Peeraya Khantaruangsakul 63070501054

CPE 393 Machine Learning : Take Home Quiz1

Template Matching

[10 points] Search for the ‘t’ using “t_character.png” as template in the text image “text_image.png”. Use a bounding box to mark where ‘t’ were found. Use the Euclidean norm. You may use OpenCV to only read and write the image, but not to call the template matching routine.

```
import numpy as np
import cv2
def myImshow(title, img):
    cv2.startWindowThread()
    cv2.imshow(title, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
#Print error massage if image is null
def checkImage(img):
    if img is None:
        print('Image load failed!')
    else:
        print('Image load succeeded!')

#RBG image in BGR order in OpenCV
img1 = cv2.imread('t_character.png', 0)
checkImage(img1)
img2 = cv2.imread('text_image.png', 0)
checkImage(img2)
# Show image
myImshow('image1', img1)
myImshow('image2', img2)
# Get the shape of the img1 and img2
h_img1, w_img1 = img1.shape
h_img2, w_img2 = img2.shape

# Euclidean distance
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1-x2)**2))
# Slide the img1 over the img2
matches = []
for y in range(h_img2 - h_img1):
    for x in range(w_img2 - w_img1):
        # Get the img2 patch
        img2_patch = img2[y:y+h_img1, x:x+w_img1]
        #  distance between img1 and img2_patch
        dist = euclidean_distance(img1,img2_patch)
        # Update the distance and match location
        if dist == 0:
            matches.append((x, y))

for match in matches:
    x, y = match
    img_bounding = cv2.rectangle(img2, (x, y),
    (x+w_img1, y+h_img1), (0, 0, 255), 1)
# Show the result
myImshow('result', img_bounding)
# Save the image with the bounding box
cv2.imwrite("text_bounding_box.png",
img_bounding)
```

Result : Image with the bounding box.

```
import numpy as np
import cv2 as cv2
import matplotlib.pyplot as plt


# In[9]:


def myImshow(title, img):
    """
    function to make windows display work in jupyter notebook
    - shows image in a separate window,
    - waits for any key to close the window.

    """

    cv2.startWindowThread()
    cv2.imshow(title, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


# In[10]:


path = "D:/data/Dropbox/ML/"
#RGB images in BGR order in penCV
img1 = cv2.imread(path+'box.png',cv2.IMREAD_GRAYSCALE)          # queryImage
# Print error message if image is null
if img1 is None:
    print('Could not read query image')
else:
    print("Query Image read success...")

img2 = cv2.imread(path+'box_in_scene.png',cv2.IMREAD_GRAYSCALE) # targetImage
# Print error message if image is null
if img2 is None:
    print('Could not read training image')
else:
    print("Target Image read success...")


# In[11]:


# Initiate SIFT detector
sift = cv2.SIFT_create()
```

Image Convolution

[10 points] Create your own Gaussian Kernel. Using Python, compute and print the matrix for Gaussian kernel with σ = 2.5. using kernel size of 15 x 15 (we use width = ceiling (6*σ)). Print the kernel as output.

[10 points] 0.5 hrs. Modify the OpenCV code shown in class to show the result of the convolution of your 15 x 15 Gaussian kernel using the Lenna image.

```
import cv2
import numpy as np
def myImshow(title, img):
    cv2.startWindowThread()
    cv2.imshow(title, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
image = cv2.imread('Lenna.png')
if image is None:
    print("Could not read input image")
else:
```

```
#print matrix for Gaussian kernel
print(kernel)
burl = cv2.filter2D(image, -1, kernel)
myImshow("Gaussian Blur", burl)
cv2.imwrite('Gaussian_Blur.png',burl)
```

```
    myImshow("Lenna", image)

# make a matrix for Gaussian kernel
s = 2.5
kernel_size = np.floor(15/2)
# Create a 2D grid of coordinates
x, y = np.mgrid[-kernel_size:kernel_size+1, -kernel_size:kernel_size+1]
# Compute the Gaussian kernel
kernel = (np.exp(-(x**2 + y**2) / (2 * s**2)))/(2 * np.pi * s**2)
# Normalize the kernel
normalizing_constant = np.sum(kernel)
kernel = kernel / normalizing_constant
```

Result : Gaussian kernel 15 x 15.

```
[[1.00755667e-05 2.85059647e-05 6.87250219e-05 1.41190967e-04
  2.47179144e-04 3.68747952e-04 4.68770520e-04 5.07813042e-04
  4.68770520e-04 3.68747952e-04 2.47179144e-04 1.41190967e-04
  6.87250219e-05 2.85059647e-05 1.00755667e-05]
 [2.85059647e-05 8.06495603e-05 1.94438001e-04 3.99459887e-04
  6.99323439e-04 1.04326798e-03 1.32625353e-03 1.43671330e-03
  1.32625353e-03 1.04326798e-03 6.99323439e-04 3.99459887e-04
  1.94438001e-04 8.06495603e-05 2.85059647e-05]
 [6.87250219e-05 1.94438001e-04 4.68770520e-04 9.63057725e-04
  1.68599867e-03 2.51521446e-03 3.19746425e-03 3.46377167e-03
  3.19746425e-03 2.51521446e-03 1.68599867e-03 9.63057725e-04
  4.68770520e-04 1.94438001e-04 6.87250219e-05]
 [1.41190967e-04 3.99459887e-04 9.63057725e-04 1.97853777e-03
  3.46377167e-03 5.16734013e-03 6.56897674e-03 7.11608755e-03
  6.56897674e-03 5.16734013e-03 3.46377167e-03 1.97853777e-03
  9.63057725e-04 3.99459887e-04 1.41190967e-04]
 [2.47179144e-04 6.99323439e-04 1.68599867e-03 3.46377167e-03
  6.06392981e-03 9.04632026e-03 1.15001269e-02 1.24579388e-02
  1.15001269e-02 9.04632026e-03 6.06392981e-03 3.46377167e-03
  1.68599867e-03 6.99323439e-04 2.47179144e-04]
 [3.68747952e-04 1.04326798e-03 2.51521446e-03 5.16734013e-03
  9.04632026e-03 1.34955240e-02 1.71561734e-02 1.85850608e-02
  1.71561734e-02 1.34955240e-02 9.04632026e-03 5.16734013e-03
  2.51521446e-03 1.04326798e-03 3.68747952e-04]
 [4.68770520e-04 1.32625353e-03 3.19746425e-03 6.56897674e-03
  1.15001269e-02 1.71561734e-02 2.18097709e-02 2.36262427e-02
```

2.18097709e-02 1.71561734e-02 1.15001269e-02 6.56897674e-03
3.19746425e-03 1.32625353e-03 4.68770520e-04]
[5.07813042e-04 1.43671330e-03 3.46377167e-03 7.11608755e-03
1.24579388e-02 1.85850608e-02 2.36262427e-02 2.55940032e-02
2.36262427e-02 1.85850608e-02 1.24579388e-02 7.11608755e-03
3.46377167e-03 1.43671330e-03 5.07813042e-04]
[4.68770520e-04 1.32625353e-03 3.19746425e-03 6.56897674e-03
1.15001269e-02 1.71561734e-02 2.18097709e-02 2.36262427e-02
2.18097709e-02 1.71561734e-02 1.15001269e-02 6.56897674e-03
3.19746425e-03 1.32625353e-03 4.68770520e-04]
[3.68747952e-04 1.04326798e-03 2.51521446e-03 5.16734013e-03
9.04632026e-03 1.34955240e-02 1.71561734e-02 1.85850608e-02
1.71561734e-02 1.34955240e-02 9.04632026e-03 5.16734013e-03
2.51521446e-03 1.04326798e-03 3.68747952e-04]
[2.47179144e-04 6.99323439e-04 1.68599867e-03 3.46377167e-03
6.06392981e-03 9.04632026e-03 1.15001269e-02 1.24579388e-02
1.15001269e-02 9.04632026e-03 6.06392981e-03 3.46377167e-03
1.68599867e-03 6.99323439e-04 2.47179144e-04]
[1.41190967e-04 3.99459887e-04 9.63057725e-04 1.97853777e-03
3.46377167e-03 5.16734013e-03 6.56897674e-03 7.11608755e-03
6.56897674e-03 5.16734013e-03 3.46377167e-03 1.97853777e-03
9.63057725e-04 3.99459887e-04 1.41190967e-04]
[6.87250219e-05 1.94438001e-04 4.68770520e-04 9.63057725e-04
1.68599867e-03 2.51521446e-03 3.19746425e-03 3.46377167e-03
3.19746425e-03 2.51521446e-03 1.68599867e-03 9.63057725e-04
4.68770520e-04 1.94438001e-04 6.87250219e-05]
[2.85059647e-05 8.06495603e-05 1.94438001e-04 3.99459887e-04
6.99323439e-04 1.04326798e-03 1.32625353e-03 1.43671330e-03
1.32625353e-03 1.04326798e-03 6.99323439e-04 3.99459887e-04
1.94438001e-04 8.06495603e-05 2.85059647e-05]
[1.00755667e-05 2.85059647e-05 6.87250219e-05 1.41190967e-04
2.47179144e-04 3.68747952e-04 4.68770520e-04 5.07813042e-04
4.68770520e-04 3.68747952e-04 2.47179144e-04 1.41190967e-04
6.87250219e-05 2.85059647e-05 1.00755667e-05]]

Result : Convolution of above 15 x 15 Gaussian kernel using the Lenna image.

Before	After
	

KNN (K nearest neighbor) for 3 Classes

[10 points] KNN (K nearest neighbor) for 3 Classes. Modify the provided program for KNN with 2 random red/blue classes shown in class to have 3 classes of red/blue/yellow instead. Then use K = 4 to classify a randomly generated sample as red, yellow, or blue.

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
# containing 25 * 2 for 25 (x,y) values of known/training data that are random integers 0-99
trainData = np.random.randint(0,100,
(25,2)).astype(np.float32)
# Label Red or Blue or yellow with numbers 0 and 1 and 2
#Response is a random integers 0-2 of 25 * 1 values
responses = np.random.randint(0,3, (25,1)).astype(np.float32)
print ("Training Data:\n", trainData)
print("Responses Ravel or flattened as 1-D:\n", responses.ravel())
#Method .ravel flattens the np array to 1-D.
#color 0 is "Red", color 1 is "blue", color 2 is "yellow"

# Make red, blue, yellow
red = trainData[responses.ravel()==0]   #red is  0
print(red)
```

```
blue = trainData[responses.ravel()==1]   #blue is  1
print(blue)
yellow = trainData[responses.ravel()==2]   #yellow is  2
print(yellow)
#matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, ...)
plt.scatter(red[:,0],red[:,1],80,'r','^')  #size 80, red, triangle
plt.scatter(blue[:,0],blue[:,1],80,'b','s')  #size 80, blue, square
plt.scatter(yellow[:,0],yellow[:,1],80,'y','o')  #size 80, yellow, circle
plt.show()

#create 1 * 2 or 1 (x,y) value with random integer 0-99 for newcomer
newcomer = np.random.randint(0,100,(1,2)).astype(np.float32)
print(newcomer)
plt.scatter(red[:,0],red[:,1],80,'r','^')   #red, triangle
plt.scatter(blue[:,0],blue[:,1],80,'b','s')   #blue, square
plt.scatter(yellow[:,0],yellow[:,1],80,'y','o')   #yellow, circle
plt.scatter(newcomer[:,0],newcomer[:,1],80,'g','x')   #green, x
plt.show()

# Find near 4 color that near green color
knn = cv.ml.KNearest_create()
knn.train(trainData, cv.ml.ROW_SAMPLE, responses)
k=4
ret, results, neighbours, dist = knn.findNearest(newcomer, k)

# Change type for show correct anf beautiful value
results = results.ravel().astype(int)
neighbours = neighbours.ravel().astype(int).astype(str)
dist = dist.ravel().astype(int)

if results[0] == 0:
    results = "Red"
elif results[0] == 1:
    results = "Blue"
elif results[0] == 2:
    results = "Yellow"

for i in range(k):
    if neighbours[i] == '0':
        neighbours[i] = "Red"
    elif neighbours[i] == '1':
        neighbours[i] = "Blue"
    elif neighbours[i] == '2':
        neighbours[i] = "Yellow"

print("Result: ", results)
print("Nearest Neighbors: ", neighbours)
print("Distance: ", dist)
```

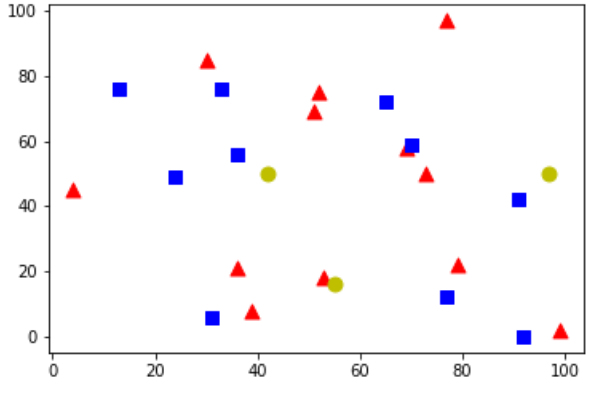
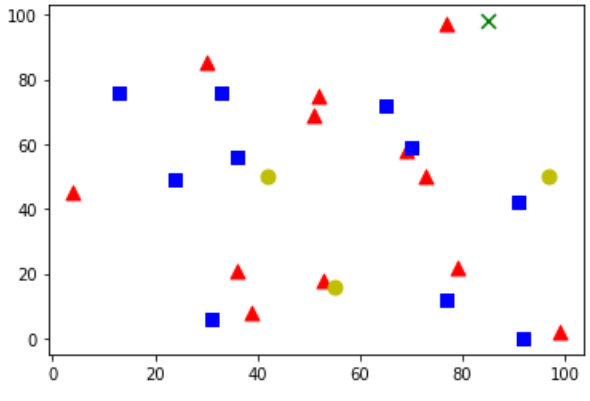
Result : A point of traning data.

[69. 58.]	[30. 85.]	[70. 59.]	[42. 50.]	[65. 72.]
[77. 97.]	[55. 16.]	[99. 2.]	[53. 18.]	[97. 50.]
[36. 21.]	[79. 22.]	[39. 8.]	[92. 0.]	[31. 6.]
[13. 76.]	[24. 49.]	[33. 76.]	[91. 42.]	[77. 12.]
[73. 50.]	[52. 75.]	[51. 69.]	[36. 56.]	[4. 45.]

Result : Chooses each point to red blue and green.

0	0	1	2	1
0	2	0	0	2
0	0	0	1	1
1	1	1	1	1
0	0	0	1	0

Result : Use matplotlib.pyplot.scatter to show each position.

Before has new comer	After has new comer
	
	New comer position : [85. 98.]

Result : KNN (K nearest neighbor) when K = 4.

Result: Red

Nearest Neighbors: ['Red' 'Blue' 'Red' 'Blue']

Distance: [65 1076 1618 1746]

Image Matching with KNN

[10 points] Image Matching with KNN. Try the provided image matching program on a test image of an object you photographed yourself. Then photograph the object in a different environment as a target image. Show your input and output image. Also, show the 2 input images with SIFT features as asked for in the jupyter notebook provided.

```
import numpy as np
import pandas as pd
import cv2 as cv2
def myImshow(title, img):
    cv2.startWindowThread()
    cv2.imshow(title, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
#Print error massage if image is null
def checkImage(img):
    if img is None:
        print('Image load failed!')
    else:
        print('Image load succeeded!')
```

```
#RBG image in BGR order in OpenCV
img1 = cv2.imread('box.png', 0)
checkImage(img1)
img2 = cv2.imread('box_in_scene.png', 0)
checkImage(img2)
myImshow('a box', img1)
myImshow('many boxes', img2)
```

```
#Initialize SIFT detector
sift = cv2.SIFT.create()
# Detect keypoints and compute descriptors for the first image
keypoints1, descriptors1 = sift.detectAndCompute(img1, None)
# Detect keypoints and compute descriptors for the second image
keypoints2, descriptors2 = sift.detectAndCompute(img2, None)
# KNN Match
matcher = cv2.BFMatcher()
matches = matcher.knnMatch(descriptors1, descriptors2, 2)
# Filter matching
good_matches = []
for m, n in matches:
    if m.distance < 0.7 * n.distance:
```

```
        good_matches.append(m)
    else:
        pass
# Draw matches
img_matches = cv2.drawMatches(img1, keypoints1, img2, keypoints2, good_matches,
None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
myImshow('Matches', img_matches)
cv2.imwrite('find_box.png',img_matches)
```

Result : Photograph the object in a different environment as a target image.