University of Liverpool

Bachelor Thesis

Artificial Neural Networks: Kohonen Self-Organising Maps

Author:
Eklavya Sarkar

Supervisors:
Dr. Irina Biktasheva
Dr. Rida Laraki

A thesis submitted in the partial fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Science

May 10, 2018

Declaration of Authorship

I, Eklavya SARKAR, declare that this thesis entitled, "Artificial Neural Networks: Kohonen Self-Organising Maps" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a Bachelor degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date: 10.05.2018

UNIVERSITY OF LIVERPOOL

Abstract

Faculty of Science and Engineering
Department of Computer Science

Bachelor of Science

Artificial Neural Networks: Kohonen Self-Organising Maps

by Eklavya SARKAR

In the coming years, the impact of Artificial Intelligence (AI) will be keenly felt, in both, our personal and professional lives. Given the pace and scale of developments in this field, it is imperative to explore AI research and potential applications.

Kohonen's Self-Organising Maps is an algorithm used to improve a machine's performance in pattern recognition problems. The algorithm is especially capable of clustering and visualising complex high-dimensional data and can potentially be applied to solve many complex real-world problems.

The aim of this thesis is to provide an in-depth study of Kohonen's algorithm, and present insights of its properties, by implementing a complete and functional model.

As part of this project, an extensive literature review on Kohonen networks was conducted first; and a brief background on its relevance to society, the technical structure, and the variables and formulas are presented. The scope, aims and objectives of the project are then defined in detail, highlighting the key differences that make Kohonen networks unique compared to other available models.

Subsequently, the project follows a design methodology, employing identified technologies to build a model, before presenting a comprehensive description of how each component of the final implementation was realised and tested.

The results of the project are then presented to provide answers to the formulated problem, before evaluating the project, and discussing its strengths, weaknesses, and the general learning points.

Acknowledgements

Writing a quality thesis alongside testing and implementing an entire software in Computer Science largely comes down to a balancing act, requiring a healthy mix of guidance, encouragement, and support. I would like to take the time sincerely thank the contributors whose inputs were critical to this project.

First and foremost, this project would not have been possible without my supervisor, Dr. Irina Biktasheva, whom I thank not only for accepting me as her student, but for her comprehensive guidance on managing each submission, and overall insight on the deeper purpose of the project.

Additionally, I would also like to thank Dr. Radi Laraki for reviewing my papers, providing additional feedback, and being part of my educational journey.

Furthermore, I have to distinctly thank my friends for providing a steady network of support, and my family for making me understand the value of excellence and the reasons one should pursue it.

Lastly, the work achieved in this project would not have been possible without the innate will to constantly explore and learn more. The desire to investigate the field of Machine Learning in depth is what drove me to undertake this project, and is an indispensable ingredient for all students aiming to develop a distinctive project.

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

AI	**A**rtificial **I**ntelligence
AJAX	**A**synchronous **J**avaScript and **X**ML
ANN	**A**rtificial **N**eural **N**etworks
BMU	**B**est **M**atching **U**nit
CDN	**C**ontent **D**elivery **N**etwork
CLI	**C**ommand **L**ine **I**nterface
CSS	**C**ascading **S**tyle **S**heets
D3	**D**ata **D**riven **D**ocuments
DOM	**D**ocument **O**bject **M**odel
EMNIST	**E**xtended **M**odified **N**ational **I**nstitute of **S**tandards and **T**echnology
GUI	**G**raphical **U**ser **I**nterface
HTML	**H**yper **T**ext **T**ransfer **P**rotocol
ML	**M**achine **L**earning
MNIST	**M**odified **N**ational **I**nstitute of **S**tandards and **T**echnology
OCR	**O**ptical **C**haracter **R**ecognition
PC	**P**ersonal **C**omputer
SOM	**S**elf-**O**rganising **M**ap
SVG	**S**calable **V**ector **G**raphics
UI	**U**ser **I**nterface
UX	**U**ser **x**perience

Glossary

The following term's definition are given specifically from a Computer Science or Machine Learning perspective.

Ajax: a set of web development techniques to create asynchronous web applications that allows for such pages and applications to change content dynamically without the need to reload the entire page.

Best Matching Unit: the vector that is the optimal fit, i.e. with the smallest Euclidian distance, for the given input vector in the Kohonen network.

Bootstrap: a popular, free and open-source front-end web framework for designing websites and web applications.

D3.js: a JavaScript library for producing dynamic, interactive data visualizations in web browsers.

Django: a high-level open-source Python web framework.

Euclidian Distance: the shortest straight-line distance between two points in Euclidean space.

Feature: a measurable property, characteristic, attribute or variable of an analysed phenomenon or observed object, e.g. a petal length of an iris, the grey scale intensity of a pixel, or the RGB values of a colour.

Feature Vector: an n-dimensional vector of features.

Flask: a micro web framework written in Python and based on the Jinja2 template engine.

Jinja2: a modern and designer-friendly template language for Python, modelled after Django's templates.

jQuery: a cross-platform JavaScript library designed to simplify the client-side scripting of HTML.

Machine Learning: a field of Computer Science and sub-field of Artificial Intelligence, which uses statistical techniques to give the computer an ability to seemingly learn from input data without being explicitly programmed.

Model: the Machine Learning network implemented according to the chosen algorithm.

Optical Character Recognition: the conversion of handwritten or printed text into electronic machine-readable text.

Pattern Recognition: a branch of Machine Learning that attempts to group data in sections based on its patterns, repetitions or differences. Depending on the availability of labels, pattern recognition can be considered to be part of supervised learning (sorting) or unsupervised learning (clustering).

Supervised Learning: a sub-field of Machine Learning where the given input data's also contains information on the total number of classes, labels, and outputs.

Topology: the structure, i.e. the distances and links between nodes, of a network.

Unsupervised Learning: a sub-field of Machine Learning where the data is given without any labels, number of total classes, or any information on the outputs.

Vector: an array containing a collection of values, usually in one-dimension unless explicity mentioned otherwise.

List of Symbols

Symbol	**Variable**	**Name**
t	`i`	Current iteration
n	`n_iterations`	Iteration limit
λ	`time_constant`	Time constant
i	`x`	Row coordinate of the nodes grid
j	`y`	Column coordinate of the nodes grid
d	`w_dist`	Distance between a node and the BMU
$\vec{w}$	-	Weight vector
$w_{ij}(t)$	`w`	Weight of the node i, j linked to input at iteration t
$\vec{x}$	`inputsValues`	Input vector
$x(t)$	`inputsValues[i]`	Input vector's instance at iteration t
$\alpha(t)$	`l`	Learning rate
$\beta_{ij}(t)$	`influence`	Influence of the neighbourhood function
$\sigma(t)$	`r`	Radius of the neighbourhood function
-	`n`	Total number of grid rows
-	`m`	Total number of grid columns
-	`net[x,y,m]`	Nodes grid
-	`n_classes`	Total number distinct classes in input
-	`labels`	Label vector of every input's instance

For my family, and my future self.

Chapter 1

Introduction

1.1 Artificial Neural Networks

1.1.1 Background

Humans and animals have always been fundamentally proficient at pattern recognition, having learnt since birth to be able to innately identify patterns and respond to them. This allows them to communicate and interact in different biological ways, thanks to the brain's intricate ability to constantly learn. Computationally complex tasks such as understanding speech and visual processing are effortless for humans, by virtue of exceedingly developed neural networks within the human brain, capable of constantly encoding and processing patterns.

Even the most advanced computers, although very competent and precise at following large sets of linear, logical and arithmetic rules, have historically not been nearly as capable as humans at discerning visual or audible patterns. Until only very recently, sub-fields of Computer Science involved in facial and speech recognition, handwriting classification, and natural language processing have not seen software implementations with highly accurate results capable of solving these problems.

Artificial neural networks (ANNs) are essentially biologically-inspired algorithms, employed in the field of Artificial Intelligence, in an attempt to enable computers to seemingly *learn* from observational data. In other words, these algorithms allow a program to improve its functionality on a task, and to go from a certain state of capability to a new one of improved performance in subsequent situations. Instead of specifically programming a software to perform tasks by following certain rules written in a coding language, information in artificial neural networks is distributed throughout the network. To fully understand the nature of how they work, a certain abstraction is required, and is substantiated below.

1.1.2 Structure

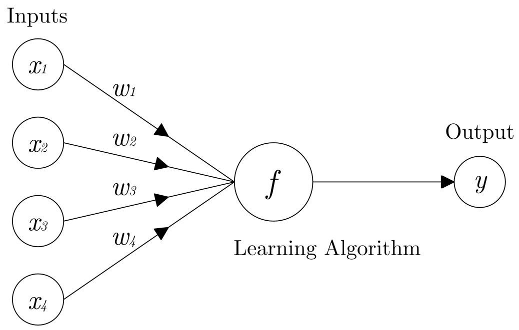


Figure 1.1: A simple artificial neural network

The information in neural networks can be visualised as *input* and *output nodes*, which are their own entities, as well as individually weighted *connections*, which are linked from nodes to nodes in various permutations, depending on the machine learning algorithm. The neural network therefore works by taking in a set of input *data* and a chosen algorithm, and then outputting data incrementally based on each input and the weights of the network's connections. The key aspect is that the weights are progressively adjusted *after each input*, a phase called *training*, allowing the network to improve *itself*, and output more and more accurate data at every iteration. After the network has gone through a certain quantity of inputs and is capable of distinguishing the data into different classes at a given accuracy, the improvement rate stabilises, and the network is said to have *converged*.

It's important to note that the set of inputs is not necessarily single-valued. Indeed, an input vector can be multi-dimensional, inserting 2, 3 or n values to the neural network at any given instance. The inputs represent *features* of the task in question, i.e. a measurable property or attribute of the observed phenomenon or object, and they are not as such necessarily limited to a single value. For example, a dataset of residents living in a university accommodation would contain several features for every single instance, such as name, gender, age, nationality, course, etc.

Name	**Gender**	**Age**	**Nationality**	**Course**
Eklavya Sarkar	M	23	Swiss	MSc Machine Learning
Polly Dawson	F	24	English	PhD Linguistics
Jérôme Besson	M	18	French	BSc Organic Chemistry

Table 1.1: A sample input vector of dimension 5 for each data instance

The number of features in an input space is thus equivalent to the *dimensionality* of its database. Furthermore, the dimension of the *output* vector of a network is *not* necessarily the same as that of the input.

Next Life Event	In x years
Work	2
Wedding	1
Education	3

TABLE 1.2: A sample output vector of dimension 2 for each instance

1.1.3 Learning Categories

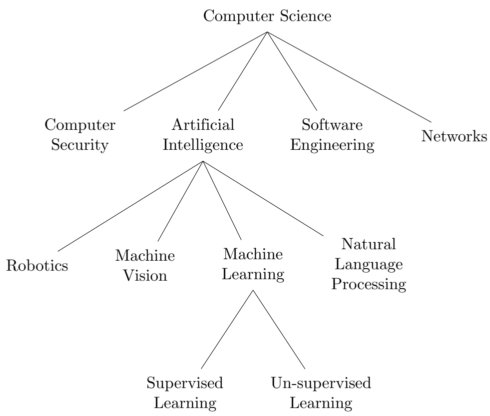


FIGURE 1.2: Branches of Computer Science

Artificial neural networks can be distinctly divided into two categorises based on their learning process. In the event where the data is *labelled*, i.e. the input training set is accompanied by an equivalent set of associated labels, the iterative process is called *supervised* learning. A label could indicate anything from whether or not a photo contains a car, to which certain words were mentioned in an audio, or else which colour is shown on a image.

Lower Bound	Upper Bound	Label
70	100	First Class
60	69	Upper Second Class
50	59	Lower Second Class
40	49	Third Class
0	39	Fail

TABLE 1.3: Grade percentages and their corresponding class

The labels can be understood as the corresponding *target* or *desired* output values, and can be used to measure and evaluate the network's accuracy, error-rate and overall convergence over time. The goal in such cases is then to train the network to a degree, that it can successfully predict - *classify* - new unknown and unlabelled *testing* data, which nonetheless belongs to the same input space as the training data.

For example, in order to classify handwritten digits (0-9), a supervised machine learning algorithm would take 9000 pictures of such drawn characters, along with a list of 9000 labels containing the number each image represents. The chosen algorithm will then learn the relationship between the images and their associated alphabet labels, and then apply that learned relationship to classify 1000 completely new unlabelled images that it hasn't seen before. If it manages to correctly classify 900 out of the total 1000 testing images, it would be said to have an accuracy of 90%, and an error rate of 10%.

The other category, where the input data space is unknown and contains no associated labels, the process is called *unsupervised* learning. The goal is then not only to *cluster* the input data into groups, but also to discover the structure and patterns - the topology - of the input space itself, by grouping them into clusters according to the similarity between one another.

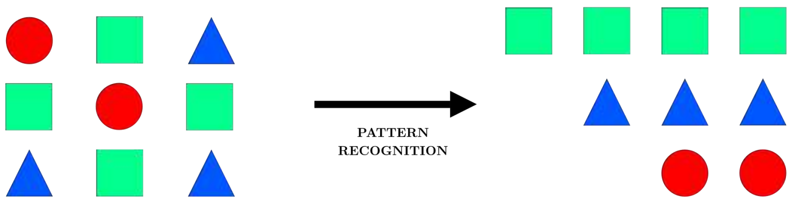


FIGURE 1.3: Unsupervised learning clusters data solely according to their feature similarities, as no labels are used

In contrast to supervised learning, we cannot directly measure the accuracy of the calculated outputs because there are no target outputs to compare them with. The performance of the network is therefore often subjective and domain-specific. The accuracy of how well a network clusters data could depend on the effectiveness of the chosen algorithm, how well it is applied, and how much useful training data is available. An important feature of this type of learning is that no human interaction is needed. Indeed, as the model requires no labels, the human necessity to review the data is bypassed, thus reducing by a considerable amount the time and effort required to assemble large datasets.

However, many datasets which can be used for unsupervised learning *do* come with labels. These can simply be ignored if the aim is to study a particular unsupervised learning algorithm and its effectiveness. In this case, the labels can be used after the network has finished training to measure the accuracy of the model, or simply aid in the visualisation of the data after clustering.

An important property of neural networks is that a small portion of bad data or a small section of non-functional nodes will not cripple the entire network. It will instead adapt, and continue working, unless the quantity of faulty data crosses the acceptable threshold, in which case incorrect outputs will be produced.

1.1.4 Learning Algorithms

Finally, the chosen algorithm is what determines two important elements: the *architecture* and the eventual output of the network. The former is essentially the number of layers, how nodes are linked to one another, and how the weight adjustments influence other connected nodes. The output node is *fired* if the inputs exceed a certain threshold.

These networks - supervised or unsupervised - can eventually become remarkably capable of doing certain tasks that conventional programs cannot. Moreover, depending on the task, the quantity and quality of the training data, the chosen algorithm, and the complexity and accuracy of a few other factors, the converged artificial neural network can match or even surpass surpass human ability at the task.

Machine Learning			
Supervised		**Unsupervised**	
Classification	**Regression**	**Clustering**	**Dimensionality Reduction**
Support Vector Machine (SVM)	Support Vector Regressor (SVR)	Hebbian Learning	Principal Component Analysis (PCA)
Logistic Regression	Linear Regression	Self Organising Maps (SOM)	Linear Discriminant Analysis (LDA)
Naive Bayes	Decision Trees	Mixture Models	Flexible Discriminant Analysis (FDA)
Nearest Neighbour (k-NN)	Random Forest	k-Means	Singular-Value Decomposition (SVA)

TABLE 1.4: A few selected machine learning algorithms from the listed categories.

One such type of neural network, Self-Organising Maps (`SOM`), and it's learning algorithm by Kohonen Teuvo will be the focus of this study.

1.2 Problem

The problem this project attempts to solve is a mix of research, communication and implementation tasks.

The principal problem of this project is the implementation of a Kohonen Network as an application, in order to demonstrate its usefulness and explain the concepts of Machine Learning, by means of a converging Self-Organising Map.

1.3 Aims

The aim of this project was to build a Kohonen network that is capable of clustering data, such as hand-drawn letters on an web program, and which would also allow users to test their *own* data. The point of the of the web application, which hosts the model, was to essentially act as an interactive learning tool for other interested students or hobbyists on Machine Learning and Artificial Intelligence.

1.4 Objectives

1.4.1 Essential Features

- Implementing a fully functional Kohonen back-end model, capable of receiving and processing data from a chosen dataset.
- Training the network with a large quantity of data until it reaches a high accuracy rate of clustering.
- Implementing a web application to host and interact with the developed model.

- The web application should communicate with the computational back-end model and retrieve the clusterisation data.
- Using different web pages for explanations of various concepts, features and parameters of the Kohonen algorithm in order to explain SOMs to users in layman's terms.

- The website should have an interactive 'Draw' page where users can draw their own letter on a Graphical User Interface (`GUI`) canvas and have the website process and display which letter it is, by interacting with the `ANN` model.
- The website should display the neural network's topological map of alphabets to the user based on training data.
- The website should have a page which displays animations or diagrams over time of neural networks and `SOM`s, to show its evolution, how its weights are adjusted and converged, and how the network is trained over time.
- The website should have a 'Database' page which contains information on the dataset used to train and test the neural network, the size of the entire database, and links to the source-files.
- The website should have an 'About' page which contains information on technologies, libraries, tools and algorithms used for building the project.

1.4.2 Desirable Features

- The website should highlight where your input would be placed on the displayed topological map.
- The users should have an 'in-depth' option of seeing the steps the network goes through, such as re-centring, cropping and down-sampling of the input, probability numbers or graphs of which letter the input corresponds to.
- Allow users to input more than one single input at a time i.e. draw more than one letter in the input canvas.
- The 'database' page, which should show a sample training data character for each class, could also show the different handwritings for a specifically selected alphabet. This is to give a visual representation and sense of scale of how many different handwritten letters were used to train the neural network for each alphabet.
- Some of the instructions sentences on the website could be written using the synthetic training data images.

1.5 Predicted Challenges

Initially, the main predicted challenge was simply the implementation difficulty of the Kohonen network model, and its visualisation on the front-end. Additionally, being able to choose particularly relevant examples and methods to illustrate SOMs as a teaching tool were also considered as a potential challenge. Finally, the vast scope and lack of real constraints were originally deemed problematic as well.

Chapter 2

Background

2.1 Problem

This problem is the implementation of a Kohonen model as a teaching tool for other interested students. It falls precisely into the category of pattern recognition in the field of Machine Learning.

2.2 Existing Solutions

There have been a number of previous implementations of neural networks that attempt to cluster data, especially hand-written digits, due to the popularity of the MNIST dataset.

However, almost none of these models employ Kohonen's algorithm for the task, as many instead favour a supervised and error-correction learning by means of convolutional neural networks.

This project's goal is not only to attempt to build a topological map of the input data, by using of an uncommon algorithm, but to do so with a much larger and complex dataset than the MNIST database. To add complexity to the task, alphabets along with digits were both used to build this implementation.

A model capable of distinguishing between similar digits and letters has certainly not been developed, especially using with Kohonen's learning process.

2.3 Research and Analysis

First of all, rigorous research went into conducting an extensive literature review on a completely new topic, to understand the nature of Self-Organising Maps: their topological mapping, competitive process, sample usages, general applications and actual implementation. Furthermore, substantial work was done reviewing Dr Irina V. Biktasheva's `COMP305`: Bio-computation module and Stanford's excellent 'Introduction to Machine Learning' course by Prof. Andrew Ng. The results of this work can be seen in Chapter 2.

Secondly, research went into the system design and how to make the SOM interactive for human users. All the extensive technologies, especially for front-end graphics visualisation and back-end algorithmic modelling, were thoroughly examined, as heavy data visualisation was planned.

Lastly, publicly available datasets on handwritten input and existing similar applications were examined in order to make a distinguished *original* project. There are many existing real-time applications that use `ANN`s to classify hand-drawn *digits* using the `MNIST` dataset, but almost none that use a `SOM` with competitive learning to cluster handwritten *letters* and display its topological map.

2.4 Project Requirements

This project firstly requires a user friendly front-end design, with interactive capabilities. HTML and CSS were both vital for this purpose. Secondly, a mathematical back-end model with significant computing power was necessary to handle large quantities of data, and the task was best suited for Python and it's libraries. Finally, to host the topological map, JavaScript's `D3.js` was perfect for this task as it required considerable data manipulation. More detailed use of technologies and programs are given in Chapter 5.

Chapter 3

Kohonen's Self-Organising Maps

3.1 Background

Pioneered in 1982 by Finnish professor and researcher Dr. Teuvo Kohonen, a self-organising map is an unsupervised learning model, intended for applications in which *maintaining* a topology between input and output spaces is of importance. The notable characteristic of this algorithm is that the input vectors that are close - similar - in high dimensional space are also mapped to nearby nodes in the 2D space. It is in essence a method for dimensionality reduction, as it maps high-dimension inputs to a low (typically two) dimensional discretised representation and conserves the underlying structure of its input space.

A valuable detail is that the entire learning occurs without supervision i.e. the nodes are *self-organising*. They are also called *feature maps*, as they are essentially retraining the features of the input data, and simply grouping themselves according to the *similarity* between one another. This has a pragmatic value for visualising complex or large quantities of high dimensional data and representing the relationship between them into a low, typically two-dimensional, field to see if the given unlabelled data has any structure to it.

3.2 Structure

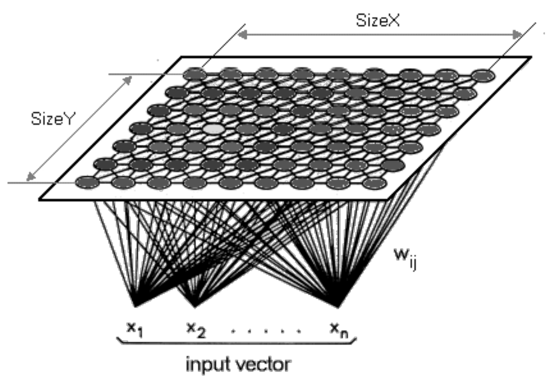
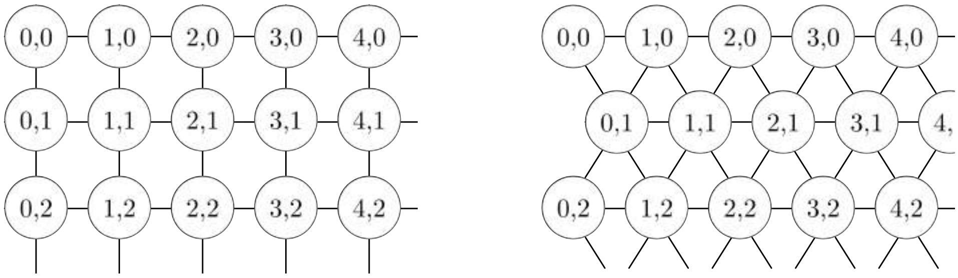


FIGURE 3.1: A Kohonen model

A SOM differs from typical ANNs both in its architecture and algorithmic properties. Firstly, its structure comprises of a single-layer linear 2D grid of neurons, instead of a series of layers. All the nodes on this grid are connected directly to the input vector, but *not to one another*, meaning the nodes do not know the values of their neighbours, and only update the weight of their connections as a function of the given inputs. The grid *itself is the map* that organises itself at each iteration as a function of the input of the input data. As such, after clustering, each node has its own (i, j) coordinate, which allows one to calculate the Euclidean distance between 2 nodes by means of the Pythagorean theorem.



(A) Rectangular (B) Hexagonal

FIGURE 3.2: Kohonen network's nodes can be in a rectangular or hexagonal topology

3.3 Properties

A Self-Organising Map, additionally, uses competitive learning as opposed to error-correction learning, to adjust it weights. This means that *only a single node* is activated at each iteration in which the features of an instance of the input vector are presented to the neural network, as all nodes compete for the right to respond to the input. The chosen node - the Best Matching Unit (`BMU`) - is selected according to the similarity, between the current input values and all the nodes in the grid. The node with the smallest Euclidean difference between the input vector and all nodes is chosen, *along with its neighbouring nodes* within a certain radius, to have their position slightly adjusted to match the input vector. By going through all the nodes present on the grid, the entire grid eventually matches the complete input dataset, with similar nodes grouped together towards one area, and dissimilar ones separated.

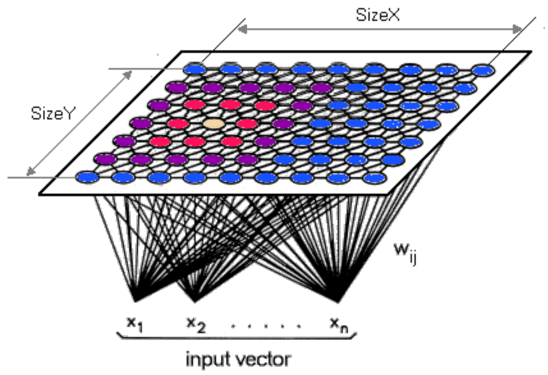


FIGURE 3.3: A Kohonen model with the BMU in yellow, the layers inside the neighbourhood radius in pink and purple, and the nodes outside in blue.

3.4 Variables

- t is the current iteration.

- n is the iteration limit, i.e. the total number of iterations the network can undergo.
- λ is the time constant, used to decay the radius and learning rate.
- i is the row coordinate of the nodes grid.
- j is the column coordinate of the nodes grid.
- d is the distance between a node and the BMU.
- $\vec{w}$ is the weight vector.
- $w_{ij}(t)$ is the weight of the connection between the node i, j in the grid, and the input vector's instance at iteration t.
- $\vec{x}$ is the input vector.
- $x(t)$ is the input vector's instance at iteration t.
- $\alpha(t)$ is the learning rate, decreasing with time in the interval [0, 1], to ensure the network converges.
- $\beta_{ij}(t)$ is the neighbourhood function, monotonically decreasing and representing a node i, j's distance from the BMU, and the influence it has on the learning at step t.
- $\sigma(t)$ is the radius of the neighbourhood function, which determines how far neighbour nodes are examined in the 2D grid when updating vectors. It is gradually reduced over time.

3.5 Algorithm

1. Initialise each node's weight w_{ij} to a random value
2. Select a random input vector $\vec{x_k}$
3. Repeat following for all nodes in the map:
   (a) Compute Euclidean distance between the input vector $\vec{x}(t)$ and the weight vector w_{ij} associated with the first node, where $t, i, j = 0$
   (b) Track the node that produces the smallest distance d
4. Find the overall Best Matching Unit (`BMU`), i.e. the node with the smallest distance from all calculated ones
5. Determine topological neighbourhood $\beta_{ij}(t)$ its radius $\sigma(t)$ of `BMU` in the Kohonen Map
6. Repeat for all nodes in the `BMU` neighbourhood:
   (a) Update the weight vector $\vec{w_{ij}}$ of the first node in the neighbourhood of the `BMU` by adding a fraction of the difference between the input vector $\vec{x}(t)$ and the weight $\vec{w}(t)$ of the neuron.
7. Repeat this whole iteration until reaching the chosen iteration limit $t = n$

Step 1 is the **initialisation** phase, while steps 2-7 represent the **training** phase.

3.6 Formulas

The updates and changes to the variables are done according to the following formulas:

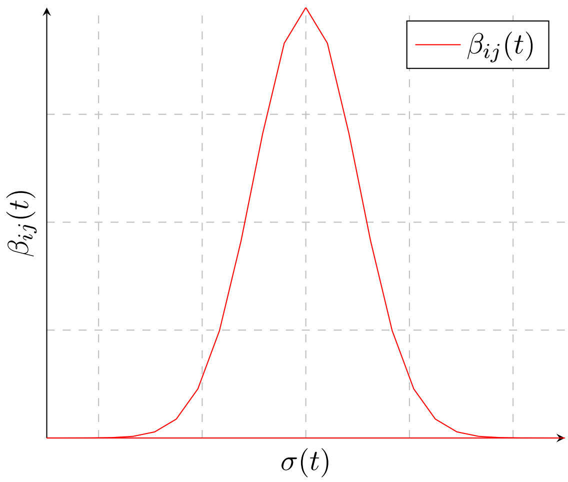
The weights within the neighbourhood are updated as:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha_i(t)[x(t) - w_{ij}(t)], \quad \text{or} \tag{3.1}$$

$$w_{ij}(t+1) = w_{ij}(t) + \alpha_i(t)\beta_{ij}(t)[x(t) - w_{ij}(t)] \tag{3.2}$$

The equation 3.1 tells us that the new updated weight $w_{ij}(t+1)$ for the node i, j is equal to the sum of old weight $w_{ij}(t)$ and a fraction of the difference between the old weight and the input vector $x(t)$. In other words, the weight vector is 'moved' closer towards the input vector. Another important element to note is that the updated weight will be proportional to the 2D distance between the nodes in the neighbourhood radius and the BMU.

Furthermore, the same equation 3.1 does not account for the influence of the learning being proportional to the distance a node is from the BMU. The updated weight should take into factor that the effect of the learning is close to none at the extremities of the neighbourhood, as the amount of learning should decrease with distance. Therefore, the equation 3.2 adds the extra neighbourhood function factor of $\beta_{ij}(t)$, and is the more precise in-depth one.



The radius and learning rate are both similarly and exponentially decayed with time:

$$\sigma(t) = \sigma_0 \cdot \exp(\frac{-t}{\lambda}), \quad \text{where } t = 1, 2, 3 \ldots n \tag{3.3}$$

$$\alpha(t) = \alpha_0 \cdot \exp(\frac{-t}{\lambda}), \quad \text{where } t = 1, 2, 3 \ldots n \tag{3.4}$$

The neighbourhood function's influence $\beta_i(t)$ is calculated by:

$$\beta_{ij}(t) = \exp(\frac{-d^2}{2\sigma^2(t)}), \quad \text{where } t = 1, 2, 3 \ldots n \tag{3.5}$$

The Euclidean distance between each node's weight vector and the current input instance is calculated by the Pythagorean formula:

$$||\vec{x} - \vec{w_{ij}}|| = \sqrt{\sum_{t=0}^{n} [\vec{x}(t) - \vec{w}_{ij}(t)]^2} \tag{3.6}$$

The BMU is selected from all the node's calculated distances as the one with the smallest:

$$d = \min(||\vec{x} - \vec{w_{ij}}||) = \min(\sqrt{\sum_{t=0}^{n} [\vec{x}(t) - \vec{w}_{ij}(t)]^2}) \tag{3.7}$$

Chapter 4

Data

4.1 Data

The data in this chapter only refers to the training and/or testing datasets that were used as inputs in the implemented Kohonen neural network in order to adjust its weights and find an optimal output at each iteration. They do **not** refer to the 3rd party feedback data explained in Chapter 10.

4.2 Ethical Use of Data

4.2.1 Real Non-Human and Synthetic Data

For the purpose of this project, only real non-human and synthetic data, specifically the `Iris`, auto-generated `RGB`, and `EMNIST` dataset were used, and these were freely available in the public domain. No human or any other type of data which requires approval from any professional ethical oversight body were ever utilised.

The `Iris` Flower dataset, created by biologist and statistician Ronald Fisher in 1936, is a published dataset containing a total of 150 training instances, each with four measurements of sepal length, the sepal width, the petal length and the petal width of the iris in question. There are 3 different iris classes, *Iris setosa*, *Iris virginica* and *Iris versicolor*, and the dataset contains 50 samples of each. The data belongs to University College Irvine's Machine Learning Repository, which contains a collection of databases that are often popular in Machine Learning communities. It represents real non-human data, measured and collected out in the field.

The `MNIST` dataset, a subset of the `NIST` database, contains the data derived from $60,000$ training and $10,000$ testing pictures of numerical handwritten digits by high school students and employees of the United States Census Bureau. A character's data is set in a 28 by 28 pixel format, giving 784 total values between 0-255, each one representing the grey scale intensity of that particular pixel. It is widely used for image processing programs and networks.

The extended MNIST, or `EMNIST` dataset, follows the same format and conventions, but also contains data of upper and lower-case alphabets, along with the digits present in the `MNIST`.

Both of these are in the public domain, and freely available in Matlab or binary data format, which can be converted to `.csv` or `.txt` files. For this project, the data was downloaded directly in .csv format from Kaggle, a popular Data Science and Machine Learning platform website, recently acquired by Google.

Full references and samples of these datasets can be found in the Bibliography and Appendix B at the end of this document. A copy of the data was uploaded to my University server, as a secure backup behind firewall.

4.2.2 Human Participation

For the realisation of this project, no human participation was involved, and therefore no permissions or approvals were required. The program is based on data already publicly available since many years, and only requires human interaction during the evaluation and usage phase, for which the consent form has been appended.

Chapter 5

Design

This chapter describes how the overall software and system was planned to work and interact with all of it's moving components, by giving an in depth explanation about the flow of data.

5.1 Software Technologies

The following table lists out the technologies, languages, libraries and frameworks used to implement this project in its entirety.

Tasks	**Technologies**	**Libraries**
Implementing Kohonen SOM	- Python	- NumPy - Pandas - Matplotlib - Argsparse
Training the network with synthetic data	- Python	- Random RGB data - Iris Database - EMNIST Database
Implementing web application to host SOM	- HTML - CSS - JavaScript - Flask	- jQuery - AJAX - Bootstrap
Displaying topological map and general model response	- JavaScript	- D3.js
Hosting and backing up source code	- Github	- Git
Hosting the website and model	- Web server	- Localhost - University server - Github.io page

TABLE 5.1: Programming languages, technologies, and libraries used for different tasks in this project.

5.2 Data Structures

5.2.1 Logical Sequence

Training

The following is the sequence of events to **train** the neural net:

1. Input image

2. Feature Extraction and Preprocessing
3. Learning and Recognition using `SOM`:
   (a) Initialise network
   (b) Present input
   (c) Best node wins via competitive learning
   (d) Update weights accordingly
   (e) Return winning node
4. Output result
5. Match output with labelled data
6. Output data
7. Repeat with different input

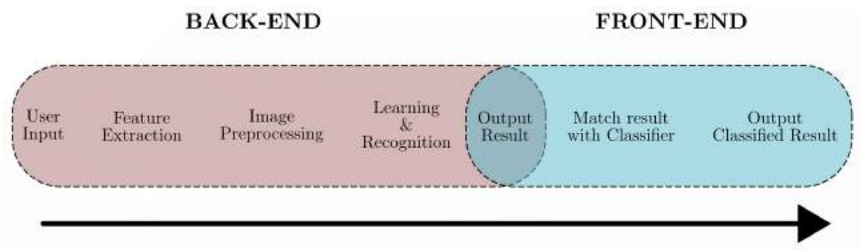


FIGURE 5.1

Testing

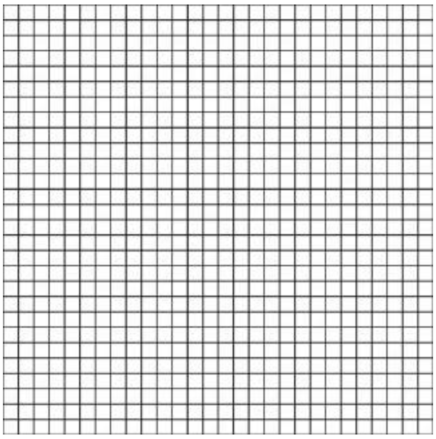
The following is the sequence of events to **test** the neural net:

1. Input image
2. Feature Extraction and Preprocessing
3. Recognition using `SOM`:
   (a) Initialise network
   (b) Present input
   (c) Return winning node
4. Output result
5. Match output with labelled data
6. Output data

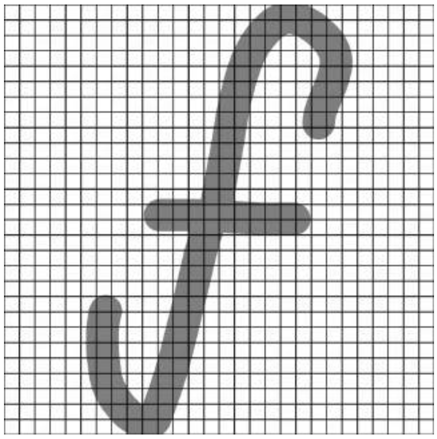
5.2.2 Image to Data Conversion

The `EMNIST` dataset contained images of handwritten characters, with each image being 28x28 pixels. Similarly, the user's input drawing had to be converted to a numeric matrix as well, based on the where the user had drawn on the canvas. A 1D vector of 784 numbers was used to convert an image to a list of greyscale black and white values in a 784-dimension array.

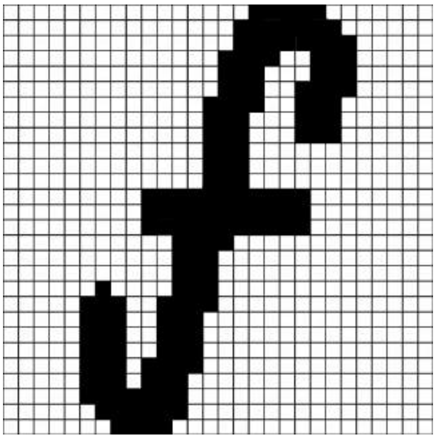
The following images show the planned sequence of image processing, to be implemented in a JavaScript front-end canvas, and its data transferred to the Python back-end.



(A) Empty grid



(B) Character drawn on grid



(C) Pixelised grid

$$\begin{bmatrix}
0&0&0&0&0&0&0&1&1&1&1&0&0&0\\
0&0&0&0&0&0&1&1&1&1&1&0&0&0\\
0&0&0&0&0&0&1&1&1&1&1&1&0&0\\
0&0&0&0&0&0&0&1&1&1&0&1&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&1&1&1&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&1&1&0&0&0&0\\
0&0&0&0&1&1&1&1&1&1&0&0&0&0\\
0&0&0&0&1&1&1&1&1&1&0&0&0&0\\
0&0&1&1&0&0&1&1&0&0&0&0&0&0\\
0&0&1&1&0&0&1&1&0&0&0&0&0&0\\
0&0&1&1&0&1&1&0&0&0&0&0&0&0\\
0&0&0&1&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&0&0&0&0&0&0&0
\end{bmatrix}$$

(D) Corresponding matrix

FIGURE 5.2: Sample hand-drawn input character converted from front-end canvas stroke to individual pixel data values.

5.3 System Design

5.3.1 UML Class Diagram

Below is the original UML class diagram, employing HTML, CSS, JavaScript, and Python files to implement both ends of the project.

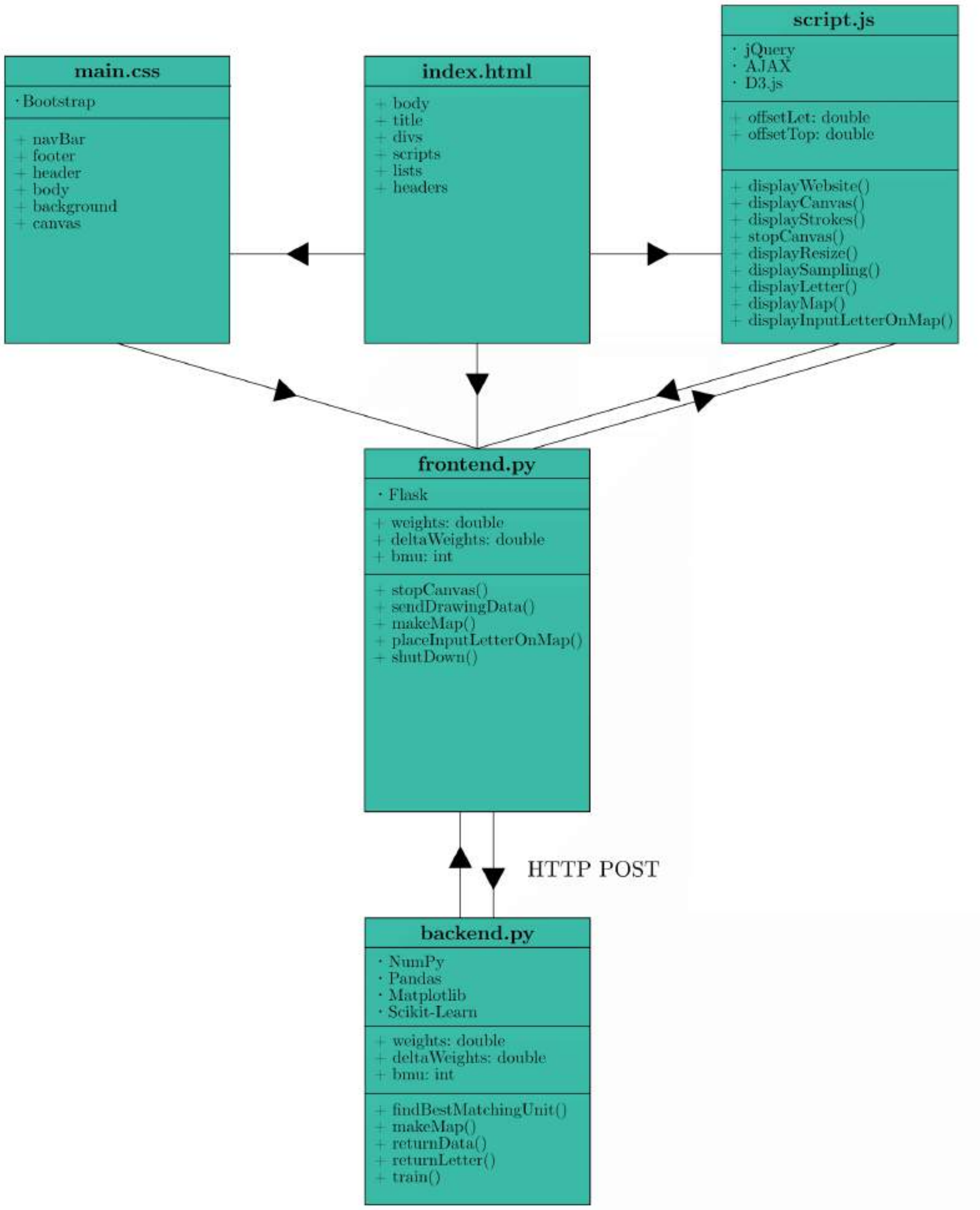


FIGURE 5.3: Use-Case Diagram

5.3.2 Use-case diagram

The following is a sample use-case diagram for the original user interface, which was later slightly altered, rendering this diagram obsolete.

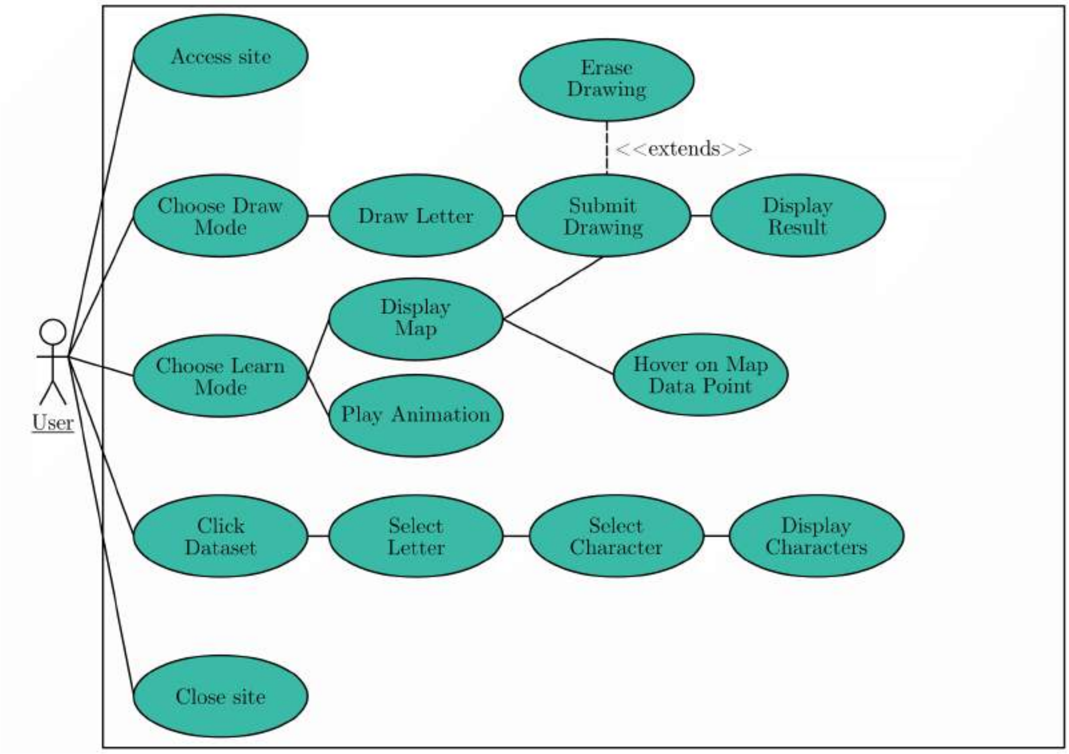


FIGURE 5.4: Use-Case Diagram

5.3.3 Use-case descriptions

The use-case descriptions for the given use-case diagram can be found in full in the Appendix E.

5.3.4 System boundary diagram

Below is the system boundary diagram for the both mobile and standard web users:

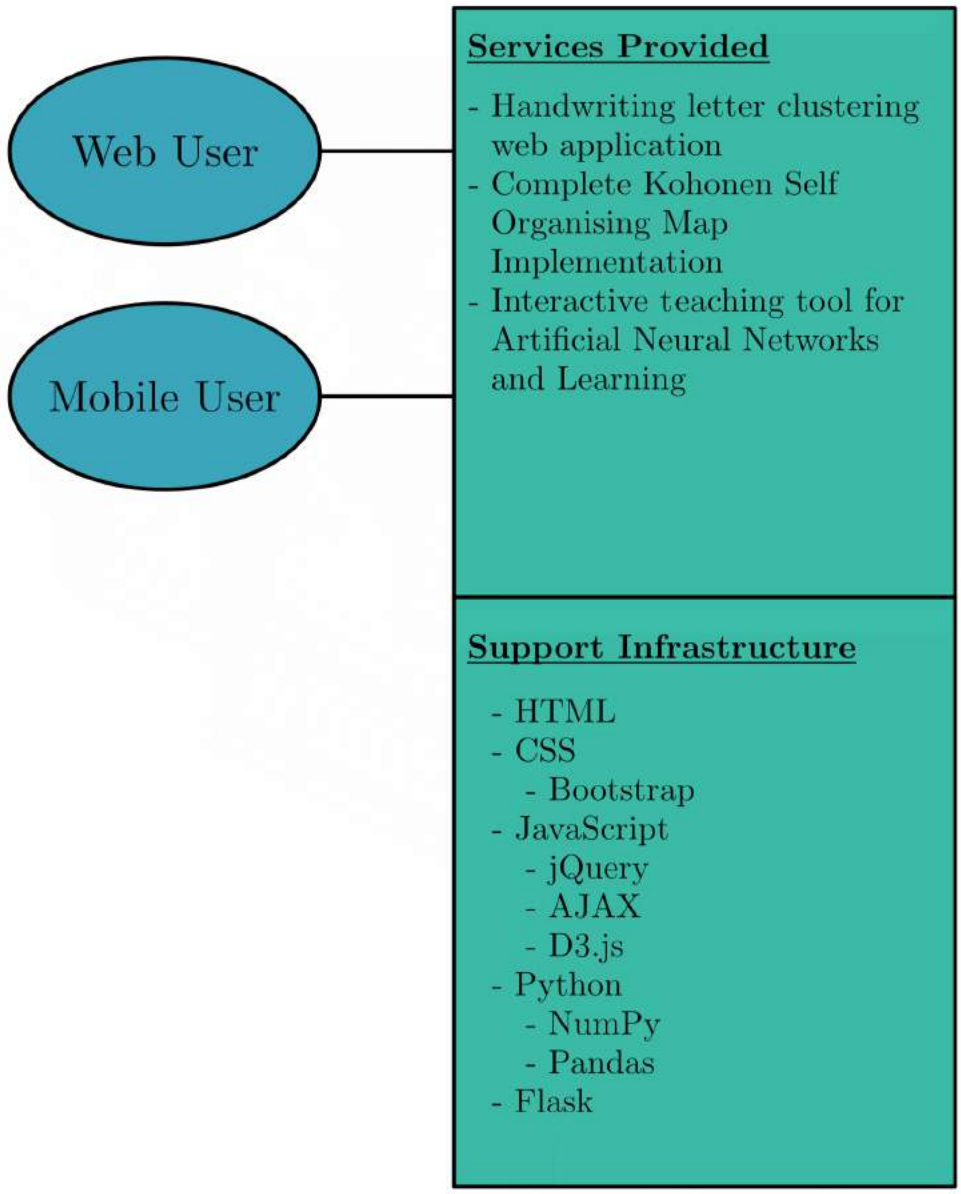


FIGURE 5.5: System boundary diagram

5.3.5 Sequence Diagram

The following is the sequence diagram when the user chooses the 'draw' option.

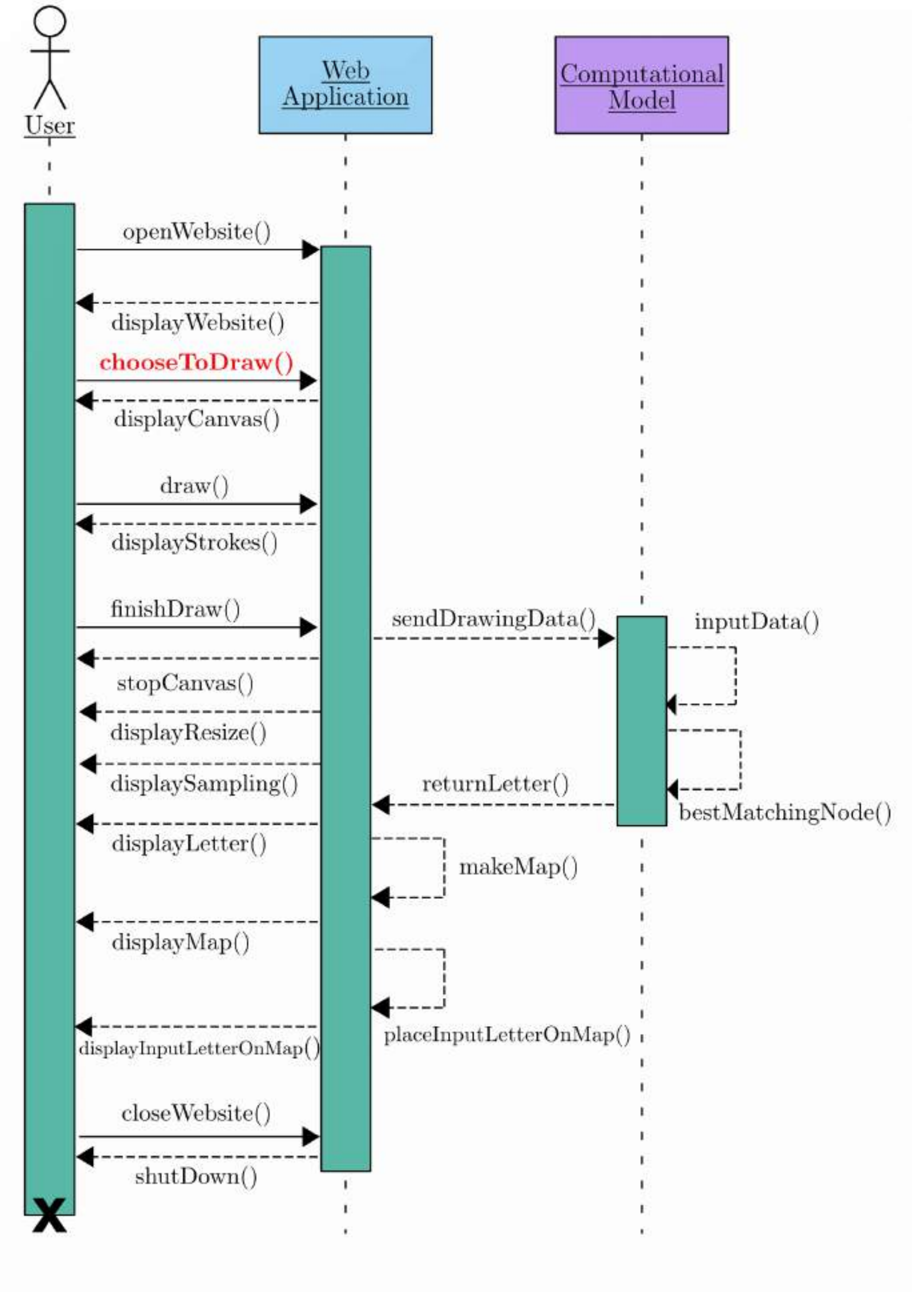


Figure 5.6: Sequence diagram for the drawing page

The sequence diagram can be broken down to the following detailed order of steps:

1. *openWebsite()*: user access the website
2. *displayWebsite()*: website content is displayed to user
3. *chooseToDraw()*: user chooses the 'draw' option button
4. *displayCanvas()*: website displays the drawable `GUI` canvas
5. *draw()*: user inputs on the canvas with his mouse or finger
6. *displayStrokes()*: website shows the strokes the user is drawing in real-time
7. *finishDraw()*: user submits his drawing
8. *sendDrawingData()*: website sends the drawing data's pixel values to the computational model as an array of integers or doubles
9. *stopCanvas()*: website stops displaying a drawable canvas to the user
10. *inputsData()*: model is fed the user's drawn data array
11. *bestMatchingUnit()*: computational model finds the best matching unit
12. *returnLetter()*: model returns the highest similarity letter's index
13. *displayResize()*: website shows the user the re-centring and re-sizing of his drawing
14. *displaySampling()*: website down-samples the user input drawing
15. *displayLetter()*: the corresponding letter with the highest similarity to the input drawing is displayed to the user
16. *makeMap()*: the topological map's data are arranged in arrays to be shown
17. *displayMap()*: the map is shown to the user using front-end graphics and the data from the array
18. *placeInputLetterOnMap()*: calculate where the user input would be placed on the map by sorting it in the array containing the other value points
19. *displayInputLetterOnMap()*: user's input letter is shown where it would belong on the map
20. *closeWebsite()*: user closes the website
21. *shutDown()*: the web application shuts down

The following is the sequence diagram when the user chooses the 'learn' option.

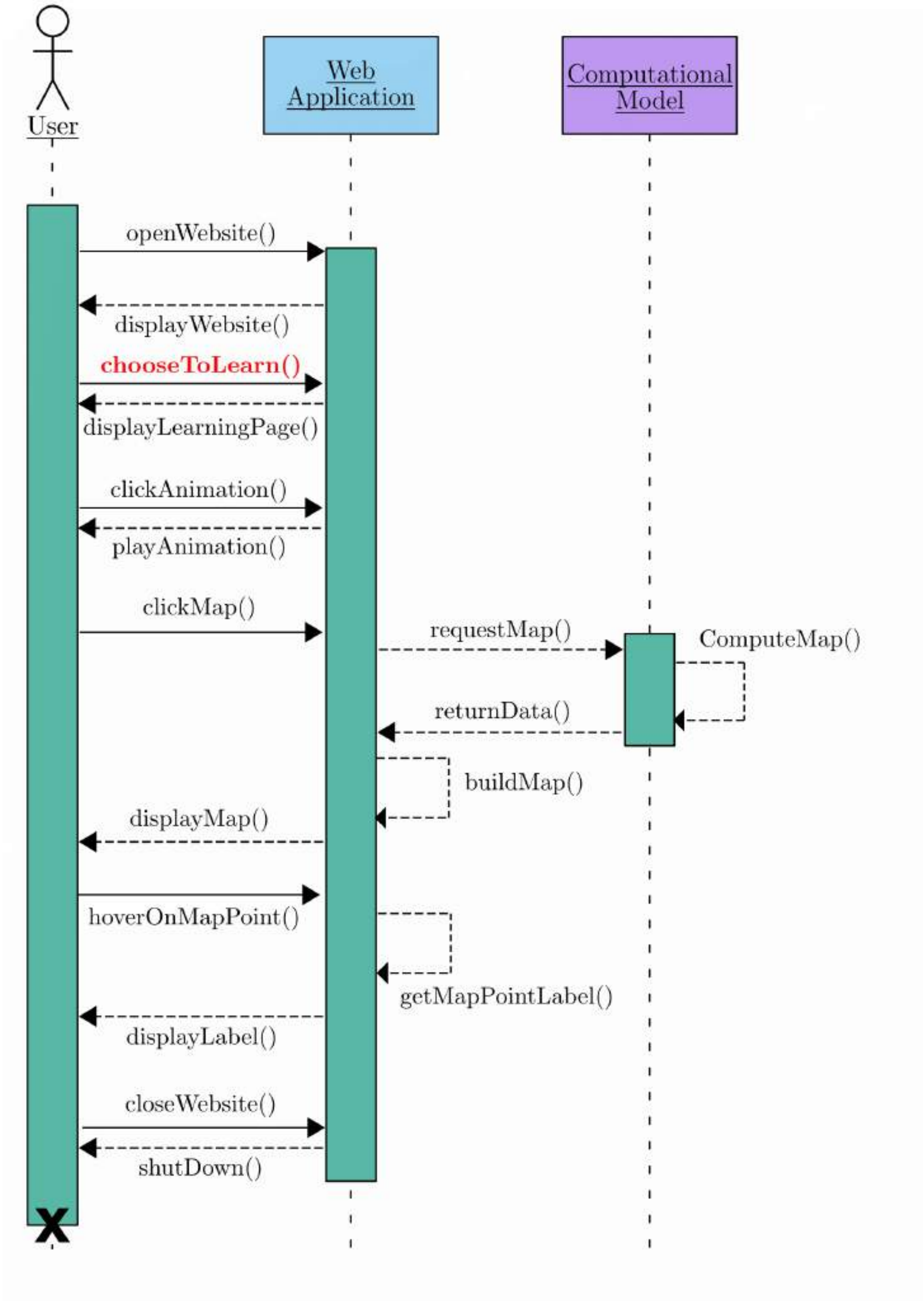


FIGURE 5.7: Sequence diagram for the learning page

The sequence diagram can be broken down to the detailed order of steps:

1. *openWebsite()*: user access the website
2. *displayWebsite()*: website content is displayed to user
3. *chooseToLearn()*: user chooses the 'learn' option button
4. *displayLearningPage()*: website displays 'learn' page
5. *clickAnimation()*: user presses play on an animation
6. *playAnimation()*: website plays the animation
7. *clickMap()*: user requests to open or see the topological map of the training set data
8. *requestMap()*: website requests the map from the computational model
9. *computeMap()*: computational model computes the map
10. *returnData()*: computational model returns the data of the map in a hash
11. *buildMap()*: website builds the map using the hash and its data
12. *displayMap()*: website displays the map to the user
13. *hoverOnMapPoint()*: user hovers on a specific map point
14. *getMapPointLabel()*: data for that specific point is fetched in the hash using the key
15. *displayLabel()*: data for that specific point is displayed
16. *closeWebsite()*: user closes the website
17. *shutDown()*: the web application shuts down

The sequence diagrams attempt to illustrate the interaction between user(s) and the pages via the computational model, and the flow of events as they happen.

5.4 Algorithm Design

The next few sub-sections contain key examples of pseudo-code and on how the interaction between components was planned.

5.4.1 Self-Organising Map

The Self-Organising Map is to be generated by the python computational model at the back end, which adjusts the network's weights during training with synthetic data and cluster similar inputs together.

1. Setup
   (a) Import necessary libraries
   (b) Create virtual environment
   (c) Create required dataframe to contain input values
   (d) Choose parameters: `SOM` size, learning parameters
   (e) Create grid
2. Normalisation
   (a) Normalise input data vectors

  - RGB: 3 vectors with values from 0 to 255.
  - Greyscale: single vector with values will be from 0 to 255.
  - Black and white: binary 0 or 1 values.

3. Learning

   (a) Initilise nodes' weights to random values

   (b) Select Random Input Vector

   (c) Repeat following for all nodes in the map:

      i. Compute Euclidian Distance between the input vector and the weight vector associated with the first node

      ii. Track the node that produces the smallest distance

   (d) Find the overall Best Matching Unit (`BMU`), i.e. the one with the smallest distance of all the nodes

   (e) Determine topological neighbourhood of `BMU` in the Kohonen Map

   (f) Repeat for all nodes in the `BMU` neighbourhood:

      i. Update the weight vector of the first node in the neighbourhood of the `BMU` by adding a fraction of the difference between the input vector and the weight of the neuron

   (g) Repeat this whole iteration until reaching the chosen iteration limit

4. Visualisation

   (a) Make use of `Matplotlib` for development, local testing and visualisation

   (b) Final visualisation for the user was to be done by the front end with `D3.js`

5.4.2 Canvas

The canvas on the front-end is the graphical user interface the user sees as the input area in which to draw his letter using a pointing devices such as a mouse, or by hand on a touch screen device. To achieve this, the canvas must have 4 event listeners for the mouse and then draw black pixels continuously along where the user inputs data in the correct events. The pseudo-code for the events can be summarised as shown below.

Algorithm 1 Mouse Move Event

```
if mouseMove then
    drawable ← true
    getCoordinates()
end if
```

Algorithm 2 Mouse Down Event

```
if mouseDown then
    drawable ← false
    getCoordinates()
end if
```

Algorithm 3 Mouse Up Event

```
if mouseUp then
    drawable ← false
end if
```

Algorithm 4 Mouse Out Event

```
if mouseOut then
    drawable ← false
end if
```

Algorithm 5 getCoordinates() Function

```
Previous_X ← Current_X
Previous_Y ← Current_Y
Current_X ← Event_X − canvas.offsetLeft
Current_Y ← Event_Y − canvas.offsetTop
if drawable ← true then
    draw()
end if
```

Algorithm 6 draw() Function

```
canvas.beginPath()
canvas.moveTo(Previous_X,Previous_Y)
canvas.lineTo(Current_X,Current_Y)
canvas.drawLine(Current_X,Current_Y)
canvas.stroke()
canvas.closePath()
```

Where:

- mouseDown is an `event` where the user only touches the screen, but does not yet draw, meaning only the fixed input coordinates are required.
- mouseMove is an `event` where the user draws on the screen, thereby continuously calling the draw function as the input position varies.
- mouseUp is an `event` where the user stops inputting.
- mouseOut is an `event` where the user leaves the canvas drawable area.
- getCoordinates and draw() are `method`s.
- drawable is a `boolean`
- offsetLeft is an `HTMLcanvas` property that returns 'the number of pixels that the upper left corner of the current element is offset to the left within the `HTMLElement.offsetParent` node'.
- offsetLeft is an `HTMLcanvas` property that returns 'the distance of the current element relative to the top of the offsetParent node'.
- Previous_X, Previous_Y, Current_X, Current_Y are `int`s about the input coordinates via the mouse or finger.
- beginPath(), moveTo(), lineTo(), drawLine(), closePath() are all `HTML` methods that reference the `canvas` tag.

Chapter 6

Front-End

6.1 Realisation

This chapter presents a comprehensive and in-depth review of how each section of the entire project, and its many components, were implemented in the chronological order, coupled with the obstacles and their respective solutions encountered during the realisation. Each part's design, structure and technical aspects are thoroughly examined and their net utility assessed.

The front-end was the first section to be implemented, with the HTML, CSS, JavaScript all developed more or less simultaneously, requiring a substantial mix of various libraries, tools, and an abundant amount of adjustments. The aesthetics of a website is a prominent part of its look and feel, and was thus carefully considered and and constructed as described below.

6.2 Bootstrap

6.2.1 Review

The Bootstrap documentation was formally reviewed to consider all the possible components such as navigation bars, footers, and headers, which could serve a purpose as part of the website and add to the UI/UX, without feeling contrived. This took precedence over writing the HTML, as a clear idea of what tools and objects were being used from the ground up was required before building the system, as any changes at a later stage would only be detrimental. The fact that newer versions of Bootstrap are continuously being released needed to be kept in mind. This project was specifically built using `Bootstrap v4.0.0`.

6.2.2 Integration

Adding the Bootstrap framework onto a project can be done in several ways. A package manager such as `npm`, `Bundler`, `RubyGems`, or `Composer` can be used to download and compile the source files. Alternatively, the compiled or source files, which contain the the minified CSS bundles and JavaScript plug-ins, can be manually downloaded and dropped into the project's directory. However, both approaches require meticulousness. Messy file management can simply be avoided by having the pre-compiled and cached version of Bootstrap's CSS and JS downloaded directly into the project as the index file is loaded.

An important side-effect of the CDN method is that an internet connection is therefore always required, even on localhost, to view the your project's files with the correct styling. On the other hand, the processing is done internally, and the correct lightweight, minified, and latest versions of the Bootstrap framework are downloaded. After testing all the different types, a slightly discrepancy between the automatic and manual versions was observed, for example in the native HTML buttons.

(A) Automatic CDN version (B) Manual download version

FIGURE 6.1: The different Bootstrap versions contain styling differences

Although these would anyway be overwritten with Bootstrap styled buttons, the automatic cached version was preferred. Furthermore, it came with an extra layer of security than the manual version by means of the integrity and crossorigin reference. Both attributes define a mechanism by which user agents can verify that a fetched resource has been delivered with the expected data. The former is to allow the browser being used to check the source file, to ensure that the code is never loaded if the source has been been manipulated. The latter ensures that origin credentials are checked.

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap
    /4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058
    RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous
    ">
```

LISTING 6.1: Bootstrap script CDN reference

Bootstrap is dependent on jQuery and Popper.js, and they both *must* be placed *before* the Bootstrap script. They are used for various features such as a colour change when a mouse hovers over a button.

```
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity
    ="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/
    GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd
    /popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/
    ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/
    bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/
    JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></
    script>
```

LISTING 6.2: jQuery, Popper.js and Bootstrap.js reference

With the Bootstrap CSS, JS, jQuery and Popper.js along with a couple more minor elements , all the necessary pre-requisites are in place, allowing for full modern Bootstrap `V4` integration.

6.2.3 Colour Theme

The first task was to fully define the look and feel of the web-application which is largely contingent on the selected colour theme and font. A peachy, light coloured background (`#fff2e7`) was instinctively chosen for it's soothing effect on the eyes. For all the other DOM objects, Bootstrap's limited handful of colours[1] were bold and complementary to both the background and one another. Attractive and user-friendly, they also maintained consistency across all pages. This was preferable over manually hand-picking a colour for a new item every time. More importantly, they natively worked for all Bootstrap components, simply by adding the colour tag to the DOM object's class names.

FIGURE 6.2: A handful of strong colour options provided natively in the Bootstrap framework and their corresponding class names

A paragraph could simply be:

```
<p class="chosen-class-name text-info">Blue paragraph, with a <a href="
    link.html" class="text-danger">red link</a></p>
```

LISTING 6.3: Class colour code

And it would produce the following output on an HTML page with two distinct colours:

FIGURE 6.3: Output

The point being that the colouring works despite the fact that there are two class names. Bootstrap's colour name tags can simply be appended to the class independently named by the developer, allowing further styling modifications in the CSS file. The modular streamlined nature of Bootstrap and its lack of dependencies is what makes it easy to grasp and work with.

6.2.4 Header

A fixed position navigation bar[2] containing the website title throughout all pages was indispensable to maintain consistency and a reference. A 'Home' and 'About' button were added to the fringes of the navbar as well. The title evolved from a lengthy *Kohonen Self-Organising Maps: Pattern Recognition and Clustering from the EMNIST*

[1] Bootstrap Getting Started. https://getbootstrap.com/docs/4.0/getting-started/theming/#theme-colors. (Accessed on 04/02/2018).

[2] Bootstrap Navbar. https://getbootstrap.com/docs/4.0/components/navbar/. (Accessed on 04/02/2018).

database over several vertical lines to a simple *Kohonen Self-Organising Maps*.

```
<nav class="navbar fixed-top bg-success">
  <a class="order-1 nav-item nav-link active" a style="color:white" href
    ="/">Home</a>
  <a class="order-2 align-self-center nav-item" a style="color:white"><b
    >Kohonen Self-Organising Maps</b></a>
  <a class="order-3 nav-item nav-link" a style="color:white" href="about
    ">About</a>
</nav>
```

LISTING 6.4: Header code

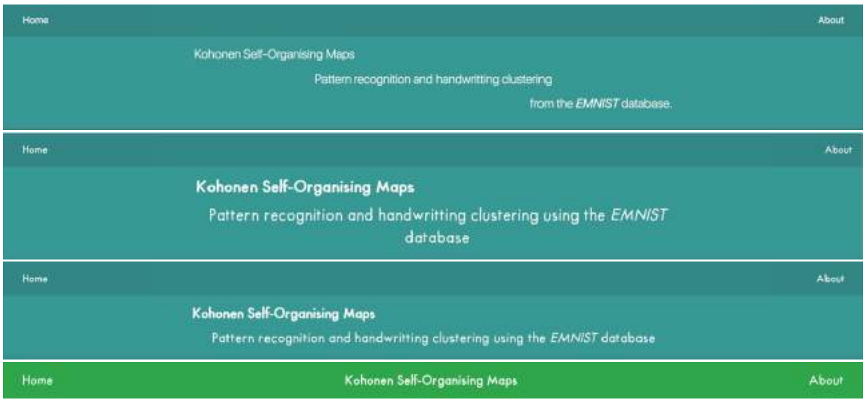


FIGURE 6.4: Header evolution from prototype to final implementation

6.2.5 Footer

Originally, a two part footer was envisioned to be put in a fixed position for all pages, similar to the navbar, containing a line on the aim of the website coupled with the website developer's name. This was disregarded early on for taking up too much screen size, and a smaller single footer was used for a large part of the development before being discarded too. The footer was pointless if it didn't contain any new information relevant to each page.

Thus, the choice was between having a pagination or progress bar. The former was first implemented and tested, but eventually disposed off as its white background did not fit into the colour scheme, and the total number of pages was not known yet. Instead, a thin, slick and *dynamic* progress bar was developed which was consistent with the colour scheme. It has all five colours, one for each section, and progressively fills each one out until reaching the last web-page.

```
<div class="footer">
  <div class="fixed-bottom">
    <div class="progress">
      <div class="progress-bar" role="progressbar" style="width: 20%"
    aria-valuenow="15" aria-valuemin="0" aria-valuemax="100"></div>
      <div class="progress-bar bg-success" role="progressbar" style="
    width: 20%" aria-valuenow="30" aria-valuemin="0" aria-valuemax="100">
    </div>
      <div class="progress-bar bg-info" role="progressbar" style="width:
     20%" aria-valuenow="20" aria-valuemin="0" aria-valuemax="100"></div>
```

```
      <div class="progress-bar bg-warning" role="progressbar" style="width: 10%" aria-valuenow="25" aria-valuemin="0" aria-valuemax="100"></div>
      <div class="progress-bar bg-danger" role="progressbar" style="width: 0%" aria-valuenow="15" aria-valuemin="0" aria-valuemax="100"></div>
    </div>
  </div>
</div>
```

Listing 6.5: Footer code

Figure 6.5: Footer

6.2.6 Flex

On top of being dynamic, it was equally important that all the web-pages be *flexible*, as modern screens come in all shapes and sizes. One of Bootstrap v4's crowning features was utilised: `flex`[3] . This made sure the header and footer were responsive to a certain degree to the width of the page, depending on the user's screen size and resolution.

```
<div class="d-sm-flex flex-wrap fixed-top">
  <nav class="navbar fixed-top bg-success">
    ...
  </nav>
</div>
```

Listing 6.6: Flex code

Figure 6.6: Header with flex implementation

6.2.7 Columns

One of Bootstrap's foundational feature is its columns grid structure[4] based on flexbox. It allows for responsive design directly in each separate class. Essentially a page's main area can be broken down into columns of a preferred size, allowing for easy manipulation and alignment of inline DOM objects.

```
<div class="col-lg-12">
  ...
</div>
```

Listing 6.7: Column code

[3] Bootstrap Flex. https://getbootstrap.com/docs/4.0/utilities/flex/. (Accessed on 04/02/2018).

[4] Bootstrap Columns Grid Layout. https://getbootstrap.com/docs/4.0/layout/grid/. (Accessed on 04/02/2018).

There is also a very useful option where the columns are **offset** by a chosen column size.

```
<div class="col-lg-8 offset-lg-2">
  ...
</div>
```

LISTING 6.8: Offset column code

6.2.8 Buttons

Bootstrap offers straightforward buttons[5] in various sizes, all of which can be coloured in any of the aforementioned tints. Small and normal sizes were used according to their importance and the available space in that particular context.

```
<button type="button" class="btn btn-lg"><a href="#">Button Title</a></
    button>
```

LISTING 6.9: Buttons code

6.2.9 Cards

Cards[6] were flexible content containers perfect for proposing the user with options. Each one of them was used for one of the three datasets, highlighting each one's features regarding their dimensionality and volume. Once again, different colours were employed to maintain colour scheme and distinguish one from the other by supposed 'difficulty'.

```
<div class="card-deck">
  <div class="card text-white bg-info mb-3" style="width: #px;">
    <img class="card-img-top" src="#" alt="##">
    <div class="card-body">
      <h5 class="card-title">Card Title</h5>
      <p class="card-text"></p>
      <a href="#" class="card-link">...</a>
    </div>
    <div class="card-footer">
      ...
    </div>
  </div>
</div>
```

LISTING 6.10: Single card code

6.2.10 jQuery

The jQuery integrated at the set-up phase with the crossorigin and integrity layer was the slim version[7], which is a streamlined and shortened version of the full jQuery. As

[5] Bootstrap Buttons. https://getbootstrap.com/docs/4.0/components/buttons/. (Accessed on 04/02/2018).

[6] Bootstrap Cards. https://getbootstrap.com/docs/4.0/components/card/. (Accessed on 04/02/2018).

[7] Bootstrap jQuery. https://getbootstrap.com/docs/4.0/getting-started/download/#bootstrapcdn. (Accessed on 04/02/2018).

it was revealed after intense debugging, this version is incompatible with Ajax and was the cause of several mysterious bugs. Therefore, in some pages it was replaced with the full version jQuery instead, at the expense of the aforementioned extra cover of security.

6.3 HTML

Once sufficient knowledge was gathered through research on the tools and components, and correctly integrated onto the foundations of the website, the main structure and content of each page had to be filled as an `.html` file. This was, like many other parts, a continuous iterative process, evolving till the very end. Thus, it was important to spend time designing a satisfactory template which could be used as a basis for all pages.

6.3.1 Template

The template consisted of bringing together all the previously researched elements, such as the background, buttons, cards, nav bar and progress bar, onto a single flexible page built with the columns layout structure and flexbox. Additionally, the minor but obligatory touches for a HTML5 page were also required. This encompassed the meta-information (such as the charset, and author's name, date, ID), the cloud bootstrap and then the personal CSS files reference, the favicons themselves, and finally the page's title just in the file's header section.

```
<head>
  <!-- META -->
  <meta charset="UTF-8"/>
  <meta name="author" content="...">
  <meta name="ID" content="...">
  <meta name="Date" content="2018-02-25" scheme="YYYY-MM-DD">
  <meta name="viewport" content="width=device-width, initial-scale=1,
    shrink-to-fit=no">

  <!-- Cloud Bootstrap CSS -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap
    /4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058
    RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous
    ">

  <!-- Main CSS -->
  <link href="static/css/main.css" rel="stylesheet" type="text/css">

  <!-- Favicons -->
  <link rel="..." sizes="..." href="...">

  <title>...</title>
</head>
```

Listing 6.11: HTML header code

The structure of the main body consisted first and foremost of a JavaScript `onload()` function in the HTML body declaration tag itself, followed by the nav bar, main content container, footer, and lastly a list of JavaScript declarations in the correct order. All script lists contained a `<noscript>` error message in case the user did not have JavaScript enabled, and were then followed by any personally developed scripts, before ending with the mandatory 3rd party jQuery, Popper.js and Bootstrap.js code required for Bootstrap v4.

```
<body onload="setUp();">

  <!-- Nav bar -->
  <div class="d-sm-flex flex-wrap fixed-top">
    <nav class="navbar fixed-top bg-success">
      ...
    </nav>
  </div>

  <!-- Main -->
  <div class="main">
      <div class="col-lg-8 offset-lg-2">
        ...
    </div>
  </div>

  <!-- Footer -->
  <div class="footer">
    <div class="fixed-bottom">
      <div class="progress">
        ...
      </div>
    </div>
  </div>

  <!-- Scripts -->
  <noscript>Your browser does not support JavaScript which is required by Bootstrap 4 for the purposes of this wep-page.</noscript>

  <!-- Personal Scripts -->
  <script src="..." type="text/javascript" charset="UTF-8"></script>

  <!-- jQuery -->
  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>

  <!-- Popper.js -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>

  <!-- Boostrap.js -->
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>
</body>
```

Listing 6.12: HTML body code

Lastly, the whole head and body content should of course be enclosed in the standard html declaration tag.

```
<!doctype html>
<html lang="en">
  <head>
    ...
  </head>
```

```
  <body>
    ...
  </body>
</html>
```

Listing 6.13: HTML declarations

This was the designed framework used by all subsequent pages.

Figure 6.7: Front page template containing Bootstrap based nav bar, column grid layout, main text container, button, progress bar and general colour theme.

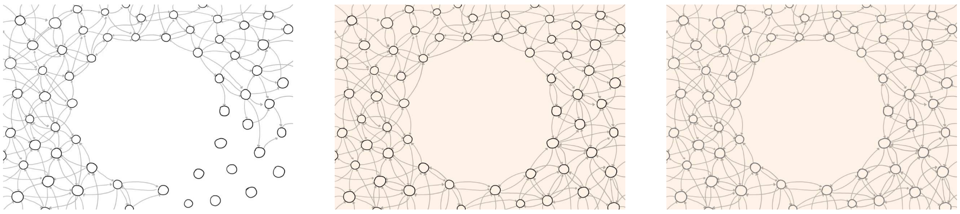
6.4 Art

In order to add a personal aspect to the website, hand-drawn art was added to the website to complement the digital features. These were drawn with a stylus on a Wacom tablet[8] linked directly to Adobe Photoshop, and exported with a `.png` image. These can be found in full size in Appendix C at the end of this document.

6.4.1 Background Nets

As the background felt too bare simply as a monotone colour, an artistic rendering of neural networks was designed to add more focus towards the centred text. The original prototype contained black outlines for each node, which took away attention from the text, and was subsequently altered to a version with grey outlined nodes. The image colour was also switched from white to the one used for the original background image, as the former would go *on top* of the latter.

[8] Wacom Intuos Pro-Medium Paper Edition Tablet



(A) Incomplete prototype (B) Complete prototype (C) Final design

FIGURE 6.8: Background art evolution

6.4.2 Volume buttons

To give users a choice to play background sound was planned from the start, and various volume buttons were designed. Eventually, a boolean design was chosen, thus only requiring two images (mute and un-mute), as users can increase or decrease volume directy from their devices.

FIGURE 6.9: Shadow volume buttons

FIGURE 6.10: Fill volume buttons

FIGURE 6.11: Dash volume buttons

6.5 CSS

The Bootstrap style sheet added a lot of components and helped with standardising the layout by making it easy to be manipulated and built upon in HTML files, and the drawn art images added a unique touch to those pages. Nonetheless, a personal `main.css` styling sheet was still imperative to meticulously refine the spaces, positioning and sizes of the DOM objects in each page in-depth. The following section details the principal elements of the complete CSS file.

6.5.1 Fonts

A font was as important to the website as a colour scheme, as it would naturally determine the tone and way information was *communicated* to the user. Initially, Google's modern Roboto font was deemed adequate for task, however it did not fit well with the drawn art. After some research on a number of 'handwritten' type of fonts, `FuturaHandwritten` font was singled-out for being user-friendly and complementary to both the art and ideology of the website. All textual content on the website was typeset using only this font by declaring it in the `@font-face` at the very top the `main.css`.

```
@font-face{
  font-family: 'FuturaHandwritten';
  font-style: normal;
  font-size: 25px;
  src: url('../Fonts/Futura/FuturaHandwritten.ttf') format('truetype');
}
```

LISTING 6.14: Font declaration

6.5.2 Background

The inclusion of the background network art was contingent on the density of the information on the page and space it took up. The title page, cards selection, and the 'About' page were easy candidates to include the background art, but the others were better off without it. A painless and elegant solution to this problem was to have the art image declared as the background for all pages, and then to simply create a different `noBackGround` class in CSS, and declare in the HTML `<body>` tag of the pages that not require the artwork.

```
body {
  margin:0;
  padding:0;

  height:100%;
  min-height: 100%;

  background-image: url("../images/nets/Net4.png");
  background-position: center;
  background-size: cover;
  background-repeat: no-repeat;
  background-color: #fff2e7;
}
```

LISTING 6.15: Background art declaration for all pages

```
.noBackGround {
  background-image:none
}
```

LISTING 6.16: No background class

6.5.3 Positioning, Padding and Alignment

After much deliberation, un-scrollable pages were deemed preferable to the alternative, as the pages were designed to be able to contain the content in a single view. Additional text could always be added with the aid of a JavaScript function, in which selected sentences were iterated through the same space on-screen.

```
body {
  overflow-x: hidden;
  overflow-y: hidden;
}
```

LISTING 6.17: Un-scrollable pages

Moreover, this allowed for easier manipulation of the header and footer. Both needed to stay in their place and never move regardless of the user interaction. The header was made sure to start from completely on top and be in its natural position, while footer's position was made absolute and without any content below it.

```
.header {
  top: 0;
  width: 100%;
}
```

LISTING 6.18: Header position

```
.footer {
  position: absolute;
  bottom: 0;
  width: 100%;
}
```

LISTING 6.19: Footer position

Practically each DOM object was almost always given a certain amount of padding on all 4 sides, and its text aligned centrally.

```
.objectClass {
  padding-top: 20px;
  padding-bottom: 20px;
  padding-left: 20px;
  padding-right: 20px;
  text-align: left;
}
```

LISTING 6.20: Sample object padding and alignment

FIGURE 6.12: Cover page with art, Bootstrap and personal CSS

6.6 JavaScript

Finally, the JavaScript is what makes the page interactive with the users, and distinguishes the website from a fancy but passive booklet or sideshow. Several different scripts were used and are outlined below.

6.6.1 Draw.js

`Draw.js` was a personal script used to initialise various variables on every page, and add user interactivity. It's principal focus was the development of the `canvas` usable by a user to input his own hand-drawn character.

The canvas was initialised in the `setUpCanvas()` function, which would get the canvas's initial values from the HTML page.

```
info = document.getElementById('status');
canvas = document.getElementById('myCanvas');
ctx = canvas.getContext('2d');
len = canvas.width;
```

Listing 6.21: Canvas Code

Simultaneously it would also call four other 3 main canvas drawing functions, corresponding to the ones detailed in Section 5.4.2.

```
// Calls
setUpMouseCanvas();
setUpTouchCanvas();
setUpScrollEvents();
```

Listing 6.22: Canvas event functions

Each one of these functions allow the user to draw inputs with a mouse or even on a mobile device using a touchscreen. For such cases it was important to *disable auto scroll* when the user would start inputting his data. Boolean values were used to decide when the could or couldn't draw in the canvas.

```
// Prevent unintended touch scroll
document.body.addEventListener("touchstart", function (e) {
if (e.target == canvas) {
  e.preventDefault();
}
}, false);
```

Listing 6.23: Disable auto-scroll on touch devices

A mysterious issue here was a random offsetting on the X-axis of the drawn lines. Indeed, everytime a line was attempted to be drawn on the canvas, it would appear a few centimers to the left, often not visible on the canvas. This was later identified to be caused by Bootstrap's grid layout structure, in which `offset-columns` were used. To disentangle this issue jQuery's `this.offset` methods proved to be useful.

```
var offsetL = this.offsetLeft + $(this).parent().offset().left - 15;
var offsetT = this.offsetTop + $(this).parent().offset().top;
```

Listing 6.24: Correcting Bootstrap column's offset on the canvas

A simple way to clear the canvas when required was to draw a rectangle of the canvas's size on it everytime the relevant requesting button was pressed. However, an even smarter solution was implemented, which *re-initialised* the canvas' height and width, thus removing any drawn strokes.

```
canvasIndImage.width = canvasIndImage.width;
```

Listing 6.25: Clearing canvas

The bulk of the work went into developing a system which could intake more than a single input drawn in the canvas. This was perhaps a bit ambitious and not really necessary, but was taken on as a challenge early on nonetheless.

The first step was to get the user's entire data from the entire canvas. Then, each individual digit drawn in the canvas could be attempted to be seperated by iterating row-by-row through all the pixels containing any greyscale value. By adding each greyscale value which is continous or adjacent to a previous value, a number of arrays could be created corresponding to the total number of drawn characters. The number of continous drawn arrays can be kept track of with a simple variable. Once we have all the required arrays, they can each individually be processed by the Kohonen network.

A last step would be to re-size the values into a correct 28x28 format processable by the Kohonen Network. To do so, the image could be rescaled to 18x18 pixels, then centered, then re-scaled to the desired 28x28 pixel format. Depending on larger height or width.

A second canvas was utilised to show that the image had indeed been processed. The values were also normalised here so they didn't have to be done later in the backend. A simple log can be used to print out the re-sized canvas input values.

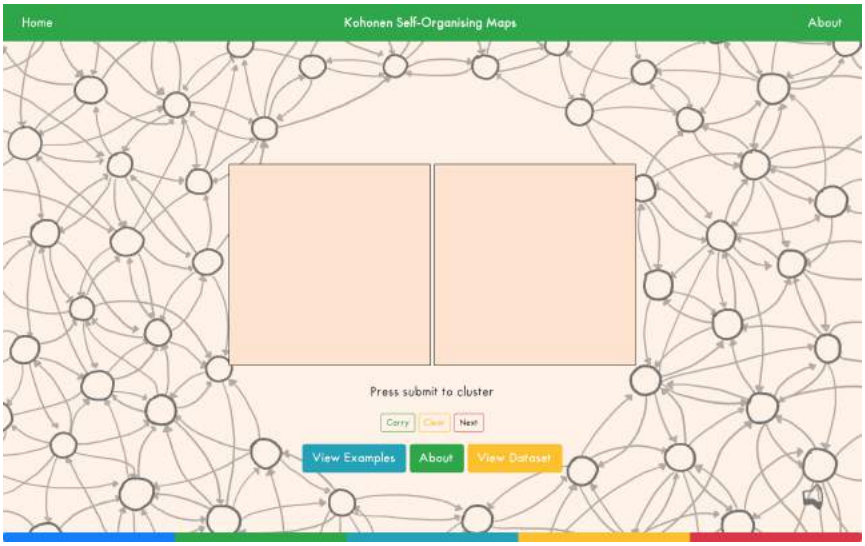


Figure 6.13: The implemented canvas

6.6.2 Howler.js

Howler.js is a popular and easy to use JavaScript library for audio manipulation. A simple working framework was developed for the system as a foundation to be easily expanded upon. It currently only contains a single audio `.mp3` file for all pages, but the groundwork for any expansion is set and easily implemented.

```
<body>
  <!-- Howler.js -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/howler/2.0.9/
    howler.min.js" type="text/javascript"></script>
</body>
```

LISTING 6.26: Importing howler.js via CDN

The current groundwork essentially consists of two JavaScript functions, `audioSetUp()` and `changeVol()`. The first method is called on through a set-up function directly embedded onto the `<body>` HTML tag on every single page, akin to the `noBackground` class in CSS.

```
<body onload="setUp();">
   ...
</body>
```

LISTING 6.27: Calling setUp() function

Its purpose was to simply have the .mp3 audio file load onto Howler.js, without playing, set at a low volume and ready to be played when asked. The second method, `changeVol()`, is a playing function with a boolean structure, that starts the audio when the user clicks upon the art icon on the front-end. The sound is played from the start if clicked on for the first time ever since loading the page, but at subsequent clicks simply alternates between muting and un-muting the audio which still ‘plays’ in the background. This was done by employing two boolean variables in an if-structure - one to see if the user had requested sound for the first time, and the other to check if the audio was currently muted or not. With these two variables all scenarios could be covered. The audio button art is also changed at each boolean call depending on its current muted or un-muted state.

```
function changeVol() {

  if (muted) { // Turning sound ON
    volIcon.src = "static/images/volume/shadow/3.png";

    if (initial) { // Start playing
      bgOST.play();
      initial = !initial;
    } else { // Resume playing
      bgOST.mute(false);
    }

  } else { // Turning sound OFF
    volIcon.src = "static/images/volume/shadow/1.png";
    bgOST.mute(true);
  }

  // Switch
  muted = !muted;
}
```

LISTING 6.28: Audio volume function

Chapter 7

Back-End

Although the front-end was enjoyable to implement, it was largely a cosmetic - albeit important - aspect coupled with a mark-up language. The back-end however, being the most demanding and time consuming task, is the real substance of this project. The first and foremost goal of this project was to implement a working mathematical Kohonen model, which would adapt to the given data, and could be adjusted according to a few modifiable variables. The following sub sections give an idea of all the different aspects that had to be tackled to implement such a model.

7.1 Software Design and Optimisation

The entire back-end is not simply an implementation of the Kohonen algorithm, as many variables have to be declared first, or input manually by the user according to the parameters they want. The following section goes through step-by-step, each fundamental component of the script.

First of all, the goal was to be able to explain Kohonen networks in layman's terms, and give insights on how various factors influence the convergence (or lack thereof) of the neural network. The factors to discuss included the volume and dimensions of the input data, the total number of classes, the effect of the learning rate, the neighbourhood function and its radius.

In and of itself, a single sample implementation did not feel sufficient to explain the variety of factors that affect the model, the subtle nuances of each parameter, and the broad range of different datasets that can be used for clustering.

It was decided therefore, to have **three** different implementations of the Kohonen artificial neural network, each one working with a different dataset and showcasing a distinct concept of the algorithm.

The first model would concretly introduce the concept of of multi-dimensional input vectors, by illustrating it with RGB vectors, which are easy to demonstrate and grasp, along with being low-dimensional (3D) but high-volume.

The second model would attempt to demonstrate the concept dimensionality reduction and touch upon the notion of topology conversation. The Iris dataset was ideal for this part, as its four dimensions are plotted on a 2D dimension space.

Finally, the last model would work on clustering similar handwritten OCR characters based on the `MNIST` and `EMNIST` dataset to emphasise the notion of topology preservation from a high to low dimension.

Model	**Dimensions**	**Volume**	**Illustrated Concept**
RGB	3	100	Multi-dimensionality
Iris	4	150	Dimensionality reduction
OCR	784	60,000	Topology conversation

TABLE 7.1: The attributes of each dataset

7.1.1 External Libraries

This project would not have been possible without crucial libraries: `Pandas` for large data handling and `NumPy` for mathematical operations and especially array restructuring. However, for the scope of this project, an obvious question is whether both were absolutely necessary. After all, being large libraries meant for similar purposes, they often overlap in their functionalities and both can perform sufficient arthimetic operations for the purposes of this project. Their distinguishing feature is actually their difference in speed and efficiency in dealing with different types of tasks. Each one has its pros and cons, and a big part of this section was to optimise the code in such a way that the best features of each library is used.

In Python, arrays are abstracted as `Lists`, NumPy uses `np.array()`, and Pandas employs `Dataframes`. Understanding the subtle differences between these three is essential, as they play a vital part in data processing and algorithmic optimisation of high-dimension high-volume inputs.

	Python	**NumPy**	**Pandas**
Import as	*native*	np	pd
Data Structure	list	array	dataFrame
Empty Declaration	[]	np.zeros((i,j))	pd.DataFrame()
Dimensions	1	n	n
Mutable	Yes	No	Yes
Starting Index	1	0	0
Iteration in loop	l[i]	np.array[i]	pd.iloc[i]
Appending	.append()	np.append()	pd.concat()
Time Complexity	$\mathcal{O}(1)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$
Sorting	l.sort	np.sort()	pd.sort()
Time Complexity	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
Length	len(l)	np.shape[0]	pd.shape

TABLE 7.2: Different aspects of Python lists, NumPy arrays and Panda data frames

There are several crucial elements to note that determine the flow of the script's development. Perhaps the most important one is that unlike NumPy's data structures, Python's native `list` is mutable. This means it can be declared as empty or of any size, and **keep on extending as new items are added**. It is a **dynamic** array, whereas both NumPy and Pandas are static, i.e. they require the developer to declare the array size beforehand, and then fill it up to the maximum declared limit. Furthermore, the time complexity for appending a value to a Python list using list.append() is simply $\mathcal{O}(1)$. NumPy is considerably slower because it declares a **new** array of the size of the sum of both arrays, and then copies, one after the other, the values of both arrays' onto the new one in $\mathcal{O}(n + m)$ time. This is simply not a feasible method when iterating over 60, 000 rows with 784 values each, due to both the time taken and memory required.

However, a pivotal concept is that Python lists can be very easily and rapidly *converted to* NumPy arrays. This is very much the key notion of the back-end development, and also at the heart of working in data science in Python. In fact, you can have NumPy perform specific operations with a function on a Python list without directly converting it. However, this way NumPy would be *forced* to construct a new array and copy its value *every* single time the function is called, giving a time complexity of $\mathcal{O}(k \cdot (n+m))$ where k is the number of times the NumPy function is called. However, it is generally a good practice to directly convert a list to NumPy array only once, after completing the appending-data phase.

Similarly, Pandas' data structures can also be converted into NumPy arrays, and also easily be appended to lists.

```
# Import
import pandas as pd
import numpy as np

# Declare empty list - O(1)
myList = []

# Add values from Panda dataframe into empty Python list - O(n)
for i in range(dataValues.shape):
  myList.append(dataValues[i])

# Convert list to NumPy array - O(n)
myArr = np.array(myList)
```

Listing 7.1: Declaring, filling and converting a Python list to a NumPy array with values from a Panda data frame

If one had to choose the most suitable library for this project, the edge would go to **NumPy** for its multi-dimensional array manipulation and processing, which are truly relevant to this project. Moreover, NumPy works well with Matplotlib, a Python data visualisation and plotting tool, which is why it was chosen to be the central working framework. All the data was eventually converted to variables which were compatible with NumPy, and the Kohonen algorithm was implemented with it.

The functions that NumPy *cannot* do efficiently, were delegated to other libraries. Specifically, Pandas was used to read the inputs from a .csv file, as it's `pd.read_csv('my_file.csv')` was vastly superior to NumPy's `genfromtxt('my_file.csv',delimiter=',')`[1], and Python lists were essentially used to fill up arrays with unknown final size. The rest of the implementation takes place primarily using NumPy's and its following functions.

7.1.2 Principal External Functions

Note that many of these functions can also contain additional parameters not listed here. Depending on the context and need, the source contains further arguments than

[1] Fastest Python library to read a CSV file - Stack Exchange. https://softwarerecs.stackexchange.com/questions/7463/fastest-python-library-to-read-a-csv-file. (Accessed on 05/04/2018).

those mentioned here for some of these functions.

NumPy:

- `np.zeros((i,j))` - Declares a multi-dimensional array of `i` rows and `j` columns.
- `np.array(myList)` - Converts the list `myList` into an NumPy array.
- `np.reshape(m,n)` - Reshapes an array from dimensions `i,j` into `m,n`.
- `np.log(x)` - Returns natural logarithm $\ln x$ of `x`.
- `np.exp(x)` - Returns the value of e^x.
- `np.sum(myArr)` - Returns the sum of the array's `myArr` elements.
- `np.add(x,y)` - Returns the sum of `x` and `y`.
- `np.max(myArr)` - Returns the maximum value of the parameter array `myArr`.
- `np.random.rand(i,j)` - Returns random values in shape of `i` rows and `j` columns.
- `np.savetxt('mySavedFile.csv',myNPArr)` - Saves the np array `myNpArray` into the current directory as `mySavedFile.csv` file.

Pandas:

- `read_csv(fileName.csv)` - Read data from a `fileName.csv` file.

Matplotlib:

- `plt.scatter(xValues,yValues,s,marker,facecolour,edgecolour)` - Plots a scattergraph with values from the NumPy arrays `xValues` and `yValues`. The size, type and colour of the marker can be customised with the remaining parameters.
- `plt.xlabel('x-axis-title')` - Inserts a title to the plot's x axis.
- `plt.ylabel('y-axis-title')` - Inserts a title to the plot's y axis.
- `plt.title('title')` - Inserts a title to the plot.
- `plt.show()` - Displays the plot after the script is executed.

Argparse:

- `argparse.ArgumentParser()` - Creates an argument parser.
- `argparse.ArgumentParser.add_argument()` - Adds an argument to the argument parser.
- `argparse.ArgumentParser.parse_args()` - Parses all the arguments added to the argument parser.

Sys:

- `sys.exit(1)` - Exits the Python script gracefully with error status 1.

Datetime:

- `datetime.datetime.now()` - Returns current date and time.

7.1.3 Variables

- `i` is the current iteration.
- `n_iterations` is the iteration limit, i.e. the total number of iterations the network can undergo.
- `time_constant` is the time constant, used to decay the radius and learning rate.
- `x` is the row coordinate of the nodes grid.
- `y` is the column coordinate of the nodes grid.
- `w_dist` is the (squared) distance between a node and the BMU.
- `w` is the weight of the connection between the node `x,y` in the grid, and the input vector's instance at iteration `i`.
- `inputsValues` is the input vector.
- `inputsValues[i]` is the input vector's instance at iteration `i`.
- `l` is the learning rate, decreasing with time in the interval [0, 1], to ensure the network converges.
- `influence` is the influence the neighbourhood function, monotonically decreasing and representing a node `x,y`'s distance from the BMU, has on the learning at step `i`. It is gradually reduced over time.
- `r` is the radius of the neighbourhood function, which determines the extent of the distance neighbour nodes are examined in the grid. It is gradually reduced over time.
- `n` is the total number of grid rows
- `m` is the total number of grid columns
- `net[x,y,m]` is the nodes grid
- `n_classes` is the total number distinct classes in input
- `labels` is the label vector of every input's instance

7.2 Software Development

7.2.1 Arguments Parser

The implemented algorithm uses several variables, which, if modified, would alter outcome of the Self-Organising Map, affecting both the value of variables and their visualisation. The whole point of this project is to discover and visualise the factors that influence and change the outcomes of this algorithm. Additionally, it is a good ideology of software engineering to develop a program which allows modification of these parameters with ease.

As such, the developed script allows users to specifically customise arguments, such as the learning rate and the number of inputs. A neat trick was to develop the scripts so that these parameters **could be modified from the command-line itself**, as is the case for many data-focused programs, instead of changing the values directly in the source code at various places at every adjustment. For this purpose, Python's argument parser, `argparse` was selected and came in very handy.

For example, to input the learning rate in the command-line directly, the code would be as follows. The arguments parser also allows for default values in the event where the user or developer chose not to modify the customisable parameters

```
# Argument Parser for debugging
parser = argparse.ArgumentParser()
parser.add_argument('-r','--rate', type=float, action='store', default
    =0.3, help='Choose learning rate (range: 0-1)')
args = parser.parse_args()
```

LISTING 7.2: Sample arguments parser declaration

If the user does input an argument for the learning rate, it would then be associated with the corresponding variable. If not, the default value in the parser itself would be used to enter in the variable instead.

```
# If a argument is input at the CLI for the learning rate
if (args.rate):
  init_learning_rate =  args.rate
```

LISTING 7.3: Sample functionality if user entered arguments via parser

Furthermore, a `debug` or `-d` flag was used to print out a detailed sequence of internal events in the CLI for debugging and testing purposes. All the variables mentioned in Section 7.1.3 implemented in the program were printed out with their values over time, as well as a progress percentage to indicate how much the network trained had trained so far.

```
parser.add_argument('-d','--debug', action='store_true', default=False,
    help='Print debug messages to stderr')
```

LISTING 7.4: Sample debug flag as an argument

A user can also view the list of possible parameters by using the help flag with `-h` or `--help` on the CLI.

```
$ python3 iris.py -h
```

LISTING 7.5: The possible arguments can be listed with the -h command

Which outputs the possible modifiable arguments and their flag names:

```
Make a 2D map of a multidimensional input

optional arguments:
  -h, --help             show this help message and exit
  -d, --debug            Print debug messages to stderr
  -r RATE, --rate RATE   Choose learning rate (range: 0-1)
```

LISTING 7.6: List of possible sample arguments

Finally, the parser can be used for input parameters in any order. `-d` and `-r` are interchangeable and don't affect their execution either.

```
$ python3 iris.py -d -r=0.8
```

LISTING 7.7: Sample parser usage

This executes the Python script, and is described in the next sections, which lists the information and variables values. The user input parameters such as the learning rate can indeed be spotted in the output generated via the debug flag.

```
Debug mode ON
Loading input files ...
Loaded inputs: <class 'numpy.ndarray'>
Loaded labels: <class 'numpy.ndarray'>
Data normalised: False
n_classes: 3
n: 150
m: 4
Network dimensions: (2,)
Number of training iterations: 150
Initial learning rate: 0.3
Inputs per class: 50
Net <class 'numpy.ndarray'>
Initial Radius 3.0
Time constant 136.5358839940256
0%
1%
...
99%
100%
Rate: 0.3
x: (150,)
y: (150,)
z: (150, 3)
BMUs: (150, 2)
Saved sorted coordinates
Saved sorted coordinates with noise
```

Listing 7.8: Sample parser usage output

7.2.2 Datasets

For importing and using the original dataset, e.g. the Iris and EMNIST dataset, inside the Python scripts, they could be downloaded in .csv format from their hosting sites. They could then be referenced by into the script by their path, and thus used for training the network.

```
data_path = 'localPath/datasetFile.csv'
data = pd.read_csv(data_path)
```

Listing 7.9: Importing the Iris dataset from a local file using Pandas

This would imply having them in the project directory along with the source code to compile every time. However, sharing this would be very problematic, as the EMNIST dataset has 188,000 lines, and weighs around `218Mb`. Even as a `.zip` file this was not an ideal way.

An elegant solution was found in Panda's documentation which allowed data to be important directly for URLs, starting from version `0.19.2`, and substantially reduces the size of the final source code folder.

```
data_path = 'http://archive.ics.uci.edu/ml/machine-learning-databases/
    iris/iris.data'
```

```
data = pd.read_csv(data_path, encoding='utf-8', header=None)
```

Listing 7.10: Importing the Iris dataset from URL using Pandas

A subsequent challenge in this method was that the EMNIST dataset was not hosted anywhere online in a .csv format. This was circumvented by uploading the data on the University of Liverpool server, and hosting them at a public URL `http://cgi.csc.liv.ac.uk/~u5es2/EMNIST/`. One might think that the data is not secure as the website is `http` not `https`, but it is important to recall that this dataset is freely available in the public domain, and does not contain any sensitive data. Furthermore, the university server files are hosted behind a firewall, which gives it an extra layer of protection.

```
data_path = 'http://cgi.csc.liv.ac.uk/~u5es2/EMNIST/Sorted/Train.csv'
data = pd.read_csv(data_path, encoding='utf-8', header=None)
```

Listing 7.11: Importing the EMNIST dataset from URL using Pandas

The contents of the uploaded .csv files are explained in more detail in Section 7.2.7.

The RGB dataset is generated in the script using random values, and therefore does not require an import statement.

```
# Argument Parser for debugging
parser = argparse.ArgumentParser()
parser.add_argument('-i','--inputs', type=int, action='store', default
    =20, help='Choose number of train inputs per class (range: 0-2400)')
args = parser.parse_args()

# Get value in variable
if (args.inputs):
  inputsQuantity =  args.inputs

# Generate requested quantity of vectors
data = np.random.randint(0, 255, (inputsQuantity, 3))
```

Listing 7.12: Sample RGB dataset creation

7.2.3 Normalisation

Once the dataset has been imported, or generated, it should be normalised so that all inputs features are given the same importance. For example in the Iris dataset, the petals might naturally be longer than the sepals, however the former attributes shouldn't be given more weight than latter ones while training. Normalising neutralises this effect, and additionally, neural networks are much more efficient if the input values are between 0 and 1.

```
# Constant
INPUTS_MAX_VALUE = data.max()

# Normalise and convert from list to array
inputs = []
inputs = data/INPUTS_MAX_VALUE
inputs = np.array(inputs)
```

Listing 7.13: Sample RGB data normalisation

The input's max value used for normalisation will be 255 for the RGB and OCR dataset, as they both read colour values, and are even (0-255) across all dimensions of each input. This also makes it easier to normalise the whole dataset all at once. For the Iris dataset, however, the maximum value used for normalisation will actually be the maximum value in the dataset *for that column*, as the variables are on different scales.

```
# Constant
INPUTS_MAX_VALUE = data.max(axis=0)

# Normalise and convert from list to array
inputs = []
inputs = data/INPUTS_MAX_VALUE[np.newaxis, :]
inputs = np.array(inputs)
```

Listing 7.14: Sample Iris data normalisation

7.2.4 Kohonen Algorithm Implementation

This section goes through the internal functions developed for the Kohonen algorithm that are the same for all three models.

```
for i in range (n_iterations):

  # ---------------- INPUT ----------------
  # 1. Select a input weight vector at each step

  # This can be random, however since we're using sorted inputs, we're
  # proceeding in a linear manner through all nodes for sake of clarity
  t = inputsValues[i, :].reshape(np.array([m, 1]))

  # ---------------- BMU ----------------
  # 2. Find the chosen input vector's BMU at each step
  bmu, bmu_idx, dist = findBMU(t, net, m)

  # ---------------- DECAY ----------------
  # 3. Determine topological neighbourhood for each step
  r = decayRadius(init_radius, i, time_constant)
  l = decayLearningRate(init_learning_rate, i, iterations)

  # ---------------- UPDATE ----------------
  # 4. Repeat for all nodes in the BMU neighbourhood
  for x in range(net.shape[0]):
    for y in range(net.shape[1]):

      # Find weight vector
      w = net[x, y, :].reshape(m, 1)

      # Get the 2-D distance (not Euclidean as no sqrt)
      w_dist = np.sum((np.array([x, y]) - bmu_idx) ** 2)

      # If the distance is within the current neighbourhood radius
      if w_dist <= r**2:

        # Calculate the degree of influence (based on the 2-D distance)
        influence = getInfluence(w_dist, r)

        # Update weight:
        new_w = w + (l * influence * (t - w))
```

```

        # Update net with new weight
        net[x, y, :] = new_w.reshape(1, m)
```

LISTING 7.15: Python implementation of the main Kohonen algorithm

If one was to compare this implementation to the Kohonen algorithm given in Section 3.5, the main noticeable difference would be that this version proceeds through all the nodes sequentially, as opposed to iterating randomly. This means at each step, the 'next' node is literally the adjacent one to be processed. As all nodes have to go through the process anyway, this does not have any impact on the final network, because the final weight values would have eventually been the same, just gone through a different route.

From a software point of view, a glaring omission in code above is that **no values are ever stored**. The variables are constantly overwritten as the network goes through the iterations, but at the end the information of the *evolution* of the network is lost, and only the values of the last iteration remain. The idea of using Python lists for dynamic arrays and subsequently converting them to NumPy ones works perfectly in this case. First they are declared inside the method:

```
bmu_idx_arr = []
radiusList = []
learnRateList = []
sqDistList = []
```

LISTING 7.16: List declarations to contain network variables over the course of its evolution

And values are added to each one during every iteration of the Kohonen algorithm.

```
for i in range (n_iterations):
  # ————————— INPUT —————————
  ...

  # ————————— BMU —————————
  bmu, bmu_idx, dist = findBMU(t, net, m)

  bmu_idx_arr.append(bmu_idx)
  sqDistList.append(dist)

  # ————————— DECAY —————————
  r = decayRadius(init_radius, i, time_constant)
  l = decayLearningRate(init_learning_rate, i, times)

  radiusList.append(r)
  learnRateList.append(l)

  # ————————— UPDATE —————————
  ...
```

LISTING 7.17: Lists appended with calculated values

The variables used in the Kohonen algorithm are initialised according to the network's structure and properties as detailed in Section 3.2 and 3.3 respectively. Choosing the number of nodes in a grid is an art in itself. As such, a good rule-of-thumb is to declare the grid to be double the size of the maximum number of classes in a model. This means for Iris dataset, which contains 3 different total classes, the network size would 6x6. For the model using only digits, the size would 20x20, as there are a total of 10 digits (0-9).

```
# Weight Matrix
net = np.random.random((n_classes*2, n_classes*2, m))

# Initial Radius for the neighbourhood
init_radius = max(network_dimensions[0], network_dimensions[1]) / 2

# Radius decay parameter
time_constant = n_iterations / np.log(init_radius)
```

Listing 7.18: Declarations

The functions are based on the formulas given in section 3.6. Recall that the radius and learning rate have to decrease with time, similar to a exponential function, and the influence like a Gaussian function.

```
# Decay the neighbourhood radius with time
def decayRadius(initial_radius, i, time_constant):
  return initial_radius * np.exp(-i / time_constant)

# Decay the learning rate with time
def decayLearningRate(initial_learning_rate, i, n_iterations):
  return initial_learning_rate * np.exp(-i / n_iterations)

# Calculate the influence
def getInfluence(distance, radius):
  return np.exp(-distance / (2* (radius**2)))
```

Listing 7.19: Functions

And finally, the function to find the BMU, which is called at each iteration in the Kohonen algorithm, can be implemented as below. Each node is evaluated in the grid is evaluated, until the one which is **the most similar** to the current input node - meaning the one with the smallest Euclidean distance - is chosen and returned as the BMU.

```
def findBMU(t, net, m):
  # A 1D array which will contain the X,Y coordinates
  # of the BMU for the given input vector t
  bmu_idx = np.array([0,0])

  # Set the initial minimum difference to large number
  min_diff = np.iinfo(np.int).max

  # To compute the high-dimension distance between
  # the given input vector and each neuron,
  # we calculate the difference between the vectors
  for x in range (net.shape[0]):
    for y in range(net.shape[1]):
      w = net[x,y,:].reshape(m, 1)

      # Don't sqrt to avoid heavy operation
      diff = np.sum((w - t) ** 2)

      if (diff < min_diff):
        min_diff = diff
        bmu_idx = np.array([x, y])

```

```
    bmu = net[bmu_idx[0], bmu_idx[1], :].reshape(m, 1)

    return(bmu, bmu_idx, min_diff)
```

Listing 7.20: Find BMU function

For practical implementation purposes, the smallest distance doesn't actually need to be 'square rooted', as we are only using it to compare with other distances which are anyway squared initially. Calculating the square root would be a time and memory consuming operation, at each iteration, and would needlessly slow down the efficiency of the already lengthy method.

7.2.5 Offset Noise

Once the algorithm is completed, the neural network stops training (and testing), and the data processing is completed. The `BMU` array (or any of its variants, depending on the model) contains the coordinates (X,Y) of clustered the nodes that make up a Self-Organised Map. These values can now be plotted on a scatter-plot on the front-end for the user to see on the web-application.

One issue however arises when the quantity of input data is *larger* than the number of possible nodes in the grid. If a grid is of size 6x6, such as the Iris net, it could only contain a maximum of $6 \cdot 6 = 36$ possible nodes. However there are 150 input instances, meaning even if each was clustered onto a separate node, there would be an overlap, only the most recent node would be shown on the graph when iterating through the coordinates array. This is an important issue as only 36 visible nodes out of a total of 150 represent only 24% of all data. For other models with a higher volume, the data representation would be even lower.

In fact, this issue would arise most times, as the whole idea of unsupervised learning is to cluster input points by using a large quantity of data. The bigger the data, the higher the accuracy. The mixed EMNIST database contains 47 classes, and would therefore have a total of $47 * 2 = 94$ possible nodes, which returns only a $\frac{47*2}{118000} = 0.07966\%$ of data representation.

Keep in mind that we *do* want data to overlap, else there would be **no similarity to cluster them with**. We do not however want to *not be able to **view*** the similarities because the nodes only show one of the many possible data points. We want to show the overlap in our data visualisation, not have it hidden.

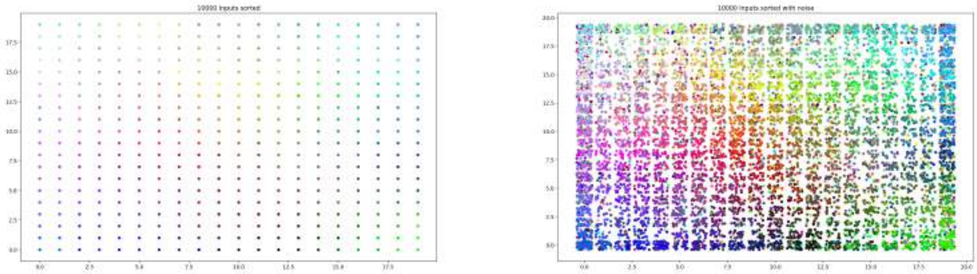
To elegantly and aesthetically counter this problem, a small **offset** was added to each data point in a *random* direction.

```
# Offset min and max values
a_x = -0.4
a_y = -0.4
b_x = 0.4
b_y = 0.4

# Calculate noise
noise_x = (b_x-a_x) * np.random.rand(bmu_idx_arr.shape[0], 1) + a_x
noise_y = (b_y-a_y) * np.random.rand(bmu_idx_arr.shape[0], 1) + a_y

# Add noise to all points in the BMU array
```

```
xPlotNoise = np.add(bmu_idx_arr[:,0], noise_x[:,0])
yPlotNoise = np.add(bmu_idx_arr[:,1], noise_y[:,0])
```

LISTING 7.21: Adding offset to each data point

This way, if a single node contained more than one data point, then they would not be hidden by virtue of being one on top of the other, but in fact 'scattered' *around* the node. The idea is simpler to grasp in a visual form.



(A) A trained SOM without noise (B) A trained SOM with noise

FIGURE 7.1: By adding an offset to each data point, a considerably improved visualisation of the entire dataset is possible.

The difference in quantity of information gathered by glancing at both plots is of *immense* value, and its depth and importance cannot be understated. Visual representation of data is very striking to the human eye, and a good rendition requires very little explanation. Adding noise to each data point was therefore absolutely vital, and perhaps the single most important feature developed in the entire back-end. It single handedly increases the quality and value of every single plot generated and viewed by the user. The quality of the neural network's training can be assessed to a certain degree by a simple glimpse at the scatterplot with noise.

7.2.6 Processing Speed vs. the Number of Classes

After implementing and testing the RGB and Iris models of the network, a **major** problem quickly became apparent for the OCR model. The final Self-Organising Map would only be produced at the end of all iterations. This was not an issue for datasets with low dimensions, low data volume, low classes, or even low-dimensions and high-volume, as the data processing would be at worst relatively slow, i.e. a couple of minutes. However, for datasets with high-volume, high-dimensions and *especially* **a large number of classes** (e.g. 47), the processing time would be very, **very** long.

The EMNIST dataset, however, contained a total of 47 classes, with a high-volume data of 112800 instances, each one being of 784 dimensions. The RGB generated dataset, on the other hand, had a low - arbitrary - amount of classes (anything between 3 and 5), 100 instances of each data point of only 3 dimensions each. The Iris dataset had 3 classes, 150 instances of 4 dimensions each.

Although the dimension and volume attribute for each dataset were known and accounted for, as shown in the table 7.1, an issue was that the current implementation created a nodes grid of `2*n_classes`. This means for the EMNIST dataset, there was a grid of $(2 \cdot 47) * (2 \cdot 47) = 94^2 = 8,836$ nodes in total, and each single one's Euclidean distance over 784 dimensions was calculated. In simpler words, calculating the difference between 2 arrays, of 784 values each, a total of $8,836$ times is what made the training laboriously slow. Even without calculating the square of each difference, the process was slow enough to easily last several *hours* for around **10,000 inputs only**.

```
for x in range (net.shape[0]):
# Net shape is the length (and width) of nodes grid. In EMNIST's case
    the size of the grid is 94x94, which gives a total of 8836 iterations
    .
  for y in range(net.shape[1]:
    w = net[x,y,:].reshape(m, 1)
     diff = np.sum((w - t) ** 2)
```

LISTING 7.22: The section of findBMU() function which took a gigantic amount of time

As learnt by this practical experience, **the size of the network is the most important factor in determining the feasibility of a network's training**. If it was decided that the size should follow a certain unalterable rule of thumb - that the length and width of a network should be double the size of its total number of classes - then the only way possible to make this network's convergence feasible was to *reduce* its total input data. 150 input data instances were sufficient to converge and visualise the Iris dataset, and the RGB model could easily go up to 60,000 instances and produce a stabilised network (by virtue of each instance being very low-dimension and the network having an overall small sized grid). Surely a quantity between several hundred and a few thousand should be able to converge a network, even with 94x94 grid.

Thus, the ideal solution was to change the implementation in a way such that only the first 20 values of each class was taken in as input data, totalling approximately a reasonable thousand values ($47 \cdot 20 = 940$). And after labourious debugging and input data visualisation, therein was discovered the **biggest challenge and set-back of the entire project**: the EMNIST's dataset, totalling 112,800 data instances of 784 dimensions each, were **not sorted according to their class**. They entire database was ordered *randomly*, making it impossible to reduce the total number of inputs for each class when training the network.

The magnitude of this realisation simply cannot be understated. This meant there was no way to work with the principal dataset of the project without waiting hours on end for the network to finish training for a single test, and even then there could be minor programming errors which could 'ruin the batch', so to speak. This took an enormous toll on the productivity and advancement of the realisation of the implementation, and was the single biggest cause of delay against the planned timeline. A string of alternative fixes, ingenious 'hacks', and innovative work-arounds were attempted under intense pressure in order to find a feasible solution for this issue within a manageable time-frame.

An obvious resolution was not to reduce the quantity of input data of each class, but to instead take a slightly bigger chunk of the total dataset, so that there was enough of a margin to encompass every class's input values at least a handful of times, and still have a total number of input instances not going beyond a couple of thousand. This would nonetheless take several minutes to an hour to compute, but could be optimised to find the perfect ratio between inputs of each class and the total computational time.

However, this method proved to be unsuccessful, as a network simply cannot converge with a couple of thousand total inputs, as they represent only around 20 instances of each class, which is very low to distinguish between data of 784 dimensions. Furthermore, the slice of data being taken from the original large dataset was too small to offset the randomness of each class. Some classes were repeated too often, and some almost none. This would lead to a distorted and converged network. Finally, a possibility was simply that the network was *not* convergable for a large number of total classes. After all, Kohonen networks were used to visualise and find pattern in data that overlapped in a few instances. In the case of EMNIST, the full dataset was too large with 47 different classes, along with being too long to train and converge. However, it seems counter-intuitive to think that there were perhaps not enough similarities between a large number of classes, as logically they should have more overlap than datasets with fewer classes.

An alternative way to 'hack' this problem, was to use several machines to process different networks, each run with different parameters values, and use each result to see and understand which hypothesis held truth to determine the principal factor that caused this non-convergence.

Again, this proved to be an impossible tasks for several logistical reasons. First of all, the number of available machines was very low. Secondly, all of them needed to have the version of Python and its various libraries such as NumPy and Pandas installed. If a machine was non-unix based, then another set-back would take place due to the additional work load of configuring a Windows machine. Finally, any update to the overall script development would have to be made on the other machine as well. The management and synchronisation of the scripts would be an absurdly strenuous task to conduct. It was simply not a feasible solution, both technically and logistically to break down an issue over several machines in order to try and understand the cause of a neural network's convergence and potentially use the results to overcome the issue. It was mentally taxing enough to work on such a problem on a single machine, with constant minor updates to the developing scripts.

The only way to overcome this problem was then to **sort the data**. If all 112,800 input could be sorted into 47 different arrays, with each one containing only the instances belonging to that distinct class, then we could a chose a specific amount of inputs all sorted arrays. Moreover, we could see if the non-convergence of a network was really due to a high grid size, and if so find the limit, by first only training a subset of the EMNIST dataset which only contained digits, and therefore only 10 total classes. Then the same could be tested on only the alphabets in the EMNIST dataset, which would have 26 classes, before finally attempting the colossal 47 classes. Sorting the data, as often restated in Computer Science education, was the key not only to implementing the OCR model of Kohonen's neural network but also the find insights of the properties and nature of this algorithm.

7.2.7 Data Sorting

The first step to sorting the data was knowing that there were indeed an equal amount of inputs for each class, specifically $112800/40 = 2400$ instances. Then, there were two ways of proceeding: manually declaring 47 arrays, and using an insertion sort algorithm to iterate over all 47 classes, and appending to the relevant array the instance that belonged to that class. This can be determined using the array labels, which thankfully contains the label of each input instance's class. It did not feel like 'smart' programming at all to declare such a large amount of arrays. Furthermore, insertion sort is a basic sorting algorithm and would take at best $\Theta(n)$ and at worst $\mathcal{O}(n^2)$ iterations to complete.

A series of alternative ways were again tested, such as using Python dictionaries, 47 of which can be easily declared by a single for-loop. However, in each alternative method, the core issue that would arise was that it was simply not possible to declare variable names with other variables. One just cannot use a for-loop to name arrays with strings.

Instead, the manual way of declaring arrays and using the unsorted data's labels to sort them into their respective classes's array was implemented with success.

```
# Read unsorted raw data
data_path = 'path/To/UnSorted/Data.csv'
data = pd.read_csv(data_path)

# Create lists per class
arr_0 = []
arr_1 = []
...
arr_46 = []

# Sort and append according to class
for i in range(data.shape[0]):
  if data.iloc[i,0]==0:
    arr_0.append(data.iloc[i,1:])
  elif data.iloc[i,0]==1:
    arr_1.append(data.iloc[i,1:])
  ...
  elif data.iloc[i,0]==47:
    arr_47.append(data.iloc[i,1:])

# Merge in order into main list
sortedInputs.extend(arr_0+arr_1+...+arr_47)


# Make sorted labels list
i = 0
for x in range(0, data.shape[0], max_inputs_per_class):
  for y in range (max_inputs_per_class):
    sortedLabels.append(i)
  i=i+1

# Convert both lists to NumPy arrays
sortedInputs = np.array(sortedInputs)
sortedLabels = np.array(sortedLabels)

# Export sorted classes
np.savetxt(save_path+'SortedInputs.csv', sortedInputs, fmt='%d',
    delimiter=',')
np.savetxt(save_path+'SortedLabels.txt', sortedLabels, fmt='%d')
```

LISTING 7.23: Compact view of the sorting script implementation

The sorting scipt was also developed with Python's `argparse`, so that a user could input the paths to his unsorted data (and labels) via the command line, using the `-c`, `-ip`, and `-sp` commands.

```
# Argument Parser
parser = argparse.ArgumentParser(description='Sort the EMNIST data in order of their class')
parser.add_argument('-d','--debug', action='store_true', default=False, help='Print debug messages')
parser.add_argument('-c','--classes', action='store', type=int, help='Insert the number of different classes in the database to be sorted')
parser.add_argument('-ip','--input_path', action='store', help='Insert the data path to the .csv file')
parser.add_argument('-sp','--save_path', action='store', help='Insert the save path for the sorted output .csv file (do not insert the file name itself)')
args = parser.parse_args()
```

LISTING 7.24: Compact view of the sorting script implementation

It is this script's sorted values that were uploaded on the University of Liverpool's departmental server at `http://cgi.csc.liv.ac.uk/~u5es2/EMNIST/`, and finally used for the OCR model's input data and labels.

7.2.8 Local Visualisation with Matplotlib

Matplotlib is a Python library for plotting and data visualisation, and was an essential tool for developing these scripts as it allowed observation of the algorithm's results locally at the back-end itself. Being integrated with NumPy, it allowed for very easy implementation: the data to be plotted could stay in separate NumPy arrays for the x and y coordinates, and the plotting method would automatically iterate and get the necessary values from the same row of the separate arrays.

Being in the back-end also had other advantages, such as visualising any variable for debugging purposes.

```
# Plot nodes
plt.scatter(x_coords, y_coords, s=20, facecolor=zPlot)
plt.title(str(n)+' Inputs unsorted without noise')
plt.show()
```

LISTING 7.25: Plotting BMUs

```
# Plot learning rate
plt.title('Learning rate evolution')
plt.xlabel('Number of iterations')
plt.ylabel('Learning rate')
plt.plot(learnRate, 'r')
plt.show()
```

LISTING 7.26: Plotting learning rate against time to visualise its evolution

Chapter 8

Linking Front to Back End

Finally, this chapter summarises how the front and back end were linked, specifically the data structures and how the data flowed from one point to another depending on user inputs and back-end outputs.

8.1 Incompatibility

Till now, all the diverse challenges encountered of various difficulties were eventually solved, or accounted for, one way or the other. Some were were purely cosmetic, such as styling each HTML web-page using CSS and JavaScript, requiring only diligent testing and updating. Others were more technically challenging but nonetheless engaging, necessitating theoretical Computer Science skills, such as algorithm complexity analysis, as well as a certain degree of creativity to solve in an elegant manner. Some were substantially more challenging to simply identify, and then gruelling to solve, such as data sorting, requiring a certain abstraction, back-tracking, re-developing parts of the software, and general meticulousness. None of these problems were fundamentally unsolvable, as the main deciding factor was simply the time, energy, and strategy required to overcome them.

There was, however, one underlying technical problem which could not be solved. The issue stems from the general incompatibility between Python and JavaScript. These two programming languages were fundamental to this project, without which this project would not have been the same. However, they do not communicate well at all, as they were not originally ever meant to interact. JavaScript was natively built to be part of the three core technologies of the World Wide Web, along with HTML and CSS, and is also proficient at working with a PHP back-end. Python, a high-level general purpose programming language is good at a lot of things, including web-development with frameworks such as Django and Flask, but is not **directly** compatible with JavaScript. Flask can host JavaScript files, but to send data from a Python script to a JavaScript one is nonetheless complex. There have been many attempts to create a simpler way of linking the two, but most of them have eventually resulted in awkward and unsuitable implementations for important projects.

When designing this project, neither language could be omitted, as JavaScript is indispensable for web-scripting, and the alternatives to Python for designing a mathematical back-end would have been very limited without data specific libraries such as NumPy and Pandas. Undertaking a data science project without employing Python would have sorely restricted the scope, modernity and **originality** of the project.

Consequently, the ambitiousness of this project resulted some incompatibility, one of which was particularly troublesome as it related to one of the core features this project promised: direct interactivity between the user and Kohonen model. Indeed, although components were build with JavaScript to take in a user's hand-drawn inputs on the front-end, they could not be sent to the back-end model in a straightforward and elegant way. Similarly, the back-end could not directly transfer back the neural network's outputs to JavaScript, although this particular direction of flow was slightly mitigated by finding a round-about way, further explained in the next chapter.

This is why, the user input data on the canvas does not return any data, despite significant time and work going into converting the drawn strokes to data values of the correct shape and size.

Despite being a very interactive feature, the input would have only been a single input instance, where as the EMNIST dataset provided over hundred thousands such values. It is important to remember that the implemented network is fully capable of handling input data, at any scale, but simply could not *receive* the data from the user. This problem was on a structural and systems level, due to the complex incompatibility between the polished front-end and highly developed back-end, and not due to one single error. If one were to manually transfer the user's letter data to a .csv file into the Python script, the network could successfully cluster that input.

8.2 Data structures

This section quickly highlights how the front-end was able to read the Python output values, despite the linking not working in the other direction.

By writing the calculated Python values to a local .csv file in the correct relative repository, these could be read by the JavaScript every time a new page was loaded.

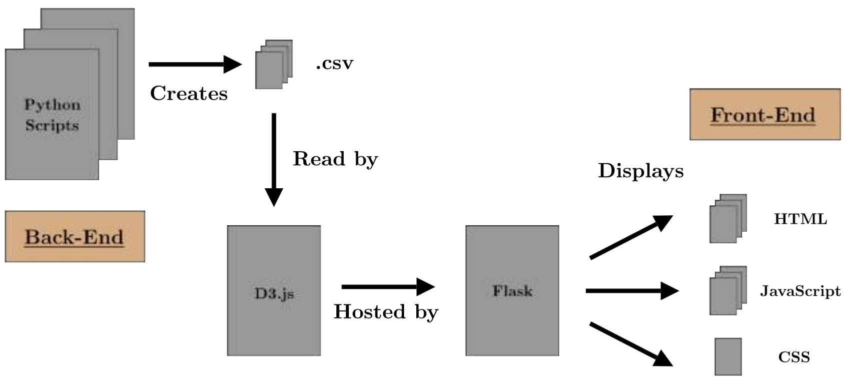


Figure 8.1: Flow of data between front and back ends

8.3 Data Visualisation

The first and most important goal was to use the output data calculated by the Kohonen back-end model, by transferring it to the front end, and representing it in a visual, comprehensive and easy-to-grasp way.

`D3.js` was chosen as front-end plotting library as it was very effective at data visualisation. Similar to Bootstrap, D3.js is a continuously updating library, with new versions being released every few months. D3's `v4` release was used when researching the library and understanding it's API, however `v5` was the final version used for the implementation.

At this point, all the sorting, processing, and number crunching was completed. All that was left was to plot the `(X,Y)` coordinates list of the BMUs onto a 2D graph, as previously done locally on Python's Matplotlib. However, this proved to be an unexpectedly and considerably challenging task, and became a critical cause for delay in adding more textual explanations and informations on the website.

The difficulty was mostly due to the *nature* of the JavaScript library itself. Despite its popularity, D3.js is not recommended for beginners on account of its very steep learning curve. Furthermore, it's Github-based API documentation was hard to understand, navigate, and lacking examples for such a dense reference. The constant updates also didn't help, as many of the examples given for D3 on other websites referred to older versions and were thus useless at the time.

An easy option was to simply avoid D3 altogether and circumvent the problem entirely by using a different plotting library. Google Charts, Plotly.js, Chartist.js and especially Chart.js were all considered as alternatives, but all permutations led to one technical issue or the other. Notably, one sticking point for most other libraries was that the points were to be read from a local file in **.csv** format, as opposed to a JSON format. Additionally, those which did offered little customisation tools in particular for scatter-plots, which, on top of being plotted, needed to be coloured according to its class value and ideally even display mouse-over text. Therefore, despite the tough learning curve, an exceptional effort was made to understand the technicalities and power through the material in a tight period of time in order to be completed by the demonstration deadline. Ultimately, this challenging endeavour was successful, and the details hereunder give an insight to the technicalities of D3.js that were overcome.

First of all, unlike most JavaScripts declared at the end of the `<body>` tag, D3 had to be important in the header along with the Bootstrap and personal CSS reference, because it is directly called as soon as the page is loaded.

```
<head>
  <!-- D3.js -->
  <script src="https://d3js.org/d3.v5.min.js"></script>
</head>
```

LISTING 8.1: Importing D3.js in the HTML header

Then the code has constructed with the following declared elements: margins, axis, SVGs, and finally plotting the graphs by reading the .csv data files.

```
// Margins
var margin = {top: 20, right: 10, bottom: 20, left: 15},
  width = 600 - margin.left - margin.right,
  height = 300 - margin.top - margin.bottom;

// Axis
var x = d3.scaleLinear()
  .range([0, width]);

var y = d3.scaleLinear()
```

```
  .range([height, 0]);

var xAxis = d3.axisBottom()
  .scale(x);

var yAxis = d3.axisLeft()
  .scale(y);
```

Listing 8.2: Margins and Axis

The number of SVGs (plots) and their respective data was naturally dependent on the number of graphs chosen to be displayed.

```
// Adding to HTML
var svg = d3.select("#chartContainer").append("svg")
  .attr("width", width + margin.left + margin.right)
  .attr("height", height + margin.top + margin.bottom)
  .append("g")
  .attr("transform", "translate(" + margin.left + "," + margin.top + ")"
    );
```

Listing 8.3: Single sample of SVG-HTML link

Firstly, to read the .csv's data values, each line had to be read in, and changed from a string to an `int` integer.

```
// Read as ints not strings
data.forEach(function(d) {
  d.xRGB = +d.xRGB;
  d.yRGB = +d.yRGB;
  d.R = +d.R
  d.G = +d.G;
  d.B = +d.B;
});
```

Listing 8.4: Converting each .csv's column from string to int

Then, the domain of both the x and y axis can be adjusted according to the given data values. Once set, they can be drawn and appended to the SVG html class. The graph's ticks (labels) can be removed if necessary, as in our case, as don't represent any values, and are only required to show how the data groups itself into 'physically' separate clusters.

```
// Define domains of x and y axis
x.domain(d3.extent(data, function(d) { return d.xRGB; })).nice();
y.domain(d3.extent(data, function(d) { return d.yRGB; })).nice();

// Draw
// X-axis
svg.append("g")
  .attr("class", "x axis")
  .attr("transform", "translate(0," + height + ")")
  .call(xAxis)
  .selectAll("text").remove();

// Y-axis
svg1.append("g")
  .attr("class", "y axis")
  .call(yAxis)
```

```
  .selectAll("text").remove();
```

Listing 8.5: X and Y axis

Finally, we can plot each data point using the (X,Y) coordinates in the data as a circle with a chosen radius. Additionally, we can colour each one according to its class.

```
svg.selectAll(".dot")
  .data(data)
  .enter().append("circle")
  .attr("class", "dot")
  .attr("r", 3.5)
  .attr("cx", function(d) { return x(d.xRGB); })
  .attr("cy", function(d) { return y(d.yRGB); })
  .style("fill", function(d) {return d3.rgb(d.R,d.G,d.B); })
});
```

Listing 8.6: Plotting the scatterplot circles for RGB dataset

Additionally, the a tooltip can be used for mouseovers.

```
var tooltip = d3.select("#chartContainer").append("div")
  .attr("class", "tooltip")
  .style("opacity", 0);
```

Listing 8.7: Mouse hover tooltip appended to html div

The clever part here was the use of the HTML tag <spanstyle='color:"#";'> containing the individually read R,G,B values. An unrelated complication was the offset value by exaggerated by the Bootstrap columns grid structure. Similar to the offset issue for the canvas, intense debugging was necessary simply to find the cause of the problem. Once understood, a partial solution was successfully implemented. The extra offset caused by the offset Bootstrap column had to be deducted from the page's `eventY` value using jQuery: `d3.event.pageY-$(this).parent().offset().top`.

```
// Mouseover Event Handler
var tipMouseover = function(d) {

  var html = "<span style='color:" + d3.rgb(d.R,d.G,d.B) + ";'>" + d.
    label;

  tooltip1.html(html)
    .style("left", (d3.event.pageX) + "px")
    .style("top", (d3.event.pageY - $(this).parent().offset().top) + "px
  ")
    .transition()
    .duration(200) // ms
    .style("opacity", .9)
};
```

Listing 8.8: Mouse hover tooltip's text content coloured according to class

The mouseover function is ended when the cursor leaves the data circle, and gently faded out.

```
// Mouseout event handler
var tipMouseout = function(d) {
```

```
  tooltip.transition()
    .duration(300) // ms
    .style("opacity", 0);
};
```

Listing 8.9: Mouse out

Each of the 3 dataset's plots were written in individual JavaScript files using the D3 library. They're named `plot.js`, `plotIris.js`, and `plotRGB.js`.

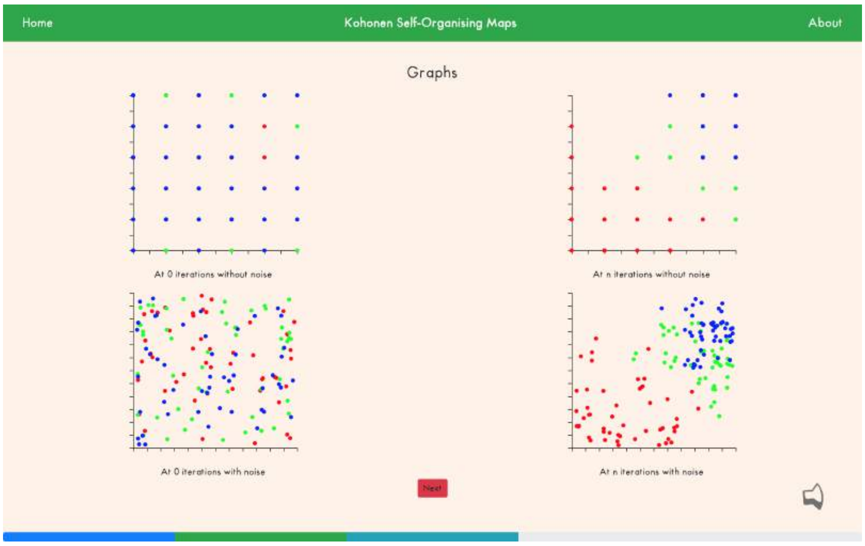


Figure 8.2: A page with four different D3 charts

8.4 Server deployment

As it turned out, Flask *cannot* be deployed on a server, at least not without being thoroughly knowledgeable on third-party Web Server Gateway Interfaces, such as Heroku or OpenShift, which were beyond the scope and intend of this project. Flask is in fact mostly used for local development and testing purposes only, and therefore this project was chosen to be developed for local-hosting purposes only as well. This was indeed an unfortunate development with regards to sharing the web-application with other users, as was originally intended.

Chapter 9

Testing

9.1 Test Results

The following is the testing results of the different scripts. Each test ID was executed with the command `$Python3ScriptName.py` following by any extra CLI parameter, such as `-d`. The parameters for each test case is given in the table, and a blank value represents no additional argument being parsed.

9.1.1 RGB

ID	Data	Data Type	Expected Result	Success?
1	(Blank)	Correct	Successful build	YES
2	`-i`	Erroneous	Native error message	YES
3	`-i=`	Erroneous	Native error message	YES
4	`-i=0`	Erroneous	Implemented error message	YES
5	`-i=-1`	Erroneous	Implemented error message	YES
6	`-i=0.5`	Erroneous	Native error message	YES
7	`-i=-0.5`	Erroneous	Native error message	YES
8	`-i=100`	Correct	Successful build	YES
9	`-r`	Erroneous	Native error message	YES
10	`-r=`	Erroneous	Native error message	YES
11	`-r=0`	Erroneous	Implemented error message	YES
12	`-r=-1`	Erroneous	Implemented error message	YES
13	`-r=0.5`	Correct	Successful build	YES
14	`-r=1`	Correct	Successful build	YES
15	`-r=1.5`	Erroneous	Implemented error message	YES
16	`-d`	Correct	Successful build	YES
17	`-d-i=100`	Correct	Successful build	YES
18	`-d-r=0.3`	Correct	Successful build	YES
19	`-r=0.3-i=100`	Correct	Successful build	YES
20	`-d-r=0.3-i=100`	Correct	Successful build	YES

Table 9.1: RGB script tests

9.1.2 Iris

ID	Data	Type	Expected Result	Success?
1	(Blank)	Correct	Successful build	YES
2	`-r`	Erroneous	Native error message	YES
3	`-r=`	Erroneous	Native error message	YES
4	`-r=0`	Erroneous	Implemented error message	YES
5	`-r=-1`	Erroneous	Implemented error message	YES
6	`-r=0.5`	Correct	Successful build	YES
7	`-r=1`	Correct	Successful build	YES
8	`-r=1.5`	Erroneous	Implemented error message	YES
9	`-d`	Correct	Successful build	YES
10	`-d-r=0.3`	Correct	Successful build	YES

Table 9.2: Iris script tests

9.1.3 OCR

ID	Data	Type	Expected Result	Success?
1	(Blank)	Correct	Successful build	YES
2	`-d`	Correct	Successful build	YES
3	`-r`	Erroneous	Native error message	YES
4	`-r=`	Erroneous	Native error message	YES
5	`-r=0`	Erroneous	Implemented error message	YES
6	`-r=-1`	Erroneous	Implemented error message	YES
7	`-r=0.5`	Correct	Successful build	YES
8	`-r=1`	Correct	Successful build	YES
9	`-r=1.5`	Erroneous	Implemented error message	YES
10	`-iTr=100`	Correct	Successful build	YES
11	`-iTr=0`	Correct	Successful build	YES
12	`-iTr=-1`	Erroneous	Implemented error message	YES
13	`-iTr=2400`	Correct	Successful build	YES
14	`-iTr=2401`	Erroneous	Implemented error message	YES
15	`-iTe=100`	Correct	Successful build	YES
16	`-iTe=0`	Correct	Successful build	YES
17	`-iTe=-1`	Erroneous	Implemented error message	YES
18	`-iTe=2400`	Correct	Successful build	YES
19	`-iTe=2401`	Erroneous	Implemented error message	YES
20	`-t=d`	Correct	Successful build	YES
21	`-t=l`	Correct	Successful build	YES
22	`-t=c`	Correct	Successful build	YES
23	`-t=z`	Erroneous	Implemented error message	YES
24	`-d-iTr=100`	Correct	Successful build	YES
25	`-d-iTe=100`	Correct	Successful build	YES
26	`-d-r=0.3`	Correct	Successful build	YES
27	`-d-r=0.3-iTr=100`	Correct	Successful build	YES
28	`-d-r=0.3-iTr=100-iTe=100`	Correct	Successful build	YES
29	`-d-r=0.3-iTr=100-iTe=100-t=d`	Correct	Successful build	YES

Table 9.3: OCR script tests

As shown above, the scripts show error handling, and graceful exit for all the cases when a user enters an incorrect or invalid parameter.

The full outputs of each cases can be seen in Appendix G, along with the details of all hardware and software used for testing.

Chapter 10

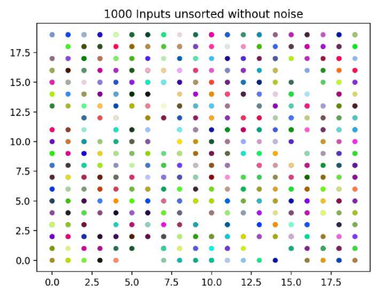
Results

This chapter presents an overview of plots generated by the Python scripts. They can be seen in full detail in Appendix H.

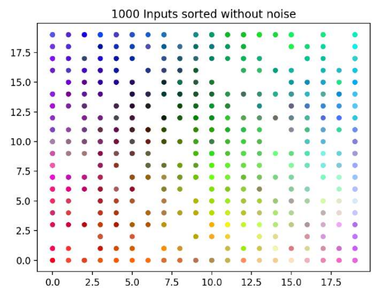
10.1 RGB

The following parameters were used to generate the sample plots shown below:

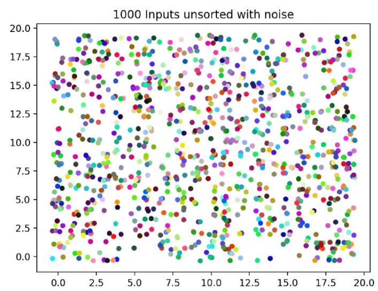
```
$python3RGB.py-d-i=1000
```



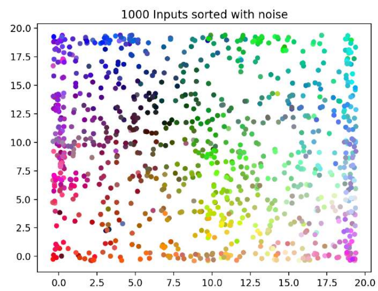
(A) Data before clustering



(B) Data after clustering



(C) Data before clustering with noise



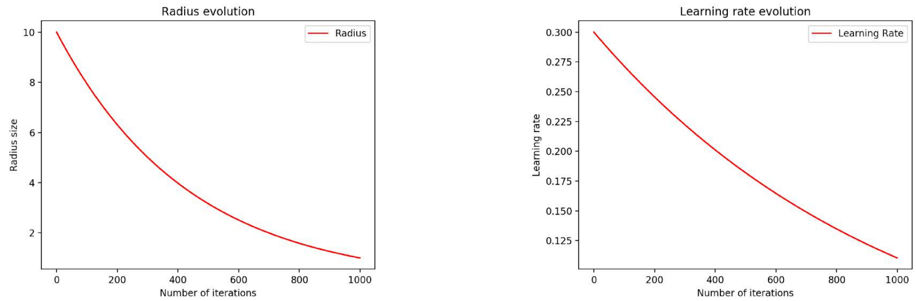
(D) Data after clustering with noise

FIGURE 10.1: RGB model plotted with 1000 inputs

10.2 Iris

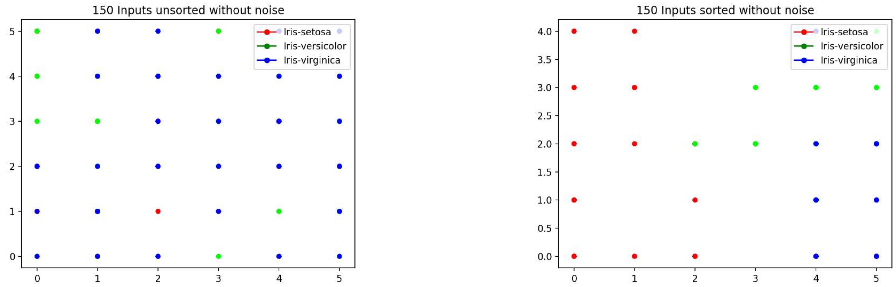
The following parameters were used to generate the sample plots shown below:

```
$python3iris.py-d-r=0.3
```

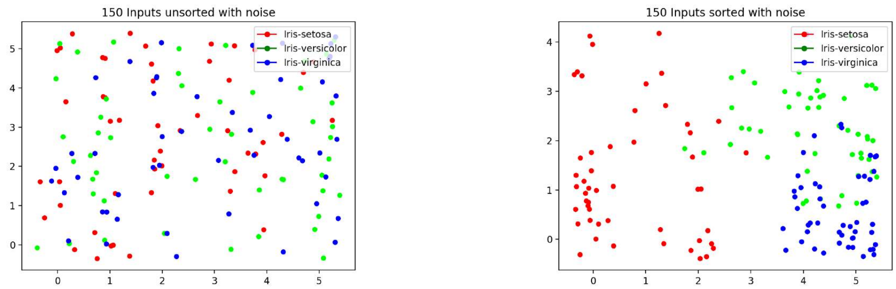


(A) Radius evolution over time (B) Learning rate evolution over time

FIGURE 10.2: Model's radius and learning rate evolution over time

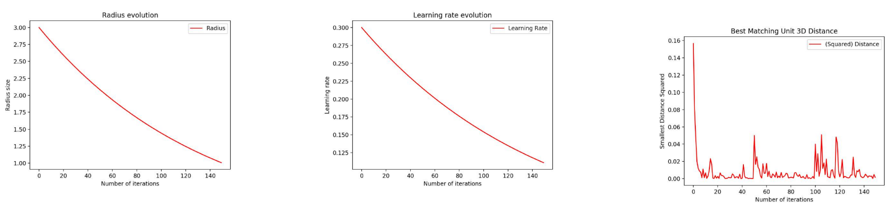


(A) Data before clustering (B) Data after clustering



(C) Data before clustering with noise (D) Data after clustering with noise

FIGURE 10.3: Iris dataset plotted with 0.3 learning rate



(A) Radius evolution over time (B) Learning rate evolution over time (C) BMU distance over time

FIGURE 10.4: Model's radius, learning rate and squared distance evolution over time

10.3 OCR

10.3.1 Digits

The following parameters were used to generate the sample plots shown below:

```
$python3som.py-r=0.3-iTr=100-iTe=10t=d
```

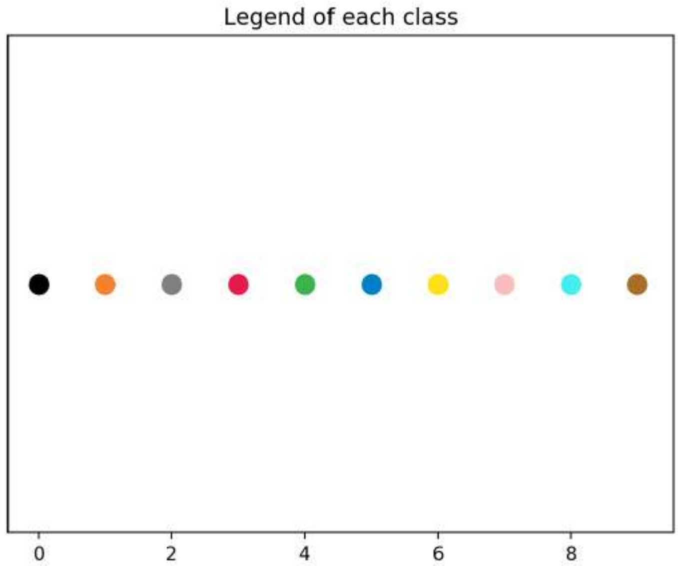
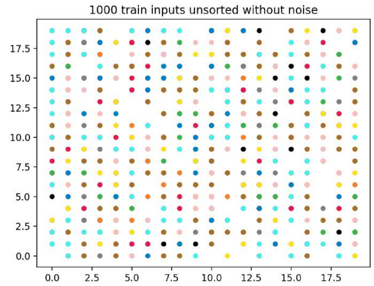
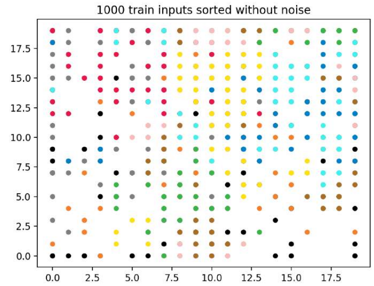


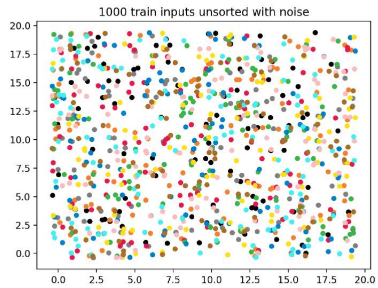
FIGURE 10.5: The legend of each letter used for the graphs below



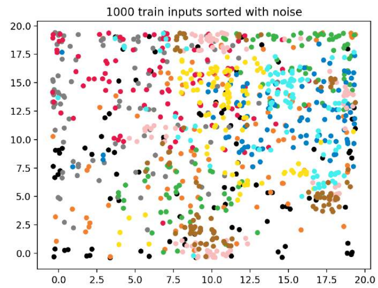
(A) Train data before clustering



(B) Train data after clustering

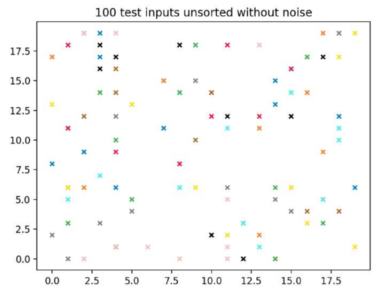


(C) Train data before clustering with noise

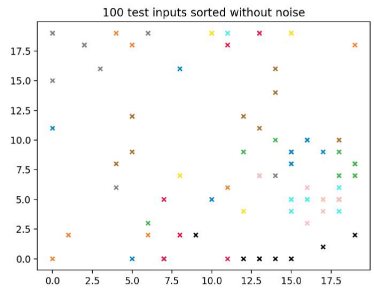


(D) Train data after clustering with noise

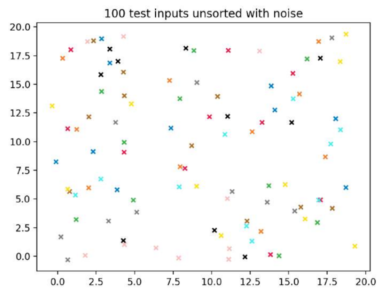
FIGURE 10.6: Digits dataset plotted with 100 training and 10 testing inputs with 0.3 learning rate (Part 1)



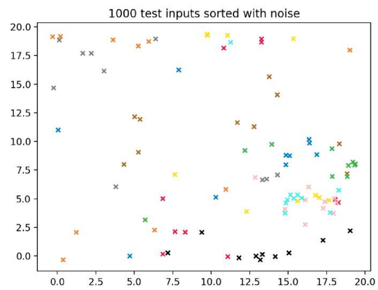
(A) Test data before clustering



(B) Test data after clustering

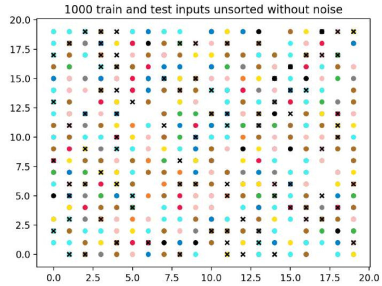


(C) Test data before clustering with noise

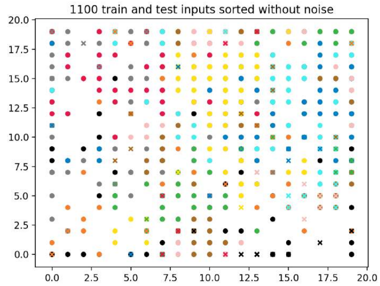


(D) Test data after clustering with noise

FIGURE 10.7: Digits dataset plotted with 100 training and 10 testing inputs with 0.3 learning rate (Part 2)

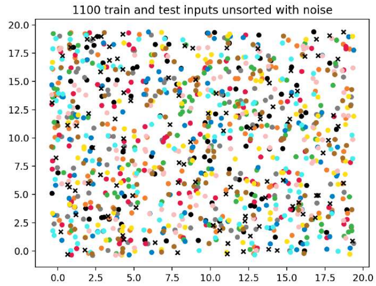


(A) Train and test data before clustering

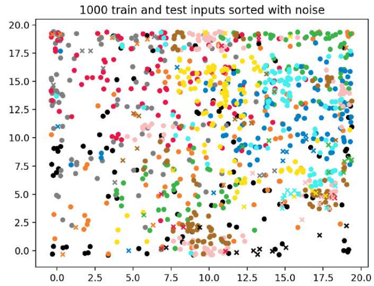


(B) Train and test data after clustering

//

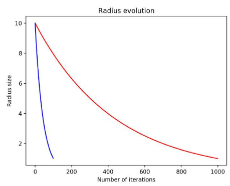


(C) Train and test data before clustering with noise

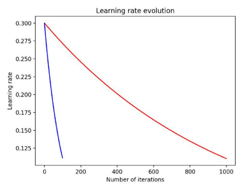


(D) Train and test data after clustering with noise

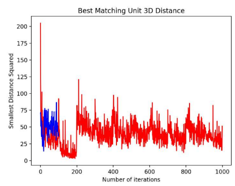
FIGURE 10.8: Digits dataset plotted with 100 training and 10 testing inputs with 0.3 learning rate (Part 3)



(A)



(B)



(C)

FIGURE 10.9: Model's radius, learning rate and squared distance evolution over time

FIGURE 10.10: An alternate plot of the entire 60,000 MNIST letters dataset

10.3.2 Letters

The following parameters were used to generate the sample plots shown below:

```
$python3som.py-r=0.3-iTr=88000t=l
```

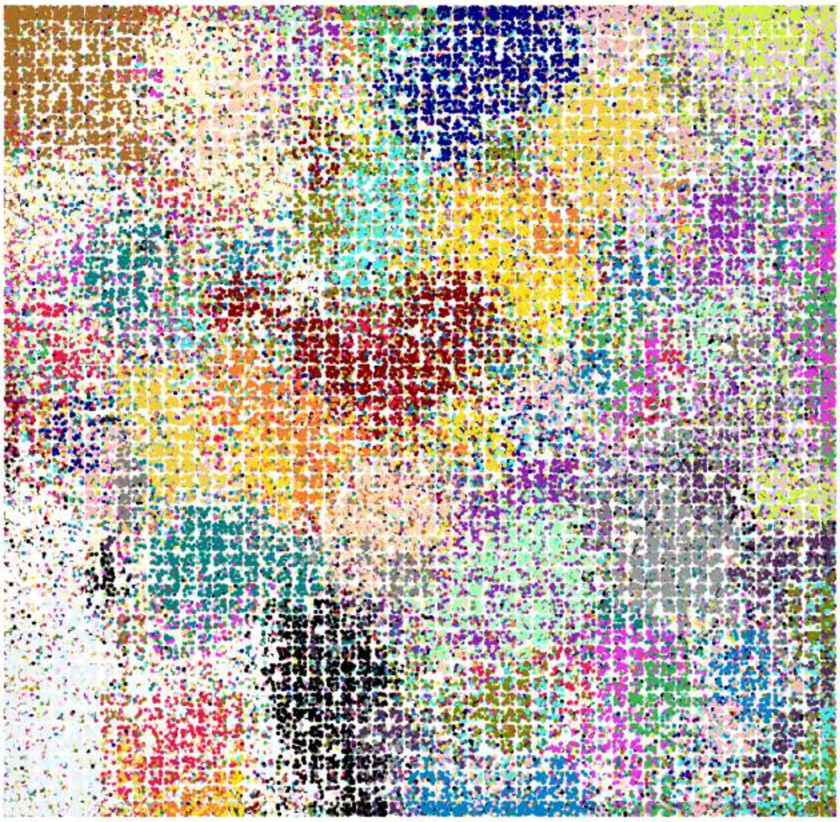


FIGURE 10.11: 88000 letters data only after clustering

Chapter 11

Evaluation

To critically assess a project as a whole, one must compare it to the tasks it set out to accomplish at the very beginning. This chapter attempts to give insights on the project's overall success by measuring its results against its original goals, couples with personal opinion and 3rd party feedback.

11.1 Evaluation Design

11.1.1 Evaluation Criteria

Firstly, it should be ensured that the overall features are working correctly individually and collectively in a asynchronous system.

Secondly, basic user interface and experience guidelines should always work, i.e. clicking on a button should lead to the correct page corresponding to it, most unexpected exceptions should be caught, and the website should be able to display 'error pages' in case of unforeseen crashes rather than native browser alerts.

Additionally, the loading times such as when launching the website, submitting the input, and visualising the training dataset should all be to a reasonable standard for 2018 and in the same league as other similar applications.

Finally, the website should **not be invasive** in any manner, and only the essential permissions should ever be requested. The privacy of the user should be respected no matter what, and absolutely no tracking or data collecting should be done on the visitors.

11.1.2 Assessment Criteria

Assessing many of the criteria on efficiency could simply be a straightforward case of measuring response-time(s) of pages, images, and graphics on various devices and browsers with different memories.

For the requested permissions, security certificates and `RAM` usage, the the browser's developer console and the device's task manager could be checked for detailed information, as another way of assessing the system.

11.2 Critical Evaluation

Each planned feature, given in italics, is evaluated against the final implementation.

11.2.1 Essential Features

- *The website should have an interactive 'Draw' page where users can draw their letter on a Graphical User Interface(GUI) canvas and have the website process and display which letter it is, by interacting with the ANN model.*
- This was implemented on both ends. The front-end could take user input data and convert it to a re-sized 28x28 pixel data value in an 784-dimensional array. The back-end could take in input instances of the same size and output a Best Matching Unit node's (X,Y) coordinates. However, although they worked individually, the data could not be sucessfully passed through to one another. Unfortunate, as each component's implementation was fully developed, an infact even customised to be able to take in several characters as inputs on a single canvas.
- *The website should display the neural network's topological map of input data to the user based on training (and/or testing) data.*
- Fully developed and implemented with Python and D3.js scripts. Took substantial time for both, but topological map can be viewed both on the front-end HTML or on Python's local Matplotlib.
- *The website should highlight where your input would be placed on the displayed topological map.*
- This was also implemented for the user's canvas data, but cannot be shown as it wasn't functional. The testing and training data however is correctly distinguished by crosses and dots.
- *The website should have a 'Learn' page which displays animations or clickable diagrams of neural networks and SOMs, to show how weights are adjusted and converged, and how the network is trained over time.*
- The 'learn' and 'draw' page were converged into a single, linear and more driven experience, in order to control more accurately the way and order a user learns. At each stage, new information and concepts were introduced to the user.
- *The website should have a 'Database' page which contains information on the dataset used to train and test the neural network, such as the number of images used, the size of the entire database, links to the source files.* This was fully implemented in the final web-application. All sources are given, and samples of the EMNIST database is also shown.

11.2.2 Desired Features

- *The users should have an 'in-depth' option of seeing the steps the network goes through, such as re-centring, cropping and down-sampling of the input, probabilities numbers or graphs of which letter the input corresponds to.*
- Partially completed, as the canvas processes invidual characters, crops and resizes them. However, this is all done under-the-surface, and is not shown to the user. Probabilities were not implemented as they are more relevant to a supervised learning model.
- *Allow users to input more than one single input i.e. enter a whole 'training set'.*
- This was implemented on the front-end canvas, which allowed more than a single character to be drawn on its canvas.

- *The 'database' page which shows a sample training data letter for each alphabet from A to Z, and after clicking on one of the letters, the entire training dataset images of different handwriting for that alphabet should be shown. This is to give a visual representation and sense of scale of how many different handwritten letters were used to train the neural network for each alphabet.*
- This was partially implemented. All *distinct* inputs from the database are shown on the database page, however having *all* inputs of the same class was not feasible. Hosting over $60,000$ on a single HTML page was too intensive, and would have needlessly bogged down the system. A smarted and leaner version was implemented instead.
- *Some of the instructions sentences on the website could be written using the synthetic training data images.* Discarded as infeasible and unnecessary.

11.3 Personal Evaluation

11.3.1 Strengths

This project's strengths are in its ambition, thoroughness and meticulousness of the front and back end, both of which are built upon an underlying theoretical foundation of Machine Learning and Kohonen's networks.

A deep understanding of Kohonen's algorithm is required to not only implement a mathematical model, but to then question the factors that influence the convergence of such a network. This project would simply not have been possible without the comprehensive literature review on Self-Organising Maps and Kohonen's algorithm.

Similar rigour was employed when reviewing tools and technologies usable for the development of the implemented components. The source code reflects the depth of the research done for each part, and how it was persistently optimised.

11.3.2 Weaknesses

The weakness of this project is clearly in its incapability to take full advantage of the developed functionalities. Even thought the implementation works, it is not nearly as strong or powerful as it should be. Both ends could be much more interactive, and potentially even usable for current modern-era applications.

Another weakness was in the inability to take a step back from the technicalities and reconnect with non-scientific users. Unexpected delays on two keys areas led to a tight schedule, and eventually the language used for *communicating* the depth of the designed product was not at the same level as its technical code.

11.4 3rd Party Evaluation

For the purposes of system evaluation, 3rd party human participants were involved to gather feedback. It is important to note that all data was completely anonymous and no individual tracking whatsoever was done.

Participants were be first asked if the system worked according to the given specification requirements and it if meets acceptable quality standards. More specifically, they were be asked to rate various factors of the system, such as speed, different functionalities, reliability of outputs, general robustness, and innovation along with the UI/UX 'feel' and 'ease-of-use' of the website. These were all mostly given positive scores, as the evaluation was all done on localhost which virtually has no delays. Furthermore, the artwork and Bootstrap were consistently singled out for praise, as they added a unique personal touch to the project.

Additional questions were on the methodology of the website as a teaching tool, and how ambitious they felt the scope of the project was. They were quizzed on how effectively the creator managed to convey concepts to users in innovative manners, and whether the website provided them with enough content and interest on the discussed topics. This was given a more mixed reception. Users could understand the visual 'before and after' plots, and the concepts behind them. They did not all however understand the nuances of each different dataset and the concept being conveyed due to a lack of textual explanations. More explanations in layman's terms were requested to gain a deeper understanding of the project.

11.5 Further Improvements and Development Ideas

The project can be further enhanced in many ways. First of all, each page's weight can be further reduced by:

- Compressing images
- Compressing resources with GZIP
- Minifying all resources (HTML, CSS, JS)

Furthermore, each page's number of browser requests could be reduced by:

- Leveraging browser caching
- Eliminating render-blocking JavaScript and CSS
- Avoiding landing page redirects

Finally, more optimisation can be done by:

- Loading visible content before CSS and JS files
- Reducing server response time (not an issue on localhost)

Chapter 12

Learning Points

This year-long project made me go through several iterations of workloads which were very enriching and helped develop my skills in a number of ways. This project was very multi-dimensional and it took a lot of different *kinds* of skills to overcome the various obstacles encountered throughout the project. From algorithmic optimisation to data visualisation, this project used the full breath of all techniques learnt in Computer Science. This chapter gives insights on the main learning points of this project.

From a technical point of view, there was an vast amount of small learning experiences related to software development, data science and algorithms, and each contributed to improving my technical skills.

For example, my Python programming skills were considerably improved by having to work regularly on this language with which I was originally fairly unfamiliar. Additionally, due to the all-inclusive nature of my project, I had to develop skills in other areas I lacked experience in, such as web development and front-end designing.

A novel experience was analysing and improving the efficiency of *my own* algorithms. When processing large quantities of data, the entire software can really slow down, and it is vital to improve the algorithms to their most efficient version possible. Learning to not neglect optimising my code was almost as big a discovery as initially learning how to code.

Even more important was perhaps working on my debugging skills, and practising solving issues that were caused by the multifaceted nature of the project. Trying to *find* the source of unidentifiable bugs in several scripts was a distinctly educational experience.

Furthermore, I got a deeper understanding of the value of proper documentation. As my project involved back-tracking at times, and re-developing certain parts, clear coding and documentation were indispensable to not get lost. Using Git and Github was another valuable experience, and I was able back up my code at every important iteration.

However, while technical knowledge and rigour remain the bedrock of any scientific endeavour, I found myself also truly learning and appreciating the merit of essential *non-technical* skills.

Being a completely independent project, I learnt to take full responsibility of delivering a final product. The regular assessments and their relevant feedback allowed me to slowly gain confidence, and allowed me to become more bold and decisive in my work.

Moreover, time management was a key factor in delivering the required products in time. This involved learning to make decisions on time, even if it meant giving up on some ideas. Computer Science essentially is a constant decision making processing on the approach to take, and can never be completely assessed from the outset. Learning, however, to become a better judge of it and to trust my intuition was an important learning point.

Over and above that, I also learnt how and when to ask for assistance, as trying to do everything by oneself is often counter-productive. Learning to work with my supervisor and staying on a viable timeline was important in being able to converge my ideas into one single complete project.

My ability to assimilate theoretical concepts was also exercised, as I often had to reflect to comprehend abstract information relating to machine learning for several days before being able to fully process them.

Lastly, and possibly the most remarkable element I felt was the sentiment of empowerment when finally completing the implementation of this personal project. It is the strongest feeling I associate with Computer Science, as I feel this entire course gives us the tools to realise our dreams and turn them into reality regardless of their ambition, scope, and technicality.

This experience has only left me wanting to do and learn more and I hope to continue working in an environment which allows me expand my repertoire of skills and thus grow both as a developer and a person.

Chapter 13

Professional Issues

As academics, it is our duty to ensure our projects are well within the principles of the British Computer Society. In particular, any Computer Science projects should follow the established common practices of relevance, and respect the key practices specific to particular IT skills.

This project is fully in accordance with British Computer Society's code of conduct. As explained in Chapter 4, these are freely accessible in the public domain, or else randomly generated in a Python script, and hence in line with the BSC's guidelines on confidentiality. Additionally, the participant's evaluation sheets were completely *anonymous* and no data was stored or collected.

Furthermore, actual code of all scripts follow the code of conduct's principles on good programming. The code is well organised, documented, and appropriately structured as highlighted in the BSC's framework of guidance.

Moreover, all sources for this project have always been cited or appropriatly listed in the Bibliography Section.

All of the following items have been followed and respected as well:

Practice common to all disciplines

- Adhere to regulations
- Act professionally as a specialist
- Use appropriate methods and tools
- Manage your workload efficiently
- Promote good practices within your organisation
- Represent the profession to the public

Key IT practice

- When managing a programme of work:
- When planning
- When closing a project

In conclusion, this project fully respects the rules defined in the BCS code of conduct with full professional competence, integrity and duty to the relevant authorities.

Appendix A

Source Codes

A.1 sort.py

```
# Name: Eklavya SARKAR,
# ID:201135564,
# Username: u5es2

# Sort the EMNIST Balanced 47 Classes (training or testing) data
# Sequence: digits (0-9), then capital letters (A-Z), then small letters
    (selected ones from a-z)

import argparse
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#--------------------------------------------------------------------------
# CONFIG
#--------------------------------------------------------------------------

# Argument Parser
parser = argparse.ArgumentParser(description='Sort the EMNIST data in
    order of their class')
parser.add_argument('-d','--debug', action='store_true', default=False,
    help='Print debug messages')
parser.add_argument('-c','--classes', action='store', type=int, help='
    Insert the number of different classes in the database to be sorted')
parser.add_argument('-ip','--input_path', action='store', help='Insert
    the data path to the .csv file')
parser.add_argument('-sp','--save_path', action='store', help='Insert
    the save path for the sorted output .csv file (do not insert the file
     name itself)')
args = parser.parse_args()

#--------------------------------------------------------------------------
# SET-UP
#--------------------------------------------------------------------------

# Enough arguments given
if not (args.input_path):
  print('ERROR - No input path given')
  print('Use -ip to insert the input file path, eg: -p=/Users/input_path
   /input_file.csv')
  sys.exit(1)

if not (args.save_path):
  print('ERROR - No save path given')
  print('Use -sp to insert a file save path, eg: -sp=/Users/save_path/')
```

```
  sys.exit(1)

if not (args.classes):
  print('ERROR - Number of classes not given')
  print('Use -c to input the total number of classes in the dataset, eg
    -c=47:')
  sys.exit(1)

# Read arguments
if args.input_path:
  data_path = args.input_path

if args.save_path:
  save_path = args.save_path

if args.classes:
  max_classes = args.classes

# Read raw data
data = pd.read_csv(data_path, encoding='utf-8', header=None)

if (args.debug):
  print('Number of classes', max_classes)
  print('Input path', data_path)
  print('Save path', save_path)
  print('')
  print('Raw data shape:', data.shape)
  print(type(data))

#---------------------------------------------------------------------------
# SORTING
#---------------------------------------------------------------------------

# Sorting into classes
# Numpy arrays are immutable, and are very inefficient for appending,
# as they create a new array, then copy entire rows/columns onto it
# We therefore use python lists (mutable), then later convert them to
    Numpy array

sortedInputs = []
sortedLabels = []

max_inputs_per_class = data.shape[0]//max_classes

# Number of classes

# Numpy arrays are immutable, and are very inefficient for appending
# (they create a new array, then copy entire rows/columns onto it).
# We therefore use python lists (mutable), then convert them to Numpy
    array

# Create lists per class
arr_0 = []
arr_1 = []
arr_2 = []
arr_3 = []
arr_4 = []
arr_5 = []
arr_6 = []
arr_7 = []
arr_8 = []
arr_9 = []
arr_10 = []
```

```
arr_11 = []
arr_12 = []
arr_13 = []
arr_14 = []
arr_15 = []
arr_16 = []
arr_17 = []
arr_18 = []
arr_19 = []
arr_20 = []
arr_21 = []
arr_22 = []
arr_23 = []
arr_24 = []
arr_25 = []
arr_26 = []
arr_27 = []
arr_28 = []
arr_29 = []
arr_30 = []
arr_31 = []
arr_32 = []
arr_33 = []
arr_34 = []
arr_35 = []
arr_36 = []
arr_37 = []
arr_38 = []
arr_39 = []
arr_40 = []
arr_41 = []
arr_42 = []
arr_43 = []
arr_44 = []
arr_45 = []
arr_46 = []

if (args.debug):
  print('Starting sorting')

# Sort and append according to class
for i in range(data.shape[0]):
    if data.iloc[i,0]==0:
        arr_0.append(data.iloc[i,1:])
    elif data.iloc[i,0]==1:
        arr_1.append(data.iloc[i,1:])
    elif data.iloc[i,0]==2:
        arr_2.append(data.iloc[i,1:])
    elif data.iloc[i,0]==3:
        arr_3.append(data.iloc[i,1:])
    elif data.iloc[i,0]==4:
        arr_4.append(data.iloc[i,1:])
    elif data.iloc[i,0]==5:
        arr_5.append(data.iloc[i,1:])
    elif data.iloc[i,0]==6:
        arr_6.append(data.iloc[i,1:])
    elif data.iloc[i,0]==7:
        arr_7.append(data.iloc[i,1:])
    elif data.iloc[i,0]==8:
        arr_8.append(data.iloc[i,1:])
    elif data.iloc[i,0]==9:
        arr_9.append(data.iloc[i,1:])
    elif data.iloc[i,0]==10:
```

```
            arr_10.append(data.iloc[i,1:])
        elif data.iloc[i,0]==11:
            arr_11.append(data.iloc[i,1:])
        elif data.iloc[i,0]==12:
            arr_12.append(data.iloc[i,1:])
        elif data.iloc[i,0]==13:
            arr_13.append(data.iloc[i,1:])
        elif data.iloc[i,0]==14:
            arr_14.append(data.iloc[i,1:])
        elif data.iloc[i,0]==15:
            arr_15.append(data.iloc[i,1:])
        elif data.iloc[i,0]==16:
            arr_16.append(data.iloc[i,1:])
        elif data.iloc[i,0]==17:
            arr_17.append(data.iloc[i,1:])
        elif data.iloc[i,0]==18:
            arr_18.append(data.iloc[i,1:])
        elif data.iloc[i,0]==19:
            arr_19.append(data.iloc[i,1:])
        elif data.iloc[i,0]==20:
            arr_20.append(data.iloc[i,1:])
        elif data.iloc[i,0]==21:
            arr_21.append(data.iloc[i,1:])
        elif data.iloc[i,0]==22:
            arr_22.append(data.iloc[i,1:])
        elif data.iloc[i,0]==23:
            arr_23.append(data.iloc[i,1:])
        elif data.iloc[i,0]==24:
            arr_24.append(data.iloc[i,1:])
        elif data.iloc[i,0]==25:
            arr_25.append(data.iloc[i,1:])
        elif data.iloc[i,0]==26:
            arr_26.append(data.iloc[i,1:])
        elif data.iloc[i,0]==27:
            arr_27.append(data.iloc[i,1:])
        elif data.iloc[i,0]==28:
            arr_28.append(data.iloc[i,1:])
        elif data.iloc[i,0]==29:
            arr_29.append(data.iloc[i,1:])
        elif data.iloc[i,0]==30:
            arr_30.append(data.iloc[i,1:])
        elif data.iloc[i,0]==31:
            arr_31.append(data.iloc[i,1:])
        elif data.iloc[i,0]==32:
            arr_32.append(data.iloc[i,1:])
        elif data.iloc[i,0]==33:
            arr_33.append(data.iloc[i,1:])
        elif data.iloc[i,0]==34:
            arr_34.append(data.iloc[i,1:])
        elif data.iloc[i,0]==35:
            arr_35.append(data.iloc[i,1:])
        elif data.iloc[i,0]==36:
            arr_36.append(data.iloc[i,1:])
        elif data.iloc[i,0]==37:
            arr_37.append(data.iloc[i,1:])
        elif data.iloc[i,0]==38:
            arr_38.append(data.iloc[i,1:])
        elif data.iloc[i,0]==39:
            arr_39.append(data.iloc[i,1:])
        elif data.iloc[i,0]==40:
            arr_40.append(data.iloc[i,1:])
        elif data.iloc[i,0]==41:
            arr_41.append(data.iloc[i,1:])
```

```
    elif data.iloc[i,0]==42:
        arr_42.append(data.iloc[i,1:])
    elif data.iloc[i,0]==43:
        arr_43.append(data.iloc[i,1:])
    elif data.iloc[i,0]==44:
        arr_44.append(data.iloc[i,1:])
    elif data.iloc[i,0]==45:
        arr_45.append(data.iloc[i,1:])
    else: # == 46
        arr_46.append(data.iloc[i,1:])

if (args.debug):
  print('Finished sorting')

# Merge in order into main list
sortedInputs.extend(arr_0+
arr_1+
arr_2+
arr_3+
arr_4+
arr_5+
arr_6+
arr_7+
arr_8+
arr_9+
arr_10+
arr_11+
arr_12+
arr_13+
arr_14+
arr_15+
arr_16+
arr_17+
arr_18+
arr_19+
arr_20+
arr_21+
arr_22+
arr_23+
arr_24+
arr_25+
arr_26+
arr_27+
arr_28+
arr_29+
arr_30+
arr_31+
arr_32+
arr_33+
arr_34+
arr_35+
arr_36+
arr_37+
arr_38+
arr_39+
arr_40+
arr_41+
arr_42+
arr_43+
arr_44+
arr_45+
arr_46)
```

```python
if (args.debug):
  print('Starting labelling')

# Make sorted labels list
i = 0
for x in range(0, data.shape[0], max_inputs_per_class):
    for y in range (max_inputs_per_class):
        sortedLabels.append(i)
    i=i+1

if (args.debug):
  print('Finished labelling')

# Convert both lists to NumPy arrays
sortedInputs = np.array(sortedInputs)
sortedLabels = np.array(sortedLabels)

# View on Matplotlib to check
def display(n_cols, n_rows, x):

  fig, ax = plt.subplots(n_rows, n_cols, sharex='col', sharey='row')

  for i in range(n_rows):
      for j in range(n_cols):
          pic = np.rot90((np.fliplr(sortedInputs[x,:].reshape((28,28)))))
          ax[i, j].imshow(pic, cmap='gray')
          ax[i, j].axis('off')
          x+=1
  plt.show()

if (args.debug):
  print('Sorted data shape:',sortedInputs.shape)
  print('Sorted labels shape:',sortedLabels.shape)
  # display(5,5,0)

#-------------------------------------------------------------------------------
# EXPORT
#-------------------------------------------------------------------------------

# Make sure to change file name to not overwrite files in case you sort both training and testing files
np.savetxt(save_path+'SortedInputs.csv', sortedInputs, fmt='%d', delimiter=',')
np.savetxt(save_path+'SortedLabels.txt', sortedLabels, fmt='%d')

if (args.debug):
  print('Sorted inputs saved at ' + save_path)
  print('Sorted labels saved at ' + save_path)
```

Listing A.1: Sorting code

A.2 RGB.py

```
# Name: Eklavya SARKAR,
# ID:201135564,
# Username: u5es2

# We're using sorted EMNIST Balanced 47 Classes data, to make a SOM

import argparse
import sys
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#-------------------------------------------------------------------------------
# CONFIG
#-------------------------------------------------------------------------------

# Argument Parser for debugging
parser = argparse.ArgumentParser(description='Make a 2D map of a multidimensional input')
parser.add_argument('-d','--debug', action='store_true', default=False, help='Print debug messages to stderr')
parser.add_argument('-r','--rate', type=float, action='store', default=0.3, help='Choose learning rate (range: 0-1)')
parser.add_argument('-i','--inputs', type=int, action='store', default=20, help='Choose number of train inputs per class (range: 0-2400)')
args = parser.parse_args()

#-------------------------------------------------------------------------------
# SET-UP
#-------------------------------------------------------------------------------

if (args.inputs):
  if (args.inputs < 0):
    print('ERROR - The number of inputs cannot be lower than 0.')
    print('Use -i to insert the correct number of inputs, eg: -i=20.')
    sys.exit(1)
  else:
    inputsQuantity = args.inputs

elif (args.inputs == 0):
  print('ERROR - The number of inputs cannot be equal to 0.')
  print('Use -i to insert the correct number of inputs, eg: -i=20.')
  sys.exit(1)

# Constants
# ========== DO NOT CHANGE ==========|
MAX_CLASSES = 10                    #|
INPUTS_PER_CLASS = inputsQuantity#|
# ==========DO NOT CHANGE==========|

if args.debug:
  print("Debug mode ON")
  print('Loading input files ...')

# We can generate random vectors in range [0-255] with the three values R,G,B
data = np.random.randint(0, 255, (INPUTS_PER_CLASS, 3))

INPUTS_MAX_VALUE = data.max()
```

```

# Normalise and convert from list to array
inputs = []
inputs = data/INPUTS_MAX_VALUE
inputs = np.array(inputs)

if args.debug:
  print('Generated inputs:', type(inputs))
  if (inputs.max()==1 and inputs.min()==0):
    normaliseCheck = True
  else:
    normaliseCheck = False
  print('Data normalised:', normaliseCheck)

# Variables
n = inputs.shape[0]
m = inputs.shape[1]

n_classes = MAX_CLASSES
network_dimensions = np.array([n_classes*2,n_classes*2])

n_iterations = n

# Learning rate (Eta), range: 0 - 1
if (args.rate):
  if (args.rate < 0):
    print('ERROR - The learning cannot be lower than 0.')
    print('Use -r to insert the correct learning rate, eg: -r=0.3.')
    sys.exit(1)
  elif (args.rate > 1):
    print('ERROR - The learning cannot be bigger than 1.')
    print('Use -r to insert the correct learning rate, eg: -r=0.3.')
    sys.exit(1)
  else:
    init_learning_rate =  args.rate
elif (args.rate == 0):
  print('ERROR - The learning cannot be equal to 0.')
  print('Use -r to insert the correct learning rate, eg: -r=0.3.')
  sys.exit(1)

if args.debug:
  print('n_classes:', n_classes)
  print('n:', n)
  print('m:', m)
  print('Network dimensions:', network_dimensions.shape)
  print('Number of training iterations:', n_iterations)
  print('Initial learning rate:', init_learning_rate)

# Variables

# Weight Matrix - same for training and testing as same number of
    classes and therefore network dimensions
net = np.random.random((network_dimensions[0], network_dimensions[1], m)
    )

# Initial Radius (sigma) for the neighbourhood - same for tranining and
    testing as same network dimensions
init_radius = max(network_dimensions[0], network_dimensions[1]) / 2

# Radius decay parameter - different as (possibly) different number of
    iterations
time_constant = n_iterations / np.log(init_radius)

```

```
if args.debug:
  print('Net', type(net))
  print('Initial Radius', init_radius)
  print('Time constant', time_constant)

#-----------------------------------------------------------------------------
# METHODS
#-----------------------------------------------------------------------------

# Find Best Matching Unit (BMU)
def findBMU(t, net, m):

  # A 1D array which will contain the X,Y coordinates
  # of the BMU for the given input vector t
  bmu_idx = np.array([0,0])

  # Set the initial minimum difference
  min_diff = np.iinfo(np.int).max

  # To compute the high-dimension distance between
  # the given input vector and each neuron,
  # we calculate the difference between the vectors
  for x in range (net.shape[0]):
    for y in range(net.shape[1]):
      w = net[x,y,:].reshape(m, 1)

      # Don't sqrt to avoid heavy operation
      diff = np.sum((w - t) ** 2)

      if (diff < min_diff):
        min_diff = diff
        bmu_idx = np.array([x, y])

  bmu = net[bmu_idx[0], bmu_idx[1], :].reshape(m, 1)

  return(bmu, bmu_idx, min_diff)

# Decay the neighbourhood radius with time
def decayRadius(initial_radius, i, time_constant):
  return initial_radius * np.exp(-i / time_constant)

# Decay the learning rate with time
def decayLearningRate(initial_learning_rate, i, n_iterations):
  return initial_learning_rate * np.exp(-i / n_iterations)

# Calculate the influence
def getInfluence(distance, radius):
  return np.exp(-distance / (2* (radius**2)))

# SOM Step Learning
def trainSOM(inputsValues, times):

  bmu_idx_arr = []
  radiusList = []
  learnRateList = []
  sqDistList = []

  for i in range (times):

    if args.debug:
      print(str(round(i/times*100))+'%')

    # ------------- INPUT -------------
```

```
    # 1. Select a input weight vector at each step

    # This can be random, however since we're using sorted inputs, we're
    # proceeding in a linear manner through all nodes for sake of clarity
    t = inputsValues[i, :].reshape(np.array([m, 1]))

    # ------------------ BMU ------------------
    # 2. Find the chosen input vector's BMU at each step
    #bmu, bmu_idx = findBMU(t, net, m)
    bmu, bmu_idx, dist = findBMU(t, net, m)

    bmu_idx_arr.append(bmu_idx)
    sqDistList.append(dist)

    # ------------------ DECAY ------------------
    # 3. Determine topological neighbourhood for each step
    r = decayRadius(init_radius, i, time_constant)
    l = decayLearningRate(init_learning_rate, i, times)

    radiusList.append(r)
    learnRateList.append(l)

    # ------------------ UPDATE ------------------
    # 4. Repeat for all nodes in the *BMU neighbourhood*
    for x in range(net.shape[0]):
      for y in range(net.shape[1]):

        # Find weight vector
        w = net[x, y, :].reshape(m, 1)
        #wList.append(w)

        # Get the 2-D distance (not Euclidean as no sqrt)
        w_dist = np.sum((np.array([x, y]) - bmu_idx) ** 2)
        #wDistList.append(w_dist)

        # If the distance is within the current neighbourhood radius
        if w_dist <= r**2:

          # Calculate the degree of influence (based on the 2-D distance )
          influence = getInfluence(w_dist, r)

          # Update weight:
          # new w = old w + (learning rate * influence * delta)
          # delta = input vector t - old w
          new_w = w + (l * influence * (t - w))
          #new_wList.append(new_w)

          # Update net with new weight
          net[x, y, :] = new_w.reshape(1, m)

    # Every 100 iterations we call for a SOM to be made to view
    #if (i>0 and i%100==0):
    # bmu_interim_arr = np.array(bmu_idx_arr)
    # makeSOM(bmu_interim_arr, labels, [], [])

  # Convert to NumPy array
  bmu_idx_arr = np.array(bmu_idx_arr)

  #np.savetxt((save_path+'%s'%timeStamped()+'_%s'%n_classes+'classes'+'_%s'%init_learning_rate+'rate'+'_%s'%chosen_inputs_per_class+'inputs'+'.csv'), bmu_idx_arr, fmt='%d', delimiter=',')
```

```
  #np.savetxt((save_path+'Net_%s'%timeStamped()+'.txt'), net, fmt='%d')

  return(bmu_idx_arr, radiusList, learnRateList, sqDistList)


def makeSOM(bmu_idx_arr):

  plotVector = np.zeros((n,5))
  x_coords = []
  y_coords = []

  x_coords = np.random.randint(0, network_dimensions[0],
    INPUTS_PER_CLASS)
  y_coords = np.random.randint(0, network_dimensions[0],
    INPUTS_PER_CLASS)

  x_coords = np.array(x_coords)
  y_coords = np.array(y_coords)

  # plotVector Format: [X, Y, R, G, B]
  # Coordinates and colours in a single vector

  # Insert training values
  for i in range(n):
    # X, Ys - Coordinates with added noise
    plotVector[i][0] = bmu_idx_arr[i][0]
    plotVector[i][1] = bmu_idx_arr[i][1]

    # R,G,Bs - Color each point according to class
    plotVector[i][2] = inputs[i][0]
    plotVector[i][3] = inputs[i][1]
    plotVector[i][4] = inputs[i][2]

  # Generate noise for each point
  if (plotVector.shape[0] > 0):
    a_x = -0.4
    a_y = -0.4
    b_x = 0.4
    b_y = 0.4

    noise_x = (b_x-a_x) * np.random.rand(plotVector.shape[0], 1) + a_x
    noise_y = (b_y-a_y) * np.random.rand(plotVector.shape[0], 1) + a_y

  zPlot = np.array(plotVector[:,2:5])

  # With noise
  xPlotNoise = np.add(plotVector[:,0], noise_x[:,0])
  yPlotNoise = np.add(plotVector[:,1], noise_y[:,0])

  x_coordsNoise = np.add(x_coords[:], noise_x[:,0])
  y_coordsNoise = np.add(y_coords[:], noise_y[:,0])

  # Witout noise
  xPlot = plotVector[:,0]
  yPlot = plotVector[:,1]

  if (args.debug):
    print('Rate:', init_learning_rate)
    print('x:', xPlot.shape)
    print('y:', yPlot.shape)
    print('z:', zPlot.shape)
    print('BMUs:', bmu_idx_arr.shape)
    print(zPlot[0])
```

```

# Plot Scatterplot
plotSize = (n_classes * 2)
figSize = 5.91
plt.figure()

# Plot nodes
plt.scatter(x_coords, y_coords, s=20, facecolor=zPlot)
plt.title(str(n)+' Inputs unsorted without noise')
plt.show()

# Plot nodes with noise
plt.scatter(x_coordsNoise, y_coordsNoise, s=20, facecolor=zPlot)
plt.title(str(n)+' Inputs unsorted with noise')
plt.show()

# Plot data without noise
plt.scatter(xPlot, yPlot, s=20, marker='o', facecolor=zPlot)
plt.title(str(n)+' Inputs sorted without noise')
plt.show()

# Plot data with noise
plt.scatter(xPlotNoise, yPlotNoise, s=20, marker='o', facecolor=zPlot)
plt.title(str(n)+' Inputs sorted with noise')
plt.show()

# Legend
#for i in range(10):
# plt.scatter(i, 1, s=20, facecolor=zPlot[i])

#for i in range(n):
# plt.text(xPlot[0], yPlot[1], labels[i], ha='center', va='center')

#plt.legend(handles=[n])

#plt.axis('off')

# Export as CSV
unClustered = np.zeros((n,5))
unClusteredNoise = np.zeros((n,5))
clustered = np.zeros((n,5))
clusteredNoise = np.zeros((n,5))

unClustered[:,0] = x_coords[:]
unClustered[:,1] = y_coords[:]
unClustered[:,2:5] = data[:]

unClusteredNoise[:,0] = x_coordsNoise[:]
unClusteredNoise[:,1] = y_coordsNoise[:]
unClusteredNoise[:,2:5] = data[:]

clustered[:,0] = xPlot[:]
clustered[:,1] = yPlot[:]
clustered[:,2:5] = data[:]

clusteredNoise[:,0] = xPlotNoise[:]
clusteredNoise[:,1] = yPlotNoise[:]
clusteredNoise[:,2:5] = data[:]

np.savetxt(('static/data/RGB/RGBUnsorted.csv'), unClustered, fmt='%d',
    delimiter=',', comments='', header='xRGB,yRGB,R,G,B')
np.savetxt(('static/data/RGB/RGBUnsortedNoise.csv'), unClusteredNoise,
    fmt='%.3f', delimiter=',', comments='', header='xRGB,yRGB,R,G,B')
```

```
    np.savetxt(('static/data/RGB/RGBSorted.csv'), clustered, fmt='%d', delimiter=',', comments='', header='xRGB,yRGB,R,G,B')
    np.savetxt(('static/data/RGB/RGBSortedNoise.csv'), clusteredNoise, fmt='%.3f', delimiter=',', comments='', header='xRGB,yRGB,R,G,B')

    if args.debug:
      print('Saved unsorted coordinates')
      print('Saved unsorted coordinates with noise')
      print('Saved sorted coordinates')
      print('Saved sorted coordinates with noise')

# Make graphical comparaisons of various parameters
def plotVariables(radius, learnRate, sqDist):

    # Plot radius
    plt.title('Radius evolution')
    plt.xlabel('Number of iterations')
    plt.ylabel('Radius size')
    plt.plot(radius, 'r', label='Radius')
    plt.legend(loc=1)
    plt.show()

    # Plot learning rate
    plt.title('Learning rate evolution')
    plt.xlabel('Number of iterations')
    plt.ylabel('Learning rate')
    plt.plot(learnRate, 'r', label='Learning Rate')
    plt.legend(loc=1)
    plt.show()

    # Plot 3D distance
    plt.title('Best Matching Unit 3D Distance')
    plt.xlabel('Number of iterations')
    plt.ylabel('Smallest Distance Squared')
    plt.plot(sqDist, 'r', label='(Squared) Distance')
    plt.legend(loc=1)
    plt.show()

#------------------------------------------------------------------------
# MAIN METHODS CALL
#------------------------------------------------------------------------
#inputs = setUp(inputsQuantity)
bmu, radius, rate, sqDist = trainSOM(inputs, inputsQuantity)
makeSOM(bmu)
plotVariables(radius, rate, sqDist)
```

LISTING A.2: RGB SOM code

A.3 Iris.py

```
# Name: Eklavya SARKAR,
# ID:201135564,
# Username: u5es2

# We're using the Iris dataset to train an ANN
import argparse
import sys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

#-----------------------------------------------------------------------
# CONFIG
#-----------------------------------------------------------------------

# Argument Parser for debugging
parser = argparse.ArgumentParser(description='Make a 2D map of a
    multidimensional input')
parser.add_argument('-d','--debug', action='store_true', default=False,
    help='Print debug messages to stderr')
parser.add_argument('-r','--rate', type=float, action='store', default
    =0.3, help='Choose learning rate (range: 0-1)')
args = parser.parse_args()

#-----------------------------------------------------------------------
# SET-UP
#-----------------------------------------------------------------------

# Constants
# ========== DO NOT CHANGE ==========|
INPUTS_MAX_VALUE = 7.9         #|
MAX_CLASSES = 3                #|
MAX_INPUTS_PER_CLASS = 50      #|
# ==========DO NOT CHANGE ==========|

chosen_inputs_per_class = 50
n_classes = MAX_CLASSES

# Learning rate (Eta), range: 0 - 1
if (args.rate):
  if (args.rate < 0):
    print('ERROR - The learning cannot be lower than 0.')
    print('Use -r to insert the correct learning rate, eg: -r=0.3.')
    sys.exit(1)
  elif (args.rate > 1):
    print('ERROR - The learning cannot be bigger than 1.')
    print('Use -r to insert the correct learning rate, eg: -r=0.3.')
    sys.exit(1)
  else:
    init_learning_rate =  args.rate
elif (args.rate == 0):
  print('ERROR - The learning cannot be equal to 0.')
  print('Use -r to insert the correct learning rate, eg: -r=0.3.')
  sys.exit(1)

if args.debug:
  print("Debug mode ON")
  print('Loading input files ...')
```

```
# Raw Data
#data_path = 'static/data/Iris/IrisOriginal.csv'
data_path = 'http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
data = pd.read_csv(data_path, encoding='utf-8', header=None)

# Add Column names
attributes = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"]
data.columns = attributes

# Looping
loopStart = 0
loopEnd = MAX_CLASSES*MAX_INPUTS_PER_CLASS
labels = []
inputs = []

for i in range(loopStart,loopEnd,MAX_INPUTS_PER_CLASS):
    for j in range(chosen_inputs_per_class):
        inputs.append(data.iloc[i+j][0:4]/INPUTS_MAX_VALUE) # Append normalised value
        labels.append(data.iloc[i][4])

# Put labels in seperate NumPy array
#labels = np.array(data['class'])
labels = np.array(labels)

# Put inputs in a a seperate NumPy Array, while normalising it
#inputs = np.array(data[["sepal_length", "sepal_width", "petal_length", "petal_width"]]/inputs.max())
inputs = np.array(inputs)

if args.debug:
  if (inputs.max()==1 and inputs.min()==0):
    normaliseCheck = True
  else:
    normaliseCheck = False

  print('Loaded inputs:', type(inputs))
  print('Loaded labels:', type(labels))
  print('Data normalised:', normaliseCheck)

# Variables
n = inputs.shape[0]
m = inputs.shape[1]

network_dimensions = np.array([n_classes*2,n_classes*2])
n_iterations = n

if args.debug:
  print('n_classes:', n_classes)
  print('n:', n)
  print('m:', m)
  print('Network dimensions:', network_dimensions.shape)
  print('Number of training iterations:', n_iterations)
  print('Initial learning rate:', init_learning_rate)
  print('Inputs per class:', chosen_inputs_per_class)


# Weight Matrix - same for training and testing as same number of classes and therefore network dimensions
net = np.random.random((network_dimensions[0], network_dimensions[1], m))
```

```python

# Initial Radius (sigma) for the neighbourhood - same for tranining and testing as same network dimensions
init_radius = max(network_dimensions[0], network_dimensions[1]) / 2

# Radius decay parameter - different as (possibly) different number of iterations
time_constant = n_iterations / np.log(init_radius)

if args.debug:
  print('Net', type(net))
  print('Initial Radius', init_radius)
  print('Time constant', time_constant)

#-----------------------------------------------------------------------
# METHODS
#-----------------------------------------------------------------------

# Find Best Matching Unit (BMU)
def findBMU(t, net, m):

  # A 1D array which will contain the X,Y coordinates
  # of the BMU for the given input vector t
  bmu_idx = np.array([0,0])

  # Set the initial minimum difference
  min_diff = np.iinfo(np.int).max

  # To compute the high-dimension distance between
  # the given input vector and each neuron,
  # we calculate the difference between the vectors
  for x in range (net.shape[0]):
    for y in range(net.shape[1]):
      w = net[x,y,:].reshape(m, 1)

      # Don't sqrt to avoid heavy operation
      diff = np.sum((w - t) ** 2)

      if (diff < min_diff):
        min_diff = diff
        bmu_idx = np.array([x, y])

  bmu = net[bmu_idx[0], bmu_idx[1], :].reshape(m, 1)

  return(bmu, bmu_idx, min_diff)

# Decay the neighbourhood radius with time
def decayRadius(initial_radius, i, time_constant):
  return initial_radius * np.exp(-i / time_constant)

# Decay the learning rate with time
def decayLearningRate(initial_learning_rate, i, n_iterations):
  return initial_learning_rate * np.exp(-i / n_iterations)

# Calculate the influence
def getInfluence(distance, radius):
  return np.exp(-distance / (2* (radius**2)))

# SOM Step Learning
def trainSOM(inputsValues, times):

  bmu_idx_arr = []
  radiusList = []
```

```
learnRateList = []
sqDistList = []

for i in range (times):

  if args.debug:
    print(str(round(i/times*100))+'%')

  # ------------ INPUT ------------
  # 1. Select a input weight vector at each step

  # This can be random, however since we're using sorted inputs, we're
  # proceeding in a linear manner through all nodes for sake of clarity
  t = inputsValues[i, :].reshape(np.array([m, 1]))

  # ------------ BMU ------------
  # 2. Find the chosen input vector's BMU at each step
  #bmu, bmu_idx = findBMU(t, net, m)
  bmu, bmu_idx, dist = findBMU(t, net, m)

  bmu_idx_arr.append(bmu_idx)
  sqDistList.append(dist)

  # ------------ DECAY ------------
  # 3. Determine topological neighbourhood for each step
  r = decayRadius(init_radius, i, time_constant)
  l = decayLearningRate(init_learning_rate, i, times)

  radiusList.append(r)
  learnRateList.append(l)

  # ------------ UPDATE ------------
  # 4. Repeat for all nodes in the *BMU neighbourhood*
  for x in range(net.shape[0]):
    for y in range(net.shape[1]):

      # Find weight vector
      w = net[x, y, :].reshape(m, 1)
      #wList.append(w)

      # Get the 2-D distance (not Euclidean as no sqrt)
      w_dist = np.sum((np.array([x, y]) - bmu_idx) ** 2)
      #wDistList.append(w_dist)

      # If the distance is within the current neighbourhood radius
      if w_dist <= r**2:

        # Calculate the degree of influence (based on the 2-D distance)
        influence = getInfluence(w_dist, r)

        # Update weight:
        # new w = old w + (learning rate * influence * delta)
        # delta = input vector t - old w
        new_w = w + (l * influence * (t - w))
        #new_wList.append(new_w)

        # Update net with new weight
        net[x, y, :] = new_w.reshape(1, m)

  # Every 100 iterations we call for a SOM to be made to view
  #if (i>0 and i%100==0):
```

```
    # bmu_interim_arr = np.array(bmu_idx_arr)
    # makeSOM(bmu_interim_arr, labels, [], [])

  # Convert to NumPy array
  bmu_idx_arr = np.array(bmu_idx_arr)

  #np.savetxt((save_path+'%s'%timeStamped()+'_%s'%n_classes+'classes'+'_%s'%init_learning_rate+'rate'+'_%s'%chosen_inputs_per_class+'inputs'+'.csv'), bmu_idx_arr, fmt='%d', delimiter=',')
  #np.savetxt((save_path+'Net_%s'%timeStamped()+'.txt'), net, fmt='%d')

  return(bmu_idx_arr, radiusList, learnRateList, sqDistList)

def makeSOM(bmu_idx_arr):

  plotVector = np.zeros((n,5))

  x_coords = []
  y_coords = []

  x_coords = np.random.randint(0, 6, chosen_inputs_per_class*n_classes)
  y_coords = np.random.randint(0, 6, chosen_inputs_per_class*n_classes)

  x_coords = np.array(x_coords)
  y_coords = np.array(y_coords)

  # plotVector Format: [X, Y, R, G, B]
  # Coordinates and colours in a single vector

  # Insert training values
  for i in range(n):
    # X, Ys - Coordinates with added noise
    plotVector[i][0] = bmu_idx_arr[i][0]
    plotVector[i][1] = bmu_idx_arr[i][1]

    # R,G,Bs - Color each point according to class
    # RGB Values are normalised
    if (labels[i]=='Iris-setosa'):
      plotVector[i][2] = 1
      plotVector[i][3] = 0
      plotVector[i][4] = 0
    elif (labels[i]=='Iris-versicolor'):
      plotVector[i][2] = 0
      plotVector[i][3] = 1
      plotVector[i][4] = 0
    elif (labels[i]=='Iris-virginica'):
      plotVector[i][2] = 0
      plotVector[i][3] = 0
      plotVector[i][4] = 1

  # Generate noise for each point
  if (plotVector.shape[0] > 0):
    a_x = -0.4
    a_y = -0.4
    b_x = 0.4
    b_y = 0.4

    noise_x = (b_x-a_x) * np.random.rand(plotVector.shape[0], 1) + a_x
    noise_y = (b_y-a_y) * np.random.rand(plotVector.shape[0], 1) + a_y

  zPlot = np.array(plotVector[:,2:5])

  # With noise
```

```
    xPlotNoise = np.add(plotVector[:,0], noise_x[:,0])
    yPlotNoise = np.add(plotVector[:,1], noise_y[:,0])

    x_coordsNoise = np.add(x_coords[:], noise_x[:,0])
    y_coordsNoise = np.add(y_coords[:], noise_y[:,0])

    # Witout noise
    xPlot = plotVector[:,0]
    yPlot = plotVector[:,1]

    if (args.debug):
      print('Rate:', init_learning_rate)
      print('x:', xPlot.shape)
      print('y:', yPlot.shape)
      print('z:', zPlot.shape)
      print('BMUs:', bmu_idx_arr.shape)

    # Legend
    legend_elements = [ Line2D([0],[0], marker='o', color='r', label='Iris
      -setosa', markerfacecolor='r', markersize=5),
                        Line2D([0],[0], marker='o', color='g', label='Iris
      -versicolor', markerfacecolor='g', markersize=5),
                        Line2D([0],[0], marker='o', color='b', label='Iris
      -virginica', markerfacecolor='b', markersize=5)]

    # Plot Scatterplot
    plotSize = (n_classes * 2)
    figSize = 5.91
    plt.figure()

    # Plot nodes
    plt.scatter(x_coords, y_coords, s=20, facecolor=zPlot)
    plt.title(str(n)+' Inputs unsorted without noise')
    plt.legend(handles=legend_elements, loc=1)
    plt.show()

    # Plot nodes with noise
    plt.scatter(x_coordsNoise, y_coordsNoise, s=20, facecolor=zPlot)
    plt.title(str(n)+' Inputs unsorted with noise')
    plt.legend(handles=legend_elements, loc=1)
    plt.show()

    # Plot data without noise
    plt.scatter(xPlot, yPlot, s=20, marker='o', facecolor=zPlot)
    plt.title(str(n)+' Inputs sorted without noise')
    plt.legend(handles=legend_elements, loc=1)
    plt.show()

    # Plot data with noise
    plt.scatter(xPlotNoise, yPlotNoise, s=20, marker='o', facecolor=zPlot)
    plt.title(str(n)+' Inputs sorted with noise')
    plt.legend(handles=legend_elements, loc=1)
    plt.show()

    # Legend
    #for i in range(10):
    # plt.scatter(i, 1, s=20, facecolor=zPlot[i])

    #for i in range(n):
    # plt.text(xPlot[0], yPlot[1], labels[i], ha='center', va='center')

    #plt.legend(handles=[n])

```

```
    #plt.axis('off')

    # Export as CSV
    unClustered = np.zeros((n,5))
    unClusteredNoise = np.zeros((n,5))
    clustered = np.zeros((n,5))
    clusteredNoise = np.zeros((n,5))

    unClustered[:,0] = x_coords[:]
    unClustered[:,1] = y_coords[:]
    unClustered[:,2:5] = zPlot*255

    unClusteredNoise[:,0] = x_coordsNoise[:]
    unClusteredNoise[:,1] = y_coordsNoise[:]
    unClusteredNoise[:,2:5] = zPlot*255

    clustered[:,0] = xPlot[:]
    clustered[:,1] = yPlot[:]
    clustered[:,2:5] = zPlot*255 # Un-normalised

    clusteredNoise[:,0] = xPlotNoise[:]
    clusteredNoise[:,1] = yPlotNoise[:]
    clusteredNoise[:,2:5] = zPlot*255 # Un-normalised

    np.savetxt(('static/data/Iris/IrisUnsorted.csv'), unClustered, fmt='%d', delimiter=',', comments='', header='xIris,yIris,R,G,B')
    np.savetxt(('static/data/Iris/IrisUnsortedNoise.csv'), unClusteredNoise, fmt='%.3f', delimiter=',', comments='', header='xIris,yIris,R,G,B')
    np.savetxt(('static/data/Iris/IrisSorted.csv'), clustered, fmt='%d', delimiter=',', comments='', header='xIris,yIris,R,G,B')
    np.savetxt(('static/data/Iris/IrisSortedNoise.csv'), clusteredNoise, fmt='%.3f', delimiter=',', comments='', header='xIris,yIris,R,G,B')

    if args.debug:
        print('Saved sorted coordinates')
        print('Saved sorted coordinates with noise')

# Make graphical comparaisons of various parameters
def plotVariables(radius, learnRate, sqDist):

    # Plot radius
    plt.title('Radius evolution')
    plt.xlabel('Number of iterations')
    plt.ylabel('Radius size')
    plt.plot(radius, 'r', label='Radius')
    plt.legend(loc=1)
    plt.show()

    # Plot learning rate
    plt.title('Learning rate evolution')
    plt.xlabel('Number of iterations')
    plt.ylabel('Learning rate')
    plt.plot(learnRate, 'r', label='Learning Rate')
    plt.legend(loc=1)
    plt.show()

    # Plot 3D distance
    plt.title('Best Matching Unit 3D Distance')
    plt.xlabel('Number of iterations')
    plt.ylabel('Smallest Distance Squared')
    plt.plot(sqDist, 'r', label='(Squared) Distance')
    plt.legend(loc=1)
```

```
    plt.show()

#------------------------------------------------------------------------
# MAIN METHOD CALLS
#------------------------------------------------------------------------
bmu, radius, rate, sqDist = trainSOM(inputs, 150)
makeSOM(bmu)
plotVariables(radius, rate, sqDist)
```

LISTING A.3: Iris SOM code

A.4 SOM.py

```
# Name: Eklavya SARKAR,
# ID:201135564,
# Username: u5es2

# We're using sorted EMNIST Balanced 47 Classes data, to make a SOM

import argparse
import sys
import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Argument Parser for debugging
parser = argparse.ArgumentParser(description='Make a 2D map of a
    multidimensional input')
parser.add_argument('-d','--debug', action='store_true', default=False,
    help='Print debug messages to stderr')
parser.add_argument('-t','--type', action='store', default="d", help='
    Choose type of dataset: letters(=l), digits(=d), or combined(=c)')
parser.add_argument('-r','--rate', type=float, action='store', default
    =0.3, help='Choose learning rate (range: 0-1)')
parser.add_argument('-iTr','--inputsTrain', type=int, action='store',
    default=20, help='Choose number of train inputs per class (range:
    0-2400)')
parser.add_argument('-iTe','--inputsTest', type=int, action='store',
    default=20, help='Choose number of test inputs per class (range:
    0-400)')
args = parser.parse_args()

#----------------------------------------------------------------------
# CONFIG
#----------------------------------------------------------------------

# Constants
# ========== DO NOT CHANGE ==========|
INPUTS_MAX_VALUE = 255         #|
MAX_CLASSES = 47               #|
MAX_INPUTS_PER_CLASS = 2400    #|
MAX_TEST_INPUTS_PER_CLASS = 400 #|
# ==========DO NOT CHANGE ==========|

# Parameters configure according to given arguments
if not len(vars(args)) > 1:
  print('Using default values')

# Number of training inputs, range: 0 - 2400
if (args.inputsTrain):
  if (args.inputsTrain < 0):
    print('ERROR - The number of training inputs cannot be lower than 0.
    ')
    print('Use -iTr to insert a correct number of inputs, eg: -iTr=20.')
    sys.exit(1)
  if (args.inputsTrain > 2400):
    print('ERROR - The number of training inputs cannot be higher than
    2400.')
    print('Use -iTr to insert a correct number of inputs, eg: -iTr=20.')
    sys.exit(1)
  else:
    chosen_inputs_per_class = args.inputsTrain
```

```

elif (args.inputsTrain == 0):
  print('ERROR - The number of training inputs cannot be equal to 0.')
  print('Use -iTr to insert a correct number of inputs, eg: -iTr=20.')
  sys.exit(1)

# Number of testing inputs, range: 0 - 2400
if (args.inputsTest):
  if (args.inputsTest < 0):
    print('ERROR - The number of testing inputs cannot be lower than 0.')
    print('Use -iTe to insert a correct number of inputs, eg: -iTe=20.')
    sys.exit(1)
  if (args.inputsTest > 2400):
    print('ERROR - The number of testing inputs cannot be higher than 2400.')
    print('Use -iTe to insert a correct number of inputs, eg: -iTe=20.')
    sys.exit(1)
  else:
    chosen_test_inputs_per_class = args.inputsTest

elif (args.inputsTest == 0):
  print('ERROR - The number of testing inputs cannot be equal to 0.')
  print('Use -iTe to insert a correct number of inputs, eg: -iTe=20.')
  sys.exit(1)

# Learning rate (Eta), range: 0 - 1
if (args.rate):
  if (args.rate < 0):
    print('ERROR - The learning cannot be lower than 0.')
    print('Use -r to insert the correct learning rate, eg: -r=0.3.')
    sys.exit(1)
  elif (args.rate > 1):
    print('ERROR - The learning cannot be bigger than 1.')
    print('Use -r to insert the correct learning rate, eg: -r=0.3.')
    sys.exit(1)
  else:
    init_learning_rate =  args.rate
elif (args.rate == 0):
  print('ERROR - The learning cannot be equal to 0.')
  print('Use -r to insert the correct learning rate, eg: -r=0.3.')
  sys.exit(1)

# Number of classes
if (args.type == 'd'): # Digits
  n_classes = 10
elif (args.type == 'l'): # Letters
  n_classes = MAX_CLASSES-10
elif (args.type == 'c'): # Combined
  n_classes = MAX_CLASSES
else:
  print('ERROR - Invalid class type.')
  print('Use -t to insert the correct class type, eg: -t=d.')
  sys.exit(1)

#------------------------------------------------------------------------------
# SET-UP
#------------------------------------------------------------------------------

if args.debug:
  print("Debug mode ON")
  print('Loading input files ...')

```

```
# Inputs (Sorted inputs of all 47 classes)
#train_inputs_path = '/Users/eklavya/Movies/EMNIST_csv/Balanced/Sorted/SortedTrainInputs.csv'
train_inputs_path = 'http://cgi.csc.liv.ac.uk/~u5es2/EMNIST/Sorted/Train.csv'
train_inputs = pd.read_csv(train_inputs_path, encoding='utf-8', header=None)

#test_inputs_path = '/Users/eklavya/Movies/EMNIST_csv/Balanced/Sorted/SortedTestInputs.csv'
test_inputs_path = 'http://cgi.csc.liv.ac.uk/~u5es2/EMNIST/Sorted/Test.csv'
test_inputs = pd.read_csv(test_inputs_path, encoding='utf-8', header=None)

if args.debug:
  print('Loaded 1/3 files')

# Labels
#train_labels_path = '/Users/eklavya/Movies/EMNIST_csv/Balanced/Sorted/SortedTrainLabels.txt'
train_labels_path = 'http://cgi.csc.liv.ac.uk/~u5es2/EMNIST/Sorted/TrainLabels.txt'
train_labels = pd.read_csv(train_labels_path, encoding='utf-8', dtype=np.int8, header=None)

#test_labels_path = '/Users/eklavya/Movies/EMNIST_csv/Balanced/Sorted/SortedTestLabels.txt'
test_labels_path = 'http://cgi.csc.liv.ac.uk/~u5es2/EMNIST/Sorted/TestLabels.txt'
test_labels = pd.read_csv(test_labels_path, encoding='utf-8', dtype=np.int8, header=None)

# Drawn input
# drawn_path = '/Users/eklavya/Dropbox/__Liverpool/_390/SourceCode/EMNIST-Kohonen-SOM/static/data/drawn.csv'
# drawn_input = pd.read_csv(drawn_path, encoding='utf-8', header=None)

if args.debug:
  print('Loaded 2/3 files')

if (args.type == 'd'):
  colours_path = '/Users/eklavya/Dropbox/__Liverpool/_390/SourceCode/10Colors.csv'
  save_path = '/Users/Eklavya/Movies/EMNIST_csv/Balanced/Runs/Digits/'
elif (args.type == 'l'):
  colours_path = '/Users/eklavya/Dropbox/__Liverpool/_390/SourceCode/47Colors.csv'
  save_path = '/Users/Eklavya/Movies/EMNIST_csv/Balanced/Runs/Letters/'
else:
  colours_path = '/Users/eklavya/Dropbox/__Liverpool/_390/SourceCode/47Colors.csv'
  save_path = '/Users/Eklavya/Movies/EMNIST_csv/Balanced/Runs/Combined/'

class_colours = pd.read_csv(colours_path, encoding='utf-8', header=None)

if args.debug:
  print('Loaded 3/3 files')
  print('Save path:', save_path)

# bmu_path = '/Users/eklavya/Movies/EMNIST_csv/Balanced/Runs/Digits/2018-04-08-18-33-38_10classes_0.5rate_200inputs.csv'
# bmu_idx_arr = pd.read_csv(bmu_path, encoding='utf-8', header=None)
```

```
# bmu_idx_arr = np.array(bmu_idx_arr)

if args.debug:
  print('Loaded train inputs:', type(train_inputs))
  print('Loaded train labels:', type(train_labels))
  print('Loaded test inputs', type(test_inputs))
  print('Loaded test labels:', type(test_labels))
  print('Loaded colors:', type(class_colours))

inputs = []
labels = []

testInputs = []
testLabels = []

if (args.type == 'd'):
  # From 0 to 24000
  loopStart = 0
  loopEnd = 10*MAX_INPUTS_PER_CLASS

  # From 0 to 4000
  loopStartTest = 0
  loopEndTest = 10*MAX_TEST_INPUTS_PER_CLASS

elif (args.type == 'l'):
  # From 24000 to 112800
  loopStart = 10*MAX_INPUTS_PER_CLASS
  loopEnd = MAX_CLASSES*MAX_INPUTS_PER_CLASS

  # From 4000 to 18800
  loopStartTest = 10*MAX_TEST_INPUTS_PER_CLASS
  loopEndTest = MAX_CLASSES*MAX_TEST_INPUTS_PER_CLASS

elif (args.type == 'c'):
  # From 0 to 112800
  loopStart = 0
  loopEnd = MAX_CLASSES*MAX_INPUTS_PER_CLASS

  # From 0 to 18800
  loopStartTest = 0
  loopEndTest = MAX_CLASSES*MAX_TEST_INPUTS_PER_CLASS

else: # Default mode is digits
  loopStart = 0
  loopEnd = 10*MAX_INPUTS_PER_CLASS

  # From 0 to 4000
  loopStartTest = 0
  loopEndTest = 10*MAX_TEST_INPUTS_PER_CLASS

for i in range(loopStart,loopEnd,MAX_INPUTS_PER_CLASS):
    for j in range(chosen_inputs_per_class):
        inputs.append(train_inputs.iloc[i+j][:]/INPUTS_MAX_VALUE) # Append normalised value
        labels.append(train_labels.iloc[i])

for i in range(loopStartTest,loopEndTest,MAX_TEST_INPUTS_PER_CLASS):
    for j in range(chosen_test_inputs_per_class):
        testInputs.append(test_inputs.iloc[i+j][:]/INPUTS_MAX_VALUE) # Normalised
        testLabels.append(test_labels.iloc[i])

# Convert to NumPy Arrays
```

```
labels = np.array(labels)
inputs = np.array(inputs)
# drawnInput = np.array(drawn_input/12) # 336 / 28 = 12

testLabels = np.array(testLabels)
testInputs = np.array(testInputs)

class_colours = np.array(class_colours)

if args.debug:
  if (inputs.max()==1 and inputs.min()==0):
    trainNormaliseCheck = True
  else:
    trainNormaliseCheck = False

  if (testInputs.max()==1 and testInputs.min()==0):
    testNormaliseCheck = True
  else:
    testNormaliseCheck = False

  print('Train labels:',labels.shape)
  print('Train inputs:', inputs.shape)
  print('Test labels:',testLabels.shape)
  print('Test inputs:', testInputs.shape)
  print('Colours:', class_colours.shape)
  print('Training data normalised:', trainNormaliseCheck)
  print('Testing data normalised:', testNormaliseCheck)

# Variables
n = inputs.shape[0]
m = inputs.shape[1]

n_test = testInputs.shape[0]
m_test = testInputs.shape[1]

network_dimensions = np.array([n_classes*2,n_classes*2])

n_iterations = n
n_iterations_test = n_test

if args.debug:
  print('n_classes:', n_classes)
  print('n:', n)
  print('m:', m)
  print('n_test:', n_test)
  print('m_test:', m_test)
  print('Network dimensions:', network_dimensions.shape)
  print('Number of training iterations:', n_iterations)
  print('Number of testing iterations:', n_iterations_test)
  print('Initial learning rate:', init_learning_rate)
  print('Inputs per class:', chosen_inputs_per_class)

# Variables

# Weight Matrix - same for training and testing as same number of
    classes and therefore network dimensions
net = np.random.random((network_dimensions[0], network_dimensions[1], m)
    )

# Initial Radius (sigma) for the neighbourhood - same for tranining and
    testing as same network dimensions
init_radius = max(network_dimensions[0], network_dimensions[1]) / 2

```

```
# Radius decay parameter - different as (possibly) different number of iterations
time_constant = n_iterations / np.log(init_radius)
time_constant_test = n_iterations_test / np.log(init_radius)
# time_constant_drawn = drawnInput.shape[0] / np.log(init_radius)

if args.debug:
  print('Net', type(net))
  print('Initial Radius', init_radius)
  print('Time constant', time_constant)
  print('Time constant test', time_constant_test)

#------------------------------------------------------------------------------
# METHODS
#------------------------------------------------------------------------------

# Saving files with timestamp
def timeStamped(fmt='%Y-%m-%d-%H-%M-%S'):
  return datetime.datetime.now().strftime(fmt)

# View on Matplotlib
#def display(n_cols, n_rows, x):
#
# fig, ax = plt.subplots(n_rows, n_cols, sharex='col', sharey='row')
#
# if args.debug:
#   for i in range(n_rows):
#       for j in range(n_cols):
#           pic = np.rot90((np.fliplr(inputs[x,:].reshape((28,28)))))
#           ax[i, j].imshow(pic, cmap='gray')
#           ax[i, j].axis('off')
#           x+=1
#   plt.show()

#if args.debug:
# display(5,5,0)


# Find Best Matching Unit (BMU)
def findBMU(t, net, m):

  # A 1D array which will contain the X,Y coordinates
  # of the BMU for the given input vector t
  bmu_idx = np.array([0,0])

  # Set the initial minimum difference
  min_diff = np.iinfo(np.int).max

  # To compute the high-dimension distance between
  # the given input vector and each neuron,
  # we calculate the difference between the vectors
  for x in range (net.shape[0]):
    for y in range(net.shape[1]):
      w = net[x,y,:].reshape(m, 1)

      # Don't sqrt to avoid heavy operation
      diff = np.sum((w - t) ** 2)

      if (diff < min_diff):
        min_diff = diff
        bmu_idx = np.array([x, y])

  bmu = net[bmu_idx[0], bmu_idx[1], :].reshape(m, 1)
```

```python

  return(bmu, bmu_idx, min_diff)

# Decay the neighbourhood radius with time
def decayRadius(initial_radius, i, time_constant):
  return initial_radius * np.exp(-i / time_constant)

# Decay the learning rate with time
def decayLearningRate(initial_learning_rate, i, n_iterations):
  return initial_learning_rate * np.exp(-i / n_iterations)

# Calculate the influence
def getInfluence(distance, radius):
  return np.exp(-distance / (2* (radius**2)))


# SOM Step Learning
def trainSOM(inputsValues, times, timeCTE):

  bmu_idx_arr = []
  radiusList = []
  learnRateList = []
  sqDistList = []

  for i in range (times):

    if args.debug:
      print(str(int(i/times * 100)) + '%') # Progress percentage

    # --------------- INPUT ---------------
    # 1. Select a input weight vector at each step

    # This can be random, however since we're using sorted inputs, we're
    # proceeding in a linear manner through all nodes for sake of clarity
    t = inputsValues[i, :].reshape(np.array([m, 1]))

    # --------------- BMU ---------------
    # 2. Find the chosen input vector's BMU at each step
    #bmu, bmu_idx = findBMU(t, net, m)
    bmu, bmu_idx, dist = findBMU(t, net, m)

    bmu_idx_arr.append(bmu_idx)
    sqDistList.append(dist)

    # --------------- DECAY ---------------
    # 3. Determine topological neighbourhood for each step
    r = decayRadius(init_radius, i, timeCTE)
    l = decayLearningRate(init_learning_rate, i, times)

    radiusList.append(r)
    learnRateList.append(l)

    # --------------- UPDATE ---------------
    # 4. Repeat for all nodes in the *BMU neighbourhood*
    for x in range(net.shape[0]):
      for y in range(net.shape[1]):

        # Find weight vector
        w = net[x, y, :].reshape(m, 1)
        #wList.append(w)

        # Get the 2-D distance (not Euclidean as no sqrt)
```

```
        w_dist = np.sum((np.array([x, y]) - bmu_idx) ** 2)
        #wDistList.append(w_dist)

        # If the distance is within the current neighbourhood radius
        if w_dist <= r**2:

          # Calculate the degree of influence (based on the 2-D distance)
          influence = getInfluence(w_dist, r)

          # Update weight:
          # new w = old w + (learning rate * influence * delta)
          # delta = input vector t - old w
          new_w = w + (l * influence * (t - w))
          #new_wList.append(new_w)

          # Update net with new weight
          net[x, y, :] = new_w.reshape(1, m)

    # Every 100 iterations we call for a SOM to be made to view
    #if (i>0 and i%100==0):
    # bmu_interim_arr = np.array(bmu_idx_arr)
    # makeSOM(bmu_interim_arr, labels, [], [])

  # Convert to NumPy array
  bmu_idx_arr = np.array(bmu_idx_arr)

  np.savetxt((save_path+'%s'%timeStamped()+'_%s'%n_classes+'classes'+'_%s'%init_learning_rate+'rate'+'_%s'%chosen_inputs_per_class+'inputs'+'.csv'), bmu_idx_arr, fmt='%d', delimiter=',')
  #np.savetxt((save_path+'Net_%s'%timeStamped()+'.txt'), net, fmt='%d')

  return(bmu_idx_arr, radiusList, learnRateList, sqDistList)

def makeSOM(bmu_idx_arr, labels, bmu_idx_arr_test, testLabels): #, bmuDrawn):

  # Declare
  x_coords = []
  y_coords = []

  x_coordsTest = []
  y_coordsTest = []

  # Fill
  x_coords = np.random.randint(0, n_classes*2, chosen_inputs_per_class*n_classes)
  y_coords = np.random.randint(0, n_classes*2, chosen_inputs_per_class*n_classes)

  x_coordsTest = np.random.randint(0, n_classes*2, chosen_test_inputs_per_class*n_classes)
  y_coordsTest = np.random.randint(0, n_classes*2, chosen_test_inputs_per_class*n_classes)

  # Convert
  x_coords = np.array(x_coords)
  y_coords = np.array(y_coords)

  x_coordsTest = np.array(x_coordsTest)
  y_coordsTest = np.array(y_coordsTest)

  if (args.type=='d'):
```

```
    labelColorLen = n_classes
  else:
    labelColorLen = MAX_CLASSES

  # plotVector Format: [X, Y, R, G, B]
  # Coordinates and colours in a single vector

  labelColor = np.zeros((labelColorLen,3))
  plotVector = np.zeros((n,5))

  labelColor_test = np.zeros((labelColorLen,3))
  plotVectorTest = np.zeros((n_test,5))

  # Insert training values
  for i in range(n):
        # Color classes
    labelColor[labels[i,0]-1][0] = class_colours[labels[i,0]-1][0]
    labelColor[labels[i,0]-1][1] = class_colours[labels[i,0]-1][1]
    labelColor[labels[i,0]-1][2] = class_colours[labels[i,0]-1][2]

    # X, Ys - Coordinates with added noise
    plotVector[i][0] = bmu_idx_arr[i][0]
    plotVector[i][1] = bmu_idx_arr[i][1]

    # R,G,Bs - Color each point according to class
    plotVector[i][2] = labelColor[labels[i,0]-1][0]
    plotVector[i][3] = labelColor[labels[i,0]-1][1]
    plotVector[i][4] = labelColor[labels[i,0]-1][2]

  # Insert testing values
  for i in range(n_test):
    # Color classes
    labelColor_test[testLabels[i,0]-1][0] = class_colours[testLabels[i
    ,0]-1][0]
    labelColor_test[testLabels[i,0]-1][1] = class_colours[testLabels[i
    ,0]-1][1]
    labelColor_test[testLabels[i,0]-1][2] = class_colours[testLabels[i
    ,0]-1][2]

    # X, Ys - Coordinates with added noise
    plotVectorTest[i][0] = bmu_idx_arr_test[i][0]
    plotVectorTest[i][1] = bmu_idx_arr_test[i][1]

    # R,G,Bs - Color each point according to class
    plotVectorTest[i][2] = labelColor_test[testLabels[i,0]-1][0]
    plotVectorTest[i][3] = labelColor_test[testLabels[i,0]-1][1]
    plotVectorTest[i][4] = labelColor_test[testLabels[i,0]-1][2]

  # Generate noise for each point
  if (plotVector.shape[0] > 0):
    a_x = -0.4
    a_y = -0.4
    b_x = 0.4
    b_y = 0.4

    noise_x = (b_x-a_x) * np.random.rand(plotVector.shape[0], 1) + a_x
    noise_y = (b_y-a_y) * np.random.rand(plotVector.shape[0], 1) + a_y

    noise_x_test = (b_x-a_x) * np.random.rand(plotVectorTest.shape[0],
    1) + a_x
    noise_y_test = (b_y-a_y) * np.random.rand(plotVectorTest.shape[0],
    1) + a_y

```

```
# Convert zPlot first as there are no noise values for RGB
zPlot = np.array(plotVector[:,2:5])
zPlot_test = np.array(plotVectorTest[:,2:5])

# With noise
xPlotNoise = np.add(plotVector[:,0], noise_x[:,0])
yPlotNoise = np.add(plotVector[:,1], noise_y[:,0])

xPlotTestNoise = np.add(plotVectorTest[:,0], noise_x_test[:,0])
yPlotTestNoise = np.add(plotVectorTest[:,1], noise_y_test[:,0])

x_coordsNoise = np.add(x_coords[:], noise_x[:,0])
y_coordsNoise = np.add(y_coords[:], noise_y[:,0])

x_coordsTestNoise = np.add(x_coordsTest[:], noise_x_test[:,0])
y_coordsTestNoise = np.add(y_coordsTest[:], noise_y_test[:,0])

# Witout noise
xPlot = plotVector[:,0]
yPlot = plotVector[:,1]

xPlotTest = plotVectorTest[:,0]
yPlotTest = plotVectorTest[:,1]

# Below values don't change but are here just to show the 4 total batches
# x_coords = x_coords
# y_coords = y_coords

# x_coordsTest = x_coordsTest
# y_coordsTest = y_coordsTest

if (args.debug):
  print('Train Inputs per class:',args.inputsTrain)
  print('Test Inputs per class:',args.inputsTest)
  print('Rate:',args.rate)
  print('Type:',args.type)
  print('')
  print('x:',xPlot.shape)
  print('y:',yPlot.shape)
  print('z:',zPlot.shape)
  print('BMUs:',bmu_idx_arr.shape)
  #print(labelColor)
  print('')
  print('x test noise:',xPlotTestNoise.shape)
  print('y test noise:',yPlotTestNoise.shape)
  print('BMUs_test:',bmu_idx_arr_test.shape)
  print('')
  print('x_test:',xPlotTest.shape)
  print('y_test:',yPlotTest.shape)
  print('z_test:',zPlot_test.shape)
  print('')
  #print('BMU drawn:',bmuDrawn.shape)
  #print(labelColor_test)

# Plot Scatterplot
#plotSize = (n_classes * 2)
#figSize = 5.91
#plt.figure(figsize=(figSize, figSize))

#-------------------------------------------------------------------------
# Legend
#-------------------------------------------------------------------------
```

```

if (args.type == 'd'): # Digits
  plotLegend = 10
elif (args.type == 'l'): # Letters
  plotLegend = MAX_CLASSES-10
elif (args.type == 'c'): # Combined
  plotLegend = MAX_CLASSES

for i in range(plotLegend):
  plt.title('Legend of each class')
  plt.scatter(i, 1, s=100, facecolor=labelColor[i], edgecolor=labelColor[i])

plt.yticks([])
plt.show()

#---------------------------------------------------------------------
# Random train nodes
#---------------------------------------------------------------------

# Plot train random nodes without noise
plt.scatter(x_coords, y_coords, s=20, marker='o', facecolor=zPlot)
plt.title(str(n)+' train inputs unsorted without noise')
plt.show()

# Plot train random nodes with noise
plt.scatter(x_coordsNoise, y_coordsNoise, s=20, marker='o', facecolor=zPlot)
plt.title(str(n)+' train inputs unsorted with noise')
plt.show()

#---------------------------------------------------------------------
# Random test nodes
#---------------------------------------------------------------------

# Plot test random nodes without noise
plt.scatter(x_coordsTest, y_coordsTest, s=20, marker='x', facecolor=zPlot_test)
plt.title(str(n_test)+' test inputs unsorted without noise')
plt.show()

# Plot test random nodes with noise
plt.scatter(x_coordsTestNoise, y_coordsTestNoise, s=20, marker='x', facecolor=zPlot_test)
plt.title(str(n_test)+' test inputs unsorted with noise')
plt.show()

#---------------------------------------------------------------------
# Random train and test nodes
#---------------------------------------------------------------------

# Plot train and test random nodes without noise
plt.scatter(x_coords, y_coords, s=20, marker='o', facecolor=zPlot)
plt.scatter(x_coordsTest, y_coordsTest, s=20, marker='x', facecolor=zPlot)
plt.title(str(n)+' train and test inputs unsorted without noise')
plt.show()

# Plot train and test random nodes with noise
plt.scatter(x_coordsNoise, y_coordsNoise, s=20, marker='o', facecolor=zPlot)
plt.scatter(x_coordsTestNoise, y_coordsTestNoise, s=20, marker='x', facecolor=zPlot)
```

```
plt.title(str(n+n_test)+' train and test inputs unsorted with noise')
plt.show()

#-------------------------------------------------------------------
# Train data
#-------------------------------------------------------------------

# Plot train data without noise
plt.scatter(xPlot, yPlot, s=20, marker='o', facecolor=zPlot)
plt.title(str(n)+' train inputs sorted without noise')
plt.show()

# Plot train data with noise
plt.scatter(xPlotNoise, yPlotNoise, s=20, marker='o', facecolor=zPlot)
plt.title(str(n)+' train inputs sorted with noise')
plt.show()

#-------------------------------------------------------------------
# Test data
#-------------------------------------------------------------------

# Plot test data without noise
plt.scatter(xPlotTest, yPlotTest, s=20, marker='x', facecolor=
  zPlot_test)
plt.title(str(n_test)+' test inputs sorted without noise')
plt.show()

# Plot test data with noise
plt.scatter(xPlotTestNoise, yPlotTestNoise, s=20, marker='x',
  facecolor=zPlot_test)
plt.title(str(n)+' test inputs sorted with noise')
plt.show()

#-------------------------------------------------------------------
# Train and Test data
#-------------------------------------------------------------------

# Plot both train and test data without noise
plt.scatter(xPlot, yPlot, s=20, marker='o', facecolor=zPlot)
plt.scatter(xPlotTest, yPlotTest, s=20, marker='x', facecolor=
  zPlot_test)
plt.title(str(n+n_test)+' train and test inputs sorted without noise')
plt.show()


# Plot both train and test data with noise
plt.scatter(xPlotNoise, yPlotNoise, s=20, marker='o', facecolor=zPlot)
plt.scatter(xPlotTestNoise, yPlotTestNoise, s=20, marker='x',
  facecolor=zPlot_test)
#plt.scatter(bmuDrawn[0][0], bmuDrawn[0][0], marker='+', s=200,
  facecolor='black')
plt.title(str(n)+' train and test inputs sorted with noise')
plt.show()

#-------------------------------------------------------------------
# View all plots together
#-------------------------------------------------------------------

#fig, ax = plt.subplots(2, 5, sharex='col', sharey='row')

#for i in range(2):
    #for j in range(5):
        #pic = np.rot90((np.fliplr(inputs[x,:].reshape((28,28)))))
```

```
            #ax[i, j].imshow(pic, cmap='gray')
            #ax[i, j].axis('off')
            #x+=1
    #plt.show()

    #plt.legend(handles=[n])
    #plt.xlim(-1, plotSize)
    #plt.ylim(-1, plotSize)
    #plt.axis('off')
    #plt.title('Train: ' + str(args.inputsTrain*n_classes) + ', Test: ' +
      str(args.inputsTest*n_classes))
    #plt.show()

    #--------------------------------------------------------------------------
    # Save all plots as .CSVs
    #--------------------------------------------------------------------------

    # Declare
    randTrain = np.zeros((n,6))
    randTest = np.zeros((n_test,6))
    randCombined = np.zeros((n+n_test,6))

    randTrainNoise = np.zeros((n,6))
    randTestNoise = np.zeros((n_test,6))
    randCombinedNoise = np.zeros((n+n_test,6))

    Train = np.zeros((n,6))
    Test = np.zeros((n_test,6))
    combined = np.zeros((n+n_test,6))

    TrainNoise = np.zeros((n,6))
    TestNoise = np.zeros((n_test,6))
    combinedNoise = np.zeros((n+n_test,6))

    # Convert for D3
    fullRGB = zPlot*255
    fullRGB_test = zPlot_test * 255
    print('fullRGB shape', fullRGB.shape)
    print('fullRGB_test shape', fullRGB_test.shape)

    # Fill by column
    # Nodes without noise
    randTrain[:,0] = x_coords
    randTrain[:,1] = y_coords
    randTrain[:,2:5] = fullRGB
    randTrain[:,5:6] = labels-1

    randTest[:,0] = x_coordsTest
    randTest[:,1] = y_coordsTest
    randTest[:,2:5] = fullRGB_test
    randTest[:,5:6] = testLabels

    randCombined[:,0] = np.concatenate((x_coords,x_coordsTest))
    randCombined[:,1] = np.concatenate((y_coords,y_coordsTest))
    randCombined[:,2:5] = np.concatenate((fullRGB,fullRGB_test))
    randCombined[:,5:6] = np.concatenate((labels-1,testLabels))

    # Nodes with noise
    randTrainNoise[:,0] = x_coordsNoise
    randTrainNoise[:,1] = y_coordsNoise
    randTrainNoise[:,2:5] = fullRGB
    randTrainNoise[:,5:6] = labels-1

```

```
randTestNoise[:,0] = x_coordsTestNoise
randTestNoise[:,1] = y_coordsTestNoise
randTestNoise[:,2:5] = fullRGB_test
randTestNoise[:,5:6] = testLabels

randCombinedNoise[:,0] = np.concatenate((x_coordsNoise,
  x_coordsTestNoise))
randCombinedNoise[:,1] = np.concatenate((y_coordsNoise,
  y_coordsTestNoise))
randCombinedNoise[:,2:5] = np.concatenate((fullRGB,fullRGB_test))
randCombinedNoise[:,5:6] = np.concatenate((labels-1,testLabels))

# Data without noise
Train[:,0] = xPlot
Train[:,1] = yPlot
Train[:,2:5] = fullRGB
Train[:,5:6] = labels-1

Test[:,0] = xPlotTest
Test[:,1] = yPlotTest
Test[:,2:5] = fullRGB_test
Test[:,5:6] = testLabels

combined[:,0] = np.concatenate((xPlot,xPlotTest))
combined[:,1] = np.concatenate((yPlot,yPlotTest))
combined[:,2:5] = np.concatenate((fullRGB,fullRGB_test))
combined[:,5:6] = np.concatenate((labels-1,testLabels))

# Data with noise
TrainNoise[:,0] = xPlotNoise
TrainNoise[:,1] = yPlotNoise
TrainNoise[:,2:5] = fullRGB
TrainNoise[:,5:6] = labels-1

TestNoise[:,0] = xPlotTestNoise
TestNoise[:,1] = yPlotTestNoise
TestNoise[:,2:5] = fullRGB_test
TestNoise[:,5:6] = testLabels

combinedNoise[:,0] = np.concatenate((xPlotNoise,xPlotTestNoise))
combinedNoise[:,1] = np.concatenate((yPlotNoise,yPlotTestNoise))
combinedNoise[:,2:5] = np.concatenate((fullRGB,fullRGB_test))
combinedNoise[:,5:6] = np.concatenate((labels-1,testLabels))

# Export
np.savetxt(('static/data/OCR/RandTrain.csv'), randTrain, fmt='%.3f',
  delimiter=',', comments='', header='xSOM,ySOM,R,G,B,label')
np.savetxt(('static/data/OCR/RandTest.csv'), randTest, fmt='%.3f',
  delimiter=',', comments='', header='xSOM,ySOM,R,G,B,label')
np.savetxt(('static/data/OCR/RandCombined.csv'), randCombined, fmt='
  %.3f', delimiter=',', comments='', header='xSOM,ySOM,R,G,B,label')

np.savetxt(('static/data/OCR/RandTrainNoise.csv'), randTrainNoise ,
  fmt='%.3f', delimiter=',', comments='', header='xSOM,ySOM,R,G,B,label
  ')
np.savetxt(('static/data/OCR/RandTestNoise.csv'), randTestNoise , fmt=
  '%.3f', delimiter=',', comments='', header='xSOM,ySOM,R,G,B,label')
np.savetxt(('static/data/OCR/randCombinedNoise.csv'),
  randCombinedNoise , fmt='%.3f', delimiter=',', comments='', header='
  xSOM,ySOM,R,G,B,label')

np.savetxt(('static/data/OCR/Train.csv'), Train , fmt='%.3f',
  delimiter=',', comments='', header='xSOM,ySOM,R,G,B,label')
```

```
    np.savetxt(('static/data/OCR/Test.csv'), Test , fmt='%.3f', delimiter=',', comments='', header='xSOM,ySOM,R,G,B,label')
    np.savetxt(('static/data/OCR/Combined.csv'), combined , fmt='%.3f', delimiter=',', comments='', header='xSOM,ySOM,R,G,B,label')

    np.savetxt(('static/data/OCR/TrainNoise.csv'), TrainNoise , fmt='%.3f', delimiter=',', comments='', header='xSOM,ySOM,R,G,B,label')
    np.savetxt(('static/data/OCR/TestNoise.csv'), TestNoise , fmt='%.3f', delimiter=',', comments='', header='xSOM,ySOM,R,G,B,label')
    np.savetxt(('static/data/OCR/CombinedNoise.csv'), combinedNoise , fmt='%.3f', delimiter=',', comments='', header='xSOM,ySOM,R,G,B,label')

    #np.savetxt(('static/data/OCR/TrainCoordinates.csv'), exportTrain, fmt='%.3f', delimiter=',', comments='', header='xSOM,ySOM,R,G,B')
    #np.savetxt(('static/data/OCR/TestCoordinates.csv'), exportTest, fmt='%.3f', delimiter=',', comments='', header='xSOM,ySOM,R,G,B')
    np.savetxt(('static/data/OCR/Labels.txt'), labels, fmt='%d', comments='', header='Labels')
    np.savetxt(('static/data/OCR/TestLabels.txt'), testLabels, fmt='%d', comments='', header='testLabels')

    #if args.debug:
    #    print('Saved train coordinates with noise')

# Make graphical comparaisons of various parameters
def plotVariables(radiusTrain, radiusTest, learnRateTrain, learnRateTest, sqDistTrain, sqDistTest): #, radiusDrawn, rateDrawn, sqDistDrawn):

    # Plot radius
    plt.title('Radius evolution')
    plt.xlabel('Number of iterations')
    plt.ylabel('Radius size')
    plt.plot(radiusTrain, 'r', label='Training Radius')
    plt.plot(radiusTest, 'b', label='Testing Radius')
    #plt.plot(radiusDrawn, 'g')
    plt.legend(loc=1)
    plt.show()

    # Plot learning rate
    plt.title('Learning rate evolution')
    plt.xlabel('Number of iterations')
    plt.ylabel('Learning rate')
    plt.plot(learnRateTrain, 'r', label='Training Learning Rate')
    plt.plot(learnRateTest, 'b', label='Testing Learning Rate')
    #plt.plot(rateDrawn, 'g')
    plt.legend(loc=1)
    plt.show()

    # Plot 3D distance
    plt.title('Best Matching Unit 3D Distance')
    plt.xlabel('Number of iterations')
    plt.ylabel('Smallest Distance Squared')
    plt.plot(sqDistTrain, 'r', label='Training (Squared) Distance')
    plt.plot(sqDistTest, 'b', label='Testing (Squared) Distance')
    #plt.plot(sqDistDrawn, 'g')
    plt.legend(loc=1)

    # We have to even out the iteration steps for the graphs to be comparable
    #step = int(chosen_inputs_per_class/chosen_test_inputs_per_class)

    #y = 0
    #for x in range(0, len(sqDistTrain), step):
```

```
        #plt.plot(x, testArr[y],'b')
        #print(testArr[y])
        #y = y+1

    plt.show()

#------------------------------------------------------------------------
# MAIN METHOD CALLS
#------------------------------------------------------------------------

bmuTrain, radiusTrain, rateTrain, sqDistTrain = trainSOM(inputs,
    n_iterations, time_constant)
bmuTest, radiusTest, rateTest, sqDistTest = trainSOM(testInputs,
    n_iterations_test, time_constant_test)
# bmuDrawn, radiusDrawn, rateDrawn, sqDistDrawn = trainSOM(drawnInput,
    drawnInput.shape[0], time_constant_drawn)

makeSOM(bmuTrain, labels, bmuTest, testLabels) #, bmuDrawn)
plotVariables(radiusTrain, radiusTest, rateTrain, rateTest, sqDistTrain,
     sqDistTest) #, radiusDrawn, rateDrawn, sqDistDrawn)
```

Listing A.4: EMNIST SOM code

A.5 app.py

```
from flask import Flask
from flask import render_template
from flask import request
from flask import jsonify
#import som
#import RGB

app = Flask(__name__)

@app.route("/")
def index():
    return render_template('index.html')

@app.route('/1')
def one():
    return render_template('1.html')

@app.route('/cards1')
def cards1():
    return render_template('cards1.html')

@app.route('/cards2')
def cards2():
    return render_template('cards2.html')

@app.route('/cards3')
def cards3():
    return render_template('cards3.html')

@app.route('/1_3')
def oneThree():
    return render_template('1_3.html')

@app.route('/1_4')
def oneFour():
    return render_template('1_4.html')

@app.route('/1_5')
def oneFive():
    return render_template('1_5.html')

@app.route('/2')
def two():
    return render_template('2.html')

@app.route('/2_5')
def twoFive():
    return render_template('2_5.html')

@app.route('/3')
def three():
    return render_template('3.html')

@app.route('/canvas')
def canvas():
    return render_template('canvas.html')

@app.route('/canvaspost', methods=['GET', 'POST'])
def canvaspost():
    if request.method == 'GET':
```

```
        #return json.dumps({'success': True}), 200, {'ContentType': 'application/json'}
        csv = request.files['myJSON']
        return jsonify(
            summary=make_summary(csv),
            csv_name=secure_filename(csv.filename)
        )
    else:
        return "Not"

    return render_template("canvaspost.html")

@app.route('/dataset')
def dataset():
    return render_template('dataset.html')

@app.route('/about')
def about():
    return render_template('about.html')

if __name__ == "__main__":
    app.run(debug=True)
```

Listing A.5: Flask code

A.6 viewInput.py

```
# Name: Eklavya SARKAR,
# ID:201135564,
# Username: u5es2

# Sort the EMNIST Balanced 47 Classes (training or testing) data
# Sequence: digits (0-9), then capital letters (A-Z), then small letters
    (selected ones from a-z)

import argparse
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#--------------------------------------------------------------------------
# CONFIG
#--------------------------------------------------------------------------

# Argument Parser
parser = argparse.ArgumentParser(description='Sort the EMNIST data in
    order of their class')
parser.add_argument('-d','--debug', action='store_true', default=False,
    help='Print debug messages')
args = parser.parse_args()

#--------------------------------------------------------------------------
# SET UP
#--------------------------------------------------------------------------

# Read raw data
#data_path = '/Users/eklavya/Movies/EMNIST_csv/Balanced/Sorted/
    SortedTestInputs.csv'
data_url = 'http://cgi.csc.liv.ac.uk/~u5es2/EMNIST/Sorted/Train.csv'
data = pd.read_csv(data_url, encoding='utf-8', header=None)

labels_url = 'http://cgi.csc.liv.ac.uk/~u5es2/EMNIST/Sorted/TrainLabels.
    txt'
labels = pd.read_csv(labels_url, encoding='utf-8', header=None)

# Convert to NumPy arrays
inputs = np.array(data)
labels = np.array(labels)

if args.debug:
    print(inputs.shape)
    print(labels.shape)

#--------------------------------------------------------------------------
# GENERATE PLOTS
#--------------------------------------------------------------------------


def display(n_cols, n_rows, x):

    plt.figure(dpi=100)

    fig, ax = plt.subplots(n_rows, n_cols, sharex='col', sharey='row')

    for i in range(n_rows):
        for j in range(n_cols):
```

```
            label = labels[i]
            pic = np.rot90((np.fliplr(inputs[x,:].reshape((28,28)))))
            ax[i, j].imshow(pic, cmap='gray')
            ax[i, j].axis('off')
            x+=2400
    fig.savefig('static/images/dataset.png', bbox_inches='tight', transparent=True)

#------------------------------------------------------------------------------
# MAIN METHOD CALLS
#------------------------------------------------------------------------------
display(9,5,0)
```

Listing A.6: View input code

Appendix B

Data

B.1 Iris Dataset

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
5.1,3.7,1.5,0.4,Iris-setosa
4.6,3.6,1.0,0.2,Iris-setosa
5.1,3.3,1.7,0.5,Iris-setosa
4.8,3.4,1.9,0.2,Iris-setosa
5.0,3.0,1.6,0.2,Iris-setosa
5.0,3.4,1.6,0.4,Iris-setosa
5.2,3.5,1.5,0.2,Iris-setosa
5.2,3.4,1.4,0.2,Iris-setosa
4.7,3.2,1.6,0.2,Iris-setosa
4.8,3.1,1.6,0.2,Iris-setosa
5.4,3.4,1.5,0.4,Iris-setosa
5.2,4.1,1.5,0.1,Iris-setosa
5.5,4.2,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.0,3.2,1.2,0.2,Iris-setosa
5.5,3.5,1.3,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
4.4,3.0,1.3,0.2,Iris-setosa
5.1,3.4,1.5,0.2,Iris-setosa
5.0,3.5,1.3,0.3,Iris-setosa
4.5,2.3,1.3,0.3,Iris-setosa
4.4,3.2,1.3,0.2,Iris-setosa
5.0,3.5,1.6,0.6,Iris-setosa
5.1,3.8,1.9,0.4,Iris-setosa
4.8,3.0,1.4,0.3,Iris-setosa
```

```
5.1,3.8,1.6,0.2,Iris-setosa
4.6,3.2,1.4,0.2,Iris-setosa
5.3,3.7,1.5,0.2,Iris-setosa
5.0,3.3,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
5.7,2.8,4.5,1.3,Iris-versicolor
6.3,3.3,4.7,1.6,Iris-versicolor
4.9,2.4,3.3,1.0,Iris-versicolor
6.6,2.9,4.6,1.3,Iris-versicolor
5.2,2.7,3.9,1.4,Iris-versicolor
5.0,2.0,3.5,1.0,Iris-versicolor
5.9,3.0,4.2,1.5,Iris-versicolor
6.0,2.2,4.0,1.0,Iris-versicolor
6.1,2.9,4.7,1.4,Iris-versicolor
5.6,2.9,3.6,1.3,Iris-versicolor
6.7,3.1,4.4,1.4,Iris-versicolor
5.6,3.0,4.5,1.5,Iris-versicolor
5.8,2.7,4.1,1.0,Iris-versicolor
6.2,2.2,4.5,1.5,Iris-versicolor
5.6,2.5,3.9,1.1,Iris-versicolor
5.9,3.2,4.8,1.8,Iris-versicolor
6.1,2.8,4.0,1.3,Iris-versicolor
6.3,2.5,4.9,1.5,Iris-versicolor
6.1,2.8,4.7,1.2,Iris-versicolor
6.4,2.9,4.3,1.3,Iris-versicolor
6.6,3.0,4.4,1.4,Iris-versicolor
6.8,2.8,4.8,1.4,Iris-versicolor
6.7,3.0,5.0,1.7,Iris-versicolor
6.0,2.9,4.5,1.5,Iris-versicolor
5.7,2.6,3.5,1.0,Iris-versicolor
5.5,2.4,3.8,1.1,Iris-versicolor
5.5,2.4,3.7,1.0,Iris-versicolor
5.8,2.7,3.9,1.2,Iris-versicolor
6.0,2.7,5.1,1.6,Iris-versicolor
5.4,3.0,4.5,1.5,Iris-versicolor
6.0,3.4,4.5,1.6,Iris-versicolor
6.7,3.1,4.7,1.5,Iris-versicolor
6.3,2.3,4.4,1.3,Iris-versicolor
5.6,3.0,4.1,1.3,Iris-versicolor
5.5,2.5,4.0,1.3,Iris-versicolor
5.5,2.6,4.4,1.2,Iris-versicolor
6.1,3.0,4.6,1.4,Iris-versicolor
5.8,2.6,4.0,1.2,Iris-versicolor
5.0,2.3,3.3,1.0,Iris-versicolor
5.6,2.7,4.2,1.3,Iris-versicolor
5.7,3.0,4.2,1.2,Iris-versicolor
5.7,2.9,4.2,1.3,Iris-versicolor
6.2,2.9,4.3,1.3,Iris-versicolor
5.1,2.5,3.0,1.1,Iris-versicolor
5.7,2.8,4.1,1.3,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
7.1,3.0,5.9,2.1,Iris-virginica
6.3,2.9,5.6,1.8,Iris-virginica
6.5,3.0,5.8,2.2,Iris-virginica
7.6,3.0,6.6,2.1,Iris-virginica
4.9,2.5,4.5,1.7,Iris-virginica
7.3,2.9,6.3,1.8,Iris-virginica
6.7,2.5,5.8,1.8,Iris-virginica
```

```
7.2,3.6,6.1,2.5,Iris-virginica
6.5,3.2,5.1,2.0,Iris-virginica
6.4,2.7,5.3,1.9,Iris-virginica
6.8,3.0,5.5,2.1,Iris-virginica
5.7,2.5,5.0,2.0,Iris-virginica
5.8,2.8,5.1,2.4,Iris-virginica
6.4,3.2,5.3,2.3,Iris-virginica
6.5,3.0,5.5,1.8,Iris-virginica
7.7,3.8,6.7,2.2,Iris-virginica
7.7,2.6,6.9,2.3,Iris-virginica
6.0,2.2,5.0,1.5,Iris-virginica
6.9,3.2,5.7,2.3,Iris-virginica
5.6,2.8,4.9,2.0,Iris-virginica
7.7,2.8,6.7,2.0,Iris-virginica
6.3,2.7,4.9,1.8,Iris-virginica
6.7,3.3,5.7,2.1,Iris-virginica
7.2,3.2,6.0,1.8,Iris-virginica
6.2,2.8,4.8,1.8,Iris-virginica
6.1,3.0,4.9,1.8,Iris-virginica
6.4,2.8,5.6,2.1,Iris-virginica
7.2,3.0,5.8,1.6,Iris-virginica
7.4,2.8,6.1,1.9,Iris-virginica
7.9,3.8,6.4,2.0,Iris-virginica
6.4,2.8,5.6,2.2,Iris-virginica
6.3,2.8,5.1,1.5,Iris-virginica
6.1,2.6,5.6,1.4,Iris-virginica
7.7,3.0,6.1,2.3,Iris-virginica
6.3,3.4,5.6,2.4,Iris-virginica
6.4,3.1,5.5,1.8,Iris-virginica
6.0,3.0,4.8,1.8,Iris-virginica
6.9,3.1,5.4,2.1,Iris-virginica
6.7,3.1,5.6,2.4,Iris-virginica
6.9,3.1,5.1,2.3,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
6.8,3.2,5.9,2.3,Iris-virginica
6.7,3.3,5.7,2.5,Iris-virginica
6.7,3.0,5.2,2.3,Iris-virginica
6.3,2.5,5.0,1.9,Iris-virginica
6.5,3.0,5.2,2.0,Iris-virginica
6.2,3.4,5.4,2.3,Iris-virginica
5.9,3.0,5.1,1.8,Iris-virginica
```

LISTING B.1: Iris CSV source code

B.2 Colours Classes

```
0.976470588,0.921568627,0.917647059
0.752941176,0.223529412,0.168627451
0.980392157,0.858823529,0.847058824
0.690196078,0.227450980,0.180392157
0.607843137,0.349019608,0.713725490
0.733333333,0.560784314,0.807843137
0.831372549,0.901960784,0.945098039
0.921568627,0.960784314,0.984313725
0.819607843,0.949019608,0.921568627
0.066666667,0.470588235,0.392156863
0.086274510,0.627450980,0.521568627
0.831372549,0.937254902,0.874509804
0.117647059,0.517647059,0.286274510
0.094117647,0.415686275,0.231372549
0.490196078,0.400000000,0.031372549
```

```
0.992156863,0.949019608,0.913725490
0.901960784,0.494117647,0.133333333
0.898039216,0.596078431,0.400000000
0.992156863,0.996078431,0.996078431
0.592156863,0.603921569,0.603921569
0.650980392,0.674509804,0.686274510
0.666666667,0.717647059,0.721568627
0.800000000,0.819607843,0.819607843
0.921568627,0.929411765,0.937254902
0.156862745,0.215686275,0.278431373
0.670588235,0.698039216,0.725490196
0.090196078,0.125490196,0.164705882
0.203921569,0.596078431,0.858823529
0.160784314,0.501960784,0.725490196
0.423529412,0.203921569,0.513725490
0.317647059,0.180392157,0.372549020
0.921568627,0.870588235,0.941176471
0.482352941,0.141176471,0.109803922
0.364705882,0.427450980,0.494117647
0.439215686,0.482352941,0.486274510
0.372549020,0.415686275,0.415686275
0.956862745,0.964705882,0.964705882
0.898039216,0.905882353,0.913725490
0.941176471,0.952941176,0.956862745
0.627450980,0.250980392,0.000000000
0.470588235,0.258823529,0.070588235
0.960784314,0.690196078,0.254901961
0.956862745,0.815686275,0.247058824
0.670588235,0.921568627,0.776470588
0.321568627,0.745098039,0.501960784
0.635294118,0.850980392,0.807843137
0.203921569,0.596078431,0.858823529
```

LISTING B.2: The colour classes's source code, employed for the OCR's mixed digits and letters database

B.3 EMNIST Dataset

Can be accessed on `http://cgi.csc.liv.ac.uk/~u5es2/EMNIST/`.

Appendix C

Art

C.1 Nets

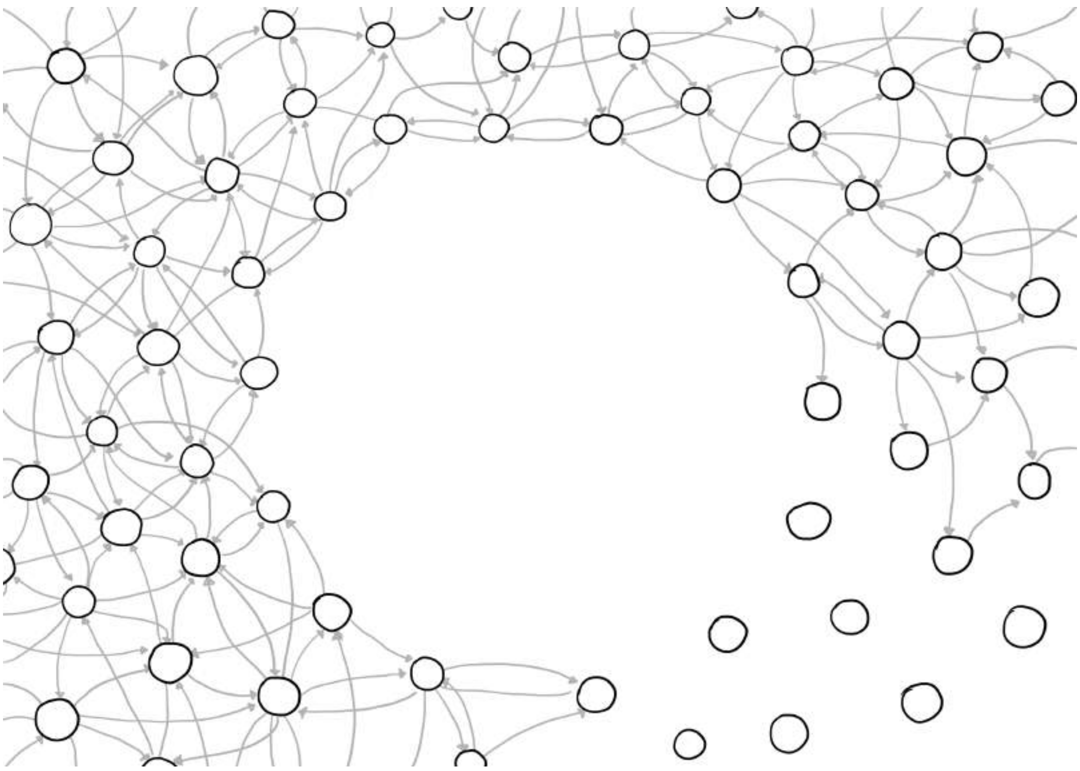


FIGURE C.1: Incomplete prototype

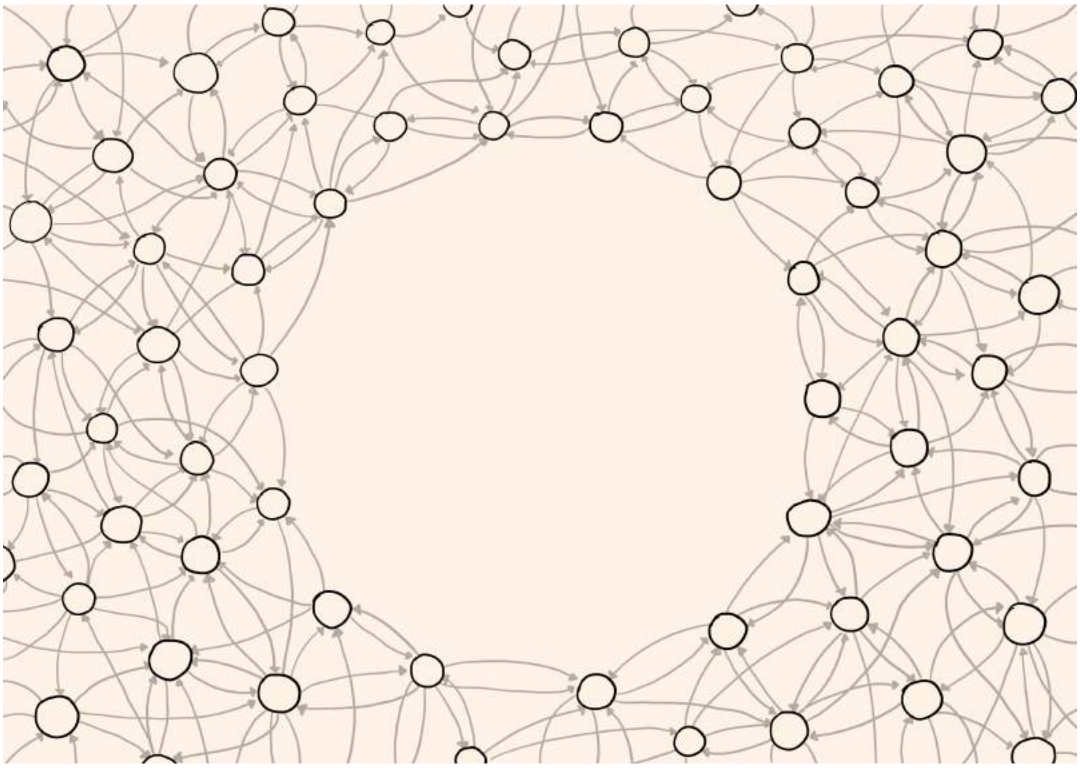


Figure C.2: Complete prototype

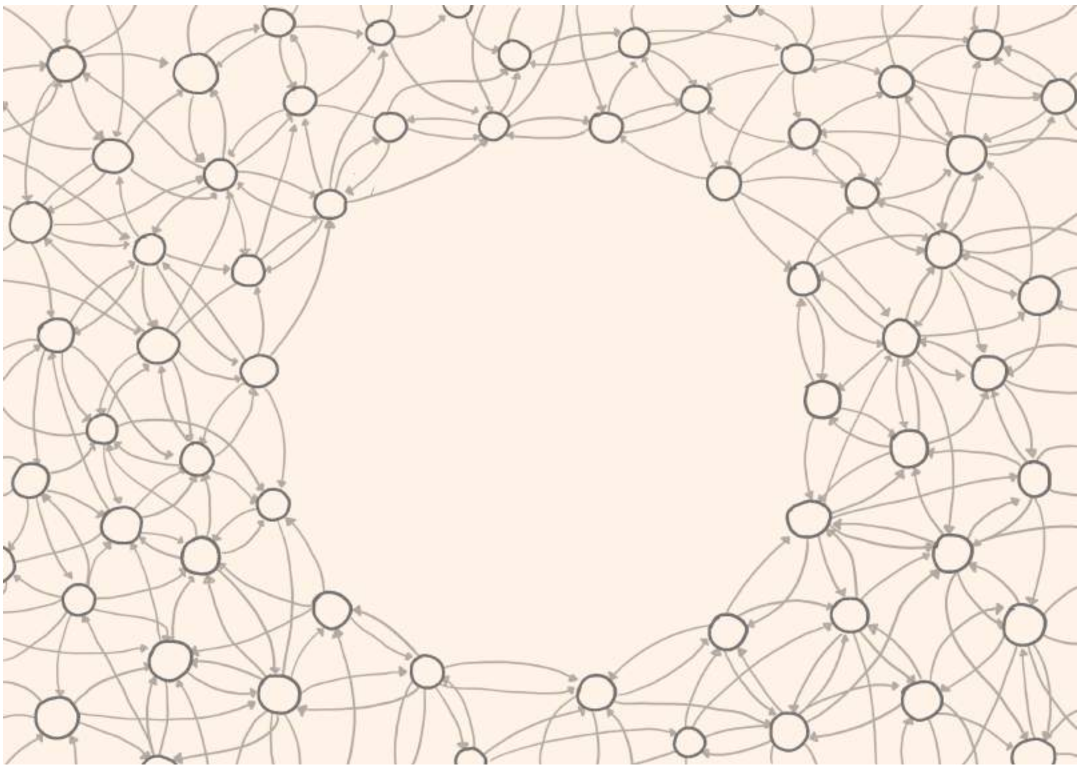


Figure C.3: Final design

C.2 Volume

FIGURE C.4: Shadow volume buttons

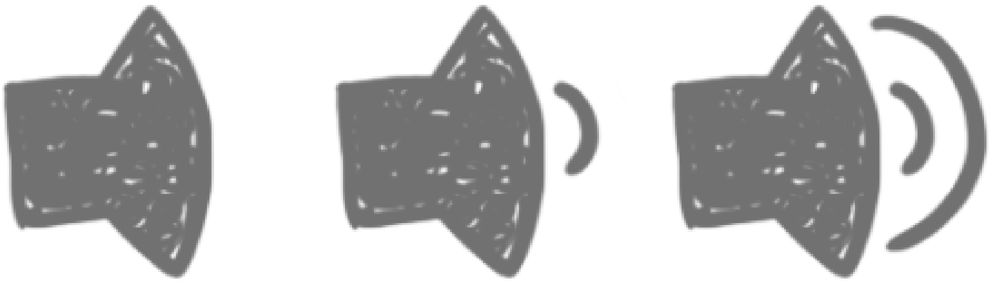


FIGURE C.5: Fill volume buttons

FIGURE C.6: Dash volume buttons

C.3 Cards

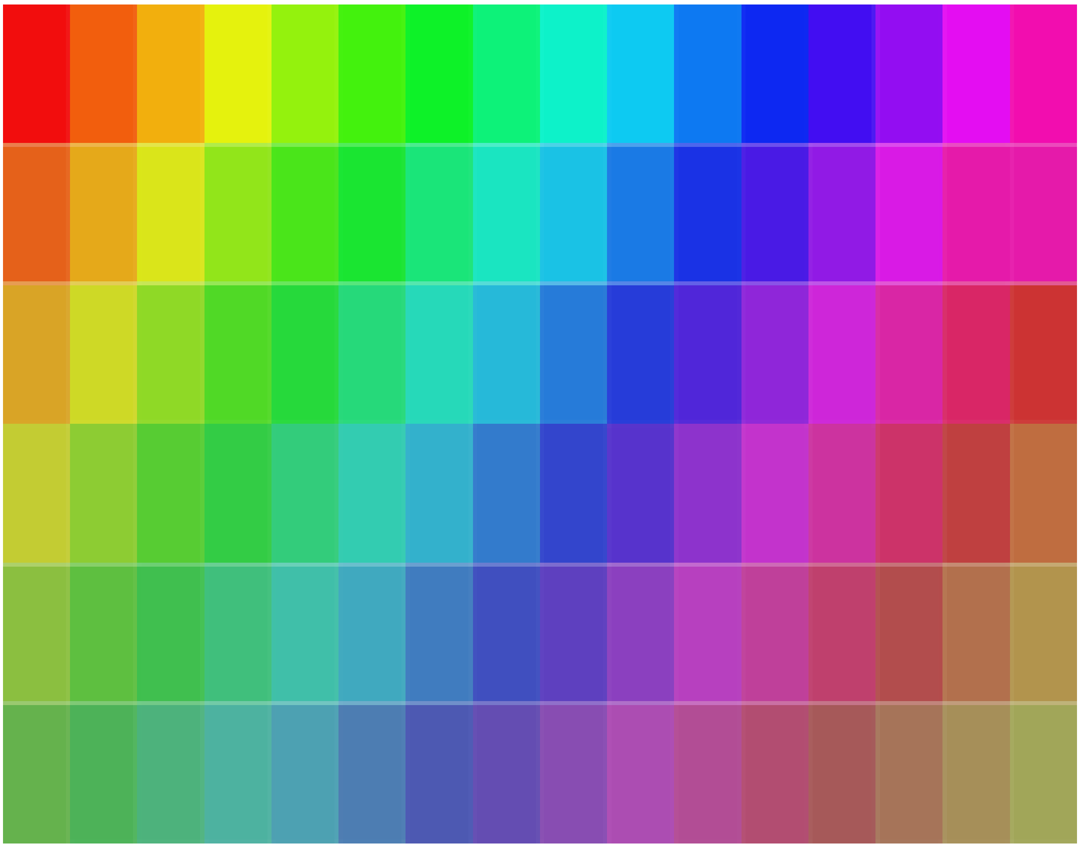


Figure C.7: RGB SOM designed for card

Appendix D

User Manual

D.1 Requirements

To execute the attached scripts, Python 3 is required as a framework.

D.2 Installation

If Python 3 is not already installed, it can be done via brew (which itself can be installed with the command given below).

Install Brew:

```
$/usr/bin/ruby-e"$(curl-fsSLhttps://raw.githubusercontent.com/Homebrew/install/
master/install)"
```

Use Brew to install Python 3:

```
$python3installpip3
```

The `pip3` package manager is recommended in order to install `Flask` or any other `Python3` package. To do install, following the steps below, given in a unix shell context.

```
$pip3installFlask
```

To run this software, the following libraries are required, and can be installed using pip3:

```
$pip3installpandas
$pip3installnumpy
$pip3installmatplotlib
```

The following used libraries are natively pre-installed in Python, but are nonetheless listed below:

- argsparse
- sys
- datetime

Virtual Environments

If necessary, virtual environments can be used to keep the libraries installed for the entire working machine seperate from those simply required for a specific task. This ensures that the libraries for this project don't get change or mix up with the development PC's native Python installation.

Navigate to `~\myPath\EMNIST-Kohonen-SOM\`

```
$pip3installvirtualenv
$cdmyPath
$virtualenvmyFolder
$sourcemyFolder/bin/activate
$pip3installmyPackages
$deactivate
```

Running Flask

Finally the proejct can be running by executing `app.py` on the terminal:

```
$python3app.py
*Runningonhttp://127.0.0.1:5000/(PressCTRL+Ctoquit)
```

And on a browser simply navigate to: `http://127.0.0.1:5000`. The website is now viewable.

Appendix E

Use-case descriptions

ID	**Use Case 1**
Name	Access site
Description	The user accesses the system either via a desktop or mobile web browser
Pre-condition	System is running
Event flow	1. Open Browser on device 2. Type in website's URL

ID	**Use Case 2**
Name	Choose Draw Mode
Description	The user chooses the draw mode option
Pre-condition	System is running
Event flow	1. Click on 'Draw' button

ID	**Use Case 3**
Name	Draw Letter
Description	The user draws a letter on the canvas
Pre-condition	System is running
Event flow	1. Use mouse on desktops, fingers on touchscreen devices 2. Click/touch and drag on canvas to draw 3. Draw an alphabet

ID	**Use Case 4**
Name	Submit Drawing
Description	The user submits their input drawing to the backend
Pre-condition	System is running
Event flow	1. Press the 'submit' button
Extension points	Erase Drawing

ID	**Use Case 5**
Name	Erase Drawing
Description	The user erases all of his current drawing
Pre-condition	System is running
Event flow	1. Click on 'Erase' 2. Canvas resets to blank

ID	**Use Case 6**
Name	Display Result
Description	The website displays the returned letter corresponding to the input
Pre-condition	System is running The computational model is functional
Event flow	1. The letter with the most resemblance to the input is displayed

ID	**Use Case 7**
Name	Choose Learn Mode
Description	The user chooses the learn mode option
Pre-condition	System is running The computational model is functional
Event flow	1. Click on 'Learn' button

ID	**Use Case 8**
Name	Display Map
Description	The website displays the SOM
Pre-condition	System is running The computational model is functional
Event flow	The topological map is printed out for the user

ID	**Use Case 9**
Name	Play Animation
Description	The website plays the neural network animation
Pre-condition	System is running
Event flow	1. User clicks on play button 2. The animation is played

ID	**Use Case 10**
Name	Hover on Map Point Data
Description	The user hovers over a particular point on the SOM
Pre-condition	System is running The computatinal model is functional
Event flow	1. User brings cursor over map point data 2. Map point shows contextual values

ID	**Use Case 11**
Name	Click Dataset
Description	The user selects to view the dataset
Pre-condition	System is running
Event flow	1. Click on 'Dataset' button

ID	**Use Case 12**
Name	Select Letter
Description	The user selects a letter from all whole alphabet
Pre-condition	System is running
Event flow	1. Click on 'Dataset' button 2. Click on a letter

ID	Use Case 13
Name	Select Character
Description	The user selects a given character of the chosen letter
Pre-condition	System is running
Event flow	1. Click on 'Dataset' button 2. Click on a letter 3. Click on a specific letter 4. Click

ID	Use Case 14
Name	Display Characters
Description	The website displays the meta data on the chosen character
Pre-condition	System is running
Event flow	The meta-data on such a character is displayed as a pop-up

ID	Use Case 15
Name	Close Site
Description	The user shuts down the browser
Pre-condition	System is running
Event flow	
Extension points	
Triggers	
Post-condition	The user exists the browser

Appendix F

Testing

F.1 Hardware

The testing of the application was done on the following hardware device:

Macbook Pro 15" Retina (1st Gen), early 2013[1]:

- OS: macOS High-Sierra
- Processor: 2.4 GHz Intel Core i7
- Memory: 8 GB 1600 MHz DDR3
- Graphics: NVIDIA GeForce GT 650M 1024 MB, Intel HD Graphics 4000 1536 MB

F.2 Software

Google Chrome's browser in developer mode also allows testing in various screen sizes and resolutions which was thoroughly used for UI formatting testing. This allowed to maintain a universal look and feel of the website across different devices, and isn't exclusively device-dependent.

The developer PC's task manager also allowed to monitor for any eventual memory leaks or excessive CPU usages, and was be used to optimise the web application. This was of relative importance as battery life is generally important, and a bad experience could deter people from using the website again. Network usage of website was also be looked at to decide whether or not to optimise or compress certain features.

F.3 Test Results

The following is the testing results of the different scripts. Each test ID was executed with the command `$Python3ScriptName.py` following by any extra CLI parameter, such as `-d`. The parameters for each test case is given in the table, and a blank value represents no additional argument being parsed.

[1] MacBook Pro (Retina, 15-inch, Early 2013) - Technical Specifications. https://support.apple.com/kb/sp669?locale=en_US. (Accessed on 05/05/2018).

F.3.1 RGB

ID	Data	Data Type	Expected Result	Success?
1	(Blank)	Correct	Successful build	YES
2	`-i`	Erroneous	Native error message	YES
3	`-i=`	Erroneous	Native error message	YES
4	`-i=0`	Erroneous	Implemented error message	YES
5	`-i=-1`	Erroneous	Implemented error message	YES
6	`-i=0.5`	Erroneous	Native error message	YES
7	`-i=-0.5`	Erroneous	Native error message	YES
8	`-i=100`	Correct	Successful build	YES
9	`-r`	Erroneous	Native error message	YES
10	`-r=`	Erroneous	Native error message	YES
11	`-r=0`	Erroneous	Implemented error message	YES
12	`-r=-1`	Erroneous	Implemented error message	YES
13	`-r=0.5`	Correct	Successful build	YES
14	`-r=1`	Correct	Successful build	YES
15	`-r=1.5`	Erroneous	Implemented error message	YES
16	`-d`	Correct	Successful build	YES
17	`-d-i=100`	Correct	Successful build	YES
18	`-d-r=0.3`	Correct	Successful build	YES
19	`-r=0.3-i=100`	Correct	Successful build	YES
20	`-d-r=0.3-i=100`	Correct	Successful build	YES

Table F.1: RGB script tests

F.3.2 Iris

ID	Data	Type	Expected Result	Success?
1	(Blank)	Correct	Successful build	YES
2	`-r`	Erroneous	Native error message	YES
3	`-r=`	Erroneous	Native error message	YES
4	`-r=0`	Erroneous	Implemented error message	YES
5	`-r=-1`	Erroneous	Implemented error message	YES
6	`-r=0.5`	Correct	Successful build	YES
7	`-r=1`	Correct	Successful build	YES
8	`-r=1.5`	Erroneous	Implemented error message	YES
9	`-d`	Correct	Successful build	YES
10	`-d-r=0.3`	Correct	Successful build	YES

Table F.2: Iris script tests

F.3.3 OCR

ID	Data	Type	Expected Result	Success?
1	(Blank)	Correct	Successful build	YES
2	`-d`	Correct	Successful build	YES
3	`-r`	Erroneous	Native error message	YES
4	`-r=`	Erroneous	Native error message	YES
5	`-r=0`	Erroneous	Implemented error message	YES
6	`-r=-1`	Erroneous	Implemented error message	YES
7	`-r=0.5`	Correct	Successful build	YES
8	`-r=1`	Correct	Successful build	YES
9	`-r=1.5`	Erroneous	Implemented error message	YES
10	`-iTr=100`	Correct	Successful build	YES
11	`-iTr=0`	Correct	Successful build	YES
12	`-iTr=-1`	Erroneous	Implemented error message	YES
13	`-iTr=2400`	Correct	Successful build	YES
14	`-iTr=2401`	Erroneous	Implemented error message	YES
15	`-iTe=100`	Correct	Successful build	YES
16	`-iTe=0`	Correct	Successful build	YES
17	`-iTe=-1`	Erroneous	Implemented error message	YES
18	`-iTe=2400`	Correct	Successful build	YES
19	`-iTe=2401`	Erroneous	Implemented error message	YES
20	`-t=d`	Correct	Successful build	YES
21	`-t=l`	Correct	Successful build	YES
22	`-t=c`	Correct	Successful build	YES
23	`-t=z`	Erroneous	Implemented error message	YES
24	`-d-iTr=100`	Correct	Successful build	YES
25	`-d-iTe=100`	Correct	Successful build	YES
26	`-d-r=0.3`	Correct	Successful build	YES
27	`-d-r=0.3-iTr=100`	Correct	Successful build	YES
28	`-d-r=0.3-iTr=100-iTe=100`	Correct	Successful build	YES
29	`-d-r=0.3-iTr=100-iTe=100-t=d`	Correct	Successful build	YES

Table F.3: OCR script tests

Appendix G

Web-Pages

FIGURE G.1: Page 1

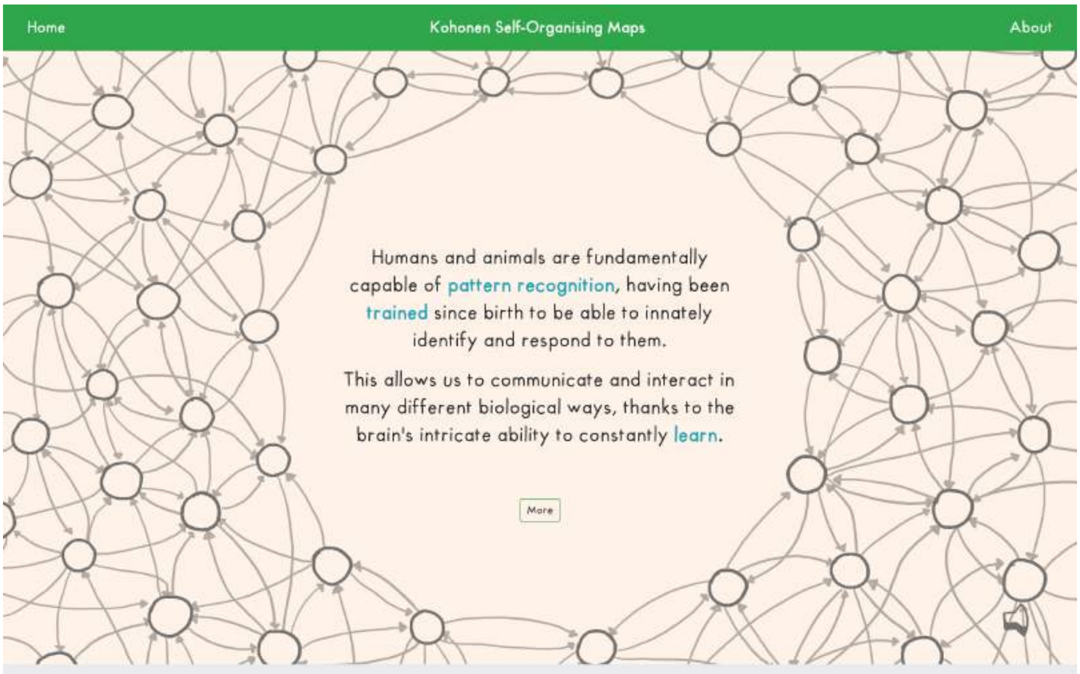


Figure G.2: Page 2

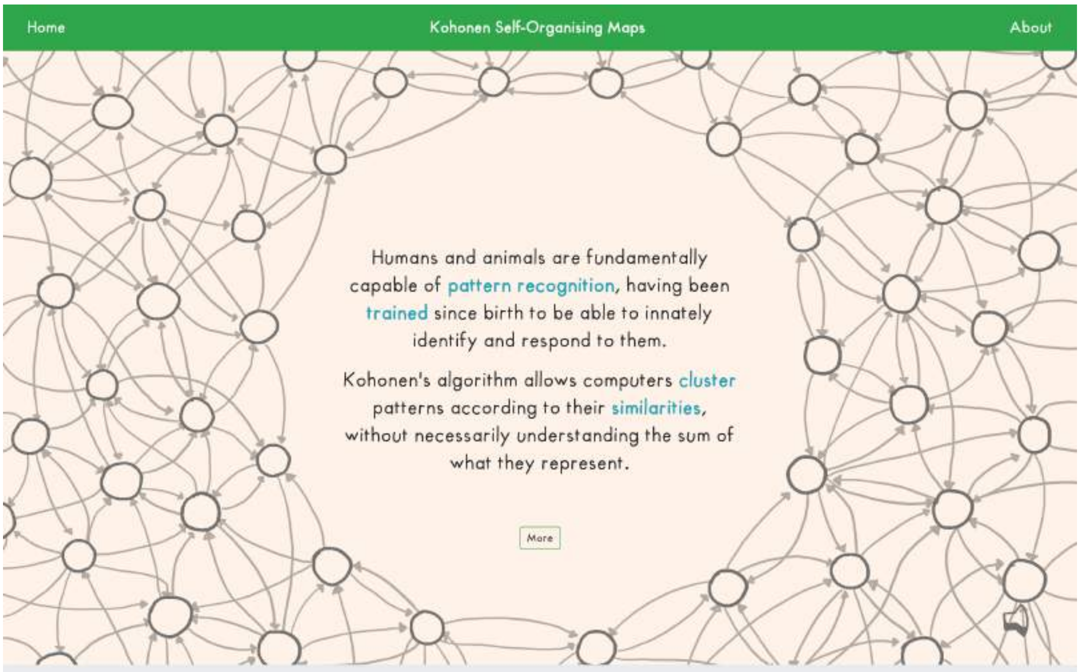


Figure G.3: Page 3

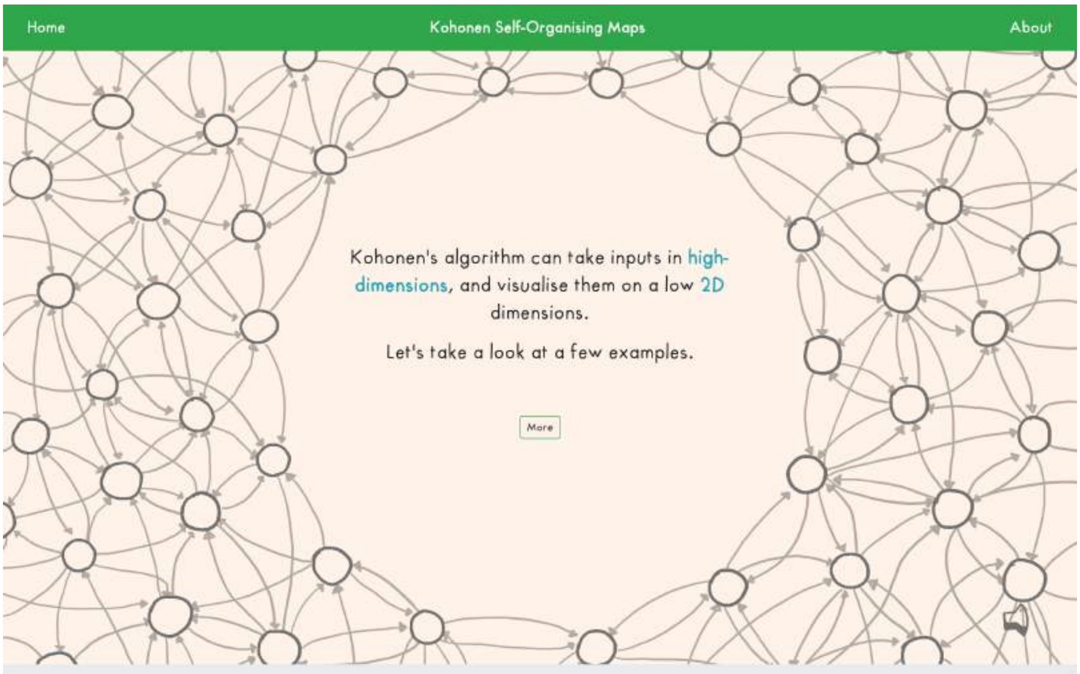


FIGURE G.4: Page 4

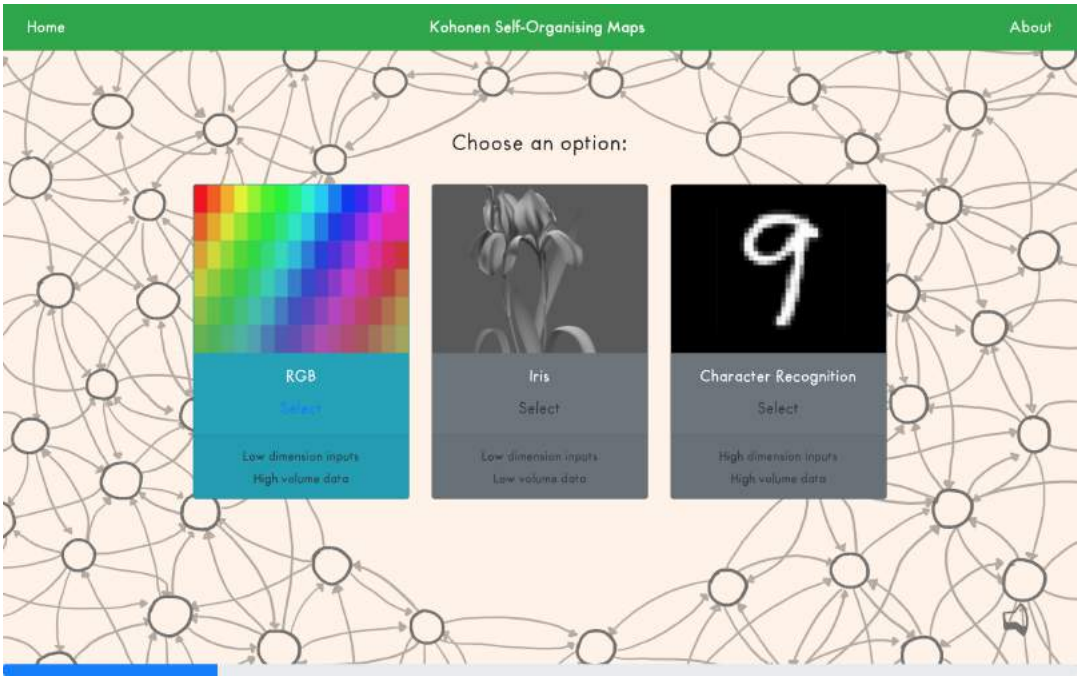


FIGURE G.5: Page 5

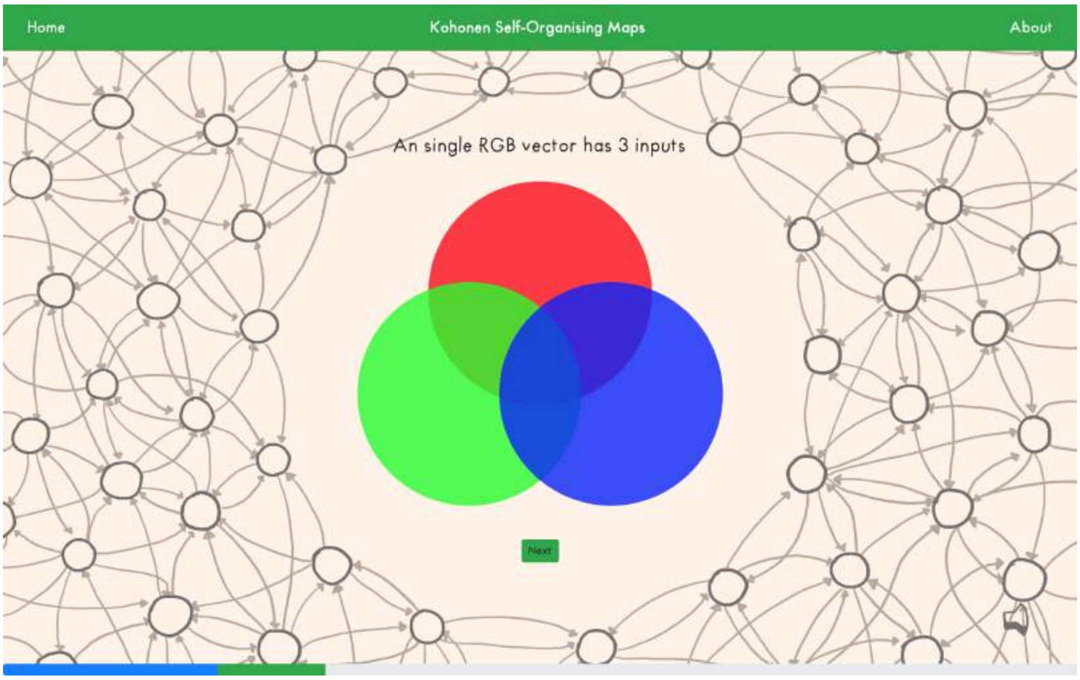


FIGURE G.6: Page 6

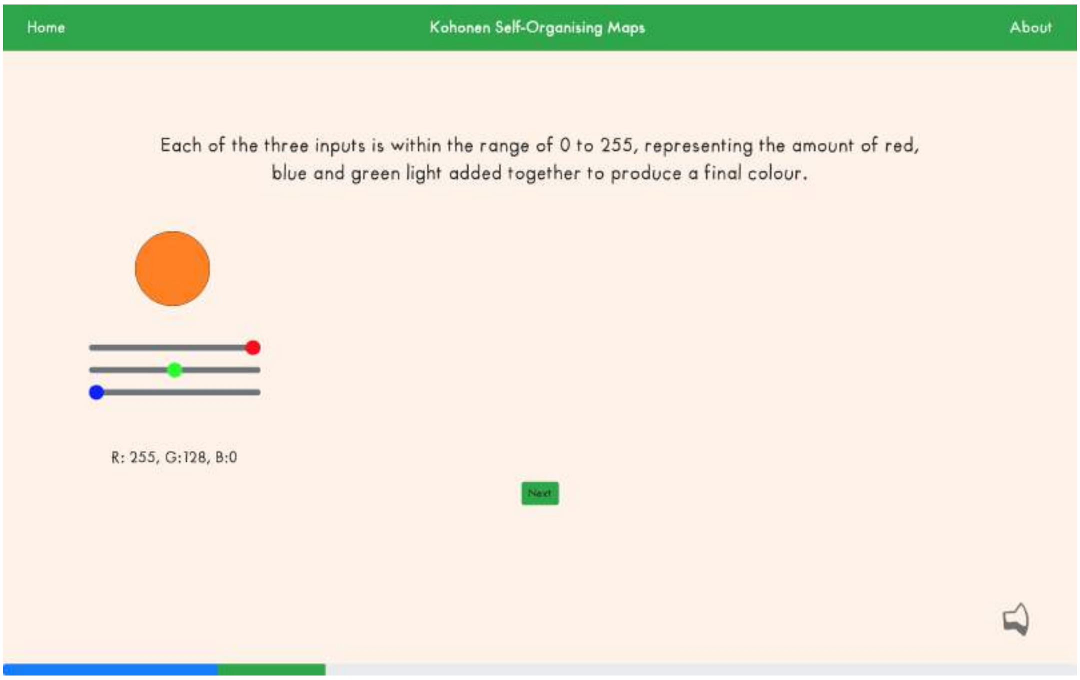


FIGURE G.7: Page 7

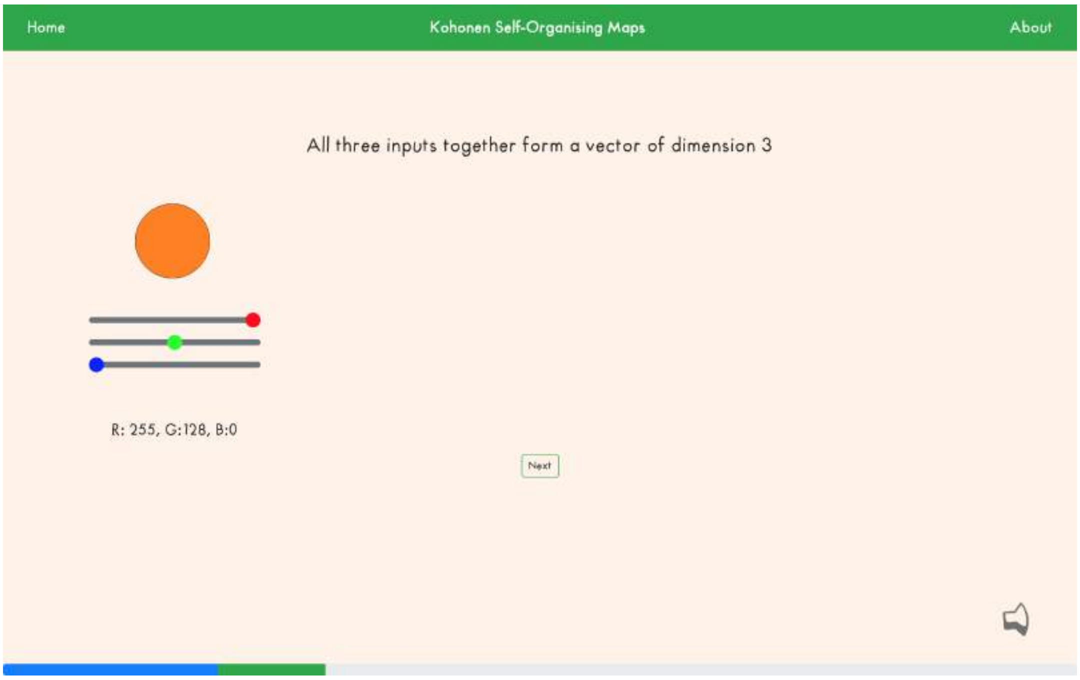


FIGURE G.8: Page 8

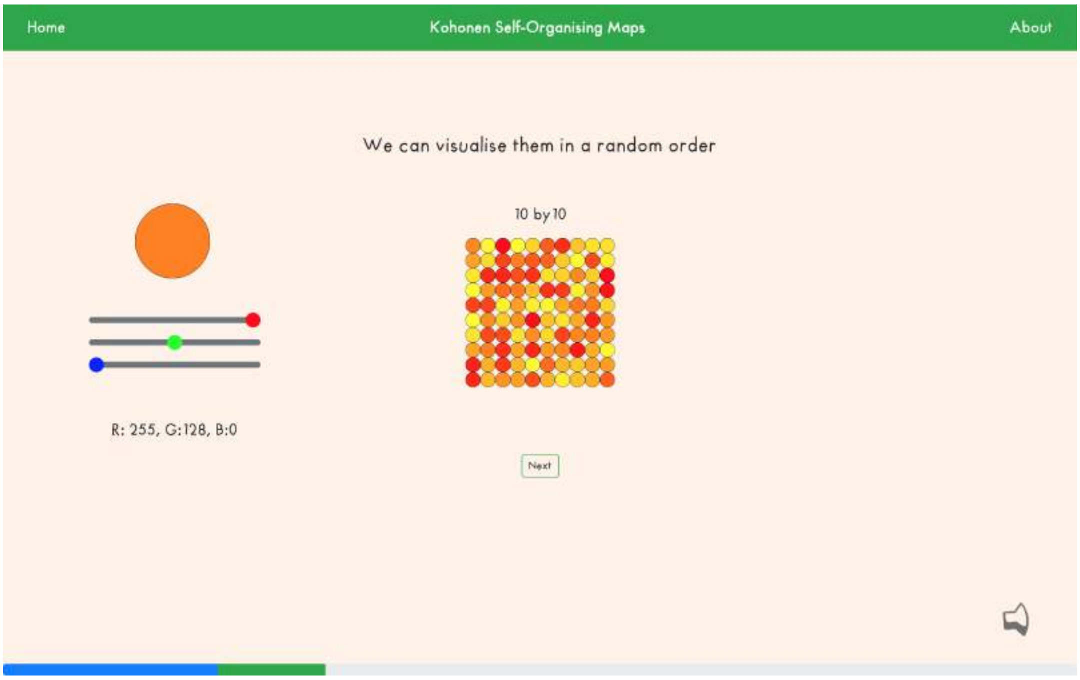


FIGURE G.9: Page 9

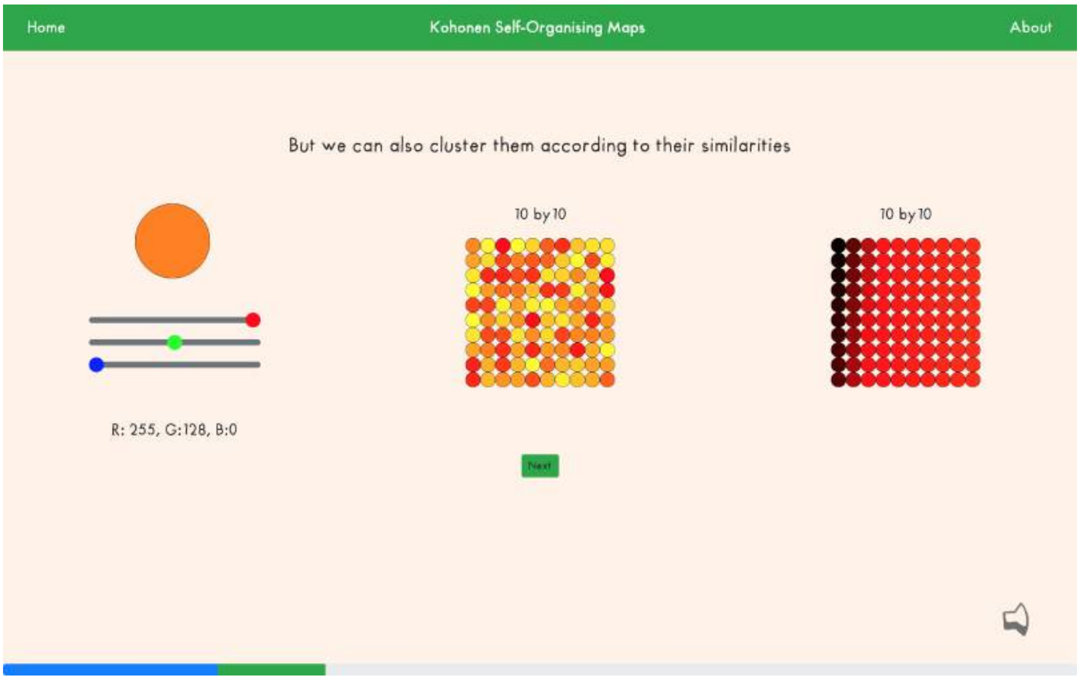


FIGURE G.10: Page 10

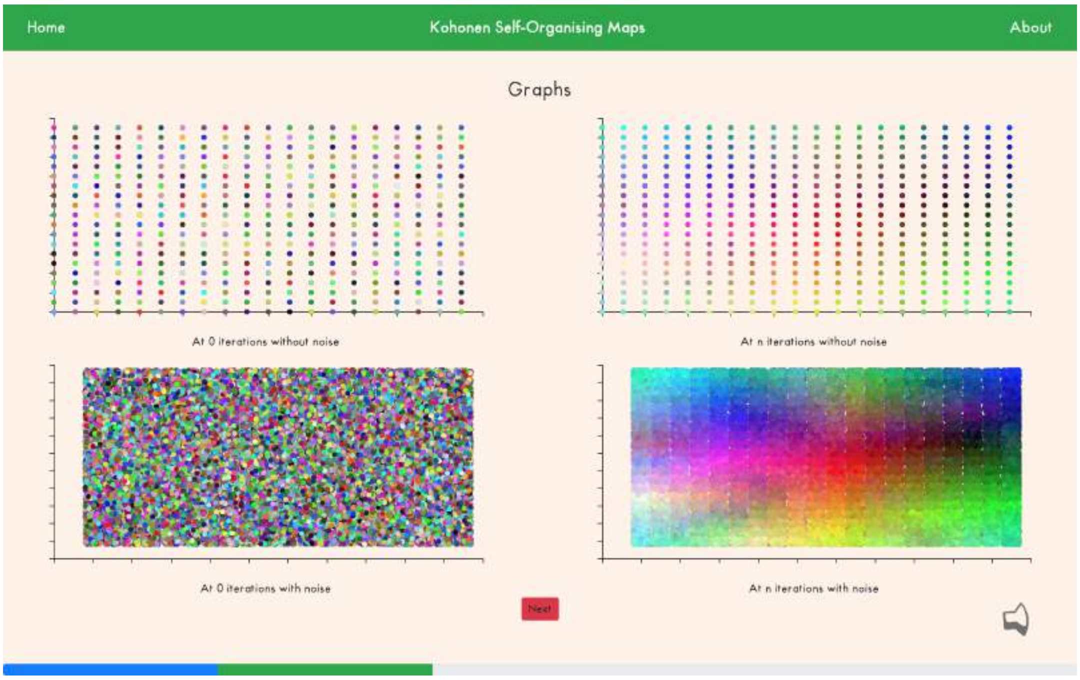


FIGURE G.11: Page 11

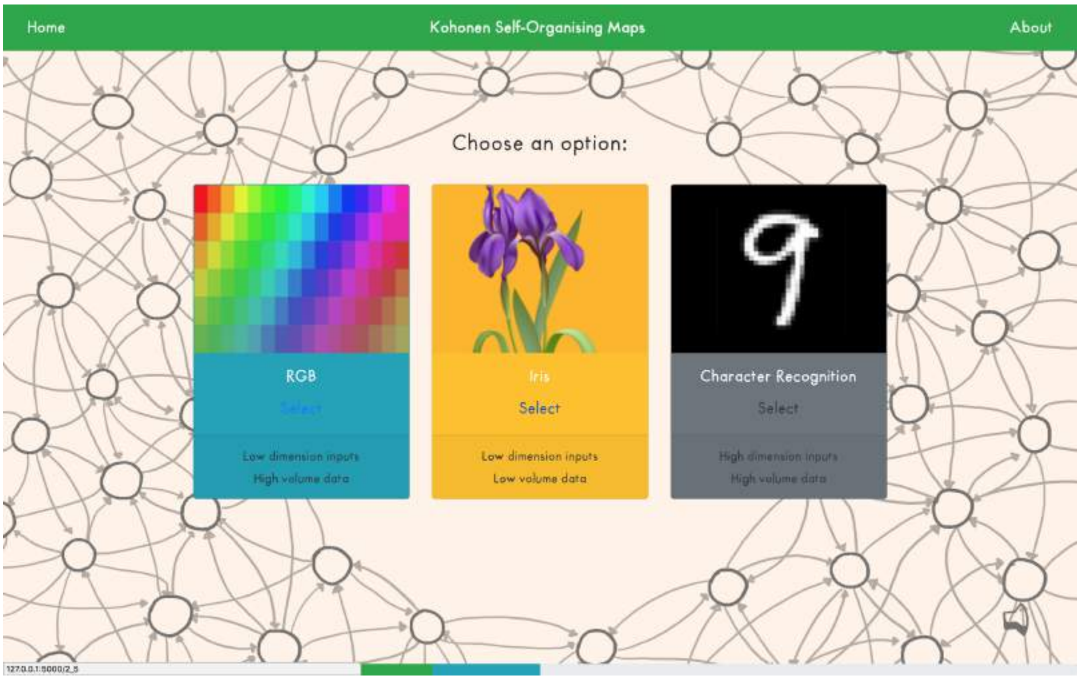


FIGURE G.12: Page 12

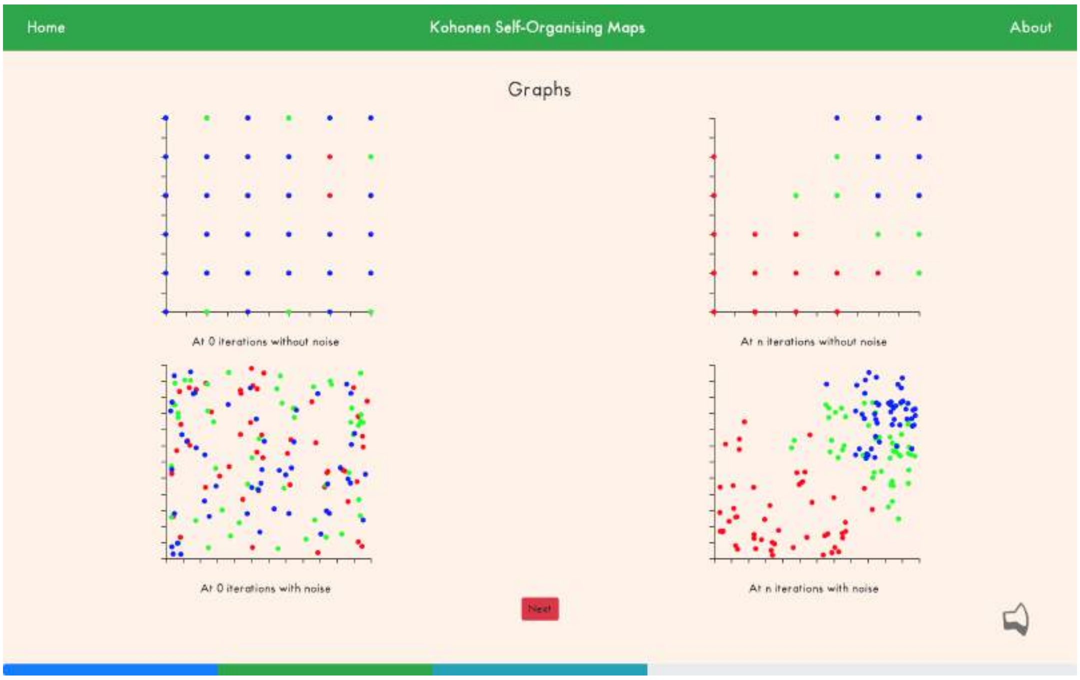


FIGURE G.13: Page 13

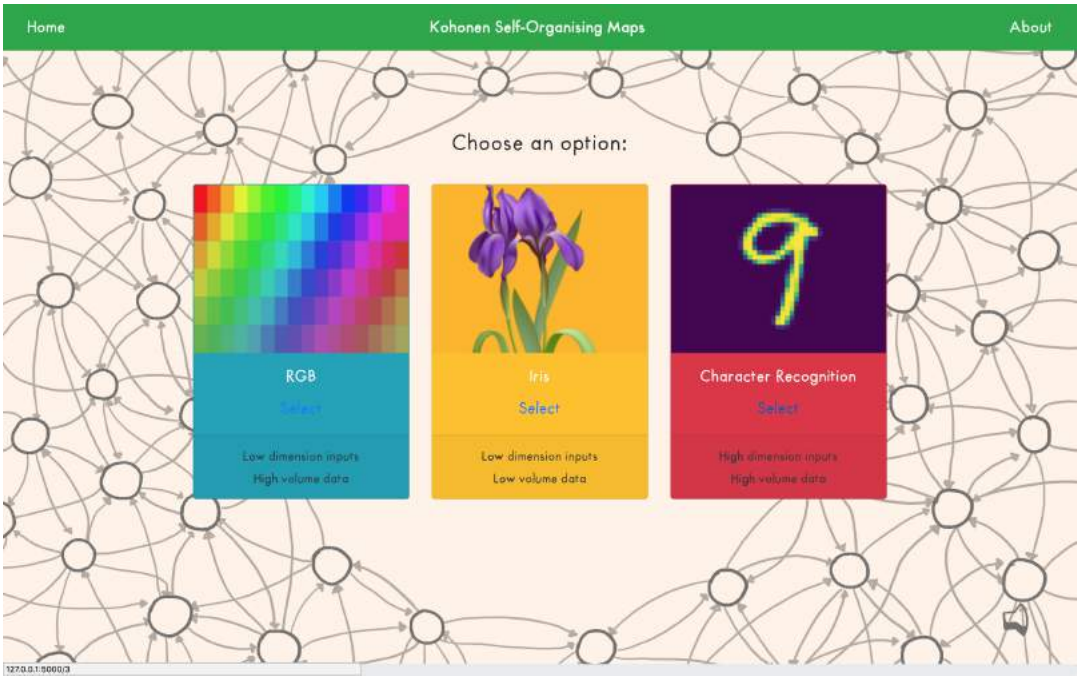


FIGURE G.14: Page 14

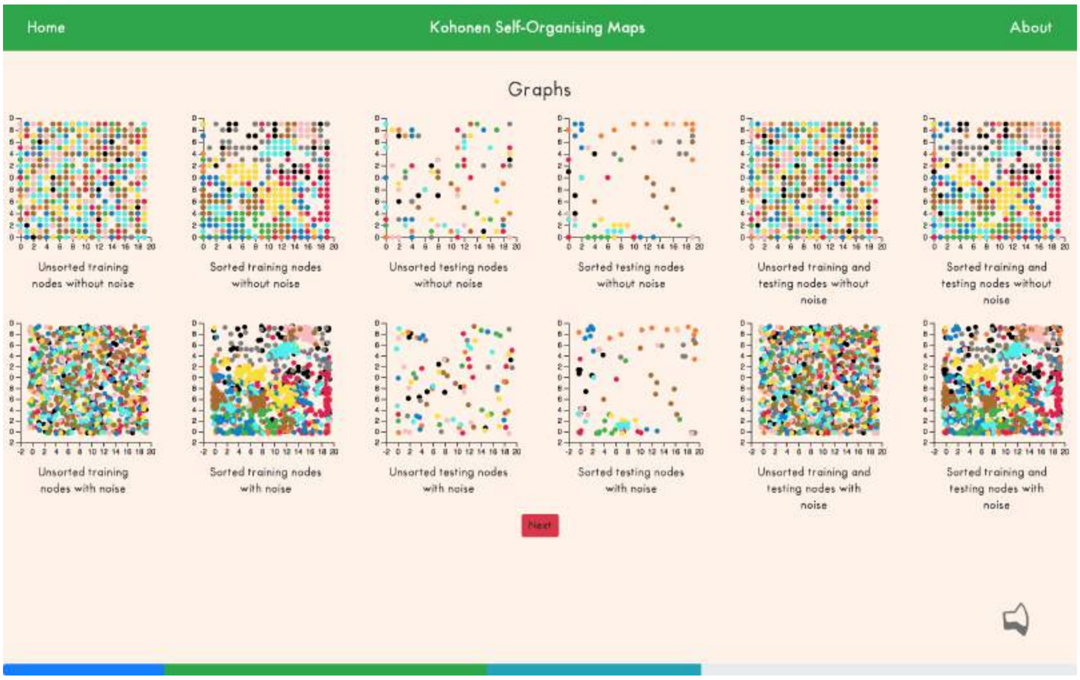


FIGURE G.15: Page 15

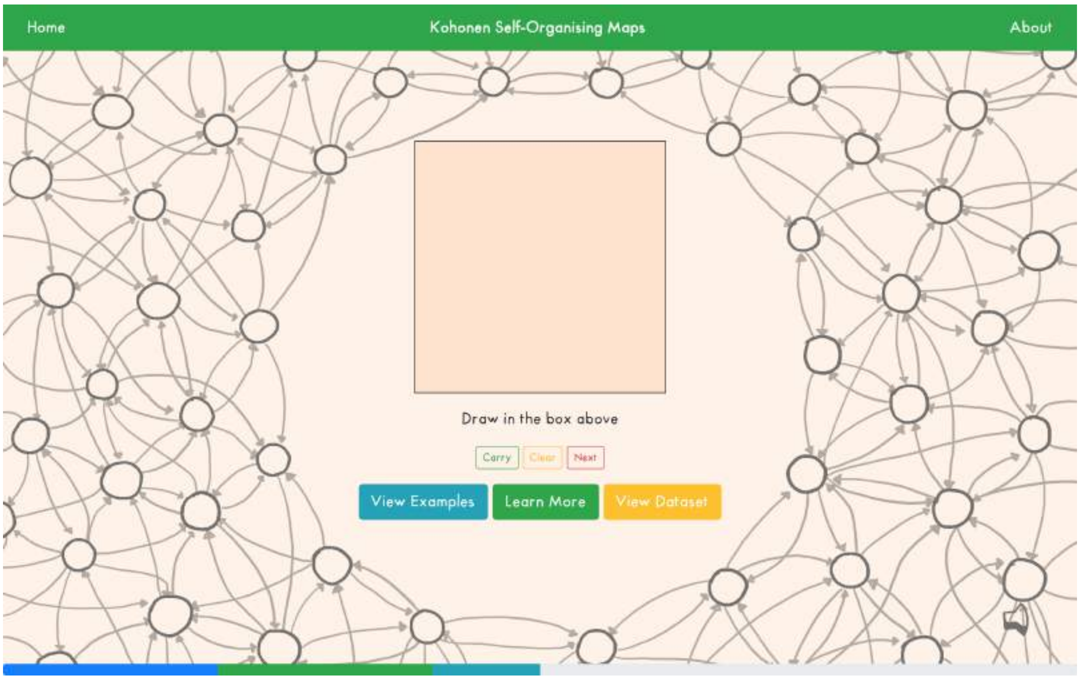


FIGURE G.16: Page 16

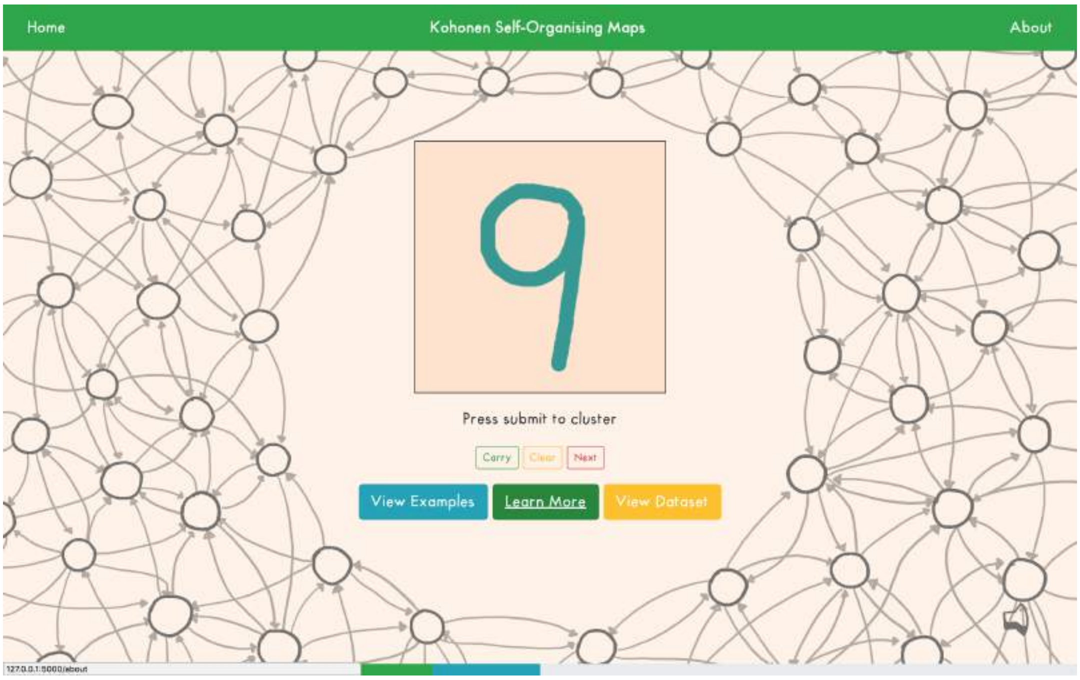


FIGURE G.17: Page 17

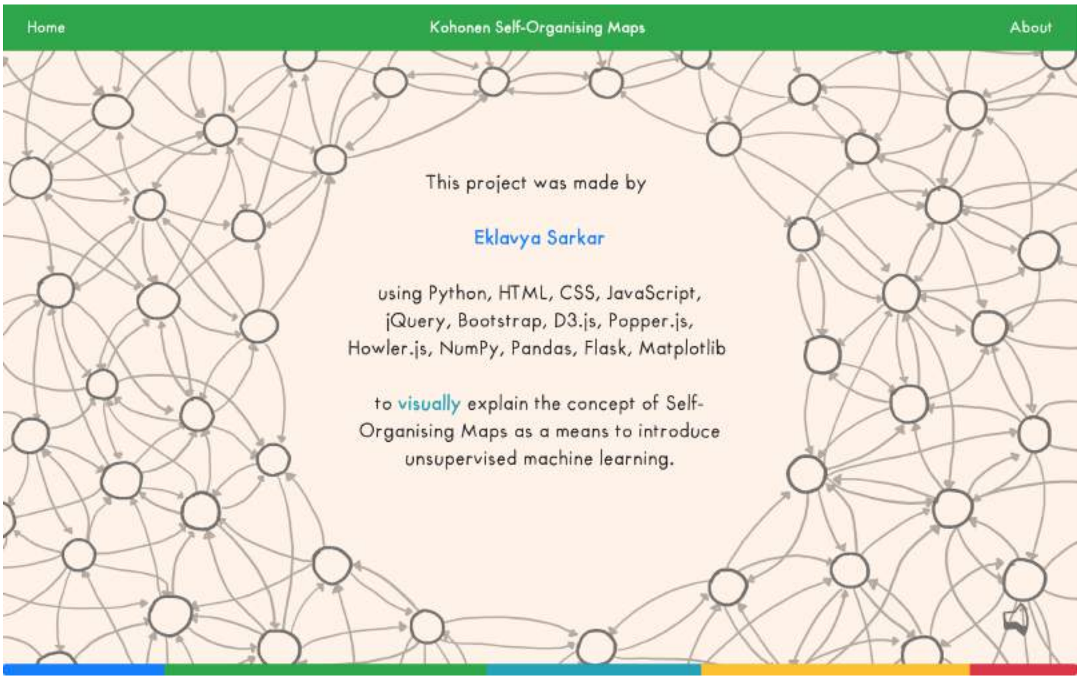


FIGURE G.18: Page 18

Appendix H

Plots

H.1 RGB

H.1.1 0.3 Learning Rate, 1000 Inputs

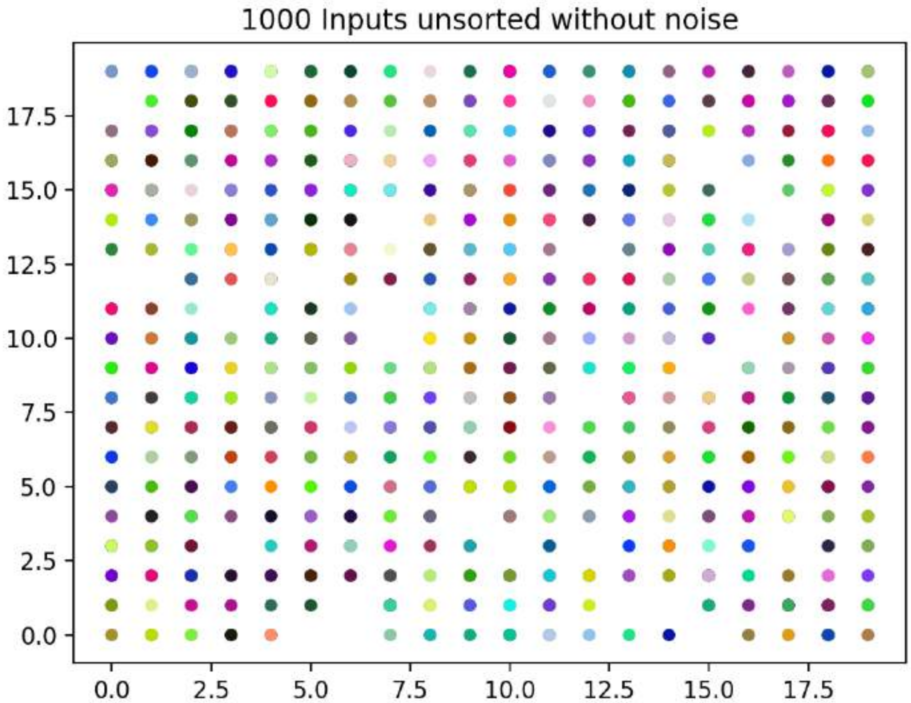


FIGURE H.1: RGB Plot 1

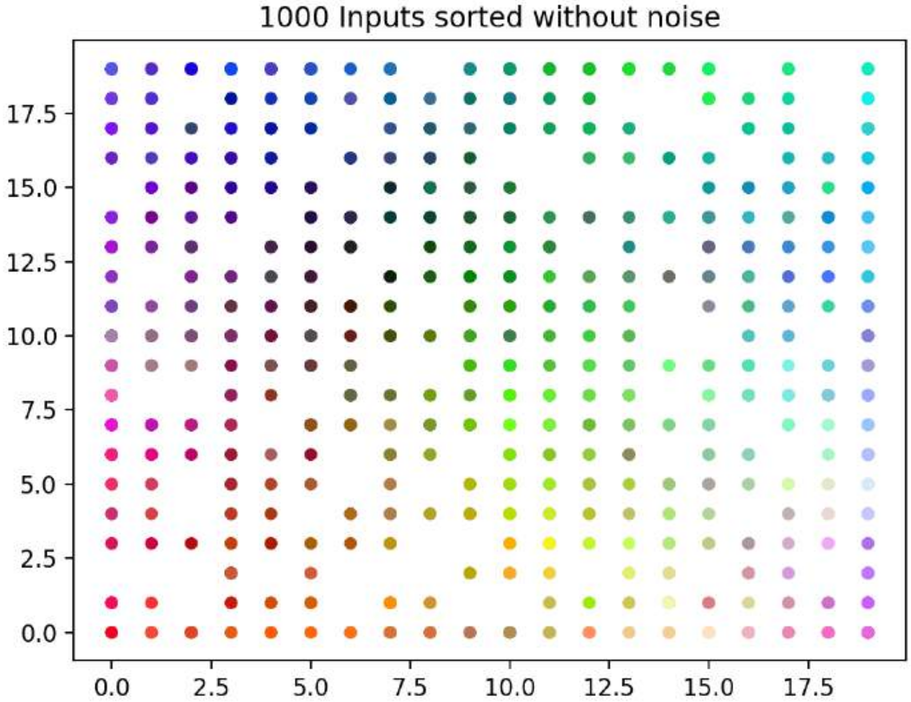


Figure H.2: RGB Plot 2

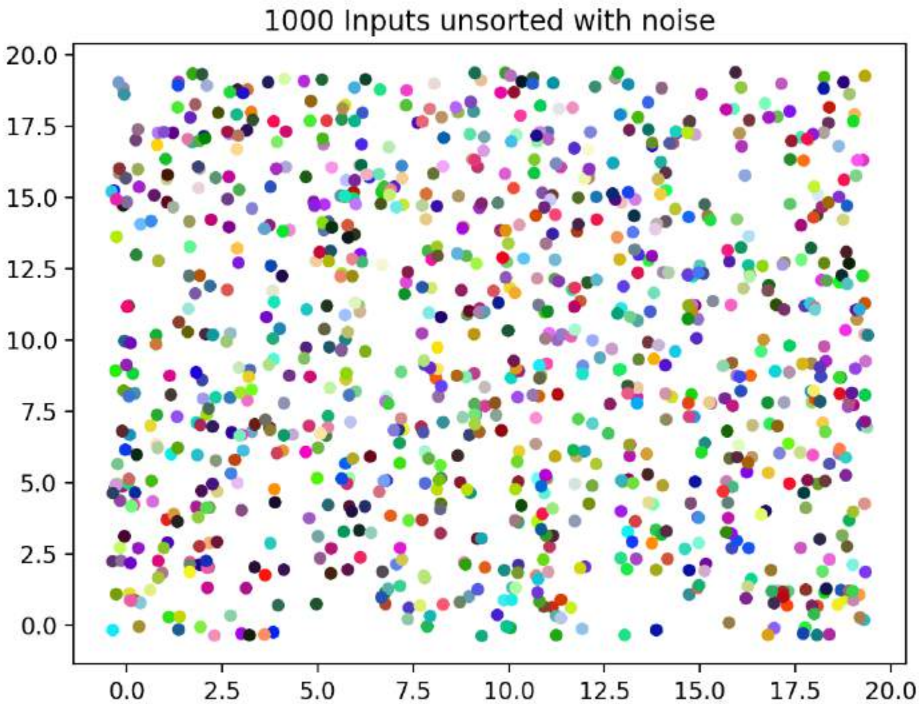


Figure H.3: RGB Plot 3

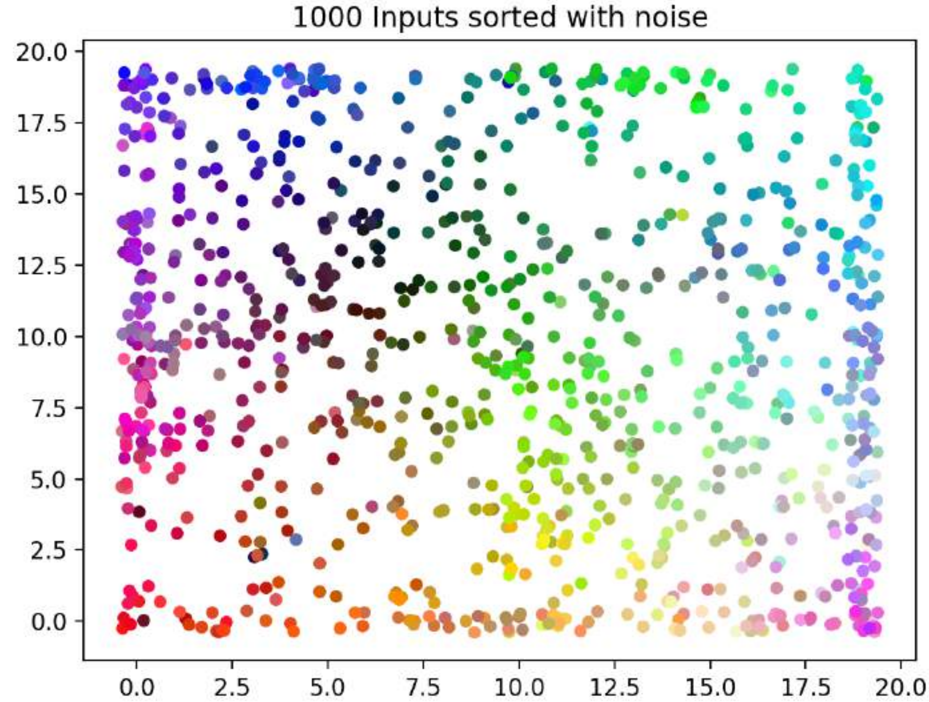


FIGURE H.4: RGB Plot 4

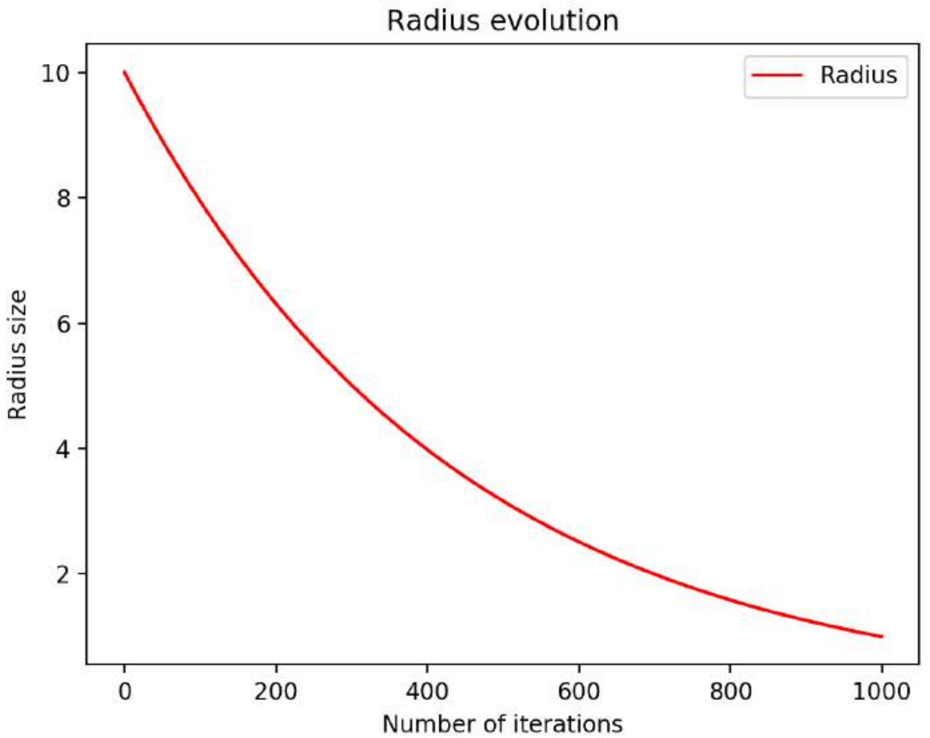


FIGURE H.5: RGB Plot 5

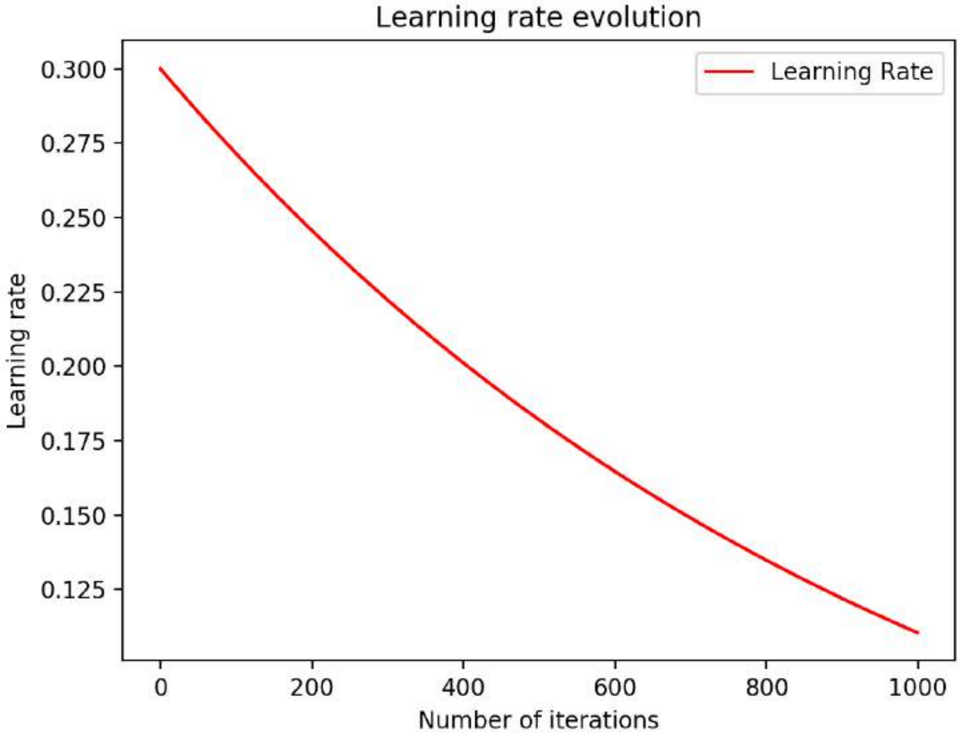


Figure H.6: RGB Plot 6

H.2 Iris

H.2.1 0.3 Learning Rate

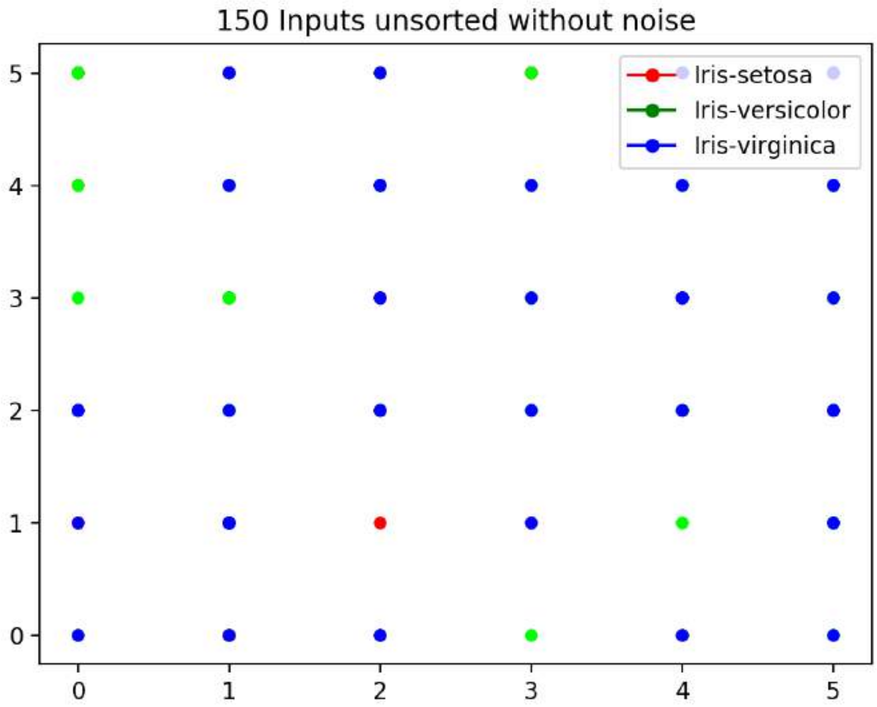


Figure H.7: Iris Plot 1

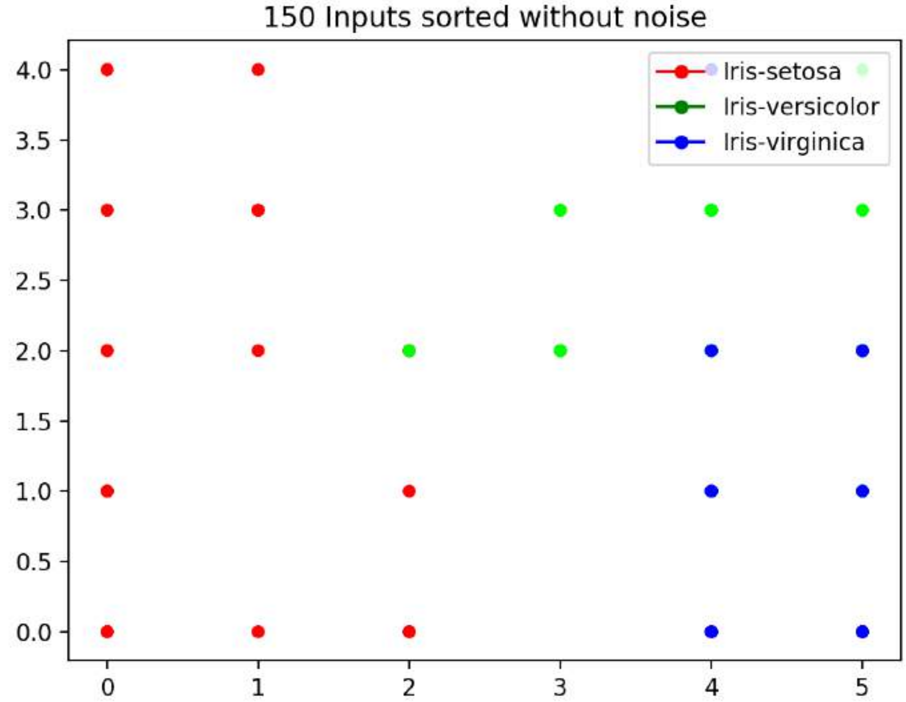


FIGURE H.8: Iris Plot 2

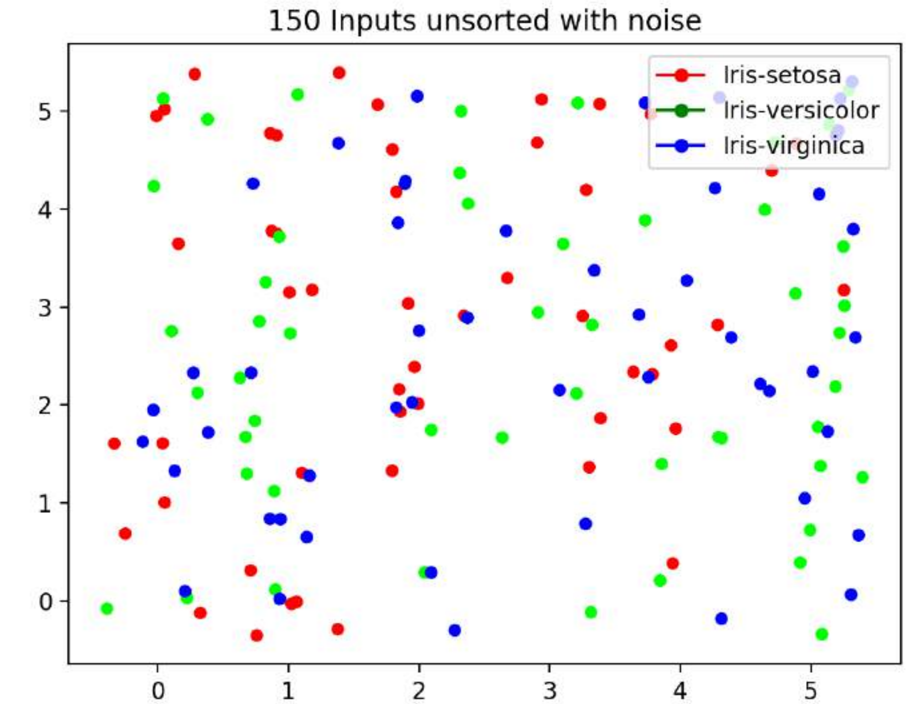


FIGURE H.9: Iris Plot 3

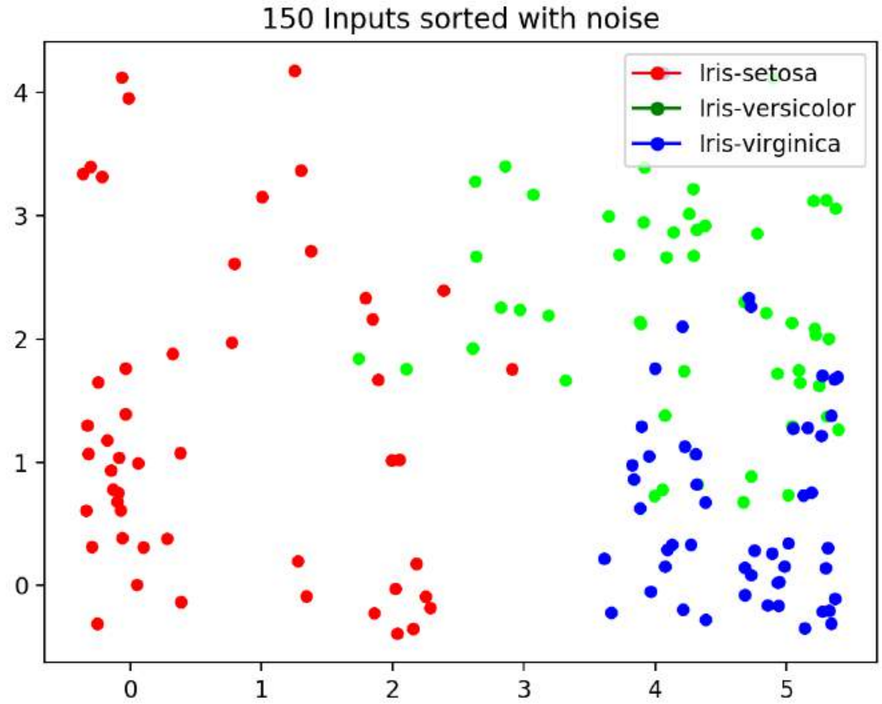


FIGURE H.10: Iris Plot 4

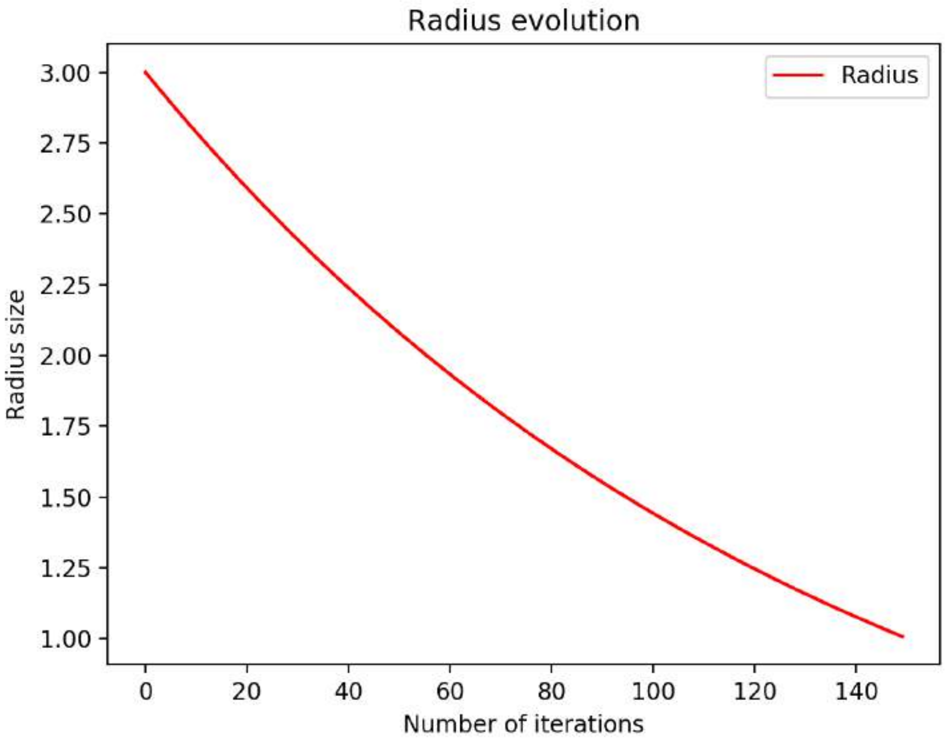


FIGURE H.11: Iris Plot 5

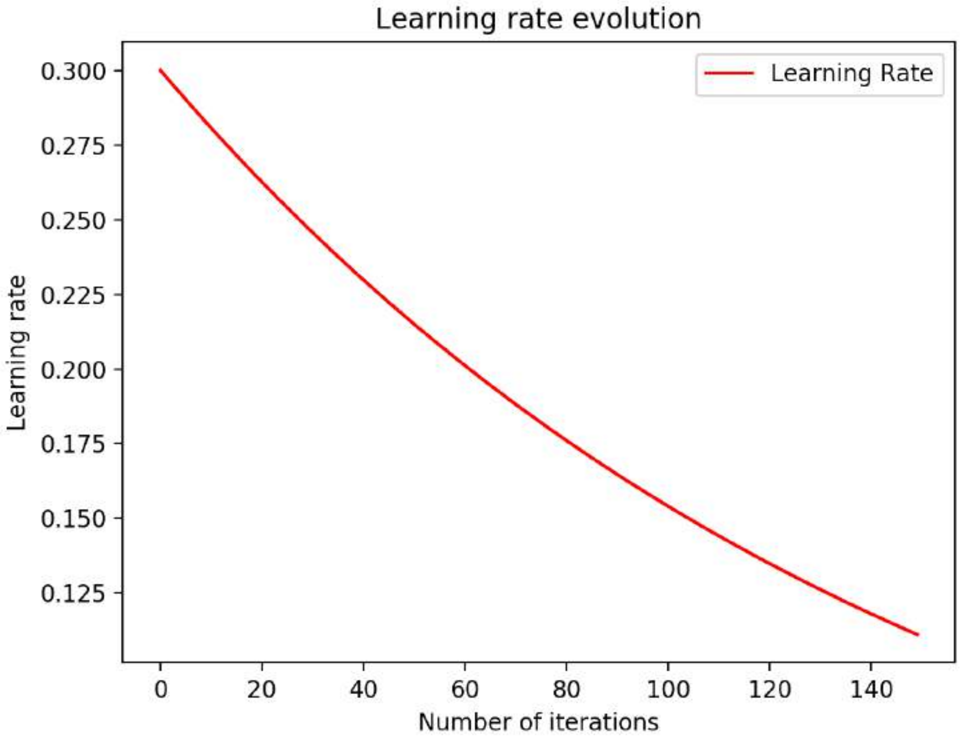


FIGURE H.12: Iris Plot 6

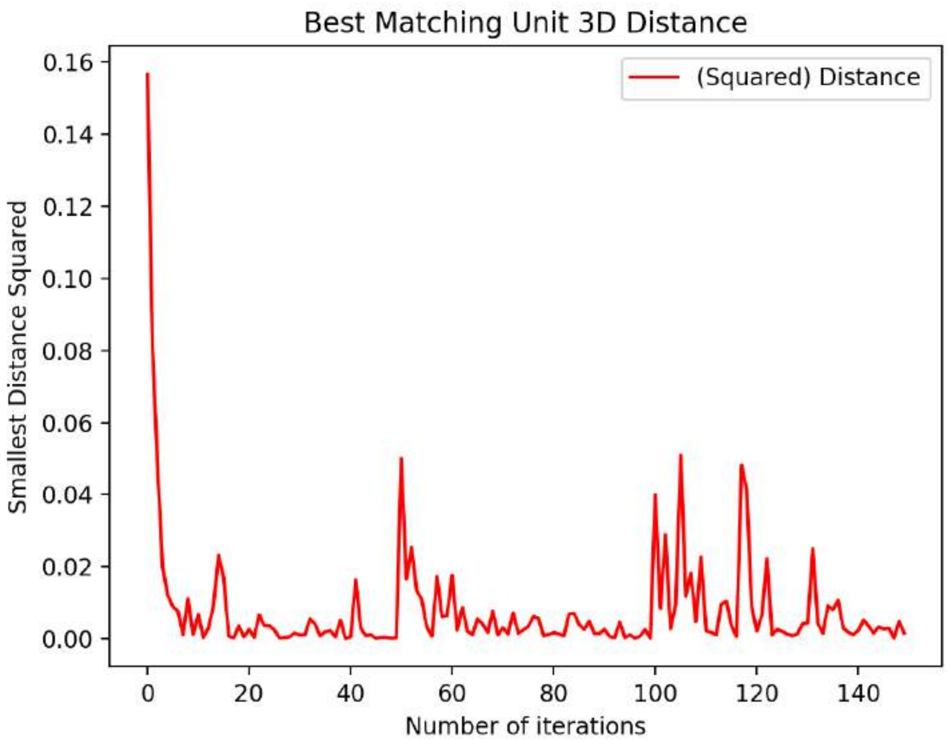


FIGURE H.13: Iris Plot 7

H.2.2 0.8 Learning Rate

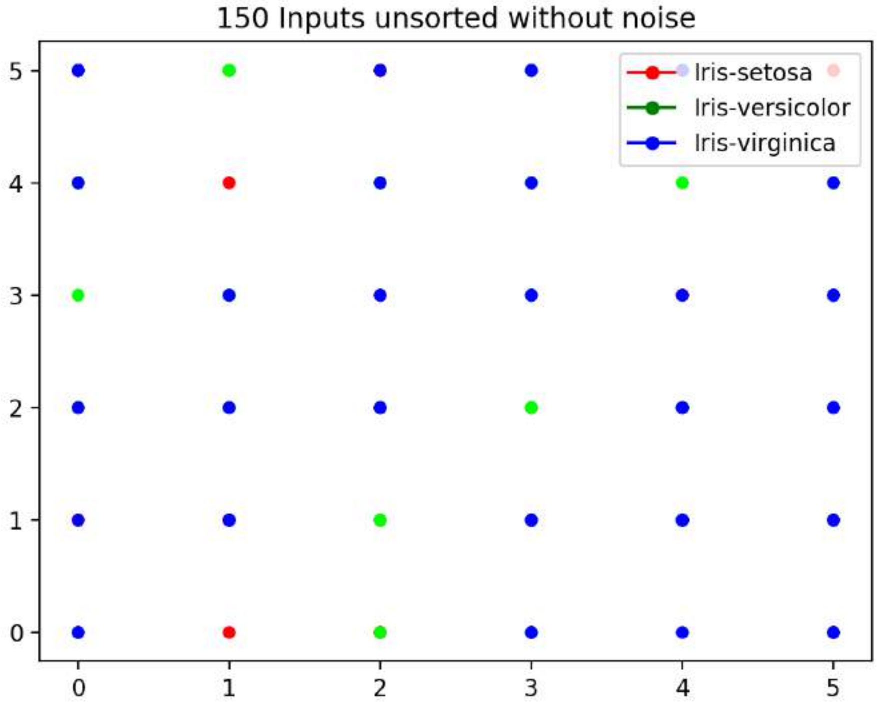


FIGURE H.14: Iris Plot 8

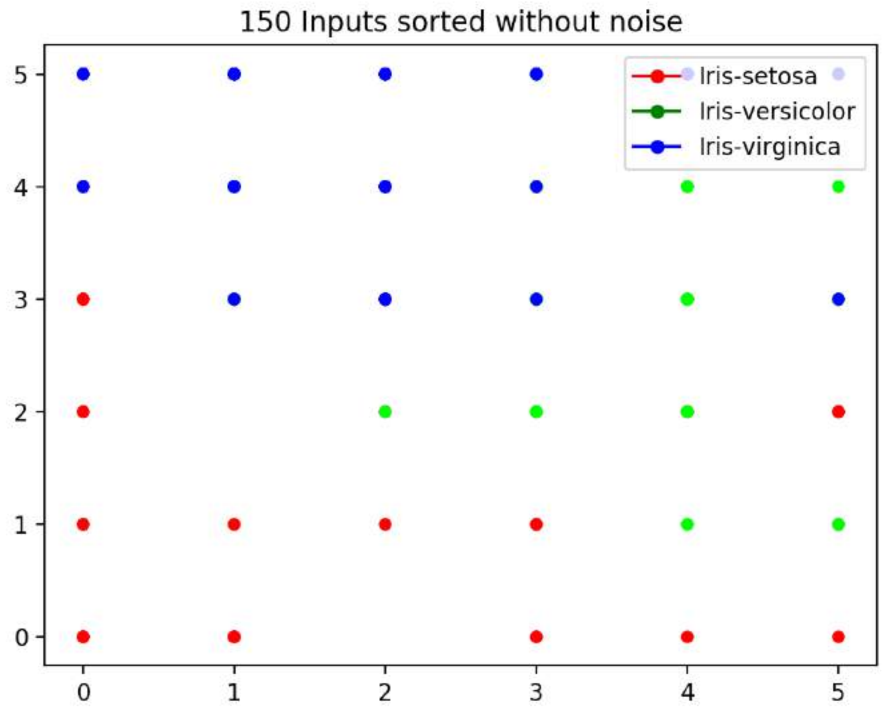


FIGURE H.15: Iris Plot 9

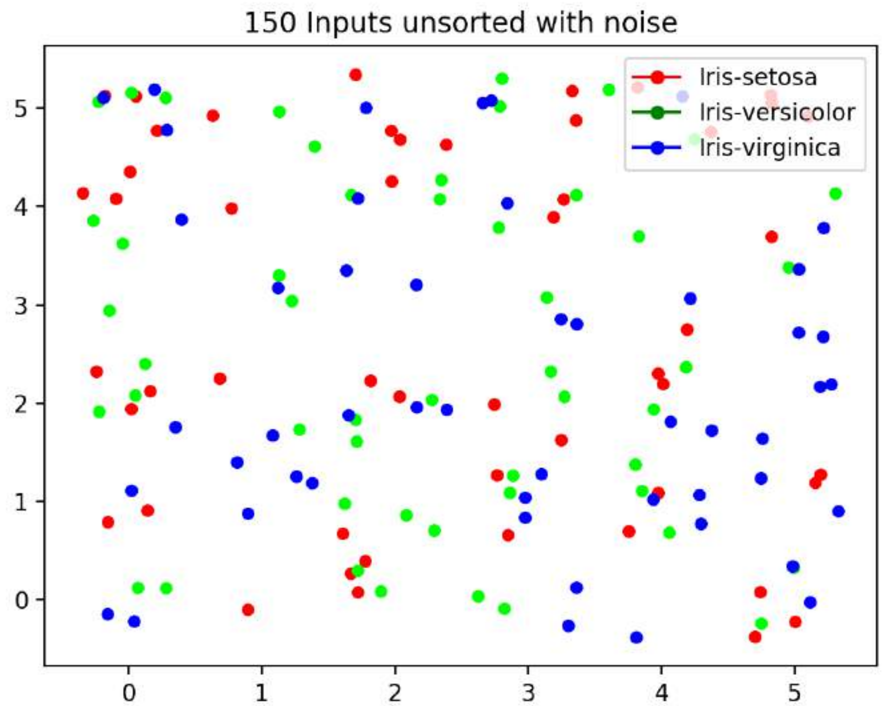


FIGURE H.16: Iris Plot 10

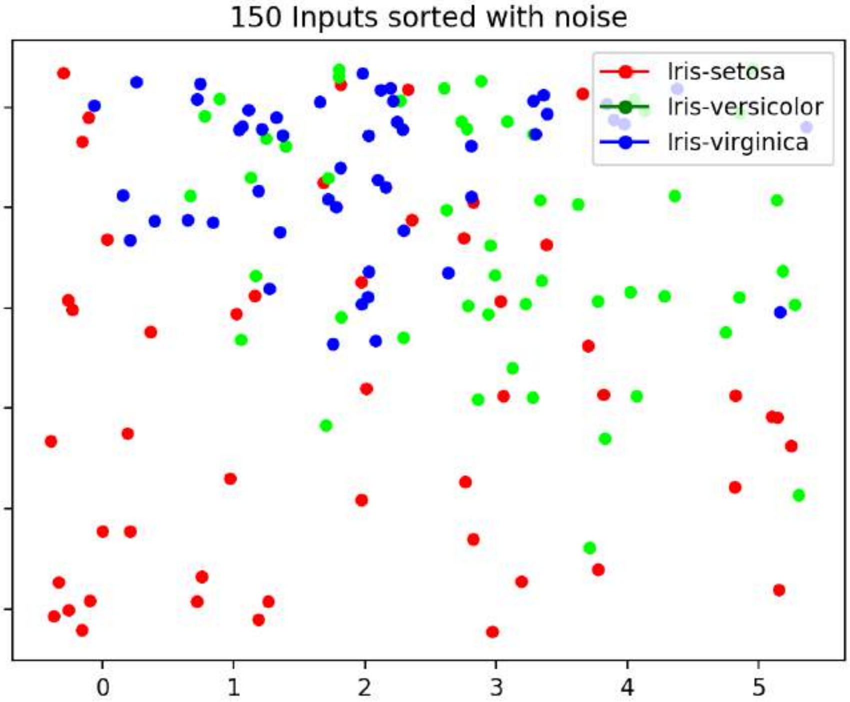


FIGURE H.17: Iris Plot 11

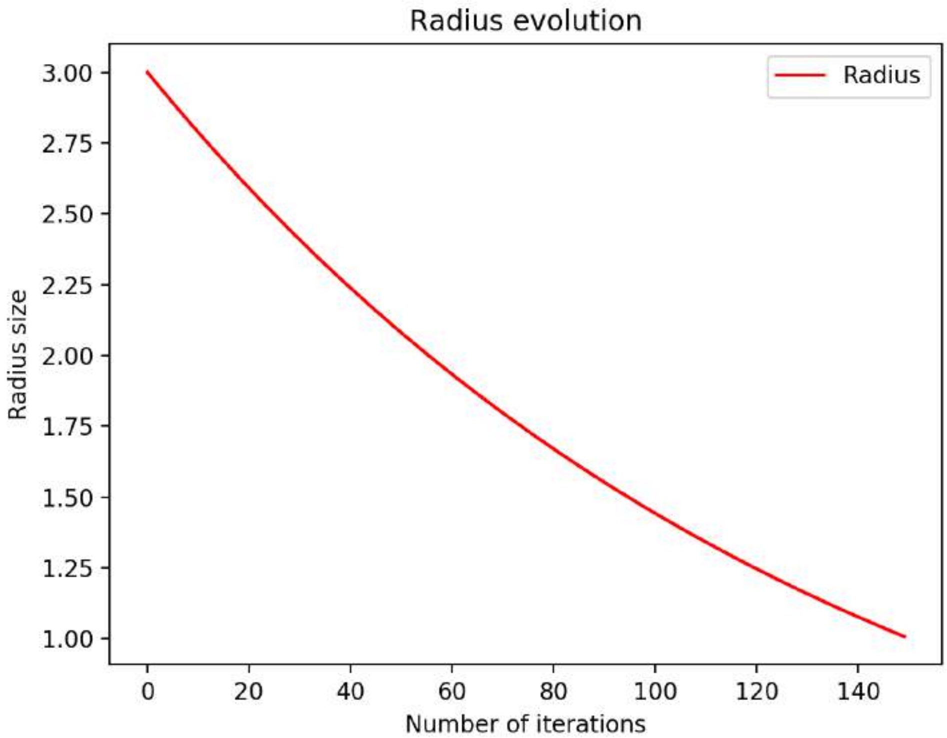


FIGURE H.18: Iris Plot 12

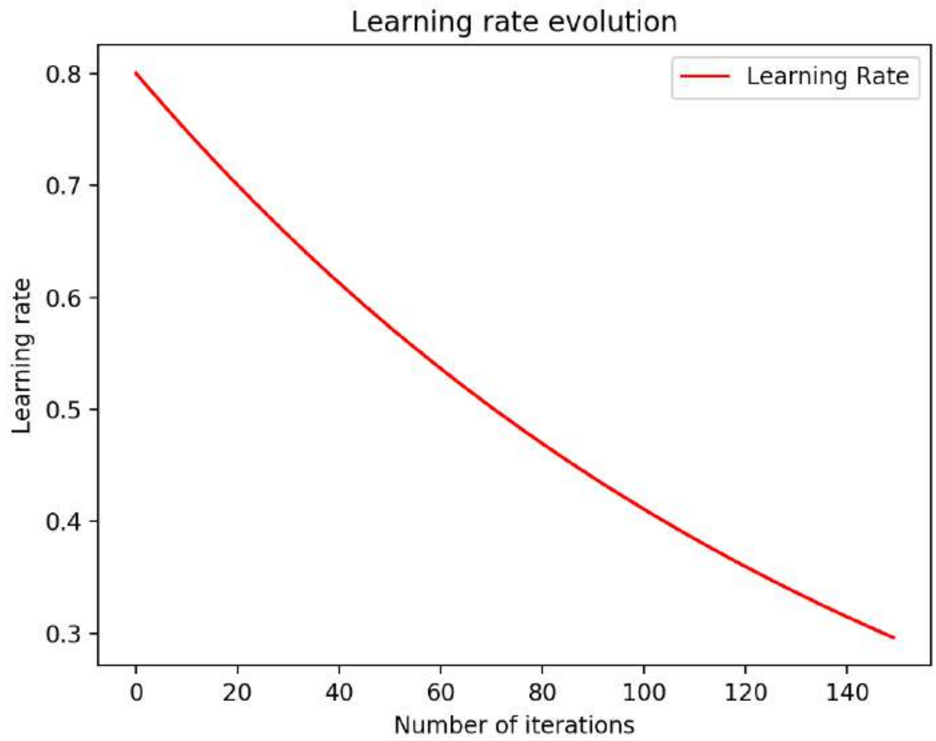


FIGURE H.19: Iris Plot 13

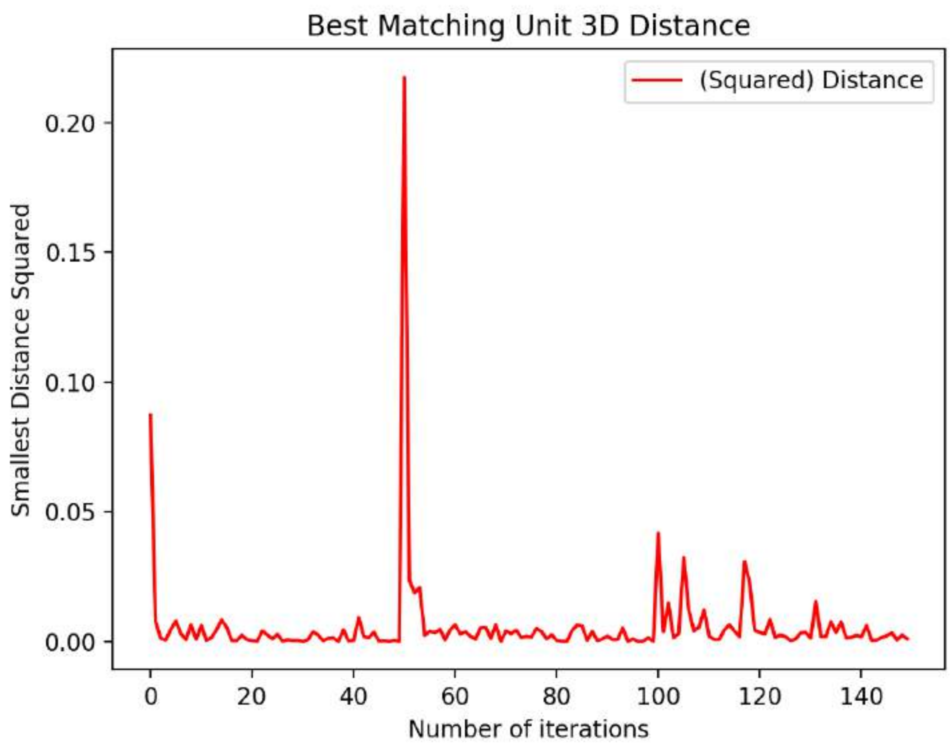


FIGURE H.20: Iris Plot 14

H.3 OCR

H.3.1 0.3 Learning Rate, 100 Training Inputs, 10 Testing Inputs

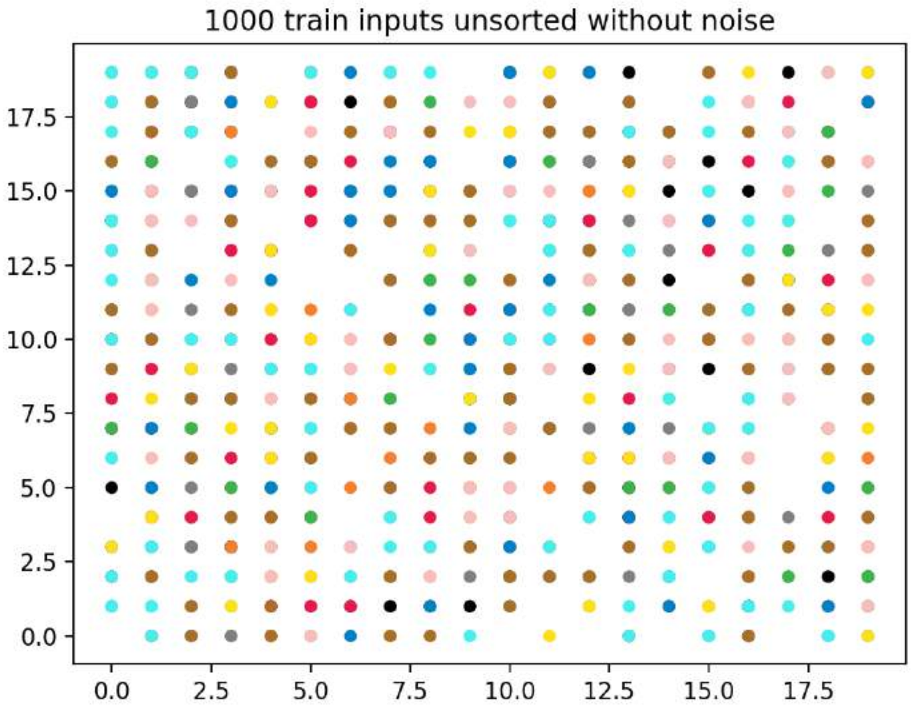


FIGURE H.21: OCR Plot 1

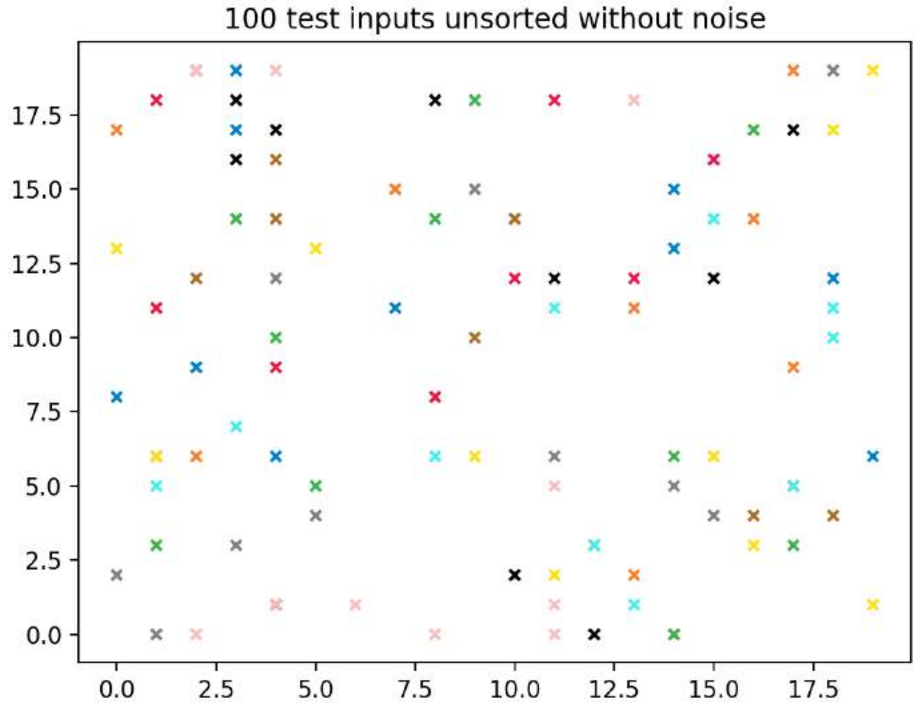


Figure H.22: OCR Plot 2

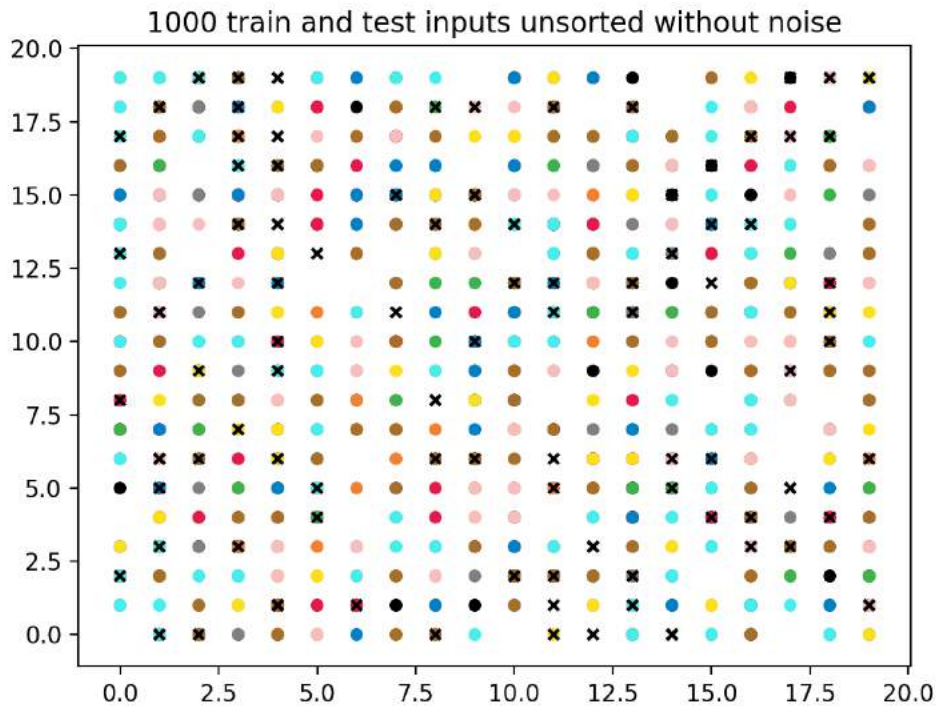


Figure H.23: OCR Plot 3

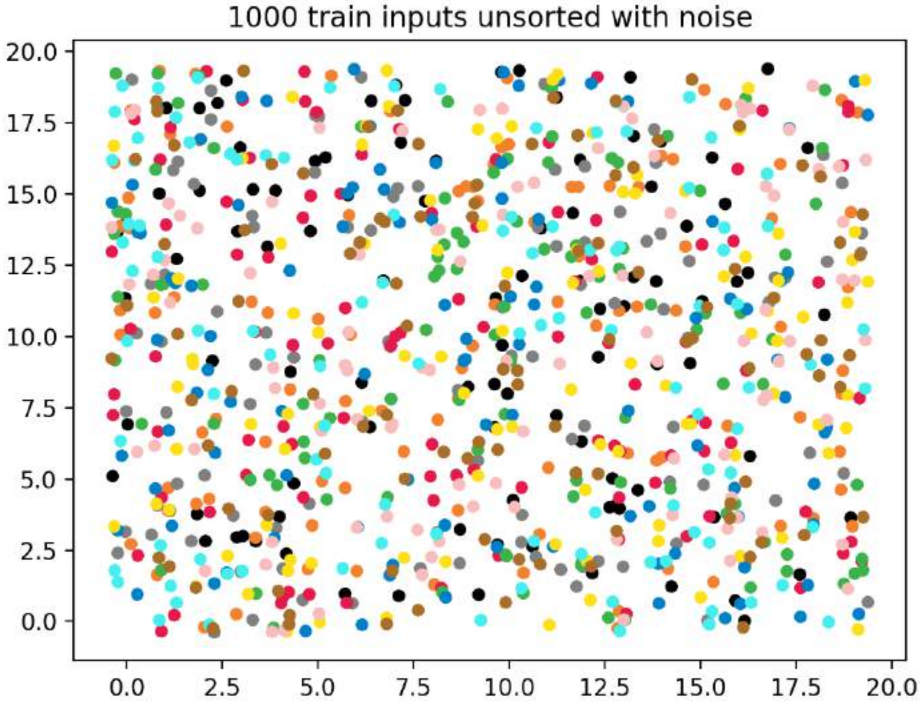


FIGURE H.24: OCR Plot 4

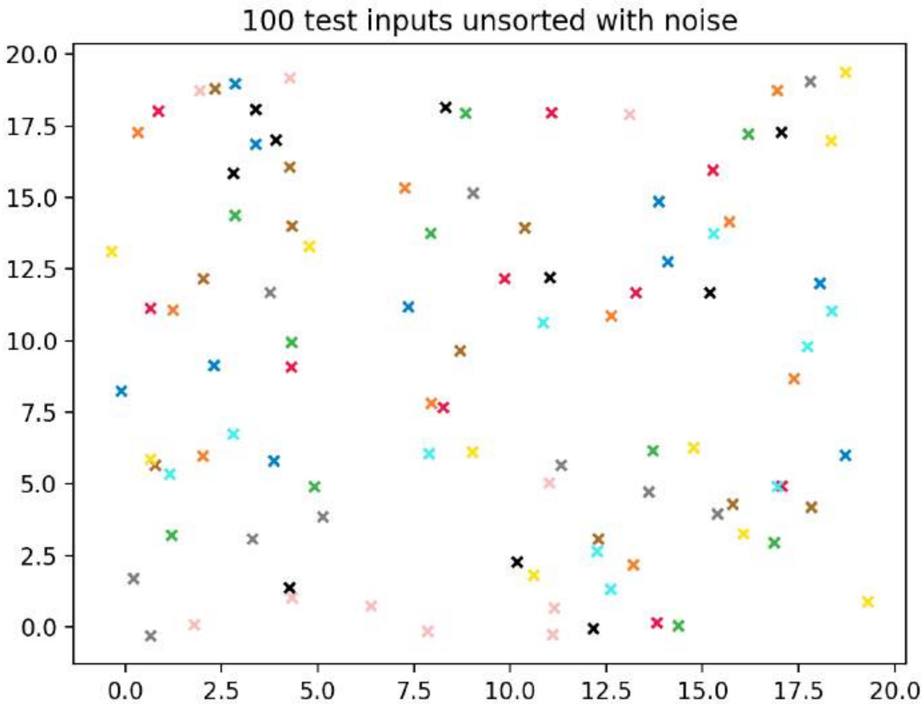


FIGURE H.25: OCR Plot 5

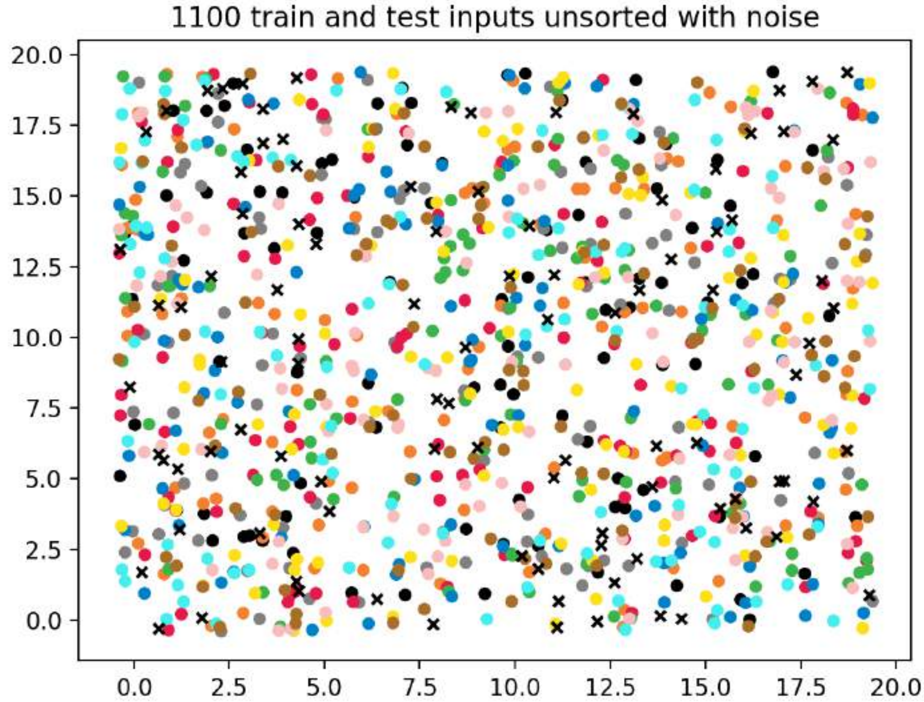


FIGURE H.26: OCR Plot 6

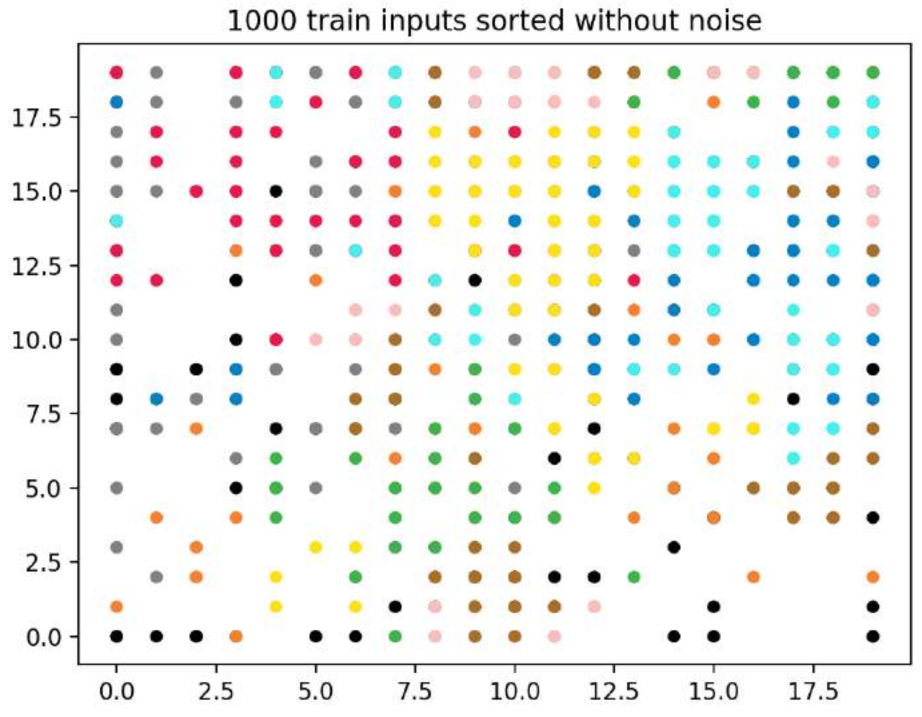


FIGURE H.27: OCR Plot 7

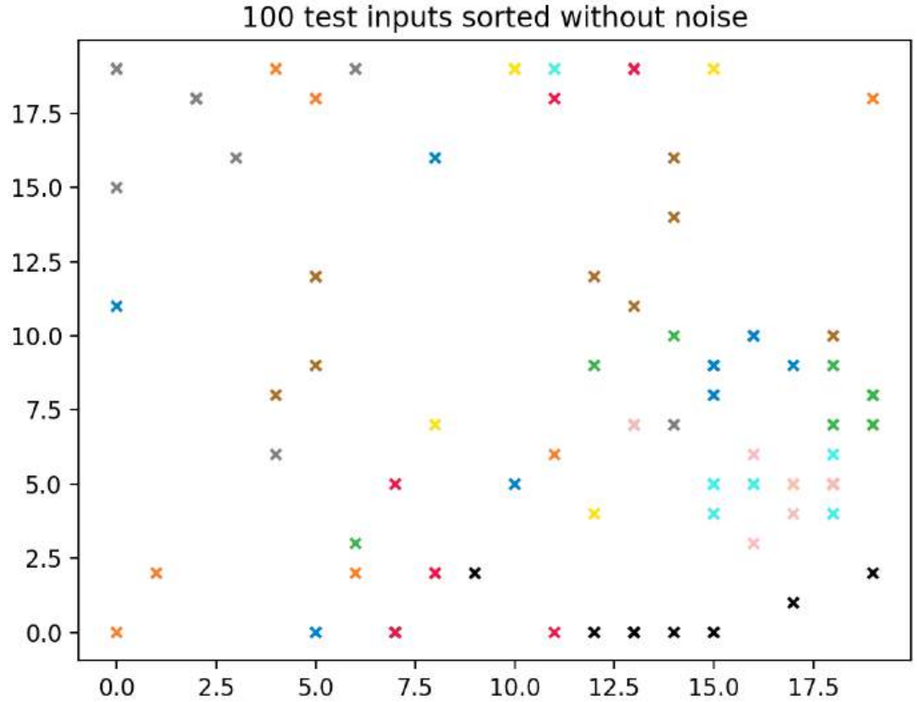


FIGURE H.28: OCR Plot 8

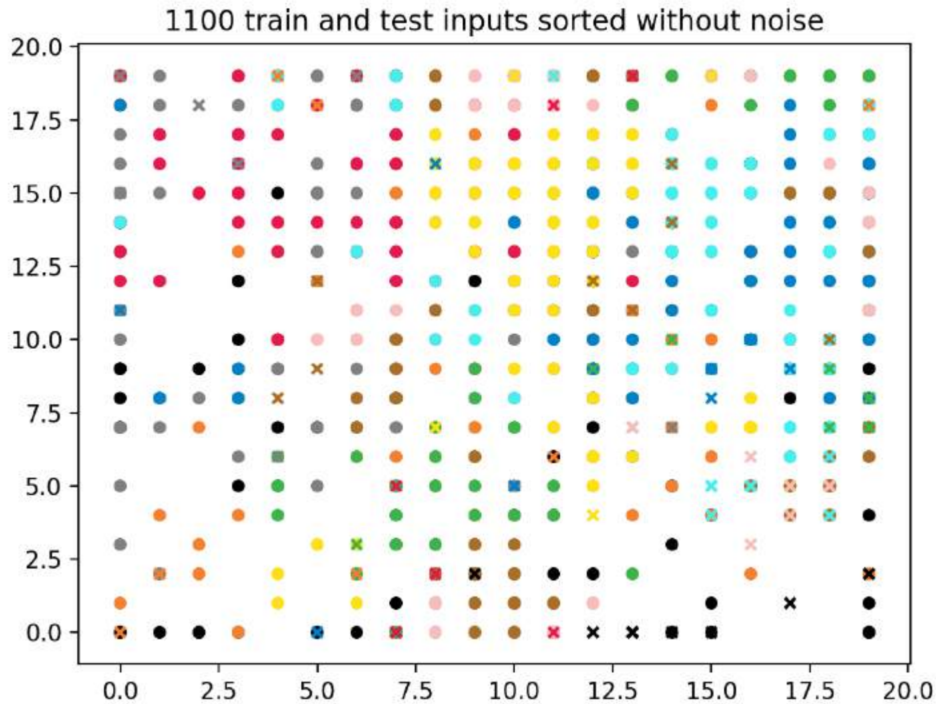


FIGURE H.29: OCR Plot 9

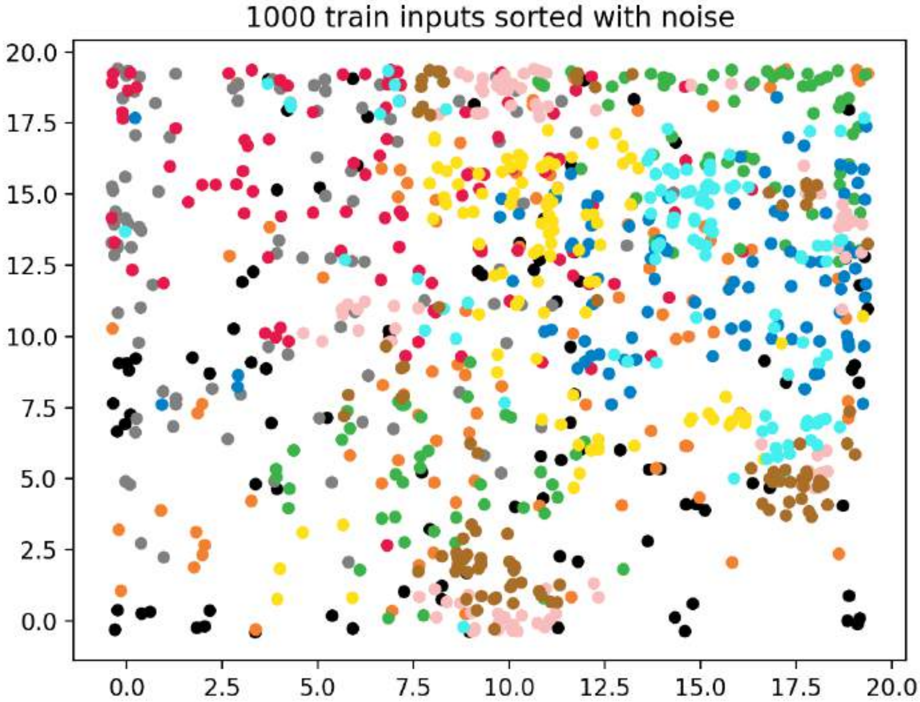


Figure H.30: OCR Plot 10

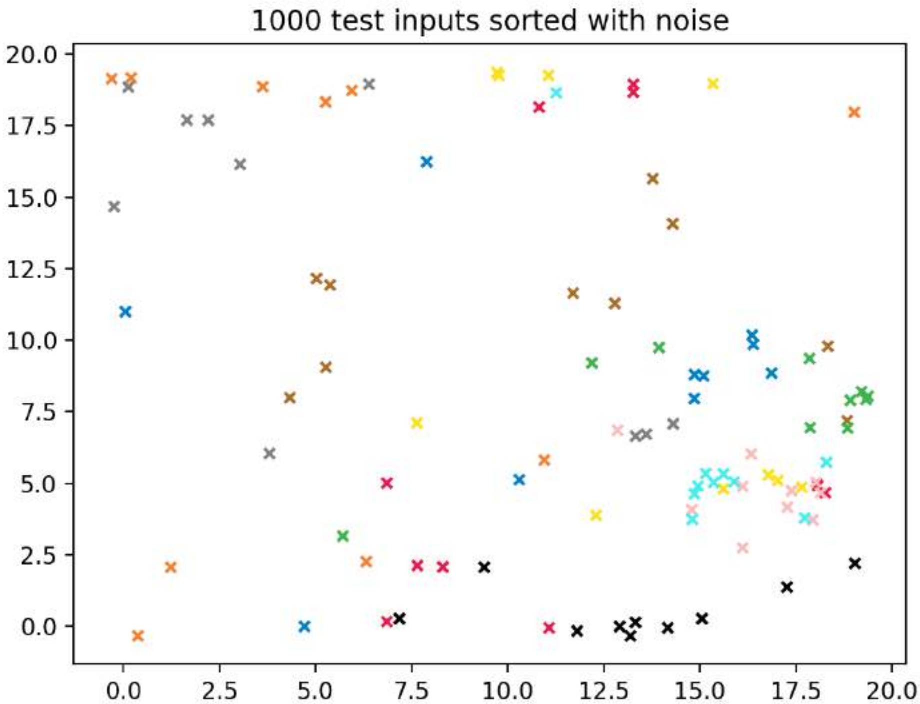


Figure H.31: OCR Plot 11

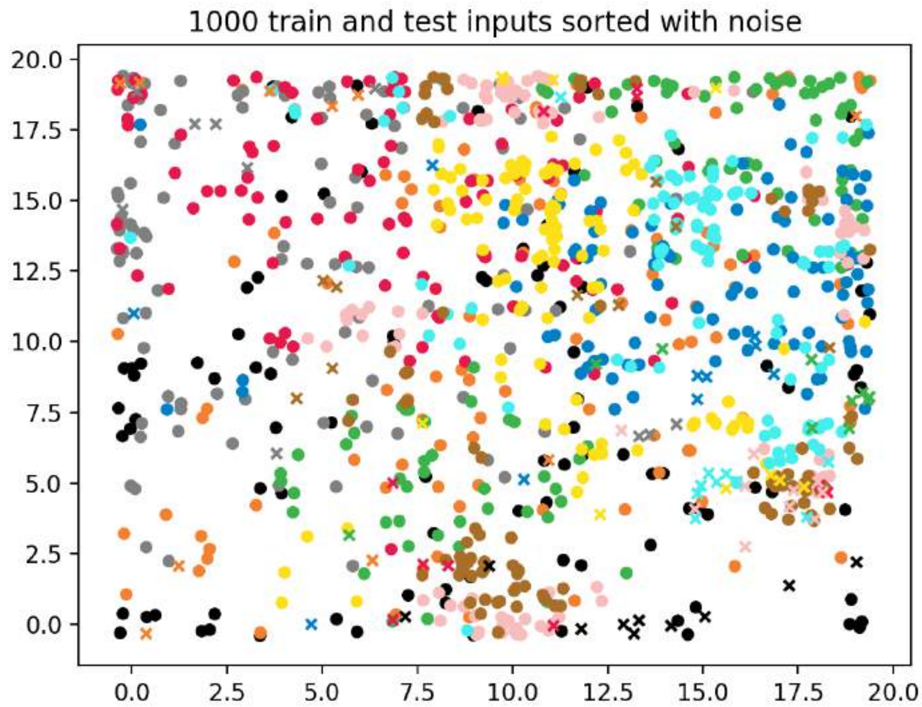


FIGURE H.32: OCR Plot 12

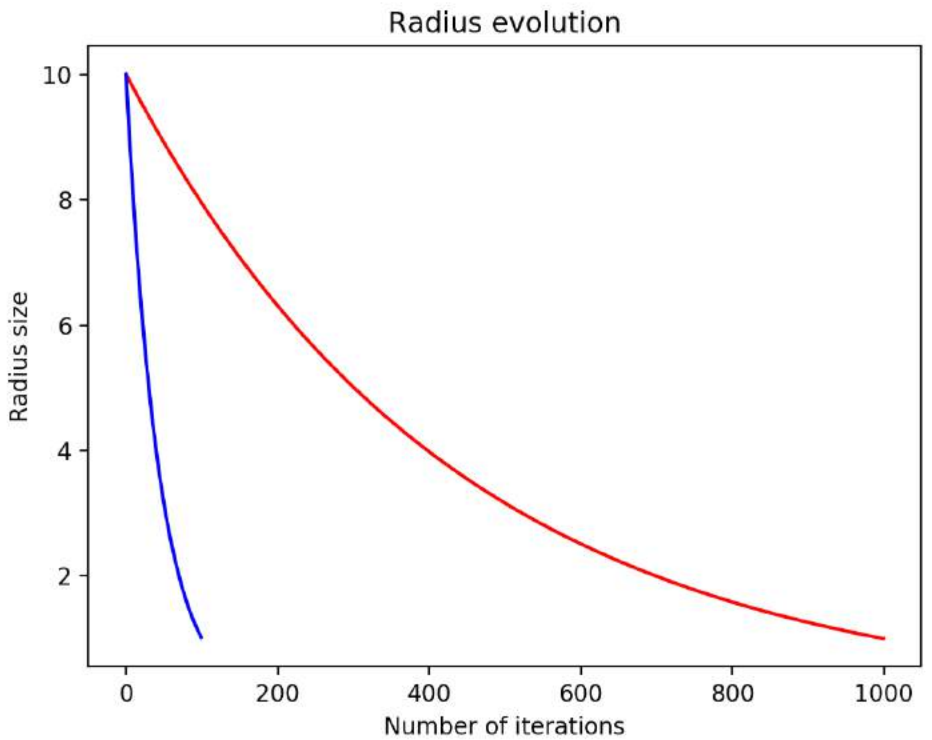


FIGURE H.33: OCR Plot 13

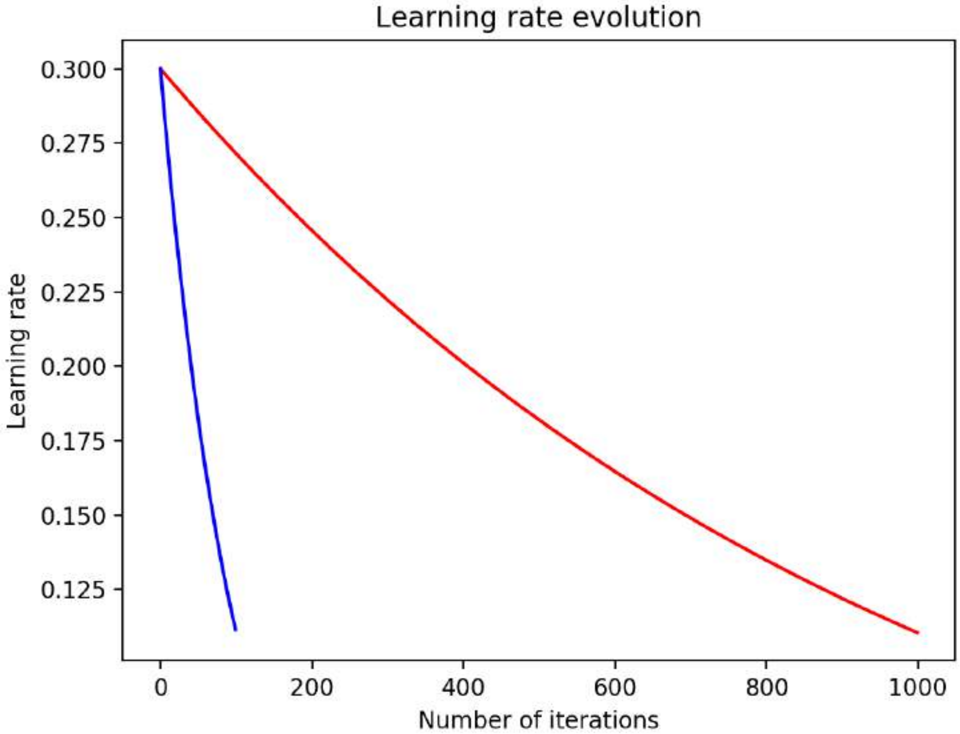


FIGURE H.34: OCR Plot 14

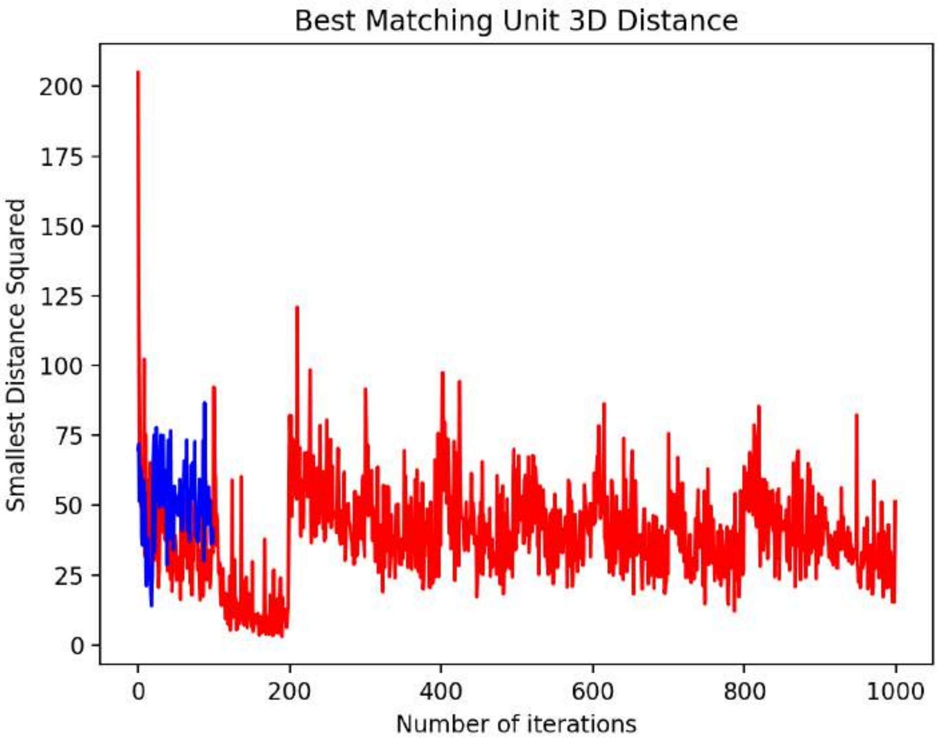


FIGURE H.35: OCR Plot 15

Bibliography

[1] Artificial neural network - wikipedia. https://en.wikipedia.org/wiki/Artificial_neural_network. (Accessed on 11/16/2017).

[2] Bootstrap. https://getbootstrap.com/. (Accessed on 04/02/2018).

[3] Cluster analysis - wikipedia. https://en.wikipedia.org/wiki/Cluster_analysis. (Accessed on 11/16/2017).

[4] The emnist dataset | nist. https://www.nist.gov/itl/iad/image-group/emnist-dataset. (Accessed on 11/16/2017).

[5] Emnist (extended mnist) | kaggle. https://www.kaggle.com/crawford/emnist. (Accessed on 04/25/2018).

[6] Machine learning | coursera. https://www.coursera.org/learn/machine-learning/. (Accessed on 11/16/2017).

[7] Matplotlib 2.2.2 documentation. https://matplotlib.org/. (Accessed on 04/02/2018).

[8] The neural network zoo - the asimov institute. http://www.asimovinstitute.org/neural-network-zoo/. (Accessed on 04/26/2018).

[9] Numpy and scipy documentation. https://docs.scipy.org/doc/. (Accessed on 04/02/2018).

[10] Pattern recognition - wikipedia. https://en.wikipedia.org/wiki/Pattern_recognition. (Accessed on 04/25/2018).

[11] Pattern recognition (psychology) - wikipedia. https://en.wikipedia.org/wiki/Pattern_recognition_(psychology). (Accessed on 04/25/2018).

[12] Self-organizing map - wikipedia. https://en.wikipedia.org/wiki/Self-organizing_map. (Accessed on 11/16/2017).

[13] Som tutorial. http://www.ai-junkie.com/ann/som/som1.html. (Accessed on 11/16/2017).

[14] Asboth, D. Self-organising maps: An introduction. http://davidasboth.com/2016/11/05/self-organising-maps-an-introduction/. (Accessed on 04/02/2018).

[15] Cohen, G., Afshar, S., Tapson, J., and Van Schaik, A. EMNIST: an extension of MNIST to handwritten letters. *CoRR abs/1702.05373* (2017).

[16] Dheeru, D., and Karra Taniskidou, E. UCI machine learning repository, 2017.

[17] Eichner, H. Neural net for handwritten digit recognition in javascript. http://myselph.de/neuralNet.html. (Accessed on 11/16/2017).

[18] Fiesler, E., and Beale, R., Eds. *Handbook of Neural Computation*. Oxford University Press, 1997.

[19] H. Ott, B. A convergence criterion for self-organising maps. Master's thesis, University of Rhode Island, 2012.

[20] Jongejan, J., Rowley, H., Kawashima, T., Kim, J., and Fox-Gieg, N. Quick, draw! https://quickdraw.withgoogle.com/. (Accessed on 11/16/2017).

[21] Kohonen, T. *Self-Organizing Maps*, 3rd ed. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[22] Kohonen, T. Essentials of the self-organizing map. *Neural Networks 37* (2013), 52 – 65. Twenty-fifth Anniversay Commemorative Issue.

[23] LeCun, Y., Cortes, C., and Burges, C. Mnist handwritten digit database. http://yann.lecun.com/exdb/mnist/. (Accessed on 11/16/2017).

[24] Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[25] Olah, C. Visualizing mnist: An exploration of dimensionality reduction. http://colah.github.io/posts/2014-10-Visualizing-MNIST. (Accessed on 11/16/2017).

[26] Ronacher, A. Flask (a python microframework). http://flask.pocoo.org/. (Accessed on 11/16/2017).

[27] Russell, S. J., and Norvig, P. *Artificial Intelligence: A Modern Approach*, 2 ed. Pearson Education, 2003.

[28] TensorFlow. Mnist for ml beginners. https://www.tensorflow.org/get_started/mnist/beginners. (Accessed on 10/15/2017).

[29] Wang, Y. Artificial neural networks: Kohonen self-organizing maps (soms). Bachelor thesis, University of Liverpool, May 2015.

[30] Westerlund, M. L. Classification with kohonen self-organising maps. *Soft Computing, Haskoli Islands* (2005).

[31] Zurada, J. M. *Introduction to Artificial Neural Systems*. West Publishing Co., St. Paul, MN, USA, 1992.