IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

NCAIT 2017

JSS Academy of Technical Education

Vol. 4, Special Issue 8, May 2017

Digit Recognizer

Anushri Ravikumar[1], Ashitha Nayak[2], Mamatha[3]

Information Science and Engineering, BMS College of Engineering, Bangalore, India[12]

Assistant Professor, Information Science and Engineering, BMS College of Engineering, Bangalore, India[3]

Abstract: The human visual system is one of the wonders of the world. The difficulty of visual pattern recognition becomes apparent if you attempt to write a computer program to recognize digits. One of the approaches to computers behaving and computing like humans involve neural networks. The neural network automatically infers rules for recognizing handwritten digits. This approach reduces human intervention in many commercial places like banks to process cheques and by post offices to recognize addresses. The neural network attempts to determine if the input data matches a pattern that the neural network has memorized. The concepts of weights and biases are conventionally used to enhance the performance of the Neural networks over a large set of test images and classify the digit into a class label. The data is stored in a very simple file format designed for storing vectors and multidimensional matrices. Tensor Flow is used to train an elaborate model that achieves state-of-the-art performance along with Softmax Regression. The current program can recognize digits with an accuracy over 96 percent, without human intervention, classifying 9,979 of 10,000 images correctly. The performance is close to human-equivalent, and is arguably better, since quite a few of the MNIST images are difficult even for humans to recognize with confidence.

Keywords: Neural network; TensorFlow; Softwax Regression; Pattern recognition.

INTRODUCTION

Pattern recognition is one of the most common use of neural networks. The neural network is provided with a target vector for identification and another vector which contains the pattern information. This could be an image or hand written data. The model will attempt to memorise patterns during training and then try to determine if the input data matches them. When a neural network is being designed for such a purpose, it must be trained to classify the input samples into groups. These groups may not have clearly defined boundaries and therefore be fuzzy. With the large number of the hand-written documents, there is a great demand to convert the handwritten documents into digital record copies, which are more accessible through digital systems, such as digital forms and databases. To accomplish the transformation of handwritten numbers to their respective digital version automatically, multi-digit recognition techniques are extremely useful and necessary.

The purpose of this paper is to take handwritten digit characters as input, process the character, train the neural network algorithm, to recognize the pattern and modify the character to a beautified version of the input.

Objectives:

- To design a neural network using a mathematical model for the required outcome and for the process of normalization and enhancing efficiency.
- To adopt multiple layers while designing the computational model of CNN for digit classification.
- This paper is aimed at development of software that can be used in applications which require recognition of digits. This paper is restricted to English digit only. It can be developed further to recognize the digits of different languages.
- It aims to engulf the concepts of machine learning, neural networks and data mining to meet the required outcome.
- This paper presents a convolutional neural network for digit recognition. The system works for images uploaded with handwritten digits in them. It is limited to grayscale images of size 784 pixels, but it can also be extended to nearly double the size with suitable changes.
- Currently the system recognizes one digit at a time. It can be upgraded to recognize multiple digits in the future.

LITERATURE SURVEY

Works in character recognition

Research attempts began in the area of character recognition as early as 1959 by Grimsdale. An approach known as analysis-by-synthesis originated in the early sixties on which a great amount of research was based. This method was suggested by Eden in 1968. Eden's work proved that a finite number of schematic features form all handwritten characters. This point made it highly important for further works. It was later used in all methods in syntactic approaches of character recognition.

Analytical Review of Pre-processing Techniques [5] is another source of information for various techniques that can be used to pre-process images. It concerns different kind of images ranging from a simple handwritten form based documents to those containing coloured and complex backgrounds which have different intensities.

IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

NCAIT 2017

JSS Academy of Technical Education

Vol. 4, Special Issue 8, May 2017

For recognizing unconstrained texts that are handwritten, a proposed model was the Hybrid Hidden Markov Model [6]. In this model, the structural parts of text are modeled using Markov chains. A multilayer perceptron is used to estimate the emission probabilities. The slope and slant from handwritten text are removed using various techniques. The size of text images are normalized with supervised learning methods. The key features of this recognition system were to develop a system having high accuracy in pre-processing and recognition, which are both based on ANNs.

A modified quadratic classifier is the Hand Written Numeral Recognition of six scripts [7]. It is based on a scheme to recognize the handwritten numerals of six popular Indian scripts. This paper has been a great success to adopt neural network for highly used local languages in India. The project uses this paper as an inspiration and reference for the work done.

Multilayer perceptron [8] is a feed forward ANN that has been used for recognizing English characters that are handwritten. Boundary tracing is for extraction of features and their Fourier Descriptors. The shape of the character is analyzed and the features that distinguish it are compared for recognition.

The paper on Diagonal based feature extraction [9], proposes a diagonal feature extraction for offline character recognition. This gives a high recognition accuracy of 97.8 % for 54 features and 98.5% for 69 features.

Offline Handwriting Recognition [10] describes a system for recognition of cursive handwriting. It is based on Hidden Markov Models using discrete and hybrid modeling techniques.

Devanagiri Character Recognition [12], used four feature extraction techniques namely, intersection, shadow feature, chain code histogram and straight line fitting features. For each character image, there is a global computation of shadow features. The character image is divided into different segments and computed for chain code histogram features, intersection features and line fitting features. The overall recognition rate for a data set of 4900 units was observed to be 92.80% on experimentation for Devanagari characters.

Recognizing Arabic characters [13] et al, describes that recognition of characters greatly depends upon the features used. Many features of the Arabic characters that are handwritten are discussed. An off-line recognition system based on the selected features was built. Realistic samples of handwritten Arabic characters are used for training and testing. The importance and accuracy of the selected features is evaluated. Based on selected features, average accuracies of 88% and 70% are obtained for the numbers and letters, respectively.

DESIGN

System Design

The following figure (Figure 1) provides a high level design of the system and the association between various modules used.

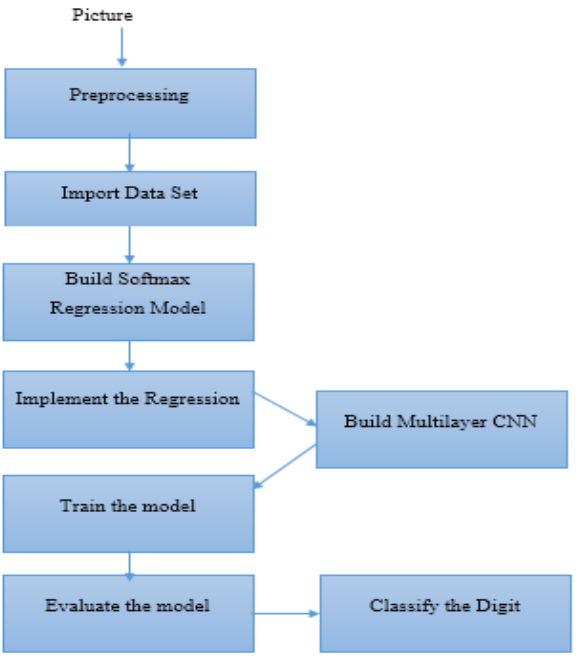

Figure 1 High level Design

Data flow diagrams

The DFD's have been shown below.

DFD Level-0

The DFD Level-0 consists of two external entities, the UI and the Output, along with a process block 1.0, representing the CNN for Digit Recognition as shown (figure 2). Output is obtained after processing.

Figure 2 DFD level-0

DFD Level-1

The DFD Level-1 (figure 3) consists of 2 external entities, the GUI and the Output, along with five process blocks and 2 data stores MNIST data and Input image store, representing the internal workings of the CNN for Digit Recognition System. Process block 1.3 imports MNIST data from data store online. Process block 1.1 imports the image and processes it and sends it to block 1.2 where regression model is built. It sends objects with probabilities to CNN where weights are updated and multiple layers are built. Block 1.5 trains and evaluates the model to generate output.

IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

NCAIT 2017

JSS Academy of Technical Education

Vol. 4, Special Issue 8, May 2017

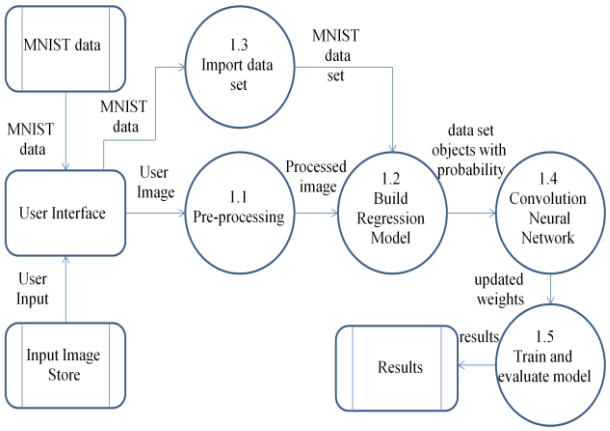

Figure 3 DFD level-1

DFD Level-2

The DFD Level-2 for import data (figure 4) consists of two external data store and one entity UI along with three process blocks, representing the three functionalities of the CNN for Digit Recognition System. It imports data from MNIST data store and stores on the System.

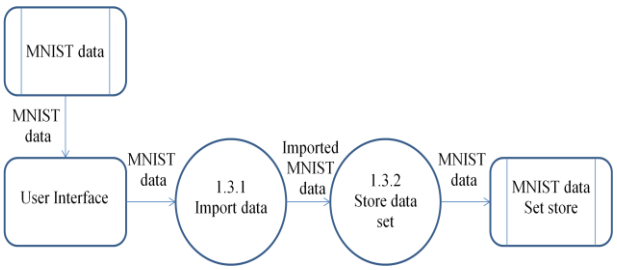

Figure 4 DFD level-2

DFD Level-2(regression model)

The DFD Level-2 for regression model (figure 5) consist of a MNIST data store and 4 process block. Block 1.2.1 creates placeholders for inputs. Variables are created and initialized (block 1.2.2) weights are assigned to variables (block 1.2.3) and regression model is built (block 1.2.4).

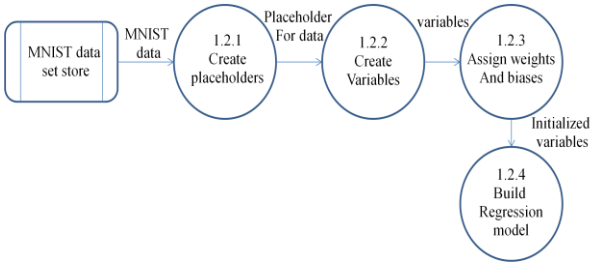

Figure 5 DFD level-2(regression model)

DFD Level-2(Build CNN)

The DFD Level-2 for build CNN (figure 6) consists of an external entity Regression Model along with 7 process blocks, weights are initialized (block 1.4.1) and passed to convolution layer and pooling layers (block 1.4.2) Then layers of CNN are built (blocks 1.4.3,1.4.4,1.4.5) and dropout (block 1.4.6) and then final readout is obtained (block 1.4.7).

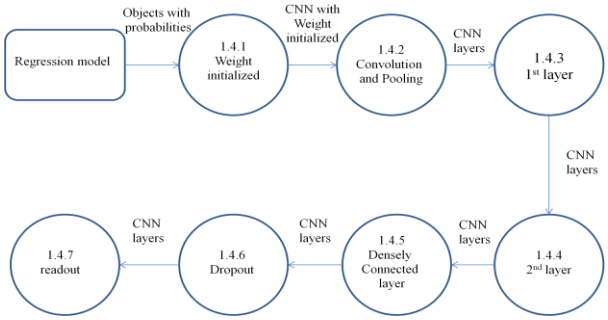

Figure 6 DFD level-2 (Build CNN)

DFD Level-2(Train and Evaluate model)

The DFD Level-2 for (Train and Evaluate Model) (figure 7) consists of an external entity CNN along with 4 process blocks, cross entropy (block 1.5.1) and optimization (block 1.5.2) is done on the results obtained from CNN block. Labels are predicted using aggregation function (block 1.5.3) and accuracy is determined (block 1.5.4) Results are stored in results data store.

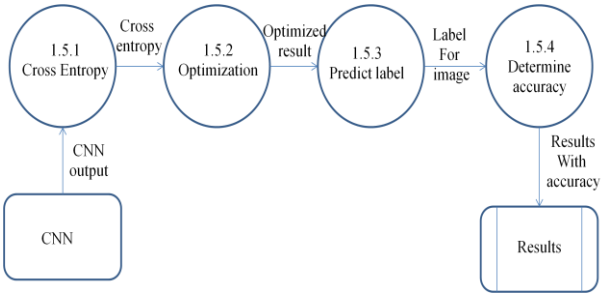

Figure 7 DFD level-2 (Train and Evaluate Model)

Detailed Design of the functional Modules

Functional description of each module is explained:

Pre-processing

Pre-processing may be required for pre-captured photo sequences. The dimensions are matched and made sure that each image contains only one digit. We use MNIST data sets for training, recognition of handwritten digits. MNIST is a simple computer vision dataset. It (figure 8) consists of images like these:

Figure 8 Data set example images

The MNIST Data

The data set being used is hosted on Yann LeCun's website. It can be imported using the following code:

fromtensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

NCAIT 2017

JSS Academy of Technical Education

Vol. 4, Special Issue 8, May 2017

The downloaded data is split into three parts, 55,000 data points of training data (mnist.train), 10,000 points of test data (mnist.test), and 5,000 points of validation data (mnist.validation). Every MNIST data point has two parts: an image of a handwritten digit and a corresponding label. We will call the images "xs" and the labels "ys". All divisions of the data set must contain these labels.Each image is 28 pixels by 28 pixels. This is interpreted as a big array of numbers (figure 9):

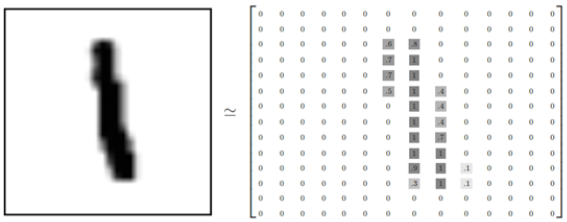

Figure 9 Matrix representation of pixels

This array can be flattened into a vector of size 28 pixels * 28 pixels = 784. The method or procedure used for flattening does not have any particular constraints other than that it must be used consistently for images. In this way, the images in the data set are visualized as a set of points in a 784-dimensional vector space, a very rich structure (computationally intensive visualizations). The result of this process is that the data set now becomes a structure known as a tensor. It will now have a shape of [55000, 784]. The x-coordinate is an index of the images in the data set and the y-coordinate is an index of the each pixel in an image. Each entered value in the tensor is the pixel intensity of that particular index(figure 10).

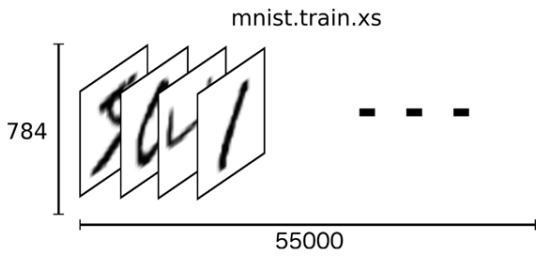

Figure 10 MNIST training data as a tensor

The labels for identifying images in the data set are numbers between 0 and 9.They are used to describe the digit that the image represents. For the purposes of this paper, we're going to encode our labels as "one-hot vectors". A one-hot vector is a vector which is 1 at a single index and 0 in all others. In this case, the nth digit will be represented as a vector which is 1 in the nth dimensions.

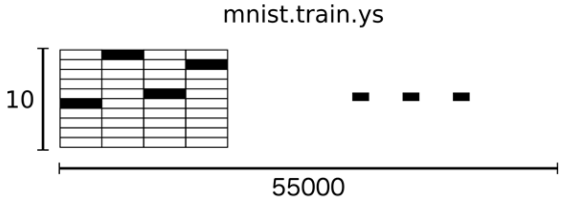

Figure 11 Encoding of lables

For example, 3 would be [0,0,0,1,0,0,0,0,0,0]. Consequently, mnist.train.labels is a [55000, 10] array of floats (figure 11).

Softmax regressions

We know that every image in the data set is a digit between 0 to 9.We want to be able to look at an image and give probabilities for it being each digit. For example, our model might look at a picture of a nine and be 80% sure it's a nine, but give a 5% chance to it being an eight (because of the top loop) and a bit of probability to all the others because it isn't sure.

This is a classic case where a softmax regression is a natural, simple model, to assign probabilities to an object being one of several different things. The best method is to use softmax. Even later on, when we train more sophisticated models, the final step will be a layer of softmax.

A softmax regression has two steps: we first need to find the evidence of our input being in certain classes to add up, and then we convert that evidence into probabilities. To tally up the evidence, that a given image is in a particular class, we do a weighted sum of the pixel intensities. The weight is negative if that pixel having a high intensity is evidence against the image being in that class, and positive if it is evidence in favor.

The following diagram (figure 12) shows the weights one model learned for each of these classes. The negative weights and positive weights are represented by red and blue respectively.

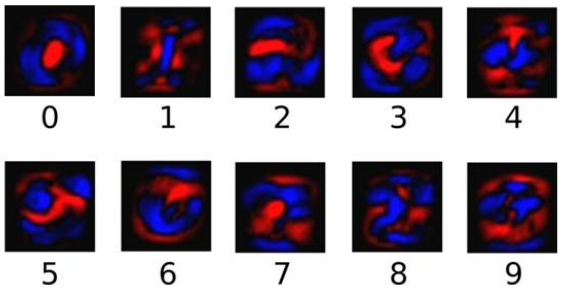

Figure 12 Weights for model

The model also adds some extra evidence called a bias. This is to consider factors that are more likely independent of the input. The result is that the evidence for a class i given an input x is:

$$\text{evidence}_i = \sum_j W_{i,j} x_j + b_i$$

where W_i is the weights and b_i is the bias for class i, and j is an index for summing over the pixels in our input image x. We then convert the evidence tallies into our predicted probabilities y using the "softmax" function:

$$y = \text{softmax}(\text{evidence})$$

Here softmax is serving as an "activation" or "link" function, shaping the output of our linear function into the form we want -- in this case, a probability distribution over 10 cases. It basically involves the converting tallies of evidence into probabilities of our input being in each class. It's defined as:

$$\text{softmax}(x) = \text{normalize}(\exp(x))$$

IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

NCAIT 2017

JSS Academy of Technical Education

Vol. 4, Special Issue 8, May 2017

But understanding softmax the first way is more helpful: a process of exponentiating its inputs and then normalizing them. This means that the weight given to any hypothesis increases multiplicatively with just one more unit of evidence. Conversely, a hypothesis gets a fraction of its earlier weight if it has one less unit of evidence. No hypothesis ever has zero or negative weight.

These weights are then normalized, so that they add up to one, forming a valid probability distribution. Our softmax regression looks like the following (figure 13), although with a lot more xs. A weighted sum of the xs, addition of a bias, and then application of softmax is done for every image.

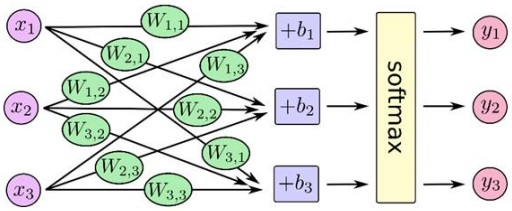

Figure 13 Graph of softmax regression function

If we write that out as equations, we get (figure 14):

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{bmatrix} \right)$$

Figure 14 Matrix representation of the function

The procedure can be "vectorized", turning it into a matrix multiplication and vector addition as follows(figure 15). This is helpful for computational efficiency.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Figure 15 Vector representation of the function

More compactly, we can just write:
y=softmax(Wx+b)

IMPLEMENTATION

Implementing the Regression

Tensor Flow needs to be imported using the below code:
importtensorflow as tf
x = tf.placeholder(tf.float32, [None, 784])

Here, x isn't a specific value. x is a placeholder, we will input the value when we need to run a computation. This is because any number of input images must be accepted and each flattened into a 784-dimensional vector. A structure [None, 784] is used to represent this as a 2-D tensor of floating-point numbers. (Here None means that a dimension can be of any length.)

Weights and biases are also needed for our model. It could be treated these like additional inputs, but Tensor Flow has an even better way to handle it: Variable. A Variable is a modifiable tensor that lives in TensorFlow's graph of interacting operations. The model parameters are generally Variables in machine learning applications.
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

But we initialize both W and b as tensors full of zeros by giving **tf.Variable** the initial value of the Variable. Since the model learns W and b, it doesn't matter very much what they initially are.

Notice that W has a shape of [784, 10] because we want to multiply the 784-dimensional image vectors by it to produce 10-dimensional vectors of evidence for the difference classes. b can be added to output since it has a shape of [10].
y = tf.nn.softmax(tf.matmul(x, W) + b)

First, multiply x by W with the expression **tf.matmul(x, W).** This is flipped from when it multiplied them in our equation, where we had Wx, as a small trick to deal with x being a 2D tensor with multiple inputs. Then add b, and finally apply tf.nn.softmax.It only took one line to define our model, after a couple short lines of setup. A softmax regression is not made particularly easy by Tensor Flow: it's just a very flexible way to describe many kinds of numerical computations, from machine learning models to physics simulations.

Training the model

In order to train the model, defining what it means for the model to be good is needed. But, we typically define what it means for a model to be bad in machine learning, called the cost or loss, and then try to minimize how bad it is. But the two are equivalent.

One very common, very nice cost function is "cross-entropy." Surprisingly, cross-entropy arises from thinking about information compressing codes in information theory but it winds up being an important idea in lots of areas, from gambling to machine learning.

The cross-entropy is measuring how inefficient our predictions are for describing the truth. To implement cross-entropy we need to first add a new placeholder to input the correct answers:
y_ = tf.placeholder(tf.float32, [None, 10])
Then we can implement the cross-entropy,
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))

First, the logarithm of each element is computed by **, tf.log**. Next, multiply each element of y_ with the corresponding element of **tf.log(y)**. Then **tf.reduce_sum** adds the elements in the second dimension of y, due to the reduction_indices [1] parameter.

Finally, The mean over all examples in the batch is computed by **tf.reduce_mean**. Since the entire graph of our computations is known, it can automatically use

IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

NCAIT 2017

JSS Academy of Technical Education

Vol. 4, Special Issue 8, May 2017

the algorithm to efficiently determine how your variables affect the cost we ask it to minimize. Then it can apply our choice of optimization algorithm to modify the variables and reduce the cost.

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

In this case, it is common to ask Tensor Flow to minimize **cross_entropy** using the gradient descent algorithm with a learning rate of 0.5. Tensor Flow simply shifts each variable a little bit in the direction that reduces the cost using a simple procedure - Gradient descent. Tensor Flow can provide many other optimization algorithms by just tweaking one line.

Now the model is set up to train. One last thing before we launch it, it adds an operation to initialize the variables we created:

init = tf.initialize_all_variables()

It is now possible to now launch the model in a Session, and run the operation that initializes the variables:

sess = tf.Session()
sess.run(init)
fori in range(1000):
batch_xs, batch_ys = mnist.train.next_batch(100)
sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

Each step of the loop gets a "batch" of one hundred random data points from our training set. It is needed to run train_step feeding in the batches data to replace the placeholders. Stochastic training involves the usage of small batches of data. In this case, stochastic gradient descent. Ideally, it is likely to use all our data for every step of training because that would give us a better sense of what we should be doing, but that's expensive. So, instead, here it uses a different subset every time. Doing this is just as beneficial and also cheaper.

RESULTS

Results are shown in figure 16, figure 17 and figure 18.

Figure 16 Result after encoding few images of data set

Figure 17 Result of softmax regression model

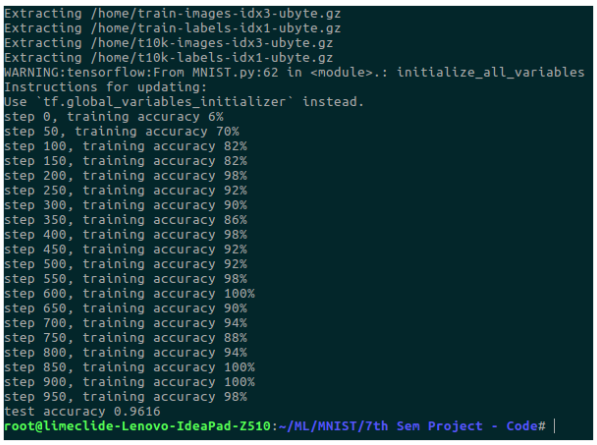

Figure 18 Result of CNN

Analysis of Experimental Results

Evaluation metric :

In TensorFlow, **tf.argmax** is an extremely useful function which gives us the index of the highest entry in a tensor along some axis. For example, tf.argmax(y,1) is the label our model thinks is most likely for each input, while tf.argmax(y_,1) is the correct label. We can use tf.equalto check if our prediction matches the truth. To determine what fraction are correct, we cast to floating point numbers and then take the mean.

accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy,feed_dict={x:mnist.test.images, y_: mnist.test.labels}))

We use the following function to compute the accuracy on our test data. This should be **around 92%.**Getting 92% accuracy on MNIST is not very good. The approach we have is a convolutional neural network. To evaluate it we will use a code that is nearly identical to that for the simple one layer SoftMax network above.

The differences are that:
Replace the steepest gradient descent optimizer with the more sophisticated ADAM optimizer.
Include the additional parameter keep_prob in feed_dict to control the dropout rate.
Add logging to every 100th iteration in the training process.
This will get us to around 99.2% accuracy, not state of the art, but respectable.

IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

NCAIT 2017

JSS Academy of Technical Education

Vol. 4, Special Issue 8, May 2017

Inference from the result

The SoftMax regression algorithm which uses a trivial modeling provides an accuracy of about 92%, which is considerably inefficient given that it uses a highly efficient algorithm. This led us to the usage of Convolutional Neural Network along with the regression method mentioned earlier to attain a higher accuracy.

The final result of over **99%** is an indicator of the power of a neural network to think and compute based on what it has been trained for. The necessity of this problem of digit classification is realized when combining it with other algorithms to form a part of solving a higher end problem.

CONCLUSION

The concepts of Neural Networks, machine learning and data mining are being implemented in almost all problems faced by technologists and programmers around the world. The idea is to train a computer to think and make decisions like a human being. The concepts used in this paper helps us to understand the essential requirements to build a Convolutional Neural network. The layers in the network harness the usage of weights, biases, pooling, etc. The model provides an accuracy of over 99% using the mathematical regression alone thus enabling it to be used in other problems. This optimal digit classifier has excellent accuracy, which is not remarkable, because unlike other high performance classifiers it does not include prior knowledge about the problem. In fact, this classifier would do just as well if the image pixels were permuted with a fixed mapping. It is still slower due to the memory hungry nature of the large dataset. When plenty of data is available, many methods can attain acceptable accuracy. Although the neural-nets methods require considerable training time, trained networks run faster and require much less space than memory based techniques. The nets' advantage will be more striking as training databases continue to increase in size.

FUTURE WORK

While this paper exploits the manipulation of the various parameters, some features may affect the optimal classification of objects more than others. As part of our future enhancements, we aim to find these features and optimize them so as to find the most accurate solution for classification.

Furthermore, it would be worthwhile to run this system for multi -digit recognition and character recognition. Most likely, this would aid in complete handling of occlusion and would lead to improved detection and classification results. Data storage should be as efficient as possible, in spite of having a large number of training samples.

REFERENCES

[1] Hayder M. Albeahdili, Haider A Alwzwazy, Naz E. Islam, "Robust convolutional neural networks for image recognition", (IJACSA), pp 10-21, Volume 6, Issue 11, 2015.

[2] Sherif Abdel Azeem, Maha E1 Meseery, Hany Ahmed, "Online Arabic Handwritten digits recognition", Frontiers in handwriting recognition (ICFHR), pp. 25-67, 2012.

[3] Liu, C.L. "Handwritten digit recognition: Benchmarking of State-of-the-art techniques", Pattern recognition 365, (IJACSA), pp 2271-2285, 2003.

[4] LeCun, Y. et al" Comparison of learning algorithms for handwritten digit recognition", International conference on Artificial Neural networks, France, pp. 53–60, 1995

[5] K. Gaurav and Bhatia P. K., "Analytical Review of Preprocessing Techniques for Offline Handwritten Character Recognition", 2nd International Conference on Emerging Trends in Engineering & Management, ICETEM, pp. 346-368, 2013.

[6] Salvador España-Boquera, Maria J. C. B., Jorge G. M. and Francisco Z. M., "Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models", IEEE Transactions on Pattern Analysis and Machine Intelligence, 13, pp.1024-1035, Vol. 33, No. 4, April 2011.

[7] U. Pal, T. Wakabayashi and F. Kimura, "Handwritten numeral recognition of six popular scripts," Ninth International conference on Document Analysis and Recognition ICDAR 07, Vol.2, pp.749-753, 2007.

[8] Anita Pal &Dayashankar Singh, "Handwritten English Character Recognition Using Neural," Network International Journal of Computer Science & Communication. Vol. 1, No. 2, pp. 141-144, July-December 2010.

[9] J. Pradeep, E. Srinivasan and S. Himavathi, "Diagonal Based Feature Extraction For Handwritten Alphabets Recognition System Using Neural Network", International Journal of Computer Science & Information Technology (IJCSIT), Vol 3, pp.326-339, No 1, Feb 2011.

[10] A. Brakensiek, J. Rottland, A. Kosmala and J. Rigoll, "Offline Handwriting Recognition using various Hybrid Modeling Techniques & Character N-Grams", Available at http://irs.ub.rug.nl/dbi/4357a84695495.

[11] Reena Bajaj, LipikaDey, and S. Chaudhury, "Devnagari numeral recognition by combining decision of multiple connectionist classifiers", (IJCSIT), Vol.27, part. 1, pp. 59-72, 2002.

[12] Sandhya Arora, "Combining Multiple Feature Extraction Techniques for Handwritten Devnagari Character Recognition", IEEE Region 10 Colloquium and the Third ICIIS, Kharagpur, pp. 376-398, December 2008.

[13] Mohammed Z. Khedher, Gheith A. Abandah, and Ahmed M. Al-Khawaldeh, "Optimizing Feature Selection for Recognizing Handwritten Arabic Characters", proceedings of World Academy of Science Engineering and Technology, vol. 4, February 2005 ISSN 1307-6884.

[14] SushreeSangita Patnaik and Anup Kumar Panda, "Particle Swarm Optimization and Bacterial Foraging Optimization Techniques for Optimal Current Harmonic Mitigation by Employing Active Power Filter Applied Computational Intelligence and Soft Computing", Volume 2012, Article ID 897127.

[15] T.Som, SumitSaha, "Handwritten Character Recognition Using Fuzzy Membership Function", International Journal of Emerging Technologies in Sciences and Engineering, Vol.5, No.2, pp. 11-15, Dec 2011.

[16] G. Pirlo and D. Impedovo, "Fuzzy Zoning Based Classification for Handwritten Characters", IEEE Transaction on pattern Recognition and Machine Intelligence, vol.19, no. 04, pp.780-785, August 2011.

[17] Nafiz Arica, and Fatos T. Yarman-Vural, ―Optical Character Recognition for Cursive Handwriting,‖ IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.24, no.6, pp. 801-113, June 2002.

[18] M. Hanmandlu, O.V. Ramana Murthy, "Fuzzy model based recognition of handwritten numerals", pattern recognition, vol.40, pp.1840-1854, 2007.

[19] Renata F.P.Neves,Alberto N. G. Lopes Filho, Carlos A.B.Mello, CleberZanchettin, "A SVM Based Off-Line Handwritten Digit Recognizer", International conference on Systems, Man and Cybernetics, IEEE Xplore, pp. 510-515, 9-12 Oct, 2011, Brazil

[20] Yoshimasa Kimura, "Feature Selection for Character Recognition Using Genetic Algorithm", Fourth International Conference on Innovative Computing, Information and Control 978-0-7695-3873-0/09© 2009 IEEE, 2009.