Capstone Project

Machine Learning Engineer Nanodegree

Abhinav Kumar Jha

July 9th,2018

Definition

Project Overview

According to the CDC motor vehicle safety division, 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 better insure their customers, by testing whether dashboard cameras can automatically detect drivers engaging in distracted behaviours.

In this project, I have created and refined machine learning models to detect what the driver is doing in a car given driver images. This is done by predicting the likelihood of what the driver is doing in each picture.

Problem Statement

Given a dataset of 2D dashboard camera images, an algorithm needs to be developed to classify each driver's behaviour and determine if they are driving attentively, wearing their seatbelt, or taking a selfie with their friends in the backseat etc..? This can then be used to automatically detect drivers engaging in distracted behaviours from dashboard cameras.

Following are needed tasks for the development of the algorithm:

1. Download and preprocess the driver images
2. Build and train the model to classify the driver images
3. Test the model and further improve the model using different techniques.

The final model is expected to achieve suitable score so that it is in top 50% of the Public Leaderboard submissions in Kaggle.

Metrics

Submissions are evaluated using the multi-class logarithmic loss. Each image has been labeled with one true class. For each image, we must submit a set of predicted probabilities (one for every image). The formula is then,

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

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.*

The submitted 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 multi-class logarithmic loss is selected than accuracy as this gives the probability of the predictions than simply saying yes or no.This gives a more nuanced view of the model and performance. Also F1 score may not be appropriate metric as it is focussed on measuring the number of true positives against total number predicted as positive, number of actual positives. In addition to this F1 score also depends on the threshold used to identify each class. Hence multi-class logarithmic loss is the most relevant metric for this problem.

Analysis

Data Exploration

The provided data set has 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). This dataset is obtained from Kaggle(State Farm Distracted Driver Detection competition).

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

- 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

https://www.kaggle.com/c/state-farm-distracted-driver-detection/download/driver_imgs_list.csv.zip
https://www.kaggle.com/c/state-farm-distracted-driver-detection/download/imgs.zip
https://www.kaggle.com/c/state-farm-distracted-driver-detection/download/sample_submission.csv.zip

The 10 classes to predict 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

There are 102150 total images. Of these 17939 are training images,4485 are validation images and 79726 are training images. All the training, validation images belong to the 10 categories shown above.The images are coloured and have 640 x 480 pixels each as shown in Fig.1

Fig.1 Images from the data set

Exploratory Visualization

The count of images for each class in the training dataset is obtained and the graph is plotted as shown below in Fig.2 . From the graph it is obvious that the distribution is uniform. Also it is evident that there is not much class imbalance in the training dataset.

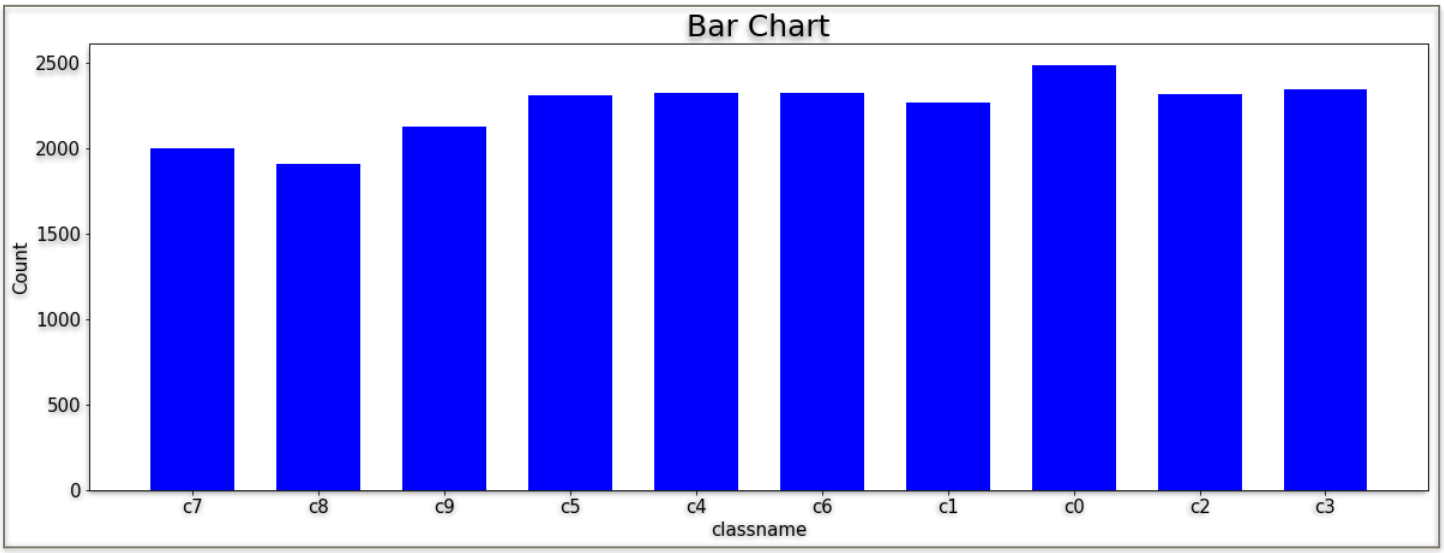

Fig.2 Image Count for each class - Training dataset

Benchmark

The model with the Public Leaderboard score(multi-class logarithmic loss) of 0.08690 is used as a benchmark model.A standard CNN architecture was initially created and trained. We have created 4 convolutional layers with 4 max pooling layers in between. Filters were increased from 64 to 512 in each of the convolutional layers. Also dropout was used along with flattening layer before using the fully connected layer. Altogether the CNN has 2 fully connected layers. Number of nodes in the last fully connected layer were setup as 10 along with softmax activation function. Relu activation function was used for all other layers.Xavier initialization was used in each of the layers.This resulted in a Public Leaderboard score(multi-class logarithmic

loss) of 2.67118 when predicted with the test dataset and submitted to Kaggle.

Methodology

Data Preprocessing

Preprocessing of data is carried out before model is built and training process is executed.

Following are the steps carried out during preprocessing.

1. Initially the images are divided into training and validation sets.
2. The images are resized to a square images i.e. 224 x 224 pixels.
3. All three channels were used during training process as these are color images.
4. The images are normalised by dividing every pixel in every image by 255.
5. To ensure the mean is zero a value of 0.5 is subtracted.

Implementation

Below is the brief description of CNN's and different layers as specified in Convolutional neural network. (2017, September 29) wikipedia article

"CNN Architecture:

In machine learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery.

Convolutional networks were inspired by biological processes in which the connectivity pattern between neurons is inspired by the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.
They have applications in image and video recognition, recommender systems and natural language processing.
A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers are either convolutional, pooling or fully connected.

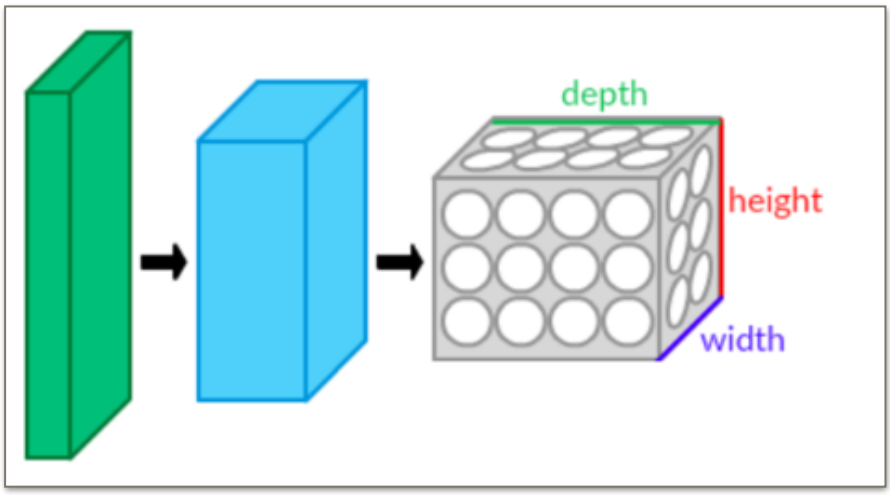

Fig.3 CNN layers arranged in 3 dimensions(File:Conv layers.png. (2016, January 19))

Aphex34 (https://commons.wikimedia.org/wiki/File:Conv_layers.png),
https://creativecommons.org/licenses/by-sa/4.0/legalcode

A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume (e.g. holding the class scores) through a differentiable function(shown in Fig.3). A few distinct types of layers are commonly used. We discuss them further below:

Convolutional layer(CNV) :*The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.*

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map(refer Fig.4).

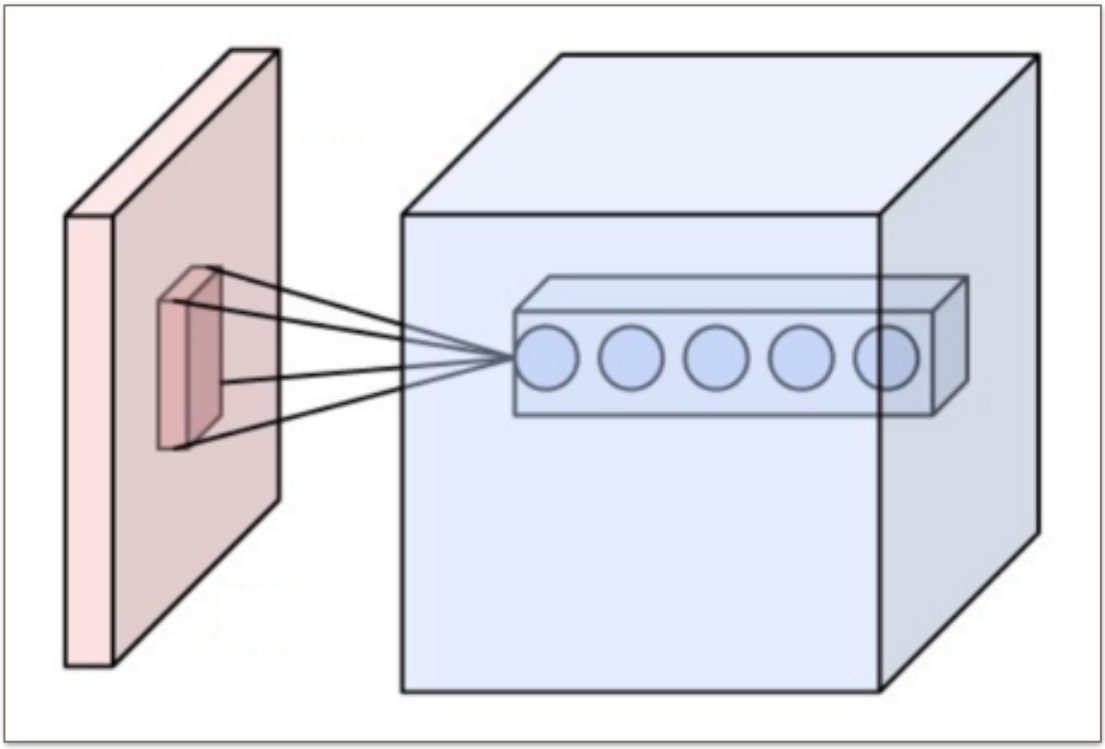

Fig.4 Neurons of a convolutional layer (blue), connected to their receptive field (red)

(File:Conv layer.png. (2016, August 18))

Aphex34 (https://commons.wikimedia.org/wiki/File:Conv_layer.png), https://creativecommons.org/licenses/by-sa/4.0/legalcode

Pooling layer(PL) : *Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The intuition is that the exact location of a feature is less important than its rough location relative to other features. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides another form of translation invariance.*

The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples at every depth slice in the input by 2 along both width and height, discarding 75% of the activations. In this case, every max operation is over 4 numbers. The depth dimension remains unchanged.This is shown in Fig.5.

In addition to max pooling, the pooling units can use other functions, such as average pooling or L2-norm pooling.

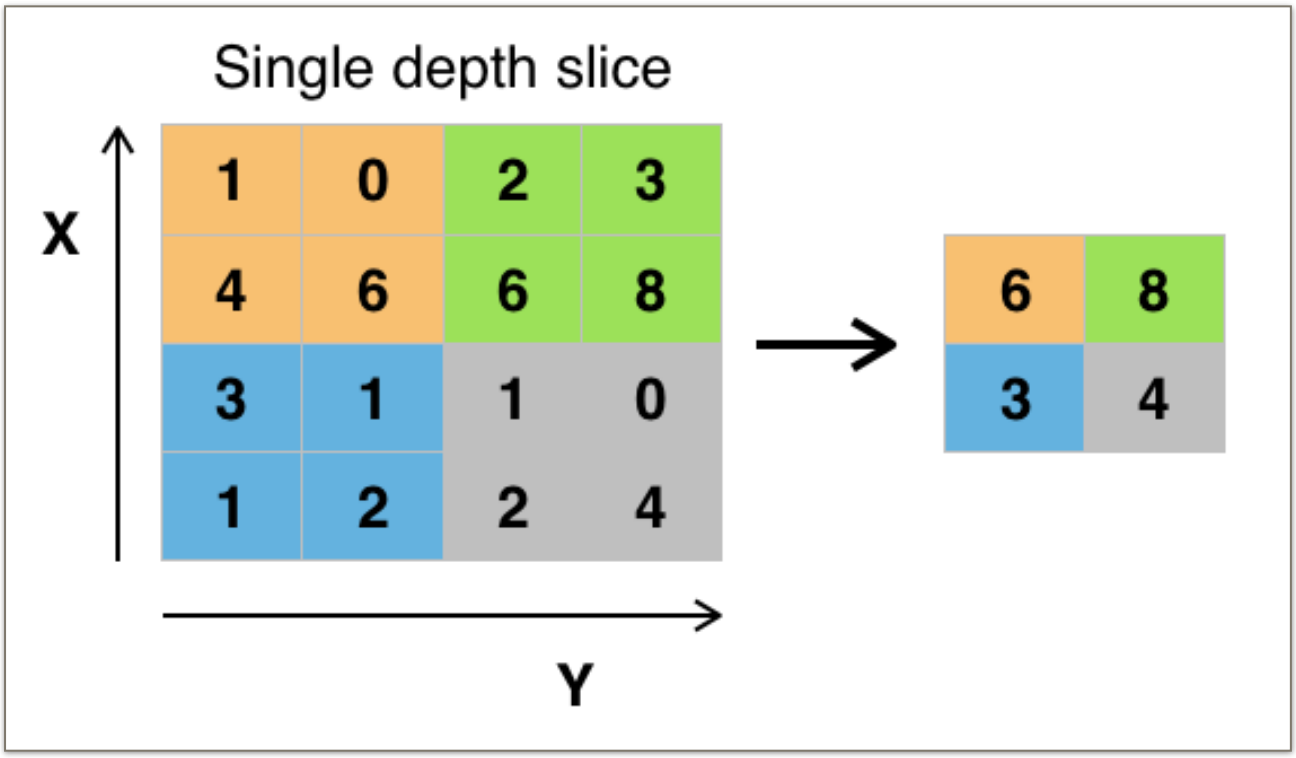

Fig.5 Max pooling with a 2x2 filter and stride = 2(File:Max pooling.png. (2016, August 18))

Aphex34 (https://commons.wikimedia.org/wiki/File:Max_pooling.png), https://creativecommons.org/licenses/by-sa/4.0/legalcode

Fully connected layer(FC) :*Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.*

Classification Layer(CL) :*The classification layer specifies how training penalizes the deviation between the predicted and true labels and is normally the final layer. Various loss functions appropriate for different tasks may be used there. Softmax loss is used for predicting a single class of K mutually exclusive classes. Sigmoid cross-entropy loss is used for predicting K independent probability values in [0,1]*

A typical CNN architecture is shown below(Fig.6)"

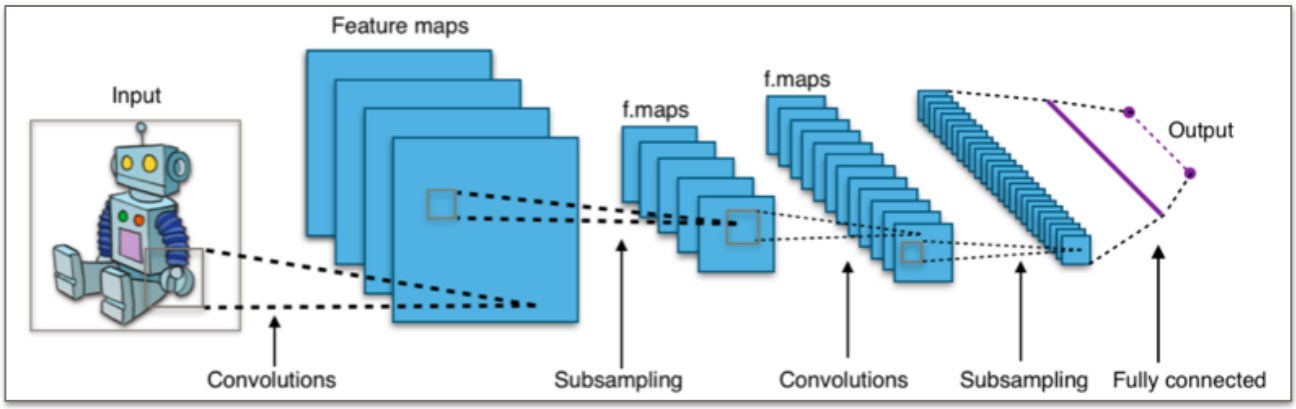

Fig.6 Typical CNN architecture(File:Typical cnn.png. (2016, August 18))

Aphex34 (https://commons.wikimedia.org/wiki/File:Typical_cnn.png), https://creativecommons.org/licenses/by-sa/4.0/legalcode

A simple CNN (Convolutional Neural Net) shown in Fig.7 is created as a initial step.This is then trained with training data.

Following are the steps carried out for training a CNN:

1. The entire dataset i.e. train, validation and test dataset is pre-processed as discussed above(Refer “Data Preprocessing” section).
2. A simple CNN is created to classify images. The CNN consists of 4 convolutional layers with 4 max pooling layers in between.
3. Filters were increased from 64 to 512 in each of the convolutional layers and drop out is added
4. Flattening layer along with two fully connected layers were added at the end of the CNN
5. Number of nodes in the last fully connected layer were setup as 10 which is number of categories in the dataset, along with using softmax activation function. This layer is being used as classification layer.
6. ”Relu” activation function was used for all other layers along with Xavier initialization.
7. The model is compiled with ‘rmprop’ as the optimizer and ‘categorical_crossentropy’ as the loss function.
8. The model is trained for 30 epochs with a batch size of 40.During training process the model parameters were saved when there is an improvement of loss for validation dataset.
9. Finally predictions were made for the test dataset and were submitted in suitable format.

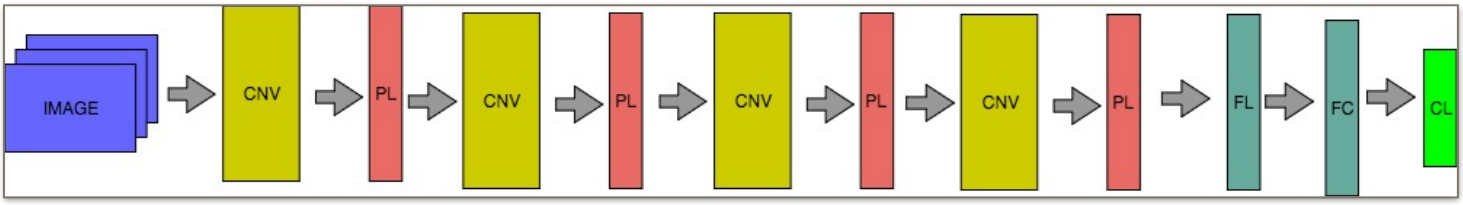

Fig.7 CNN(from Scratch)

Transfer Learning : In this technique instead of training a CNN from scratch a pre-trained model is used as an initialization or fixed feature extractor. Below are two types of transfer learning techniques.

ConvNet as fixed feature extractor : In this technique a pre-trained network is initialized and the last fully-connected layer is removed.After removing this is treated as a fixed feature extractor.Once these fixed features were extracted, the last year is trained with these extracted fixed features.

As shown below in Fig.8 instantiate the convolutional part of the model. Execute the model on training and validation data once, recording the output (the "bottleneck features" from th VGG16 model: the last activation maps before the fully-connected layers) in two numpy arrays. Then a small fully-connected model is trained on top of the stored features.

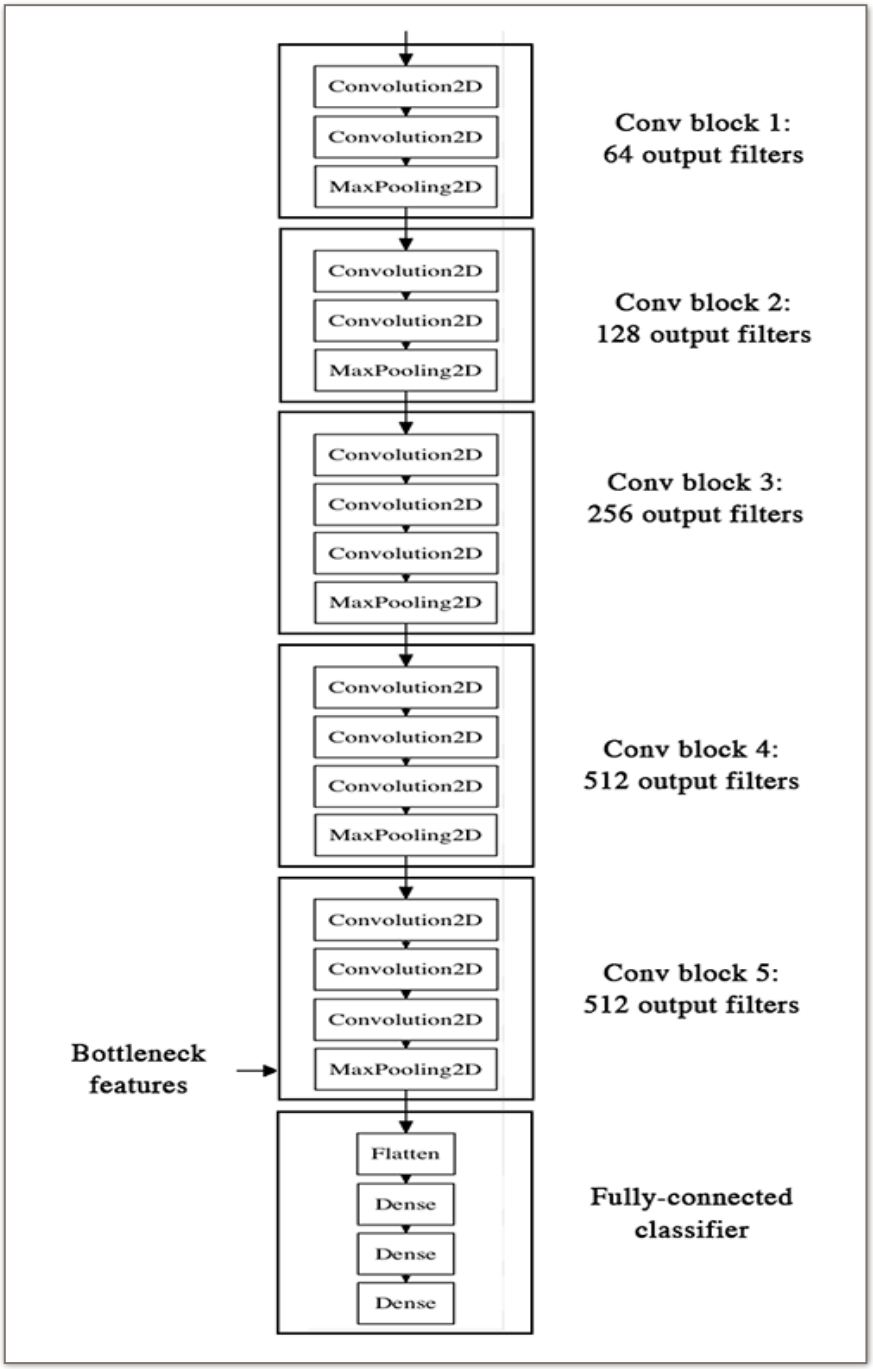

Fig.8(File:vgg16_original.png. (n.d.))

Fine-tuning the ConvNet: In this technique in-addition to replacing and retraining only the classifier, the weights of the pre-trained network are also fine-tuned. "*It is possible to fine-tune all the layers of the ConvNet, or it's possible to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network. This is motivated by the observation that the earlier features of a ConvNet contain more generic features (e.g. edge detectors or colour blob detectors) that should be useful to many tasks, but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset."* (CS231n Convolutional Neural Networks for Visual Recognition(n.d.))

“As shown below in Fig.9 instantiate the convolutional base of VGG16 and load its weights.Add previously defined fully-connected model on top, and load its weights.Freeze the layers of the VGG16 model up to the last convolutional block.Fine-tune the model.This is done with a very slow learning rate, and typically with the SGD optimizer rather than an adaptive learning rate optimizer such as RMSProp. This is to make sure that the magnitude of the updates stays very small, so as not to wreck the previously learned features."(Francois Chollet(Sun 05 June 2016))

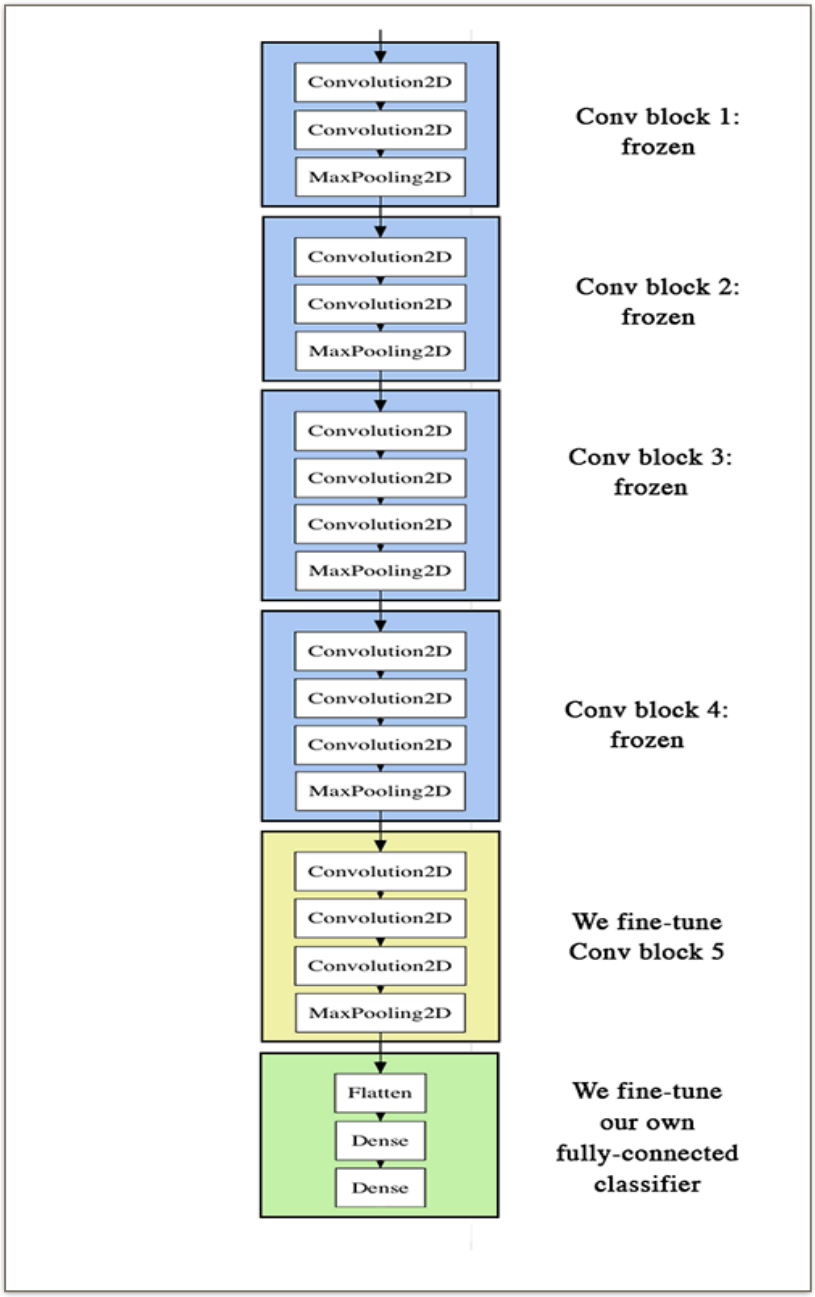

Fig.9(File:vgg16_modified.png. (n.d.))

Programming Language used: Python and it’s libraries keras and TensorFlow as backend.
Recommended Software Requirement:GPU for training the CNN.

Problems encountered:During the training process huge memory usage was observed when all the images were loaded and pre -processed.
Solution:This step was executed multiple times by changing the capacity of the compute instances to account for huge RAM.

Problems encountered: During the transfer learning process, obtaining bottleneck features took time.
Solution:Use a powerful GPU from cloud for reducing the learning process.

Problem encountered:Training process during fine tuning took huge time(~ 6 hours). This may be due to the usage of a very slow learning rate.
Solution:Can use a higher learning rate on the cost of prediction accuracy.

Refinement

To get the initial result simple CNN architecture was built and evaluated. This resulted in a decent loss. The public score for the initial simple CNN architecture(initial unoptimized model) was 2.67118.

After this to further improve the loss, transfer learning was applied to VGG16 along with investigating 2 types of architectures for fully connected layer. Model Architecture1 showed good results and was improved further by using the below techniques

- Drop out layer was added to account for overfitting.
- Xavier initialization was used instead of random initialization of weights
- Zero mean was ensured by subtracting 0.5 during Pre-processing.
- Training was carried out with 400 epochs and with a batch size of 16

To further improve the loss metric ,VGG16 along with Model Architecture1 was selected and fine-tuning was applied. SGD optimiser was used with very slow learning rate of 1e-4. A momentum of 0.9 was applied with SGD.

The fine-tuned model was finally selected as that provided the best score and rank in the Public Leadership board. The public score for fine-tuned model(final optimized model) was 1.26397. This is best score than that of the initial simple CNN architecture(initial unoptimized model) score i.e. 2.67118.

Results

Model Evaluation and Validation

During model creation and development , a validation set was used to evaluate the model.

The comparison of the Public Scores for all the model architectures considered for this data set is shown in Fig.10

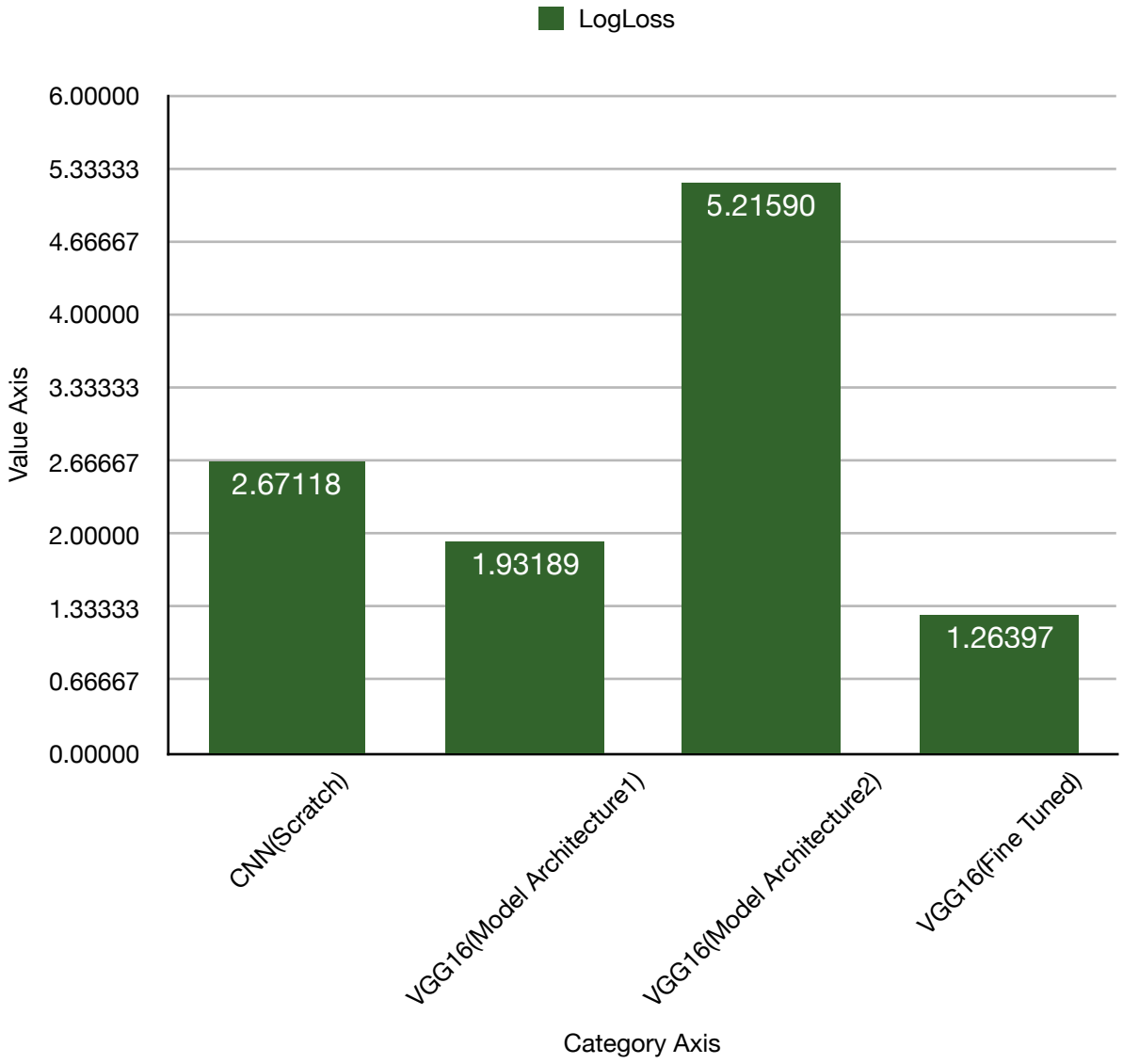

Fig.10

The final architecture chosen was the VGG16 fine-tuned model. This architecture along with hyper-parameters were chosen as they performed the best among all model combinations.

Following are the details of the final model parameters and training process :

1. VGG16 is instantiated and first 15 layers were frozen.
2. The last Conv layer i.e. Conv block 5 is fine tuned.
3. Our own layer(Global Average Pooling + Fully Connected layer) is added and fine tuned
4. Fine tuning is carried out with very slow learning rate and SGD optimizer
5. Training is carried out for 10 epochs with a batch size of 16

Finally the model is tried on the test data set. This resulted in Public Score of 1.26397. This can result in rank of 617 out of 1440 in Public Leaderboard i.e. in top 42.84%. It was observed the loss on validation dataset was 0.00751.This in caparison with public score on test dataset would suggest that there is overfitting. Inorder to resolve overfitting we need to consider adding/increasing the drop out and L2 regularization.

Model improvement against bench mark model

The model with the Public Leaderboard score(multi-class logarithmic loss) of 0.08690 is used as a benchmark model.

1.Using the bottleneck features of a pre-trained network" for VGG16 with model architecture1 was trained and tested resulted in rank of 1120 out of 1440 in Public Leaderboard i.e. in top 77.77%.

2."Fine-tuning the top layers of a a pre-trained network" was applied to VGG16 with model architecture1 as top layer. This resulted in further improvement of loss metric and when tested resulted in rank of 617 out of 1440 in Public Leaderboard i.e. in top 42.84%

This resulted in a Public Leaderboard score(multi-class logarithmicloss) of 2.67118 when predicted with the test dataset and submitted to Kaggle.

Justification

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

In order for further improve the model and reduce the loss, we need to take advantage of different algorithms and hardware(Refer “Improvement” section).

Conclusion

Free-Form Visualization

Below are some of images classified by the VGG16 fine-tuned model.

Fig.11

Fig.12

The first image in Fig.11 was misclassified as “c3: texting - left" although it is showing as driving. Similarly the second image in Fig.11 was misclassified as “c9: talking to passenger” although it is showing as talking on the phone - right

The image in Fig.12 was an example of correctly classified image. It is correctly classified as “c1: texting - right”

Misclassifications may be due to the following:

- Less clear image
- Too many objects(Clutter) in the image

Reflection

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

1. An initial problem is identified and relevant datasets were obtained
2. The datasets were downloaded and pre-processed
3. Relevant hardware is obtained from cloud to account for huge number of images in the dataset
4. The highest score in the Public Leadership board was taken as a benchmark and a target rank was defined.
5. An initial CNN was created from Scratch. After training this was tested and resulted in rank of 1362 out of 1440 in Public Leaderboard i.e. in top 94.58%
6. In-order for further improve the loss metric, pre-trained model such as VGG16 was used and transfer learning is applied
7. Technique such as “Using the bottleneck features of a pre-trained network” is applied to VGG16 for two types of model architectures as the top layer
8. “Using the bottleneck features of a pre-trained network” for VGG16 with model architecture1 was trained and tested resulted in rank of 1120 out of 1440 in Public Leaderboard i.e. in top 77.77%.
9. However when “Using the bottleneck features of a pre-trained network” for VGG16 with model architecture2 was trained and tested, did not result in any improvement of loss metric.
10. Finally “Fine-tuning the top layers of a a pre-trained network” was applied to VGG16 with model architecture1 as top layer. This resulted in further improvement of loss metric.

I found steps 8,9,10 to be difficult. This is because of the fact that training and testing took large time despite running on GPU instances. This may be due to model complexity and huge size of data.

The interesting part of the project is the use of transfer learning to obtain good score even with decent sized datasets. Also I also enjoyed using Keras with Tensorflow as the backend due to its ease of use and simplicity.

Improvement

Following are the areas for improvement

1. To further improve the score , we need to consider using bigger size when re-sizing the image. Currently the algorithm was trained by using 224x224 re-sized images. An image size of 480x480 might be relevant as original image size is 640x480.
2. Currently VGG16 architecture was only investigated during Transfer learning. We also need to try using the below pre-trained models and verify the score.

    - VGG-19 bottleneck features
    - ResNet-50 bottleneck features
    - Inception bottleneck features
    - Xception bottleneck features

3. Model Ensembles and use of advanced Image Segmentation algorithms such as R-CNN,Fast R-CNN,Faster R-CNN and Mask R-CNN can also be investigated.