Homework 4

December 8, 2021

Name - Parthasarathi Kumar
PID - A59003519
ECE 253 - Fundamentals of Digital Image Processing

Academic Integrity Policy: Integrity of scholarship is essential for an academic community. The University expects that both faculty and students will honor this principle and in so doing protect the validity of University intellectual work. For students, this means that all academic work will be done by the individual to whom it is assigned, without unauthorized aid of any kind.By including this in my report, I agree to abide by the Academic Integrity Policy mentioned above.

Assignmnet4_Problem1

December 8, 2021

Load required libraries

[1]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

[2]:
```
I_bird1 = cv2.imread('birds1.jpeg',0)/255
I_template = cv2.imread('template.jpeg',0)/255

plt.figure(figsize=(8,8))
plt.subplot(2,1,1)
plt.axis('off')
plt.imshow(I_bird1, cmap='gray')

plt.figure(figsize=(6,6))
plt.subplot(2,1,2)
plt.axis('off')
plt.imshow(I_template, cmap='gray')
```

[2]:
```
<matplotlib.image.AxesImage at 0x7f3a8b8e1dc0>
```

Function to perform cross correlation filter

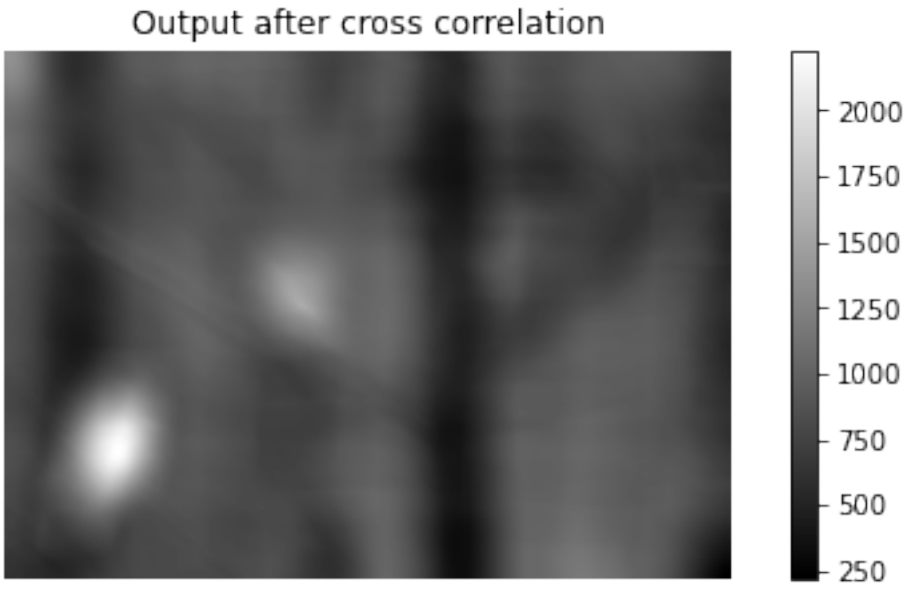
[3]:
```
def crossCorrelelationFilter(I, template):
    '''
    This function implements cross correlation filter
    '''
    I_crosscor = cv2.filter2D(I, ddepth=-1, kernel= template)
    idx = np.unravel_index(np.argmax(I_crosscor),I_crosscor.shape)

    w = template.shape[1]
    h = template.shape[0]
    cv2.rectangle(I, (idx[1]-int(w/2), idx[0]-int(h/2)), (idx[1]+int(w/2),␣
↪idx[0] + int(h/2)), (1,0,0), 2)

    return I_crosscor, I
```

Apply cross - correlation on birds1.jpeg using template

[4]:
```
template = I_template
I_crosscor, I_detected = crossCorrelelationFilter(np.copy(I_bird1), template)
plt.figure(figsize=(8,8))
plt.subplot(2,1,1)
plt.axis('off')
plt.title('Output after cross correlation')
im = plt.imshow(I_crosscor, cmap='gray')
plt.colorbar(im)

plt.figure(figsize=(8,8))
plt.subplot(2,1,2)
plt.axis('off')
plt.title('Detected object')
```

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

[4]: <matplotlib.image.AxesImage at 0x7f3a8aac9760>



Function to perform normalized cross correlation

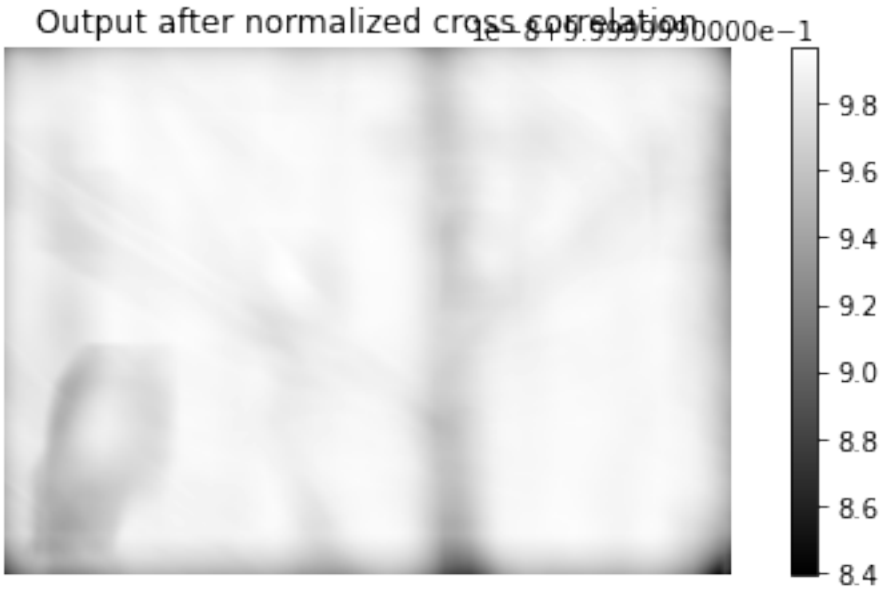
[5]:
```
def norCrossCor(I, template):
    '''
    Function implements normalized cross-correlation fiter
    '''
    # pad image
    m = int(template.shape[0]/2)
    n = int(template.shape[1]/2)
    I_pad = np.pad(I, ((m,m),(n,n)))

    # loop over image
    I_norcorr = np.zeros_like(I)
    template_pix_sum = np.sum(template)
    template = np.flip(np.flip(template,1),0)
    for i in range(I.shape[0]):
        for j in range(I.shape[1]):
            # Extract window
            I_window = I_pad[i:i+I_template.shape[0],j:j+I_template.shape[1]]
            window_pix_sum = np.sum(I_window)

            # Normalized cross correlation value
            A = I_window - window_pix_sum
            B = template - template_pix_sum

            N = np.sum(np.multiply(A,B))
            D = np.sqrt(np.multiply(np.sum(np.square(A)),np.sum(np.square(B))))
            I_norcorr[i,j] = N/D

    # Detect maxima
    idx = np.unravel_index(np.argmax(I_norcorr),I_norcorr.shape)
    w = template.shape[1]
    h = template.shape[0]
    I_detected = np.copy(I)
    # Annotate rectangle on detected object
    cv2.rectangle(I_detected, (idx[1]-int(w/2), idx[0]-int(h/2)), (idx[1]+int(w/
↪2), idx[0] + int(h/2)), (1,0,0), 2)

    return I_detected, I_norcorr
```

Normalized cross correlation on birds1.jpeg

[6]:
```
I_detected, I_norcorr = norCrossCor(I_bird1, I_template)

plt.figure(figsize=(8,8))
plt.subplot(2,1,1)
plt.axis('off')
im = plt.imshow(I_norcorr, cmap='gray')
plt.title('Output after normalized cross correlation')
```

```
plt.colorbar(im)

plt.figure(figsize=(8,8))
plt.subplot(2,1,2)
plt.axis('off')
plt.title('Detected object')
plt.imshow(I_detected, cmap = 'gray')
```

[6]: <matplotlib.image.AxesImage at 0x7f3a891095b0>



Normalized cross correlation on birds2.jpeg

[7]:
```
I_bird2 = cv2.imread('birds2.jpeg',0)/255
I_detected, I_norcorr = norCrossCor(I_bird2, I_template)
```

[8]:
```
plt.figure(figsize=(8,8))
plt.subplot(2,1,1)
plt.axis('off')
im = plt.imshow(I_norcorr, cmap='gray')
plt.title('Op after NCC')
plt.colorbar(im)

plt.figure(figsize=(8,8))
plt.subplot(2,1,2)
plt.axis('off')
plt.title('Detected object')
plt.imshow(I_detected, cmap = 'gray')
```

[8]:
```
<matplotlib.image.AxesImage at 0x7f3a89004eb0>
```


Detected object

[ ]:

Assignment4_Problem2

November 27, 2021

Import libraries

[25]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
```

Function to generate max ρ value

[26]:
```
def get_rho_val(I):
    '''
    Function generates the max rho value
    given the image.
    '''
    D_max = np.sqrt(np.square(I.shape[0])+np.square(I.shape[1]))
    D_max_int = np.rint(D_max).astype(np.int32)
    return D_max, D_max_int
```

Function generates the hough transform for a given image

[27]:
```
def houghTransform(I,pts):
    '''
    Function generates the hough tranform of a given image
    and candidate edge points.
    '''
    D_max, D_max_int = get_rho_val(I)
    theta = np.linspace(-90,90,181)
    acc = np.zeros(shape = (2*D_max_int+1,theta.shape[0]))

    for i in range(len(pts[0])):
        # Sweep across all theta vaues
        rho = pts[1][i]*np.cos(np.pi*theta/180) + pts[0][i]*np.sin(np.pi*theta/
↪180)

        # Accumalate votes
        rho_bins = np.rint(rho).astype(np.int32)
        np.add.at(acc, (rho_bins+D_max_int,range(theta.shape[0])), 1)
    extent = [-90 , 90, -D_max_int , D_max_int]
    return extent,acc
```

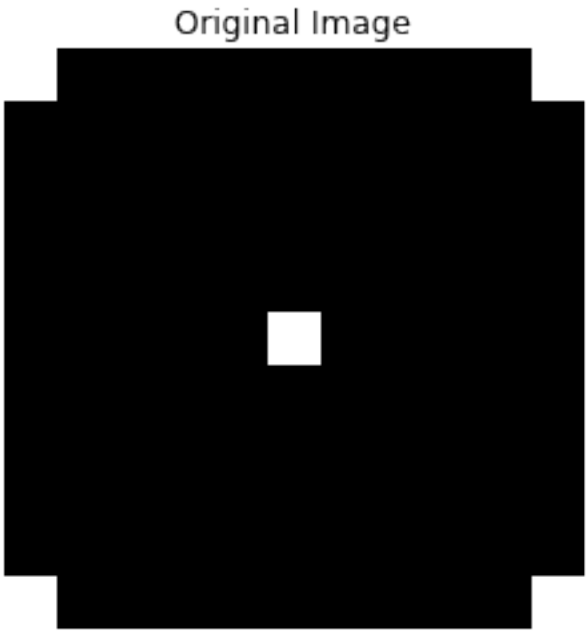
Function used to draw line.

[28]:
```
def drawLine(I, acc, threshold = 2,color = (1,0,0), thickness =1,␣
↪angleThreshold = [-90,90]):
    '''
    Function filters candidate based on votes and angle.
    Lines corresponding to filtered candidates are plotted
    on the given image (I).
    '''
    candidates = np.where(np.greater(acc,threshold))
    D_max, D_max_int = get_rho_val(I)

    for i in range(len(candidates[0])):

        x = candidates[0][i]
        rho = x - D_max_int
        theta_deg = (candidates[1][i]-90)

        if(np.logical_or(theta_deg < angleThreshold[0] , theta_deg >␣
↪angleThreshold[1])):
            continue

        theta = theta_deg/180*np.pi
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 + 1200*(-b))
        y1 = int(y0 + 1200*(a))
        x2 = int(x0 - 1200*(-b))
        y2 = int(y0 - 1200*(a))

        cv2.line(I,(x1,y1),(x2,y2),color,thickness)

    return I;
```

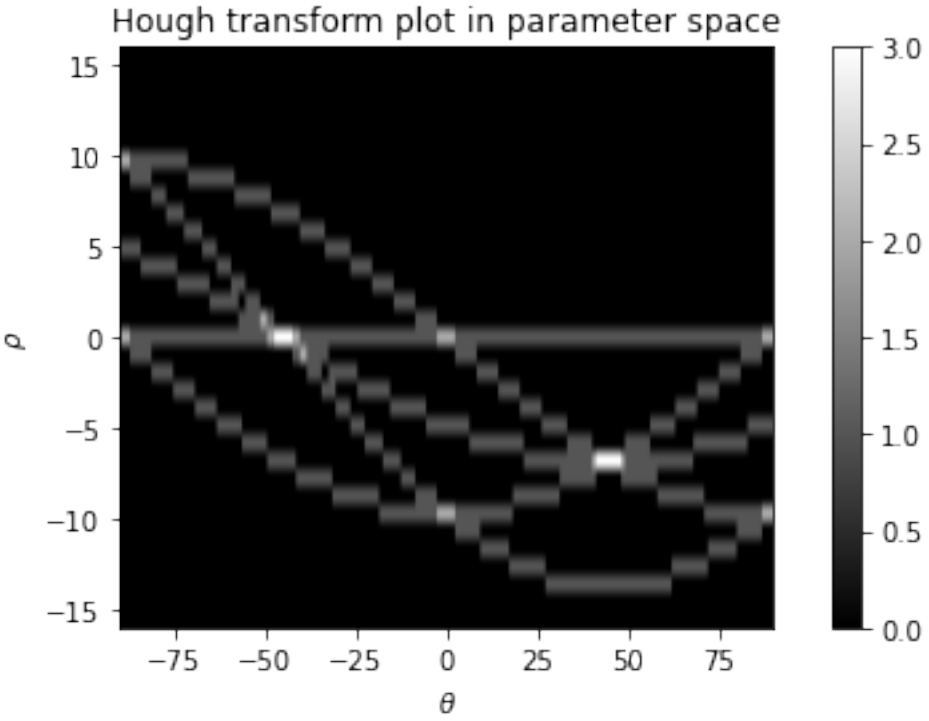
(i)
Hough transform on 11x11 image with 5 points

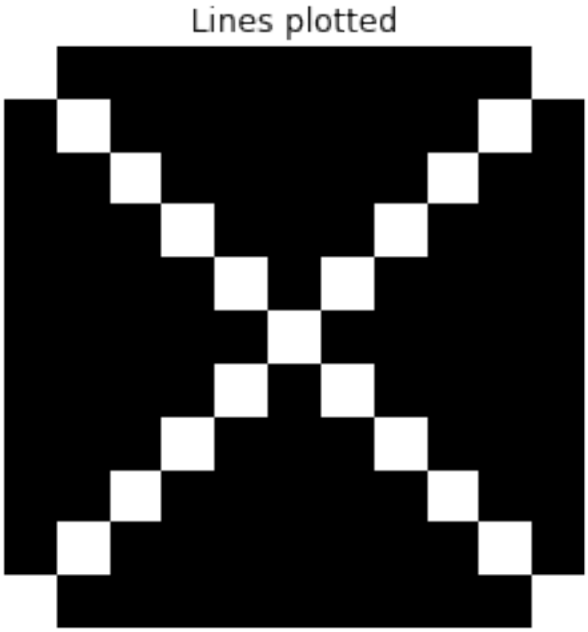
[29]:
```
# Generate test image
s = 11
img = np.zeros(shape = (s,s))
pts = ([0,0,s-1,s-1,int(s/2)],[0,s-1,0,s-1,int(s/2)])
img[pts[:]] = 1

plt.imshow(img,cmap='gray')
plt.title('Original Image')
plt.axis('off')
```

```
plt.show()
```



[30]:
```
extent, acc = houghTransform(img,pts)

# Plot Accumalator plot in rho-theta parameter space
fig, ax = plt.subplots(1,1, figsize = (8,4))
im = plt.imshow(acc, cmap='gray',extent=extent,aspect = 5)
plt.xlabel(r'$\theta$')
plt.ylabel(r'$\rho$')
plt.title('Hough transform plot in parameter space')
plt.colorbar(im, fraction = 0.05)
plt.show()

drawLine(img, acc, 2, [255, 0, 0], 1)

plt.title('Lines plotted')
plt.axis('off')
plt.imshow(img,cmap='gray')
plt.axis('off')
plt.show()
```





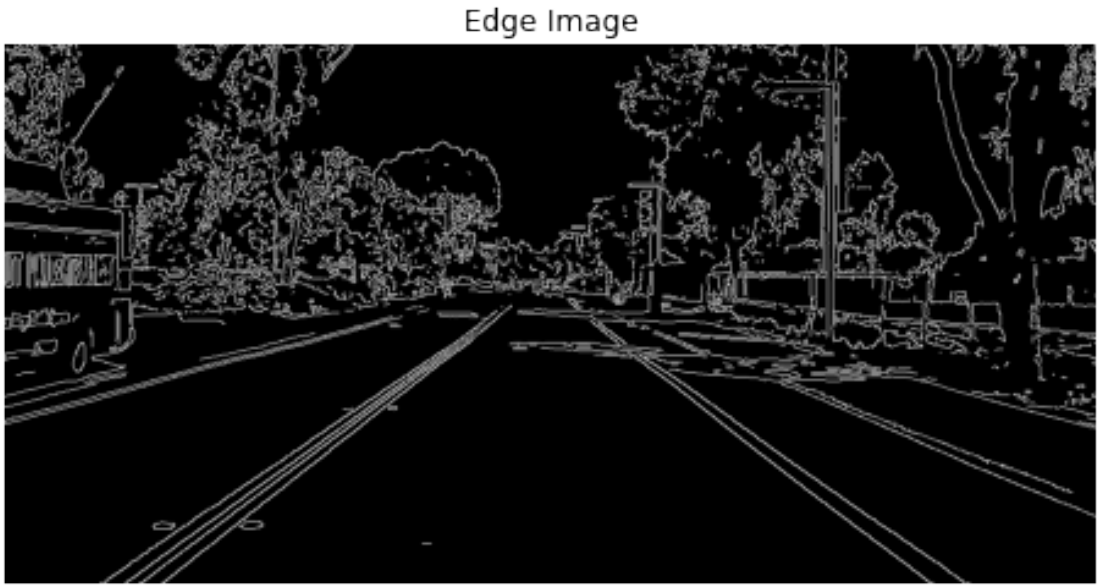
(iii)
Hough transform on lane image

[31]:
```
img = cv2.imread('lane.png',0)
plt.figure(figsize=(8,8))
plt.title('Original Image')
plt.axis('off')
plt.imshow(img, cmap = 'gray')
plt.show()
```

Performing canny edge detection on given image

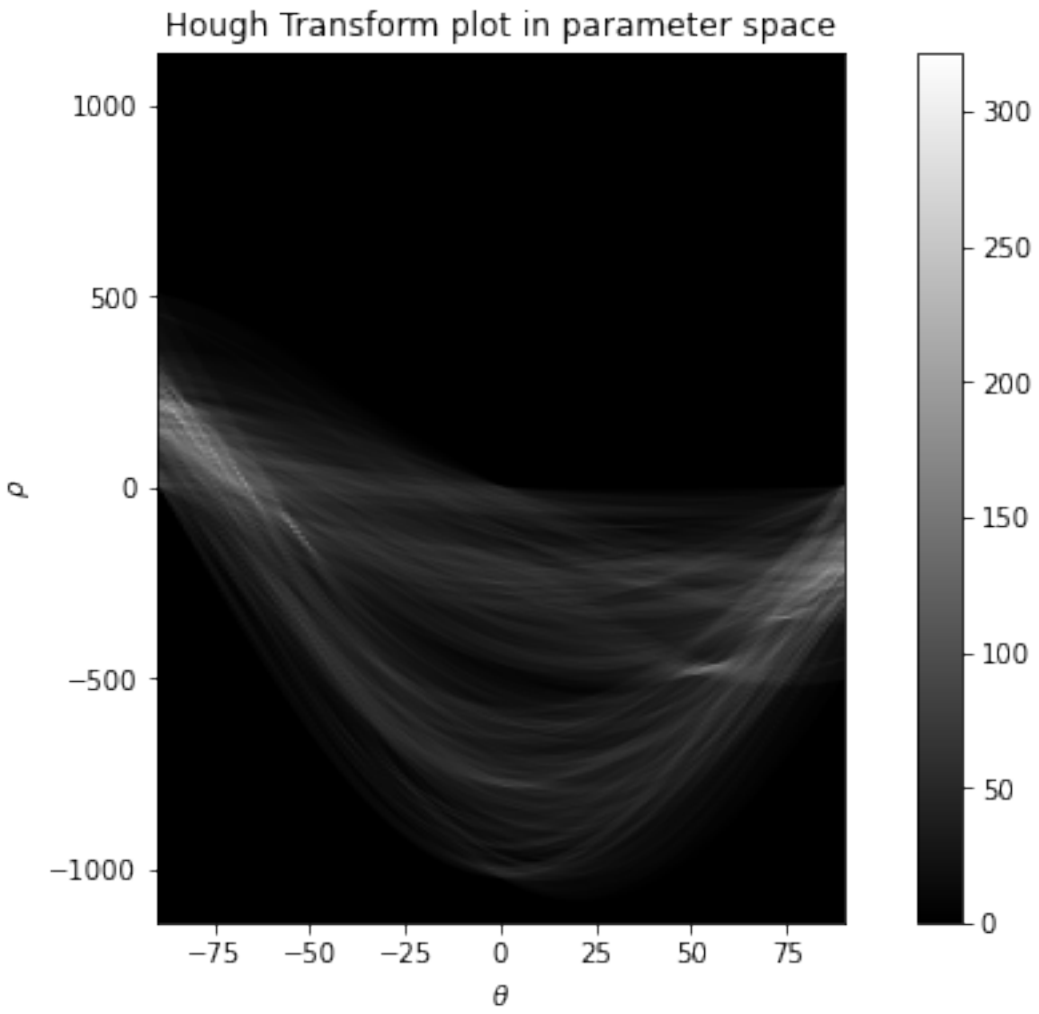
[32]:
```
# Perform Canny edge detection
edges = cv2.Canny(np.copy(img),150,200)
plt.figure(figsize=(8,8))
plt.imshow(edges,cmap = 'gray')
plt.axis('off')
plt.title('Edge Image')
np.unique(edges)
```

[32]:
```
array([  0, 255], dtype=uint8)
```



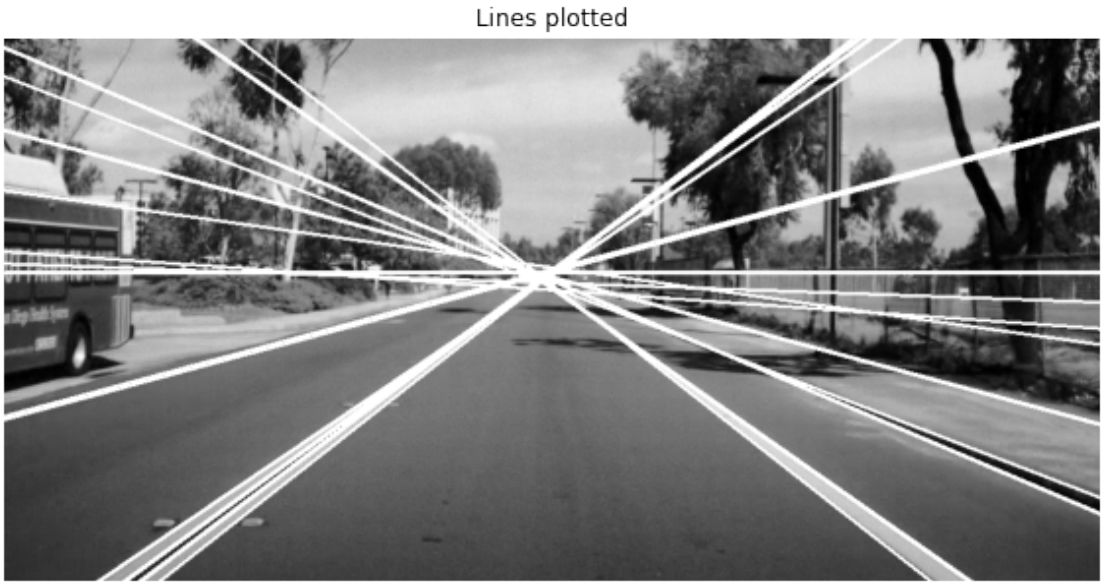
[33]:
```
pts = np.where(edges == 255)
extent, acc = houghTransform(edges, pts)
```

[34]:
```
# Plot Hough transgorm plot in rho-theta parameter space
plt.subplots(1,1, figsize = (10,6))

im = plt.imshow(acc, cmap='gray',extent=extent,aspect = 0.1)
plt.xlabel(r'$\theta$')
plt.ylabel(r'$\rho$')
plt.title('Hough Transform plot in parameter space')
plt.colorbar(im, fraction = 0.05)
plt.show()
```



Plotting all lines in the greater than 0.75 times the max value

[35]:
```
drawLine(img, acc, np.max(acc)*0.75,[255,0,0],2)

plt.figure(figsize = (10,6))
plt.title('Lines plotted')
plt.axis('off')
plt.imshow(img,cmap='gray')
plt.show()
```



Thresholding lines based on angle to extract lines corresponding to the lines. These lines are in the range [-55, -50] and [50,55] degree.

[36]:
```
img1 = cv2.imread('lane.png')
drawLine(img1, acc, np.max(acc)*0.75,[255,0,0],2,[50,55])
drawLine(img1, acc, np.max(acc)*0.75,[255,0,0],2,[-55,-50])
plt.figure(figsize = (10,6))
plt.title('Lines plotted corresponding to lanes')
plt.axis('off')
plt.imshow(img1,cmap='gray')
plt.show()
```

Lines plotted corresponding to lanes

Assignment4_Problem3

December 8, 2021

Import Libraries

[1]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
import pandas as pd
```

Function to create a NxM feature vector of given RGB image where N = number of pixels , M = 3 color channel

[2]:
```
def createDataset(I):
    '''
    Given a RGB Image of size (MxNx3)
    this function returns a 2D matrix
    of size MNx3
    '''
    return np.reshape(I,(I.shape[0]*I.shape[1],3))
```

Function to perform k-mean clustering

[3]:
```
def kMeansCluster(features, centers, max_iters):
    '''
    Performs k-means clustering given the intial
    k cluster centers on the dataset features
    Clustering done for max_Iter time
    If cluster center has no points, then update
    by assigning it to data center
    '''
    idx = np.ones((features.shape[0],1))*-1
    n_iters = 0
    n_idx = np.copy(idx)
    while(n_iters < max_iters):
        n_iters = n_iters + 1

        # Assign closest cluster centers
        dist = np.linalg.norm(features[:,None,:]-centers[None,:,:],axis=-1)
        n_idx = np.argmin(dist,axis = 1)

        # Break if no change in cluster centers
```

```
        if (np.array_equal(n_idx,idx)):
            break
        idx = np.copy(n_idx)
        # Recompute cluster centers based on new assignments
        for i in range(centers.shape[0]):
            if(np.any(idx == i)):
                centers[i] = np.mean(features[idx == i],axis =0)
            else:
                centers[i] = np.ones((1,3))*np.inf
    return idx,centers
```

Generates label mask corresponding to idx passed

[4]:
```
def mapValues(I, idx, centers):
    '''
    Given a list of cluster centers
    create a mask for given image
    '''
    segmented_I = np.zeros((I.shape[0],I.shape[1],3))
    mask_labels = idx.reshape(I.shape[0],I.shape[1])

    for i in range(centers.shape[0]):
        segmented_I[mask_labels==i,:] = centers[i]
    segmented_I = segmented_I.astype(np.uint8)
    return segmented_I
```

Read original image

[5]:
```
I = cv2.imread('white-tower.png')
I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(10,10))
plt.imshow(I)
plt.title('Original Image')
plt.axis('off')
```

[5]:
```
(-0.5, 1279.5, 719.5, -0.5)
```

Original Image

[6]:
```
nClusters = 7
#centers = features[np.random.choice(features.shape[0],7)]
centers = np.random.randint(0,255,(nClusters,3))
features = createDataset(I)
max_Iter = 100
idx,centers = kMeansCluster(features, centers, max_Iter)
```

Following are the cluster centers (0-255 range pixel value)

[7]:
```
print(centers)
```

```
[[ 32  29  21]
 [ 85  76  56]
 [ 72  98 109]
 [100 125 153]
 [137 152 165]
 [155 130 107]
 [202 170 141]]
```

[8]:
```
segmented_I = mapValues(I, idx, centers)

plt.figure(figsize=(10,10))
plt.imshow(segmented_I)
plt.title('Segmented Image')
plt.axis('off')
```

[8]: (-0.5, 1279.5, 719.5, -0.5)

Assignment4_Problem4

December 8, 2021

[12]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.patches as mpatches
```

[13]:
```
def plotImage(I, title,figSize = (12,12)):
    I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=figSize)
    plt.title(title)
    plt.imshow(I)
    plt.axis('off')
    plt.show()
```

1)
The model is a fully convoluted network. It can be broadly divided into 2 parts, the first part has a series of convolutions and max pooling layers that downsample the image and in the process get deep-features. This is followed by the part that is responsible for upsampling the image to generate a mask.

[24]:
```
plotImage(cv2.imread('Model_summary.png'), 'Model Summary',(15,15))
```

Model Summary

```
<bound method Module.parameters of fcn8s(
  (conv_block1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(100, 100))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=True)
  )
  (conv_block2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=True)
  )
  (conv_block3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=True)
  )
  (conv_block4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=True)
  )
  (conv_block5): Sequential(
    (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=True)
  )
  (classifier): Sequential(
    (0): Conv2d(512, 4096, kernel_size=(7, 7), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): Dropout2d(p=0.5, inplace=False)
    (3): Conv2d(4096, 4096, kernel_size=(1, 1), stride=(1, 1))
    (4): ReLU(inplace=True)
    (5): Dropout2d(p=0.5, inplace=False)
    (6): Conv2d(4096, 19, kernel_size=(1, 1), stride=(1, 1))
  )
  (score_pool4): Conv2d(512, 19, kernel_size=(1, 1), stride=(1, 1))
  (score_pool3): Conv2d(256, 19, kernel_size=(1, 1), stride=(1, 1))
  (upscore2): ConvTranspose2d(19, 19, kernel_size=(4, 4), stride=(2, 2), bias=False)
  (upscore4): ConvTranspose2d(19, 19, kernel_size=(4, 4), stride=(2, 2), bias=False)
  (upscore8): ConvTranspose2d(19, 19, kernel_size=(16, 16), stride=(8, 8), bias=False)
)>
```

2)
We are initializing the models using the parameters of a pretrained VGG16 model.

3) Training Loss

[15]:
```
plotImage(cv2.imread('Training_Loss.png'), 'Training Loss')
```

Validation Loss

[16]:
```
plotImage(cv2.imread('Validation_loss.png'),'Validation Loss')
```

4)
Following are the results on the validation set. It is evident that class 0 i.e. The class "ROADS" have the best performance. On the other hand, the classes 'RIDER' and 'MOTORCYLE' have the worst performance

[17]:
```
plotImage(cv2.imread('Validation_Run.png'), 'Training Loss')
```

Training Loss

```
Overall Acc:     0.9126775339962867
Mean Acc :       0.6303604594520074
FreqW Acc :      0.8477698473832183
Mean IoU :       0.5394424016602788
0 0.9548895279193593
1 0.6775241434854732
2 0.8452083805552156
3 0.415631112588026 26
4 0.34751554954606917
5 0.2414010122691258
6 0.30121989133492233
7 0.4034389892071633
8 0.8585675144902462
9 0.5286599277392117
10 0.8817178849858368
11 0.5433454893874735
12 0.2130477676478614
13 0.8509891571518705
14 0.5027899462823842
15 0.489207062646273 67
16 0.4226288081470614
17 0.2197325623418865
18 0.5518909038198346
```

The paper used 4 metrics to quabtify the results of the model -
1.Pixel Accuracy
2.Mean Accuracy
3.Mean Intersection over Union (IoU)
4.Frequency weighted IoU

5)
Code to visualize model results on a test image

[18]:
```
# Define colors and labels
colors = [[  0,   0,   0],
        [128, 64, 128],
        [244, 35, 232],
        [70, 70, 70],
        [102, 102, 156],
        [190, 153, 153],
        [153, 153, 153],
        [250, 170, 30],
```

```
        [220, 220, 0],
        [107, 142, 35],
        [152, 251, 152],
        [0, 130, 180],
        [220, 20, 60],
        [255, 0, 0],
        [0, 0, 142],
        [0, 0, 70],
        [0, 60, 100],
        [0, 80, 100],
        [0, 0, 230],
        [119, 11, 32],
    ]

class_names = [
        "unlabelled",
        "road",
        "sidewalk",
        "building",
        "wall",
        "fence",
        "pole",
        "traffic_light",
        "traffic_sign",
        "vegetation",
        "terrain",
        "sky",
        "person",
        "rider",
        "car",
        "truck",
        "bus",
        "train",
        "motorcycle",
        "bicycle",
    ]
colors = np.array(colors)
colors = colors/255
```

Function to plot mask

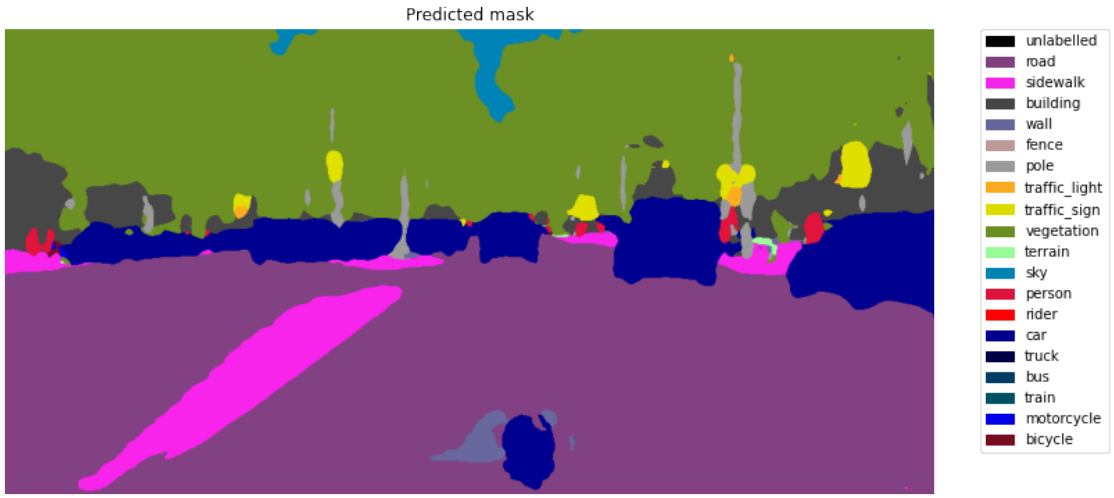
[19]:
```
def plotMask(I, title):
    plt.figure(figsize=(12,12))
    I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
    plt.imshow(I)
    plt.axis('off')
    plt.title(title)
```

```
    patches =[mpatches.Patch(color=colors[i],label=class_names[i]) for i in␣
↪range(len(class_names))]
    plt.legend(handles=patches, bbox_to_anchor=(1.05, 1), loc=2,␣
↪borderaxespad=0., prop={'size': 10})
    plt.show()
```

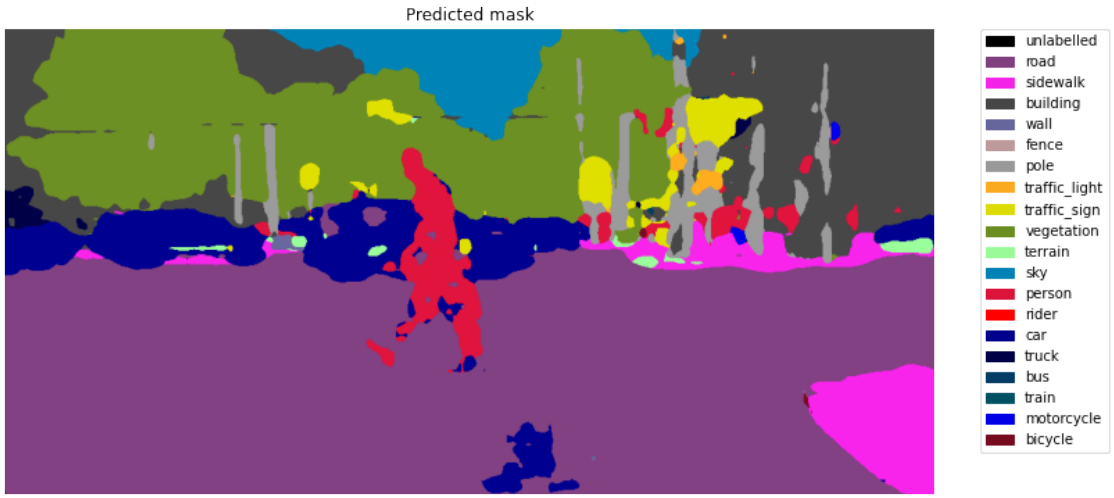
[20]:
```
def decode_segmap(temp, label_colours):
    r = temp.copy()
    g = temp.copy()
    b = temp.copy()
    for l in range(0, 20):
        r[temp == l] = label_colours[l][0]
        g[temp == l] = label_colours[l][1]
        b[temp == l] = label_colours[l][2]

    rgb = np.zeros((temp.shape[0], temp.shape[1], 3))
    print(rgb.shape)
    print(r.shape)
    rgb[:, :, 0] = r / 255.0
    rgb[:, :, 1] = g / 255.0
    rgb[:, :, 2] = b / 255.0
    return rgb
```

Original image and predicted mask on 2 test images

[21]:
```
I_test_1 = cv2.imread('berlin_000001_000019_leftImg8bit.png')
plotImage(I_test_1,'Original Image')

I_test_1_mask = cv2.imread('berlin_001_op.png')
plotMask(I_test_1_mask, 'Predicted mask')
```

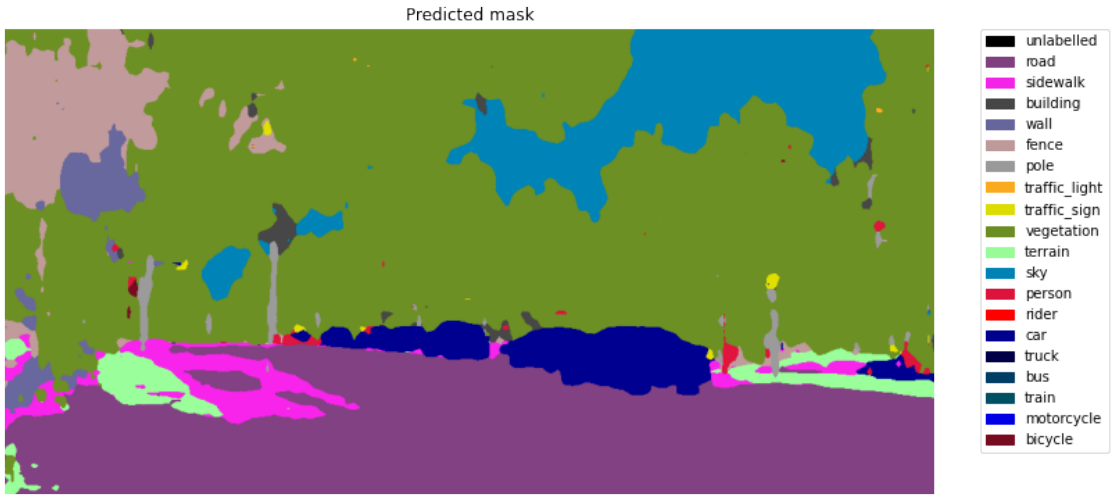


[22]:
```
I_test_1 = cv2.imread('berlin_000002_000019_leftImg8bit.png')
plotImage(I_test_1,'Original Image')

I_test_1_mask = cv2.imread('berlin_002_op.png')
plotMask(I_test_1_mask, 'Predicted mask')
```



6) Test image out of dataset with predicted msak

[23]:
```
I_test_1 = cv2.imread('test_phone.png')
plotImage(I_test_1,'Original Image')

I_test_1_mask = cv2.imread('test_op.png')
plotMask(I_test_1_mask, 'Predicted mask')
```



7)
The output of the model can be imporoved in following ways -

Making architecture improvements like the U-Net will improve the performance by providing relevant contextual information in the upsampling part of the network.

Use of more sophisticated convolutions like dilated convolution as done in deeplab v3

Assignment4_Problem5

December 8, 2021

[47]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
```

Following function stylize takes in a image and produces a stlized version of it using simple Gaussian blurring and k-means clustering

[58]:
```
def stylize(IM):
    '''
    stylize takes in a image and returns a
    stylized version of it as output

    The function internally does Gaussian
    blurring and k-means clustering to generate
    the image. (k = 4)

    input - RGB/ grayscale image
    output - stylized version of input

    '''
    # Blur Image
    kernel = np.ones((3,3),np.float32)/9
    I = cv2.filter2D(np.copy(IM),-1,kernel)
    # Define number of clusters
    nClusters = 4
    # Pick cluster centers
    centers = np.random.randint(0,255,(nClusters,3))
    features = np.reshape(I,(I.shape[0]*I.shape[1],3))
    # Perform k-means clustering in color space
    max_iters = 100
    idx = np.ones((features.shape[0],1))*-1
    n_iters = 0
    n_idx = np.copy(idx)
    while(n_iters < max_iters):
        n_iters = n_iters + 1

        # Assign closest cluster centers
        dist = np.linalg.norm(features[:,None,:]-centers[None,:,:],axis=-1)
```

```
        n_idx = np.argmin(dist,axis = 1)

        # Break if no change in cluster centers
        if (np.array_equal(n_idx,idx)):
            break
        idx = np.copy(n_idx)
        # Recompute cluster centers based on new assignments
        for i in range(centers.shape[0]):
            if(np.any(idx == i)):
                centers[i] = np.mean(features[idx == i],axis =0)
            else:
                centers[i] = np.ones((1,3))*np.inf
    # Generate mask
    segmented_I = np.zeros((I.shape[0],I.shape[1],3))
    mask_labels = idx.reshape(I.shape[0],I.shape[1])

    for i in range(centers.shape[0]):
        segmented_I[mask_labels==i,:] = centers[i]
    segmented_I = segmented_I.astype(np.uint8)


    return segmented_I
```

Helper function to plot images

[59]:
```
def plotImage(I):
    plt.figure(figsize=(10,10))
    plt.axis('off')
    plt.imshow(I)
```

Example 1

[60]:
```
I = cv2.imread('I1.jpeg')
I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
I_stylized = stylizeImage(I)
plotImage(I_stylized)
```

Example 2

[61]:
```
I = cv2.imread('I2.jpeg')
I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
I_stylized = stylizeImage(I)
plotImage(I_stylized)
```

Example 3

[62]:
```
I = cv2.imread('I3.jpg')
I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
I_stylized = stylizeImage(I)
plotImage(I_stylized)
```

[ ]:
```

```