MACHINE LEARNING HW 4

- ***GAUTAM SHENDE (50245840)***
- ***SUJIT SINGH (50247206)***

ENVIRONMENT:

TENSOR FLOW

In python, tensorflow, provides the efficient platform for running convolutional neural networks and with minimum lag.

Tensorflow provides the flexible environment to set variable and inbuilt function to train and test dataset.

It has multiple inbuilt optimizers and activation functions making it easier to implement Deep Learning problems. In this project, we have used AdamOptimizer with ReLU activation function.

AWS

We have used EC2 Instance with Deep Learning AMI using tensorflow_p36 evironment to train and test our codes on the AWS server.

PROGRAM FLOW:

Extraction of data:

In extraction.py file, we constructed the dataset for training, validation and testing using an 8:1:1 split and 0.1% of the original dataset by default. These parameters can be changed in the code by changing the variables dataFraction, trainFraction and validationFraction respectively.

- ➔ Extracted the RGB images
- ➔ Resized RGB images
- ➔ Flattened RGB images
- ➔ Save flattened RGB training Vectors
- ➔ Extracted labels from the eyeglasses column in the attributes.csv
- ➔ We also added choice to add augmented dataset which can be used by activating the variable *flag.* We augmented new images to the training data set by translation (12.5% of the training set), rotation (7.5% of training set) and cropping (5% of training set)

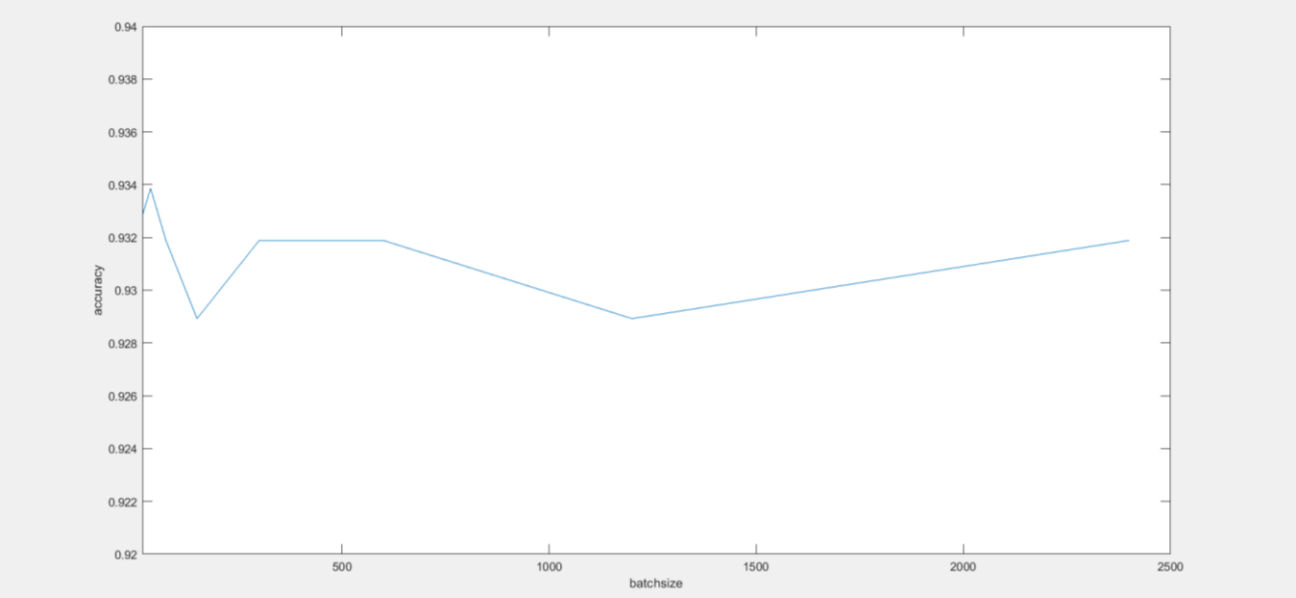
Implementation of CNN:

We have realized the concept of Convolutional Neural Networks in CovolutionalNN.py file. As we have to train the RGB (3 channel images), we have used 3D convolution and pooling. The CNN performs following series to operations to train and test the dataset.

- ➔ Extracted the RGB images
- ➔ The placeholder for input images and output labels are created with x and y_
- ➔ Defining the weights and the bias of the first convolution layer where weights. The last column of the weight vector is the number of channels of the filter.
- ➔ Convolution: Reshaping the image into the required format and applying 3D convolution to the reshaped image. The convolution is done between the image and weight and bias is added thereafter. Here, weight act as a filter. Then we apply ReLU activation function to the result of the convolution.
- ➔ Pooling: After convolution the next stage of CNN - pooling is initiated. In pooling the aggregated result across the neuron cluster in the current layer goes into the single neuron in the subsequent layer. Here we have used max pooling i.e. the maximum result from each layer neuron cluster goes into one of the neuron of the subsequent layer.
- ➔ The step 3 and 4 are done twice for two hidden layers using different dimensions of filter (weights).
- ➔ Fully connected Layer: The fully connected network connects every neuron of one layer with every other neuron to the other layer. This is done twice starting from the 1024 neurons and ending to 2 neurons (dimension of the image label).
- ➔ Dropout Layer: In order to avoid overfitting of data, dropout layer is created. It is a regularization technique which prevents co-adaptation of training data.
- ➔ Readout Layer: The last layer is a readout layer which is similar to that of the softmax layer used previously

PARAMETER TUNING

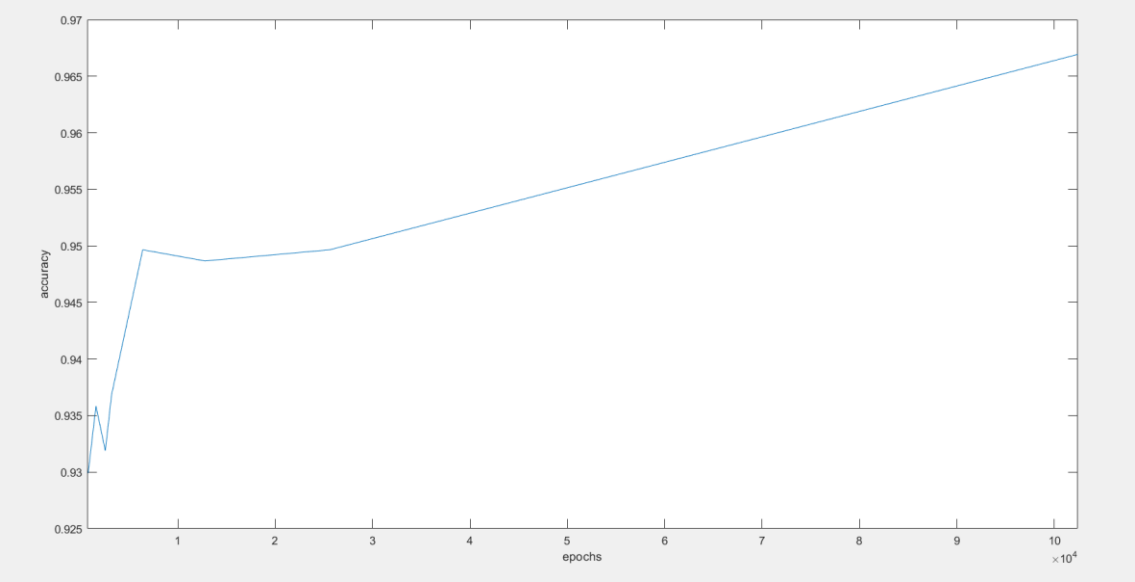
1) **Batch Size**
   Increasing the batch size along with decreasing epochs, keeping the multiplication epochs * batch size constant. We achieved the following graph:



We couldn't draw anything concrete from these results i.e what should be the ratio of the batchsize to the number of epochs.

2) Number of epochs:

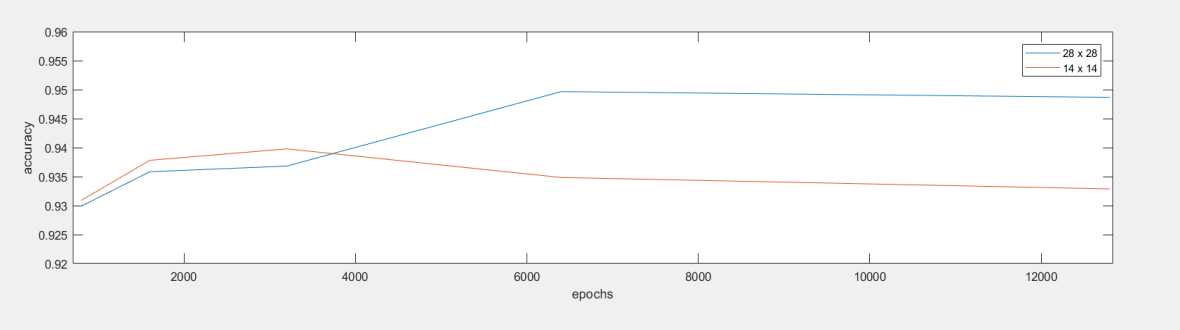
We tried increasing the number of epochs for a toy dataset of around 10% of the original one and achieved the following graph:



Thus the general trend here followed was there was increase in accuracy with increase in the number of epochs. This was expected because there were too many nodes within the architecture for which the weights could be adjusted and even after close to 10^5 epochs, the weights still had not reached convergence.

3) Size of compressed Image:

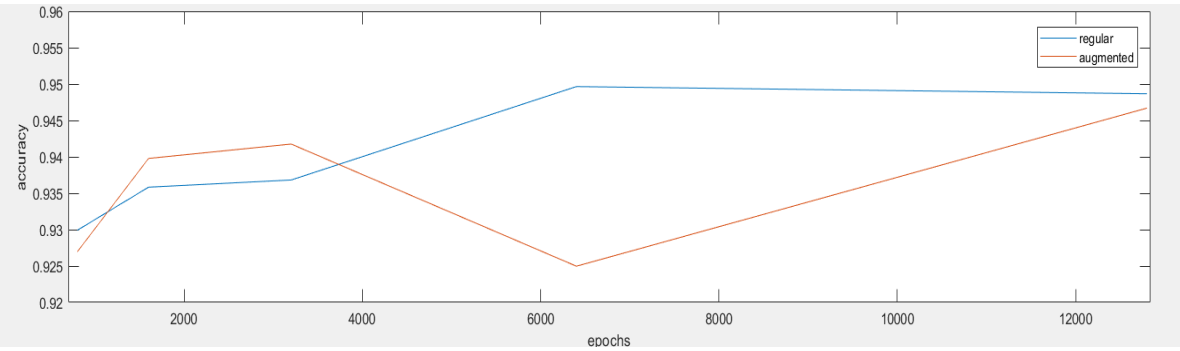
Here is a plot of accuracy vs number of epochs for size of compressed image = 14 x 14 and 28 x 28.



The 14 x 14 initially performed better with less number of epochs because it was able to learn fast. However, the 28 x 28 case clearly outperforms the 14 x 14 once the number of epochs is increased and model starts to learn. There is just too much compression in the 14 x 14 case t keep up with the accuracy of the 28 x 28 model.

4) Augmenting Data Set:

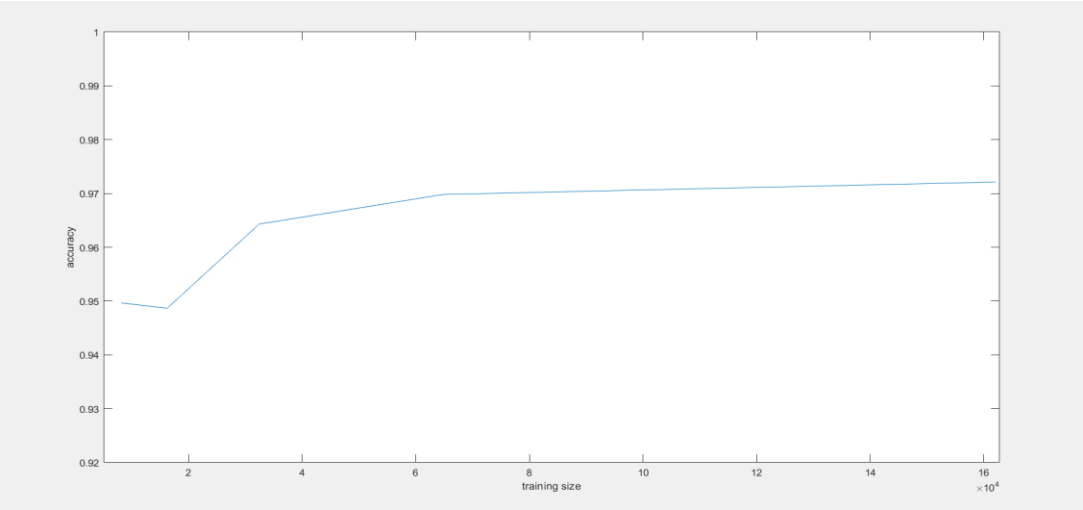
As explained, we augmented data by performing translation on 12.5% of the training set, rotation on 7.5% of training set and cropping on 5% of training set. Due to the added images, we did not see any improvement in accuracy. Here is a plot for epochs vs accuracy on both the original data set as well as the augmented one.



Since, there was not much improvement in accuracy but considerable increase in computation for creating the augmented data set, we decided to skip/not include the augmented data) it in the final model.

5) Size of Training set:

We tried playing with the size of the data set but limited resources only allowed us to do it for a few iterations. Even on the AWS server, when we considered > 30,000 samples, it took more than an hour to run for >5000 epochs.



We found out after increasing the data set beyond a certain point close to 30,000, the tradeoff between the time taken for execution and improvement in accuracy was not worth it because the accuracy only increased marginally and it became almost constant.

RESULTS

Here is a set of results we found for different values of the hyper parameters:

Training Set Size	Image Height	Image Width	Epochs	BatchSize	Accuracy
8103	28	28	800	38	0.929911
8103	28	28	1600	38	0.935834
8103	28	28	3200	38	0.936821
8103	28	28	6400	38	0.949655
8103	28	28	12800	38	0.948667
8103	28	28	2560	38	0.931885
8103	28	28	25600	38	0.949655
8103	28	28	100	600	0.931885
8103	28	28	50	1200	0.928924
8103	28	28	25	2400	0.931885
8103	28	28	200	300	0.931885
8103	28	28	400	150	0.928924
8103	28	28	800	75	0.931885
8103	28	28	1600	38	0.93386
8103	28	28	1600	37	0.93386
8103	28	28	3200	19	0.932873
8103	28	28	6400	150	0.944719
8103	28	28	12800	75	0.936821
8103	28	28	6400	38	0.948667
16207	28	28	6400	38	0.948667
16207	28	28	12800	38	0.949655
16207	28	28	25600	38	0.943238
16207	28	28	102400	38	0.96693
32414	28	28	25600	38	0.96431
64828	28	28	12800	75	0.96981
162079	28	28	6400	150	0.97212
809	14	14	800	38	0.960396
8103	14	14	800	38	0.930898
8103	14	14	1600	38	0.937809
8103	14	14	3200	38	0.939783
8103	14	14	6400	38	0.934847
8103	14	14	12800	38	0.932873

Augmented Data Set Results:

10128	28	28	800	38	0.92695
10128	28	28	1600	38	0.939783
10128	28	28	3200	38	0.941757
10128	28	28	6400	38	0.924975
10128	28	28	12800	38	0.946693

REFERENCES

1. *www.tensorflow.org/tutorials*
2. https://docs.opencv.org/3.0 beta/doc/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html
3. https://www.kaggle.com/sentdex/first-pass-through-data-w-3d-convnet

***