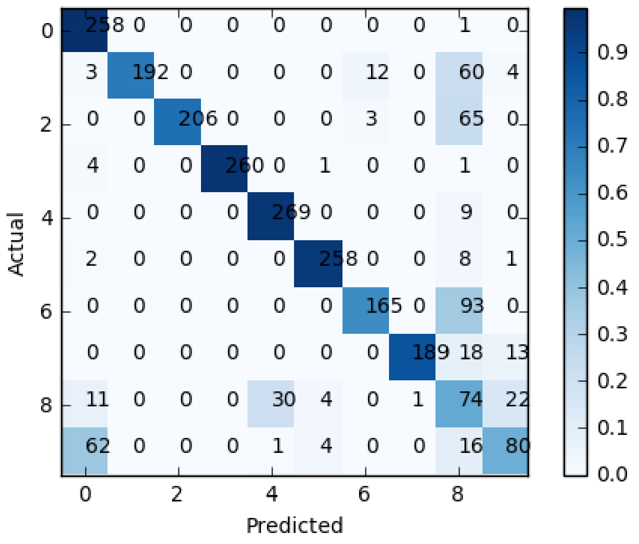


Fig. 2 - 1500 iteration VGG-16 Confusion Matrix

By plotting the test and training accuracies (shown in Fig. 3 below), we found that our VGG model suffered from overfitting - i.e. low bias and high variance.

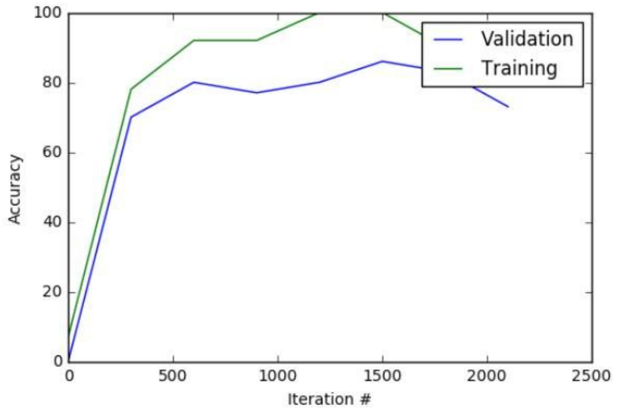


Fig. 3 - VGG-16 Training/Validation Accuracy vs Iterations

To combat overfitting, we tried creating an ensemble of VGG-16 models. We trained VGG repeatedly with iterations ranging from 1200 to 1700 and calculated the average of the top nine predictions to lower the generalization error. The final averaged prediction achieved an LB log loss score of 0.31746.

4.4 GoogleNet

GoogleNet is a 22-layer CNN developed by Google. Similar to how we trained VGG-16, we iterated several times for GoogleNet with a batch size of 32 (we were able to increase the batch size from the previous 14 by using a new graphics processing unit with more memory). Eventually, the best single model was achieved by using an initial learning rate of 0.001, a learning rate multiplying factor γ of 0.1 and a weight decay (regularization) of 5×10^{-4}. Fig. 4 and Fig. 5 show the results. The validation accuracy plot in Fig. 4 indicates that after 300 iterations, the network began to overfit. The confusion matrix shown in Fig. 5 offers more insight, indicating that the classifier is particularly bad at classifying Class 8 and mis-classifies it as Class 4 roughly half the time.

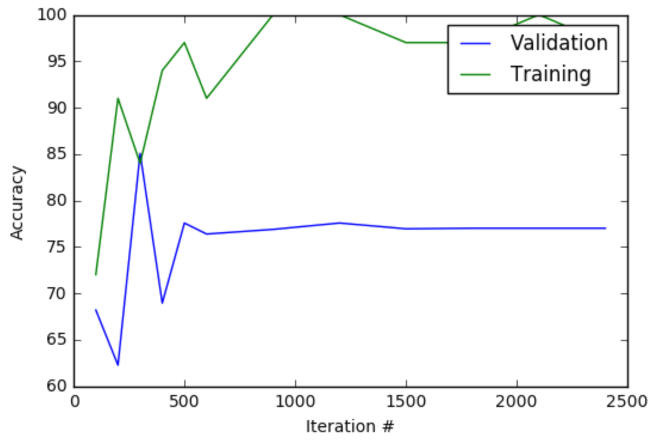


Fig. 4 - GoogleNet Training/Validation Accuracy vs Iterations

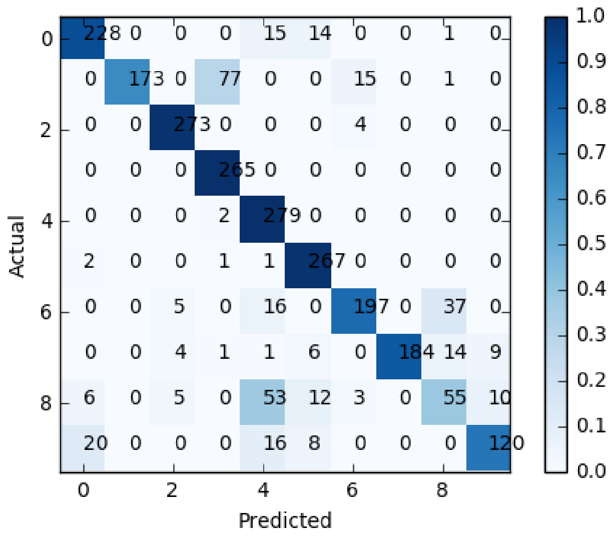


Fig. 5 - 300 iteration GoogleNet Confusion Matrix

4.5 Network Ensemble

An ensemble is the idea of combining (averaging) the results of several classifiers. Given that the test set is about four times as large as the training set, overfitting is the major problem. By averaging different models, the variance of the trained model can be reduced and a lower loss can be achieved. The best leaderboard score achieved using a single VGG-16 model was 0.42. The ensemble created by averaging several VGG-16 models together (with scores of 0.42, 0.45, 0.48, 0.49, 0.50, 0.55, 0.58, 0.44 and 0.445) achieved a leaderboard score of 0.31746. Table 1 shows the impact of creating an ensemble and applying KNN (described in section 4.6).

4.6 K Nearest Neighbors

As mentioned in Section 2, since the images are taken from a video clip, there exist many similar pictures which should belong to the same categories. This implies that applying K Nearest Neighbor (KNN) can improve our classification. The idea is that after the CNN models complete predictions for all testing images, for each test image, we find its K most similar images (including itself) based on pixel-wise L2 euclidean norm. Due to the large quantity of images to be processed, we had to shrink the original test images to 40 × 30 to shorten the computation tie. Fig. 6, as an example, shows one original test image and its four most similar images. We then replace the predicted probability of the test image with the average probabilities of all of these K images. This increases the stability of the performance of the model and lowered the log loss.

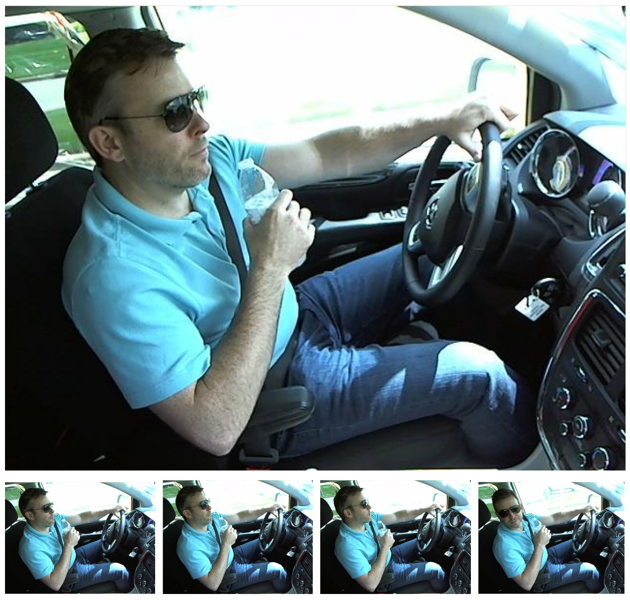


Fig. 6. One testing image and its four most similar images

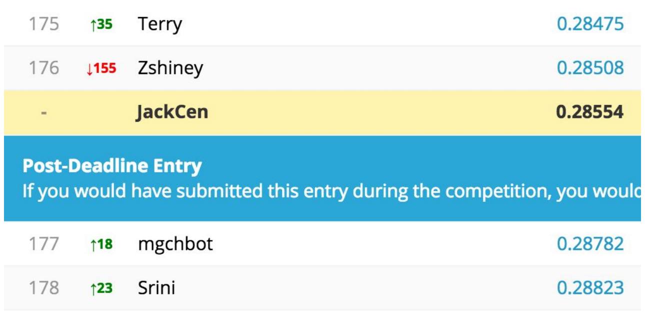
In practice, applying KNN to our 10-class predictions turns out to be very effective in terms of reducing the log loss. It generally can improve the LB score (log loss) by 0.03 ~ 0.1. For average calculation, we used both normal average and weighted average (weight 1 for image itself, 0.9 for its first neighbor, 0.8 for second neighbor... etc). The weighted average yields slightly lower log loss when K is large. We tried with different K values and found optimal results for when K = 10.

Table 1. Top 4 performing model results

	Approaches	LB Score
1.	VGG (0.31) + GoogleNet (0.85) + KNN (K = 10)	0.28554
2.	VGG (0.31) + GoogleNet (0.85) + KNN (K = 7)	0.28646
3.	VGG (0.31) + KNN (K = 9)	0.28684
4.	VGG (0.31) + KNN (K = 7)	0.28732

5. Conclusion/Future Work

After exploring various CNN models combined with network ensemble and KNN, our best performing results were achieved by calculating the average of our best predictions from our VGG-16 and GoogleNet and finally applied KNN with weighted average (K = 10). The final LB score (log loss) we got is 0.28554 which ranked 177 among 1440 total participants (top 12%) on Kaggle's leaderboard (shown in Fig. 7).



175	↑35	Terry	0.28475
176	↓155	Zshiney	0.28508
-		**JackCen**	**0.28554**

Post-Deadline Entry
If you would have submitted this entry during the competition, you would

177	↑18	mgchbot	0.28782
178	↑23	Srini	0.28823

Fig. 7. Final submission achieved a score of 0.28554

As none of the members of this team had much knowledge about deep learning and neural networks before we started the project, we really enjoyed the learning experience and knowledge that we gained. It is the first time for us to participate in a challenge on Kaggle and are satisfied by what we have achieved so far. Given more computing resources and time, we would have tried the following to improve our results:

1. Try ResNet-152. Many top participants reported that apart from VGG and GoogleNet, ResNet achieves promising performance for this challenge.
2. More parameter tuning and optimize our tuning process. Many other participants achieved very promising log loss around 0.2 ~ 0.1 with a single CNN model like VGG, ResNet or GoogleNet. This indicates that there is still space for us to improve our single model's performance.
3. Try data augmentation. So far we have applied KNN only to the test set. We have read that it can be applied to the training images as a way of data augmentation [5]. Apart from applying KNN, we can also try simpler augmentation such as rotation, zooming and shifting of the images. Doing this will effectively give us a larger training set, and thus help to minimise overfitting.
4. Try feature extraction. We have read that cropping out the driver or extracting the driver's right hand can improve the results since these features are more informational.

6. References

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

[2] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

[3] He, Kaiming, et al. "Deep residual learning for image recognition." *arXiv preprint arXiv:1512.03385* (2015).

[4] Kaggle. *A brief summary.* https://www.kaggle.com/c/state-farm-distracted-driver-detection/forums/t/22906/a-brief-summary

[5] Kaggle. *#3 BR POWER - Solution* https://www.kaggle.com/c/state-farm-distracted-driver-detection/forums/t/22631/3-br-power-solution

[6] Singh, Diveesh. "Using Convolutional Neural Networks to Perform Classification on State Farm Insurance Driver Images" (2016)

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

[8] Kaggle. State Farm Distracted Driver Detection - Evaluation. https://www.kaggle.com/c/state-farm-distracted-driver-detection/details/evaluation

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

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

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

[12] GitHub. *kaggle-statefarm.* https://github.com/alireza-a/kaggle-statefarm/blob/master/src/fine_tune_caffe_net.ipynb

[13] BVLC. Solver http://caffe.berkeleyvision.org/tutorial/solver.html