ENPM673: Perception for Autonomous Robots

Project 6

Umang Rastogi UID: 116773232

Sayani Roy UID: 116818766

Prateek Bhargava UID: 116947992

Date: 16 May 2020

1. Introduction

A Convolutional Neural Network (CNN) is a class of deep neural network, most commonly applied to analyzing visual imagery which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

In this project our aim is to classify a given set of data containing 25000 images into dogs or cats using CNN.

2. Pipeline

- First we perform pre-processing on our data-set where it is standardized. This was done using keras as the images in the dataset were of different sizes, thus being resized to 224 by 224, converted into a numpy array, classified and saved in two different folders cats and dogs.
- We re-scaled the pixel values from (0, 255) to (0,1).
- A model containing of 4 layers is defined.
- Batch Sizes are defined.
- Binary class is established due to rescaling between (0, 1).
- Training of the dataset begins, where first for validation we use half of the images in the dataset which are 12500 images and then for evaluation we use the full dataset.

2.1 Architecture

Convolution Layers are the major building blocks which are used in convolution neural networks. Convolution is the repeated application of a filter to an input that results in the activation of a map called feature map, which indicates the locations and strengths of a detected feature in an input, such as a dog or cat in our image.

In our model, we have used 4 layers out of which 3 were convolution layers and 1 was fully connected layer. The filter size of the convolution layers are 32, 64 and 128 respectively. The final fully connected layer also has a filter size of 128 in order to correct map with the last convolution layer. The input to the first convolution layer is of fixed size 224 x 224 RGB image, which is used to obtain the low level features of the image such as edges, orientation and color. The first convolution layer is used to get the low level features of the image such as color, orientation, edges, etc. With more and more convolution layer added to the network it is much more easier to get more high level features of the image. The image is passed through a stack of convolution layers, where the filters were used with a very small receptive field: 3×3. After the above operation padding is applied to the convoluted layer where either the dimensions of the convoluted image can be increased or it can be kept the same. In our case where we have already defined the dimensions we have kept the padding to be same so as to preserve the spatial resolution.

We then carry out Spatial pooling by three max-pooling layers, which follow some of the convolution layers. Max-pooling is performed over a 2×2 pixel window, with stride 2.

All hidden layers are equipped with the rectification (ReLU) non-linearity. RELU or Rectified Linear Activation is function which given an output for the input if it is positive and if no output is present it returns a zero value.

Then we perform Stochastic gradient descent (SGD), so as to enhance the results obtained, predicting the data not obtained before. Stochastic gradient descent is an iterative method for optimizing an objective function with suitable smoothness properties, as it replaces the actual gradient by an estimate thereof.

3. Results Obtained

The model, described above, is common for all 3 layer images. The model was optimized using Adam. It can be seen Fig.1 that the training accuracy has increased consistently and has gone above 90 %. However, the testing accuracy has consistently low which implies our model did not learn how to classify the images.
Next, the batch size was increased from 32 to 64 but there was no significant improvement in testing accuracy as seen in Fig.2a.
Since increasing the batch size did not work, we then tried data augmentation by adjusting the image height and width by a factor 0.1. This gave better results than the previous model. We can see in Fig.2b that the testing accuracy, however low, is still consistent unlike in Fig.2a.
The model was then changed to add dropout in each layer. A dropout factor of 0.2 was added after all the 3 convolution layers and a dropout factor of 0.5 was added after the dense layer. We see in Fig.2c that the results were better than model where batch size was increased (See Fig.2a) but worse than the model where data augmentation was done (See Fig.2b).

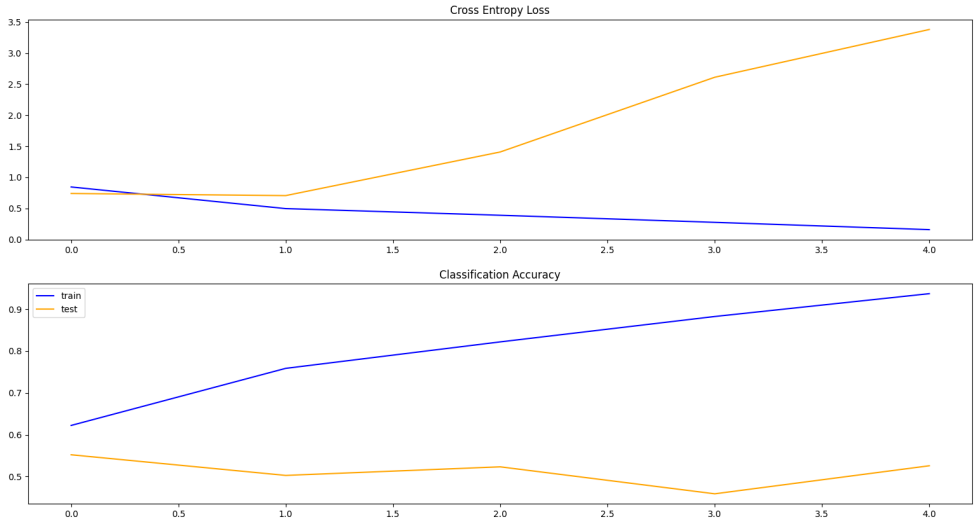


Figure 1: First Implementation of our 4 layer model

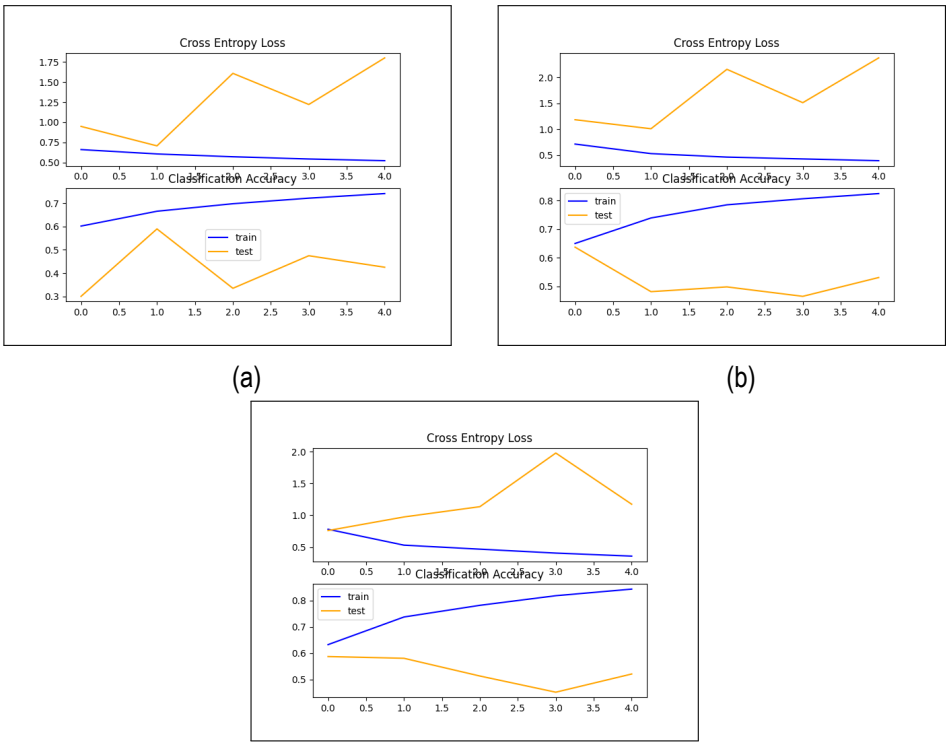


Figure 2: Accuracy and losses of various models created while tweaking training parameters

As we were not getting good results, the optimizer of the model was changed. In this model, the input size was changes from $224 \times 224 \times 3$ to $200 \times 200 \times 3$. Changing the optimizer led to us to some comparatively good results as seen in Fig. 3.

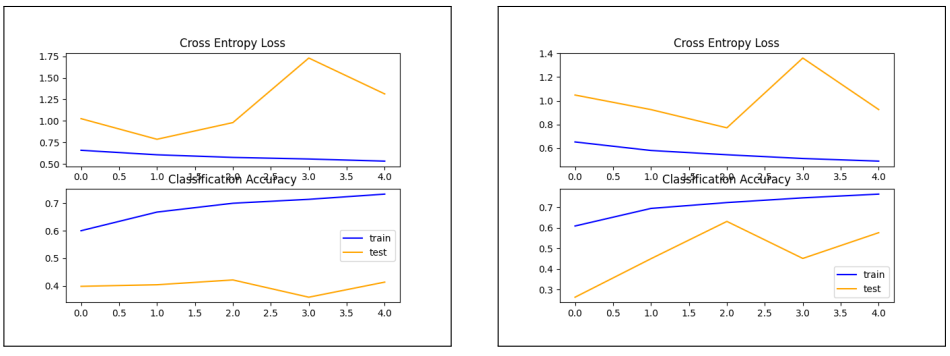
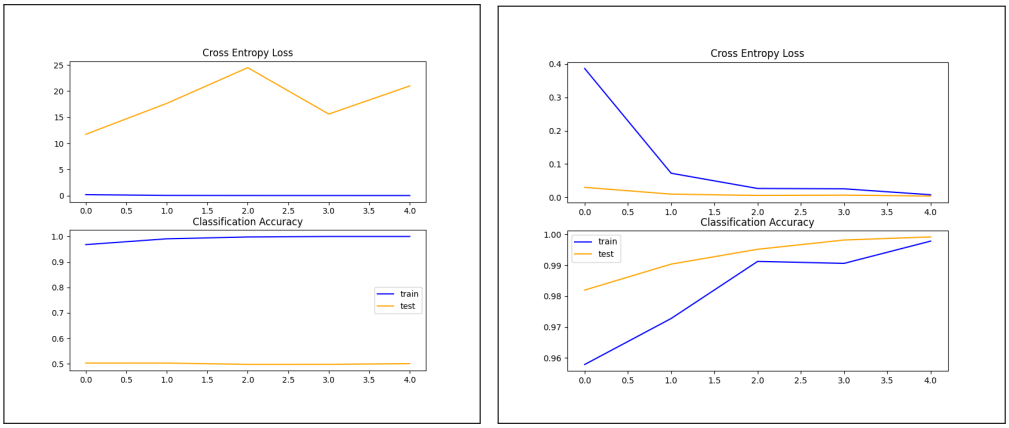


Figure 3: Accuracy and losses after changing the optimizer

To check if our implementation was correct, we used the pre-trained model VGG-16. It can be seen in Fig. 4a that the results were not good enough even with the pre-trained model. It was then realized that our implementation of the validation data-set was wrong. The VGG-16 model was then run again with the correct validation data-set and we obtained fairly good results as seen in Fig. 4b.



(a) (b)

Figure 4: Accuracy and losses of VGG-16 model before and after using correct validation data-set

This correction of validation data-set was then implemented in the original model and the results as seen in Fig. 5 were achieved. We first corrected our implementation in the our original 4-layer model with input image size as $200 \times 200 \times 3$ and optimizer as SGD. The final model was the same as above with addition of batch normalization and data augmentation. Test accuracy of our final model was 85.48%.

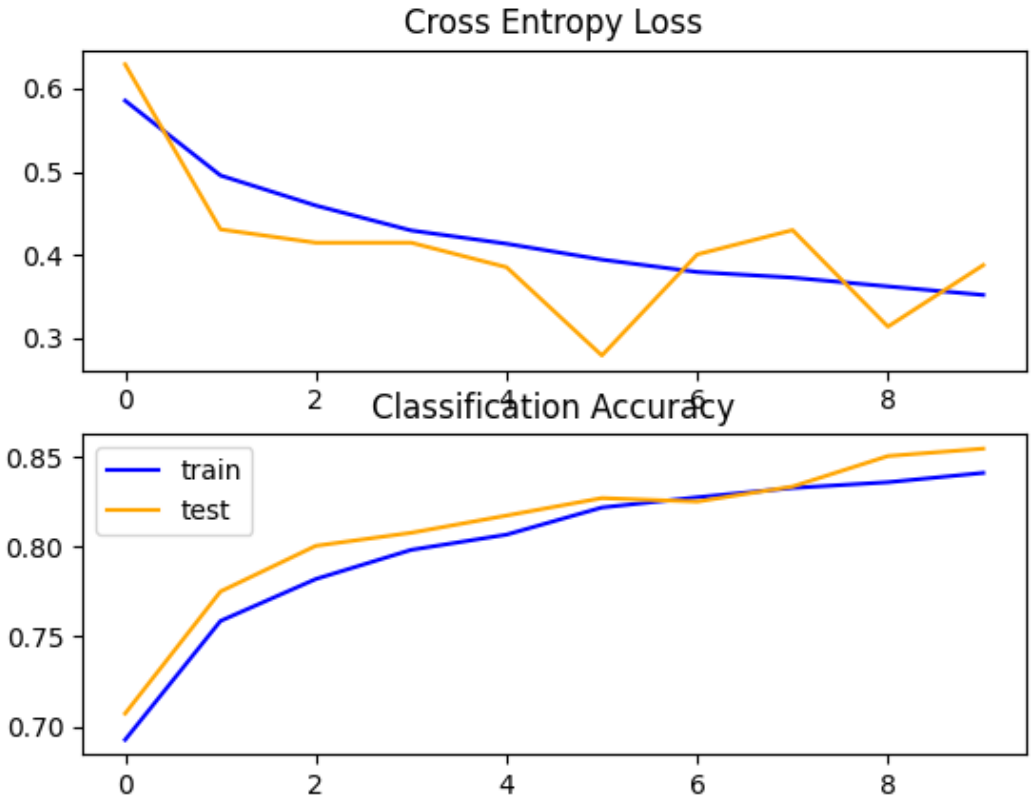


Figure 5: Accuracy and losses of our final model

4. Problems encountered

The problems faced are as follows:

- We wanted to use Google Colab to train our model. However, Colab kept repeatedly disconnecting after about 30 minutes when left inactive during training. A lot of time was spent to debug the disconnection problem.
- Eventually, we shifted to our local machines that were taking about 10 minutes per epoch. This was much lower than the machine on Colab that was taking around 2 hours per epoch for a 4 layer model.
- We could not set up Tensorflow with GPU on our machines despite almost spending a day on trying to configure it.

5. Conclusion

Despite tuning various parameters, such as batch size, input shape, switching optimizers, and adding batch normalization, we could only get an accuracy of 85.48% from our 4 layer model. It seems that the usage of the entire training dataset caused our model to overfit. In the future, we would like to properly set up our local machines for GPU-enabled training and check if results differ. We would also like to add more layers to our model for better results.

6. Links

GitHub repository link : image-classifier

7. References

- Simple Classification Tutorial using Keras
- Custom Implementation of VGG16 Architecture
- Adding layers to a sequential model - Keras Documentation