# TEXT LINE SEGMENTATION FOR MEDIEVAL MANUSCRIPTS

*A project report submitted to*

**Rajiv Gandhi University of Knowledge Technologies**

**SRIKAKULAM**

**In partial fulfilment of the requirements for the**

**Award of the degree of**

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted by**

**3rd year B. Tech 2nd semester**

**B. Manasa S180122**

**I. Praveen Kumar S180221**

**J. Bhuvana Sree S180073**

**Under the Esteemed Guidance of**

**Mr. G. Shiva Rama Sastry, (M. Tech, PhD)**



**Rajiv Gandhi University of Knowledge Technologies - SKLM**

TEXT LINE SEGMENTATION FOR MEDIEVAL MANUSCRIPTS

**BONAFIED CERTIFICATE**

This is to certify that the mini project report titled **“TEXTLINE SEGMENTATION FOR MEDIEVAL MANUSCRIPTS”** was successfully completed by **B. MANASA (S180122), I. PRAVEEN KUMAR (S180221), J. BHUVANA SREE (S180073)** under the guidance of **Mr. G. Shiva Rama Sastry, (M. Tech, PhD)** In partial fulfillment of the requirements for the Mini Project in Computer Science and Engineering of **Rajiv Gandhi University of Knowledge Technologies** under my guidance and output of the work carried out is satisfactory.

Mr. G. Shiva Rama Sastry, (M. Tech, PhD) Mr. N. Sesha Kumar, (M.Tech)

Assistant Professor Head of the department

PROJECT GUIDE PROJECT CO-ORDINATOR

TEXT LINE SEGMENTATION FOR MEDIVAL MANUSCRIPTS

**DECLARATION**

We declared that this thesis work titled **“TEXT LINE SEGMENTATION FOR MEDIEVAL MANUSCRIPTS”** is carried out by us during the year 2022-2023 in partial fulfillment of the requirements for the Mini Project in **Computer Science and Engineering.** We further declare that this dissertation has not been submitted elsewhere for any Degree. The matter embodied in this dissertation report has not been submitted elsewhere for any other degree. Furthermore, the technical details furnished in various chapters of this thesis are purely relevant to the above project and there is no deviation from the theoretical point of view for design, development and implementation.

**With Regards**

**Bojja Manasa S180122**

**Ippili Praveen Kumar S180221**

**Jami Bhuvana Sree S180073**

TEXT LINE SEGMENTATION FOR MEDIEVAL MANUSCRIPTS

## ACKNOWLEDGEMENT

We would like to articulate my profound gratitude and indebtedness to our project guide **Mr. G. Shiva Rama Sastry**, who has always been a constant motivation and guiding factor throughout the project time. It has been a great pleasure for us to get an opportunity to work under his guidance and complete the thesis work successfully.

We wish to extend our sincere thanks to **Mr. N. Shesha Kumar** Head of the Computer Science and Engineering Department, for her constant encouragement throughout the project. We are also grateful to other members of the department without their support our work would have been carried out so successfully.

I thank one and all who have rendered help to me directly or indirectly in the completion of my thesis work.

**Project Associate:**

**Bojja Manasa S180122**

**Ippili Praveen Kumar S180221**

**Jami Bhuvana Sree S180073**

TEXT LINE SEGMENTATION FOR MEDIEVAL MANUSCRIPTS

**ABSTRACT**

Text line segmentation is an essential step in the digitization of medieval manuscripts. These manuscripts are often written in scripts that vary in size, style, and spacing. This process involves identifying individual lines of text within an image of a manuscript page. It is a challenging task due to noise, degradation, and variations in the manuscript layout and format. In recent years, text line segmentation techniques have advanced, including traditional image processing-based methods. This paper provides an overview of these techniques and their performance on medieval manuscript datasets, highlighting the advantages and limitations of each approach. Additionally, the paper identifies open research challenges and future directions for text line segmentation for medieval manuscripts.

**Keywords:** Text line segmentation, medieval manuscripts, Computer Vision, Image processing, Seam Carving Algorithm.

**TABLE OF CONTENTS**

[TEXT LINE SEGMENTATION FOR MEDIEVAL MANUSCRIPTS i](#_Toc140403982)

[ACKNOWLEDGEMENT iv](#_Toc140403983)

[CHAPTER-1 1](#_Toc140403984)

[INTRODUCTION 1](#_Toc140403985)

[1.1 Introduction 1](#_Toc140403986)

[1.2 Statement of the problem 1](#_Toc140403987)

[1.3 Objectives 1](#_Toc140403988)

[1.4 Goal 1](#_Toc140403989)

[1.5 Scope 2](#_Toc140403990)

[1.6 Applications 2](#_Toc140403991)

[1.7 Limitations 2](#_Toc140403992)

[CHAPTER-2 3](#_Toc140403993)

[LITERATURE SURVEY 3](#_Toc140403994)

[2.1 Collect Information 3](#_Toc140403995)

[2.2 Study 3](#_Toc140403996)

[2.3 Benefits 3](#_Toc140403997)

[2.4 summary 3](#_Toc140403998)

[CHAPTER -3 4](#_Toc140403999)

[ANALYSIS 4](#_Toc140404000)

[3.1 Existing system 4](#_Toc140404001)

[3.2 Disadvantages 4](#_Toc140404002)

[3.3 Proposed system 4](#_Toc140404003)

[3.4 Advantages 4](#_Toc140404004)

[3.5 system requirements 4](#_Toc140404005)

[**3.5.1 Software requirements: 4**](#_Toc140404006)

[**3.5.2 Hardware requirements: 4**](#_Toc140404007)

[CHAPTER -4 5](#_Toc140404008)

[SYSTEM IMPLEMENTATION 5](#_Toc140404009)

[4.1 TEXT LINE SEGMENTATION FOR MEDIEVAL MANUSCRIPTS 5](#_Toc140404010)

[CHAPTER-5 9](#_Toc140404011)

[SOURCE CODE 9](#_Toc140404012)

[5.1 CODE 9](#_Toc140404013)

[CHAPTER-6 53](#_Toc140404014)

[SYSTEM TESTING 53](#_Toc140404015)

[6.1 INTRODUCTION 53](#_Toc140404016)

[6.2 TYPES OF TESTS : 53](#_Toc140404017)

[Conclusion 55](#_Toc140404018)

[Appendices 55](#_Toc140404019)

[References: 55](#_Toc140404020)

# CHAPTER-1

# INTRODUCTION

## 1.1 Introduction

Text Line Segmentation is an essential step in the digitalization of medieval manuscripts which uses computer vision and traditional image-processing techniques. These manuscripts, rich in the historical and cultural significance, contain hand written text arranges in lines, often with decorations and irregular layouts. Traditional segmentation algorithms designed for modern printed text often struggle to handle the complexities variability present in the medieval manuscripts. Therefore the development of specialized and techniques tailored to the specific requirements of medieval manuscripts is necessary. This project is to identify and separate individual lines of text, providing a foundation for subsequent analysis and processing.

## 1.2 Statement of the problem

The problem at hand is the precise segmentation of text lines in mediaeval manuscripts using a seam carving method. Due to variable line spacing, decorative embellishments, fading or damaged text, and different handwriting styles, mediaeval manuscripts provide special difficulties for text line segmentation. The algorithm must effectively divide text into lines while supporting irregular line spacing, correctly identifying and handling decorative components, and adjusting to variances in handwriting styles frequently seen in these manuscripts.

## 1.3 Objectives

The major goal of our effort is to use the following steps to split the text lines of mediaeval manuscripts for further analysis.

1) Image loading

2) Preprocessing

3) Generate an energy map

4) Seam carving

5) Binning

6) Polygon manager

7) Text line segmentation

## 1.4 Goal

The goal of this research is to accurately segment text into individual lines in mediaeval manuscripts using a seam carving algorithm. To handle the special qualities of mediaeval manuscripts, such as inconsistent line spacing, artistic embellishments, faded or damaged text, and a variety of handwriting styles, the chosen algorithm will be customized and fine-tuned. In order to effectively digitize and analyze mediaeval manuscripts for historical study and preservation purposes, it is important to obtain high accuracy and reliability in text line segmentation.

## 1.5 Scope

**The scope of The Text Line Segmentation for medieval manuscripts includes:**

* The seam carving method is better suited to tackle the particular difficulties of mediaeval manuscripts, such as irregular line spacing and changes in handwriting styles.
* Our project uses seam carving algorithm for text line segmentation of mediaeval manuscripts.
* For our project, we selected one dataset of mediaeval manuscripts and processed them digitally.
* Our project does some preprocessing and postprocessing procedures on the input image in order to implement the modified algorithm.
* To show the algorithm's value for historical study and preservation efforts, utilize actual samples of mediaeval manuscripts.

## 1.6 Applications

Text line segmentation used in various application areas some of those are

1. Document Layout Analysis: Text line segmentation is an essential step in document layout analysis, where the goal is to understand the structure and organization of a document. The segmented text lines can be used to identify paragraphs, headings, captions, and other structural elements within a document.

2. OCR Systems: Optical character recognition (OCR) systems aim to convert scanned or printed documents into machine-readable text. Text line segmentation is a crucial component of OCR systems as it helps isolate individual lines of text, enabling accurate character recognition and text extraction.

3. Text Recognition in Images: The segmented text lines can be used for text recognition in images, such as street signs, vehicle license plates, or product labels. By segmenting the text lines, the system can focus on recognizing and understanding the text content more accurately.

## 1.7 Limitations

Currently, our project only supports one chosen dataset we need to add additional functionalities for this project to work any other type of Manuscripts.

# CHAPTER-2

# LITERATURE SURVEY

## 2.1 Collect Information

Information for the Text Line segmentation for Medieval Manuscripts system was collected from various sources, including research papers, existing manuscript databases, online resources and academic publications.

## 2.2 Study

**TEXT LINE SEGEMENTATION FOR MEDIEVAL MANUSCRIPTS**

Image and Text Segmentation pipeline for the paper **"Labeling, Cutting, Grouping: an Efficient Text Line Segmentation Method for Medieval Manuscripts",** published at the 15th IAPR International Conference on Document Analysis and Recognition (ICDAR) in 2019.

## 2.3 Benefits

* Text line segmentation
* Optical character recognition
* Text recognition in medieval manuscripts

## 2.4 summary

Text line segmentation for medieval manuscripts involves a series of steps to accurately detect and separate lines of text within the historical documents which are in various languages.

# CHAPTER -3

# ANALYSIS

## 3.1 Existing system

Text line segmentation use modern text documents as their dataset which aims to improve the accuracy of OCR systems or assist in document analysis tasks.

## 3.2 Disadvantages

Text line segmentation for modern text document is not that much complex. But when coming to the historical manuscripts its often challenging task because these papers suffer from degradation, noisy, contain ornaments and decorations, contain varying font sizes and scripts.

## 3.3 Proposed system

Text line segmentation for medieval manuscripts use images of historical manuscripts which based on seam carving algorithm and focus on preserving and digitizing historical documents.

## 3.4 Advantages

1. Improved Document Analysis

2. Precise text recognition

3. Efficient Document Processing

4. Research Contribution

## 3.5 system requirements

### 3.5.1 Software requirements:

* Google Colab
* Windows 10

### 3.5.2 Hardware requirements:

* RAM: 4GB above
* Hard disk: 5 GB above

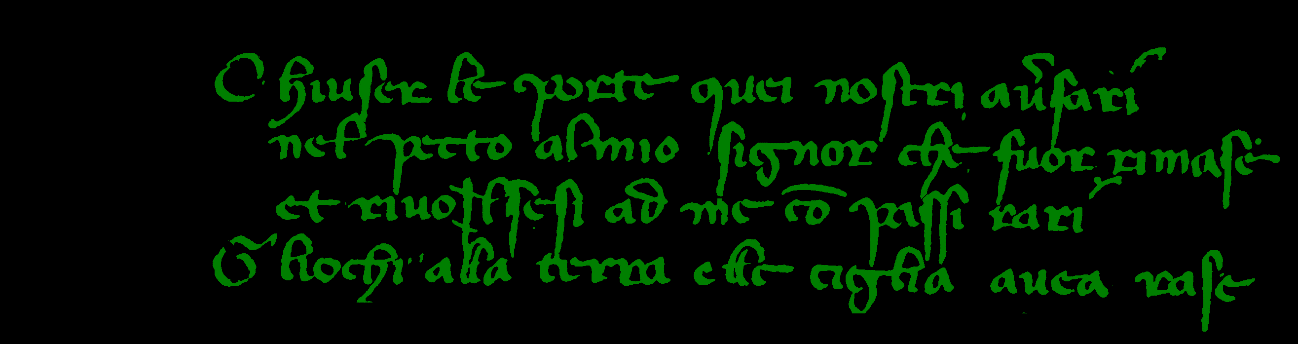
# CHAPTER -4

# SYSTEM IMPLEMENTATION

## 4.1 TEXT LINE SEGMENTATION FOR MEDIEVAL MANUSCRIPTS

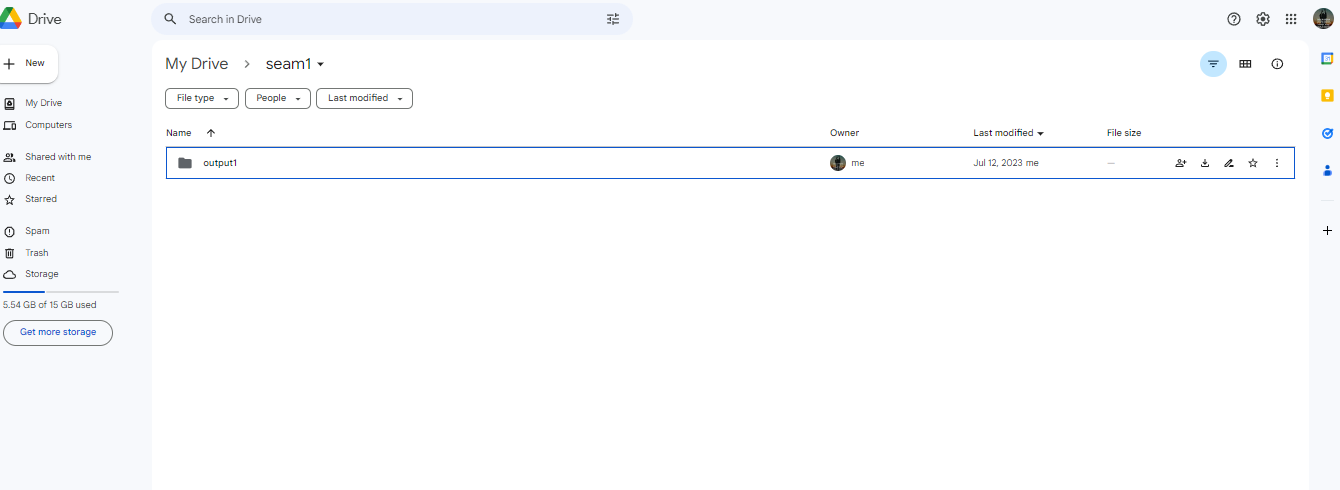
This project is done by using python language based on computer vision and traditional image processing techniques .Mainly we are using Open CV module in this project .In this project we load, preprocess the input image and calculates the energy map of the preprocessed image and then we apply seam carving algorithm for generating seams after that we applied binning and polygon manager for efficient output.

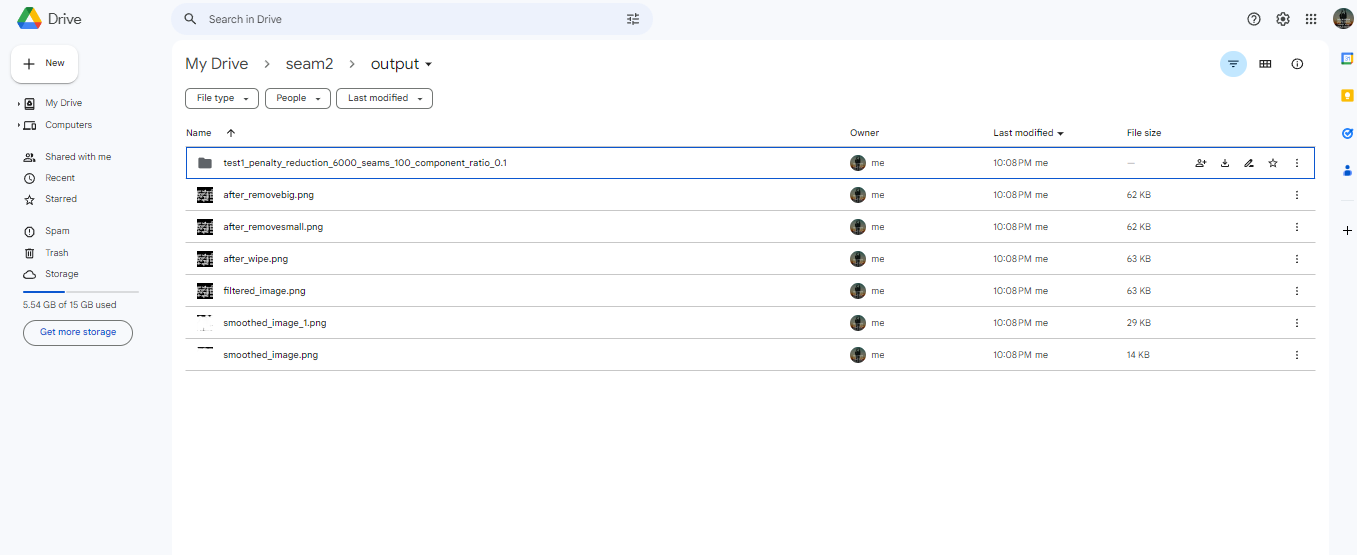
**Input Image:**

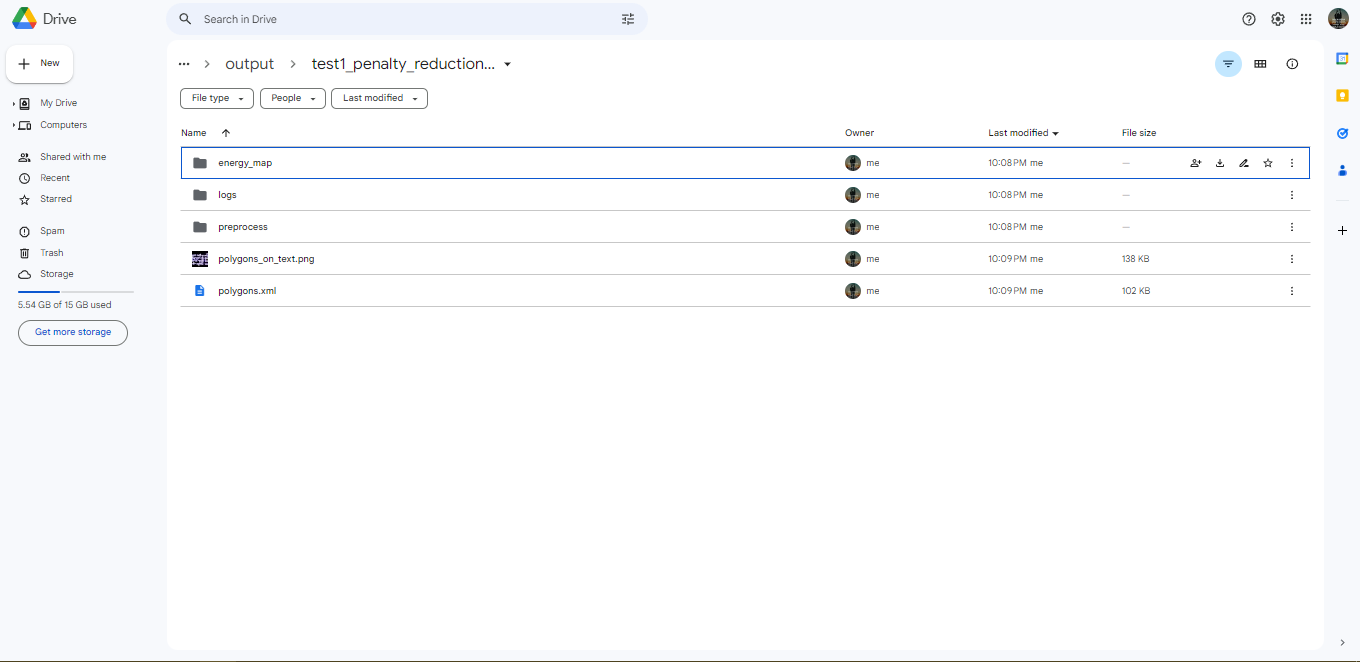
****

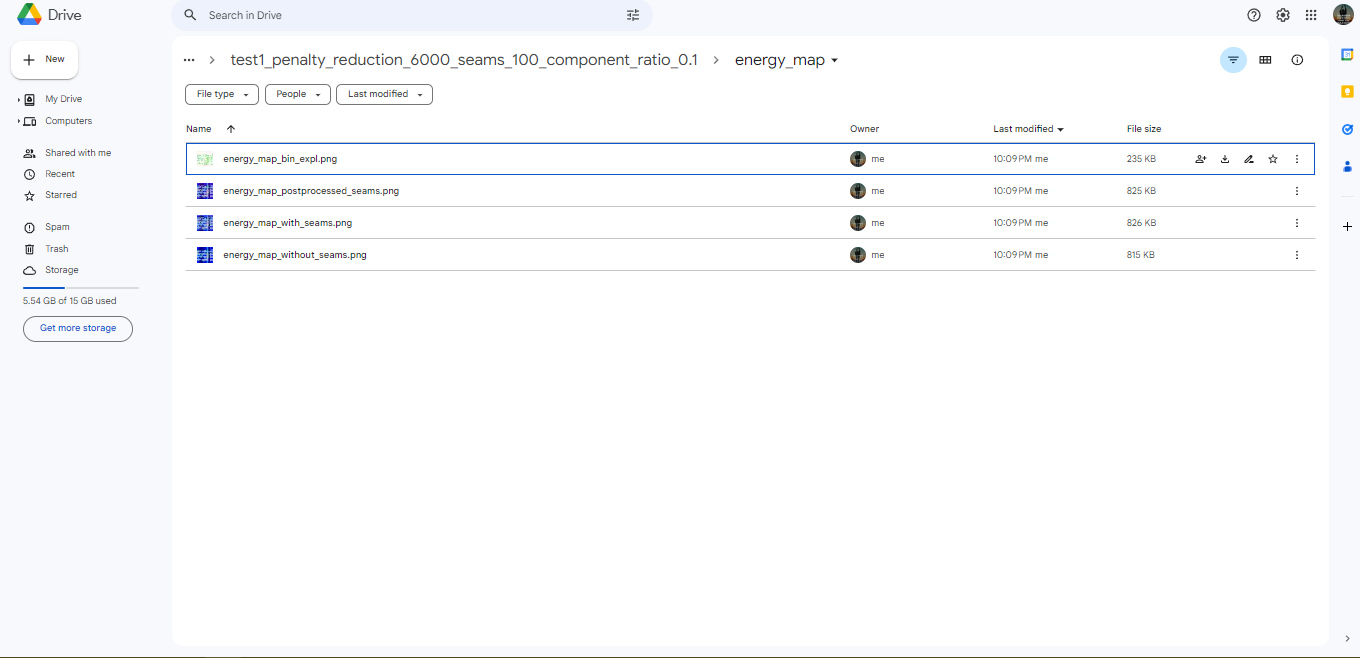
The input image is given to the system it produce output in a folder through the path specified in the code

**Output folder:**

****

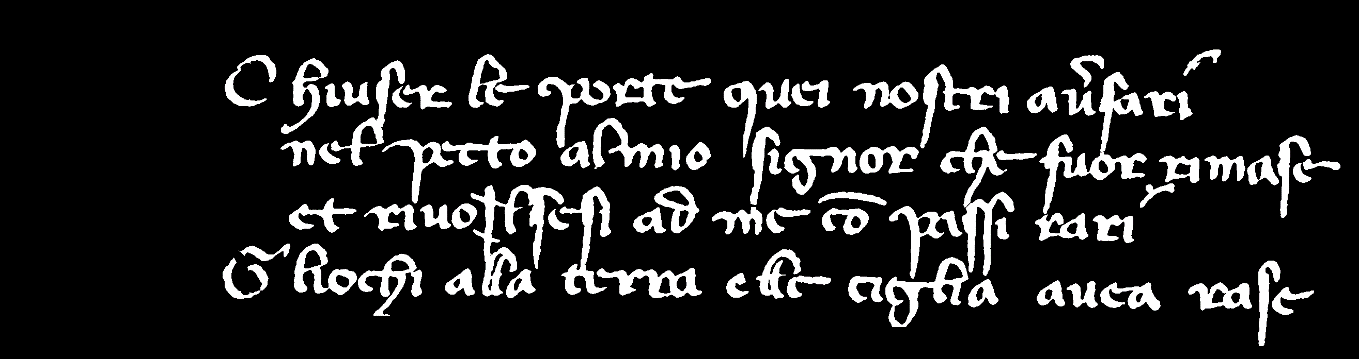
****

****

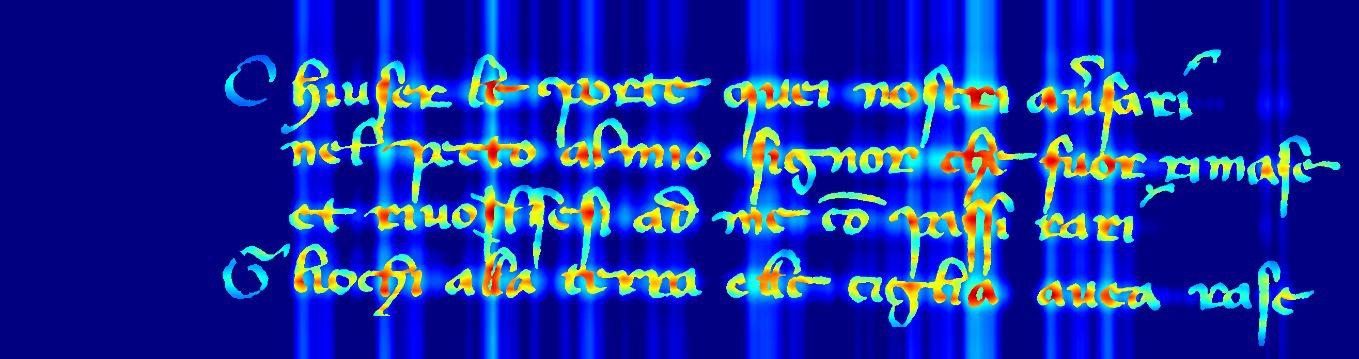
****

**Output**

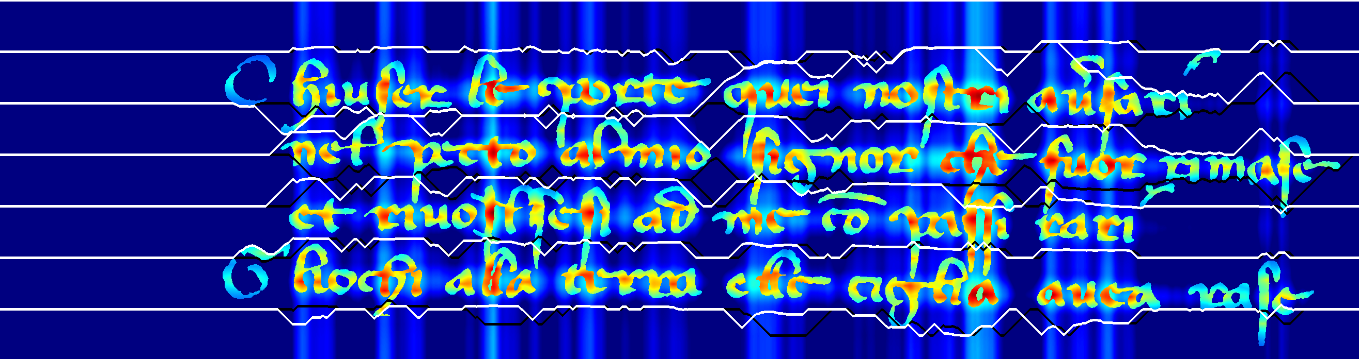
**After preprocessing**

****

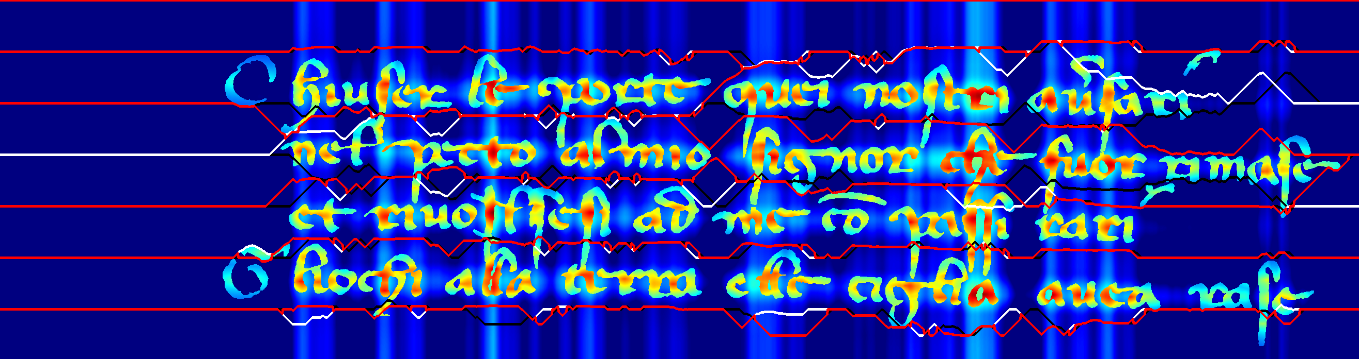
**Energy map**

****

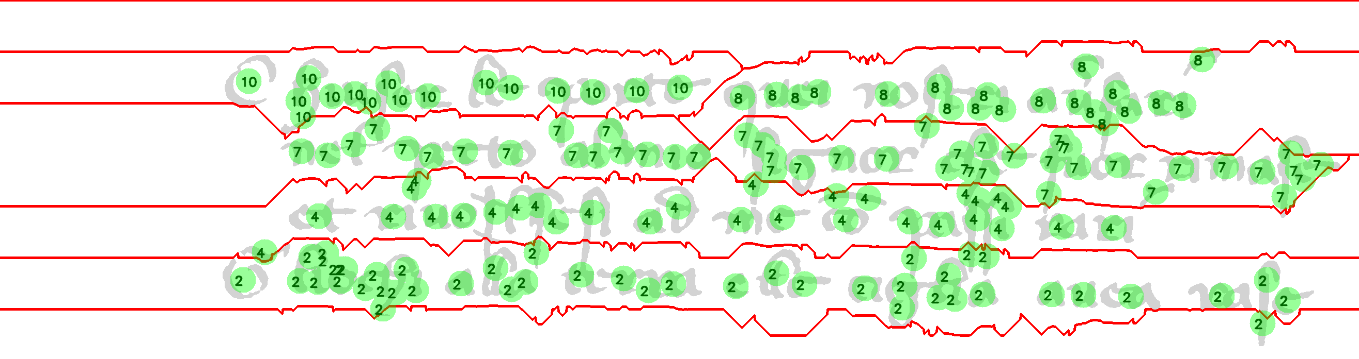
**Energy map with seams**

****

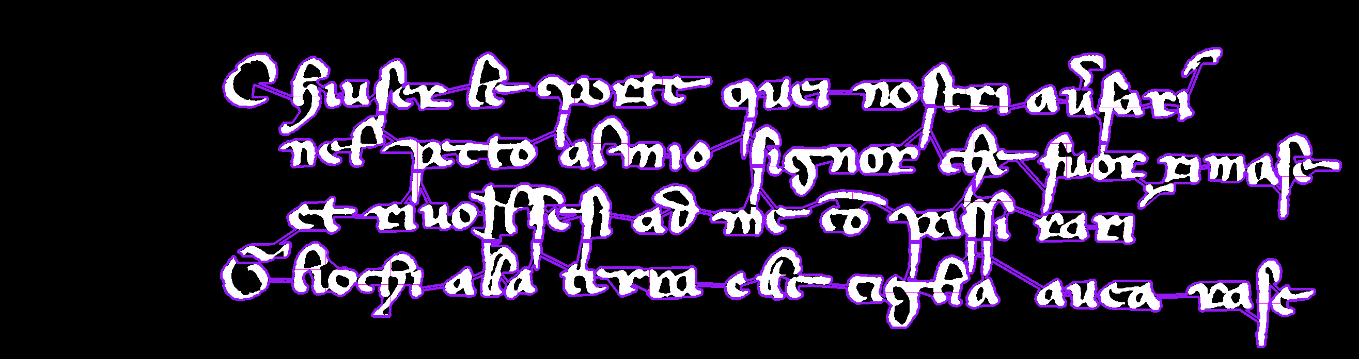
**Energy map with post preprocessed seams**

****

**After binning:**

****

**Polygons on text:**

****

# CHAPTER-5

# SOURCE CODE

## 5.1 CODE

**Drive Mount**

from google.colab import drive

drive.mount("/content/drive", force\_remount=True)

import sys

sys.path.insert(0, '/content/drive/MyDrive/seam1')

%cd /content/drive/MyDrive/seam1**/**

**Load Image**

import errno

import cv2

import logging

import os

import time

import numpy as np

from PIL import Image

def load\_image(path):

if not os.path.exists(path):

raise FileNotFoundError(errno.ENOENT, os.strerror(errno.ENOENT), path)

if os.path.splitext(path)[1] == '.tif':

img = np.asarray(Image.open(path), dtype=int)

img[np.where(img == 0)] = 8

img[np.where(img == 1)] = 0

img = np.stack((img,)\*3, axis=-1)

img[:, :, 1] = 0

img[:, :, 2] = 0

else:

img = cv2.imread(path)

if img is None:

raise Exception("Image is empty or corrupted", path)

return img

def prepare\_image(img, testing, cropping=True, vertical=False):

start = time.time()

if testing:

img[:, :, 0] = 0

img[:, :, 2] = 0

locations = np.where(img == 127)

img[:, :, 1] = 0

img[locations[0], locations[1]] = 255

if cropping:

locs = np.array(np.where(img == 255))[0:2, ]

img = img[np.min(locs[0, :]):np.max(locs[0, :]), np.min(locs[1, :]):np.max(locs[1, :])]

else:

assert len(np.unique(img[:, :, 1])) == 1

assert np.unique(img[:, :, 1])[0] == 0

assert len(np.unique(img[:, :, 2])) <= 2

assert np.unique(img[:, :, 2])[0] == 0

if len(np.unique(img[:, :, 2])) > 1:

assert np.unique(img[:, :, 2])[1] == 128

locations = np.where(img == 128)

img[locations[0], locations[1]] = 0

locations\_text = np.where(img == 8)

locations\_text\_decoration = np.where(img == 12)

img[:, :, :] = 0

img[locations\_text[0], locations\_text[1]] = 255

img[locations\_text\_decoration[0], locations\_text\_decoration[1]] = 255

if vertical:

img = cv2.rotate(img, cv2.ROTATE\_90\_COUNTERCLOCKWISE)

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return img

**Preprocessing Image**

import logging

import os

import sys

import time

import cv2

from skimage import measure

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

import numpy as np

def preprocess(image, small\_component\_ratio):

start = time.time()

image = wipe\_outside\_textarea(image)

save\_img(image, path=os.path.join('./output', 'after\_wipe.png'), show=False)

image = remove\_small\_components(image, small\_component\_ratio)

save\_img(image, path=os.path.join('./output', 'after\_removesmall.png'), show=False)

image = remove\_big\_components(image)

save\_img(image, path=os.path.join('./output', 'after\_removebig.png'), show=False)

image[image > 255] = 255

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return image

def wipe\_outside\_textarea(image):

ORIGINAL = image

image = image[:, :, 1]

filter\_size\_H = 64

filter\_size\_V = 192

kernel = np.ones((filter\_size\_V, filter\_size\_H)) / filter\_size\_H

image = cv2.filter2D(image, -1, kernel)

image[5:-5, int(image.shape[1] / 2) - 5:int(image.shape[1] / 2) + 5] = 255

save\_img(image, path=os.path.join('./output', 'smoothed\_image\_1.png'), show=False)

tmp = np.ones((image.shape[0], image.shape[1], 3), dtype=np.uint8)//

cc = measure.find\_contours(image, 200, fully\_connected='high')[0]

cc[:, 0], cc[:, 1] = cc[:, 1], cc[:, 0].copy()

cc = [cc.astype(np.int32, copy=False)]

cv2.fillPoly(tmp, cc, (255, 255, 255))

save\_img(tmp, path=os.path.join('./output', 'smoothed\_image.png'), show=False)

tmp = tmp - ORIGINAL

image = np.stack((image,) \* 3, axis=-1)

image[np.where(tmp != 0)] = 0

save\_img(image, path=os.path.join('./output', 'filtered\_image.png'), show=False)

image[:, 0:left] = 0

image[:, right:] = 0

plt.figure()

plt.plot(ver)

plt.axhline(y=np.mean(ver), color='r', linestyle='-')

plt.axvline(x=left, color='r', linestyle='-')

plt.axvline(x=right, color='r', linestyle='-')

plt.savefig('./output/ver.png')

hor = np.sum(SMOOTH\_IMAGE, axis=1)

hor\_indexes = np.where(hor > np.mean(hor))

top = np.min(hor\_indexes)

bottom = np.max(hor\_indexes)

image[0:top, :] = 0

image[bottom:, :] = 0

plt.figure()

plt.plot(hor)

plt.axhline(y=np.mean(hor), color='r', linestyle='-')

plt.axvline(x=top, color='r', linestyle='-')

plt.axvline(x=bottom, color='r', linestyle='-')

plt.savefig('./output/hor.png')

return image

def remove\_small\_components(image, small\_component\_ratio):

cc\_properties = measure.regionprops(measure.label(image[:, :, 1], background=0),

cache=True)

avg\_area = np.mean([item.area for item in cc\_properties])

for cc in cc\_properties:

if cc.area < small\_component\_ratio \* avg\_area:

image[(cc.coords[:, 0], cc.coords[:, 1])] = 0

return image

def remove\_big\_components(image):

cc\_properties = measure.regionprops(measure.label(image[:, :, 1], background=0),

cache=True)

avg\_area = np.mean([item.area for item in cc\_properties])

for cc in cc\_properties:

if cc.area > 10 \* avg\_area:

image[(cc.coords[:, 0], cc.coords[:, 1])] = 0

return image

import logging

import time

import cv2

import numpy as np

def blow\_up\_image(image, seams):

start = time.time()

new\_image = []

ori\_height, \_, \_ = image.shape

height = ori\_height + len(seams)

seams = np.array(seams)

for i in range(0, image.shape[1]):

col = np.copy(image[:, i])

y\_cords\_seams = seams[:, i, 1]

seam\_nb = 0

for y\_seam in y\_cords\_seams:

col = np.insert(col, y\_seam + seam\_nb, [0, 0, 0], axis=0)

seam\_nb += 1

new\_image.append(col)

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return np.swapaxes(np.asarray(new\_image), 0, 1), (((100 / ori\_height) \* height) - 100) / 100

def blur\_image(img, save\_name="blur\_image.png", save=False, show=False, filter\_size=1000, horizontal=True):

kernel\_motion\_blur = np.zeros((filter\_size, filter\_size))

if horizontal:

kernel\_motion\_blur[int((filter\_size - 1) / 2), :] = np.ones(filter\_size)

kernel\_motion\_blur[:, int((filter\_size - 1) / 2)] = np.ones(filter\_size)

kernel\_motion\_blur = kernel\_motion\_blur / filter\_size

output = cv2.filter2D(img, -1, kernel\_motion\_blur)

if save:

cv2.imwrite(save\_name, output)

if show:

cv2.imshow('image', output)

cv2.waitKey(0)

cv2.destroyAllWindows()

return output

**Energy map**

import logging

import os

import sys

import time

import cv2

import numpy as np

from scipy.spatial import distance

from skimage import measure

def create\_heat\_map\_visualization(ori\_energy\_map):

start = time.time()

heatmap = ((np.copy(ori\_energy\_map) / np.max(ori\_energy\_map)))

heatmap = (np.stack((heatmap,) \* 3, axis=-1)) \* 255

heatmap = np.array(heatmap, dtype=np.uint8)

heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP\_JET)

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return heatmap

def prepare\_energy(ori\_map, left\_column, right\_column, y):

y\_value\_left, y\_value\_right = left\_column[y], right\_column[y]

ori\_map[:, 0] = sys.maxsize / 2

ori\_map[:, -1] = sys.maxsize / 2

ori\_map[y][0], ori\_map[y][-1] = y\_value\_left, y\_value\_right

return ori\_map

def create\_distance\_matrix(img\_shape, centroids, asymmetric=False, side\_length=1000):

start = time.time()

template = np.zeros((side\_length, side\_length))

center\_template = np.array([[np.ceil(side\_length / 2), np.ceil(side\_length / 2)]])

pixel\_coordinates = np.asarray([[x, y] for x in range(template.shape[0]) for y in range(template.shape[1])])

if asymmetric:

template = np.array([calculate\_asymmetric\_distance(center\_template, pxl, 1, 10)

for pxl in pixel\_coordinates]) \

.flatten().reshape((side\_length, side\_length))

else:

template = distance.cdist(center\_template, pixel\_coordinates).flatten().reshape(

(side\_length, side\_length))

distance\_matrix = np.ones(img\_shape) \* np.max(template)

centroids = np.rint(centroids).astype(int)

for centroid in centroids:

pos\_v, pos\_h = (centroid - np.ceil(side\_length / 2)).astype(int) # offset

v\_range1 = slice(max(0, pos\_v), max(min(pos\_v + template.shape[0], distance\_matrix.shape[0]), 0))

h\_range1 = slice(max(0, pos\_h), max(min(pos\_h + template.shape[1], distance\_matrix.shape[1]), 0))

v\_range2 = slice(max(0, -pos\_v), min(-pos\_v + distance\_matrix.shape[0],

template.shape[0]))

h\_range2 = slice(max(0, -pos\_h), min(-pos\_h + distance\_matrix.shape[1],

template.shape[1]))

distance\_matrix[v\_range1, h\_range1] = np.minimum(template[v\_range2, h\_range2],distance\_matrix[v\_range1, h\_range1])

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return distance\_matrix.flatten()

def create\_energy\_map(img, blurring=True, projection=True, asymmetric=False):

start = time.time()

cc, centroids, areas = find\_cc\_centroids\_areas(img)

centroids = np.asarray([[point[0], point[1]] for point in centroids])

areas = (areas - np.min(areas)) / (np.max(areas) - np.min(areas))

areas = areas - np.mean(areas)

areas = - np.abs(areas)

areas \*= 500

distance\_matrix = create\_distance\_matrix(img.shape[0:2], centroids,

asymmetric=asymmetric)

distance\_matrix /= 30

distance\_matrix += 1

energy\_background = ((np.ones(img.shape[0] \* img.shape[1]) \* 100) / distance\_matrix).transpose()

locs = np.array(np.where(img[:, :, 0].reshape(-1) == 0))[0:2, :]

energy\_text = energy\_background

energy\_text[locs] = 0

energy\_map = energy\_background + energy\_text

energy\_map = energy\_map.reshape(img.shape[0:2])

if blurring:

blurred\_energy\_map = blur\_image(img=energy\_map, filter\_size=300)

energy\_map = blurred\_energy\_map

if projection:

projection\_profile = create\_projection\_profile(energy\_map)

projection\_profile \*= np.max(energy\_map) / 2

projection\_matrix = np.zeros(img.shape[0:2])

projection\_matrix = (projection\_matrix.transpose() + projection\_profile).transpose()

projection\_matrix = blur\_image(projection\_matrix, filter\_size=1000)

energy\_map = energy\_map + projection\_matrix

if True:

filter\_size\_H = img.shape[0]

filter\_size\_V = img.shape[1]

kernel = np.zeros((filter\_size\_V, filter\_size\_H))

kernel[int(filter\_size\_V/2), :] = 1

kernel[:, int(filter\_size\_H/2)] = 1

smoothed = cv2.filter2D(energy\_map, -1, kernel)

filter\_size\_H = 32

filter\_size\_V = 32

kernel = np.ones((filter\_size\_V, filter\_size\_H)) / (filter\_size\_V\*filter\_size\_H)

smoothed = cv2.filter2D(smoothed, -1, kernel)

smoothed -= np.mean(smoothed)

smoothed[smoothed < 0] = 0

smoothed = ((smoothed - np.min(smoothed)) \* np.max(energy\_map)) /

(np.max(smoothed) - np.min(smoothed))

energy\_map = energy\_map + smoothed

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return energy\_map, cc

def create\_projection\_profile(energy\_map):

pp = np.sum(energy\_map, axis=1)

WINDOW\_SIZE = 100

pp = smooth(pp, WINDOW\_SIZE)[int(WINDOW\_SIZE/2):-int(WINDOW\_SIZE/2-1)]

pp -= np.mean(pp)

pp[pp < 0] = 0

pp = (pp - np.min(pp)) / (np.max(pp) - np.min(pp))

return pp

def smooth(x, window\_len=11, window='hanning'):

if x.ndim != 1:

raise ValueError("smooth only accepts 1 dimension arrays.")

if x.size < window\_len:

raise ValueError("Input vector needs to be bigger than window size.")

if window\_len < 3:

return x

if not window in ['flat', 'hanning', 'hamming', 'bartlett', 'blackman']:

raise ValueError("Window is on of 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'")

s = np.r\_[x[window\_len - 1:0:-1], x, x[-2:-window\_len - 1:-1]]

if window == 'flat':

w = np.ones(window\_len, 'd')

else:

w = eval('np.' + window + '(window\_len)')

y = np.convolve(w / w.sum(), s, mode='valid')

return y

def find\_cc\_centroids\_areas(img):

start = time.time()

cc\_labels, cc\_properties = get\_connected\_components(img)

amount\_of\_properties = 0

avg\_area = np.mean([item.area for item in cc\_properties])

std\_area = np.std([item.area for item in cc\_properties])

avg\_height = np.mean([item.bbox[2] - item.bbox[0] for item in cc\_properties])

avg\_width = np.mean([item.bbox[3] - item.bbox[1] for item in cc\_properties])

while amount\_of\_properties != len(cc\_properties):

amount\_of\_properties = len(cc\_properties)

image = img[:, :, 1]

coef = 1.5

for item in cc\_properties:

if item.area > coef \* avg\_area \

or item.bbox[2] - item.bbox[0] > coef \* avg\_height \

or item.bbox[3] - item.bbox[1] > coef \* avg\_width:

v\_size = abs(item.bbox[0] - item.bbox[2])

h\_size = abs(item.bbox[1] - item.bbox[3])

y1, x1, y2, x2 = item.bbox

if float(h\_size) / v\_size > 1.5:

image[y1:y2, np.round((x1 + x2) / 2).astype(int)] = 0

elif float(v\_size) / h\_size > 1.5:

image[np.round((y1 + y2) / 2).astype(int), x1:x2] = 0

else:

image[y1:y2, np.round((x1 + x2) / 2).astype(int)] = 0

image[np.round((y1 + y2) / 2).astype(int), x1:x2] = 0

img[:, :, 1] = image

cc\_labels, cc\_properties = get\_connected\_components(img)

all\_centroids = np.asarray([cc.centroid[0:2] for cc in cc\_properties])

all\_areas = np.asarray([cc.area for cc in cc\_properties])

no\_outliers = detect\_outliers(all\_areas, avg\_area, std\_area)

centroids = all\_centroids[no\_outliers, :]

filtered\_area = all\_areas[no\_outliers]

all\_areas = filtered\_area[np.argsort(centroids[:, 0])]

all\_centroids = centroids[np.argsort(centroids[:, 0]), :]

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return (cc\_labels, cc\_properties), all\_centroids, all\_areas

def get\_connected\_components(img):

cc\_labels = measure.label(img[:, :, 1], background=0)

cc\_properties = measure.regionprops(cc\_labels, cache=True)

return cc\_labels, cc\_properties

def detect\_outliers(area, mean, std):

start = time.time()

if mean is not None:

mean = np.mean(area)

if std is not None:

std = np.std(area)

no\_outliers = area - 0.25\*mean > 0

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return no\_outliers

**Seam Carving**

import itertools

import sys

import cv2

import numba

import numpy as np

@numba.jit()

def horizontal\_seam(energies, penalty\_reduction, bidirectional=False):

height, width = energies.shape[:2]

ori\_y = 0

previous = 0

seam\_forward = []

seam\_backward = []

for i in range(0, width, 1):

col = energies[:, i]

if i == 0:

ori\_y = previous = np.argmin(col)

else:

top = col[previous - 1] if previous - 1 >= 0 else sys.maxsize

middle = col[previous]

bottom = col[previous + 1] if previous + 1 < height else sys.maxsize

if penalty\_reduction > 0:

top += ((ori\_y - (previous - 1)) \*\* 2) / penalty\_reduction

middle += ((ori\_y - previous) \*\* 2) / penalty\_reduction

bottom += + ((ori\_y - (previous + 1)) \*\* 2) / penalty\_reduction

previous = previous + np.argmin([top, middle, bottom]) - 1

seam\_forward.append([i, previous])

if bidirectional:

for i in range(width-1, -1, -1):

col = energies[:, i]

if i == width-1:

ori\_y = previous = np.argmin(col)

else:

top = col[previous - 1] if previous - 1 >= 0 else sys.maxsize

middle = col[previous]

bottom = col[previous + 1] if previous + 1 < height else sys.maxsize

if penalty\_reduction > 0:

top += ((ori\_y - (previous - 1)) \*\* 2) / penalty\_reduction

middle += ((ori\_y - previous) \*\* 2) / penalty\_reduction

bottom += + ((ori\_y - (previous + 1)) \*\* 2) / penalty\_reduction

previous = previous + np.argmin([top, middle, bottom]) - 1

seam\_backward.append([i, previous])

return [seam\_forward, seam\_backward[::-1]]

def draw\_seams(img, seams, bidirectional=True):

x\_axis = np.expand\_dims(np.array(range(0, len(seams[0]))), -1)

seams = [np.concatenate((x, np.expand\_dims(seam, -1)), axis=1) for seam, x in zip(seams, itertools.repeat(x\_axis))]

for i, seam in enumerate(seams):

if bidirectional and i % 2 == 0:

cv2.polylines(img, np.int32([seam]), False, (0, 0, 0), 3)

else:

cv2.polylines(img, np.int32([seam]), False, (255, 255, 255), 3)

def draw\_seams\_red(img, seams, bidirectional=True):

x\_axis = np.expand\_dims(np.array(range(0, len(seams[0]))), -1)

seams = [np.concatenate((x, np.expand\_dims(seam, -1)), axis=1) for seam, x in zip(seams, itertools.repeat(x\_axis))]

for i, seam in enumerate(seams):

cv2.polylines(img, np.int32([seam]), False, (0, 0, 255), 3)

def get\_seams(ori\_energy\_map, penalty\_reduction, seam\_every\_x\_pxl):

seams = []

left\_column\_energy\_map = np.copy(ori\_energy\_map[:, 0])

right\_column\_energy\_map = np.copy(ori\_energy\_map[:, -1])

for seam\_at in range(0, ori\_energy\_map.shape[0], seam\_every\_x\_pxl):

energy\_map = prepare\_energy(ori\_energy\_map,left\_column\_energy\_map, right\_column\_energy\_map, seam\_at)

seams.extend(horizontal\_seam(energy\_map, penalty\_reduction=penalty\_reduction, bidirectional=True))

seams = np.array([np.array(s)[:, 1] for s in seams])

return seams

def post\_process\_seams(energy\_map, seams):

assert energy\_map.shape[1] == len(seams[0])

SAFETY\_STOP = 100

iteration = 0

repeat = True

while repeat:

iteration += 1

if iteration >= SAFETY\_STOP:

break

repeat = False

for index, seam\_A in enumerate(seams):

for seam\_B in seams[index:]:

overlap = seam\_A - seam\_B

overlap[abs(overlap) < 10] = 0

seam\_A[overlap == 0] = seam\_B[overlap == 0]

sequences = non\_zero\_runs(overlap)

if len(sequences) > 0:

for i, sequence in enumerate(sequences):

target = sequence[1] - sequence[0]

left = sequence[0] - sequences[i - 1, 1] if i > 0 else sequence[0]

right = sequences[i + 1, 0] - sequence[1] if i < len(sequences)-1

else

energy\_map.shape[1] - sequence[1]

if target > left and target > right:

continue

repeat = True

sequence = range(\*sequence)

energy\_A = measure\_energy(energy\_map, seam\_A, sequence)

energy\_B = measure\_energy(energy\_map, seam\_B, sequence)

if energy\_A > energy\_B:

seam\_A[sequence] = seam\_B[sequence]

else:

seam\_B[sequence] = seam\_A[sequence]

return seams

def non\_zero\_runs(a):

iszero = np.concatenate(([1], np.equal(a, 0).view(np.int8), [1]))

absdiff = np.abs(np.diff(iszero))

ranges = np.where(absdiff == 1)[0].reshape(-1, 2)

return ranges

def measure\_energy(energy\_map, seam, sequence):

return energy\_map[seam[sequence], sequence].sum()

**Binning algorithm**

import itertools

import logging

import os

import time

import cv2

import numpy as np

def majority\_voting(connected\_components, seams):

start = time.time()

centroids = np.asarray([cc.centroid[0:2] for cc in connected\_components[1]])

centroids = centroids[np.argsort(centroids[:, 0]), :]

values = count\_seams\_below(centroids, seams)

small\_bins = [42]

while len(small\_bins) > 0:

bin\_index, bin\_size, unique\_bins = split\_into\_bins\_and\_index(values)

if small\_bins[0] == 42:

avg = np.mean(bin\_size[bin\_size>1])\*0.25

small\_bins = unique\_bins[np.where(bin\_size < avg)

merge\_small\_bins(bin\_index, centroids, small\_bins, values)

lines = []

for bin in unique\_bins:

lines.append(list(centroids[np.where(bin\_index == bin)]))

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return lines, centroids, values

def count\_seams\_below(centroids, seams):

values = np.zeros([len(centroids)])

for i, centroid in enumerate(centroids):

cx = int(centroid[1])

cy = int(centroid[0])

for seam in seams:

if seam[cx] > cy:

values[i] = values[i] + 1

return values

def merge\_small\_bins(bin\_index, centroids, small\_bins, values):

for bin in small\_bins:

for loc in np.where(bin\_index == bin)[0]:

loc\_p = loc + 1 if loc + 1 < len(values) else loc

while bin\_index[loc\_p] == bin\_index[loc]:

if loc\_p + 1 < len(values):

loc\_p += 1

else:

break

loc\_m = loc - 1 if loc > 0 else loc

while bin\_index[loc\_m] == bin\_index[loc]:

if loc\_m > 0:

loc\_m -= 1

else:

break

XA = np.expand\_dims(centroids[loc], axis=0)

upper = np.array([calculate\_asymmetric\_distance(XA, c, 1, 5) for c in

centroids[np.where(bin\_index == bin\_index[loc\_p])]]).min()

lower = np.array([calculate\_asymmetric\_distance(XA, c, 1, 5) for c in

centroids[np.where(bin\_index == bin\_index[loc\_m])]]).min()

values[loc] = values[loc\_m] if (upper == 0 or upper > lower) and lower != 0 else values[loc\_p]

def split\_into\_bins\_and\_index(values):

bin\_index = np.digitize(values, np.unique(values))

unique\_bins, bin\_size = np.unique(bin\_index, return\_counts=True)

return bin\_index, bin\_size, unique\_bins

def compute\_avg\_pairwise\_distance(centroids):

centroids = centroids[np.argsort(centroids[:, 1]), :]

dist = []

for c1, c2 in pairwise(centroids):

dist.append(c2[1] - c1[1])

return dist

def check\_for\_anomaly(centroids, threshold):

centroids = centroids[np.argsort(centroids[:, 1]), :]

for c1, c2 in pairwise(centroids):

if c2[1] - c1[1] > threshold:

return True

return False

def pairwise(iterable):

a, b = itertools.tee(iterable)

next(b, None)

return zip(a, b)

def draw\_bins(img, centroids, root\_output\_path, seams, bins):

binning\_img = np.zeros(img.shape[0:2], dtype=np.uint8)

binning\_img.fill(255)

locs = np.array(np.where(img[:, :, 0].reshape(-1) != 0))[0:2, :]

binning\_img = binning\_img.flatten()

binning\_img[locs] = 211

binning\_img = binning\_img.reshape(img.shape[0:2])

binning\_img = np.stack((binning\_img,) \* 3, axis=-1)

draw\_seams\_red(binning\_img, seams)

overlay\_img = binning\_img.copy()

for centroid, value in zip(centroids, bins):

cv2.circle(overlay\_img, (int(centroid[1]), int(centroid[0])), 25, (0, 255, 0), -1)

cv2.putText(binning\_img, str(int(value)), (int(centroid[1]) - 16, int(centroid[0]) + 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (0, 0, 0), 2, cv2.LINE\_AA)

cv2.addWeighted(overlay\_img, 0.4, binning\_img, 0.6, 0, binning\_img)

save\_img(binning\_img, path=os.path.join(root\_output\_path, 'energy\_map',

'energy\_map\_bin\_expl.png'))

**Polygon Manager**

import logging

import time

import cv2

import networkx as nx

import numpy as np

from skimage import measure

def get\_polygons\_from\_lines(img, lines, connected\_components, vertical):

polygon\_coords = []

for i, line in enumerate(lines):

cc\_coords = []

graph\_nodes = []

polygon\_img = np.zeros(img.shape)

for c in line:

cc = find\_cc\_from\_centroid(c, connected\_components[1])

points = cc.coords[::3, 0:2]

points = np.asarray([[point[1], point[0]] for point in points])

cc\_coords.append(points)

graph\_nodes.append(find\_graph\_node(cc.coords, cc.centroid))

overlay\_graph = createTINgraph(np.array(list(set(tuple(p) for p in graph\_nodes))))

overlay\_graph = nx.minimum\_spanning\_tree(overlay\_graph)

polygon\_img = print\_graph\_on\_img(polygon\_img, [overlay\_graph], color=(255, 255, 255), thickness=1)

cv2.fillPoly(polygon\_img, cc\_coords, color=(255, 255, 255))

if vertical:

polygon\_img = cv2.rotate(polygon\_img, cv2.ROTATE\_90\_CLOCKWISE)

filter\_size\_H = 5

filter\_size\_V = 5

kernel = np.ones((filter\_size\_V, filter\_size\_H)) / filter\_size\_H

polygon\_img = cv2.filter2D(polygon\_img, -1, kernel)

polygon\_coords.append(measure.find\_contours(polygon\_img[:, :, 0], 5, fully\_connected='high')[0])

return polygon\_coords

def find\_graph\_node(coords, centroid):

centroid = np.asarray(centroid, dtype=int)

if centroid in coords:

return centroid

return [coords[0][0], coords[0][1]]

def find\_cc\_from\_centroid(c, cc\_properties):

for cc in cc\_properties:

if cc.centroid[0] == c[0] and cc.centroid[1] == c[1]:

return cc

print("If this is printed, you might want to uncomment the line swapping the coordinates!")

return None

def draw\_polygons(image, polygons, vertical):

if vertical:

image = cv2.rotate(image, cv2.ROTATE\_90\_CLOCKWISE)

for polygon in polygons:

cv2.polylines(image, np.array([[[np.int(p[1]), np.int(p[0])] for p in polygon]]), 1,

color=(248, 24, 148), thickness=3)

return image

def polygon\_to\_string(polygons):

start = time.time()

strings = []

for polygon in polygons:

line\_string = []

for i, point in enumerate(polygon):

if i % 3 != 0:

continue

line\_string.append("{},{}".format(int(point[1]), int(point[0])))

strings.append(' '.join(line\_string))

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return strings

**Utils**

**Util XML:**

import logging

import time

import cv2

import networkx as nx

import numpy as np

from skimage import measure

def get\_polygons\_from\_lines(img, lines, connected\_components, vertical):

polygon\_coords = []

for i, line in enumerate(lines):

cc\_coords = []

graph\_nodes = []

polygon\_img = np.zeros(img.shape)

for c in line:

cc = find\_cc\_from\_centroid(c, connected\_components[1])

points = cc.coords[::3, 0:2]

points = np.asarray([[point[1], point[0]] for point in points])

cc\_coords.append(points)

graph\_nodes.append(find\_graph\_node(cc.coords, cc.centroid))

overlay\_graph = createTINgraph(np.array(list(set(tuple(p) for p in graph\_nodes))))

overlay\_graph = nx.minimum\_spanning\_tree(overlay\_graph)

polygon\_img = print\_graph\_on\_img(polygon\_img, [overlay\_graph], color=(255, 255, 255), thickness=1)

cv2.fillPoly(polygon\_img, cc\_coords, color=(255, 255, 255))

if vertical:

polygon\_img = cv2.rotate(polygon\_img, cv2.ROTATE\_90\_CLOCKWISE)

filter\_size\_H = 5

filter\_size\_V = 5

kernel = np.ones((filter\_size\_V, filter\_size\_H)) / filter\_size\_H

polygon\_img = cv2.filter2D(polygon\_img, -1, kernel)

polygon\_coords.append(measure.find\_contours(polygon\_img[:, :, 0], 5, fully\_connected='high')[0])

return polygon\_coords

def find\_graph\_node(coords, centroid):

centroid = np.asarray(centroid, dtype=int)

if centroid in coords:

return centroid

return [coords[0][0], coords[0][1]]

def find\_cc\_from\_centroid(c, cc\_properties):

for cc in cc\_properties:

if cc.centroid[0] == c[0] and cc.centroid[1] == c[1]:

return cc

print("If this is printed, you might want to uncomment the line swapping the coordinates!")

return None

def draw\_polygons(image, polygons, vertical):

if vertical:

image = cv2.rotate(image, cv2.ROTATE\_90\_CLOCKWISE)

for polygon in polygons:

cv2.polylines(image, np.array([[[np.int(p[1]), np.int(p[0])] for p in polygon]]), 1,

color=(248, 24, 148), thickness=3)

return image

def polygon\_to\_string(polygons):

start = time.time()

strings = []

for polygon in polygons:

line\_string = []

for i, point in enumerate(polygon):

if i % 3 != 0:

continue

line\_string.append("{},{}".format(int(point[1]), int(point[0])))

strings.append(' '.join(line\_string))

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return strings

import shutil

import cv2

import os

import numpy as np

def create\_folder\_structure(input\_file, output\_path, params):

fileName = os.path.basename(input\_file).split('.')[0]

if not os.path.exists(output\_path):

os.mkdir(output\_path)

basefolder\_path = os.path.join(output\_path, fileName + '\_penalty\_reduction\_{}\_seams\_{}\_component\_ratio\_{}'.format(\*params))

if not os.path.exists(basefolder\_path):

os.mkdir(basefolder\_path)

os.mkdir(os.path.join(basefolder\_path, 'energy\_map'))

os.mkdir(os.path.join(basefolder\_path, 'logs'))

os.mkdir(os.path.join(basefolder\_path, 'preprocess'))

return basefolder\_path

def save\_img(img, path='experiment.png', show=False):

if show:

cv2.imshow('img', img)

cv2.waitKey(0)

cv2.destroyAllWindows()

cv2.imwrite(path, img)

def calculate\_asymmetric\_distance(x, y, h\_weight=1, v\_weight=5):

return [np.sqrt((((y[0] - x[0][0]) \*\* 2) \* v\_weight + ((y[1] - x[0][1]) \*\* 2) \* h\_weight)/ (h\_weight+v\_weight))]

def dict\_to\_string(dictionay):

string = []

for entry in dictionay.items():

string.append('\_'.join(entry))

return '\_'.join(string)

**Util-graph:**

import bisect

import logging

import os

import time

import networkx as nx

import numpy as np

import cv2

from scipy.spatial import Delaunay

from shapely.geometry import LineString

def createTINgraph(points):

start = time.time()

TIN = Delaunay(points)

edges = set()

for n in range(TIN.nsimplex):

edge = sorted([TIN.vertices[n, 0], TIN.vertices[n, 1]])

edges.add((edge[0], edge[1], asymetric\_distance(edge, points)))

edge = sorted([TIN.vertices[n, 0], TIN.vertices[n, 2]])

edges.add((edge[0], edge[1], asymetric\_distance(edge, points)))

edge = sorted([TIN.vertices[n, 1], TIN.vertices[n, 2]]

edges.add((edge[0], edge[1], asymetric\_distance(edge, points)))

graph = nx.Graph()

graph.add\_weighted\_edges\_from(edges)

original\_nodes = points

assert len(original\_nodes) == graph.number\_of\_nodes()

attributes = {}

for n in range(len(original\_nodes)):

XY = original\_nodes[n]

attributes[n] = [XY[1], XY[0]]

nx.set\_node\_attributes(graph, attributes, 'XY')

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return graph

def asymetric\_distance(edge, points):

return np.linalg.norm(

np.asarray(points[edge[0]]) \* np.array([1, 3]) - np.asarray(points[edge[1]]) \*

np.array([1, 3]))

def print\_graph\_on\_img(img, graphs, color=(0, 255, 0), thickness=3):

img = img.copy()

for graph in graphs:

for edge in graph.edges:

p1, p2 = get\_edge\_node\_coordinates(edge, graph)

cv2.line(img, p1, p2, color, thickness=thickness)

return img

def get\_edge\_node\_coordinates(edge, graph):

node\_attributes = nx.get\_node\_attributes(graph, 'XY')

p1 = np.asarray(node\_attributes[edge[0]], dtype=np.uint32)

p1 = (p1[0], p1[1])

p2 = np.asarray(node\_attributes[edge[1]], dtype=np.uint32)

p2 = (p2[0], p2[1])

return p1, p2

def cut\_graph\_with\_seams(graph, seams, too\_small\_pc):

start = time.time()

tic = time.time()

unique\_edges, weights, occurrences = find\_intersected\_edges(graph, seams)

logging.info("find\_intersected\_edges: {}".format(time.time()-tic))

graph.remove\_edges\_from(unique\_edges)

if nx.is\_connected(graph):

return list([graph])

graphs = np.asarray(list(nx.connected\_component\_subgraphs(graph)))

small\_graphs = detect\_small\_graphs(graphs, too\_small\_pc).tolist()

while small\_graphs:

graph = merge\_small\_graphs(graph, list(small\_graphs), unique\_edges, weights)

graphs = np.asarray(list(nx.connected\_component\_subgraphs(graph)))

small\_graphs = detect\_small\_graphs(graphs, too\_small\_pc)

if nx.is\_connected(graph):

return list([graph])

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return graphs

def get\_neighbouring\_seams\_index(seams\_max\_y, seams\_min\_y, edge\_max\_y, edge\_min\_y):

return [bisect.bisect\_left(seams\_max\_y, edge\_min\_y), bisect.bisect\_left(seams\_min\_y, edge\_max\_y)]

def chunks(l, n):

n = np.min([n, len(l)])

size = int(np.ceil(len(l)/n))

for i in range(0, len(l), size):

yield l[i:i + size]

def find\_intersected\_edges(graph, seams):

seams\_y = [np.array(s)[:, 1] for s in seams]

seams\_max\_y = np.max(seams\_y, axis=1)

seams\_min\_y = np.min(seams\_y, axis=1)

node\_attributes = nx.get\_node\_attributes(graph, 'XY')

seams = [LineString(seam) for seam in seams]

edges = [e for e in graph.edges]

edges.sort(key=lambda tup: tup[0])

edges\_to\_remove = []

for chunk in chunks(edges, 250):

tmp = np.array(chunk)

p1 = np.array([node\_attributes[edge[0]] for edge in tmp])

p2 = np.array([node\_attributes[edge[1]] for edge in tmp])

edge\_max\_y = np.max((p1[:, 1], p2[:, 1]))

edge\_min\_y = np.min((p1[:, 1], p2[:, 1]))

index = get\_neighbouring\_seams\_index(seams\_max\_y, seams\_min\_y, edge\_max\_y, edge\_min\_y)

for start, end, edge in zip(p1, p2, chunk):

line\_edge = LineString([start, end])

for seam in seams[index[0]:index[1]]:

if line\_edge.intersects(seam):

edges\_to\_remove.append(edge)

break

unique\_edges, occurrences = np.unique(np.array(edges\_to\_remove), return\_counts=True,

axis=0)

weights = [graph.edges[edge]['weight'] for edge in unique\_edges]

return unique\_edges, weights, occurrences

def merge\_small\_graphs(graph, small\_graphs, unique\_edges, weights):

edges\_to\_add = []

weights = np.asarray(weights)

while small\_graphs:

small\_graph = small\_graphs.pop()

edge\_idxs = np.unique(np.hstack(np.asarray(

[np.where(unique\_edges == node)[0] for node in list(small\_graph.nodes)])))

min\_edge\_idx = edge\_idxs[np.argmin(weights[edge\_idxs])]

edge = unique\_edges[min\_edge\_idx]

unique\_edges = np.delete(unique\_edges, min\_edge\_idx, axis=0)

weights = np.delete(weights, min\_edge\_idx, axis=0)

edges\_to\_add.append((edge[0], edge[1], weights[min\_edge\_idx]))

graph.add\_weighted\_edges\_from(edges\_to\_add)

return graph

def graph\_to\_point\_lists(graphs):

return [list(nx.get\_node\_attributes(graph, 'XY').values()) for graph in graphs]

def detect\_small\_graphs(graphs, too\_small\_pc):

start = time.time()

graph\_sizes = np.asarray([len(g.nodes) for g in graphs])

too\_small = graph\_sizes < too\_small\_pc \* np.mean(graph\_sizes)

stop = time.time()

logging.info("finished after: {diff} s".format(diff=stop - start))

return graphs[too\_small]

**util-graph\_logger:**

import cv2

import os

import numpy as np

class GraphLogger:

IMG\_SHAPE = ()

ROOT\_OUTPUT\_PATH = ''

@classmethod

def draw\_graphs(cls, img, graphs, color=(0, 255, 0), thickness=3, name='graph.png'):

if not list(img):

img = np.zeros(cls.IMG\_SHAPE)

else:

img = img.copy()

for graph in graphs:

img = cls.draw\_graph(img, graph, color, thickness, False)

save\_img(img, path=os.path.join(cls.ROOT\_OUTPUT\_PATH, 'graph', name),

show=False)

return img

@classmethod

def draw\_graph(cls, img, graph, color=(0, 255, 0), thickness=3, save=False,

name='graph.png'):

if not list(img):

img = np.zeros(cls.IMG\_SHAPE)

else:

img = img.copy()

cls.draw\_edges(img, graph.edges, graph, color, thickness, save=False)

if save:

save\_img(img, path=os.path.join(cls.ROOT\_OUTPUT\_PATH, 'graph', name),

show=False)

return img

@classmethod

def draw\_edges(cls, img, edges, graph, color, thickness, save=False, name='graph.png'):

for edge in edges:

p1, p2 = get\_edge\_node\_coordinates(edge, graph)

cv2.line(img, p1, p2, color, thickness=thickness)

if save:

save\_img(img, path=os.path.join(cls.ROOT\_OUTPUT\_PATH, 'graph', name),

show=False)

**Line Segmentation**

import cv2

import os

import numpy as np

class GraphLogger:

IMG\_SHAPE = ()

ROOT\_OUTPUT\_PATH = ''

@classmethod

def draw\_graphs(cls, img, graphs, color=(0, 255, 0), thickness=3, name='graph.png'):

if not list(img):

img = np.zeros(cls.IMG\_SHAPE)

else:

img = img.copy()

for graph in graphs:

img = cls.draw\_graph(img, graph, color, thickness, False)

save\_img(img, path=os.path.join(cls.ROOT\_OUTPUT\_PATH, 'graph', name)

, show=False)

return img

@classmethod

def draw\_graph(cls, img, graph, color=(0, 255, 0), thickness=3, save=False,

name='graph.png'):

if not list(img):

img = np.zeros(cls.IMG\_SHAPE)

else:

img = img.copy()

cls.draw\_edges(img, graph.edges, graph, color, thickness, save=False)

if save:

save\_img(img, path=os.path.join(cls.ROOT\_OUTPUT\_PATH, 'graph', name),

show=False)

return img

@classmethod

def draw\_edges(cls, img, edges, graph, color, thickness, save=False, name='graph.png'):

for edge in edges:

p1, p2 = get\_edge\_node\_coordinates(edge, graph)

cv2.line(img, p1, p2, color, thickness=thickness)

if save:

save\_img(img, path=os.path.join(cls.ROOT\_OUTPUT\_PATH, 'graph', name),

show=False)

extract\_textline(input\_path='test1.png', output\_path= './output', penalty\_reduction = 6000, sseam\_every\_x\_pxl= 100,testing=True, vertical=False, console\_log= False, small\_component\_

ratio= 0.1)

logging.info('Terminated')

# 

# CHAPTER-6

# SYSTEM TESTING

## 6.1 INTRODUCTION

The cause of testing is to detect mistakes. Making an attempt out is the technique of looking for to

realize each viable fault or weakness in a piece product. It presents a method to determine the

performance of add-ons, sub-assemblies, assemblies and/or a completed product. It is the

method of ex excising g program with the intent of constructing certain that the application

procedure meets its necessities and client expectations and does no longer fail in an unacceptable

process. There are rather plenty of forms of scan. Each experiment sort addresses a special

trying out requirement.

## 6.2 TYPES OF TESTS:

**Unit testing:**

Unit testing is undertaken when a module has been created and successfully reviewed. In

order to test a single module we need to provide a complete environment i.e. besides the

module we would require

• The procedures belonging to other modules that the module under test calls.

• Non-local data structures that module accesses.

• A procedure to call the functions of the module under test with appropriate parameters

In this testing individual components of the project status detecting, sound alerting,

automatic mail sub functions are verified successfully.

**Integration testing:**

The documents created using the main code should be sent by mailing module correctly.

If not reports of the documents won’t reach the agency correctly.

**Functional testing:**

Testing on the functionality of the driver monitoring system for travel agencies include whether

system correctly detecting the state of the driver correctly or not and mail is sent to correct

destination or not and sound alert testing. Functionality testing done successfully.

**System Testing:**

Testing of whole system is done after the completion all the required sub functions like automatic mail

sending and sound alert. System testing is completed successfully.

**White Box Testing:**

This testing is a trying out wherein where the application tester has competencies of the interior

workings, constitution and software language, or at least its cause. It's rationale. It's used to test

areas that can't be reached from a black box stage.

**Black Box Testing:**

This is testing the software with none advantage of the inside workings, establishment or words

of the unit life form veteran.

**Acceptance Testing:**

User Acceptance testing trying out is a crucial section of any mission and requires enormous

participation by the tip user. It additionally ensures that the procedure meets the functional

specifications.

**Test Results:**

The entire test cases recounted above passed effectually. No defects Encountered.

# Conclusion

This project “TEXT LINE SEGMENTATION FOR MEDIEVAL MANUSCRIPTS” has been successfully executed by our team to produce segmented lines of handwritten text for our dataset and it is found that our proposed system provides efficient and effective result when compared to existing system. Our project provides text with polygons which specifies the segmentation of handwritten text.

In conclusion, text line segmentation for medieval manuscripts is a critical task that plays a vital role in the analysis, digitization, and preservation of these valuable historical documents.

# Appendices

# References:

* Image and Text Segmentation pipeline for the paper **"Labeling, Cutting, Grouping : an**

**Efficient Text Line Segmentation Method for Medieval Manuscripts",** published at the

15th IAPR International Conference on Document Analysis and Recognition (ICDAR) in

2019.

* Research Gate – Text line segmentation for handwritten documents.
* IEEE Papers for Text Line Segmentation.