Capstone Project-Machine Learning Engineer Nanodegree

Raghu Dharmavaram
March 20th, 2018

Project Overview

As per the Centers for Disease Control and Prevention (CDC)'s motor vehicle safety division, each day in the United States, approximately 9 people are killed and more than 1000 injured in crashes that are reported to involve a distracted driver.[1] Distracted driving means doing another activity while driving which can take attention away from driving and can cause vehicle accidents.

State farm hopes to detect drivers distracted behavior by observing dashboard images, hence to provide better insurance to their customers.[2]

As part of Machine Learning Nano degree Cap Stone project, I used Kaggle state farm data set (images)[2] to develop and fine-tune ML models to detect driver's behavior by analyzing the given driver images.

Problem Statement

Needs to develop an algorithm to classify driver's behavior and identify whether driver is driving responsively or distracted by various actions such as taking selfie, doing make-up, using mobile while driving or taken safety precautions such as wearing seatbelt etc., by using 2D dashboard camera images.

Necessary steps:

1. Get images from Kaggle dataset[2] and pre-process them.
2. Develop a model (build and train) to classify images.
3. Test the model with test dataset by using various techniques and make sure model is expected to achieve at least top 50% of the overall kaggle leadership submission.

Evaluation Metrics

Often Kaggle Submissions are evaluated using the logarithmic loss.[3] and success of the completion depends on minimizing the logloss.[3] In order to evaluate in logarithmic loss[3], first every image needs to be labeled with a class (C0, C1) and in order to calculate logloss the classifier has to assign a probability to each image/class. So, for each image, I must submit a set of predicted probabilities. The formula is,

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

N = no of images in the test set
M = no of image class
y_{ij} = if i is equal to class j then 1 else 0
p_{ij} = predicted probability observation i belongs to class j.

The probabilities for a given image are not required to sum to one because they are rescaled prior to being scored (each row is divided by the row sum). In order to avoid the extremes of the log function, predicted probabilities are replaced with max (min (p,1 − 10^{-15}),10^{-15})

The metric logarithmic loss[3] gives a refined view of the model and performance as this provides probabilities of predication rather than yes or no. Apart from this, F1 score is focused on measuring the no of true positives by comparing total no of predicted positive vs no of actual positives and F1 score solely depends on threshold of each class.

Considering above two points, logarithmic loss[3] considered most relevant metric for this problem.

Analysis:

Datasets and Inputs:

Driver images, each taken in a car with a driver doing something in the car (texting, eating, talking on the phone, makeup, reaching behind, etc.) were provided.[4]

Following are the file descriptions and URL's from which the data can be obtained:

- imgs.zip - zipped folder of all (train/test) images.[4]
- sample_submission.csv - a sample submission file in the correct format.[5]
- driver_imgs_list.csv - a list of training images, their subject (driver) id, and class id.[6]

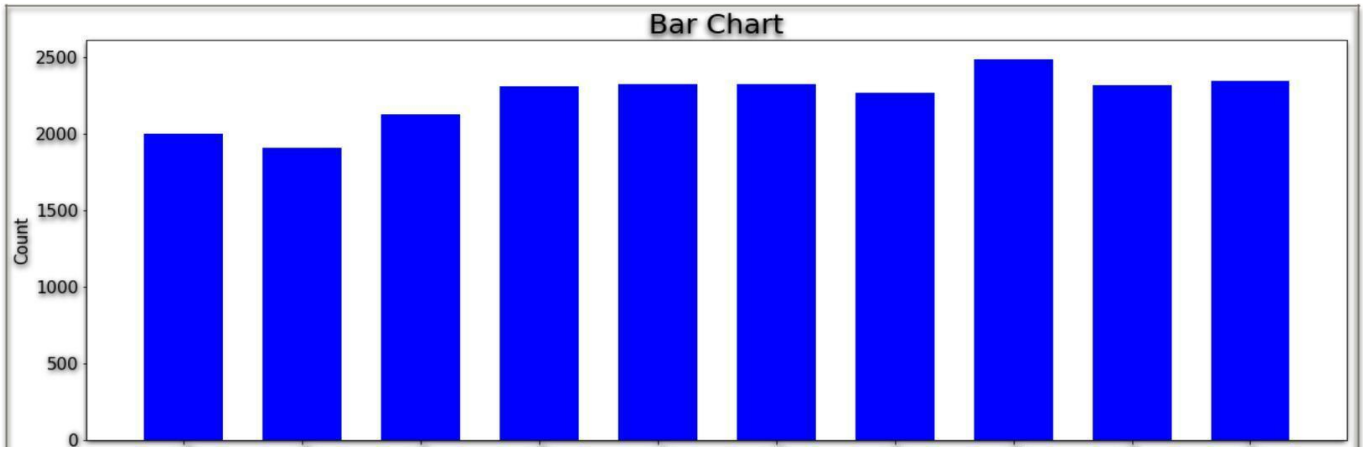
The 10 classes to predict are:

C0	Safe Driving
C1	Testing-right
C2	Taking 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

We have total of 102150 colored images with 640 X 480 pixels each. Out of these 17939 are training images, 4485 are validation images and 79726 are test images. Total images size is around 4gb hence output of test data sets without images submitted in the final project. Training, validation images belong to the categories shown above.

Data Visualization:

The graph is plotted for the count of images for each class in the training dataset. The distribution is uniform for each class and also it is evident that there is not much class imbalance in the training dataset.



Sample images from the dataset:

Benchmark Model

I am going to use top model of Kaggle model with the Public Leader board score (multi-class logarithmic loss) of 0.08739.[7] as a bench mark. so, I created standard Convolutional Neural Network (CNN) model and trained.

In this CNN, I have 2 fully connected layers which means 4 conventional layers with 4 max pooling layers in between and a flattening layer with a dropout before using as a fully connected layer. Here filters were increased to 512 in each conventional layer and Xavier initialization[8] was used in each of the layers. The no of nodes in the fully connected layer were set as 10 along with softmax [17] activation function. Relu activation function[9] was used for all other layers.

When tested with test data set this model is predicted in Kaggle's Public Leaderboard score of 2.67118.

Methodology

Data Preprocessing steps:

Data is preprocessed before model is built and train.

1. with randomized data, images are divided into training and validation sets.
2. Later images have been resized to square images with size of 224 X 224 pixels.
3. Image Normalization: Each image pixel is divided by 255 and a value of 0.5 is subtracted to ensure mean of zero

Implementation:

Algorithms and Techniques:

Convolutional Neural Network: as name suggests, CNN consists one or more convolutional layers and contains one or more fully connected layers which forms as standard multilayer neural network.[10]

The architecture of CNN takes images as inputs this allows developers to encode certain image properties into the model. For example, a 2D image structure can be used with tied weights, local connections to form a pooling[11] and form a translational feature.[12]

Another advantage of CNN is they are easy to train with many fewer parameters instead of fully connected networks with hidden units.

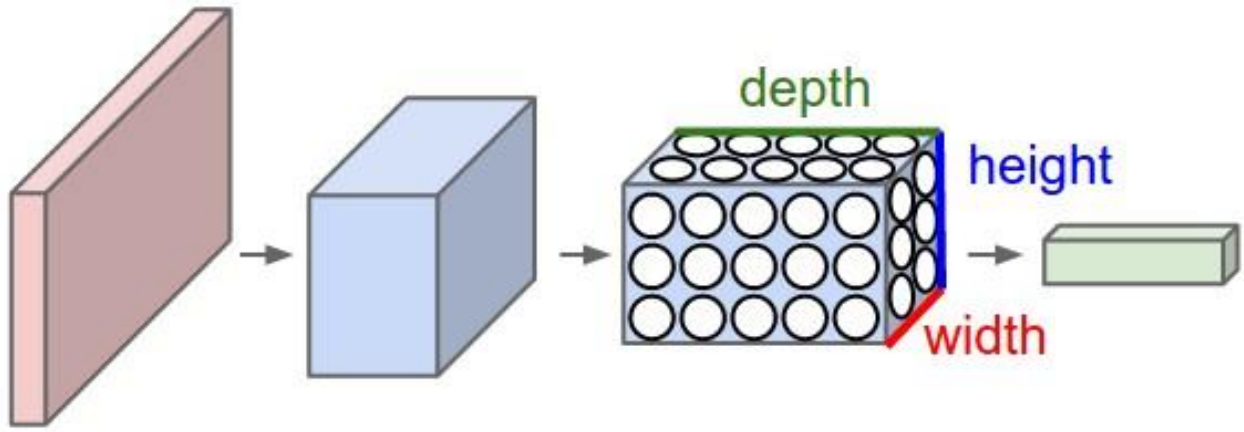


Image Source: http://cs231n.github.io/convolutional-networks/

A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).[13]

CNN Architecture:

Briefly, CNN takes the image, pass it thru series of convolutional, non-linear, fully connected layers, pooling and provides an output which can best describes about the input image.

CNN/ConvNet Layers[13]:

- Convolutional Layer
- Pooling Layer
- Fully Connected Layer
- Classification Layer

Convolutional Layer: This is the first layer in CNN. In CNN, the primary purpose of Convolutional layer is to extract features from the input image. Convolutional maintains the relationship between pixels by learning image features using small squares of input data.

To understand better, imagine input image (5 X 5) as a matrix of pixel values whose values are only 0 and 1; another 3 X 3 matrix for convolution purpose.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

1	0	1
0	1	0
1	0	1

The convolution of the 5 X 5 image and the 3 X 3 matrix can be computed as below.

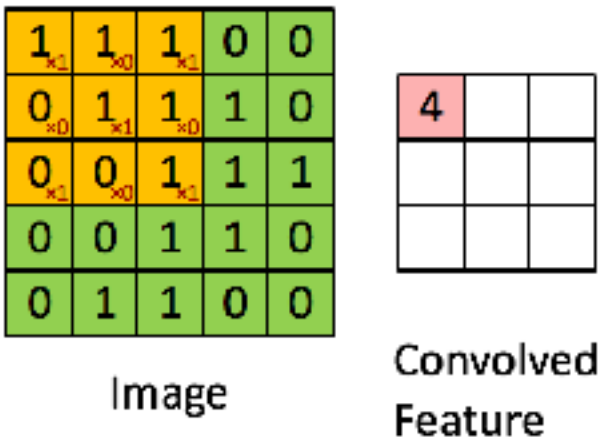


Image Source - http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

We slide the orange matrix over green matrix by 1 pixel (called as stride), for every position we have to compute element wise multiplication and add the output

of multiplication to get final integer which forms a single element of the output matrix (pink matrix). Here 3 x 3 matrix sees only a part of the input image in each stride.

In the CNN architecture – 3 x 3 matrix is called a 'Filter' or 'Kernel' or 'Feature detector' and the pink matrix is called as 'Convolved Feature' or 'Activation Map' or 'Feature Map'. It is important to note the filters acts as feature detectors from the input.[13, 14]

Pooling Layer: Pooling layer used to control over-fitting and Spatial pooling.

Spatial Pooling: Progressively reduce the dimensionality of each feature map but retains most important information. Spatial pooling or subsampling or down-sampling can be of different types - Max, Average, Sum etc.
The function of pooling is to progressively reduce the spatial size of the input representation. In particular, pooling[13]

- Makes the input representations (feature dimensions) smaller and more manageable.
- To controller over fitting it reduces the number of parameters and computations in the network.
- Makes the network invariant to small transformations, distortions and translations in the input image.
- Helps us arrive at an almost scale invariant representation of input image. With this feature model can detect objects in an image. Further reference – [15] and [16].

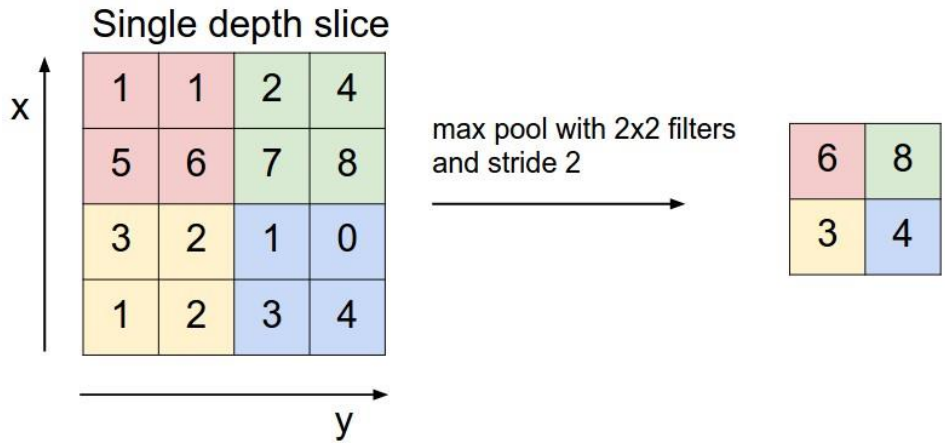


Image source - http://cs231n.github.io/convolutional-networks/#pool

The most common down-sampling operation is max, giving rise to max pooling, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2X2 square).[13]

Fully connected layer (FC):
FC is uses a softmax[17] activation function in the output layer. The output from the convolutional and pooling layers gives us high-level features of the input. FC layer uses these features to classify the image into various classes based on the training set.[13]

Classification Layer (CL):
CL used in final layer of CNN to corrects the training dataset and used to predict the final output as a probability vector. Softmax[17] loss is used for predicating a single class of K mutually exclusive classes. [17]

Graphical representation of a typical CNN architecture

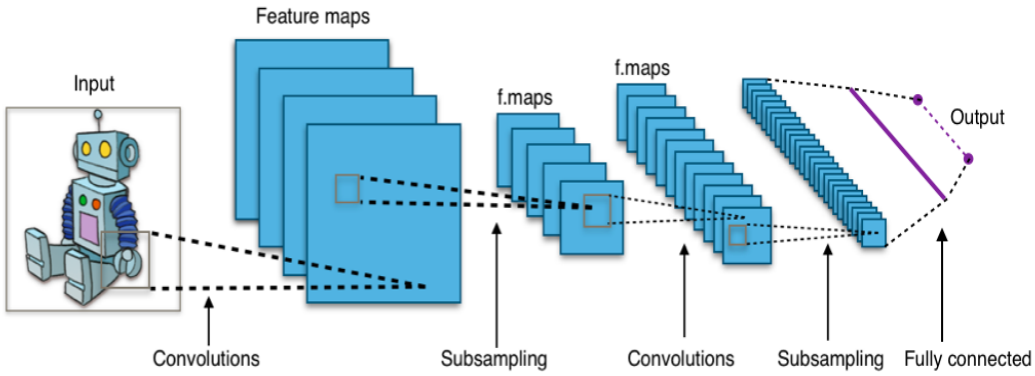


Image source:
https://upload.wikimedia.org/wikipedia/commons/6/63/Typical_cnn.png

Following are the steps carried out for training a CNN:

1. After preprocessing of images, a simple CNN is created to classify images.
2. CNN consists of 2 fully connected layers, with filters increased to 512 in each of the convolutional layer and Xavier initialization[8] was used in each of the layers.
3. For the classification layer – softmax[17] activation function is used for the last fully connected layer with node size as 10.

Pictorial representation of simple CNN:

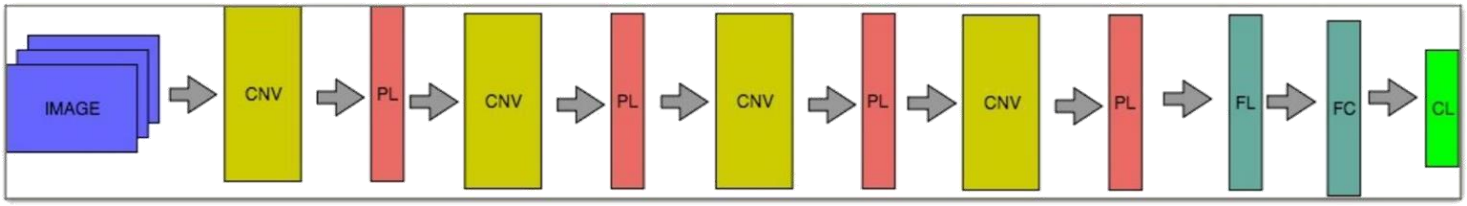


image source: https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-2.pdf

Implementation:

1. Model used keras rmsprop[18] optimizer and keras categorical_crossentropy[19] as loss function.
2. Trained with batch size of 40 for 30 epochs.[20]

Since data is huge and training CNN from scratch is rare, so Transfer Learning[21] is incorporated.

Transfer Learning[21]: In this technique it is common to pre-train a ConvNet on a large data set and then use that model as an initializer or fixed feature extractor. Out of three major learning scenarios I have tried feature extractor and fine-tuning.

ConvNet as fixed feature extractor[21]:

In this technique a pre-trained network is initialized and the last fully-connected layer is removed. After removal this is treated as a fixed feature extractor for the new dataset.

For this, VGG16 architecture has been used- first instantiated the convolutional part of the model, then model is executed once on training and validation dataset to record the output in 2 numpy arrays. Later this small fully-connected model is trained on top of stored features using above arrays.[22]

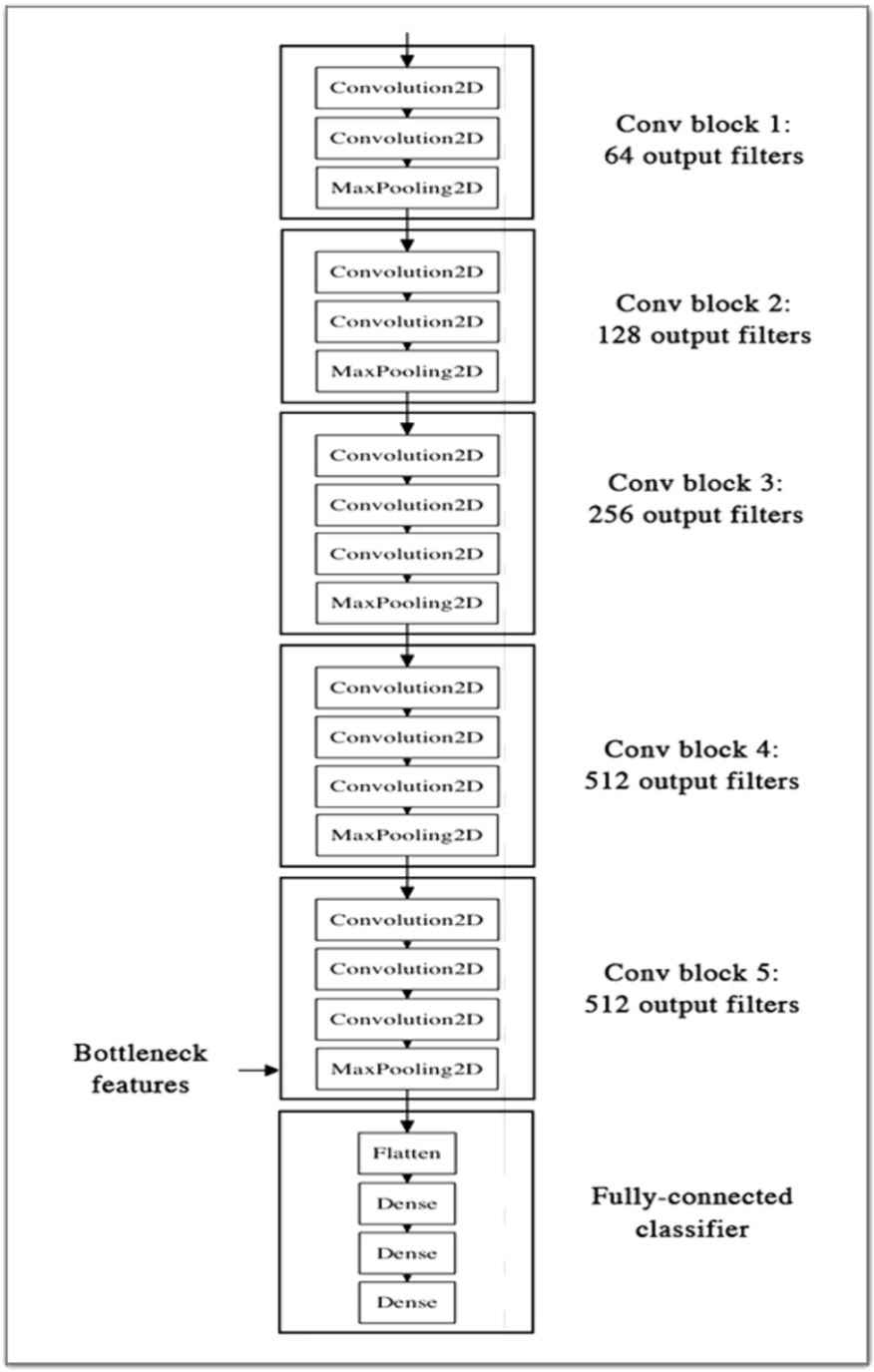
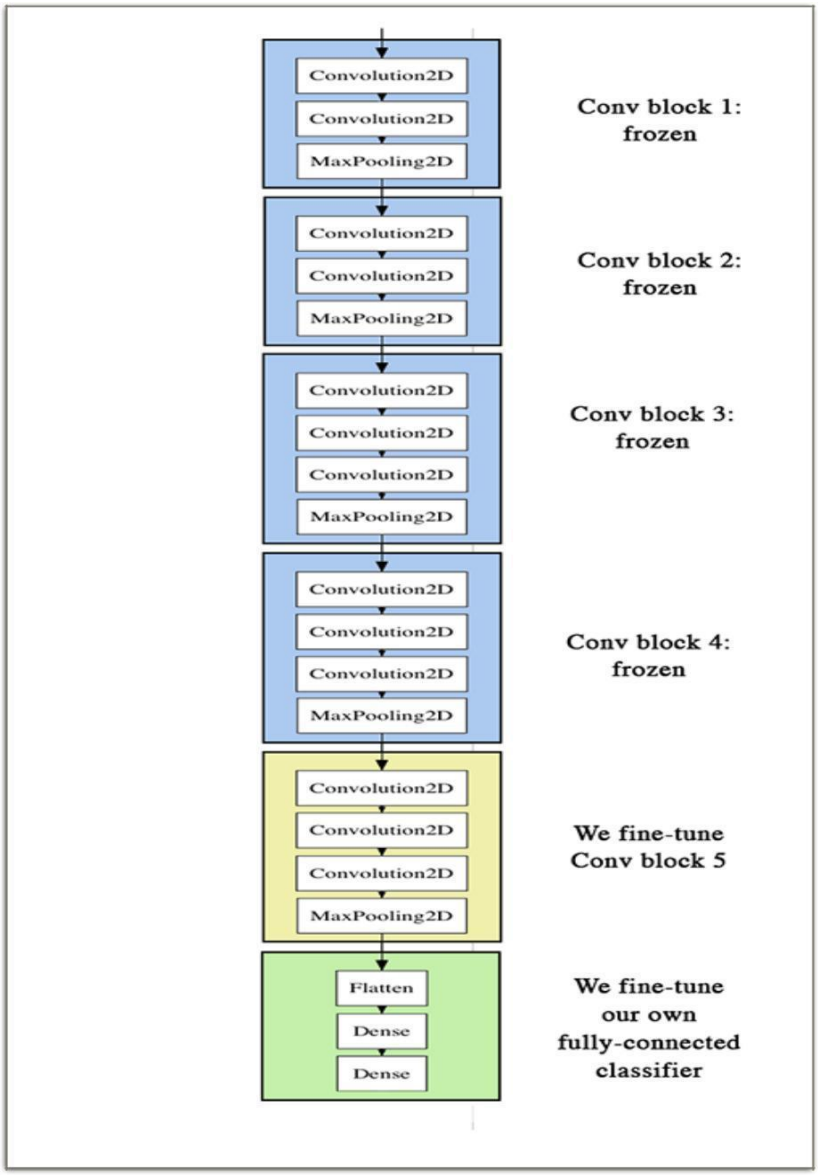


Image source - https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Fine-tuning the ConvNet[21]:
Fine tuning consists starting from a trained model, then re-training model on a new data set using small weight updates.

- Instantiate the convolutional base of VGG16 by loading its weights.
- Add previously defined fully-connected model on top with corresponding weights.
- Freeze the layers of VGG16 model up to the last convolutional block

Image source: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html



- Python, keras with Tensorflow being used to develop model.
- Pre-processing of images, training process, executing bottleneck features and saving data for Kaggle submission took long hours due to lesser capacity system and slow learning rate.
- Zero mean was ensured by subtracting 0.5 during pre-processing.
- Training carried with 400 epochs[20] with a batch size of 16.

Fine Tuning/Refinement:

Simple CNN model with decent loss resulted in public score of 2.67118.

To improve the loss, Transfer Learning methodology has been incorporated with VGG16 with 2 types of architectures. Model 1 showed good results hence it was improved further.

Improvements:

- VGG16 is instantiated as first 15 layers frozen.
- Last Convolutional block is fine-tuned.
- To avoid over-fitting drop-out layer was added.
- Xavier Initialization[8] has been used.
- Global avg pooling and fully connected layer has been added and fine-tuned.
- SGD Optimizer [23] was used with a momentum of 0.9
- Training is carried out for 10 epochs[20] with a batch size of 10.

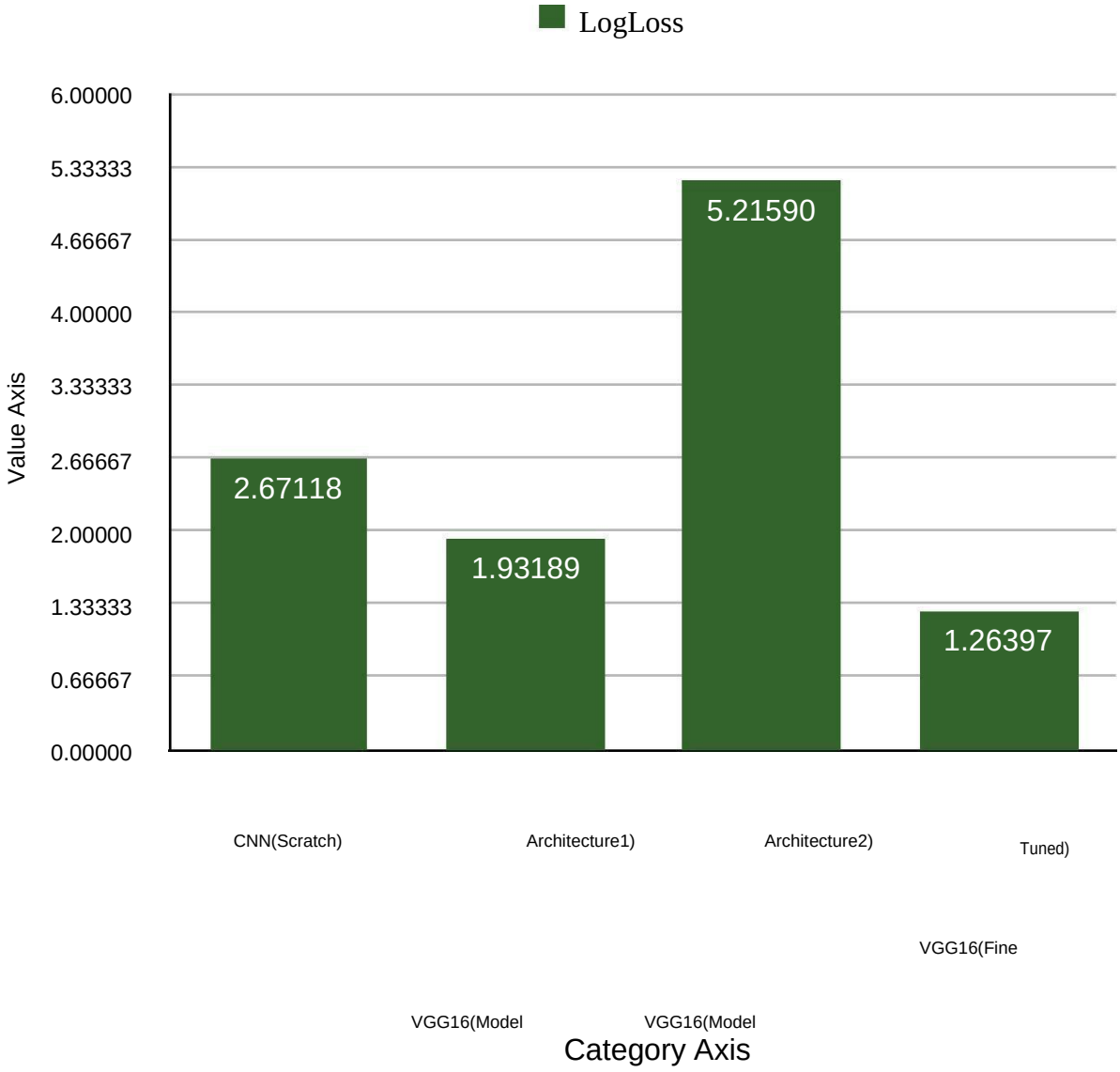
This fine-tuned model was selected to submit to Kaggle public leadership board. The public score for fine-tuned model was 1.26397. This is best score than that of the initial simple CNN architecture score of 2.67118.

Fine-tuned model output resulted in rank of 600 out of 1440 in Kaggle public leadership board i.e. in top 41.66% with loss of validation dataset as 0.00751. In order to address over-fitting as future enhancement consider adding the drop-out and L2 regulation.[24]

Results

Model Evaluation and Validation

The comparison of the Public Scores for all the model architectures considered for this data set is shown in below image.



Justification:

The selected model ensued in a decent public score. This can be used in a mobile application where the driver behavior can be detected. This can aid in the safety of the driver and can help insurance companies identify risky driver.

Conclusion

Reflection/Summary:

The process used to define and develop the model can be summarized as follows:

1. An initial usage of machine learning is identified and corresponding datasets were obtained.
2. The datasets were pre-processed.
3. Cloud hardware is utilized to process the dataset.
4. The highest score in the Public Leadership board was taken as a benchmark and a target rank was defined.
5. Using Dog breed image processing project [25] of Udacity, an initial CNN was created from Scratch. After training this was tested and resulted in rank of 1348 out of 1440 in Public Leaderboard i.e. in top 93.61%
6. To better the loss metric, pre-trained model such as VGG16 was used and Transfer Learning [21] is applied.
7. Bottleneck feature of a pre-trained network is applied to VGG16 for two types of model architectures as the top layer.
   7.1. Model architecture-1 was trained and tested resulted in rank of 1021 out of 1440 in Kaggle public leaderboard i.e. in top 70.90%.
   7.2. Model architecture-2 was trained and tested but did not result in any improvement.
8. Fine-tuned VGG16 model architecture 1 as mentioned in Fine-Tuning section.
   8.1. This best the improvement of loss metric and when tested resulted in rank of 600 out of 1440 in Public leaderboard i.e. in top 41.66%.

Improvement

Following areas are identified for improvement

- Bigger size when resizing of images.
- VGG16 [31]architecture used in this project to pre-training the model, we can try VGG-19 [32], ResNet [33], Inception[30] deep learning.

References

[1] National Center for Statistics and Analysis. Distracted Driving: 2015, in Traffic Safety Research Notes. DOT HS 812 381. March 2017, National Highway Traffic Safety Administration: Washington, D.C. (https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/812_381_distracteddriving2015.pdf)

[2]. Kaggle state farm project dataset - https://www.kaggle.com/c/state-farm-distracted-driver-detection

[3]. Loss functions for classification- https://en.wikipedia.org/wiki/Loss_functions_for_classification

[4]. https://www.kaggle.com/c/state-farm-distracted-driver-detection/download/ imgs.zip

[5]. https://www.kaggle.com/c/state-farm-distracted-driver-detection/download/ sample_submission.csv.zip

[6]https://www.kaggle.com/c/state-farm-distracted-driver-detection/download/driver_imgs_list.csv.zip

[7]. https://www.kaggle.com/c/state-farm-distracted-driver-detection/leaderboard

[8]. http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization

[9]. Rectifier liner Unit - https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

[10] http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

[11] http://ufldl.stanford.edu/tutorial/supervised/Pooling/

[12] http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/

[13] http://cs231n.github.io/convolutional-networks/

[14] http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

[15] https://github.com/rasbt/python-machine-learning-book/blob/master/faq/difference-deep-and-normal-learning.md

[16] https://www.quora.com/How-is-a-convolutional-neural-network-able-to-learn-invariant-features

[17] https://www.quora.com/How-does-the-softmax-classification-layer-of-a-neural-network-work

[18] https://keras.io/optimizers/#rmsprop
[19] https://keras.io/losses/#categorical_crossentropy
[20] https://www.quora.com/What-is-epochs-in-machine-learning
[21] http://cs231n.github.io/transfer-learning/
[22] https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
[23] https://keras.io/optimizers/#sgd
[24] http://www.ritchieng.com/machine-learning/deep-learning/tensorflow/regularization/
[25] https://github.com/udacity/machine-learning/tree/master/projects/image-classification
[26] https://keras.io/
[27] https://www.tensorflow.org/
[28] https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-2.pdf
[29] https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-1.pdf
[30] https://nicolovaligi.com/history-inception-deep-learning-architecture.html
[31] https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3
[32] https://gist.github.com/baraldilorenzo/8d096f48a1be4a2d660d
[33] https://arxiv.org/abs/1512.03385
[34] https://github.com/udacity/machine-learning/blob/master/projects/capstone/capstone_report_template.md