

**ANKARA UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT**



**COM491 PROJECT REPORT
Optical Character Recognition**

**Mustafa Ahmet DENİZ 14290087
Mustafa GÖKSEVER 14290099**

Supervisor: Prof. Dr. Refik SAMET

OCAK 2019

1. ABSTRACT

Image Processing is nowadays considered to be a favorite topic in the IT industry. One of its major applications is Optical Character Recognition (OCR). When the object to be matched is presented then the general recognition system starts extracting the important features of the object that includes color, depth, shape, and size. These features are stored in the part of the memory. The computer starts finding the closest match for these extracted features in the whole collection of objects. For humans, character recognition seems to be a simple task but to make a computer analyze and finally correctly recognize a character is a difficult task. OCR is one such technique that gives the power of vision to the computer to extract data from images and finds important data from it and makes it computer editable.

This paper describes the basic character recognition process from printed documents containing English text and the digital image processing technologies, applications. Developments made using tools and library such as PyCharm, GIT, OpenCV, Anaconda, Tkinter, *k*-Nearest Neighbors Algorithm were mentioned.

The aim is to develop an efficient method which uses a custom image to train the classifier. This OCR extracts distinct features from the input image for classifying its contents as characters specifically letters and digits. The input to the system is digital images containing the patterns to be classified. The analysis and recognition of the patterns in images are becoming more complex, yet easy with advances in technological knowledge. The present work involves the application of pattern recognition using KNN to recognize printed text.

OCR system that separates and identify English characters. The modern English alphabet is a Latin alphabet consisting of 26 letters (each having an uppercase and a lowercase form). It consists of five vowels – A, E, I, O, U and the rest 21 alphabets are consonants. Compound characters may be a combination of consonant and vowel or consonant and consonant. The words become difficult to recognize as a result.

2. TABLE OF CONTENTS

1. ABSTRACT.....	2
2. TABLE OF CONTENTS	3
3. ABBREVIATIONS DIRECTORY	4
5. CHARTS DIRECTORY	4
7. MOTIVATION.....	6
7.1. DIGITAL IMAGE	6
7.1.1. Digital Image Types.....	7
7.1.2. Converting the color image to Gray-Tone	8
7.2. OPTICAL CHARACTER RECOGNITION.....	9
7.2.1. Image Preprocess	9
7.2.2. Character Segmentation.....	10
7.2.3. Character Recognition with Template Matching.....	10
7.2.4. Character Recognition with k-Nearest Neighbor	10
7.2.5. Character Recognition with Neural Network	10
7.2.6. Character Recognition with Support Vector Machine.....	11
8. MATERIALS and METHODS	12
8.1. MATERIALS	12
8.1.1. Python.....	12
8.1.2. OpenCV (Open Source Computer Vision)	12
8.1.3. Anaconda	12
8.1.4. PyCharm	13
8.1.5. Git	13
8.1.6. Tkinter	13
8.2. METHODS.....	13
8.2.1. Preprocess	13
8.2.2. Segmentation	15
8.2.3. Characters Recognition	16
9. DISCUSSION AND CONCLUSION.....	20
10. REFERENCE	21
11. APPENDIX.....	22
11.1. APPENDIX 1 (IMAGES).....	22
11.2. APPENDIX 2 (CODES).....	26

3. ABBREVIATIONS DIRECTORY

Optical Character Recognition(OCR), k -Nearest Neighbors Algorithm(k -NN)
Character recognition (CR), Multilayer Perceptron (MLP), Support Vector
Machine(SVM), Preprocess, Segmentation, Training, Dataset, Classification

4. FIGURES DIRECTORY

Figure 6.1. OCR Flow	6
Figure 7.1. Example pixel of an image	8
Figure 7.2. Binary Image	8
Figure 7.3. Grayscale image	4
Figure 7.5. Grayscale image from RGB image	10
Figure 8.1. Image	12
Figure 8.2. Original Image	13
Figure 8.3. Gray Image	13
Figure 8.4. Gaussian Blur Image	13
Figure 8.5. Binary Image	14
Figure 8.6. Finding Contours	14
Figure 8.7. Segmented Image	14
Figure 8.7. Discovered Text	15
Figure 8.8. K-nearest neighbors	17
Figure 8.9. Dataset	18

5. CHARTS DIRECTORY

Table 7.4. Table of RGB Colors	9
--------------------------------	---

6. INTRODUCTION

Digital image processing is the help to enhance the quality of the image. Image processing is performed to extract information from the image digitally by the use of the computer algorithm.

Optical character recognition (OCR) is a process of converting a printed document, scanned a page or digital image into ASCII characters that a computer can recognize. An OCR system enables us to feed a book or a magazine article directly into an electronic computer file and edit the file using a word processor. Though academic research in the field continues.

Optical Character Recognition is broadly divided into two parts, offline recognition, and online recognition. Offline recognition deal with the system where the input is either an image or a scanned form of the document. In this paper, we are dealing only with offline recognition technique.

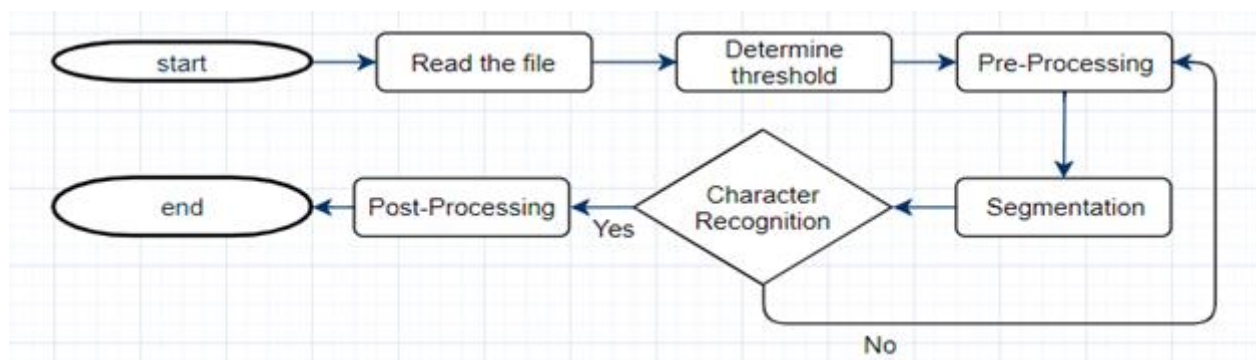


Figure 6.1. OCR Flow

A digital image is a representation of a real image as a set of numbers that can be stored and handled by a digital computer. The basic component of a digital image is the pixel element. In image processing, there are three basic processes on which OCR works: Preprocessing, Segmentation and Character Recognition.

7. MOTIVATION

In many environments, huge amounts of time are spent on unnecessary tasks such as inputting data or searching through piles of documents and files to retrieve information needed to complete a task. The growing trend of document digitization, enabled by document scanning, is helping to reduce time wasted and when coupled with OCR technology it is revolutionizing businesses and studies all over.

Optical Character Recognition is the process of converting a scanned document into fully editable and searchable virtual files, transforming paper documents into formats such as Microsoft Word, Excel, CV files and PDF searchable documents. When processing a scanned digital file with OCR technology, highly useful documents are created as a result which can allow businesses to have easier and quicker access to streams of data in a digital format.

7.1. DIGITAL IMAGE

A digital image is an array, or a matrix, of square pixels (picture elements) arranged in columns and rows. The basic component of a digital image is the pixel-picture element. Therefore, an image should be $m \times n$ dimensional matrix of pixels. To refer to a specific pixel within the image matrix, we define its coordinate at x and y . The coordinate system of image matrices defines x as increasing from left to right and y as increasing from top to bottom.

A pixel is the smallest unit of a digital image or graphic that can be displayed and represented on a digital display device. A pixel is the basic logical unit in digital graphics. Pixels are combined to form a complete image, video, text or any visible thing on a computer display. A pixel is also known as a picture element. Image size specifically describes the number of pixels within a digital image.

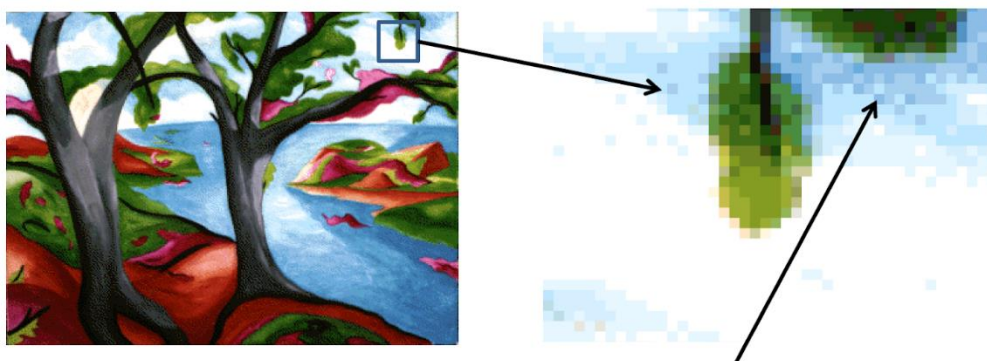


Figure 7.1. Example pixel of an image

A blue pixel

7.1.1. Digital Image Types

7.1.1.1. Binary Digital Image

A binary image is a digital image that has only two possible values for each pixel. Typically, the two colors used for a binary image are black and white. The color used for the objects in the image is the foreground color while the rest of the image is the background color.

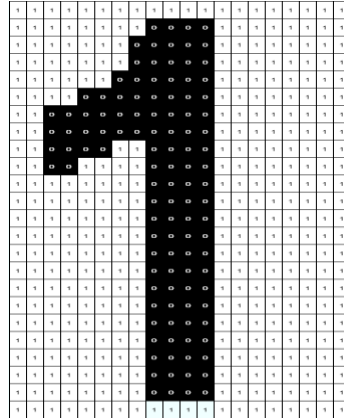


Figure 7.2. Binary Image

7.1.1.2. Grayscale Digital Image

The brightness value of each pixel of the digital image is called grey levels. The grey level range is determined by the number of bits that the brightness value at each pixel is encoded.

There are two colors in grey level borders, black and white. Each pixel commonly used in the application is encoded with 8 bits. In this type of image, each pixel consists of

$2^8 = 256$ different grayscale (brightness level) values and the gray range of values is expressed as $G = \{0, 1, 2, \dots, 255\}$.

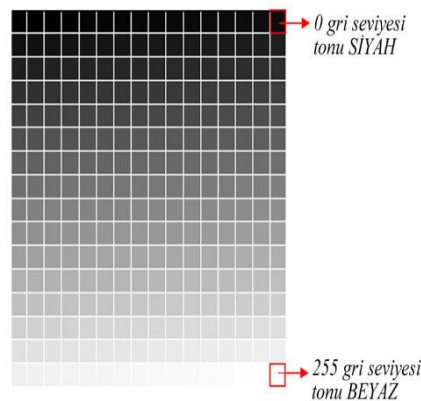


Figure 7.3. Grayscale image

7.1.1.3. RGB Image

Color images, on the other hand, have intensity from the darkest and lightest of three different colors, Red, Green, and Blue. Intensity values in digital images are defined by bits. An 8-bit intensity range has 256 possible values, 0 to 255. This can be seen mathematically by $2^{(\# \text{ of bits})}$. For an 8-bit image, $2^8 = 256$ possible values. The various mixtures of these color intensities produce a color image. The standard digital photo uses an 8-bit range of values; RGB images contain 3 x 8-bit intensities they are also referred to as 24-bit color images.


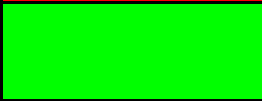



Colors Name	R	G	B	Colors
Red	255	0	0	
Green	0	255	0	
Blue	0	0	255	
White	255	255	255	
Dark Grey	100	100	100	

Table 7.4. Table of RGB Colors

7.1.2. Converting the color image to Gray-Tone

The process of converting a color digital image to a gray-toned image is, in fact, the scaling of the gray-tone images that correspond to each color band specified in the RGB color model.

The formula is: **Gray = 0.299 x R + 0.587 x G + 0.114 x B**

The grayscale of the color image is obtained using this equation.

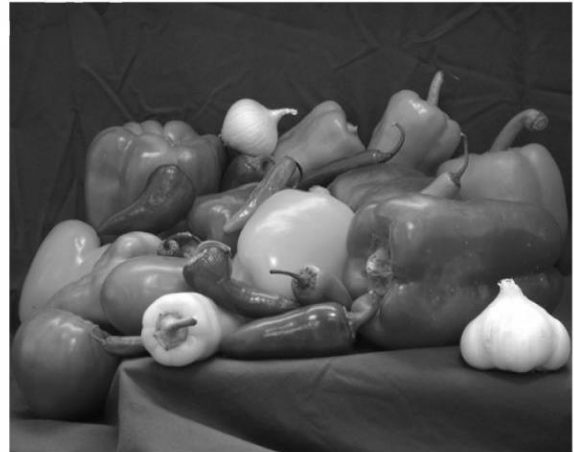


Figure 7.5. Grayscale image from RGB image.

7.2. OPTICAL CHARACTER RECOGNITION

There are many different approaches to solving the optical character recognition problem. First of all, there need to do image process like preprocessing, segmentation. These image processes are for finding the objects in the digital images with high rate. After finding objects from the image, there are some algorithms to reveal what the object is.

7.2.1. Image Preprocess

Preprocessing techniques are needed on color, grey-level or binary document images containing text and/or graphics. In character recognition systems most of the applications use grey or binary images since processing color images are computationally high. Such images may also contain non-uniform background and/or watermarks making it difficult to extract the document text from the image without performing some kind of preprocessing, therefore; the desired result from preprocessing is a binary image containing text only. Thus, to achieve this, several steps are needed, first, some image enhancement techniques to remove noise or correct the contrast in the image, second, thresholding to remove the background containing any scenes, watermarks and/or noise, third, page segmentation to separate graphics from text, fourth, character segmentation to separate characters from each other and, finally, morphological processing to enhance the characters in cases where thresholding and/or other preprocessing techniques eroded parts of the characters or added pixels to them. The above techniques present few of those which may be used in character recognition systems and in some applications; few or some of these techniques or others may be used at different stages of the OCR system.

7.2.2. Character Segmentation

Character segmentation is a necessary preprocessing step for character recognition in many OCR systems. It is an important step because incorrectly segmented characters are unlikely to be recognized correctly.

The most difficult case in character segmentation is the cursive script.

7.2.3. Character Recognition with Template Matching

Optical Character Recognition by using **Template Matching** is a system useful to recognize the character or alphabet by comparing two images of the alphabet. The objectives of this system are to develop for the Optical Character Recognition (OCR) system and to implement the Template Matching algorithm in developing the system. The purpose of this system is to solve the problem in recognizing the character which is before that it is difficult to recognize the character without using any techniques and Template Matching is as one of the solutions to overcome the problem. There are a few processes that were involved. The processes are starting from the acquisition process, filtering process, a threshold the image, clustering the image of the alphabet and lastly recognize the alphabet. All of these processes are very important to get the result of recognition after comparing the two-character images.

7.2.4. Character Recognition with k-Nearest Neighbor

The nearest neighbor (**k-NN**) is a well-known approach used to classify characters and predictions. The idea is to search for the closest match of the test data in feature space. The distance measured between the two characters images is needed in order to use this rule. The rule involves a training set of positive and negative cases and a new sample is classified by calculating the distance to the nearest one.

7.2.5. Character Recognition with Neural Network

An optical character recognition system, which uses a multilayer perceptron (MLP) is **neural network classifier**. Neural networks are good pattern recognition engines and robust classifiers, with the ability to generalize in making decisions based on imprecise input data. The neural network classifier has the advantage of being fast, easily trainable, and capable of creating arbitrary partitions of the input feature space. Issues in the design of the neural network that we examine include the selection of input features, the choice of network learning and momentum parameters, and the selection of training patterns.

7.2.6. Character Recognition with Support Vector Machine

Support Vector Machines – SVMs, represent the cutting edge of ranking algorithms and have been receiving special attention from the international scientific community. Many successful applications, based on SVMs, can be found in different domains of knowledge, such as in text categorization, digital image analysis, character recognition, and bioinformatics.

Support vector machines are computational algorithms that construct a hyperplane or a set of hyperplanes in a high or infinite dimensional space. SVMs can be used for classification, regression, or other tasks

8. MATERIALS and METHODS

The OCR can only be successful after some process. These processes include pre-processing, segmentation, and characters recognition.

To achieve higher recognition rate there need to improve the quality of the picture. Preprocess provide clean, high-quality image. Preprocess removes noise, detect and remove the unrelated object in the image and convert the binary image.

In large and complex images, there need to segment the image and then extract the characters by using character segmentation methods. Segmentation is performed by considering the analysis of geometric features and ligatures of characters.

Character recognition processes will recognize the text and convert the document to an editable text file with an algorithm such as match template, k-Nearest Neighbor, support vector machine, neural networks.

8.1. MATERIALS

Technologies, applications, tools, library, and algorithm were used to achieve character recognition.

8.1.1. Python

It is a high-level programming language which puts emphasis on code readability. Its user-friendly syntax assists in expressing concepts in fewer lines of code.

8.1.2. OpenCV (Open Source Computer Vision)

It is a library of programming functions mainly aimed at real-time computer vision, which is the field of processing and analyzing digital images

8.1.3. Anaconda

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system conda.

8.1.4. PyCharm

PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains.

8.1.5. Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

8.1.6. Tkinter

Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit and is Python's *de facto* standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS of Python.

8.2. METHODS

Test data was created for all methods.

```
• img = np.zeros((80, 280, 3), np.uint8)
• img.fill(255)
• font = cv2.FONT_HERSHEY_SIMPLEX
• cv2.putText(img, "MUSTAFA", (5, 60), font,
  2, (0, 0, 255), 2, cv2.LINE_AA)
• cv2.imwrite("Test Data", img)
```

The image shows the word 'ANKARA' in a large, red, blocky, and slightly irregular font. The letters are thick and have a hand-drawn or stencil-like appearance. The color is a bright red.

Figure 8.1. Image

8.2.1. Preprocess

The output of the scanning process may contain a certain amount of noisy result. The quality of character is highly dependent on the resolution on the scanner and the technique applied for thresholding. The preprocessing is used to eliminate these

defects which may later cause bad recognition rates and smooth the digitized characters.

In order to achieve higher recognition rate, preprocessing algorithms makes the OCR system more robust by accurate image enhancement, noise removal, image thresholding, skew detection/correction.

First, the steps start to browse an image from your files. A full path of the image should be taken from the browse menu. Then used the function **cv2.imread()** to read an image.

```
• self.image = cv2.imread(self.image_path)
```

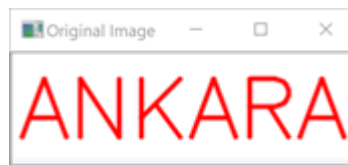


Figure 8.2. Orjinal image

8.2.1.1. Gray Conversion

Grayscale images have many shades of gray. For achieving accuracy input document should be grayscale. To convert a color from a color space based on an RGB color model to a grayscale representation following function is used

$$\text{Gray} = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

When the preprocess button was pressed, this chosen image was converted to gray image by `cvtColor()` method. In this method converts an input image from one color space to another. `COLOR_BGR2GRAY` was used in our project. After the calculation gray image was shown.

```
• gray_image = cv2.cvtColor(cvImage, cv2.COLOR_BGR2GRAY)
```



Figure 8.3. Gray image

8.2.1.2. Blurring

We can see that the image above needs further enhancement, therefore, we apply another blur to improve the looks.

```
• blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
```



Figure 8.4. Gaussian Blur Image

8.2.1.3. Binarization

This step converts a multicolored image (RGB) to a black and white image. There are several algorithms to convert a color image to a binary image, ranging from simple thresholding to more sophisticated zonal analysis.

If the pixel value is greater than a threshold value, it is assigned 0, else it is assigned 255. We applied thresholding with value 127 using `cv2.threshold()` method.

```
• binary = cv2.threshold(blurred_image, 127, 255,  
    cv2.THRESH_BINARY_INV)
```

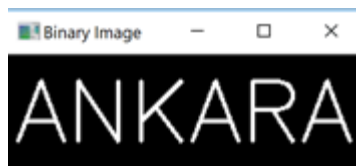


Figure 8.5. Binary Image

8.2.2. Segmentation

In Character Recognition techniques, the Segmentation is the most important process. In the segmentation stage, a sequence of characters of an image is decomposed into

sub-images of the individual character. Segmentation is done to make each character isolated by providing the separation between them.

Segmentation partitions an image into distinct regions containing each pixel with similar attributes. To be meaningful and useful for image analysis and interpretation, the regions should strongly relate to depicted objects or features of interest.

Contours in the binary image were found and drawn a green rectangle around the characters one by one. After that contours were sorted from left to right.

Characters (imgROI) were found and saved in IMAGEROI Folder. It's shown below.

- contours= cv2.findContours(self.binary,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)



Figure 8.6. Finding contours



Figure 8.7. Segmented image

8.2.3. Characters Recognition

There are two basic types of core OCR algorithm, which may produce a ranked list of candidate characters.

Matrix matching involves comparing an image to a stored glyph on a pixel-by-pixel basis; it is also known as "pattern matching". This relies on the input glyph being correctly isolated from the rest of the image, and on the stored glyph being in a similar font and on the same scale. This technique works best with typewritten text and does not work well when new fonts are encountered.

Feature extraction decomposes glyphs into "features" like lines, closed loops, line direction, and line intersections. Feature Extraction serves two purposes; one is to extract properties that can identify a character uniquely. Second is to extract properties that can differentiate between similar characters.

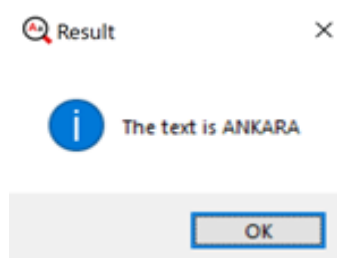


Figure 8.7. Discovered Text

8.2.3.1. Match Template

Optical Character Recognition by using Template Matching is a system prototype that useful to recognize the character or alphabet by comparing two images of the alphabet. Template matching works by "sliding" the template across the original image. As it slides, it compares or matches the template to the portion of the image directly under it. This method calculates all the correlation coefficients for every displacement between the input images. After the template matching, you can filter the results by descent threshold, let's say 0.85, to check if the images are similar (this threshold depends on your input images, e.g. lightning, different sensor types etc.), then look for the max value displacement.

Dataset was created and saved as name of character in dataset folder by using ASCII Uppercase Characters.

```
• for x in list (string.ascii_uppercase ):
•     img = np.zeros((80, 60,3), np.uint8)
•     img.fill(255)
•     cv2.putText(img,x,(10,60), cv2.FONT_HERSHEY_SIMPLEX,
•         2,(0,0,0),2,cv2.LINE_AA)
•     cv2.imwrite("dataset/"+x+".png",img)
```

First, dataset image was preprocessed for match template function. Match template function works by "sliding" the ImgROI (template) across the Dataset_binary. As it slides, it compares or matches the template to the portion of the image directly under it after that found all characters on the image.

```
• for x in list(string.ascii_uppercase):
•     Dataset = cv2.imread("dataset/" + x + ".png")
•     Dataset_gray = cv2.cvtColor(Dataset,
•         cv2.COLOR_BGR2GRAY)
•     ret, Dataset_binary = cv2.threshold(Dataset_gray,
•         127, 256, cv2.THRESH_BINARY_INV)
•     we, he = imgROI.shape[::-1]
•     res = cv2.matchTemplate(Dataset_binary , imgROI,
•         cv2.TM_CCOEFF_NORMED)
•     threshold = 0.85
•     min_val, max_val, min_loc, max_loc =
•         cv2.minMaxLoc(res)
•     if (max_val >= threshold):
•         print(x + " FOUND....")
```

- `text = text + x`
- `break`

8.2.3.2. k-Nearest Neighbor

The nearest neighbor (k-NN) is a well-known approach used to classify characters and predictions. The idea is to search for the closest match of the test data in feature space. The distance measured between the two characters images is needed in order to use this rule. The rule involves a training set of positive and negative cases and a new sample is classified by calculating the distance to the nearest one.

When the amount of the pre-classified points is large, it is good to use a majority vote of the nearest k neighbors instead of the single nearest neighbor. This method is called the k nearest neighbor (k-NN).

We will look into it with below image.

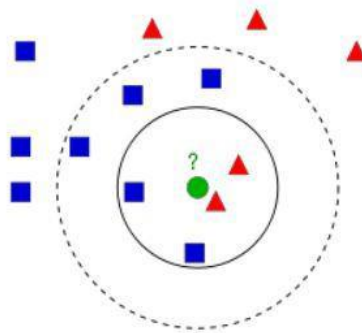


Figure 8.8. K-nearest neighbors

In the image, there are two objects, Blue Squares and Red Triangles. And their feature space that we can consider as space where all data is projected. For example,

consider a 2D coordinate space. Each data has two features, x and y coordinates. We can represent this data in 2D coordinate space. If there are three features, there need 3D space. This N-dimensional space is its feature space. In our image, we consider it as a 2D case with two features. A new object is shown as a green circle which should be added to one of these Blue/Red objects. That process called as Classification. One method is to check its nearest neighbor. From the image, it is clear it is the Red Triangle object.

So, it is also added into the Red Triangle. This method is called simply Nearest Neighbor because classification depends only on the nearest neighbor. Red Triangle may be the nearest. There might be a situation where there are more of Blue Squares near to it. Then Blue Squares will have more strength in that locality than Red Triangle. Hence, to check the nearest one is not sufficient. Instead, we check some k nearest object. Then, whichever is the majority in them, the new object belongs to that object. In our image, if taking $k=3$, i.e. 3 nearest objects. It has two

Red and one Blue so it should be added to Red object. If we take $k=7$, then it has 5 Blue objects and 2 Red objects. It should be added to the Blue object type. So, it all changes with the value of k . If $k = 4$, it has 2 Red and 2 Blue neighbors. It is preferable to take k as an odd number. Hence, this method is called k -Nearest Neighbor since classification depends on k nearest neighbors.

Training dataset was shown below for k -NN classification.

```

0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

```

Figure 8.9. Dataset

The characters found in the dataset have converted a row (a single line) named `npaFlattenImages`. After the characters in the original image are found one by one, characters are expected to be entered from the user and written a file what is named `npaClassification`. By using algorithm k -NN, these files are trained.

The characters found in the original image are converted a row (a single line) named `npaROIResized`. The categorization of the created object is finalized. ($k=1$ means chosen the nearest row)

```

• self.kNearest = cv2.ml.KNearest_create() # instantiate
  KNN object
• self.kNearest.train(np flattenedImages,
  cv2.ml.ROW_SAMPLE, npaClassifications)
• retval, npaResults, neigh_resp, dists =
  self.kNearest.findNearest(self.npaROIResized, k=1)

```

9. DISCUSSION AND CONCLUSION

What does the future hold for OCR? OCR can become a powerful tool for future data entry applications. However, the limited availability of funds in a capital-short environment could restrict the growth of this technology. But, given the proper impetus and encouragement, a lot of benefits can be provided by the OCR system. It provides advantages such as:

- The automated entry of data by OCR is one of the most attractive, labor-reducing technology
- The recognition of new font characters by the system is very easy and quick.
- We can edit the information of the documents more conveniently and we can reuse the edited information as and when required.

Optical Character Recognition system can be efficiently used to speed up the translation of image-based documents into structured documents that are currently easy to discover, search and process.

We have shown that k-NN algorithm can be implemented successfully better than match template in optical character recognition. The system has image preprocessing, segmentation and recognition modules for the image.

Applications of our approach are varied. We hope to implement our ideas further on the various image for multiple characters and lower case. The development of OCR to directly using machine learning algorithm which is known SVM and Naive Bayes, using particular data set, adding handwriting characters, adding complex images are also something we would like to pursue in the near future. In addition to we want to implement an OCR using the tesseract library which is Tesseract is an optical character recognition engine for various operating systems. It is free software, released under the Apache License, Version 2.0, and development has been sponsored by Google since 2006. We can improve our project to convert the image to voice and convert to a particular format like Word, pdf, etc.

After all of these processes, we can still improve the accuracy of recognition. We can minimize the number of incorrect words in the obtained text by doing post-processing. Post-processing step reduces the number of errors in the obtained text. First detects errors in the given OCR text with the help of a support vector machine trained using given training dataset, followed by rectifying the errors by applying a confidence-based mechanism. OCR post-processed text correction is an important and challenging problem that needs to be addressed to facilitate digitization.

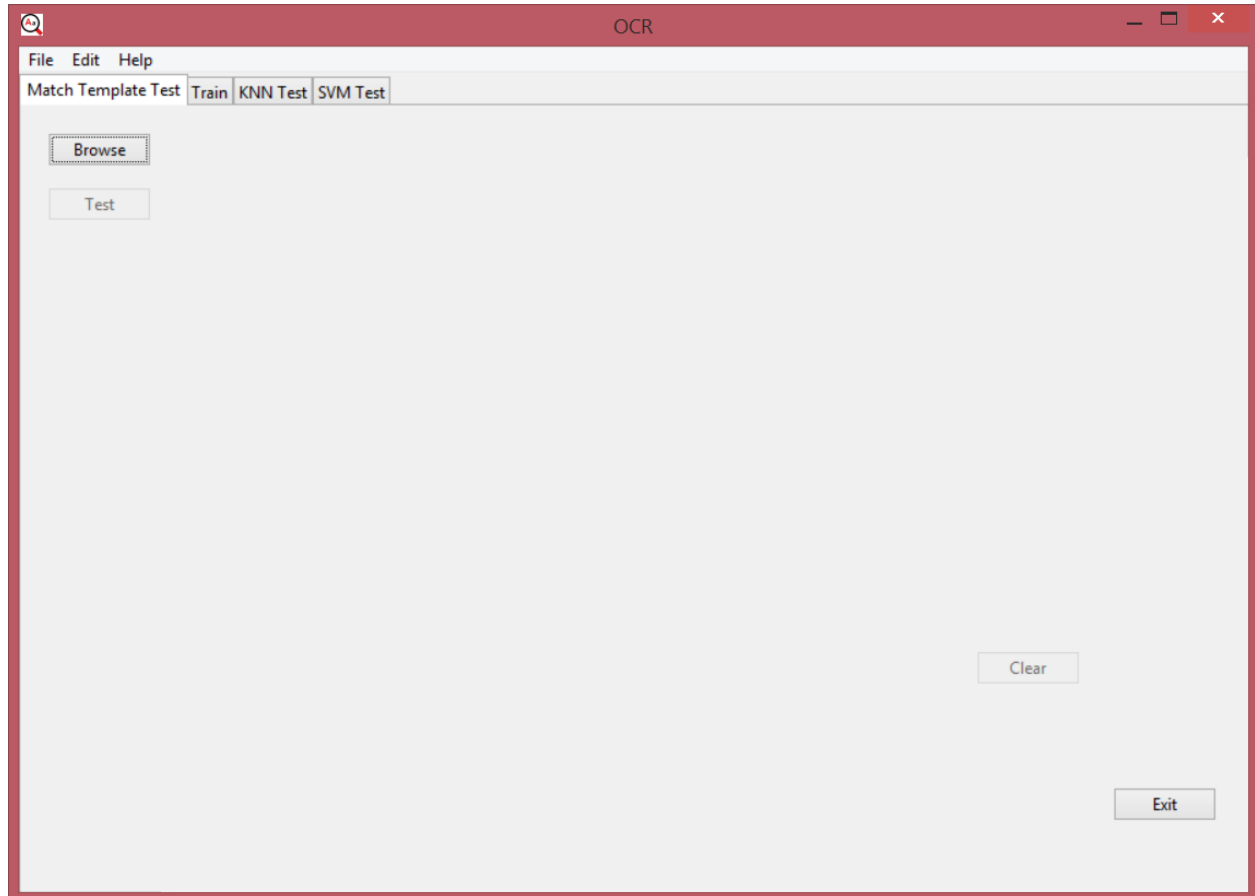
We would like to thank Prof. Dr. Refik SAMET for his immense help and support, useful discussions and valuable recommendations.

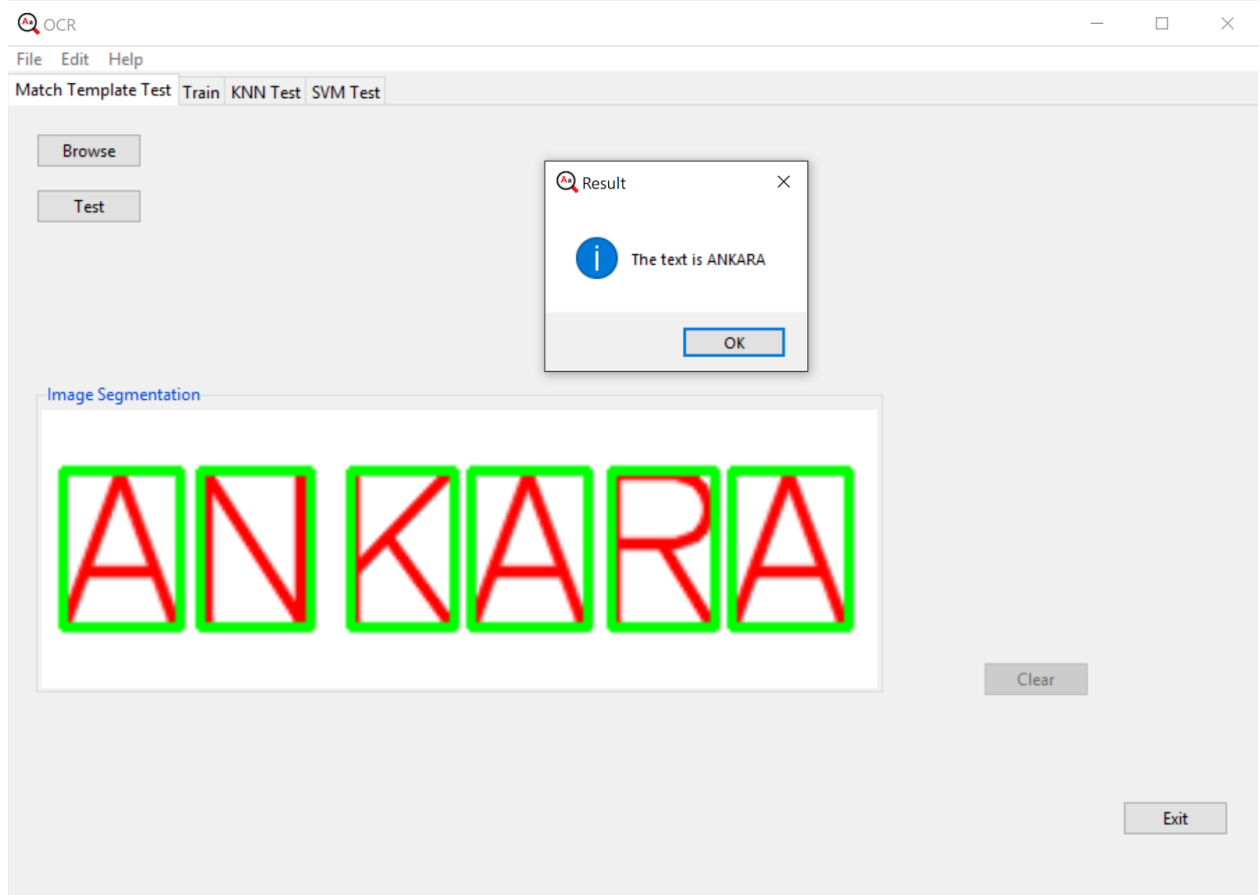
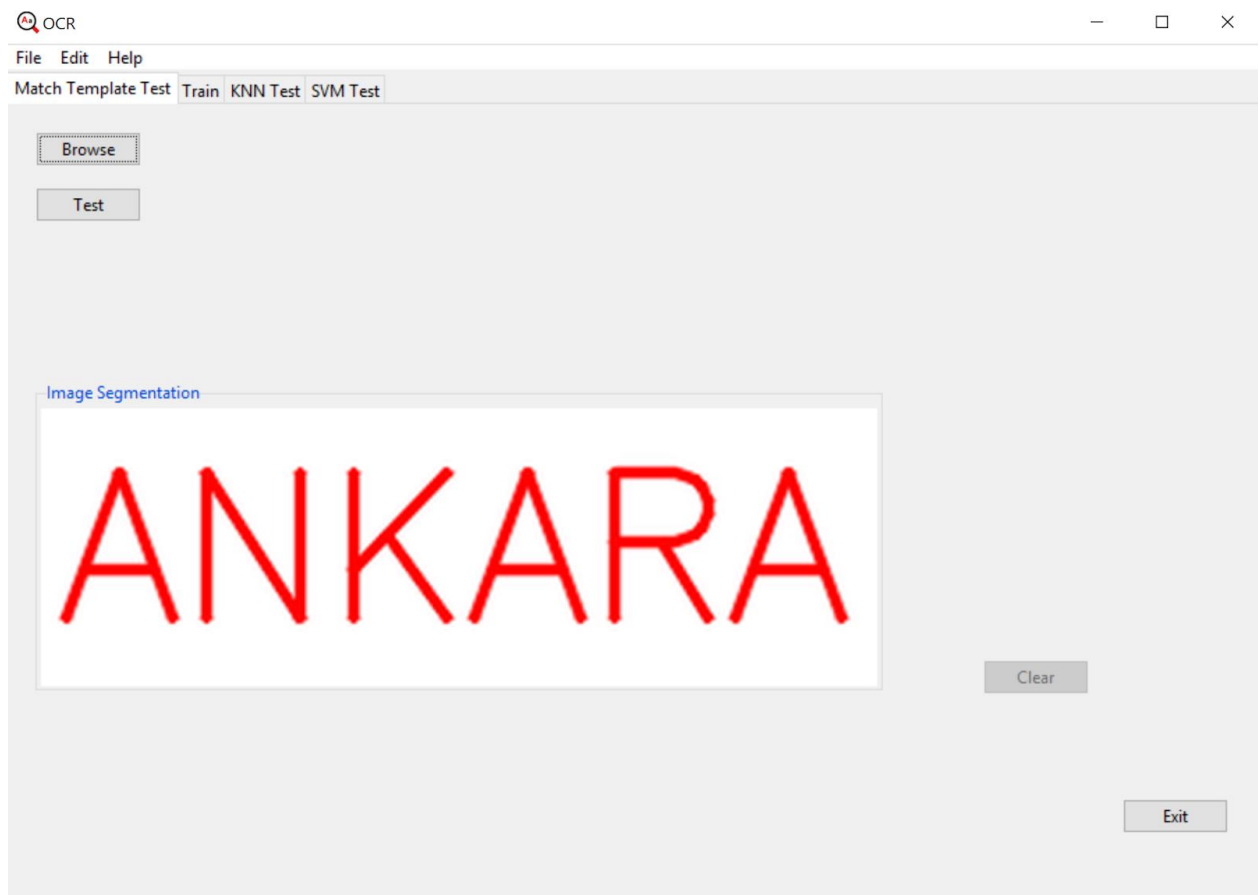
10. REFERENCE

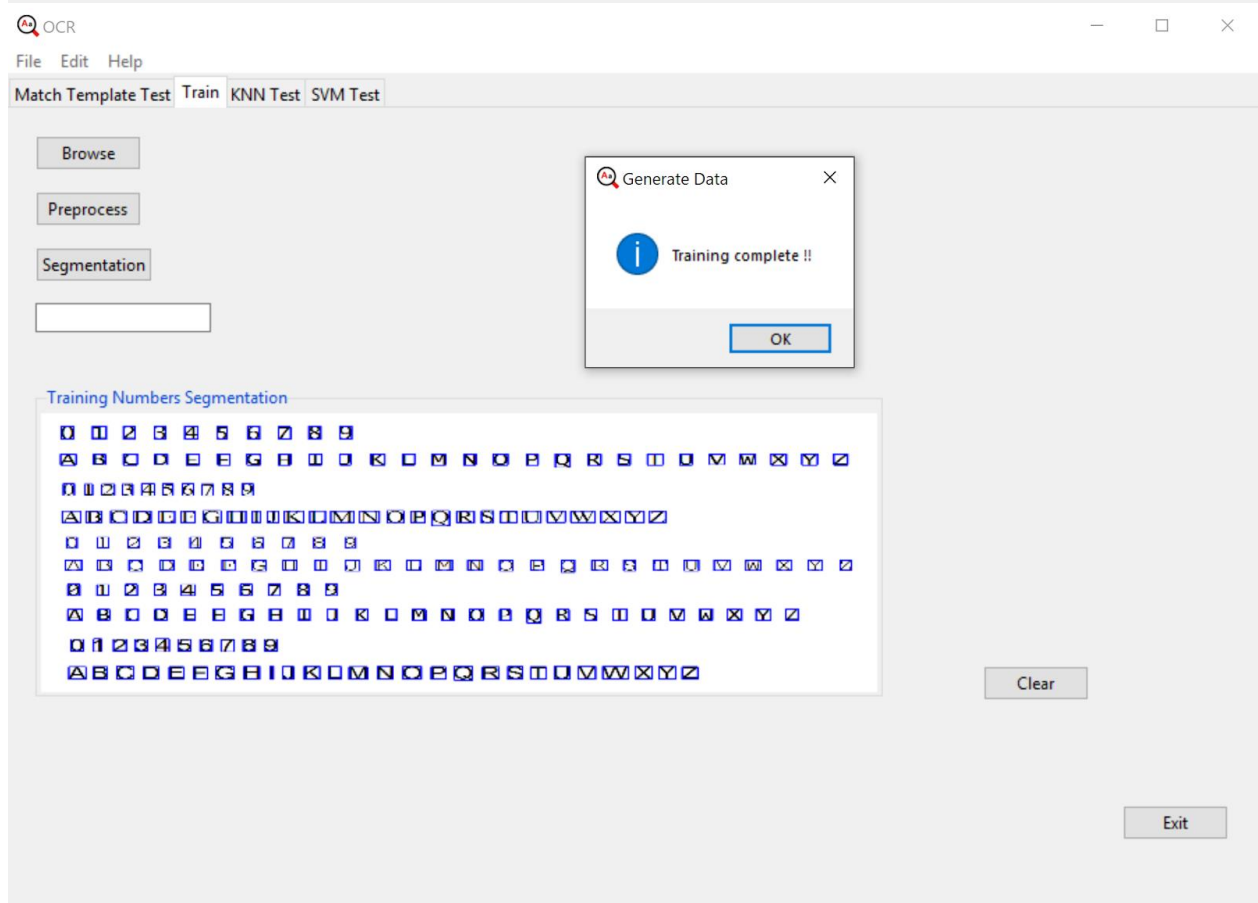
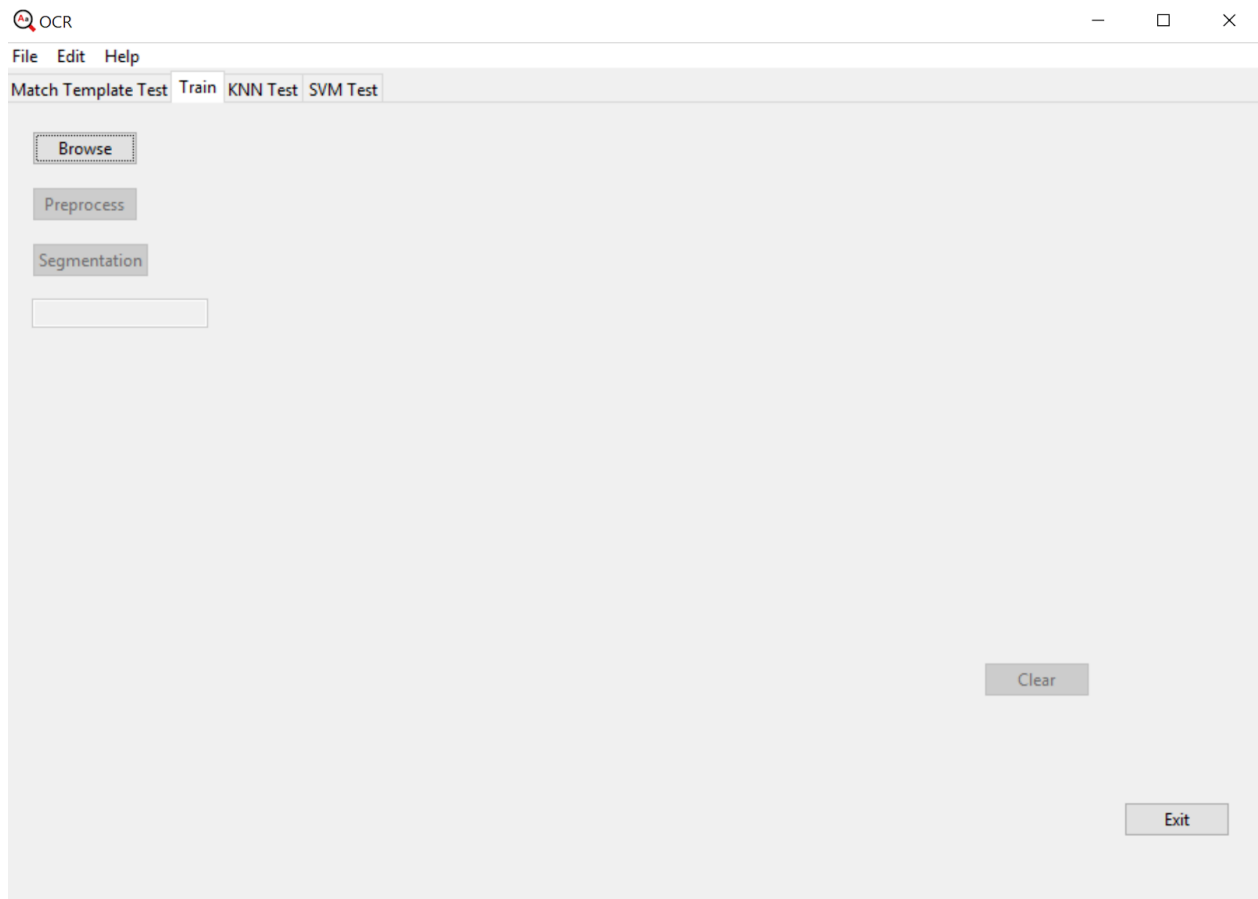
- Lecture Notes, Ibrahim Cayiroglu, Karabük Üniversitesi, Mühendislik Fakültesi
- Lecture Notes, Introduction to Image Processing, Doç. Dr. Aybars UĞUR, 2013
- Gonzalez, R.C., Woods, R., “Digital Image Processing”, 3rd Edition, Prentice-Hall, 2008.
- <http://www.yildiz.edu.tr/~bayram/sgi/saygi.htm>
- “Image Processing”, https://en.wikipedia.org/wiki/Image_processing
- https://docs.opencv.org/3.4/d8/d4b/tutorial_py_knn_opencv.html
- The article, Primer Defects Detection on Military Cartridge Cases, Refik Samet, Anar Bayram, Serhat Tural, Semra Aydin
- Article, Fuzzy Rule-Based Image Segmentation Technique for Rock Thin Section, Refik Samet, Şahin Emrah Amrahov, Ali Hikmet Ziroğlu,
- Article, Comparison: KNN & SVM Algorithm Dr.Amita Goel, Saarthak Mahajan
- Article, An Application of SVM in Character Recognition with Chain Code Dipti Singh, Mohd. Aamir Khan, Atul Bansal
- Article, Support Vector Machine (SVM) Based Classifier For Khmer Printed Character-set Recognition, Pongsametrey Sok

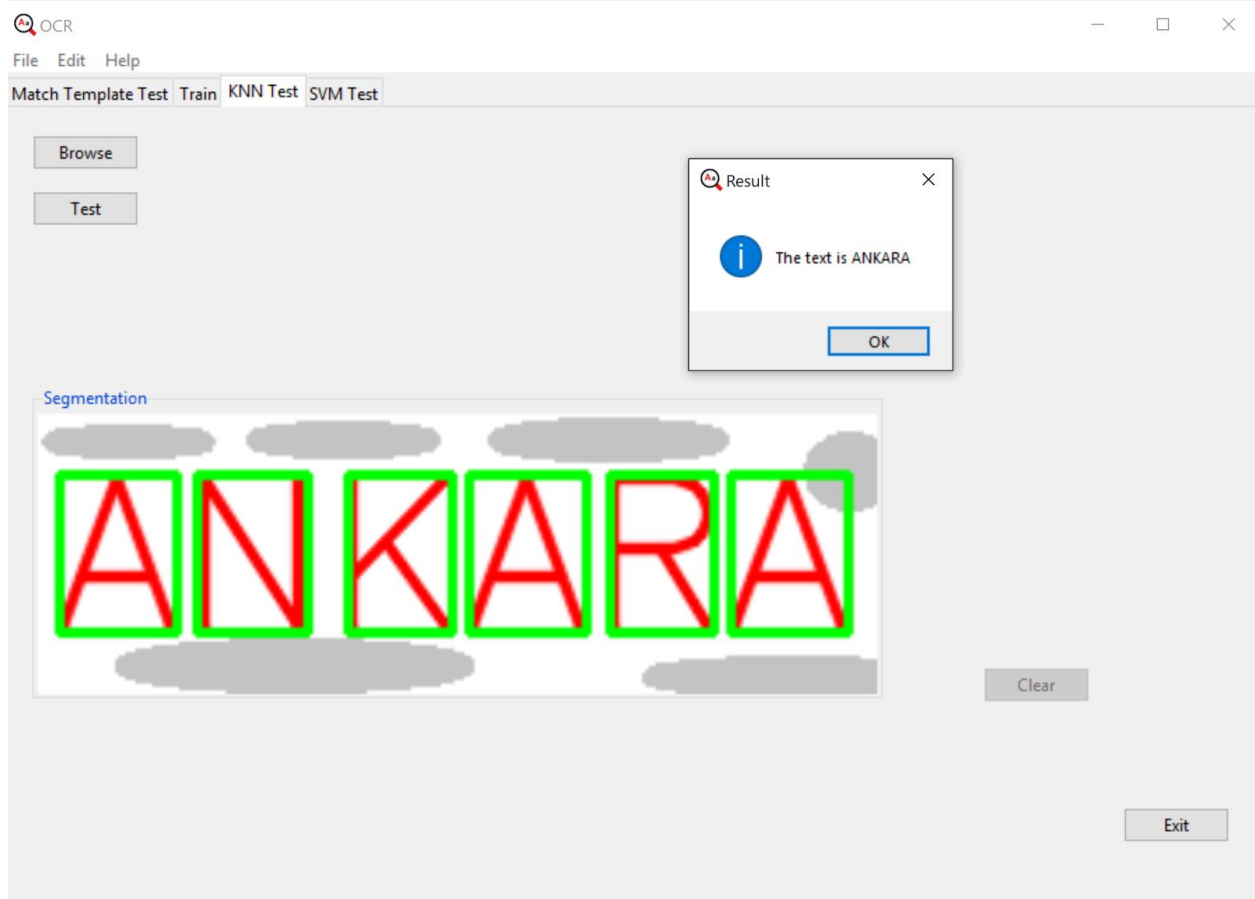
11. APPENDIX

11.1. APPENDIX 1 (IMAGES)









11.2. APPENDIX 2 (CODES)

UserInterface.py

```
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
from tkinter import messagebox
from PIL import ImageTk, Image
import numpy as np
from source import OCR, MatchTemplate, Train, KNearest
import cv2

class userInterface(Tk):
    filename: object
    text = ""
    image_path = ""
    counter = 0
    page1: object
    page2: object
    page3: object
    page4: object
    binaryImage: object
    myOcr: object
    myKNN: object
    myMatch:object
    myTrain:object
    DatasetFrame: object
    DatasetFrameMatch: object
    DatasetFrameKNN: object
    clearButtonMatch :object
    clearButtonKNN :object
    clearButtonTrain:object

    def __init__(self):
        super(userInterface, self).__init__()
        self.title("OCR")
        self.minsize(900, 600)
        self.maxsize(900, 600)
        self.wm_iconbitmap('myicon.ico')
        self.configure(background='#FFFFFF', )
        self.createMenu()
        self.createTabs()
        self.trainUI()
        self.matchTemplateUI()
        self.KNNUI()

    def createTabs(self):
        tab_control = ttk.Notebook(self, width=self.winfo_width(),
height=self.winfo_height())
```

```

self.page2 = ttk.Frame(tab_control)
tab_control.add(self.page2, text='Match Template Test')
self.page1 = ttk.Frame(tab_control)
tab_control.add(self.page1, text='Train')
self.page3 = ttk.Frame(tab_control)
tab_control.add(self.page3, text='KNN Test')
self.page4 = ttk.Frame(tab_control)
tab_control.add(self.page4, text='SVM Test')
tab_control.pack(expand=True, fill=BOTH)
tab_control.grid(column=0, row=0)
def createMenu(self):
    menubar = Menu(self)
    filemenu = Menu(menubar, tearoff=0)
    filemenu.add_command(label="New")
    filemenu.add_separator() # -----
    filemenu.add_command(label="Exit", command=self.quit)
    menubar.add_cascade(label="File", menu=filemenu)
    editmenu = Menu(menubar, tearoff=0)
    editmenu.add_command(label="Cut")
    editmenu.add_command(label="Copy")
    editmenu.add_command(label="Delete")
    menubar.add_cascade(label="Edit", menu=editmenu)
    helpmenu = Menu(menubar, tearoff=0)
    helpmenu.add_command(label="Help Index", command=self.help)
    helpmenu.add_command(label="About...", command=self.About)
    helpmenu.add_command(label="Do you like us?", command=self.like)
    menubar.add_cascade(label="Help", menu=helpmenu)

    self.config(menu=menubar)
def help(self):
    messagebox.showinfo("Help", "Optical Character Recognition(OCR)\n\nOCR is
the mechanical or electronic conversion of images of typed, printed text into
machine-encoded text.")
def like(self):
    messagebox.askyesno("Hey!", "Do you like us?")
def About(self):
    messagebox.showinfo("About Us", "MUSTAFA GOKSEVER\n14290099\n\nMUSTAFA
AHMET DENIZ\n14290087\n\n\nANKARA UNIVERSITY\n2018")
def Clear(self, method, frameName):
    method()
    frameName.destroy()
def trainUI(self):
    self.browseButtonTrain = ttk.Button(self.page1, text="Browse",
                                         command=lambda:
self.filedialog(frame=self.DatasetFrame,
                                         btn=self.preprocessButton))
    self.browseButtonTrain.place(x=20, y=20)
    self.DatasetFrame = ttk.LabelFrame(self.page1, text="Dataset")
    self.preprocessButton = ttk.Button(self.page1, text="Preprocess",
command=lambda:
[self.myTrain.preprocess(),self.segmentationButton.configure(state=NORMAL)])

```

```

self.preprocessButton.configure(state=DISABLED)
self.preprocessButton.place(x=20, y=60)
self.segmentationButton = ttk.Button(self.page1,
                                     text="Segmentation", command=lambda:
self.myTrain.segmentation())
    self.segmentationButton.bind('<Button-1>', self.entryEnable)
    self.segmentationButton.configure(state=DISABLED)
    self.segmentationButton.place(x=20, y=100)
    sv = StringVar()
    sv.trace("w", lambda name, index, mode, sv=sv: self.callback(sv))
    self.entry = ttk.Entry(self.page1, textvariable=sv)
    self.entry.configure(state=DISABLED)
    self.entry.place(x=20, y=140)
    self.exitButton = ttk.Button(self.page1,
                                text="Exit", command=sys.exit)
    self.exitButton.place(x=800, y=500)

    # self.clipboard_clear()
    self.clearButtonTrain = ttk.Button(self.page1, text="Clear",
                                       command=lambda:
self.Clear(method=self.trainUI, frameName=self.DatasetFrame))
    self.clearButtonTrain.configure(state=DISABLED)
    self.clearButtonTrain.place(x=700, y=400)
    def newmethod(self, event):
        self.a = 1
        self.entry.delete(0, END)
        print("enter")
        self.update()
    def callback(self, sv):
        c = sv.get()[0:1]
        sv.set(c)
    def entryEnable(self, event):
        self.entry.configure(state=NORMAL)
        self.update()
    def matchTemplateUI(self):
        self.browseButtonMatch = ttk.Button(self.page2, text="Browse",
command=lambda:
self.filedialog(frame=self.DatasetFrameMatch,
                                     btn=self.testButton1))

        self.browseButtonMatch.place(x=20, y=20)
        self.DatasetFrameMatch = ttk.LabelFrame(self.page2, text="Dataset")
        self.exitButton = ttk.Button(self.page2,
                                     text="Exit", command=sys.exit)
        self.exitButton.place(x=800, y=500)
        self.testButton1 = ttk.Button(self.page2,
                                     text="Test", command=lambda:
self.myMatch.matchTemplate())

        self.testButton1.configure(state=DISABLED)
        self.testButton1.place(x=20, y=60)

```

```

        self.clearButtonMatch = ttk.Button(self.page2, text="Clear",
                                             command=lambda:
self.Clear(method=self.matchTemplateUI,
                                             frameName=self.DatasetFrameMatch))
        self.clearButtonMatch.place(x=700, y=400)
        self.clearButtonMatch.configure(state=DISABLED)

    def KNNUI(self):
        self.browseButtonKNN = ttk.Button(self.page3, text="Browse",
                                             command=lambda:
self.filedialog(frame=self.DatasetFrameKNN,
                                             btn=self.testButton2))

        self.browseButtonKNN.place(x=20, y=20)
        self.exitButton = ttk.Button(self.page3,
                                     text="Exit", command=sys.exit)
        self.exitButton.place(x=800, y=500)
        self.DatasetFrameKNN = ttk.LabelFrame(self.page3, text="Dataset")
        self.testButton2 = ttk.Button(self.page3,
                                       text="Test", command=lambda:
self.myKNN.kNearest())
        self.testButton2.place(x=20, y=60)
        self.testButton2.configure(state=DISABLED)
        self.clearButtonKNN = ttk.Button(self.page3, text="Clear",
                                             command=lambda: self.Clear(method=self.KNNUI,
frameName=self.DatasetFrameKNN))
        self.clearButtonKNN.place(x=700, y=400)
        self.clearButtonKNN.configure(state=DISABLED)

        self.exitButton = ttk.Button(self.page3,
                                     text="Exit", command=sys.exit)
        self.exitButton.place(x=800, y=500)

    def filedialog(self, frame , btn):

        self.filename = filedialog.askopenfilename(initialdir="/", title="Select a
Picture",
                                                    filetype=(('jpeg', '*.jpg'),
('png', '*.png'))))
        if self.filename is "":
            messagebox.showerror("Error", "You did not select any photo! Browse
again!")
        else:
            self.myKNN = KNearest.KNearest(filename=self.filename, gui=myGUI)
            self.myMatch = MatchTemplate.MatchTemplate(filename=self.filename,
gui=myGUI)
            self.myTrain = Train.Train(filename=self.filename, gui=myGUI)
            selectedImage = self.imageOpen(self.filename)
            self.showDatasetfromImage(selectedImage, frame)
            self.image_path = self.filename

```

```

        btn.configure(state=NORMAL)

def showDatasetfromImage(self, image, frame):
    self.Datasetpanel = ttk.Label(frame, image=image)
    self.Datasetpanel.grid(column=0, row=0)
    frame.place(x=20, y=200)
    frame.image = (image)
def getImagePath(self):
    return self.image_path

def imageOpen(self,filename):
    image = Image.open(filename)
    image = image.resize((600, 200), Image.ANTIALIAS)
    img = ImageTk.PhotoImage(image)
    return img
if __name__ == '__main__':
    myGUI = userInterface()

    myGUI.mainloop()

def getGUI():
    return myGUI

```

OCR.py

```

def preprocess(cvImage,gui,DataSetFrameName):
    gray_image = cv2.cvtColor(cvImage, cv2.COLOR_BGR2GRAY)
    showDatasetfromImage(gray_image, "Gray Image",
                           DataSetFrameName, gui=gui)

    blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
    # threshold eksik
    showDatasetfromImage(blurred_image, "Blurred Image", DataSetFrameName,
gui=gui)

    ret, binary = cv2.threshold(gray_image, 127, 256, cv2.THRESH_BINARY_INV)
    showDatasetfromImage(binary, "Binary Image", DataSetFrameName, gui=gui)
    return binary,gray_image

def imageConvert(cvImage):
    cvImage = cv2.resize(cvImage, (600, 200))
    image = Image.fromarray(cvImage)
    image = ImageTk.PhotoImage(image)
    return image

def showDatasetfromImage(image, string, DataSetFrame,gui): # parametre olarak
sleep alabilir
    newImage = imageConvert(image)
    gui.showDatasetfromImage(newImage, DataSetFrame)
    DataSetFrame.configure(text=string)
    gui.update()
    time.sleep(0.3)

```

KNearest.py

```
import os
import cv2
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
from tkinter import messagebox
import numpy as np
import time
import operator
from source import OCR
MIN_CONTOUR_AREA = 100
RESIZED_IMAGE_WIDTH = 20
RESIZED_IMAGE_HEIGHT = 30

class ContourWithData():
    npaContour = None
    boundingRect = None
    intRectX = 0
    intRectY = 0
    intRectWidth = 0
    intRectHeight = 0
    fltArea = 0.0
    def calculateRectTopLeftPointAndWidthAndHeight(self):
        [intX, intY, intWidth, intHeight] = self.boundingRect
        self.intRectX = intX
        self.intRectY = intY
        self.intRectWidth = intWidth
        self.intRectHeight = intHeight

    def checkIfContourIsValid(self):
        if self.fltArea < MIN_CONTOUR_AREA: return False
        return True

class KNearest():
    image: object
    binaryImage: object
    allContoursWithData = []
    validContoursWithData = []
    kNearest: object
    npaROIResized: object
    strFinalString = ""

    def __init__(self, filename, gui):
        self.gui = gui
        self.image_path = filename
        self.image = cv2.imread(self.image_path)
        self.image = cv2.cvtColor(self.image, cv2.COLOR_BGR2RGB)
        self.ocr = OCR
```

```

print("KNN nesnesi olustu")

def kNearest(self):
    time.sleep(0.5)
    self.binary, self.gray_image = self.ocr.preprocess(self.image, self.gui,
self.gui.DatasetFrameKNN)
    im2, contours, hierarchy = cv2.findContours(self.binary, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    for npaContour in contours:
        contourWithData = ContourWithData()
        contourWithData.npaContour = npaContour
        contourWithData.boundingRect =
cv2.boundingRect(contourWithData.npaContour)
        contourWithData.calculateRectTopLeftPointAndWidthAndHeight()
        contourWithData.flTArea = cv2.contourArea(contourWithData.npaContour)
        self.allContoursWithData.append(
            contourWithData)
    self.allContoursWithData.sort(key=operator.attrgetter("intRectX"))
    for a, contourWithData in enumerate(self.allContoursWithData): # for all
contours
        if contourWithData.checkIfContourIsValid():
            self.validContoursWithData.append(contourWithData)
            (x, y, w, h) = cv2.boundingRect(contourWithData.npaContour)
            cv2.rectangle(self.image, (x, y), (x + w, y + h), (0, 255, 0), 2)
            imgROI = self.binary[y:y + h, x:x + w] # imgRoi kullanılmıyor
            self.ocr.showDatasetfromImage(self.image, "Segmentation",
self.gui.DatasetFrameKNN, gui=self.gui)
        try:
            npaClassifications = np.loadtxt("Classifications.txt", np.float32)
        except:
            print("error, unable to open classifications.txt, exiting program\n")
            os.system("pause")
            return
        try:
            npaFlattenedImages = np.loadtxt("Flattened_images.txt", np.float32)
        except:
            print("error, unable to open flattened_images.txt, exiting program\n")
            os.system("pause")
            return
        npaClassifications = npaClassifications.reshape((npaClassifications.size,
1))

        self.kNearest = cv2.ml.KNearest_create() # instantiate KNN object
        self.kNearest.train(npaFlattenedImages, cv2.ml.ROW_SAMPLE,
npaClassifications)
        for contourWithData in self.validContoursWithData:
            cv2.rectangle(self.image,
                (contourWithData.intRectX, contourWithData.intRectY), #
upper left corner
                (contourWithData.intRectX + contourWithData.intRectWidth,
                contourWithData.intRectY +
                contourWithData.intRectHeight), # lower right corner

```



```

        (0, 255, 0), # green
        2) # thickness
    imgROI = self.binary[contourWithData.intRectY: contourWithData.intRectY
+ contourWithData.intRectHeight,
        # crop char out of threshold image
        contourWithData.intRectX: contourWithData.intRectX +
contourWithData.intRectWidth]

    imgROIResized = cv2.resize(imgROI, (RESIZED_IMAGE_WIDTH,
RESIZED_IMAGE_HEIGHT))
    self.npaROIResized = imgROIResized.reshape(
        (1, RESIZED_IMAGE_WIDTH * RESIZED_IMAGE_HEIGHT)) # flatten image
into 1d numpy array
    self.npaROIResized = np.float32(
        self.npaROIResized) # convert from 1d numpy array of ints to 1d
numpy array of floats
    retval, npaResults, neigh_resp, dists =
self.kNearest.findNearest(self.npaROIResized, k=1)
    strCurrentChar = str(chr(int(npaResults[0][0])))
    print(strCurrentChar)
    self.strFinalString = self.strFinalString + strCurrentChar
    print("\n" + self.strFinalString + "\n")
    messagebox.showinfo("Result", "The text is " + self.strFinalString)
    self.strFinalString = ""
    cv2.waitKey(0)
    self.gui.clearButtonKNN.configure(state=NORMAL)

```

MatchTemplate.py

```

import os

import cv2
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
from tkinter import messagebox
import numpy as np
from PIL import ImageTk, Image
import sys
import time
import string
import operator
from source import OCR
MIN_CONTOUR_AREA = 100

RESIZED_IMAGE_WIDTH = 20
RESIZED_IMAGE_HEIGHT = 30
class MatchTemplate(object):
    image: object

```

```

binaryImage: object
def __init__(self, filename, gui):
    self.ocr = OCR
    self.gui = gui
    self.image_path = filename
    self.image = cv2.imread(self.image_path)
    self.image = cv2.cvtColor(self.image, cv2.COLOR_BGR2RGB)
    print("Match Template nesnesi olustu")
def matchTemplate(self):
    text: object
    text = ""
    self.gui.update()
    time.sleep(0.5)
    self.binary, self.gray_image =
self.ocr.preprocess(self.image, self.gui, self.gui.DatasetFrameMatch)
    im2, contours, hierarchy = cv2.findContours(self.binary, cv2.RETR_EXTERNAL,
                                                cv2.CHAIN_APPROX_SIMPLE)

    contours.sort(key=lambda c: np.min(c[:, :, 0]))
    contours = sorted(contours,
                      key=lambda ctr: cv2.boundingRect(ctr)[0] +
cv2.boundingRect(ctr)[1] * self.image.shape[1])
    for a, contour in enumerate(contours):
        (x, y, w, h) = cv2.boundingRect(contour)
        cv2.rectangle(self.image, (x, y), (x + w, y + h), (0, 255, 0), 2)
        imgROI = self.binary[y:y + h, x:x + w]
        self.ocr.showDatasetfromImage(self.image, "Image Segmentation",
                                      self.gui.DatasetFrameMatch, gui=self.gui)
        cv2.imwrite("roi/" + str(a) + '.png', imgROI)
        for x in list(string.ascii_uppercase):
            Adataset = cv2.imread("dataset/" + x + ".png")
            Adataset_gray = cv2.cvtColor(Adataset, cv2.COLOR_BGR2GRAY)
            ret, binarydataset = cv2.threshold(Adataset_gray, 127, 256,
cv2.THRESH_BINARY_INV)
            we, he = imgROI.shape[:,-1]
            res = cv2.matchTemplate(binarydataset, imgROI,
cv2.TM_CCoeff_NORMED)
            threshold = 0.85
            min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
            top_left = max_loc
            bottom_right = (top_left[0] + we, top_left[1] + he)
            if (max_val >= threshold):
                print(x + " bulundu....")
                text = text + x
                break
            time.sleep(0.3)
        messagebox.showinfo("Result", "The text is " + text)
        print("The text is " + text)
        text = ""
        self.gui.clearButtonMatch.configure(state=NORMAL)

```

Train.py

```
import cv2
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
from tkinter import messagebox
import numpy as np
import sys
import time
from source import OCR

class Train(object):
    image: object
    binaryImage: object

    def __init__(self, filename, gui):
        self.gui = gui
        self.ocr = OCR
        self.image_path = filename
        self.image = cv2.imread(self.image_path)
        self.image = cv2.cvtColor(self.image, cv2.COLOR_BGR2RGB)
        print("Train nesnesi olustu")

    def preprocess(self):
        self.binaryImage, self.gray_image = self.ocr.preprocess(self.image,
self.gui, self.gui.DatasetFrame)

    def segmentation(self):
        self.gui.update()
        im2, contours, hierarchy = cv2.findContours(self.binaryImage,
cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

        npaFlattenedImages = np.empty((0, 20 * 30))
        intClassifications = []
        intValidChars = [ord('0'), ord('1'), ord('2'), ord('3'), ord('4'),
ord('5'), ord('6'), ord('7'), ord('8'),
ord('9'), ord('A'), ord('B'), ord('C'), ord('D'),
ord('E'), ord('F'), ord('G'), ord('H'),
ord('I'), ord('J'), ord('K'), ord('L'), ord('M'),
ord('N'), ord('O'), ord('P'), ord('Q'),
ord('R'), ord('S'), ord('T'), ord('U'), ord('V'),
ord('W'), ord('X'), ord('Y'), ord('Z')]
        for npaContour in contours:
            if cv2.contourArea(npaContour) > 100:
                [intX, intY, intW, intH] = cv2.boundingRect(npaContour)
                cv2.rectangle(self.image, #dogru dimi bu
(intX, intY), # upper left corner
(intX + intW, intY + intH), # lower right corner
(0, 0, 255), # red
```

```

        2) # thickness
imgROI = self.binaryImage[intY:intY + intH, intX:intX + intW]
imgROIResized = cv2.resize(imgROI, (20, 30))
self.ocr.showDatasetfromImage(self.image, "Training Numbers
Segmentation",
                                self.gui.DatasetFrame,gui=self.gui)
intChar = self.gui.entry.get()
self.gui.update()
print(intChar)
time.sleep(0.5)
if intChar == 27: # esc
    sys.exit()
elif intChar in intValidChars:
    intClassifications.append(intChar)
    npaFlattenedImage = imgROIResized.reshape((1, 20 * 30))
    npaFlattenedImages = np.append(npaFlattenedImages,
npaFlattenedImage, 0)
    self.gui.entry.bind('<Return>', self.gui.newmethod)
    floatClassifications = np.array(intClassifications, np.float32)
    npaClassifications =
floatClassifications.reshape((floatClassifications.size, 1))
    print("\ntraining complete !!\n")
    np.savetxt("Classifications.txt", npaClassifications)
    np.savetxt("Flattened_images.txt", npaFlattenedImages)
    messagebox.showinfo("Generate Data", "Training complete !!")
    cv2.destroyAllWindows()
    self.gui.clearButtonTrain.configure(state=NORMAL)

```