TOUCHSCREEN HANDWRITTEN DIGITS RECOGNIZER

Reda Eloualladi

EGR 4402 – Capstone Design

Supervised by Dr. Assem

Fall 2018
Student Statement:

I hereby state that I have applied ethics to the design process and in the selection of the final proposed design. I have also held the safety of the public to be paramount and have addressed this in the presented design wherever may be applicable.

______________________________
Reda Eloualladi

Approved by the Supervisor(s)

______________________________
Dr. N. Assem
ACKNOWLEDGEMENTS

I am thankful to Dr. Assem for being my supervisor for this project, and for having guided and advised me through it. I am also thankful to Călin Enăchescu and Cristian-Dumitru Miron from "Petru Maior" in University of Tîrgu Mureș for writing and publishing their paper named Handwritten Digits Recognition Using Neural Computing. It helped me understand the logic and implement an important part of my project. I also would like to thank the National Institute of Standards and Technology for providing the dataset of labeled handwritten digits I worked with. I also thank Yann LeCun from Courant Institute in NYU, Corinna Cortes from Google Labs in New York and Christopher J.C. Burges from Microsoft Research in Redmond for editing and normalizing the images in that dataset in terms of size and opacity to be more adapted and easier to process for Machine Learning. Finally, I would like to thank the Google Brain team from Google’s AI organization for producing the Machine Learning library for Tensorflow for Python. It speeds up the implementation and allows for focusing the efforts on the logic behind the algorithms and ways to optimizing the precision of the Machine Learning algorithms.
CONTENTS

ACKNOWLEDGEMENTS ......................................................................................... 3

CONTENTS ........................................................................................................ 4

TABLE OF FIGURES .............................................................................................. 6

ABSTRACT ........................................................................................................... 8

1 INTRODUCTION .................................................................................................. 10

2 STEEPLE ANALYSIS ........................................................................................... 12
   2.1 Social ........................................................................................................ 12
   2.2 Technological ............................................................................................ 12
   2.3 Economic .................................................................................................. 12
   2.4 Environmental .......................................................................................... 12
   2.5 Political .................................................................................................... 13
   2.6 Legal ......................................................................................................... 13

3 REQUIREMENTS SPECIFICATION ..................................................................... 14
   3.1 Functional requirements .......................................................................... 14
   3.2 Non-functional requirements ................................................................... 14

4 FEASIBILITY STUDY .......................................................................................... 15

5 THE ALGORITHM ............................................................................................... 17
   5.1 Supervised Machine Learning .................................................................. 17
   5.2 The Neural Network .................................................................................. 18
      5.2.1 From the perceptron to our Neural Network ....................................... 18
      5.2.2 The dataset ........................................................................................ 23
   5.3 The Forward Propagation .......................................................................... 24
5.4 The Backward Propagation ................................................................. 25

5.4.1 Computing the Loss ................................................................. 25

5.4.2 Updating the weights and biases ................................................ 25

6 THE PROGRAM .................................................................................. 28

6.1 Machine Learning Part ..................................................................... 28

6.1.1 Libraries ..................................................................................... 28

6.1.2 Dataset ....................................................................................... 28

6.2 Building and Using the Model .......................................................... 29

6.2.1 Creating the Model ...................................................................... 29

6.2.2 Compiling the model ................................................................. 30

6.2.3 Learning and Testing ................................................................. 31

6.2.4 Predicting our own number ....................................................... 31

6.2.5 Snapshots of Execution ............................................................. 33

6.3 Building our Graphical User Interface in Order to Start and Display
predictions ............................................................................................. 35

6.3.1 Screenshots/snapshots of our Graphical User Interface for prediction of
input number image .............................................................................. 35

6.3.2 Code of our Graphical User Interface for prediction of input number
image ..................................................................................................... 36

CONCLUSION ......................................................................................... 39

REFERENCES ......................................................................................... 40
TABLE OF FIGURES

Figure 1 Training set [3]........................................................................................................... 17
Figure 2 Perceptron [4]........................................................................................................... 18
Figure 3 Curve of the sigmoid function [5]............................................................................. 19
Figure 4 Neural Network [5].................................................................................................. 20
Figure 5 Number shape 1 [4].................................................................................................. 21
Figure 6 Number shapes 2, 3, 4 [4]....................................................................................... 22
Figure 7 Complete shape [4]................................................................................................... 22
Figure 8 Example of Shape of Cost Function [7]..................................................................... 27
Figure 9 Code 1 ....................................................................................................................... 28
Figure 10 Code 2 .................................................................................................................... 28
Figure 11 Code 3 .................................................................................................................... 29
Figure 12 Code 4 .................................................................................................................... 30
Figure 13 Code 5 .................................................................................................................... 31
Figure 14 Code 6 .................................................................................................................... 31
Figure 15 Code 7 .................................................................................................................... 31
Figure 16 Code 8 .................................................................................................................... 32
Figure 17 Code 9 .................................................................................................................... 33
Figure 18 Snapshot 1 ............................................................................................................. 33
Figure 19 Snapshot 2 ............................................................................................................. 34
Figure 20 Snapshot 3 ............................................................................................................. 34
Figure 21 Snapshot 4 ............................................................................................................. 34
Figure 22 Snapshot 5 ............................................................................................................. 35
Figure 23 Snapshot 6 ............................................................................................................. 36
Figure 24 Code 10 ............................................................................................................. 37
Figure 25 Code 11 ............................................................................................................. 37
Figure 26 Code 12 ............................................................................................................. 37
ABSTRACT

In this document, we are going to report the steps and results of the journey we took towards accomplishing the design of a computer program that allows recognizing handwritten digits input in real time through a touchscreen. The making of that software was divided into two parts. The first one is the interface part and the second one is Machine Learning part. It uses an Artificial Neural Network algorithm to learn from the Modified National Institute of Science and Technology dataset of labeled handwritten digits. Then, it presents an interface for the user to enter their own digit, and then displays the result. This document will show the steps followed to augment the precision of the result produced by the back-end part and the creation and adaptation of the front-end interface to comply precisely with the recognition mechanism.
LIST OF ACRONYMS AND ABBREVIATIONS

HDR: Handwritten Digits Recognition
ML: Machine Learning
NN: Neural Network
ReLU: Rectified Linear Unit
STEEPLE: Socio-cultural, Technological, Economic, Environmental, Political, Legal, Ethical
1 INTRODUCTION

Machine learning (ML) is a field in computer science aiming at enabling computers to learn a behavior from a set of empirical data. Its goal is to develop algorithms that make the computer display behaviors that it learned from experience instead of human instruction [1]. It is used mainly (but not only) in Artificial Intelligence. The latter is a field specialized in designing “intelligent agents”, which are devices that perceive their environment and uses that in making actions to maximize their chance of achieving their goals successfully [2].

As for the history of ML, a first algorithm, called Artificial Neural Network, or simply Neural Network (NN), based on the functioning of the human brain with its neurons, was proposed in the 1957. It is only in 1980’s that the algorithm was revisited by scientists for use in Artificial Intelligence, and its capacities were extended. Since then, many algorithms were developed (Decision Trees, Support Vector Machines and Naïve Bayes classifiers…) [1] along with an extension of the range of applications of that technology: security, bioinformatics, business advertising, language translation, and computer vision with the recognition of visual shapes (in which we find Handwriting Recognition).

My motivation for choosing that as a theme for my capstone is my fascination for Artificial Intelligence. I believe that in the process of creating machines and algorithms that mimic humans’ way of thinking, humans need to investigate their own way of thinking, and that gives them a better understanding of their own way of thinking and perception of the world. A second motivation is the extent to which that field allows for creativity. One can think of new ways to modify algorithms and new ways of solving problems and create their own custom-made agent. My third motivation is how helpful that is to humanity, and that will be developed
in my Socio-cultural, Technological, Economic, Environmental, Political, Legal and Ethical (STEEPLE) analysis.

After the STEEPLE analysis, I will expose the functional and non-functional requirements of my program. Then, I will provide a feasibility study to determine if it is possible to develop the program in term of resources and technicalities. Finally, I will present the development of the program step by step, which problems I ran into and how I tackled them, but also which resources I used for each step.

The methodology we will be using consists in implementing algorithms described in our sources, test our algorithm for precision and then implement an interface for the user.
2 STEEPLE ANALYSIS

2.1 Social
Handwriting recognition using Artificial Intelligence is done for the purpose of computerizing texts written by hand. This aims at recent and old texts. Instead of taking an extensive time to re-type those texts, they could be digitalized with artificial intelligence. It is important because digital texts are a lot easier to review and share and make it simpler to search for information in them.

2.2 Technological
Besides, Handwriting Recognition helped in the progress of Machine Learning, Computer Vision and Artificial Intelligence. This means making technology evolve, which has positive purposes. New technologies make people’s lives easier and allow them to communicate easier.

2.3 Economic
On the economic level, adding features of handwriting recognition, especially on phones or tablets that have a stylus can interest some people. If performant enough, the technology can be a good investment for professionals and students, for whom it would make notes-taking faster and easier. Still, the technology should try to be affordable enough for people to choose it as an alternative.

2.4 Environmental
On the environmental level, writing with a stylus on a device can help lessening the use of paper and pens/pencils. Paper and pencils (wood ones) are made using trees, so it would decrease deforestation. Besides pens and other pencils are made using plastic, and
plastic cannot be recycled. Thus, handwriting on digital devices would lower waste in the world.

2.5 Political

There seems not to be any direct correlation between politics and Handwriting Recognition applications. However, companies providing the application can gather data about the users, and that may give to those companies the power influence their political choices and tendencies. We do not know, however, of any concrete evidence of that. We will talk about how companies can gather data through Handwriting Recognition apps.

2.6 Legal

On the legal level, the main restrictions are linked to the privacy of data. Handwriting Recognition can be programmed to learn to recognize better the user’s handwriting. It does that by guessing the words that the user is writing and by learning from his corrections. That data (the words written) can be used by the companies that provide the application, to know the customer better, by knowing the words they use. That technology can be used for commercial purposes, to know the customer better and know how to target them with products advertisements.

2.6 Ethical

One of the most important uses of Machine Learning and AI is Bioinformatics. Machine Learning is used by medical research to detect or identify diseases or know the human body better, in order to provide better care and cures, and that helps sustain human life.
3 REQUIREMENTS SPECIFICATION

3.1 Functional requirements

- Import the Modified National Institute of Science and Technology Dataset:
  o Import dataset file directly to the program from a command that will download the dataset from its website
  o Save the dataset file in the same directory as the program

- Create the model
  o Take the value of the color is the pixels
  o Put the value of all the pixels in a one-dimensional array
  o Build a Neural Network with a number of nodes in the input layer equal to the number of pixels in the arrays
  o Activate the Neural Network
  o Test the precision of the model in a testing set

- Recognize handwritten number input
  o Allow user to input a number using a touchscreen
  o Predict in real-time the value of the number written

3.2 Non-functional requirements

- Windows Application
- Using Python
- Using Tensorflow
The aim of our feasibility study is to select the system that fits the most our performance requirements. We would like to determine if it is possible to develop our product in terms of resources and technicalities. Thus, we will analyze the problem and collect information for the product, including the data we will input to our system, how we will carry our process on that data, and the output we wish to obtain following our process, as well as the constraints applied on how the system behaves.

On the technical level, our two main concerns will be finding the right data and finding and coding the right algorithms. As for the input data, we will use a famous handwritten digits dataset assembled by the National Institute of Standards and Technology and arranged by Yann Lecun, professor at NYU, available at http://yann.lecun.com/exdb/mnist/. It contains a training set of 60,000 examples and a testing set of 10,000 examples. They have been centered and size-normalized in fixed-size labeled images, each number in one image.

Then, to input the data, the images will be scanned pixel by pixel by our program. Each pixel will have a value saying how dark its color is, and the value for each pixel will be put in an array. Then, the value of each element from the array will be fed to the input layer of our neural network. Each pixel value will be given to a unit from the input layer.

We will then use different types of neural network algorithms and select the one that gives us the most precise predictions. Part of our work will consist of adapting the different codes for neural networks available to our program. Another part will consist in building an interface for the user to input a handwritten number with a touchscreen.
In what concerns the resources, we will be programming on our laptop, using Internet connection to download the data and search for the algorithms to adapt. Our work will not need any additional expenses. We will use the programming language Python 3.6 for 64-bit architecture, and the IDE PyCharm 2018.2.2 (Community Edition). We have been introduced to Python IDEs in the Language and Compilers course in AUI. We will be using the Python library TensorFlow (and/or others, if necessary) for the neural networks. The work will be done by me (Réda Eloualladi), under the supervision of Dr. Nasser Assem. We estimate the time we will need to do it to two months.
5 THE ALGORITHM

5.1 Supervised Machine Learning

In order to do our Handwritten Digits Recognition (HDR), our program will learn from a set of images of handwritten digits. With each image, there is a label, allowing the computer to know which digit it is. The images, in addition to the labels, form our dataset. Our program will divide the dataset into two groups of labeled numbers. It will use the first one to train, and to learn to recognize numbers. We call it the training set. It will try to guess the label of each number of the training set, check whether it guessed correctly or not, and learn from that how to recognize the upcoming numbers. Then, when it has learned, it goes to the second set, which is called the testing set. The program uses it to calculate how accurate it is to recognize handwritten numbers. Again, the program will try to guess the label of each number of the testing set and see if it guessed correct or not. However, this time, it will do that in order to count how accurate it was and output a percentage that represents how accurate it was.
5.2 The Neural Network

5.2.1 From the perceptron to our Neural Network

Before explaining how the neural network for recognizing handwritten digits works, we will start by explaining how a perceptron works. A perceptron is a neural network that takes inputs equal to 0 or 1 (which here are $x_1, x_2, x_3$), applies a computation to them in a neuron, or unit (the circle in the drawing), and then delivers a binary output, which means outputs 0 or 1 [4]. (The neural network we will use will not have binary output).

Each of the links has a weight $w_1, w_2, \ldots$ and the output if defined by first calculating $\sum_j w_j x_j$, then comparing it to a threshold value. Thus,

- If $\sum_j w_j x_j \leq 0$, output $= 0$
- If $\sum_j w_j x_j > 0$, output $= 1$

The weight of the link determines how important the input is to determine the output. Instead of writing $\sum_j w_j x_j$ all the time, we will write $w \cdot x$, which makes sense considering that all $w_j$ constitute a vector $w$ and all $x_j$ constitute a vector $x$. In what concerns the threshold, we will name is bias, noted as $b$. Thus we can actually write, more simply:

- If $w \cdot x + b \leq 0$, output $= 0$
- If $w \cdot x + b > 0$, output $= 1$ [4]
The neural network then works by readjusting itself the weights and biases depending on how close it was to the rights value, as we keep giving it a different input, comparing its guesses the results it should be predicting. The problem with that model is that a small change in weights or biases can completely shift the output. For example if the neural network modifies its weight and biases after one input example, the changes could be too big and we would never converge to the wanted output.

To change that, instead of just calculating $w \cdot x + b$, the unit will do $w \cdot x + b$, take the result and apply to it a function, called the sigmoid function, noted $\sigma$.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \text{ with } z = w \cdot x + b$$

The result of that function will always be between 0 and 1 and the graph looks like this:

![Sigmoid function graph](image)

**Figure 3 Curve of the sigmoid function [5]**

In that function, small changes of bias or weights will result in small changes of output. However, neural networks are often more complex than outputting 0 or 1. Neural networks are often not perceptrons. A neural network is generally a bigger graph. The nodes,
also called neurons or units, are separated into three groups: the Input layer, the Hidden layers and the Output layer. The one we will use for our problem is called a feedforward neural network. (There are others called recurrent neural networks for example, which are called recurrent neural network, which take time in consideration to adjust weights and biases. Those can be more efficient in other problems.). In our feedforward neural network, neurons are only connected to neurons of adjacent layers and there are no loops. It gets the input in the Input layer, processes it in the hidden layers and returns its guess in the output layer. It learns by adjusting the weight of each edge (link) of the graph depending on how close that guess was to the right result.

![Neural Network Diagram](image)

**Figure 4 Neural Network [5]**

In our example, we input an image. The image is in black and white (grayscale), and its format is 28*28 pixels. Each pixel has a value between 0 and 1 corresponding to how dark it is. That value is called its activation. Furthermore, we have 28*28 pixels, thus 784 pixels in total. Therefore, in our neural network’s input layer, we will have 784 nodes.

Since we are trying to classify an input number according to the digit it represents, we have 10 possible classes (0, 1, ..., 8, 9). We want to separate our images into those classes and
each one should have only one class. Our problem is a segmentation problem. In the output layer, we will have 10 neurons, each corresponding to a class. The value that the output nodes will have corresponds to how much the neural network thinks the number belongs to each class. For example, if we input a 6 and the Neural Network tries to make a prediction, it could say it is 90% sure that it is a 6, 10% sure it is a 5, and 0% sure it is any of the other digits. We will actually use numbers between 0 and 1, so 0.9 and 0.2 instead of 90% and 10%. Then, the number is assigned to the class with the highest value, or activation. We could have used a binary output, which means 0 or 1 for the appurtenance of an input to a class. However, empirically, it is better to use the classes as a continuous value between 0 and 1.

Let us take an example to explain why. For example, let us suppose that the first output neuron catches whether the image presents the first shape:

![Figure 5 Number shape 1](image)

**Figure 5 Number shape 1 [4]**

It does this by giving a high weight to input pixels that are dark in that image. Let us also suppose that the 2nd, 3rd and 4th neurons of the hidden layer are detecting if it is true or not that the following shapes are occurring in our input image:
Those four images constitute the digit zero.

Thus, if those four neurons we just spoke about are firing (with high value), we can conclude that the input number we are trying to assimilate to a class is zero. The reality is more complex, but that is a good example where to start. Having ten classes is better than four. The reason behind that is that, if we had only four, the neural network would make the hypothesis that some of those neurons are more important than others to determine whether that image is a zero or not, and will give them weights accordingly. The reality is that we need all those neurons to fire in order to be able to determine that is a zero, because all four shapes need to be there in a zero. We need more neurons in order to be able to add other elements in order to determine that it is a zero [4].
Then, the weights of the links are adjusted in a repetition of two steps, which we will explain in the next paragraph. Those steps are: the forward propagation and the backward propagation. These two propagations are done using mathematical functions called activation functions, loss function and optimizer function. Some functions give more precise results depending on the problem and the dataset, and we will examine some in the next part. The precision also depends on the number of hidden layers. Note that each hidden layer can have its own activation function and number of neurons.

5.2.2 The dataset

Before going further in our explanation, we should give a short description of the dataset we will use, which is called the MNIST dataset. The MNIST dataset contains 60000 images that we can use as a training data, which means the data that is going to be input, number by number, to our neural network. They were created by scanning handwritten samples of digits from 250 people, half of that sample being employees of the US Census Bureau, and half of them being high school students. The images were, after the were scanned, converted in a greyscale color and their size was reduced to 28x28 pixels. The second part is our testing set. It is made of 10000 images, which we will use to measure how accurate the neural network was in its prediction of the class to which belongs each sample number image. Again, in our next explanations, we will use $x$ to identify the input. Thus, $x$ is a vector of size 784 dimensions. This time, the output that is the correct number that the neural network should predict for an image will be noted $y$, or $y(x)$. $y$ is not the number itself but actually another vector, consisting this time of only 10 dimensions. That is because, for example, if we input 5, the desired input is actually 0 for all the classes (digits) except the class that represents the digit 5. As we do in linear algebra, in order to note a column vector,
we use the following notation \( y(x) = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0) \). The superscript \( T \) means the transpose of a vector. We use it here simply because what we wrote is a line vector and what we need is a column vector. The superscript \( T \) here turns vectors’ lines from lines to columns and the vectors columns from columns to lines. Here we only have a row, so the row will be turned into a column vector [6].

5.3 The Forward Propagation

Once the data is input, it will go from the input nodes to the output nodes through a process called Forward Propagation. The weights of the links are first initialized to random values. Then, starts a process called Logistic regression. We go layer by layer in that process. Then, inside each layer, we go neuron by neuron. The activation (or value) of a neuron is calculated through steps. We first multiply the weight of each link between the neuron and the neurons of the previous layer. Then, we sum those products and add a bias, which is a value that is proper to each neuron. The bias works as a threshold that measures whether the activation is significant enough. That step is called linear regression. However, the result we will get will not be between 0 and 1. Thus, we finally apply an activation function to the value obtained in order to obtain a value between 0 and 1.

An example of activation function is the sigmoid function, which is the inverse of the sum of one and the exponential of minus the activation. The output value of that function will always be between 0 and 1. Another activation function is the Rectifier Linear Unit (ReLU) function. It equals the maximum between the input number and zero. Finally, another example is the Softmax function (i.e. Normalized Exponential function). It is a function that turns the numbers into probabilities so that their sum is equal to one. For example, \([2.0, 1.0, 0.1]\) as
input will be turned into $[0.7, 0.2, 0.1]$. It is equal to the division of the exponential of the logistic regression by the sum of the exponential of the logistic regression for each class.

5.4 The Backward Propagation

5.4.1 Computing the Loss

Once we have done a forward propagation, we do the backward propagation. This means that, this time, we will go layer by layer backwards. That is to say we will go through the layers in the opposite order, starting by the output layer. Furthermore, we obtain certain values in the output layer corresponding to the guess of the neural network. That value can be close or far from the actual value. For example, if we input a 7 in the input layer, the right activation we should get for the neurons of the output layer is 1 for node 7 and zero for all the other nodes. The Loss Function measures how close our guesses were to the right answer. An appropriate function for logistic regression we can use is the Cross-Entropy function. It is the negative of the sum of the product of the right value by the logarithm of the guessed value, and the product of one minus the right value and the logarithm of one minus the guessed value.

5.4.2 Updating the weights and biases

Once we computed the loss function, we need to use it to update the weights and the biases. Thus, before going to the next neuron in the layer, we will update all the weights linking the neuron to the previous layer, and the bias of each neuron. The new weight will be the difference between the old weight and the product of alpha and the derivative of the loss function with respect to the weight. The new bias will be the difference between the old weight and the product of alpha and the derivative of the loss function with respect to the bias. Alpha is the learning rate. It is a constant we configure for the neural network.
5.4.3 Cost function/gradient descent

To measure how far the neural network is from the real values, we use the Cost function. It first computes the sum of the squares of the differences between the predicted values and the correct values. Its result is called the cost of a single training example. That cost is smaller when the network’s guesses are closer to the correct values.

We can think of the whole network is a function that takes as input 784 values (input layer) and outputs 10 values, using the weights and biases as parameters. The Cost function is more complicated because it takes as input the whole weights and biases between all layers during one training example, and outputs a single value measuring the performance of the network during a training example, and then computes the average of those numbers over all the training examples.

For some functions in math, it is easy to find the minimum, but not for that one. If we call the cost function $C$ and a matrix containing all the weights during a training example $w$, we can represent the cost of a training example as $C(w)$.

We are trying to modify the weights and biases in order to minimize that cost. Thus, we are looking for the minimum of $C(w)$. What we can do is, every time compute the slope of the function. Then, we can go to a higher $w$ if the slope is negative, which means take a step to the right and take a lower $w$ if the slope is positive, which translates to going to the left. We can even choose to take. We can even make the size of the steps we take proportional to how steep the function is at that point. That means if the absolute value of the slope is small, we will take big steps, and, if it is large, we will take big steps.
Figure 8 Example of Shape of Cost Function [7]

Since $w$ is actually a vector, in reality, we will need to compute, as we do in Multivariable Calculus, the Gradient vector of the Cost function, rather than the derivative of the function. We will actually focus on the negative of the gradient: $-\nabla C(w)$. That vector will allow us to, somehow, compute the derivative for each component of $w$. If, for example, the first component of the gradient is positive, we will increase the first weight, and, if it is negative, we will decrease the first weight. Moreover, the more the value of that first component of the gradient is high, the more modifying that weight will impact the loss. That descent is called the Gradient descent.

5.4.4 Modifying activations

Once we have modified the weights, we will modify the activations of the neurons on the previous layer. The changes we will apply will depend on the size of the weights. Normally, we should do that for all training examples, but that would be slow. Thus, we set batches of training examples.
6 THE PROGRAM

6.1 Machine Learning Part

6.1.1 Libraries

We start by importing our libraries and dataset. The first library we import is Tensorflow. It is an open-source library, developed by Google and released in 2015, that is very popular in Machine Learning and Neural Networks. Within Tensorflow, we import Keras, which is an Application Programming Interface. It helped us because it has the potential to make building and modifying a model very fast, which allows focusing more on experimenting and finding the right model for our purposes. Next, Numpy and PIL are two libraries we will use to input our own digit image.

6.1.2 Dataset

We start by importing our libraries and dataset. The first library we import is Tensorflow. It is an open-source library, developed by Google and released in 2015, that is very popular in Machine Learning and Neural Networks. Within Tensorflow, we import Keras, which is an Application Programming Interface. It helped us because it has the potential to make building and modifying a model very fast, which allows focusing more on experimenting and finding the right model for our purposes. Next, Numpy and PIL are two libraries we will use to input our own digit image.
Subsequently, we import the MNIST dataset. As we presented in our feasibility study, it has 60000 images in the training set. It was taken from the National Institute of Standards and Technology (NIST). It is a sciences laboratory and an agency of the USA government that aims to promote innovation. Then that set was modified: MNIST dataset means Modified NIST dataset by Yann LeCun from NYU (et. al.). It was normalized in size and color in order to make it easier to use for image recognition purposes. Those images are imported as arrays representing the pixels. Their values are between 0 and 255, 0 for the brightest pixels and 255 for the darkest pixels. Loading the MNIST data (in line 13) returns two sets of data in the form of arrays: the testing set and the training set. We divide the value of every pixel by 255 in order to get a value between 0 and 1 for each pixel, describing its level of darkness.

6.2 Building and Using the Model

6.2.1 Creating the Model

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(196, activation=tf.nn.sigmoid),
    keras.layers.Dense(10, activation=tf.nn.sigmoid)
])
```

**Figure 11 Code 3**

Next, we use Keras to build the model. In line 34, we describe the input layer. The command “Flatten” allow for converting the image from a 2d-array into a 1-d array. It is the equivalent of taking all the lines of the image and putting them one after the other horizontally. After that, we code two hidden layers. They are “Dense” layers, which means fully-connected. That means all neurons of a layer are connected to all neurons of adjacent layers. The number of layers and neurons for each layer is arbitrary. We chose 196 for the first
one, which is the quarter of 784 pixels. This is because we imagine the neural network as determining the importance of each pixel in recognizing each number. For example, the digit zero has no black pixel in the middle of the image. Thus, that pixel is very important in determining whether the digit is zero, because if that pixel is white, there is a high chance that the number is a zero. Thus, we think grouping the pixels four by four is close to the actual image. We tried using 748 input nodes, but the precision of the results did not change a lot and the learning was a lot slower. As for the second hidden layer, we chose 10 because the output layer contains 10 neurons too. The activation functions are chosen arbitrarily too. We chose the Sigmoid function as a starter because it is the first one we learn when studying neural networks, and it gave great results.

6.2.2 Compiling the model

```python
model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

**Figure 12 Code 4**

The next step is to compile the model. For that, we need a loss function (line 40). It measures the accuracy of the model while it is training. The optimizer function (line 39) determines how the Neural Network is updated according to the data that goes through it and the loss function. Finally, the “metrics” is the function that measures the performance of our Neural Network.
6.2.3 Learning and Testing

```python
model.fit(train_images, train_labels, epochs=5)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

**Figure 13 Code 5**

After that, we input the training data (images of digits) to the model. We use the function `fit()` for that. Its first argument is the training images and the second one is the training labels. Finally, the number of epochs is the number of times the whole training set will be fed to the Neural Network.

```python
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

**Figure 14 Code 6**

Finally, we test our model and print the accuracy (which we named “test_acc” earlier).

6.2.4 Predicting our own number

```python
while(True):
    prompt = input("Predict?")

    im = Image.open("./Untitled.png").convert('L')
    newImage = Image.new('L', (28, 28), [255])
    im = im.resize((20, 20), Image.ANTIALIAS).filter(ImageFilter.SHARPEN)
    wtop = int(round(((28 - 20) / 2), 0))
    newImage.paste(im, (4, wtop))
```

**Figure 15 Code 7**

Now, let us move to the goal of our project, which is predicting our own number. For that, we start a loop, so that the user can predict as many times a number as they want to. We
also prompt their input so that the prediction can start. We use the library Image in order to open our input image, and we convert it to ‘L’, which means converting it to greyscale (black and white) if it is not yet in greyscale. Then we create a new blank image with the same properties as the images from the MNIST dataset. That means in greyscale, configured with the first argument ‘L’, with dimensions 28*28 and with value of the pixels’ darkness going from 0 to 255. In the MNIST dataset, the numbers do not take more space than 20*20 pixels in reality. Thus, we resize our image to 20*20. We also apply antialiasing and sharpening, which are filters that modify the image for quality. This is especially useful when we are inputting a photography of a number. In our demonstration, we will input a number using a touchscreen. Then, we paste our image in the center of the new image we created, leaving 4 digits up, down, left and right.

```python
      tv = list(newImage.getdata())  # get pixel values
      tva = [(255 - x) ** 1.0 / 255.0 for x in tv]
```

**Figure 16 Code 8**

Now, we use the function `getdata()` on our image `NewImage` in order to get the values of the pixels. We then invert the colors of our own image (by computing 255-x) in order to conform to the format of the MNIST dataset, and divide the value of the pixels by 255 in order to get, like previously, a value between 0 and 1 for the darkness of the pixels.
After that, we put those values in our list called `myList` and finally predict the image using `model.predict(myList)`.

6.2.5 Snapshots of Execution

We execute the program and the learning starts. We can see to what image of the set the program got. We can also see the loss that reduces and the accuracy that augments. On top of that, we can see which epoch is running.

Then, we obtain the accuracy of our recognition.
We use Paint 3D, the new version of Windows Paint in order to draw a number:

![Paint 3D screenshot](image)

Finally, we get out number predicted in the last line of our input.

![Test accuracy screenshot](image)
6.3 Building our Graphical User Interface in Order to Start and Display predictions

6.3.1 Screenshots/snapshots of our Graphical User Interface for prediction of input number image

Instead of only displaying the text predicted by the neural network, or the class with the biggest activation output after the machine learning part of our program, we decided to add a graphical user interface. For that, we imported the library Tkinter. It is a standard library in Python, very commonly used for graphical user interfaces, that we will help us to build ours. Our graphical user interface will allow for clicking on a button to start the neural network’s prediction process. It will act as a starter for the prediction, but, on top of that, it will also display the number. First, it will only show an empty window with only a button with “PREDICT” written on it. We decide to capitalize the prompt for visibility while keeping the size of the window adapted to displaying the number. This means having a button visible without having too much empty space.

Figure 22 Snapshot 5
Once the user has decided to start the prediction, and this means when the image they have
decided to input was drawn and saved, the can click our predict button. Once the button was
clicked, the number will appear. And this is what has happened in the following screenshot of
the graphical user interface:

![Image of a predict button with the number 8]

**Figure 23 Snapshot 6**

6.3.2 Code of our Graphical User Interface for prediction of input number image

A problem appeared when we decided to import the `tkinter` library. We first tried
importing the whole library with the command “`from tkinter import *`”, which means import
everything from `tkinter`. However, when we tried that, there was a conflict with the Image
library. The previous commands that used the library Image presented errors. The error was
that the library Image did not contain the desired commands. We remediated to that by
choosing “`import Tkinter`” rather. When we did that, we had, however, to use “`tkinter`” before
using a component from the library **Tkinter**. We had to do that in when making buttons, labels and windows.

```
49  root = tkinter.Tk()
50  root.configure(background='white')
```

**Figure 24 Code 10**

We start by creating the main window of our program, which we called “root” in line 49. In line 50, we configured the background of the image.

```
52  nb=tkinter.StringVar()
53  nb.set(" ")
```

**Figure 25 Code 11**

Then, we created a string variable called `nb`, which will be the label where the predicted number will appear. To create it, we did not directly use a string, but we used “*tkinter.StringVar()*”. That is due to the fact that we are going to need that string in that format later for our `predict` button. We initialize that label to an empty character so that it takes the space required for it from the beginning.

```
54  numberlbl = tkinter.Label(root, bg="white", text=nb.get(), font="none 200")
55  numberlbl.pack()
```

**Figure 26 Code 12**

After that, we set the label, or the bit of space, where we are going to display the predicted number. For that, we use the function `Button` (from the `tkinter` library again. We give it the following arguments. We first set the window in which it will be displayed. That is `root`. Then, we specify its background with the argument `bg="white"`. Above that, we need to
specify the text that will be displayed on the label. It is the text that we put earlier in the variable `nb`. Since `nb` is not an integer or a double, not even a string, but rather an object of resulting from the function `StringVar()`, we need to call the function `nb.get()` in order to get its value. Note that there is also a type obtained by the function `IntVar()`, but here we needed to first set the label as having a white space at first of font 200. This brings us to the font, which we select again in the arguments. We type `font="none 200"`. The `none` means that the text is neither bold nor italic nor underlined. And, of course, the `200` adjusts the size desired for the font. The line after that, `button.pack()` means that the label will be packed, or inserted automatically in the root window. We could have inserted different parameters to the function `pack()`, such as `side="left"` in order to insert the label at the left of the root window.
CONCLUSION

All in all, we were able to program a handwritten digit recognizing program, made using machine learning and neural networks, more precisely. We needed to make a program that, first, could as input the Modified National Institute of Science and Technology dataset, then use neural networks to learn from that dataset how to recognize handwritten digits, and then apply what it learned on a testing set, first, with a precision greater than 95%. Then, we were able to implement an interactive part of the program, which implies making the program read a number that would be input through a touchscreen.

That first part was, however, just as interesting as actually understanding in depth the algorithm and how it works. That means, first, understanding what makes up a neural network, starting from a simple model (the one neuron perceptron) and exploring a more complicated one (with more neurons, layers and activation functions). We were also able to explore the two steps of the neural network algorithm (feedforward neural network algorithm) we used: the forward propagation and backward propagation. In the forward propagation we learned about the activation function (Sigmoid, Softmax, Rectifier Linear Unit). As for the Backward propagation, we understood the loss function and the mathematical logic that makes it a good solution to solving our machine learning problem.

Finally, we were able to learn more about different libraries in Python. That includes Tensorflow for machine learning and Keras (contained in Tensorflow) for machine learning for image recognition. But it also includes the library Image that allowed us to convert and filter our own input image into one that can be read and used by our Neural Network. Finally, we learned about the Tkinter library, which is the standard graphical user library interface for Python.
REFERENCES


