ICE-7. MPEG-7 Color Layout Descriptor

Please download the images from the following link:
https://drive.google.com/file/d/1h0JFzpyc7zNZiln5Ls74CfvOfhtl51XK/view?usp=sharing

Please refer to the following link for the code of Color Layout Descriptor:
https://github.com/scferrada/imgpedia

Task 1. Please using the given code to extract the color layout features from the downloaded image 001.jpg, and display it.

```
# write you code here
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# import the required libraries
import cv2, sys, os
import numpy as np

# Create an array to store average colors
averages = np.zeros((8, 8, 3))

# Read the input image
img = cv2.imread(r'/content/drive/MyDrive/ICE-7_data/ICE-7_data/001.jpg')

# Get the dimensions of the image
imgH, imgW, _ = img.shape

# Divide the image into 8x8 slices
for row in range(8):
    for col in range(8):
        # Extract the current slice
        slice = img[imgH // 8 * row: imgH // 8 * (row + 1), imgW // 8 * col: imgW // 8 * (col + 1)]

        # Calculate the average color of the slice
        average_color_per_row = np.mean(slice, axis=0)
        average_color = np.mean(average_color_per_row, axis=0)
        average_color = np.uint8(average_color)

        # Store the average color in the array
        averages[row][col][0] = average_color[0]
        averages[row][col][1] = average_color[1]
        averages[row][col][2] = average_color[2]

# Convert the average color array to YCrCb color space
icon = cv2.cvtColor(np.array(averages, dtype=np.uint8), cv2.COLOR_BGR2YCR_CB)
```

```
# Split the YCrCb image into Y, Cr, and Cb channels
y, cr, cb = cv2.split(icon)

# Apply Discrete Cosine Transform (DCT) to Y, Cr, and Cb channels
dct_y = cv2.dct(np.float32(y))
dct_cb = cv2.dct(np.float32(cb))
dct_cr = cv2.dct(np.float32(cr))

# Perform zigzag scanning on DCT coefficients
dct_y_zigzag = []
dct_cb_zigzag = []
dct_cr_zigzag = []
flip = True
flipped_dct_y = np.fliplr(dct_y)
flipped_dct_cb = np.fliplr(dct_cb)
flipped_dct_cr = np.fliplr(dct_cr)

for i in range(8 + 8 - 1):
    k_diag = 8 - 1 - i
    diag_y = np.diag(flipped_dct_y, k=k_diag)
    diag_cb = np.diag(flipped_dct_cb, k=k_diag)
    diag_cr = np.diag(flipped_dct_cr, k=k_diag)

    if flip:
        diag_y = diag_y[::-1]
        diag_cb = diag_cb[::-1]
        diag_cr = diag_cr[::-1]

    dct_y_zigzag.append(diag_y)
    dct_cb_zigzag.append(diag_cb)
    dct_cr_zigzag.append(diag_cr)
    flip = not flip

# Concatenate the zigzag scanned DCT coefficients
array = np.concatenate([np.concatenate(dct_y_zigzag), np.concatenate(dct_cb_zigzag), np.concatenate(d

# Display the resulting array
array
```

```
array([ 7.54875000e+02, -3.71453323e+01, -2.75390893e-01, -2.05506821e+02,
        7.78468102e-02, -2.34176540e+01, -3.93178177e+01,  3.43435440e+01,
        2.75409832e+01, -5.34272051e+00, -4.38750000e+01,  5.65796041e+00,
        2.59727240e+00, -2.54444008e+01, -1.43750000e+01, -2.34878922e+01,
        2.55022869e+01,  2.20043163e+01, -3.80048065e+01, -5.47946072e+00,
        6.65629059e-02,  5.35990372e+01, -1.31532040e+01,  2.22179368e-01,
        4.08514328e+01, -2.71923709e+00, -2.33335533e+01, -7.21246767e+00,
       -2.86610723e+00,  1.22206163e+01,  3.80418777e+00, -3.12251759e+01,
        3.03049412e+01,  1.38942504e+00,  7.60543251e+00,  1.85286961e+01,
        1.82548904e+01, -2.60280678e-03, -1.26275692e+01,  2.58750000e+01,
        2.63148155e+01, -3.00260210e+00, -2.30272079e+00, -2.49541059e-01,
       -1.75480080e+01,  2.62883568e+01,  8.04879284e+00, -2.93222847e+01,
        1.81053467e+01, -1.63744450e+01, -1.93038082e+01,  1.69019365e+00,
        8.70240688e+00,  1.83429193e+00,  4.75085020e+00,  5.05936050e+00,
       -1.27258711e+01, -8.98755074e+00, -8.12548637e+00,  6.52728260e-01,
        2.36595964e+00, -1.14373207e+00,  6.95866251e+00, -5.61947727e+00,
         1.05687500e+03,  2.33692932e+00,  1.66218597e+02,  3.59732704e+01,
        -3.10048962e+00,  2.19854088e+01,  1.39373083e+01, -6.82184505e+00,
        -4.46828651e+00, -3.38446426e+00, -7.37500000e+00,  3.54915285e+00,
        -2.25270977e+01,  5.51983976e+00, -3.37500000e+00,  1.10027876e+01,
        -1.13995914e+01, -1.18014374e+01,  1.19812572e+00,  3.00066352e+00,
        -2.82964516e+00, -1.01651497e+01, -1.14870477e+00,  6.32986593e+00,
        -1.06097631e+01,  1.85936594e+00,  5.95567417e+00,  1.15940971e+01,
         8.16266537e-01, -6.40579581e-01, -7.39151764e+00,  1.85883541e+01,
        -4.99310207e+00, -1.57048213e+00, -2.85975337e+00, -8.51349068e+00,
        -3.92626905e+00,  2.77252483e+00,  4.44167137e+00,  1.87500000e+00,
        -8.44741535e+00, -1.07274761e+01,  1.18464339e+00, -1.16959870e+00,
         1.55625910e-01, -4.35215998e+00, -1.55539522e+01,  6.63819361e+00,
         4.67327356e+00,  2.48151040e+00, -1.71726787e+00,  1.32423019e+00,
         3.19594240e+00, -6.56751931e-01, -1.34578872e+00, -4.01528955e-01,
         2.45730233e+00,  1.15069304e+01, -1.74703538e-01,  2.77096897e-01,
        -3.88031542e-01,  7.75154054e-01,  1.54322861e-02,  8.86023760e-01,
         9.87000000e+02, -2.39751339e+00, -6.25978966e+01, -5.24261856e+01,
         1.81337571e+00, -1.63088245e+01, -1.11743298e+01,  3.89959693e+00,
         3.86709118e+00,  1.93433590e+01, -3.25000000e+00, -4.26588202e+00,
         1.71102829e+01, -3.98335147e+00,  1.50000000e+00, -8.58107281e+00,
         6.53243399e+00,  7.82848549e+00, -3.61910248e+00, -1.39890611e+00,
         4.19096994e+00,  1.06211147e+01, -6.69988990e-01, -2.99580979e+00,
         7.71191835e+00, -3.66268933e-01, -4.68244267e+00, -8.66875267e+00,
        -1.06134844e+00,  8.25644076e-01,  4.77057743e+00, -1.31332035e+01,
         2.45378351e+00,  1.17903984e+00,  2.83975005e+00,  2.85453773e+00,
         3.69694328e+00, -2.22334933e+00, -3.73916650e+00, -2.25000000e+00,
         6.01603699e+00,  7.52665091e+00, -4.78526317e-02,  4.12503779e-01,
        -1.63812026e-01,  3.05486250e+00,  1.04304724e+01, -5.49839830e+00,
        -2.80205131e+00, -1.90118897e+00,  4.22311604e-01, -2.06715301e-01,
        -2.00627184e+00,  7.25445569e-01,  5.61091483e-01, -4.76779729e-01,
        -1.54695201e+00, -6.57088232e+00, -3.88076119e-02,  1.39718518e-01,
         8.17748547e-01, -3.77117991e-01, -1.33306444e-01, -3.18579614e-01],
       dtype=float32)
```

Task 2. Retrieve the image that is the most similar to image 001.jpg from the downloaded images (not itself).

Please try to use at least three different metrics to measure the distance between color layout descriptors (L1 distance, L2 distance, earth mover's distance, etc)

```
# write you code here

import os

# Change the current working directory to the specified path
os.chdir('/content/drive/MyDrive/ICE-7_data/ICE-7_data/')

# Get the list of files in the current directory
path = os.listdir()

# Create an empty list to store the image filenames
images = []

# Iterate over each file in the directory
for i in path:
    images.append(i)

# Print the current working directory
print(os.getcwd())

# Remove the '001.jpg' file from the list of images
images.remove('001.jpg')

# Display the updated list of image filenames
images
```

```
/content/drive/MyDrive/ICE-7_data/ICE-7_data
['004.jpg',
 '003.jpg',
 '005.jpg',
 '002.jpg',
 '011.jpg',
 '014.jpg',
 '007.jpg',
 '013.jpg',
 '008.jpg',
 '015.jpg',
 '006.jpg',
 '010.jpg',
 '012.jpg',
 '016.jpg',
 '017.jpg',
 '009.jpg']
```

```
from math import sqrt
from scipy import spatial
from scipy.spatial import distance
import numpy as np
from scipy.stats import wasserstein_distance
from sklearn.metrics.pairwise import cosine_similarity

# Set the path to the directory containing the images
path = "/content/drive/MyDrive/ICE-7_data/ICE-7_data"

# Get the list of entries (files and directories) in the specified path
entries = os.listdir()

# Create an empty list to store the image data
dat = []

# Iterate over each entry in the directory
for entry in entries:
    # Create the complete path to the image file
    pat = path + '/' + entry

    # Read the image
    da = cv2.imread(pat)

    # Append the image data to the list
    dat.append(da)
```

```
# Create empty lists to store the calculated distances
array = [0] * len(dat)
icon = [0] * len(dat)
l1dist = [0] * len(dat)
l2dist = [0] * len(dat)
emd = [0] * len(dat)

# Iterate over each image in the list
for m in range(0, len(dat)):
    # Initialize an array to store average colors
    averages = np.zeros((8, 8, 3))

    # Get the current image
    img = dat[m]

    # Get the dimensions of the image
    imgH, imgW, _ = img.shape

    # Divide the image into 8x8 slices and calculate average colors
    for row in range(8):
        for col in range(8):
            slice = img[imgH // 8 * row: imgH // 8 * (row + 1), imgW // 8 * col: imgW // 8 * (col + 1
            average_color_per_row = np.mean(slice, axis=0)
            average_color = np.mean(average_color_per_row, axis=0)
            average_color = np.uint8(average_color)
            averages[row][col][0] = average_color[0]
            averages[row][col][1] = average_color[1]
            averages[row][col][2] = average_color[2]

        # Convert the average color array to YCrCb color space
        icon[m] = cv2.cvtColor(np.array(averages, dtype=np.uint8), cv2.COLOR_BGR2YCR_CB)

    # Split the YCrCb image into Y, Cr, and Cb channels
    y, cr, cb = cv2.split(icon[m])

    # Apply Discrete Cosine Transform (DCT) to Y, Cr, and Cb channels
    dct_y = cv2.dct(np.float32(y))
    dct_cb = cv2.dct(np.float32(cb))
    dct_cr = cv2.dct(np.float32(cr))

    # Perform zigzag scanning on DCT coefficients
    dct_y_zigzag = []
    dct_cb_zigzag = []
    dct_cr_zigzag = []
    flip = True
    flipped_dct_y = np.fliplr(dct_y)
    flipped_dct_cb = np.fliplr(dct_cb)
    flipped_dct_cr = np.fliplr(dct_cr)

    for i in range(8 + 8 - 1):
        k_diag = 8 - 1 - i
        diag_y = np.diag(flipped_dct_y, k=k_diag)
        diag_cb = np.diag(flipped_dct_cb, k=k_diag)
        diag_cr = np.diag(flipped_dct_cr, k=k_diag)
        if flip:
            diag_y = diag_y[::-1]
            diag_cb = diag_cb[::-1]
            diag_cr = diag_cr[::-1]

        dct_y_zigzag.append(diag_y)
        dct_cb_zigzag.append(diag_cb)
        dct_cr_zigzag.append(diag_cr)
        flip = not flip

    # Concatenate the zigzag scanned DCT coefficients
    array[m] = np.concatenate([np.concatenate(dct_y_zigzag), np.concatenate(dct_cb_zigzag), np.concat

    # Calculate the L1 distance between the current image and the reference image (array[0])
    l1dist[m] = np.linalg.norm(array[0] - array[m], ord=1)

    # Calculate the L2 distance between the current image and the reference image (array[0])
    l2dist[m] = np.linalg.norm(array[0] - array[m], ord=2)

    # Calculate the Earth Mover's distance between the current image and the reference image (array[0
    emd[m] = sum(abs(array[0] - array[m])) / (len(dat) - 1)

# Print the calculated distances
print("\nUsing L1-DISTANCE: ", l1dist)
print("\nUsing L2-DISTANCE: ", l2dist)
print("\nUsing Earth Mover's DISTANCE: ", emd)
```

```
Using L1-DISTANCE:  [0.0, 3270.5764, 3885.7505, 4138.3555, 3414.1968, 3869.8684, 3911.4028, 3897

Using L2-DISTANCE:  [0.0, 383.69128, 798.213, 599.48816, 513.87933, 560.43555, 716.0182, 519.581

Using Earth Mover's DISTANCE:  [0.0, 204.41103096678853, 242.85941885225475, 258.64721602760255,
```

Question 1. Please answer if the outputs of Task 2 are the same. If the outputs are different, based on your perspective, which metrics gives the most similar retrieval? If the outputs are the same, do you think we can use any metrics to compare the color layout features in any cases? Give an explanation to your answer.

Answer to Q1:

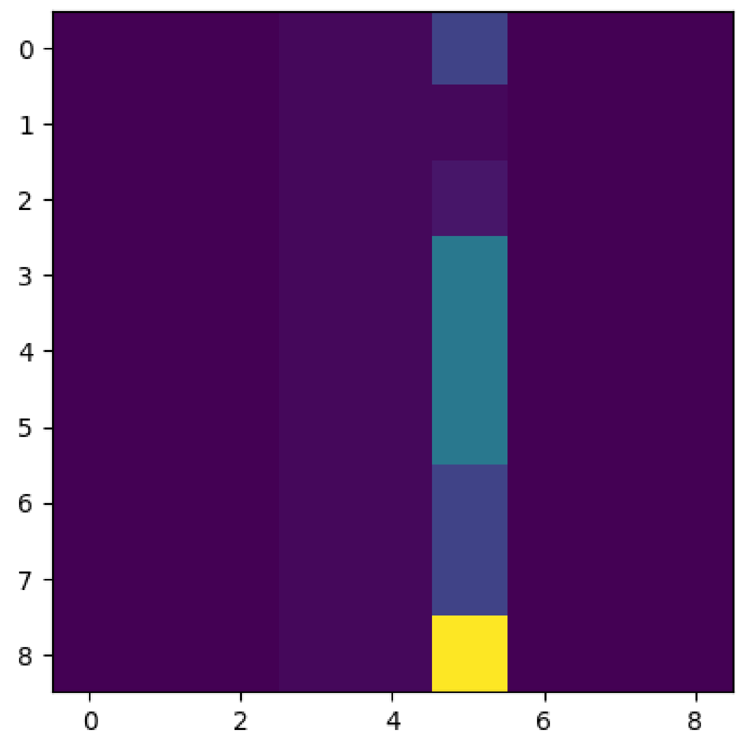
(Tutorial) Convolutional filter

Please review the following article for further understanding of Convolutional layer in Convolutional Neural Network https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/
Please download the image from the following link for the your program
https://drive.google.com/file/d/1MSQ8e0nxoBIkcp5sdo8Vd-65b9okx6hX/view?usp=sharing

- use pip install opencv-python to install cv2

Example. Convolutional filter

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

# build the test data
data = np.array([[0, 0, 0, 1, 1, 10, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 0, 0, 0],
       [0, 0, 0, 1, 1, 3, 0, 0, 0],
       [0, 0, 0, 1, 1, 20, 0, 0, 0],
       [0, 0, 0, 1, 1, 20, 0, 0, 0],
       [0, 0, 0, 1, 1, 20, 0, 0, 0],
       [0, 0, 0, 1, 1, 10, 0, 0, 0],
       [0, 0, 0, 1, 1, 10, 0, 0, 0],
       [0, 0, 0, 1, 1, 50, 0, 0, 0]], dtype='uint8')
plt.imshow(data)
pylab.show()

#build the convolutional kernel
fil = np.array([[ -1, -1, -1],
                [ -1, 8, -1],
                [ -1, -1, -1]])
#use filter2D to apply convolution
res = cv2.filter2D(data,-1,fil, borderType=cv2.BORDER_CONSTANT)
print(res)

plt.imshow(res)
pylab.show()
```



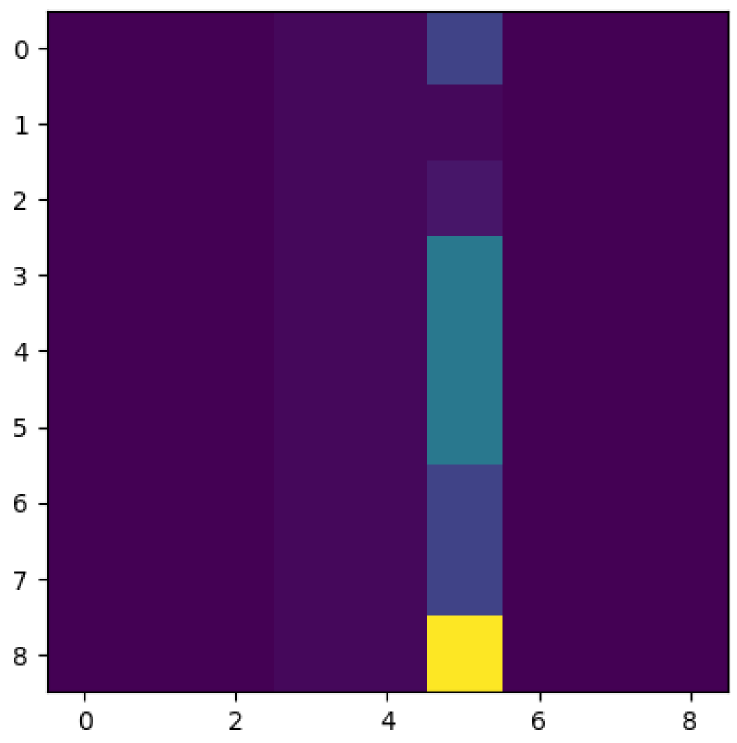
```
[[  0   0   0   5   0  77   0   0   0]
 [  0   0   0   3   0   0   0   0   0]
 [  0   0   0   3   0   0   0   0   0]
 [  0   0   0   3   0 134   0   0   0]
 [  0   0   0   3   0 117   0   0   0]
 [  0   0   0   3   0 127   0   0   0]
 [  0   0   0   3   0  47   0   0   0]
 [  0   0   0   3   0  17   0   0   0]
 [  0   0   0   5   0 255   0   0   0]]
```

Task 3. Building three different kernels for different purposes, and commenting the purpose of each kernel

For instance, the kernel in the example is using for edge extraction

3.1 Sharpen Kernel

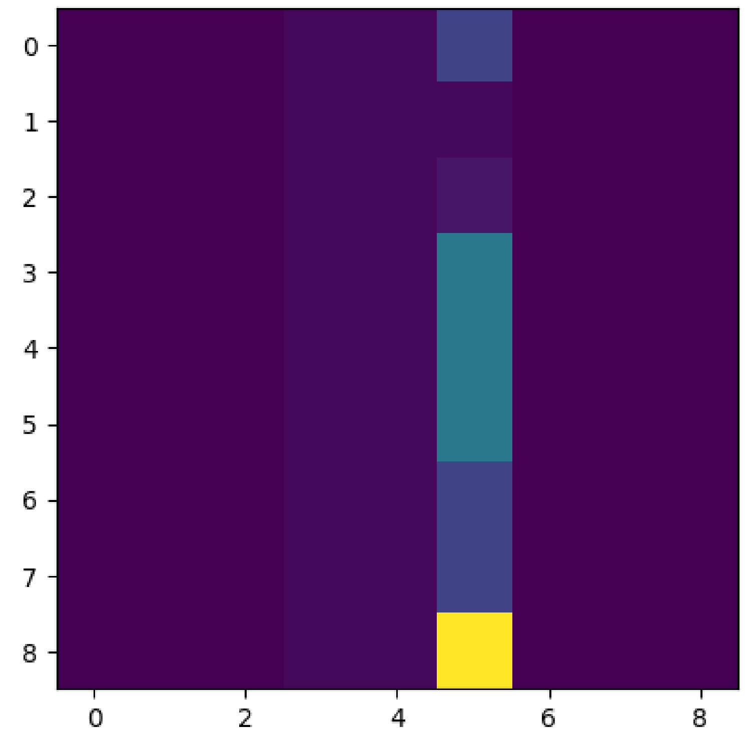
```
# write your code here

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

# build the test data
data = np.array([[0, 0, 0, 1, 1, 10, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 0, 0, 0],
       [0, 0, 0, 1, 1, 3, 0, 0, 0],
       [0, 0, 0, 1, 1, 20, 0, 0, 0],
       [0, 0, 0, 1, 1, 20, 0, 0, 0],
       [0, 0, 0, 1, 1, 20, 0, 0, 0],
       [0, 0, 0, 1, 1, 10, 0, 0, 0],
       [0, 0, 0, 1, 1, 10, 0, 0, 0],
       [0, 0, 0, 1, 1, 50, 0, 0, 0]], dtype='uint8')
plt.imshow(data)
pylab.show()

#build the convolutional kernel
fil1 = np.array([[ 0, -1, 0],
                [ -1, 5, -1],
                [ 0, -1, 0]])
#use filter2D to apply convolution
res1 = cv2.filter2D(data,-1,fil1, borderType=cv2.BORDER_CONSTANT)
print(res1)

plt.imshow(res1)
pylab.show()
```



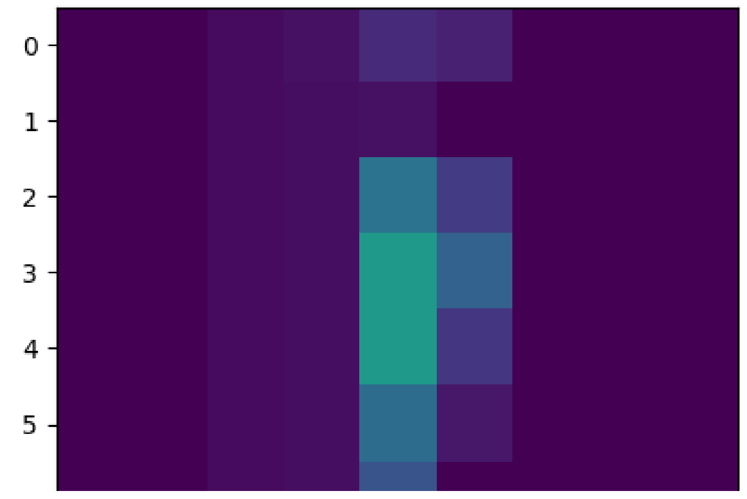
```
[[  0   0   0   3   0  48   0   0   0]
 [  0   0   0   2   1   0   0   0   0]
 [  0   0   0   2   0   0   0   0   0]
 [  0   0   0   2   0  76   0   0   0]
 [  0   0   0   2   0  59   0   0   0]
 [  0   0   0   2   0  69   0   0   0]
```

3.2 The emboss kernel

```
plt.imshow(data)
pylab.show()

#build the convolutional kernel
fil2 = np.array([[ -2, -1, 0],
                [ -1, 1, 1],
                [ 0, 1, 2]])

#use filter2D to apply convolution
res2 = cv2.filter2D(data,-1,fil2, borderType=cv2.BORDER_CONSTANT)
print(res2)

plt.imshow(res2)
pylab.show()
```



```
[[  0   0   3   5  13  10   0   0   0]
 [  0   0   3   4   5   0   0   0   0]
 [  0   0   3   4  41  19   0   0   0]
 [  0   0   3   4  58  34   0   0   0]
 [  0   0   3   4  58  17   0   0   0]
 [  0   0   3   4  38   7   0   0   0]
 [  0   0   3   4  28   0   0   0   0]
 [  0   0   3   4 108  47   0   0   0]
 [  0   0   1   1  47  37   0   0   0]]
```



3.3 The Top Sabol Kernel :

```
plt.imshow(data)
pylab.show()

#build the convolutional kernel
fil3 = np.array([[ 1, 2, 1],
                 [ 0, 0, 0],
                [ -1, -2, -1]])

#use filter2D to apply convolution
res3 = cv2.filter2D(data,-1,fil3, borderType=cv2.BORDER_CONSTANT)
print(res3)

plt.imshow(res3)
pylab.show()
```

Task 4. Refer to the multiple channels section in the tutorial material, apply the three kernels you built in task 1 to the downloaded image. Visualize the feature maps produced by the three kernels. Combine the three feature maps to a 3-channels feature map and visualize it.

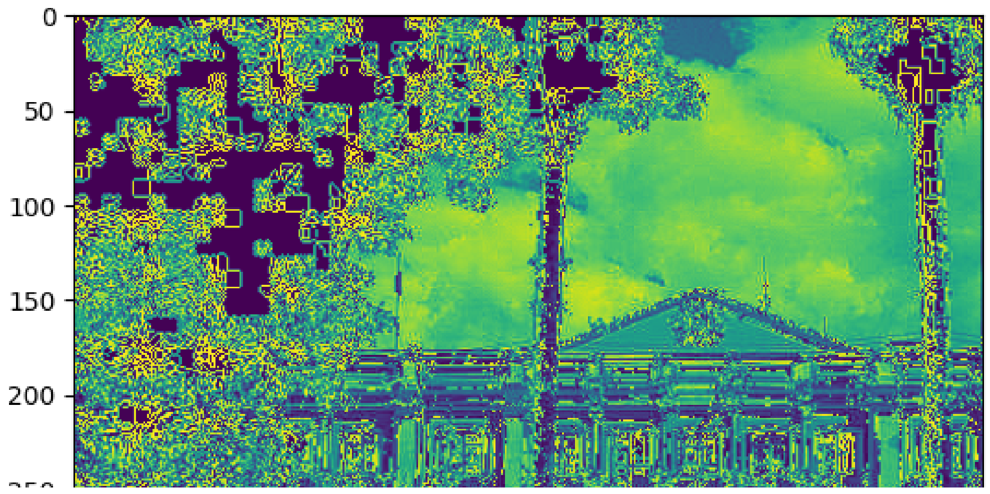
```
# write your code here

# import required libraries

import cv2
import matplotlib.pyplot as plt
import pylab
import numpy as np
from PIL import Image
from numpy import asarray

# write your code here

originalImage = cv2.imread(r'/content/1.jpg')

# convert image to numpy array
originalData = asarray(originalImage)
print(type(originalData))

# summarize shape and print the data
print(originalData.shape)
print(originalData)

# Display the original image

plt.title("Original Image")
plt.imshow(originalImage)
```

```
  ...
  [  9  17  16]
  [ 13  16  14]
  [ 54  55  51]]

 [[  0   1   0]
  [  0   1   0]
  [  0   0   0]
  ...
  [  8  16  15]
  [  7  10   8]
  [ 35  36  32]]

 [[  0   1   0]
  [  0   1   0]
  [  0   0   0]
  ...
  [  6  12  11]
  [  3   4   2]
  [ 15  14  10]]

 ...

 [[ 87 108 136]
  [ 87 108 136]
  [ 88 109 137]
  ...
  [ 96 116 141]
  [ 97 117 142]
  [ 97 117 142]]

 [[ 89 110 138]
  [ 89 110 138]
  [ 89 110 138]
  ...
  [ 84 104 129]
  [ 83 103 128]
  [ 81 101 126]]

 [[ 89 110 138]
  [ 89 110 138]
  [ 89 110 138]
  ...
  [104 124 149]
  [103 123 148]
  [100 120 145]]]
<matplotlib.image.AxesImage at 0x7e73a31176a0>
```

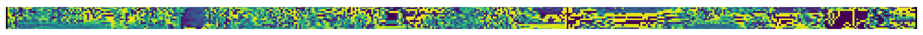
```
import cv2
from PIL import Image
from matplotlib import pyplot

def load_image(image_path):
    coloured_image = cv2.imread(image_path)
    grey_image = cv2.cvtColor(coloured_image, cv2.COLOR_BGR2GRAY)
    return grey_image

def convolve2d(image, kernel):

    # Flip the kernel
    kernel = np.flipud(np.fliplr(kernel))
    # convolution output
    output = np.zeros_like(image)

    # Add zero padding to the input image
    image_padded = np.zeros((image.shape[0] + 2, image.shape[1] + 2))
    image_padded[1:-1, 1:-1] = image

    # Loop over every pixel of the image
    for x in range(image.shape[1]):
        for y in range(image.shape[0]):
            # element-wise multiplication of the kernel and the image
            output[y, x]=(kernel * image_padded[y: y+3, x: x+3]).sum()

    return output
```

4.1 The Sharpen Kernel

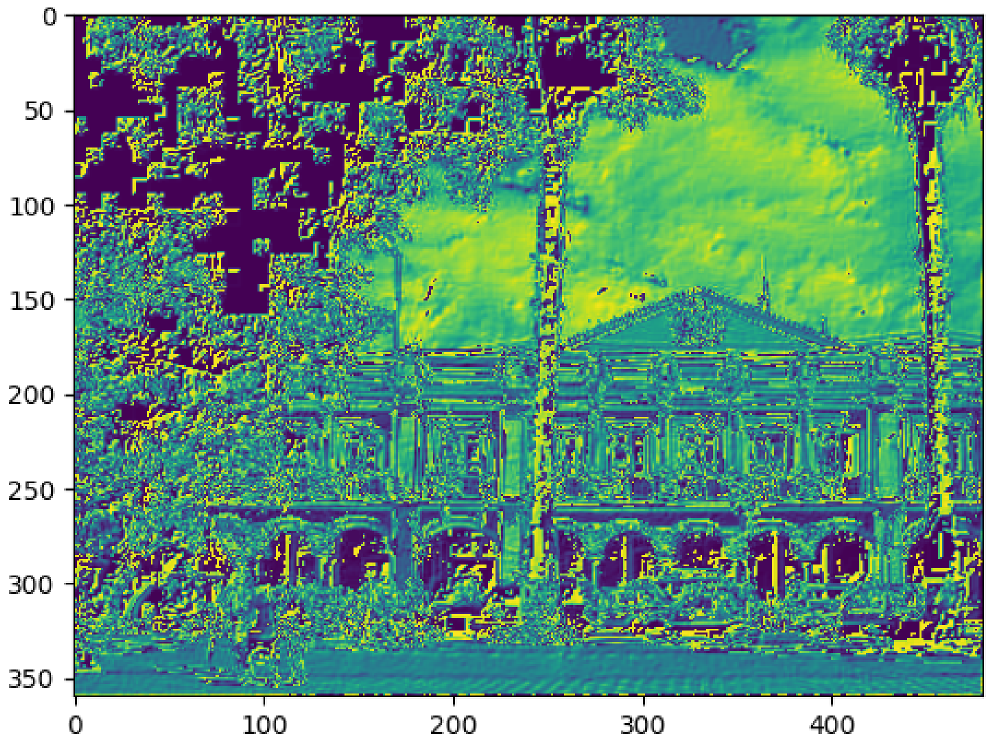
```
input_image = load_image('/content/1.jpg')

# kernel to be used to get sharpened image
KERNEL1 = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
image_sharpen = convolve2d(input_image, kernel=KERNEL1)

plt.imshow(image_sharpen)
pylab.show()
```



4.2 The emboss kernel

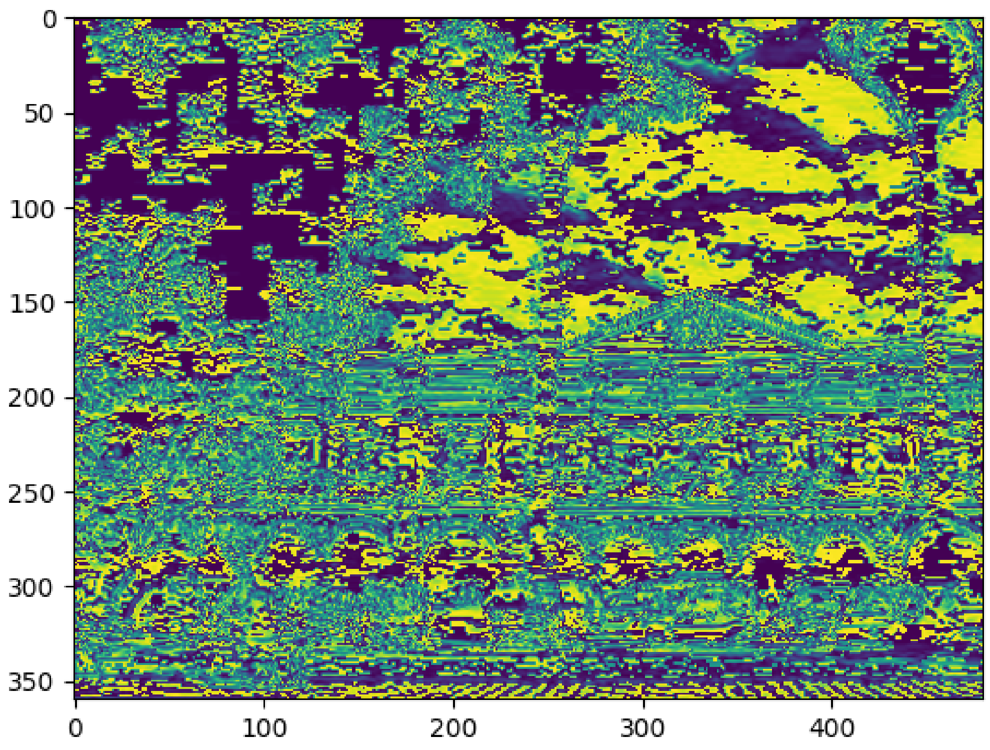


```
# kernel to find depth by emphasizing the differences of pixels

KERNEL2 = np.array([[ -2, -1, 0],[ -1, 1, 1],[ 0, 1, 2]])
image_emboss = convolve2d(input_image, kernel=KERNEL2)

plt.imshow(image_emboss)
pylab.show()
```

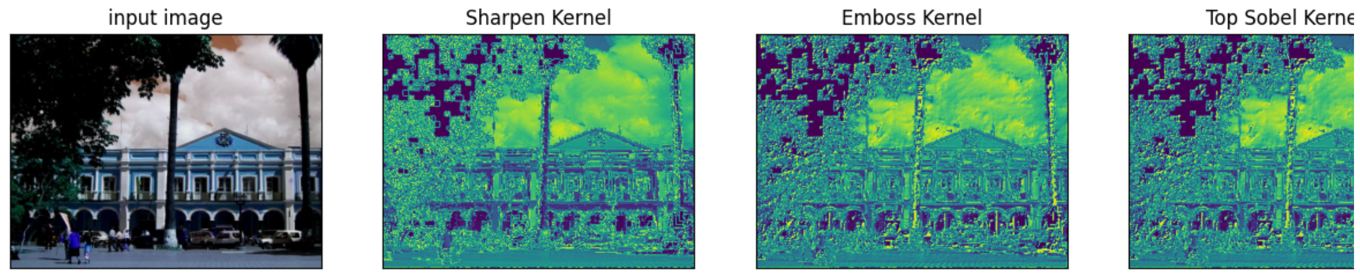


4.3 The Top Sobel Kernel

```
# kernel to be used to show differences in adjacent pixels

KERNEL3 = np.array([[ 1, 2, 1],[ 0, 0, 0],[ -1, -2, -1]])
image_Sobel = convolve2d(input_image, kernel=KERNEL3)

plt.imshow(image_Sobel)
pylab.show()
```

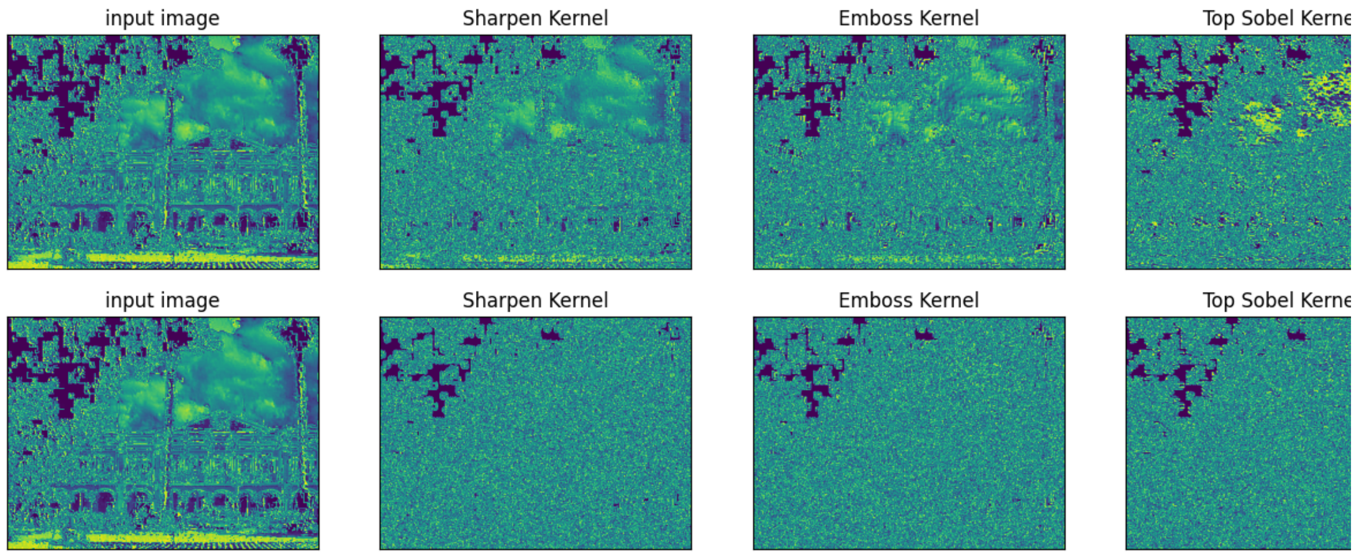


```
def featuremap(originalImage, sharpenImage, embossImage, sobelImage):
    pyplot.figure(figsize=(16, 8))
    # Subplot all the above kernel outputs
    ax = pyplot.subplot(1, 4, 1)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_title("input image")
    # plot filter channel in grayscale
    pyplot.imshow(originalImage)
    ax = pyplot.subplot(1, 4, 2)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_title('Sharpen Kernel')
    # plot filter channel in grayscale
    pyplot.imshow(sharpenImage)
    ax = pyplot.subplot(1, 4, 3)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_title('Emboss Kernel')
    # plot filter channel in grayscale
    pyplot.imshow(embossImage)
    ax = pyplot.subplot(1, 4, 4)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_title('Top Sobel Kernel')
    # plot filter channel in grayscale
    pyplot.imshow(sobelImage)
    pyplot.show()


featuremap(originalImage, image_sharpen, image_emboss, image_emboss)
```



Task 5. Refer to the multiple layers section in the tutorial material, repeatly apply the three kernels to the combined feature map that created by the previous round two times. Visualize the feature maps produced in the process

```
# write your code here

# Define the combined feature map as the input for the next round
input_image = image_sharpen + image_emboss + image_Sobel

# First round
output1_sharpen = convolve2d(input_image, kernel=KERNEL1)
output1_emboss = convolve2d(input_image, kernel=KERNEL2)
output1_sobel = convolve2d(input_image, kernel=KERNEL3)

# Second round
output2_sharpen = convolve2d(output1_sharpen, kernel=KERNEL1)
output2_emboss = convolve2d(output1_emboss, kernel=KERNEL2)
output2_sobel = convolve2d(output1_sobel, kernel=KERNEL3)

# Visualize the feature maps produced in the process
featuremap(input_image, output1_sharpen, output1_emboss, output1_sobel)
featuremap(input_image, output2_sharpen, output2_emboss, output2_sobel)
```



✓ 0s completed at 16:25