DoodleRec - Foundations of AI and Image Recognition using Machine Learning.

Tuesday 18th July, 2023 - 12:58

Tinouert Alexandre
University of Luxembourg
Email: alexandre.tinouert.001@student.uni.lu

This report has been produced under the supervision of:
De Jesus Tiago
University of Luxembourg
Email: tiago.dejesus.001@student.uni.lu

Abstract

Nowadays, scientific and technical interest has emerged for the elaboration of modern artificial intelligence, and different algorithms have been developed. The purpose of this project is to see what is a neural network, what is machine learning, and how recognize images using this process, by developing an Artificial Intelligence capable of Deep Learning over huge amounts of data sets. Hence, machine learning became a popular domain in computer science, with a plethora of algorithms and applications made over decades, and is likely to gain even more traction in the future, knowing what trained algorithms can do today. This BSP focuses on the foundations of machine learning, and one of its applications in the domain of image recognition. The scientific deliverable proposes a model to follow according to a use case in order to answer to a scientific question and fulfill its requirements, while the technical deliverable handles the development of the web / back-end application, where other requirements also met. Thus, this report includes the explanation of the scientific and technical deliverable done for this first Bachelor Semester Project, from mathematical definitions of notions and models to technical developments, via interpretations of results.

1. Introduction

Since the wake of modern computing, scientists elaborated methods to use computers in order to answer problems that might not have a defined answer. Thus, training over large amounts of already defined data, meaning rewarding the algorithm if it guesses it right, and punishing it when wrong, emerged as the idea of Machine Learning.

Over the years, Machine Learning algorithms spread over many areas. For example, AlphaGo and DeepBlue from Google toppled the best players in go and chess, with an almost excellent accuracy for the best possible moves.

Even nowadays, the creation of algorithms capable of creating images from one sentence seemed impossible decades ago. For example, OpenAI released multiple AI, with capabalities ranging from constructive conversations and problem solving to creation of medias such as images, videos and face swapping. As such, the progression of Machine Learning is exponential, learning from even bigger data sets, and with the improvement of processors and computers, it is even faster.

Thus, knowing what Machine Learning is conceptually, this BSP revolves around what Machine Learning is mathematically and in the context of modern computer sciences. Moreover, to validate this mathematical model, the creation of an application, here performing image recognition is appropriate.

2. Project Description

2.1. Domains

2.1.1. Scientific

What are the foundations of Image Recognition using Machine Learning?

Almost everyone might describe AI as Machine Learning, where a program, using neural networks, gets data inputs in order to create patterns [3], in this case from images, that are similar and conform to what the program should case in order to detect objects, even though this schema applies to many domains [4]. But what are the foundations of Artificial Intelligence ? What might be some constraints and limits that might be encountered while experimenting on it ? Along this BSP, the topic of AI will be developed, first with basic examples on a tiny scale [2] and try to reach rudimentary limits of image recognition [5] and distinction between small doodles that will be pretty much similar (for example the Sun, Saturn, or a bowling bowl described mainly as spheres). It is expected that the program will be able to distinguish different sets of image with a decent accuracy [1]. By the end of this BSP, we will be able to correctly define what Machine Learning is [1] on a basic scale with an opening to foresee the development of AI onto a more advanced scale. Image Recognition will also be defined in this context [5] with some basic limits to it.

2.1.2. Technical

The program, written in Python using Numpy in order to grasps large amount of data sets and convert them into tables, as well as HTML and CSS as GUI for the interface, and, will take libraries of doodles as input. The data set must comprise both realistic and pictorial representations (for example, the Moon might be described as a crescent shape for the pictorial part, or a circle for the realistic representation). The user will then input its doodle on a relatively small canvas, 28x28 pixels only using black and white as color inputs. Since Flask is mainly used in order to link the Python back-end program with an HTML's GUI, JavaScript will also be needed in order to convert the user's doodle input on the website into an image data array usable by the main Python program. Using a trained neural network, the program will then process the database of images in order to detect patterns of possible similarities between the user's doodle and the database. The program will then output a list of similarities, and what the user's doodle might be according to the algorithm (i.e most likely to be a bowling bowl…). All in all, the main purpose of the program is to recognize doodles, and by decreasing the level of pictorial details (as for example all objects can be described as spheres), it is expected to get mildly surprising results regarding technical limits, especially considering details and the canvas' size.

2.2. Targeted Deliverables

2.2.1. Scientific

To answer to "What are the foundations of Image Recognition using Machine Learning", many boundaries are to be made. First of all, what is Image Recogntion, and what is Machine Learning. This suggests the creation of a global use case, where Image Recognition will be performed on images, in this case on black and white images. Next it is essential to define what Machine Learning is, with its mathematical definitions to see and computations to demonstrate. Finally, a link between images and Machine Learning has to be made. This will be the core of the Scientific Deliverable of this Bachelor Semester Project.

2.2.2. Technical

To assess how Machine Learning works on a large scale, creating an interface to train and to test Machine Learning seems crucial. First of all, it is needed to grasp large amount of image data to create a data set for training. Then, training must be dictated by the developed notions in the Scientific Deliverable with its concepts to translate to code. Finally, a general interface must be created, namely a website, to allow users to test the fully trained algorithm.

3. Pre-requisites

3.1. Scientific Pre-requisites

For this project, knowledge on Analysis and Algebra notions such as matrices, derivatives, as well as usual functions such as exponential are crucial to know in order to grasp the developed notions.

3.2. Technical Pre-requisites

For this project, medium to advanced knowledge on Python (Flask, Numpy), but also on HTML, CSS and JavaScript are needed to well pursue the development of DoodleRec.

4. Foundations of AI and Image Recognition using Machine Learning

4.1. Requirements

The main purpose of the scientific deliverable is to answer "How can Machine Learning compute Image Recognition?". Many subjects and sub-questions can be deduced from this main interrogation, as it would require to know how to analyse images and image data, then elaborate a mathematical model around Machine Learning, with definitions. Finally, interpretations on output are given, as computations need to be correctly contextualized in the topic of Image Recognition.

Namely, requirements will be ordered as such:

State data setting and cleaning
As images need to be usable for training, testing and so on, image data setting and cleaning need to create a set of images so that they can be used for general computations. As a result, image data needs to be in a compatibly mathematical form (regarding other requirements). This quite introductory requirement also serves in the interpretation part, though other definitions like data set shuffling are explained in this part. Finally, a general use case is also present, in order to concretely understand the overall input of this scientific deliverable.

Define Machine Learning conceptually and mathematically, with a defined model
Machine Learning is a magic bullet statement regarding overall definitions, with some ambiguity around it. This requirement leads to a conceptual and mathematical definition of the global process, with concrete examples related to the presented use case. All in all, this part defines key child concepts, as the already mentioned mathematical model, but also the training process with again, some examples.

Interpret the process and the output
Finally, a link has to be put between image data and Machine Learning, giving a contextual definition of Image Recognition regard Machine Learning. As such, elaborations of the process related to the aforementioned image data, thanks to the defined mathematical concept, an output can exist and overall be considered as valid, as long as it can be interpreted.

In order to answer the scientific question, which means creating, proving and computing mathematical entries and concepts, definitions must be stated to form an inventory of notions to use in the Production part.

4.2. Design

4.2.1. Overview

This part focuses on the inventory of notions that will be used in the Production section. Points from the Requirements section will be elaborated in order to create a pack of definitions, mathematical and contextual models to fulfill said requirements, namely definitions for data setting, cleaning and Machine Learning. Interpretations on the output and the general process are the main point of the final subsection of the Production part.

First and foremost, as for every function, inputs come first as how to categorize such functions. In this case, the inputs are mainly raw images, though a computer cannot directly interpret them, conventions are needed. [5]

4.2.2. Cleaning and setting up the data for the algorithm

Since raw images are inputted to the algorithm, ensuring that they can be read one way or another is vital for the process. In this case, they will be interpreted as matrices and arrays [3], especially since the main inputs are black and white (and gradient) images that are 28x28 pixels in dimension. RGB could be used, although sRGB can feature a contrast value which will simplify the interpretation. As such, inputs will use 784 entries arrays to represent images conveyed by their sRGB contrast pixel values, now knowing the algorithm can interpret them. In order to prepare for the next step, setting up the set by randomizing the entries of labeled images randomly ordered is critical, as the algorithm might calibrate itself on only one type of images it is too redundant. From now on, definitions on the concepts of Artificial Intelligence, Neural Networks and Deep Learning will be expressed.

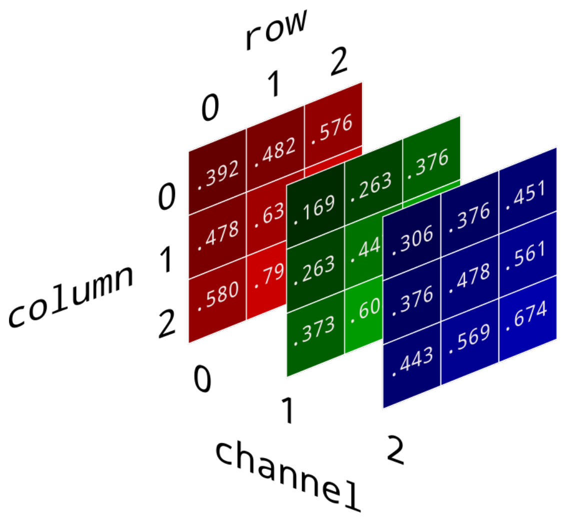


Figure 0 - Example of RGB components of an image by e2eml.school

4.2.3. Definitions for Machine Learning

In order to make a program that can "learn" with some amount of inputs, many modules, components and notions have to be seen.

First, a perceptron [1] is a node structure composed of a simple input layer of neurons, linked with an output value with weight, under the resolution of an activation function.

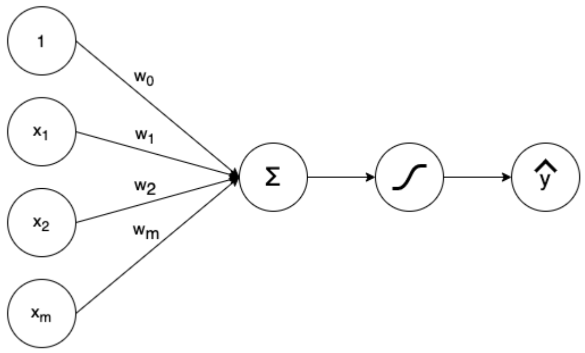


Figure 1 - Perceptron, provided by makshay.com

For example, neurons (denoted by $\{1,\ x_1,\ ...,\ x_m\}$ on the picture above) may have a higher input if a pixel is darker, lower if a pixel is lighter and so on and so forth [5]. Those values are multiplied by their weights ($\{w_0,\ ...,\ w_m\}$) and summed as a weighted sum (here, denoted as $\sum$). As such:

$$\sum = \sum_{i=0}^{m} x_i * w_i \tag{1}$$

Finally, in order to normalize the weighted sum into an average kind of value, this weighted sum value will pass into an activation function (denoted above as $\int$). There exists several kind of activation functions, but for now, the process will use one of the most famous activation function, which is the Sigmoïd function [1] [2] [3]. This function will "normalize" the entries in a range from 0 to 1, which will be more useful to manipulate later:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} = 1 - S(-x) \tag{2}$$

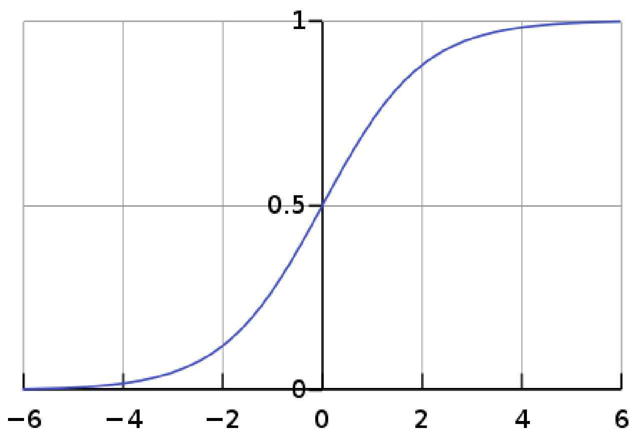


Figure 3 - Graph of the Sigmoïd function, provided by Wikipedia

At this point, values are normalized to correspond to an output value ($\hat{y}$) ranging from 0 to 1. This schema of Perceptron, which has **one input layer** and **one output layer** can be replicated, according to the use cases. Many layers can be inbetween those input and output layers, with different activation functions and weights, that are called **hidden layers**. This will allow the algorithm to distribute computations throughout several tasks and values, in order to define more properties that are to be distinguished, which will be the image patterns for later on. Here is an example of a multi-layer Perceptron, or Neural Network [1] [3]:

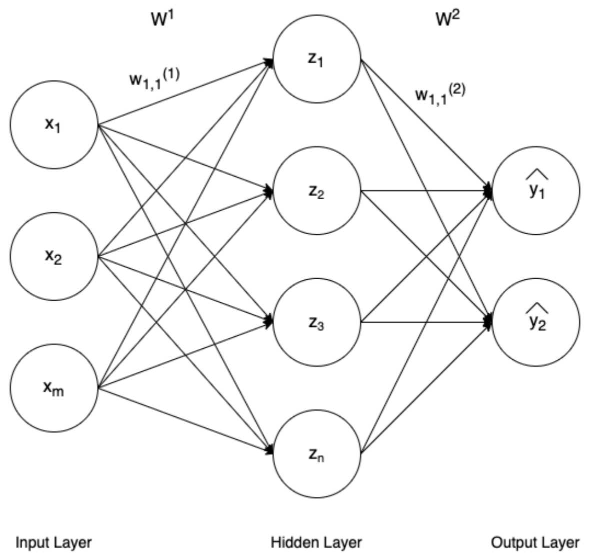


Figure 2 - Multi-layer Perceptron, provided by makshay.com

Now that the basic structure of a Neural Network is covered, Some adaptation to the use case of Image Recognition will be made.

4.3. Production

Thanks to the mathematical notions defined in the Design section, which consists of what are the main inputs and notions that can be used to construct mathematical models to answer to the scientific question. Recall that the main purpose of this Production section is to define the process of Machine Learning in the context of Image Recognition. Hence, by this point, this means computing a general model and interpret the global output and process. First of all, inputs must be written mathematically:

4.3.1. Image data - Input

As defined in the Design section, the raw images can be interpreted as RGB array by computers. An image would thus resemble such an array:

$$Let\ Image = [(r_1, g_1, b_1), (r_2, g_2, b_2)..., (r_n, g_n, b_n)] \quad (3)$$

With every $r_k, g_k, b_k \in [0, 255]$ $(k \in Z)$ or in [0, 1] according to the convention (see Figure 0).

However, for the sake of simplicity and compatibility, every value will be comprised in [0, 255] as it is easier to represent contextually, being more widespread than [0, 1].

Moreover, recall that using the sRGB colorspace adds an alpha value. In this case:

$$\begin{aligned} Let\ Image = [(r_1, g_1, b_1, \alpha_1), (r_2, g_2, b_2, \alpha_2)...,\\ (r_n, g_n, b_n, \alpha_n)] \end{aligned} \quad (4)$$

This time with all $\alpha_k \in [0, 1],\ \forall k \in Z$.
Considering that the process only uses black and white images, the alpha value acts as a contrast value, with 0 being white and 1 being black on a grey scale. Hence, Image can be reduced to:

$$Image = [label, \alpha_1, \alpha_2, ...\ \alpha_{784}] \quad (5)$$

with label as an integer refering to an arbitrary value pointing to a name. As images are defined to be 28x28, hence with 784 pixels, the Image array counts from row to row, meaning that α_{28} refers to the last pixel of the first row while α_{29} refers to the first pixel of the second row for example.

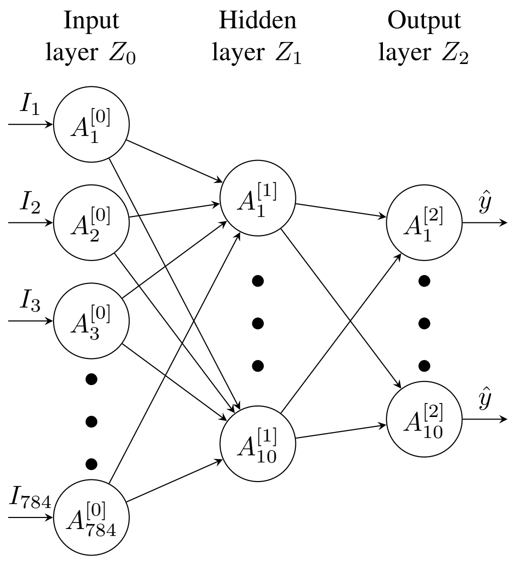
Having a sufficiently large, at least thousands of images in this form, shuffling the entire set of images to create a data set for the Neural Network to train on is needed. As such:

$$\begin{aligned} Let\ Data = [Image_1,\ ...,\ Image_n]\\ Let\ Labels = [label_1,\ ...,\ label_n] \end{aligned} \quad (6)$$

respectively the set of Images with their labels stripped and the set of respective labels. Having identified the inputs of the process, it is now possible to produce the mathematical model to train a Neural Network.

4.3.2. Training - Forward Propagation

The Neural Network's structure used for this situation is as follows:



with all $I_k,\ k \in Z$ being the pixel values from the image ($I_1 = \alpha_1$ for an image, and so on). Note that between every layer are series of weights, though computations can be explained as such:

$$\begin{aligned} A^{[0]} &= I(i_i), A_{784,1}\ represents\ an\ image \\ Let\ Z^{[1]} &= A^{[0]} * W^{[1]} + B^{[1]} \\ Let\ A^{[1]} &= Sig(Z^{[1]}) \end{aligned} \tag{7}$$

This first propagation from the input layer to the hidden layer features the use of the first activation function, already mentionned as example in equation (2) in the Design section. Here, $Z^{[1]}$ is the matrix containing the values of the of the $A^{[0]}$ neurons, multiplied by their assigned weights ($W^{[1]}$) and summed with a bias matrix ($B^{[1]}$) in order to replicate a linear function for each entry. Then, the Sigmoïd function is applied to the summed matrix and computations continues onto the next layer for the new input matrix $^{[1]}$ [1] [3].

Given the values of the hidden layer, for the output layer, let:

$$Z^{[2]} = A^{[1]} * W^{[2]} + B^{[2]} \tag{8}$$

For this step, images are to be differentiated as per 10 different states. As such, several patterns have to be found, but as an output, it is expected to have a list of possibilities on whether or not an image has patterns that would make the image labelled, as it resemble other images for instance. Without a loss of generality, and without digression, a more probabilistic approach has to be envisioned rather than a normalization one on this step (this image is 90% this object). Having such approach will lead computations to a single matrix where the sum of all entries will be equal to 1, and where all entries will correspond to probabilities. And for such transition, the Sigmoïd function will be discarded, like any conventional function, and a Softmax activation function [1] [2] [4] would be more appropriated on a set as follows:

$$\begin{aligned} Softmax(\mathbf{z})_j &= \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \\ Let\ A^{[2]} &= Softmax(Z^{[2]}) \end{aligned} \tag{9}$$

Where $Z^{[2]}$ has all the wanted properties.

However, this is only understanding how to compute the inputs to outputs, as it is now expected to train the program to a larger scale.

4.3.3. Training - Backward Propagation

Since the Neural Network will be initialized with random values, those need to be calibrated to correspond to something more appropriate, thanks to tests and training.
At the end of the first computation (let's say with an image labeled "A"), $Z^{[2]}$ is expected with all probabilities of what the image is. For example, $Z^{[2]}$ could say that the image can be categorized as "A". Thus, let us encode this certainty, or guess in $Z^{[2]}$ in place of label A, and rewind calculations, this time with derivatives of functions, generally called "loss functions" [1] that will apply to weights and biases in order to see how much they have contributed to the error [1] [3].

This step begins by taking the probabilities and put them through a OneHot encoding process. In this use case, the OneHot encoding process is behaves as a function to transform an array into a weighted matrix that is compatible with the process. To put it in perspective of this use case, the output array, being a matrix of size 10×1 needs to be transformed back into a 784×1 matrix to be computed. Hence, values need to be normalized between 0 and 1 again, as so that the loss function will be able to turn the values back into image-like pixel values. Thus:

$$\begin{aligned} &OneHot([1,2,3,4,5,6,7,8,9,784]) \\ &= [1,1,1,1,1,1,1,1,1,0,...,1]\ of\ size\ 784 \times 1 \end{aligned} \tag{10}$$

By applying this technique:

$$\begin{aligned} dZ^{[2]} &= A^{[2]} - Y \\ dW^{[2]} &= \frac{1}{m} * dZ^{[2]} A^{[1]T} \end{aligned} \tag{11}$$

Y being the weighted label (OneHot(Output) - label). This process continues so on and so forth for each output value pegged to the corresponding weights and biases. It is however needed to use the derivative of the Sigmoïd function where Sigmoïd has been used before, now as a "loss function". That is:

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * Sig'(Z^{[1]}) \tag{12}$$

Finally, weights and biases have to be redefined to be more accurate, with a learning rate (r) defined to be arbitrarily small (since the Sigmoïd function is ranging from 0 to 1 only) that will be seen in details in the Technical Deliverable section:

$$\begin{aligned} W^{[1]} &:= W^{[1]} - r * dW^{[1]} \\ B^{[1]} &:= B^{[1]} - r * dB^{[1]} \\ W^{[2]} &:= W^{[2]} - r * dW^{[2]} \\ B^{[2]} &:= B^{[2]} - r * dB^{[2]} \end{aligned} \tag{13}$$

4.3.4. Training - Machine Learning

Now, the training continues on the rest of the randomly shuffled selected data set, as it is now possible to interpret what the output is.
For now, **Machine Learning** is to take a large amount of labelled input data, pass it into a Neural Network with weights and biases and compute it thanks to the activation functions between every layer to obtain outputs, which is the process of **Forward Propagation**. Then, those outputs are compared with the default label, in order to update the weights and biases so that corresponding neurons will not be activated by the same weights and biases for a wrong output. Note that if the output is correct, no modification is done to the weights and biases. This is the **Backward Propagation** step. To put it in a larger scale, and in order to calibrate the weights and biases to the set of data, the steps of Forward and Backward Propagation are repeated over and over again on every input of all the data set, for thousands of iterations to refine the weights and biases as precisely as possible. Nonetheless, the defined model allows to fulfill the set requirement concerning Machine Learning and is a perfect fit for the incoming Technical Delivery.

4.3.5. Interpretation, Results and Conclusion

To quickly resume on the image set, the process of Machine Learning over the image set imposes that the data set must be randomly shuffled in order to avoid discrepancy. Namely, as Machine Learning goes on the entire set, if the images are well ordered by labels, and sufficiently large enough, the Neural Network's weights and biases might calibrate too much on the last subset of labeled images, since weights and biases used for detecting other images might be overlapping those used to detect the last subset of labeled images.
Finally, concerning images, to put it in a nutshell, images are conceptually what a user see, and mathematically, at least for a computer, an array, in this case a matrix of values, here filled with contrast values oscillating between 0 for black and 1 for white, with a whole data set of such images used as **input**.

Secondly, Machine Learning relies on the use of activation functions over summed values made from input times their attributed weights, plus their biases, over every layer to obtain an output, and of loss functions over the guess, weights and biases to measure the error and update the weights and biases.

Thirdly, a the obtained output at the end of a Forward Propagation process is actually a label, which corresponds to the index of the maximum found in the output. Put it more concretely, since values are at this point and thereon normalized between 0 and 1, such an output could be:

$$Out = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] \tag{14}$$

Here, "Out" has a 1 at index 3 (or 2 if calculated by a computer) and corresponds to the third category that has been inputted as "to guess". In the same way, here what could a name array look like:

$$\begin{aligned} name = [&"Airplane", "Cookie", "Cat, "Cup", "Hat", \\ &"Snowman", "Star", "StopSign", \\ &"Television", "Tree"] \end{aligned} \tag{15}$$

Which is the name array used in the Technical Deliverable section. In this case, as 1 is at index 3, the image would refer to a "Cat" according to the trained Neural Network if Forward Propagation is performed.

Finally, to promptly answer to the question thanks to this developed section, Image Recognition can easily rely on Machine Network if images are numerized as arrays of pixel data if said steps are applied over and over to make an algorithm calibrate its weights and biases to what is expected for the algorithm to guess [5]. Note that Machine Learning is gradual, and reaching one hundred percent of accuracy of guessing is nonetheless theoretical.

As food for thoughts: tree-based graphs can thus be elaborated to understand how a machine classify an image per step [2] [4].

4.4. Assessment

With the Production part complete, thanks to the developed definitions in the Design sections, many points have been developed that properly meet the requirements

First of all, images have correctly been defined and can be interpreted by computers using colorspaces. Furthermore, black and white images particularly rely on contrast values thanks to sRGB and the associated alpha value. Images can be put in data set and randomly shuffled.

Next, the core of Machine Learning has correctly been defined, with the structure of a Neural Network, definitions of values like weights and biases, functions like activation and loss functions, but also overall computations around matrices that make Forward and Backward Propagation to either guess an image, or refine the weights and biases.

Finally, an interpretation over the input, consisting of a matrix of images, and over the output, being an array of a leading indexed label. The overall process of Machine Learning has been put into the context of Image Recognition, where it is now easy to understand the computations.

To conclude, the computations and points developed previously, satisfying the requirements, can now be accepted as definitions, as they are key components to the Scientific Deliverable, which consists of a software using this Scientific Deliverable as structure.

5. DoodleRec - A Web Application and a Testing/Training Pack

5.1. Requirements

The main technical deliverable consists of building a web application, ordered around a main back-end program that can process arrays that represent images, of contrast values, thus black and white images. Rendering a website is also a priority, with full on integrated front-end and back-end application, from which a user can input a doodle and where the program can retrieve data to process, and display results to the user. This means the user can input a doodle on a canvas, a front-end application, in JavaScript will retrieve the image data and send it to a back-end application to analyze and return an output result.

Aside with this web application comes other programs needed to train a Neural Network thanks to a data set of images, but also create a compatible data set from raw array data set, meaning shuffling and labeling.

All in all, those programs can interact with each other by creating, reading and computing data and allow a synergy by initializing a data set, training, testing and using the Neural Network.

5.1.1. Functional Requirements

Create a usable data set for training

In the initial inventory, the process begins with a large database of arrays representing images. However, this huge data set may not fit the requirements for training, as it needs to be randomly shuffled and labeled. The first task of this Technical Delivery is to scale and transform the data set by creating a new one with labeled and shuffled data coming from the initial bigger set.

Training and Testing the Neural Network

Next, as mentioned in the Scientific Deliverable, the Neural Network shall be trained thanks to the defined notions that will be transcribed, like the Neural Network's structure, activation and loss functions and so on. Then, the program must loop on every image in the data set, and train, hence update the weights and biases thanks to the developed step in the Scientific Deliverable. Finally, a secondary program will perform Forward Propagation, by using the final weights and biases on random images from the data set to naively assess its viability.

Share, fetch and send data between the Training Pack and the Web Application

This step accounts for sharing values like weights and biases values between the Training Pack (what composes the two aforementioned requirements) and the Web Application, as those will be used for the Web Application on its own.

Create a Web Application for drawing and processing doodles

Thanks to the training of the Neural Network, what remains is to create a web application capable of rendering doodles by interacting with, allowing the user to draw doodles. Then, the program will perform Forward Propagation on it and return a result.

5.2. Design

The main design decisions are listed below:

5.2.1. Back-end and Front-end

For this project, Python (Flask) has been used as a back-end program in order to fetch and analyze the inputted data, as having more experience in Python is better for this program. JavaScript is useful for the creation of a ”canvas” (input zone for doodles) and retrieving the image data and sending it is easier in that case.

5.2.2. Training Pack

Among this deliverable, a training pack is available to train a Neural Network if some compatible data is available in the correct directory. The training pack is included as one of the programs in the technical deliverable to develop the process of Machine Learning. It is not accessible with the web application's interface, but with an IDLE with the appropriate Python environment. and serves as a ”developer pack” in contrast with the web application, easier to use.

The training data set is composed of 10 .npy files provided by Google's Quick Draw data set (Airplane, Cup, Cookie, Cat, Hat, Snowman, Star, Stop sign, Television and Tree) [6].

5.2.3. Libraries

For Python, different libraries are used to implement and develop DoodleRec:

- **Flask** is used to run the Web application. It is a Python based library running on HTML templates allowing the creation of dynamic websites.
- **Numpy**'s methods are useful to perform matrix operations using its own special types. Thus, its various methods and functions are mainly used to not reinvent functions computing such operations (dot products, functions on entire matrices and so on). Note that the ”raw” set of data comes in .npy files, where Numpy is also used to read them.
- **Pandas**'s CSV method allows the creation and deletion of .csv files which constitutes the cleaned shuffled data

set. Pandas is thus used to create the data set, but also to read it when training the Neural Network and also to get the size of the data set as matrix operations are performed on it, hence matrices must be compatible by taking the size of the data set.
- **Matplotlib** is only used in the testing program to display images by taking arrays.
- **os** is used to manipulate files and directories by getting paths, and sharing it with other functions and libraries to modify said files.
- **json** is used read and create JSON data used to be parsed and fetched between Python and JavaScript.
- **random** is used in the data setting and testing programs to respectively select random images to create a data set and to select other images to test the Neural Network on those.

For JavaScript, no extension, module or special library are used.

5.2.4. Special Types

Regarding Python, and especially Numpy, different special types are used regarding matrix multiplications:

- **numpy.ndarray** is used in order to facilitate operations such as dot products, matrix multiplications, transpose and reshapes. It can be retransformed into a normal type list with .tolist() on Python. numpy.ndarray(s) will thereon be referred to as "Numpy arrays".
- **JSON**, namely JavaScript Object Notation is used as dynamic data for parsing and fetching. It consists of dictionary like data structure, mainly used in JavaScript but compatible with Python, here serving as data to communicate between Python and JavaScript when running the web application live. Other static data like the image data set and Neural Network values use respectibely .csv and .txt files.

5.2.5. User Interface and Experience

(Please, see the image of the software in the appendix below). For the user experience, CSS will be used to glaze the Graphical User Interface in order to be nicer to the user's eyes. The web application is a simple page composed of essential elements:

- A canvas to input doodles.
- A Description in the right hand size, that can be modified with JavaScript.
- A button to retrieve data in a .txt file containing the image data and the eventual guess by the application.

5.2.6. Functional Decomposition

Here are the main files used for this project's software:

Web Application's files:

- **main.py** which is the main back-end Python file, containing the main back-end script for receiving data, retrieving it and analyzing it.
- **script.js** which is the main front-end JavaScript front-end program, that develops a canvas, but also sends the image data every second as JSON POST requests. It is among the statics recognized by the Python Flask application.
- **index.html** is the main HTML template used by the Flask application to render the main web page. It contains Jinja to change dynamically the website to the willing of the main.py application.
- **style.css** is the static used to render the page with a modern look.
- **yourmatrix.txt** is the user's matrix data file that is download every time a user request it via the site and the Flask application.
- **nn.txt** is the text file where the Neural Network's Weights and Biases values are stored. It is written by training.py and read by main.py.
- **todisp.txt** is the text file where the program's guess is located, waiting to be read by JavaScript.

Training Pack's files:

- **setdata.py** prepares a random set of data from .npy files (which are Numpy arrays) and writes it into a .csv file to be used for training.
- **doc.py** is the main file containing the functions required to train a set of data from a .csv file and returns matrixes of weights and biases calibrated from the training.
- **training.py** is the file to run in order to process training. It takes doc.py functions and prompts the user for a Learning Rate, and a number of Iterations.
- **test.py** allows the user to test the trained Neural Network on data.csv and naively assess its performance.

5.2.7. Data Structures for Training

This subsection will focus on the data structures (files and variables) used specifically for training, since the structures used for the web application are JSON arrays.

- **Images** for the training data set come as .npy files which are files of Numpy arrays. This type of array has been used in this training program to keep the whole computation homogeneous, having easier calculations processes from Numpy. They can be reconstructed as images using the arrays' values which represent the contrast value (0 being white and 255 being black).
- **Weights and Biases** are also Numpy arrays. Again, calculations like the dot product for a matrix, as well as the argmax are simplified in order to avoid reinventing the wheel.

5.3. Production

5.3.1. Functional Details:

This part focuses on how does a file works (with its functions) and how does it interact with other file. For the files and folders layout, please refer to this subsection of the

Appendix. 8
3

Web Application's files:

- **main.py** imports functions from doc.py
  - **Compute()** for route "/" is executed every second. It then writes in "yourmatrix.txt" a matrix version of the user's input fetched by JSON POST requests. It then takes, from "nn.txt" (using **Params()**, imported from **doc.py**) the weights and biases values to perform **ForwardPropagation** (again imported from **doc.py**) on the Numpy array version of the user's input, and prints out a guess. It also renders "index.html" as a template.
  - **download_data()** for route "/download" is executed every time the user clicks on "Retrieve your Data!". It downloads "yourmatrix.txt" from which "Compute()" has written the user's data on.
- **script.js**
  - **Canvas Creation**, though not directly a function, defines a Canvas display area as well as a Contextual input area. It defines functions like **"startPos()", "finPos()", "draw(e)"** that takes into input the coordinates of the mouse (thanks to event listeners) on the Contextual input area, and transcribes it in order to color in white where the user has clicked.
  - **setInterval(function()...)** is executed every second. It takes the image data (**"ctx.getImageData(0,0,28,28)"** which is an array with 4 values for every pixel, as sRGB is applied. It then reduces the image data array to every 4th values (contrast values for each pixel). Finally, it creates a JSON POST request and sends the stringified image data.
- **index.html** is the main template, used by Flask. It makes the link with script.js and style.css.

Training Pack's files:

- **setdata.py** Asserts first for the user's prompts. It then takes, for the number of iterations, a random array in the .npy files (with labels, using for loops) and plug it into data.csv.
- **doc.py**
  - **Prompt()** prompts the user for a Learning Rate and a number of Iterations.
  - **Parameters()** sets random weights and biases.
  - **Sigmoid(Matrix)** uses Sigmoid on an entire Numpy array (please refer to the code in the Appendix, or Scientific Deliverable).
  - **SigmoidDerivate(Matrix)** applies the derivative of Sigmoid on a Numpy array.
  - **Softmax(Matrix)** computes Softmax on a Numpy Array.
  - **OneHot(Matrix)** applies a OneHot encoding process to a Numpy array.
  - **ForwardPropagation** takes a Numpy arrays, multiplies it with the first set of weights and sum it with biases, then applies **Sigmoid(Matrix)** on it, and repeats the process with the second set of weights and biases and uses the **Softmax(Matrix)** function on it.
  - **BackwardPropagation** applies OneHot on a Numpy array, creates a set of weights and biases to update, then does the same with the **DerivativeSigmoid** function for another set of weights and biases.
  - **NewParameters** updates the regular weights and biases by substracting their corresponding values with the product of the weights and biases by the learning rate (W1 = W1 - Rate * derW1) created by **BackwardPropagation** (derW1,...).
  - **Prediction** returns the argmax of a Numpy array, which is its largest entry.
  - **Accuracy** returns the ratio of similarities of the entries between two Numpy arrays.
  - **LowOnCurve** initializes random parameters with **Parameters()**, performs **ForwardPropagation, BackwardPropagation, NewParameters** as many times as the user has requested (using Iteration, prompted at **Prompt()**) and occasionally prints the accuracy and predictions thanks to the two aforementioned functions.
  - **Params()** opens "nn.txt", and for each line (respectively weights and biases), it transforms the stringed arrays into regular arrays json.loads(), then transform them into Numpy arrays.
- **training.py** uses all functions from **doc.py** to train the Neural Network. It then writes the weights and biases values into nn.txt.
- **test.py** uses ForwardPropagation (and all its child functions) from **doc.py** to test on data.csv. It takes random images from it and applies the functions, then returns a result and displays the image to the user.

5.3.2. Proper Highlights

As the deliverable is divided into many source files, here are some notable highlights.

- **Sigmoid Function, Derivative Sigmoid Function, Softmax Function and OneHot Function** in doc.py:

```
#Sigmoid function for an entire matrix
def Sigmoid(Matrix):
    return 1 / (1 + np.exp(-Matrix))

#Sigmoid derivative function for an
    entire matrix
def SigmoidDerivative(Matrix):
    Sigma = Sigmoid(Matrix)
    return Sigma * (1 - Sigma)

#Softmax function for an entire matrix
def Softmax(Matrix):
    A = np.exp(Matrix) /
        sum(np.exp(Matrix))
    return A

#OneHot function for an entire matrix
def OneHot(Matrix):
    OneHotMatrix = np.zeros((Matrix.size,
        Matrix.max() + 1))
    OneHotMatrix[np.arange(Matrix.size),
        Matrix] = 1
    OneHotMatrix = OneHotMatrix.T
    return OneHotMatrix
```

Those functions are mainly used in steps for Machine Learning, namely Forward and Backward propagation. Notice how Numpy allows the program to be developed synthetically, as it is not needed to write additional lines of code that would cover what is already accessible, especially regarding Numpy arrays and elementary operations on matrices (.zeros, .exp, .arange, .T) since a vanilla Python code would have to account those operations for each element in the array.

- **Parsing and Fetching** data from script.js to main.py. **script.js**:

```
//Every 1s, this function takes the image
    data of the input image and returns an
    array of it
//It then parses it to a JSON request,
    that will be fetched by the Flask
    application
setInterval(function(){
    imgData = ctx.getImageData(0,0,28,28);
    imgDataBW = [];
    imgDataBW.push(imgData.data[3]);
    for (var i = 7; i <
        imgData.data.length + 1; i = i + 4)
        {
       imgDataBW.push(imgData.data[i]);
    }

    var sendData = {
       "userinput" : imgDataBW
       }
         fetch(`${window.origin}/`, {
          method: "POST",
          credentials: "include",
          body: JSON.stringify(sendData),
          cache: "no-cache",
          headers: new Headers({
            "content-type":
                "application/json"
          })
         })

}, 1000);
```

main.py

```
#Receive the data from the JavaScript
    that arrives every second (1000ms)
@app.route("/", methods = ["post"])
def printer():
    userdata = request.get_json()
    userinput = userdata['userinput']

    #Open the "yourmatrix.txt" file and
        write an introductory line
    f = open("yourmatrix.txt", "w")
    f.write("Here is the image data of your
        doodle, with values ranging from 0
        to 255 from the least contrasted to
        most contrasted parts of your
        doodle." + 2*"\n")

    #Transform the parsed data by writing
        the proper usermatrix and
        yourmatrix.txt file in a matrix
        format
    for i in range(0, 28):
     for j in range(0, 28):
      usermatrix[i][j] = userinput[28*i+j]
     f.write(str(usermatrix[i]) + "\n")
    print(usermatrix)

    #Close the .txt and return
    f.close()
    return usermatrix
```

Here, JSON POST requests are used to parse and fetch data. Having a continuous flow of data every second allows the program (main.py) to run a function every second, where here the user's data is transferred to a matrix and is written in a .txt file.

To conclude, knowing the pattern of Machine Learning described in the Scientific Deliverable part, then implementing it technically is straightforward. The use of libraries is targeted to avoid wasting time on essential functions. The implementation of a data selection/parsing/fetching system is also based on libraries, this time to interact with the different file extensions (.csv, .txt).

5.4. Assessment

Every described file have correctly been implemented and fulfill the developed technical requirements.

Namely, the creation of different programs allowing the cleaning and shuffling of the initial data set is a success, with setdata.py functioning correctly and outputting data.csv with a correct size, and shuffled images.

Secondly, the elaboration of a Machine Learning program that permits the training of a neural network over the said data set is also a success, with doc.py, training.py and nn.txt resulting from it. Notice that notions from the scientific deliverable section have been used to correctly program the functions.

Thirdly, thanks to script.js, the web application is able to retrieve data from nn.txt, allowing it to setup the computations for the live website.

Finally, the website allows the user to draw doodles, thanks from script.js creating a canvas, and analyze it to retrieve a result to the user thanks to main.py. The only inconvenience resides in the difference of image compression and rendering between the original data set and the website (anti-aliasing differences). As such, although the program can guess correctly 80% of images in the data set, those imperfections can be seen in the website, as its performances downgrades a bit, with more inconsistencies. Nonetheless, the software correctly fullfil its requirements.

6. Conclusion

This section concludes this Bachelor Semester Project, which had as objectives the elaboration of a Machine Learning model regarding Image Recognition and the creation of a software, comprised with a web application and a training pack allowing the creation of a Machine Learning program. The model developed in the Scientific Deliverable is a success, as for this use case, for black and white images, it is possible to use the mentioned computations to describe the concept of Machine Learning conceptually and mathematically, and to elaborate a Technical Deliverable on this said use case. The software developed in the Technical Deliverable is also a success as it allows a user to create a data set, to train a Neural Network, and to test it either on the data set or on a website, despite image rendering being different between the input dataset and the drawings made on the website.

6.1. Acknowledgment

The author would like to thank his Project Academic Tutor De Jesus Tiago for their assistance, their helpful continuous feedback and general suggestions for improvement. Lastly, the author would also like to thank the BiCS man- agement and the rest of the education team for creating and enhancing the Bachelor Semester Project concept on a continuing basis while providing the most useful resources

7. Plagiarism statement

I declare that I am aware of the following facts:

- As a student at the University of Luxembourg I must respect the rules of intellectual honesty, in particular not to resort to plagiarism, fraud or any other method that is illegal or contrary to scientific integrity.
- My report will be checked for plagiarism and if the plagiarism check is positive, an internal procedure will be started by my tutor. I am advised to request a precheck by my tutor to avoid any issue.
- As declared in the assessment procedure of the University of Luxembourg, plagiarism is committed whenever the source of information used in an assignment, research report, paper or otherwise published/circulated piece of work is not properly acknowledged. In other words, plagiarism is the passing off as one's own the words, ideas or work of another person, without attribution to the author. The omission of such proper acknowledgement amounts to claiming authorship for the work of another person. Plagiarism is committed regardless of the language of the original work used. Plagiarism can be deliberate or accidental. Instances of plagiarism include, but are not limited to:
  1) Not putting quotation marks around a quote from another person's work
  2) Pretending to paraphrase while in fact quoting
  3) Citing incorrectly or incompletely
  4) Failing to cite the source of a quoted or paraphrased work
  5) Copying/reproducing sections of another person's work without acknowledging the source
  6) Paraphrasing another person's work without acknowledging the source
  7) Having another person write/author a work for oneself and submitting/publishing it (with permission, with or without compensation) in one's own name ('ghost-writing')
  8) Using another person's unpublished work without attribution and permission ('stealing')
  9) Presenting a piece of work as one's own that contains a high proportion of quoted/copied or paraphrased text (images, graphs, etc.), even if adequately referenced

  Auto- or self-plagiarism, that is the reproduction of (portions of a) text previously written by the author without citing that text, i.e. passing previously authored text as new, may be regarded as fraud if deemed sufficiently severe.

References

[1] Müller, A. C., Guido, S. (2016). Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.".
Most complete introductory book with definitions, use cases on various libraries as well as practicals and concepts.
Available here

[2] Bonaccorso, G. (2017). Machine learning algorithms. Packt Publishing Ltd.
Containing synthetic examples of Bayesian representations and theories, and linear regressions, related to how a neural network uses its weight to analyze data.
Available here

[3] RASCHKA, Sebastian. Python Machine Learning. Packt publishing ltd, 2015.
Insights on machine learning related to Flask web, as well as how to train such a network with different types of labeled and unlabeled data sets.
Available here

[4] Surdeanu, M. (2020). A Gentle Introduction to Deep Learning for Natural Language Processing. Google Scholar, 1-63.
In order to define what can be grasped on machine learning, different examples can be used to understand how it works, and how it acts in this case.
Available here

[5] Fujiyoshi, H., Hirakawa, T., Yamashita, T. (2019). Deep learning-based image recognition for autonomous driving. IATSS research, 43(4), 244-252.
This book features practicals insight on how to do image recognition, and tracking how does a network groups and classify data with images in that case.
Available here

[6] Halfdas J. Jongejan. Google Creative Lab, The Quick, Draw! Dataset. Adjusted on June 12 2022.
Training dataset used in the Technical Deliverable.
Available here

8. Appendix

8.1. Mockup and Software

`(Pre-input)`

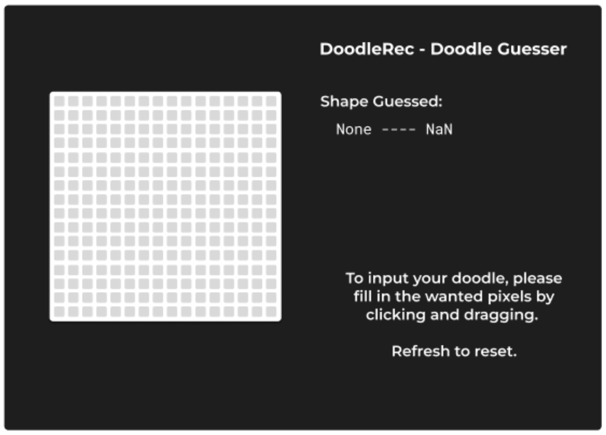

DoodleRec - Doodle Guesser
Shape Guessed:
None ---- NaN
To input your doodle, please fill in the wanted pixels by clicking and dragging.
Refresh to reset.

Figure 1 - Figma mockup of the software, blank of inputs, as when it is loaded.

`(Post-input)`

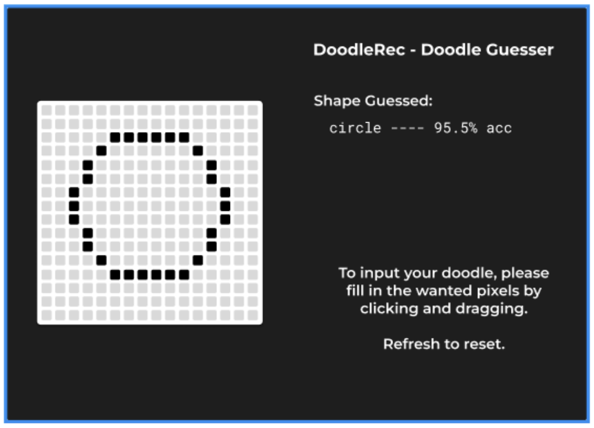

DoodleRec - Doodle Guesser
Shape Guessed:
circle ---- 95.5% acc
To input your doodle, please fill in the wanted pixels by clicking and dragging.
Refresh to reset.

Figure 2 - Figma mockup of the software, the user has chosen to input what resembles to a circle. This applies to every other shape.

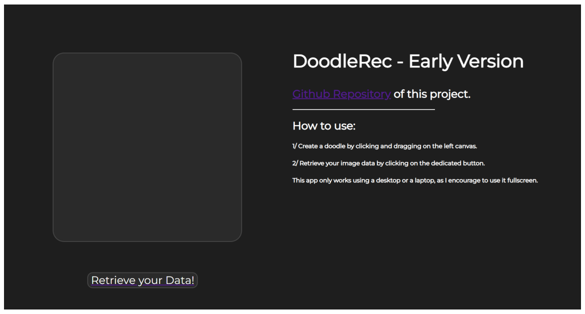

DoodleRec - Early Version
Github Repository of this project.
How to use:
1/ Create a doodle by clicking and dragging on the left canvas.
2/ Retrieve your image data by clicking on the dedicated button.
This app only works using a desktop or a laptop, as I encourage to use it fullscreen.
Retrieve your Data!

Figure 3 - DoodleRec - Early Version software, usable as is. Image taken on the 30th of October, 2022.

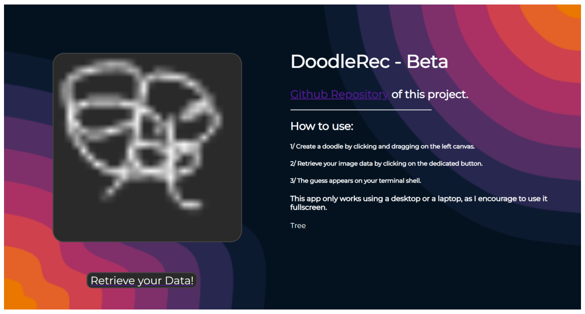
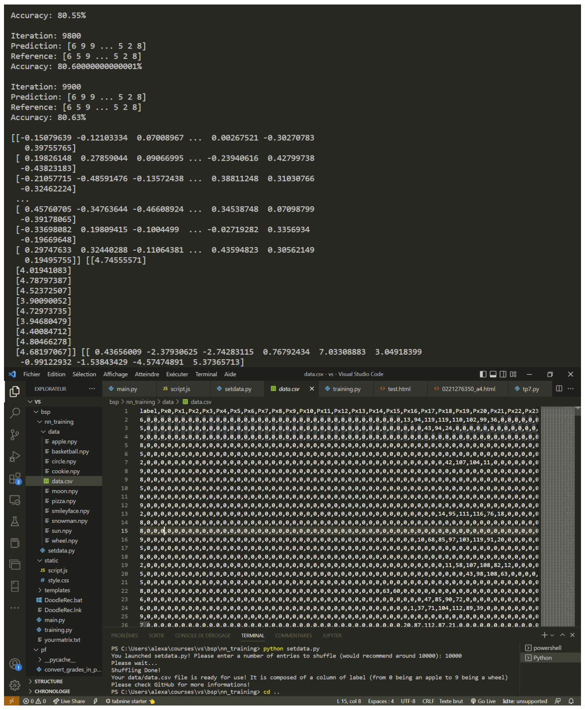

DoodleRec - Beta
Github Repository of this project.
How to use:
1/ Create a doodle by clicking and dragging on the left canvas.
2/ Retrieve your image data by clicking on the dedicated button.
3/ The guess appears on your terminal shell.
This app only works using a desktop or a laptop, as I encourage to use it fullscreen.
Tree
Retrieve your Data!

Figure 4 - DoodleRec - Beta software, with input and guess. Image taken on the 20th of December, 2022.



```
Accuracy: 80.55%

Iteration: 9800
Prediction: [6 9 9 ... 5 2 8]
Reference: [6 5 9 ... 5 2 8]
Accuracy: 80.60000000000001%

Iteration: 9900
Prediction: [6 9 9 ... 5 2 8]
Reference: [6 5 9 ... 5 2 8]
Accuracy: 80.63%

[[-0.15079639 -0.12103334  0.07008967 ...  0.00267521 -0.30270783
   0.39755765]
 [ 0.19826148  0.27859044  0.09066995 ... -0.23940616  0.42799738
  -0.43823183]
 [-0.21057715 -0.48591476 -0.13572438 ...  0.38811248  0.31030766
  -0.32462224]
 ...
 [ 0.45760705 -0.34763644 -0.46608924 ...  0.34538748  0.07098799
  -0.39178065]
 [-0.33698082  0.19809415 -0.1004499  ... -0.02719282  0.3356934
  -0.19669648]
 [ 0.29747633  0.32440288 -0.11064381 ...  0.43594823  0.30562149
   0.19495755]] [[4.74555571]
 [4.01941083]
 [4.78797387]
 [4.52372507]
 [3.90090052]
 [4.72973735]
 [3.94680479]
 [4.40084712]
 [4.80466278]
 [4.68197067]] [[ 0.43656009 -2.37930625 -2.74283115  0.76792434  7.03308883  3.04918399
  -0.99122932 -1.53843429 -4.57474891  5.37365713]
```

```
label,Px0,Px1,Px2,Px3,Px4,Px5,Px6,Px7,Px8,Px9,Px10,Px11,Px12,Px13,Px14,Px15,Px16,Px17,Px18,Px19,Px20,Px21,Px22,Px23
6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,94,119,119,110,102,99,36,0,0,0,0,0
5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,43,94,24,0,0,0,0,0,0,0,0,0,0,0,0,
9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,42,107,104,11,0,0,0,0,0,0,0
9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,14,95,111,116,76,18,0,0,0,0,0,0,0
8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,10,68,85,97,103,119,91,20,0,0,0,0,0,0
5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,58,107,108,82,12,0,0,0,0
5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,43,98,108,63,0,0,0,0,0,0,
5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,63,60,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,47,85,90,72,0,0,0,0,0,0,0,0,0,0
6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,37,71,104,112,89,39,0,0,0,0,0,0,0
9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,20,87,112,87,21,0,0,0,0,0,0,0,0,0,0,0,0
```

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL COMMENTAIRES JUPYTER

```
PS C:\Users\alexa\courses\vs\bsp\nn_training> python setdata.py
You launched setdata.py! Please enter a number of entries to shuffle (would recommend around 10000): 10000
Please wait...
Shuffling Done!
Your data/data.csv file is ready for use! It is composed of a column of label (from 0 being an apple to 9 being a wheel)
Please check GitHub for more informations!
PS C:\Users\alexa\courses\vs\bsp\nn_training> cd ..
```

Figure 5 - DoodleRec - Example of training.py shell when executing and of a shuffled data.csv using setadata.py.

8.2. Functional Properties

Training the neural network

Users: Creator / administrator only, additional, but not included to the website.
Description: Train a random generated algorithm to a measured algorithm capable of distinguishing 10 types of images.
Parameters: Input: large amount of arrays that represents images (matrices).
Pre-condition: The algorithm is initialized with random values.
Post-condition: The algorithm must now be able to categorize 10 types of images.
Trigger: Cannot be triggered on the website, comes aside as ”administrator only”, can be launched with an IDLE.

Taking user’s doodle input

Users: Everyone using the website.
Description: Input section as a canvas section on the website for the user.
Trigger and parameters: Input by clicking and dragging.
Pre-condition: The page must be refreshed in order to reset the input area.
Post-condition: The image can be rendered into an array on the front-end application.

Fetching the input to a back-end application

Description: Using the said trained algorithm, the data is processed and a result is outputted to the user.
Parameters: Image data array, fetched from the front-end application.
Pre-condition: The user must have inputted a doodle, and must have been fetched to the back-end application.
Post-condition: Defined results are returned, modifying a Jinja section of the website to inform the user on its doodle caracteristics.
Trigger: Automatically triggered after a new result has been received.

Processing the user’s input and returning a result

Description: The front-end application takes the image data then sends it to a back-end application to process.
Pre-condition: The user has inputted a doodle.
Post-condition: The back-end application has now the user’s input.
Trigger: Automatically triggered every time a user inputs a doodle.

8.3. Files and Folders layout

In order for DoodleRec to function properly, please follow this layout below. Informative and non-essential files (README.md, requirements.txt) have been omitted.

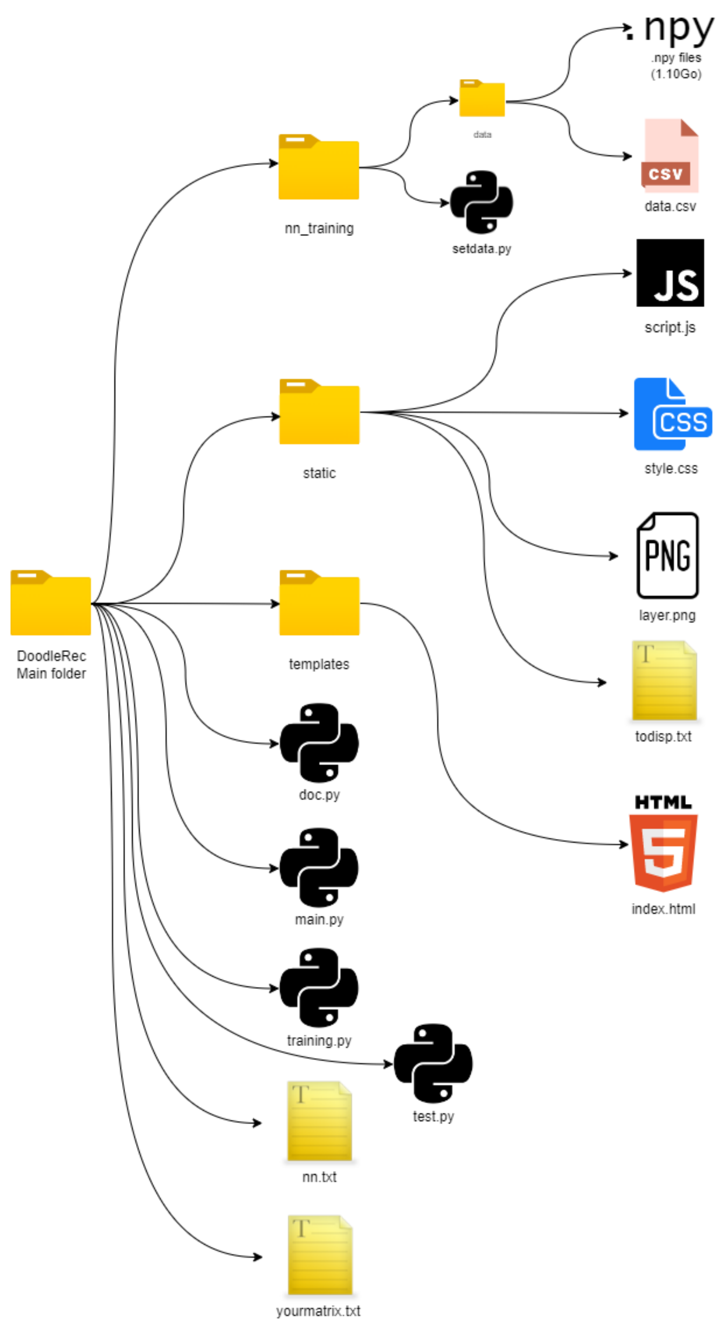


Figure 6 - DoodleRec - Layout of main files and folders.

8.4. Code

8.4.1. main.py

```python
'''main.py'''
'''Github repository for this BSP: https://github.com/CookNChips/BSP1---DoodleRec-Deep-Learning-Image-Recognition'''
'''For easier use, please execute the DoodleRec.bat file provided with this, or read the README file'''

import os #To open the files needed for the Flask application, as well as the user's matrix yourmatrix.txt file
import json #To receive the JSON data from JavaScript requests
from flask import * #For the web application
import numpy as np #For testing datasets and open the data base that are .npy files
import pandas as pd #To get the length of the data set


#From training.py, we import the necessary functions in order to have redundancy between main.py and training.py
from doc import Sigmoid, Softmax, OneHot, ForwardPropagation, Prediction, Params

#Import the Weights and Biases from nn.txt
Weight1, Weight2, Bias1, Bias2 = Params()

#Initialize the Flask applications with the static files
app = Flask(__name__)

path_cwd = os.path.dirname(os.path.realpath(__file__))
path_templates = os.path.join(path_cwd,"templates")
path_static = os.path.join(path_cwd,"static")


#List of possible guesses by the software. Will be used to display the result.
names = ["Airplane", "Cookie", "Cat", "Cup", "Hat", "Snowman", "Star", "Stop_sign", "Television", "Tree"]


#Initialize the user's matrix from imported data, which is a 28x28 matrix
usermatrix = []
for i in range(0, 28):
  usermatrix.append([])
  for j in range(0, 28):
    usermatrix[i].append(0)


#Render the main page from the index.html static
@app.route('/')
def hello():
    return render_template('index.html', data="True")

#Receive the data from the JavaScript that arrives every second (1000ms)
@app.route("/", methods = ["post"])
def Compute():
  userdata = request.get_json()
  userinput = userdata['userinput']

  #Open the "yourmatrix.txt" file and write an introductory line
  f = open("yourmatrix.txt", "w")
  f.write("Here is the image data of your doodle, with values ranging from 0 to 255 from the least contrasted to most contrasted parts of your doodle." + 2*"\n")

  #Transform the parsed data by writing the proper usermatrix and yourmatrix.txt file in a matrix format
  for i in range(0, 28):
    for j in range(0, 28):
      usermatrix[i][j] = userinput[28*i + j]
    f.write(str(usermatrix[i]) + "\n")


  #Initialize the Final matrix
  Final = [userinput]

  #Take the user input and transform it into a numpy array to compute
  Final = np.tile(np.array(Final).T, (1, len(pd.read_csv('nn_training/data/data.csv'))))

  #Check if Final is not a 0 matrix
  if not np.any(Final):
   print(None)
   return "None"

  #Pass into the Neural Network only one time
  Z1, A1, Z2, A2 = ForwardPropagation(Final, Weight1, Bias1, Weight2, Bias2)

  #Find the prediction according to the Neural Network
  Guess = Prediction(A2)
  Guess = names[Guess[0].item()]
  print(Guess)

  #Write the Guess in the yourmatrix.exe file
  f.write(Guess)

  #Close the .txt
  f.close()

  #Open todisp.txt, write the guess and close the file
  f1 = open("static/todisp.txt", "w")
  f1.write(Guess)
  f1.close()

  return Guess

#Download function - Downloads yourmatrix.txt
@app.route("/download")
def download_data():
  file = "yourmatrix.txt"
  return send_file(file, as_attachment=True)

#Run the application
app.run()
```

8.4.2. doc.py

```
'''doc.py'''
'''Github repository for this BSP:
    https://github.com/CookNChips/BSP1---DoodleRec-Deep-Learning-Image-Recognition'''
'''Only usable via IDLE for now'''
'''Please execute training.py for training'''


'''Part 0. Initialisation of data set and parameters'''

#Import needed libraries to read data (csv and .npy files)
import numpy as np
import pandas as pd
import os
import json


#Prompt the user for a LearningRate and an Iteration number
def Prompt():

    LearningRate = input("Welcome to training.py! Make sure the .csv file is ready to use (by
        launching setdata.py). Please specify a learning rate (would recommend no more than 1,
        must be an integer or a float): ")

    try:
        LearningRate = float(LearningRate)
    except:
        os.sys.exit("Error: Must specify a float.")

    Iterations = input("How many iterations? (Please specify an integer) ")

    try:
        Iterations = int(Iterations)
    except:
        os.sys.exit("Error: Must specify an integer.")

    return LearningRate, Iterations



#Initialize data into into a set of training data
data = pd.read_csv('nn_training/data/data.csv')
data = np.array(data)
x, y = data.shape

TrainingData = data.T
Labels = TrainingData[0]
DoodleData = TrainingData[1:y]
DoodleData = DoodleData


#Initialize random parameters for weights and biases
def Parameters():
    Weight1 = np.random.rand(10, 784) - 0.5
    Bias1 = np.random.rand(10, 1) - 0.5
    Weight2 = np.random.rand(10, 10) - 0.5
    Bias2 = np.random.rand(10, 1) - 0.5
    return Weight1, Bias1, Weight2, Bias2


'''Part 1. Mathematical functions: Activation, Derivative, and Loss functions'''

#Sigmoid function for an entire matrix
def Sigmoid(Matrix):
    return 1 / (1 + np.exp(-Matrix))

#Sigmoid derivative function for an entire matrix
def SigmoidDerivative(Matrix):
    Sigma = Sigmoid(Matrix)
    return Sigma * (1 - Sigma)

#Softmax function for an entire matrix
def Softmax(Matrix):
    A = np.exp(Matrix) / sum(np.exp(Matrix))
    return A

#OneHot encoding function for an entire matrix
def OneHot(Matrix):
    OneHotMatrix = np.zeros((Matrix.size, Matrix.max() + 1))
    OneHotMatrix[np.arange(Matrix.size), Matrix] = 1
    OneHotMatrix = OneHotMatrix.T
    return OneHotMatrix


'''Part 2. Neural Network functions: ForwardPropagation, BackwardPropagation, NewParameters
    and LowOnCurve'''

```

```python
#Step 1) ForwardPropagation: we apply weights and biases to the matrix with Sigmoid and
    Softmax as activation functions
def ForwardPropagation(Matrix, Weight1, Bias1, Weight2, Bias2):
   Z1 = Weight1.dot(Matrix) + Bias1
   A1 = Sigmoid(Z1)
   Z2 = Weight2.dot(A1) + Bias2
   A2 = Softmax(Z2)
   return Z1, A1, Z2, A2

#Step 2) BackwardPropagation: we apply the obtained results with the SigmoidDerivative and
    OneHot as loss functions
def BackwardPropagation(Mat1, Mat2, Z1, A1, Z2, A2, Weight1, Weight2):
   OneHotMatrix = OneHot(Mat2)
   dZ2 = A2 - OneHotMatrix
   derWeight2 = 1 / x * dZ2.dot(A1.T)
   derBias2 = 1 / x * np.sum(dZ2)
   dZ1 = Weight2.T.dot(dZ2) * SigmoidDerivative(Z1)
   derWeight1 = 1 / x * dZ1.dot(Mat1.T)
   derBias1 = 1 / x * np.sum(dZ1)
   return derWeight1, derBias1, derWeight2, derBias2

#Step 3) According to the error margin, we update the parameters while taking into account the
    learning rate
def NewParamaters(Rate, Weight1, Bias1, Weight2, Bias2, derWeight1, derBias1, derWeight2,
    derBias2):
   Weight1 = Weight1 - Rate * derWeight1
   Bias1 = Bias1 - Rate * derBias1
   Weight2 = Weight2 - Rate * derWeight2
   Bias2 = Bias2 - Rate * derBias2
   return Weight1, Bias1, Weight2, Bias2


'''Part 3. Accuracy and Prediction functions'''

#Prediction function: we use np.argmax() to take the prediction label
def Prediction(A2):
   return np.argmax(A2, 0)

#Accuracy function
def Accuracy(Predictions, Matrix):
   print("Prediction: " + str(Predictions) + "\n" + "Reference: " + str(Matrix))
   return np.sum(Predictions == Matrix) / Matrix.size


'''Part 4. Execute every said function, which happens to find low points on the probability
    curve using derivatives'''

#LowOnCurve function
def LowOnCurve(Mat1, Mat2, Rate, Iteration):

   #Initialize our parameters
   Weight1, Bias1, Weight2, Bias2 = Parameters()

   #Apply all cited steps
   for i in range(Iteration):
      Z1, A1, Z2, A2 = ForwardPropagation(Mat1, Weight1, Bias1, Weight2, Bias2)
      derWeight1, derBias1, derWeight2, derBias2 = BackwardPropagation(Mat1, Mat2, Z1, A1, Z2,
          A2, Weight1, Weight2)
      Weight1, Bias1, Weight2, Bias2 = NewParamaters(Rate, Weight1, Bias1, Weight2, Bias2,
          derWeight1, derBias1, derWeight2, derBias2)

      #Every 100th iteration, we print the prediction and accuracy of the process
      if i % 100 == 0:
         Predictions = Prediction(A2)
         print("Iteration: " + str(i))
         print("Accuracy: " + str(Accuracy(Predictions, Mat2).item()*100) + "%" + "\n")
   return Weight1, Bias1, Weight2, Bias2


'''Part 5. Appendix / Support'''
#Params() function, to import the Neural Network's Weights and Biases to compute.
def Params():

   #Open nn.txt as read
   f = open("nn.txt","r")

   #Create a counter
   counter = 1

   #Iterate through the lines to find Weights and Biases in nn.txt
   for line in f:
      if counter == 3:
         Weight1 = json.loads(line)
      if counter == 4:
         Weight2 = json.loads(line)
      if counter == 5:
         Bias1 = json.loads(line)
      if counter == 6:
         Bias2 = json.loads(line)
      counter = counter + 1

   f.close()

   #Transform the arrays back into numpy arrays
   Weight1 = np.array(Weight1)
   Weight2 = np.array(Weight2)
   Bias1 = np.array(Bias1)
   Bias2 = np.array(Bias2)

   return Weight1, Weight2, Bias1, Bias2
```

8.4.3. training.py

```python
'''training.py'''
'''Github repository for this BSP:
    https://github.com/CookNChips/BSP1---DoodleRec-Deep-Learning-Image-Recognition'''
'''Only usable via IDLE for now'''

#Import needed libraries to read data (csv and .npy files)
import numpy as np
import pandas as pd
import os

#Import functions from doc.py
from doc import *

#Prompt user for LearningRate and number of Iterations
LearningRate, Iterations = Prompt()

#Compute the Weights and Biases
Weight1, Bias1, Weight2, Bias2 = LowOnCurve(DoodleData, Labels, LearningRate, Iterations)

#Transform them into strings
Weight1 = str(Weight1.tolist())
Weight2 = str(Weight2.tolist())
Bias1 = str(Bias1.tolist())
Bias2 = str(Bias2.tolist())

#Write the values in nn.txt
f = open("nn.txt", "w")
```

```
f.write("Neural Network Weights (1, 2) and Biases (1, 2):" + 2*"\n" + Weight1 + "\n" + Weight2
    + "\n" + Bias1 + "\n" + Bias2)

print("Training is done! nn.txt is now correctly updated with your Weight1, Weight2, Bias1 and
    Bias2 matrices. Please rerun DoodleRec (main.py) to update the Neural Network.")
```

8.4.4. setdata.py

```
'''setdata.py'''
'''Prepares a random set of data of 10000 doodles from .npy files to a .csv file'''

#Import needed libraries to exploit data
import csv
import numpy as np
import random
import os

#Initialize the user to choose how many entries need to be generated and shuffled
nb = input("You launched setdata.py! Please enter a number of entries to shuffle (would
    recommend around 10000): ")

#Verifies if the user has inputted an integer
try:
   nb = int(nb)
except:
   os.sys.exit("Invalid input. Please make sure to only enter integers.")

print("Please wait...")

#Import all the data from .npy files
airplane = np.load(os.getcwd() + "\\data\\airplane.npy", allow_pickle=True)
cookie = np.load(os.getcwd() + "\\data\\cookie.npy", allow_pickle=True)
cat = np.load(os.getcwd() + "\\data\\cat.npy", allow_pickle=True)
cup = np.load(os.getcwd() + "\\data\\cup.npy", allow_pickle=True)
hat = np.load(os.getcwd() + "\\data\\hat.npy", allow_pickle=True)
snowman = np.load(os.getcwd() + "\\data\\snowman.npy", allow_pickle=True)
star = np.load(os.getcwd() + "\\data\\star.npy", allow_pickle=True)
stop_sign = np.load(os.getcwd() + "\\data\\stop_sign.npy", allow_pickle=True)
television = np.load(os.getcwd() + "\\data\\television.npy", allow_pickle=True)
tree = np.load(os.getcwd() + "\\data\\tree.npy", allow_pickle=True)

#Indexing the data in a list
index = [airplane, cookie, cat, cup, hat, snowman, star, stop_sign, television, tree]

#Prepare the header of the csv output file
header = ["label"]

#Add the pixel numbers to the header
for i in range(784):
   header.append("Px" + str(i))

#Prepare the data array
data = []

#Choose random doodles among the .npy files and append them to the data array
for i in range(nb):
   rand1 = random.randint(0, 9)
   rand2 = random.randint(0, 100000)
   toapp = np.insert(index[rand1][rand2], 0, rand1)
   toapp = toapp.tolist()
   data.append(toapp)

#Write the header and the data array in the csv file
with open(os.getcwd() + "\\data\\data.csv", 'w', newline='') as file:
   writer = csv.writer(file)
```

```
    writer.writerow(header)
    writer.writerows(data)

#Inform the user that the process is done
print("Shuffling Done!" + "\n" + "Your data/data.csv file is ready for use! It is composed of
    a column of label (from 0 being an apple to 9 being a wheel) with pixel values on rows." +
    "\n" + "You can now safely close this terminal instance." + "\n" + "Please check GitHub
    for more informations!")
```

8.4.5. test.py

```
'''test.py'''
'''Github repository for this BSP:
    https://github.com/CookNChips/BSP1---DoodleRec-Deep-Learning-Image-Recognition'''
'''For easier use, please execute the DoodleRec.bat file provided with this, or read the
    README file'''

import os #To open the files needed for the Flask application, as well as the user's matrix
    yourmatrix.txt file
import pandas as pd #To read data.csv
import numpy as np #For testing datasets and open the data base that are .npy files
import matplotlib.pyplot as plt #For displaying the image
import random as rd #For selecting a random number

#List of possible guesses by the software. Will be used to display the result.
names = ["Airplane", "Cookie", "Cat", "Cup", "Hat", "Snowman", "Star", "Stop_sign",
    "Television", "Tree"]

#From training.py, we import the necessary functions in order to have redundancy between
    main.py and training.py
from doc import Sigmoid, Softmax, OneHot, ForwardPropagation, Prediction, Params

#Init() function to prompt the user for a number of images to analyze
def Init():

    nb = input("Welcome to test.py! Test the Neural Network over random images from data.csv.
        If you want new images to test on, you can use setdata.py! Please specify a number of
        images to test on: ")

    try:
        nb = int(nb)
    except:
        os.sys.exit("Error: Must specify an integer.")

    return nb

nb = Init()

#Import the Weights and Biases from nn.txt
Weight1, Weight2, Bias1, Bias2 = Params()

#Read the data in data.csv
data = pd.read_csv('nn_training/data/data.csv')
data = np.array(data)

#Iterate through random images and analyze them
for i in range(nb):
    r = rd.randint(0, len(data) - 1)
    image = data[r]
    label = image[0]
    image = image[1:]
    imageR = image.reshape((28, 28)).tolist()
    image = np.tile(image, (len(data), 1)).T

    Z1, A1, Z2, A2 = ForwardPropagation(image, Weight1, Bias1, Weight2, Bias2)
```

```python

    #Find the prediction according to the Neural Network
    Guess = Prediction(A2)
    Guess = names[Guess[0].item()]
    print(f"Image {i + 1}: \n Label: {names[label]} \n Prediction: {Guess} \n Remaining image:
        {nb - i - 1}, please close the image to continue. \n \n \n")

    #Display the image
    plt.gray()
    plt.imshow(imageR, interpolation='nearest')
    plt.show()
```

8.4.6. index.html

```html
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta name="viewport" content="width=device-width", initial-scale=1>
      <meta charset="UTF-8"/>
      <meta http-equiv="X-UA-Compatible" content="ie=edge"/>
      <link href="{{ url_for('static', filename='/style.css') }}" rel="stylesheet">
      <title>DoodleRec</title>
   </head>
   <body>
      <aside>
         <div><h1 class="heading1">DoodleRec - Beta</h1></div>
         <div><h2><a
            href="https://github.com/CookNChips/BSP1---DoodleRec-Deep-Learning-Image-Recognition">Github
            Repository</a> of this project.</h2></div><hr>
         <div><h2>How to use:</h2></div>
         <div><h5>1/ Create a doodle by clicking and dragging on the left canvas.</h5></div>
         <div><h5>2/ Retrieve your image data by clicking on the dedicated button.</h5></div>
         <div><h5>3/ The guess appears on your terminal shell. </h5></div>
         <div><h4>This app only works using a desktop or a laptop, as I encourage to use it
            fullscreen.</h4></div>
         <div id="data"></div>
      </aside>
      <canvas id="canvas" height="28" width="28"></canvas>
      <a href="{{ url_for('download_data') }}"><button type="button">Retrieve your
         Data!</button></a>
      <script src="{{ url_for('static', filename='/script.js') }}"></script>
      <style>
         * {
            background-color: rgba(12, 12, 12, 0);
            background-size: cover;
         }
         body {
            background-repeat: no-repeat;
            background-position: center;
            height: 98%;
            width: 98%;
            background-image: url({{ url_for('static', filename='layer.png') }});
         }
      </style>
   </body>
</html>
```

8.4.7. script.js

```javascript
/*script.js
Github repository for this BSP:
    https://github.com/CookNChips/BSP1---DoodleRec-Deep-Learning-Image-Recognition
For easier use, please execute the DoodleRec.bat file provided with this, or read the README
    file*/
```

```
const canvas = document.getElementById("canvas"); //Create a canvas element
const ctx = canvas.getContext("2d"); //As well as a context zone for intercation

let painting = false; //Initialize painting state

//Initialize starting position to avoid errors with the user’s mouse position
function startPos(e) {
   painting = true;
   draw(e);
}

function finPos() {
  painting = false;
  ctx.beginPath();
}

//Initialize canvas’ size and context zone to be 28x28 pixels rendered into 420x420px
canvas.width = 28;
canvas.height = 28;
canvas.style.width = "420px";
canvas.style.height = "420px";
ctx.scale((28/420), (28/420));

//draw function with event listeners
function draw(e) {
   if (!painting) return;
   ctx.imageSmoothingEnabled = false;
   ctx.lineWidth = 15;
   ctx.lineCap = "circle";
   ctx.lineTo(e.clientX - 100, e.clientY - 100);
   ctx.stroke();
   ctx.strokeStyle = "#FFFFFF";
   ctx.beginPath()
   ctx.moveTo(e.clientX - 100, e.clientY - 100);
}

canvas.addEventListener(’mousedown’, startPos);
canvas.addEventListener(’mouseup’, finPos);
canvas.addEventListener(’mousemove’, draw);

//Every 1s, this function takes the image data of the input image and returns an array of it
//It then parses it to a JSON request, that will be fetched by the Flask application
setInterval(function(){
   imgData = ctx.getImageData(0,0,28,28);
   imgDataBW = [];
   imgDataBW.push(imgData.data[3]);
   for (var i = 7; i < imgData.data.length + 1; i = i + 4) {
      imgDataBW.push(imgData.data[i]);
   }

   var sendData = {
      "userinput" : imgDataBW
      }
        fetch(‘‘, {
          method: "POST",
          credentials: "include",
          body: JSON.stringify(sendData),
          cache: "no-cache",
          headers: new Headers({
            "content-type": "application/json"
          })
        })

   fetch(’/static/todisp.txt’)
   .then(function(response) {return response.text();})
   .then(function(data){document.getElementById("data").innerHTML = data;})

}, 1000);
```

8.4.8. style.css

```css
@import url('https://fonts.googleapis.com/css?family=Montserrat');

* {
   margin: 300;
   padding: 300;
   box-sizing: border-box;
   background-color: #1E1E1E;
   font-family: 'Montserrat';
}

aside {
   width: 50%;
   float: right;
   color: #FFFFFF;
   font-family: 'Montserrat';
   margin-top: 60px;
}

.heading1 {
   font-weight: 1000;
   font-size: 40px;
   line-height: 50px;
   align-items: left;
   text-align: left;
}

button {
   color: #FFFFFF;
   border: solid #444444;
   cursor: pointer;
   background-color: #2a2a2a;
   font-size: 24px;
   border-radius: 12px;
   align-items: center;
   display: flex;
   margin-left: 14%;
   margin-top: 5%;
}

canvas {
   border: solid #444444;
   border-radius: 25px;
   background-color: #2a2a2a;
   margin-left: 100px;
   margin-top: 100px;
}

hr {
   width: 50%;
   color:#FFFFFF;
   background-color: #FFFFFF;
   margin-left: 0;
}
```