assignment7

June 12, 2020

1 Machine Learning and Computer Vision

1.1 Assigment 7

This assignment contains 2 programming exercises.

1.2 Problem 1: Template Matching

Template matching is a naive technique for object detection and recognition in an image. It makes use of normalized cross correlation (NCC), which is a very similar operation to convolution. The differences between NCC and convolution are subtle3; you can think of it as a modified convolution. Follow the four steps to perform template matching on two different images:

(i) Correlation by Convolution: Using the Letter.jpg and LetterTemplate.jpg perform the following steps to find the template in the original image.

Convert both image and template to grayscale, double type.
Flip LetterTemplate.jpg vertically and horizontally.
Convolve the Letter.jpg and LetterTemplate.jpg.
Find the maximum point of the resulting convolved image. This is the location of the template match in the original image.

Draw a rectangle (using matplotlib.patches.rectangle() or cv2.rectangle()) over the original image centered at the location where the template match occurred. The size of this rectangle is the same as the template.

(ii) Normalized Cross Correlation: try to use functions like skimage.feature.match_template(), or scipy.signal.correlate2d() to performs a normalized cross correlation. It returns a correlation matrix that can be interpreted as an image. The maximum value of this image represents the location of the largest correlation between the image and the template. Repeat the process in (i) using Normalized Cross Correlation instead of convolution, and compare their result.

(iii) Multiple Matches: Now use Cross Correlation to detect faces in crowd.jpg using face1.jpeg as a template. This time instead of finding the maximum value in the correlation matrix, use a percentage of the maximum as the threshold to detect multiple faces instead of just the best match. Draw rectangles over the original image to indicate where the template matches occurred.

(iv) Multiple Templates: Repeat part (iii), but this time use all three templates, face1.jpeg, face2.jpeg and face3.jpeg, to generate three separate correlation matrices. Use each of the three to detect multiple face orientations in the crowd. Draw rectangles over the original image to indicate where the template matches occur (for all three templates). Make sure you use rectangles of different colors to represent matches corresponding to different templates.

Things to turn in:

Part (i): image after convolution (use imagesc()) and original image with rectangle depicting template match.

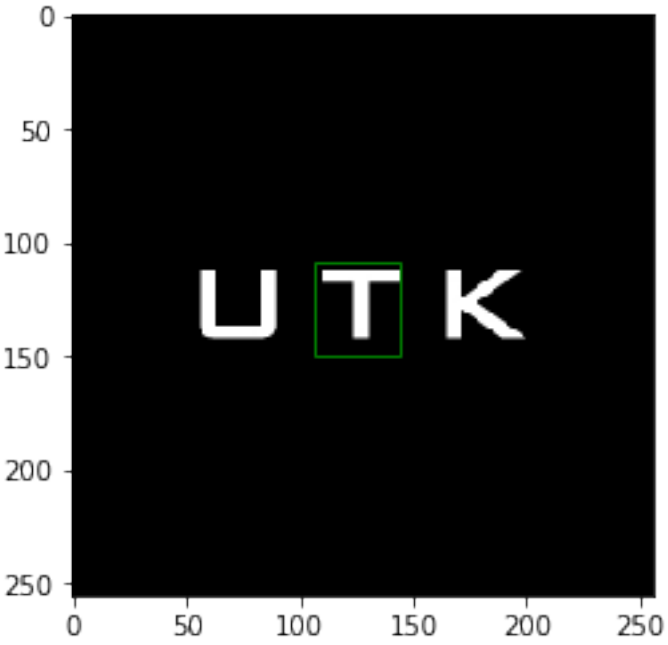
Part (ii): image after Cross Correlation (use imagesc()) and original image with rectangle depicting template match.

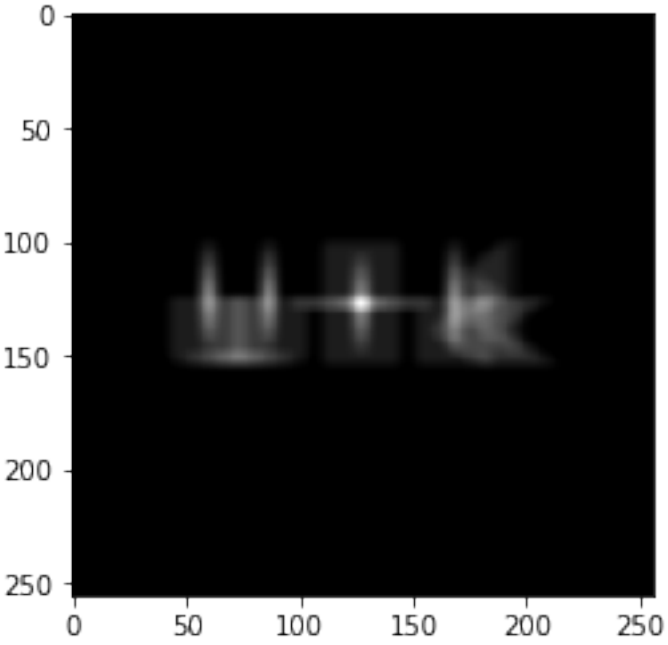
Part (iii): image after Cross Correlation (use imagesc()) and original image with rectangles depicting template matches. Also mention NCC threshold used to detect faces.

Part (iv): images after Cross Correlation (use imagesc()) and original image with rectangles depicting template matches (color coded for each template).

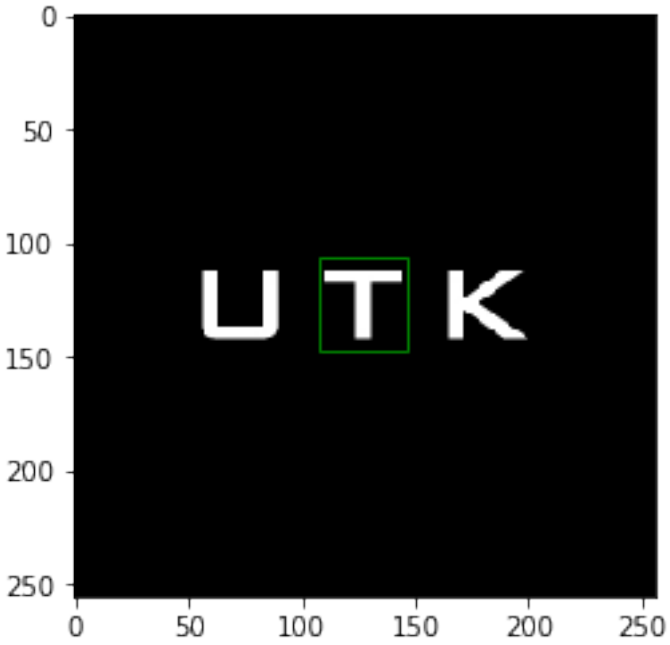
[34]:
```
import numpy as np
from imageio import imread
import matplotlib.pyplot as plt
from scipy.signal import convolve2d, convolve
from skimage.feature import match_template

#Function
def flipVertical(img):
    return img[::-1,: ]
def flipHorizontal(img):
    return img[:, ::-1]

#Import image here
# Convert to Gray scale image, select Threshold
img = imread('Letters.jpg')
template = imread('LettersTemplate.jpg')
# template = np.fliplr(np.flipud(template))
H, W = img.shape
template_h, template_w = template.shape

# flip image
template = flipVertical(template)
template = flipHorizontal(template)
# img = flipHorizontal(img)

# convolution
result = convolve2d(img.astype(float), template, mode='same')
m = np.max(result)
x, y = np.where(result==m)

plt.imshow(img, cmap='gray')
```

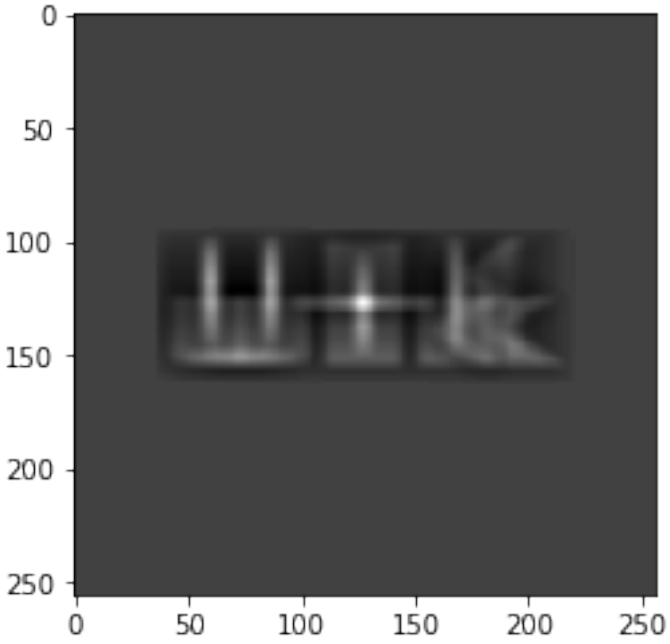
```
plt.gca().add_patch(plt.Rectangle((y-template_w//2, x-template_h//2),␣
 ↪template_h, template_w, fill=False, edgecolor='green'))
plt.figure()
plt.imshow(result, cmap='gray')
plt.show()
```





[14]:
```
# part ii

import numpy as np
from imageio import imread
import matplotlib.pyplot as plt
from scipy.signal import correlate2d, convolve2d
from skimage.feature import match_template


#Import image here
# Convert to Gray scale image, select Threshold
img = imread('Letters.jpg')
template = imread('LettersTemplate.jpg')
H, W = img.shape
template_h, template_w = template.shape
img.astype = 'int64'
template.astype = 'int64'


# cross correlation
result = match_template(img, template, pad_input=True)
# result = np.absolute(result)
m = np.max(result)
x, y = np.where(result==m)

plt.imshow(img, cmap='gray')
plt.gca().add_patch(plt.Rectangle((x-template_h//2, y-template_w//2),␣
 ↪template_h, template_w, fill=False, edgecolor='green'))
plt.figure()
plt.imshow(result, cmap='gray')
plt.show()
```

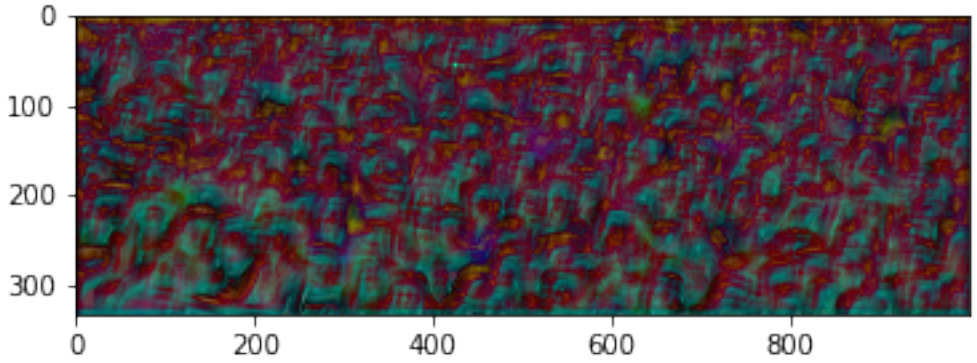




[13]:
```
# part iii
import numpy as np
from imageio import imread
import matplotlib.pyplot as plt
```

```
from scipy.signal import correlate2d, convolve2d
from skimage.feature import match_template


#Import image here
# Convert to Gray scale image, select Threshold
img = imread('crowd.jpg')
template = imread('face1.jpeg')
threshold = 0.55
H, W, D = img.shape
template_h, template_w, template_d = template.shape
img.astype = 'int64'
template.astype = 'int64'


# cross correlation
result = match_template(img, template, pad_input=True)
result = np.absolute(result)
x, y, z = np.where(result>threshold)
print('NCC threshold =',threshold)
# print(x, y, z)

plt.imshow(img)
for i, j in zip(x, y):
    plt.gca().add_patch(plt.Rectangle((j-template_w,i-template_h),␣
 ↪template_w*2, template_h*2, fill=False, edgecolor='yellow', linewidth='2'))

# plt.figure()
# plt.imshow(template, cmap='gray')
plt.figure()
plt.imshow(result)
plt.show()
```

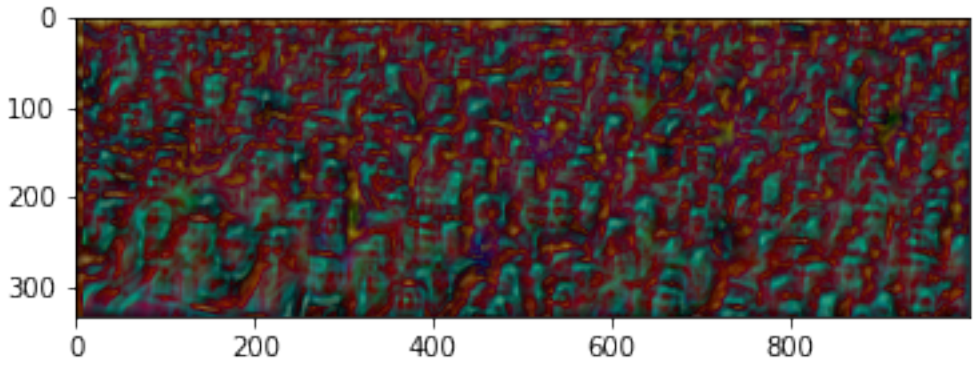
```
NCC threshold = 0.55
```



[153]:
```
# part iv
import numpy as np
from imageio import imread
import matplotlib.pyplot as plt
from scipy.signal import correlate2d, convolve2d
from skimage.feature import match_template


#Import image here
# Convert to Gray scale image, select Threshold
img = imread('crowd.jpg')
template_n = 3
templateList=[]
for i in range(1, template_n+1):
    name = 'face'+str(i)+'.jpeg'
    templateList.append(imread(name))
threshold = 0.8
H, W, D = img.shape
template_h, template_w = 25, 20
# img.astype = 'int64'
# template.astype = 'int64'

# print(len(templateList))
# print(templateList)
# cross correlation
locationList=[]
for t in templateList:
    result = match_template(img, t, pad_input=True)
    result = np.absolute(result)
    x, y, z = np.where(result>threshold)
    locationList.append(zip(x, y))
print('NCC threshold =',threshold)
```

```
# print(len(locationList))
# for tmatching in locationList:
#     for x,y in tmatching:
#         print(x,y)


colors=['red', 'green', 'blue']
plt.imshow(img)
for tmatching, color in zip(locationList,colors) :
    for i, j in tmatching:
#         print(i, j)
        plt.gca().add_patch(plt.Rectangle((j-template_w,i-template_h),␣
 ↪template_w*2, template_h*2, fill=False, edgecolor=color, linewidth='2'))

# plt.figure()
# plt.imshow(template, cmap='gray')
plt.figure()
plt.imshow(result)
plt.show()
```

NCC threshold = 0.8



1.3 Problem 2 K-Means Segmentation

In this problem, we shall implement a K-Means based segmentation algorithm from scratch. To do this, you are required to implement the following three functions -

def createDataset(im): This function takes in an RGB image as input, and returns a dataset of features which are to be clustered. The function returns features, which is an N Œ M matrix where N is the number of pixels in the image im, and M = 3 (to store the RGB value of each pixel).

def kMeansCluster(features, centers): This function is intended to perform K-Means based clustering on the dataset features (of size N Œ M). The function returns a list [idx, centers]. Each row in features represents a data point, and each column represents a feature. centers is a k Œ M matrix, where each row is the initial value of a cluster center. The output idx is an N Œ 1 vector that stores the final cluster membership (1, 2, ů ů ů , k) of each data point. The output centers are the final cluster centers after K-Means. Note that you may need to set a maximum iteration count to exit K-Means in case the algorithm fails to converge. You may use loops in this function.

def mapValues(im, idx): This function takes in the cluster membership vector idx (N Œ 1), and returns the segmented image im_seg as the output. Each pixel in the segmented image must have the RGB value of the cluster center to which it belongs. You may use loops for this part.

With the above functions set up, perform image segmentation on the image white-tower.png, with the number of clusters, nclusters = 7. To maintain uniformity in the output image, please initialize clusters centers for K-Means as follows -

```
np.random.seed(5)

nclusters = 7

features = createDataset(im)

id = np.random.randint(np.shape(features)[0], size=(1, nclusters))
```

Cluster Center points will be the corresponding row of each id value in features.

Things to turn in:

The input image, and the image after segmentation.

The final cluster centers that you obtain after K-Means.

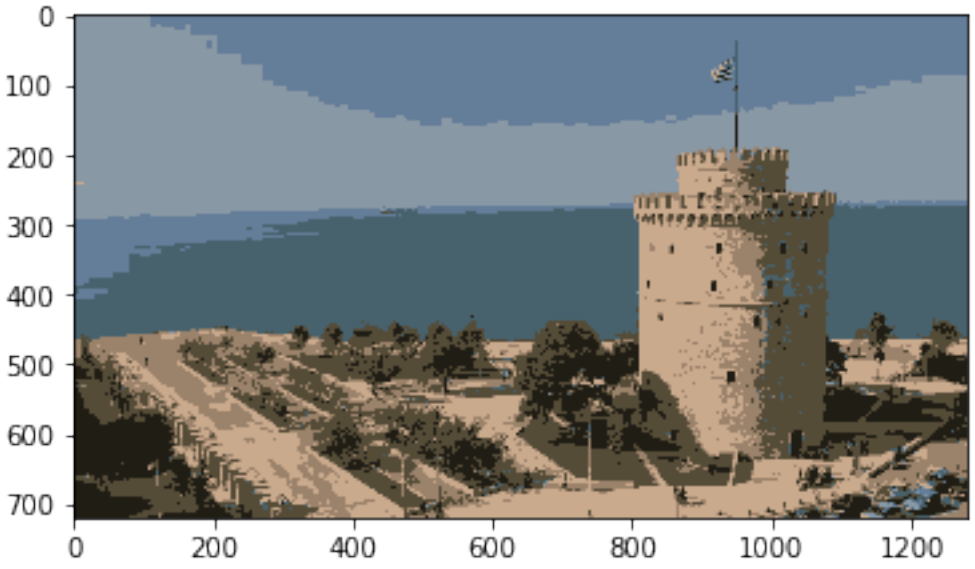
[30]:
```
#K mean segmentation Function
import numpy as np
from scipy.spatial.distance import cdist

def createDataset(im):
    h, w, d = im.shape
    features = np.reshape(im, (h*w, d))
    features = np.array(features)
    print(h*w)
    return features
```

```
# def kMeansCluster(features, centers, iterMax=20):
#     m = features.shape[0]
#     k = centers.shape[0]
#     idx = np.full((m, 1), -1)
#     clusterAssment = np.zeros((m,2)) #m*2
#     clusterChanged = True
#     iterNum=0
#     while clusterChanged and iterNum < iterMax:
#         clusterChanged = False
#         iterNum += 1
#         for i, pt in enumerate(features):
#             pts = np.tile(pt, (k,1))# tile pts for k rows
#             dists = np.sum((pts-centers)**2, axis=1)# calculate a dist array
#             minDist = np.min(dists)
#             minIndex = np.where(dists == minDist)[0][0] # find the min idx of␣
 ↪centers
#             if idx[i] != minIndex:
#                 clusterChanged = True
#                 idx[i] = minIndex

#         for cent in range(k):
#             ptsInClust = features[np.where(idx == cent)[0]]
#             centers[cent,:] = np.mean(ptsInClust, axis=0)
#     print('iterate %d times\n'%(iterNum))
#     print(centers)
#     return [idx,centers]

def kMeansCluster(features, centers, max_iter=10):
    idx = np.zeros(features.shape[0]);
    for iter in range(max_iter):
        dist = cdist(features,centers)
        idx_new = dist.argmin(axis=1)
        for k in range(centers.shape[0]):
            centers[k, :] = np.mean(features[idx_new == k, :] , 0)
        if np.array_equal(idx_new, idx):
            break
        idx = idx_new;
        if iter == max_iter:
            break
    return [idx,centers]

def mapValues(im, idx, colors):
    h, w, d = im.shape
    features = np.reshape(im, (h*w, d))
    features = np.array(features)
    for i in range(h*w):
        features[i,:] = colors[int(idx[i]), :]
```

```
    im_seg = np.reshape(features, (h, w, d))
    return im_seg
```

[32]:
```
from imageio import imread
import numpy as np
import matplotlib.pyplot as plt
#load image
im = imread('white-tower.png')
# im = imread('onion.png')


#Sample call and plot
np.random.seed(5)

nclusters = 7

features = createDataset(im)
m = im.shape[0] * im.shape[1]
idx = np.random.randint(np.shape(features)[0], size=(1, nclusters))
centers = np.array([features[i][:] for i in idx[0]])
print(centers)
[ptsAssign, centers] = kMeansCluster(features, centers, max_iter=100)
img_seg = mapValues(im, ptsAssign, centers)

plt.imshow(im)
plt.figure()
plt.imshow(img_seg)
plt.show()
```

```
921600
[[172 153 139]
 [ 82 111 143]
 [ 64  91 100]
 [107 131 159]
 [168 137 109]
 [200 165 143]
 [ 89  85  50]]
```



1.4 Conclusion

Have you accomplished all parts of your assignment? What concepts did you used or learned in this assignment? What difficulties have you encountered? Explain your result for each section. Please wirte one or two short paragraph in the below Markdown window (double click to edit).

**** Your Conclusion: ****

– For problem 1, I finished part ii to iv successfully, the functions are easy to use and it only needs seconds to give results. For each part from ii to iv, two images are shown: a result image

after template matching and a original image with rectangles to mark out the matching areas. According to the guides and with the help of functions, the results seems ok. But part i troubled me, I didn't fully understand the formula and there were always some bugs in the program.Then I asked lan for help. For problem 2, for k-means algorithm, I use matrix operation to simplify loops. The functions can do what I want, according to the formula, but the result is just like a gray image. The result seems to be correct because it can be segmented when I load a image with small area and give distinct colors, but 'white-tower.png' is too big. If the maximum of iteration times is small, the centers are not there, while if it is big, it will need a long long time to give a result. I tried to simplify several parts to test the functions: to load small-area image, to reduce the number of clusters, to point out centers, and to give distinct colors for each clusters. After testing with simplified conditions, the results met my expectation.

After asking lan, he modified the functions and I finally accomplish all the problems. For problem 1, we should flip the template and then use it to convolve. But I didn't fully understand the formula and didn't flip the template, so my result was not right. For problem 2, lan showed me to use funcitons to calculate distance and it really helped. I didn't know these functions before and used numpy functions to calculate distances and find the minimum, but using functions in scipy can greatly reduce the time cost. Thanks again to lan for helping me and showing me how to make the programs run in an efficient way.

** Submission Instructions**
Remember to submit you pdf version of this notebook to Gradescope. You can find the export option at File $\rightarrow$ Download as $\rightarrow$ PDF via LaTeX