Chapter No. 05

Data Gathering

The purpose of this chapter is to explain what data was collected, how data was collected and how the collected data will be processed.

- **Explaining what data was collected :**

We have used MNIST dataset for this project, this dataset is famous for digit recognition

The MNIST database (National Institute of Standards and Technology) is a large database of handwritten digits that is commonly used for training various image processing systems.The database is also widely used for training and testing in the field of machine learning.It was created by "re-mixing" the samples from NIST's original datasets.The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset The original creators of the database keep a list of some of the methods tested on it. An extended dataset similar to MNIST called EMNIST has been published in 2017, which contains 240,000 training images, and 40,000 testing images of handwritten digits and characters.
The set of images in the MNIST database is a combination of two of NIST's databases: Special Database 1 and Special Database 3. Special Database 1 and Special Database 3 consist of digits written by high school students and employees of the United States Census Bureau, respectively.

The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census

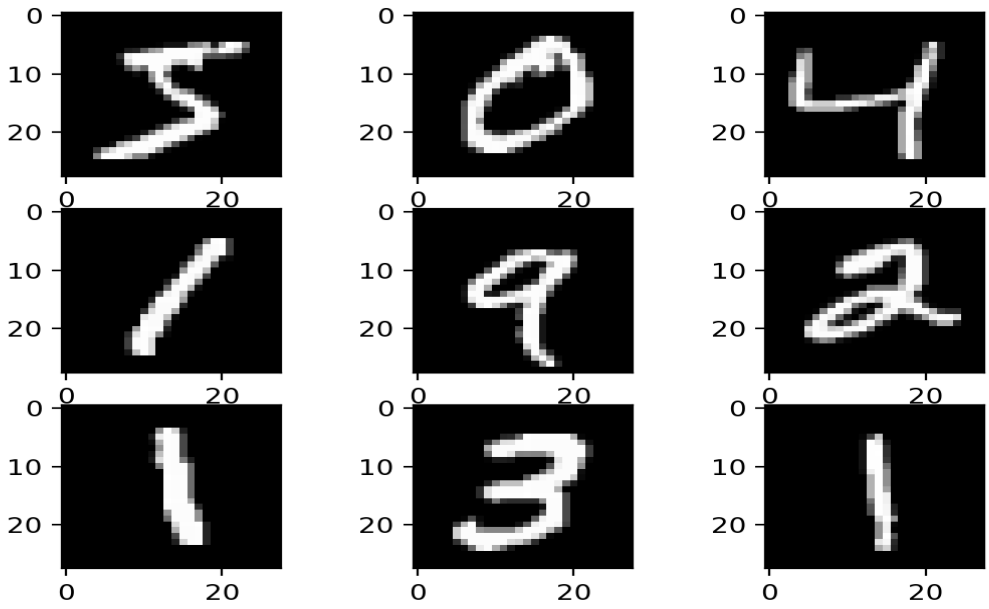
Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Therefore it was necessary to build a new database by mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1. The 60,000 pattern training set contained examples from approximately 250 writers. We made sure that the sets of writers of the training set and test set were disjoint.

SD-1 contains 58,527 digit images written by 500 different writers. In contrast to SD-3, where blocks of data from each writer appeared in sequence, the data in SD-1 is scrambled. Writer identities for SD-1 is available and we used this information to unscramble the writers. We then split SD-1 in two: characters written by the first 250 writers went into our new training set. The remaining 250 writers were placed in our test set. Thus we had two sets with nearly 30,000 examples each. The new training set was completed with enough examples from SD-3, starting at pattern # 0, to make a full set of 60,000 training patterns. Similarly, the new test set was completed with SD-3 examples starting at pattern # 35,000 to make a full set with 60,000 test patterns. Only a subset of 10,000 test images (5,000 from SD-1 and 5,000 from SD-3) is available on this site. The full 60,000 sample training set is available.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

Sample images from MNIST test dataset



- **Explaining how the data is collected and processed:**

The MNIST dataset is one of the most common datasets used for image classification and accessible from many different sources. In fact, even Tensorflow and Keras allow us to import and download the MNIST dataset directly from their API. Therefore, I will start with the following two lines to import TensorFlow and MNIST dataset under the Keras API.

In [1]:
```
from keras.datasets import mnist
import cv2
import os
os.mkdir('dataset')
# load train and test dataset
# load dataset
(trainX, trainY), (testX, testY) = mnist.load_data()
# reshape dataset to have a single channel
trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
testX = testX.reshape((testX.shape[0], 28, 28, 1))
count=0
for img,label in zip(trainX,trainY):
    if not os.path.exists('dataset/'+str(label)):
        os.mkdir('dataset/'+str(label))
    count=count+1
    cv2.imwrite('dataset/'+str(label)+"/"+str(count)+'.jpg',img)
for img,label in zip(testX,testY):
    if not os.path.exists('dataset/'+str(label)):
        os.mkdir('dataset/'+str(label))
    count=count+1
    cv2.imwrite('dataset/'+str(label)+"/"+str(count)+'.jpg',img)
```

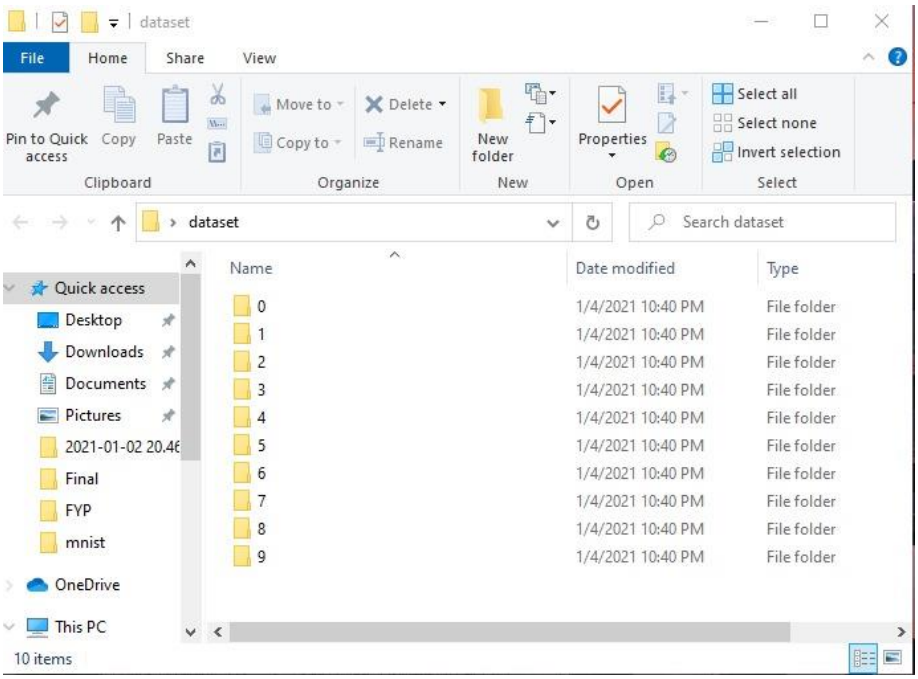
We have used MNST Dataset for following reasons:

- MNSIT dataset is publicly available.
- The data requires little to no processing before using.
- It is a voluminous dataset.

Additionally, this dataset is commonly used in courses on image processing and machine learning.

In processing a dataset we look for null values if there exist any null value we fill it up, the dataset which we are using is i.e. MNIST dataset, we get an advantage that this data does not need a lot of processing because it does not contain null values so, no cleansing of data is needed. In the figure given above we have given command in such a way that data is loaded and saved in folders (e.g. all the images of 1 in 1's folder, same with 2, 3, 4 etc.) , this command also reshapes dataset into channel. For example if we null/delete values in dataset the accuracy can be compromised as we have saved all images folder wise (in the class of 1 we have all images of 1and same with 2 3 4 so on).

Basically whenever we look for dataset first we cleanse dataset if there exist any column that we want to delete or also if we want to add a column we have to to cleansing , and also if null values exist in dataset we fill up all the null values by giving different commands but in the case og mnist dataset we have not done any of.



Chapter NO. 06

Results and discussion

In this chapter first we will show all the results in the form of graphs, tables, images etc. and then we will discuss those results we have got from data gathering phase (i.e. chapter no. 5).

Model summary:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 32)        896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3136)              0
_________________________________________________________________
dropout_5 (Dropout)          (None, 3136)              0
_________________________________________________________________
dense_4 (Dense)              (None, 100)               313700
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1010
=================================================================
Total params: 334,102
Trainable params: 334,102
Non-trainable params: 0
```

The above given picture is summary of the model that we have built i.e. Convolutional Neural Network. In this image we can see that multiple convolutional of layers are used here and types of layers can also be seen and we can also see that how many of parameters are trainable and which of them are not trainable. It actually clarifies our model, its layers and its output shape. E.g. flatten_2 is one of the layers and its output shape is (none, 3136) and its parameter is 0.

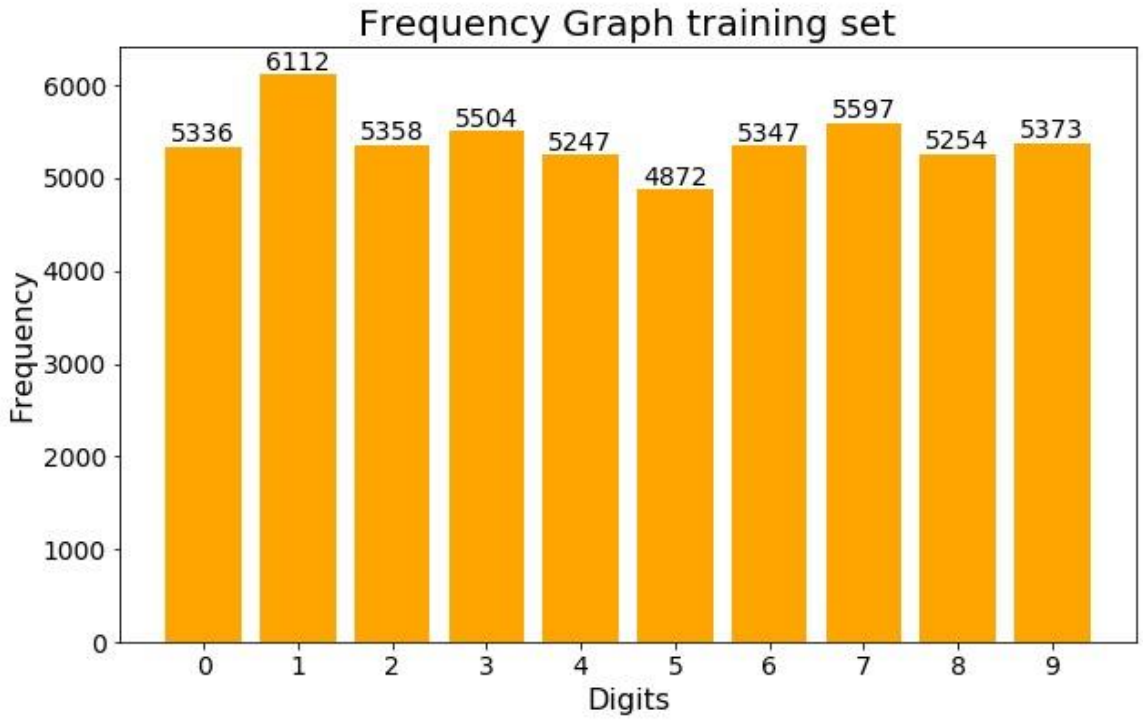
Training and testing samples dimensions:

Training sample dimension	60000, 28, 28, 1
Testing sample dimensions	10000, 28, 28, 1

In MNIST dataset we have 60,000 training samples on which we will train the model and 10,000 testing samples on which we will test samples.

Frequency graph training set:

Now we will discuss about the frequency graph of training set. In the graph given below we can see that every digit from 0 to 9 is classified, in every graph there are different number of digits e.g. in the class of 8 there are 5,254 images. After data gathering phase when we evaluated the frequencies images were divided into these classes.



Accuracy of model:

Now we will discuss the overall accuracy of the model when it was trained.

```
Epoch 1/8
6563/6563 - 20s - loss: 0.3035 - accuracy: 0.9384 - val_loss: 0.0991 - val_accuracy: 0.9694
Epoch 2/8
6563/6563 - 19s - loss: 0.1177 - accuracy: 0.9671 - val_loss: 0.0769 - val_accuracy: 0.9763
Epoch 3/8
6563/6563 - 19s - loss: 0.0960 - accuracy: 0.9737 - val_loss: 0.0920 - val_accuracy: 0.9754
Epoch 4/8
6563/6563 - 19s - loss: 0.0872 - accuracy: 0.9761 - val_loss: 0.0804 - val_accuracy: 0.9791
Epoch 5/8
6563/6563 - 19s - loss: 0.0893 - accuracy: 0.9768 - val_loss: 0.0812 - val_accuracy: 0.9787
Epoch 6/8
6563/6563 - 18s - loss: 0.0786 - accuracy: 0.9799 - val_loss: 0.0895 - val_accuracy: 0.9785
Epoch 7/8
6563/6563 - 18s - loss: 0.0785 - accuracy: 0.9801 - val_loss: 0.0774 - val_accuracy: 0.9823
Epoch 8/8
6563/6563 - 18s - loss: 0.0771 - accuracy: 0.9809 - val_loss: 0.0810 - val_accuracy: 0.9807
```

When we loaded MNIST dataset, built model and made it fit. After training model we got accuracy results e.g. in the image we can see in the image that we have made the size of Epoch 8 and at every Epoch we got different accuracy and loss. E.g. in the first Epoch we have got 0.3035 loss, 0.9384 accuracy, 0.0991 validation loss and 0.9694 validation accuracy. As we can see that at every Epochs the accuracy of the model is is improving gradually. At Epoch 1/8 the accuracy is 0.9384 and after that when the model is being trained the accuracy at Epoch 8/8 is 0.9809 with loss of 0.0771. We can see in the above image the accuracy of every epoch from Epoch 1/8 to the Epoch 8/8.

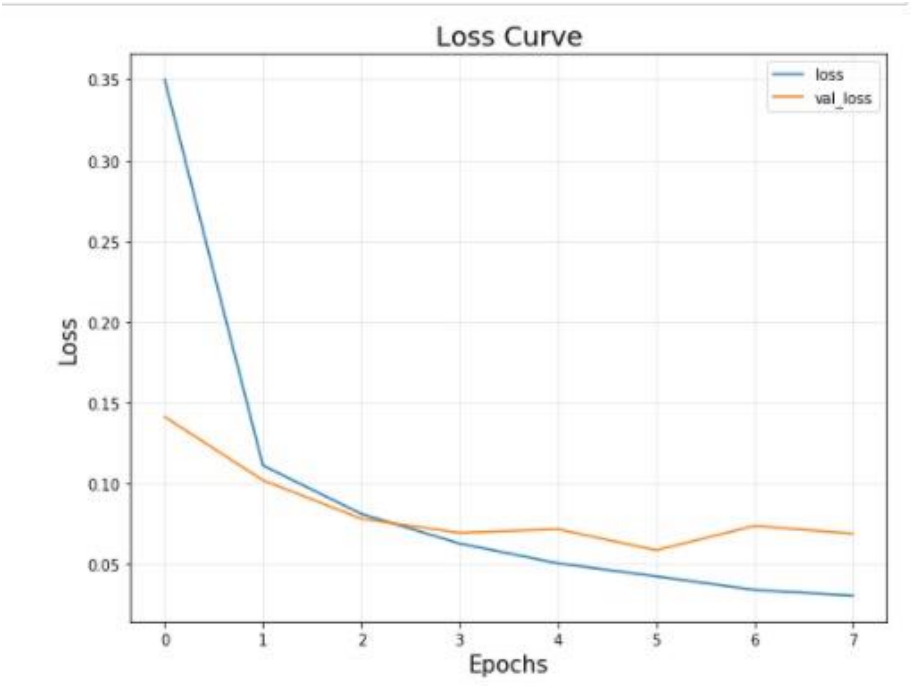
So the final Accuracy is 0.98 with the loss of 0.07 Loss:0.07 Accuracy :0.98

Classification report:

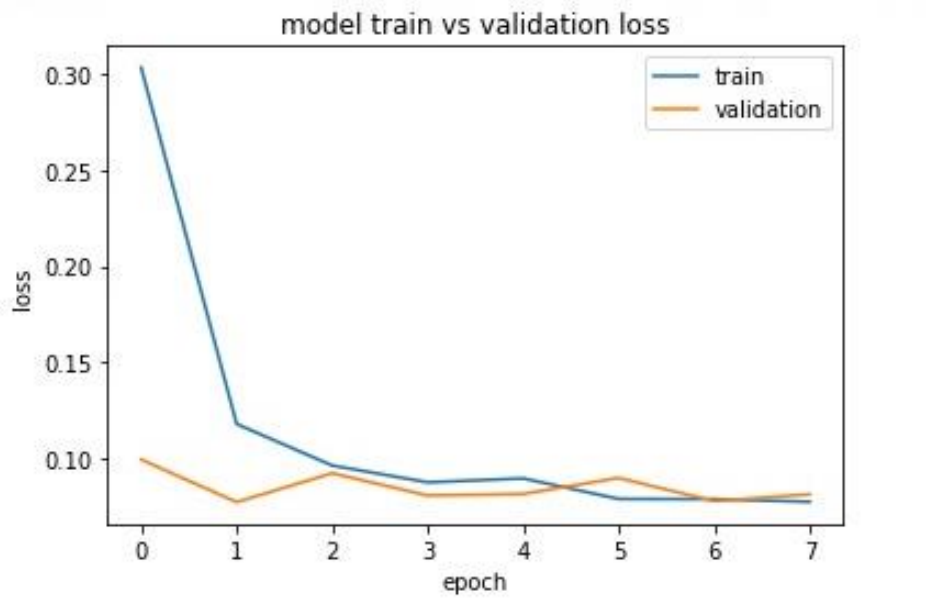
This is the classification report of trained model. In this report we can see that the accuracy of every class from 0 to 9. When we loaded data classes of separate digits were made e.g. the accuracy of class zero is 0.99 the accuracy of class 1 is 0.98 at the accuracy of class 2 to is 0.99 and so on we can see accuracy of every class in this report.

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1969
1	0.98	0.99	0.98	1748
2	0.99	0.98	0.99	1785
3	0.99	0.98	0.98	1706
4	0.98	0.98	0.98	1578
5	0.99	0.99	0.99	1719
6	0.99	0.98	0.98	1823
7	0.97	0.98	0.98	1706
8	0.96	0.98	0.97	1740
9	0.98	0.99	0.99	1726
micro avg	0.98	0.98	0.98	17500
macro avg	0.98	0.98	0.98	17500
weighted avg	0.98	0.98	0.98	17500
samples avg	0.98	0.98	0.98	17500

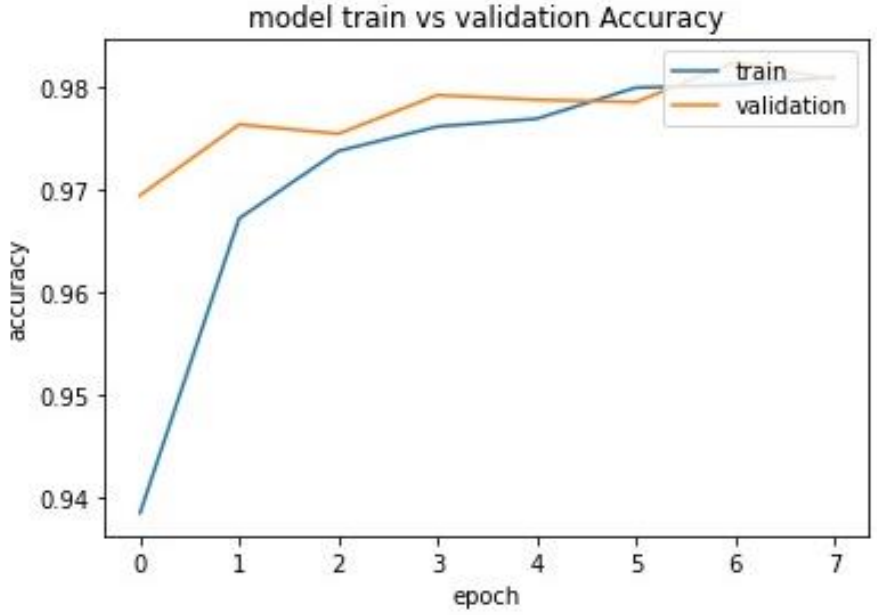
Graph for accuracy and loss:



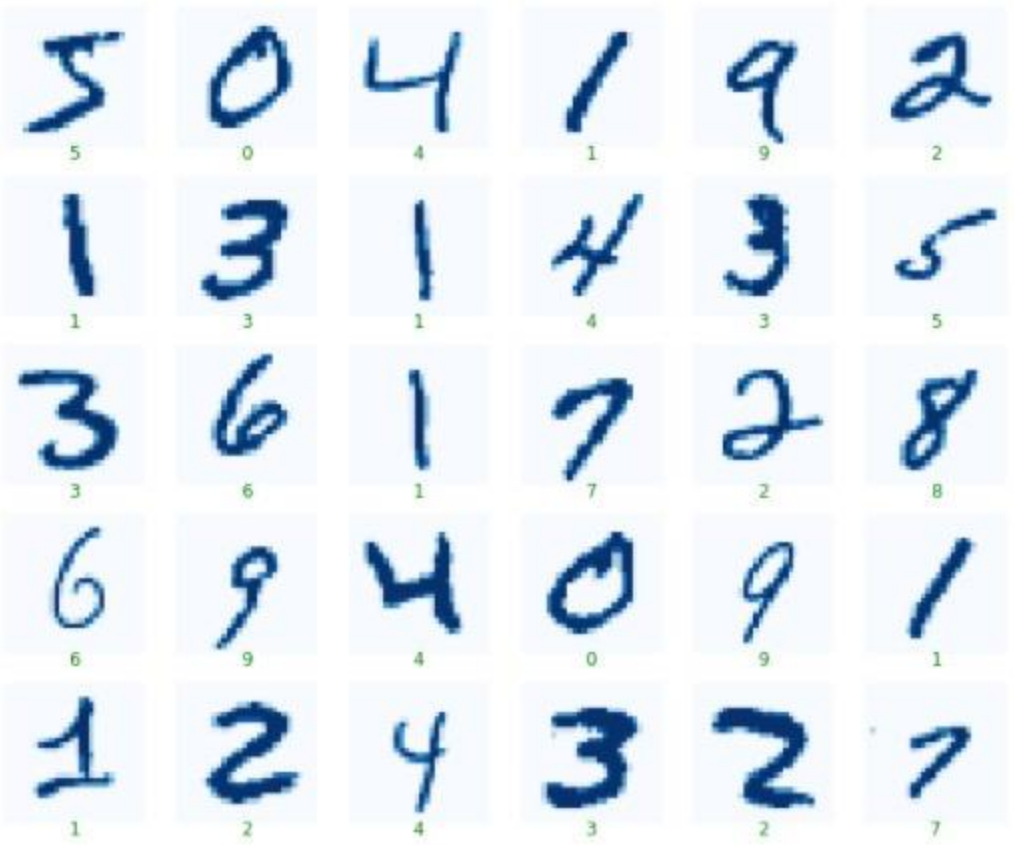
In this graph we can see that as we are approaching towards 8/8 Epochs the loss is generally decreasing and our accuracy is increasing.



As the size of Epoch here is 8 so as we are approaching towards 8/8 Epoch loss validation is also generally decreasing



When we are approaching Epoch 8/8 validation accuracy of the model is also increasing as we can also see this in the graph given above.



In this image we can see that when the model was trained we tested our model to see that if it can predict numbers accurately or not and we can clearly see that different digits are written in different hand writings but still our model can predict them.

As a final verdict we can say that after explaining and showing all tables and graphs of the results the accuracy of our model got improved as we loaded data, we can see that CNN is a best classifier as it classified all the digits very perfectly.

Comparison with previous work:

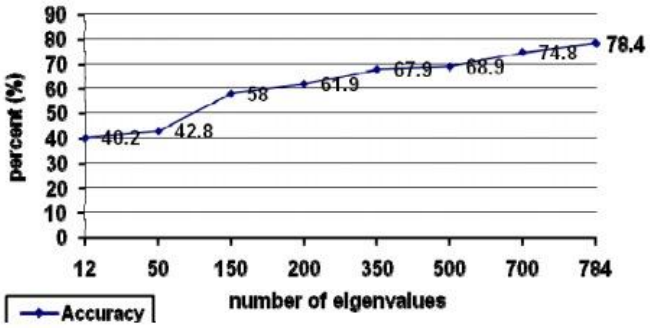
A lot of work have been done on this project and people have used different types of classifiers they also have improved the accuracy of the model, but when we did this project our self our accuracy is 98% from the previous work we can see that in this paper that I have just given in the figure accuracy is 78.4%. previous work is on different dataset different models different classifiers like SVM, K-NN, A-NN, S-NN etc. (Gaurav Jain, Jason Ko, Handwritten DigitsRecognition,Project Report, University of Toronto, 11/21/2008.) he used PCA algorithm to get this accuracy. A lot of other researchers also have got good accuracies like 80%, 90% etc.

Handwritten digits recognition

Gaurav Jain, Jason Ko

Multimedia Systems, Project Report, University of Toronto, 1-3, 2008

The aim of this project is to implement a classification algorithm to recognize handwritten digits (0-9). It has been shown in pattern recognition that no single classifier performs the best for all pattern classification problems consistently. Hence, the scope of the project also included the elementary study the different classifiers and combination methods, and evaluate the caveats around their performance in this particular problem of handwritten digit recognition. This report presents our implementation of the Principal Component Analysis (PCA) combined with 1-Nearest Neighbor to recognize the numeral digits, and discusses the other different classification patterns. We were able to achieve an accuracy rate of 78.4%.



Graph for Accuracy using different number of eigenvalues

Chapter No. 07

Conclusion

This work aims to improve an accuracy and performance of handwritten digit recognition. We evaluated variants of a convolutional neural network to avoid complex pre-processing, costly feature extraction and a complex ensemble (classifier combination) approach of a traditional recognition system. Through extensive evaluation using a MNIST dataset, the present work suggests the role of various hyper-parameters.

We also verified that fine tuning of hyper-parameters is essential in improving an accuracy and performance of CNN architecture. We achieved an accuracy rate of 98% using CNN model with the Adam optimizer for the MNIST database which gives better results than previously reported optimizers and for the backend, Tensorflow was used while building this model. Alongside these many python libraries were used (imported) for several purposes and different techniques were applied to achieve our desired accuracy of the model. Comparing to other research methods, CNN architecture works better by improving the accuracy of the classification models. Utilizing these deep learning techniques, a high amount of accuracy can be obtained.

The effect of increasing the number of convolutional layers in CNN architecture on the performance of handwritten digit recognition is clearly presented through the experiments. The novelty of the present work is that it thoroughly investigates all the parameters of CNN architecture that deliver best recognition accuracy for a MNIST dataset.

Some researchers used ensemble CNN network architectures for the same dataset to improve their recognition accuracy at the cost of increased computational cost and high testing complexity but with comparable accuracy as achieved in the present work.

Future Direction of the Research:

In future, different architectures of CNN, namely, hybrid CNN, viz., CNN-RNN and CNN-HMM models, and domain-specific recognition systems, can be investigated. Evolutionary algorithms can be explored for optimizing CNN learning parameters, namely, the number of layers, learning rate and kernel sizes of convolutional filters.

References:

[1]. Gaurav Jain, Jason Ko, Handwritten Digits Recognition, Project Report, University of Toronto, 11/21/2008

[2]. A. Dutta and A. Dutta, Handwritten digit recognition using deep learning, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), vol. 6, no. 7, July 2017.

[3]. Al Maadeed, Somaya, and Abdelaali Hassaine, Automatic prediction of age, gender, and nationality in offline handwriting. EURASIP Journal on Image and Video Processing, no. 1 2014.

[4]. M. Wu and Z. Zhang, Handwritten Digit Classification using the MNIST Dataset, 2010.

[5]. Hamid, Norhidayu Abdul, and NilamNur Amir Sjarif,Handwritten recognition using SVM, KNN and neural network, arXiv preprint arXiv:1702.00723 (2017).

[6]. R.G.Mihalyi, Handwritten digit classification using support vector machines, 2011.

[7]. Z. Dan, C. Xu, The Recognition of Handwritten Digits Based on BP Neural Networks and the Implementation on Android, In: 3rd International Conference on Intelligent System Design and Engineering Applications, pp. 1498-1509, 2013.

[8].Hayder M. Albeahdili, Haider A. Alwzwazy, Naz E. Islam.” Robust Convolutional Neural Networks for Image Recognition”. (IJACSA)

International Journal of Advanced Computer Science and Applications, Vol. 6, No. 11, 2015

[9]. Fabien Lauer, Ching Y. Suen, and G´erard Bloch “A trainable feature extractor for handwritten digit recognition‖” , Journal Pattern

Recognition,Elsevier, 40 (6), pp.1816-1824, 2007.

[10]. Kaiming, He and Xiangyu, Zhang and Shaoqing, Ren and Jian Sun “ Spatial pyramid pooling in deep convolutional networks for visual

recognition‖ European” , Conference on Computer Vision, arXiv:1406.4729v4 [cs.CV] 23 Apr 2015.

[11]. Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey. “ImageNet classification with deep convolutional neural networks”. In Advances in

Neural Information Processing Systems 25 (NIPS’2012). 2012.

[12]. Omkar M. Parkhi, Andrea Vedaldi,and Andrew Zisserman “Deep Face Recognition” British Machine Vision Conference, 2015

[13]. Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, Juergen Schmidhuber, “Deep Big Simple Neural Nets Excel on Handwritten Digit

Recognition” Neural Computation, Volume 22, Number 12, December 2010

[14]. Sherif Abdel Azeem, Maha El Meseery, Hany Ahmed ,”Online Arabic Handwritten Digits Recognition “,Frontiers in Handwriting Recognition

(ICFHR), 2012

[15]. Liu, C.L. et al.,.” Handwritten digit recognition: Benchmarking of state-of-the-art techniques”. Pattern Recognition 36, 2271–2285. 2003

[16]. LeCun, Y. et al.,.” Comparison of learning algorithms for handwritten digit recognition”. In: International conference on Artificial Neural

networks, France, pp. 53–60. 1995

[17]. Xiao-Xiao Niu n , Ching Y. Suen “A novel hybrid CNN–SVM classifier for recognizing handwritten digits” Pattern Recognition 45 (2012)

1318–1325, 2011

[18]. M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional neural networks,” arXiv:1311.2901, 2013

[19]. Gil Levi and Tal Hassner,” Age and Gender Classification using Convolutional Neural Networks”, Computer Vision and Pattern Recognition

Workshops (CVPRW) IEEE, 2015