Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Bachelor's diploma thesis

in the field of study Computer Science

Handwriting synthesis with the help of machine learning

Martin Mrugała

student record book number 290551

Patryk Walczak

student record book number 290567

Bartłomiej Żyła

student record book number 290570

thesis supervisor

Agnieszka Jastrzębska, Ph.D. Eng.

WARSZAWA 2021

....

supervisor's signature author's signature

Abstract

Handwriting synthesis with the help of machine learning

This engineering work aimed to create a program that imitates handwriting, based on a dataset consisting of handwritten text images. For this reason, the project contains two main parts.

The first is responsible for processing the input image, recognising the handwriting and extracting the individual characters. The recognition module is based on an existing mechanical handwriting recognition model, modified to recognise handwriting. A collection containing examples of different handwriting styles has been used to achieve this.

The second, much more extensive part of the work is responsible for mapping the appropriate handwriting style. It is based on the results of the first module. From the extracted images, skeletons and corresponding sequences of control points are created. This data is then processed and combined in various configurations to create a new, correct skeleton. The resulting letter base is different from all input data while retaining its unique character. Then comes the final stage of synthesis. The new letters' final version is created based on the skeletons created and using the previously trained model. The only thing left to do is to combine letters into words and words into sentences.

For the convenience of use, a graphic interface has also been created, allowing for an easy and clear way not only to recognise the text in a selected photo or create a synthesis of a letter but also to interfere with these processes or create a new style of imitation. By inputting photos of the script, the user can create their font, and thanks to extensive parameterisable options, they can improve the results. Of course, the program is configurable, and the authors have left the door open for entering other models.

Keywords: handwriting synthesis, text recognition, machine learning, image processing, graphic interface

Streszczenie

Syntezowanie pisma odręcznego za pomocą uczenia maszynowego

Celem pracy inżynierskiej było stworzenie programu imitującego pismo odręczne, bazującego na zbiorze danych składających się ze zdjęć zawierających ręcznie pisany tekst. Z tego powodu projekt zawiera dwie główne części.

Pierwszą odpowiedzialną za przetwarzanie wprowadzanego obrazu, rozpoznanie pisma i wyodrębnienie poszczególnych znaków. Moduł rozpoznawania jest oparty na istniejącym modelu rozróżniającym pismo mechaniczne, który następnie został zmieniony, żeby rozpoznawał pismo odręczne. Do osiągniecia takiego stanu został wykorzystany zbiór zawierający przykłady różnych styli pisma.

Natomiast druga dużo bardziej rozbudowana cześć pracy jest odpowiedzialna za odwzorowanie odpowiedniego stylu pisma. Bazuje ona na wynikach pierwszego modułu. Z wyodrębnionych zdjęć tworzone są szkielety oraz odpowiednie sekwencje punktów kontrolnych. Tak przygotowane dane są następnie odpowiednio przetwarzane, łączone w różnych konfiguracjach aby stworzyć poprawny nowy szkielet znaku. Powstała w taki sposób podstawa litery jest różna od wszystkich wprowadzonych danych, a jednocześnie zachowuje ich unikalny charakter. Potem następuje ostatni etap syntezy. Za pomocą wcześniej wytrenowanego modelu, bazując na powstałych szkieletach tworzona jest ostateczna wersja nowych liter. Pozostaje jedynie połączyć litery w słowa, a słowa w zdania.

Dla wygody użytkowania został stworzony także interfejs graficzny, pozwalający w łatwy i czytelny sposób nie tylko rozpoznawać tekst na wybranym zdjęciu, czy tworzyć syntezę pisma, ale także ingerować w te procesy czy stworzyć nowy styl imitacji. Wprowadzając zdjęcia pisma można stworzyć własną czcionkę, a dzięki rozbudowanym opcjom parametryzacji można poprawić wyniki. Oczywiście program jest konfigurowalny i autorzy pozostawili wolną furtkę, aby wprowadzić własne modele.

Słowa kluczowe: rozpoznawanie tekstu, synteza pisma, pismo ręczne, uczenie maszynowe, przetwarzanie obrazu, interfejs graficzny

Warsaw, ...

Declaration

I hereby declare that the thesis entitled „Handwriting synthesis with the help of machine learning", submitted for the Engineer degree, supervised by Agnieszka Jastrzębska, Ph.D. Eng., is entirely my original work apart from the recognized reference.

.....

Contents

1. Introduction

Nowadays, handwriting it is used less and less. Letters are mostly replaced by e-mail messages, computations are done by computers or with their help. Everything shows that this process will accelerate rather than release or withdraw. Today, it is easier to type the document using the keyboard, and nobody denies that fact. In such reality, the plain text is not impressive. Everyday peoples read instructions, documentations, advertising or search some information, and all of that is written in the plain, printed text.

Therefore, the occurrence of handwriting is exciting. Getting such a message is very personal, and addressee remarks the uniqueness of nicely written text. Surely, the advertising industry would make frequent use of this because such a practice would interest a potential customer. In many computer games, information is passed on through various types of letters or messages. These are currently written in plain text, but creating them in a handwriting-like fashion could convey additional information. The player would know if the fictional character was in a hurry or had time to write a letter from the first sight. Moreover, there are undoubtedly many more possibilities to use such technology. Millions of people all over the world participate in various classes, conferences, courses, and workshops every day. The participants of such events typically take notes. Even though electronic devices supersede a pen and a sheet, the ink remains the first choice for many people. The handwriting synthesis allows joining these two approaches. The notes are presented as handwriting, whereas the author may use a handy mobile device.

In this thesis project, we have tried to create a solution that is as convenient for the user as possible and at the same time, produces the best possible result. There already exist several applications which create handwritten text like *Handwriting.io* or *My Text in Your Handwriting*, but they have some drawbacks. The main flaws are that they were either created in several years or are paid high above university standards. We wondered whether it was possible to produce such software on university terms during a single semester and without huge investments.

Taking into account all the information mentioned above, this work describes the process of creating a program that allows the user to enter his photo with the text in order to recognise its content - faster and easier to transfer to computer text, but also to create a new style on its basis with the help of which any text can be generated.

1.1. Main steps

To create this project many algorithms, additional packages, and two models using machine learning have been used, but they can be divided into two parts to explain the whole process.

The first one is responsible for recognising text from a photo. In this part, there are many algorithms for processing dataset and photos entered by the user. All these functions are described below, as well as the whole process of training the model.

The second, more complex one, is responsible for imitating handwriting. It is done using extracted letters from the base image, which are then skeletonized, and control points are created on their basis. Then, using the base skeleton, the correct order of connecting the resulting points is searched for. This process is followed by the creation of a new skeleton using the control points created on the basis of the dataset. This skeleton is then used by the model trained earlier which applies a layer imitating a writing tool. In this way, letters are created, which are combined in the final effect.

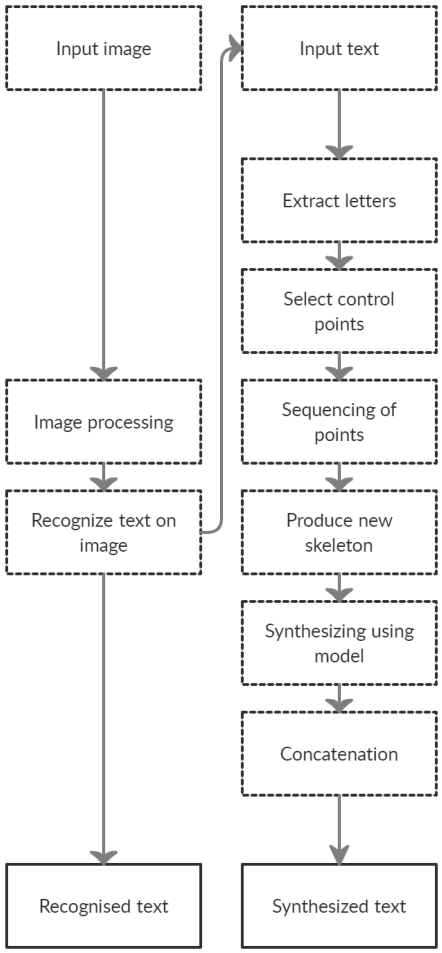


Figure 1.1: Main steps

For the convenience of use, a graphical interface has been created, which allows the use of all the created options in a user-friendly way.

The following sections describe the theory behind every part of the project, the implementation process and the results complemented with a few ideas for further improvement.

1.2. Organization of work

The division of work between individual team members was as follows:

Martin Mrugała:

- Training recognition model
- Implementing algorithms for handwriting reconstruction
- Implementing algorithms for concatenation functions
- Preparing algorithms for finding control points
- Preparing matching algorithm
- Preparing algorithms for filtering control points

Patryk Walczak:

- Preparing the environment
- Processing of the dataset
- Implementing algorithms for handwriting reconstruction
- Implementing algorithm for sequencing of control points
- Compilation to executable application
- Preparing graphical user interface
- Preparing pipelines for repository

Bartłomiej Żyła:

- Preparing dataset for recognition model
- Preparing scripts for creating initial dataset for synthesis model
- Implementing synthesis models' functionalities
- Training synthesis models
- Adapting concatenation functions for synthesis models

All authors collaborated in the creation of the work. Every person's main tasks are listed above, but each author contributed to every part of the application and integration of the components.

2. Theory

Many algorithmic techniques and concepts are necessary to understand the overall process used in this work. Therefore, all the theoretical information has been collected and explained in this chapter. It explains the approach to the problems and how to solve them. It also touches on aspects of why such a solution was implemented but does not discuss its implications. The individual implementations of the respective modules are described in the next chapter entitled "Implementation".

2.1. Image processing

Image processing is a branch of computer graphics and digital signal processing. The study field includes all images algorithm such as filtering, segmentation, transformation, comparison and a lot more. In the thesis, only a small part of the set was used. The project is mainly based on machine learning which is common practice to join the trained models with image processing. Others subsections contain a description of each included tool. However, the assumption was to accept only pictures of the text and find the letters by the trained model, and many algorithms only improve the recognition effect. Nonetheless, the synthesis process applies the skeletonization and finding control points.

2.1.1. Segmentation

Image segmentation is a process dividing the input image into parts or finding specified areas. In the case of the project, the dataset images were segmented because they contain two types of text. On the upper part, there is a printed description, while in the lower main section, there is handwritten text. The dataset contains the files with that description. Hence the first part is negligible from the viewpoint of training the recognition model. That was the task of the segmentation function - pull out only the handwritten text.

We must not forget that segmentation plays a significant role in the process of text recognition, and thus also in synthesis. The module responsible for text recognition divides the processed material into letters used in the second part of the project. It is from the process of image segmentation that the synthesis begins.

2.1.2. Thresholding

Thresholding is one of the simplest methods of image segmentation. It is primarily used to transform an image in grayscale format into a binary image. The most basic form of thresholding is changing each pixel into pure black or pure white, based on whether its intensity value is below or above some fixed constant [19].

$$I_{i,j} = \begin{cases} 255 & \text{if } I_{i,j} > T \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

$I_{i,j}$ - intensity value of pixel in i-th row and j-th column,

T - thresholding constant

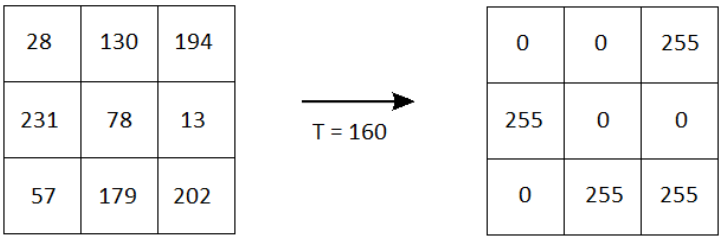


Figure 2.1: Example of the thresholding procedure for a given matrix

2.1.3. Skeletonization

Every handwritten character has a unique shape even though some of them perceptually seem to be identical. The synthesis requires to perform dozens of procedures on numerous characters' images. Their shape features are exploited for symbols creation which is why the consistency of the outcomes is crucial. Hence, the skeletons of the figures should form the basis for further processing.

The skeletonization is a process of thinning an object located on an image. The operation itself can be considered as removal of the coloured areas of a figure leaving thin lines. The skeleton preserves the outline of the original object. It is a set of single-pixel width lines which may be connected. The points belonging to it are equidistant to the edges of the original figure.

In order to skeletonize an image, a thinning algorithm is applied. There exist two main types, as well as their subtypes of thinning algorithms - iterative and non-iterative. The Figure 2.2 shows the depicts the entire hierarchy.

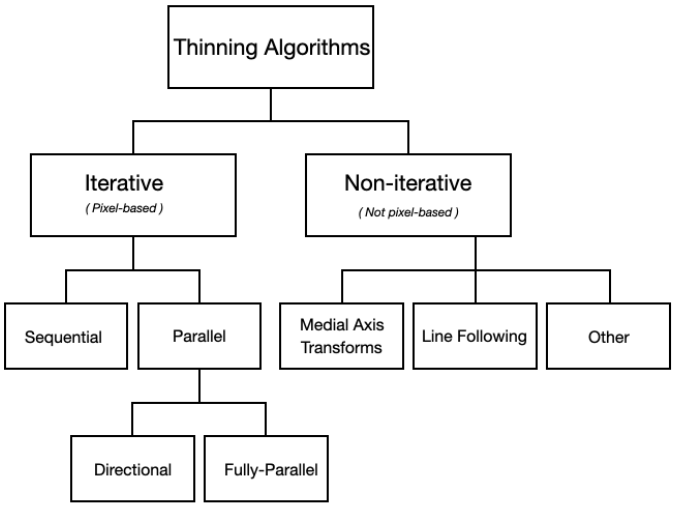


Figure 2.2: Types of thinning algorithms [12]

The following algorithms belong to the group of popular thinning algorithms [13]:

- Gou-Hall algorithm
- Rutovitz algorithm
- Zhang–Suen algorithm
- Lee-Kashyap-Chu algorithm

Implementations of the last two algorithms are used for creating skeletons in the project. Zhang-Suen algorithm concerns 2D pictures, whereas Lee-Kashyap-Chu algorithm deals with 3D images. Both the Zhang–Suen method [15] and the Lee-Kashyap-Chu method [4] are clarified based on the highlighted articles.

Zhang–Suen algorithm

In the case of images, two-dimensional arrays may represent only grayscale images and binary images. For example, RGB images have three channels, which means there is a need for a 3D array to store them in memory. The Zhang–Suen algorithm is suitable for 2D arrays with possible values 0 and 1.[1] Hence, it applies the thinning to binary images. The three channels will be discussed in the Lee-Kashyap-Chu algorithm section.

[1] In the other part of this section, it is assumed without loss of the generality that white colour represents the value 0 while the black colour stands for the value 1.

The Zhang-Suen algorithm is an iterative, parallel thinning algorithm. In every iteration, the algorithm verifies if a black pixel's value should be set to zero. In this way, all pixels belonging to the shape, yet not being in the skeleton, are removed. The decision is made based on the concerned pixel's and its eight neighbours' values. They are ordered in the manner presented in the Figure 2.3.

P_9 $(i-1, j-1)$	P_2 $(i-1, j)$	P_3 $(i-1, j+1)$
P_8 $(i, j-1)$	P_1 (i, j)	P_4 $(i, j+1)$
P_7 $(i+1, j-1)$	P_6 $(i+1, j)$	P_5 $(i+1, j+1)$

Figure 2.3: Pixels' labels [15]

The algorithm is parallel. In every iteration, the computations are performed on the initial, per each iteration, values of the pixels. It allows taking advantage of the multiprocessing. The values are updated at the end of every repetition.

The iteration itself is divided into two parts. The first one gets rid of pixels residing on the shapes' bottom or right edges, as well as top-left corners pixels, while the second one, on the contrary, zeros the pixels located on the shapes' top or left edges, as well as bottom-right corner pixels. A pixel's value is zeroed if all conditions from at least one part are satisfied. The conditions are as follows:

Part I:

1) $2 \leqslant B(P_1) \leqslant 6$
2) $A(P_1) = 1$
3) $P_2 \cdot P_4 \cdot P_6 = 0$
4) $P_4 \cdot P_6 \cdot P_8 = 0$

Part II:

1) $2 \leqslant B(P_1) \leqslant 6$
2) $A(P_1) = 1$
3) $P_2 \cdot P_4 \cdot P_8 = 0$
4) $P_2 \cdot P_6 \cdot P_8 = 0$

where

- $A(P_1)$ is the number of occurrences of the $\{0, 1\}$ sequence in the sequence $\{P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9\}$
- $B(P_1)$ is the number of neighbours of P_1 with value 1 i.e.

$$B(P_1) = P_2 + P_3 + P_4 + P_5 + P_6 + P_7 + P_8 + P_9$$

Conditions 1 and 2 are the same. The first one prevents deleting endpoints. In turn, the second one ensures that a point between the endpoints cannot be zeroed, as shown in the Figure 2.4.

Figure 2.4: Deletion of the endpoints [15]

In the case of conditions 3 and 4, they differ slightly. As mentioned above, each part is responsible for dealing with appropriate types of black pixels in given directions. While considering the first part, these two equations must be satisfied:

3) $P_2 \cdot P_4 \cdot P_6 = 0$

4) $P_4 \cdot P_6 \cdot P_8 = 0$

This implies that $P_4 = 0 \vee P_6 = 0 \vee (P_2 = 0 \wedge P_8 = 0)$. Assuming that only P_4 is 0, P_1 is in a right edge pixel. Similarly, if P_6 is 0, then P_1 is in a bottom edge. The case when both P_2 and P_8 are zeros indicates that P_1 is a corner pixel. These cases are explained graphically in the Figures 2.5, 2.6, 2.7.

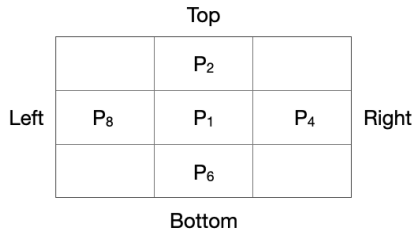


Figure 2.5: Points and their locations [15]

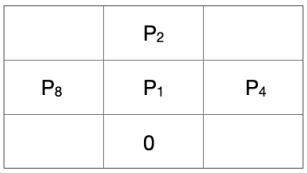


Figure 2.6: $P_6 = 0$. This is the case of bottom edge pixel [15]

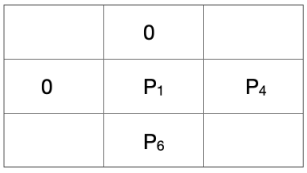


Figure 2.7: $P_2 = 0 \wedge P_8 = 0$. This is the case of top-left corner pixel [15]

Lee-Kashyap-Chu algorithm

As suggested in the previous section, this algorithm deals with three-dimensional images. In this case, the skeleton consists of points equidistant from the edges. Therefore, these points are located in the solid's interior. During skeletonization, special attention should be paid to the sustainability of the topological properties.

The image is represented by a three-dimensional array with values 0, 1. Each point may have 6 or 26 neighbours depending on the assumptions.

- 6-connected neighbours
  $N_6(v = (x, y, z)) = \{(a, b, c) : |a\ - x| + |b - y| + |c - z| = 1\}$
- 26-connected neighbours
  $N_{26}(v = (x, y, z)) = \{(a, b, c) : max(|a\ - x|, |b - y|, |c - z|) = 1\}$

These three conditions have to be satisfied in order to delete a point:

- This is a simple point - after deletion of this point all the connections between it and its 26-connected neighbours remain
- The Euler characteristics does not change after the deletion
- The point has a 26-connected neighbour

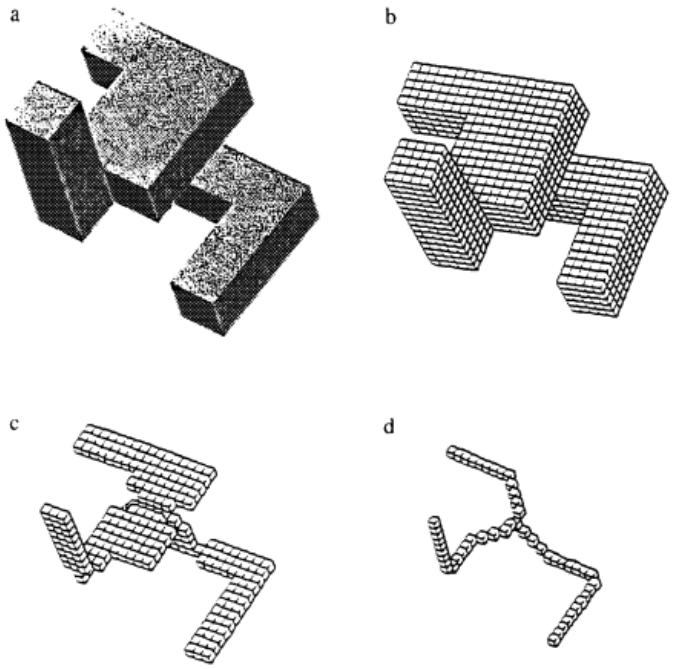


Figure 2.8: The result of a skeletonization using Lee-Kashyap-Chu algorithm [4]

2.1.4. Control Points

An extracted skeleton may consist of a tremendous number of pixels. Therefore there is a need for reducing the number of those pixels. This selection cannot be accidental. The set of the points left, namely control points, should preserve the original figure's shape. It means that an object formed by appropriately connecting these points should be as akin to the original object as possible. The most common methods to single out control points are:

- Distance Transform
- Gabor Filter

The application of Gabor filter produces satisfactory results. For images the two-dimensional variant is suitable.

Gabor filter

It investigates if there exist any content arranged in a given direction. That means that it belongs to the set of linear filters and more precisely to the band-pass filters. Thus, it is sensitive to orientation filter, which is frequently used for texture analysis [5].

The filter itself is tied to the human visual system. It is proven that there exist two-dimensional Gabor filters which match both the spatial and spectral response profiles of perceptive fields in the visual space of the human's visual system [9]. A dataset with information about pen's shifts in horizontal and vertical directions against time while handwriting, one-dimensional Gabor filter may be used for handwriting synthesis. It consists of complex sinusoid modulated by Gaussian function.

$$G(x, \omega, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{\frac{-x^2}{2\sigma^2}} \cdot e^{j(\omega\ x)} \tag{2.2}$$

The intuitive idea of the two-dimensional version is shown in the Figure 2.9. In case of the spatial filter, the kernel is created. The assumptions remain the same.

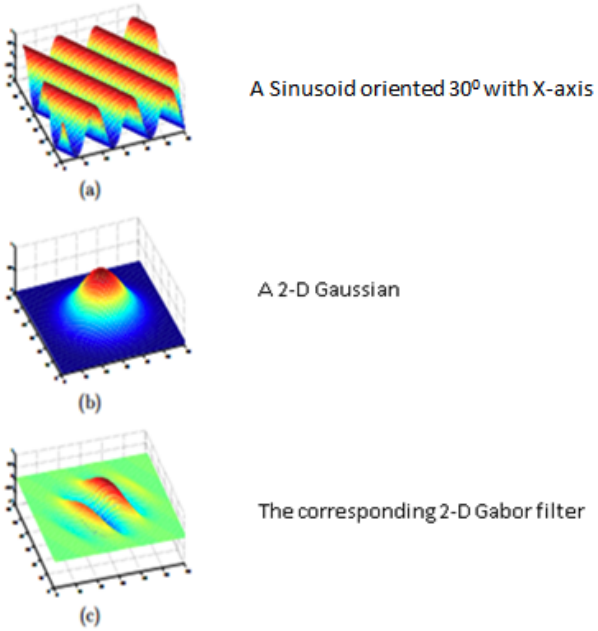


Figure 2.9: Gabor filter as result of multiplication of Gaussian function and complex sinusoid [1]

The formula is extended by the second dimension yielding the following equations [5]:

$$G(x, y; \lambda, \theta, \psi, \sigma, \gamma) = exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2} \right) exp\left(i\left(2\pi \frac{x'}{\lambda} + \psi \right) \right) \tag{2.3}$$

Real

$$G(x, y; \lambda, \theta, \psi, \sigma, \gamma) = exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2} \right) \cos\left(2\pi \frac{x'}{\lambda} + \psi \right) \tag{2.4}$$

Imaginary

$$G(x, y; \lambda, \theta, \psi, \sigma, \gamma) = exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2} \right) \sin\left(2\pi \frac{x'}{\lambda} + \psi \right) \tag{2.5}$$

where

$x' = x \cos\theta \ + y \sin\theta$

$y' = -x \sin\theta \ + y \cos\theta$

λ - wavelength of the sinusoidal factor

θ - orientation of the normal to the parallel stripes of a Gabor atom

ψ - phase offset

σ - standard deviation of the Gaussian envelope

γ - spatial aspect ratio

The Figure 2.10 reveals a set of Gabor filters. The wavelength - λ - is related to the thickness of the filter. θ, in turn, rotates the filter. γ is responsible for changing the height of the filter. σ waves the filter, which results in changing the number of bands. The influence of these parameters is shown in the Figures 2.11, 2.12, 2.13, 2.14.

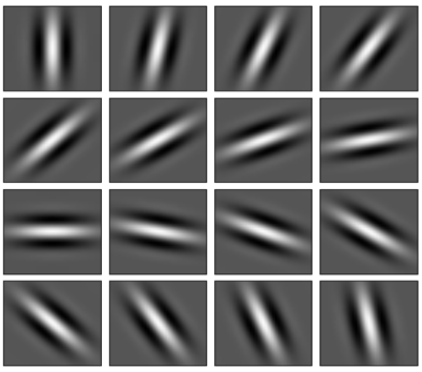


Figure 2.10: a set of exemplary Gabor filters [1]

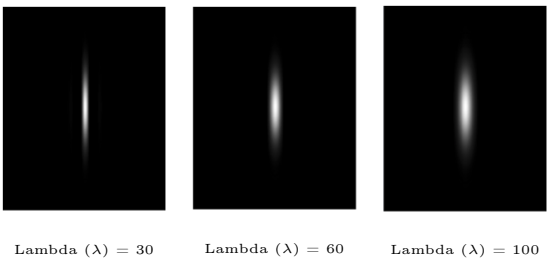


Figure 2.11: Impact of the λ parameter on a Gabor filter [1]

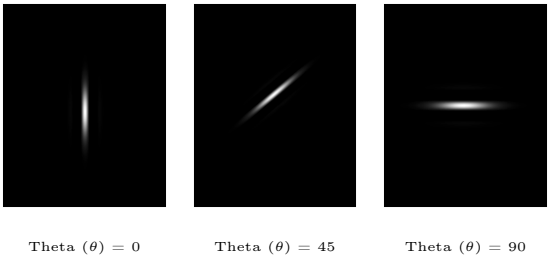


Figure 2.12: Impact of the θ parameter on a Gabor filter [1]

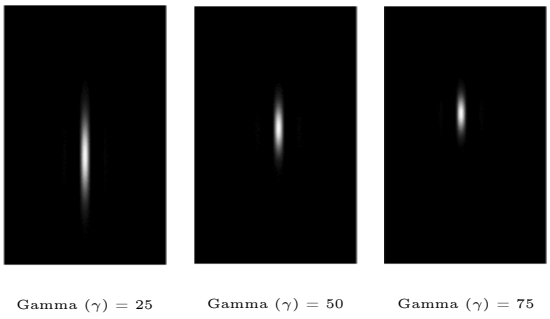


Figure 2.13: Impact of the γ parameter on a Gabor filter [1]

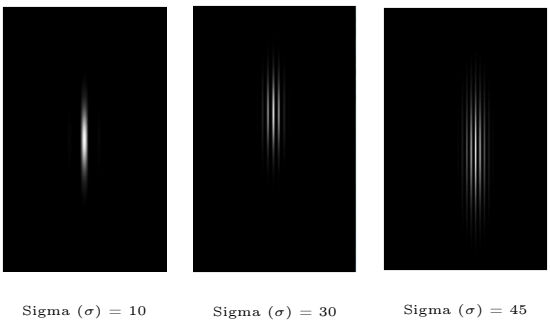


Figure 2.14: Impact of the σ parameter on a Gabor filter [1]

The Gabor function consists of two functions residing in real and imaginary parts of the complex function. Hence, the application of the Gabor kernel to an image results in creating two filtered images.

2.2. Optical character recognition

An optical character recognition (OCR) system deals with the transformation of a text image into its corresponding textual representation. In the case of this work, we will focus on the handwritten text, but it can also be printed text. This process can be divided into five main sub-processes:

- image processing
- finding text in a photo
- letter segmentation
- letter recognition
- post-processing

Of course, the list of subprocesses may vary depending on their techniques, but this is their basis. Optical letter recognition systems aim to identify and detect unique words in each text and for each language. However, this requires a large enough dataset and time to train the model. This thesis focuses only on the English language. Everything, including results, is discussed for this language. The use of any other language can result in a substantial degradation of the recognition module.

2.2.1. Tesseract OCR

The development of the Tesseract OCR was initiated at Hewlett Packard labs in Bristol. In the years 1985 - 1994, the project has been improved in Greeley. On the eve of the 20th century, in 1996, there appeared a Windows version. Nine years later, it became an open-source software under Apache 2.0 license. Google started to develop it in 2006. Since then nothing changed up to today [16][17].

This OCR can recognize a vast number of languages. This procedure is based on a file with the extension *traineddata*. If there is such need, every user may create a new traineddata file with custom language and symbols. The supported encoding is UTF-8. The output of recognition may be saved as *TXT*, *HTML*, *PDF* or *TSV* file [16].

In the beginning, it was written in C. Later, in 1998, some parts were rewritten to C++. There are wrappers for such languages like C, Java, .NET, Python, C++, Go, Node.js, Ruby, Swift - depending on the version. The newest stable version - 4.1.1 - supports all of them. By default, it is utilized in Command-Line-Interface. The version 4.00 includes as the first the LSTM network.

This results in an extension of the duration of some processes but ensures more accurate results. The architecture of this neural network type will be discussed in the next section. There exist tools for training Tesseract models in case a custom language or set of symbols is needed. There exist three types of learning [7]:

- Fine-tuning - works fine when the new data is almost identical to the trained data. This type may improve minor deficiencies.
- Removal of several layers - it allows deleting a few layers. This option is suitable for learning new languages or fonts when the trained data knows a similar script.
- Training from scratch - as the name suggests, the model may be trained from scratch. It requires a massive dataset.

The training itself may be customized to a large extent. User can train every model. The network specification, caching memory size, the maximal number of iterations, replacement of the head of the network, custom evaluation and other options are available.

2.2.2. Long short-term memory

An artificial neuron is an entity computing value based on a predefined expression. A set of connected artificial neurons is called a neural network. There exist three main classes of neural networks [11]:

- Artificial Neural Network (ANN)
- Convolution Neural Networks (CNN)
- Recurrent Neural Networks (RNN)

Long short-term memory (LSTM) is a Recurrent Neural Network. RNN's can use previous information and adds a new value in every iteration, as shown in the Figure 2.15.

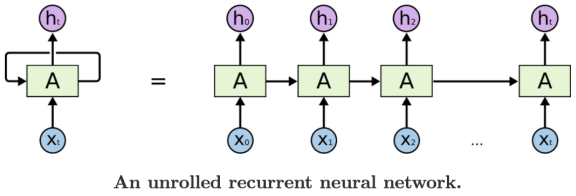


Figure 2.15: The architecture of a Recurrent Neural Network [20]

The problem arises when a piece of information is supplied long before it is needed. In such a case, the network may not manage to select the information. It is presented on the Figure 2.16.

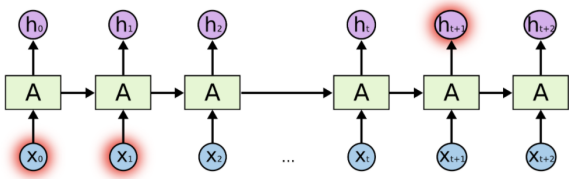


Figure 2.16: A gap between time of inputting information and the time when it is needed [20]

LSTM can remember or forget information allowing at any moment to retrieve any remembered data from the past. The structure of an LSTM cell is shown in the Figure 2.17. Such cells are connected as in a usual Recurrent Neural Network. The elements of the image are as follows:

- Yellow rectangle - layer of a neural network
- Pink circle - arithmetic operation
- Black arrow - transfer of a vector
- Blue circle - input vector
- Purple vector - cell's new state

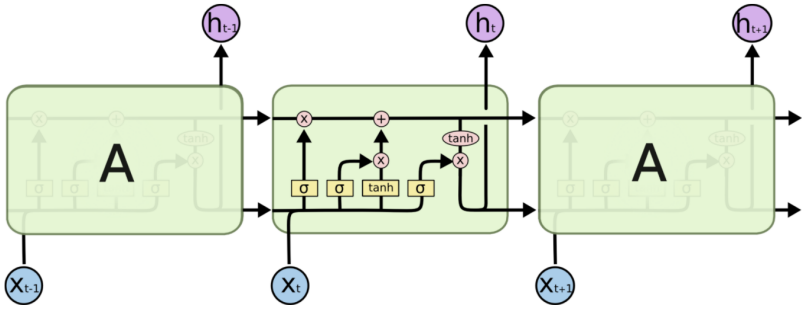


Figure 2.17: The architecture of an LSTM cell [20]

The top horizontal arrow in each cell is the state of the cell. Each cell includes four layers while an ordinary RNN only one. If there is an inscription σ, then the layer is a gate. There are three gates - input gate, forget gate and output gate. The functions assigned to these layers are called sigmoid functions. Wikipedia gives the following example of them [14]:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

This function has to satisfy the condition $S^{-1}[\mathbb{R}] = \langle 0; 1\rangle$. It computes the bandwidth. If S(x) is 0, then no information will be passed through. For S(x) equal to 1, all data will be pushed. The first gate is called forget gate. It computes the f_t value, which determines what part of the initial cell's state should remain. It is done by multiplying C_{t-1} by f_t. The second gate is the input gate. This time the amount of the input data that should be remembered is determined. The tanh layer computes the entire input data's value ($\overline{C_t}$), which could be added to the cell's state. The tanh stands for the hyperbolic tangent. It is applied to the input values, because it has desired properties, namely $\tanh^{-1}[\mathbb{R}] = \langle -1; 1\rangle$. Thus, it normalizes the values, preventing rapid divergence. However, it is multiplied by i_t - the second gate's value - before added to the cell's state. The last - output - gate filters the output data. The cell's state is scaled with tanh and filtered with o_t. This is the output used as the output of the previous iteration in the next one, denoted by h_t. The formulas are like this [20]:

- $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
- $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
- $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$
- $h_t = o_t * \tanh(C_t)$
- $\overline{C_t} = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$
- $C_t = f_t * C_{t-1} + i_t * \overline{C_t}$

where:
- W_f, W_i, W_c are weight matrices
- b_f, b_i, b_o, b_c are biases
- σ is a sigmoid function
- $[h_{t-1}, x_t]$ is a vector containing the output of the previous cell and the input of the current cell, marked as h_{t-1} and x_t, respectively

2.3. Letter Synthesis

Synthesizing handwriting is broadly defined as imitating handwriting in a style that makes it impossible to distinguish between human-made and machine-made. This approach to the original, but not its reproduction, is a fascinating and powerful technology. The proper personalised of the handwriting is a combination of many factors, which the handwriting synthesis field deals with.

However, the handwriting generated in this way should be different, as a human being will not write the same sentence in the same way. This uncertainty, which differentiates each letter from one another and at the same time allows us to find a suitable pattern for recognising their similarity, is extremely difficult to grasp.

2.3.1. Points filtering

Extraction of control points from skeletons of a set of an arbitrary character results in obtaining different numbers of them very often. Two generated letters may then be drawn based on different numbers of control points. It neither breaks down the synthesis nor produces unreliable outcomes. However, control points filtering may eliminate small distortions. The algorithms rely entirely on pure mathematics. There are three approaches of filtering which will be described in this section. All of them divides initially the image into $n \times n$ rectangles of the same size. Exceptions are the borders with maximal indices. For example, if an image's width is not divisible by n, then the last column of cells will be narrower. The division is important in the centre of the image with the greatest concentration of control points. For each cell, the number of control points residing in it is counted.
The control points of an instance of letter "b" are presented in Figure 2.18. The comparison of filter types in next sections is based on this image.

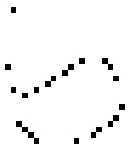


Figure 2.18: The control points of a letter "b" instance

Consecutive method

There is a loop checking if n control points are selected. The control points of consecutive non-empty cells are chosen (The upper-left cells are privileged concerning the order). When there are no more cells with control points, the looping is broken.

Algorithm 1: Consecutive filter

Result: image

Input: n - desired number of control points, image - empty image
cells - image's list of cells and their control points' locations
index - 0

```
1  while n > 0 do
2      if cells[index] is not empty then
3          image[cells[index][0]] ← 1
4          remove cells[index][0]
5          n ← n − 1
6      else
7          min_index ← max(cells)
8          if cells[mix_index] is empty then
9              break
10     index ← index + 1
11     index ← index mod count(cells)
```

In this approach, consecutive control points of consecutive cells are retrieved. It is very fast, but the results are not satisfactory. Usually, only one part of a letter remains (first points are these located in the upper-left part), as shown in the Figure 2.19.

Figure 2.19: Control points from the Figure 2.18
filtered with the consecutive filter ($n = 5$, $k = 10$)

Random method

As in the previous method, the filtering algorithm's essential part is a loop investigating if the requested number of control points is already chosen. If the cell with randomized index is not-empty, then a random control point from it is selected. In case the cell is empty, it should be examined if there are any non-empty cells yet. If not, the looping is broken, because all control points have been filtered. However, after selecting a control point, the cell is marked as visited, and the number of added points is increased by one. These assignments aim to create indirect iterations. If the cell's indices are randomized, control points of one cell may be selected ten times while none will be chosen from another cell. That is why the next index's randomization is done on the range $\langle 0; initial\ \ no.\ \ of\ \ nonempty\ \ cells - no.\ \ of\ \ visited\ \ cells - 1\rangle$. Next, it has to be mapped on the list. When the randomization gives number i, then the next iteration index is of the i^{th} non-empty cell index. When all possible cells are visited, the list is cleared, and the number of visited cells is zeroed.

Algorithm 2: Random filter

Result: image

Input: n - desired number of control points, image - empty image
cells - image's list of cells and their control points' locations
ne_cells - list containing non empty cells
checked - list of ne_cell's size indicating if a cell is already visited
no_checked - number of visited cells
index - random(0, count(checked) - 1 - no_checked)

```
1  while n > 0 do
2      if ne_cells[index] is not empty then
3          random_index = random(0, count(ne_cells) − 1)
4          image[cells[index][random_index][0]]] ← 1
5          remove cells[index][random_index]
6          n ← n − 1
7      else
8          min_index ← max(cells)
9          break if cells[mix_index] is empty
10     update checked[index] and no_checked
11     i ← random(0, count(checked) − no_checked)
12     index ← i^th element of checked with value 0
```

Figure 2.20: Control points from the Figure 2.18
filtered with the random filter ($n = 5$, $k = 10$)

Binary-search method

This method is the fairest but also the most complicated way of filtering. There is a primary loop checking if the number of filtered control points exceeds the requested number. Inside, it is investigated if there are any non-empty cells. If not, the loop is broken. Otherwise, control points from the first and the last cells are selected. The selection of a control point in a cell is random. Again, in each iteration points from every non-empty cell should be chosen. That is why the "halves" list is initialized. It represents the indices of the "cells" list. The addition of a point to the set of selected control points is based on inserting a point from the cell with the index being the midpoint of this list's two adjacent elements, hence the name. The process of filling it with the number of non-empty cells will be called a sub-iteration. The second loop controls if the number of selected points in a subiteration is not equal to the number of non-empty cells or the requested number n. The third loop is broken whenever there is no space to insert an element between two consecutive elements of the "halves" list.

Figure 2.21: Control points from the Figure 2.18
filtered with the binary-search filter ($n = 5$, $k = 10$)

Algorithm 3: Binary-seach filter

Result: image

Input: n - desired number of control points
cells - image's list of cells and their control points' locations
ne_cells - list containing non empty cells
image - empty image

```
1  append one end point if n == 1 and return
2  index ← 2
3  while index < n do
4      compute ne_cells and init halves as empty list
5      break if ne_cells is empty
6      append end points
7      while count(halves) < n and count(halves) < count(ne_cells) do
8          offset ← 0
9          for i in (0; count(halves) − 1) do
10             continue if halves[i] == halves[i + 1] − 1
11             add point offset ← offset + 1
12             state ← state + 1
13             break if state == n
14         break if state == n
```

2.3.2. Matching

This section describes the creation of a new character based on its two instances. As mentioned in the "Control points" section, these points approximate the shape of a figure. Combining two sets of control points of a letter gives a brand new instance preserving the unique style. It is assumed without loss of generality, that one of the figures is the base. First, the figures have to be overlaid. Therefore their centroids are calculated using the balancing method [2]:

$$C = \frac{\sum_{i=1}^{k} x_i}{k} \tag{2.7}$$

where x_i are all control points and k is the number of control points. The centroid of the second figure is shifted to the geometric centre of the base. No affine transformations are needed because the objects on the images have their original sizes.

New points are obtained by iterating through every control point from the base character and choosing the second character's nearest control point. The midpoint for every such couple is computed. The points of a new letter are slightly distorted via randomized shift to make every new instance unique. Hence, the newly generated object will consist of the same number of control points as the base. The nearest point is found with respect to the smallest Euclidean distance. Given two points $P1 = (x_1, y_1), P2 = (x_2, y_2)$ the Euclidean distance is:

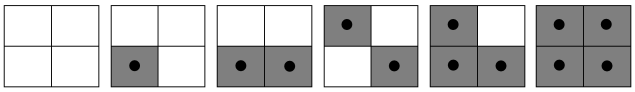
$$d = \sqrt{(\Delta\ x)^2 + (\Delta\ y)^2} \tag{2.8}$$

Figure 2.22: Blue points represent a newly generated letter

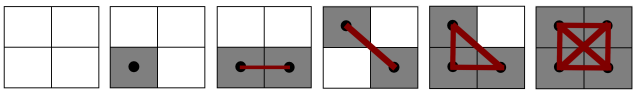
2.3.3. Sequencing

Many papers on handwriting synthesis based on the dataset containing the timeline which is excellent facilitation for letter reconstruction, however, the thesis's goal was to use the text images as the input. After extracting the words from the text and letters from the words, there was no information about how exactly the letter was created, where the line starts, and where the end is. Without the details, the reconstructing algorithm (described in the next section) is ineffectual.

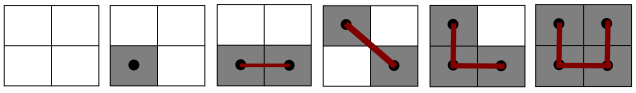
Numerous way was check before the algorithm with the best results was chosen. The algorithm's first step is based on the work entitled *Spatio-Temporal Handwriting Imitation*, where authors start considerations from taking letter as a graph. As the input the algorithm takes a skeleton of a letter (the process is explained in Section 2.1.3 titled **Skeletonization**). The pixels are treated as vertices, and there are six types of such relations, rest are the rotations.



The edges are also needed to generate a graph. So algorithm check which pixels are neighbours and such pair is treated as connected vertices.



The generated graph includes many small undesirable cycles which can produce the problem in the final result. Therefore all the cycles have to be removed. To achieve that function finds in the graph the three and four adjacent vertices. All the additional unnecessary edges are deleted. Below the result of the six basic types is located.



Such a graph may create the letter using the B-spline function described in the next section if the vertices are on proper order. However, that would create a similar letter to the input one.

Hence, there should remain only ordered control points. There is given a collection of point pairs which can be seen as edges. It is transformed into a collection of single points and their number of occurrences. If a point is an endpoint, then its number of occurrences is one. If it connects three strokes, then it is three. The sequences are created in an infinite loop which will be broken whenever there is no more point to append. First, all endpoints are found, and sequences creating strokes starting at these endpoints are generated. After that, strokes originating in points which connect more than two points are processed. The appendage is done by searching points interconnected points in the collection of edges. Every time a point is appended to a sequence, its number of occurrences is decreased by one. A point may be appended to a sequence only if its number occurrences is greater than zero. The result is a collection of sequences representing the strokes.

There exist an alternative version. It includes additional conditions that result in dividing a character into a set of sequences representing strokes consisting of only points (except endpoints) with double occurrences.

The last step is to recreate the sequences leaving only control points. It is important to preserve the order and number of sequences. A simple looping through all sequences and their elements with checking if the element is a control point solves the problem.

2.3.4. Handwriting Reconstruction

Another problematic issue included in our project was the reconstruction of handwriting. New generated letters should be realistic and indistinguishable from the primary dataset. Many weeks of research was done before the most suitable solution was implemented.

In a few words, the reconstruction takes the control points and uses connections to produce new letters, but the algorithm has to work for every letter (at least as many as possible). Our program generates any synthesized text, so considering each letter as an apart case was rejected at the start. At the same point of work, not using machine learning at the synthesis stage was made because of many theses based on that solution. Eventually, in our thesis, the B-spline interpolation is the tool used to connect the control points and generate skeletons of the synthesized letters.

Start from scratch B-spline or basis spline or B-spline curve is mainly described by degree (denote by n) and the number of subsets defined by next parts of the spline (denote by m). Of course, it bases on the sequence of control points also. Evaluating the curve de Boor's algorithm is used, which is a generalization of de Casteljau's algorithm for Bézier curves. Moreover the curve is defined for $t \in [0, 1]$ but $m + 1$ values (denoted by u_i) divide the values for subset for which the curve is defined. The u_i values are named "knots", and they are sorted into nondecreasing order.

The main equation is a linear combinations of control points denoted by p_i and function $B_{i,k}(t)$ given by:

$$p(t) = \sum_{i=1}^{m-n-1} p_i B_i^n(t) \quad \text{for} \quad t \in [u_n, u_{m-n}] \tag{2.9}$$

where:

$m + 1$ - number of knots,

n - degree of the spline,

p_i - i^{th} control point,

$B_i^n(t)$ - the value of basis function for t value.

The basis function is defined as:

$$B_i^0(t) = \begin{cases} 1 & \text{if } u_i \leqslant t \leqslant u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

$$B_i^n(t) = \frac{t - u_i}{u_{i+1} - u_i} B_i^{n-1}(t) + \frac{u_{i+n+1} - t}{u_{i+n+1} - u_{i+1}} B_{i+1}^{n-1}(t) \tag{2.11}$$

That is all essential information about the B-spline, however, there is one more demanding step. The pure B-spline result creates the line between the control points what is inadequate because the points are generated from the letter. They are taken from each letter, to be exact they are on the lines. So the algorithm has to connect the points. It is not enough to produce the line between the points. On the Figure 2.23, there is an example of such inappropriate (for project purposes) effect - computing only the B-spline.

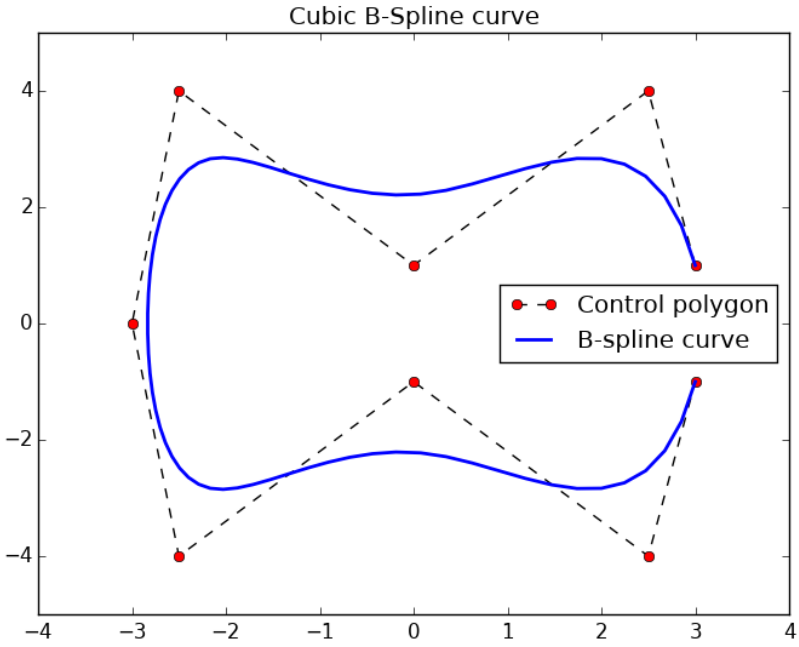


Figure 2.23: B-spline example [10]

Hence, additional step is necessary - move the line to the points. In the algorithm such step is obtained by interpolating the B-spline.

To be exact, polynomial interpolation of function f, where $y_0 = f(x_0)$, $y_1 = f(x_1)$, ..., $y_n = f(x_n)$ are known, searches for the polynomial of degree at most n that meets all the listed points. The general way of solving starts from:

$$\varphi(x) = a_0\varphi_0(x) + a_1\varphi_1(x) + ... + a_n\varphi_n(x) \quad x \in [x_0, x_n] \tag{2.12}$$

Functions $\varphi_0, \varphi_1, ...\varphi_n$ can be chosen, and one example of such selection is mentioned in the next part of the section. Next the function $\varphi(x)$ is denoted by two matrices:

$$\varphi(x) = B(x)A \tag{2.13}$$

$$B = \begin{bmatrix} \varphi_0(x), \varphi_1(x), ..., \varphi_n(x) \end{bmatrix} \quad A = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \tag{2.14}$$

After substitution, the finial equation is as below:

$$\begin{bmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \dots & \varphi_n(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \dots & \varphi_n(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_0(x_n) & \varphi_1(x_n) & \dots & \varphi_n(x_n) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix} \tag{2.15}$$

$$VA = F \tag{2.16}$$

And after performing the matrix transformations on Equation 2.16 and including Equation 2.13 the final equation is presented bellow.

$$\varphi(x) = B(x)V^{-1}F \tag{2.17}$$

As Wikipedia [8] states, the quality of the interpolation is determined by two aspects. The choice of $B(x)$ and selection of points' positions. Moreover, it says that the easiest base is formulated as $B(x) = (1, x, x^2, ..., x^n)$ and the method is named Lagrange interpolation. This is a specific case but the general formula is presented in the upper Equations (2.12 - 2.17).

So after all the steps and computing the result for the same points as in Figure 2.23 is as it was expected. The line meets all the red points.

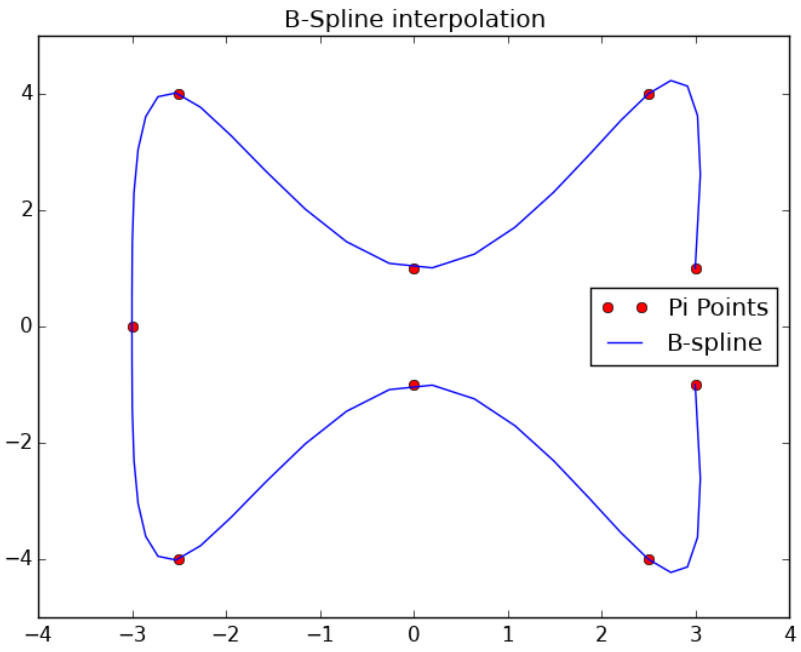


Figure 2.24: Interpolated B-spline example [10]

Such solution is included in the project, but external packages mainly make the computation with properer parameters. It is worth mentioning that the algorithm has to receive the points in the order. The computations are base on that n^{th} point is the predecessor of $n+1^{th}$, and it is the successor of $n-1^{th}$ point. The sorting algorithm of the sequence is described in Section 2.3.3 titled **Sorting**. The handwriting reconstruction outcome without proper sequences is chaotic and is not similar to the letter at all.

2.3.5. Generative Adversarial Network

A generative adversarial network (GAN) is a machine learning framework class with two different neural networks competing in a game. This is a zero-sum game - if one of them is a winner, then the other must be a loser. This network's main idea is for one model called discriminator to distinguish between real input and generated one, while the other named generator to create such output that can fool the discriminator. Both models are dynamically updated throughout the training, making the generator learn in an unsupervised manner. Since both networks are learning from each other, it may lead to potentially far better results than any other approach [6]. This makes generative adversarial networks a proposed solution for image-to-image translation problem present in this project.

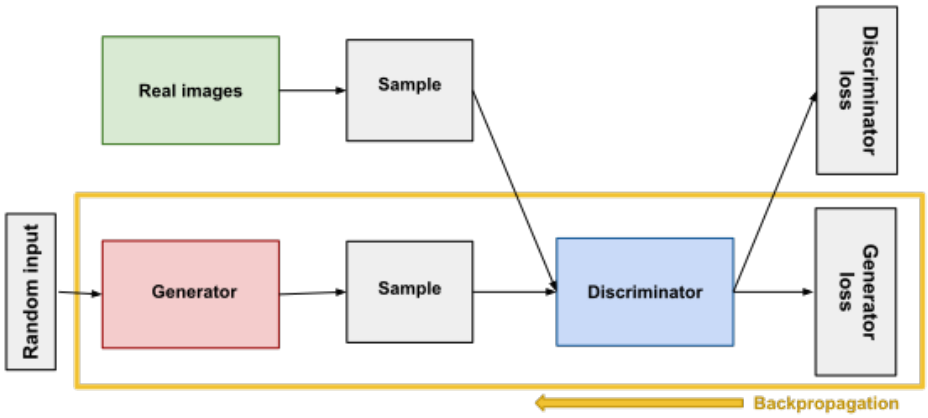


Figure 2.25: Example of GAN Architecture [18]

3. Implementation

In this section, only the descriptions of the production process, the dataset processing, the models learning and the creation of all algorithms are included. To illustrate the sequence of procedures, image 3.1 shows a diagram with the following main steps, also appearing in Section 1.1 titled **Main Steps**. Moreover, it contains the description of the creation of the graphical interface, all tests performed with the results, and the creation of executable files.

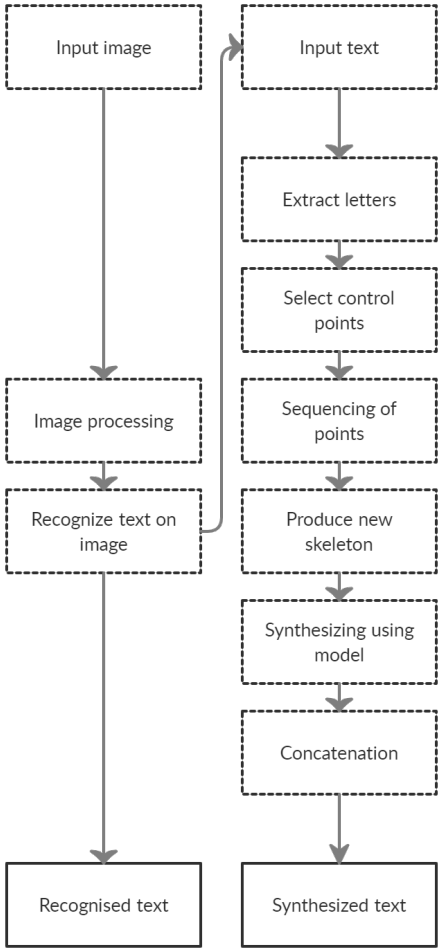


Figure 3.1: Main steps

3.1. Recognition

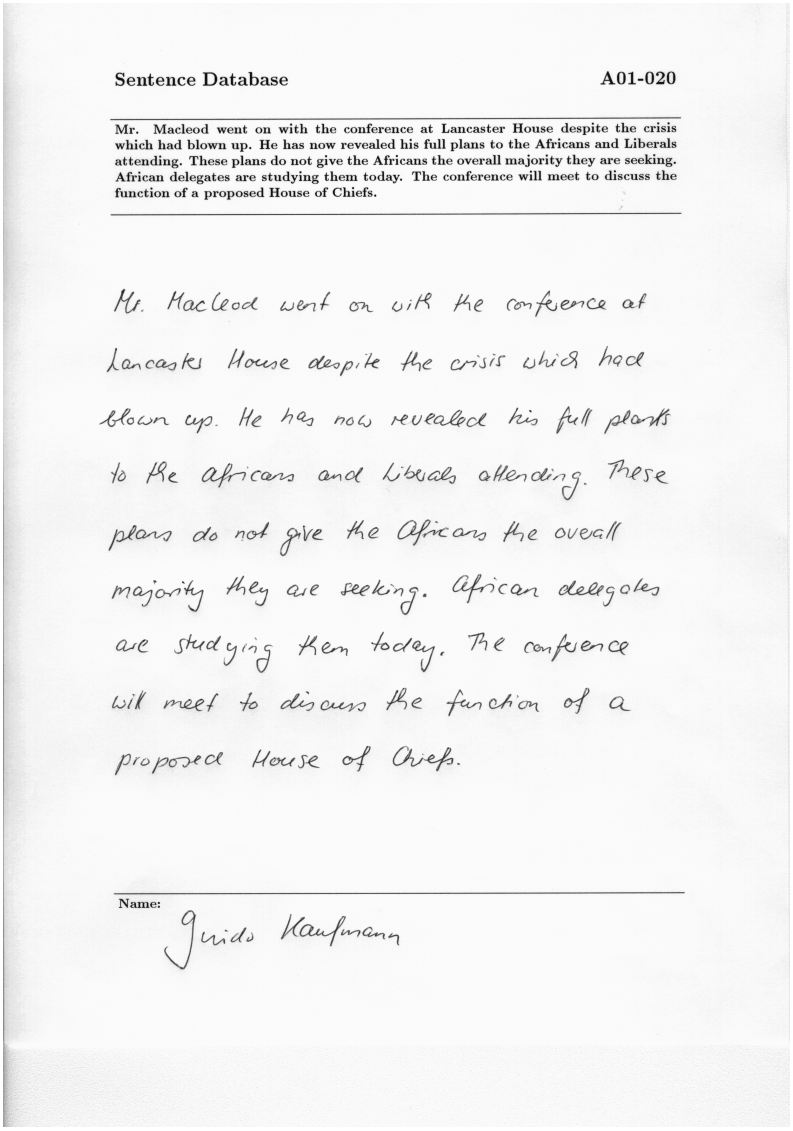
The implementation started with the text recognition part, as it is the basis of the synthesis. It starts from the discussion of the dataset, through its processing - proper preparation for the training process to the training process itself. These processes are described in detail in the following sections, including descriptions of the parameters used.

3.1.1. Dataset

The one thing that is necessary to train the machine learning model is a proper dataset. The one we used in this thesis in named IAM Handwriting Database and came from Research Group on Computer Vision and Artificial Intelligence from the Institute of Computer Science on the University of Bern.

This particular database consists of 1539 pages from 657 different writers, making it large enough to teach the recognition model properly and diverse in writing styles, which may help recognise even most specific handwriting styles properly.

Each of those 1539 pages consists of 2 main parts - machine and handwritten typed text which can be seen on the Figure 3.2.


Sentence Database A01-020

Mr. Macleod went on with the conference at Lancaster House despite the crisis which had blown up. He has now revealed his full plans to the Africans and Liberals attending. These plans do not give the Africans the overall majority they are seeking. African delegates are studying them today. The conference will meet to discuss the function of a proposed House of Chiefs.

Mr. Macleod went on with the conference at Lancaster House despite the crisis which had blown up. He has now revealed his full plans to the Africans and Liberals attending. These plans do not give the Africans the overall majority they are seeking. African delegates are studying them today. The conference will meet to discuss the function of a proposed House of Chiefs.

Name: Guido Kaufmann

Figure 3.2: Example of a page from the database

From this image, only the handwritten part is needed for the dataset. The other parts may also cause problems in the learning process; thus, we need to isolate the rest's handwritten part. Since every page has the same structure, we used the horizontal bars separating each part to differentiate them. Using appropriate image processing functions to find those bars and their positions, we cropped the image to leave only the area between the second and third bar. (Figure 3.3).

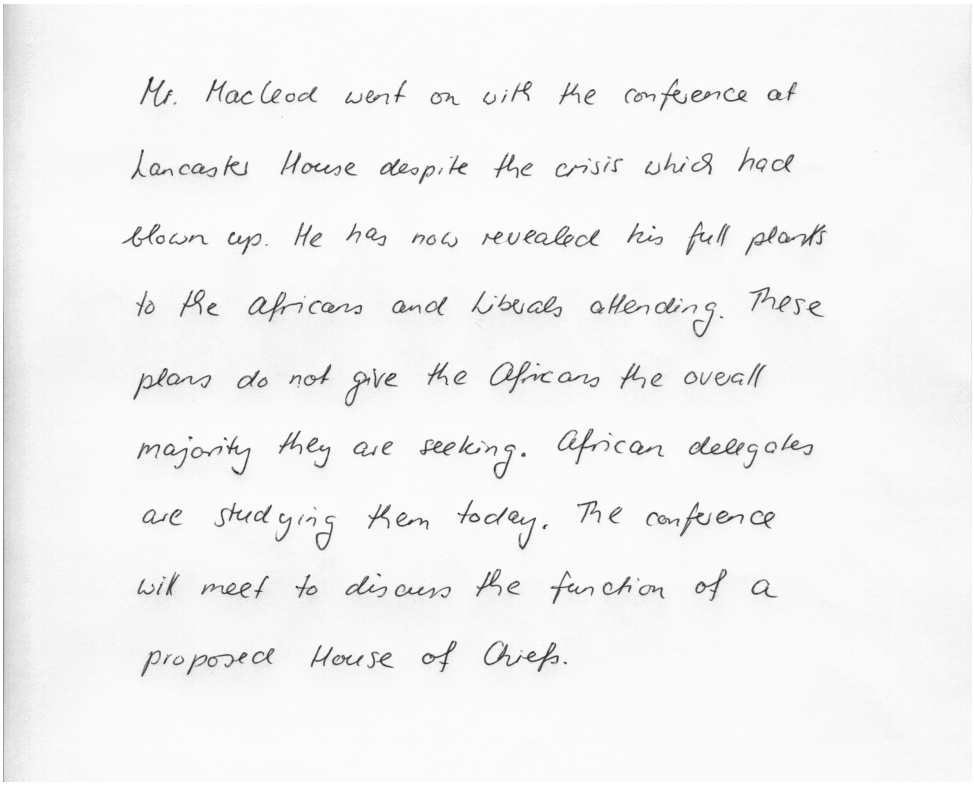


Figure 3.3: Cropped page

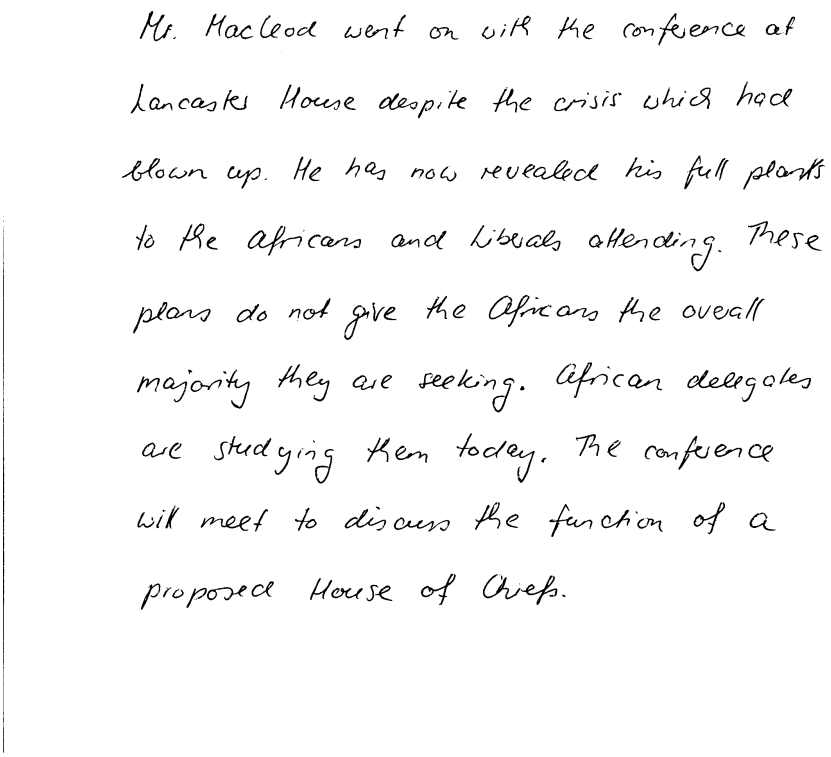


Figure 3.4: Thresholded page

To further prepare the dataset, we decided to apply thresholding to the images (Figure 3.4). Although it may distance the result from the letters' original appearance, it comes with many advantages. The biggest one is since all the pixels are either entirely black or white, the model will recognize the letter based solely on its shape rather than pixel value. It also removes all of the noise present on the image from paper and photo quality. The last advantage came circumstantially - the size of each image is reduced dramatically, which results in overall dataset size reduction by 40 times.

The second part of the dataset is to prepare appropriate labels corresponding to the images. Unfortunately, we could not use the machine typed text at the top of the pages, due to the discrepancy in line length between the handwritten and printed text. To obtain the proper labels we used another piece of the database which were the XML files describing contents and positions of each line and each letter individually on the page (example below).

```
...
<machine-printed-part>
    <machine-print-line text="Mr. Macleod went on with the conference at Lancaster House despite
     the crisis" />
    <machine-print-line text="which had blown up. He has now revealed his full plans to the
        Africans and Liberals" />
    <machine-print-line text="attending. These plans do not give the Africans the overall
        majority they are seeking." />
    <machine-print-line text="African delegates are studying them today. The conference will
        meet to discuss the" />
    <machine-print-line text="function~of~a~proposed House~of~Chiefs." />
  </machine-printed-part>
  <handwritten-part>
    <line ass="154" asx="0" asy="903" character-width="999" dss="154" dsx="0" dsy="992" fd0="
    13911" fd1="2799" fd2="-536" filter-width="3" id="a01-020-00" lbs="204" lbx="0" lby="972"
    segmentation="ok" slant="66000" stroke-width="5272" threshold="176" text="Mr. Macleod went
    on with the conference at" ubs="55" ubx="0" uby="937">
      <word id="a01-020-00-00" sentence-start="yes" tag="NPT" text="Mr.">
        <cmp x="352" y="903" width="81" height="75" />
        <cmp x="435" y="966" width="8" height="6" />
      </word>
      <word id="a01-020-00-01" tag="NP" text="Macleod">
        <cmp x="518" y="908" width="53" height="57" />
        <cmp x="568" y="936" width="74" height="37" />
        <cmp x="658" y="912" width="53" height="58" />
        <cmp x="721" y="942" width="25" height="25" />
        ...
```

With them, the only thing needed was to extract from these files appropriate parts describing line contents and write them into TXT files using python script (example presented in Figure 3.5).

Mr. Macleod went on with the conference at Lancaster House despite the crisis which had blown up. He has now revealed his full plans to the Africans and Liberals attending. These plans do not give the Africans the overall majority they are seeking. African delegates are studying them today. The conference will meet to discuss the function of a proposed House of Chiefs.

Figure 3.5: Example of text from dataset

3.1.2. Model

The recognition model's training is based on the standard English trained data available on the official Tesseract OCR repository. This fact implies that the current configuration of the project works preferably with the English language. The customization for arbitrary language is possible. In such a case, new dataset and training are required. It had been training for approximately two months on a machine with 64-bit Windows 7 operating system, 16 GB of RAM and 3.4 GHz processor.

For each image from the training dataset, the appropriate file with extension "box" is created. There exist a CLI command for creating such files. This file contains coordinates of lines' locations, as well as their content as simple text. It is possible to create files with every single letter's coordinates, but this assumes a different training approach. Based on "box" files appropriate files with *lstm* extension are created. These are binary files used during training LSTM. The page segmentation mode of the model was assumed to be a uniform block of text.

Moreover, a list of paths to the *lstm* files is required. Tesseract uses images and *lstm* files for the training. The names establish the relations between them. Hence, special attention should be paid to the consistency of the filenames. The type of implemented training is training from scratch.

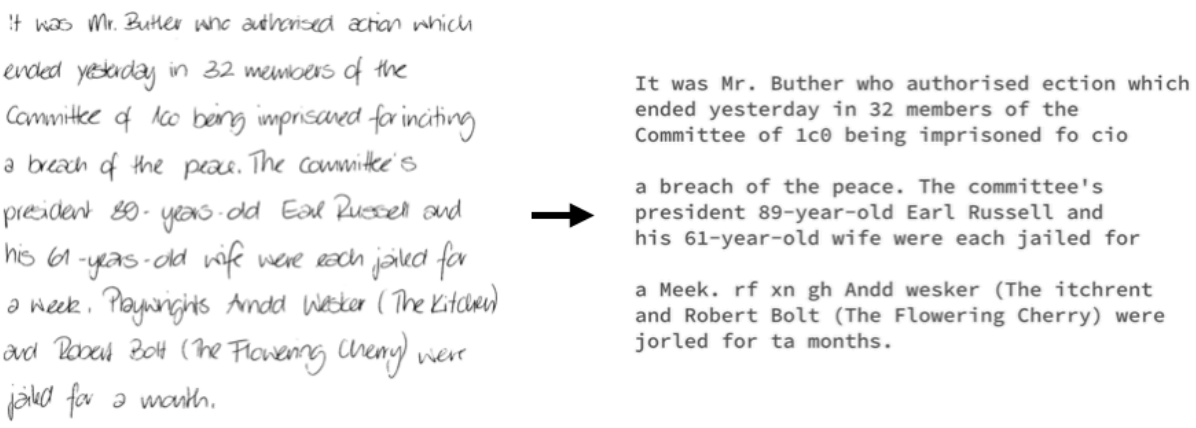


Figure 3.6: Result of a handwriting recognition by Tesseract OCR

3.2. Synthesis

This section, as the name suggests, contains a description of the entire implementation process of the module responsible for handwriting synthesis. Similarly to the previous section, it contains a description of the dataset and all its processing, but this module is much more extensive and therefore more comprehensive. This is the main part of this work, and it is what authors have spent the most time on.

3.2.1. Dataset

The synthesis models' dataset utilises the same images as the recognition model, but on a much smaller scale. For recognition, more than 1500 images were used on a single model, where for each of the synthesis models no more than five images were used. There are two reasons for such actions. The first one is that each writer in the database wrote between 2 to 5 pages of text, and each model is supposed to be an imitation of a single person. The second one is the correlation between the size of input and time needed to train the model.

It was one of the goals of the project to be able to create such a model directly in the application; thus, the time of training should be relatively short. It is also worth noting that any picture of handwriting can be used as a source for this part.

For the first step of creating the dataset, the previously trained recognition model distinguishes letters from the selected photos. After executing the command, the text file with coordinates for each letter is created. The x and y values from the file are then used to crop the letters into individual files stored in directories describing an appropriate letter (Figure 3.7).

Figure 3.7: Folder containing lowercase 'a'

Since recognising individual letters is imperfect, there is a need to further select letters to get rid of mistaken or defective ones. For this purpose, the Human-in-the-loop (HITL) approach is utilised. After assigning cropped pictures to the appropriate directory, this directory is then opened for user to select all images that are not suited for the dataset, which are then removed. Proper selection rests on the user, so it is crucial to eliminate all instances of miss-assignment of even slight imperfection if possible. Otherwise, the model and the synthesis results itself may be significantly degraded. An example of letter dataset after correction is presented on Figure 3.8.

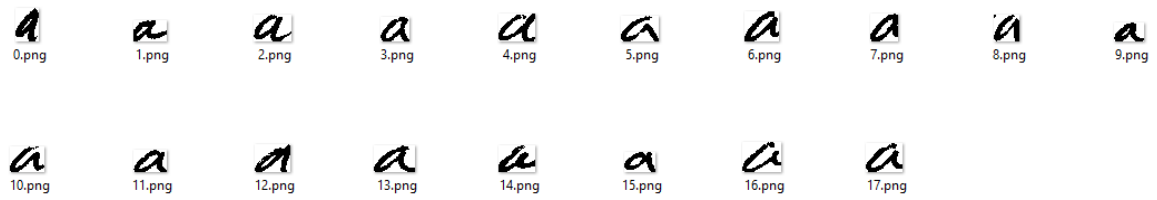


Figure 3.8: Folder with lowercase 'a' after manual correction

Such a dataset structure is afterwards used for two purposes - model training and actual letter synthesis. For each of the subsequent, changes are needed. Changes needed for the synthesis are described in the following sections. There is a need to assign a second picture for each entry in the dataset as for the training. It is different from the recognition model in which an image has a label assigned because the synthesis models are image-to-image translation instances.

The second image, in this case, is a skeletonized version of a letter. Skeletonization is performed on each image, and the results are saved in a proper subdirectory (Figure 3.9). Both images are then resized into the same size (256x256) by placing the smaller image on the center of bigger bitmap, for the purpose described in Section 3.2.4 titled **Model**. Afterwards they are combined into a single file and saved in a directory identical for all letters (Figure 3.10).

Figure 3.9: Folder containing skeletonized letters

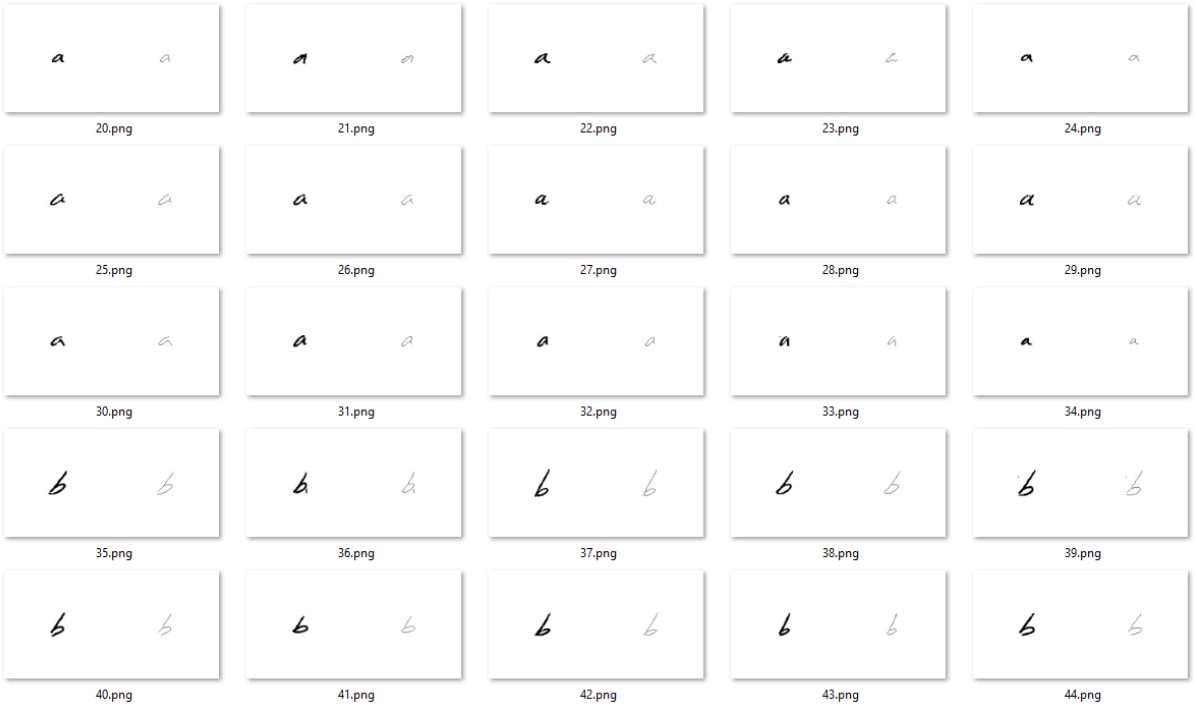


Figure 3.10: Ready dataset with combined letters

3.2.2. Control points

The control points extraction consists of three steps:

- Thresholding
- Skeletonization
- Application of Gabor filter

In this section, the implementation of the skeletonization and Gabor filter functions is discussed. In all cases, a free to use, external Python library is used. It is designed for image processing and provides developers with convenient and efficient tools. It is called scikit-image.

Skeletonization

Before an image is skeletonized, thresholding is applied. The skeletonized images are exposed to a particular thresholding function. In the other parts of the project, another function performs this operation. There exists a function *threshold_otsu* which accepts an image as a parameter. It returns a threshold based on the Otsu's method, which is an automatic thresholding method. The threshold should separate the foreground from the background. Such a prepared image is skeletonized with scikit-image skeletonize method. The implementation of this function is based on the Zhang–Suen algorithm, as well as the Lee-Kashyap-Chu algorithm. The obtained image with a skeleton is thresholded with the method described in Section 2.1.2 titled **Thresholding**.

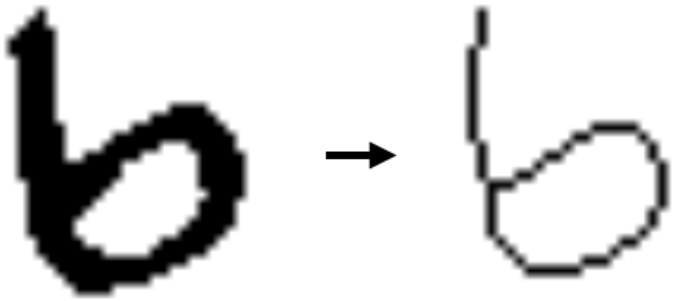


Figure 3.11: Result of skeletonization

Gabor filter

As mentioned before, the Gabor filter function is sourced from scikit-image library. However, the attention must be paid to the selection of parameters. A myriad of configurations was tested. Eventually, it was decided that the imaginary part of the results of filtering skeletons with two configurations will be combined, giving the final result. The configurations are as follows:

I	II
$frequency = 0.16 * \pi$	$frequency = 0.16 * \pi$
$theta \ = 0.25 * \pi$	$theta \ = 0.75 * \pi$

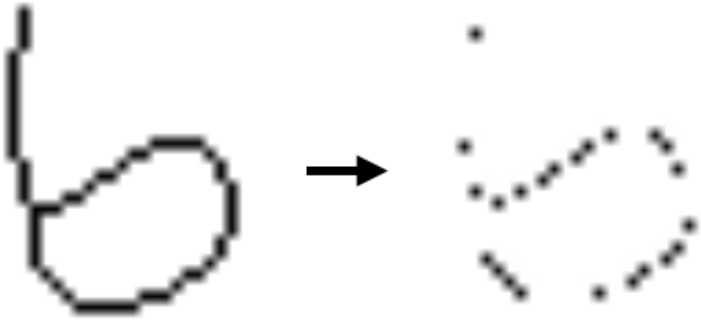


Figure 3.12: Result of applying Gabor filter to a skeleton

3.2.3. Letter drawing

As described in the sequencing and handwriting reconstruction sections, the main problem was to find the right sequence of control points. It was already known that b spline would be used for the mapping, but it was a difficult task to find the right sequence of points. Most of the work used timeline data, but this was not what our project was about. To prove the importance of this part of the work, here is a photo showing the control points of the lowercase b. The blue line in no way resembles this character.

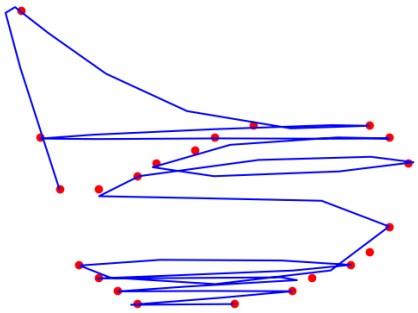


Figure 3.13: Small letter "b" before sequencing

Nevertheless, after applying the appropriate transformations, the result is already appropriate.

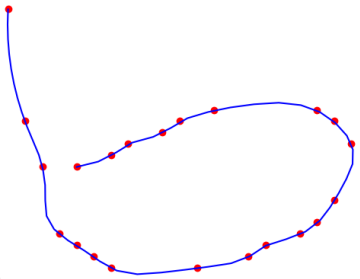


Figure 3.14: Small letter "b" after sequencing

However, it is best to start by listing the tools used to create this section for the sake of formality. The Python packages Matplotlib and SciPy were used to draw the letter. With their help, it was easy to generate a graph containing the B-spline after the previous interpolation. For ease of testing and elimination of errors, this process was broken down into several minor functions. For ease of testing and eliminating errors, this process was broken down into several smaller functions, which took many different parameters. The effect of one of these can be seen in the image above. Of course, control points are a significant part of the project, but they should not be visible in the final image. Therefore, the effect could be freely edited, which helped define the final result's quality.

When the drawing itself was ready, the arduous process of finding the correct sequence of control points began. There have been many unsuccessful attempts, but this paper will only describe the latest algorithm that solved this problem. It is worth mentioning that no additional packages are needed to reproduce this algorithm. Main task and mode of operation were explained in the theoretical part. So in this one, the effects of the different steps will be presented. The picture presented below shows a graph based on the letter A, which is shown next.

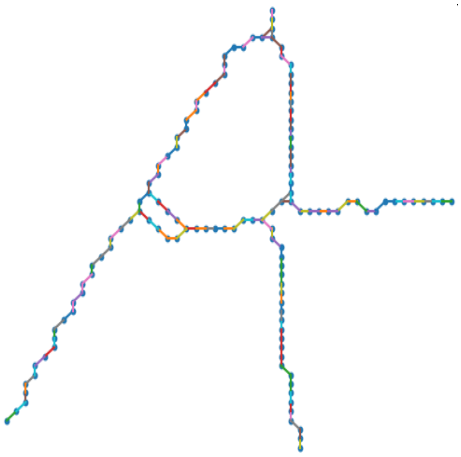


Figure 3.15: Graph of "A" letter

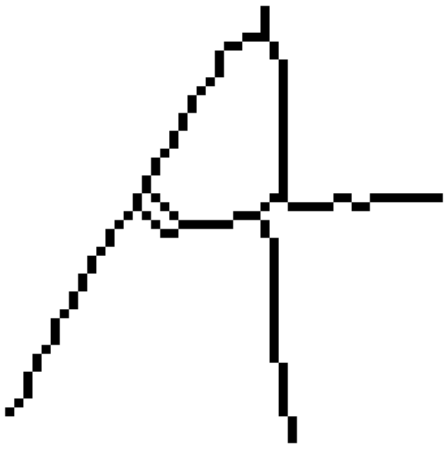


Figure 3.16: Skeleton of "A" letter

This graph is not directed yet, which could help a lot with the task at hand. However, before the right order is chosen, which is equivalent to making the graph directed, small cycles must be removed. Only small ones, because there is a cycle in the letter a and it should stay there.

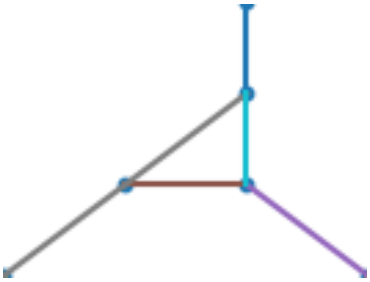


Figure 3.17: Graph of "A" letter

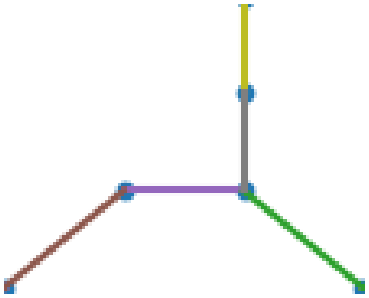


Figure 3.18: Skeleton of "A" letter

A character is drown based on a set of sequences which forms the entire shape. The algorithm presented in the theoretical part is implemented in Python. There is no external library included in this functionality.

There is an important detail in this implementation. It turns out that the points extracted with Gabor filter are shifted by one compared to the skeleton. It happens because this filter is a band-pass filter, and it looks for directed content. The solution is to shift the control points by one in a direction estimated with relation to the skeleton. An exemplary set of sequences extracted from a character is shown in the Figure 3.19.

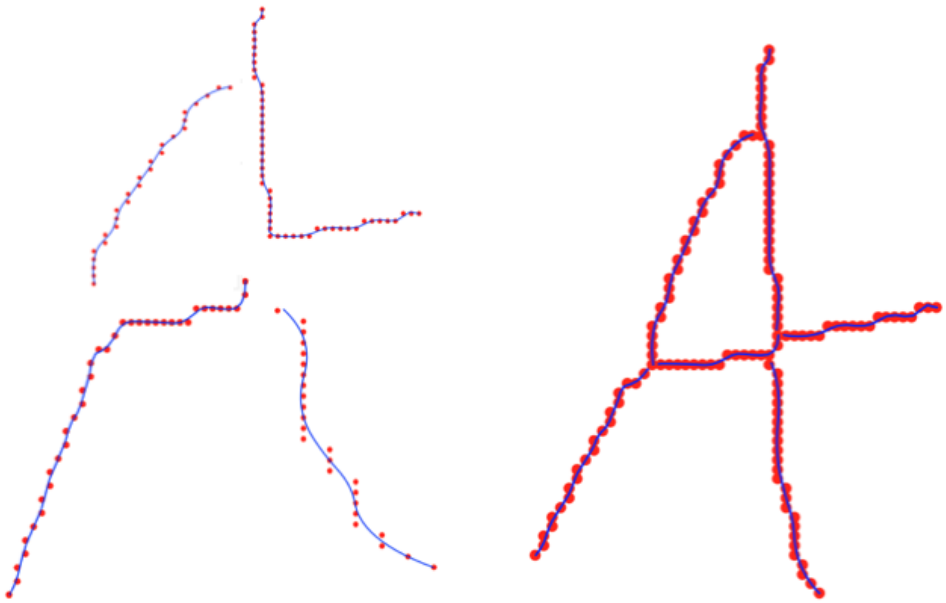


Figure 3.19: Sequences of a instance of letter A

3.2.4. Model

The need to use a machine learning model comes from the fact, that there is no simple way to imitate handwriting style, like the ink thickness in specific parts of the letter using just algorithms. Neural network, or more specifically an image-to-image model, is much more suited for this task. Our algorithm for new letter creation is focused on the letter's shape itself in form of a skeleton. The skeleton-letter pair from the recognized text is used as a training dataset for the model, which then can transform the newly-created skeleton into a unique letter, keeping the author's style.

Implementation of a training of the model is based on a project named *pix2pix* utilizing generative adversarial network with *TensorFlow* library. Thanks to this library, only a couple of commands have to be used to begin the training process. This, however, does not mean, that there are no obstacles in said process. A condition has to be met, and several variables needing specification to create a working model successfully.

The condition is that all images have to be in the identical size of 256x256 pixels. This constraint comes directly from the specific version of the library's source code used, which cannot be easily changed due to the package's problems on various systems.

As for the variables, the first one which had to be altered from the original value, is the margin of error for discriminator and generator loss. The value has been changed by one order of magnitude from the initial one, due to frequent errors during the learning process. The remaining variables are the number of epochs, generator filters and discriminator filters - all of which has to be appropriately adjusted for each style. In most cases, improperly selected variables resulted in the model producing images with different color or texture. Example of an output produced by a model trained with incorrectly chosen variables is presented in Figure 3.20.

Figure 3.20: Example of a bad output

The average time needed to train a single model is approximately 6-8 hours. It is quite long, but fortunately, there is a way to shorten this process significantly. *TensorFlow* library supports the usage of dedicated graphics processing unit (GPU), which makes fair use of multiple threads to speed up training. This solution requires additional software to run correctly on top of possessing one of the supported cards. If the setup is correct, then the process's length shortens to about 20-30 minutes per model.

After the model has been trained, it is then exported to another format, which reduces its size and makes it ready to use. However, the solution of how to use the newly created model was another problem that needed to be solved. The library's intended use has changed since project creation from local usage to the model integration into an online server, and no solution for local model usage exists in the library anymore. After a considerable amount of research, we found a way to use a now obsolete version of the library to create an output in binary format locally and save it as an image. For this step, a dedicated GPU can be used as well, which improves the time needed to process many images at once. Example of successful image creation is presented on Figure 3.21.

Figure 3.21: Example of a successful output

3.2.5. Letter concatenation

This project's main scope was creating individual letters that were supposed to resemble the style of writing of a person whose handwriting was analyzed. This fact means that another part of actual handwriting which is how various letters are connected has to be simplified to finish the project on time. Based on papers that were used as an inspiration for this project, more complex letter concatenation methods might have taken a lot more time and knowledge that the amount needed for this project. Taking this into account, the algorithm for connecting characters is a simple one. Its goal is for the synthesized letters to show a resemblance to the human handwriting.

Since the synthesis model produces each letter as an individual image, the algorithm's main task is to place those images side by side the most natural way possible. To archive this goal, the letters are treated differently based on their shape. All places for capital letters and most of the lowercase letters are calculated by subtracting their height from the bottom edge of the current line. Such action places them so that their most bottom pixel is always at the same height as all others in the same line. For the letters which lower part should be lower than the line (i.e. descenders), only half of their full height is subtracted from the starting point.

Another aspect of placing letters is their horizontal position. Although the model creates the letters on the bigger plain of fixed size, all images are then cropped such that no blank spaces are left beyond marginal parts of the characters. Each image is then placed on the final bitmap right behind the previous one, which to some extend imitates letter connection.

If the next letter's width is too large to be placed at a current line, then the current horizontal position is reset to zero, and the current height is increased by fixed value bigger than any of the letters. This method allows for multiple lines of text to be generated, but it has a downside, resulting from the fact that letters are treated independently. The word may be broken in the middle without taking rules of grammar into account.

There may exist a case when there are no instances of the letter that should be generated in the dataset. If this letter is the one with diacritics, it is replaced with its closest equivalent in ASCII. If a said character is an uppercase character, then the lowercase for this letter is used. In other cases, such a letter is omitted, and the next letter is processed.

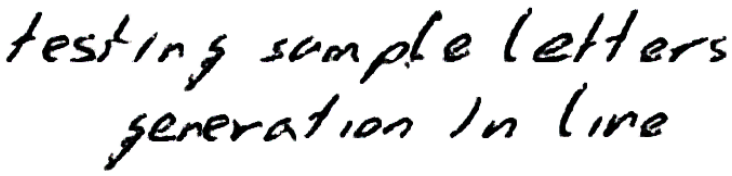


Figure 3.22: Example of letter concatenation

3.3. Graphical user interface

Initially, it was assumed that the graphical interface would only be an addition to the rest of the components and facilitate the use of the listed functions. However, at the end of the development process, the application became more extensive and more complicated. It was also taken for granted that the application should work on the most popular systems (at the time of writing) such as Windows 10 and macOS.

Knowing such assumptions, the authors chose the WxPython package as the one allowing to create graphical interfaces. Its advantages were compatibility with the systems mentioned above and ease of use. As soon as the package was added, work could begin.

At the start of developing, an application with a single panel was created, in which it was possible to use all the functions available at that time. At first, only text recognition and only later the possibility of synthesis.

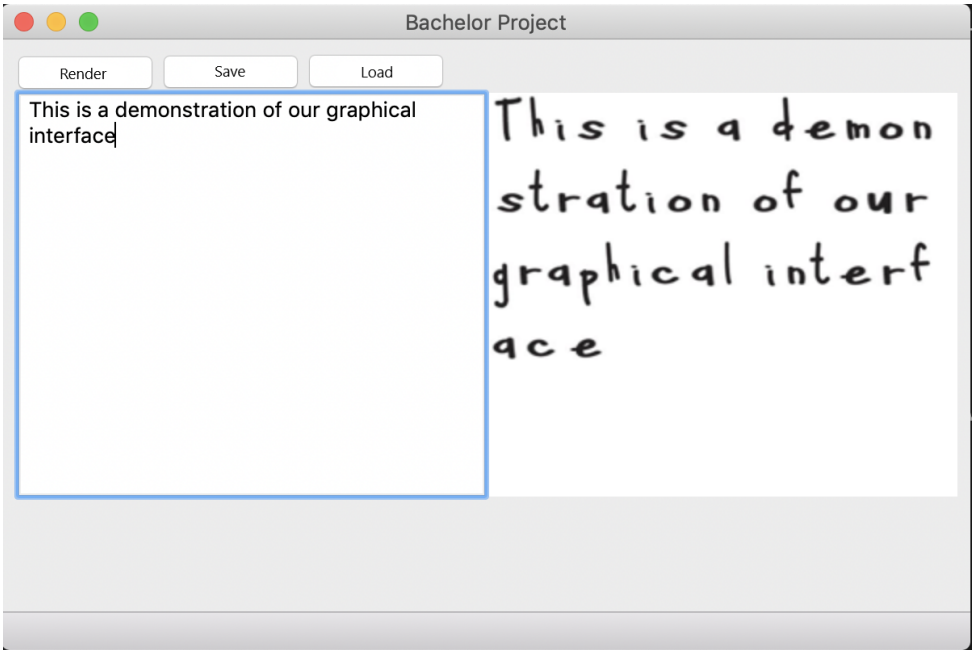


Figure 3.23: GUI version 0.1

As it is shown in the picture above, the GUI was very simple and not very intuitive. Rather everyone would guess that the "Render" button started the process of generating a photo with synthesis, but not everyone would know how to run text recognition from a photo. That functionality was done via the "Load" button, which opened a dialogue window. One had to select a photo, which was loaded into the application, and then the recognition process started. The result appeared on the right side of the application in text form. Such an interface was very primitive, but it made it much easier for developers to test the essential functions.

As the GUI was not a priority of the thesis for most of the production process, it existed in this form or slightly changed until the creation of new functionality was finished. Then the integration and bug fixing began, during which the GUI underwent major changes. The basis for that was choosing the right colour palette and deciding to split the functionality into two panels. The first one was responsible for text recognition and the second one creating a synthesis. Thus, the first colour palette was selected.

Figure 3.24: First palette

It was used to create the first more advanced version of the interface, which had more functions and was divided into two panels.

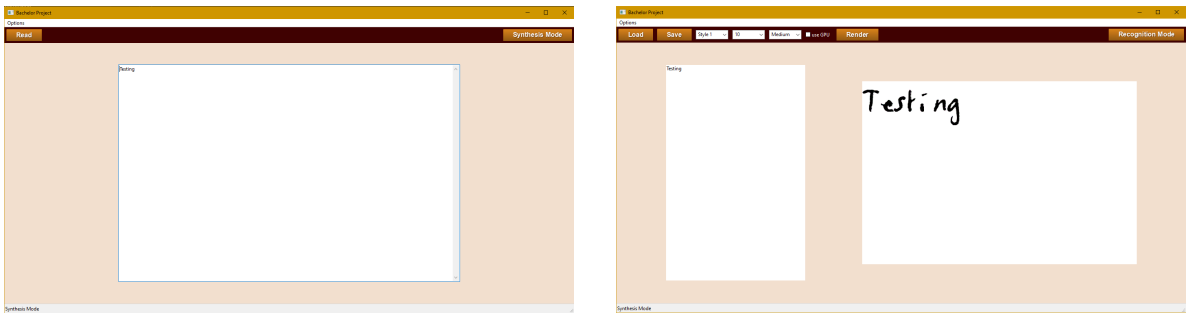


Figure 3.25: GUI version 0.5

There are as many views as there are people, but if there is a majority, then it starts to matter. Therefore, after many votes, the colour palette was changed to a more subdued and serious one.

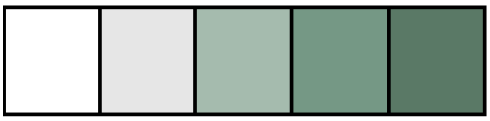


Figure 3.26: Second palette

Everything in the app has been adjusted to the new colour palette. This also involved re-creating the buttons using Cascading Style Sheets (CSS).

Figure 3.27: Examples of buttons

Many of the less useful functions were moved to the menu bar so as not to confuse the readability of the interface. Also, the button designations were changed to make the functionality of the program more understandable for new users. The size of the text box and bitmap has also been better adapted to the size of the window. After such treatments, the effect of the next version is shown in the pictures below.

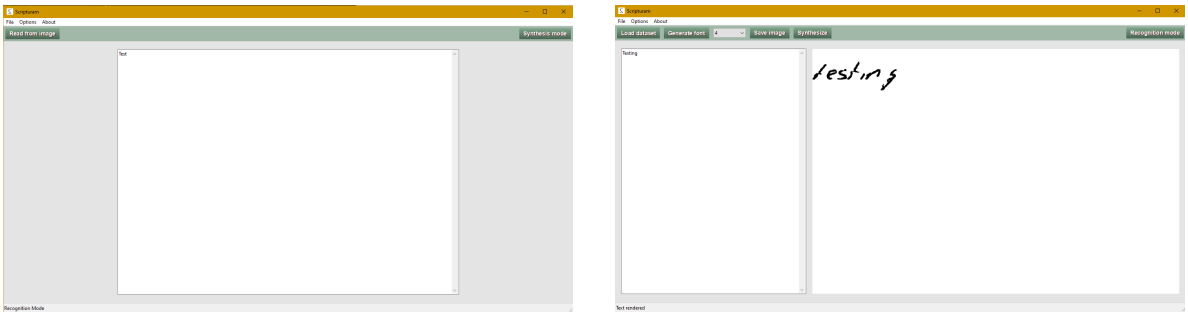


Figure 3.28: GUI version 1.0

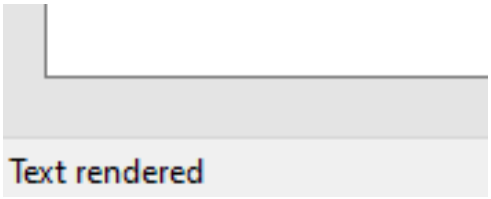


Figure 3.29: Status bar

To change the panel, use the button at the top right of the respective panel. When pressed by the user, an event is sent, and it is handled by a higher window class. In the last stage of implementation, many options were also created, e.g. allowing to change synthesis parameters or turning it off completely in order to see the effects of simply joining the drawn letters. Also, the status bar gives a lot of information, among others, about the current stage of synthesis or its lack of success.

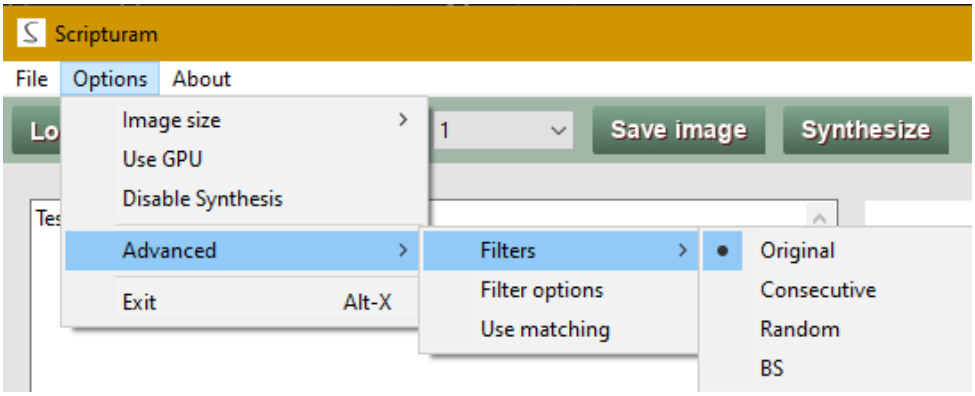


Figure 3.30: Menu bar

The final additional step was to give the application a name. In the end, we settled on the Latin word - Scripturam - which means 'writing'. The choice of icon depended on the name, and represented the synthesised first letter, in this case, "S".

Figure 3.31: Scripturam icon

This is what the process of creating a graphical user interface for the Scripturam application looked like. A user manual with all the functionalities listed is included in Section A.5 titled **Manual**, and the process of launching and installing the necessary components is described in Section A.4 titled **Installation instruction**.

3.4. Tests

For this kind of work, testing is a problematic assumption because only a human can correctly assess whether the synthesis is good enough. The writing should not be identical to the original, but also not too different from it. It is not easy to define limits for which a human will not distinguish between synthesis and handwriting. Nevertheless, unit, integration and acceptance tests have been created to meet the requirements of engineering work. The evaluation of the work itself is in a separate section and is very subjective for the reasons described above.

The tests themselves use the unittest library and are called using the pytest package. It is also worth mentioning that the Coverage package was used to evaluate the tests and their coverage. It produced a comprehensive report containing, among other things, information on which functions had already been tested. None of the above packages is needed to run the application, but only for testing, so they are not listed in the "requirements.txt" file. Additionally to preserve code quality all tests have to be successfully completed in order to commit any changes to the main repository.

For ease of use, a script has been created which can be found in "scripts/tests/test_report.sh" and contains the test calls with the appropriate parameters. All tests are located in the "tests" folder, together with test data and functions calling all and individual tests. Both unit and integration tests cover above 80% of the code.

3.4.1. Acceptance tests

All acceptance tests are presented in the form of a table with the relevant sections. They begin from the functionality to be tested, through the action to be performed, ending with the expected and actual result. The tests were conducted on the macOS operating system and contained only scenarios accessible from the graphical interface.

Tested item	Action	Expected outcome	Actual outcome
Read from button	Click the button and choose an image with handwriting to be read	The text present on the selected image should and written into the textbox	The text is recognised and written into the textbox

Tested item	Action	Expected outcome	Actual outcome
Read from button	Click the button and choose an image without text	Display error message on the info bar	The error message is printed on the Info bar, and the text in the textbox does not change
Synthesis mode button	Click the button	The Synthesis panel shows up	The Synthesis panel shows up
Recognition mode button	Click the button	The Recognition panel shows up	The Recognition panel shows up
Synthesize button	Click the button when the textbox contains text with elements belonging to the selected dataset	The text is rendered, the image is generated and displayed	The text is rendered, the image is generated and displayed
Synthesize button	Click the button when the textbox contains text with some elements present in the dataset	The text is rendered, the image without missing elements is generated and displayed	The text is rendered, the image without missing elements is generated and displayed
Synthesize button	Click the button when the textbox contains text with no elements belonging to the selected dataset	The error messages displayed and the empty image is showed	The error messages displayed and the empty image is showed
Save image button	Click the button and select a path	The image with the synthesised text is saved in the selected directory	The image with the synthesised text is saved in the selected directory
Droplist	Change the selected dataset	The selected font is changed	The selected font is changed
Droplist	Click New font	a new font is created	a new font is created

Tested item	Action	Expected outcome	Actual outcome
Generate font button	Click the button	a new font based on the current dataset is generated	a new font based on the current dataset is generated
Load dataset	Click the button and select a directory with images containing handwriting. Select incorrect or misassigned images of the letters.	The dataset without the selected letters is created	The dataset without the selected letters is created
File / Save menu item	Click the menu item in the Synthesis mode and select a path	The image with the synthesised text is saved to the selected path with the *png* extension and a text file with the text from the textbox is saved to the selected path as *txt* file	The image with the synthesised text is saved to the selected path with the *png* extension and a text file with the text from the textbox is saved to the selected path as *txt* file
File / Load menu item	Click the menu item and select a directory with images containing handwriting. Select incorrect or misassigned images of the letters.	The dataset without the selected letters is created	The dataset without the selected letters is created
Options / image size	Select size	The image is resized appropriately	The image is resized appropriately
Options / use GPU	Click first the menu item and then synthesize button	The GPU is utilized for rendering letters and text is created	The GPU is utilized for rendering letters and text is created
Options / Disable synthesis	Click first the menu item and then synthesize button	The text is concatenated using only images from the dataset	The text is concatenated using only images from the dataset

Tested item	Action	Expected outcome	Actual outcome
Options / Advanced / Use matching	Click the menu item	The flag value is changed to its negation	The flag value is changed to its negation
Options / Advanced / Filter option	Click the menu item and input positive natural numbers	The control points are filtered while rendering an image according to selected values	The control points are filtered while rendering an image according to selected values
Options / Advanced / Filter option	Click the menu item and select some non-numeric symbols	The textbox validation fails, and the change cannot be confirmed	The textbox becomes red and OK button is disabled
Options / Advanced / Filters	Choose a filter	The selected filter is changed	The selected filter is changed
About / Authors	Click the menu item	The information window is displayed	The information window is displayed
Synthesize button	Click the button when the selected font is empty	The error message is displayed	The error message is displayed

Table 3.1: Acceptance tests

3.5. Compilation

During the work, the correct compilation was essential, and at the beginning, there was also the problem that different parts of the code did not work together in separate files. A file “setup.py“ was created to solve such problems, defining the development environment. It is started with the command “python install .“, then it checks if all necessary packages exist, and if not, they are installed. After this process, all files work together.

Once the development process was complete, the project was compiled, and an executable was created for Windows 10 and macOS. This process differed primarily in the packages used. For the Microsoft system, the PyInstaller package was chosen. However, there were problems with it on Apple's system, so the py2app package was used. The only difference was in the additional libraries needed to run the application. The authors stated that the libraries should be sufficient to run the application - not to use all the features entirely. Tesseract is an external program not included in the application but required for proper text recognition. Of course, it is necessary from user to only install the above program. The learned model is already in the application files and is used during recognition.

For ease of use in compiling newer versions, the “scripts“ folder contains appropriate folders for each of the above systems with bash scripts exe.sh for Windows 10 and app.sh for macOS, respectively. These contain a call to the previously discussed packages with the appropriate parameters and the relevant files' addition. Since the application uses button images, as WxPython is not a package that allows button processing, the buttons were created early and added as images to the application. Therefore, the application needs to have a “resources“ folder. It also needs a “data“ folder containing files needed for synthesis, including trained models.

4. Results

The end-user may examine the outcomes only thanks to the GUI provided that solely the executable is available. Exceptions are the developers who have an insight into a part of the partial results. In case the program is run from the source code, the debugger may help. The unit and integration tests monitor the proper functioning, but the subjective assessment related to personal aesthetics cannot be made. Hence, the results depend, to a large extent, on the input data. Moreover, the input data consists of images containing handwriting. Other solutions to this problem use datasets which include such details as the change of the position in time. All features extracted in this project are the outcomes of image processing operations.

4.1. Recognition module

This module depends entirely on external software, namely Tesseract OCR. The model has been trained for two months. The dataset huge and the hardware resources were limited to a single PC. Hence, the model deals with some handwriting style well, but not satisfactory results may be yielded by new, sophisticated script.

In the Figure 4.1 an almost ideal case is presented. The next figure proves that the recognition is not always perfect. However, the trained model produces output more similar to the handwritten text.

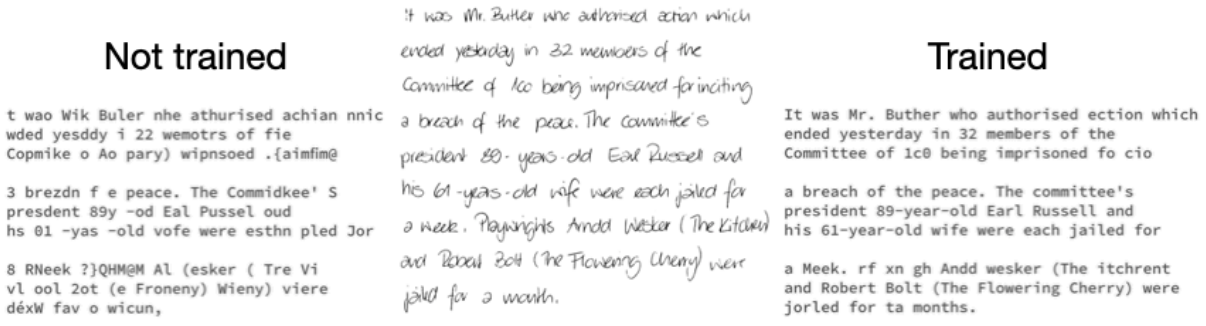


Figure 4.1: Comparison of trained and not trained Tesseract OCR outcomes

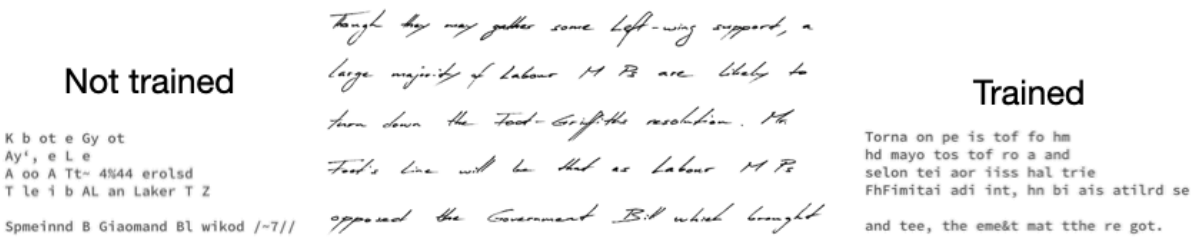


Figure 4.2: Comparison of trained and not trained Tesseract OCR outcomes

4.2. Synthesis module

The synthesis requires an interaction with the end-user in order to create a new font. In case there is an already generated font, it may be used without any limitations. The generation, though, is a crucial process. The appearance of a new font depends on the decisions made during dataset creation. Moreover, the recognition model plays a part, too. Its ability to recognize letters for a given handwriting influence the synthesis. The Figure 4.3 shows a difference between an original handwriting and a synthesized text. There are many additional options for synthesis which may be enabled.

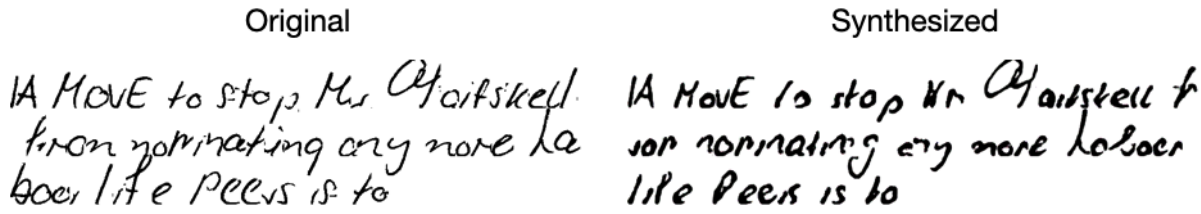


Figure 4.3: Comparison of an original text and synthesized equivalent

The Figure 4.4 shows a text consisting of original, no-changed letters. The Figure 4.5, in turn, the same text synthesized based solely on extracted control points. The synthesis produces text similar to the original one but with small losses. The parameters have to be adjusted appropriately. Figure 4.6 shows a generation with additional consecutive filtering where the number of requested points is 10. Integral parts of letters are cut off because the filters sequentially select only 10 control points. A synthesis with matching is shown in Figure 4.7. The result is satisfactory.

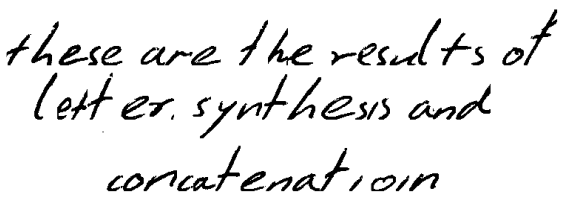


Figure 4.4: Text without synthesis

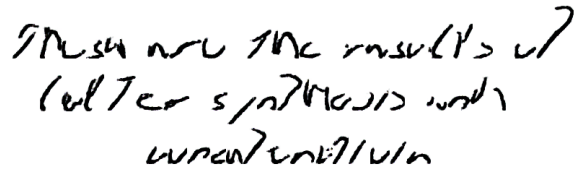


Figure 4.6: Text generated with consecutive filter - 10 control points

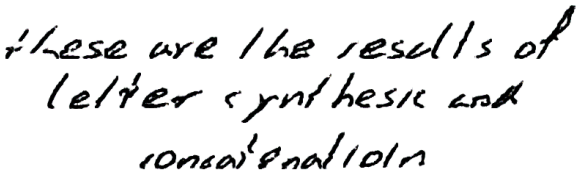


Figure 4.5: Text generated with neither control points' filtering, nor matching

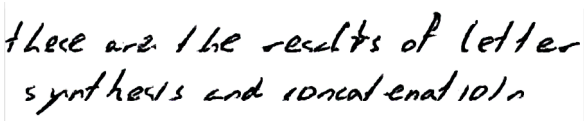


Figure 4.7: Text generated with matching, without filtering of control points

As it was mentioned above, the synthesis is based on the dataset.
Figure 4.8 shows tenuous result.

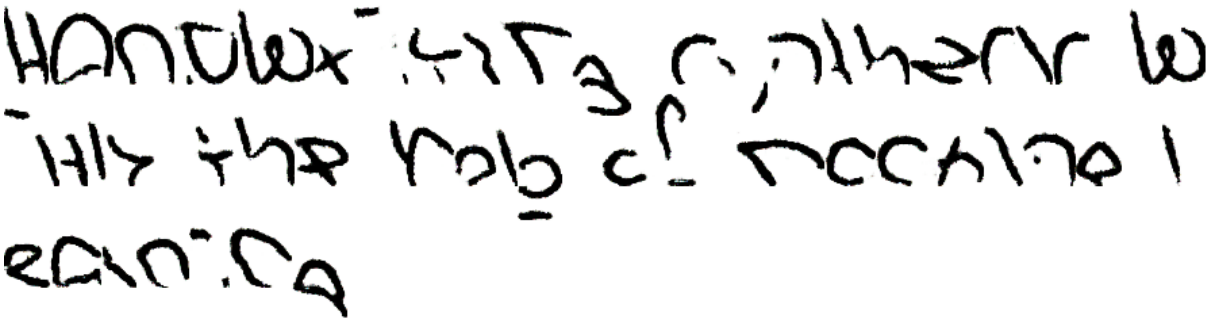


Figure 4.8: "Handwriting synthesis with the help of machine learning" written with bad configuration and dataset

5. Summary

The final version of the project with results of both recognition and synthesis leads to several reflections. They can be easily divided into positive and negative ones.

5.1. Successes

Out of all the successes, the most important one is that all deadlines were met and all functionalities have been delivered. Both functional and non-functional requirements were satisfied.

Each team member has completed his work thoroughly, on time and covering all parts assigned to him. Integration of individual modules went smoothly, with minimal complications, and each problem was solved on an ongoing basis without any delay. It proves good coordination and teamwork between members with full involvement in the project.

Another success was a drastic improvement of the tesseract model's letter recognition on handwritten text compared to the basic model. This change results from the vast and diverse dataset used for training.

The synthesis process can be characterized by a reliable representation of letters, despite the lack of letter creation timeline in the dataset, which was present in papers on which this project was based. Additionally, the possibility of adjusting various variables used for the synthesis process (e.g. the number of control points) on the fly, improves chances to obtain the desired results.

The last one is the functionality to create and train synthesis models representing writing styles directly through the graphical user interface. Additionally, using GPU to speed up the process significantly allows for relatively practical use of the project, rather than more theoretical one.

5.2. Failures

Although the recognition model can distinguish individual letters, it is far less reliable in the proper translation of whole words or sentences. This flaw makes recognition panel more of an addition to model training and synthesis, rather than full-fledged functionality.

Letter concatenation is another part with less than ideal results. With focus heavily placed on creating letters rather than connections between them, created words' appearance is visibly artificial. While the synthesised individual characters may be mistaken for the real ones, the final image with words and sentences sadly cannot.

In many cases, model-created characters are similar to the original ones and, within the same handwriting style. However, the randomized nature of the creation algorithm leads to numerous failed cases. The whole process has several sections of potential errors. From possible imperfections in a dataset, random choice of template from a dataset, to the unpredictability of control point extraction and letter matching. All of the above influence the result of the synthesis, to the point where the produced letter may not even resemble the desired effect.

5.3. Next steps

Considering both successes and failures of the project, there are several possible solutions for improving and expanding upon the project's idea:

- Extend the learning time for the recognition model to increase its accuracy. It would improve the recognition panel usability and reduce the number of errors in the letter's datasets and increase the sample size of letters instances by more proper character assignment.
- Further develop the creating algorithm. Additional steps like the rough comparison of created letter to the original from dataset may eliminate the problem with unlucky errors during synthesis. If the produced letter is vastly different from the initial one, it should be discarded and recreated. Since the generated characters will never be identical to each other, it will eventually lead to an acceptable result.
- Create a separate algorithm for connections between letters. While the complete and detailed solution for believable concatenation may be out of reach for this project, it may be possible to devise a more straightforward replacement. The algorithm would check if the connections exist in the initial sample of handwriting and then try to create artificial connections between letters imitating some of their characteristics like points of origin or line width.

A. Application

For ease of use of the functionalities developed during the development process, an application with a graphical interface has been created. It allows user to conveniently check all the functions, including recognising a given text in the photo and creating a synthesis of any text. All requirements that are needed to use the application, the entire process of installing the relevant components and the application configuration are described in the following section.

A.1. Requirements

The recognition model based on the Tesseract OCR program and installation of such software is required to work the part of the project. Of course, the trained model includes program files, so users do not need to train any new model, but it is possible to substitute the model with better results. Currently, developing the highest stable version is 4.0, and that variant used in the project.

Similarly the base version of Python was chosen. The highest stable version and the one used in the developing process was 3.8, so version 3.8 or higher is required for the project's proper working. Additionally, the project was compiled with the Python 3.7 but not thoroughly tested what prove that every version of Python which is compatible with the list of libraries (described below) may be used, but all functionalities were tested on Python 3.8.

Many features and algorithms are using the libraries. The number of libraries, including their sub-libraries is around 50, but all essential packages with suitable versions are listed in the requirements file. Below is the description of the essential packages.

- opencv-python v.4.4.0.46
- numpy v.1.19.4
- Pillow v.8.0.1
- scikit-image v.0.18.1
- matplotlib v.3.3.3
- tensorflow v.2.3.1
- Unidecode v.1.1.2
- wxPython v.4.0.7

First four are used mainly in image processing. The next one applies for creating handwriting reconstruction with the B-spline. Tensorflow is part of pix2pix module used in synthesis and Unidecode similarly used in the same part but the package decode text typed by the user. The GUI should be compatible with every popular system, and for the purposes, the WxPython is the base of the interface.

In order to utilize a dedicated graphics card during model training (described in Subsection A.2 titled **Hardware resources**) additional software is needed:

- NVIDIa GPU drivers 450.x or higher
- CUDa Toolkit 10.1
- CUPTI (ships with the CUDa Toolkit)
- cuDNN SDK 8.0.4

A.2. Hardware resources

Although using better hardware may visibly speed up the computation in the project, but the minimum requirements for the relatively smooth performance are:

- 4 GB of RAM
- 20 GB of storage space
- 4 core 3.4GHz processor

Additionally, if one wanted to speed up training new handwriting models significantly, it is possible to use a dedicated graphics card for such purpose. In that case, as for now, the only known compatible solution would be one of the Nvidia GTX 1xxx series cards.

A.3. Configuration

All of the project' communication is handled through files, thus if one wanted to change the data manually, he should use the directory *data* located in the project' directory. If one has trained his own recognition model, he should replace the model stored in $data/recognition\_model$.

A.4. Installation instruction

The installation process is straightforward, consisting of just a few steps. In this section, instructions on installing and running the application depending on the selected system are explained. It is enough for the user to choose an appropriate section and follow the described steps.

A.4.1. Tesseract

The Tesseract OCR is suitable for many operating systems. There are prepared installation instructions for Windows, Mac OS and Linux.

Windows

On Windows, the Tesseract OCR may not be installed with the code from it's official GitHub repository. The Tesseract at UB Mannheim version is for Windows. In order to install it, one has to download the installer exec file and run it. The training tools may be added during the installation.

Mac OS

On this operating system the Tesseract OCR may be installed with Homebrew with the following command:

```
brew install tesseract # install tesseract
```

Linux

- Pacman:

```
sudo pacman -Sy tesseract # install tesseract
sudo pacman -Sy tesseract-data-eng # install english language
```

- Apt-get:

```
sudo apt-get install tesseract-ocr # install tesseract
```

A.4.2. Repository

For the control version, the GitHub platform was used, and there the **repository** is available. Moreover, the project's set up contains only three commands (if the project is already downloaded).

First step after downloading the project (and suitable Python) is checking and install all required libraries by

```
pip install -r requirements.txt
```

Next compile project using

```
pip install .
```

And last step, run graphical_interface.py

```
python src/graphical_interface/graphical_interface.py
```

After all the steps, the GUI shows, and the whole project is ready to use if Tesseract is installed. Otherwise, the first subsection describes the process of installing Tesseract.

A.4.3. Executable

The program has been compiled, and two executable versions of the application are available on **the MEGa drive**. The compressed application contains all needed libraries and is ready to use after decompression. There is no need for installation process if the Tesseract is already installed. Otherwise no then he first subsection describes the process of installing Tesseract.

Of course, choose Scripturam_Win10.zip for Windows 10 operating system and Scripturam_macOS.zip for computers with the macOS system.

A.5. Manual

Available Modes:

- Recognition (Default)
- Synthesis

Recognition Mode:

- “Read from image“ button (1): opens a file dialog allowing to read handwriting from a PNG file. The result is written to the textbox in the middle.
- “Synthesis mode“ button (2): switches to the synthesis mode.

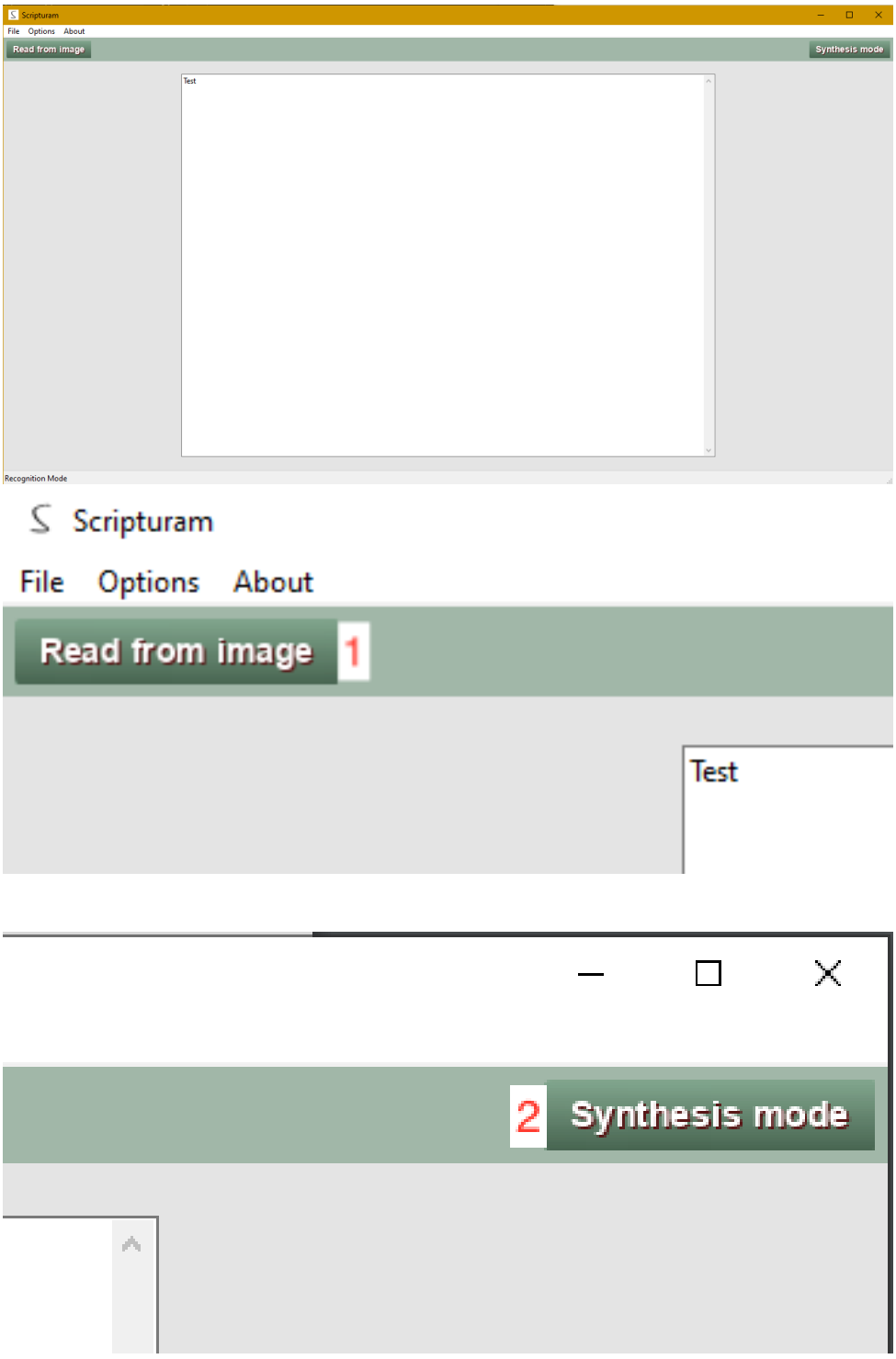


Figure 1.1: Recognition panel

Synthesis Mode:

- "Load dataset" button (3): opens a dirdialog. All files with the extension PNG residing in the selected directory will be parsed and a new dataset will be inserted into the font

selected in the dropbox located between “Generate font“ and “Save image“ buttons.

- “Generate font“ button (4): Creates a font model based on the dataset selected in the dropbox located between “Generate font“ and “Save image“ buttons.
- Dropbox located between “Generate font“ and “Save image“ buttons (5): Availabe datasets and fonts for them (if generated). (*New font* option creates an empty entry, for which “Load dataset“ and “Generate font“ functionalities must be utilized in order to sythesize a text based on it.)
- “Save image“ button (6): saves synthesized image in the selected location.
- “Synthesize“ button (7): creates an image with the handwritten text based on the text from the textbox using selected font from the dropbox.
- “Recognition mode“ button (8): switches to the recognition mode.

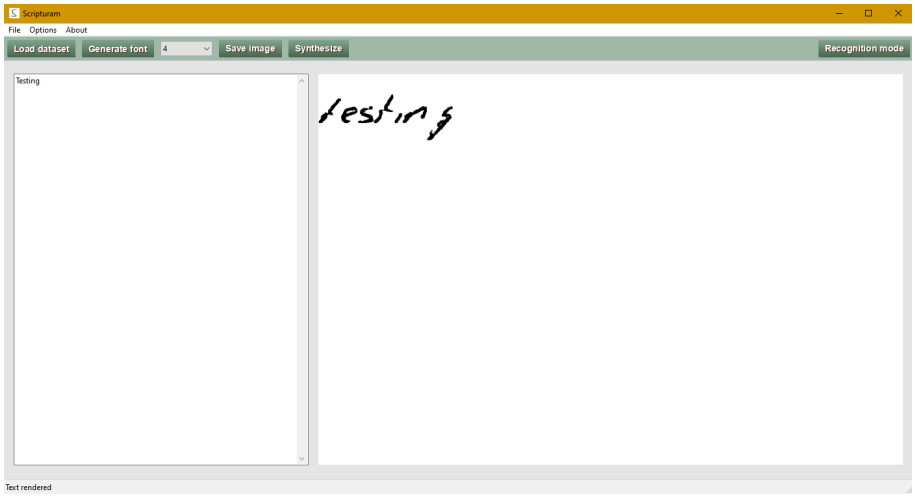


Figure 1.2: Synthesis panel

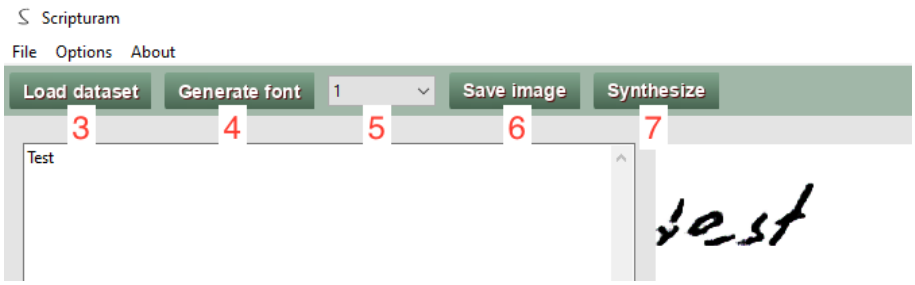


Figure 1.3: Synthesis panel buttons

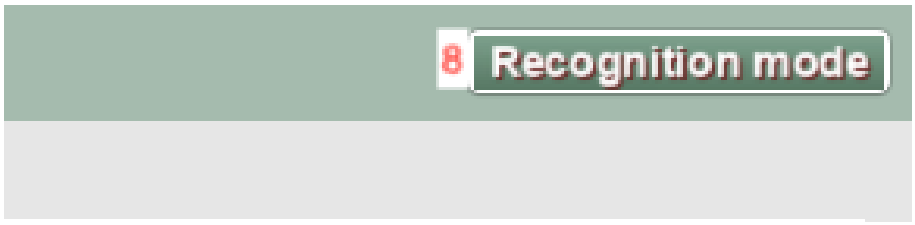


Figure 1.4: Synthesis panel right side

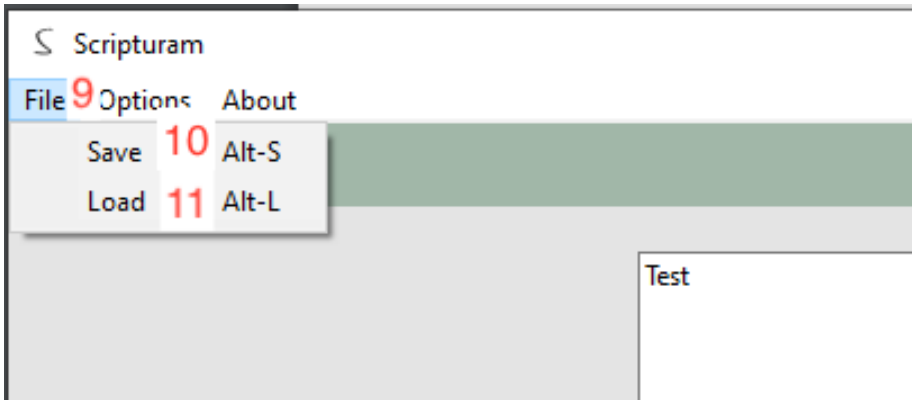


Figure 1.5: Synthesis panel file menu

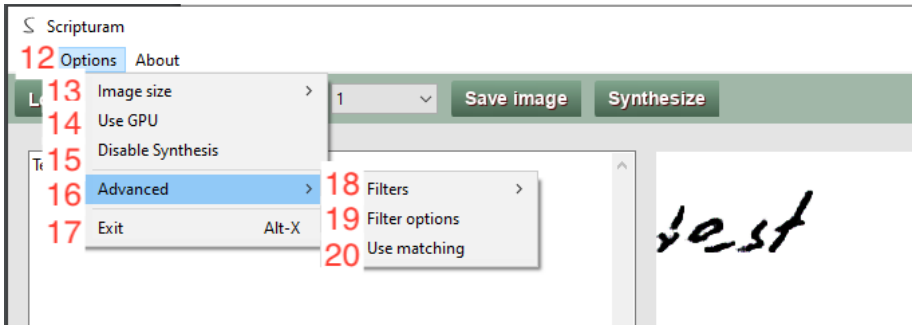


Figure 1.6: Synthesis panel options menu

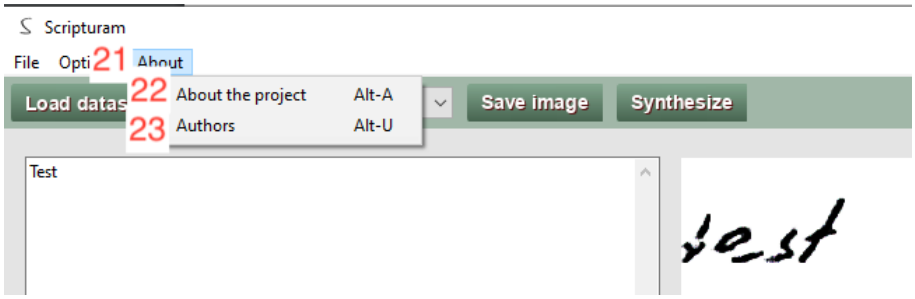


Figure 1.7: Synthesis panel about menu

Menu:

- “File“ menuitem (9):
  - “Save“ menuitem (10): Saves the text from the recongiton panel's textbox as txt file when the recognition mode is active. When the synthesis mode is active, the menuitem saves the image generated with “Synthesize“ button.
  - “Load“ menuitem (11): Loads the content of a text file into the textbox despite the active mode.
- “Options“ menuitem (12):
  - “Image size“ menuitem (enabled only in synthesis mode) (13): Changes the size of the image with handwritten text. There are the avaiable variants (Large, Medium, Small)
  - “Use GPU“ menuitem (14): Determines whether to use GPU for text synthesis and model training.
  - “Disable Synthesis“ menuitem (15): This option turns off the synthesis and the original letters are used.
  - “Advanced“ menuitem (16):
    - “Exit“ menuitem (17): terminates the program and closes the window.
    - “Filters“ menuitem (18): Selects type of filtering control points (which are used for the synthesis). (Original option assumes no filtering)
    - “Filter options“ menuitem (19): Shows a filedialog for filtering options configuration. Grid shape value determines the division of the image with control points. For example, if the value is 5, then the image will be divided into 5x5 equal “cells“. No of points, in turn, is the number of control points that has to be filtered.
    - - “Use matching“ menuitem (20): if enabled, the algorithm uses the control points of another instance of the given letter for synthesis. If not, the letter is generated based on its skeleton and control points.
- “About“ menuitem (21):
  - “About the project“ menuitem (22): a messagebox with general information about the project.
  - “Authors“ menuitem (23): displays a messagebox with information about the authors of the application.

Additionally, information about current processes are displayed on the bar at the bottom of the application window.

Bibliography

[1] Shah Anuj. *Through The Eyes of Gabor Filter*. June 2018. URL: https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97.

[2] *Centroid*. Jan. 2021. URL: https://en.wikipedia.org/wiki/Centroid.

[3] Vincent Christlein, Andreas Maier, Martin Mayr, Anguelos Nikolaou, Mathias Seuret, and Martin Stumpf. *Spatio-Temporal Handwriting Imitation*. Pattern Recognition Lab, Friedrich–Alexander-Universitat Erlangen–Nurnberg. Mar. 2020.

[4] Chong-Nam Chu, Rangasami Kashyap, and Tah-Chih Lee. "Building Skeleton Models via 3-D Medial Surface/Axis Thinning Algorithms". In: *Graphical Models and Image Processing* 56.6 (Nov. 1994), pp. 462–478.

[5] *Gabor filter*. Dec. 2020. URL: https://en.wikipedia.org/wiki/Gabor_filter.

[6] *Generative adversarial network*. Jan. 2021. URL: https://en.wikipedia.org/wiki/Generative_adversarial_network.

[7] *How to use the tools provided to train Tesseract 4.00*. URL: https://tesseract-ocr.github.io/tessdoc/TrainingTesseract-4.00.html.

[8] *Interpolacja wielomianowa*. July 2020. URL: https://pl.wikipedia.org/wiki/Interpolacja_wielomianowa.

[9] J. P. Jones and L. A. Palmer. "An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex". In: *Journal of Neurophysiology* 58.6 (1987), 1233–1258. DOI: 10.1152/jn.1987.58.6.1233.

[10] Kawache. *kawache/Python-B-spline-examples*. Oct. 2015. URL: https://github.com/kawache/Python-B-spline-examples.

[11] Aravind PaiAravind. *ANN vs CNN vs RNN: Types of Neural Networks*. Oct. 2020. URL: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/.

[12] Kaur Paramjeet and Harish Kumar. “A Comparative Study of Iterative Thinning Algorithms for BMP Images”. In: *International Journal of Computer Science and Information Technologies* 2.5 (2011).

[13] Khalid Saed, Adam Szczepański, and Marek Tabedzki. “A modified K3M thinning algorithm”. In: *International Journal of Applied Mathematics and Computer Science* 26.2 (2016), 439–450. DOI: 10.1515/amcs-2016-0031.

[14] *Sigmoid function*. Jan. 2021. URL: https://en.wikipedia.org/wiki/Sigmoid_function.

[15] C. Y. Suen and T. Y Shang. “A Fast Parallel Algorithm for Thinning Digital Patterns”. In: *Communications of the ACM* 27.3 (1984).

[16] *tesseract-ocr/tesseract*. URL: https://github.com/tesseract-ocr/tesseract.

[17] *Tesseract (software)*. Dec. 2020. URL: https://en.wikipedia.org/wiki/Tesseract_(software).

[18] *The Generator | Generative Adversarial Networks | Google Developers*. URL: https://developers.google.com/machine-learning/gan/generator.

[19] *Thresholding (image processing)*. Aug. 2020. URL: https://en.wikipedia.org/wiki/Thresholding_(image_processing).

[20] *Understanding LSTM Networks*. Aug. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

List of Figures

List of tables