NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

POSTGRADUATE STUDIES
“COMPUTING SYSTEMS: SOFTWARE AND HARDWARE”

MASTER THESIS

Exploring Character Pattern Recognition Techniques: A case study for Greek Polytonic Machine-Printed Characters

Rizart A. Dona

Supervisors: **Sergios Theodoridis,** Professor NKUA
Basilis G. Gatos, Researcher NCSR ”Demokritos”

ATHENS

JULY 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
“ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜΑΤΑ: ΛΟΓΙΣΜΙΚΟ ΚΑΙ ΥΛΙΚΟ”

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εξερευνώντας Τεχνικές Αναγνώρισης Προτύπων για Χαρακτήρες: Μια μελέτη περίπτωσης για Ελληνικούς Πολυτονικούς Τυπωμένους Χαρακτήρες

Ριζάρτ Α. Ντόνα

Επιβλέποντες: **Σέργιος Θεοδωρίδης,** Καθηγητής ΕΚΠΑ
Βασίλης Γ. Γάτος, Ερευνητής ΕΚΕΦΕ ”Δημόκριτος”

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2019

MASTER THESIS

Exploring Character Pattern Recognition Techniques: A case study for Greek Polytonic Machine-Printed Characters

Rizart A. Dona

S.N.: M1528

SUPERVISORS: **Sergios Theodoridis,** Professor NKUA
Basilis G. Gatos, Researcher NCSR ”Demokritos”

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εξερευνώντας Τεχνικές Αναγνώρισης Προτύπων για Χαρακτήρες: Μια μελέτη περίπτωσης για Ελληνικούς Πολυτονικούς Τυπωμένους Χαρακτήρες

Ριζάρτ Α. Ντόνα
Α.Μ.: Μ1528

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Σέργιος Θεοδωρίδης,** Καθηγητής ΕΚΠΑ
Βασίλης Γ. Γάτος, Ερευνητής ΕΚΕΦΕ ”Δημόκριτος”

ABSTRACT

In this thesis we explore various character pattern recognition techniques and we present a case study for Greek polytonic machine-printed characters where those techniques are applicable. We implement and describe statistical feature engineering techniques such as character zoning, adaptive character zoning, extraction of horizontal and vertical projection histograms as well as a feature extraction technique based on recursive subdivisions of the character. We also implement and discuss two classification techniques, one based on the template matching model and the other one based on artificial neural networks. Additionally, the python-based open source library that implements those functionalities is presented along with a how-to-use section. Finally, we evaluate the aforementioned techniques on two separate datasets that contain Greek polytonic characters and we present our results on the performance of our methods.

SUBJECT AREA: Character Pattern Recognition

KEYWORDS: Optical Character Recognition, Pattern Recognition, Feature Extraction, Character Classification, Artificial Neural Networks, Greek Polytonic Characters

ΠΕΡΙΛΗΨΗ

Σε αυτη την διπλωματική εργασία εξερευνούμε διάφορες τεχνικές αναγνώρισης προτύπων για χαρακτήρες και παρουσιάζουμε μια μελέτη περίπτωσης για Ελληνικούς πολυτονικούς τυπωμένους χαρακτήρες όπου οι τεχνικές αυτές είναι εφαρμόσιμες. Υλοποιούμε και περιγράφουμε στατιστικές τεχνικές μηχανικής χαρακτηριστικών (feature engineering) όπως είναι ο διαχωρισμός του χαρακτήρα σε ζώνες, ο διαχωρισμός του χαρακτήρα σε προσαρμοστικές ζώνες, η εξαγωγή ιστογραμμάτων κάθετων και οριζόντιων προβολών καθώς και μια τεχνική εξαγωγής χαρακτηριστικών που βασίζεται σε αναδρομικές υποδιαιρέσεις του χαρακτήρα. Επιπλέον, υλοποιούμε και συζητάμε δύο τεχνικές κατηγοριοποίησης, η μια βασίζεται στο μοντέλο του ταιριάσματος προτύπου (template matching) και η άλλη βασίζεται στα τεχνητά νευρωνικά δίκτυα. Επιπρόσθετα, παρουσιάζουμε την υλοποιημένη σε python βιβλιοθήκη ανοικτού κώδικα που διεκπεραιώνει αυτές τις λειτουργίες μαζί με μια ενότητα για το πώς να την χρησιμοποιήσει κάποιος. Τέλος, αξιολογούμε τις προαναφερθείσες τεχνικές σε δύο διαφορετικά σύνολα δεδομένων που περιέχουν Ελληνικούς πολυτονικούς χαρακτήρες και παρουσιάζουμε τα αποτελέσματα μας για όσον αφορά την απόδοση των μεθόδων μας.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αναγώριση Προτύπων για Χαρακτήρες

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Οπτική Αναγνώριση Χαρακτήρων, Αναγνώριση Προτύπων, Εξαγωγή Χαρακτηριστικών, Κατηγοριοποίηση Χαρακτήρων, Τεχνητά Νευρωνικά Δίκτυα, Ελληνικοί Πολυτονικοί Χαρακτήρες

ACKNOWLEDGEMENTS

I would like to thank researcher Dr. Basilis Gatos for our excellent collaboration and PhD candidate Panagiotis Kaddas for his contribution and his helpful reviews. I would also like to thank professor Sergios Theodoridis for giving me the chance to participate on this project.

July 2019

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This Master thesis is a research project that took place in Athens, Greece by postgraduate student Rizart Dona under the supervision of researcher Basilis Gatos, in the context of the CIL [7] research interests.

1. INTRODUCTION

1.1 Background

Optical character recognition (OCR) is the process of classification of optical patterns contained in a digital image corresponding to alphanumeric or other characters (machine-printed or handwritten). The character recognition is achieved through important steps of segmentation, feature extraction and classification. OCR technology enables us to convert different types of documents such as scanned paper documents, PDF files or images captured by a digital camera into editable and searchable data.

Each OCR system has a basic architecture that determines the steps that are needed in order to recognize and classify the desired characters. The typical OCR system architecture is illustrated in Figure 1.1. Some of the steps follow:

Optical Scanning: This is the first step of the workflow, here through a scanning process the digital image of the original document is captured. The scanner consists of a transport mechanism and a sensing device that converts light intensity into grey levels, thus giving us a digital representation of the input document.

Location Segmentation: Location segmentation determines constituents of an image. It is necessary to locate regions of a document which have printed data and are distinguished from figures and graphics. Those constituents could be lines of text, words in those lines and finally the segmentation of individual characters. Some problems that may arise in this process are distinguishing noise from text or misinterpreting graphics and geometry with text and vice versa.

Preprocessing: The raw data depending on the data acquisition type is subjected to a number of preliminary processing steps to make it usable in the descriptive stages of character analysis. Some preprocessing steps include:

Binarization - Thresholding: This is a very common step that is needed in order to remove noise from the characters and speedup the classification process in the future steps. What happens here essentially is that the grey-level document is transformed into a black-white image (containing only 0 and 1 as pixel values, 0 for background and 1 for foreground content) through a process that is called thresholding. The simplest thresholding methods traverse each pixel value in the grey-level document and given a threshold they check whether the given value is over or under the threshold, if it is found to be bellow the threshold then it is assigned the background content pixel value (0) and respectively if it is found to be above the threshold then it is assigned the foreground content pixel value (1).

Noise reduction: The noise introduced by the optical scanning device or the writing instrument causes disconnected line segments, bumps and gaps in lines, filled loops, etc. The distortion including local variations, rounding of corners, dilation and erosion is a potential problem. It is necessary to eliminate these imperfections prior to actual processing the data.

Slant normalization: In the case of handwritten documents, one of the measurable factors of different handwriting styles is the slant angle between longest stroke in a word and the vertical direction. Slant normalization is used to normalize all characters to a standard form.

Skew normalization and baseline extraction: Due to inaccuracies in the scanning process and writing style (for handwritten input) the writing may be slightly tilted or curved within the image. This can hurt the effectiveness of the algorithms and thus should be detected and corrected. Additionally, some characters are distinguished according to the relative position with respect to the baseline, such as 9 and g. The methods of baseline extraction include using the projection profile of the image, nearest neighbor clustering and the cross correlation method between lines and the Hough transform. After skew detection the character or word is translated to the origin, rotated or stretched until the baseline is horizontal and retranslated back into the display screen space.

Size Normalization: This step is used to adjust the character size to a certain standard. The OCR methods may apply for both horizontal and vertical size normalizations. This process is essential for the future training and recognition of the characters because most of the recognition techniques require a standard size in order to function properly.

Representation - Feature Extraction: The image representation plays one of the most important roles in any recognition system. In the simplest case, gray level or binary images are fed to a recognizer. However, in most of the recognition systems in order to avoid extra complexity and to increase the accuracy of the algorithms, a more compact and characteristic representation is required. For this purpose, a set of features is extracted for each class that helps distinguish it from other classes while remaining invariant to characteristic differences within the class. This step is one of the two main subjects of this thesis concerning pattern recognition techniques and several examples are presented in section 2.1.

Training and Recognition: OCR systems extensively use the methodologies of pattern recognition which assigns an unknown sample into a predefined class. In this step an OCR system can employ several approaches for training the labeled characters[1] and for predicting new unlabeled characters[2] which the model has never seen before. This step is the second main subject of this thesis concerning pattern recognition techniques and two approaches are presented in section 2.2.

A more comprehensive description and analysis of the components of an OCR system can be found in [6].

[1] *characters which their class is known*
[2] *characters which their class is unknown*

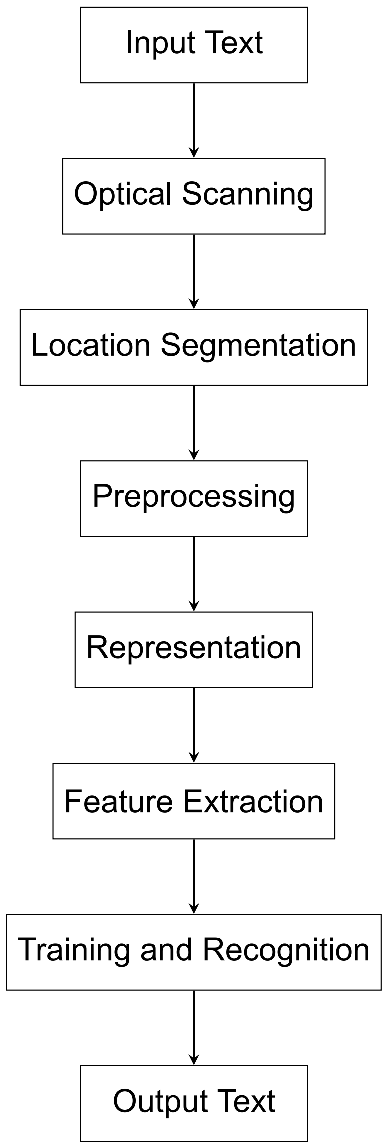


Figure 1.1: The components of an OCR system

1.2 Motivation

As we saw in the previous section, OCR systems contain multiple components that facilitate the character pattern recognition process. The motivation for this thesis is to survey feature extraction techniques and pattern recognition techniques that can help us get good results in our recognition tasks.

Our case study for Greek polytonic machine-printed characters is another challenge due to the large number of existing character classes which cannot be handled sufficiently by current OCR technologies. Taking into account that the Greek polytonic system was used from the late antiquity until recently, a large amount of scanned Greek documents still remains without full text search capabilities. This project also aims at progressing the research in that direction.

1.3 Overview

The rest of this thesis is organized as follows: In chapter 2 the feature extraction techniques are described along with the pattern recognition algorithms. In chapter 3 the implementation of those techniques is presented, namely the python library and a how-to-use section about it. In chapter 4 we discuss the experimental results on the performance of our techniques on two sets of data that contain Greek polytonic machine-printed characters. Some related work is mentioned in chapter 5 and some future work is discussed in chapter 6. Finally, we conclude in chapter 7.

2. METHODOLOGY

2.1 Feature Engineering

In this section we describe and analyze three feature extraction techniques that can be used to extract feature vectors from a character.

2.1.1 Projection Histograms

Projection histograms have been used in OCR systems since 1956 [11]. This idea is also known as histogram projection count and can be represented as $H_i = \sum_j f(i,j)$ for a horizontal projection where $f(i,j)$ is the pixel value of ith row and jth column of the image. Here, the background pixel is considered to be 0 and the foreground pixel to be 1. Similarly, a vertical projection histogram can be calculated as $H_j = \sum_i f(i,j)$. We basically calculate the pixel density for each projection.

We can see an example in Figure 2.1, here the vertical and horizontal projection histograms of the character ”5” have been calculated. The feature vector of a given character combines both vertical and horizontal values.

In this method, we can select the number of projections that we want to extract from our character. For example, if we choose 5 projections we will have 5 vertical and 5 horizontal values for our feature vector, 10 values in total. Our character will be divided in 5 equal horizontal and vertical sections and for each section the pixel density will be calculated. In the same example, if our character dimension is 10x10 it will indicate that each projection will consist of 2 rows and 2 columns.

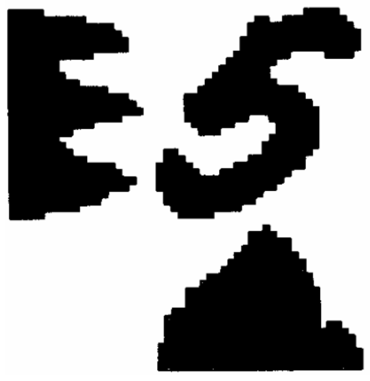


Figure 2.1: Horizontal and vertical projection histograms of a character

2.1.2 Zones

The commercial OCR system named Calera which is reported in [5] was developed based on a zonal feature extraction method. In order to extract this feature, an image is divided into same-size non overlapping zones and then the number of foreground pixels is counted for each of those zones. In that manner the pixel density is computed for each zone.

In Figure 2.2a we can see an example. Our character is divided into zones that each has dimensions of 10x10. For each zone we calculate the pixel density and thus the feature vector consists of those counts (36 values in this case, since we have 6x6 number of zones).

Adaptive Zones

One alternative technique that involves zoning is reported in [9], namely adaptive zones. Adaptive zoning features are extracted after adjusting the position of every zone based on local pattern information. This adjustment is performed by moving every zone towards the pattern body and is based on the maximization of the local pixel density around each zone.

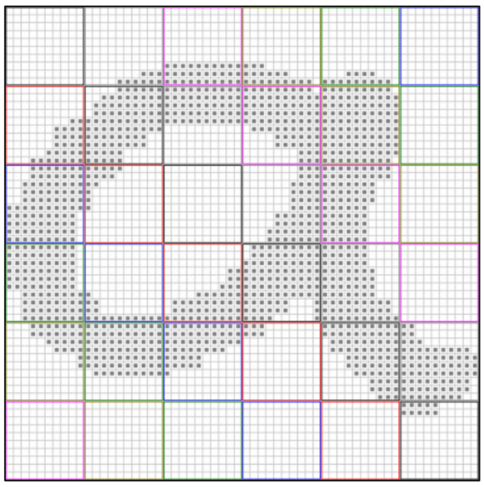
In more detail, given a character C where $C(i, j)$ is the pixel value of ith row and jth column, and coordinates $z^{x_0}, z^{x_1}, z^{y_0}, z^{y_1}$ for a particular zone z, and parameters λ_x, λ_y, we reallocate that zone by computing the offsets as follows:

$$(d_x, d_y) = \textit{arg max}_{x\in[-\lambda_x \dots \lambda_x], y\in[-\lambda_y \dots \lambda_y]} \sum_{i=z^{x_0}}^{z^{x_1}} \sum_{j=z^{y_0}}^{z^{y_1}} C(x+i, y+j) \tag{2.1}$$

Parameters λ_x and λ_y define the horizontal and vertical range for adjusting the position of the zones. Since large values for parameters λ_x and λ_y can affect the computational time needed for feature extraction, we propose that these parameters have to range between 1 and 3. Finally, the new coordinates for zone z are:

$$\begin{aligned} z^{x'_0} &= z^{x_0} + d_x \\ z^{x'_1} &= z^{x_1} + d_x \\ z^{y'_0} &= z^{y_0} + d_y \\ z^{y'_1} &= z^{y_1} + d_y \end{aligned} \tag{2.2}$$

So, once we find the new coordinates for each zone we calculate the pixel density like in the typical zoning case and we extract our features like before. One example is shown on Figure 2.2b where the zones from 2.2a have been adjusted.


(a) Non overlapping zones for a character

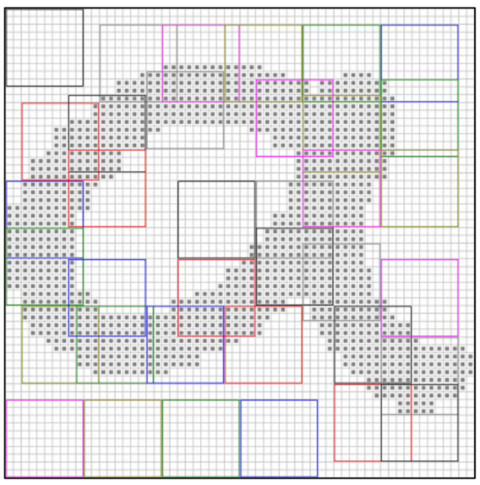

(b) Overlapping adjusted zones for a character

Figure 2.2: Character zoning

2.1.3 Recursive Subdivisions

This feature extraction method [18] [19] relies on a technique based on recursive subdivisions of the image as well as on calculation of the centre of masses of each sub-image with sub-pixel accuracy. It is based on structural features extracted directly from the character, that provide a good representation of the character at different levels of granularity.

More specifically, let $C(x, y)$ be the character image array having 1s for foreground and 0s for background pixels and x_{max} and y_{max} be the width and the height of the character image. This method relies on recursive sub-divisions of the character image based on the centre of mass of each sub-image, so, we initially calculate the co-ordinates (x_0, y_0) of the centre of mass of the initial character image. The vertical co-ordinate x_0 is found according to the following procedure:

1. Let V_0 be the vertical projection histogram array of the initial character image with size x_{max}.
2. Create V_1 array from V_0 as follows:

```
for $x := 1$ to $2 * x_{max}$ do
  if $x$ mod $2 = 1$ then
    $V_1[x] = 0$
  else
    $V_1[x] = V_0[x \text{ div } 2]$
  end if
end for
```

3. Find x_q from V_1 using the following equation:

$$x_q = \arg\min_{x_t} \left\{ \sum_{x=1}^{x=x_i-1} V_1(x) - \sum_{x=x_t+1}^{x=2^* x_{\max}} V_1(x) \right\} \tag{2.3}$$

4. The vertical co-ordinate x_0 is then estimated as:

$$x_0 = x_q \text{ div } 2 \tag{2.4}$$

As already mentioned, in order to improve the precision, the centre of mass for each of the following sub-images is calculated with sub-pixel accuracy. That is, the initial image is divided vertically into two rectangular sub-images depending on the value of x_q (Eq. 2.3). If x_q mod $2 = 0$ then the vertex co-ordinates of these two sub-images are: $\{(1, 1), (x_0, y_{max})\}$ and $\{(x_0, 1), (x_{max}, y_{max})\}$. Otherwise, if x_q mod $2 = 1$, then the vertex co-ordinates are: $\{(1, 1), (x_0, y_{max})\}$ and $\{(x_0 + 1, 1), (x_{max}, y_{max})\}$. An example of the aforementioned procedure is shown on Figure 2.3.

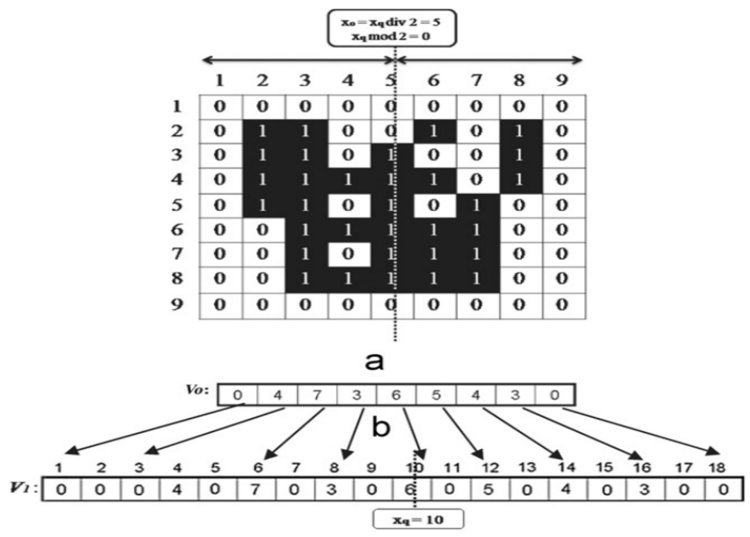


Figure 2.3: Extraction of coordinates for the subdivision procedure

Likewise, the horizontal co-ordinate y_0 is calculated thus resulting to the division of the initial image into four rectangular sub-images. The whole procedure is applied recursively for every sub-image (Figure 2.4).

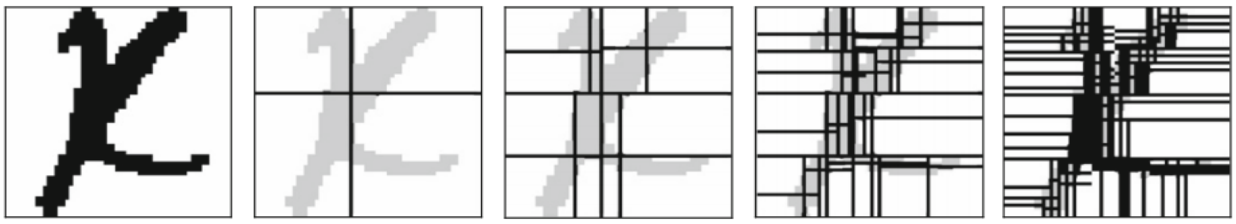


Figure 2.4: Recursive subdivisions of a character

Let L be the current level of the granularity. At this level the number of the sub-images is 4^{L+1}, for example, when $L = 0$ the number of sub-images is 4. The number of the center of masses at level L equals to 4^L. At level L, the co-ordinates (x, y) of all the centre of masses become the features of the character. So, for every L a 2×4^L - dimensional feature vector is extracted. As Figure 2.5 shows, the larger the L the better representation of the character is obtained.

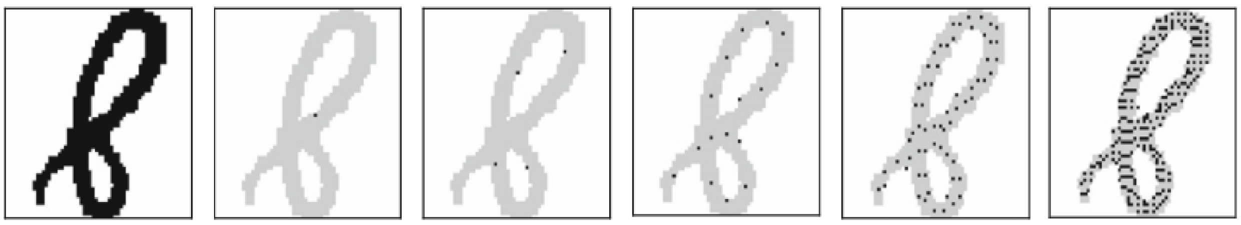


Figure 2.5: Division points (center of masses) of a character

2.2 Classifiers

In this section, we present two pattern recognition models that can be trained and used to predict the class of previously unseen characters.

2.2.1 Template Matching

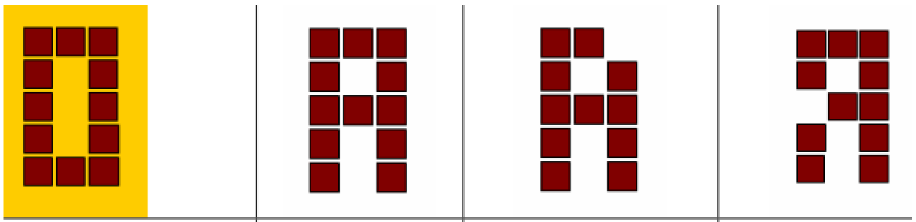
This pattern recognition technique considers the character body as the feature vector of the character [14]. If we want to compare two same-sized characters C1 and C2 (C1 is the template character and C2 is the character that we want to predict) that consist of n pixels each, then the count of points n_{ij} where character C1 has value i and character C2 has value j, with $i, j \; \varepsilon \; \{0, 1\}$, is given by the following equation:

$$n_{ij} = \sum_{m=1}^{n} \delta_m(i, j) \tag{2.5}$$

where:

$$\delta_m(i, j) = \begin{cases} 1, \text{ if } (x_m = i) \wedge (y_m = j) \\ 0, \text{ otherwise} \end{cases} \tag{2.6}$$

and x_m, y_m are the mth pixels that are compared in characters C1 and C2. We can see an example of those computations in Figure 2.6.



n_{11}	11	10	10
n_{10}	1	2	2
n_{01}	1	1	1
n_{00}	2	2	2

Figure 2.6: Calculation of n_{ij} values for template matching

Given those values, we need a distance function that can indicate the similarity of the two characters. Two well known distances are mentioned in [17], namely Jaccard distance and Yule distance. Their equations follow:

$$d_J = \frac{n_{11}}{n_{11} + n_{10} + n_{01}} \tag{2.7}$$

$$d_Y = \frac{n_{11}n_{00} - n_{10}n_{01}}{n_{11}n_{00} + n_{10}n_{01}} \tag{2.8}$$

For a given pair of characters, the proximity of those distances to 1 indicates the proximity between the characters. Basically, for each unlabeled character that we want to predict it's class we need to compute one of those distances with each labeled character and keep the closest one as the candidate class.

2.2.2 Artificial Neural Networks

Artificial neural networks (ANNs) are inspired by the biological neural networks that constitute animal brains. They are learning systems that perform tasks by considering examples, generally without being programmed with any task-specific rules [20]. An ANN is based on a collection of connected units or nodes called artificial neurons. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.

There are different types of ANNs such as Multilayer Perceptrons (MLPs) [25], Convolutional Neural Networks (CNNs) [21], Recurrent Neural Networks (RNNs) [26], and others. In the context of this thesis we present a typical MLP that performs the pattern recognition task. An example of a MLP is shown in Figure 2.7

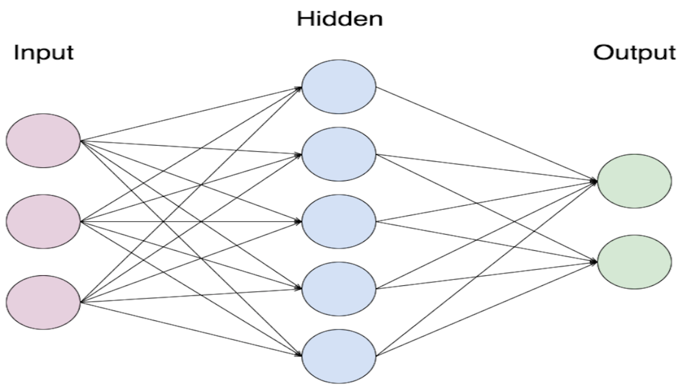


Figure 2.7: An example ANN (MLP)

Before we present our model we first need to mention some of the basics of ANNs:

Neuron: Just like a neuron forms the basic element of our brain, a neuron forms the basic structure of a neural network. Like when we get the information, we process it and then we generate an output, similarly, in case of a neural network, a neuron receives an input, processes it and generates an output which is either sent to other neurons for further processing or it is the final output.

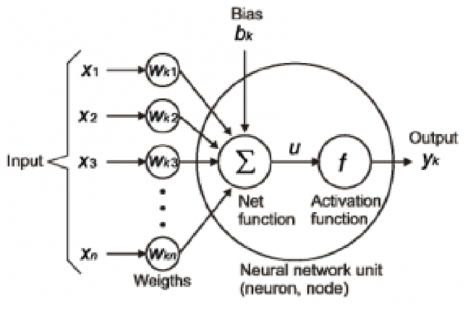
Input / Output / Hidden Layer: As the name suggests the input layer is the one which receives the input and is essentially the first layer of the network. The output layer is the one which generates the output or is the final layer of the network. The processing layers are the hidden layers within the network. These hidden layers are the ones which perform specific tasks on the incoming data and pass on the output generated by them to the next layer. The input and output layers are the ones visible to us, while the intermediate layers are hidden. We already saw this structure in Figure 2.7.

Weights: When input enters the neuron, it is multiplied by a weight. For example, if a neuron has two inputs, then each input will have has an associated weight assigned to it. We initialize the weights randomly and these weights are updated during the model training process. The neural network after training assigns a higher weight to the input it considers more important as compared to the ones which are considered less important. A weight of zero denotes that the particular feature is insignificant. If we assume the input to be a, and the weight associated to be w_1, then after passing through the node the input becomes $a \times w_1$.

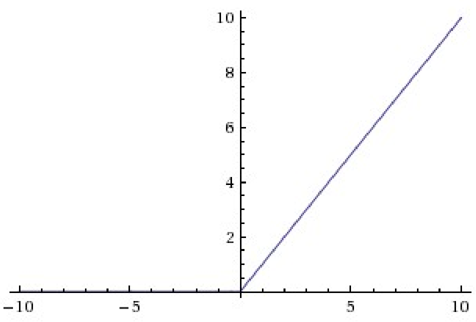
Bias: In addition to the weights, another linear component is applied to the input, called as the bias. It is added to the result of weight multiplication to the input. The bias is basically added to change the range of the weight multiplied input. After adding the bias, the result would look like $(a \times w_1) + bias$. This is the final linear component of the input transformation.

Activation Function: Once the linear component is applied to the input, a non-linear function is applied to it. This is done by applying the activation function to the linear combination.The activation function translates the input signals to output signals. The output after application of the activation function would look something like $f((a \times w_1) + bias)$ where $f()$ is the activation function.

For example, in Figure 2.8a we have n inputs given as X_1 to X_n and corresponding weights W_{k1} to W_{kn}. We have a bias given as bk. The weights are first multiplied to its corresponding input and are then added together along with the bias and we have the value u computed as $u = \sum((w \times x) + b)$. The activation function is applied to u i.e. $f(u)$ and we receive the final output from the neuron as $y_k = f(u)$. In our case, the activation function is the Rectified Linear Units (ReLU) function that is defined as $f(x) = max(x, 0)$ and is shown in Figure 2.8b.



(a) Activation function output



(b) ReLU function

Figure 2.8: Activation functions

For our output layer we use the Softmax activation function [27]. This function makes it easy to assign values to each class which can be easily interpreted as probabilities. For example, if we need to classify elements that belong to 5 categories, this function will output a probability for each category and we will select the category with the highest probability.

Forward Propagation: Forward Propagation refers to the movement of the input through the hidden layers to the output layers. In forward propagation, the information travels in a single direction forward. The input layer supplies the input to the hidden layers and then the output is generated. There is no backward movement.

Cost Function: When we build a network, the network tries to predict the output as close as possible to the actual value. We measure this accuracy of the network using the cost/loss function. The cost or loss function tries to penalize the network when it makes errors. Our objective while running the network is to increase our prediction accuracy and to reduce the error, hence minimizing the cost function. The most optimized output is the one with least value of the cost or loss function. In our case, the categorical crossentropy (Eq. 2.9) loss function is used where N is the number of observations and $p_{model}[y_i \in C_c]$ is the probability predicted by the model for the ith observation to belong to the cth category.

$$-\frac{1}{N}\sum_{i=1}^{N} \log p_{\text{model}}\left[y_i \in C_{y_i}\right] \tag{2.9}$$

Gradient Descent: Gradient descent [22] is an optimization algorithm for minimizing the cost. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point. If, instead, one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent. We use this algorithm to minimize the loss function.

Learning Rate: The learning rate is defined as the amount of minimization in the cost function in each iteration. In simple terms, the rate at which we descend towards the minima of the cost function is the learning rate. We should choose the learning rate very carefully since it should neither be very large that the optimal solution is missed and nor should be very low that it takes forever for the network to converge.

Back-propagation: When we define a neural network, we assign random weights and bias values to our nodes. Once we have received the output for a single iteration, we can calculate the error of the network. This error is then fed back to the network along with the gradient of the cost function to update the weights of the network. These weights are then updated so that the errors in the subsequent iterations is reduced. This updating of weights using the gradient of the cost function is known as back-propagation. In back-propagation the movement of the network is backwards, the error along with the gradient flows back from the out layer through the hidden layers and the weights are updated.

Batches While training a neural network, instead of sending the entire input in one go, we divide in input into several chunks of equal size randomly. Training the data on batches makes the model more generalized as compared to the model built when the entire data set is fed to the network in one go.

Epochs An epoch is defined as a single training iteration of all batches in both forward and back propagation. This means 1 epoch is a single forward and backward pass of the entire input data. It's highly likely that more number of epochs would show higher accuracy of the network, however, it would also take longer for the network to converge. Also, if the number of epochs is too high, then the network might over-fit the training data.

Dropout Dropout is a regularization technique which prevents over-fitting of the network. As the name suggests, during training a certain number of neurons in the hidden layer is randomly dropped. This means that the training happens on several architectures of the neural network on different combinations of the neurons. We can think of dropout as an ensemble technique, where the output of multiple networks is then used to produce the final output. An example can be seen in Figure 2.9

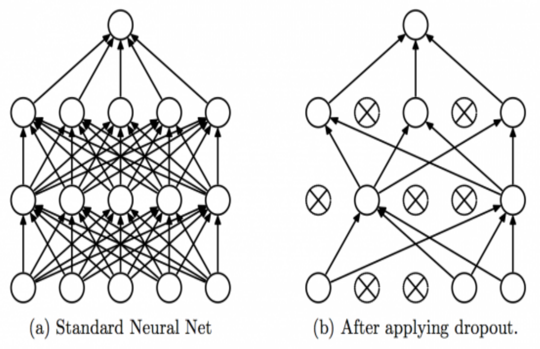


Figure 2.9: A dropout example

Now that we described the basics of ANNs we can present our model setup. The layers of our model follow:

- **1st Layer (Input Layer):** This is the input layer, here a 2-d image is flatten and we end up with a 1-d array that contains the character pixel values.
- **2nd Layer (1st Hidden):** This layer consists of 784 neurons and it uses ReLU as the activation function. This layer is densely connected, meaning that each neuron receives input from all the neurons in the previous layer (in this case the input layer).
- **3rd Layer (2nd Hidden):** This layer is a dropout layer that randomly drops 20% of the previous layer's neurons.
- **4th Layer (3rd Hidden):** This layer consists of 2000 neurons and it uses ReLU as the activation function. This layer is also densely connected.
- **5th Layer (4th Hidden):** This layer is a dropout layer that randomly drops 50% of the previous layer's neurons.
- **6th Layer (5th Hidden):** This layer consists of 2000 neurons and it uses ReLU as the activation function. This layer is also densely connected.
- **7th Layer (6th Hidden):** This layer is a dropout layer that randomly drops 50% of the previous layer's neurons.
- **8th Layer (Output Layer):** This layer consists of neurons that are as many as the number of categories that we need to predict and it uses Softmax as the activation function. This layer is also densely connected. This is the final layer that will eventually produce the predictions for our characters.

Beyond this fixed model setup we need to choose the number of epochs, the number of batches as well as a learning rate value in order to perform the recognition task.

3. IMPLEMENTATION

In this chapter we describe the implementation of the techniques that we saw in chapter 2. We present the python library design that implements those functionalities and a how-to-use section is included for researchers/developers who want to use it.

3.1 The Python Library

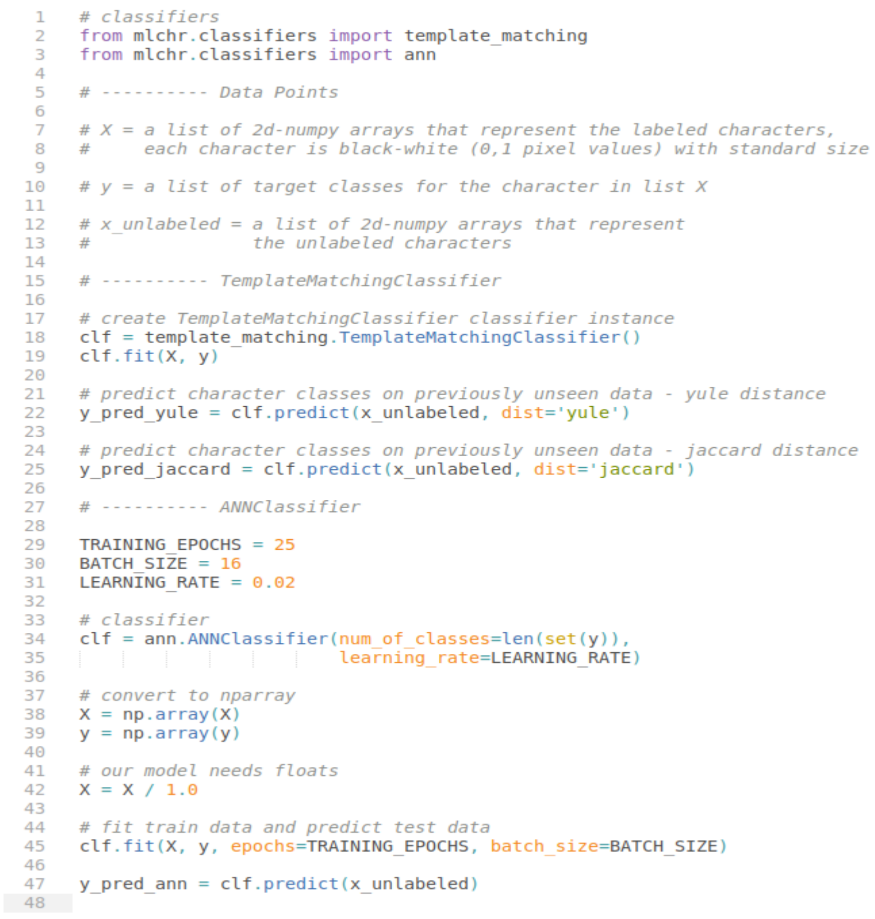
The developed library (**mlchr**) is open source [15] and was designed in the context of the NumPy [1], SciPy [3] and scikit-learn [2] ecosystem. For the ANN implementation TensorFlow [4] was used. The mlchr package has 4 modules that structure it's capabilities. Those are:

- The **feature_extraction** module: This module contains the feature extraction techniques that we presented in section 2.1.
- The **classifiers** module: This module contains the pattern recognition techniques that we presented in section 2.2.
- The **normalization** module: This module contains a character image size normalizer.
- The **utils** module: This module contains various utilities for reading images, extracting statistics, etc.

Essentially, the core functionalities are in the first two modules. In the next section we are going to see how a user is able to use those modules in order to perform character pattern recognition tasks.

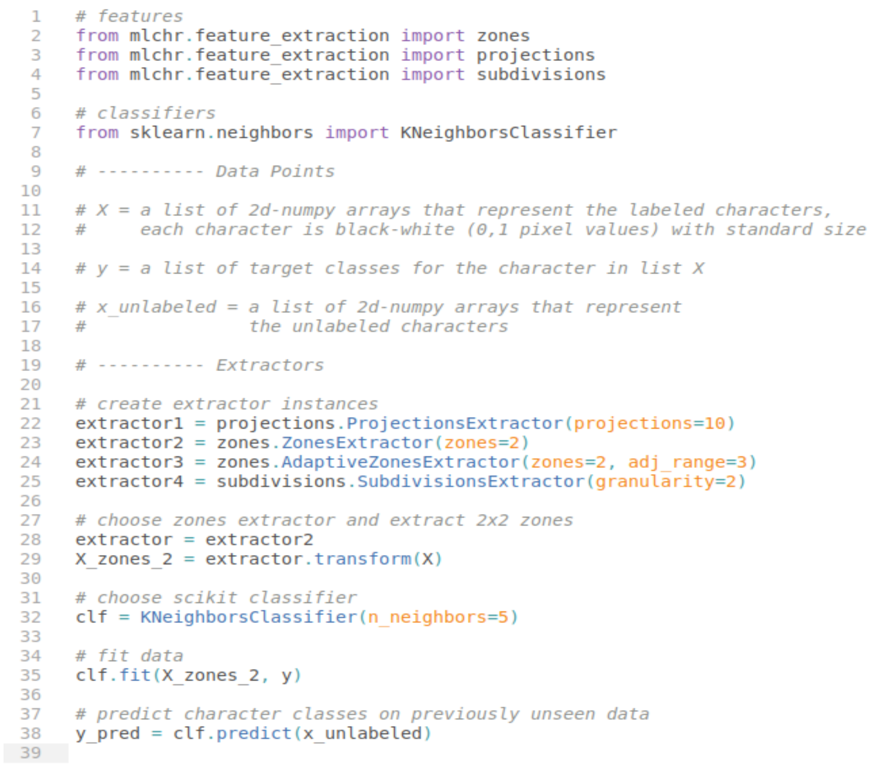
3.2 How To Use

The library facilitates simple imports with a scikit-like abstraction paradigm and the user can use it with very few lines of code. We can see two examples in Figure 3.1 and in Figure 3.2. In the first case, we use the two pattern recognition techniques, via the TemplateMatchingClassifier instance and the ANNClassifier instance. In the second case, we show how feature extraction is performed and how one can train and use for prediction a scikit-learn classifier with that extracted data.



```python
# classifiers
from mlchr.classifiers import template_matching
from mlchr.classifiers import ann

# ---------- Data Points

# X = a list of 2d-numpy arrays that represent the labeled characters,
#     each character is black-white (0,1 pixel values) with standard size

# y = a list of target classes for the character in list X

# x_unlabeled = a list of 2d-numpy arrays that represent
#                the unlabeled characters

# ---------- TemplateMatchingClassifier

# create TemplateMatchingClassifier classifier instance
clf = template_matching.TemplateMatchingClassifier()
clf.fit(X, y)

# predict character classes on previously unseen data - yule distance
y_pred_yule = clf.predict(x_unlabeled, dist='yule')

# predict character classes on previously unseen data - jaccard distance
y_pred_jaccard = clf.predict(x_unlabeled, dist='jaccard')

# ---------- ANNClassifier

TRAINING_EPOCHS = 25
BATCH_SIZE = 16
LEARNING_RATE = 0.02

# classifier
clf = ann.ANNClassifier(num_of_classes=len(set(y)),
                        learning_rate=LEARNING_RATE)

# convert to nparray
X = np.array(X)
y = np.array(y)

# our model needs floats
X = X / 1.0

# fit train data and predict test data
clf.fit(X, y, epochs=TRAINING_EPOCHS, batch_size=BATCH_SIZE)

y_pred_ann = clf.predict(x_unlabeled)
```

Figure 3.1: Classification code example



```python
# features
from mlchr.feature_extraction import zones
from mlchr.feature_extraction import projections
from mlchr.feature_extraction import subdivisions

# classifiers
from sklearn.neighbors import KNeighborsClassifier

# ---------- Data Points

# X = a list of 2d-numpy arrays that represent the labeled characters,
#     each character is black-white (0,1 pixel values) with standard size

# y = a list of target classes for the character in list X

# x_unlabeled = a list of 2d-numpy arrays that represent
#                the unlabeled characters

# ---------- Extractors

# create extractor instances
extractor1 = projections.ProjectionsExtractor(projections=10)
extractor2 = zones.ZonesExtractor(zones=2)
extractor3 = zones.AdaptiveZonesExtractor(zones=2, adj_range=3)
extractor4 = subdivisions.SubdivisionsExtractor(granularity=2)

# choose zones extractor and extract 2x2 zones
extractor = extractor2
X_zones_2 = extractor.transform(X)

# choose scikit classifier
clf = KNeighborsClassifier(n_neighbors=5)

# fit data
clf.fit(X_zones_2, y)

# predict character classes on previously unseen data
y_pred = clf.predict(x_unlabeled)
```

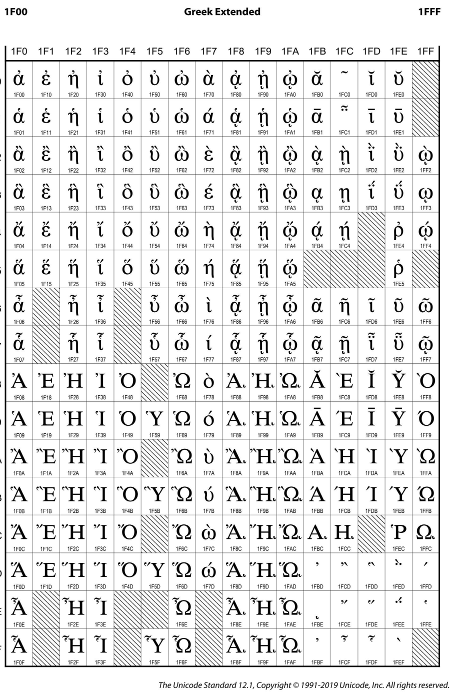
Figure 3.2: Feature extraction code example

4. EXPERIMENTAL RESULTS

In this chapter we present the experimental results for our methods on two sets of data that contain Greek polytonic machine-printed characters. In section 4.1, we describe the data and experimental setup that is used and in sections 4.2, 4.3 we present the results for each set of data.

4.1 Experimental Setup

Our evaluation data comes from two sources. The first set of data (from now on set1) comes from 21 separate book pages that contain Greek polytonic machine-printed characters. The second set of data (from now on set2) comes from GROPOLY-DB [10], a publicly available old Greek polytonic database with the same characteristics as set1. The Greek polytonic language contains more than 270 character classes as can be seen in Figure 4.1a. A sample page that illustrates the quality of our characters can be seen in Figure 4.1b. All characters, in both sets, are already binarized meaning that they have pixel values of 0 (for background content) and 1 f(or foreground content).

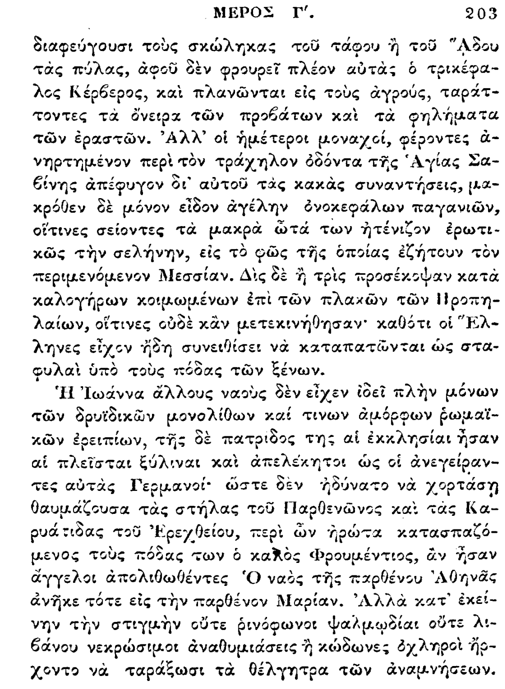


1F00 Greek Extended 1FFF

	1F0	1F1	1F2	1F3	1F4	1F5	1F6	1F7	1F8	1F9	1FA	1FB	1FC	1FD	1FE	1FF
0	ἀ 1F00	ἐ 1F10	ἠ 1F20	ἰ 1F30	ὀ 1F40	ὐ 1F50	ὠ 1F60	ὰ 1F70	ᾀ 1F80	ᾐ 1F90	ᾠ 1FA0	ᾰ 1FB0	῀ 1FC0	ῐ 1FD0	ῠ 1FE0	▨
1	ἁ 1F01	ἑ 1F11	ἡ 1F21	ἱ 1F31	ὁ 1F41	ὑ 1F51	ὡ 1F61	ά 1F71	ᾁ 1F81	ᾑ 1F91	ᾡ 1FA1	ᾱ 1FB1	῁ 1FC1	ῑ 1FD1	ῡ 1FE1	▨
2	ἂ 1F02	ἒ 1F12	ἢ 1F22	ἲ 1F32	ὂ 1F42	ὒ 1F52	ὢ 1F62	ὲ 1F72	ᾂ 1F82	ᾒ 1F92	ᾢ 1FA2	ᾲ 1FB2	ῂ 1FC2	ῒ 1FD2	ῢ 1FE2	ῲ 1FF2
3	ἃ 1F03	ἓ 1F13	ἣ 1F23	ἳ 1F33	ὃ 1F43	ὓ 1F53	ὣ 1F63	έ 1F73	ᾃ 1F83	ᾓ 1F93	ᾣ 1FA3	ᾳ 1FB3	ῃ 1FC3	ΐ 1FD3	ΰ 1FE3	ῳ 1FF3
4	ἄ 1F04	ἔ 1F14	ἤ 1F24	ἴ 1F34	ὄ 1F44	ὔ 1F54	ὤ 1F64	ὴ 1F74	ᾄ 1F84	ᾔ 1F94	ᾤ 1FA4	ᾴ 1FB4	ῄ 1FC4	▨	ῤ 1FE4	ῴ 1FF4
5	ἅ 1F05	ἕ 1F15	ἥ 1F25	ἵ 1F35	ὅ 1F45	ὕ 1F55	ὥ 1F65	ή 1F75	ᾅ 1F85	ᾕ 1F95	ᾥ 1FA5	▨	▨	▨	ῥ 1FE5	▨
6	ἆ 1F06	▨	ἦ 1F26	ἶ 1F36	▨	ὖ 1F56	ὦ 1F66	ὶ 1F76	ᾆ 1F86	ᾖ 1F96	ᾦ 1FA6	ᾶ 1FB6	ῆ 1FC6	ῖ 1FD6	ῦ 1FE6	ῶ 1FF6
7	ἇ 1F07	▨	ἧ 1F27	ἷ 1F37	▨	ὗ 1F57	ὧ 1F67	ί 1F77	ᾇ 1F87	ᾗ 1F97	ᾧ 1FA7	ᾷ 1FB7	ῇ 1FC7	ῗ 1FD7	ῧ 1FE7	ῷ 1FF7
8	Ἀ 1F08	Ἐ 1F18	Ἠ 1F28	Ἰ 1F38	Ὀ 1F48	▨	Ὠ 1F68	ὸ 1F78	ᾈ 1F88	ᾘ 1F98	ᾨ 1FA8	Ᾰ 1FB8	Ὲ 1FC8	Ῐ 1FD8	Ῠ 1FE8	Ὸ 1FF8
9	Ἁ 1F09	Ἑ 1F19	Ἡ 1F29	Ἱ 1F39	Ὁ 1F49	Ὑ 1F59	Ὡ 1F69	ό 1F79	ᾉ 1F89	ᾙ 1F99	ᾩ 1FA9	Ᾱ 1FB9	Έ 1FC9	Ῑ 1FD9	Ῡ 1FE9	Ό 1FF9
A	Ἂ 1F0A	Ἒ 1F1A	Ἢ 1F2A	Ἲ 1F3A	Ὂ 1F4A	▨	Ὢ 1F6A	ὺ 1F7A	ᾊ 1F8A	ᾚ 1F9A	ᾪ 1FAA	Ὰ 1FBA	Ὴ 1FCA	Ὶ 1FDA	Ὺ 1FEA	Ὼ 1FFA
B	Ἃ 1F0B	Ἓ 1F1B	Ἣ 1F2B	Ἳ 1F3B	Ὃ 1F4B	Ὓ 1F5B	Ὣ 1F6B	ύ 1F7B	ᾋ 1F8B	ᾛ 1F9B	ᾫ 1FAB	Ά 1FBB	Ή 1FCB	Ί 1FDB	Ύ 1FEB	Ώ 1FFB
C	Ἄ 1F0C	Ἔ 1F1C	Ἤ 1F2C	Ἴ 1F3C	Ὄ 1F4C	▨	Ὤ 1F6C	ὼ 1F7C	ᾌ 1F8C	ᾜ 1F9C	ᾬ 1FAC	ᾼ 1FBC	ῌ 1FCC	▨	Ῥ 1FEC	ῼ 1FFC
D	Ἅ 1F0D	Ἕ 1F1D	Ἥ 1F2D	Ἵ 1F3D	Ὅ 1F4D	Ὕ 1F5D	Ὥ 1F6D	ώ 1F7D	ᾍ 1F8D	ᾝ 1F9D	ᾭ 1FAD	᾽ 1FBD	῍ 1FCD	῝ 1FDD	῭ 1FED	´ 1FFD
E	Ἆ 1F0E	▨	Ἦ 1F2E	Ἶ 1F3E	▨	▨	Ὦ 1F6E	▨	ᾎ 1F8E	ᾞ 1F9E	ᾮ 1FAE	ι 1FBE	῎ 1FCE	῞ 1FDE	΅ 1FEE	῾ 1FFE
F	Ἇ 1F0F	▨	Ἧ 1F2F	Ἷ 1F3F	▨	Ὗ 1F5F	Ὧ 1F6F	▨	ᾏ 1F8F	ᾟ 1F9F	ᾯ 1FAF	᾿ 1FBF	῏ 1FCF	῟ 1FDF	` 1FEF	▨

The Unicode Standard 12.1, Copyright © 1991-2019 Unicode, Inc. All rights reserved.

(a) Greek polytonic character classes



ΜΕΡΟΣ Γ′. 203

διαφεύγουσι τοὺς σκώληκας τοῦ τάφου ἢ τοῦ Ἅδου τὰς πύλας, ἀφοῦ δὲν φρουρεῖ πλέον αὐτὰς ὁ τρικέφαλος Κέρβερος, καὶ πλανῶνται εἰς τοὺς ἀγρούς, ταράττοντες τὰ ὄνειρα τῶν προβάτων καὶ τὰ φιλήματα τῶν ἐραστῶν. Ἀλλ᾽ οἱ ἡμέτεροι μοναχοί, φέροντες ἀνηρτημένον περὶ τὸν τράχηλον ὀδόντα τῆς Ἁγίας Σαβίνης ἀπέφυγον δι᾽ αὐτοῦ τὰς κακὰς συναντήσεις, μακρόθεν δὲ μόνον εἶδον ἀγέλην ὀνοκεφάλων παγανιῶν, οἵτινες σείοντες τὰ μακρὰ ὦτά των ἠτένιζον ἐρωτικῶς τὴν σελήνην, εἰς τὸ φῶς τῆς ὁποίας ἐζήτουν τὸν περιμενόμενον Μεσσίαν. Δὶς δὲ ἢ τρὶς προσέκοψαν κατὰ καλογήρων κοιμωμένων ἐπὶ τῶν πλακῶν τῶν Προπηλαίων, οἵτινες οὐδὲ κἂν μετεκινήθησαν· καθότι οἱ Ἕλληνες εἶχον ἤδη συνειθίσει νὰ καταπατῶνται ὡς σταφυλαὶ ὑπὸ τοὺς πόδας τῶν ξένων.

Ἡ Ἰωάννα ἄλλους ναοὺς δὲν εἶχεν ἰδεῖ πλὴν μόνων τῶν δρυϊδικῶν μονολίθων καί τινων ἀμόρφων ῥωμαϊκῶν ἐρειπίων, τῆς δὲ πατρίδος της αἱ ἐκκλησίαι ἦσαν αἱ πλεῖσται ξύλιναι καὶ ἀπελέκητοι ὡς οἱ ἀνεγείραντες αὐτὰς Γερμανοί· ὥστε δὲν ἠδύνατο νὰ χορτάσῃ θαυμάζουσα τὰς στήλας τοῦ Παρθενῶνος καὶ τὰς Καρυάτιδας τοῦ Ἐρεχθείου, περὶ ὧν ἠρώτα κατασπαζόμενος τοὺς πόδας των ὁ καλὸς Φρουμέντιος, ἂν ἦσαν ἄγγελοι ἀπολιθωθέντες Ὁ ναὸς τῆς παρθένου Ἀθηνᾶς ἀνῆκε τότε εἰς τὴν παρθένον Μαρίαν. Ἀλλὰ κατ᾽ ἐκείνην τὴν στιγμὴν οὔτε ῥινόφωνοι ψαλμῳδίαι οὔτε λιβάνου νεκρώσιμοι ἀναθυμιάσεις ἢ κώδωνες ὀχληροὶ ἤρχοντο νὰ ταράξωσι τὰ θέλγητρα τῶν ἀναμνήσεων.

(b) Page example from where the characters are extracted

Figure 4.1: Data Description

From each set, we keep only the characters that have at least 10 samples per each class. This brings the total count of characters in set1 to 21.209 for 89 distinct classes and in set2 to 164.772 for 123 distinct classes.

Our evaluation strategy goes as follows: Before we run our experiments, we normalize the character size to dimensions of 30×30. Our strategy involves a stratified[1] 5-fold cross validation process [23] (except in the case of ANNs where only one fold is performed with 80% train data and 20% test data, due to limited computing resources). We evaluate the feature extraction methods that we implemented and the recognition methods as well. In the case of feature extraction, the feature vectors are combined with a k-NN classifier [24] with $k = 1$. This classifier is chosen as a simple case so we can evaluate better the extraction methods per se, without having a complex model that can shadow the performance. In the case of classification, since our models work only with the character pixels as input values, no additional component is needed.

As the evaluation metric for our methods, the accuracy is used. This metric is defined as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions made}} \tag{4.1}$$

Since in our case we perform 5-fold cross validation, we present the average accuracy of those 5 folds.

4.2 Results for Dataset 1

In the following tables, we can see the performance of our methods for the first set of data. Table 4.1 combines both the simple zones extraction as well as the adaptive zones extraction. If we set $\lambda_x = \lambda_y = 0$ in Equation 2.1 we can see that we get the standard zones that we saw in subsection 2.1.2.

Table 4.1: Accuracy for zone features - Dataset 1

	Standard Zones	**Adaptive Zones**		
Dimension of Zones	$\lambda_x = \lambda_y = 0$	$\lambda_x = \lambda_y = 1$	$\lambda_x = \lambda_y = 2$	$\lambda_x = \lambda_y = 3$
Zones 2x2	98.25	**98.29**	97.23	94.17
Zones 3x3	98.23	98.24	97.24	94.8
Zones 5x5	98.01	97.85	96.96	94.66

Table 4.2: Accuracy for projection features - Dataset 1

Projections	**Accuracy**
3	67.94
5	82.28
10	85.14
15	85.35
30	**85.55**

Table 4.3: Accuracy for subdivision features - Dataset 1

Level of Granularity L	**Accuracy**
$L = 0$	26.5
$L = 1$	84.11
$L = 2$	**95.11**
$L = 3$	91.83
$L = 4$	77.38

[1] *stratified means that each class keeps it's equal representation across the partitions of the folds*

Table 4.4: Accuracy for classifiers - Dataset 1

Method	Accuracy
Template Matching - Jaccard	98.02
Template Matching - Yule	95.96
ANN [$epochs = 25$, $batch\_size = 16$, $learning\_rate = 0.01$]	**98.18**

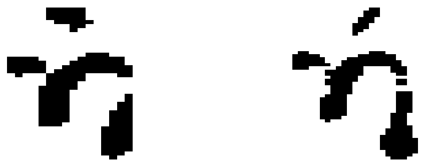
As we can see in Table 4.1, out of all cases, we perform the best in the adaptive zones case for zones of dimensions 2×2 and $\lambda_x = \lambda_y = 1$. The performance in the projections extraction is rather poor while the performance in the subdivisions extraction is promising for $L = 2$, considering the speed of the method at that granularity. Finally, we see in Table 4.4 that our ANN performs better than the other two classification models, but still, it doesn't surpass the accuracy of the adaptive zones that are mentioned in the beginning of this paragraph. Overall, we see promising results with the best classification combination reaching an accuracy of 98.29%.

Considering the best result (adaptive zones with $\lambda_x = \lambda_y = 1$), we now present a subset of the confusion matrix with the 5 classes that we have the worst performance. This is a way to see which character classes are mostly mispredicted as another character. This sample comes from the last fold of the evaluation process. We can see that information in Table 4.5.

Table 4.5: Five characters with the worst performance - Dataset 1

Character Class	Count	Accuracy	Falsely Predicted As	Percentage of False Prediction
GREEK SMALL LETTER EPSILON WITH DASIA AND OXIA	2	50%	GREEK SMALL LETTER EPSILON WITH PSILI AND OXIA	50%
GREEK SMALL LETTER ETA WITH PSILI	2	50%	GREEK SMALL LETTER ETA WITH OXIA	50%
GREEK SMALL LETTER UPSILON WITH DASIA AND OXIA	2	50%	GREEK SMALL LETTER UPSILON WITH PSILI AND OXIA	50%
GREEK SMALL LETTER IOTA WITH DASIA	9	44.44%	GREEK SMALL LETTER IOTA WITH OXIA	22.22%
			GREEK SMALL LETTER EPSILON	11.11%
			GREEK SMALL LETTER IOTA WITH PSILI AND PERISPOMENI	11.11%
			GREEK SMALL LETTER IOTA WITH VARIA	11.11%
GREEK SMALL LETTER OMICRON WITH PSILI	5	40%	GREEK SMALL LETTER OMICRON WITH VARIA	40%
			GREEK SMALL LETTER OMICRON WITH DASIA	20%

As we observe, most character classes in this table have a small sample of testing data, meaning also a small sample of training data since for each fold the testing data consists of 20% the data sample. Even so, we see that characters that are mainly mispredicted are characters from the same Greek character but with difference diacritic marks (e.g. "eta with psili" and "eta with oxia"). We can see such an example in Figure 4.2.



(a) "Eta with psili" character sample **(b) "Eta with oxia" character sample**

Figure 4.2: Two characters that are mispredicted in Dataset 1

4.3 Results for Dataset 2

Like in the previous section, here we can see the performance of our methods for the second set of data. We present the same tables as in set1.

Table 4.6: Accuracy for zone features - Dataset 2

	Standard Zones	**Adaptive Zones**		
Dimension of Zones	$\lambda_x = \lambda_y = 0$	$\lambda_x = \lambda_y = 1$	$\lambda_x = \lambda_y = 2$	$\lambda_x = \lambda_y = 3$
Zones 2x2	96.53	**96.94**	95.84	93
Zones 3x3	96.6	96.8	95.97	94.01
Zones 5x5	95.87	96.09	95.26	93.26

Table 4.7: Accuracy for projection features - Dataset 2

Projections	**Accuracy**
3	49.78
5	75.29
10	81.2
15	81.62
30	**81.75**

Table 4.8: Accuracy for subdivision features - Dataset 2

Level of Granularity L	**Accuracy**
$L = 0$	13.9
$L = 1$	76.98
$L = 2$	**91.81**
$L = 3$	86.46
$L = 4$	66.06

Table 4.9: Accuracy for classifiers - Dataset 2

Method	**Accuracy**
Template Matching - Jaccard	$-$
Template Matching - Yule	$-$
ANN [$epochs = 25$, $batch\_size = 16$, $learning\_rate = 0.01$]	**97.64**

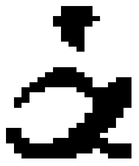
As we can see in Table 4.9, out of all cases, we perform the best in the ANN case. The performance in the projections extraction here is also poor while the performance in the subdivisions extraction has the same pattern as in set1. In this set, Table 4.9 shows only the results for the ANN model. That is because set2 is much bigger than set1 and the two template matching models would take weeks to train and evaluate (making them useless in practice for that many points of data). Compared to the results from set1, here the best accuracy score is by a small margin lower, namely 97.64% (vs 98.29% in set1). Still, we can see that with ANNs we can get very good results even for big sets of data with many categories to predict from.

Considering the best result (ANN), like before, we present a subset of the confusion matrix with the 5 classes that we have the worst performance. We can see that information in Table 4.10.

Table 4.10: 5 characters with worst performance - Dataset 2

Character Class	Count	Accuracy	Falsely Predicted As	Percentage of False Prediction
GREEK SMALL LETTER UPSILON WITH DASIA AND PERISPOMENI	4	50%	GREEK SMALL LETTER TAU	25%
			GREEK SMALL LETTER UPSILON WITH DASIA	25%
GREEK SMALL LETTER ETA WITH DASIA AND OXIA	5	40%	GREEK SMALL LETTER ETA WITH DASIA	20%
			GREEK SMALL LETTER ETA WITH PSILI AND VARIA	20%
			GREEK SMALL LETTER THETA	20%
GREEK SMALL LETTER UPSILON WITH PSILI AND PERISPOMENI	5	40%	GREEK SMALL LETTER UPSILON WITH DASIA AND OXIA	20%
			GREEK SMALL LETTER UPSILON WITH DASIA AND PERISPOMENI	20%
			GREEK SMALL LETTER UPSILON WITH PERISPOMENI	20%
GREEK CAPITAL LETTER OMICRON WITH DASIA	9	33.33%	GREEK CAPITAL LETTER OMICRON WITH PSILI	33.33%
			GREEK SMALL LETTER OMICRON	22.22%
			GREEK CAPITAL LETTER OMICRON WITH DASIA AND OXIA	11.11%
GREEK SMALL LETTER ALPHA WITH DASIA	7	28.57%	GREEK SMALL LETTER ALPHA WITH PSILI	57.14%
			GREEK SMALL LETTER ALPHA WITH TONOS	14.28%

As we also observe here, most character classes have a small sample of testing data. We see that characters that are mainly mispredicted are also characters from the same Greek character but with difference diacritic marks (e.g. ”alpha with dasia” and ”alpha with psili”). We can see that example in Figure 4.3.


(a) ”Alpha with dasia” character sample

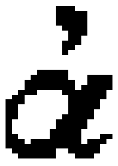

(b) ”Alpha with psili” character sample

Figure 4.3: Two characters that are mispredicted in Dataset 2

5. RELATED WORK

In this chapter we are referencing some related work that has to do with character feature extraction techniques and generally with OCR technologies.

Many feature extraction techniques are mentioned in [16] for binary images. Those include geometric moment invariants, unitary image transforms, spline curve approximation, Fourier transforms, and other. This paper provides a useful overview of many feature extraction methods, including some that are presented in this thesis. Additionally, in [12] we see a probabilistic neural network that is used as a pattern recognition model along with feature extraction methods such as celled projections and crossings.

Concerning OCR in Greek characters, [13] and [8] provide some interesting approaches to character pattern recognition. The first publication uses Hidden Markov Models in order to recognize and predict Greek polytonic degraded texts while the second one proposes a OCR framework for the recognition of machine-printed Greek polytonic documents that is based on combining different recognition modules in order to have a small number of classes in each module.

6. FUTURE WORK

In this chapter we are proposing some future work that can advance the scope of this thesis.

One important step is to implement and build more feature extraction and pattern recognition techniques for the mlchr library that was presented in Chapter 3. This will enable an enhanced integrated library that can be used both in research as in applications.

Another future endeavor would be to try to get more precise predictions by using the context of the characters that were extraced from the book pages. For example, GRPOLY-DB that we saw in Chapter 4 has also information about the lines and paragraphs where those characters belong. We could use that information to extract statistical properties that would boost the model's accuracy.

Finally, more experiments have to be done on other sets of data in order to evaluate the implemented methods and compare the performance across other use cases.

7. CONCLUSIONS

In this thesis we explored character pattern recognition techniques that were used to recognize Greek polytonic machine-printed characters from two different sets of data. We presented feature extraction methods and character classification models and we implemented a python library that performs those functionalities in a simple and intuitive way. Finally, we tested our methods and we saw promising results in terms of performance for both our sets of data.

ABBREVIATIONS - ACRONYMS

OCR	Optical Character Recognition
PDF	Portable Document Format
ANN	Artificial Neural Network
MLP	Multilayer Perceptron
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
ReLU	Rectified Linear Units

BIBLIOGRAPHY

[1] Numpy. `https://www.numpy.org/`, 2019. [Online; accessed July-2019].

[2] scikit-learn. `https://scikit-learn.org/stable/`, 2019. [Online; accessed July-2019].

[3] Scipy. `https://www.scipy.org/`, 2019. [Online; accessed July-2019].

[4] Tensorflow. `https://www.tensorflow.org/`, 2019. [Online; accessed July-2019].

[5] M. Bokser. Omnidocument technologies. *Proceedings of the IEEE*, 80(7):1066–1078, 1992.

[6] A. Chaudhuri, K. Mandaviya, P. Badelia, and S.K. Ghosh. *Optical Character Recognition Systems for Different Languages with Soft Computing*. Studies in Fuzziness and Soft Computing. Springer International Publishing, 2016.

[7] CIL. Computational intelligence laboratory. `https://www.iit.demokritos.gr/cil/`, 2019. [Online; accessed July-2019].

[8] B. Gatos, G. Louloudis, and N. Stamatopoulos. Greek polytonic ocr based on efficient character class number reduction. In *2011 International Conference on Document Analysis and Recognition*, pages 1155–1159, Sep. 2011.

[9] Basilios Gatos, Anastasios L. Kesidis, and A. Papandreou. Adaptive zoning features for character and word recognition. *2011 International Conference on Document Analysis and Recognition*, pages 1160–1164, 2011.

[10] Basilis Gatos, Nikos Stamatopoulos, Giorgos Sfikas, George Rekatsinas, Vassilis Papavassiliou, Fotini Simistira, and Vassilis Katsouros. Grpoly-db: An old Greek polytonic document image database. pages 646–650. ICDAR, 2015.

[11] MH Glauberman. Character recognition for business machines. *Electronics*, 29(2):132–136, 1956.

[12] M Zahid Hossain, M Ashraful Amin, and Hong Yan. Rapid feature extraction for optical character recognition. *arXiv preprint arXiv:1206.0238*, 2012.

[13] V. Katsouros, V. Papavassiliou, F. Simistira, and B. Gatos. Recognition of greek polytonic on historical degraded texts using hmms. In *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*, pages 346–351, 2016.

[14] William K. Pratt. *Digital image processing*. John Wiley & Sons, 2nd edition, 1991.

[15] Rizart Dona. mlchr. `https://github.com/rizart/mlchr`, 2019. [Online; accessed July-2019].

[16] Øivind Due Trier, Anil K. Jain, and Torfinn Taxt. Feature extraction methods for character recognition-a survey. *Pattern Recognition*, 29:641–662, 1996.

[17] J. D. Tubbs. A note on binary template matching. *Pattern Recognition*, 22(4):359–366, 1989.

[18] Georgios Vamvakas, Basilis Gatos, and Stavros J Perantonis. A novel feature extraction and classification methodology for the recognition of historical documents. In *2009 10th International Conference on Document Analysis and Recognition*, pages 491–495. IEEE, 2009.

[19] Georgios Vamvakas, Basilis Gatos, and Stavros J. Perantonis. Handwritten character recognition through two-stage foreground sub-sampling. *Pattern Recognition*, 43(8):2807 – 2816, 2010.

[20] Wikipedia. Artificial neural networks. `https://en.wikipedia.org/wiki/Artificial_neural_network`, 2019. [Online; accessed July-2019].

[21] Wikipedia. Convolutional neural networks. `https://en.wikipedia.org/wiki/Convolutional_neural_network`, 2019. [Online; accessed July-2019].

[22] Wikipedia. Gradient descent. `https://en.wikipedia.org/wiki/Gradient_descent`, 2019. [Online; accessed July-2019].

[23] Wikipedia. K-fold cross validation. `https://en.wikipedia.org/wiki/Cross-validation_(statistics)#k-fold_cross-validation`, 2019. [Online; accessed July-2019].

[24] Wikipedia. K-nearest neighbors algorithm. `https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm`, 2019. [Online; accessed July-2019].

[25] Wikipedia. Multilayer perceptrons. `https://en.wikipedia.org/wiki/Multilayer_perceptron`, 2019. [Online; accessed July-2019].

[26] Wikipedia. Recurrent neural networks. `https://en.wikipedia.org/wiki/Recurrent_neural_network`, 2019. [Online; accessed July-2019].

[27] Wikipedia. Softmax actication function. `https://en.wikipedia.org/wiki/Softmax_function`, 2019. [Online; accessed July-2019].