CPE383 Machine Learning: Quiz1

1. 10 points. 1.5 hrs. **Template Matching**. 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.

```
[1] import cv2
    import numpy as np

    #this was created because Google Colab does not allow cv2.imshow, so must patch by cv2_imshow.
    #If we switch over to regular jupyter notebook not on Colab, we can change c2_imshow to cv2.imshow.
    from google.colab.patches import cv2_imshow #only used when running in Google Colab
    def my_imshow(title, img ):
      cv2_imshow(img) #should be changed to c2.imshow when not in Colab

    # Set a threshold for detection
    threshold = 0.9
    match_x = []
    match_y = []

    # Read the template and larger images
    template = cv2.imread("t_character.png", 2)
    image = cv2.imread("text_image.png", 2)

    # Get the size of the template and input images
    template_height, template_width = template.shape
    image_height, image_width = image.shape
    print("template:", template_height, template_width)
    print("image:", image_height, image_width)
    print(template)
```

```
template: 9 8
image: 851 634
[[255 255 165  68 245 255 255 255]
 [255 255 165  68 245 255 255 255]
 [165   0   0   0   0   0  21 205]
 [255 255 165  68 245 255 255 255]
 [255 255 165  68 245 255 255 255]
 [255 255 165  68 245 255 255 255]
 [255 255 165  68 245 255 255 255]
 [255 255 201  21 205 255 255 255]
 [255 255 255 128   0   0  21 205]]
```

```
# Iterate over the input image
for i in range(8, image_height-8):
    for j in range(7, image_width-7):
        # Extract the sub-image
        sub_image = image[i:i+template_height, j:j+template_width]
        if (template == sub_image).all():
            # Append the x, y coordinates to the matching lists
            match_x.append(j)
            match_y.append(i)

print(match_x)
print(match_y)

# Draw bounding boxes on the image
for i in range(len(match_x)):
  markedImage = cv2.rectangle(image, (match_x[i], match_y[i]), (match_x[i]+template_width, match_y[i]+template_height), (0, 0, 255), 1)

# Show the image
my_imshow("Template Matching", markedImage)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
[43, 43, 43, 75, 107, 187, 243, 107, 123, 67, 107, 395, 443, 235, 75, 195, 267, 75, 99, 123, 139, 91, 91, 19, 107, 163, 51, 67, 155, 67, 163, 531, 571, 51, 67, 155, 211, 67, 131, 43, 67, 147, 17
[8, 26, 44, 44, 44, 44, 44, 152, 152, 188, 188, 188, 188, 206, 224, 224, 224, 296, 296, 314, 314, 332, 350, 458, 458, 494, 512, 548, 548, 584, 620, 620, 620, 638, 674, 674, 674, 710, 710, 818, 8
```

Result:

```
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()
```

2. **Image Convolution**. Create your own Gaussian Kernel

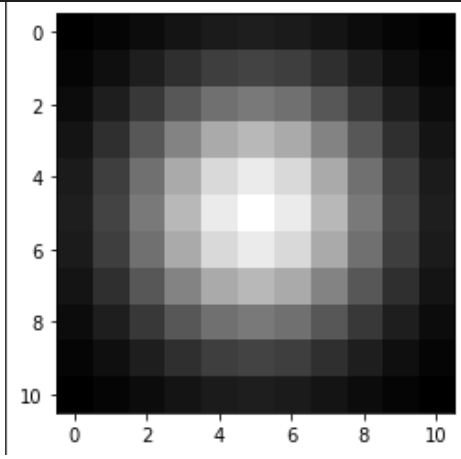
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

#this was created because Google Colab does not allow cv2.imshow, so must patch by cv2_imshow.
#If we switch over to regular jupyter notebook not on Colab, we can change c2_imshow to cv2.imshow.
from google.colab.patches import cv2_imshow #only used when running in Google Colab
def my_imshow(title, img ):
  cv2_imshow(img) #should be changed to c2.imshow when not in Colab

path = ""
fileName = path + "Lenna.png" #+ "cat_dog_hug.jfif" #+ "Lenna.png" #  + "printed.jpg"  + "cat_dog_hug.jfif" + "Lenna.png"

#RGB images in BGR order in OpenCV
image = cv2.imread(fileName, cv2.IMREAD_COLOR)

# Print error message if image is null
if image is None:
    print('Could not read image')
else:
    print("Image file read success...")

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()
    my_imshow(title, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```
Image file read success...
```

2.1. 10 points. 2 hrs. 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.

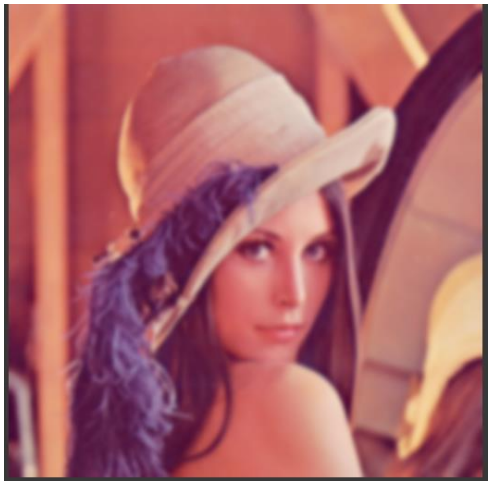
```
[11] # Define kernel size, sigma value and initialize gaussian matrix
    n = 5
    sigma = 2.5
    kernel_size = (2*n+1, 2*n+1)
    gaussian = np.zeros(kernel_size)

    # Compute the Gaussian filter
    for i in range(-n, n+1):
        for j in range(-n, n+1):
            i2 = i + n
            j2 = j + n
            gaussian[i2, j2] = np.exp(-(i**2 + j**2) / (2 * sigma**2))

    # Normalizing the kernel
    min_val = np.min(gaussian)
    gaussian = gaussian / min_val
    gaussian = gaussian / gaussian.sum()
    gaussian = gaussian.round(decimals=4, out=None)

    # Display gaussian filter in gray scale and matrix
    print(gaussian)
    plt.imshow(gaussian, cmap='gray')
    plt.show()
```

```
[[0.0005 0.001  0.0018 0.0026 0.0034 0.0036 0.0034 0.0026 0.0018 0.001
  0.0005]
 [0.001  0.0021 0.0036 0.0054 0.0069 0.0075 0.0069 0.0054 0.0036 0.0021
  0.001 ]
 [0.0018 0.0036 0.0064 0.0095 0.0121 0.0131 0.0121 0.0095 0.0064 0.0036
  0.0018]
 [0.0026 0.0054 0.0095 0.0142 0.018  0.0195 0.018  0.0142 0.0095 0.0054
  0.0026]
 [0.0034 0.0069 0.0121 0.018  0.0229 0.0248 0.0229 0.018  0.0121 0.0069
  0.0034]
 [0.0036 0.0075 0.0131 0.0195 0.0248 0.0269 0.0248 0.0195 0.0131 0.0075
  0.0036]
 [0.0034 0.0069 0.0121 0.018  0.0229 0.0248 0.0229 0.018  0.0121 0.0069
  0.0034]
 [0.0026 0.0054 0.0095 0.0142 0.018  0.0195 0.018  0.0142 0.0095 0.0054
  0.0026]
 [0.0018 0.0036 0.0064 0.0095 0.0121 0.0131 0.0121 0.0095 0.0064 0.0036
  0.0018]
 [0.001  0.0021 0.0036 0.0054 0.0069 0.0075 0.0069 0.0054 0.0036 0.0021
  0.001 ]
 [0.0005 0.001  0.0018 0.0026 0.0034 0.0036 0.0034 0.0026 0.0018 0.001
  0.0005]]
```



2.2. 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.

Original image



Blurred image

3. 10 points. 1 hrs. **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.

```
# Feature set 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 each one either Red or Blue with numbers 0 and 1. Response is a random integers 0-1 of 25 * 1 values
responses = np.random.randint(0,3,(25,1)).astype(np.float32) #responses 25 x 1 matrix

print ("Training Data:\n", trainData)
print("\n Responses:\n", responses)
print("Responses Ravel or flattened as 1-D:\n", responses.ravel()) #Method .ravel flattens the np array to 1-D.
print("Red Responses: \n ", responses.ravel()== 0) #color 0 is "Red", color 1 is blue
```

```
[ ] # Make red and blue
    red = trainData[responses.ravel()==0] #red is now trained data with responses of 0
    print(red)
```

```
[[75. 76.]
 [52. 26.]
 [87. 63.]
 [69. 28.]
 [96. 35.]
 [87. 90.]
 [97. 88.]]
```

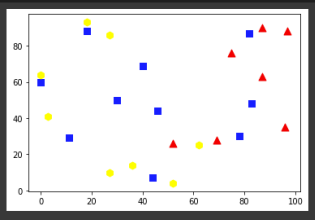
```
[ ] blue = trainData[responses.ravel()==1] #blue is now the trained data with responses of 1
    print(blue)
```

```
[[46. 44.]
 [30. 50.]
 [11. 29.]
 [82. 87.]
 [83. 48.]
 [18. 88.]
 [40. 69.]
 [78. 30.]
 [44.  7.]
 [ 0. 60.]]
```

```
[ ] yellow = trainData[responses.ravel()==2] #blue is now the trained data with responses of 1
    print(yellow)
```

```
[[36. 14.]
 [18. 93.]
 [27. 10.]
 [27. 86.]
 [62. 25.]
 [ 3. 41.]
 [ 0. 64.]
 [52.  4.]]
```

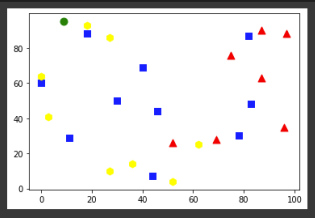
```
[ ] #matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, ...)
    plt.scatter(red[:,0],red[:,1],80,'r','^') #size 80, red, triangle
    plt.scatter(blue[:,0],blue[:,1],80,'b','s') #size 80, blue, square
    plt.scatter(yellow[:,0],yellow[:,1],80,'yellow','h') #size 80, yellow, hexagon
    plt.show()
```



```
[ ] #create 1 * 2 or 1 (x,y) value with random integer 0-99
    newcomer = np.random.randint(0,100,(1,2)).astype(np.float32)
    print(newcomer)
```

```
[[ 9. 95.]]
```

```
    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,'yellow','h') #yellow, hexagon
    plt.scatter(newcomer[:,0],newcomer[:,1],80,'g','o') #green, circle
    plt.show()
```



```
[ ] colorName = np.array(['Red', 'Blue', 'Yellow'])

    knn = cv.ml.KNearest_create()
    knn.train(trainData, cv.ml.ROW_SAMPLE, responses)
    k = 4
    ret, results, neighbors, dist = knn.findNearest(newcomer, k)

    '''
    resultColor = []
    for result in results:
        if result[0] == 0:
            resultColor.append("Red")
        else:
            resultColor.append("Blue")

    neighborColors = []
    for neighbor in neighbors[0]:
        if neighbor == 0:
            neighborColors.append("Red")
        else:
            neighborColors.append("Blue")
    '''
    resultColor = colorName[results[0].astype(int)]
    neighborColors = colorName[neighbors[0].astype(int)]

    print( "result color:  {}\n".format(resultColor) )
    print( "neighbors:  {}\n".format(neighbors) )
    print( "neighbor colors:  {}\n".format(neighborColors) )
    print( "neighbor distances:  {}\n".format(dist[0]) )
```

```
result color:  ['Yellow']

neighbors:  [[2. 1. 2. 2.]]

neighbor colors:  ['Yellow' 'Blue' 'Yellow' 'Yellow']

neighbor distances:  [  85.  130.  405. 1042.]
```

4. 10 points. 0.5 hrs. **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 cv2 as cv2
import matplotlib.pyplot as plt

#this was created because Google Colab does not allow my_imshow, so must patch by cv2_imshow.
#If we switch over to regular jupyter notebook not on Colab, we can change c2_imshow to my_imshow.
from google.colab.patches import cv2_imshow #only used when running in Google Colab
def my_imshow(title, img ):
  print(title)
  cv2_imshow(img) #should be changed to c2.imshow(img, title) when not in Colab

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()
    my_imshow(title, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

path = ""
#RGB images in BGR order in penCV
img1 = cv2.imread(path+'pic1.jpg',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+'pic2.jpg',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...")
```

```
Query Image read success...
Target Image read success...
```

```
# Initiate SIFT detector
sift = cv2.SIFT_create()
# find the keypoints and descriptors with SIFT
#cv.KeyPoint(pt, size[, angle[, response[, octave[, class_id]]]])
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

print(kp1[0], '\n',des1[0], '\n Size = ', des1[0].size)
```

```
< cv2.KeyPoint 0x7fad4f7021b0>
 [143.   6.   0.   0.   0.   0.   0.   9. 174.   2.   0.   0.   0.   0.
   0.  20. 174.   2.   0.   0.   0.   0.   0.  17.  67.   2.   0.   0.
   0.   0.   0.  10. 113.   3.   0.   0.   0.   0.   0.  10. 174.   9.
   0.   0.   0.   0.   0.  31. 174.   6.   0.   0.   0.   0.   0.  35.
  81.   2.   0.   0.   0.   0.   0.  20.  15.   0.   0.   0.   0.   0.
   0.   4. 174.  20.   0.   0.   0.   0.   0.  28. 174.  31.   0.   0.
   0.   0.   0.  38.  48.   1.   0.   0.   0.   0.   0.  46.   0.   0.
   0.   0.   0.   0.   0.   0.  24.   2.   0.   0.   0.   0.   0.   4.
 142.  21.   0.   0.   0.   0.   0.  12.  50.  13.   0.   0.   0.   0.
   2.  16.]
 Size =  128
```

```
kp1[0].pt, kp1[0].size, kp1[0].angle
```

```
((2.675405740737915, 2631.305908203125), 1.9794273376464844, 39.77043151855469)
```

```
# BFMatcher with default params
bf = cv2.BFMatcher()
#if you use knnMatch, it will return a list of (the best) k matches instead of a single DMatch.
#in our example k=2, so will get a list of best 2 matches per feature point
matches = bf.knnMatch(des1,des2,k=2)

print (matches)
```

```
((< cv2.DMatch 0x7fad406e76f0>, < cv2.DMatch 0x7fad406e7670>), (< cv2.DMatch 0x7fad406e7ab0>, < cv2.DMatch 0x7fad406e7b10>), (< cv2.DMatch 0x7fad406e74b0>, < cv2.DMatch 0x7fad406e7950>), (< cv2.I
```

```
[ ]  # Apply ratio test
     good = []
     for m,n in matches:
         if m.distance < 0.75*n.distance:
             good.append([m])

     # cv2.drawMatchesKnn expects list of lists as matches.
     #DrawMatchesFlags_DEFAULT
     #DrawMatchesFlags_DRAW_OVER_OUTIMG
     #DrawMatchesFlags_DRAW_RICH_KEYPOINTS
     #DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
     img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)
     #plt.imshow(img3),plt.show()
```

```
[ ]  myImshow('Original directly', img3)
```