Assignment 4
Find the *Largest* Digit

Section: 551-001, Kaggle team: 2gb

Barleen Kaur
ID: 260783838
barleen.kaur@mail.mcgill.ca

Daniel Morales
ID: 260818482
daniel.morales2@mail.mcgill.ca

Somayeh Ghazalbash
ID: 260845408
somayeh.ghazalbash@mail.mcgill.ca

I. Introduction

The MNIST dataset is a large collection of handwritten digits which is popular for training and testing machine learning algorithms for image classification. Modified MNIST dataset[1]is a variant of MNIST dataset in which every image contains two or three handwritten digits. The goal of this project is to predict the label for each image in the modified MNIST dataset where the correct label corresponds to the digit having the maximum area. Examples of images from this dataset are shown in Figure 1 We have used three popular machine learning algorithms for this task. In the first part, five-fold cross-validation with baseline classifiers like logistic regression and Linear Support Vector Machine (SVM) has been performed to establish a floor for classification performance. In the second part, feed forward Neural Network has been implemented from scratch by us. The performance of all the above algorithms is then compared against Convolutional Neural Networks (CNN), a famous deep-learning algorithm designed specifically for image classification. Our results show that CNN significantly outperforms all the other methods.

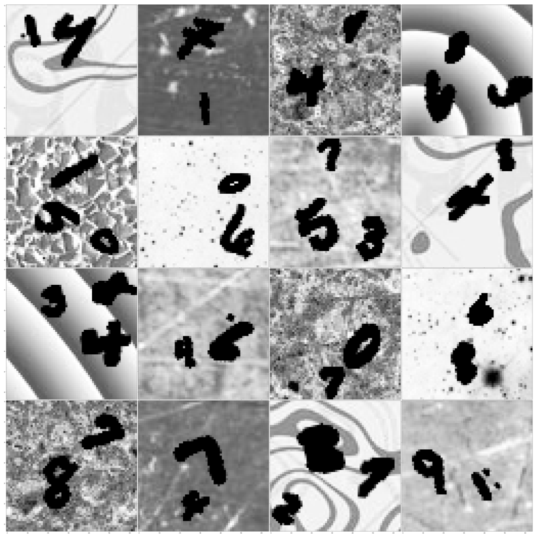

Fig. 1. Examples of modified MNIST dataset

II. Feature Design

Preprocessing of images and feature selection has been performed using the following techniques:

A. Image Thresholding

Image thresholding is used to generate binary images from grayscale images. The idea behind *Image Thresholding* is to remove background noise and clutter by setting all pixel values whose intensity $I_{(x,y)}$ is less than the chosen threshold T to black pixel (i.e. $I_{(x,y)} < T$) or white pixel if $I_{(x,y)} >= T$.

B. Normalization

The value of each pixel is set to be between 0 and 1, by dividing the value by 255.

C. One-Hot encoding

A one-hot encoding (or vector) is how a class is numerically represented. A one-hot vector is a vector which has a single element set to one and all the remaining elements set to zero.This was used for the $y - values$ in the output layer of Neural Network. Decoding was also necessary for accuracy evaluation.

D. Principal Component Analysis (PCA)

The basic idea of principle component analysis (PCA) is to reduce the dimensionality of the dataset into a set of successive orthogonal components that captures as much possible variance in the data[1]. We have used sklearn library[2]to implement PCA.

E. Histograms of Oriented Gradient

Histograms of Oriented Gradient (HOG) is the common feature descriptor in computer vision and image processing. In this method, the distribution of the image gradient in each of selected square cells are used as features[2].

[1]It contains grayscale images of size 64X64 pixels, each containing two or three hand-written digits of different sizes.

[2]PCA using sklearn

F. *Zernike Moments*

Zernike moments are a class of orthogonal moments and it is proved to be a good descriptor for handwritten digit recognition. A set of orthogonal polynomials which is defined on the unit disk are called Zernike Polynomials. The projection of the image (or the interested region) onto orthogonal polynomials could be considered as Zernike Moments[3].

III. Algorithms

We used four machine learning algorithms for our image classification task:

A. *Baseline*

1) SVM: Support vector machine is a supervised machine learning algorithm for solving problems in classification and regression. The idea behind this approach is finding a hyper-plane that best differentiates different classes. Maximizing the margin which is the distance between the nearest support vector and hyper-plane will help to find the best hyper-plane. A complete formulation of this method can be found at a number of publications[4].

2) Logistic regression: Logistic Regression (LR) is a classification model that performs very well on binary categorical variables. The aim of this model is to find the fitting model to explain the relation between categorical outcome variable (dependent variable) and one or more predictors (independent variables). Logistic Regression uses the concept of odd ratio, which is the odds of an event happening to its not happening, to compute the conditional probability. The parameters of the model are computed usually by maximum-likelihood estimation. Further information can be found here[4].

B. *Neural Network*

A Neural Network (NN) is a function that can be used to make predictions; given an input x, it can give a prediction $\hat{y}$. To be able to do that, the NN assigns weights and bias to the dimensions (features) of x. Those weights and bias, known as parameters, are learned through an iterative process. The learning process is composed of the following steps:

1) **Initialization**: The parameters (weights $W = W_0$ and bias $b = b_0$) are initialized randomly
2) **Forward Propagation**: Weight the inputs x with the parameters as follows: $Z = Wx + b$ and generate a prediction $\hat{y} = f(Z)$, where $f(Z)$ is a non-linear function that aims to model non-linearity and complexity of input x to output y. Note that in the case of 2 hidden layers (Deep NN), we consider for the first layer $Z_1 = W_1 x + b_1$, $A_1 = f(Z_1)$, for the second layer $Z_2 = W_2 A_1 + b_2$ and $y = f(Z_2)$. $f(Z) = ReLU(Z)$ for the hidden layers and $f(Z) = Softmax(Z)$ for last layer (also known has output layer). Softmax is defined by relation 1:
$$S(y_i) = \frac{e^{y_i}}{\sum_j^J e^(y_j)} \tag{1}$$
ReLU activation function is defined by relation 2:
$$ReLU(Z) = max(0, Z) \tag{2}$$
3) **Cost**: It measures the difference between the predicted label $\hat{y}$ and actual labels y. A sample cost function C can be of the form $C = |y - \hat{y}|$. Note that C is a scalar.
4) **Backward Propagation**: Calculate gradients of the parameters with respect to the cost function C. This is basically the application of chain rule to get the relation described in 3 for each hidden layer.
$$\frac{\partial C}{\partial W}, \frac{\partial C}{\partial b} \tag{3}$$
5) **Update Parameters**: The parameters W and b are updated in order to minimize the cost C. This is done by using relations 4 and 5:
$$W_i = W_{i-1} - \alpha \frac{\partial C}{\partial W} \tag{4}$$
$$b_i = b_{i-1} - \alpha \frac{\partial C}{\partial b} \tag{5}$$
where α is an hyper-parameter representing learning rate.

Steps 2 to 5 are iterated until a certain arbitrary accuracy is achieved, or certain number of iterations have been performed.

C. *Convolutional Neural Network*

The Convolutional Neural Networks (CNNs) are special kinds of deep neural networks specifically designed for performing tasks related to image classification, image captioning and object detection. The simple architecture of CNNs consists of blocks of convolutional layers, pooling layers and fully connected layers which are stacked up in order to find the spatial dependencies in an image. A convolutional layer has multiple filters which are convolved around the entire image to build multiple feature maps. Each unit in the feature map is obtained by the dot product of filter weights with pixel values of the local region in the image captured by the filter. The size of filters, number of feature maps and stride are hyper-parameters that need to be tuned for obtaining the best model. Moreover, all units in a feature map share the same set of parameters, i.e. the same filter is convolved over all the different local regions in the image.

The output of convolutional layer is passed as an input to the pooling layer which performs a down-sampling operation along the spatial dimensions. Batch normalization[5] is used to normalize the inputs of each layer so that there is zero mean output activation and standard deviation of one. Dropout layer[6] helps in regularization of the model by randomly setting some activation neurons to zero.

The parameters of the model are learned using gradient descent as the optimization strategy with a categorical cross-entropy loss function described as in relation 6:
$$L = \sum_{i=1}^{n} \sum_{c=1}^{K} t_{i,c} \log{(p_{i,c})} \tag{6}$$

where $t_{i,c}$ is the true label for an instance i and $p_{i,c}$ is the probability of classifying an instance i to class c by our model.

The output layer in a CNN is a fully connected layer which calculates the scores of an input image belonging to each class.

IV. Methodology

A. Baseline

All computations were implemented using the Python software package. We used OpenCV library[3]for Thresholding, skimage library[4], pyzernikemoment library[7] for feature extraction and scikit-learn library for PCA[5]and classification[6]. We applied a bunch of features extraction techniques to input data. Firstly, we reduced the dimensionality of features from 4096 to 256, 512, 1024 using PCA. We noticed that increasing the number of components from 256 to 1024 didn't make a different in terms of the amount of variance it captured in the data. So, we used only 256 components to train the classifiers: Logistic Regression and Linear SVM. Hyper-parameters of the model like learning rate [1e-8,1e-3], regularization coefficient[1e-3,100] were tuned using five-fold cross validation. This prevented the model from getting overfitted. In the case of Histograms of Oriented Gradient(HOG), the image was divided into 16x16 cells and a histogram of gradients was calculated for each of 16x16 cells. The computed gradient of all the pixels in the 16x16 cells were integrated to create an 8-bin histogram. Then, the cells were congregated into blocks and for each block, all cell histograms were normalized. To compute Zernike moment, we set order and the repetition of the moment to 4 and 2, respectively. The amplitude and angle of the moment was used as final features. Then, the number of features after applying Zernike moment changed from 4096 to 2 features. In case of HOG and Zernike moment, error term penalty (1e-7,1e-4, ,0.1,1) and inverse of regularization strength (0.0001 to 100) were considered as hyper-parameters for SVM and logistic regression respectively.

B. Neural Network

One hidden layer (1HL) was implemented, for then to be extended to more layers (Deep), with the expectation to increase prediction accuracy, by adding complexity and nonlinearity. For both NN: 1HL and Deep NN, the dataset was split into two sets: 40,000 samples as training set and 1,000 samples as validation set. Image thresholding, Normalization were used as pre-processing step in all computations. Feature reduction technique, PCA[8], was also implemented aiming to have faster computations. Input dimensions were reduced from 4,096 to 512 and 256.

The strategy was to increase the quantity of neurons and layers in the network to then evaluate its performance based on validation accuracy.

Given the high quantity of hyper-parameters to tune, different combinations of them were used, to have a broader spectrum of solutions.

C. Convolutional Neural Network

As training a CNN model takes a lot of time, doing hyper-parameter tuning on a cross validation dataset is not feasible. Hence, we used 5000 training data points to train a CNN model varying the learning rate in the range [1e-4,1]. In order to ensure if our network was learning and gradients were propagating back through the network well, we tried to overfit our model on this small set of data and checked if the loss reduced with the number of iterations. Over 10 experimentations on the architecture were conducted by varying the depth of the network and number of filters in each layer. The implementation of CNN[7]was done using Pytorch [8]. We used filter size of (3,3), stride of 1 and zero-padding of 1 inspired by the VGGNet architecture[9]. Batch normalization and dropout layers were also added to ensure regularization. We used ReLU activation as it is solves the vanishing gradient problem as shown in literature[10]. In order to find the best model, we trained our CNN models using 80% of our data as training data and evaluated the performance of these models using rest 20% as validation data. We also experimented by preprocessing our input using thresholding and normalization techniques. For prediction on the test set, we used monte-carlo approximation by making 10 forward passes through the network with different dropouts masks and then taking the final label as the one which is predicted the most number of times[11].

V. Results

A. Baseline

The classification accuracy of LinearSVM and Logistic Regression(LR) on the cross-validation set based on different feature extraction methods including PCA, HOG and Zernike method was compared with baseline performance. For baseline performance, we ran LinearSVM classifier without doing any pre-processing of the images. For PCA, the best learning rate for Logistic regression and LinearSVM was 2.1e-06 and 4.6e-07 whereas the regularization coefficient was 1.29 and 3.59 respectively. For HOG and Zernike, the best five-fold cross-validation performance was observed when the error term penalty (C) was set to 0.01 for LinearSVM and inverse of regularization coefficient was set to 10 in case of Logistic regression as shown in Figure **??**. The accuracy on test and validation set using different feature extraction techniques on logistic regression and LinearSVM are shown in Figure 2

B. Neural Network

The best validation accuracy achieved with one hidden layer NN was 18% and it's shown in Figure 3. It's the result of the hidden layer with the highest quantity of neurons tried (100 and 500 neurons were tried).

[6]link to OpenCV
[6]http://scikit-image.org/
[6]PCA using sklearn
[6]scikit-learn library for Classification

[8]Colab link for CNN's implementation
[8]http://pytorch.org/

Algorithm	Accuracy	No Preprocessing	PCA	HOG	Zernike
Linear SVM	Cross Validation	11.57%	15.9%	17.8%	12.7%
	Test (Kaggle)	-	15.59%	17.32%	12.61%
Logistic Regression	Cross Validation	-	15.9%	18.1%	12.9%
	Test (Kaggle)	-	15.36%	17.57%	12.15%

Fig. 2. Test (Kaggle) and five-fold cross validation accuracy obtained on classifiers Logistic regression and LinearSVM using different features extraction techniques.

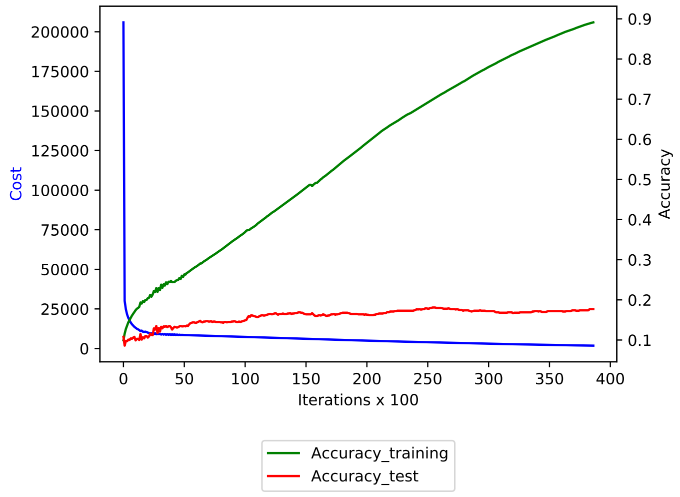

Fig. 3. NN 1HL, 500 neurons activated through ReLu and Softmax for hidden and output layer respectively. Maximum accuracy on validation set $\sim 18\%$. It took $\sim$ 24 hrs to run. Learning rate: $1e-4$.

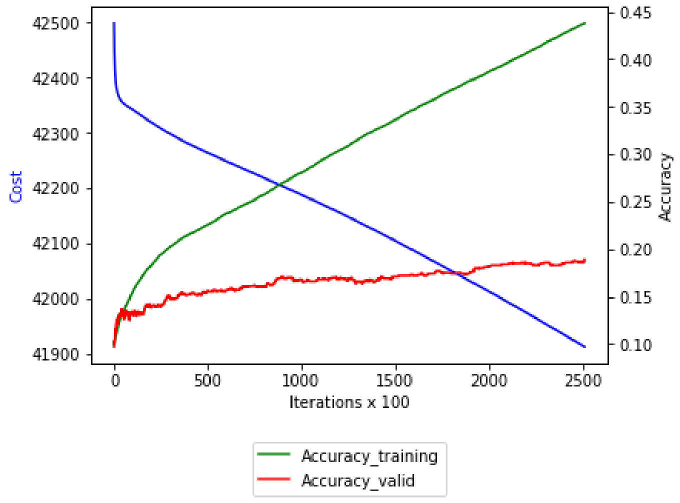

Fig. 4. NN Deep1. 4,096,200,100,10 nodes in each layer. Maximum accuracy on validation set $\sim$ 19%. It took $\sim$ 72 hrs to run. Computation interrupted because of time constraint. Learning rate: $1e-2$.

The best validation accuracy achieved with NN Deep were 19% and 21% and are shown in Figures 4 and 5 respectively.

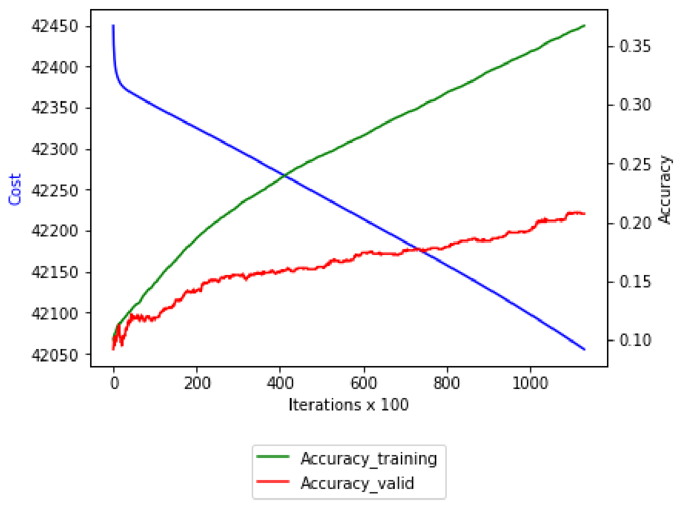

Fig. 5. NN Deep2. 4,096,500,250,125,50,10 nodes in each layer. Maximum accuracy on validation set $\sim$ 21%. It took $\sim$ 72 hrs to run. Computation interrupted because of time constraint. Learning rate: $1e-2$.

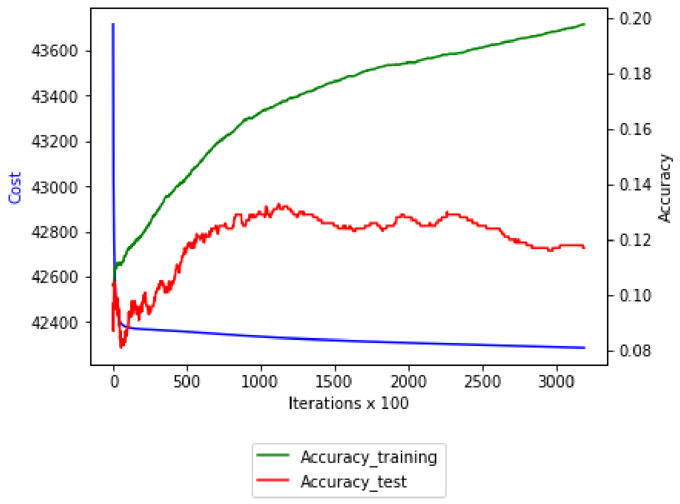

Fig. 6. NN Deep, PCA256. 256,200,100,10 nodes per input, hidden layer 1, hidden layer 2 and output respectively. Maximum accuracy on validation set $\sim$ 14%. It took $\sim$ 24 hrs to run. Computation interrupted because of diminishing performance on test set. Learning rate $5e-3$.

PCA was also implemented with the aim to reduce computation time. The results are shown in Figures 6 and 7, but for the values tried, and given time constraint, they didn't show better improvement in validation accuracy than analyzing the original 4,096 dimensions. They also showed signs of over-fitting, by reducing the accuracy in the validation set as iterations went on. The summary of validation and test accuracy for NN is shown in Table I.

C. Convolutional Neural Network

After evaluating over 10 architectures for CNN, we observed that as the depth of the network increased, the model was able to capture more complex patterns in the input image. However, the model got saturated if more layers were added

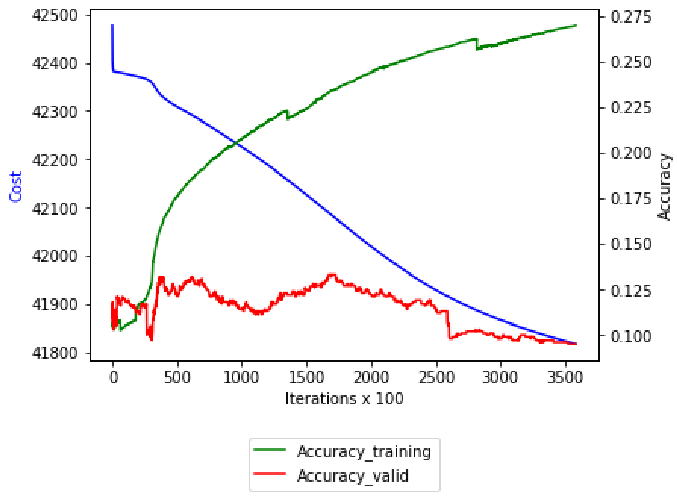

Fig. 7. NN Deep, PCA512. 512,200,50,10 nodes per input, hidden layer 1, hidden layer 2 and output respectively. Maximum accuracy on validation set $\sim$ 13%. It took $\sim$ 24 hrs to run. Computation interrupted because of diminishing performance on test set. Learning rate $5e-2$.

TABLE I
RESULTS FOR NEURAL NETWORK

Algorithm	Max Accuracy Valid	Max Accuracy Test
NN 1HL	18%	-
NN Deep1	19%	20%
NN Deep2	21%	-
NN Deep,PCA256	13%	-
NN Deep,PCA512	13%	-

to it beyond a certain limit. Best results on the validation set were obtained using the VGGNet like architecture as shown in Figure 8. The optimal learning rate was found to be in the range [1e-3,1e-1]. Any learning rate higher than the optimal range caused the model to diverge from the minima and smaller learning rates slowed down the process of learning. The initial learning rate was set to be 1e-1 and then reduced by a factor of 10 every 7 epochs as the performance would increase by annealing the learning which is also inline with the theory[12]. We ran our model for a total of 25 epochs as the loss on training and validation data saturates after a certain number of iterations as shown in Figure 9. We observed an increase of over 35% in the train accuracy using batch normalization and dropout as they avoid overfitting on the training data. Using thresholding (keeping the threshold value at 250) as a preprocessing step improved accuracy on the training and validation set since it removes the background clutter in the image. We achieved test accuracy of 96.7% as shown in Table II.

TABLE II
ACCURACY RESULTS FOR BEST CNN MODEL

Algorithm	Train	Validation	Test (Kaggle)
Convolutional NN	99.94%	95.89%	96.7% [9]

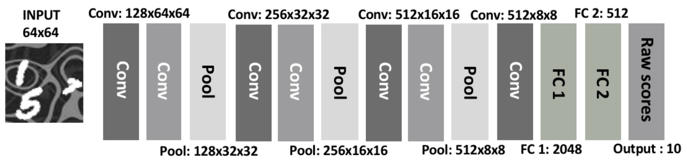

Fig. 8. Best CNN architecture

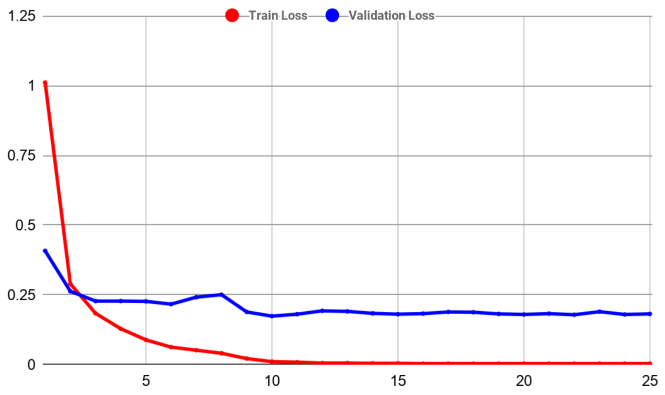

Fig. 9. Variation of loss with number of epochs (25 total) in CNN

VI. DISCUSSION

A. Baseline

In this part, several feature extraction methods are applied before classification. An advantage of using feature extraction algorithm is that it is less computationally expensive compared to using the non-preprocessed images as an input to classifier.

B. Neural Network

Results obtained suggests that deeper the network, better the accuracy. Note that the deepest NN (NN Deep2, Figure 5) shows less difference between training and validation performance, than the shallower NN (NN Deep1, Figure 4). For both NN shown in Figures 4 (NN Deep1) and 5 (NN Deep2), computations were interrupted because of time constraints, and neither showed signs of over-fitting[10]. This suggest a future development to improve the computation through the use of GPU's, which would make computation faster, and produce fast feedback to adjust hyper-parameters. Adaptive learning rates could also be useful. Other pre-processing techniques could apply, but that's against the purpose of minimizing human intervention in computation, which could add bias.

With results obtained, it is not clear if PCA (to 256 and 512 dimensions) is a good pre-processing technique. It can be said that reducing the dimensions diminishes the performance of NN. It appears that to give a better prediction, all the information available in the images should be used. It can be interpreted that the accuracy depends strongly in the details of the images.

[9]Submitted to Kaggle competition
[10]Validation accuracy diminishing while increasing training accuracy.

It's also important to note the non-linearity of the computations. In Figures 6 and 7, the difference is with the number of dimensions, the learning rate and the size of last hidden layer. The difference shows a massive difference in the behaviour of cost curve.

C. Convolutional Neural Network

Our best VGG-Net like architecture of CNN gives an accuracy of 96.7% on test data as per kaggle leaderboard. As initial layers in the network capture simple patterns and later layers in CNN capture more complex patterns[13], our experimentation results aligns with the same fact that the increase in depth of the CNN network, to some extent, improves the accuracy of the model. However, we believe that there are much better architectures in the literature like ResNet[14] and Google Inception model[15] which we couldn't try due to time constraints. We also wanted to experiment with pre-processing techniques like contouring to get the digit with highest area and then pass these highest digits as input to our Convolutional Neural network though it introduces bias into the model.

VII. Conclusions

The classification algorithm which achieved the best accuracy of 96.7% on test set was Convolutional Neural Network. Single and multiple hidden layers Neural Network were successfully implemented from the scratch. The quantity of hyperparameters and the non-linearity makes the tuning of Deep Neural Networks an art which requires experience to get the best results.

VIII. Statement of Contributions

Beside mutual comments, discussions and support, the work was divided as:

- Barleen: Baseline with PCA and Convolutional Neural Network.
- Daniel : Neural Network implementation from scratch.
- Somayeh: Baseline with pre-processing techniques(HOG, Zernike moment) and helped in NN implementation.

All of the authors participated in the elaborations of their respective work in the report.

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

References

[1] I. T. Jolliffe, "Principal component analysis and factor analysis," in *Principal component analysis*. Springer, 1986, pp. 115–128.

[2] Y. Li and G. Su, "Simplified histograms of oriented gradient features extraction algorithm for the hardware implementation," in *Computers, Communications, and Systems (ICCCS), International Conference on*. IEEE, 2015, pp. 192–195.

[3] P. K. Singh, R. Sarkar, and M. Nasipuri, "A study of moment based features on handwritten digit recognition," *Applied Computational Intelligence and Soft Computing*, vol. 2016, p. 4, 2016.

[4] M. B. Christopher, *PATTERN RECOGNITION AND MACHINE LEARNING*. Springer-Verlag New York, 2016.

[5] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. JMLR Proceedings, vol. 37. JMLR.org, 2015, pp. 448–456.

[6] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014. [Online]. Available: http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

[7] A. Tahmasbi, F. Saki, and S. B. Shokouhi, "Classification of benign and malignant masses based on zernike moments," *Computers in biology and medicine*, vol. 41, no. 8, pp. 726–735, 2011.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[11] F. F. L. Andrej Karpathy and J. Johnson, *Standford CS 231n*, 2016, vol. Lecture 6, no. Slide 55. [Online]. Available: http://cs231n.stanford.edu/slides/2016/winter1516_lecture6.pdf

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. Equation 2.7.

[13] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[14] 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.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going deeper with convolutions." Cvpr, 2015.