STATE FARM DISTRACTED DRIVER DETECTION

CS452 - Deep Learning for Perception

K163645 - Radheem Razi
K163620 - Abdul Mannan
K163618 - Murtaza Multanwala
K163750 - Shahyar
K163665 - Saira

Section - BCS-8A

July 16, 2020

Abstract

This report describes the process of using the keras deep learning framework to train and test four convolutional neural network models (VGG-19, MobileNet, ResNet, and Xception) to classify distracted drivers for the State Farm challenge on Kaggle. To improve the results, we created an ensemble by weighted average of model predictions and also averaged over K nearest neighbors that were grouped using K-Dimensional trees. Our best result is a log loss of 0.18110 using the combined ensemble with K = 10. This score would put us within the top 2% among all 1438 participants on Kaggle.

Introduction

The Center for Disease Control and Prevention (CDC) found that nearly one out of five car accidents is caused by a distracted driver [1]. Unfortunately, this means around 425,000 individuals get injured and 3,000 deaths are noted because of distracted driving each year.

State Farm is a large group of insurance companies throughout the United States with corporate headquarters in Bloomington, Illinois. Their initiative is to improve these disturbing statistics, and better ensure their customers, by examining whether dashboard cameras can automatically detect drivers participating in distracting practices. Provided a dataset of dashboard camera pictures, State Farm is challenging Kagglers to classify each driver's behavior, for example, what drivers are doing, and whether they are distracted.

Related Work

This challenge was published four years ago due to which many solutions exist on Kaggle. This competition was hosted by State Farm, an insurance company. Transfer learning has been widely used in this competition for better classification results. Some of the popular models that have been used in the submissions are VGG-16, and ResNet, etc.

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][3]. First, because the test size in this competition is approximately four times that of the training set which is not a common practice. This might result in overfitting. To overcome this problem, data augmentation and ensembling have been used in our submission. The models that have been used in the ensemble are MobileNet, VGG-19, ResNet-50, and Xception. These models have been trained to perform classification on image net dataset that contains 1000 classes. The models have to be adjusted to predict 10 classes.

MobileNets are based on a streamlined architecture that uses depthwise separable convolutions to build light weight deep neural networks. The big idea behind MobileNet is that convolutional layers, which are essential to computer vision tasks but are quite expensive to compute, can be replaced by so-called depthwise separable convolutions.

ResNet, short for Residual Networks, is a classic neural network used as a backbone for many computer vision tasks. This model was the winner of the ImageNet challenge in 2015. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+layers successfully. Before ResNet training very deep neural networks were difficult due to the problem of vanishing gradients. It overcomes the problem of vanishing gradient by using the technique of residual learning with the help of skip connections. This skip connection provides an alternate shortcut path for gradients to flow through. This allows the model to learn an identity function which

ensures that the higher layer will perform at least as good as the lower layer, and not worse.
Xception is proposed by google and is a modified form of inception models. It also uses the concept of depthwise separable convolutions but in a modified way. The modified depthwise separable convolution is the pointwise convolution followed by a depthwise convolution. This modification is motivated by the inception module in Inception-v3 that 1×1 convolution is done first before any n×n spatial convolutions. Thus, it is a bit different from the original one.
VGG-19 is an innovative model containing 19 layers. Built as a deep CNN, VGG also outperforms baselines on many tasks and datasets outside of ImageNet. VGG is now still one of the most used models in transfer learning.

Dataset

The dataset used in this project was provided by State Farm through a Kaggle competition, which is a set of images of drivers taken inside a car capturing their activities such as texting, talking on the phone, eating, reaching behind, putting on makeup, etc. As shown in Figure 1. These activities are classified into 10 classes as:

- 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 a passenger

The dataset contains a total of 102150 images split into a training set of 22424 images and a testing set of 79726 images (640 × 480 full color).

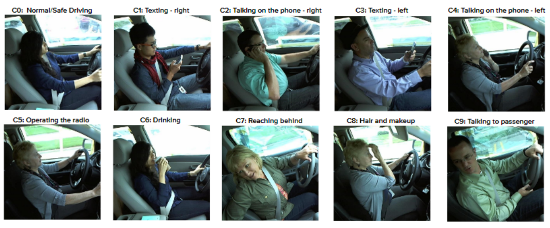


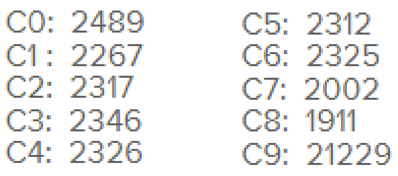
Figure 1: Sample images from each of the 10 categories

Methodology

The approach used in this study employees ensembles of large convolutional networks. These networks require immense amounts of training hence we used Transfer learning to speed up this process. We used learnt weights of famous benchmark networks to initialize our networks and attached a network head of ten units with softmax according to our problem. Weighted average ensemble predictions were produced as the final predictions in order to decrease the validation log loss.

Data Prepocessing

Data pre-processing includes down sampling of the images from 640x480 to 224x224 so that the dimension is compatible with that required by the pre-trained Convnets we use for fine-tuning. Down sample input images is a common practice to ensure the model and the corresponding training samples (from a single mini-batch) can fit into the memory of the graphics card. This could detract from performance by throwing away some information, but shorten the computing time. Since our training image set had only approximately 22K images, we wanted to synthetically get more images from the training set to make sure the models don't overfit as the neural nets have millions of parameters. Image Augmentation is a technique that creates more images from the original by performing actions such as shifting width and/or height, rotation, and zoom. The given dataset had imbalance classes, with following number of images belonging to their respective categories in the train set:



C0:	2489	C5:	2312
C1 :	2267	C6:	2325
C2:	2317	C7:	2002
C3:	2346	C8:	1911
C4:	2326	C9:	21229

Figure 2: Class Imbalance in the given dataset

Since the imbalance ratio is not that vast, we were not sure whether dealing with this problem is more costly (reducing an already small dataset) than to ignore this imbalance. We decided to train our models once with imbalance and also by dealing with imbalance using under sampling technique. The empirical results for both are discussed in upcoming sections. 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, in order to avoid data leakage problem. Only then can these validation images be independent from the remaining training images of other drivers and thus avoid a falsely high test accuracy [6]. Four drivers were chosen at random and all of their images were separated as validation set.

The id's of the drivers that were selected are:

$$'p061','p035','p056','p052'$$

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:

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

Figure 3: Multi-class Logarithmic loss formula

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

Transfer learning

To speed up the learning process we used transfer learning specifically the following networks were used:

- VGG19
- RESNET50
- Xception
- Mobilenet

We trained each network for 40-50 epochs until the loss and accuracy plateaued(Converged) which we monitored by deploying early stopping with patience of 20 .

Xception

The architecture of Xception was kept intact except the head. Instead, we removed the head and attached a GlobalAveragePooling2D, to keep the total number of parameters low, followed by a DenseLayer with 1024 neurons, relu activation and a dropout of 0.1. This was followed by another dense layer of 1024 neurons, relu activation, dropout of 0.5 and then batchNormalization. Finally we attached another Dense layer 512 neurons with relu activation and then the output layer with softmax activation. Furthermore, we used Stochastic gradient descent with categorical_crossentropy as the loss function and Accuracy as the metric also Label smoothing of 5 x 10^{-5} was used. After fine-tuning the value of learning rate was set to 5 x 10^{-5} and momentum to 0.9. Dropout is used to induce a regularization effect after the global average pooling layer. Whereas the next layer has batch normalization to improve the performance. After training on both, dataset with class imbalance and under sampled dataset the validation log loss and accuracy are stated in the Figure 4 and 5.

ResNet50

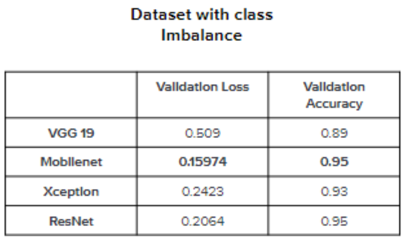
The Resnet50 architecture was kept intact just the head was replaced with our custom net. This net first flattens the resnet50 output which is an activation map. This is followed by two dense layers with 500 neurons and relu activation. Lastly we have the output layer with softmax activation. The Net uses Stochastic gradient descent with categorical_crossentropy as loss function and Accuracy as metric along with Label smoothing of 5 x 10^{-5} . Resnet50 already has batch Normalization after each block hence no additional normalization or regularization was required. Although we did experiment with the idea and no significant increase in accuracy was found.

MobileNet

The architecture of MobileNet was kept intact except the head. Instead, we removed the head and attached a GlobalAveragePooling2D, to keep the total number of parameters low, followed by a DenseLayer with 1024 neurons, relu activation and a dropout of 0.1. This was followed by another dense layer of 1024 neurons, relu activation, dropout of 0.5 and then batchNormalization. Finally we attached another Dense layer 512 neurons with relu activation and then the output layer with softmax activation. Furthermore, we used Stochastic gradient descent with categorical_crossentropy as the loss function and Accuracy as the metric also Label smoothing of 5 x 10^{-5} was used. Dropout is used to induce a regularization effect after the global average pooling layer. Whereas the next layer has batch normalization to improve the performance. MobileNet model performance without extra layers except for Global Average Pooling (GAP) layer was better than with extra layers attached.

VGG19

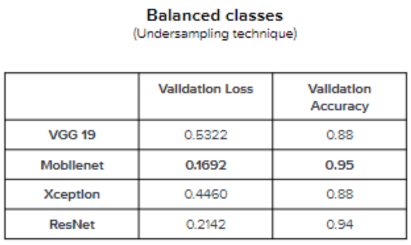
The architecture of VGG was kept intact except the head. Instead, we removed the head and attached a GlobalAveragePooling2D, to keep the total number of parameters low, followed by a DenseLayer with 1024 neurons, relu activation and a dropout of 0.1. This was followed by another dense layer of 1024 neurons, relu activation, dropout of 0.5 and then batchNormalization. Finally we attached another Dense layer 512 neurons with relu activation and then the output layer with softmax activation. Furthermore, we used Stochastic gradient descent with categorical_crossentropy as the loss function and Accuracy as the metric also Label smoothing of 5 x 10^{-5} was used. Dropout is used to induce a regularization effect after the global average pooling layer. Whereas the next layer has batch normalization to improve the performance.



Dataset with class Imbalance

	Validation Loss	Validation Accuracy
VGG 19	0.509	0.89
Mobilenet	**0.15974**	**0.95**
Xception	0.2423	0.93
ResNet	0.2064	0.95

Figure 4: Table showcasing validation log loss and accuracy values for the dataset with class imbalance. Values in bold show the best values that we achieved comparative to other models.



Balanced classes
(Undersampling technique)

	Validation Loss	Validation Accuracy
VGG 19	0.5322	0.88
Mobilenet	**0.1692**	**0.95**
Xception	0.4460	0.88
ResNet	0.2142	0.94

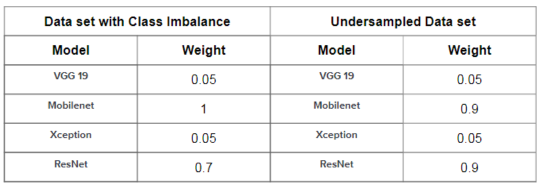
Figure 5: Table showcasing validation log loss and accuracy values for the under sampled dataset to balance images belonging to each category.

Network Ensemble

It was found that using an ensemble of different models yielded better results than using a single model as over fitting is a major concern in this problem. Thus, instead of relying on the predictions of one single model, we averaged the results of 4 of our models namely VGG-19, Xception, MobileNet and ResNet50 to get the final prediction values. 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 MobileNet model was 0.38585. We tried multiple network ensemble techniques to further improve the log loss score [5].

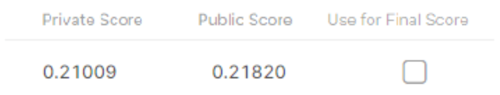
- **Mean Ensemble:** This is the easiest and most widely used ensemble method where the posterior probability is calculated as the mean of the predicted probabilities from the component models.
- **Trimmed Mean Ensemble:** This is Mean Ensemble by excluding the maximum and minimum probabilities from the component models for each image. It helps in further smoothing our predictions leading to a lower log loss value.
- **Weighted average Ensemble:** The models assigned weights when averaging their predictions, according to their performance during validation and these predictions were then added up and divided by the number of models used.

The ensemble created by averaging several models trained on both untouched and reduced datasets based on the priorities of each model given on the basis of their performance on validation set (As shown in Figure 6) proved to give least LB (log loss) score of 0.21820, as shown in Figure 7.



Data set with Class Imbalance		Undersampled Data set	
Model	**Weight**	**Model**	**Weight**
VGG 19	0.05	VGG 19	0.05
Mobilenet	1	Mobilenet	0.9
Xception	0.05	Xception	0.05
ResNet	0.7	ResNet	0.9

Figure 6: Weights for each model based on their performance on validation set



Private Score	Public Score	Use for Final Score
0.21009	0.21820	☐

Figure 7: Log loss scores after Ensemble

Temporal Context

Since the images are taken from a video clip, there exist many similar pictures which should belong to the same categories. Based on this premise, finding similar images and averaging the probabilities over these images helped us smoothen predicted probabilities for each class. To find the 10 nearest neighbors, we used outputs from the penultimate layer of VGG16 transfer learning model as features on the test set [5].

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 100 × 100 to shorten the computation tie. To find these K neighbors, K Nearest-neighbor classification based on K Dimensional-tree is used as KNN is computationally expensive as compared to other methods. A K-D Tree(also called as K-Dimensional Tree) is a binary search tree where data in each node is a K-Dimensional point in space. It is a space partitioning data structure for organizing points in a K-Dimensional space [7]. Figure. 8, as an example, shows one original test image and its 9 most similar images. We then replace the predicted probability of the test image with the average probabilities of all of its K images. It took about *3 hours and 56 minutes* to average over NN predictions for each 79726 test samples. Since K-D trees can be queried, we were able to run the process in parallel to reduce time it took to generate final predictions, as compared to 16 hours if we used KNN. This implementation further lowered the log loss to about 20% of the log loss we acquired after network ensemble.

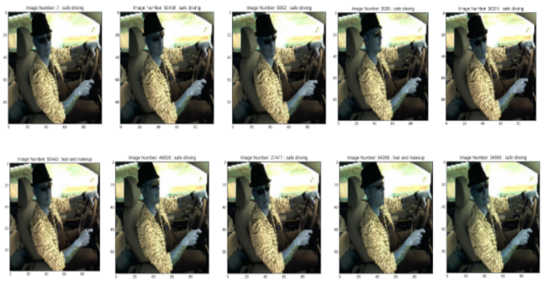
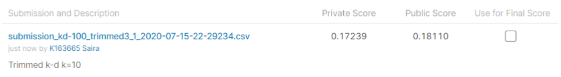


Image Number: 7 : safe driving
Image Number: 50458 : safe driving
Image Number: 9002 : safe driving
Image Number: 3535 : safe driving
Image Number: 56374 : safe driving
Image Number: 50443 : hair and makeup
Image Number: 46818 : safe driving
Image Number: 27477 : safe driving
Image Number: 64266 : hair and makeup
Image Number: 34565 : safe driving

Figure 8: Output of K-d tree

Results

After exploring various CNN models combined with network ensemble and KNN, our best performing results were achieved by calculating the weighted trimmed average of models predictions and finally calculated trimmed average for predictions across its K Nearest Neighbors with (K = 10). The final LB score (log loss) we got is 0.18110 which ranked 40 among 1438 total participants (top 2%) on Kaggle's public leaderboard (shown in Fig. 4).



Submission and Description	Private Score	Public Score	Use for Final Score
submission_kd-100_trimmed3_1_2020-07-15-22-29234.csv just now by K163665 Saira Trimmed k-d k=10	0.17239	0.18110	☐

Figure 9: Final submission log loss score

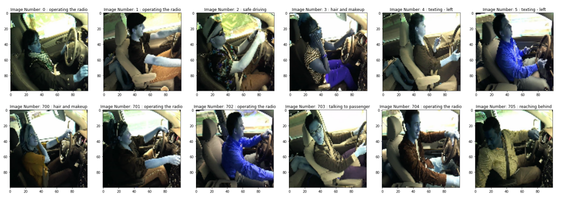


Image Number: 0 : operating the radio
Image Number: 1 : operating the radio
Image Number: 2 : safe driving
Image Number: 3 : hair and makeup
Image Number: 4 : texting - left
Image Number: 5 : texting - left
Image Number: 700 : hair and makeup
Image Number: 701 : operating the radio
Image Number: 702 : operating the radio
Image Number: 703 : talking to passenger
Image Number: 704 : operating the radio
Image Number: 705 : reaching behind

Figure 10: Predictions on test set

Conclusion/Future Work

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 (even though it was an inactive challenge) 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. KNN while training. 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 [3].

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.

References

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

[2] Kaggle. State Farm Distracted Driver Detection - Overview, `https://www.kaggle.com/c/state-farm-distracted-driver-detection/overview`

[3] Kaggle. 3 BR POWER - Solution `https://www.kaggle.com/c/state-farm-distracted-driver-detection/discussion/22631`

[4] Kaggle. A brief summary by Jacobie (Winner of this competition) `https://www.kaggle.com/c/state-farm-distracted-driver-detection/forums/t/22906/a-brief-summary`

[5] Satya Naren Pachigolla. Distracted Driver Detection using Deep Learning, December 2019, `https://towardsdatascience.com/distracted-driver-detection\-using-deep-learning-e893715e02a4`

[6] Samuel Colbran,et al. Classification of Driver Distraction, `http://cs229.stanford.edu/proj2016/report/SamCenLuo-\ClassificationOfDriverDistraction-report.pdf`

[7] Wikipedia. K-dimensional tree, `https://en.wikipedia.org/wiki/K-d_tree`