ENEE 439M

Foundations of Machine Learning

Final Project

Rajeshwar N S
116921237

1. **Support Vector Machines:**

Data and Preprocessing :

Here, we use scikit-learn tool box to implement the classifier.
First, we import the data from mnist.mat file which has two columns namely data(images) and labels. Totally, there are 70,000 rows in which dimension of the data is (70,000 x 784) 784 being the pixel dimension of each image (28x28).
On the other hand, Labels have the dimension (70,000 x 1). Now for visualizing sample images, we reshape the images into (28 x 28) vectors using matplot.
Also, we divide the images by 255 ie. normalizing the RGB value to feed into the classifier.

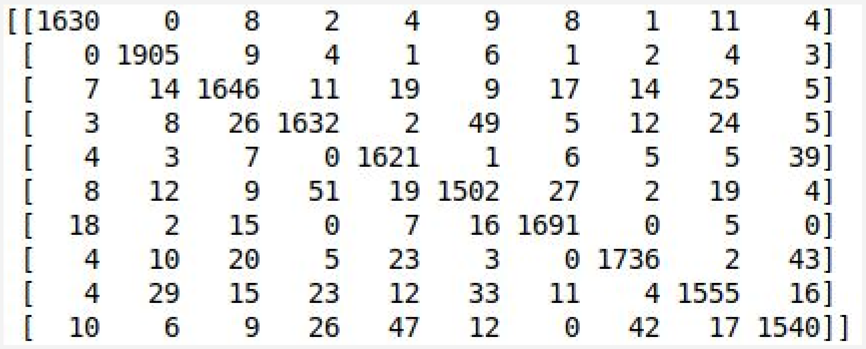
Test-Train Data Split:

The 70,000 Image data are split into 75% for training and remaining data for testing which is about 17500 rows of data.

The Classifier:

The Classifier is implemented using built in tool called svm.SVC. Two different kernels are used. The default kernel gave about 93% accuracy while using the radial base kernel (rbf) increased the accuracy upto 95%.
Also, important parameters in SVM kernel are C and gamma. C is the cost of misclassification. A large C gives a low bias and high variance which may cause a overfitting problem. A very small C gives a high bias and low variance which may cause underfitting problem. Gamma, on the other hand, is a parameter of the Gaussian kernel. In brief, they control the shape of peaks when the data is raised to a higher dimension (usually used in non-linear classification) A small gamma gives you a pointed peak in the higher dimensions, a large gamma gives you a softer, broader bump. So a small gamma will give low bias and high variance while a large gamma will give you higher bias and low variance. The best parameters of C and Gamma are found using function GridSearchCv function of sklearn which returns the best parameters for the model after running all the possible

combinations. ie. a simpler way of hyperparameter tuning. But due to computational complexities, we didn't use the GridSearchCV as it takes a longer time for the SVMs. Rather, we use dimensionality reduction techniques like PCA/LDA to reduce the dimension of the data and then feed to the classifier. For this, we used sklearn's decomposition PCA. Here we reduced the dimension from 784 to 16. Not just the calculation time had drastically reduced to about 1 minute since the dimension is very less but also accuracy is increased to about 96%. Also for every classifier, confusion matrix is calculated which gives the no. of correct and incorrect classifications for each digit as shown below.



```
[[1630    0    8    2    4    9    8    1   11    4]
 [   0 1905    9    4    1    6    1    2    4    3]
 [   7   14 1646   11   19    9   17   14   25    5]
 [   3    8   26 1632    2   49    5   12   24    5]
 [   4    3    7    0 1621    1    6    5    5   39]
 [   8   12    9   51   19 1502   27    2   19    4]
 [  18    2   15    0    7   16 1691    0    5    0]
 [   4   10   20    5   23    3    0 1736    2   43]
 [   4   29   15   23   12   33   11    4 1555   16]
 [  10    6    9   26   47   12    0   42   17 1540]]
```

Fig -1 : Confusion matrix resulted from RBF-SVM classifier

2. <u>Neural Network:</u>

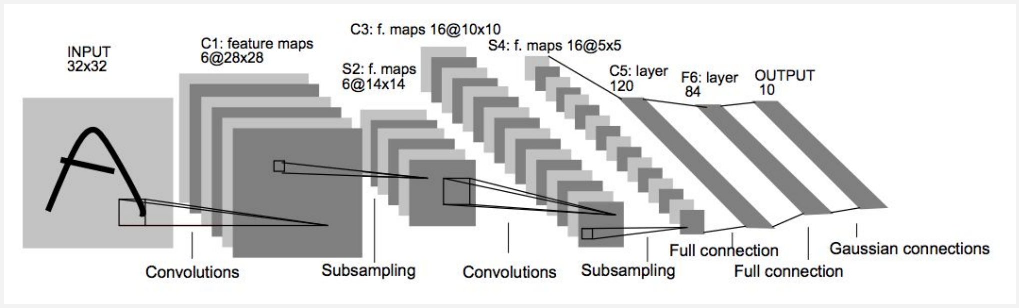


Fig-2: CNN Architecture (LeNet-5)

This method uses a CNN for classifying digits and the architecture is defined as follows:

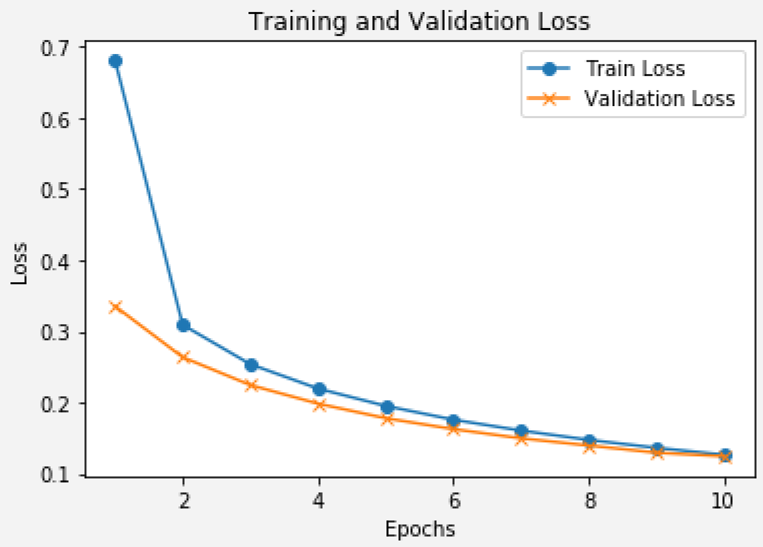
- Convolution Layer -1 . Input = 32x32x1. Output = 28x28x6
- SubSampling -1 . Input = 28x28x6. Output = 14x14x6.

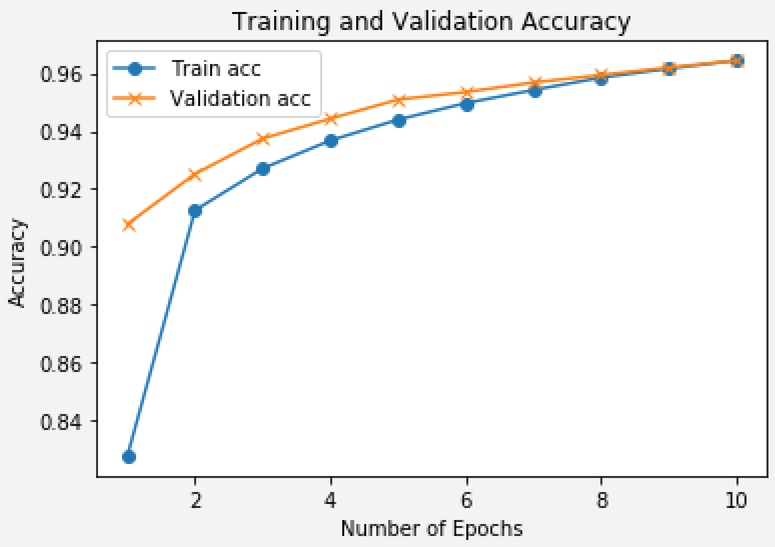
- Convolution Layer -2 . Input = 14x14x6. Output = 10x10x16
- SubSampling - 2. Input = 10x10x16. Output = 5x5x16
- Fully Connected -1 . Input = 5x5x16. Output = 120
- Fully Connected -2. Input = 120. Output = 84
- Fully Connected -3. Input = 84. Output = 10

Library:
This model is setup using Keras, a high level API implementation which uses tensorflow as backend. The code was also tried in tensorflow. But since keras offers easy fine tuning operations in a single line of code, it was used over the tensorflow.

Model:
Since the Input Image is of dimension (28x28), the first convolution layer is modified as (28x28x1) instead of standard (32x32x1). Also another method is to go with the standard (32x32) by padding zeros appropriately to the input images.and fed to first Convolution layer. A standard filter size of (5x5) and a pooling layer of (2x2) is used throughout the network. The activation function used here is hyperbolic tangent function (tanh) which is s-shaped sigmoid function ranging from -1 to 1. Other activation functions like 'relu' and 'sigmoid' have been used. But tanh performed better. Also, Different optimizers have been used like 'adam', 'rmsprop' and 'sgd'. Also different combinations of loss functions have been used like 'categorical_crossentropy' and 'mean_Squared_error' etc. Since MNIST is a popular and a relatively easy dataset (ie.comprising good data), all optimizers/loss functions almost give an accuracy for more than 90%. However, in our case, the 'sgd' optimizer and cross categorical entropy loss function gave a relatively better accuracy of more than 96%. Batch sampling was taken as 128 and the training examples were iterated 10 times (10 Epochs). Approximately, the total time was around 20 mins which was less compared to the SVM without any dimensionality reduction.

The accuracy and loss plots are shown below. The graphs shows that model is performing well with the selected optimizer, and activation functions.





Conclusion:

Thus we have performed MNIST hand digit recognition using a conventional supervised Learning ie. SVM and using a deep learning network. While both performed well with appropriate pre-processing, the neural network because of its sophisticated architecture have performed better over SVM on validation set minimizing the loss.