Project - Largest Digit Recognition

404 NOT FOUND - TEAM PROJECT REPORT

COMP 551-001

Ibtihel Amara
ibtihel.amara@mail.mcgill.ca
260819506

Sunanda Bansal
sunanda.bansal@mail.mcgill.ca
260775958

Yimeng Wu
yimeng.wu@mail.mcgill.ca
260788916

I. Introduction

Handwritten Digit Recognition has various applications in the field of computer vision and pattern recognition since it englobes different crucial utilizations such as automatic banking, plate number recognition, etc. In the context of this work, our goal is to extract and recognize the largest handwritten digit in an image. Our work is divided into two major parts: **exploration and implementation of different computer vision techniques** to find the optimal way of extracting and defining the features of the largest handwritten digit from a raw image data, and **the implementation of various machine learning techniques** for the digit recognition. All of the techniques were applied to the modified MNIST dataset provided for the in-class Kaggle competition. In the following sections, we both, discuss and report, the results and observations obtained from experimenting and tuning various algorithms using machine learning approach.

II. Feature Design

A. Image Pre-Processing and Segmentation

We experimented with more than one preprocessing techniques and the underlying ideas behind each preprocessing step are given below:

Normalization: We performed image normalization by dividing each pixel value by the maximum intensity in the image giving a narrower range of the intensity distribution (i.e. [0 1] instead of [0 255].)

Image thresholding and background suppression: The approach we took to handle the "noisy" background of the images where through performing image smoothing via a mean filter and binarization in order to highlight the most important pixels in an image and subtract the unwanted background information.

Segmentation: large digit extraction: The purpose of this step is to extract the digit having largest relative scaling factor. We were able to extract the digits for most of the training data accurately using our approach. We used an edge detector operator (i.e. Canny Edge Detector [1]) to find all existing contours and compute then compare the area of the minimum square bounding box. For each detected digit, we center, rotate and crop it to 28x28 sized image without distorting the aspect ratio.

B. Feature Extraction

We experimented with different feature extraction techniques. The latter is commonly used when dealing with computer vision problems because they can properly describe and distinguish the different characteristic of an image and all the classes (which is in our case numbers from 0 to 9). In our work, we implemented these techniques:

Local Binary Patterns (LBP): It is a feature-based descriptor and was primarily introduced by Ojala et al. [2]. Since then, its uses have expanded, covering different pattern recognition problems such as Face Recognition [3] and Handwriting Recognition [4]. This descriptor is a gray-scale invariant texture measure that is obtained through labeling the pixels of an image by comparing it to its neighboring pixels within a window. The result of this comparison is a binary number that will be used as a feature descriptor. There are many variants of LBP but in our work we used the uniform pattern variant because this approach of LBP tends to minimize redundancy found in the binary pattern therefore reducing the features to a lower dimension. In our case, the length of the LPB descriptor for each image was 26. Details can be found in the original paper [2].

Histogram of Oriented Gradient (HOG): It is an appearance descriptor and was first proposed by Dalal and Triggs [5] and currently, it is one of the most successful and most used descriptors in computer vision and pattern recognition. This technique is based on counting the occurrences in the gradient orientation in a localized region of an image;. HOG has been commonly used in many recognition tasks especially digit recognition [6]. The orientation vector in our work was set to 9 through a 16 by 16 window giving us a 144 dimension per feature.

C. Feature selection

We experimented with dimensionality reduction and implemented our algorithms after reducing the dimensions of our dataset as well.

We used Principal Component Analysis (PCA) to reduce dimensionality of our dataset. For 28x28 pre-processed image data we kept 100 principal axes after analyzing the variance ratio which is a reduction of 87.24%.

On the other hand for other dataset with HOG features since we only had 144 HOG features, we analyzed and kept 21 principal axes.

III. Algorithms

A. Baseline Classifier - Linear SVM

We used Support Vector Machines (SVMs) among the linear classifiers in order to establish a baseline for our classification. The basic idea of Support Vector Machines is to find a hyperplane that separates the data points of two different classes by as large a margin as possible. Since our data is multiclass, the approach used is one-vs-one for the purpose of classification.

B. Implemented Neural Network

Neural Network is a computing system made of interconnected layers, where neurons process the weighted input according to an activation function and backpropagate the error in prediction to adjust the weights. [7] Though various activation functions can be used, for our implementation we have used Sigmoid activation function for both forward and back propagation. The output function is sigmoid as well.

C. Other Non-linear classifiers

K-Nearest Neighbors: K-Nearest Neighbors assigns a class to the test set based on the voting from the `k` closest neighbors based on the training data points.

Neural Network through Keras: Though, We have implemented Neural networks from scratch, we also implemented it using Keras for the initial stages of our experiment for the in-class competition on Kaggle. The details of the experiment can be found in Appendix.

Convolutional Neural Network (CNN): To increase the classification accuracy, we then considered Convolutional Neural Network. We implemeted this method using the original dataset and the 28x28 pre-processed images without the use of feature extraction techniques. As a matter of fact, the configuration of the CNN enables the model to learn the features from raw images. Compared with traditional neural networks, CNN could exploit the structure in the input and benefit from local field, shared weights and subsampling. Through local field, each neuron could extract basic visual features, such as line segments, endpoints and corners. Shared weights enable CNN to have fewer parameters and require relatively less training data. Subsampling can reduce the resolution of features and achieve invariance to displacement, scaling, and other forms of distortion. These advantages make CNN extremely effective in image processing task, such as image classification and image recognition.

Residual Neural Network : We observed that when CNN is deep enough, adding layers had little contributions to increase classification accuracy. Thus, we changed to a recently presented CNN architecture, Residual Neural Network (ResNet), which is proposed by Kaiming He et al. in 2015 [8].The major difference between residual network and plain network is that residual network introduces residual block which is realized by shortcut connection. To a large extent, this method solves the degradation and vanishing gradient problem in deep neural network. In our project, we learnt from the design of a residual network with 34 parameter layers [8].

IV. Methodology

A. Training and Validation split Strategy

While using various machine learning algorithms mentioned above, we used the 80-20 Split approach and trained our model on the 80% of the training data and selected the hyper-parameters for the model after looking at its performance on the remaining 20% of the data, i.e. Validation Data. With a validation split of 20% we were able to keep a check on overfitting. Upon observing overfitting in data we used various regularization measures dependingon the algorithm.

B. Hyper-parameter Tunings

Linear SVM: This baseline classifier was applied on different feature representation of the dataset mainly using **HOG features**, **HOG with PCA**, **LBP**, and through the use of the **segmented image without feature extraction**. We proceeded in tuning this linear classifier by defining two hyper-parameters, which are the **Penalty parameter C of the error term** and **the loss function**. For each feature representation, different ranges of hyper-parameters were defined. Below are the summary of ranges that are considered for each feature space for the hyper-parameter ”C”. Concerning the second hyper-parameter, we took two values of loss functions which are the ”squared hinge” and the ”hinge” loss functions.

TABLE I: Range of the penalty hyper-parameter for the Linear SVM

Feature Space	Penalty hyper parameter C
28x28 Preprocessed Images	[0.4e-5, 0.5e-5, 0.6e-5, 0.7e-5, 0.8e-5, 1e-5,2e-5]
LBP	[1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1,1e2,1e3]
HOG Features	[90,100,110,120,130,140,150,160]
HOG Features with 21 Principal Components	[0.08,0.1,0.2,0.4,0.6,0.8,1,2,5,8]

Neural Network: The L-layer and fully connected Neural Network, implemented from scratch, was applied onto the 28 by 28 segmented images (784 dimensional features), This choice was not arbitrary. Instead it was concluded through the experiments done on the baseline classifier (Please refer to the results in Table II). This model was tuned using different methods. A summary of these tuning approaches along with the considered ranges of hyper-parameters are summarized below.

- First, we varied the number of hidden layers (i.e our hyper-parameter; from 1 to 4 layers) and kept the number of nodes at one layer as a constant (set to 50 units per layer).
- We considered one single hidden layer of 50 units and we wanted to see the effect of varying the learning rate (i.e our new hyper-parameter) of the Gradient

Descent algorithm on the performance of the model. The range considered for our hyper-parameter is $\alpha = [0.001, 0.002, 0.004, 0.006, 0.008, 0.01, 0.012]$

- Finally, we considered a single hidden layer for the network and we defined the number of nodes (i.e units) as a hyper-parameter. The range of the hyper-parameter is $HiddenLayerUnits = [300, 400, 500, 600, 700, 800, 900]$

k Nearest Neighbors: We tested k Nearest Neighbors classifier for a log spaced range of values of `k`, i.e. the number of neighbors to consider for voting, in the powers of 2 - `(2, 4, 8, 16, 32, 64)`. Since we found the values to be steadily decreasing after the local maxima at `k=4` , we did not go any further than 64.

Convolution Neural Network: We first considered a simple CNN with three convolutional layers and tuned two hyper-parameters on it:learning rate and batch size.We chose learning rate from `(0.1, 0.001, 0.0001)` and chose batch size from `(50, 80,100)`. Then we considered to tune the structure of CNN, including kernel size, depth of output volume and number of layers. The loss function we used was cross entropy function.

C. Regularization Strategy

Without regularization, the neural networks tend to overfit the training data. We used various approaches depending on the machine learning algorithm, like dropout, image augmentation and early stopping for Neural Networks and its variants.

Dropout: randomly dropping out the neurons with probability `p`

Image augmentation: We artificially expanded the training data by randomly adding several transformations, including rotation, crop and flipping images horizontally

Early stopping: stopping training when validation error did not decrease for few epochs

D. Optimization Strategy

For CNN and ResNet, we used Adam (Adaptive Moment Estimation) optimization and batch normalization (BN).

Adam: Adam is a first-order gradient-based optimization algorithm which optimizes the traditional stochastic gradient descent algorithm. It is based on adaptive estimates of lower-order moments.

Batch Normalization: BN is also an optimization trick. It is proposed to solve the vanish and exploding gradient problem in the back-propagation process, while making the overall update pace of weights in different scales more accordant.

E. Tools

In order to facilitate our experiments we used various toolkits and libraries provided for development in Python - Scikit-Learn (for K Nearest Neighbors, Support Vector Machines), Keras (Neural Network, Convolutional Neural Network), Google Colab (GPU Runtime), Pytorch (Residual Neural Network Network, Convolutional Neural Network)

V. Results

A. *Linear SVM as a baseline*

For the baseline classifier, we first implemented the default function provided by Scikit-learn for LinearSVC on different types of feature representations.

TABLE II: Validation accuracy on Linear SVM with various feature representations without hyper-parameter tuning

Feature Space	Validation Accuracy
28x28 Preprocessed Images	**68.77%**
LBP	33.86 %
HOG Features	62.63 %
HOG Features with 21 Principal Components	54.21 %

We then proceeded in tuning certain hyper-parameters of this linear classifier as per mentioned in the methodology section of this report, in order to enhance the validation accuracy score of the classifier for each defined feature space. Table II provides the validation accuracy using the default configuration of the LinearSVC.

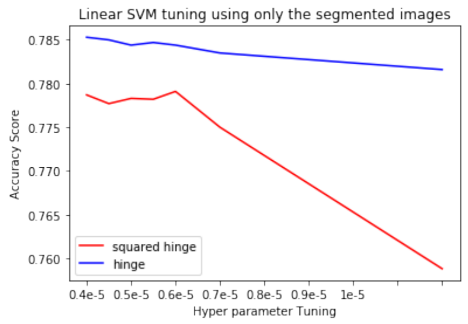

Fig. 1: Tuning linear SVM with respect to the penalty "C" and the loss function hyper-parameters

In Figure1, we presented the tuning of the 28 by 28 pre-processed images. (Figures of remaining tuning experiments are provided in the Appendix for further verification). As we can see, using a loss function of hinge provides us a higher performance in terms of accuracy in comparison with using the squared hinge loss function. The highest accuracy achieved was 78.53% from tuning this feature space corresponding to a value of $C = 4e-06$ and for the hinge loss function. Different experiments were held on the different feature spaces. Therefore, we are providing a summary of the validation accuracy in Table III obtained after tuning this baseline classifier for different feature representations and hyper-parameters.

TABLE III: Validation accuracy on the selected model of Linear SVM with various feature representations after hyper-parameter tuning

Feature space	Validation accuracy after tuning	Values of the hyper-parameters
28x28 Preprocessed images	78.53 %	C= 4e-6; loss function = hinge
LBP	50.37%	C=100; loss function = squared hinge
HOG features	62.96%	C=100; loss function = squared hinge
HOG Features with 21 Principal Components	54.58%	C=0.8; loss function = squared hinge

This table shows that we were able to enhance the accuracy of the baseline Linear SVM for different feature representations through hyper-parameter tweaking. Somehow, certain feature spaces are providing better results in terms of accuracy. In the following subsection (i.e the implemented Neural Network) we will consider the 28x28 feature space) for model selection.

B. Implemented Neural Network

As described in the methodology we did various experiments on the implemented Neural Network (from scratch) by tuning certain hyper-parameters. In Figure 2 , we presented the resulted hyper parameter tuning. As we can notice, by keeping the number of nodes fixed to 50 units, adding hidden layers did not help much in enhancing the performance of our Neural Network in both validation and training sets. On the other hand, keeping the number of hidden layers to one and tuning the learning rate was successful in terms of boosting the performance. Indeed, the accuracy of both validation and training set went from 30% to more than 70%. The highest recorded accuracy was for a learning rate of 0.01. Similarly, varying the number of nodes within a single hidden layer has a drastic effect on accuracy enhancement. By adding more nodes in the layer we tend to enhance the performance of the model. The highest accuracy was recorded for number of nodes equal to 800.

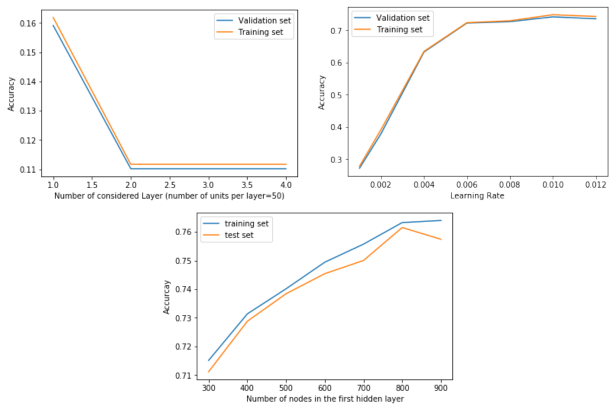

Fig. 2: Tuning the implemented Neural Network (from scratch)

Besides this implementation of Neural Network from scratch, we also performed tuning using Keras Library. We tuned both Neural Networks separately. We considered the number of hidden layers as our hyper-parameters in this case and the number of hidden layers in each hidden layer. We have provided the output of this tuning in the Appendix.

C. Non-linear Classifiers

k Nearest Neighbors : We tested k Nearest Neighbors classifier for a log spaced range of values of **k**, i.e. the number of neighbors to consider for voting:

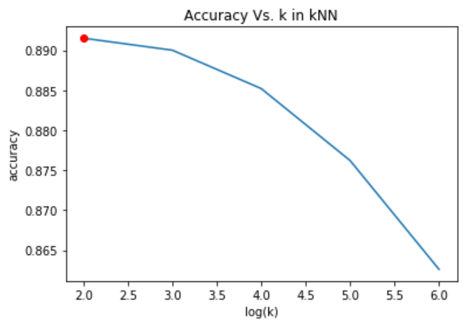

Fig. 3: Tuning kNN with respect to k

Using our preprocessed data, we could achieve accuracy of 89.15% on validation set.

TABLE IV: Accuracy on kNN with Various Preprocessing

	Validation Accuracy (%)
28x28 Preprocessed Images	89.15
Preprocessed with 100 Principal components	88.65
HOG Features	68.55
HOG Features with 21 Principal Components	68.66

Convolutional Neural Networks (CNN): We used thresholding in the original image (T=240). Then considered a simple CNN with three convolutional layers and tuned two hyperparameters on it:

i. **Learning Rate**
From Figure 4, we could observe that when the learning rate is too large, the loss cannot converge so the accuracy is very low. But too small learning rate had little contributions to increase classification accuracy. Thus, we set learning rate to 0.001.

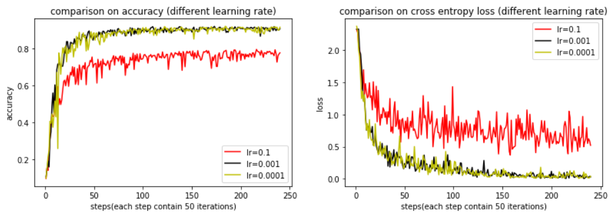

Fig. 4: Classification performance among different learning rate

ii. **Batch Size**

After adding Batch Normalization layers, there is no obvious difference in accuracy for different batch size. In general, within a certain range, the larger the Batch Size is, the more accurate the descending direction is, and the smaller the training oscillation is. For our result, the loss figure also shows that the best batch size is 100.

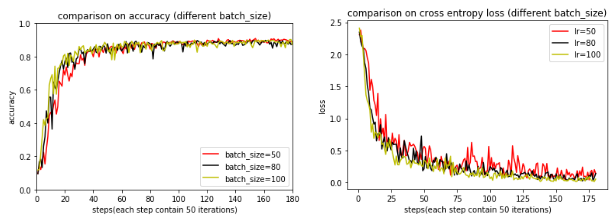

Fig. 5: Classification performance among different batch size

iii. **Structure**

We also tuned the structure of CNN, such as kernel size, depth of output volume and number of layers. Given space limitation, we just show the tuning result of number of convolutional layers.

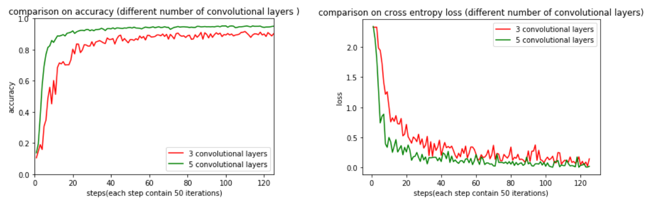

Fig. 6: Classification performance (different number of convolutional layers)

According to Figure 6, adding layers could significantly increase classification accuracy. Besides, the loss of CNN with more convolutional layers converged faster than the loss of CNN with less convolutional layers.

Residual Neural Network (ResNet): We found that the accuracy was saturated at about 95% no matter how we tuned the hyper parameters. Thus, we used ResNet to further improve the accuracy. The architecture of our ResNet was learnt from the 34-layer ResNet [8]. Based on this, we added two more residual blocks so our ResNet had 38 layers. We trained the plain CNN and ResNet, using both original images and cropped images. Results are shown in Figure 7 and Table V.

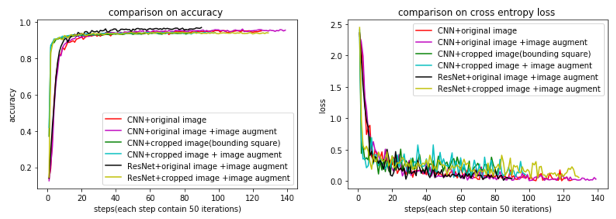

Fig. 7: Performance on classification

TABLE V: Valid accuracy on plain network and ResNet

	Original Images	Cropped Images
CNN	0.9542	0.9342
CNN+image augment	0.9576	0.938
ResNet+image augment	0.9690	0.9428

- **Plain Networks vs ResNet:**
  We found that for both original images and cropped images, ResNet performed better than the plain network. This is reasonable because on the one hand, our ResNet had far more layers than our plain CNN. On the other hand, the residual blocks used in ResNet made ResNet easier to be optimized.
- **Original images vs cropped images:**
  Both networks had higher valid accuracy in original images than in cropped images. However, networks using cropped images converged faster than networks using original images. Using cropped images could reach relative high accuracy in a shorter time. Artificially extracting the largest digit filtered out useless information for the neural networks so they could learn features faster. However, our extraction was not 100% correct even though we used bounding square instead of bounding rectangle in largest digit extraction. Such error degraded the overall classification accuracy.

VI. Discussion

In the previous sections, we have shown different classification models that we have implemented in order to solve the recognition problem of finding the largest digit given an image. In order to find the optimal solution for this problem, we have dealt with both computer vision and machine learning techniques. For the feature design, the main challenge was to find the most appropriate feature representation for our implemented classifiers. Indeed, through different experiments we have done throughout the project we have noticed that certain classifiers works best with pre-processed and feature extracted data. We have witnessed this with our baseline classifier Linear SVM. However, using Neural Networks, CNNs, and ResNet we noticed that working on direct images are more efficient and provides higher performance. In each approach, we have taken into account different hyper-parameters. Tuning these had different outcomes on the performance of our model. For example, it was seen that tuning the learning rate in the implemented Neural Network boosted the accuracy by twice. Same thing for KNNs, tuning the number of neighbors k, has a major impact in enhancing the accuracy of the model. A limitation found in these last mentioned techniques is that it is too dependent on the quality of the data obtained through preprocessing, feature extraction and feature selection. Therefore, we opted for the CNN approach since these networks are best known for handling images. To make sure of this statement we implemented and tuned CNN with preprocessed images instead of the original one. As shown in the result section, the error found in the extraction method of the largest digit degraded the classification accuracy (using the preprocessed images). However, using the 28x28 cropped images is still a promising method since it filters out useless information for the network, so the network could learn features faster. Finally, we also tried Resnet, which gave the highest classification accuracy and which was used in the Kaggle competition. Compared with our plain CNN, Resnet benefits from deep layers and residual blocks. Those advantages make ResNet easy to be optimized. Our future work could revolve into these main ideas: First, we could work on enhancing the image segmentation and feature extraction in order to have a better quality description of each image to feed to our system. We could also, proceed in tweaking other hyper-parameters for CNNs or ResNet in our data using GPUs for faster training. Another idea is to use autoencoders or variational auto encoders (VAE) to generate new and enhanced images for the training of our network.

VII. Statement of Contribution

We divided the task of preprocessing, where each took up a different thread of idea on how to extract the largest digit, where we came together to agree with a single preprocessing for extracting largest digit. We all established the baseline individually, and compared the results. The best results of the three were uploaded. In a nutshell, Sunanda and Yimeng worked on improving accuracy for the task on the Kaggle and Ibtihel focused on the tuning one of the baseline classifiers and exploring the effects of different feature spaces that we have came up with through feature design, and implementing the Neural Network from scratch. The report and results was put together by all of us, together.

We hereby state that all the work presented in this report is that of the authors.

A. *Ibtihel Amara*

She introduced the HOG/LBP features while preprocessing which we further used in the experiments to get a comparison. She implemented the neural network, from scratch.

B. *Sunanda Bansal*

She improved the preprocessing accuracy by implementing minimum area bounding square as a measure of scaling factor. She also implemented and tuned Neural Network and Convolutional Neural Network using Keras.

C. *Yimeng Wu*

She implemented extraction and rotation of the largest digit in the image using the minimum area rectangle in the preprocessing. She also implemented and tuned Convolutional Neural Network and Residual Neural Network using Pytorch.

VIII. Appendix

A. *Linear SVM*

Tuning Linear SVM with other preprocessed data sets

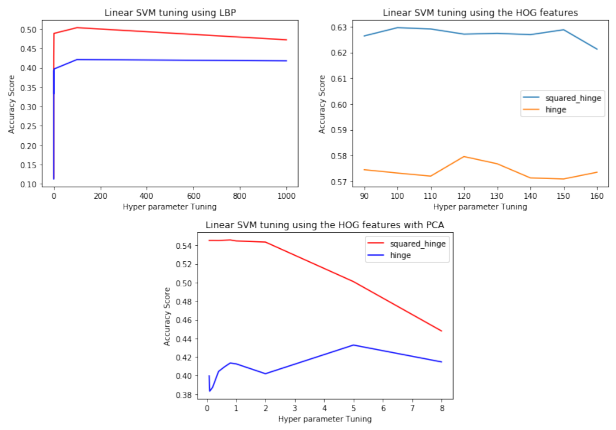

Fig. 8: Tuning Linear SVM using different feature representations

B. *Neural Networks - Keras*

We implemented Neural Networks with Keras and were able to achieve 90.76% of accuracy on the processed data of 28x28 image vectors, flattened out to a vector of 784 features each, with the following model. We tuned the 2-Hidden Layer Neural Network on number of Nodes in each layers in a log spaced range of values :

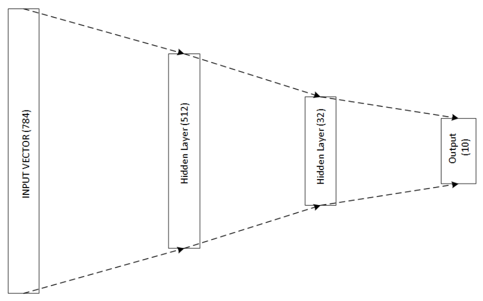

Fig. 9: Neural Network Model (Keras implementation)

With this model we used dropouts for regularization with dropout rate of 20% after each hidden layer. The activation function in Hidden Layer is 'ReLU' and we have used Softmax for output function. The Training and Validation (called Test in the plot) Acuuracies with respect to Epochs are plotted as give below:

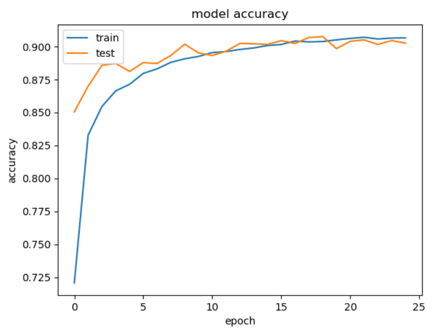

Fig. 10: Plot - Model Accuracy - Neural Network Model (Keras implementation)

We Also tuned our model for dataset with 100 principal components -

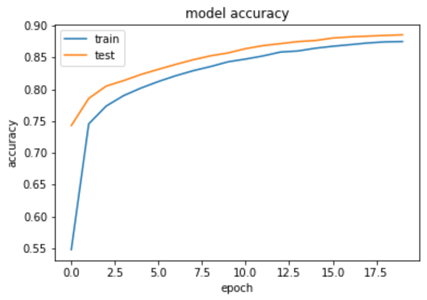

Fig. 11: Plot - Dataset: Preprocessed with PCA

For preprocessed dataset with 100 components we were able to achieve an accuracy of 88.6% with 81 neurons in both hidden layers.

C. CNN structure

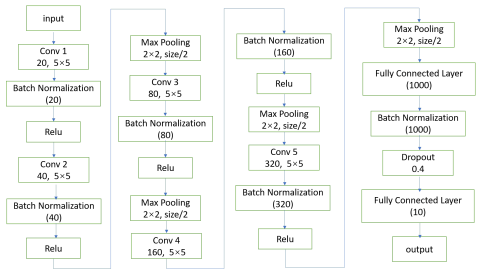

Fig. 12: Structure of the implemented CNN

Note : Because of our regularization measure we did not overfit the training data. Therefore, our training accuracy was marginally different from our Validation accuracy.

References

[1] J. Canny, "A computational approach to edge detection," in *Readings in Computer Vision*. Elsevier, 1987, pp. 184–203.

[2] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern recognition*, vol. 29, no. 1, pp. 51–59, 1996.

[3] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 12, pp. 2037–2041, 2006.

[4] T. Hassan and H. A. Khan, "Handwritten bangla numeral recognition using local binary pattern," in *Electrical Engineering and Information Communication Technology (ICEEICT), 2015 International Conference on*. IEEE, 2015, pp. 1–4.

[5] N. Dalal, B. Triggs, and C. Schmid, "Human detection using oriented histograms of flow and appearance," in *European conference on computer vision*. Springer, 2006, pp. 428–441.

[6] S. Maji and J. Malik, "Fast and accurate digit classification," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-159*, 2009.

[7] "A basic introduction to neural networks," http://pages.cs.wisc.edu/ bolo/shipyard/neural/local.html, accessed: 2018-03-22.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.