INTRODUCTION TO MACHINE LEARNING

PROJECT-4 REPORT

Prof. Dr. Srihari

GROUP MEMBERS

NITISH SHOKEEN (UB Person Number: 50247681)

MAHALAKSHMI PADMA SRI HARSHA MADDU (UB Person Number: 50246769)

CHARANYA SUDHARSANAN (UB person Number: 50245956)

Problem Statement: To implement a convolution neural network to determine whether the person in a portrait image is wearing glasses or not.

Convolutional Neural Network Classification.

Convolutional Neural Networks (CNN) are biologically-inspired variants of MLPs. CNNs exploit spatially-local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. In other words, the inputs of hidden units in layer **m** are from a subset of units in layer **m-1**, units that have spatially contiguous receptive fields.

In addition, in CNNs, each filter h_i is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a *feature map*.

Additionally, weight sharing increases learning efficiency by greatly reducing the number of free parameters being learnt. The constraints on the model enable CNNs to achieve better generalization on vision problems.

A feature map is obtained by repeated application of a function across sub-regions of the entire image, in other words, by *convolution* of the input image with a linear filter, adding a bias term and then applying a non-linear function.

A CNN is composed of a stack of convolutional modules that perform feature extraction. Each module consists of a convolutional layer followed by a pooling layer. The last convolutional module is followed by one or more dense layers that perform classification.

Implementation:

- We first import the labels for eyeglasses column (16^{th} column) and store it as an np array.
- Now import all the .jpg files in the folder into a numpy array called data_images by using a function glob(os.path.join())
- Loop through the entire dataset of images and store the index in n and imagename in file_name.
- Now, read each of the obtained images and resize it.
- Convert each image into an array and store all the arrays of images in another array named data.
- Split the data into training validation and test sets. Here, we've taken training as 75% of the dataset and the rest 25% is split equally into validation and test set.
- Convert the output predictor (Wearing glasses or not) into a binary class matrix.
- We now create a CNN Architecture with three layers: Convolutional and Dense.
- Pass the data through two layers of convolutional and pooling and then through two layers of Dense by stacking the layers using an add function.
- Apply Dropout regularization to the model. A dropout of 0.4 is applied, i.e., randomly 40% of the training data is dropped.
- Convolution Layer applies a specified number of filters to the image.
- We apply an activation function called ReLu to the output to introduce the nonlinearities into the model.

- Pooling layer down sample the image data extracted by convolutional layer to reduce the dimensionality to give a better processing time.
- Each module consists of a convolutional and pooling layer. After two modules we pass the output to the dense layer 1.
- Dense layer performs the classification. We use one dense layer with 1024 neurons and a final dense layer with 50 neurons.
- Configure the learning process with compile(), apply a loss function(cross entropy) and optimizer (SGD optimizer).
- Fit the model by training the model by iterating on the data in batches of 20.
- Find the accuracy on test data using evaluate function.

Hyper-parameter tuning:

We create a convolutional neural network model with following layers.

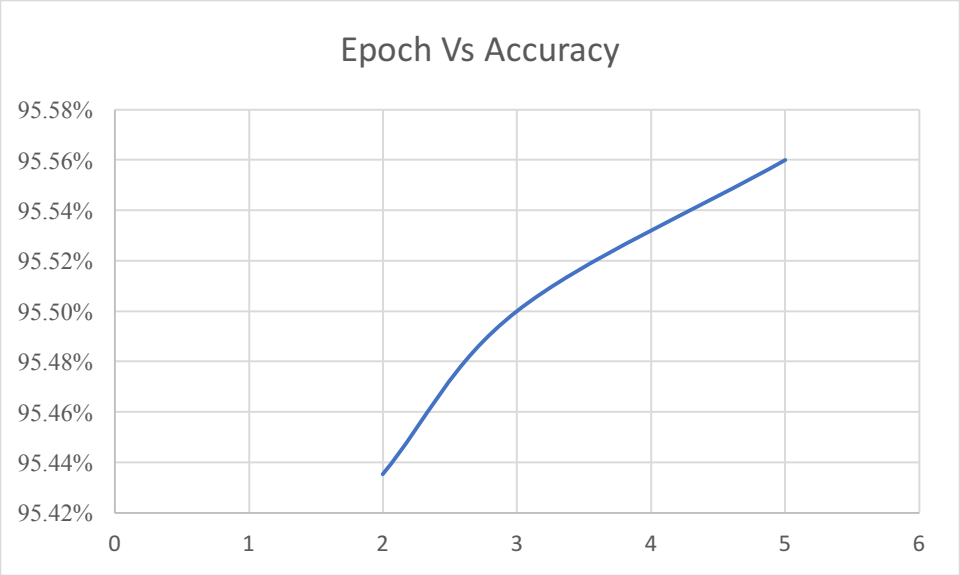
Convolutional layer with 32 2x2 filters with ReLu Activation function.
Maxpooling with 2x2 filter with stride 2.
Convolutional layer with 16 1x1 filters with ReLu Activation function.
Maxpooling with 1x1 filter.

After Hyper parameter tuning , We arrived at the following values for the model.

Epoch: 5
Batch Size: 20
Dense1: 1024 Neurons
Dense2: 500 Neurons
Dropout: 0.4

Keeping Batch Size as constant, Epoch values changed for a minuscule amount.

Epoch	Accuracy
5	95.56%
3	95.50%
2	95.44%



Hyper-parameter tuning on neurons in dense layer:

Dense Layer Neurons	Accuracy
125	94.1
1024	95.43

Final Output after Hyperparameter tuning :

```
In [7]: runfile('/Users/nitishshokeen/Desktop/MLHW4/we_hw4.py', wdir='/Users/nitishshokeen/Desktop/MLHW4')
Importing Labels

36it [00:00, 356.52it/s]creating training validation and test data from the images:

202599it [11:06, 304.11it/s]
Dividing the data into training validation and test:

creating the model:

fitting the model:

Train on 151949 samples, validate on 25325 samples
Epoch 1/5
 - 557s - loss: 0.7635 - acc: 0.9526 - val_loss: 0.7510 - val_acc: 0.9534
Epoch 2/5
 - 314s - loss: 0.7547 - acc: 0.9532 - val_loss: 0.7510 - val_acc: 0.9534
Epoch 3/5
 - 282s - loss: 0.7547 - acc: 0.9532 - val_loss: 0.7510 - val_acc: 0.9534
Epoch 4/5
 - 274s - loss: 0.7547 - acc: 0.9532 - val_loss: 0.7510 - val_acc: 0.9534
Epoch 5/5
 - 276s - loss: 0.7547 - acc: 0.9532 - val_loss: 0.7510 - val_acc: 0.9534
calculating the score:

Error for Deep CNN : 4.60%
```

Result:

Error on test data: 4.60%
Accuracy on test data: 95.40%