

# Department of Computing

# CS370: Artificial Intelligence

# Name: M. Hasnain Naeem

# Reg#: 212728

# Class: BSCS-7B

Lab 2: Offline Signature Verification

**Date: January 30th, 2019**

# Time: 10am-01pm & 2pm-5pm

# Instructor: Dr. Dr Imran Malik

**Introduction**

The purpose of this lab is to get familiar with offline signature verification and perform initial verification routines by extracting basic features.

**Objectives**

After completing this lab, students will be able to understand how to:

* Process images in Python
* Extract features from signatures

# Software Tools/Requirements

* Solutions should be made in Python
* Use PIL or OpenCV

# Prerequisites

Before you begin working on this lab, make sure that you have:

* Set up Python on your machine
* Installed either Python Imaging Library (PIL) or OpenCV.
* Downloaded the 4NSigComp2010 dataset from [http://www.iaprtc11.org/mediawiki/index.php/ICFHR\_2010\_Signature\_Verification\_Compet ition\_(4NSigComp2010)](http://www.iaprtc11.org/mediawiki/index.php/ICFHR_2010_Signature_Verification_Competition_(4NSigComp2010))

# Lab Task

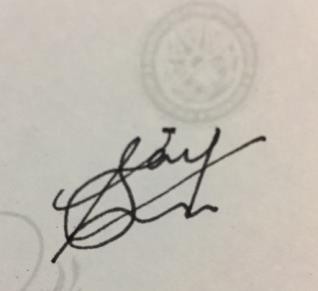
Take one Ref signature from downloaded dataset1 and perform the following tasks:

1. Develop a bounding box around the signature content.
2. Find out the centroid of the signature.
3. Segment signature from centroid vertically and horizontally (the signature will be divided into four pieces)
4. Calculate black to white transitions for each of the four segments.

1 In case you are unable to download the dataset, you can use any image of a signature to perform this lab.

**Description**

In this section, we will walk through the lab tasks and see how to complete them. This walkthrough is to provide you a starting point for your own implementations, and hence is abstract and leaves out many implementation details. You are supposed to figure them out yourself. We will use the following signature as a reference.



Preprocessing

Convert the signature to a binary (black-and-white) image before extracting features. With PIL, this can be done by using the Image.convert() method. For further information on how to use PIL, refer to PIL handbook at [http://www.pythonware.com/media/data/pil-handbook.pdf.](http://www.pythonware.com/media/data/pil-handbook.pdf) At this point, your signature image should look like this:



### Task 1: Developing a bounding box

In this task, you have to locate the signature in the image and develop a bounding box around the signature content only, ignoring the white space around it.

**Algorithm** left := width right := 0 top := height bottom := 0

for x in (0, width) and y in (0, height) color := image.getpixel(x, y)

if color is 0

if x > right

right := x if x < left

left := x if y > bottom

bottom := y if y < top

top := y

After completing this task, your signature image should look like this:



### Task 2: Locating the centroid

Centroid is the point where center of mass of the signature image is located. It can be computed using the algorithm given below.

#### Algorithm

cx := 0

cy := 0

n := 0

for x in (0, width) and y in (0, height): if image.color(x, y) is 0:

cx := cx + x cy := cy + y n := n + 1

cx := cx / n cy := cy / n

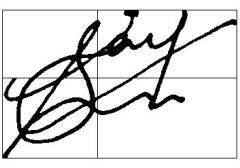
### Task 3: Dividing the image at centroid to create four segments

You should have the following at this point:

1. Bounding box = (left, right, top, bottom)
2. Centroid = (cx, cy)

You now have to divide the image into four segments. The four segments can be computed by locating the boundaries of each of the four segments:

1. (left, cx, top, cy) are the boundaries of top-left segment
2. (cx, right, top, cy) are the boundaries of top-right segment
3. (left, cx, cy, bottom) are the boundaries of bottom-left segment
4. (cx, right, cy, bottom) are the boundaries of bottom-right segment After completing this task, your signature image should look like this:



### Task 4: Finding black to white transitions

In this lab task, you have to find black to white transitions for each of the four segments/ blocks of the signature you made in Task3. That is, you have to calculate the number of white pixels in the neighborhood (use 8-connectivity) of each black pixel in the image. For details about pixel connectivity, refer to your famous friend Image result for google logo

#### Algorithm

prev := image.color(0,0) n := 0

for x in (1, width) and y in (1, height): curr = image.color(x,y)

if curr is 255 and prev is 0 n := n + 1

prev := curr

### Completion check

On successful completion of the lab tasks, you should have the following:

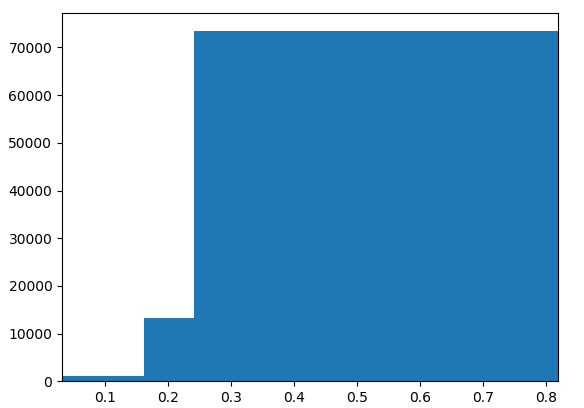
1. Bounding box of the signature in image
   1. B = (left, right, top, bottom)
2. Coordinates of centroid of the signature
   1. C = (cx, cy)
3. Four values of black to white transitions, for each of the four segments
   1. T = (TL, TR, BL, BR)

### Code

|  |
| --- |
| *""" AI - Lab 2 - Offline Signature Verification Written by: M. Hasnain Naeem """  # imports from* PIL *import* Image *import* os *import* numpy *as* np *import* cv2 *import* matplotlib.pyplot *as* plt   *# functions def* show\_image(img):  *# show image using OpenCV; extra lines are for jupyter notebook* cv2.imshow("Gray Image", img)  cv2.waitKey(0)  cv2.destroyAllWindows()   *def* bounded\_box(bin\_img):  width = bin\_img.shape[1] - 1  height = bin\_img.shape[0] - 1   left = width  right = 0  top = height  bottom = 0   *for* x *in* range(0, width):  *for* y *in* range(0, height):  color = bin\_img[y, x]  *if* color == 0:  *if* x > right:  right = x  *if* x < left:  left = x  *if* y > bottom:  bottom = y  *if* y < top:  top = y   bounded\_box\_dims = (left, right, top, bottom)  bounded\_img = bin\_img[top: bottom, left:right]  *return* (bounded\_img, bounded\_box\_dims)   *def* draw\_bounded\_box(img, bounded\_box\_dims):  img\_with\_box = img  *for* i *in* range(bounded\_box\_dims[0], bounded\_box\_dims[1] + 1):  img\_with\_box[bounded\_box\_dims[2]][i] = 0  img\_with\_box[bounded\_box\_dims[3]][i] = 0   *for* i *in* range(bounded\_box\_dims[2], bounded\_box\_dims[3] + 1):  img\_with\_box[i][bounded\_box\_dims[0]] = 0  img\_with\_box[i][bounded\_box\_dims[1]] = 0  *return* img\_with\_box   *def* centroid(bin\_img):  width = bin\_img.shape[1] - 1  height = bin\_img.shape[0] - 1  cx = 0  cy = 0  n = 0  *for* x *in* range(0, width):  *for* y *in* range(0, height):  color = bin\_img[y, x]  *if* color == 0:  cx = cx + x  cy = cy + y  n = n + 1  cx = cx // n  cy = cy // n  centroid = (cx, cy)  *return* centroid   *def* segments(bin\_img, centroid\_dims, bounded\_box\_dims):  *"""  Get 4 segments from a image.   Parameters:   big\_img: binary image with 1 color channel  centroid: centroid tuple  bounded\_box: bounding box tuple with arrangement: (left, right, top, bottom)   returns:   segment\_imgs: list of segments  coordinates: list of coordinates of each segment  """* (left, right, top, bottom) = bounded\_box\_dims  (cx, cy) = centroid\_dims  *# coordinates of image segment* top\_left\_coords = ((top, cy), (left, cx))  *# image segment* top\_left\_seg = bin\_img[top:cy + 1, left:cx + 1]  *# similarly for other segments:* top\_right\_coords = ((top, cy), (cx, right))  top\_right\_seg = bin\_img[top:cy + 1, cx:right + 1]   bottom\_left\_coords = ((cy, bottom), (left, cx))  bottom\_left\_seg = bin\_img[cy:bottom + 1, left:cx + 1]   bottom\_right\_coords = ((cy, bottom), (cx, right))  bottom\_right\_seg = bin\_img[cy:bottom + 1, cx:right + 1]   *# list of segments* segment\_imgs = (top\_left\_seg, top\_right\_seg, bottom\_left\_seg, bottom\_right\_seg)  *# list of coordinates* coordinates = (top\_left\_coords, top\_right\_coords, bottom\_left\_coords, bottom\_right\_coords)   *return* segment\_imgs, coordinates   *def* draw\_segment\_lines(img, segment\_coordinates):  (top\_left\_coords, top\_right\_coords, bottom\_left\_coords, bottom\_right\_coords) = segment\_coordinates   img\_with\_lines = img  *for* i *in* range(top\_left\_coords[0][0], top\_left\_coords[0][1]):  img\_with\_lines[i][top\_left\_coords[1][1]] = 0   *for* i *in* range(bottom\_left\_coords[0][0], bottom\_left\_coords[0][1]):  img\_with\_lines[i][bottom\_left\_coords[1][1]] = 0   *for* i *in* range(top\_left\_coords[1][0], top\_left\_coords[1][1]):  img\_with\_lines[top\_left\_coords[0][1]][i] = 0   *for* i *in* range(bottom\_right\_coords[1][0], bottom\_right\_coords[1][1]):  img\_with\_lines[bottom\_right\_coords[0][0]][i] = 0   *return* img\_with\_lines   *def* black\_to\_white\_trans(img):  height = img.shape[0]  width = img.shape[1]   prev = img[0, 0]  n = 0  *for* x *in* range(1, width):  *for* y *in* range(1, height):  curr = img[y, x]  *if* curr == 1 *and* prev == 0:  n += 1  prev = curr  *return* n   *def* \_\_main\_\_():  *# creating image object* sign\_img = Image.open(r"Files/sample\_sign.jpg")   *# using 8-bit pixel convertion to b&w* sign\_img\_l = sign\_img.convert("L")   sign\_img\_l.save(r"Files/sample\_sign\_bw.jpg")   file\_dir = "Files/"   *# open file and convert to grey scale* filename = os.path.join(file\_dir, "sample\_sign\_bw.jpg")  img = cv2.imread(filename)[:, :, 0] *# keep only one color dimension   print*("Shape of image:")  *print*(img.shape)  *# show\_image(img)   # normalize* img = img / 255  show\_image(img)   *# to analyze the pixel values* hist, bin\_edges = np.histogram(img)  plt.bar(bin\_edges[:-1], hist, width=1)  plt.xlim(min(bin\_edges), max(bin\_edges))  plt.show()   *# convert to black and white* img[img > .5] = 1  img[img != 1] = 0   *# Task 1* bounded\_img, bounded\_box\_dims = bounded\_box(img)  show\_image(bounded\_img)  *# find and show image with bounding box* img\_with\_box = draw\_bounded\_box(img, bounded\_box\_dims)  show\_image(img\_with\_box)   *# Task 2* centroid\_dims = centroid(img)  cx, cy = centroid\_dims  *print*("Centroid is: ({0}, {1})".format(cx, cy))   *# Task 3  # get list of segment images & their coordinates in original image* segment\_imgs, coordinates = segments(img, centroid\_dims, bounded\_box\_dims)   img\_with\_lines = draw\_segment\_lines(img, coordinates)  show\_image(img\_with\_lines)   show\_image(segment\_imgs[0]) *# top left* show\_image(segment\_imgs[1]) *# top right* show\_image(segment\_imgs[2]) *# bottom left* show\_image(segment\_imgs[3]) *# bottom right   # task 4* tl\_trans = black\_to\_white\_trans(segment\_imgs[0])  *print*("Tansitions in TL: {0}".format(tl\_trans))  tr\_trans = black\_to\_white\_trans(segment\_imgs[1])  *print*("Tansitions in TR: {0}".format(tr\_trans))  bl\_trans = black\_to\_white\_trans(segment\_imgs[2])  *print*("Tansitions in BL: {0}".format(bl\_trans))  br\_trans = black\_to\_white\_trans(segment\_imgs[3])  *print*("Tansitions in BR: {0}".format(br\_trans))   T = (tl\_trans, tr\_trans, bl\_trans, br\_trans)  *print*()   *# Printing results of all tasks  print*("Value of B: " + str(bounded\_box\_dims))  *print*("Value of C: " + str(centroid\_dims))  *print*("Value of T: " + str(T))   *# Main* \_\_main\_\_() |

### Screenshots:

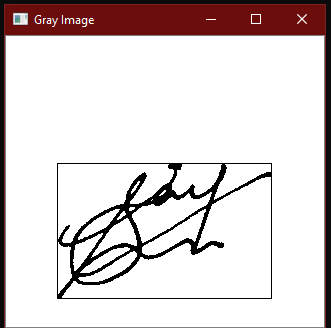
1. **Normalized Pixel Values Histogram to select the right threshold for binary image**

****

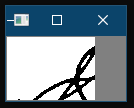
1. **Converted Black & White Bounded Image without Box**

****

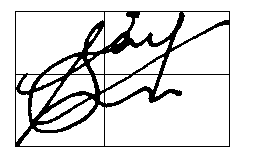
1. **Image with Bounded Box**

****

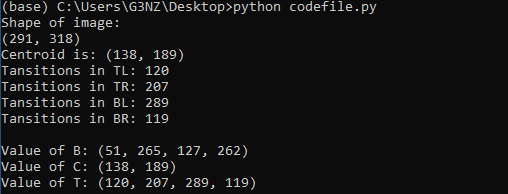
1. **Segments**

****

1. **Segment Lines**

****

1. **Console Output**



# Deliverables

Submit single word file having source code and output on LMS. Your submission should follow the following naming convention: YourName\_RegNo\_Section\_lab02