CS577 Assignment 4: Report

Shivi Srivastava
Department Of Computer Science
Illinois Institute of Technology
April 29, 2021

This report has the details of the 2 Assignment programs

Problem 1 –

1. **Problem statement**

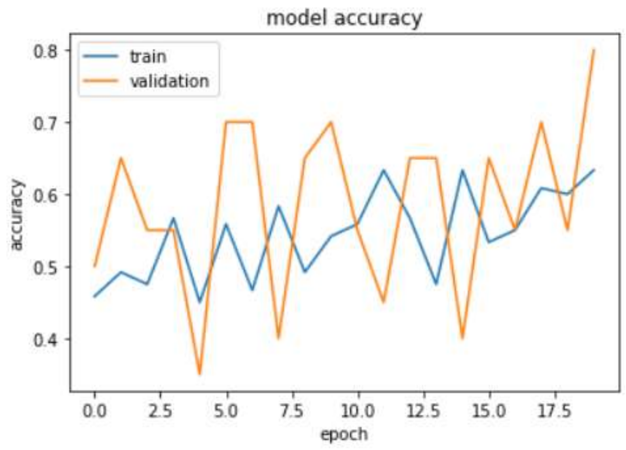
   This is a Binary Classification problem for Cats and Dogs(2000 each).
   Following needs to be implemented-
   a. A simple convolution neural network.
   b. Replacing the CNN convolution layers with a pre trained convolution base of VGG16. Frozen convolution base.
   c. Unfreeze the convolution Base and fine tune the model.
   d. Modify the data generator to augment the data and retrain with frozen convolution base.

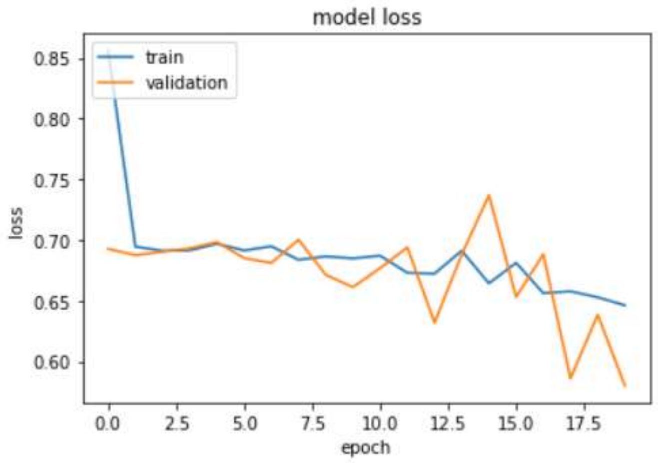
2. **Proposed solution**
   - data in Cats and Dogs is downloaded and divided in the following directories-
     Training
     Testing
     Validation
   - We will start by giving the input dim as 150 x150 to the neural network.
   - Will use sigmoid as the activation function and binary_crossentropy as the loss function.
   - We will see which optimizer we are going to use as we tune the model.
   - Also activation function for the middle layers we kept as Relu.
   - As this is the convolution neural network followed by dense Fully connected NN we will be using a combination of Convolution and Maxpooling layers.
3. **Implementation details**
   - downloading the necessary libraries like tensorflow, keras, matplotlib, numpy etc.
   - Downloading the data and keeping it in the base directory.
   - Dividing the Cats and Dogs data into 3 subsets of training/testing/validation
   - we will use data augmentation or image data generator to re scale the images
   - We are using three convolution layers after the number of experiments to obtain the final model
   - The second implementation that we did for cats and dogs data set history place the convolution layers with the pre trained convolution based model VGG 16
   - The first implementation that we did with VGG 16 model was to freeze the convolution layers that is to say the weights of the convolution layer cannot be modified and then we used the dense layer or the fully connected network do train the model.

- We extracted the feature using the VGG 16 model and based on those features trained the fully connected neural network to obtain the result.
- After this we fine tuned the model bye unfreezing the top players in of the network and jointly training our custom network with the unfrozen layers
- Lastly we took the data already present and reshaped it using the data augmentation approach to form the new images and retraining it with VGG 16 model with the frozen layers.

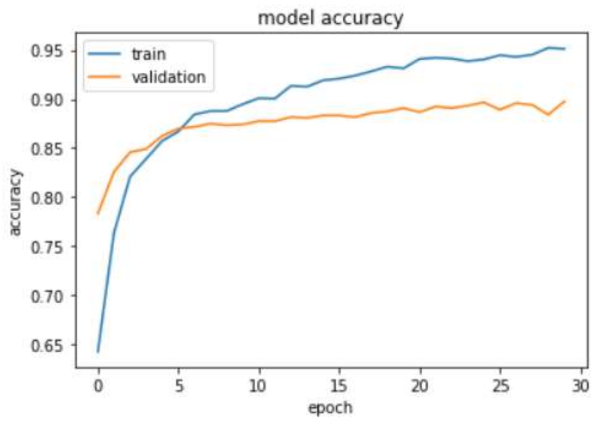
4. <u>Results and Discussions</u>

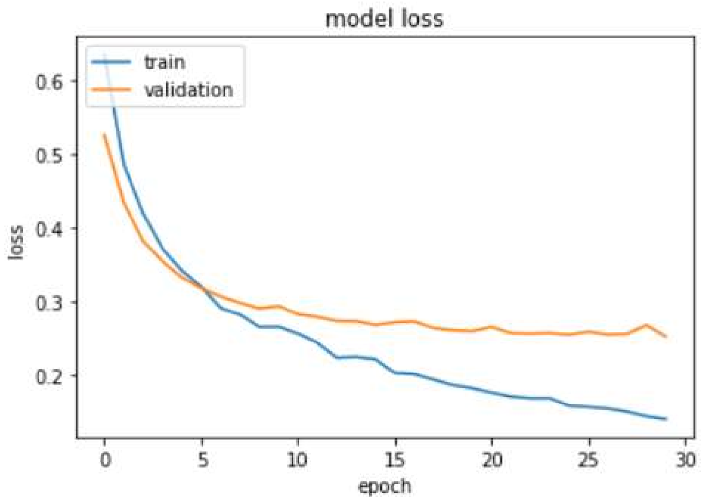
With custom CNN we got the result as –





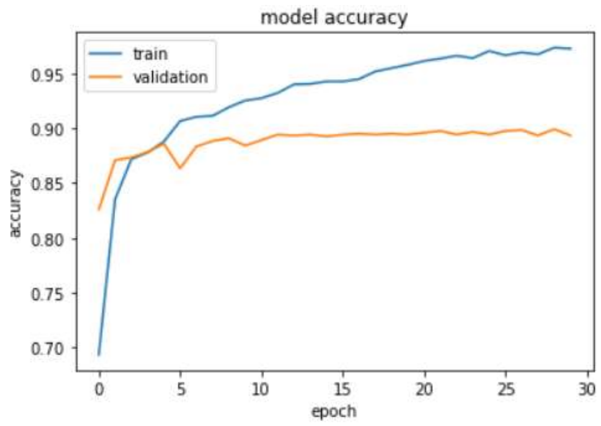
As we can see graph is not smooth and the accuracy is quite low and then we tested with VGG16 model and the result is below –

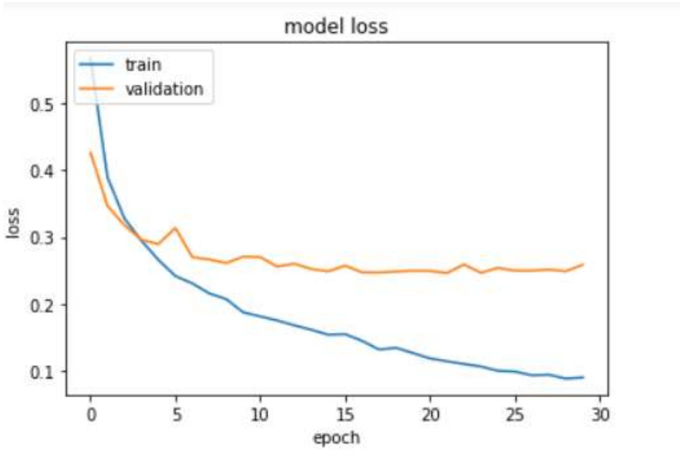


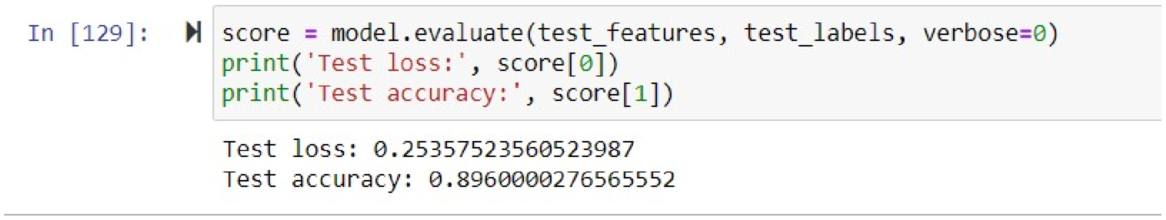


The graph is going to smooth and the accuracy is above 85% Lewis still the overfitting continues to exist, this proves that the VGG 16 model that is trained on the imagenet is better trained than the custom convolution network created above.

After fine tuning the model that is unfreezing the convolution layer and retraining it along with the custom convolution neural network






```
In [129]:  score = model.evaluate(test_features, test_labels, verbose=0)
           print('Test loss:', score[0])
           print('Test accuracy:', score[1])

           Test loss: 0.25357523560523987
           Test accuracy: 0.8960000276565552
```

The accuracy improves maybe because the part of VGG 16 is trained along with the custom convolution network.
After data augmentation the accuracy improves to 90%, maybe because the training data got more data and overfitting was reduced.

Problem 2-

1. **Problem statement**

   CIFAR 10 problem, build a basic CNN. Then add the inception blocks and check the performance, after that remove the inception block and add residual blocks and checking the performance.

2. **Proposed solution-**

   The data is divided in two five batches and one test data set. We and unpickled the data.
   - Adding the convolution and maxpooling layer.
   - Then replacing with inception block and converging to get the output from all the layers in the block.

3. **Implementation details**
   - We preprocess the data and added the convolution neural networks and Max pooling and got the accuracy of around 60%.
   - retuned the model and the accuracy improved by one or 2%.
   - then we added the inception block-

     incep_1 = Conv2D(64, (1,1), padding='same', activation='relu')(input_img)
     incep_1 = Conv2D(64, (3,3), padding='same', activation='relu')(incep_1)
     incep_2 = Conv2D(64, (1,1), padding='same', activation='relu')(input_img)

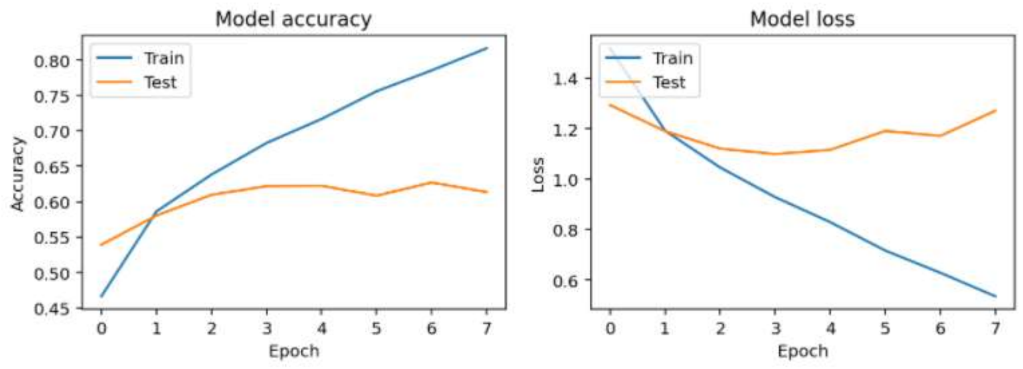
```
incep_2 = Conv2D(64, (5,5), padding='same', activation='relu')(incep_2)
incep_3 = MaxPooling2D((3,3), strides=(1,1), padding='same')(input_img)
incep_3 = Conv2D(64, (1,1), padding='same', activation='relu')(incep_3)
```

and to get the output –

```
output = keras.layers.concatenate([incep_1, incep_2, incep_3]
```

This we passed through a Fully Connected NN and

5. <u>Results and Discussions</u>

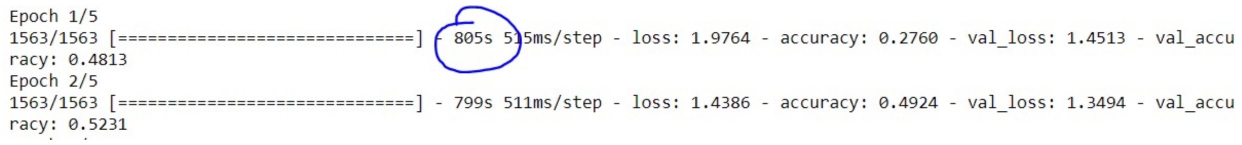


The accuracy here is quite low and it's around 60% and the model is quite overfitting,

Also we don't have the idea weather B have the vanishing gradient issue in this model.

However, when we added the inception blocks the performance improved to around 70% , for 25 epochs. this may be the cause we are using deconvolution layers in parallel rather than in sequence

The important point to note here is the performance is quite slow without GPU for the inception block.



```
Epoch 1/5
1563/1563 [==============================] - 805s 515ms/step - loss: 1.9764 - accuracy: 0.2760 - val_loss: 1.4513 - val_accu
racy: 0.4813
Epoch 2/5
1563/1563 [==============================] - 799s 511ms/step - loss: 1.4386 - accuracy: 0.4924 - val_loss: 1.3494 - val_accu
racy: 0.5231
```