Second assignment on Digital Image Processing

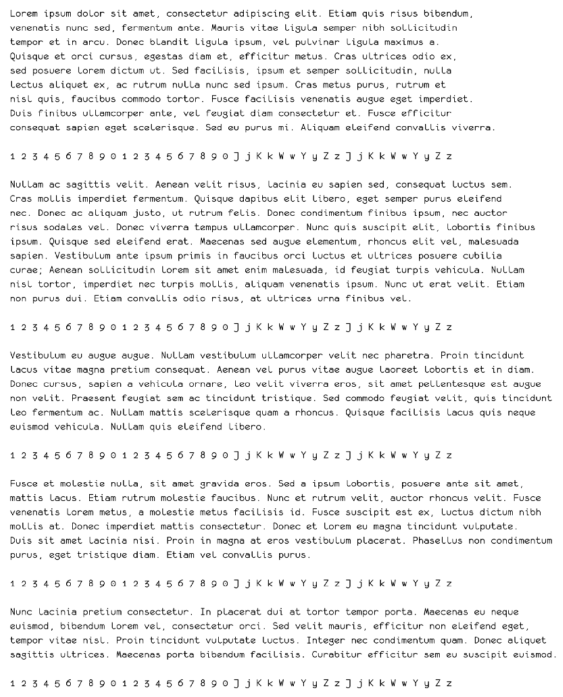
Dimitrios Tikvinas
ECE AUTH
Supervised by:Anastasios Ntelopoulos, Leonidas Alagialoglou
AEM:9998
dtikvina@ece.auth.gr

May 2023

Contents

Abstract

In this assignment we are tasked to create an **OCR** (*Optical Character Recognition*) from scratch. Its usage will be the automatic character recognition of an image containing text of a certain font. We are going to split the whole procedure into 3 main steps. Firstly, the preprocessing of the input text image in order to process it and locate and extract the characters. Secondly, the characters' description as a feature variable using the Discrete Fourier Transform. Lastly, the image's characters' recognition after training and evaluating a KNN model. After completing the OCR, it will be capable of taking any image with any applied rotation containing only text with white background and black letters written in a specific font similar to the training dataset's one and give as output the text in digital form. We will use the *text1* text image and text file incorporating the depicted text for training and the *text2* for testing and evaluation.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam quis risus bibendum,
venenatis nunc sed, fermentum ante. Mauris vitae ligula semper nibh sollicitudin
tempor et in arcu. Donec blandit ligula ipsum, vel pulvinar ligula maximus a.
Quisque et orci cursus, egestas diam et, efficitur metus. Cras ultrices odio ex,
sed posuere lorem dictum ut. Sed facilisis, ipsum et semper sollicitudin, nulla
lectus aliquet ex, ac rutrum nulla nunc sed ipsum. Cras metus purus, rutrum et
nisl quis, faucibus commodo tortor. Fusce facilisis venenatis augue eget imperdiet.
Duis finibus ullamcorper ante, vel feugiat diam consectetur et. Fusce efficitur
consequat sapien eget scelerisque. Sed eu purus mi. Aliquam eleifend convallis viverra.

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 J j K k W w Y y Z z J j K k W w Y y Z z

Nullam ac sagittis velit. Aenean velit risus, lacinia eu sapien sed, consequat luctus sem.
Cras mollis imperdiet fermentum. Quisque dapibus elit libero, eget semper purus eleifend
nec. Donec ac aliquam justo, ut rutrum felis. Donec condimentum finibus ipsum, nec auctor
risus sodales vel. Donec viverra tempus ullamcorper. Nunc quis suscipit elit, lobortis finibus
ipsum. Quisque sed eleifend erat. Maecenas sed augue elementum, rhoncus elit vel, malesuada
sapien. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia
curae; Aenean sollicitudin lorem sit amet enim malesuada, id feugiat turpis vehicula. Nullam
nisl tortor, imperdiet nec turpis mollis, aliquam venenatis ipsum. Nunc ut erat velit. Etiam
non purus dui. Etiam convallis odio risus, at ultrices urna finibus vel.

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 J j K k W w Y y Z z J j K k W w Y y Z z

Vestibulum eu augue augue. Nullam vestibulum ullamcorper velit nec pharetra. Proin tincidunt
lacus vitae magna pretium consequat. Aenean vel purus vitae augue laoreet lobortis et in diam.
Donec cursus, sapien a vehicula ornare, leo velit viverra eros, sit amet pellentesque est augue
non velit. Praesent feugiat sem ac tincidunt tristique. Sed commodo feugiat velit, quis tincidunt
leo fermentum ac. Nullam mattis scelerisque quam a rhoncus. Quisque facilisis lacus quis neque
euismod vehicula. Nullam quis eleifend libero.

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 J j K k W w Y y Z z J j K k W w Y y Z z

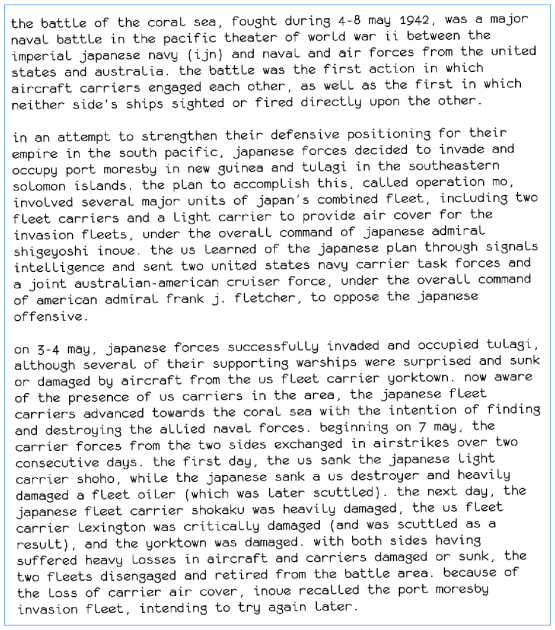
Fusce et molestie nulla, sit amet gravida eros. Sed a ipsum lobortis, posuere ante sit amet,
mattis lacus. Etiam rutrum molestie faucibus. Nunc et rutrum velit, auctor rhoncus velit. Fusce
venenatis lorem metus, a molestie metus facilisis id. Fusce suscipit est ex, luctus dictum nibh
mollis at. Donec imperdiet mattis consectetur. Donec et lorem eu magna tincidunt vulputate.
Duis sit amet lacinia nisi. Proin in magna at eros vestibulum placerat. Phasellus non condimentum
purus, eget tristique diam. Etiam vel convallis purus.

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 J j K k W w Y y Z z J j K k W w Y y Z z

Nunc lacinia pretium consectetur. In placerat dui at tortor tempor porta. Maecenas eu neque
euismod, bibendum lorem vel, consectetur orci. Sed velit mauris, efficitur non eleifend eget,
tempor vitae nisl. Proin tincidunt vulputate luctus. Integer nec condimentum quam. Donec aliquet
sagittis ultrices. Maecenas porta bibendum facilisis. Curabitur efficitur sem eu suscipit euismod.

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 J j K k W w Y y Z z J j K k W w Y y Z z

Figure 1: text1 used for training



the battle of the coral sea, fought during 4-8 may 1942, was a major
naval battle in the pacific theater of world war ii between the
imperial japanese navy (ijn) and naval and air forces from the united
states and australia. the battle was the first action in which
aircraft carriers engaged each other, as well as the first in which
neither side's ships sighted or fired directly upon the other.

in an attempt to strengthen their defensive positioning for their
empire in the south pacific, japanese forces decided to invade and
occupy port moresby in new guinea and tulagi in the southeastern
solomon islands. the plan to accomplish this, called operation mo,
involved several major units of japan's combined fleet, including two
fleet carriers and a light carrier to provide air cover for the
invasion fleets, under the overall command of japanese admiral
shigeyoshi inoue. the us learned of the japanese plan through signals
intelligence and sent two united states navy carrier task forces and
a joint australian-american cruiser force, under the overall command
of american admiral frank j. fletcher, to oppose the japanese
offensive.

on 3-4 may, japanese forces successfully invaded and occupied tulagi,
although several of their supporting warships were surprised and sunk
or damaged by aircraft from the us fleet carrier yorktown. now aware
of the presence of us carriers in the area, the japanese fleet
carriers advanced towards the coral sea with the intention of finding
and destroying the allied naval forces. beginning on 7 may, the
carrier forces from the two sides exchanged in airstrikes over two
consecutive days. the first day, the us sank the japanese light
carrier shoho, while the japanese sank a us destroyer and heavily
damaged a fleet oiler (which was later scuttled). the next day, the
japanese fleet carrier shokaku was heavily damaged, the us fleet
carrier lexington was critically damaged (and was scuttled as a
result), and the yorktown was damaged. with both sides having
suffered heavy losses in aircraft and carriers damaged or sunk, the
two fleets disengaged and retired from the battle area. because of
the loss of carrier air cover, inoue recalled the port moresby
invasion fleet, intending to try again later.

Figure 2: text2 used for testing

Chapter 1

Preprocessing-Rotation undo for text alignment

With the aim of undoing the applied rotation on the given text image input, we will use the function $findRotationAngle$, which takes an input image and computes the angle by which the image is rotated.

Firstly, we apply a Gaussian Filter in the grayscale text image to conjoin the characters of each line with each other to create continuing lines. Then, we take the DFT of the filtered image, remove the DC component to neglect the offset from our following calculations and then shift it to have the zero-frequency component in the center. To get the magnitude spectrum of the image, we employ the equation $spectrum = 10log(1 + |DFT_{shifted}|)$, its peaks and their position in the image array being the decisive factor of a first approximation of the rotation the image initially had.

Moving to a more precise calculation, we utilize a serial search around the first attempt, where for each angle candidate we undo the rotation, calculate the horizontal projections of the brightness and for the maximum ones, meaning that the text lines are aligned with each other horizontally, we will acquire the last rotation angle's approximation. This procedure, due to the range used $(-1 : 0.02 : 1)$ takes the longest to run, slowing the entire project but simultaneously achieving near perfect results.

Using findRotationAngle on the *text1.png* we get

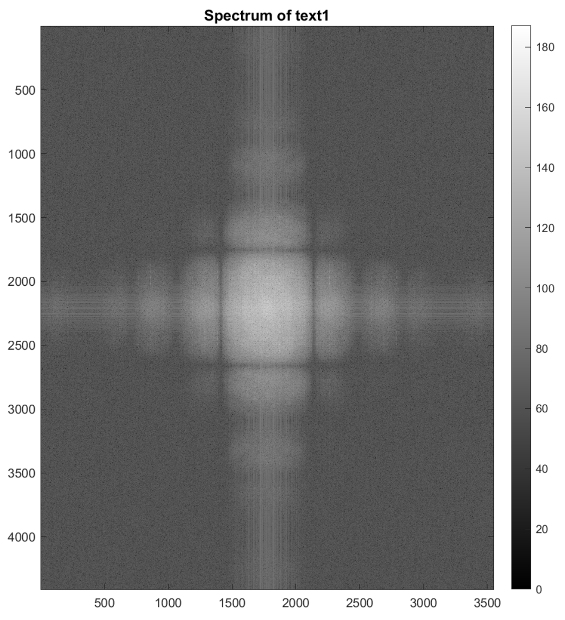
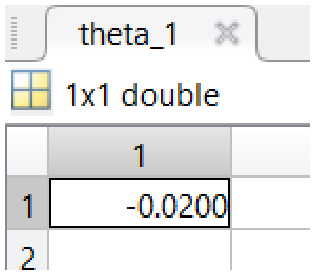


Figure 1.1: Spectrum of text1


theta_1

1x1 double

	1
1	-0.0200
2	

Figure 1.2: Rotation angle of text1

Using findRotationAngle on the *text2.png* we get

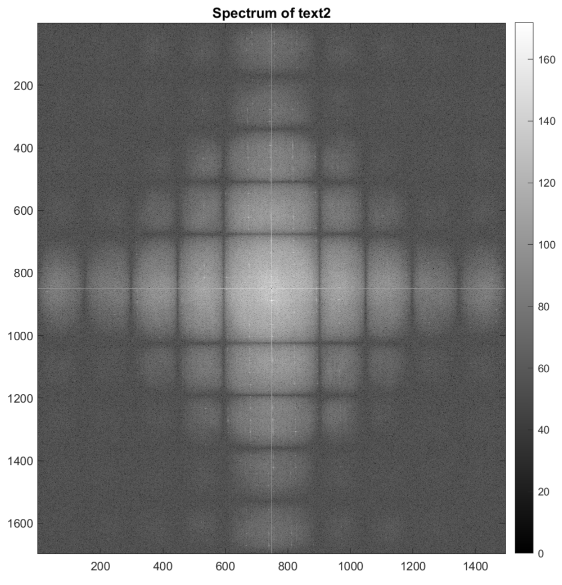
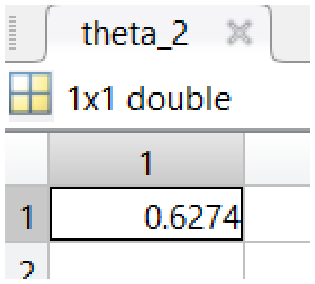


Figure 1.3: Spectrum of text2


theta_2

1x1 double

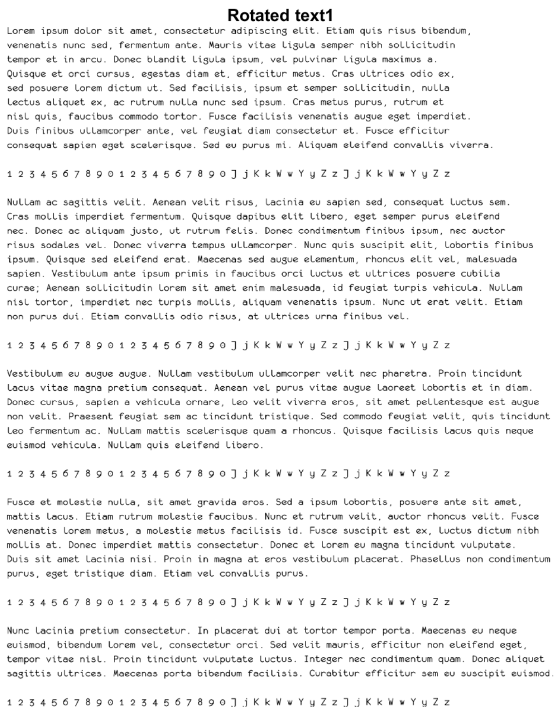
	1
1	0.6274
2	

Figure 1.4: Rotation angle of text2

The white lines appearing on each spectrum are the lines having values above a certain threshold (here $> 0.97 * peak\_value$) , which will be used to approximate the rotation angle. After the serial search, the rotation angles we get are shown in Figure.(1.2) and Figure.(1.3). We deduce that the $theta_1 = -0.02^o$ is actually 0^o, an minimal error due to the range we used for the precise calculation.

So, after finding the rotation angles for both text image, we use the function $rotateImage$, which rotates the input image by the angle we previously found from $findRotationAngle$, generating the output image with aligned text lines with the horizontal axis, the desired outcome.

Firstly, we manually turn any bizarre pixels appearing near the picture's frame into white to get rid of their negative and useless impact in the optical recognition. Afterwards, we calculate the height and width of the generated image and map each pixel from the original image with its new position in the rotated image based on the original center and the rotation angle, all that using trigonometry. Any new pixel that doesn't correspond to some old one is turned into white by default. The final results are shown below.



Rotated text1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam quis risus bibendum,
venenatis nunc sed, fermentum ante. Mauris vitae ligula semper nibh sollicitudin
tempor et in arcu. Donec blandit ligula ipsum, vel pulvinar ligula maximus a.
Quisque et orci cursus, egestas diam et, efficitur metus. Cras ultrices odio ex,
sed posuere lorem dictum ut. Sed facilisis, ipsum et semper sollicitudin, nulla
lectus aliquet ex, ac rutrum nulla nunc sed ipsum. Cras metus purus, rutrum et
nisl quis, faucibus commodo tortor. Fusce facilisis venenatis augue eget imperdiet.
Duis finibus ullamcorper ante, vel feugiat diam consectetur et. Fusce efficitur
consequat sapien eget scelerisque. Sed eu purus mi. Aliquam eleifend convallis viverra.

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 J j K k W w Y y Z z J j K k W w Y y Z z

Nullam ac sagittis velit. Aenean velit risus, lacinia eu sapien sed, consequat luctus sem.
Cras mollis imperdiet fermentum. Quisque dapibus elit libero, eget semper purus eleifend
nec. Donec ac aliquam justo, ut rutrum felis. Donec condimentum finibus ipsum, nec auctor
risus sodales vel. Donec viverra tempus ullamcorper. Nunc quis suscipit elit, lobortis finibus
ipsum. Quisque sed eleifend erat. Maecenas sed augue elementum, rhoncus elit vel, malesuada
sapien. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia
curae; Aenean sollicitudin lorem sit amet enim malesuada, id feugiat turpis vehicula. Nullam
nisl tortor, imperdiet nec turpis mollis, aliquam venenatis ipsum. Nunc ut erat velit. Etiam
non purus dui. Etiam convallis odio risus, at ultrices urna finibus vel.

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 J j K k W w Y y Z z J j K k W w Y y Z z

Vestibulum eu augue augue. Nullam vestibulum ullamcorper velit nec pharetra. Proin tincidunt
lacus vitae magna pretium consequat. Aenean vel purus vitae augue laoreet lobortis et in diam.
Donec cursus, sapien a vehicula ornare, leo velit viverra eros, sit amet pellentesque est augue
non velit. Praesent feugiat sem ac tincidunt tristique. Sed commodo feugiat velit, quis tincidunt
leo fermentum ac. Nullam mattis scelerisque quam a rhoncus. Quisque facilisis lacus quis neque
euismod vehicula. Nullam quis eleifend libero.

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 J j K k W w Y y Z z J j K k W w Y y Z z

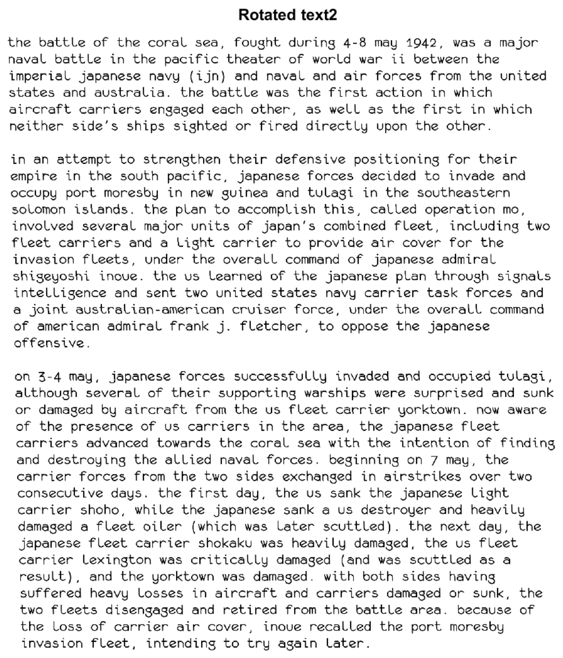
Fusce et molestie nulla, sit amet gravida eros. Sed a ipsum lobortis, posuere ante sit amet,
mattis lacus. Etiam rutrum molestie faucibus. Nunc et rutrum velit, auctor rhoncus velit. Fusce
venenatis lorem metus, a molestie metus facilisis id. Fusce suscipit est ex, luctus dictum nibh
mollis at. Donec imperdiet mattis consectetur. Donec et lorem eu magna tincidunt vulputate.
Duis sit amet lacinia nisi. Proin in magna at eros vestibulum placerat. Phasellus non condimentum
purus, eget tristique diam. Etiam vel convallis purus.

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 J j K k W w Y y Z z J j K k W w Y y Z z

Nunc lacinia pretium consectetur. In placerat dui at tortor tempor porta. Maecenas eu neque
euismod, bibendum lorem vel, consectetur orci. Sed velit mauris, efficitur non eleifend eget,
tempor vitae nisl. Proin tincidunt vulputate luctus. Integer nec condimentum quam. Donec aliquet
sagittis ultrices. Maecenas porta bibendum facilisis. Curabitur efficitur sem eu suscipit euismod.

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 J j K k W w Y y Z z J j K k W w Y y Z z

Figure 1.5: Rotated text1



Rotated text2

the battle of the coral sea, fought during 4-8 may 1942, was a major
naval battle in the pacific theater of world war ii between the
imperial japanese navy (ijn) and naval and air forces from the united
states and australia. the battle was the first action in which
aircraft carriers engaged each other, as well as the first in which
neither side's ships sighted or fired directly upon the other.

in an attempt to strengthen their defensive positioning for their
empire in the south pacific, japanese forces decided to invade and
occupy port moresby in new guinea and tulagi in the southeastern
solomon islands. the plan to accomplish this, called operation mo,
involved several major units of japan's combined fleet, including two
fleet carriers and a light carrier to provide air cover for the
invasion fleets, under the overall command of japanese admiral
shigeyoshi inoue. the us learned of the japanese plan through signals
intelligence and sent two united states navy carrier task forces and
a joint australian-american cruiser force, under the overall command
of american admiral frank j. fletcher, to oppose the japanese
offensive.

on 3-4 may, japanese forces successfully invaded and occupied tulagi,
although several of their supporting warships were surprised and sunk
or damaged by aircraft from the us fleet carrier yorktown. now aware
of the presence of us carriers in the area, the japanese fleet
carriers advanced towards the coral sea with the intention of finding
and destroying the allied naval forces. beginning on 7 may, the
carrier forces from the two sides exchanged in airstrikes over two
consecutive days. the first day, the us sank the japanese light
carrier shoho, while the japanese sank a us destroyer and heavily
damaged a fleet oiler (which was later scuttled). the next day, the
japanese fleet carrier shokaku was heavily damaged, the us fleet
carrier lexington was critically damaged (and was scuttled as a
result), and the yorktown was damaged. with both sides having
suffered heavy losses in aircraft and carriers damaged or sunk, the
two fleets disengaged and retired from the battle area. because of
the loss of carrier air cover, inoue recalled the port moresby
invasion fleet, intending to try again later.

Figure 1.6: Rotated text2

The end result for the text2 image doesn't have the lines aligned vertically. However, this doesn't affect the performance of the OCR, as long as they are aligned horizontally.

Our preprocessing of the input text images comes to an end. Now we can start characters extraction and description.

Chapter 2

Contour Descriptors

Our approach concerning the letters' recognition will be based upon diving them into *classes* according to the number of contours each one possesses and working with each one individually. For example, for our text images, we could split the letters *l, b, B* into 3 classes. "l" will be in class 1(one contour), "b" in class 2 (two contours) and "B" in class 3 (three contours). This line of action simplifies the whole procedure, as we will see in the followings.

2.1 Contours Detection

To implement the above idea, we will first need a way to acknowledge the contours a character has. To do that, we will use the function *getcontour*, which takes an input character image extracted from the text image and produces its contours. As its input we also use the dimensions of the train and testset (meaning text1 and text2 respectively). We need these in order to resize the test characters to the size of the train ones, to be able to compare them correctly and minimize the false predictions occurring due to size incompatibility. Also, the resizing of the lowercase "short" letters and of the uppercase with the lowercase "tall" letters (see "b", "l" against "e", "r"), is done differently. The output will be the *c*, a cell array containing cells of the contours of the input character.

We acquire the contours by turning the letter image into binary and by taking its complement, a line of attack used because we prefer to work with white characters and black background. By subtracting the eroded image from the dilated one and thinning the result as much as possible, we get the contours of the letter. We, then, label them for easier separation and we find its points' coordinates. Lastly, we store each contour into its own cell with dimensions *Nx2*, where 2 is the (x, y) coordinates of each one of the N points belonging in the contour. We will show some examples of this process for the letters :"a", "f", "l", "8" and "e" of the *text1* image.

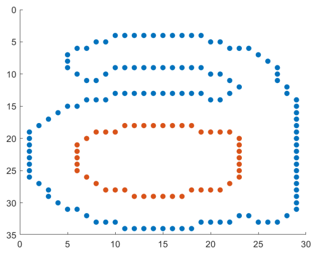


Figure 2.1: Scatter plot of the "a" contours

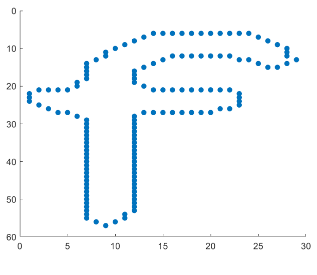


Figure 2.2: Scatter plot of the "f" contours

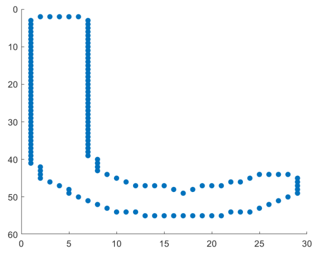


Figure 2.3: Scatter plot of the "l" contours

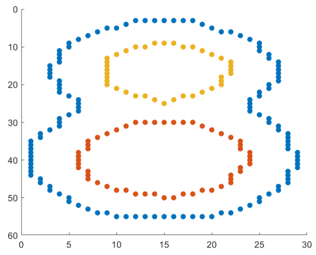


Figure 2.4: Scatter plot of the "8" contours

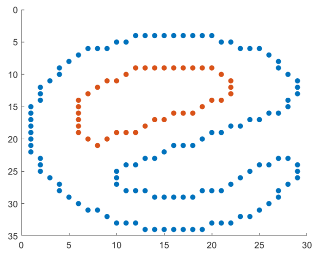


Figure 2.5: Scatter plot of the "e" contours

We can see how accurately the *getcontour* function detects and separates each class's character's contours in these few examples, an extremely useful operation and step for the characters' recognition process.

2.2 Contour Description and Comparison

After acquiring the contours of each character, we want to use a suitable method to operate as descriptor of the contours themselves to provide a comparison metric with an efficient manner. We implement this descriptor with the function *contourDescriptors*, which takes as input the extracted letter's image and the number of contours N we want the descriptor to have for comparison reasons and outputs the desired descriptor.

We structure the descriptor with getting the contours of the input letter image and, for each one it has, we calculate the DFT of the complex sequence $r[i] = x[i]+jy[i]$, where $(x[i], y[i])$ are the coordinates of the point(i), i=1,..,N , exclude the DC component being situated in the first element of the DFT and then, using linear interpolation to set the points of the descriptor to the desired N number, we finally acquire our descriptor. If there are more than one contours in a letter, we add the descriptor of each contour altogether for simplicity.

To compare complex sequences with each other to decide if they look alike or not, we will use the square error of their descriptors as metric and the lesser its value is, the more they are alike. This metric will be later used in the training of K-Nearest Neighbors model as a hyperparameter.

Chapter 3

Characters Recognition and Classification Model

3.1 Characters Extraction from Image

We composed the functions needed for the description of the letter image. Now, we have to implement the function which will detect the letters of the provided image. This will be the *lettersExtraction* function, which takes as input the text image after the restoration of its initial rotation and outputs useful information about it, including the letters found in it.

Its functionality is summarized in taking the text image's gray-scale format and complement in order to have white letters and black background and in each step of the function we calculate the lines, the words in each one line and then the letters in each one word.

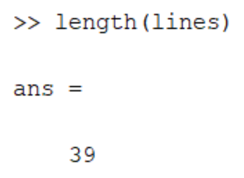
Firstly, by computing the horizontal projection of brightness, we detect the image's rows where text is located and by using block of continuing rows separated by zero projections, we detect the lines.

Secondly, for each line, by computing the vertical projection of brightness after conjoining the letters of each word using the *close* morphological operator, we gather the blocks of continuing columns separated by zero projections and subsequently detect the words. The size of the structural element used is chosen arbitrarily. We could use a metric to decide its size regarding the average spacing between the words of each line.

Thirdly, for each word, we use once again the vertical projections of the brightness to distinguish the letters with each other a continuous set of columns separated by zero projections. This way we acquire the letters of each word. These depictions of the letters not only include the useful pixels describing the contours, but also the pixels of the background. However, this isn't a point of concern, because, when we will feed these letter images into the *getcontour* and *contourDescriptors* functions, only the meaningful pixels will be processed.

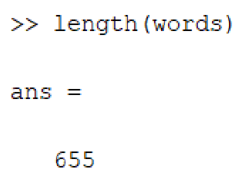
To show the successful character detection it ultimately provides, we use the function on the two text images *text1* and *text2*.

text1:



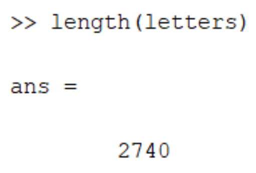
```
>> length(lines)

ans =

    39
```

Figure 3.1: Number of lines detected in text1



```
>> length(words)

ans =

   655
```

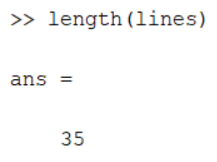
Figure 3.2: Number of words detected in text1



```
>> length(letters)

ans =

        2740
```

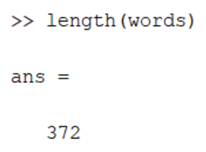
Figure 3.3: Number of letters detected in text1

text2:



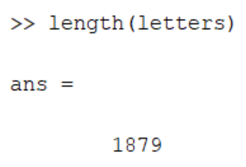
```
>> length(lines)

ans =

    35
```

Figure 3.4: Number of lines detected in text2



```
>> length(words)

ans =

   372
```

Figure 3.5: Number of words detected in text2



```
>> length(letters)

ans =

        1879
```

Figure 3.6: Number of letters detected in text2

Using multiple online word counters on the .txt files of the text images, we find exactly the same measurements, marking the letters extraction a success story.

3.2 Splitting the Dataset into Train and Validation set

For the rest of the report, we will treat the *text2* as an entire test set, while the *text1* will be used to train our classification model on after splitting it into train and validation set. The percentage we are about to split our dataset in will be 70%/30% for train and validation set respectively. We use these proportions in order to succeed in having each and every character of the alphabet represented in the classification model as a feature. We implement this partition using the function *splittingData*, which splits the dataset according to the desired ratio and stores the first part of the dataset as training data and the second as validation data. We don't encourage the usage of a shuffling and random selection due to the nature of the *text1*, being designed to be used for training. A random seed would drop significantly the performance of the classifiers.

3.3 Splitting each set according to contours

Still following the same approach of dividing the classification problem in categories based on the number of contours a letter has, we use the function *splittingDataContours*, which takes as input a set of letters and classifies them accordingly.

3.4 Training a KNN model for each contour class

Our selected classification model will be the **KNN**(*K Nearest Neighbors*), each one trained upon the three contour class train sets (1^{st}, 2^{nd}, 3^{rd}), using as features the contour descriptors . This whole implementation is done using the function $KNNClassifiers$.

To evaluate our models after the training, we use the metrics *confusion matrix* and *weighted accuracy* for each contour class. Also, we define the N number of points in the descriptors for each unique category as an extra hyperparameter. As we stated before, the metric to decide how much two complex sequences produced by the contour's points look alike is the squared error between the descriptors we formulated. That's the reason why in the KNN' classifier's training we will use the *euclidean* as the distance metric. Another noteworthy hyperparameter is the pixel size of the structural element we used to detect the words of the text image. After multiple experiments on our datasets, we found that for disk's pixel size = 8, we get the most accurate results. Moreover, another hyperparameter worth mentioning is the extra scale by which we resize the letter images after the size alignment between the training set and the entire test set, before extracting its contours. We found that for *scale_ratio* = 1.25, we get the most precise results.

After dealing with these hyperparameters, let's see some metrics for our models, after predicting on the validation set of *text1*. Each class works independently of the others. That's why we run three individual grid searches, breaking the entire hyperparameter optimization problem into three parts.

Class 1:

N_1	K_1	WA_1
100	5	99.36%
150	5	98.89%
200	5	99.00%
250	5	99.06%
300	5	98.96%
350	5	99.06%
250	3	99.56%
250	5	99.06%
250	10	99.20%
250	15	98.19%
250	20	97.78%

Class 2:

N_2	K_2	WA_2
200	2	100%
250	2	100%
250	3	99.70%

Class 3:

N_3	K_3	WA_3
200	2	100%
250	2	100%
250	3	100%

We can conclude that for the classes 2 and 3 we achieve easily enough 100% weighted accuracy. In the other hand, for the class 1, which contains the highest number of characters, changes either on the N number of descriptor points or K Nearest Neighbors parameter have some impact, keeping the weighted accuracy at most times above 99%. The best hyperparameters values we choose after these grid searches on the validation set are ***Class 3***:

N_1	K_1	WA_1
250	3	99.56%
N_2	K_2	WA_2
250	3	99.70%
N_3	K_3	WA_3
250	3	100%

We tried to avoid the minimum values in order to not converge into overfitting.

Also, we could increase the performance of our OCR system using ways such as using the morphological operators open or close (depends whether we act upon black or white letters respectively) to diminish any noise around the necessary letter pixels.

3.5 Using the trained OCR on a new Text Image

To optically recognize the *text2* image, we will use the trained KNN model. The image contains characters with font similar to the one we trained our model on, leading to an a-priori acknowledgment of the high weighted accuracies we are about to achieve. We implement the predictions using the function *readtext*.

Let's see the predictions made for the above trained model and after some changes.

Class 1:

N_1	K_1	WA_1
250	3	98.38%
250	5	99.33%

Class 2:

N_2	K_2	WA_2
200	2	98.06%
250	3	98.69%

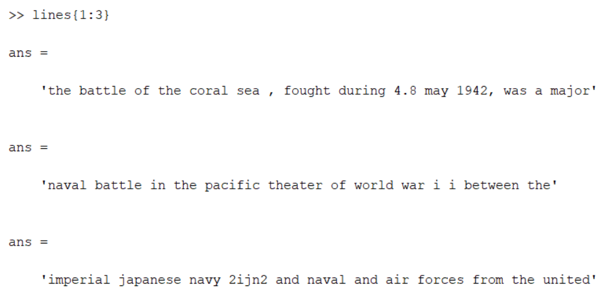
Class 3:

N_3	K_3	WA_3
250	3	100%

We come to the conclusion that for the text2 prediction, the most suitable hyperparameters values are $(\mathbf{K_1}, \mathbf{K_2}, \mathbf{K_3} = \mathbf{5}, \mathbf{3}, \mathbf{2})$ and $(\mathbf{N_1}, \mathbf{N_2}, \mathbf{N_3} = \mathbf{250}, \mathbf{250}, \mathbf{250})$, achieving weighted accuracies **99.33%**, **98.69% and 100%** for each contour class respectively.

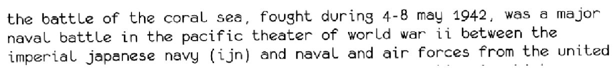
3.6 Conclusion

The whole purpose of this OCR has been from the right beginning to store in digital form the text extracted from an image, for instance a book's page, if we train our KNN classifiers with the a complete dataset of the target's font. To show that we succeeded in doing that, we store in the *lines* cell array the predicted text, where each cell corresponds to a text's line. For example, in Figure.3.7, we have the first 3 lines of the predicted test set *text2* and for comparison reasons, in Figure.3.8, we have the first line of the image. The classifier couldn't recognize the '(' and ')' characters due to their absence in the training set. That's why it was of paramount importance each and every known character to be included in it, specifically here in *text1*.



```
>> lines{1:3}

ans =

    'the battle of the coral sea , fought during 4.8 may 1942, was a major'


ans =

    'naval battle in the pacific theater of world war i i between the'


ans =

    'imperial japanese navy 2ijn2 and naval and air forces from the united'
```

Figure 3.7: Predicted first line of text2 image



the battle of the coral sea, fought during 4-8 may 1942, was a major
naval battle in the pacific theater of world war ii between the
imperial japanese navy (ijn) and naval and air forces from the united

Figure 3.8: Actual first line of text2 image

We mark our endeavor a success