UDACITY Capstone Proposal ---

State Farm Distracted Driver Detection
Can computer vision spot distracted drivers?

Bang-Sin Dai
May 4, 2018

Domain Background
Driving safety is an important issue. Many traffic accidents are caused by improper or distracted driving behaviors. The drivers may operate their mobile phones, talks to passengers, pick up some things around them and even drink beverages during driving. These inadvertent dangerous driving behaviors often cause the loss of not only the driver's live but also other people's lives. Many people get different levels of body hurts due to such traffic accidents.

According to the CDC motor vehicle safety division (https://www.cdc.gov/motorvehiclesafety/distracted_driving/), 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 also to reduce the traffic accidents and better insure their customers, by testing whether dashboard cameras can automatically detect drivers engaging in distracted behaviors (https://www.kaggle.com/c/state-farm-distracted-driver-detection#description).

Problem Statement
Given a data set of 2D dashboard camera images, State Farm (https://www.statefarm.com/) is challenging data analysts and ML engineers to classify each driver's behavior. For example, are they driving attentively, wearing their seat-belt, or taking a selfie with their friends in the backseat? In other words, by using technology of computer vision or image recognition, can we detect and highlight these dangerous and distracted driving behaviors? Formally speaking, given an image of a driver on the seat, our algorithm requires to identify which one of the ten labels this image belongs to with the highest possibility. If we can design an algorithm (using deep learning technique like CNN model for example) to identify these unsafe driving behaviors in a relatively high classification accuracy automatically and then alarm the dangerous drivers in real time to stop their unsafe behaviors, the objective of the problem can be regarded as to be achieved successfully.

Data Sets and Inputs
The input data is a set of 2D driver images (https://www.kaggle.com/c/state-farm-distracted-driver-detection/data), each taken in a car from a dashboard camera with a driver doing something in this car, for example, texting, eating, talking on the phone, makeup, reaching behind, etc. Each image is in one of the ten classes listed below, and our mission in brief is to identify (the likelihood of) what the driver in each image is doing**.** The data provider is State Farm. State Farm set up these experiments in a controlled environment - a truck dragging the car around on the streets - so these "drivers" were not really driving. The data images include training and testing sets. We can use training set to build the ML model (to construct a CNN: convolution neural network for example) and do validation and use the testing set for prediction quality test.

The 10 classes (labels) of driver images are:

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		

Note that the metadata such as creation dates is removed by State Farm. Besides, the training and testing data are split on the drivers, so that one driver can only appear on either training or testing set. The data provider has supplemented the testing data set with some images that are resized, and these processed images are ignored.

Data files and the corresponding descriptions can be found in website (https://www.kaggle.com/c/state-farm-distracted-driver-detection/data) :
imgs.zip - zipped folder of all (train/test) images
sample_submission.csv - a sample submission file in the correct format
driver_imgs_list.csv - a list of training images, their subject (driver) id, and class id

Solution Statement

We can use the deep learning technique like the CNN (convolution neural network) model, which is quite appropriate to deal with problems in the fields of computer vision and image/pattern recognition. Our problem here is a multi-classes (ten classes) image classification problem. The solution we provide needs to identify the objects (drivers) in the images and try the best to classify what they are doing. So a typical approach to handle this issue is to build a CNN model. The CNN model can be implemented by Keras and Tensorflow. We can train this CNN model by using training set and do validation by the validation set. Then we can test the model prediction quality, e.g., classification accuracy or the score of "evaluation metrics" mentioned below, by using testing set. In addition, we can also leverage the technique of transfer learning. That is, we can use good CNN models, which are trained by others before, as initial parameter settings or a part of our CNN model on this problem, without training or fine-tuning the whole network parameters by ourselves completely. Transfer learning is expected to save much training time and have better quality.

Benchmark Model

In public leader board of Kaggle competition on this problem (https://www.kaggle.com/c/state-farm-distracted-driver-detection/leaderboard), there are many models (1440 in total) constructed by different teams on this problem. The top one model has the score (see "evaluation metrics") of 0.08689. For simplicity and convenience, we can choose the median of the 1440 scores of all the models, i.e., 1.514365, as the benchmark model to be compared with our model.

Evaluation Metrics

We adopt the same evaluation metrics as that in Kaggle competition on this problem. Please see the website (https://www.kaggle.com/c/state-farm-distracted-driver-detection#evaluation) for a reference. The error function to evaluate the performance scores of models is the (multi-class) logarithmic loss function:

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

where *N* is the number of images in the test set, *M* is the number of image class labels, *log* is the natural logarithm, y_{ij} is 1 if observation *i* belongs to class *j* and 0 otherwise, and p_{ij} is the predicted probability that observation *i* belongs to class *j*.

Error function needs to properly represent the "distance" between prediction results and actual results. That is, the error function serves as a clear performance measuring metric, and we want to minimize (optimize) the error function to obtain best possible performance (or say maximum likelihood).

Log-loss error function has nice properties of differentiable and continuous, hence it is suitable to be applied gradient descent on it to improve the model performance iteratively. Since probability is the major tool used to evaluate and improve the model, the cross entropy form of the log-loss function appropriately calculates the likelihood of a prediction model given the overall actual results in a concrete way, and that is the most important key for log-loss function to become a proper (and also popular) evaluation metric.

Project Design

The outlines of our approach workflow are as follows.

- Import data sets
First we import the "train" and "test" data sets, which are provided by State Farm. Then we split the "train" image data set into the training set and the validation set. We will print out the number of images in each data set.

- Data analysis and preprocessing
First we will explore the image data set. Since there are ten classes, we will check the number of images in each class and whether the data set is roughly balanced in size on different labels. Then we will preprocess the images by converting each image to 3D tensor with (224, 224, 3) shape, i.e., 224 x 224 pixels with three channels of '*r*', '*g*', and '*b*'. We will re-scale the images by dividing every pixel in every image by 255.

- Create a CNN to classify driver images (from scratch)
We will use Keras and Tensorflow to implement our CNN model. In this step, we will provide the first architecture of the CNN model we design. And then test the performance result of the first CNN model.

- Use a CNN to classify driver images (using transfer learning)
To reduce training time without sacrificing accuracy, we will train a CNN model using transfer learning. In this step, our CNN model will use the pre-trained VGG-16 model as a fixed feature extractor, where the last convolutional output of VGG-16 is fed as input to our model.

- Create a CNN to classify driver images (using transfer learning)
In this step, instead of VGG-16, we may try to use the other pre-trained model, which is like Xception or ResNet-50 or Inception, for different potential choices. We can compare this CNN model with the above one and check the difference of prediction scores of them.

- Algorithm test result
We will choose the best result (the lowest metric score) among the CNN models we construct above to be the final output of our proposed algorithm.