Machine Learning Engineer Nanodegree

Capstone Proposal

Jordan Milushev
October 7th, 2018

Proposal

Domain Background

Distracted driving is one of the leading causes of auto accidents in the US today. According to NHTSA, in 2016 alone, there were 3,450 lives lost due to distracted driving (https://www.nhtsa.gov/risky-driving/distracted-driving). Distracted driving may be caused by many factors, including but not limited to conversations with passengers in the car, conversation on the phone, adjusting radio, navigation or other controls in the car, eating, texting, checking e-mail, drowsiness, etc. Passengers should always remind the driver to keep their eyes on the road and hands on the wheel. A lot of times, drivers are alone in the car, and in the case of teenager drivers, having one or more teenagers in the car may further increase the teenage driver distraction.

Problem Statement

The accidents caused by distracted driving can easily be reduced or eliminated altogether if this behavior can be minimized. Usually, every one of us knows the risk associated with distractions during driving but still engage in this behavior without even noticing. If there was a way for the environment in the car to change when the driver becomes distracted, it could serve as a reminder to the driver to correct himself or herself. For example, if the driver becomes distracted, the radio could start playing or turn its volume up. If the driver starts texting, there could be a sound signal or voice warning played. In addition, insurance companies could reward good behavior by lowering premiums for drivers who do not engage in distracted driving. This project will propose a solution by creating an algorithm that could process frames from a dash board camera and determine if a driver is engaging in some kind of distracted driving or not. The idea is to analyze the driving behavior in real time, and when unwanted behavior is detected, signal a warning appropriately. The latter part is not part of the proposed solution, which only addresses the frame analysis part.

Datasets and Inputs

The data for the solution is used from a Kaggle competition (https://www.kaggle.com/c/state-farm-distracted-driver-detection/data). The data is not provided for public use and can only be used for the purpose of the competition. It can be obtained by logging into the Kaggle web-site, accepting the rules of the competition, after which the data can be downloaded.

The dataset consists of driver images taken from a dashboard camera divided into a training (22,424 images) and test (79,726 images) set. The images are categorized in one of the following categories:

- 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

The train and test data are split on the drivers such that one driver can appear in only one of the data sets.

In addition there is a file driver_imgs_list,csv, which associates an image from the training set with its class name and driver.

Solution Statement

The solution would be able to accept image frames of the driver from a dashboard camera, pass it through the model, which would output the state of the driver which would be one of the 10 categories described above. This can be later input to another system, which will act upon this information. The downstream system is not part of this solution.

The model will be trained and validated with the dataset provided. For that purpose, the provided training set will be split into two 80% / 20%. (training / validation) sets. The performance of the model will be determined with the help of the validation data set.

Benchmark Model

The proposed solution is essentially a solution for an existing Kaggle competition from 2 years ago. Because of that the competition leaderboard provides a great benchmark for the purpose of this project. The closer the score from the created model in this solution is to the leader on the leader board, the better. In order to be able to compare the results on the other side of the scale, a score of the prediction on the test dataset using uniform random probabilities will be utilized. The score achieved by the solution in this project needs to be better than the one achieved by the uniform random prediction. Once this is achieved, the score will be rated on how close it is to the leader.

Evaluation Metrics

Models will be evaluated by submitting the test dataset predictions to the Kaggle competition, which will calculate the multi-class logarithmic loss.

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} Yi \log(Pij),$$

Where N is the number of images in the test set, M is the number of image class labels (10), 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

Project Design

This is an image classification project, solutions for which in the last 5-7 years have proven that the application of deep learning can result in high accuracy classifications. With the help of the ImageNet competition, multiple architectures have been created in the last 5-7 years, which have been reducing the classification error while improving the training speed.

Today, Keras provides easy access to pre-trained algorithms on the ImageNet set (https://keras.io/applications/):

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
ResNet50	99 MB	0.749	0.921	25,636,712	168
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

The ImageNet competition is a multi-class classification, which gives a probability for an image to be classified as each one of the classes. If the highest predicted probability for a category happens to be what the image category truly is, then it is considered Top-1 Accuracy. If, the true category is in the top 5 predicted classes, then it is considered a Top-5 Accuracy.

The table also contains the number of parameters, number of hidden layers and size of the model. These characteristics of the model are indicative of the computational complexity.

Models with higher accuracy and lower computational complexity are preferable.

The idea for the proposed solution is to select one of the above pre-trained models and optimize for the image set described for this problem. For the sake of time, only 3 models will be examined based on their accuracy and computational requiremets:

- Xception
- InceptionResNetV2
- MobileNetV2

The workflow is as follows:

1. Pre-process the input data.

The expected shape of the input image data by the Keras models is
(nb_samples, rows, columns, channels),
where
nb_samples – number of samples
rows – height of the image in pixels
columns – width of image in pixels
channels – each corresponds to an RGB color
the ImageNet dataset is comprised of color images with the size 224 x 224. In order to use the pre-trained Keras models, each image from this project will first be converted to a tensor with size (1, 224, 224, 3)

2. Separate the training data set into training and validation dataset using an 80/20% ratio, keeping individual drivers in just one of the datasets in order to reduce cross-training between the training and validation datasets.

3. Augment the training dataset, in order to increase its size, which are quite small and could lead to over-fitting.

4. Create the architecture of the new model
   In this solution, pre-trained models will be used without the top (fully connected layer at the top of the network). Different architectures will be considered, i.e. additional convolutional layers and/or additional pooling layers, fully connected layers, etc.

5. Choose activation functions, optimizer and loss function

6. Evaluate regularization technics such as batch normalization, dropout layers, etc.

7. Train the model using the dataset.
   Utilize early stopping of the training process. Parameters of the added on layers to the pre-trained models will be considered for tuning. Depending on the chosen architecture, some or all of the following parameters will be considered for tuning:
   - learning rate
   - dropout rate
   - batch size
   - batch normalization momentum
   - activation function
   - number of added layers and units
   - image augmentation parameters, etc.
   - kernel optimizer
   - loss function

8. Once the optimal model is trained on the training set, use the test dataset to make a prediction for the images in it. Prepare the submission file and submit to the Kaggle competition in order to get back the score
9. Compare the score with the benchmark score