Machine Learning Capstone Project Report

Distracted Driver Detection

Project Overview

Distracted driving is a growing public safety hazard. The dramatic rise in texting and constant updates from social media platforms appear to be contributing to an alarming increase in distracted driving fatalities. In this project, we will classify a dataset of images of drivers at the wheel to identify distracted drivers and categorize them based on the type of distraction using CNNs.

Problem

The problem we are trying to solve is a multi-class classification problem. The model built should correctly classify driver's behaviour from an image of the driver at the wheel. The model will be trained using a dataset of images of drivers taken using a dashboard camera. The dataset used, "State Farm Distracted Driver Detection," is provided by State Farm and it is published on the Kaggle website.

https://www.kaggle.com/c/state-farm-distracted-driver-detection/

Each image in the dataset is classified into one of the following ten classes.
c0: safe driving,
1: 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.

We will predict the likelihood of what the driver is doing in each picture. For each image in the test set, the probability of the image belonging to each of the classes is calculated. Since images in the test set are not labelled we will calculate the performance of the model by submitting the predictions on test data to Kaggle’s State Farm Distracted Driver challenge.
https://www.kaggle.com/c/state-farm-distracted-driver-detection/submit
Submissions are evaluated using the multi-class logarithmic loss. Each image has been labelled with one true class. For each image, the submission file includes a set of predicted probabilities, one for every image. The formula is then,

$$F = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{M} y_{ij} \cdot Ln(p_{ij})) = \sum_{j}^{M}\left(-\frac{1}{N}\sum_{i}^{N} y_{ij} \cdot Ln(p_{ij}))\right) = \sum_{j}^{M} F_i$$

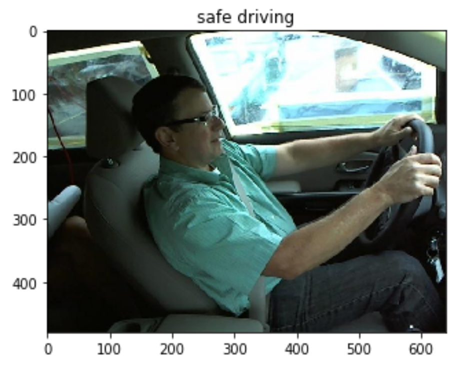
where N is the number of instances, M is the number of different labels, yij is the binary variable with the expected labels and pij is the classification probability output by the classifier for the i-instance and the j-label.

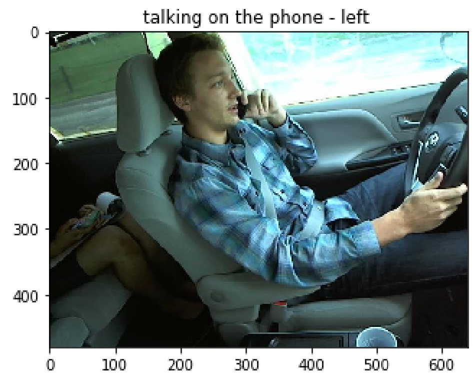
Metrics

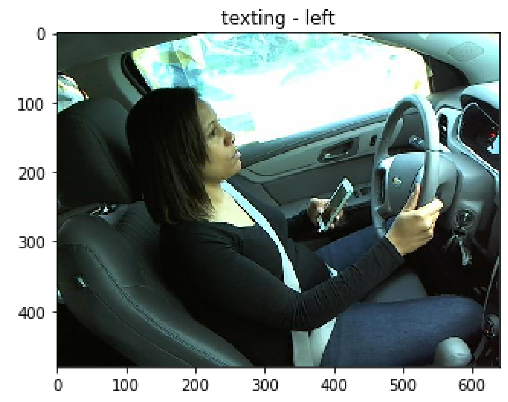
I used Multi-class logarithmic loss as the metric to measure the performance of the model. This metric measures the accuracy of a classifier by penalizing false classifications which makes it a good metric for Computer Vision classification problem. Because, to calculate log-loss, the classifier must assign a probability to each class rather than outputting the most likely class. Log loss ensures that we are generalizing all categories well.

Data Exploration

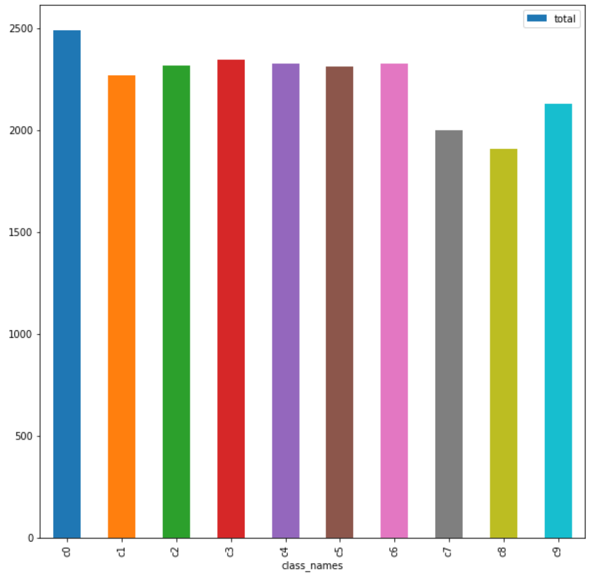
The dataset consists of images of the driver captured from a side-view dashboard camera. Here are few sample images from the dataset.







All the images have a shape of 640 x 480 x 3.
The train set has 26 unique drivers and a total of 22424 training images that belong to 10 different classes with the following distributions per class.



Some categories have more images than the others but the difference is not significant.

The test dataset has 79726 images. These images are not labelled. To calculate the log loss on test dataset, the metric used to the performance of the model for this project, I made submissions on the Kaggle website. Additional details on data exploration that is done for this project is documented in Data_Expolaration.ipynb

Exploratory Visualization

Each driver data is distributed across all the available classes for all the drivers in the train set.

All the images posted here belong to the same class c0: Safe driving behaviour. From these examples, you can see that an image could be classified as safe driving even when the driver is not holding the wheel with both hands.

Also, Images vary in angle, the distance of the camera from the driver, lighting etc. These differences will help the model to generalize better.

Algorithms and Techniques

I used Python and Keras framework to implement this project and trained it on an AWS EC2 instance.

I used Convolutional Neural Networks(CNN) to classify the images. Convolutional Neural Networks are a category of Neural Networks that have proven to be very effective in areas such as image recognition and classification. CNNs have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self-driving cars.

For spatial data like images where most features are localized Convolution neural networks work well because CNN applies small filters to explore and identify features of the dataset. Convolutional layers are not fully connected which reduces the number of trainable parameters significantly and allows CNNs to use many filters to extract interesting features.

I applied transfer learning on the pertained models available in Keras Application model. The architectures of these models have been carefully selected and the pre-trained weights for

Imagenet are obtained by training on million images. Making use of these trained weights and architectures will drastically reduce the training time.

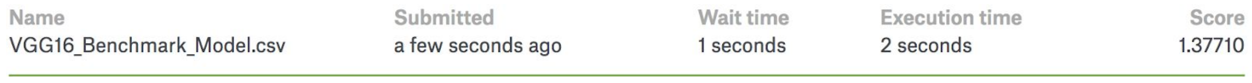
I used VGG16 as the benchmark model. The VGGNet has a simple layout of basic conv nets: a series of convolutional, max-pooling, and activation layers before some fully-connected classification layers at the end.

To improve on the benchmark model, I trained the dataset on VGG19, ResNet50, InceptionV3, InceptionResNetV2 networks. I expected ResNet50, InceptionV3 model to perform better than the benchmark model because of their better architecture and higher performance on imageNet. But they did not work well for this dataset. So I tried VGG19 which is an improvement on VGG16 and got good results. So I fine-tuned the VGG19 model and got an improved score of 1.20704 after training on 50 epochs. This involved initializing a model with pre-trained weights from VGG19 and fine-tuning the last few layers on a new dataset while tweaking hyperparameters.

Benchmark

I used VGG16's bottleneck features as the benchmark model.
This involved initializing a model with pre-trained weights from VGG16 adding a Dense layer with 10 outputs one for each class, training the model for few epochs and using it as a feature extractor.

The test images are not labelled. So I created a submission file with the predictions from the test set and got a score of 1.3770. Submissions file is included in the Submission folder on the Github project.



Name	Submitted	Wait time	Execution time	Score
VGG16_Benchmark_Model.csv	a few seconds ago	1 seconds	2 seconds	1.37710

The result is calculated using multi-class logarithmic loss, log loss value of the predicted labels against the actual labels of the 79726 test images.

Methodology

Data Preprocessing

The following preprocessing steps were applied to the data:

- Resize images to 100 x 100 to make it easier to load into memory and train in a reasonable amount of time.
- Shuffle images to change the default order.
- Normalize pixel values of the image by diving all the values in the image array by 255 which is the maximum RGB value and reduces our data to be within the range of 0 and 1
- Data augmentation to improve generalization like Zoom, height shift, width shift and rotation.

Implementation

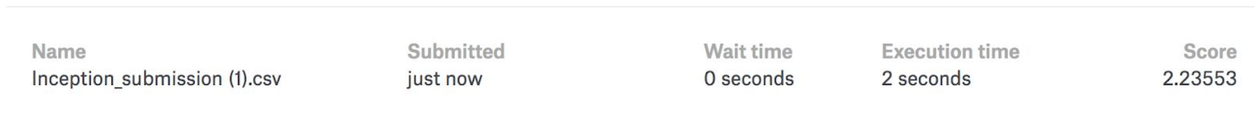
I followed the following steps to implement the model.

1. Data loading and visualization: Load the data and get familiar with the data and distribution amongst categories using visualizations.
All the data exploration steps are documented in IPython notebook, “Data_Expolaration.ipynb”
2. Create a benchmark model using pre-trained VGG16 by adding a fully-connected softmax layer with ten outputs for classification.
The steps to train and test this model are documented in “VGGNet.ipynb” I saved the model in h5 format to load and compare results with other models.
3. Train models using fine-tuned InceptionV3, VGG19, ResNet50 and InceptionResNetV2 models pre-trained on Imagenet data, compare it with the benchmark model and choose the model that performs best on the training set as the feature encoder.
5. Enhance the results of the model chosen using hyperparameter tuning and data augmentation.
Final Model is documented in “Final Model - VGG19.ipynb ”

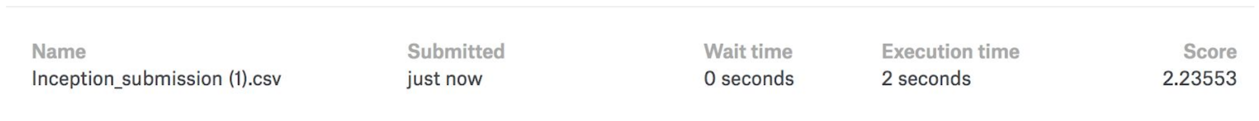
For all the models training was a time-consuming process and I had to make tweaks to batch size and image shape to optimize memory usage and time to train.

Refinement

I started off with VGG16 as the benchmark model. I trained and tested the model and got a log loss score of 1.37710 when submitted to the Kaggle submission page. I expected InceptionV3 and ResNetV2 to perform better than VGG16. But Inception's log loss of 2.23553 and ResNet50’s score of 2.78177 on test data. Inception models and ResNetV2 are deeper architecture, and for this dataset, these modes are overfitting and not generalizing well even after tweaking the hyperparameters and adding dropout layers.



Name	Submitted	Wait time	Execution time	Score
Inception_submission (1).csv	just now	0 seconds	2 seconds	2.23553



Name	Submitted	Wait time	Execution time	Score
Inception_submission (1).csv	just now	0 seconds	2 seconds	2.23553

I trained on multiple models multiple models ResNet50, InceptionV3, InceptionResNetV2 and VGG19.

VGG19 performed the best of all, so I fine-tuned the VGG19 model to improve the performance of the model. To further refine the model, I had to tweak the optimizers used, learning rate, momentum and batch size. In addition to hyperparameter tuning, another refinement made was to use Early stopping technique. This technique reduced the time it took to train when there are no improvements in validation loss.

I started off with VGG16 as the benchmark model. I trained and tested the model and got a log loss score of 1.37710 when submitted to the Kaggle submission page.

The final model is an improved VGG19 model for this dataset with a log loss score of 1.20704 on test data.

Name	Submitted	Wait time	Execution time	Score
VGG19_submission.csv	just now	0 seconds	2 seconds	1.20704

Results and Discussion

The final solution for this project is a Convolutional Neural Network-based system for distracted driver detection. The pre-trained ImageNet model is used for weight initialization and the concept of transfer learning is applied. Weights of all the layers of the network are updated with respect to the dataset. After rigorous experimentation, all the hyperparameters are finetuned. The training is carried out using Stochastic Gradient Descent with a learning rate of XXXX, the decay rate of 0.0009 and momentum value 0.9. The batch size and number of epochs are set to 32 and 50 respectively. Training and testing are carried out using GPU instance on AWS(p2.xlarge with Ubuntu 16.04+k80 GPU). The framework used to build the project is Keras.

On VGG-16 model, the Benchmark model a log loss score of XXXXX on the validation set and log loss score of 1.37710 for the Kaggle submission on the test set.

Performance of the system has significantly improved with the fine-tuned VGG19 model which resulted in log loss score of 1.20704 on the test set.

Name	Submitted	Wait time	Execution time	Score
VGG19_submission.csv	just now	0 seconds	2 seconds	1.20704

Conclusion and Future Work

Driver distraction is a serious problem leading to a large number of road crashes worldwide. Hence detection of the distracted driver becomes an essential system component until we achieve level 5, complete automation, with self-driving cars. Here, I built a robust Convolutional Neural Network based system to detect distracted driver and also identify the

cause of distraction. We modified the VGG19 architecture for this particular task which has several regularization techniques to prevent overfitting to the training data. With the log loss score of 1.20704, this proposed system outperforms the benchmark approach of distracted driver detection by a significant amount.
As an extension of this work, we can create an ensemble model to improve the accuracy further. We can also incorporate a temporal context that may help in reducing misclassification errors and thereby increasing accuracy.