**Summary:**

This project has two main parts to it which revolve around using Convolutional Neural Networks to classify the data given as either faces or backgrounds. The first part is where we build the Convolutional Neural Network from the Scratch. We will be training the CNN using the data supplied and then build the architecture of the CNN using different layers and different input settings. Once this task is done we will train the CNN and then test it on the testing data to get the accuracy. We change the settings to obtain different outputs. The next part of the project is to use a pre-trained network and perform Transfer Learning, this is the process where the final layers of the pre-trained model are adjusted to solve our problem of classifying the data into two different classes, either face or background.

**Algorithmic Approach:**

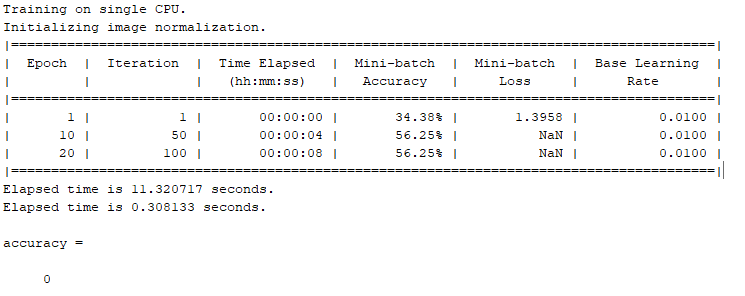
Since a boiler plate has been provided by the professor, we just have to perform few adjustments to make the CNN work on the currently provided data. But the general approach to this problem would be following.

1. Load the dataset into the program. This data is imported and stored as an ImageDatastore object as its difficult to load the whole dataset into memory. So, using it as an ImageDatastore object we can easily handle the data in a flow.
2. Next task is to define the architecture of the CNN, here we make some changes to the architecture in order to make it work.
   1. We mention *DataAugmentation* in the *imageInputLayer* as either ***randcrop*** or ***randfliplr.***
   2. We apply ***crossChannelNormalizationLayer*** before the *maxPooling2dLayer* to perform the normalization per channel.
   3. We also apply dropout layer to just activate the specified number of neurons before the *FullyConnectedLayer* (this is done as not all neurons contribute towards the answer).

Apart from the above additions we also have the *convolution2dLayer, reluLayer, softmaxLayer and classificationLayer.* All these layers work in cahoots to make the process work. The final layers are responsible to classify the test data into either face or background.

1. We then apply different parameters like ***MiniBatchSize***, ***InitialLearnRate***, ***MaxEpochs*** in the *trainingOptions.* These control the way the CNN learns features from the training data in order to accurately predict the result.
2. Finally, we classify the test data and calculate the accuracy.
3. For the next part of the assignment everything is the same except the architecture of the CNN is from a pre-trained network and we just change the final 3 layers to make to fit out problem.

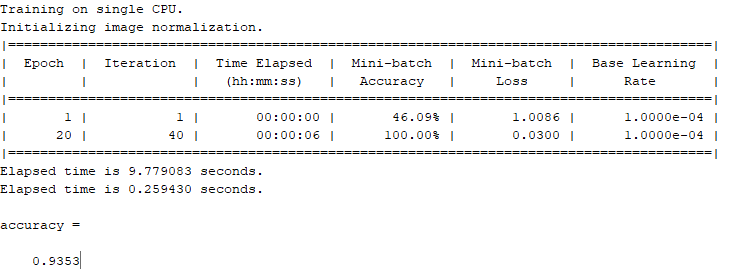
**Results:**



**Figure 1: Using only MiniBatch**

***Observation:***

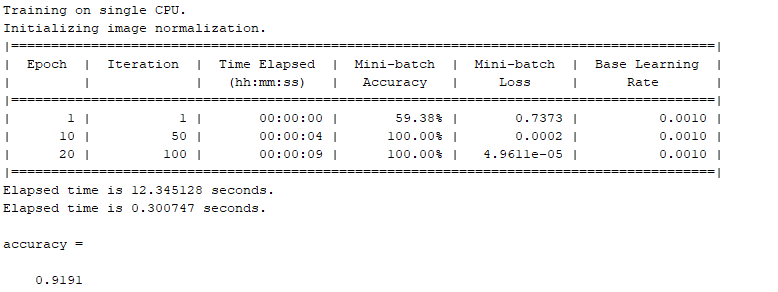
We can see that only using the MiniBatch size of 64 without other parameters doesn’t make the CNN work. I have tried different batch sizes but had no luck with any batch size.]



**Figure 2: Using both MiniBatch and InitialLearnRate**

***Observation:***

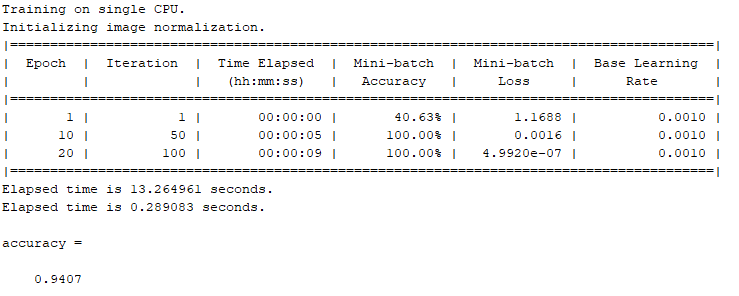
After using InitialLearnRate of 0.001 with the MiniBatchSize of 128, I have obtained an accuracy of 93.53 percent.



**Figure 3: After adding crossChannelNormalizationLayer to above task 2**

***Observation:***

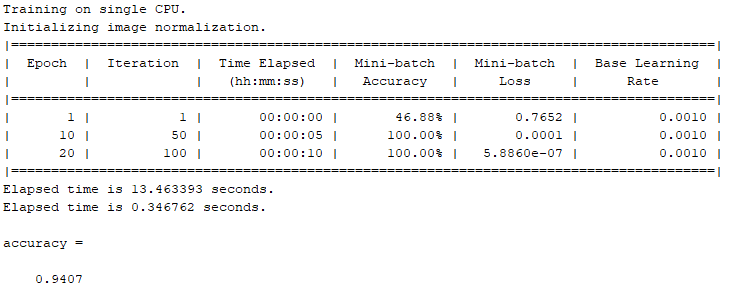
The normalization layer added has some effect on making the data normal to get acceptable results, here the batch size is reduced to 64 again.



**Figure 4: Adding the dropout layer**

***Observation:***

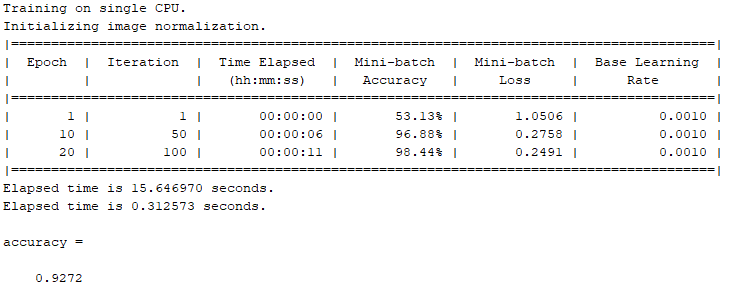
In this task we have added the dropout layer and can see that the accuracy has increased by 3 percent. This proves that not all layers have equal contribution towards the final result.



**Figure 5: After adding DataAugmentation RandCrop**

***Observation:***

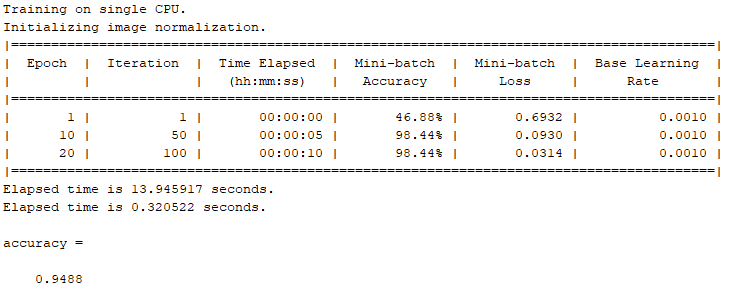
Adding the DataAugmentation Feature of RandCrop has helped retain the same accuracy with little change in the execution time of the data.

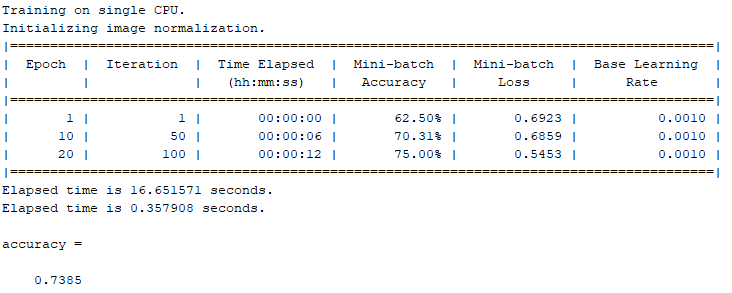


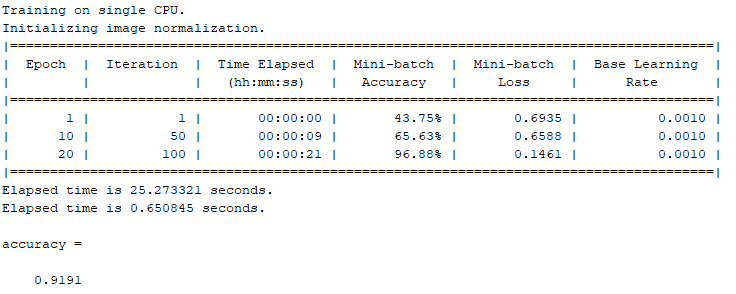
**Figure 6: After adding DataAugmentation RandFlipLr**

***Observation:***

Adding the DataAugmentation Feature of RandFlipLR has a different approach and has reduced the overall accuracy of the CNN by approx. 2 percent.



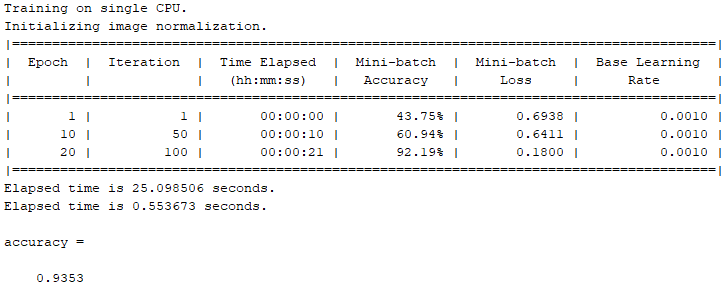




**Figure 7,8,9: Making the Layer deeper**

***Observation:***

We can observe that from the initial image, as we increase the depth the accuracy suddenly falls to an all time low at 73 and then as we increase the depth again, the accuracy increases to 91 percent

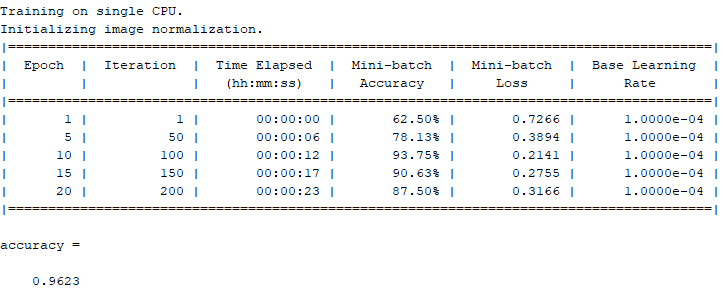


**Figure 10: After adding DropOutLayer after each MaxPooling2dLayer**

***Observation:***

The random dropping of neurons after each convolution is helpful, we can see that the accuracy has increased by 2 percent in the deeper CNN.

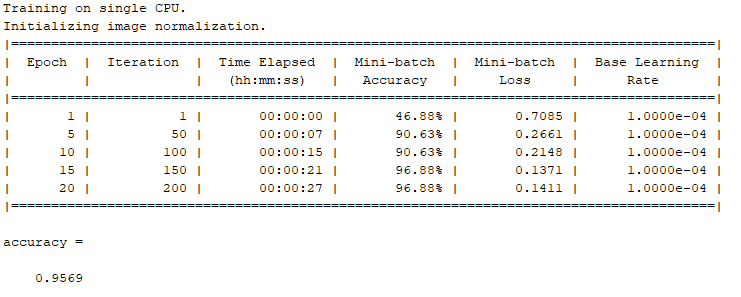
**Problem 2:**



**Figure 11: Adding WeightLearnRateFactor and BiasLearnRateFactor for the pre-trained network**

***Observation:***

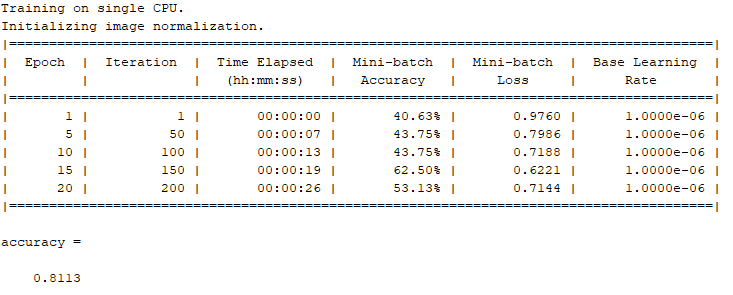
The all-time high 96 percent has been achieved using the WeightLearnRateFactor = 2 and BiasLearnRateFactor = 2 and InitialLearnRate = 0.0001 and batch size = 64. The low InitialLearnRate ensures the earlier layers of the pre-trained network to be retained.



**Figure 12: Adding WeightLearnRateFactor and BiasLearnRateFactor for the pre-trained network**

***Observation:***

Increasing WeightLearnRateFactor and BiasLearnRateFactor to 3 each, with a smaller batch size = 64 has led to decrease in the accuracy by 1 percent.



**Figure 13: Adding WeightLearnRateFactor and BiasLearnRateFactor for the pre-trained network**

***Observation:***

With the same settings as used for Figure 12 and decreasing the InitialLearnRate to 0.000001 has shown that as the LearnRate decreases further the accuracy gets worse. This could probably mean that we are focusing more on the initial layers rather than the final layers.