ENPM 673 - Perception of Autonomous Robots

Pradeep Gopal
Sahana Anbazhagan
Srikumar Muralidharan

May,15 2020

1 Introduction

Data classification is the process of sorting and organizing data into categories so that it is easier for us to retrieve and use this data anytime in the future. As per our problem statement, we are required to classify the given data as dogs and cats. This classification is to be performed using Convolution Neural Networks. Convolution network is a type of artificial neural network which uses the concept of perceptrons for supervised learning to analyze data. A convolution layer has an input layer, output layer, and some hidden layers which consists of convolution layers that uses a mathematical model to pass on the results to successive layers.
In the given data set, we have 12500 testing data and 25000 training data that consists of dogs and cats. The training data is labeled and follows the format of dog/cat.0.jpg whereas testing data is unlabeled and the format is 0.jpg and so on.

2 Architecture of CNN

2.1 Convolution layer

A convolution layer in a neural network has the following components:

- Convolution kernels defined with their width and height
- Number of input and output channels
- The depth of the convolution filter must be equal to the number of channels of the input feature map.

Even though a fully connected feed forward neural network can be used to learn features as well as classify data, applying this architecture to images is not very practical. The convolution operation reduces the number of free parameters and allows the network to be deeper with fewer parameters. Usage of regularized weights over fewer parameters aids in avoiding the vanishing gradient and

exploding gradient problems that is generally seen during back propagation in traditional neural networks.

2.2 Pooling

Pooling layer reduces the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Pooling could compute a max value or an average value. Max pooling is generally used at times where the maximum value from each of a cluster of neurons at the previous layer is used, whereas, average pooling uses the average value from each of the cluster of neurons at the previous layer.

2.3 Fully connected layer

A fully connected layer is the one in which every neuron in one layer is connected to every neuron in another layer. The flattened matrix should go through a fully connected layer in order to classify images.

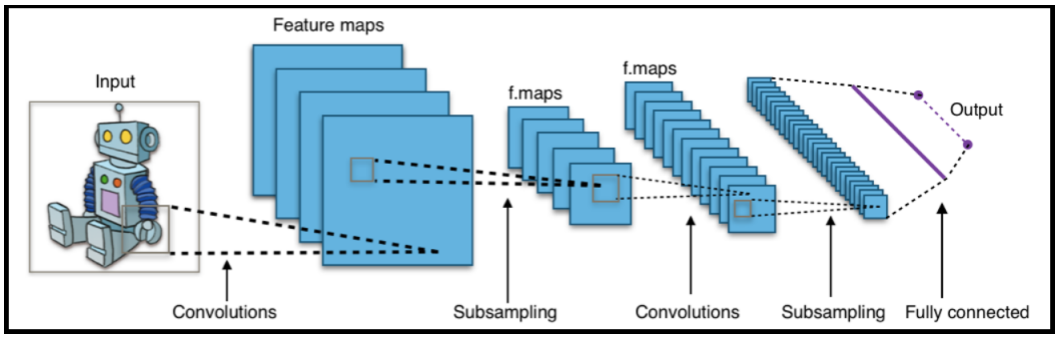

Figure 1: The basic CNN architecture

3 Implementation

We have implemented the project using TensorFlow and ran it on the GPU. For this we had to install the TensorFLow modules and we also installed Cuda so that we could run it on the GPU for quicker results. We also used TensorBoard for viewing the accuracy plots. TensorBoard is a tool which provides the measurements and visualizations needed during the project workflow. This enables tracking experiment metrics like loss and accuracy, visualizing the model graph among a few others.

As the first step, a function was written to convert the given data into a 2 dimensional array of images and labels. This function uses shuffle to modify variables in place so that we don't have to redefine it elsewhere. This function

also saves both the variables and return the array. So if wee change anything in the neural network and no changes in the image or the image size then this function can be loaded. This saves a lot of the processing time. This function has been written for training data and testing data individually. A little bit of preprocessing required was also performed in this function itself.

The next step in the process is to build the neural network. To do this we needed to import a few modules which are listed below.

```
import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression
```

We have used 7 convolution layers to obtain a better accuracy. We first used the input data function where we give a size for each frame, which is initially defined as 200 in our case. In the 7 layers we have used a kernel size of 32, 64, 128, 64, 128, 64 and 32 respectively and each layer has 3x3 kernels, with 3 filters. These values were selected on trial and error basis for an improved accuracy. The the activation function of a node defines the output of that node given an input or set of inputs. We have done this for 1,2 and 15 epochs. We have used rectified linear activation function. The biggest advantage of using **ReLu** is the non-saturation of its gradient, which greatly accelerates the convergence of stochastic gradient descent compared to the sigmoid or tanh functions. Following every convolution layer is a layer of max pooling. After the 7 layers comes the fully connected layer. We have used **softmax** activation layer in this layer. We have used the **Adam** optimizer and **cross-entropy** loss function and we have kept our learning rate at 0.001 and we have set our momentum to be 0.9.

3.1 Challenges Faced and lessons learnt

- While installing tensorflow, we understood that for tflearn to function properly, we needed an earlier version. Being completely new to Machine learning, this was a steep learning curve point.
- We had problems with np.load. It was not compatible with the latest version of numpy, when we tried loading npy files that had the testing and training dataset. So, we had to load an alternate version as shown in the code and revert back to the older np.load after this operation was carried.
- Working with tflearn and Tensorboard integration was difficult as Tensorboard tends to be extremely buggy. We have managed to attach the graphs, but event files have not been attached in the final submission. Event files can be provided on request.
- Trade-of between learning rate and image size. Bigger the size and lower the learning rate did not guarantee good accuracy results. We had to do a

lot of trial and error to reach the final level of accuracy value we desired. We have implemented our own model rather than use a pre defined model, which lead to difficulties in loading meta files. This was simply overcome by placing the meta file created in the same folder as the code and making sure we pass a return statement earlier in the training validation function.

- We managed to reduce run-time effectively by saving models and loading them rather than using traditional methods like training and testing in the same run.

3.2 Result and conclusion

We implemented a custom model and tried to run the same with an image size of 100, having around 3 convolution layers for different learning rates and varied kernel sizes for each layer. We did not achieve very good accuracy. We had achieved validation accuracy around 50% and 80% for the first 2 trials. After we adjusted the parameters as mentioned above, we were able to obtain a validation accuracy of 89.2% and epoch 15 accuracy of about 98%. We were able to generate the following output graphs and we have also attached herewith a window that shows classification data for training data. In the two graphs shown below, the red curve represents the loss/accuracy curve for 1 epoch. The blue curve represents the loss/accuracy curve for 2 epochs and the orange curve represents the loos/accuracy curve for 15 epochs.

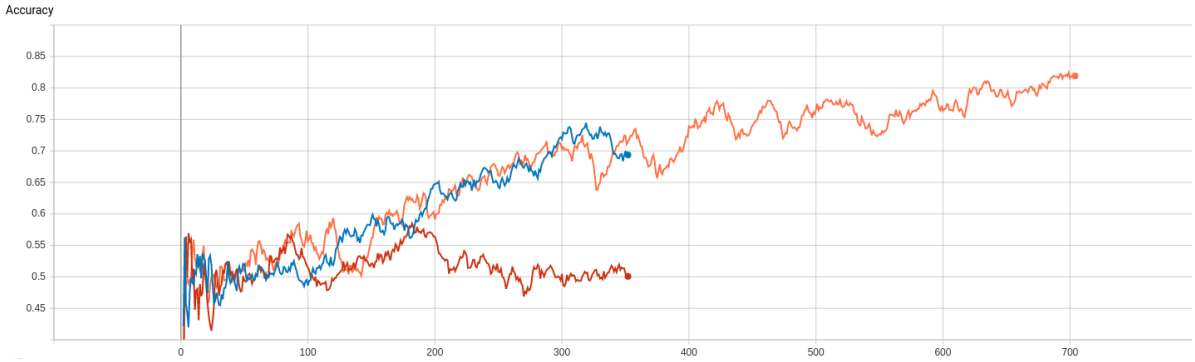

Figure 2: Validation accuracy vs epochs plot

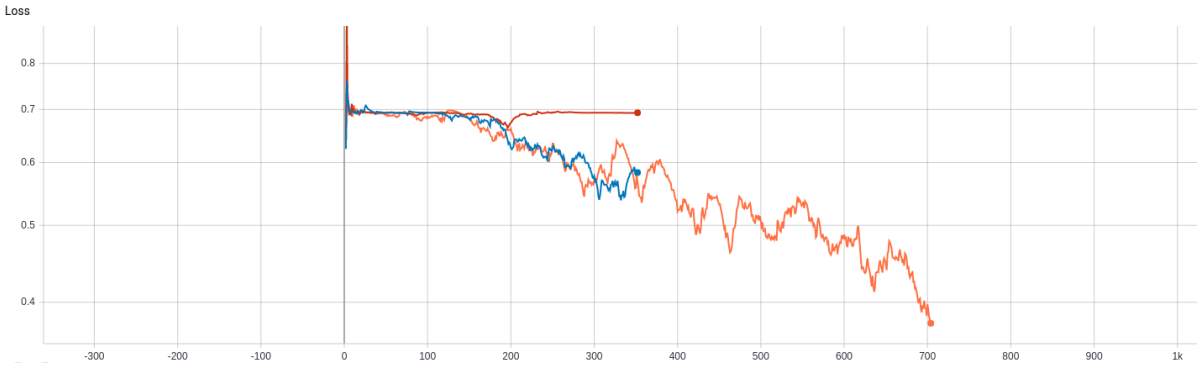

Figure 3: Validation loss vs epochs plot

Figure 4: Output for random images for 1 epoch

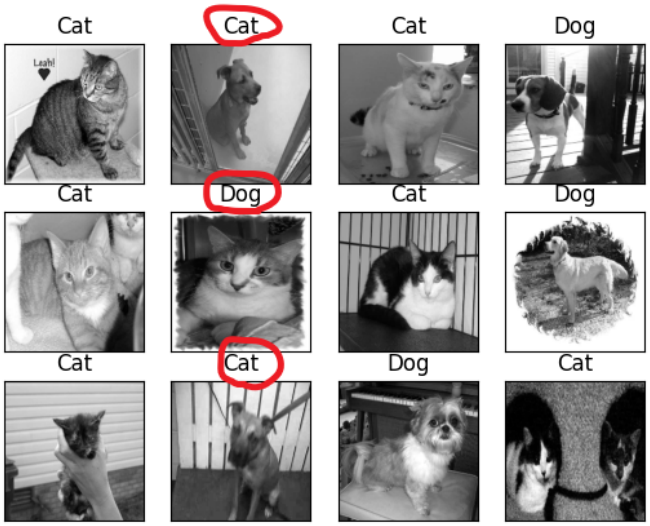

Figure 5: Output for random images for 2 epochs

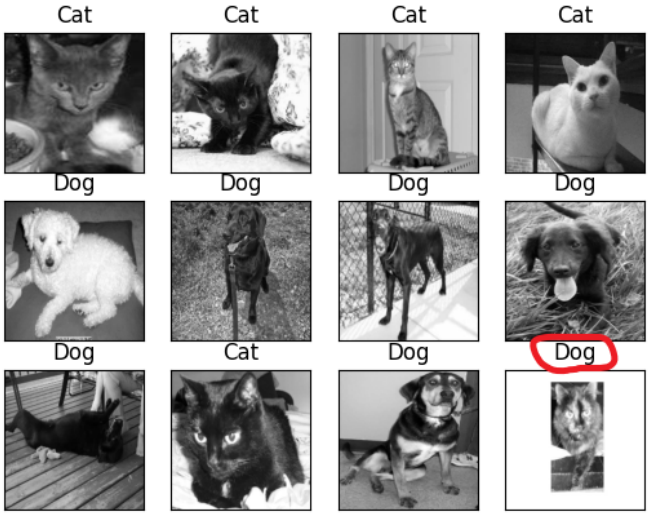

Figure 6: Output for random images for 15 epochs

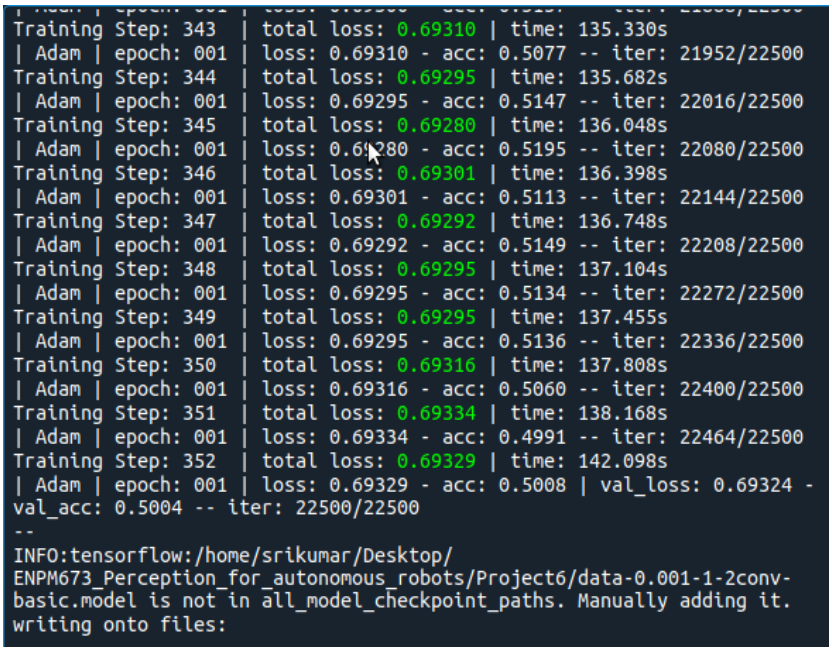

Figure 7: Accuracy output for 1 epoch

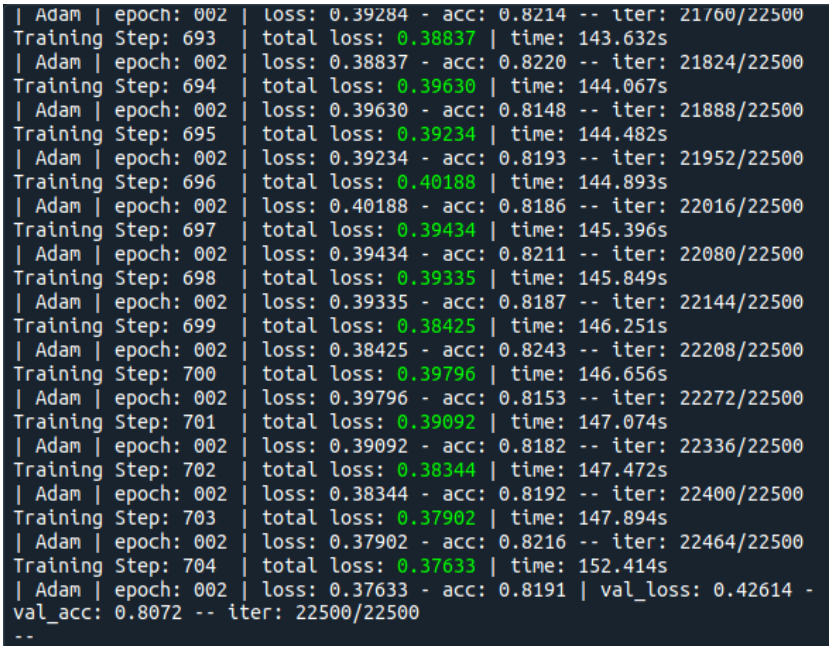

Figure 8: Accuracy output for 2 epoch

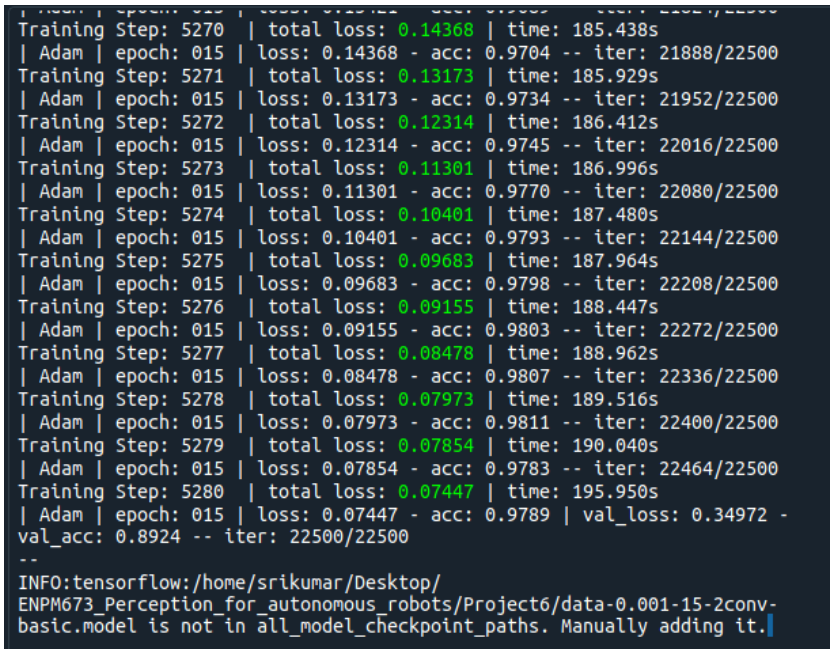

Figure 9: Accuracy output for 15 epoch