ECE_253_hw4_Kalgundi_Srinivas_A59010584

December 9, 2021

1 Srinidhi Bharadwaj Kalgundi Srinivas

2 A59010584

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.

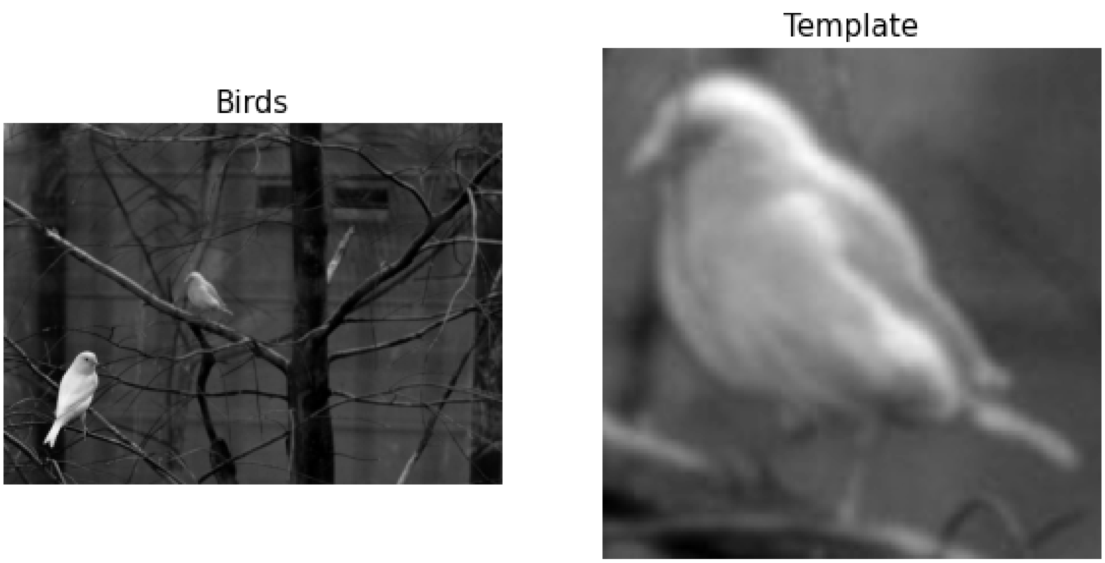
2.0.1 Problem 1: Detecting Objects with Template Matching

[1]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
import scipy.signal as sig
import math
```

[2]:
```
def plot_subplots(imageList, imageNames, rows, cols, gray=False, size=(10, 5),␣
 ↪colorbar=False):
    fig, axs = plt.subplots(rows, cols,figsize=size)
    [axi.set_axis_off() for axi in axs.ravel()]
    for i in range(rows*cols):
        ax1= fig.add_subplot(rows,cols,i+1)
        ax1.title.set_text(imageNames[i])
        ax1.title.set_size(15)
        ax1.axis('off')
        if gray == True:
            ax = ax1.imshow(imageList[i], cmap='gray')
        else:
            ax = ax1.imshow(imageList[i])
        if colorbar:
            plt.colorbar(ax)
```

Cross Correlation

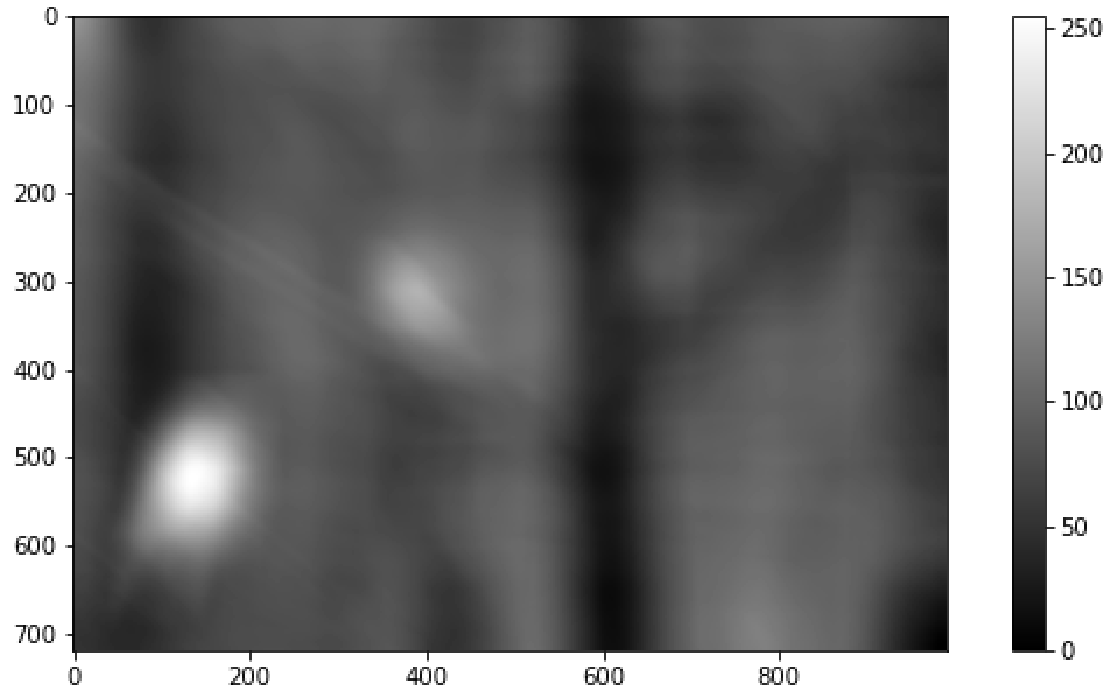
[3]:
```
birds = cv2.imread("birds1.jpeg")
template = cv2.imread("template.jpeg")
birds_color = cv2.cvtColor(birds, cv2.COLOR_BGR2RGB) #Used for plotting

birds_gray = cv2.cvtColor(birds, cv2.COLOR_BGR2GRAY).astype(np.float64)
template_gray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY).astype(np.float64)

images = [birds_gray, template_gray]
names = ["Birds", "Template"]
plot_subplots(images, names, 1, 2, True, colorbar=False)
```



[4]:
```
template_flip = cv2.flip(template_gray, -1)
convolved_out = cv2.filter2D(src=birds_gray, ddepth=-1, kernel=template_flip)
convolved_out = convolved_out.astype(np.float64)
#Normalizing
min_val = np.min(convolved_out)
max_val = np.max(convolved_out)
convolved_out = (convolved_out - min_val)
convolved_out = (convolved_out / (max_val - min_val)) * 255.0

print("Maximum value is:",np.max(convolved_out))
plt.figure(figsize=(10, 5))
plt.imshow(convolved_out, cmap='gray')
plt.colorbar()
```

```
Maximum value is: 255.0
```

[4]:
```
<matplotlib.colorbar.Colorbar at 0x7f9500be5610>
```



Normalized Cross Correlation

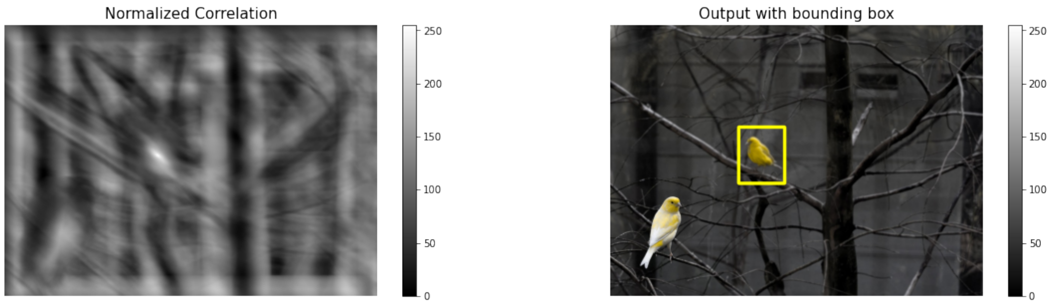
[5]:
```
birds = cv2.imread("birds1.jpeg")
template = cv2.imread("template.jpeg")

birds_gray = cv2.cvtColor(birds, cv2.COLOR_BGR2GRAY)
template_gray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
```

[6]:
```
def norm_corr(image, kernel):
    template = np.asarray(kernel, dtype=np.float64)
    template = template - np.mean(template)
    template_norm = math.sqrt(np.sum(np.square(template)))
    template = template / template_norm

    mean_filter = np.ones(np.shape(template))

    image = np.asarray(image, dtype=np.float64)
    image_squared = np.square(image)

    # compute sums of values and sums of values squared under template
    image_sum = sig.correlate2d(image, mean_filter, 'same')
    image_squared_sum = sig.correlate2d(image_squared, mean_filter, 'same')

    numer = sig.correlate2d(image, template, 'same')
    denom = np.sqrt(image_squared_sum - np.square(image_sum)/np.size(template))
```

```
    tol = np.sqrt(np.finfo(denom.dtype).eps)
    normalCorr = np.where(denom < tol, 0, numer/denom)

    normalCorr = np.where(np.abs(nxcorr-1.) > np.sqrt(np.finfo(nxcorr.dtype).
 ↪eps),nxcorr,0)

    #Normalizing
    min_val = np.min(nxcorr)
    max_val = np.max(nxcorr)
    nxcorr = (nxcorr - min_val)
    nxcorr = (nxcorr / (max_val - min_val)) * 255.0

    return nxcorr

def get_max_loc(image):
    return np.unravel_index(np.argmax(image, axis=None), image.shape)

def draw_rectangle(image, start_index, end_index):
    rect = cv2.rectangle(np.copy(image), start_index, end_index, (255,255,0), 8␣
 ↪)
    return rect
```

[7]:
```
norm_corr_image = norm_corr(birds_gray, template_gray)
max_loc_index = get_max_loc(norm_corr_image)
print([max_loc_index])

start_index_x = int(max_loc_index[1] - template_gray.shape[0]/2) -10
start_index_y = int(max_loc_index[0] - template_gray.shape[1]/2) - 20
print(start_index_x, start_index_y)
end_index_x = int(max_loc_index[1] + template_gray.shape[0]/2)
end_index_y = int(max_loc_index[0] + template_gray.shape[1]/2) + 20

rect_out = draw_rectangle(birds_color, (start_index_x, start_index_y),␣
 ↪(end_index_x, end_index_y))

images = [norm_corr_image, rect_out]
names = ["Normalized Correlation", "Output with bounding box"]
plot_subplots(images, names, 1, 2, True, colorbar=True, size=(20, 5))
```

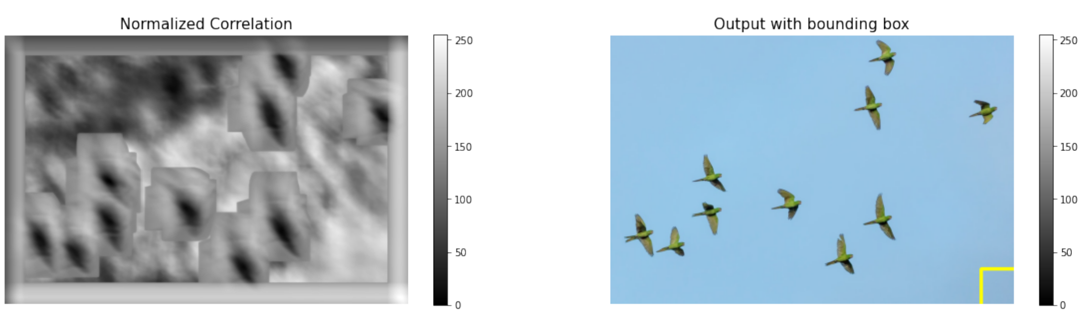
```
[(345, 407)]
341 270
```



[8]:
```python
birds = cv2.imread("birds2.jpeg")
template = cv2.imread("template.jpeg")
birds_color = cv2.cvtColor(birds, cv2.COLOR_BGR2RGB) #Used for plotting

birds_gray = cv2.cvtColor(birds, cv2.COLOR_BGR2GRAY)
template_gray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
```

[9]:
```python
norm_corr_image = norm_corr(birds_gray, template_gray)
max_loc_index = get_max_loc(norm_corr_image)
print([max_loc_index])

start_index_x = int(max_loc_index[1] - template_gray.shape[0]/2) -10
start_index_y = int(max_loc_index[0] - template_gray.shape[1]/2) - 20
print(start_index_x, start_index_y)
end_index_x = int(max_loc_index[1] + template_gray.shape[0]/2)
end_index_y = int(max_loc_index[0] + template_gray.shape[1]/2) + 20

rect_out = draw_rectangle(birds_color, (start_index_x, start_index_y), (end_index_x, end_index_y))

images = [norm_corr_image, rect_out]
names = ["Normalized Correlation", "Output with bounding box"]
plot_subplots(images, names, 1, 2, True, colorbar=True, size=(20, 5))
```

```
[(684, 1031)]
965 609
```



As it can be seen in the above image, there is no matching bird in the original image and hence the bound box is somewhere near the corner of the image. Matched location varies depending on the type of padding done to the image

Problem 2: Hough Transform

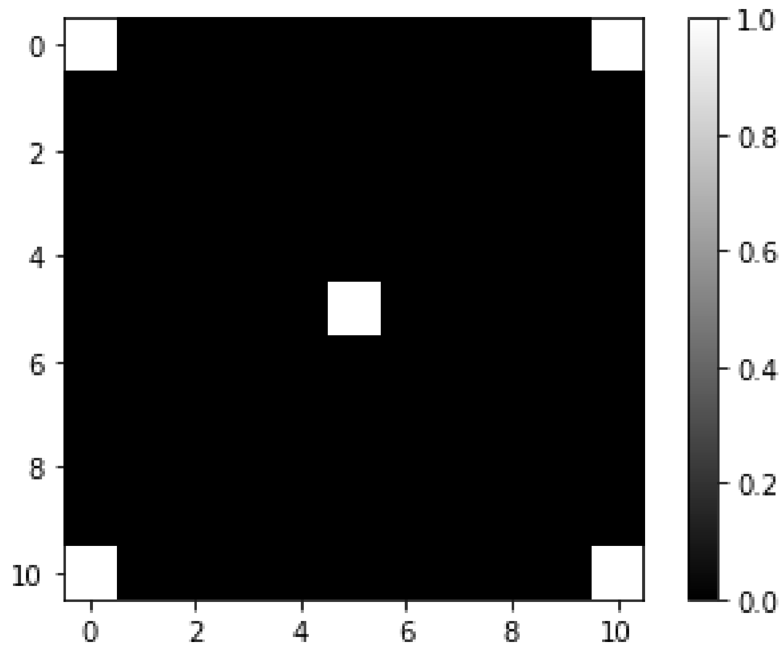
[10]:
```
def Hough_Transform(image):

    rows, cols = image.shape
    thetas = np.deg2rad(np.arange(-90.0, 90.0))
    lin_spacing = len(thetas)

    rho_max = int(np.ceil(np.sqrt(rows * rows + cols * cols)) )
    rhos = np.linspace(-rho_max, rho_max, rho_max * 2)

    lut_cos = np.cos(thetas)
    lut_sin = np.sin(thetas)

    accumulator = np.zeros((2 * rho_max, lin_spacing), dtype=np.uint64)
    # Get X and Y indices where the image is non-zero
    # Vectorized for running faster
    y_idxs, x_idxs = np.nonzero(image)
    for i in range(len(x_idxs)):
        x = x_idxs[i]
        y = y_idxs[i]
        for theta in range(lin_spacing):
            rho = round(x * lut_cos[theta] + y * lut_sin[theta]) + rho_max
            accumulator[rho, theta] += 1
    return accumulator, thetas, rhos
```

[19]:
```
def draw_line(hough_space, image, rhos, thetas, threshold, normalize=False):
    ret_image = np.copy(image)
    rows, cols = image.shape
    x1, x2, y1, y2 = [], [],[],[]
    for i in range(hough_space.shape[0]):
        for j in range(hough_space.shape[1]):
            if hough_space[i, j] > threshold:
                rho = rhos[i]
                theta = thetas[j]
                a = np.cos(np.deg2rad(theta))
                b = np.sin(np.deg2rad(theta))
                x0 = (a * rho)
                y0 = (b * rho)
                x1.append(round((x0 + 1000 * (-b)))
                y1.append(round((y0 + 1000 * (a)))
```

```
                x2.append(round((x0 - 1000 * (-b))))
                y2.append(round((y0 - 1000 * (a))))

    x1_arr = np.array(x1)
    x2_arr = np.array(x2)
    y1_arr = np.array(y1)
    y2_arr = np.array(y2)

    if normalize == True:
        #Normalize x and y axis within the range of original image
        x1_arr = (x1_arr - np.min(x1_arr))/(np.max(x1_arr) - np.min(x1_arr)) *␣
 ↪(image.shape[1] - 1)
        x2_arr = (x2_arr - np.min(x2_arr))/(np.max(x2_arr) - np.min(x2_arr)) *␣
 ↪(image.shape[1]-1)
        y1_arr = y1_arr/100
        y2_arr = (y2_arr - np.min(y2_arr))/(np.max(y2_arr) - np.
 ↪min(y2_arr)+1e-15) * (image.shape[0])
#     print(np.min(y2_arr))
#     print(np.max(y2_arr))
    plt.plot([(x1_arr), (x2_arr)], [(y1_arr), (y2_arr)], color='red',␣
 ↪linestyle='-', linewidth=2)
    #for i in range(x1_arr.shape[0]):
        #cv2.line(image, (int(x1_arr[i]), int(y1_arr[i])), (int(x2_arr[i]),␣
 ↪int(y2_arr[i])), (255, 0,0), thickness=5)
```

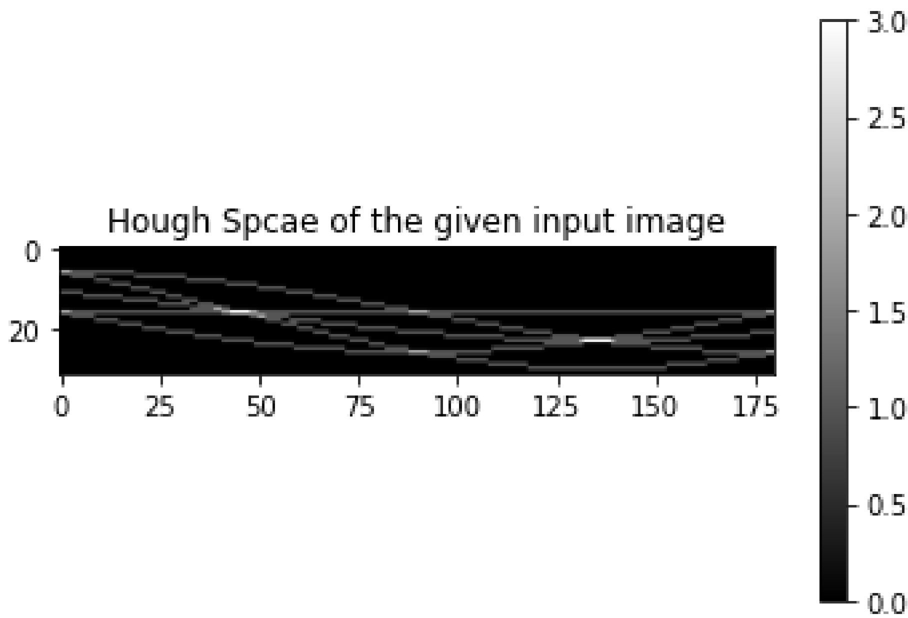
[20]:
```
rows, cols = 11, 11
image = np.zeros((rows, cols))
image[0, 0] = 1
image[0, 10] = 1
image[5, 5] = 1
image[10, 0], image[10,10] = 1, 1
plt.imshow(image, cmap='gray')
plt.colorbar()
```

[20]: <matplotlib.colorbar.Colorbar at 0x7f95143fba00>



[21]:
```
acc, theta, rho = Hough_Transform(image)
#plt.figure(figsize=(10, 20))

plt.imshow(acc, cmap='gray')
plt.title("Hough Spcae of the given input image")
plt.colorbar()
```

[21]: `<matplotlib.colorbar.Colorbar at 0x7f9501f1fa00>`

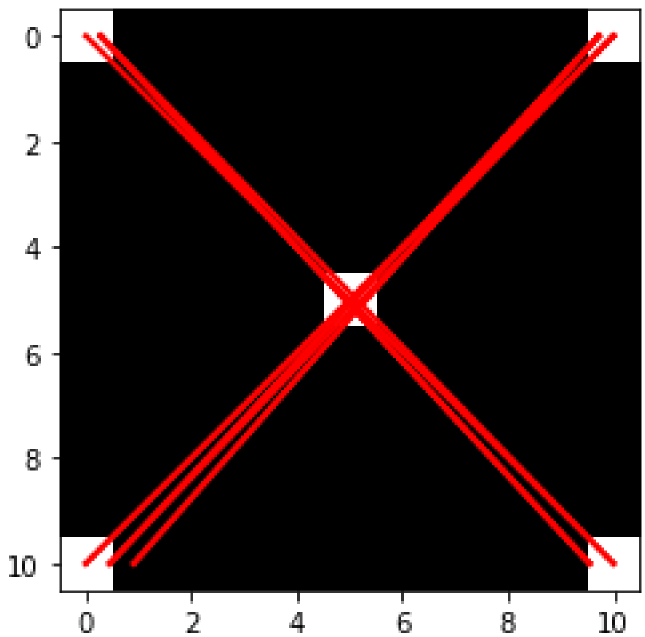


[22]:
```
acc.shape
```

[22]:
```
(32, 180)
```

[23]:
```
draw_line(acc, image, rho, theta, 2, True)
plt.imshow(image, cmap='gray')
```

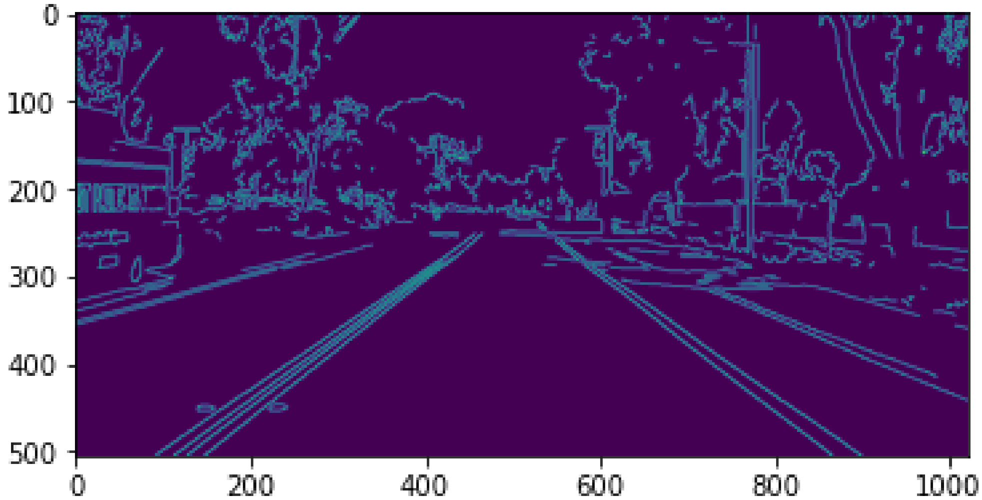
[23]:
```
<matplotlib.image.AxesImage at 0x7f94f09cf430>
```



[24]:
```
image = cv2.imread("lane.png")
edge_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edge_image = cv2.GaussianBlur(edge_image, (3, 3), 1)
edge_image = cv2.Canny(edge_image, 150, 220)
```

[25]:
```
plt.imshow(edge_image)
```

[25]:
```
<matplotlib.image.AxesImage at 0x7f9501f7e2e0>
```

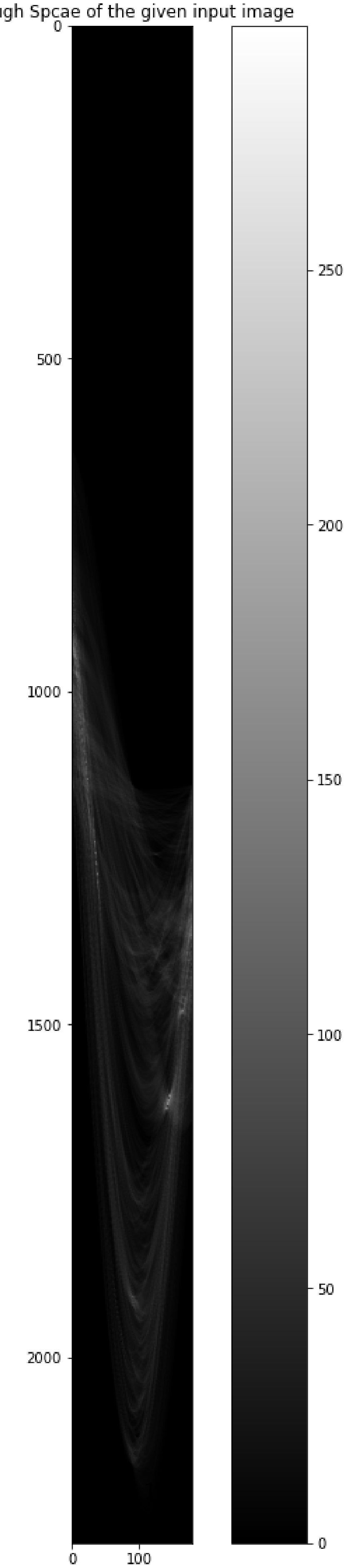


[27]:
```
acc, theta, rho = Hough_Transform(edge_image)
```

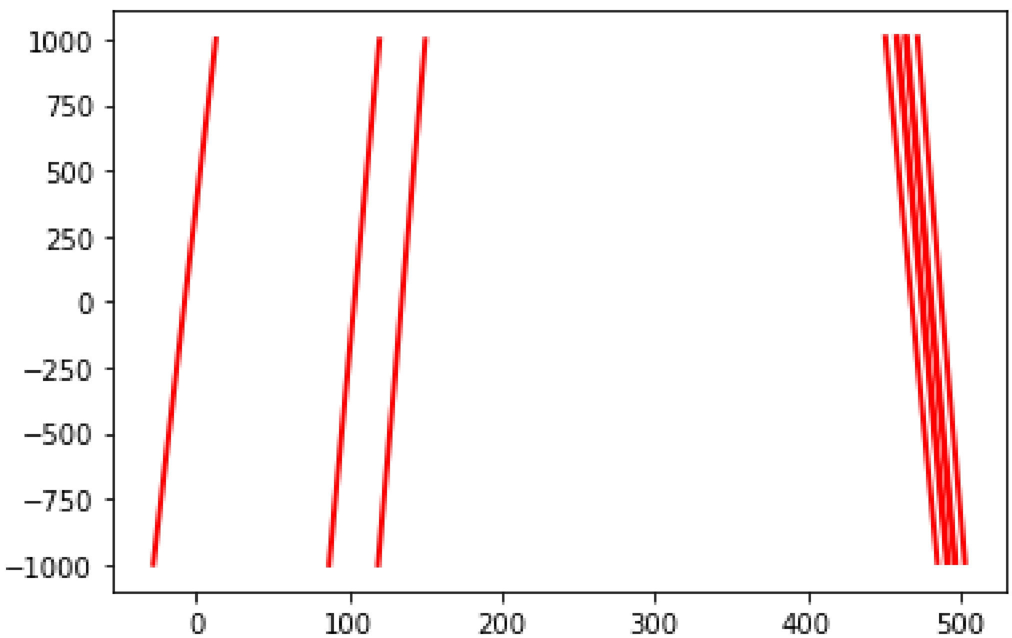
[28]:
```
plt.figure(figsize=(10, 20))
plt.imshow(acc, cmap='gray')
plt.title("Hough Spcae of the given input image")
plt.colorbar()
```

[28]: <matplotlib.colorbar.Colorbar at 0x7f94c883b8e0>

Hough Spcae of the given input image



[29]:
```
draw_line(acc, edge_image, rho, theta, 0.75*np.max(acc))
#plt.imshow(image,cmap='gray')
```



Problem 3: K-Means Segmentation

[30]:
```
import cv2
import math
from matplotlib import pyplot as plt
```

[133]:
```
def distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))

def createDataset(im):
    im_pixels = im.shape[0] * im.shape[1]
    M = 3
    features = im.reshape((im_pixels, M))
    features = np.float32(features)
    return features

def mapValues(im, idx, centers):
    updated_image_values = np.copy(im)
    print(updated_image_values.shape)
    for i in range(0, 7):
        indices_current_cluster = np.where(idx == i)[0]
```

```
        #print(indices_current_cluster)
        updated_image_values[indices_current_cluster] = centers[i]

    im_seg = updated_image_values.reshape(720,1280,3)
    return im_seg
```

[134]:
```
# Helper function to intialize random centers
# Note: I have used a Gaussian initialization, there are other methods as well
def initialize_centers(points, clusters):
    row, col = points.shape
    retArr = np.empty([clusters, col])
    for number in range(clusters):
        randIndex = np.random.randint(row)
        retArr[number] = points[randIndex]

    return retArr


# Helper method to check the loss in every iteration
# This is one of the criterion used for exit condition
def loss_function(centers, cluster_idx, points):
    dists = eucledian_distance(points, centers)
    loss = 0.0
    N, D = points.shape
    for i in range(N):
        loss = loss + np.square(dists[i][cluster_idx[i]])
    return loss

# Helper function to calculate the Eucledian distance between 2 points
def eucledian_distance(x, y):
    x_sum = np.sum(np.square(x),axis=1);
    y_sum = np.sum(np.square(y),axis=1);
    dot_product = np.dot(x, y.T);
    dists = np.sqrt(abs(x_sum[:, np.newaxis] + y_sum-2*dot_product))
    return dists

# Moves the points to new centers
def update_assignment(centers, points):
    row, col = points.shape
    cluster_idx = np.empty([row])
    distances = eucledian_distance(points, centers)
    cluster_idx = np.argmin(distances, axis=1)

    return cluster_idx

# Updates the old centers to new centers
def update_centers(old_centers, cluster_idx, points):
```

```
    N = old_centers.shape[0]
    new_centers = np.empty(old_centers.shape)
    for i in range(N):
        new_centers[i] = np.mean(points[cluster_idx == i], axis = 0)
    return new_centers

# Actual function
def kMeansCluster(features, centers, K=7, print_loss=True):
    max_iterations = 100
    abs_tol=1e-16
    for i in range(max_iterations):
        cluster_idx = update_assignment(centers, features)
        centers = update_centers(centers, cluster_idx, features)
        loss = loss_function(centers, cluster_idx, features)
        K = centers.shape[0]
        if True:
            diff = np.abs(prev_loss - loss)
            if diff < abs_tol:
                break
        prev_loss = loss
        if print_loss:
            print('Loop %d, Loss: %.4f' % (i, loss))
    print(loss)
    return cluster_idx, centers
```

[135]:
```
tower = cv2.imread("white-tower.png")
tower = cv2.cvtColor(tower, cv2.COLOR_BGR2RGB)
#print(tower.shape[0] * tower.shape[1])
# plt.figure(figsize=(10, 5))
# plt.imshow(tower)
```

[136]:
```
feature_set = createDataset(tower)
#print(feature_set.shape)
```

[137]:
```
#Initialize random centers
centers = initialize_centers(feature_set, 7)
idx, centers = kMeansCluster(feature_set, centers, print_loss=False)
```

```
452644459.49086976
```

[143]:
```
print(centers.shape)
seg_out = mapValues(feature_set, idx, centers)
seg_out = (seg_out - np.min(seg_out)) / (np.max(seg_out) - np.min(seg_out)) *␣
 ↪255.0
images = [tower, seg_out.astype(int)]
names = ["Original Image", "Segmented Image"]
plot_subplots(images, names, 1, 2, True, colorbar=False, size=(20, 5))
```

```
(7, 3)
(921600, 3)
```

2.0.2 Note: I have added multiple helper functions along with 3 expected functions in the questions. The function mapValues in the question expects only image and idx as its arguments, however, centers are required as well.

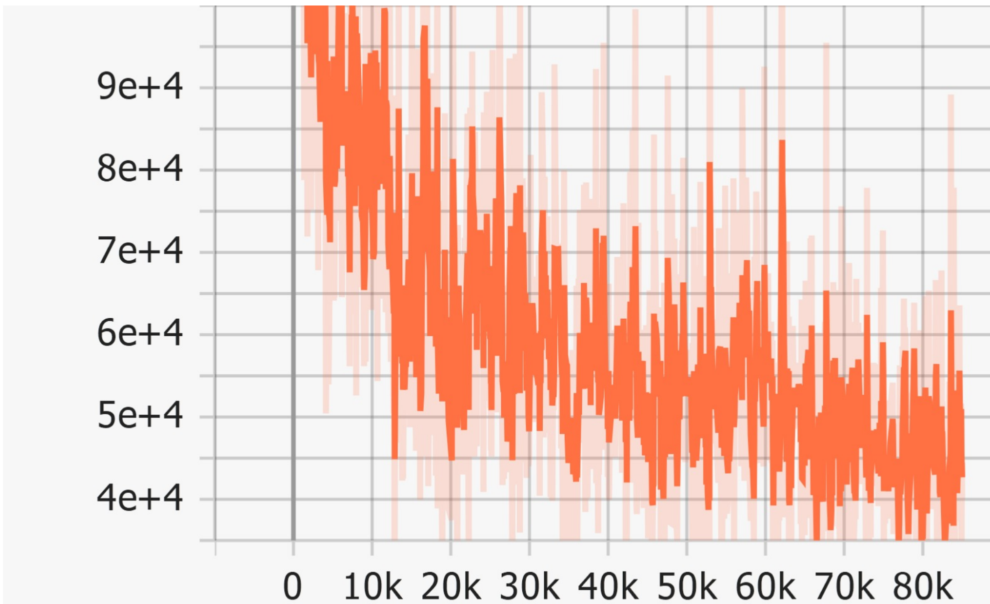
Problem 4: Semantic Segmentation

2.1 1. Fully convolutional network is used for semantic segmentation. It contains convolution, max-pool layers (includes ReLu and dropouts as well).

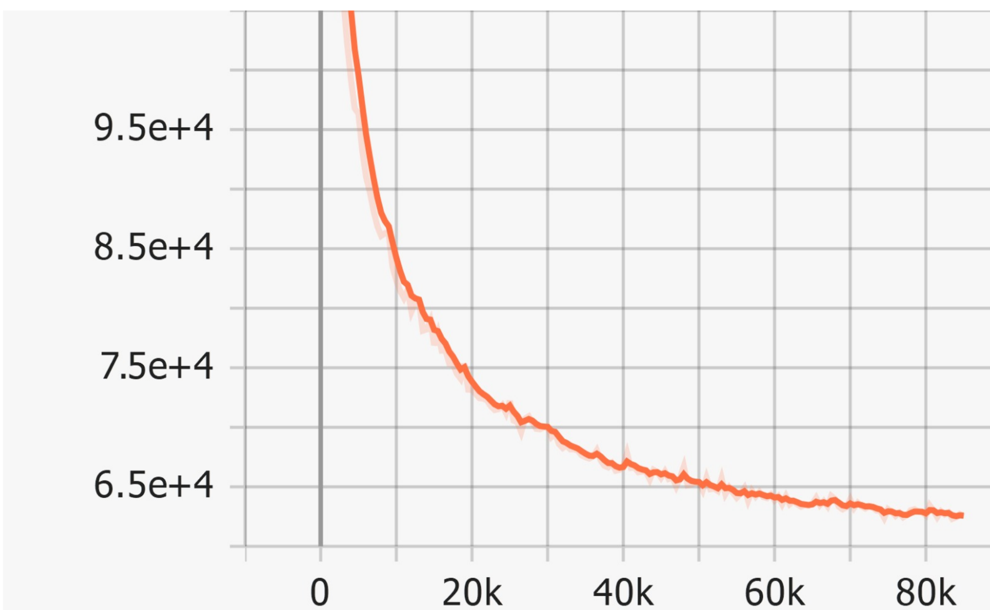
```
- The network contains 5 convolutional blocks each with different parameters.
    - First convolution block contains 2 convolution operations, 2 ReLu operations and 1 maxpoo
    - Second convolution block contains the same number of operations as the first. Input chann
    - Third convolution block contains 3 convolution operations, 2 ReLu and 1 maxpooling operat
    - Fourth convolution block contains the same number of operations as the third convolution 
    - Fifth convolution block contains 3 convolution, 2 ReLu and 1 maxpool operation. Input and
    - Last layer is a classification layer that contains 3 convolution operations. The first co
    - Last layers are upsampling layers which are 3 in number and use bilinear upsampling opera
```

2.2 2. The model is trained from scratch. Batch size of 2 is used (tried with 4) but due to GPU constraints, max batch size could only be 2. Training took about 12 hours.

2.3 3. Training and validation curves are as shown below:



Training Curve



Validation Curve

2.4 4. Metrics used by the original paper are:

```
- Validation accuracy
```

```
- Mean Intersection over Union (IoU)
```

Class 0, 2, 8, 10, 13 has good accuracy. Class 0 has the best accuracy of all. Classes with not so good accuracies are 3, 4, 5, 6, 7. Worst accuracies are obtained for 12 and 17.

2.5 5.

2.6 6. Ouput of the streets: Output does look reasonable, could have been slightly better in terms of IoU a different model were used.

San Diego Street

Segmented Street Output

2.7 7. Options to improve segmentation output

- Better model architecture such as U-Net can be used for better output. U-Net architecture preserves spatial connections by using skip connections between encoding and decoding

layers.
- Adding more data to the training set will help with model learning better features and generalizing well to streets of different cities as well.
- Using optimizers such as Adam will help with the learning process and hopefully yield better mIoU.

Problem 5: Tritonogram

[31]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def Tritonogram_Filter(alpha, images_path_list=None, convertMatplotlib=False):
    # Can only add 2 images
    if images_path_list == None or len(images_path_list) > 2:
        return -1

    images = []
    # Read all the images in the imageList
    for i in range(len(images_path_list)):
        img = cv2.imread(images_path_list[i])
        img_rgb = img
        #Condition to check if cv2.imshow or plt.imshow is used as cv2 show␣
↪function internally handles BGRformat
        if convertMatplotlib == True:
            img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        images.append(img_rgb)

    #Calculating foreground and background shapes for slicing
    # Due to lack of time, I am only considering the left-smallest images size␣
↪and top-smallest image size slice
    # It can be modified to find the most interesting feature in the image and␣
↪consider that part
    foreground, background = images[0].copy(), images[0].copy()

    foreground_height = foreground.shape[0]
    foreground_width = foreground.shape[1]
    #Add the 2 images as weighted sum
    beta = 1 - alpha
    output = cv2.addWeighted(images[0], alpha, images[1][:foreground_height,:
↪foreground_width,:], beta, 0)

    return output, images
```

[42]:
```
image_path1 = "Mandrill.tiff"
image_path2 = "ucsd-trident.jpeg"
```

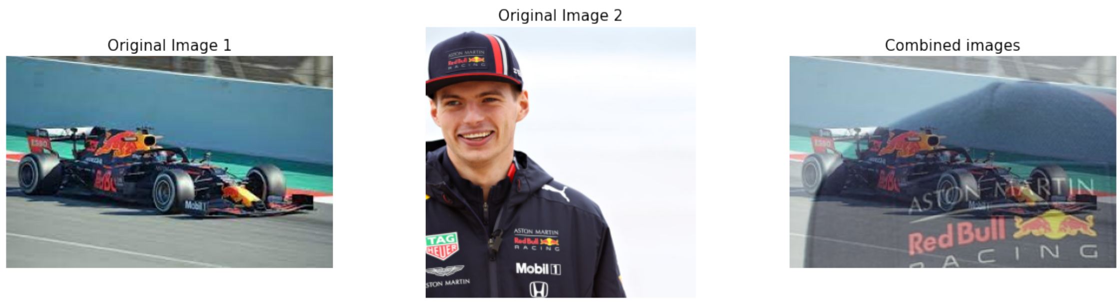
```
images_list = []
images_list.append(image_path1)
images_list.append(image_path2)

#Changing alpha changes the transparency
triton_out, original_images = Tritonogram_Filter(0.3, images_list,␣
 ↪convertMatplotlib=True)

image_path1 = "F1.jpeg"
image_path2 = "max-f1.jpeg"
images_list = []
images_list.append(image_path1)
images_list.append(image_path2)

#Changing alpha changes the transparency
max_out, original_f1 = Tritonogram_Filter(0.5, images_list,␣
 ↪convertMatplotlib=True)

image_path1 = "giesel.jpeg"
image_path2 = "ucsd-logo.png"
images_list = []
images_list.append(image_path1)
images_list.append(image_path2)

#Changing alpha changes the transparency
ucsd_out, original_lib = Tritonogram_Filter(0.8, images_list,␣
 ↪convertMatplotlib=True)
```

[43]:
```
fig, axs = plt.subplots(1, 3,figsize=(20, 5))
ax1= fig.add_subplot(1,3,1)
ax1.title.set_text("Original Image 1")
ax1.title.set_size(15)
ax1.axis('off')
ax = ax1.imshow(original_images[0], cmap='gray')
axs[0].axis('off')

ax1= fig.add_subplot(1,3,2)
ax1.title.set_text("Original Image 2")
ax1.title.set_size(15)
ax1.axis('off')
ax = ax1.imshow(original_images[1], cmap='gray')
axs[1].axis('off')

ax1= fig.add_subplot(1,3,3)
ax1.title.set_text("Combined images")
ax1.title.set_size(15)
ax1.axis('off')
```

```python
ax = ax1.imshow(triton_out)
axs[2].axis('off')

# F1 output
fig, axs = plt.subplots(1, 3,figsize=(20, 5))
ax1= fig.add_subplot(1,3,1)
ax1.title.set_text("Original Image 1")
ax1.title.set_size(15)
ax1.axis('off')
ax = ax1.imshow(original_f1[0], cmap='gray')
axs[0].axis('off')

ax1= fig.add_subplot(1,3,2)
ax1.title.set_text("Original Image 2")
ax1.title.set_size(15)
ax1.axis('off')
ax = ax1.imshow(original_f1[1], cmap='gray')
axs[1].axis('off')

ax1= fig.add_subplot(1,3,3)
ax1.title.set_text("Combined images")
ax1.title.set_size(15)
ax1.axis('off')
ax = ax1.imshow(max_out)
axs[2].axis('off')

fig, axs = plt.subplots(1, 3,figsize=(20, 5))
ax1= fig.add_subplot(1,3,1)
ax1.title.set_text("Original Image 1")
ax1.title.set_size(15)
ax1.axis('off')
ax = ax1.imshow(original_lib[0], cmap='gray')
axs[0].axis('off')

ax1= fig.add_subplot(1,3,2)
ax1.title.set_text("Original Image 2")
ax1.title.set_size(15)
ax1.axis('off')
ax = ax1.imshow(original_lib[1], cmap='gray')
axs[1].axis('off')

ax1= fig.add_subplot(1,3,3)
ax1.title.set_text("Combined images")
ax1.title.set_size(15)
ax1.axis('off')
ax = ax1.imshow(ucsd_out)
axs[2].axis('off')
```

[43]: (0.0, 1.0, 0.0, 1.0)


Original Image 1
Original Image 2
Combined images


Original Image 1
Original Image 2
Combined images
ASTON MARTIN
Red Bull
RACING


Original Image 1
Original Image 2
Combined images
UNIVERSITY OF CALIFORNIA
LET THERE BE LIGHT
SAN DIEGO

[ ]: