Sujeeth Shetty 03/10/2020

BUAN 6357 Advanced Business Analytics with R

Spring 2020 - Project Report

Distarcted Driver Detection

Executive Summary:

One efficient way to reduce car accidents is to prevent distracted driving, which is the act of driving while engaging in other activities such as texting, talking on the phone, etc. Activities of that nature distract the driver from paying attention to the road. These distractions in turn compromise the safety of the driver, passengers, bystanders and others in other vehicles. The United States Department of Transportation states that one in five car accidents are caused by distracted drivers, meaning that distracted driving causes injuries to 400,000 people and claims the lives of 2,841 just in 2018. 2.9% of the driver using handheld cell phones in 2017, which was down from 3.3% in 2016.

Introduction:

To improve the alarming statistics of car accidents, innovative methods should be tested. One such method would be to develop an algorithm to detect drivers engaging in distracted behaviors by feeding it dashboard camera images. This algorithm can then be used as an API in a device to classify the driver's behavior by checking if they are driving attentively, wearing their seatbelt and remind them if they are not.

The input of this model is images of the driver taken in the car. The dataset is provided by State Farm consists of images, which means that the most efficient way to tackle the problem at hand is to develop a neural network model and then train it on a training subset of the dataset with the objective to obtain a high-accuracy model. The model will then be tested on a testing subset of the dataset and evaluated.

Data Exploration:

The dataset used in this project was provided by State Farm through a Kaggle competition, which has 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. 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

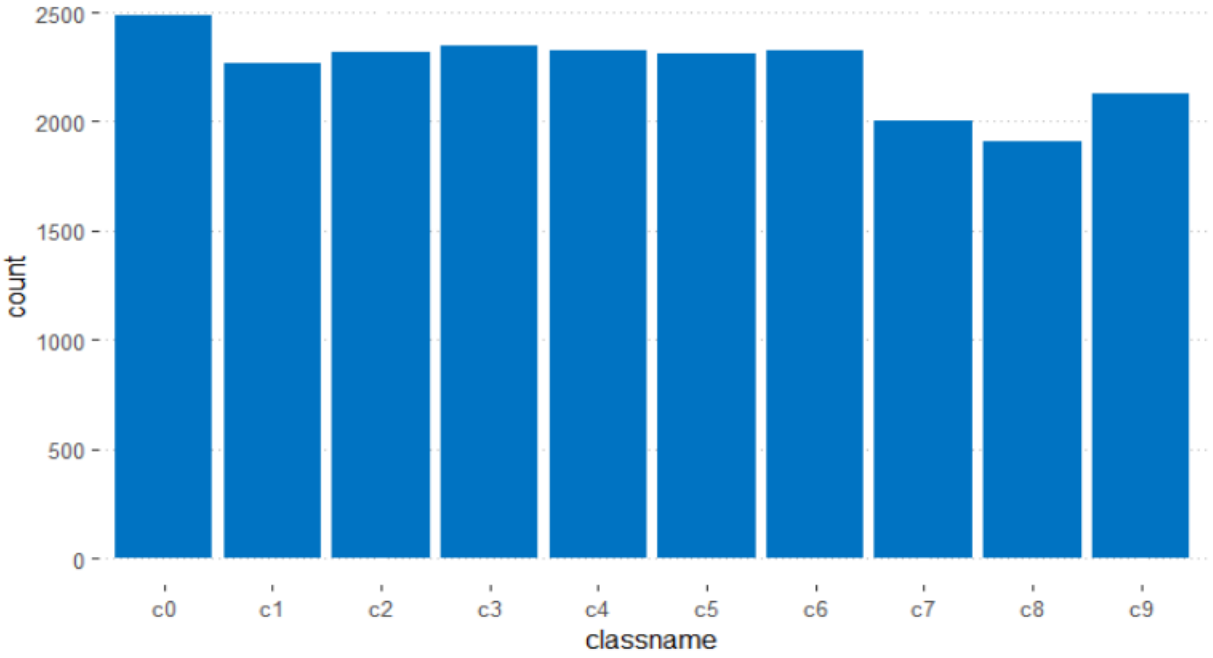
c9: talking to a passenger

The dataset contains a total of 102150 images split into a training set of 22424 images and a testing set of 79726 images. Here is a sample of the dataset images.

The images are 480 X 640 pixels and the distribution of the classes in the training set are relatively uniform. The total size of the dataset is 81 GB, which is publicly available at: https://www.kaggle.com/c/state-farm-distracted-driver-detection/data

Data Visualization:

One important factor in choosing an evaluation metric and a validation method is the uniformity of the target class distribution. The class distribution for this dataset is relatively uniform.



Algorithms and Techniques

The project was executed using a convolutional neural network (CNN) of 'keras' Package in R. CNNs are made of neurons that have learnable weight and biases, each neuron receives an input, performs a dot product and usually followed by a non-linear function. The inputs of CNNs are image pixels as vectors and their outputs are class scores. As the vectors go from the input layer to the output layer they pass through a series of hidden layers, each hidden layer consists of neurons where each neuron is fully connected to every neuron in the previous layer while remaining completely independent from the other neurons in the same hidden layer.

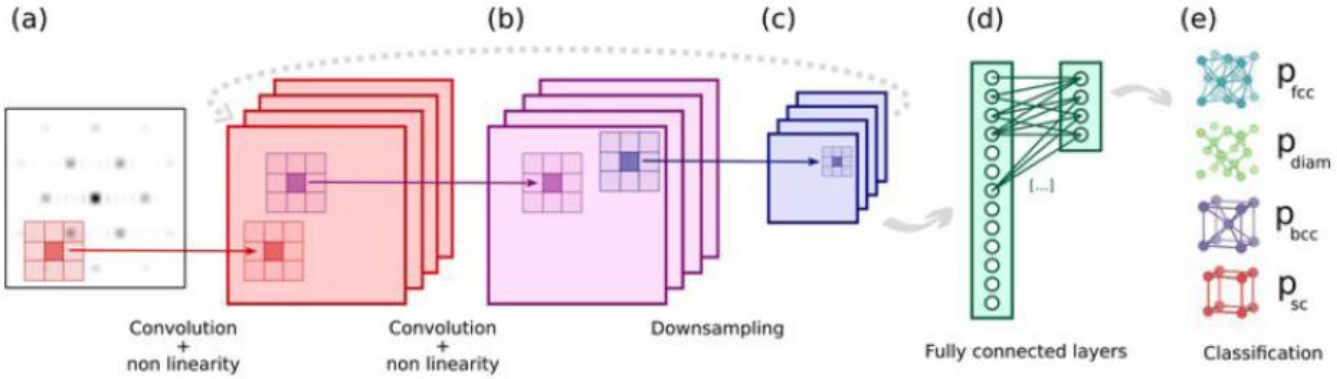


Figure 1 Visual representation of CNN

Data Preprocessing:

The training set which holds 22424 images has taken over 20 GB of memory and the test set has a memory of 61 GB which holds 79726 images with no labels. Batches of tensor image data were generated by splitting the entire training dataset into the validation set and another 50% for training the model.

0	1	2	3	4	5	6	7	8	9
1245	1134	1159	1173	1163	1156	1163	1001	956	1065

Figure 2 Number of Training Images per Class

Moreover, the project was set to use keras with TensorFlow as backend which means that the input has to be a 4D tensors to be compatible with keras CNN, so the images were first resized to 256 x 256 pixels from 480X640 pixels, then converted into a 3D tensors and then into a 4D tensors of shape (N, 256, 256, 3) where N is the number of images. After that, the tensors were scaled by dividing them over 255.

Implementation:

The final model was implemented first by creating a model through the use of keras function Sequential and adding a Convolutional layer of 16 filters, kernel size 2X2, same padding, relu activation, and input shape the same as the 4D tensors that are (None, 256, 256, 3). A second hidden layer of 128 filters, kernel size 2X2, relu activation was added which followed by a 2X2 max-pooling layer. Followed by 2 additional hidden layers of 64 filters, kernel size 4X4 & relu activation. The key difference between the first Conv layer and the rest is that no input shape was specified as that only required for the first layer of the model. The activations of layers at each batch were normalized to maintain the mean activation close to 0 and the activation standard deviation close to 1. The model was regularized by adding the Dropout layer of rate 0.2 followed by a Flatten layer. The output had a Dense layer of 10 outputs representing the data classes with a softmax activation.

```
Model
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 256, 256, 16)      208
_________________________________________________________________
batch_normalization_16 (BatchNormal (None, 256, 256, 16)      64
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 255, 255, 128)     8320
_________________________________________________________________
batch_normalization_17 (BatchNormal (None, 255, 255, 128)     512
_________________________________________________________________
max_pooling2d_8 (MaxPooling2D)  (None, 127, 127, 128)     0
_________________________________________________________________
dropout_11 (Dropout)         (None, 127, 127, 128)     0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 127, 127, 64)      131136
_________________________________________________________________
batch_normalization_18 (BatchNormal (None, 127, 127, 64)      256
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 124, 124, 64)      65600
_________________________________________________________________
batch_normalization_19 (BatchNormal (None, 124, 124, 64)      256
_________________________________________________________________
max_pooling2d_9 (MaxPooling2D)  (None, 62, 62, 64)        0
_________________________________________________________________
dropout_12 (Dropout)         (None, 62, 62, 64)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 246016)            0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2460170
=================================================================
Total params: 2,666,522
Trainable params: 2,665,978
Non-trainable params: 544
_________________________________________________________________
```

Figure 3 Structure of final model

Model Evaluation :

The benchmark performance was relatively good although it was a simple model and only trained for 10 epochs. In terms of accuracy, it achieved **99.5%** in training and **100%** in the validation set, while in loss function it reached 0.021 & 0.003 in the training and validation sets respectively. However, it suffered from

high bias as its results in validation sets are better than those in the training data set. The loss vs epochs & accuracy vs epochs is shown below.

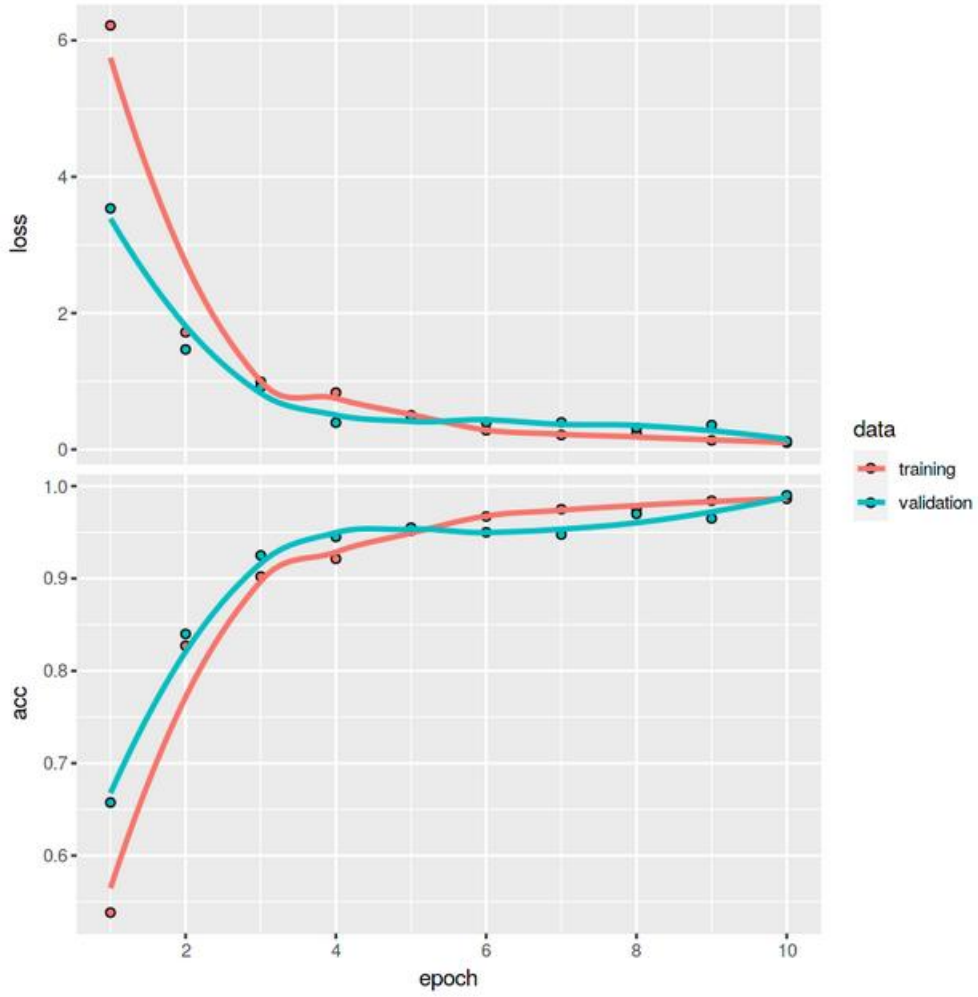


Figure 4 Loss & Accuracy vs iteration number

Results:

The model was tested by randomly selecting the images from the test dataset. The algorithm was able to identify a wide range of distracting activities as can be seen from the below figures. However, in certain cases, it was able to classify the image as distracted but failed to put in the appropriate category. For example, the appropriate classification for Figure 7 is “reaching behind” but the algorithm misclassified it as “talking to the passenger”.

Figure 5 Safe Driving

Figure 6 texting - right

Figure 7 talking to passenger

Figure 8 Texting - right

Conclusion and Future work:

In conclusion, a three-layer Neural Network model was successfully implemented that has great performance on the distracted driver detection task and gives a validation accuracy of 100%. For the future improvement of these self-implemented models, a more sophisticated weight initialization method like Xavier Initialization or LeCun uniform Initialization can be applied. Besides, the dataset also provides 77,000 test images without labels, so if these images are labeled or use a semi-supervised learning method, this model could be developed with these images. Finally, pre-trained CNN-based models like ResNet or VGG can be used to get a better result.

References:

[1]D. G. Kidd, N. K. Chaudhary, "Changes in the sources of distracted driving among Northern Virginia drivers in 2014and 2018: A comparison of results from two roadside observation surveys", Journal of Safety Research2019,69,131-138;doi: 10.1016/j.jsr.2018.12.004

[2]R.L. Olson, R.J. Hanowski, J.S. Hickman, J. Bocanegra. Driver Distraction in Commercial Vehicle Opera-tonus's. Department of Transportation, Washington, DC. Data available online since September 2009 on:https://www.fmcsa.dot.gov/sites/fmcsa.dot.gov/files/docs/FMCSA-RRR-09-042.pdf

[3] keras sample example available at: https://keras.rstudio.com/articles/examples/cifar10_cnn.html

[4] https://github.com/rstudio/keras/blob/master/R/initializers.R