NCC

**Import necessary libraries:**

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from skimage.color import rgb2gray

import cv2 as cv

import numpy as np

import math

from PIL import Image

import glob

import os

import skimage

from os import listdir

from os.path import join, isfile

from skimage import morphology

from skimage import measure,color

from skimage import io, data

from numpy.linalg import eig

from scipy import ndimage, misc

from scipy.ndimage import median\_filter

import matplotlib.patches as patches

## Image 1

**Read and display searchImage1 and template image**

searchImage1 = skimage.io.imread('input1.png')

# Generating a template

img1 = Image.open(r"input1.png")

left = 421

top = 191

right =609

bottom = 385

img2 = img1.crop((left, top, right, bottom))

img2.save('templateImage.png')

# Read template image

templateImage = skimage.io.imread('templateImage.png')

# Display searchImage1 and templateImage

plt.subplot(1,1,1)

plt.imshow(searchImage1)

plt.show()

#print(searchImage1.shape)

plt.subplot(1,1,1)

plt.imshow(templateImage)

plt.show()

#print(templateImage.shape)

**Create a function to generate Image Pyramid and obtain ideal size for images**

def imagePyramid(image1, image2):

image1Level1 = image1[0::2,0::2]

image1Level2 = image1Level1[0::2,0::2]

image1Level3 = image1Level2[0::2,0::2]

image1Level4 = image1Level3[0::2,0::2]

image2Level1 = image2[0::2,0::2]

image2Level2 = image2Level1[0::2,0::2]

image2Level3 = image2Level2[0::2,0::2]

image2Level4 = image2Level3[0::2,0::2]

if(image1.size > 4000000):

image1Out = image1Level4

image2Out = image2Level2

elif(image1.size > 2000000):

image1Out = image1Level3

image2Out = image2Level2

elif(image1.size <= 2000000 and image1.size > 1000000):

image1Out = image1Level2

image2Out = image2Level2

elif(image1.size <= 1000000):

image1Out = image1Level1

image2Out = image2Level1

return image1Out, image2Out

### Display ideal search and template image from the pyramid

searchImage1, templateImage = imagePyramid(searchImage1, templateImage)

plt.subplot(1,1,1)

plt.imshow(searchImage1)

plt.show()

#print(searchImage1Downscaled.shape)

plt.subplot(1,1,1)

plt.imshow(templateImage)

plt.show()

#print(templateImageDownscaled.shape)

# We will use searchImage1Downscaled and templateImageDownscaled for NCC calculations

### Calculate NCC of patches

def compute\_NCC(temp2,temp,rtempmean,gtempmean,btempmean,rtempstd,gtempstd,btempstd):

rpatmean = np.mean(temp2[:,:,0])

gpatmean = np.mean(temp2[:,:,1])

bpatmean = np.mean(temp2[:,:,2])

rpatstd = np.std(temp2[:,:,0])

gpatstd = np.std(temp2[:,:,1])

bpatstd = np.std(temp2[:,:,2])

n1 = 0

n2 = 0

n3 = 0

tempr,tempc,ch=temp2.shape

for k in range(tempr):

for l in range(tempc):

n1 = n1 + ((temp2[k,l,0]-rpatmean)\*(temp[k,l,0]-rtempmean)/(rpatstd\*rtempstd));

n2 = n2 + ((temp2[k,l,1]-gpatmean)\*(temp[k,l,1]-gtempmean)/(gpatstd\*gtempstd));

n3 = n3 + ((temp2[k,l,2]-bpatmean)\*(temp[k,l,2]-btempmean)/(bpatstd\*btempstd));

return [n1,n2,n3]

### Calculate all possible NCC windows

def calculateNCC(searchImage,templateImage):

nrowtemp,ncoltemp,ch=templateImage.shape

nrows,ncols,ch=searchImage.shape

nc1=np.zeros((abs(-nrowtemp+nrows),abs(-ncoltemp+ncols)))

nc2=np.zeros((abs(-nrowtemp+nrows),abs(-ncoltemp+ncols)))

nc3=np.zeros((nrowtemp,ncoltemp))

rtempmean = np.mean(searchImage[:,:,0])

gtempmean = np.mean(searchImage[:,:,1])

btempmean = np.mean(searchImage[:,:,2])

rtempstd = np.std(searchImage[:,:,0])

gtempstd = np.std(searchImage[:,:,1])

btempstd = np.std(searchImage[:,:,2])

for i in range(nrows//2,nrowtemp-nrows//2):

for j in range(ncols//2,ncoltemp-ncols//2):

temp2=searchImage[i-nrowtemp//2:i+nrowtemp//2+1,j-ncoltemp//2:j+ncoltemp//2+1,:]

nc1[i-nrows//2 - 1,j-ncols//2-1],nc2[i-nrows//2 - 1,j-ncols//2-1],nc3[i-nrows//2 - 1,j-ncols//2-1]=compute\_NCC(temp2,searchImage,rtempmean,gtempmean,btempmean,rtempstd,gtempstd,btempstd)

NCC=np.zeros((-nrowtemp+nrows,-ncoltemp+ncols))

for i in range(nrowtemp-nrows):

for j in range(ncoltemp-ncols):

NCC[i,j]=1/(nrows\*ncols)\*(nc1[i,j]+nc2[i,j]+nc3[i,j])

NCC=NCC/3

return NCC

NCC=calculateNCC(searchImage,templateImage)

array = NCC.flatten()

flattenIndex = np.argmax(array)

y = int(flattenIndex / (searchImage.shape[1] - templateImage.shape[1]))

x = flattenIndex % (searchImage.shape[0] - templateImage.shape[0])

print(y,x)

### Display output image

# Find coordinates of maximumum similarity in original search image

originalX = x

originalY = y

fig,ax = plt.subplots()

ax.imshow(searchImage1)

rect = patches.Rectangle((originalX,originalY),templateImage.shape[1]\*1.4,templateImage.shape[0]\*1.4,linewidth=1,edgecolor='r',facecolor='none')

ax.add\_patch(rect)

plt.show()

### Output vs Ground Truth

# Ground truth - Blue

# Output - Red

fig,ax = plt.subplots()

ax.imshow(searchImage1)

rect1 = patches.Rectangle((originalX,originalY),templateImage.shape[1]\*1.4,templateImage.shape[0]\*1.4,linewidth=1,edgecolor='r',facecolor='none')

rect2 = patches.Rectangle((350,120),templateImage.shape[1]\*1.4,templateImage.shape[0]\*1.4,linewidth=1,edgecolor='b',facecolor='none')

ax.add\_patch(rect1)

ax.add\_patch(rect2)

plt.savefig('IoU1\_NCC.png')

plt.axis('off')

plt.show()

### Censor infant face by applying Gaussian Blurring on the detected subregion

# Read in image

image = searchImage1.copy()

# Create ROI coordinates

topLeft = math.floor(originalX), math.floor(originalY)

bottomRight = math.floor(originalX + (templateImage.shape[1]\*1.4)), math.floor(originalY + (templateImage.shape[0]\*1.4))

x, y = topLeft[0], topLeft[1]

w, h = bottomRight[0] - topLeft[0], bottomRight[1] - topLeft[1]

# Grab ROI with Numpy slicing and blur

ROI = image[y:y+h, x:x+w]

blur = cv.GaussianBlur(ROI, (81,81), 0)

# Insert ROI back into image

image[y:y+h, x:x+w] = blur

plt.subplot(1,1,1)

plt.imshow(image)

plt.axis('off')

plt.savefig('blur1NCC.png')

### Create a function to calculate censor accuracy using intersection over union

def bb\_intersection\_over\_union(patchA, patchB):

boxA = np.zeros(4)

boxA[0] = patchA.xy[0]

boxA[1] = patchA.xy[1]

boxA[2] = patchA.xy[0] + patches.Rectangle.get\_width(patchA)

boxA[3] = patchA.xy[1] + patches.Rectangle.get\_height(patchA)

boxB = np.zeros(4)

boxB[0] = patchB.xy[0]

boxB[1] = patchB.xy[1]

boxB[2] = patchB.xy[0] + patches.Rectangle.get\_width(patchB)

boxB[3] = patchB.xy[1] + patches.Rectangle.get\_height(patchB)

# determine the (x, y)-coordinates of the intersection rectangle

xA = max(boxA[0], boxB[0])

yA = max(boxA[1], boxB[1])

xB = min(boxA[2], boxB[2])

yB = min(boxA[3], boxB[3])

# compute the area of intersection rectangle

interArea = max(0, xB - xA + 1) \* max(0, yB - yA + 1)

# compute the area of both the prediction and ground-truth

# rectangles

boxAArea = (boxA[2] - boxA[0] + 1) \* (boxA[3] - boxA[1] + 1)

boxBArea = (boxB[2] - boxB[0] + 1) \* (boxB[3] - boxB[1] + 1)

# compute the intersection over union by taking the intersection

# area and dividing it by the sum of prediction + ground-truth

# areas - the interesection area

iou = interArea / float(boxAArea + boxBArea - interArea)

# return the intersection over union value

return iou

### Censor accuracy calculation

censorAccuracy1 = bb\_intersection\_over\_union(rect1, rect2)

print("Censor accuracy in Image1 is: " + str(censorAccuracy1\*100) + "%")

## Image 2

### Read and display searchImage2 and templateImage

searchImage2 = skimage.io.imread('input2.png')

# Generating a template

img1 = Image.open(r"input1.png")

left = 421

top = 191

right =609

bottom = 385

img2 = img1.crop((left, top, right, bottom))

img2.save('templateImage.png')

# Read template image

templateImage = skimage.io.imread('templateImage.png')

# Display searchImage2 and templateImage

plt.subplot(1,1,1)

plt.imshow(searchImage2)

plt.show()

#print(searchImage2.shape)

plt.subplot(1,1,1)

plt.imshow(templateImage)

plt.show()

#print(templateImage.shape)

### Display ideal search and template images from the pyramid

searchImage2Downscaled, templateImageDownscaled = imagePyramid(searchImage2, templateImage)

plt.subplot(1,1,1)

plt.imshow(searchImage2Downscaled)

plt.show()

#print(searchImage2Downscaled.shape)

plt.subplot(1,1,1)

plt.imshow(templateImageDownscaled)

plt.show()

#print(templateImageDownscaled.shape)

# We will use searchImage2Downscaled and templateImageDownscaled for NCC calculations

### Calculate NCC for Image 2

NCC=calculateNCC(searchImage2Downscaled,templateImageDownscaled)

array = NCC.flatten()

flattenIndex = np.argmax(array)

row = int(flattenIndex / (searchImage.shape[1] - templateImage.shape[1]))

column = flattenIndex % (searchImage.shape[0] - templateImage.shape[0])

print(row,column)

# Find coordinates of maximumum similarity in original search image

originalX = row

originalY = column

print(originalX,originalY)

fig,ax = plt.subplots()

ax.imshow(searchImage2Downscaled)

rect1 = patches.Rectangle((originalY,originalX),templateImageDownscaled.shape[0]\*0.8,templateImageDownscaled.shape[1]\*0.8,linewidth=1,edgecolor='r',facecolor='none')

ax.add\_patch(rect1)

plt.show()

### Display output vs ground truth

# Ground truth - Blue

# Output - Red

fig,ax = plt.subplots()

ax.imshow(searchImage2)

rect1 = patches.Rectangle((615,40),templateImage.shape[1]\*0.8,templateImage.shape[0]\*0.8,linewidth=1,edgecolor='b',facecolor='none')

rect2 = patches.Rectangle((originalX,originalY),templateImage.shape[1]\*0.8,templateImage.shape[0]\*0.8,linewidth=1,edgecolor='r',facecolor='none')

ax.add\_patch(rect2)

ax.add\_patch(rect1)

plt.axis('off')

plt.savefig('IoU2\_NCC.png')

plt.show()

### Censor infant face by applying Gaussian Blurring on the detected subregion

### Read in image

image = searchImage2.copy()

# Create ROI coordinates

topLeft = math.floor(originalX), math.floor(originalY)

bottomRight = math.floor(originalX + (templateImageDownscaled.shape[1]\*4)), math.floor(originalY + (templateImageDownscaled.shape[0]\*4))

print(topLeft,bottomRight)

x, y = topLeft[0], topLeft[1]

w, h = bottomRight[0] - topLeft[0], bottomRight[1] - topLeft[1]

# Grab ROI with Numpy slicing and blur

ROI = image[y:y+h, x:x+w]

blur = cv.GaussianBlur(ROI, (81,81), 0)

# Insert ROI back into image

image[y:y+h, x:x+w] = blur

plt.subplot(1,1,1)

plt.imshow(image)

plt.axis('off')

plt.savefig('blur2NCC.png')

plt.show()

### Censor accuracy calculation

censorAccuracy1 = bb\_intersection\_over\_union(rect1, rect2)

print("Censor accuracy in Image1 is: " + str(censorAccuracy1\*100) + "%")

## Image 3

### Read and display searchImage3 and templateImage

searchImage3 = skimage.io.imread('input3.png')

# Generating a template

img1 = Image.open(r"input1.png")

left = 421

top = 191

right =609

bottom = 385

img2 = img1.crop((left, top, right, bottom))

img2.save('templateImage.png')

# Read template image

templateImage = skimage.io.imread('templateImage.png')

# Display searchImage2 and templateImage

plt.subplot(1,1,1)

plt.imshow(searchImage3)

plt.show()

#print(searchImage3.shape)

plt.subplot(1,1,1)

plt.imshow(templateImage)

plt.show()

#print(templateImage.shape)

### Display ideal search and template images from the pyramid

searchImage3Downscaled, templateImageDownscaled = imagePyramid(searchImage3, templateImage)

plt.subplot(1,1,1)

plt.imshow(searchImage3Downscaled)

plt.show()

#print(searchImage3Downscaled.shape)

plt.subplot(1,1,1)

plt.imshow(templateImageDownscaled)

plt.show()

#print(templateImageDownscaled.shape)

# We will use searchImage3Downscaled and templateImageDownscaled for NCC calculations

## **Calculate NCC**

NCC=calculateNCC(searchImage3Downscaled,templateImageDownscaled)

array = NCC.flatten()

flattenIndex = np.argmax(array)

row = int(flattenIndex / (searchImage.shape[1] - templateImage.shape[0]))

column = flattenIndex % (searchImage.shape[1] - templateImage.shape[0])

# Find coordinates of maximumum similarity in original search image

originalX = row

originalY = column

print(originalX,originalY)

fig,ax = plt.subplots()

ax.imshow(searchImage3Downscaled)

rect = patches.Rectangle((originalX,originalY),templateImage.shape[1]\*1.2,templateImage.shape[0]\*1.2,linewidth=1,edgecolor='r',facecolor='none')

ax.add\_patch(rect1)

plt.show()

# Find coordinates of maximumum similarity in original search image

originalX = (originalX / searchImage3Downscaled.shape[1]) \* searchImage3.shape[1]

originalY = (originalY / searchImage3Downscaled.shape[0]) \* searchImage3.shape[0]

print(originalX,originalY)

# Ground truth - Blue

# Output - Red

fig,ax = plt.subplots()

ax.imshow(searchImage3)

rect1 = patches.Rectangle((1200,75),templateImage.shape[1]\*1.2,templateImage.shape[0]\*1.2,linewidth=1,edgecolor='b',facecolor='none')

rect2 = patches.Rectangle((originalX,originalY),templateImage.shape[1]\*1.2,templateImage.shape[0]\*1.2,linewidth=1,edgecolor='r',facecolor='none')

ax.add\_patch(rect2)

ax.add\_patch(rect1)

plt.axis('off')

plt.savefig('IoU3\_NCC.png')

plt.show()

### Read in image

image = searchImage3.copy()

# Create ROI coordinates

topLeft = math.floor(originalX), math.floor(originalY)

bottomRight = math.floor(originalX + (templateImageDownscaled.shape[1]\*4)), math.floor(originalY + (templateImageDownscaled.shape[0]\*4))

x, y = topLeft[0], topLeft[1]

w, h = bottomRight[0] - topLeft[0], bottomRight[1] - topLeft[1]

# Grab ROI with Numpy slicing and blur

ROI = image[y:y+h, x:x+w]

blur = cv.GaussianBlur(ROI, (81,81), 0)

# Insert ROI back into image

image[y:y+h, x:x+w] = blur

plt.subplot(1,1,1)

plt.imshow(image)

plt.axis('off')

plt.savefig('blur3NCC.png')

plt.show()

### Censor accuracy calculation

### censorAccuracy1 = bb\_intersection\_over\_union(rect1, rect2)

### print("Censor accuracy in Image3 is: " + str(censorAccuracy1\*100) + "%")

## Image 4

### Read and display searchImage4 and templateImage

### searchImage4 = skimage.io.imread('input4.png')

### # Generating a template

### img1 = Image.open(r"input1.png")

### left = 421

### top = 191

### right =609

### bottom = 385

### img2 = img1.crop((left, top, right, bottom))

### img2.save('templateImage.png')

### # Read template image

### templateImage = skimage.io.imread('templateImage.png')

### # Display searchImage2 and templateImage

### plt.subplot(1,1,1)

### plt.imshow(searchImage4)

### plt.show()

### #print(searchImage4.shape)

### plt.subplot(1,1,1)

### plt.imshow(templateImage)

### plt.show()

### #print(templateImage.shape)

### Display ideal search and template images from the pyramid

searchImage4Downscaled, templateImageDownscaled = imagePyramid(searchImage4, templateImage)

plt.subplot(1,1,1)

plt.imshow(searchImage4Downscaled)

plt.show()

#print(searchImage4Downscaled.shape)

plt.subplot(1,1,1)

plt.imshow(templateImageDownscaled)

plt.show()

#print(templateImageDownscaled.shape)

# We will use searchImage4Downscaled and templateImageDownscaled for NCC calculations

## **Calculate NCC**

NCC=calculateNCC(searchImage3Downscaled,templateImageDownscaled)

array = NCC.flatten()

flattenIndex = np.argmax(array)

row = int(flattenIndex / (searchImage.shape[1] - templateImage.shape[0]))

column = flattenIndex % (searchImage.shape[1] - templateImage.shape[0])

### Display output image

originalX=row

originalY=column

# Find coordinates of maximumum similarity in original search image

originalX = (originalX / searchImage4Downscaled.shape[1]) \* searchImage4.shape[1]

originalY = (originalY / searchImage4Downscaled.shape[0]) \* searchImage4.shape[0]

fig,ax = plt.subplots()

ax.imshow(searchImage4)

rect = patches.Rectangle((originalX,originalY),templateImageDownscaled.shape[1]\*5,templateImageDownscaled.shape[0]\*5,linewidth=1,edgecolor='r',facecolor='none')

ax.add\_patch(rect)

plt.show()

### Display output vs ground truth

fig,ax = plt.subplots()

ax.imshow(searchImage4)

print(originalX,originalY)

rect1 = patches.Rectangle((originalX,originalY),templateImageDownscaled.shape[1]\*5,templateImageDownscaled.shape[0]\*5,linewidth=1,edgecolor='r',facecolor='none')

rect2 = patches.Rectangle((840,75),templateImageDownscaled.shape[1]\*5,templateImageDownscaled.shape[0]\*5,linewidth=1,edgecolor='b',facecolor='none')

ax.add\_patch(rect1)

ax.add\_patch(rect2)

plt.axis('off')

plt.savefig('IOU\_NCC4.png')

plt.show()

### Censor infant face by applying Gaussian Blurring on the detected subregion

### Read in image

image = searchImage4.copy()

# Create ROI coordinates

topLeft = math.floor(originalX), math.floor(originalY)

bottomRight = math.floor(originalX + (templateImageDownscaled.shape[1]\*5)), math.floor(originalY + (templateImageDownscaled.shape[0]\*5))

x, y = topLeft[0], topLeft[1]

w, h = bottomRight[0] - topLeft[0], bottomRight[1] - topLeft[1]

# Grab ROI with Numpy slicing and blur

ROI = image[y:y+h, x:x+w]

blur = cv.GaussianBlur(ROI, (81,81), 0)

# Insert ROI back into image

image[y:y+h, x:x+w] = blur

plt.subplot(1,1,1)

plt.imshow(image)

plt.axis('off')

plt.savefig('blur4NCC.png')

plt.show()

### Censor accuracy calculation

censorAccuracy1 = bb\_intersection\_over\_union(rect1, rect2)

print("Censor accuracy in Image4 is: " + str(censorAccuracy1\*100) + "%")

Covariance

Image 1

### Read and display searchImage 1 and template image

### searchImage1 = skimage.io.imread('input1.png')

### # Generating a template

### img1 = Image.open(r"input1.png")

### left = 421

### top = 191

### right =609

### bottom = 385

### img2 = img1.crop((left, top, right, bottom))

### img2.save('templateImage.png')

### # Read template image

### templateImage = skimage.io.imread('templateImage.png')

### # Display searchImage1 and templateImage

### plt.subplot(1,1,1)

### plt.imshow(searchImage1)

### plt.show()

### #print(searchImage1.shape)

### plt.subplot(1,1,1)

### plt.imshow(templateImage)

### plt.show()

### #print(templateImage.shape)

### Display ideal search and template image from the pyramid

### searchImage1Downscaled, templateImageDownscaled = imagePyramid(searchImage1, templateImage)

### plt.subplot(1,1,1)

### plt.imshow(searchImage1Downscaled)

### plt.show()

### #print(searchImage1Downscaled.shape)

### plt.subplot(1,1,1)

### plt.imshow(templateImageDownscaled)

### plt.show()

### #print(templateImageDownscaled.shape)

### # We will use searchImage1Downscaled and templateImageDownscaled for covariance calculations

### Calculate covariance matrix of template

### # Creates a 5x5 covariance matrix of template image

### x,y,z = templateImageDownscaled.shape

### featureTemplate = np.zeros((x,y,5))

### for i in range(x):

### for j in range(y):

### xCoordinate = j

### yCoordinate = i

### R = templateImageDownscaled[yCoordinate][xCoordinate][0]

### G = templateImageDownscaled[yCoordinate][xCoordinate][1]

### B = templateImageDownscaled[yCoordinate][xCoordinate][2]

### featureTemplate[i][j] = xCoordinate, yCoordinate, R, G, B

### 

### reshapedFeatureTemplate = featureTemplate.reshape(featureTemplate.shape[0]\*featureTemplate.shape[1],(featureTemplate.shape[2]))

### covMatrixTemplate = np.cov(reshapedFeatureTemplate.transpose(),bias=True)

### Generate list containing all possible overlapping windows

a,b,c = searchImage1Downscaled.shape

x,y,z = templateImageDownscaled.shape

featureList = []

for i in range(a-x):

for j in range(b-y):

window = np.zeros((x,y,5))

for k in range(x):

for l in range(y):

xCoordinate = j + l

yCoordinate = i + k

R = searchImage1Downscaled[yCoordinate][xCoordinate][0]

G = searchImage1Downscaled[yCoordinate][xCoordinate][1]

B = searchImage1Downscaled[yCoordinate][xCoordinate][2]

window[k][l] = xCoordinate, yCoordinate, R, G, B

featureList.append(window)

### Reshape overlapping windows from 3D to 2D

featureListReshaped = []

for matrix in featureList:

reshapedMatrix = matrix.reshape(matrix.shape[0]\*matrix.shape[1],(matrix.shape[2]))

featureListReshaped.append(reshapedMatrix)

### Calculate candidate covariance matrices and store in a list

candidateCovMatrix = []

for matrix in featureListReshaped:

covMatrix = np.cov(matrix.transpose(),bias=True)

candidateCovMatrix.append(covMatrix)

### Riemannian Mannifold Calculation

#Following section creates a list that contains distances of all candidate covariances from model covariance matrix

from scipy.linalg import eigh

distanceMetric = []

alpha = 0

for matrix in candidateCovMatrix:

eigvals = eigh(covMatrixTemplate, matrix, eigvals\_only=True)

for values in eigvals:

if (values != 0):

alpha += (math.log(values))\*\*2

beta = math.sqrt(alpha)

distanceMetric.append(beta)

alpha=0

### Display coordinates of where maximum similarity is found

# Find coordinates of maximumum similarity in downscaled search image

valueOfMaximumSimilarity = min(distanceMetric)

indexOfMaximumSimilarity = distanceMetric.index(valueOfMaximumSimilarity)

coordinatesOfMaximumSimilarity = featureListReshaped[indexOfMaximumSimilarity][0][0:2]

### Display output image

# Find coordinates of maximumum similarity in original search image

originalX = (coordinatesOfMaximumSimilarity[0] / searchImage1Downscaled.shape[1]) \* searchImage1.shape[1]

originalY = (coordinatesOfMaximumSimilarity[1] / searchImage1Downscaled.shape[0]) \* searchImage1.shape[0]

fig,ax = plt.subplots()

ax.imshow(searchImage1)

rect = patches.Rectangle((originalX,originalY),templateImageDownscaled.shape[1]\*8,templateImageDownscaled.shape[0]\*8,linewidth=1,edgecolor='r',facecolor='none')

ax.add\_patch(rect)

plt.show()

### Output vs Ground Truth

# Ground truth - Blue

# Output - Red

fig,ax = plt.subplots()

ax.imshow(searchImage1)

rect1 = patches.Rectangle((originalX,originalY),templateImageDownscaled.shape[1]\*8,templateImageDownscaled.shape[0]\*8,linewidth=1,edgecolor='r',facecolor='none')

rect2 = patches.Rectangle((300,60),templateImageDownscaled.shape[1]\*8,templateImageDownscaled.shape[0]\*8,linewidth=1,edgecolor='b',facecolor='none')

ax.add\_patch(rect1)

ax.add\_patch(rect2)

plt.savefig('output1\_IOU.png')

plt.show()

### Censor infant face by applying Gaussian Blurring on the detected subregion

### # Read in image

### image = searchImage1.copy()

### # Create ROI coordinates

### topLeft = math.floor(originalX), math.floor(originalY)

### bottomRight = math.floor(originalX + (templateImageDownscaled.shape[1]\*8)), math.floor(originalY + (templateImageDownscaled.shape[0]\*8))

### x, y = topLeft[0], topLeft[1]

### w, h = bottomRight[0] - topLeft[0], bottomRight[1] - topLeft[1]

### # Grab ROI with Numpy slicing and blur

### ROI = image[y:y+h, x:x+w]

### blur = cv.GaussianBlur(ROI, (81,81), 0)

### # Insert ROI back into image

### image[y:y+h, x:x+w] = blur

### plt.subplot(1,1,1)

### plt.imshow(image)

### plt.savefig('output1\_Blurred.png')

### plt.show()

### Censor accuracy calculation

censorAccuracy1 = bb\_intersection\_over\_union(rect1, rect2)

print("Censor accuracy in Image1 is: " + str(censorAccuracy1\*100) + "%")

## Image 2

### Read and display searchImage2 and templateImage

searchImage2 = skimage.io.imread('input2.png')

# Generating a template

img1 = Image.open(r"input1.png")

left = 421

top = 191

right =609

bottom = 385

img2 = img1.crop((left, top, right, bottom))

img2.save('templateImage.png')

# Read template image

templateImage = skimage.io.imread('templateImage.png')

# Display searchImage2 and templateImage

plt.subplot(1,1,1)

plt.imshow(searchImage2)

plt.show()

#print(searchImage2.shape)

plt.subplot(1,1,1)

plt.imshow(templateImage)

plt.show()

#print(templateImage.shape)

### Display ideal search and template images from the pyramid

searchImage2Downscaled, templateImageDownscaled = imagePyramid(searchImage2, templateImage)

plt.subplot(1,1,1)

plt.imshow(searchImage2Downscaled)

plt.show()

#print(searchImage2Downscaled.shape)

plt.subplot(1,1,1)

plt.imshow(templateImageDownscaled)

plt.show()

#print(templateImageDownscaled.shape)

# We will use searchImage2Downscaled and templateImageDownscaled for covariance calculations

### Calculate covariance matrix of template

# Creates a 5x5 covariance matrix of template image

x,y,z = templateImageDownscaled.shape

featureTemplate = np.zeros((x,y,5))

for i in range(x):

for j in range(y):

xCoordinate = j

yCoordinate = i

R = templateImageDownscaled[yCoordinate][xCoordinate][0]

G = templateImageDownscaled[yCoordinate][xCoordinate][1]

B = templateImageDownscaled[yCoordinate][xCoordinate][2]

featureTemplate[i][j] = xCoordinate, yCoordinate, R, G, B

reshapedFeatureTemplate = featureTemplate.reshape(featureTemplate.shape[0]\*featureTemplate.shape[1],(featureTemplate.shape[2]))

covMatrixTemplate = np.cov(reshapedFeatureTemplate.transpose(),bias=True)

### Generate list containing all possible overlapping windows

a,b,c = searchImage2Downscaled.shape

x,y,z = templateImageDownscaled.shape

featureList = []

for i in range(a-x):

for j in range(b-y):

window = np.zeros((x,y,5))

for k in range(x):

for l in range(y):

xCoordinate = j + l

yCoordinate = i + k

R = searchImage2Downscaled[yCoordinate][xCoordinate][0]

G = searchImage2Downscaled[yCoordinate][xCoordinate][1]

B = searchImage2Downscaled[yCoordinate][xCoordinate][2]

window[k][l] = xCoordinate, yCoordinate, R, G, B

featureList.append(window)

### Reshape overlapping windows from 3D to 2D

featureListReshaped = []

for matrix in featureList:

reshapedMatrix = matrix.reshape(matrix.shape[0]\*matrix.shape[1],(matrix.shape[2]))

featureListReshaped.append(reshapedMatrix)

### Calculate candidate covariance matrices and store in a list

candidateCovMatrix = []

for matrix in featureListReshaped:

covMatrix = np.cov(matrix.transpose(),bias=True)

candidateCovMatrix.append(covMatrix)

### Riemannian Mannifold Calculation

#Following section creates a list that contains distances of all candidate covariances from model covariance matrix

from scipy.linalg import eigh

distanceMetric = []

alpha = 0

for matrix in candidateCovMatrix:

eigvals = eigh(covMatrixTemplate, matrix, eigvals\_only=True)

for values in eigvals:

if (values != 0):

alpha += (math.log(values))\*\*2

beta = math.sqrt(alpha)

distanceMetric.append(beta)

alpha=0

### Display coordinates of where maximum similarity is found

# Find coordinates of maximumum similarity in downscaled search image

valueOfMaximumSimilarity = min(distanceMetric)

indexOfMaximumSimilarity = distanceMetric.index(valueOfMaximumSimilarity)

coordinatesOfMaximumSimilarity = featureListReshaped[indexOfMaximumSimilarity][0][0:2]

### Display output image

# Find coordinates of maximumum similarity in original search image

originalX = (coordinatesOfMaximumSimilarity[0] / searchImage2Downscaled.shape[1]) \* searchImage2.shape[1]

originalY = (coordinatesOfMaximumSimilarity[1] / searchImage2Downscaled.shape[0]) \* searchImage2.shape[0]

fig,ax = plt.subplots()

ax.imshow(searchImage2)

rect1 = patches.Rectangle((originalX,originalY),templateImageDownscaled.shape[1]\*4,templateImageDownscaled.shape[0]\*4,linewidth=1,edgecolor='r',facecolor='none')

ax.add\_patch(rect1)

plt.show()

### Display output vs ground truth

# Ground truth - Blue

# Output - Red

fig,ax = plt.subplots()

ax.imshow(searchImage2)

rect1 = patches.Rectangle((originalX,originalY),templateImageDownscaled.shape[1]\*4,templateImageDownscaled.shape[0]\*4,linewidth=1,edgecolor='r',facecolor='none')

rect2 = patches.Rectangle((590,20),templateImageDownscaled.shape[1]\*4,templateImageDownscaled.shape[0]\*4,linewidth=1,edgecolor='b',facecolor='none')

ax.add\_patch(rect1)

ax.add\_patch(rect2)

plt.savefig('output2\_IOU.png')

plt.show()

### Censor infant face by applying Gaussian Blurring on the detected subregion

### Read in image

image = searchImage2.copy()

# Create ROI coordinates

topLeft = math.floor(originalX), math.floor(originalY)

bottomRight = math.floor(originalX + (templateImageDownscaled.shape[1]\*4)), math.floor(originalY + (templateImageDownscaled.shape[0]\*4))

x, y = topLeft[0], topLeft[1]

w, h = bottomRight[0] - topLeft[0], bottomRight[1] - topLeft[1]

# Grab ROI with Numpy slicing and blur

ROI = image[y:y+h, x:x+w]

blur = cv.GaussianBlur(ROI, (81,81), 0)

# Insert ROI back into image

image[y:y+h, x:x+w] = blur

plt.subplot(1,1,1)

plt.imshow(image)

plt.savefig('output2\_Blurred.png')

plt.show()

### Censor accuracy calculation

censorAccuracy1 = bb\_intersection\_over\_union(rect1, rect2)

print("Censor accuracy in Image1 is: " + str(censorAccuracy1\*100) + "%")

## Image 3

### Read and display searchImage3 and templateImage

searchImage3 = skimage.io.imread('input3.png')

# Generating a template

img1 = Image.open(r"input1.png")

left = 421

top = 191

right =609

bottom = 385

img2 = img1.crop((left, top, right, bottom))

img2.save('templateImage.png')

# Read template image

templateImage = skimage.io.imread('templateImage.png')

# Display searchImage2 and templateImage

plt.subplot(1,1,1)

plt.imshow(searchImage3)

plt.show()

#print(searchImage3.shape)

plt.subplot(1,1,1)

plt.imshow(templateImage)

plt.show()

#print(templateImage.shape)

### Display ideal search and template images from the pyramid

searchImage3Downscaled, templateImageDownscaled = imagePyramid(searchImage3, templateImage)

plt.subplot(1,1,1)

plt.imshow(searchImage3Downscaled)

plt.show()

#print(searchImage3Downscaled.shape)

plt.subplot(1,1,1)

plt.imshow(templateImageDownscaled)

plt.show()

#print(templateImageDownscaled.shape)

# We will use searchImage3Downscaled and templateImageDownscaled for covariance calculations

### Calculate covariance matrix of template

# Creates a 5x5 covariance matrix of template image

x,y,z = templateImageDownscaled.shape

featureTemplate = np.zeros((x,y,5))

for i in range(x):

for j in range(y):

xCoordinate = j

yCoordinate = i

R = templateImageDownscaled[yCoordinate][xCoordinate][0]

G = templateImageDownscaled[yCoordinate][xCoordinate][1]

B = templateImageDownscaled[yCoordinate][xCoordinate][2]

featureTemplate[i][j] = xCoordinate, yCoordinate, R, G, B

reshapedFeatureTemplate = featureTemplate.reshape(featureTemplate.shape[0]\*featureTemplate.shape[1],(featureTemplate.shape[2]))

covMatrixTemplate = np.cov(reshapedFeatureTemplate.transpose(),bias=True)

### Generate list containing all possible overlapping windows

a,b,c = searchImage3Downscaled.shape

x,y,z = templateImageDownscaled.shape

featureList = []

for i in range(a-x):

for j in range(b-y):

window = np.zeros((x,y,5))

for k in range(x):

for l in range(y):

xCoordinate = j + l

yCoordinate = i + k

R = searchImage3Downscaled[yCoordinate][xCoordinate][0]

G = searchImage3Downscaled[yCoordinate][xCoordinate][1]

B = searchImage3Downscaled[yCoordinate][xCoordinate][2]

window[k][l] = xCoordinate, yCoordinate, R, G, B

featureList.append(window)

### Reshape overlapping windows from 3D to 2D

featureListReshaped = []

for matrix in featureList:

reshapedMatrix = matrix.reshape(matrix.shape[0]\*matrix.shape[1],(matrix.shape[2]))

featureListReshaped.append(reshapedMatrix)

### Calculate candidate covariance matrices and store in a list

candidateCovMatrix = []

for matrix in featureListReshaped:

covMatrix = np.cov(matrix.transpose(),bias=True)

candidateCovMatrix.append(covMatrix)

### Riemannian Mannifold Calculation

#Following section creates a list that contains distances of all candidate covariances from model covariance matrix

from scipy.linalg import eigh

distanceMetric = []

alpha = 0

for matrix in candidateCovMatrix:

eigvals = eigh(covMatrixTemplate, matrix, eigvals\_only=True)

for values in eigvals:

if (values != 0):

alpha += (math.log(values))\*\*2

beta = math.sqrt(alpha)

distanceMetric.append(beta)

alpha=0

### Display coordinates of where maximum similarity is found

# Find coordinates of maximumum similarity in downscaled search image

valueOfMaximumSimilarity = min(distanceMetric)

indexOfMaximumSimilarity = distanceMetric.index(valueOfMaximumSimilarity)

coordinatesOfMaximumSimilarity = featureListReshaped[indexOfMaximumSimilarity][0][0:2]

### Display output image

# Find coordinates of maximumum similarity in original search image

originalX = (coordinatesOfMaximumSimilarity[0] / searchImage3Downscaled.shape[1]) \* searchImage3.shape[1]

originalY = (coordinatesOfMaximumSimilarity[1] / searchImage3Downscaled.shape[0]) \* searchImage3.shape[0]

fig,ax = plt.subplots()

ax.imshow(searchImage3)

rect1 = patches.Rectangle((originalX,originalY),templateImageDownscaled.shape[1]\*4,templateImageDownscaled.shape[0]\*4,linewidth=1,edgecolor='r',facecolor='none')

ax.add\_patch(rect1)

plt.show()

### Display output vs ground truth

# Ground truth - Blue

# Output - Red

fig,ax = plt.subplots()

ax.imshow(searchImage3)

rect1 = patches.Rectangle((originalX,originalY),templateImageDownscaled.shape[1]\*4,templateImageDownscaled.shape[0]\*4,linewidth=1,edgecolor='r',facecolor='none')

rect2 = patches.Rectangle((1200,120),templateImageDownscaled.shape[1]\*4,templateImageDownscaled.shape[0]\*4,linewidth=1,edgecolor='b',facecolor='none')

ax.add\_patch(rect1)

ax.add\_patch(rect2)

plt.savefig('output3\_IOU.png')

plt.show()

### Censor infant face by applying Gaussian Blurring on the detected subregion

### Read in image

image = searchImage3.copy()

# Create ROI coordinates

topLeft = math.floor(originalX), math.floor(originalY)

bottomRight = math.floor(originalX + (templateImageDownscaled.shape[1]\*4)), math.floor(originalY + (templateImageDownscaled.shape[0]\*4))

x, y = topLeft[0], topLeft[1]

w, h = bottomRight[0] - topLeft[0], bottomRight[1] - topLeft[1]

# Grab ROI with Numpy slicing and blur

ROI = image[y:y+h, x:x+w]

blur = cv.GaussianBlur(ROI, (81,81), 0)

# Insert ROI back into image

image[y:y+h, x:x+w] = blur

plt.subplot(1,1,1)

plt.imshow(image)

plt.savefig('output3\_Blurred.png')

plt.show()

### Censor accuracy calculation

censorAccuracy1 = bb\_intersection\_over\_union(rect1, rect2)

print("Censor accuracy in Image3 is: " + str(censorAccuracy1\*100) + "%")

## Image 4

### Read and display searchImage4 and templateImage

searchImage4 = skimage.io.imread('input4.png')

# Generating a template

img1 = Image.open(r"input1.png")

left = 421

top = 191

right =609

bottom = 385

img2 = img1.crop((left, top, right, bottom))

img2.save('templateImage.png')

# Read template image

templateImage = skimage.io.imread('templateImage.png')

# Display searchImage2 and templateImage

plt.subplot(1,1,1)

plt.imshow(searchImage4)

plt.show()

#print(searchImage4.shape)

plt.subplot(1,1,1)

plt.imshow(templateImage)

plt.show()

#print(templateImage.shape)

### Display ideal search and template images from the pyramid

searchImage4Downscaled, templateImageDownscaled = imagePyramid(searchImage4, templateImage)

plt.subplot(1,1,1)

plt.imshow(searchImage4Downscaled)

plt.show()

#print(searchImage4Downscaled.shape)

plt.subplot(1,1,1)

plt.imshow(templateImageDownscaled)

plt.show()

#print(templateImageDownscaled.shape)

# We will use searchImage4Downscaled and templateImageDownscaled for covariance calculations

### Calculate covariance matrix of template

# Creates a 5x5 covariance matrix of template image

x,y,z = templateImageDownscaled.shape

featureTemplate = np.zeros((x,y,5))

for i in range(x):

for j in range(y):

xCoordinate = j

yCoordinate = i

R = templateImageDownscaled[yCoordinate][xCoordinate][0]

G = templateImageDownscaled[yCoordinate][xCoordinate][1]

B = templateImageDownscaled[yCoordinate][xCoordinate][2]

featureTemplate[i][j] = xCoordinate, yCoordinate, R, G, B

reshapedFeatureTemplate = featureTemplate.reshape(featureTemplate.shape[0]\*featureTemplate.shape[1],(featureTemplate.shape[2]))

covMatrixTemplate = np.cov(reshapedFeatureTemplate.transpose(),bias=True)

### Generate list containing all possible overlapping windows

a,b,c = searchImage4Downscaled.shape

x,y,z = templateImageDownscaled.shape

featureList = []

for i in range(a-x):

for j in range(b-y):

window = np.zeros((x,y,5))

for k in range(x):

for l in range(y):

xCoordinate = j + l

yCoordinate = i + k

R = searchImage4Downscaled[yCoordinate][xCoordinate][0]

G = searchImage4Downscaled[yCoordinate][xCoordinate][1]

B = searchImage4Downscaled[yCoordinate][xCoordinate][2]

window[k][l] = xCoordinate, yCoordinate, R, G, B

featureList.append(window)

### Reshape overlapping windows from 3D to 2D

featureListReshaped = []

for matrix in featureList:

reshapedMatrix = matrix.reshape(matrix.shape[0]\*matrix.shape[1],(matrix.shape[2]))

featureListReshaped.append(reshapedMatrix)

### Calculate candidate covariance matrices and store in a list

candidateCovMatrix = []

for matrix in featureListReshaped:

covMatrix = np.cov(matrix.transpose(),bias=True)

candidateCovMatrix.append(covMatrix)

### Riemannian Mannifold Calculation

#Following section creates a list that contains distances of all candidate covariances from model covariance matrix

from scipy.linalg import eigh

distanceMetric = []

alpha = 0

for matrix in candidateCovMatrix:

eigvals = eigh(covMatrixTemplate, matrix, eigvals\_only=True)

for values in eigvals:

if (values != 0):

alpha += (math.log(values))\*\*2

beta = math.sqrt(alpha)

distanceMetric.append(beta)

alpha=0

### Display coordinates of where maximum similarity is found

# Find coordinates of maximumum similarity in downscaled search image

valueOfMaximumSimilarity = min(distanceMetric)

indexOfMaximumSimilarity = distanceMetric.index(valueOfMaximumSimilarity)

coordinatesOfMaximumSimilarity = featureListReshaped[indexOfMaximumSimilarity][0][0:2]

### Display output image

# Find coordinates of maximumum similarity in original search image

originalX = (coordinatesOfMaximumSimilarity[0] / searchImage4Downscaled.shape[1]) \* searchImage4.shape[1]

originalY = (coordinatesOfMaximumSimilarity[1] / searchImage4Downscaled.shape[0]) \* searchImage4.shape[0]

fig,ax = plt.subplots()

ax.imshow(searchImage4)

rect = patches.Rectangle((originalX,originalY),templateImageDownscaled.shape[1]\*5,templateImageDownscaled.shape[0]\*5,linewidth=1,edgecolor='r',facecolor='none')

ax.add\_patch(rect)

plt.show()

### Display output vs ground truth

fig,ax = plt.subplots()

ax.imshow(searchImage4)

rect1 = patches.Rectangle((originalX,originalY),templateImageDownscaled.shape[1]\*5,templateImageDownscaled.shape[0]\*5,linewidth=1,edgecolor='r',facecolor='none')

rect2 = patches.Rectangle((830,60),templateImageDownscaled.shape[1]\*5,templateImageDownscaled.shape[0]\*5,linewidth=1,edgecolor='b',facecolor='none')

ax.add\_patch(rect1)

ax.add\_patch(rect2)

plt.savefig('output4\_IOU.png')

plt.show()

### Censor infant face by applying Gaussian Blurring on the detected subregion

### Read in image

image = searchImage4.copy()

# Create ROI coordinates

topLeft = math.floor(originalX), math.floor(originalY)

bottomRight = math.floor(originalX + (templateImageDownscaled.shape[1]\*5)), math.floor(originalY + (templateImageDownscaled.shape[0]\*5))

x, y = topLeft[0], topLeft[1]

w, h = bottomRight[0] - topLeft[0], bottomRight[1] - topLeft[1]

# Grab ROI with Numpy slicing and blur

ROI = image[y:y+h, x:x+w]

blur = cv.GaussianBlur(ROI, (81,81), 0)

# Insert ROI back into image

image[y:y+h, x:x+w] = blur

plt.subplot(1,1,1)

plt.imshow(image)

plt.savefig('output4\_Blurred.png')

plt.show()

### Censor accuracy calculation

censorAccuracy1 = bb\_intersection\_over\_union(rect1, rect2)

print("Censor accuracy in Image4 is: " + str(censorAccuracy1\*100) + "%")