Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

Multi-Digit Number Classification using MNIST and ANN

Priyansh Pandey
Department Of Information Technology
Amity University
Noida, India

Mazhar Khan
Department Of Information Technology
Amity University
Noida, India

Ritu Gupta
Department Of Information Technology
Amity University
Noida, India

Sajid Iqbal
Department Of Information Technology
Amity University
Noida, India

***Abstract*— Modern machines have difficulty in reading the handwritten numbers, as every person's handwriting is different and unique. However, in the modern era where everyone is shifting towards digital technology. Retyping of handwritten ledgers and documents into computer systems will be a hefty process. Thanks to Artificial Intelligence computer vision techniques, that has eased the path. Thus, we have developed an Artificial Neural Network System that can read any type of handwritten number with an accuracy percentage of more than 93. Our Project consists of both research work and practical implementation. The Neural Network tests and learns with the help of MNSIT Data after the successful completion of the code we get a system that can read and process the images with an accuracy of 93% which is also supported by the graph. We have stored this system in a file and now we can use this system and with the help of some libraries like opencv, etc. We can read and save the processed output in a CSV file in just few clicks and get the data in real-time. The project is in its earlier stages. Once finalized it can help in saving a lot of Human Efforts.**

Keywords*— *Artificial Intelligence, Artificial Neural Networks, Handwritten Number, Machine Learning, Deep Learning, MNIST Dataset,

INTRODUCTION

Handwritten digit recognition is an important problem in optical character recognition, and it can be used as a test case for theories of pattern recognition and machine learning algorithms. The handwritten digits are preprocessed, including segmentation and normalization, so that researchers can compare recognition results of their techniques on a common basis as well as reduce the workload [1].

The Modern machines have difficulty in reading the handwritten numbers, as every person's handwriting is different and unique. However, in the modern era where everyone is shifting towards digital technology. Retyping of handwritten ledgers and documents into computer systems will be a hefty process. Thanks to Artificial Intelligence computer vision techniques, that has eased the path. Thus, we have developed an Artificial Neural Network System that can read any type of handwritten number with an accuracy of more than 99.5%. This system will help teachers upload marks of students in the online portal with ease. Moreover, we can implement this system in the finance and accounts departments of both government and private sectors for uploading number related queries. That will also help reduce human error while uploading number-based data. Our motive is to reduce human error, effort and time.

Not only is this, as the main method of teaching in school and colleges still pen paper-based there are lakhs of hard copies that need to digitalized every year. And that only from school and colleges think about the other sectors such as aadhar generation. There are Billions of handwritten data that needed to be inserted into the computer.

Conversion of hardcopy to softcopy in convention way take a lot of human effort and time. Our objective is to predict the handwritten images with maximum accuracy using Artificial Neural Networks.

I. LITERATURE REVIEW

In Handwritten number classification, different methods are used like methods based on low-level image feature representation which consider the handwritten number image as a group of low-level characteristics like texture, shape, size, color, etc. and methods based on mid-level visual feature constructions for image classification tasks. Nowadays, the usage of deep neural networks and neural-networks to get an image representation is trending. Such architectures allow us to extract features from a specific layer of trained neural network and so use extract feature maps as a numeric image representation. There are a large number of publications related to the image processing with neural network. Our work is related to this type of research, where ANN is used for classifying images. Image classification in the fashion domain has numerous benefits and applications and has various research works have been published about it. [2].

Handwriting recognition is one of the most favorite topics among researchers because every individual in this world has their style of writing. The computer can identify and understand handwritten digits but It is very difficult to make the computer understand handwritten work as each person's

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

handwriting is different and unique. The main motive of our system is to reduce the human effort.

Data Entry workers spend hundreds of their work hours just to type the handwritten data into the computer. It is a very time-consuming job as well it takes lots of precision and high-speed typing as there are lakhs of records needed to be entered.
Different Organizations spend a huge amount of money for converting their document from one format to another.[4]

Deep learning and machine learning plays an important role in computer technology and artificial intelligence. With the use of deep learning and machine learning, human effort can be reduced in recognizing, learning, predictions and many more areas. Thus we created an Artificial Neural Network System that can automatically convert handwritten images to Digital Format with and accuracy of more than 93%

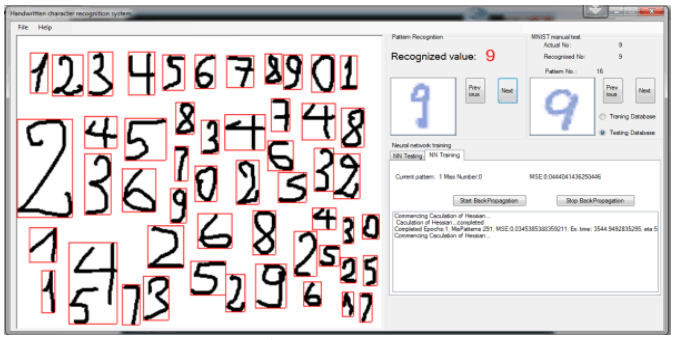


Figure 1: Examples of Handwritten Images to Dgital Format

II. PROPOSED SOLUTION

II.1 IMAGE SEGMENTER

One of the most important feature of our program is Image Segmenter. We have inserted an Image segmenter in our program to give it a benefit over other MNSIT programs that is it give a more practical look to over program. Since MNSIT dataset contains only single digits numbers from 0 to 9 so it will not be able to process multi digit numbers. So to process multidigit numbers we have impleted an image segmenter program. Since, the identification of a single segmented digit is an easier task as compared to identification and segmentation of a multi digit sting. It is also identified as a Hello World program example of machine learning.
We will give you a brief summary of how our image segmenter works-
The identification process of handwritten number passes through 3 steps preprocessing, segmentation of image into single digits, and identification of each digits.

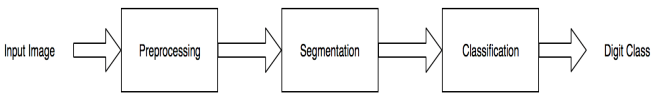


Figure 2: Multi- Digit Identification Diagram

The preprocessing step involves conversion of image to grayscale, binarization and dilation.

Figure 3: Preprocessing Steps

Preprocessing Steps

1. Load Image and converting it into Grayscale – We assigned each pixel a value from the range of monochromatic shades from white to black to represents its amount of lights in the image.

```
# Load image in grayscale
    original = gray_image.copy()
```

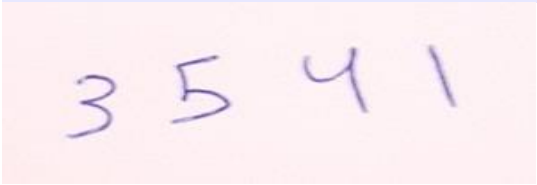


Figure 4: Sample Image

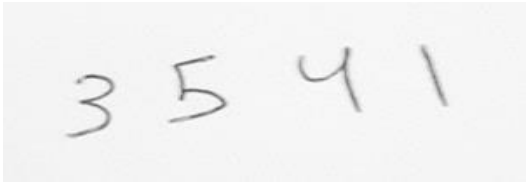


Figure 5: Grayscale Image

2. Binarize the image – Assigning each pixel to only 2 colors possible either black or white.

```
# Binarize the image
bin_image = cv2.threshold(gray_image,200,255,cv2.THRESH_TRUNC)[1]
#    bin_image = cv2.threshold(gray_image,180,255,cv2.THRESH_BINARY)[1]
#bin_image = cv2.adaptiveThreshold(image,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY,11,2)
```

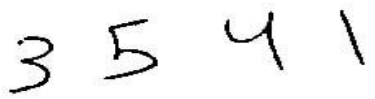


Figure 6: Binarized Image

3. Invert and Dilate image – We assigned each output pixel to the max value of all the pixel in inputted pixels neighborhoods. This results in overall increase of the thickness of the digits.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

```
# invert image
inverted_image = cv2.bitwise_not(bin_image)

# Dilate image
kernel = np.ones((1,1), np.uint8)  # Taking a matrix of size 1 as the kernel
img_dilation = cv2.dilate(inverted_image, kernel, iterations=1)

# detect edge
edged=cv2.Canny(img_dilation,30,200)

# Dilate it
kernel = np.ones((3,3), np.uint8)  # Taking a matrix of size 1 as the kernel
edged = cv2.dilate(edged, kernel, iterations=1)
```

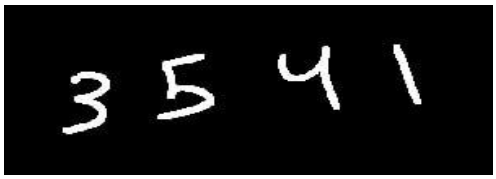


Figure 7: Dilated Image

4. Segment Multi digits number into individual Digits and storing it.

```
# Sort contours as per their x-coordinate
cnts = sorted(cnts, key= cv2.boundingRect)

# store segmented images
image_list = []
for c in cnts:
    x,y,w,h = cv2.boundingRect(c)
    ROI = 255 - original[y:y+h, x:x+w]
    image_list.append(ROI)
    cv2.rectangle(gray_image, (x, y), (x + w, y + h), (36,255,12), 2)
```

Figure 8: Segmented Image

5. Sort Digits and adding border to every number, this results in increase in the accuracy of classification.

```
# Modify segmented images
resized_image_list = []
border_fraction = 0.05  # 10%
for img in image_list:
    # add padding of black border
    img = cv2.copyMakeBorder(
                img,
                int(np.ceil(img.shape[1]*border_fraction)),
                int(np.ceil(img.shape[1]*border_fraction)),
                int(np.ceil(img.shape[0]*border_fraction)),
                int(np.ceil(img.shape[0]*border_fraction)),
                cv2.BORDER_CONSTANT,
                value=0)
```

Before adding border

After adding border

6. Forwarding it to ANN so the numbers can be read and recognized by the system.

```
    # Resize the image
    img = cv2.resize(img, (28, 28), interpolation = cv2.INTER_AREA)

    # Binarize it
    img = cv2.threshold(img,50,255,cv2.THRESH_BINARY)[1]

    # save it in a list
    resized_image_list.append(img)

return np.asarray(resized_image_list)
```

II.2 NEURAL NETWORK

The ANN for the digit dataset is developed into 3 layers. The first layer is the input layer then we have a hidden layer followed by an output layer. It is a fully connected network. The ANN is depicted in Fig 9.

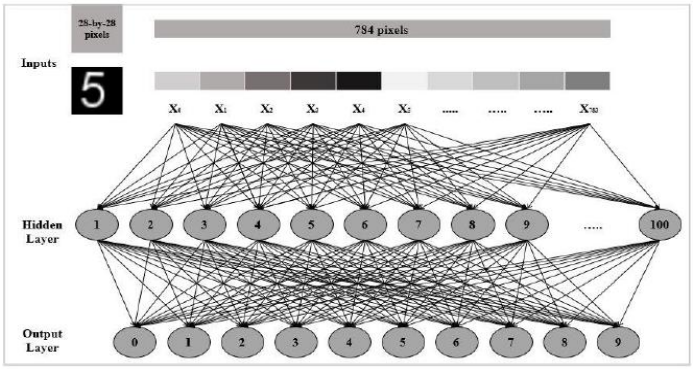


Figure 9: Artificial Neural Network

The MNIST dataset images have 28*28 pixels size that are flattened to 784 tensor array.

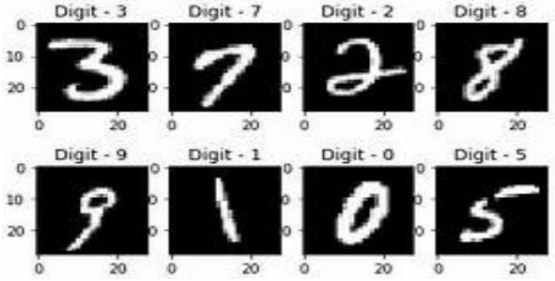


Figure 10: Example of MNIST Image

Each of these array values are assigned to a single neuron in the input layer. The input layer is fully connected to the hidden layer. The initial weights are initialized using random normal distribution. The weight values are between 0 and 1. The neural network also has bias weight added to both the layers. The bias weight is shown as X0 in the Fig. 10 for the input layer.

The input layer has then 784 neurons which is then fully connected to a hidden layer consisting of 75 neurons. The weight matrix representation for the input layer are size 20X784. The number 20 is the batch size. The weights are not updated after every training example but after 20 training examples. These 20 examples are called as batch. This way

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

3000 batches are generated out of our 60,000 training examples.
The input is then propagated to a hidden layer which consists of 75 hidden neurons. There is no thumb rule to select the number of hidden neurons. It varies from problem to problem. After several choices, this 75 neuron is the best selection for the hidden layer. The hidden layer then has a matrix equivalent of size 784X75. The hidden layer also has a bias weight depicted by h0 in Fig. 2. Using simple matrix multiplication of two matrices of size 20X784 and 784X75 an output matrix of 20X75 will be generated. An activation function is applied to all the elements of the output matrix. Here, the sigmoid function is used which has a range of -1 to 1 for the input values.[20]
The hidden layer is then connected to the output layer which has 10 neurons which is depicted in Fig.2 as 0 to 9. These are the class labels or the actual numbers. The output layer has a weight matrix of 75X10 neurons. Using matrix multiplication of size 20X75 and 75X10 an output of 20X10 is generated that is for 20 training example a probability for all these numbers is generated. The Soft Max activation function is then used to get these outputs in the range of 0 to 1 and the total sum of 1. The highest probability is selected for a training example as 1 and rest other less probabilities are converted to 0. Instead of getting an actual number value the output labels of our training examples are also converted to 0 and 1 values, one-hot encoding. [6]
After getting the predicted result for the first batch of 20 training examples. The error i.e. difference between actual and predicted output is calculated using the mean squared error loss function. The error is then back propagated through the network to adjust the weights. Various learning rates are chosen to see which best suits the model and ANN doesn't stick to the local minima or overshoot the global minimum. After training the neural network with all of our training examples and testing it on the testing set. The whole ANN is trained multiple times bypassing the whole dataset again and again a total of 350 times called several epochs. With every epoch, the accuracy of the neural network should increase and loss should do down until the ANN hits a plateau. The predicted outputs are then converted back to numeric digits for better human readability.

II.3 Training Model

We have added a special feature in our program that can help in reduce processing time to a great extent. We all know that training a systems takes a lot of time and lots of resources including processing power of the system. For an example to process an image using MNSIT will take approx. 10 to 15 minutes for a normal system to provide the result. To overcome this error we have put a different approach of saving the resulted code into a pickle file. A pickle file is a file that is used to serializing and de serializing objects in python. Any python object can be converted into pickle file. The basic idea behind it was that the pickle file contains all the necessary information required to reconstruct the same object in another python program. Which in our case our trained system to recognize the digits.
This is the following line of code where the program ask for user input if they want to train the model or not.

```
train_model = True
```

If the line is set to True. The program will run from beginning and will re train the model that is process all the MNIST images again and re learn the whole process and achieving the accuracy and then it will save the object into the pickle file which can be used for future process.

```
train_model = False
```

If the line is set to False. The program will not re run from beginning. It will just use the last saved pickle and recognize the digits on the basis of that file.

It is recommended that we should train the model once before starting every cycle. This helps in getting proper results.

II.4 Processed Output

In this part, we are going to know about what happens after the result is declared i.e. when the final image is processed by the system. The program then asks for user input if they want to save the output in the csv file or not. IF the user gives permission then The result gets saved in the excel file with its name and value. The image is now saved in the excel file so we can open it whenever we want to and we can make changes in it according to our needs. It is helpful for our future references if we need any we can easily go to the saved file and take a look on whatever we need to find through that. It's less time taking and efficient. And at the end of the day, all we need is more efficient and less time consuming features. It goes without saying that it keeps the files organized as the file gets saved with its name and value by default. The file which gets saved observed in the output as in one column there is file path gets saved and in the other column the value of the file is saved. This feature is more organized and convenient to look and understand. But as this feature is not a compulsion, it is an added feature so there is no hard and fast to use this feature. It's all up to us to use it or not.

III. TOOLS REQUIRED

A. MNIST DATASET

The MNIST database (Modified National Institute of Standards and Technology database) is a huge database of handwritten digits that is usually utilized for preparing different picture handling frameworks. The database is additionally broadly utilized for preparing and testing in the field of AI. It was made by "re-blending" the examples from NIST's unique datasets. The makers felt that since NIST's preparation dataset was taken from American Census Bureau representatives, while the testing dataset was taken from American secondary school understudies, it was not appropriate for AI tests. Moreover, the high contrast pictures from NIST were standardized to fit into a 28x28 pixel jumping box and hostile to associated, which presented grayscale levels.
The MNIST database in total has 70,000 images out of which 60,000 are preparing pictures and 10,000 testing pictures.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

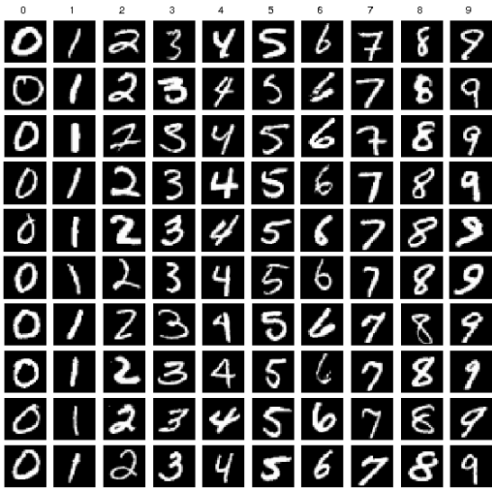


Figure 11: Examples of Image set

B. JUPYTER NOTEBOOK

It is an open-source application that allows writing and sharing files that contain working code, equations, visualizations, and narrative text. Some of its uses are - data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, etc.

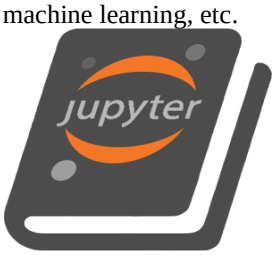


Figure 12: Jupyter Notebook Setup for Web application

C. ANACONDA PYTHON

Anaconda is a free and open-source conveyance of the Python and R programming dialects for logical registering (information science, AI applications, enormous scale information preparing, prescient investigation, and so on.), that plans to disentangle bundle the board and arrangement. Bundle-adaptations are overseen by the bundle executive framework conda. The Anaconda circulation is utilized by more than 15 million clients and incorporates more than 1500 prevalent information science bundles appropriate for Windows, Linux, and Mac OS.

Figure 13: Example of Anaconda Python using for framework

D. NUMPY LIBRARY

NumPy is a library for the Python programming language, including support for enormous, multi-dimensional clusters and matrices, along with a huge assortment of high-level numerical capacities to work on these exhibits. The progenitor of NumPy, Numeric, was initially made by Jim Hugunin with commitments from a few different engineers. In 2005, Travis Oliphant made NumPy by consolidating highlights of the contending Numarray into Numeric, with broad adjustments. NumPy is open-source programming and has numerous patrons.

Figure 14: Numpy is basically used for Python Programming languages

E. ANN

Artificial neural networks (ANNs) are factual learning calculations that are reused by properties of the natural neural networks. They are utilized for a wide assortment of errands, from moderately straightforward arrangement issues to discourse acknowledgment and PC vision.

ANNs are approximately founded on organic neural networks it might be said that they are actualized as an arrangement of interconnected handling components, now and again called hubs, which are practically comparable to natural neurons. The associations between various hubs have numerical qualities, called loads, and by modifying these qualities in an orderly way, the system is, in the end, ready to rough the ideal capacity.

Every hub in the system takes numerous contributions from different hubs and computes a solitary yield dependent on the information sources and the association loads. This yield is, for the most part, bolstered into another neuron, rehashing the procedure. At the point when outfitted with the data given in the last sentence, one can without much of a stretch imagine the inner various leveled structure of the artificial neural system, where neurons are sorted out into various layers, as delineated underneath. The info layer gets the information sources and the yield layer creates a yield. The layers that lye in the middle of these two are called shrouded layers.

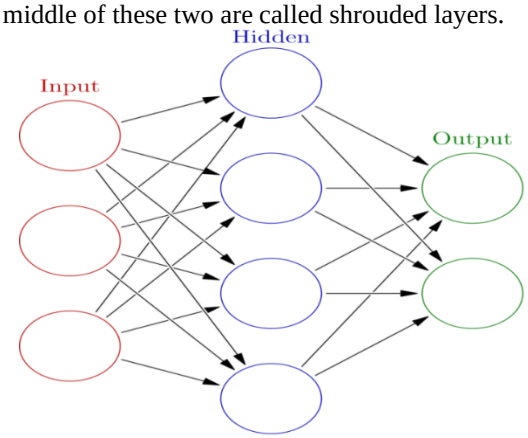


Figure 15: Examples of Input, Hidden and Output Layers with Nodes

F. ARTIFICIAL INTELLIGENCE

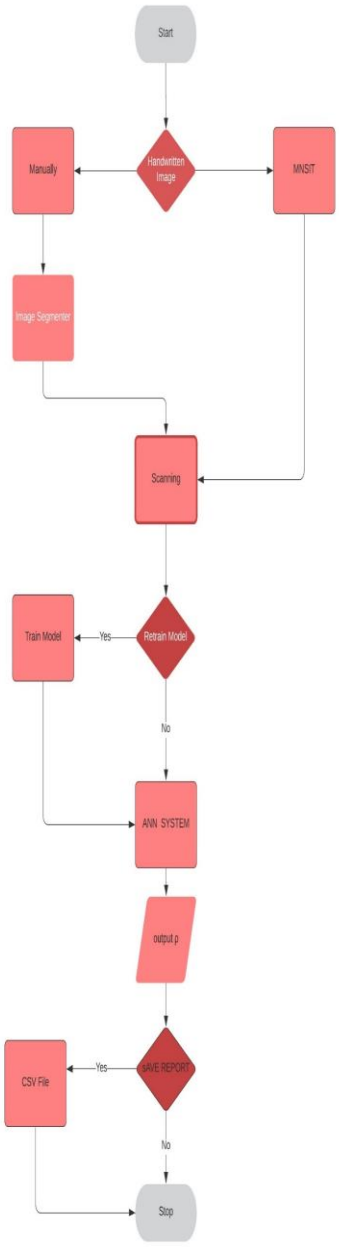
Artificial Intelligence, often referred to as Machine Intelligence, is best defined as the simulated human intelligence by mainly computers but in general machines. The machines take decisions as outputs. These decisions are based

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

upon a set of defined rule and the machines render approximately correct or definite conclusions. These rules are also used for corrections in decision.

IV. FLOWCHART DIAGRAM

As you can see in the above flowchart, first of all we start the program and observe that whether the handwritten image is manual or MNSIT. If the image happens to be manual then we place an image segmenter into it and then scan the image. And if the image comes out to be MNSIT then we directly scan the image. After scanning the image, a retrained model has been received if we wish to train the model, we go through the train model stage and if not then directly jump on the ANN system, which further takes us to the output p and finalize the step by saving the report and then go to the CSV file and stop the program. And if we do not wish to save the report we can stop the program straight.



V. CONCLUSIONS

The optical character recognition is quite an interesting field for machine learning researchers. The goal is to get as much accuracy as possible. An attempt to solve the similar problem of digit recognition is done using the ANN.

The ANN built has 3 layers the input layer with 784 neurons fully connected to a hidden layer of 75 neurons which is further fully connected to the output layer with 10 neurons which classify whether the inputted image is a number between 0 and 9

```
self.w1 = np.random.randn(input_size, no_hidden_neurons)
self.w2 = np.random.randn(no_hidden_neurons, output_size)
self.b1 = np.zeros([1, no_hidden_neurons])
self.b2 = np.zeros([1, output_size])
self.learning_rate = 0.002
```

Figure 16: Examples of ANN Initialization

The Fig. 16 shows the initialization of weights and learning rate. Then the training set is passed in a forward direction in the ANN

```
z1 = np.dot(x, self.w1) + self.b1
self.a1 = self.sigmoid_activation(z1)
z2 = np.dot(self.a1, self.w2) + self.b2
self.a2 = self.softmax_activation(z2)
return self.a2
```

Figure 17: Examples of ANN Forward Pass

as shown in Fig. 17 which is called the forward pass of the ANN. The ANN will generate some output for the training set which is then compared to actual output and the difference is calculated using mean squared error.

```
output_error = y_actual - y_pred
output_delta = output_error * self.softmax_prime(y_pred)

hidden_error = output_delta.dot(self.w2.T)
hidden_delta = hidden_error * self.sigmoid_prime(self.a1)

self.w1 += self.learning_rate * x.T.dot(hidden_delta)
self.w2 += self.learning_rate * self.a1.T.dot(output_delta)

self.b1 += self.learning_rate * np.sum(hidden_delta, axis=0)
self.b2 += self.learning_rate * np.sum(output_delta, axis=0)
```

Figure 18: Examples of ANN Backward Propagation

The error is back propagated as shown in Fig. 18 through the ANN so that the weights can be optimized.

To summarize, The ANN is trained on the MNIST dataset for 350 number of epochs on a batch size of 20 with 60,000 training examples and 10,000 test examples. After various hyper parameters values experiment. The best value to choose for learning rate is 0.002 and the number of hidden neurons is 75. More epochs cannot be taken as the accuracy has reached the plateau values i.e. no more increase in the performance of the model even if the model is trained again. The first epoch accuracy of the model is 33.14% which kept on increasing

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

with the number of epochs until the number of epoch where it hit an accuracy of 88% after that the accuracy started to increase very slowly and after 350 epochs the model hits an accuracy of 92%.
When the model is tested on the training set it shows an accuracy of 95%.

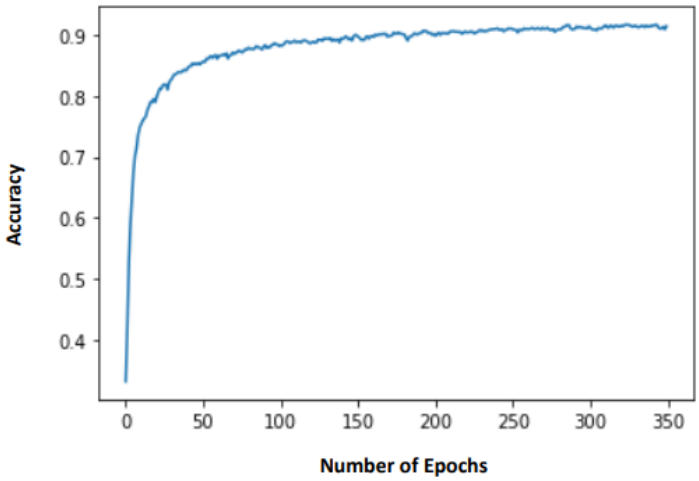


Figure 19: Examples of MNIST Simulation result

The MNIST website also provides the best model for their dataset which suggests that using a 2 layered neural network with 800 hidden neurons and cross-entropy can lead to an accuracy of 93%.

ACKNOWLEDGMENT

The Authors would like to thank Dr. A. Sai Sabitha, HOD, IT, Amity University for giving us the opportunity to undertake this project. We would like to thank our faculty guide Ms. Ritu Gupta who is the biggest driving force behind our successful completion of the project. She has been always there to solve any query and also guide us in the right direction regarding the project. Without her help and inspiration, We would not have been able to complete the project. Also the authors would like to thank Mr Deepansh Pandey, University Bonn,Germany who guided us, helped us and gave ideas and motivation at each step.

REFERENCES

1. Deng, L., 2012. The MNIST database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Processing Magazine, 29(6), pp.141-142.
2. Al-Mansoori, Saeed. (2015). Intelligent Handwritten Digit Recognition using Artificial Neural Network. 10.13140/RG.2.1.2466.0649
3. Ashworth, Q. Vuong, B. Rossion, M. Tarr, Q. Vuong, M. Tarr, et al., "Object Recognition in Man, Monkey, and Machine," Visual Cognition, vol. 5, pp. 365-366, 2017.
4. J. Janai, F. Güney, A. Behl, and A. Geiger, "Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art," arXiv preprint arXiv:1704.05519, 2017.
5. Wu, Ming & Zhang, Zhen. (2019). Handwritten Digit Classification using the MNIST Data Set
6. D. Arpit, Y. Zhou, B. Kota, and V. Govindaraju, "Normalization propagation: A parametric technique for removing internal covariate shift in deep networks," in International Conference on Machine Learning, 2016, pp. 1168-1176.
7. I. Patel, V. Jagtap, and O. Kale, "A Survey on Feature Extraction Methods for Handwritten Digits Recognition," International Journal of Computer Applications, vol. 107, 2014.
8. D. Arpit, Y. Zhou, B. Kota, and V. Govindaraju, "Normalization propagation: A parametric technique for removing internal covariate shift in deep networks," in International Conference on Machine Learning, 2016, pp. 1168-1176.
9. I. Patel, V. Jagtap, and O. Kale, "A Survey on Feature Extraction Methods for Handwritten Digits Recognition," International Journal of Computer Applications, vol. 107, 2014.
10. K. T. Islam, R. G. Raj, and G. Mujtaba, "Recognition of Traffic Sign Based on Bag-of-Words and Artificial Neural Network," Symmetry, vol. 9, p. 138, 2017.
11. P. M. Kamble and R. S. Hegadi, "Handwritten Marathi character recognition using R-HOG Feature," Procedia Computer Science, vol. 45, pp. 266-274, 2015.
12. J. Varagul and T. Ito, "Simulation of Detecting Function object for AGV Using Computer Vision with Neural Network," Procedia Computer Science, vol. 96, pp. 159-168, 2016.
13. A. Boukharouba and A. Bennia, "Novel feature extraction technique for the recognition of handwritten digits," Applied Computing and Informatics.
14. S. Abdleazeem and E. El-Sherif, “Arabic handwritten digit recognition,” International Journal on Document Analysis and Recognition, vol. 11, no. 3, pp. 127–141, 2008.
15. R. Ebrahimzadeh and M. Jampour, “Efficient handwritten digit recognition based on Histogram of oriented gradients and SVM,” International Journal of Computer Applications, vol. 104, no. 9, pp. 10–13, 2014.
16. A. M. Gil, C. F. F. C. Filho, and M. G. F. Costa, “Handwritten digit recognition using SVM binary classifiers and unbalanced decision trees,” in Image Analysis and Recognition, vol. 8814 of Lecture Notes in Computer Science, pp. 246–255, Springer, Basel, Switzerland, 2014.
17. B. El Qacimy, M. A. Kerroum, and A. Hammouch, “Feature extraction based on DCT for handwritten digit recognition,” International Journal of Computer Science Issues, vol. 11, no. 6(2), pp. 27–33, 2014.
18. S. AL-Mansoori, “Intelligent handwritten digit recognition using artificial neural network,” International Journal of Engineering Research and Applications, vol. 5, no. 5, pp. 46–51, 2015.
19. R. C. Gonzalez and R. E. Woods, Digital Image Processing, vol. I, Prentice-Hall, New Delhi, India, 1992.
20. F.Hausdorff,“Summationsmethoden und omentfolgen. I,” Mathematische Zeitschrift, vol. 9, no. 1-2, pp. 74–109, 1921.
21. M.-K. Hu, “Visual pattern recognition by moment invariants,” IRE Transactions on Information Theory, vol. 8, no. 2, pp. 179–187, 1962.
22. M. Petrou and A. Kadyrov, “Affine invariant features from the trace transform,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 1, pp. 30–44, 2004.
23. P.-T. Yap and R. Paramesran, “An efficient method for the computation of Legendre moments,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 12, pp. 1996–2002, 2005.
24. S. X. Liao and M. Pawlak, “On image analysis by moments,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18, no. 3, pp. 254–266, 1996.
25. A. Khotanzad and Y. H. Hong, “Invariant image recognition by Zernike moments,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, no. 5, pp. 489–497, 1990.