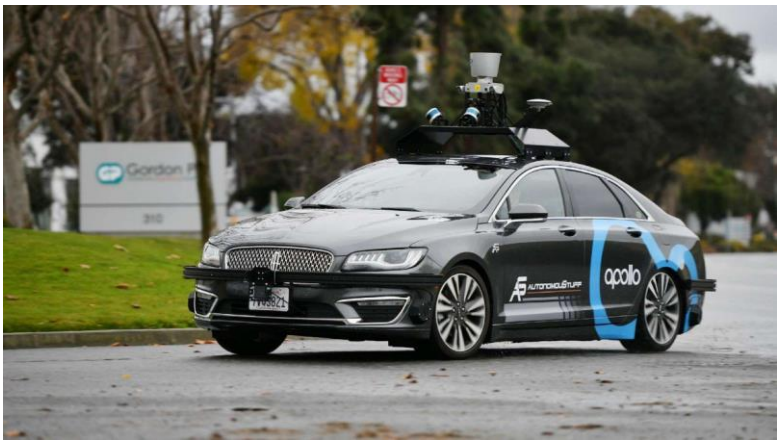
DEEP LEARNING

FOR

SELF DRIVING CARS



MACHINE LEARNING

CAPSTONE

TEJAS SHANBHAG

TABLE OF CONTENTS

PROJECT OVERVIEW

The project that I will be working on is the simulation of a self-driving car. Every year there are lot of deaths due to road accidents. According to World Health Organization there have been about 1.25 million deaths due to the road accidents in the year 2010.The main cause of accidents is over speeding and reckless driving. This problem motivated research in development of autonomous cars. The autonomous cars use sophisticated circuitry and sensors such as Lidars, Inertial Measurement Sensors, Proximity sensors, GPS, computer vision. The control systems interpret these sensory signals and help to plan the navigation for the vehicle. We would be exploring Computer Vision and Neural networks to simulate a self-driving car.

PROBLEM STATEMENT

The goal of this project is basically to create a deep learning model using Convolutional Neural Networks and Deep Neural Networks to simulate a self-driving car .The final Objective is to correctly predict the steering angles depending upon the curvature of the road .The Neural Network model should be capable to accurately extract the road features and output the correct values of the steering angles .The various tasks that need to be carried out for successful implementation of the project are:

- The first and the most important part is the preparation of the dataset. The dataset consists of about 40,000 images while the car was driven and the measured steering angles for each of the images.

- The next part involves designing a neural network and train it on the input images.

- The final task is to visualize the images as well as the steering angles. The steering wheel should make appropriate turns according to the curvature of the road.

ACCURACY METRICS

The problem we have defined involves predicting the steering angles with the image as the input. Hence this is a regression problem because we are predicting continuous values instead of a discrete or a binary output. Our accuracy would be best measured by the difference between the predicted value and the true value i.e. the difference between the predicted steering angle and the true steering angle. So, the best evaluation metric which computes this deviation would be **Mean Squared Error**. There can be other metrics available for regression problems such as R score or root mean squared error.

The mean squared error can be calculated as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Here Yi represents the true value and Y^i represents the value which would be predicted by the model. We take the mean of all the examples to finally get the mean squared error. Eras has a built in metric for mean squares error.

However, keras do not have built in methods for R2 Score and Root mean squares error and we write custom functions to evaluate the built model using these metrics.

The root mean square is the mean of square of the difference between the predicted and the true value and can be illustrated as,

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Also, the R2 score could be used as an evaluation metric when dealing with regression problems. The R2 score can be calculated as:

$$r^2 = \frac{SSR}{SSTO} = 1 - \frac{SSE}{SSTO}$$

$$SSR = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2 = 6679.3$$

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = 1708.5$$

$$SSTO = \sum_{i=1}^{n}(y_i - \bar{y})^2 = 8487.8$$

- SSR is the "regression sum of squares" and quantifies how far the estimated sloped regression line, y^i, is from the horizontal "no relationship line," the sample mean or $\bar{y}$.
- SSE is the "error sum of squares" and quantifies how much the data points, yi, vary around the estimated regression line, y^i.
- SSTO is the "total sum of squares" and quantifies how much the data points, yi, vary around their mean, $\bar{y}$.

ANALYSIS

Data Exploration

The dataset consists of about 45,000 images of the road when the car was driven. The dataset can be downloaded at the following link: https://GitHub/Sullychen. The dataset is prepared by recording a 25 minutes video by a 30 FPS camera. Hence a 25-minute video accounts to 25x60x30 =45,000 images. Also, an angular displacement measuring instrument was used to record the steering angles at each instant. Considering 80:20 train test split about 36,000 images would be used for training. The images are colored, and each image has a pixel of 500x500.The dataset also consists of a text file containing the image path and the steering angle recorded for each image.

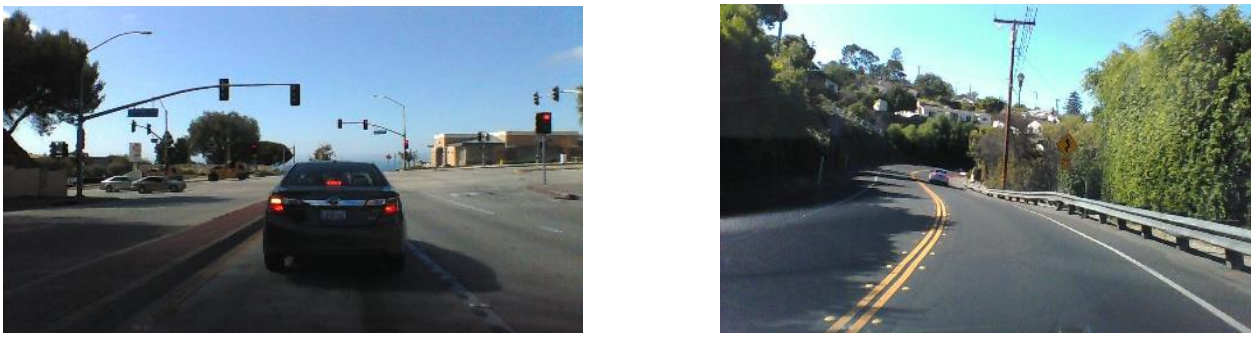


Fig.1 Images of the driving dataset

Exploratory Visualization

The dataset mainly consists of images and a text file containing the steering angles for each image. We can carry out some exploratory visualization to find the range of the steering angles in our dataset. We can also plot a Histogram to visualize the frequency distribution of the steering angles and find the value of the angles at which the steering angles are centered around. The Histogram is basically a plot that helps us to find the frequency distribution of a continuous data. It helps us to identify various characteristic present in the data like its normal distribution, skewness and to find out if there are any outliers present in the data.

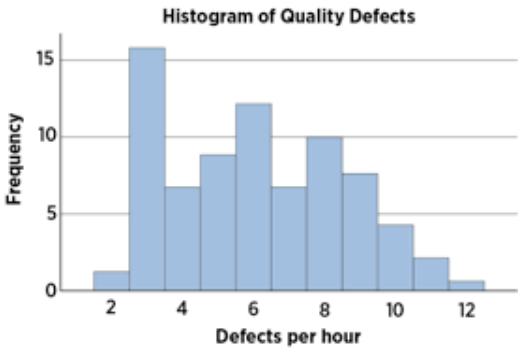


Fig.2 A sample Histogram

Several Python packages are available for data visualization such as Matplotlib and Seaborn. Hence the histogram plot using the steering angles as the data points can be visualized as

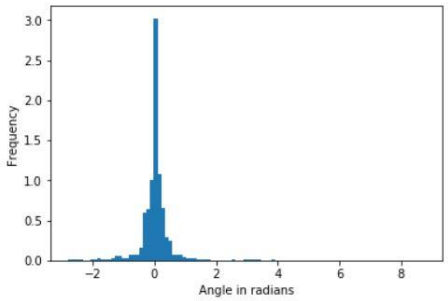
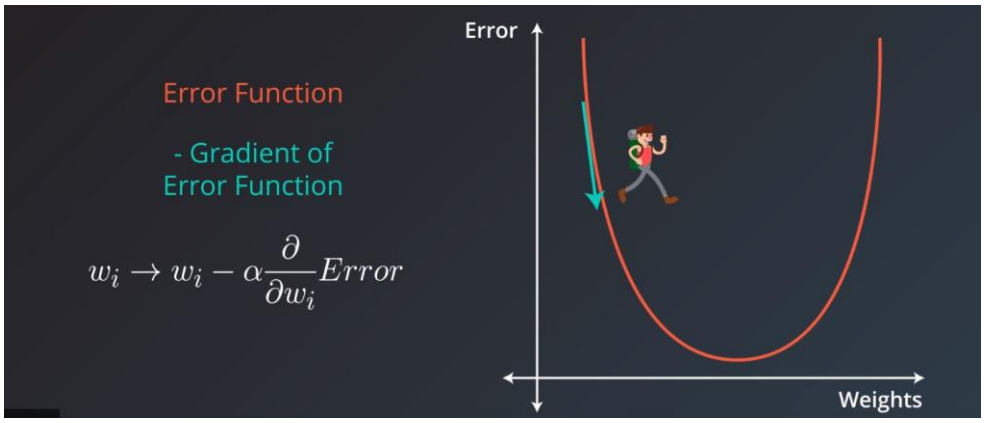


Fig.3 Plot of the histogram to visualize the frequency distribution.

The steering angles are centered near zero and have a mean nearing 0 radians.

Algorithms and Techniques

The model which is finally chosen for our application consists of a combination of convolutional neural networks and the deep networks. The CNN are found to perform extremely well when it comes to image classification task. CNN stands for Convolutional Neural Networks. They perform extremely well as compared to deep fully connected neural networks. CNN can accept input in form of a matrix and unlike fully connected dense layers they do not require the input to be flattened. This helps to preserve important features in the image and also require less number of parameters. They consist of filters which slide through the entire image and after a convolution produce an image with different sizes depending upon the number of strides. A stride of 2 produces an output image which is half the size of the input image. Initially 5 CNN's would be used and their output would be fed to 4 deep layers and 5 output layers. The Convolutional Neural Networks use the traditional gradient descendant algorithm to adjust their weights and fit the model. Gradient descent employs backpropagation to minimize the error until it converges to a global minimum. Below figure summarizes this. (picture courtesy – Udacity)



However, that are many parameters which can be tuned to increase the accuracy of the model. The first task is to split the images into train and test sets. The general approach during splitting is to randomly split the entire dataset but for this application the images are in a certain sequence so randomly splitting the data would not be a good approach. The images are then fed to the first CNN layer in form of small batches of size 20. It is then passed through certain hidden layers and finally through an output layer. The output layer has a 'tanh' activation function because the problem we are aiming to solve is clearly a regression problem and for each image there should be a continuous output value. A particular label or a Class is not being predicted, so it is not feasible to use the SoftMax activation function at the output. Several parameters such as neural network architecture, filter size, learning rate, optimizers, number of epochs, learning rate could be slightly varied to fine tune the model to achieve a better performance.

Benchmark

Several different types of networks such as shallow neural networks, deep neural networks. The goal was to minimize the mean squared error and the validation loss. The shallow Neural Networks performed poorly in the task thus setting a very poor benchmark. To create an initial benchmark a two layer feed forward network consisting of two hidden layers was then used. This model performed better than shallow networks but had considerably high mean squared error and didn't accurately predict the steering angles.

- The goal was to create a model which performs better than the initial benchmark.
- In terms of the mean squared error the goal was to bring the value below 1.
- The main objective of the final model would be to accurately predict the steering angles based on the curvature of the road.

METHODOLOGY

Data Preprocessing

Almost all the machine learning and data science projects require some form of data preprocessing. Preprocessing is the most important step in any Machine Learning project. The performance of any classifier is mainly dependent on the data being processed. If the data is unstructured or contains outliers, skewness then the performance goes down. Preprocessing steps involve dealing with Nan values, outliers and bringing the entire data into a correct format which could be used by classifiers for training.

This application uses images as the data to be processed. Lot of preprocessing has to be done on the images as well. The important preprocessing steps that have been incorporated in this application are listed below.

- First and foremost, it is important to split the data into training and testing data. The training data will be used to train the neural network and the test data will be used to evaluate the performance of the model. An ideal split to for train and test is in the ratio 80:20. However this data follows a certain sequence, so it would not be a good idea to split the data randomly. Hence first 80% of the data is considered for training and the remaining 20% of the data would be used for testing.

- The images that are being used are in .png format. Any image needs to be converted into an array or a tensor before it can be fed to a neural network. So, the first step involves converting the RGB image into a tensor. The path_to_img function displayed in fig4 convert an input image to a corresponding tensor.

```
def path_to_tensor(img_path):
    img = image.load_img(img_path, target_size=(66, 200))
    # convert PIL.Image.Image type to 3D tensor with shape (66, 200, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor
    return np.expand_dims(x, axis=0)
```

Fig.5 Function that converts input image to the corresponding tensor

- The next most important step during data preprocessing is **Normalization**. There are several different ways of Normalizing the data. This project deals with normalizing the pixel values. The black has an intensity of 0 while white color has the maximum pixel intensity of 255.Hence all the corresponding pixels in the image are divided by a pixel value of 255in order to normalize them.

- Also, the size of the input image sis 500 x 500 pixels. Nvidia end to end CNN architecture would be used to build the model architecture. This is a highly robust model specifically developed by Nvidia for self-driving cars. The first layer is a CNN and the documentation suggests that the size of the input image being fed to the first layer must be about 66x200 pixels. Hence it is necessary to center crop the image of size 500x500 and bring it to a value of 66x200 pixels.

Implementation

Once the data has been processed and is in the correct format the next step is the implementation of algorithms on the data to realize the objective and solution. The important stages of the implementation process for this application are:

- Define and build the model.
- Choose appropriate hyperparameters such as learning rate and select the best optimizers.
- Train the model.
- Visualize the performance on test data.

The entire project is built in **Python** language. Several built-in packages and libraries are used. Some of them have been listed below:

- **Matplotlib:** Matplotlib is a plotting library in python which helps to plot graphs, figures etc. pyplot module further simplifies the process and provides MATLAB like interface for plotting.
- **Keras:** It is a high-level Neural Network API which uses TensorFlow backend. Keras allows user to build, compile and train Neural Network models.
- **OpenCV:** It stands for open source computer vision library. It supports java ,python and C++.It is a powerful library which helps in manipulating and visualizing images.
- **SciPy**: It is a free and open source python library used for scientific computing and technical computing.

Each of the project implementation stage has been elaborated below.

Model Development Stage

The architecture of the model plays an important role in the performance of a neural network. Mainly the Convolutional Neural Networks are used for this application. CNN ‘s have greatly revolutionized the image classifier models and are one of the widely used models when it comes to image recognition tasks. Unlike fully connected neural network the CNN doesn’t requires the input image to be flattened. CNN requires very a smaller number of parameters which makes the training process extremely efficient.

The model architecture for this application would be referred from NVidia’s end to end CNN model which was specifically developed for its application in self-driving cars. Fig 6 clearly represents the model developed by NVidia. It consists of five convolutional layers. The first three convolutional layers have a filter size of 5 x 5 and a stride of 2. The next two convolutional layers have a filter size of 3 x 3and a stride of 1. Finally, the output of the fifth convolutional layer is flattened and is fed to a fully connected layer with 1164 neurons. There are 4 fully connected layers of size 1164 ,199, 59,10 and finally an output layer which has a tanh activation function.

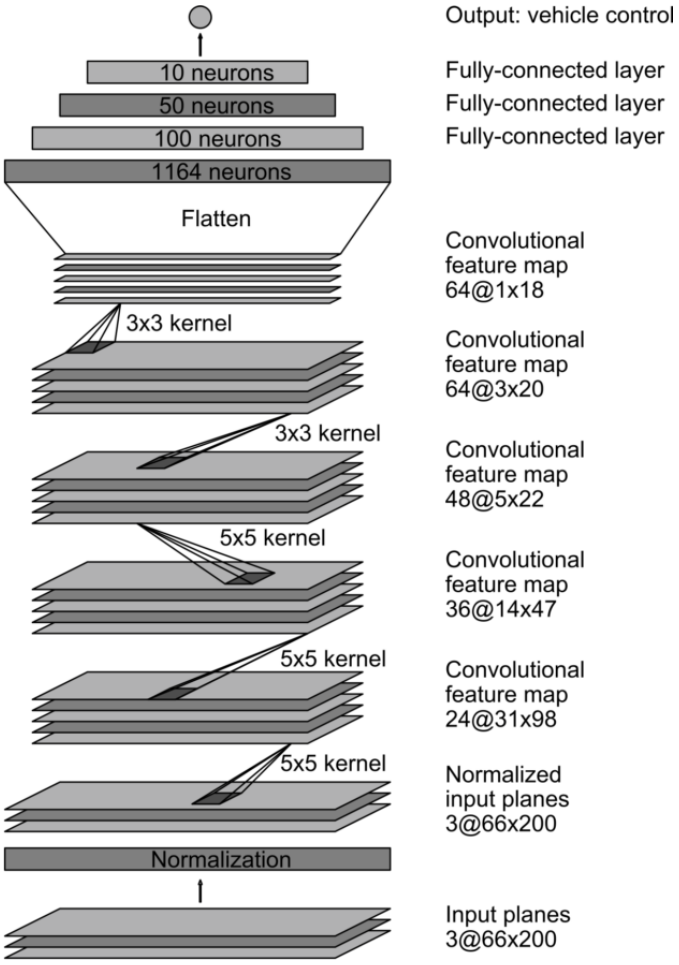


Fig.6 Nvidia CNN Architecture

Choosing Hyperparameters and Optimizers (Model Compilation)

Choice of the hyperparameters and Optimizers play an important role in the performance of a model. A number of optimizers were used and different learning rates were tried. The best results were obtained by using Adam as the optimizer and the learning rate was chosen to be 0. 0001.This problem is a regression problem, so the accuracy metric would be the Mean squared Error which is the difference between the predicted value and the true value.

Training and Visualization

Once the model is compiled it is then trained for a specific number of epochs. Training Neural Networks require a lot of computing power. The model can either be trained on CPU (Central Processing Unit) or on GPU (Graphic Processing Unit). Training on CPU takes a lot of time . Nvidia's Cuda package allows the networks to be trained via GPU.Training on GPU saves a lot of time and hence is computationally very efficient. One the model is trained on a given number of epochs one can save the best model weights. This could be used in the future to load the trained weights to further save the training time.

Figure 7 shows a block diagram of how CNN training works. Images are fed to the neural network and for each image a steering angle is predicted. The predicted steering angle is then compared to the actual steering angle and the model weights are adjusted to bring the predicted value close to the actual value and minimize the error in between them.

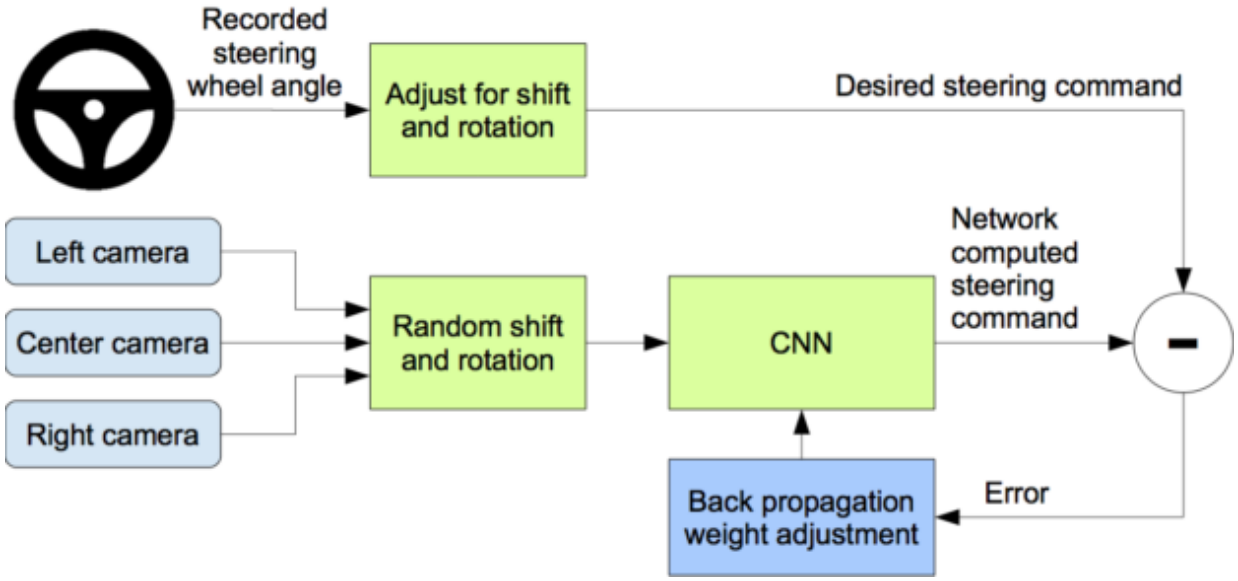


Fig 7: Training Block Diagram of CNN (Referred from Nvidia Research paper)

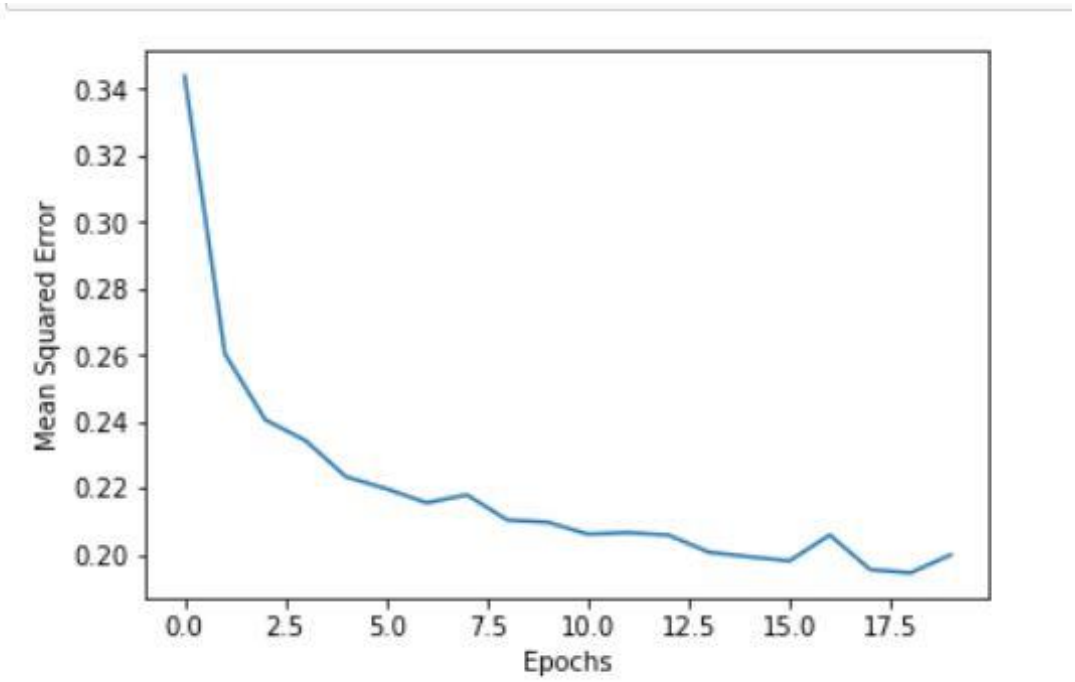
Finally, after training and loading the best model weights the final implementation step is the visualization of our model on real data. OpenCV module would be utilized to display two windows. One window would show the image and the next window would show the predicted angle for that image. Actual rotation of the steering wheel is incorporated to better visualize the predicted steering angles. We would loop through the entire data and visualize the steering angle and ensure it is in sync with the curvature of the road.

Refinement

Several hyperparameters were refined and fine tuned to achieve better performance and minimize the mean squared error. However, the Nvidia architecture by itself is found to perform extremely well and we can visualize that during initial stages of training that error is already very less. But to further reduce the error different optimizers and hyperparameters are varied to fine tune and refine the network.

After trying different hyperparameters it was found that the model performance was best when the learning rate was 0.0001 and optimizer was Adam.

Below figure summarizes the performance of the model when trained for 20 epochs.



RESULTS

Model Evaluation and Validation

After thorough testing and trying out different architectures, optimizers and varying the hyperparameters the final model is chosen. Following are the features of the model which was finally selected and used to train on the input data.

1. Firstly, the model requires the input image to be in the shape of 66 x 200 pixels. So, the input image is processed and converted into the tensor of required shape.

2. The first three convolution layer has a filter size of 3x3 and a stride of 2. The padding is set to valid. The output image after passing through first three convolutional layers has a size of 5 x 22 and has a depth of 48.

3. Later the image tensor is passed through two more convolutions with a filter size of 3 x 3 and a stride of 1. The size of the tensor after a total of five convolutions is 1 X 18 with a depth of 64.

4. The array or the tensor is finally flattened and the fed as an input to a fully connected later consisting of 1164 neurons. Subsequently it is then passed through three fully connected layers of size 100, 50, 10 and then finally through an output layer with a tanh activation function.

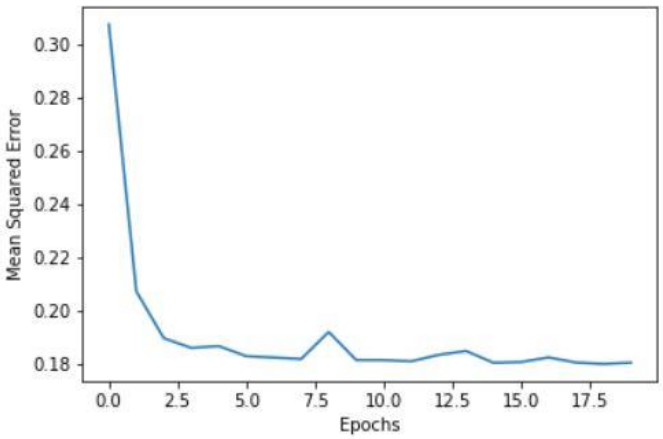
Once the model architecture was designed the model was compiled using Adam as the optimizer and the loss and the accuracy metric was chosen to be mean squared error as we want to minimize the error between the predicted steering angle and the true steering angle.

Finally, the model is trained for hundred epochs and the best weights are saved in the directory which could be later used to load the model.

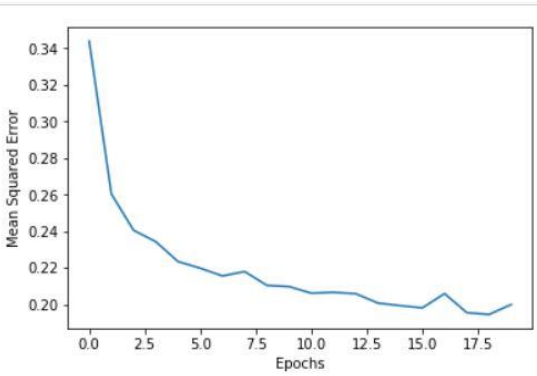
However, the model was then run on a set of training and validation images to ensure it fulfilled the intended functionality. During training and validation, the following observations were made:

- For every image the model does predict a value for the steering angle.
- The model is accurately able to identify the curvature of the road and correctly predicts the steering angle depending upon whether the road turn towards left or right.
- As the number of training epochs is increased the mean squared error decreases.
- The model performs well not only on the training set but also on the test set

Model performance on test set

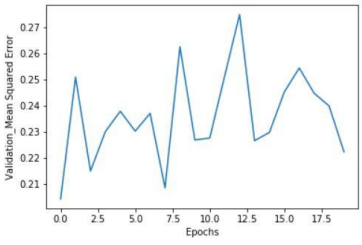


Mean squared and R2 on train set error on train set for 20 epochs



Model performance on Test set for 20 epochs

```
# mean squared error after 50 epochs
plt.plot(histor.history['val_mean_squared_error'])
plt.xlabel("Epochs")
plt.ylabel("Validation_Mean Squared Error")
plt.show()
```



The model performs extremely well on the test set as well. It can be seen in the plot that the validation mean square error is somewhere near 0.22 after 20 epochs which is extremely good. Training the model for more number of epochs which greater computing capabilities will increase the performance and further reduce the error.

Justification

The Visualization of the final output of the network on real images proves that the model performs extremely well in detecting the road and correctly predicts the steering angles. This could be a great application specially for self-driving and autonomous cars.

We had initially set a benchmark that our model should correctly predict the steering angles and should have a mean square error less than 1.

The evaluation of the performance and the plots show that the mean square error on the train set is near 0.17 and on validation set is near 0.22 which is extremely well. Hence this model fulfills the set benchmark and could be a great solution which could be applied in self-driving cars.

However, there are some limitations of this model as well. This model has no arrangement for speed control and braking. Also, there is no mechanism to detect any obstacle or pedestrians on the road.

Hence it can be seen that Neural Networks for feature detection is a great concept and provides a good framework for realizing self-driving cars. However, the final prototype would need to be incorporated with sophisticated sensors, electronics and control systems to further add safety as well as speed control.

CONCLUSION

Free form Visualization

```
The Predicted steering angle is: [-5.017941] degrees
The Predicted steering angle is: [-3.2690754] degrees
The Predicted steering angle is: [-4.7446795] degrees
The Predicted steering angle is: [-5.834346] degrees
The Predicted steering angle is: [-4.6322455] degrees
The Predicted steering angle is: [-6.8706627] degrees
The Predicted steering angle is: [-9.45764] degrees
The Predicted steering angle is: [-8.855452] degrees
The Predicted steering angle is: [-8.197015] degrees
The Predicted steering angle is: [-9.233618] degrees
The Predicted steering angle is: [-8.632962] degrees
The Predicted steering angle is: [-9.480685] degrees
The Predicted steering angle is: [-11.519356] degrees
The Predicted steering angle is: [-13.198104] degrees
The Predicted steering angle is: [-10.687017] degrees
The Predicted steering angle is: [-5.901938] degrees
The Predicted steering angle is: [-2.6498075] degrees
The Predicted steering angle is: [-2.0594807] degrees
The Predicted steering angle is: [-1.0849901] degrees
The Predicted steering angle is: [0.84492284] degrees
The Predicted steering angle is: [1.4475853] degrees
The Predicted steering angle is: [0.4483015] degrees
The Predicted steering angle is: [-0.1587374] degrees
The Predicted steering angle is: [-1.2069285] degrees
The Predicted steering angle is: [-1.2332076] degrees
The Predicted steering angle is: [-0.53623015] degrees
The Predicted steering angle is: [-1.3577983] degrees
```

It can be clearly seen that the rotation of the steering wheel is in sync with the curvature of the road. Please watch the video included in the zip for a more deeper understanding of the project working

Reflection

The complete process in the realization of this projects could be summarized as:

1. Firstly, different problem domains were explored and a final domain related to autonomous vehicles was chosen.
2. Once the problem domain was identified, a clear and a concise problem statement was written.
3. Different datasets were explored and finally the driving dataset made by Sully Chen was used.
4. A benchmark as set for the application to fulfill the intended functionality accurately.
5. Later a network architecture built by Nvidia was selected and the model was built.
6. The model was compiled using different hyperparameters and optimizers and finally the best one was chosen.
7. The model was trained, and the best model weights were saved.
8. OpenCV was used to visualize the images and predicted steering angles.

Overall it was a great experience accomplishing this project. However, there were certain challenges faced. Firstly, finding a concise and a good dataset was a big challenge. Most of the datasets related to self-driving cars are huge and are more than 100gb.I am thankful to Sully Chen for developing a small and an efficient dataset with steering labels included in a text file. Secondly importing the dataset was a big challenge because it consisted of images as well as text file containing steering angles. Python 'open' function provided a good solution to read the text file as well as import the images. Also, I was a bit confused choosing a neural network framework. Finally I chose to work with keras. Last but not the least I was exploring different options of displaying the output as actual rotation of the steering wheel. OpenCV helped me tackle this problem.

Improvement

Almost all Engineering solutions are iterable and always there is a scope of improvement. No solution is perfect and should never be considered as the final solution that completely solves a problem. The Technology is advancing everyday and with the passage of time a new and efficient way to solve a problem is discovered.

This project also has a great scope of improvement which has been listed below:

1. The performance of a machine learning or a neural network is totally dependent on the dataset. The better the data, the better would be our model. This dataset is relatively small compared to the large driving datasets that are available. This dataset is around 2gb and was selected because of the computation limitations of my laptop. However, there are some datasets which range from 100 -200 GB. The model could be trained on such large dataset which would help the model to train on more images and capture more features. However, training the model on such large datasets would require a great computational power and powerful GPU.

2. Secondly only CNN and Deep Neural Networks are used for the model implementation. The model could be improved by combining Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) to achieve a better performance.

3. In the real-world implementation this could be integrated with sophisticated circuitry and control systems to increase the autonomy and including speed control and braking systems.

