IMAGE SEGMENTATION AND ITS TECHNIQUES

Kesha Chavda[1], Gauraja Deo[2], Sakshi Dhanu[3], Prof. Mohan Bonde[4]

[1,2,3] *Information Technology, Usha Mittal Institute of Technology, Mumbai, India*
[4]*Assistant Professor, Usha Mittal Institute of Technology, Mumbai, India*

***Abstract*— In digital image processing and computer vision, image segmentation is a process that involves separating a wide variety of images into various segments. The goal of this procedure is to simplify and improve the representation of an image. The role of image segmentation is very important in image processing. The partitioning of an image into multiple segments makes it easier for further process. Thus, after completing the operations, the image will be re-joined. Segmentation increases the accuracy of recognizing the object in an image and reduces the loss. Semantic segmentation and Instance segmentation are the types of image segmentation established on the problem we use image segmentation.**

***Index Terms*— LSTM, SVR, Linear Regression, Sentimental Analysis, Stock Market Prediction**

I. INTRODUCTION

An image contains a lot of useful information. Understanding the image and extracting information from the image to accomplish something is an application of digital image technology. Therefore, the start in understanding the image is Image Segmentation. In practice, it's often not inquisitive about all parts of the image, but just for some certain areas which have identical characteristics. Image segmentation is a vital basis for image recognition. It is based on the group to divide an input image into a number of the same nature of the group in order to obtain the area in which people are interested. And it's the idea behind Image Analysis and understanding of Image Feature Extraction and Recognition. Image Segmentation is a way to partition an image into numerous segments (subgroups) that help in reducing the intricacy of the image, hence making the analysis of the image easier. Various algorithms are used to allocate a certain set of pixels together to form an image. We are basically assigning labels to the pixels by doing so. Pixels with the same tag fall under a category where they have some or the other thing familiar in them. Using these labels, we are able to specify boundaries, draw lines, and separate the foremost required objects in a picture from the remainder of the not-so-important ones.

II. PROPOSED SYSTEM

The aim of this project is to compare various Image Segmentation techniques with Machine Learning, using OpenCV, NumPy, TensorFlow, Keras, Scikit-Image, Python Image Library (Pillow/PIL), and other python libraries, that would help detect and identify the objects present around one's surroundings and compare their results.

III. RELATED WORK

In [1], the K-mean technique was used to implement image segmentation on the image. An RGB image was transformed into l*a*b* colour space because the RGB image was very large. They concluded that in the K-mean algorithm, the number of clusters was very important. If the number of clusters was very high or very low, then the result was not so good. K-mean showed every cluster in a new window, and it made it easier to analyse the image for further information. In Köhler's method [2], Adaptive thresholding was one of the most frequently used techniques in many applications because it was fast to evaluate and when merged with previous filters, it gave sturdy decision rules for pattern recognition. In [3], the proposed segmentation system provided a complete solution for both unsupervised and supervised segmentation of colour images built on neural networks. In the system, unsupervised segmentation was implemented by SOM-based colour reduction and SA-based colour clustering. The supervised segmentation was achieved by HPL learning and pixel classification. The system proposed in [4], presented and compared different criteria to optimize segmentation parameters, when examples are available. They also exposed another way to take advantage of ground truth, in changing the data space before applying the segmentation algorithm. It was shown that using this knowledge to guide the segmentation enables to produce better results, even better than manually produced segmentation by an expert. The paper [5] mainly focused on the study of the soft computing approach to edge detection for image segmentation. The soft computing approaches namely, fuzzy-based approach, Genetic algorithm-based approach, and Neural network-based approach were applied on a real-life example image of a nature scene and the results showed the efficiency of image segmentation. In [6] they proposed a conceptually easy, pliable, and general framework for object instance segmentation. Their approach efficiently detected objects in an image while at once generating a high-quality segmentation mask for each instance by using Mask R-CNN. Mask R-CNN extends Faster R-CNN by appending a branch for predicting an object mask aligned with the existing branch for bounding box recognition. The paper [7]

proposed that the histogram thresholding was proposed to enhance image segmentation. Thresholding of the final histogram is done relatively easy with the definition of a low pass filter and the amplification and attenuation of the peaks and valleys respectively or the standard deviation of the presumed Gaussian modes.

IV. IMPLEMENTATION

1. Region-based Segmentation

We started our implementation by installing and importing some very important modules namely NumPy, Matplotlib, Scikit-image, TensorFlow, Keras and OpenCV.

We then imported the Mask-RCNN or the MRCNN model. And then we imported the Coco dataset. COCO signifies Common Objects in Context. It is wide-ranging object detection, segmentation, and captioning dataset. It presents a thorough statistical analysis of the dataset in comparison to PASCAL, ImageNet, and SUN. And it also provides baseline performance analysis for bounding box and segmentation detection results. We then defined the path for the pretrained weights and the images on which we performed the segmentation.

We then created an inference class that was used to infer the Mask R-CNN model. The inference is the method of taking a model and implementing it onto a machine, which will then process inbound data to look for and classify whatever it has been trained to perceive. The inference is the step in which a trained model is used to foresee the testing.

```
Configurations:
BACKBONE_SHAPES                [[256 256]
 [128 128]
 [ 64  64]
 [ 32  32]
 [ 16  16]]
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     1
BBOX_STD_DEV                   [ 0.1  0.1  0.2  0.2]
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.5
DETECTION_NMS_THRESHOLD        0.3
GPU_COUNT                      1
IMAGES_PER_GPU                 1
IMAGE_MAX_DIM                  1024
IMAGE_MIN_DIM                  800
IMAGE_PADDING                  True
IMAGE_SHAPE                    [1024 1024    3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.002
MASK_POOL_SIZE                 14
MASK_SHAPE                     [28, 28]
MAX_GT_INSTANCES               100
MEAN_PIXEL                     [ 123.7  116.8  103.9]
MINI_MASK_SHAPE                (56, 56)
NAME                           coco
NUM_CLASSES                    81
POOL_SIZE                      7
POST_NMS_ROIS_INFERENCE        1000
POST_NMS_ROIS_TRAINING         2000
ROI_POSITIVE_RATIO             0.33
RPN_ANCHOR_RATIOS              [0.5, 1, 2]
RPN_ANCHOR_SCALES              (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE              2
RPN_BBOX_STD_DEV               [ 0.1  0.1  0.2  0.2]
RPN_TRAIN_ANCHORS_PER_IMAGE    256
STEPS_PER_EPOCH                1000
TRAIN_ROIS_PER_IMAGE           128
USE_MINI_MASK                  True
USE_RPN_ROIS                   True
VALIDATION_STEPS               50
WEIGHT_DECAY                   0.0001
```

Fig. 1.1: Model Configuration

We can see the various specifications of the Mask R-CNN model that we have used. The backbone is Resnet-101. ResNet, is an ideal neural network used as a backbone for various computer vision tasks.

ResNet-101 is a convolutional neural network that is 101 layers deep. The pretrained network can classify images into many groups, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a variety of images. The network has an image input size of 1024-by-1024.

The mask shape that will be restored by the model is 28X28, as it is trained on the COCO dataset. And we have an overall 81 classes (including the background). We can also see various other statistics as well, like the input shapes, the Number of GPUs to be used, and Validation steps, among other things.

```
class_names = ['BG', 'person', 'bicycle', 'car', 'motorcycle', 'airplane',
               'bus', 'train', 'truck', 'boat', 'traffic light',
               'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird',
               'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear',
               'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie',
               'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
               'kite', 'baseball bat', 'baseball glove', 'skateboard',
               'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup',
               'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
               'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza',
               'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed',
               'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote',
               'keyboard', 'cell phone', 'microwave', 'oven', 'toaster',
               'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors',
               'teddy bear', 'hair drier', 'toothbrush']
```

Fig. 1.2: COCO classes

Next, we created our model and loaded the pretrained weights. The model is pre-trained on the COCO dataset. This dataset includes overall 80 classes (plus one background class) that you can detect and segment from an input image (with the first class being the background class). We then defined the classes of the COCO dataset which will assist us in the prediction phase.

A random image was loaded and then the objects were detected within that image. An abounding box was created around each object and each object was identified correctly with great accuracy.

Fig. 1.3: RCNN Output

2. Thresholding-based Segmentation

A. Simple Thresholding

We started off the implementation by importing a few modules: NumPy, matplotlib, and Skimage. We then loaded the image and displayed it.

We want the pixels belonging to the shapes to be "on," while turning the remaining pixels "off," by setting their colour channel values to zeros. The skimage library has several different methods of thresholding. We started with the only version, which involves a crucial step, i.e., human input. Specifically, during this simple, fixed-level thresholding, we must provide a threshold value t. We then loaded the original image, converted it to grayscale, and de-noised it.

Grayscale images contain pixel values within the range from 0 to 1, so we are trying to find a threshold therein closed range and the geometric shapes are "darker" than the white background so, a method to work out a "good" value for t is to refer the histogram of the image and inspect to identify what grayscale ranges correspond to the shapes within the image or the background. Since the image features a white background, most of the pixels within the image are white. This corresponds nicely to what we see within the histogram: there is a peak near 1.0.

If we might wish to pick the shapes and not the background, we would like to show off the white background pixels, while leaving the pixels for the shapes turned on.

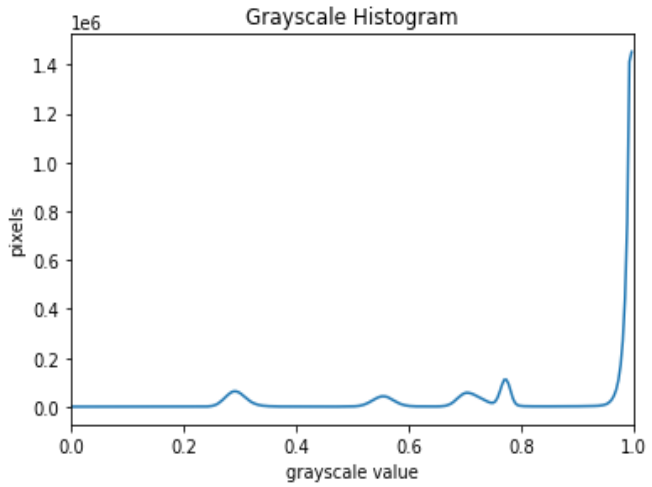

Fig.2.1: Plotting Histogram of the grayscale and de-noised image

So, we should always choose the value of 't' somewhere before the massive peak. So, we chose t as 0.9. To apply the threshold, we used the comparison operators to make a mask. We want to show "on" all pixels which have values smaller than the threshold, so we use the less than operator and the operator returns a mask. It has just one channel, and every one of its values is either 0 or 1. The areas where the shapes were within the original image are now white, while the remainder of the mask image is black. We can now apply the binary mask to the original-coloured image and what we are left with is only the coloured shapes from the original image.

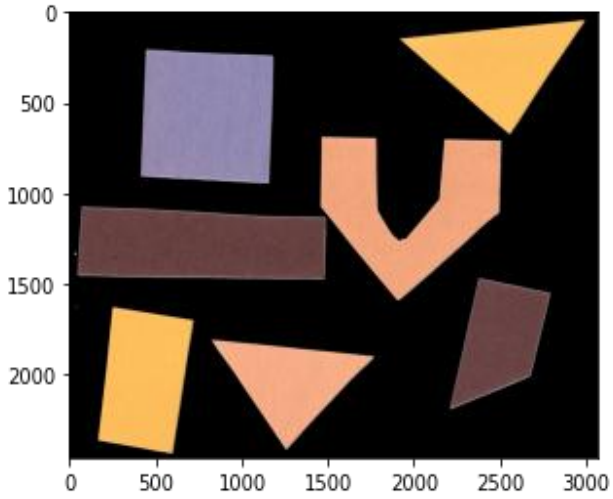

Fig. 2.2: Applying binary mask and showing the foreground

B. Automatic Thresholding

As ahead, we started off by importing the same modules. We also loaded the image and displayed it. We also used the Gaussian blur to denoise the image and colluded a histogram of the denoised image. The histogram has a significant peak around 0.2 and an alternate, lower peak veritably near1.0. So, we will say that this image may be a good candidate for thresholding with Otsu's system. The Otsu's system finds a threshold value linking the 2 peaks of a grayscale histogram.

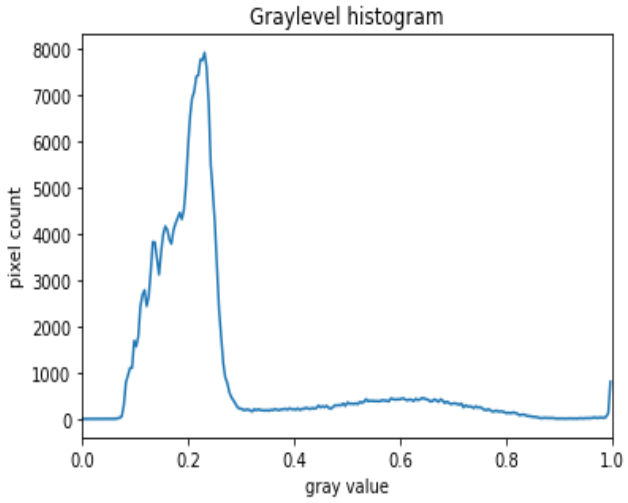

Fig. 2.3: Blurring, de-noising the Image, and plotting its Histogram

We've automatic thresholding strategies that can determine the threshold automatically for us. One similar system is Otsu's system. It's particularly useful for situations where the grayscale histogram has two or further peaks that resemble the background and objects of interest.

The Otsu function from the Skimage library can be used to ascertain the threshold automatically through Otsu's method. Also, comparison operators can be used to apply it. For this image, the estimated threshold value is0.4113. Now we can produce a binary mask. As we've seen ahead, pixels above the threshold value are going to be turned on, and those below the edge are going to be turned off. Eventually, we use the mask to choose the focus.

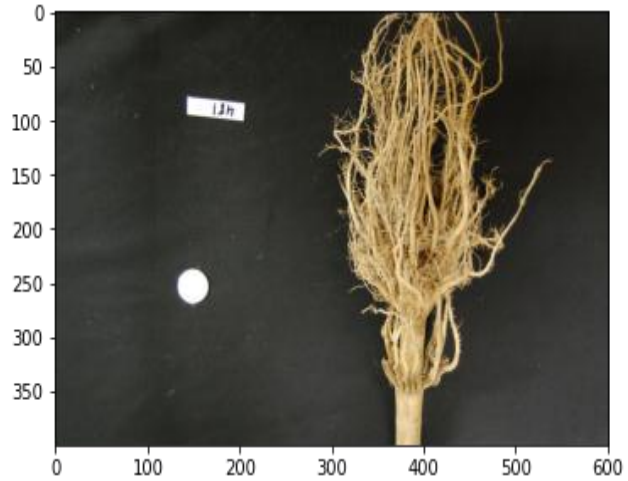

Fig. 2.4: Applying binary mask and showing the forepart

3. Clustering-based Segmentation

A. K-means Clustering

We started off the implementation by importing a few modules namely NumPy, MatplotLib, and OpenCV. We then read the image and transformed it to an RGB image. We then organised the data for the K-Means process. The image was a 3-dimensional but to utilize k-means clustering on it we required to reform it to a 2-dimensional array. We used the NumPy reshape() function for this.

We designed a criterion for the algorithm to stop executing, which will occur if 100 iterations are executed or the epsilon (which is the required accuracy) inclines to 85%. We then performed the k-means clustering with total number of clusters as 3, and random centres were randomly chosen for k-means clustering. We then converted the data into 8-bit values, reshaped the data into the original image dimensions, and plotted it.

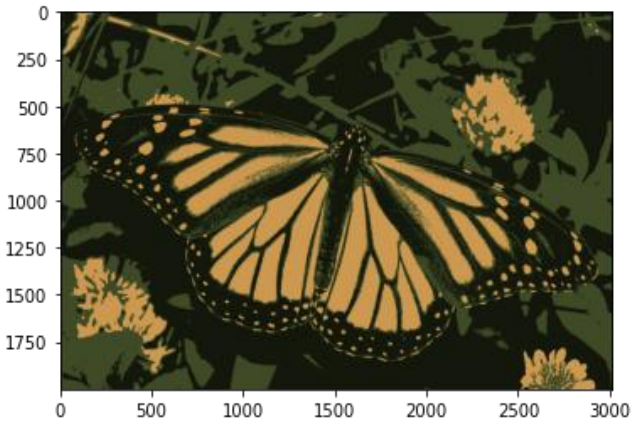

Fig 3.1: Image after K-Means, k=3

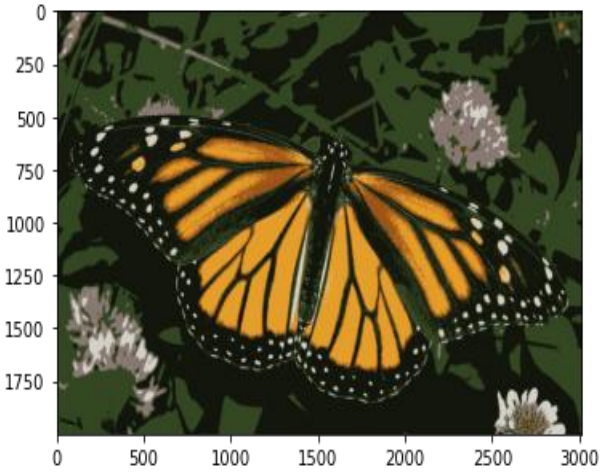

Fig 3.2: Image after K-Means, k=6

B. Fuzzy C-means Clustering

In Fuzzy - c means clustering we started off the implementation by importing a few modules namely NumPy, MatPlotlib, PIL, Skimage, Skfuzzy, and Google.Collab.Patches and OpenCV. We then read the image and transformed it to an RGB image.

The image was a 3-dimensional but to utilize k-means clustering on it we required to reform it to a 2-dimensional array, similar to k means clustering. We used the NumPy reshape() function for this. We used the transpose function to focus the cluster prediction based on the colour pixels which is RGB format rather than other features.

Fig. 3.3 Image after applying Fuzzy C-means, k=6

We then organised the data for the Fuzzy-c means algorithm. We created a change_color_fuzzycmeans() function to predict the features of particular pixels and append them to an array.

We then performed the fuzzy-c means clustering with a number of clusters defined as 6, and various other parameters were chosen for fuzzy-c means clustering. Finally, we reshaped the finalized data into the original image dimensions and plotted it.

Fig. 3.4: Image after applying Fuzzy C-means, k=10

4. Edge-based Segmentation

A. Gradient-based Segmentation

Under Gradient-based segmentation, we first implemented the Sobel operator followed by the Prewitt Operator. We started off the implementation by importing a few modules: NumPy, MatplotLib, OpenCV, and PIL.

We then loaded the image and converted it to grayscale. We then rounded the pixel values to their nearest integers, in this case, 0s and 1s. We then assigned variables h, w to the image's respective height and width. And defined the Gx and Gy kernels.

```
# Defining filters
horizontal = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])  # Gx
vertical = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])  # Gy
```

Fig. 4.1: Sobel Kernel

```
# Defining filters
horizontal = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])  # Gx
vertical = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])  # Gy
```

Fig. 4.2: Prewitt Kernel

Then by using loops, we applied both the horizontal and vertical filter to the image. And, with the new Gx and Gy values, we calculated the gradient magnitude and displayed the image.

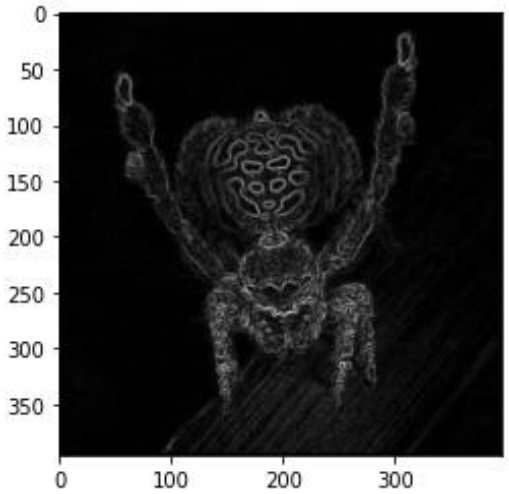

Fig. 4.3: Output image after applying Sobel filter

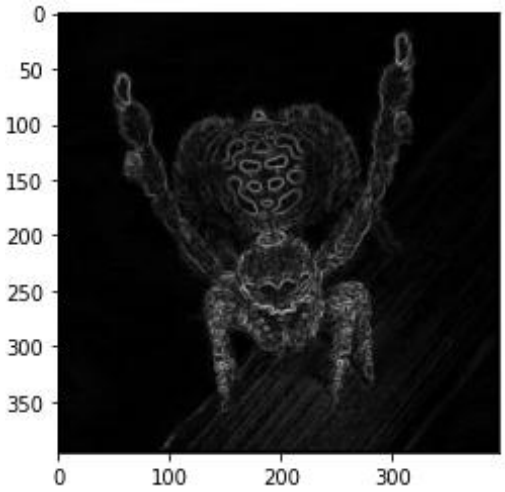

Fig. 4.4: Output image after applying Prewitt filter

B. Gaussian-based Segmentation

Under Gaussian-based segmentation, we first implemented the Laplacian operator followed by the Canny Operator. We started off by importing the cv2 and MatplotLib modules and reading the input image. We then applied the Gaussian filter to denoise the image and converted the de-noised image to grayscale. Using the in-built Laplacian and Canny filters, we detected the edges.

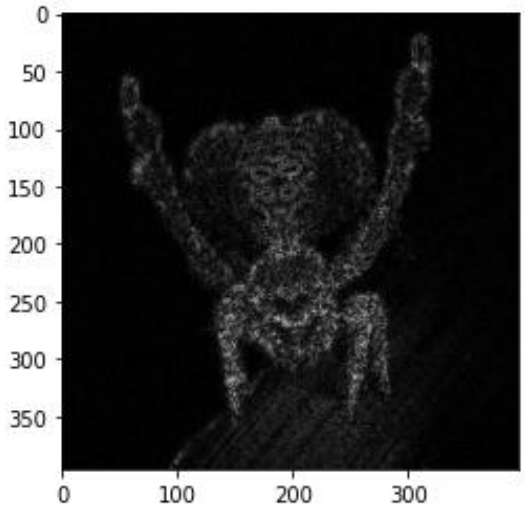

Fig. 4.5: Output image after applying Laplacian filter

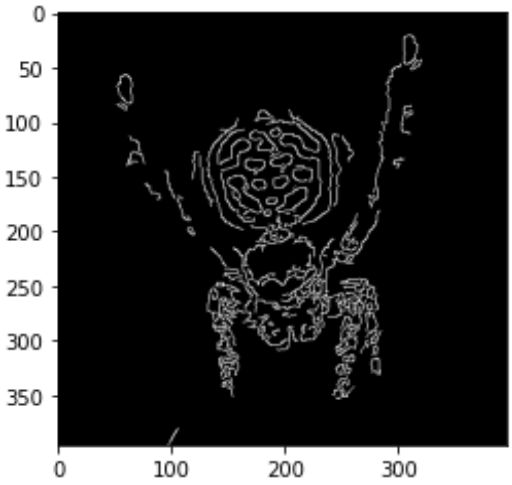

Fig. 4.6: Output image after applying Sobel filter

V. CONCLUSION

We implemented 4 types of Image segmentation techniques i.e.

Region-based segmentation where we learned how to use Mask R-CNN to perform instance segmentation. Contrary to object detection, which only gives you the bounding box (x, y)-coordinates for an object in an image, instance segmentation takes it a step further, complying with pixel-wise masks for each object. By applying instance segmentation, we actually segment an object from the input picture. One of the most prominent applications of region-based segmentation is objection detection and recognition.

Thresholding-based segmentation is used to convert a multilevel/grayscale image into a binary image. The advantage of acquiring first a binary image is that it minimizes the complexity of the data and simplifies the process of recognition and classification. The binary images acquired by thresholding are held in two-dimensional NumPy arrays as they need just one colour value channel. they're Boolean, and accordingly contain the values 0 (off) and 1 (on). Thresholding-based segmentations are used in cases where the foreground needs to be separated from the background, therefore, they are mostly used for object detection.

Clustering-based segmentation during which we used (a) K-means algorithm which is an iterative algorithm that tries to divide the dataset into K pre-defined clear non-overlapping subgroups (clusters) where each data point belongs to just one group. It aims to form the intra-cluster data points as alike as possible while also keeping the clusters as contrasting (far) as possible. (b) Fuzzy-C means an algorithm that works by assigning membership to each data point corresponding to each cluster centre on the basis of the distance between the cluster centre and the data point. The more the data is nearby to the cluster centre, the

closer its membership is close to the precise cluster centre. Clustering-based segmentation are primarily used for pattern recognition and data analysis.

Edge-based segmentation remits to the process of identifying and locating sharp discontinuities in an image. The discontinuities are instantaneous changes in pixel intensity that distinguish the boundaries of objects in a scene. In this technique, we implemented the Sobel Operator, Prewitt Operator, Canny Operator, and Laplacian of Gaussian. From the experiment performed, it was observed that the Canny result is the superior one when compared to the other detectors for the selected image since different edge detectors work better under different conditions. Edge-based segmentation techniques are used where finding out edges is important such as fingerprint sensors.

VI. REFERENCES

[1] B. Tanwar, R. Kumar, and G. Gopal, "Clustering Techniques for Digital Image Segmentation", vol. 7, no. 12, pp. 55–60, 2016.

[2] Ralf Kohler, "A segmentation system based on thresholding, Computer Graphics and Image Processing", Volume 15, Issue 4, ISSN 0146-664X, 1981.

[3] Guo Dong and Ming Xie, "Learning for image segmentation based on neural networks," in IEEE Transactions on Neural Networks, vol. 16, no. 4, pp., July 2005.

[4] I. Levner and H. Zhang, "Classification-Driven Watershed Segmentation," in IEEE Transactions on Image Processing, vol. 16, no. 5, pp., May 2007.

[5] N Senthilkumaran and Rajesh, Reghunadhan, "Edge Detection Techniques for Image Segmentation - A Survey of Soft Computing Approaches", International Journal of Recent Trends in Engineering, November 2007.

[6] Kaiming He, Georgia Gkioxari, Piotr Doll´ar, and Ross Girshick,"Mask RCNN" ,IEEE International Conference on Computer Vision (ICCV), 2017.

[7] P.Daniel Ratna Raju and G.Neelima, "Image Segmentation by using Histogram Thresholding", 2012.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for largescale image recognition", 2015.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep Residual Learning for Image Recognition", December 2015