Table of Contents

INTRODUCTION

❏ K-Means Algorithm

This is the unsupervised learning technique which is used to classify data into clusters through different centroids. I am using this technique in order to find the points in an image and will group all similar points in one cluster so that different objects can be differentiated because points in the image are unlabelled and not categorized and the goal of this technique is to group all similar points into one cluster.

❏ Library CV2

CV2 is the python library which is used for several functions and it is designed to solve computer vision problems. I have used it for many tasks in this project like reading an image, displaying an image and finding corners of an image.

❏ Library Matplotlib

It is a visualisation or plotting library in python to plot graphs of arrays. It is beneficial to use because it gives access to visual data. It is based on different graphs like line, Bar, Histogram etc. I have used this library to show plot images in the console.

❏ Library Numpy

NumPy is the core library for computation.it provides high performance with multi-dimensional arrays. I have used this library to calculate the distance in the two-dimensional array and to calculate the mean/average of clustered points.

PROGRAM OUTLINE

❏ Methodology

Firstly, I imported some libraries such as CV2, Matplotlib and NumPy to run my script as built-in functions belong to these libraries. I have made use of various functions to perform multiple tasks. For Reading an image Cv2.imread() is used, for converting into grayscale image, cv2.cvtColor() is used, for finding corners in the image, cv2.goodFeaturesToTrack() is used. In this method I have given "0" as value in the arguments to find all detected points and "100" to find the strongest 100 points. Then in order to make circles for the points detected, cv2.circle() is used. For image display, plt.imshow() and cv2.imshow() is used.

For Calculation of K clusters, I have used the adopted the following method:

1. First, I have read an image, convert it into a grey scale and find corners with cv2.goodFeaturesToTrack() and store the length of points in a variable called "count".
2. I then declared and initialized K clusters as in my case K=3 and K=4.
3. I randomly selected 3 centroids for the first time where K=3.
4. I calculated distance with this function np.linalg.norm() and stored the each distance in arrDist1= [] , arrDist2= [] , arrDist3= [] and used append() function to store values in the list.
5. Then I labelled the nearest class as 1,2 and 3 and assigned the points to that specific class.
6. Stored new classified points into three different arrays as labelOnePoints=[] , labelTwoPoints=[] labelThreePoints=[]

7. Then I calculate the mean(average) of those classified points with np.mean(labelOnePoints,axis=0) function and store the new centroid in the new variable called newC1, newc2 and newc3.
8. This process is an iterative process and will go on until old centroids and new centroids are matched.

For Bounding Box, the following technique was used.

1. After performing the above mentioned steps, I have stored classified points to 3 different arrays.
2. Used cv2.boundingRect() to find the coordinates which are returned by this function as a, b, c, d. As labelled point array points are float64 so I have to convert it to float 32 as np.float32() as shown below:

   a,b,c,d = cv2.boundingRect(np.float32(labelOnePoints))
3. Using these coordinates, I have used this function to draw a box cv2.rectangle(img,(a,b),(a+c,b+d),(255,0,0),3)

❏ Implementing Harris Corner Detection:

Harris Corner Detector is a corner detection operator that is commonly used in computer vision algorithms to extract corners and infer features of an image. Harris' corner detector takes the differential of the corner score into account with reference to direction directly, instead of using shifting patches for every 45-degree angle, and has been proved to be more accurate in distinguishing between edges and corners.

The features in the image are all pixels that have large values of E(u,v), as defined by some threshold:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2$$

We have to maximize this function $E(u,v)$ for corner detection. That means, we have to maximize the second term. Applying Taylor Expansion to the above equation and using some mathematical steps, we get the final equation as:

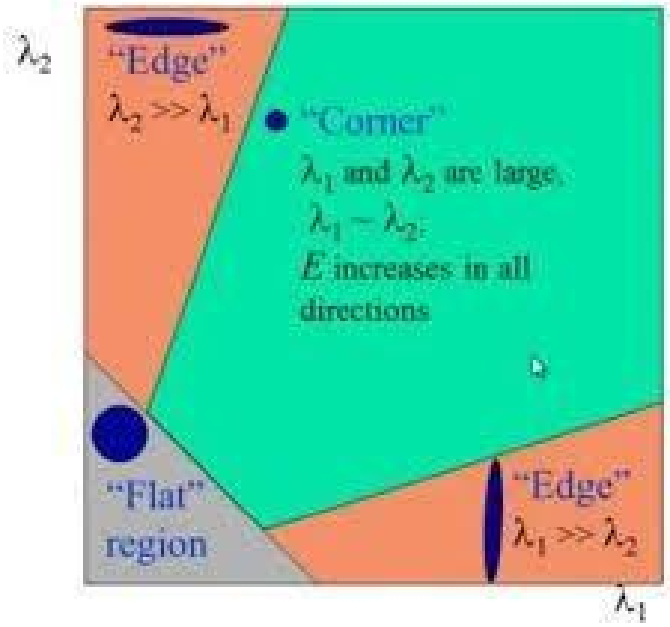
$$E(u,v) \approx [u \quad v] \left(\sum \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

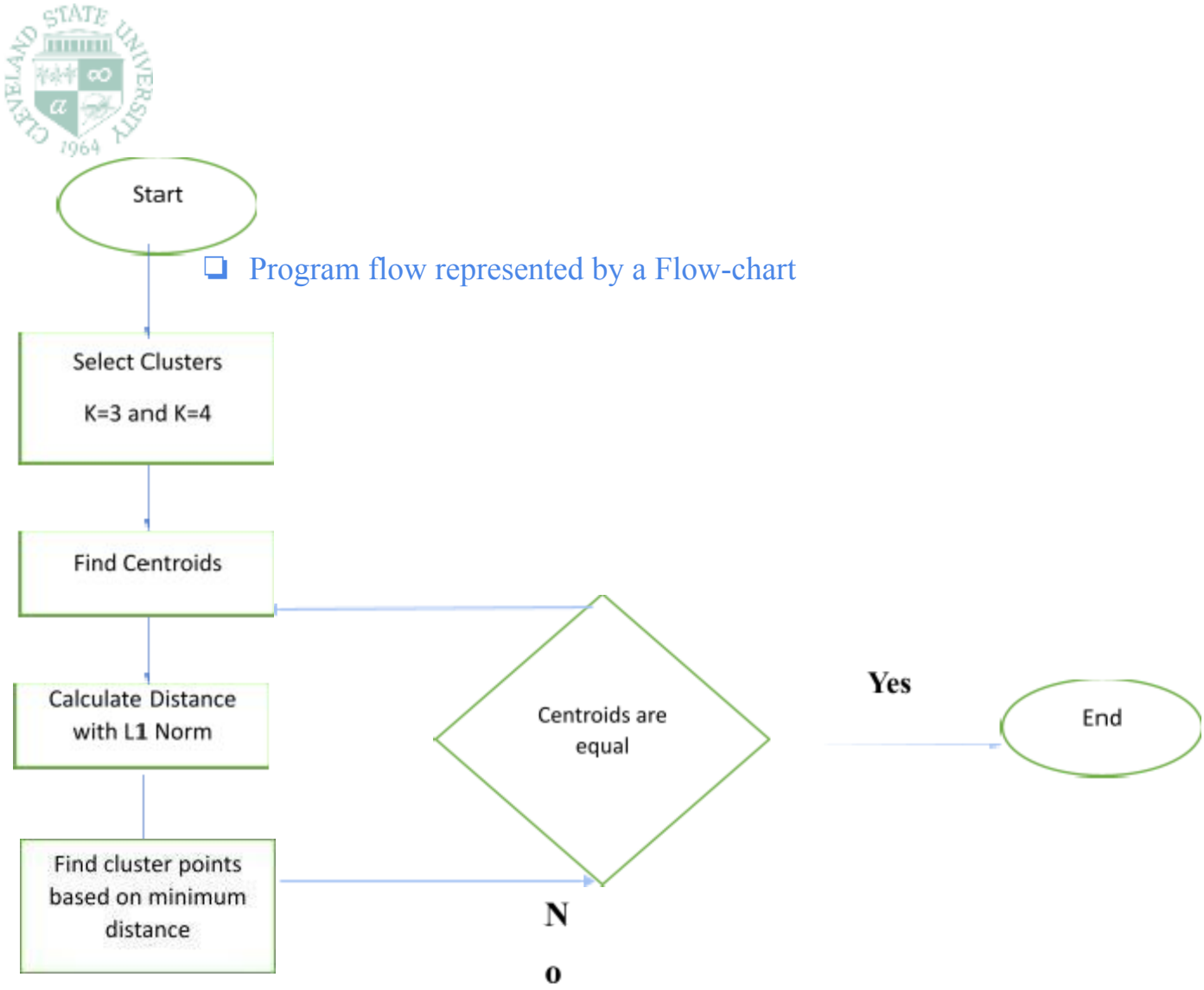
Now, we rename the summed-matrix, and put it to be M:

$$M = \sum w(x,y) \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix}$$

So the equation now becomes:

$$E(u,v) \approx [u \quad v] \, M \, \begin{bmatrix} u \\ v \end{bmatrix}$$





- Program flow represented by a Flow-chart

RESULTS:

Task 1 - Use Harris corner detection technique to find the data points (key points) of the enclosed image.

Task 2 - To view 100 Strongest detected points on the image

Task 3 - Compare the outcomes of multiple runs with different K and choose the best one based on a predefined criterion.

----------K=3----------

Iteration 1

[[254.94737 522.1316]] [[559.8571 635.2857]] [[437.69092 239.38182]]

Iteration 2

[[236.5 568.26666]] [[568.7368 577.7895]] [[387.82352 184.86275]]

Iteration 3

[[234.44827 575.62067]] [[568.7368 577.7895]] [[386.05768 188.13461]]

----------Following are new centroids----------

[[234.44827 575.62067]] [[568.7368 577.7895]] [[386.05768 188.13461]]

----------K=4----------

Iteration 1

[[282.08334 702.25]] [[399.73077 523.8077]] [[641.8 708.2]] [[363.0351 208.19298]]

Iteration 2

[[210.94118 641.35297]] [[394.44 488.04]] [[616.375 667.125]] [[386.04 180.26]]

Iteration 3

[[184.05882 619.35297]] [[391. 477.96]] [[597.2727 636.5455]] [[387.3617 169.65958]]

Iteration 4

[[184.05882 619.35297]] [[374.47827 477.69565]] [[594.7692 612.61536]] [[387.3617 169.65958]]

Iteration 5

[[180.1875 626.625]] [[330.42105 471.21054]] [[572.94446 583.3889]] [[387.3617 169.65958]]

Iteration 6

[[180.1875 626.625]] [[312.05884 478.94116]] [[568.7368 577.7895]] [[389.29166 173.08333]]

----------Following are new centroids----------

[[180.1875 626.625]] [[312.05884 478.94116]] [[568.7368 577.7895]] [[389.29166 173.08333]]

Task 4 - Display the original image with the best detected clusters on it (use different colours).

----------K=3----------

----------K=4----------

Task 5 - Display the original image with the bounding boxes on it (use different colours).

----------K=3----------

----------K=4----------

CONCLUSION:

I would like to summarise by comparing the results of various values of K. I think k=3 fits in better because it detects all three images whereas k=4 does not fit in because it considers one picture as two different pictures.

REFERENCES:

https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html

https://pythonprogramming.net/corner-detection-python-opencv-tutorial/

https://medium.com/programming-fever/color-detection-using-opencv-python-6eec8dcde8c7

https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1