IMAGE RECOGNITION USING CIFAR-10 DATASET

BHAVIKA BAGARIA (NUID: 001825309)
KARTHIK ENJETI (NUID: 001476969)
SAVANKUMAR PATEL (NUID: 001470065)

I. ABSTRACT

The goal of the project is to understand and implement Image Recognition and Classification of the CIFAR-10 dataset using Convolutional Neural Networks (CNNs). The current industry baseline accuracy is 82%. The aim is to try and match the accuracy and understand the nuances of building a Convolutional Neural Network. The result is to identify the optimal parameters to achieve justifiable accuracy.

II. INTRODUCTION

Image Classification is the task of assigning an input image one label from a fixed set of categories. This is one of the core problems in Computer Vision that, despite its simplicity, has a large variety of practical applications. Many other seemingly distinct Computer Vision tasks such as object detection, segmentation can be reduced to image classification. As a significant application of machine learning and pattern recognition theory, image classification has become an important topic in individual's life. A few examples are that use this algorithm are Face recognition, vehicle detection and signature verification.

To classify the images, a CNN model is trained using Tensorflow. Image segmentation with CNN involves feeding segments of an image as input to a convolutional neural network, which labels the pixels. The CNN scans the image, looking at a small filter of several pixels each time until it has mapped the entire image. Pre-processing of data needs to be done before training the model. This is done by normalizing the data and one hot encoding. Once the pre-processing is done, the data is trained using the CNN model and tuning of hyperparameters is done.

The convolutional neural network (CNN) is a class of deep learning Neural Networks. CNNs represent a huge breakthrough in image recognition. They're most commonly used to analyze visual imagery and are frequently working behind the scenes in image classification. They can be found at the core of everything from Facebook's photo tagging to self-driving cars. They're working hard behind the scenes in everything from healthcare to security.

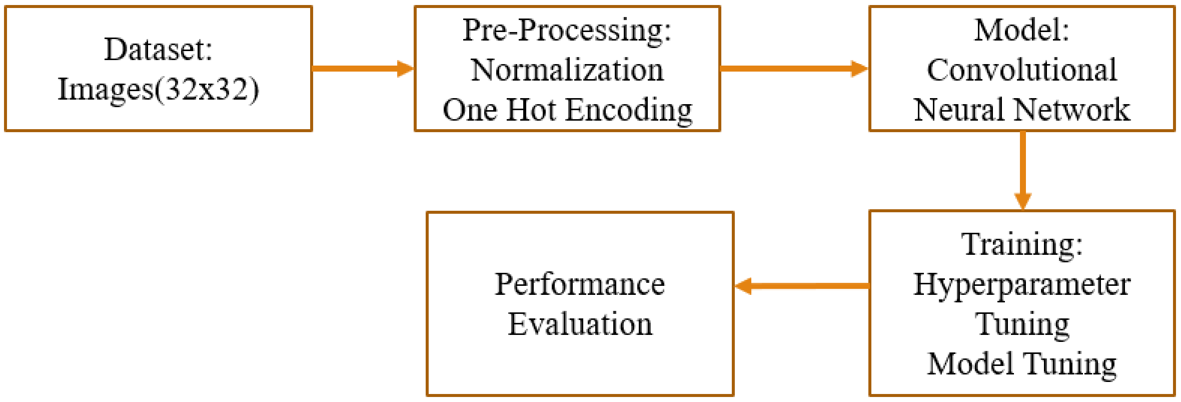
The reason behind choosing to build a Convolutional Neural Network model is because CNN can automatically extract features from the image. While if an algorithm with pixel vector is used, a lot of spatial interaction between pixels is lost; a CNN effectively uses adjacent pixel information to effectively down sample the image first by convolution and then uses a prediction layer at the end.

The dataset used is the CIFAR-10 dataset. It consists of 60,000 32X32 colored images which are divided into 10 categories (Airplanes, Automobile, Bird, Cat, Dog, Deer, Frog, Horse, Truck and Ship) and each category has 6000 images in which the dominant object in the image is of that class. The classes are designed to be completely mutually exclusive; for example, neither automobile nor truck contains images of pickup trucks. The images are split into training and testing sets. The training set contains 50,000 observations and the testing set contains 10,000 observations. The library used to process the dataset is Keras, a high-level neural networks API, which runs on Python and uses TensorFlow in the background. It allows for fast prototyping.

III. BACKGROUND

The dataset was created by MIT and NYU in 2009 together and is a reliable source to train an object recognition model. Multiple models have been implemented and the highest accuracy achieved is 82%. It was achieved by Jasper Snoek. CNNs have been agreed to be a reliable method to build a model for the same.

IV. APPROACH


Dataset: Images(32x32)
Pre-Processing: Normalization One Hot Encoding
Model: Convolutional Neural Network
Training: Hyperparameter Tuning Model Tuning
Performance Evaluation

4.1 PRE-PROCESSING

4.1.1 DATA NORMALIZATION

The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. To overcome the model learning problem, we normalize the data**.** We make sure that the different features take on similar ranges of values so that gradient descents can converge more quickly**.** Normalization avoids problems by creating new values that maintain the general distribution and ratios in the source data, while keeping values within a scale applied across all numeric columns used in the model.

4.1.2 ONE HOT ENCODING

Since the output of the model is going to show the probabilities of where an image should be categorized as a prediction, there should be a vector having the same number of elements as the number of image classes. CIFAR-10 provides 10 different classes of image, so a vector in size of 10 is needed. Each element represents the predicting probability of each classes. Also, the model should be able to compare the prediction with the ground truth label. It means the shape of the label data should also be transformed into a vector in size of 10 too. Instead, because label is the ground truth, the value is set to 1 to the corresponding element.

one_hot_encode function takes the input, **x**, which is a list of labels(ground truth). The total number of elements in the list is the total number of samples in a batch. **one_hot_encode** function returns a two-dimensional tensor, where the number of row is the size of the batch, and the number of column is the number of image classes.

4.2 BUILDING THE MODEL

The entire breakdown of the model is:

- 20 layers
- 6 convoluted layers:
    - Kernel size 3x3
    - Filter size: 16,32,64
    - Padding: Yes
    - Activation Function: Relu
- Batch Normalization
- Max Pooling: 2x2 and 3x3
- Dropout Layer: 0.25
- Flatten Layer
- Dense Layers : 1000 nodes x 2
- Final Dense Layer: 10 nodes ~ category ~ softmax

The model consists of 20 layers in total.

4.2.1 CONVOLUTED LAYERS

The main purpose of the convolution step is to extract features from the input image. The convolutional layer is always the first step in a CNN. There is an input image, a feature detector, and a feature map. Take the filter and apply it pixel block by pixel block to the input image. This can be done using matrix multiplication. Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. Sometimes filter does not fit perfectly fit the input image. There are two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

As for the activation function, ReLU is used as it does not saturate; the gradient is always high (equal to 1) if the neuron activates. ReLU is also very quick to evaluate.

4.2.2 MAX POOLING

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling reduces the dimensionality of each map but retains important information. **Max Pooling** is a type of spatial pooling. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality by keeping the max value(activated features) in the sub-regions binned.

4.2.3 BATCH NORMALIZATION

Batch normalization is a technique that standardizes the inputs to a layer for each mini batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

4.2.4 DROPOUT LAYER

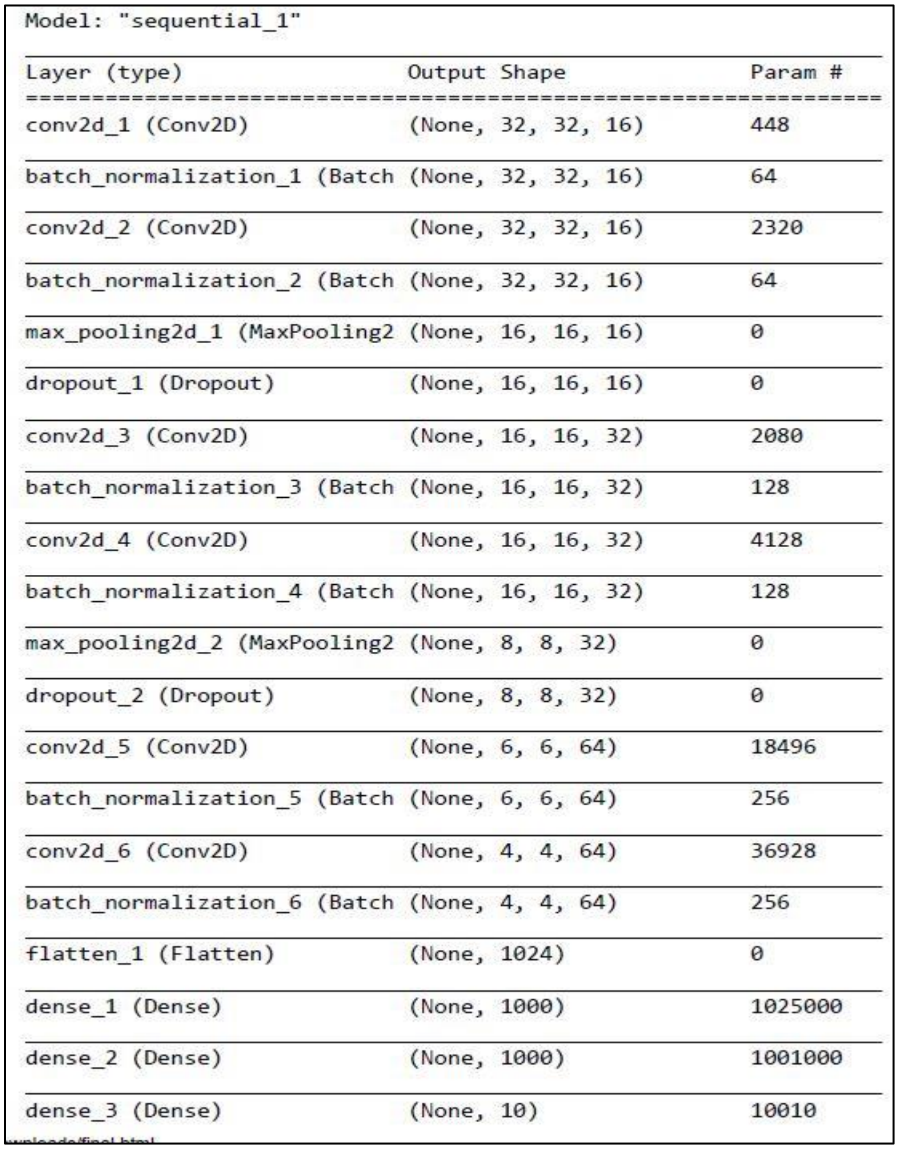
Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. It is used to prevent overfitting of the model.

4.2.5 FLATTEN LAYER

Between the convolutional layer and the fully connected layer, there is a ‘Flatten’ layer. Flattening transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier.

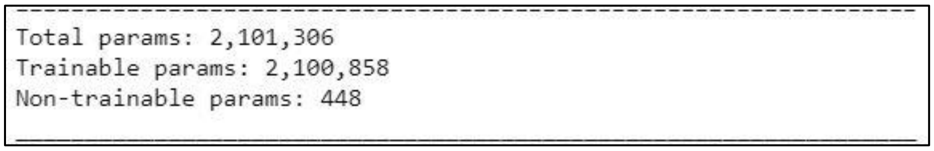
4.2.6 DENSE LAYERS

The dense layers are fully connected.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 16)        448
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 16)        64
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 16)        2320
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 16)        64
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 32)        2080
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 32)        128
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 32)        4128
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 32)        128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 6, 6, 64)          18496
_________________________________________________________________
batch_normalization_5 (Batch (None, 6, 6, 64)          256
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
batch_normalization_6 (Batch (None, 4, 4, 64)          256
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1000)              1025000
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              1001000
_________________________________________________________________
dense_3 (Dense)              (None, 10)                10010
```

Model Parameters



```
================================================================
Total params: 2,101,306
Trainable params: 2,100,858
Non-trainable params: 448
________________________________________________________________
```

V. RESULTS

5.1 HYPERPARAMETER TUNING

Activation Function – Different activation functions such as Sigmoid, tanh and ReLu were used. ReLu was found to be the most optimal activation function.

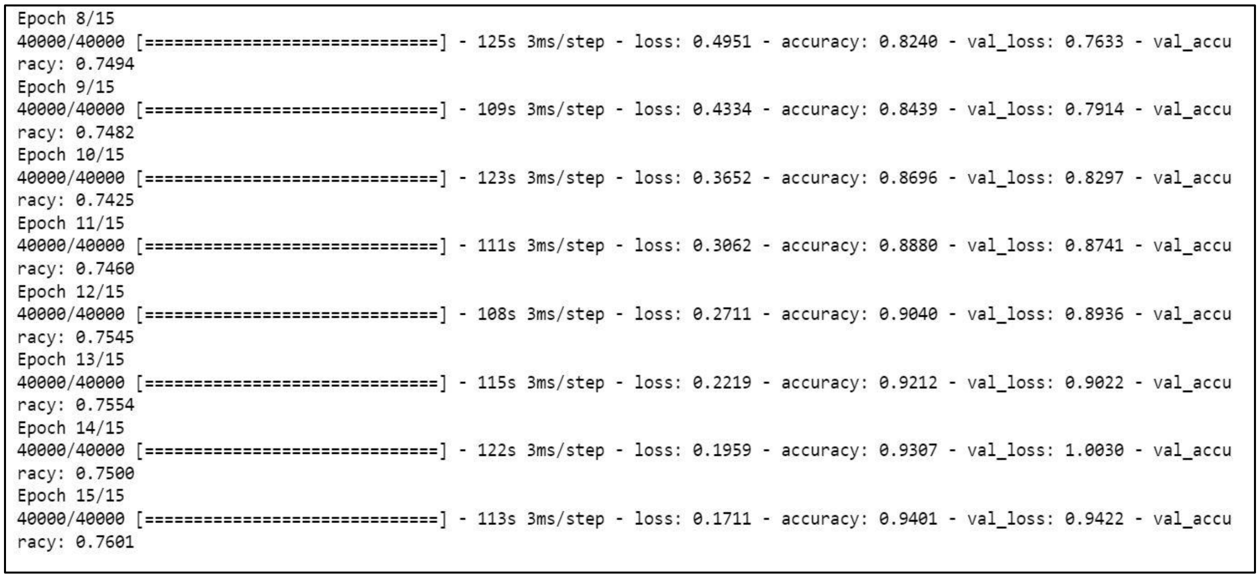
Epochs – Different values of epochs were used to compare the Validation Loss and Training Loss. The final decision of choosing the number of epochs was decided based on the value at which minimum validation loss occurred.

Loss – Categorical Cross-Entropy and Sparse Categorical Cross- Entropy options were considered and the final decision was to go with Categorical Cross-Entropy as it was a better fit for the model architecture.

Optimizer – Adam Optimizer was chosen as it is one of the most widely used optimizers to train deep neural networks.

5.2 PERFORMANCE EVALUATION

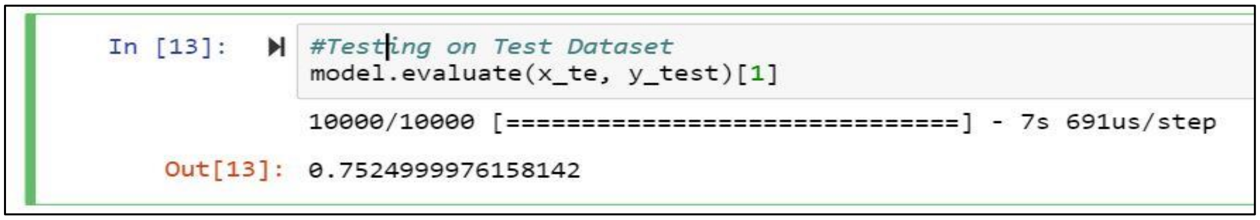
Once the model was implemented for 15 epochs, the testing accuracy was achieved at 75.24%
If the model is trained for more than 15 epochs, it seems to overfit.



```
Epoch 8/15
40000/40000 [==============================] - 125s 3ms/step - loss: 0.4951 - accuracy: 0.8240 - val_loss: 0.7633 - val_accu
racy: 0.7494
Epoch 9/15
40000/40000 [==============================] - 109s 3ms/step - loss: 0.4334 - accuracy: 0.8439 - val_loss: 0.7914 - val_accu
racy: 0.7482
Epoch 10/15
40000/40000 [==============================] - 123s 3ms/step - loss: 0.3652 - accuracy: 0.8696 - val_loss: 0.8297 - val_accu
racy: 0.7425
Epoch 11/15
40000/40000 [==============================] - 111s 3ms/step - loss: 0.3062 - accuracy: 0.8880 - val_loss: 0.8741 - val_accu
racy: 0.7460
Epoch 12/15
40000/40000 [==============================] - 108s 3ms/step - loss: 0.2711 - accuracy: 0.9040 - val_loss: 0.8936 - val_accu
racy: 0.7545
Epoch 13/15
40000/40000 [==============================] - 115s 3ms/step - loss: 0.2219 - accuracy: 0.9212 - val_loss: 0.9022 - val_accu
racy: 0.7554
Epoch 14/15
40000/40000 [==============================] - 122s 3ms/step - loss: 0.1959 - accuracy: 0.9307 - val_loss: 1.0030 - val_accu
racy: 0.7500
Epoch 15/15
40000/40000 [==============================] - 113s 3ms/step - loss: 0.1711 - accuracy: 0.9401 - val_loss: 0.9422 - val_accu
racy: 0.7601
```

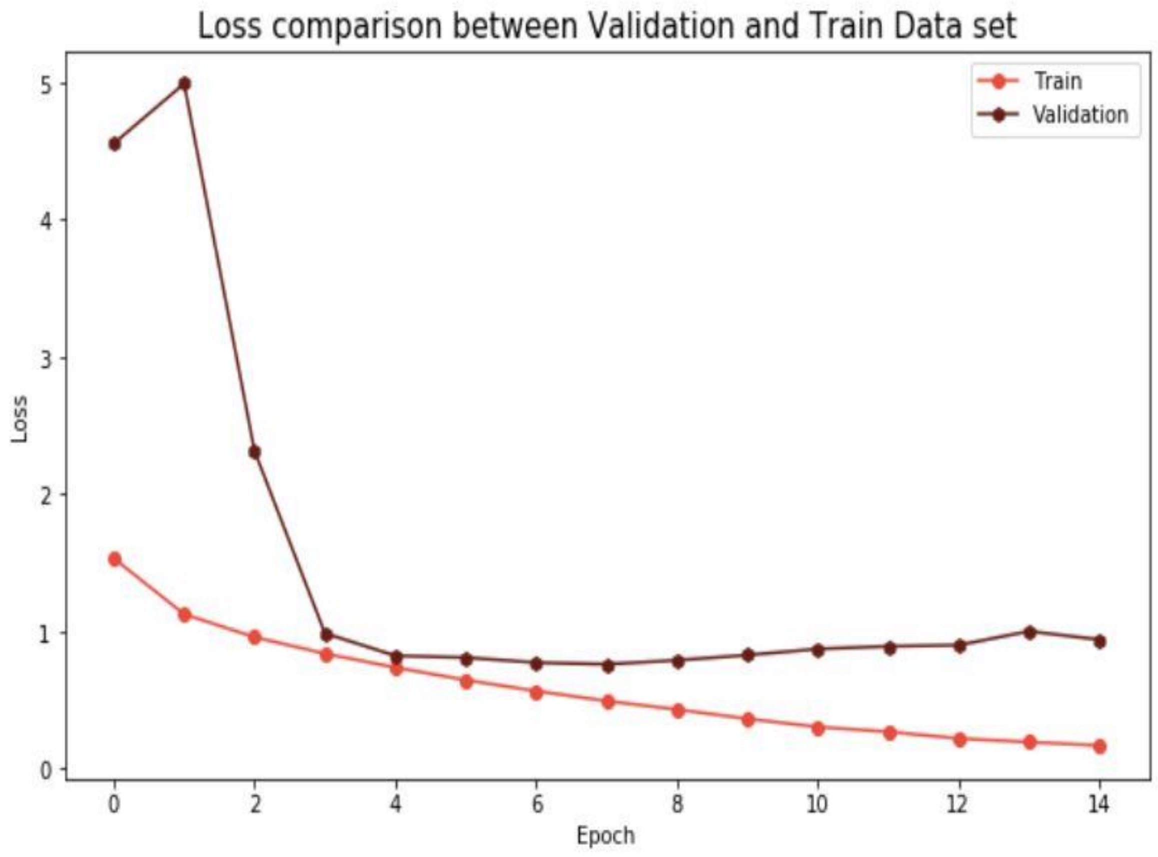
TESTING ACCURACY

With the final model we have achieved 75.24% accuracy on the test dataset.



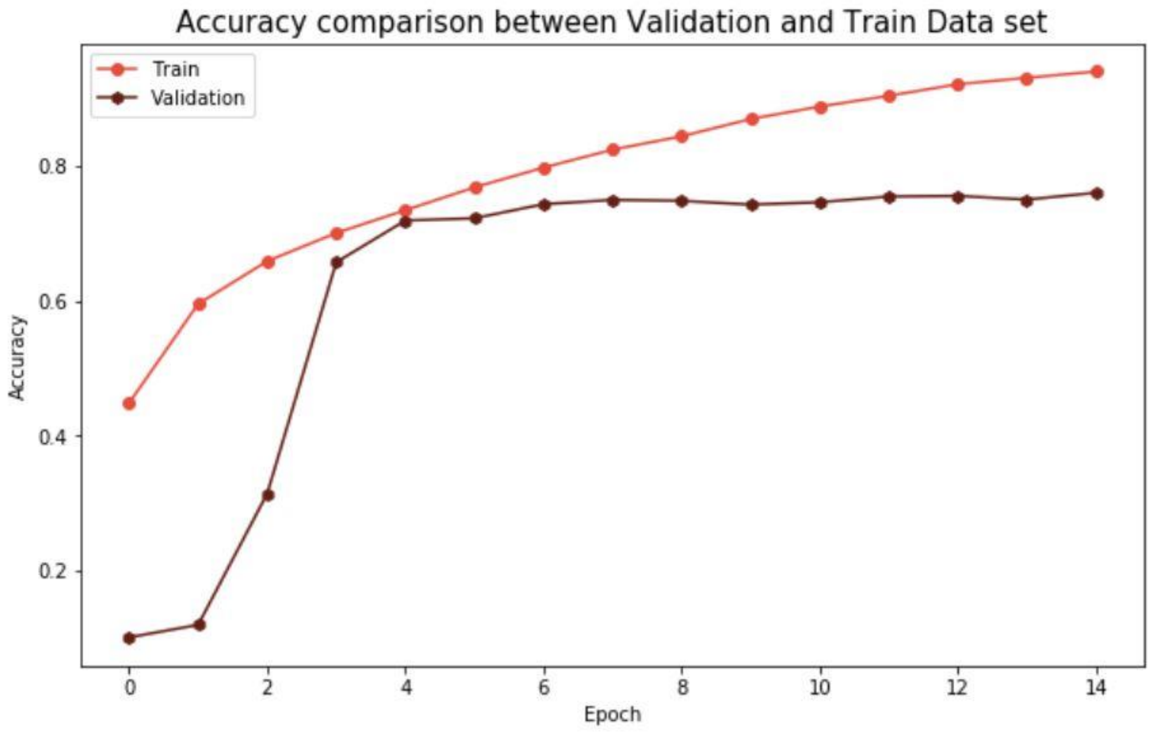
```
In [13]:  #Testing on Test Dataset
          model.evaluate(x_te, y_test)[1]

          10000/10000 [==============================] - 7s 691us/step

Out[13]:  0.7524999976158142
```

This graph plots the Loss Comparison between Validation and Train Data Set



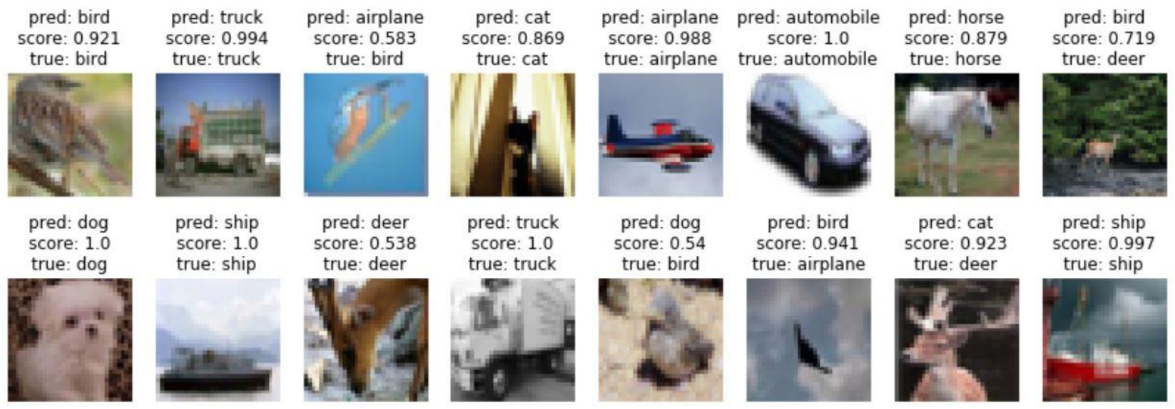
We notice through the above graph that minimal validation loss is seen at 15 epochs.

This graph plots the Accuracy Comparison between Validation and Train Data Set



Through the above graph best accuracy is seen at 15 epochs.

SCORES OF SAMPLES OF TEST IMAGE CATEGORIES



Above image shows a sample of test images and how they were scored by the model.

VI. CONCLUSION

The model has been trained to achieve a reliable accuracy even on different subsets of the dataset along with some images which are not part of the dataset. Through the implementation, the factors that go in to building a reliable model and how to fine tune a CNN have been explored.

There is a scope for improvement by slightly distorting the image and tweaking the image parameters, robustness of model can be increased, in turn achieving higher accuracy.

VII. REFERENCES

[1] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009.

[2] M. F. D. Rasim Caner Calik, "Cifar-10 Image Classification with Convolutional Neural Networks for Embedded Systems," IEEE, Aqaba, Jordan, 2018.

[3] X. H. Ming Liang, "Recurrent Convolutional Neural Network for Object Recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, 2015.

[4] "https://pythonistaplanet.com/cifar-10-image-classification-using-keras/," [Online].

[i]

[i] Equal Contribution by each member of the team.