Recognizing Handwritten Digits Using Artificial Neural Network

Mohammad J. Sharara
Department of Electrical Engineering
Lebanese University
Beirut, Lebanon

Haidar Y.Mehsen
Department of Electrical Engineering
Lebanese University
Beirut, Lebanon

Terms of Reference

This report is submitted in fulfillment of the requirements of the Communication and Mini-Project course, under the supervision of Dr. Y. Harkouss, department of Electrical Engineering, Lebanese University.

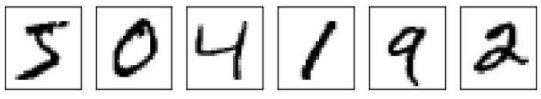
I. Introduction

Trying to comprehend and decipher intelligence in its core has various consequences on pursuing knowledge and uncovering answers on unanswered questions. For instance, simulating this ability by the means of a machine may contribute to solving complex real word tasks, leveraging the speed and efficiency of solving problems, and retain a high level of legitimacy and accuracy. As a result, the scientists went on a long journey to uncover the essence of the cognitive process and to understand the behavior and the functioning of the human brain down to the level of individual neurons. Driven by the advancements in discovering the mechanisms of functioning of neurons, and by the philosophical view that the process of human thinking can be mechanized, scientists went on the quest of building intelligent machines that simulates human cognitive functions as vision, creativity, and language processing. Out of that and many other factors, a completely new discipline emanated: Artificial Intelligence: a broad field that aims to mimic the cognitive functions of humans. This field branches into many domains out of which machine learning is a bold one that targets at giving the computer the ability to learn without being explicitly programmed. A manifestation of the latter subfield was developing a mathematical model that mimics - to some extent and with some assumptions - the operation of neurons. This model is called “neural networks”. Implementing this model to employ it in problem solving was not possible without the emergence and the evolution of programmable digital computers and the take off in their computational power and efficiency. The prominence of neural network protrudes from the fact that they can learn and model the relationships between inputs and outputs that are nonlinear and complex, make generalizations and inferences, reveal hidden relationships, patterns and predictions, and model highly volatile data (such as financial time series data) and variances needed to predict rare events (such as fraud detection). As a result, neural networks can improve decision processes in a vast spectrum of areas. For instance, characterizing dark matter in particle physics, finding patterns in astronomical data in astronomy, using genomic data to predict protein structures in genetics, modeling and predicting plants’ biosynthesis in complex organic chemistry, and the list goes on. Moreover, they lead to a paradigm shift in scientific research: instead of starting from known set of rules and utilizing this set to find new discoveries, feed an enough amount of data and let the network find hidden patterns and relations on its own. One of the fundamental applications of neural network is pattern recognition. Its applications extending to computer vision, language processing, recommender systems, and much more. Optical character recognition (known as OCR) plays an important role in transforming

printed materials into digital text files, saturating the rising need for automation of systems. Specifically, the object in this report is to handle the process of recognition of handwritten digits using a deep neural network that will be trained by the MNIST dataset of handwritten digits images. An assessment of whether the network is able to generalize and perform well on handwritten digits of different handwritings outside the training set will be carried out. That is, to what extent such model is able to learn the pattern of handwritten digits? The network will be implemented in an android application that uses the camera to take a picture of a handwritten digit and is able, by feeding it to the network and obtaining its output, to recognize this digit by writing to the screen the digital digit corresponding to it. This report is organized as follows: The materials and methods section that states the instruments and software used in addition to the methodology and scientific approach that was adopted to construct the model/classifier. The results section included the reached accuracy and cost at the end of the model training. In the discussion section, the validity and the consequences of the results was discussed. Finally, the conclusion included a brief summary of the procedure and a comment about the overall work.

II. Materials and Methods

The devices and software tools used in the overall procedure are to be mentioned. The didactic materials used in this process consisted of raw data called the MNIST dataset, standing for "Modified National Institute of Standards and Technology". It 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 The MNIST database contains 60,000 training images and 10,000 testing images. Each image is a greyscale and 28 by 28 pixels in size. Here is a few images from MNIST:



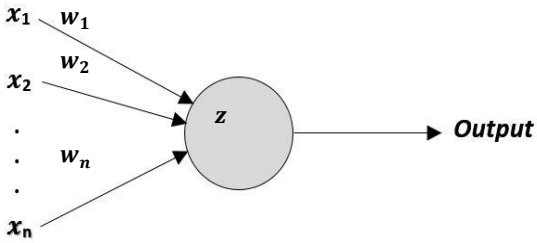
The images in this set are scanned handwriting samples from 250 people, half of whom were US Census Bureau employees, and half of whom were high school students. The set name comes from the fact that it is a modified subset of two data sets collected by the United States' national institute of standards and technology. When 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 bounding box and anti-aliased, which introduced grayscale levels. They attempted to "re-mix" the samples from NIST's original datasets. 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. To make this a good test of performance, the test data was taken from a different set of 250 people than the original training data (albeit still a group split between Census Bureau employees and high school students).

The neural network code was written in Python programming language, which is interpreted, high-level general-purpose programming language and the most widely used language for machine learning. It provides simplicity, consistency, access to substantial libraries and frameworks for AI and machine learning, flexibility, platform independence, and a wide community. The libraries that were used to facilitate the creation and execution of the neural network include Numpy, Matplotlib, and Keras. The usage of the latter was restricted to the datasets module that allowed loading the MNIST dataset as a 2-tuple. Numpy is an extensive library that allows the manipulation of matrices carried out as n-dimensional arrays (called Ndarrays) providing tons of methods to perform various operations on them. It also supports single instruction multiple data (SIMD) vector processing (parallelization) as it realizes fixed-type contiguous memory blocks arrays that, unlike standard Python lists, results in faster execution time and lower memory overhead. Matplotlib was used to plot as a 2-dimensional line graphs any necessary visualization of results.

The platform that was used to execute the written program is Google Colab, which provides a cloud based system that allows anybody to write and execute arbitrary python code through the browser. The cloud platform used version 3.6.9 of Python at the time of writing this report. The computational power that is allocated on the cloud of this platform is specified by two 22-cores Intel ® Xeon ® CPU with a frequency of 2.2 GHz each, 55 MB cache

each, a memory of 13 GB, and a 12GB NVIDIA Tesla K80 GPU that can be used for up to 12 hours continuously.

The constructed neural network consisted of a set of interconnected nodes, called artificial neurons, organized across several layers. Where the number of layers and the number of nodes in each layer can be set as necessary. They fall under the category of hyper parameters that can be tuned to leverage the accuracy of the network. An artificial neuron is a mathematical model that mimics to some extent how a biological neuron works. That is, it accepts a certain combination of real inputs, does some evaluations, and then outputs a corresponding result. Thus, it can be seen as a device that decides by "weighing" up evidence. To illustrate, consider the following figure of a node:



Two operations are carried out in each node:

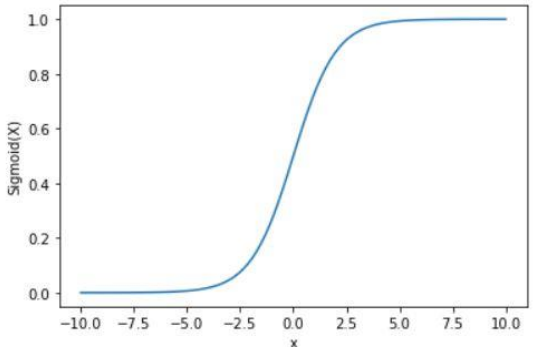
- Summation and addition of a real number $\boldsymbol{b}$ called the bias.

$$z = \sum_{i=1}^{n} w_i x_i + b \qquad (1)$$

- Applying a function called an activation function that can have different forms depending on the case. We'll use a function called sigmoid function:

$$f(z) = \sigma(z) = \frac{1}{1+e^{-z}} \qquad (2)$$

The job of an activation function is, in general, to transform the relation between the input and the output of the node, hence the network, from linear to non-linear one, thus enabling the network to learn more complex patterns. The sigmoid function maps any real number in $\boldsymbol{R}$ to a real number between zero and one. As much as this number is large, the output will be near one. Similarly, as much the number is negative, the output will be near zero. The result $f(z)$ of any node is called an activation. Note that, by convention, the input vector x is denoted the first activation of the network. The following figure shows the plot of sigmoid function:



The intuition behind the bias $\boldsymbol{b}$ is that it represents how hard it is for the corresponding node to fire, i.e. to output a value that is near one. Instead of writing $\boldsymbol{z}$ as a summation, which is cumbersome, we can write it as a scalar product of two vectors $\boldsymbol{w}$ and $\boldsymbol{x}$. Where each one is an n-dimensional vector of dimension determined by the number of inputs. Henceforth, the expression of $\boldsymbol{z}$ will be:

$$z = w.x + b$$

Zooming out from a single neuron, a layer consists of a set of nodes such that the outputs from each node in the previous layer is connected to the input of each node in the current layer. The same thing is between each two layers of the network. It is mandatory to simplify the notations using linear algebra, not only to be clear, but also to contribute to transform the mechanism of the network to a written code. Thus, each layer is denoted a weights' matrix $\boldsymbol{W}$ that contains the weights associated between the nodes of the current layer and the previous one. A weight w_{ij} in this matrix denotes the connection between the j^{th} neuron in the previous layer and the i^{th} neuron in the current one. Hence, this matrix can be seen as the vertical stacking of weights' vectors corresponding to each neuron in the layer. Similarly, each layer is denoted a bias vector $\boldsymbol{b}$ that contains the set of biases corresponding to each neuron in the current layer. The first layer of a network is denoted the input layer, that is the vector that contains the data that we would feed to the network. The last layer is called the output layer, and it contains the result of computation of the inputted data. It might be obvious now that each layer except the input one is associated a weights' matrix. Any layer in between the input and the output layer is called a hidden

layer. A network is called "Deep neural network" if it contains at least one hidden layer. In this way, a neuron in the second layer can make a decision at a more complex and more abstract level than neurons in the first layer. And even more complex decisions can be made by the neurons in the third layer. In this way, a many-layer network of neurons can engage in sophisticated decision making. In our case, the output layer will consist of 10 nodes denoting the ten digits (from 0 to 9) that we want to detect. The node with the largest activation will signify that its corresponding digit is detected. Sticking to the described properties and architecture of a neural network, a deep network was initialized with two hidden layers: the first layer has 40 nodes while the second one has 20 nodes. In addition, the weights of the network were randomly initialized with a standard normal distribution, and the biases were initialized as zeros. This is because it is found that randomly initializing the network parameters resulted in faster learning. The network is ready to feed forward the training data inside it. Thus, having a 28 by 28 pixels picture of handwritten digit, it is transformed to an array of numbers denoting the pixels. Then it is reshaped to the dimension of 784 by 1 in order to be inserted to the input layer. Moreover, the output component of the training set and the testing set was vectorized to fit to the network output, meaning that it was transformed from a number between 0 and 9 to vector of ten components all equal to zero except the component that corresponds to the initial value, it was set to one. Finally, the data was normalized by dividing each number by 256 (where each pixel is a grayscale number between 0 and 256 representing the degree of blackness). By normalizing all of our inputs to a standard scale, we are allowing the network to learn more quickly the optimal parameters for each input node. After feeding data to the network, the activations from each layer that are calculated according to equations **(1)** and **(2)** are sent after computation to the following layer until they reach the output node. This is in essence the forward propagation. Up to this point, the network is ready to accept data, but it has not been trained yet. Here comes the algorithm of "Backpropagation", which: "repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector" [4]. To quantify how well we are achieving this goal we define a cost function:

$$C = \frac{1}{2n}\sum_x ||y(x) - a||^2 \quad \textbf{(3)}$$

Which represents the mean square error (**MSE**) Euclidean distance in the m-dimensional space between the output vector a and the actual output vector $y(x)$ associated with the input vector x from the training set. n is the total number of training inputs and m is the size of the output layer, which is, in our case, 10. This cost function, also called objective function or loss function, is chosen such that it is a quadratic function, admitting an important property, which is that it has an absolute minimum. This will help us later when applying the optimization algorithm that aims to reduce the cost function by trying to find an absolute minimum.

Since the calculation of the cost function depends on the output of the network, where the latter depends on the set of weights and biases corresponding to every layer in the network, thus the cost function also depends on all those weights and biases. Backpropagation makes use of the partial derivatives and the chain rule in order to calculate, let us say, the contribution of each element of the network parameters to the cost function through calculating its partial derivative. More precisely, the partial derivative of the cost function with respect to a certain parameter indicates how much perturbing this parameter by a differential value will affect the cost function. Some notations are defined to facilitate the process. For instance, $\frac{\partial C}{\partial w^l_{jk}}$ represents the derivative of the cost function with respect to the weight of connecting the j^{th} neuron in the l^{th} layer to the k^{th} neuron in the $(l-1)^{th}$ layer. Similarly, $\frac{\partial C}{\partial b^l_j}$ represents the derivative of the cost function with respect to the bias of the j^{th} neuron in the l^{th} layer. Backpropagation will compute those partial derivative for every node in the network. But to compute those, we first introduce an intermediate quantity, δ^l_j , which we call the local gradient, which is the error in the j-th neuron in the l-th layer. Backpropagation will give us a procedure to compute the error δ^l_j, and then will relate δ^l_j to $\frac{\partial C}{\partial w^l_{jk}}$ and $\frac{\partial C}{\partial b^l_j}$. It turns out after working the linear algebra of the network model and chasing the indices that the four main equations used in backpropagation algorithm are given below [2]:

- $\delta^l = \nabla_a C \odot \sigma'(z^l)$
- $\delta^l = ((w^{l+1})^T \delta^l) \odot \sigma'(z^l)$
- $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \, \delta^l$
- $\frac{\partial C}{\partial b^l_j} = \delta^l$

The $\odot$ denotes the hadamard product that is the element-wise product of vectors. And the superscript T is the transpose of the weights matrix of the $(l+1)^{th}$ layer. One thing to note is that the training set is divided into mini-batches where the size of the batch is a hyper parameter that can be tuned to increase the efficiency of the network. And the Backpropagation is done as a mini-batch stochastic gradient descent. Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated accurate steps in the opposite direction of the gradient of the function at the current point, because this is the direction of steepest descent. A problem that arises here is that for each step that should be taken we have to iterate over the whole dataset to get the accurate steepest direction to the minimum resulting in a great time inefficiency due to computational limitation. Thus, it is more convenient to iterate over a subset of training examples in order to get an approximation of the direction (that is determined by the gradient), and that is the point behind splitting the training set into mini batches. From here arises the term stochastic, were the path taken by the algorithm is to some extent random or noisy, but will eventually reach the minima. Hence, in this way, a tradeoff was done between path accuracy and time efficiency. This means that for every mini-batch we will loop over each training example, feed forward it, calculate the output gradient, and then back propagate the error. After traversing the whole mini-batch, the parameters of the network will be updated based on the average of the calculated gradients over the mini-batch, meaning that a step of gradient descent will be taken. An epoch is when an entire dataset is passed forward and backward through the neural network only once. The number of epochs is also a hyper parameter that can be set on demand. In each epoch, the training set is shuffled to ensure that the batches are representative of the training set.
So, the algorithm of backpropagation becomes as follows:

1) Input a set of training examples.

2) For each training example x: Set the corresponding input activation $a^{x,1}$ and perform the following steps:
   • Feedforward: For each l= 2, 3, . . . , L compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$.
   • Output error $\delta^{x,L}$: Compute the vector $\delta^{x,l} = \nabla_a C_x \odot \sigma'(z^{x,l})$
   • Back propagate the error: For each l = L−1, L−2,..., 2 compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l}) \odot \sigma'(z^{x,l})$

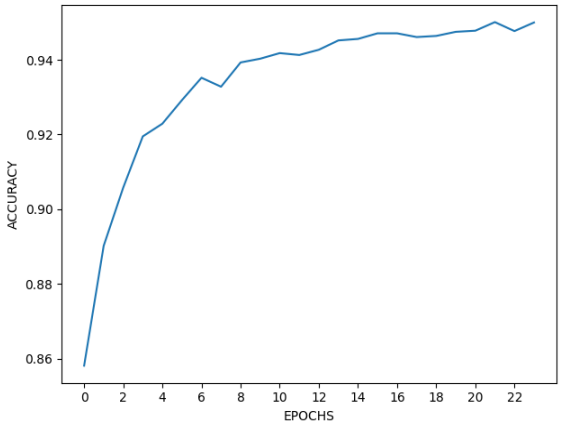
3) Gradient descent: For each l = L, L −1, . . . , 2 update the weights according to the rule $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$ and the biases according to the rule $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$

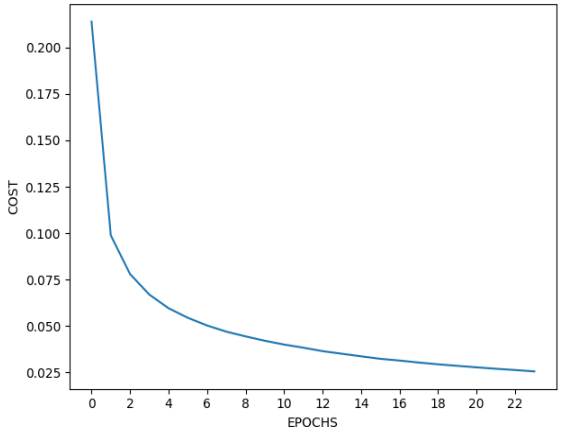
η is a hyper parameter that defines the step size of the gradient descent algorithm.

The network training was carried out over 24 epochs, a mini batch size of 50, and a learning rate of three. Then the network was serialized and saved in order to be used in the android application. The code of the network was put on a cloud server, alongside with the file containing the data of the trained model. The android application is a graphical interface that allows capturing a picture of a handwritten digit, sends it to the server where the image will be converted to an array and processed in the network, then returns the result and displays it on the phone screen.

III. Results

The training of the network took about two and a half minute, and the accuracy jumped from 85.81% at the first epoch to a very good 95% percent at 24^{th} epoch. Also, the cost determined by the function **(3)** decrease from 0.214 at the first epoch to 0.025 at the last epoch. The variation of the accuracy and the cost with respect to each epoch during the training phase is shown on the following two figures:





Notice the fluctuations in accuracy due to the stochastic sense of the optimization algorithm, where the direction of the steepest descent is approximated not from the whole training set, but from the mini-batch.

The android application was tested on many pictures of digits and performed very well in detecting the corresponding digit.

IV. Discussion

Given that the aim of implementing the neural network model was to be able to detect handwritten digits with high precision, it can be affirmed that the objective was reached. This is interpreted from the results. The goal of training a network is to reduce as much as possible the objective function that measures the distance between ground true label (the true digit) and the result outputted by the network. Arriving at a cost of 0.025 clearly emphasizes that the network is "trained". This is further confirmed by another measure, which is the accuracy. However, there is one thing to note before this. A major problem that may be faced during training a neural network model is the overfitting problem, also known as the problem of generalization. That is, an over fitted model is one that becomes too complex, and does not generalize well on data outside the training set. Thus, detecting whether a model is over fitted is essential for declaring that this model will function well in reality. Overfitting can be detected from the performance of the network on the testing data that are different from the training data. The testing data were not fed to the network to update its internal weights and biases based on the gradients on these data. Hence, we can rely on the performance that is the accuracy, of the model measured from feeding the testing data in order to decide whether we have overfitting. In the written code of the network, the testing data were fed to the network, and the result spitted by the network was compared with the true value of the testing instance (the digit), that is the label: if they match, that is a point for the network. The number of true classifications of the network after feeding all the testing data was divided by the total instances of testing data that were fed to the network to obtain a percentage of the performance or the accuracy of the network. In this way, we can know from the plot of the accuracy if there is an overfitting. Precisely, the point where overfitting starts is the point that the accuracy of the network-measure on the testing data- start to decrease instead of continuing to increase. Looking at the plot of the accuracy as a function of training epochs clearly signifies that we do not have a case of overfitting. As a result, we can confidently say that we obtained a model with a very good accuracy of 95%.

An annoying property of a neural network model is its "black box" nature. The weights and biases of the network model learned automatically. What is really happening behind the scenes? Why would the network perform well on testing data while it was trained over the training data? To put it simple, there is no sort of magic happening here. Rather, what is happening at the core is a game of averaging. You are feeding thousands of training data that are scanned pictures of handwritten digits. That is, the network is fed, for each digit, thousands of scanned handwritten form of that digit representing a vast collection of different possible representations that enables the network to tune its parameters in accordance to that collection. Finally, the network arrives to a point where the weights are averages of all the collection of a digit's representations so that any image array of a digit we feed to the network from outside the set will probably fall under a form that is near the average of the trained data corresponding to that digit. In addition, recall that a single node can be

thought of as a device that makes a decision by weighing up evidence. This can be easily elicited from its underlying mathematical principles that governs its functioning: an input is multiplied by a certain weight, added to a bias, and then outputted through a non-linear activation function, where the output can be between zero and one. We can think of zero and one as yes or no: if the inputted data values do not agree with the node's weight and bias, then the answer will be probably no that is near zero, and vice versa. Hence, a node will give a value that determines the degree of agreement of the node with the input. In conclusion, the nodes at the first layer are performing simple decisions, and the nodes at the following layers are performing decisions that are more abstract and more complex. For example, the number nine consists of a loop and a line. In turn, the loop consists of round edges, and the line consists of smaller dashes. Being fed to the network, the nodes of the first layer can be heuristically thought of as deciding whether the data corresponds to dashes or round edges. If so, the corresponding nodes will fire to the next layer. In the next layer, the nodes will decide whether the arrived data corresponds to a line or a loop. If so, the corresponding nodes will fire to the last layer. The nodes of the last layer will decide if the arriving data corresponds to number nine or not and if so the 9^{th} node will fire. This is in essence a heuristic approach of what is happening inside the network.

Upon training the network, two major drawbacks were encountered. The first one is the need to try different hyper parameters so that we can choose those that corresponds to the best performance. There is no explicit rule that defines how to choose precisely those parameters except for some general heuristics. For instance, we cannot analytically calculate the optimal learning rate for a given model on a given dataset. But this parameter cannot be very small in order to avoid extremely slow learning, and it cannot be very large in order to avoid overshooting. Thus, we need to try the training many times to arrive at a rate that is good enough to reach our aim. Fortunately, a solution was proposed to this problem that may be applied to this network as a future improvement. This solution consists of modifying the code so that the learning rate becomes adaptive: that is it adapts itself according to gradient value, learning speed, size of weights, etc… Moreover, layers and layers sizes are also parameters that the network needs to be tested on different values of them in order to get a combination with high performance. The second drawback that was noticed is that the learning reaches a point where it becomes very slow, as if the network converges to a certain parameters values. This happens mainly due to the nature of the sigmoid activation function: Sigmoid neurons get saturated on the boundaries and hence the local gradients at these regions is almost zero. One solution could be to use different activation function. For instance softmax and ReLU functions are good replacements that could leverage the training speed and accuracy.

In fact, some researchers achieved an accuracy that is significantly higher than 95%, were they used different network type called convolutional neural network and an optimizer than stochastic gradient descent called Adam. They were able to reach an accuracy of 99.89% [5].

What have been done so far hangs great hopes on such models in order to leverage automating systems. The network can farther be tweaked to detect any alphanumeric digit, and to detect many characters at once. Deep neural networks are very promising in terms of upgrading the efficiency of optical digit recognition (OCR) systems in order to increase the accuracy and the speed in detecting characters.

V. Conclusion

In summary, neural networks power and importance come from the fact that they can work on their own to identify features implicitly with very high accuracy and speed. They are used in tons of domains and as a utility for various applications. One of the applications that we've implemented deep learning in in this project is the optical image recognition, specifically classifying handwritten digits. A training process for the network model is carried out over a large training set comprising of handwritten digits images where the network at the end of the training phase would have been learned the features of the training set and is able to generalize and perform very well on any image outside the training set. And this was assessed by the accuracy which is in essence an amount that points to how well the model can perform in classifying new images. After training the network and passing an accuracy of 95% it was implemented in an interface in order to use it in real life. This is done alongside with a function that preprocess the data from rescaling and inverting, to thresholding, and centering the digit in the image. This is very essential step that without it the networks performance outside the training set would have been bad. From our experience in the overall procedure we can infer that there is a lot of improvement and tweaking that can be done on such project to smoothen the performance even in

bad conditions such as a noisy image. This can involve using another models other than the standard model of neural network such as convolutional neural networks, and using another better optimizers and activation function that can leverage the accuracy to a value higher than ninety five percent.

Resources:

[1] Neural Networks
What they are & why they matter https:// www.sas.com.

[2] Michael A. Nielson, "Neural Networks and Deep Learning", Determination Press, 2015

[3]
https://en.wikipedia.org/wiki/MNIST_database

[4] Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0

[5] Savita, Ahlawat & Choudhary, Amit & Nayyar, Anand & Singh, Saurabh & Yoon, Byungun. (2020). Improved Handwritten Digit Recognition Using Convolutional Neural Networks (CNN). Sensors. 20. 3344. 10.3390/s20123344.