**Capstone: State Farm Distracted Driver Detection**

Natraj Rajput January 17, 2018

1. **Definition**
   1. **Project Overview**

The domain of this problem is computer vision. Computer vision is a branch of machine learning concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images (BMVA, n.d.). Computer vision began in the 1960’s, when a person named Larry Roberts wrote his PhD thesis on the possibility of extracting 3D details and information from 2D images(T.S. Huang, n.d.). In the 70’s, some progress was made on the interpretation of 2d images to 3d images (Hari Narayanan, et al, n.d.). In the 80’s, optical character recognition systems that recognize letters, symbols and numbers were used in several industries (Quick history, n.d.). In the 90’s, new applications of computer vision were possible as computers became more powerful and common (Quick history, n.d.). In the 2000’s, computer vision was used to process large datasets, videos and could understand motion, patterns and predict outcomes (Hari Narayanan, et al, n.d.). The [dataset](State%20Farm%20Distracted%20Driver%20Dataset.%20(2016).%20In%20Kaggle.%20Retrieved%20October%2025,%202017,%20from%20https:/www.kaggle.com/c/state-farm-distracted-driver-detection/data) being considered is the one provided in the Kaggle competition. It contains 2 folders, one with the training images and the other with the test images. The images capture the driver from a side-view dashboard camera.

* 1. **Problem Statement**

The problem that we are trying to solve is a multi-class classification problem. We are tasked to properly predict and classify driver’s behavior given the dashboard images of people doing 10 different actions, 9 of which are considered actions of distracted behavior. The 10 classes are as follows: 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.

A solution to this problem is using machine learning computer vision models to classify driver actions. Working with Convolutional Neural Networks would be a good idea because CNN’s are known to yield the most accurate results in the computer vision field. Keras application models with pre-trained weights could reduce the time it takes to train while still maintaining good results. Reducing image size and dividing the images into the RGB channels could make processing of the images more manageable. Pre-processing the images may be necessary to reduce overfitting and improve generalization. Once the model is fit, we will need to predict the labels of the test set to determine which of 10 categories each picture belongs to. The validation results and benchmark result will then tell us if we still need to improve the model. We are trying to achieve a log loss that would be closest to zero and should at least be within the top 10% of the Kaggle public leaderboard.

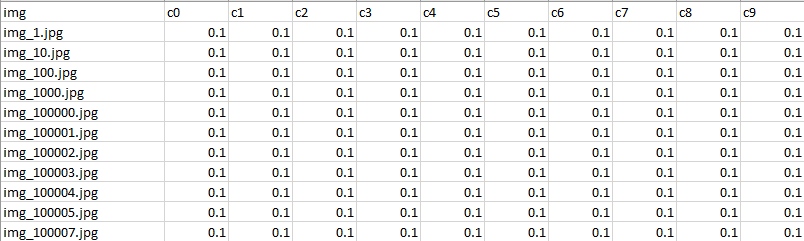
An outlined solution is, Step 1, import the training data. Step 2, split the training data into 2 subsets, a train subset and a validation subset. Step 3, Reduce and scale images from the dataset to a more manageable size. Step 4, augment the dataset by adding images with noise. Step 5, make the CNN architecture but most optimally make use of the keras application and pre-trained model weights such as Xception, VGG-16, Resnet50. Step 6, fit the model and save the resulted weights. Step 7, test using the validation subset. Step 8, validate results/accuracy. Step 9, Plot the train and test dataset against the fitted values/epoch to see how much the model is overfitting. Step 10, Predict the labels of the test dataset then submit it to Kaggle. Step 11, compare Kaggle result to see if it meets benchmark results. Step 11, Adjust architecture, model, parameters and data augmentation. Step 12, Repeat until it meets target benchmark results. Step 13, make final adjustments.

* 1. **Metrics**

I will make use of the Multi-class logarithmic loss. This metric is used in many computer vision classification problems because it measures the accuracy of a classifier by penalizing false classifications. It is also a good metric for this problem because in order to calculate log-loss, the classifier must assign a probability to each class rather than yielding the most likely class. Having a good log loss would mean we are generalizing all categories well and not only favoring and generalizing few categories. Since we also need to compare our results to a benchmark, it would be best to stick with the metric made use in the Kaggle competition.

-log P(yt|yp) = -(yt log(yp) + (1 - yt) log(1 - yp))

Sample output:



Multi class log loss example:

**>**LogLossMulti (["bam", "ham", "spam"], [[1, 0, 0], [0, 1, 0], [0, 0, 1]])

[1] 2.1094237467877998e-15

**>**LogLossMulti (["bam", "ham", "spam"], [[0, 0, 1], [1, 0, 0], [0, 1, 0]])

[1] 34.538776394910684

**>**LogLossMulti (["bam", "ham", "spam", "spam"], [[0.8, 0.1, 0.1], [0.3, 0.6, 0.1], [0.15, 0.15, 0.7], [0.05, 0.05, 0.9]])

[1] 0.2990

***Retrieved from Mark Needham,first steps with log loss***<http://www.markhneedham.com/blog/2016/09/14/scikit-learn-first-steps-with-log_loss/>

From this example, we can see that when the prediction is completely the same as the actual value, the log loss results to a 2.11e-15 and when the prediction is completely wrong, the log loss reaches 34.54.

1. **Analysis(2-4 pages)**
   1. **Data Exploration**

There are a total of 22424 images in the train dataset and 79726 images for the test dataset. All images are colored and 640 x 480 in size. The images capture the driver from a side-view dashboard camera.

2489,2267,2317,2346,2326,2312,2325,2002,1911,2129

Sample image:



* 1. **Exploratory Visualization**
  2. **Algorithms and Techniques**

The main model architecture that will be used is a Keras Application Model. These architectures have been perfected to classify images. Keras also has the option of making use of pretrained weights based of **ImageNet**. Making use of these pretrained weights can drastically reduce the time it will take to train and optimize the model.

The algorithms and techniques I intend to use and update are:

#REMOVE THOSE THAT WERE NOT PLANNED

* + 1. Keras Application Models
       1. Model -This is the main architecture that will be used to train our images. Architectures that I will try are VGG16, VGG19, ResNet50, Xception, Inceptionv3, InceptionResNetv2, MobileNet
       2. Weights- Random initialization or pretrained on ‘ImageNet’
    2. Neural Network Architecture
       1. Number of layers
       2. Layer types – Include Core, Pooling, Convolutional, Normalization and Noise layers
       3. Layer Parameters
    3. Preprocessing Parameters (see the Data Preprocessing section)
    4. Training Parameters
       1. Epochs- Training length
       2. Batch size- Number of tensors/images to be trained per epoch.
       3. Activation- The optimization algorithm to use for learning. Activation functions include softmax, elu, selu, softplus,softsign, relu, tanh, sigmoid, hard sigmoid and linear.
       4. Learning Rate- The learning speed of the algorithm
       5. Weight Decay- Regularization method used to prevent weights from growing too large
       6. Momentum- Technique used for accelerating gradient descent by making use of accumulated velocity.
    5. Callbacks
       1. Early Stopping- Technique used for reducing the time it takes to train when loss is not improving for a certain number of epochs.
       2. Model Checkpoint- Technique used for saving important weights.
    6. Cross Validation Techniques
       1. Kfold cross validation – technique used to train and evaluate models by randomly partitioning original samples into k equal sized subsamples. Each subsample is divided into a train and validation dataset.
    7. Other techniques
       1. Mean/Averaging Predictions- technique used to improve generalization by averaging out multiple predictions.

* 1. **Benchmark**

The benchmark result will be based on the Kaggle public leaderboard as I was unable to find a benchmark model as such. Since everyone in the leaderboard must follow the same rules and evaluation metric, it makes it good for benchmarking. My target result for this project is to reach the top 10% (≤ 144 of 1440) people with a logloss ≤ 0.24859. The target result is based on the log loss value of the predicted labels against the actual labels of the 79726 test images.

1. **Methodology(3-5 pages)**
   1. **Data Preprocessing**

The data preprocessing done are as follows:

1. Images are converted into 3 channels, RGB – This preprocessing is done so models may use the 3 channels to learn features with the objective of improving accuracy and log loss.
2. Images are resized to 224 x 224 – Resizing images makes it easier to load on memory at the cost of losing some details.
3. Image labels are converted to categorical integer features/vectors – This is done using the one-hot scheme. This encoding is needed for feeding categorical data to our models because it is the most practical way for models to read categorical data.
4. The list of Images is shuffled– Randomizing images is simply done to change the default order.
5. The list of images are divided into a train set and validation set - This division is important so that we could validate if our model is improving or not when it is training.
6. Pixel values are converted to 32 bit floats – This is done so we could rescale our images
7. Pixel values are divided by 255 – We divide our data by 255 because it is the maximum RGB value and we want our data to be within the range of 0 and 1.
8. Some images are randomly augmented using the following:
   * + 1. Random – Adding a touch of chance is known to improve accuracy and reduce overfitting in deep learning. This can be seen in everything from random weight initializations of models to dropout layers.
       2. Augmentations - Since not all dashboard images are taken in exactly the same spot, these augmentations provide a means of improving generalization.
          1. Zoom
          2. Width Shift
          3. Height Shift
          4. Rotation

Sample Output:

Step 1, import the training data. Step 2, split the training data into 2 subsets, a train subset and a validation subset. Step 3, Reduce and scale images from the dataset to a more manageable size. Step 4, augment the dataset by adding images with noise. Step 5, make the CNN architecture but most optimally make use of the keras application and pre-trained model weights such as Xception, VGG-16, Resnet50. Step 6, fit the model and save the resulted weights. Step 7, test using the validation subset. Step 8, validate results/accuracy. Step 9, Plot the train and test dataset against the fitted values/epoch to see how much the model is overfitting. Step 10, Predict the labels of the test dataset then submit it to Kaggle. Step 11, compare Kaggle result to see if it meets benchmark results. Step 11, Adjust architecture, model, parameters and data augmentation. Step 12, Repeat until it meets target benchmark results. Step 13, make final adjustments.

* 1. **Implementation**

The initial implementation went as follows.

1. Import data
   * + 1. Implementation: Created a function that would read all images. One function for reading the train dataset and another function for reading the test dataset
       2. Complications: It was a very time consuming process
       3. Solution: Implement
2. Split data into train and validation subsets
   * + 1. Implementation: Split the dataset
       2. Complications:
3. Preprocess dataset
   * + 1. Implementation:
       2. Complications:
4. Create Model Architecture
   * + 1. Implementation: Created a function that would read all images and then immediately rescale them into desired size.
       2. Complications:
5. Train model
   * + 1. Implementation: Created a function that would read all images and then immediately rescale them into desired size.
       2. Complications:
6. Test data
   * + 1. Implementation: Created a function that would read all images and then immediately rescale them into desired size.
       2. Complications:

The final implementation

* 1. **Refinement**

There were several refinements I

1. **Results(2-3 pages)**
   1. **Model Evaluation and Validation**
   2. **Justification**
2. **Conclusion(1-2 pages)**
   1. **Free-form Visualization**
   2. **Reflection**
   3. **Improvement**

**Sources:**

Motor Vehicle. (n.d.). In Merriam Webster’s online dictionary. Retrieved October 25, 2017, from <https://www.merriam-webster.com/dictionary/motor%20vehicle>

Distracted Driving. (2017). In Centers for Disease Control and Prevention. Retrieved October 25, 2017, from <https://www.cdc.gov/motorvehiclesafety/distracted_driving/>

Road Traffic Injuries. (2017). In World Health Organization. Retrieved October 25, 2017, from <http://www.who.int/mediacentre/factsheets/fs358/en/>

Distracted driving global fact sheet. (n.d.). In U.S. Department of Transportation National Highway Traffic Safety Administration. Retrieved October 25, 2017, from <http://usdotblog.typepad.com/files/6983_distracteddrivingfs_5-17_v2.pdf>

Annual Global Road Crash Statistics. (n.d.). In Association for safe international road travel. Retrieved October 25, 2017, from <http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics>

Ward, Marry. (2007). In Offaly History. Retrieved October 25, 2017, from <http://www.offalyhistory.com/reading-resources/history/famous-offaly-people/mary-ward-1827-1869>

Automobile History. (2010). In History. Retrieved October 25, 2017, from <http://www.history.com/topics/automobiles>

Texting and Driving Accident Statistics. (n.d.). In Edgar Snyder. Retrieved October 25, 2017, from <https://www.edgarsnyder.com/car-accident/cause-of-accident/cell-phone/cell-phone-statistics.html>

Andrew B. Collier(2015)Making sense of Logarithmic Loss. In Exegetic. Retrieved Octover 26,2017 from <http://www.exegetic.biz/blog/2015/12/making-sense-logarithmic-loss/>

State Farm Distracted Driver Dataset. (2016). In Kaggle. Retrieved October 25, 2017, from <https://www.kaggle.com/c/state-farm-distracted-driver-detection/data>

Ritchie Ng. (n.d.). Evaluating a classification model. Retrieved October 30,2017, from <http://www.ritchieng.com/machine-learning-evaluate-classification-model/>

What is computer vision? (n.d.). In BMVA. Retrieved October 30,2017, from <http://www.bmva.org/visionoverview>

Quick History of Machine Vision. (n.d.). In EPIC systems. Retrieved October 30,2017, from <https://www.epicsysinc.com/blog/machine-vision-history>

Hari Narayanan, Libin Sun, Greg Yauney, et al. (n.d.). Introduction to computer vision. Retrieved October 30,2017, from <https://cs.brown.edu/courses/cs143/lectures/01.pdf>

T. S. Huang. (n.d.) Computer vision: Evolution and Promise. Retrieved October 30,2017, from <https://cds.cern.ch/record/400313/files/p21.pdf>