SCHOOL OF PURE APPLIED AND HEALTH SCIENCES

DEPARTMENT OF COMPUTING AND INFORMATION SCIENCES

PROJECT TITLE: GRABCUT IMAGE SEGMENTATION SYSTEM

BY

ACHIENG BEATRICE

SB06/SR/MN/9914/2019

Email: achieng9914@student.mmarau.ac.ke

SUPERVISED BY MRS.DEBORA

A RESEARCH PROJECT REPORT SUBMITTED TO THE SCHOOL OF PURE, APPLIED AND HEALTH SCIENCES, DEPARTMENT OF COMPUTING AND INFORMATION SCIENCES, IN PARTIAL FULLFILMENT OF THE REQUIREMENT FOR THE AWARD OF A BACHELORS DEGREE IN COMPUTER SCIENCE

MAASAI MARA UNIVERSITY

APRIL 2023

DECLARATION

This project report is my original work done by me under the guidance and supervision of Madam Deborah, and I am responsible for the work submitted in this project. Except as noted in the references and acknowledgement, this is my original work and the original work in this project have not been submitted to any other university or institute but Maasai Mara University.

NAME: ACHIENG BEATRICE

SIGNATURE: ………………………

DATE: …………………………

APPROVAL

This project has been submitted with my approval as the university student supervisor.

NAME: MADAM DEBORAH

SIGNATURE: ……………………

DATE: …………………………

ABSTRACT

Image segmentation is a critical task in computer vision that involves partitioning an image into multiple regions or segments. In this project, I proposed a machine learning based approach for image segmentation that leverages the power of convolutional neural networks(CNNs). I use a modified version of the U-Net architecture, which has been shown to achieve state-of-the-art performance in various of image segmentation tasks. My approach takes a raw input image outputs pixel-wise segmentation map, where each pixel is assigned a label corresponding to its segment. I evaluate my approach on several benchmark datasets and compare my result with other state-of-art image segmentation methods. My experimental results show that the proposed algorithm achieve high accuracy and outperforms other methods in terms of speed and accuracy. I also demonstrate the robustness of my algorithm by evaluating it on noisy or low-quality images. To train this model, I use a large dataset of labeled images and employ augment techniques to improve the generalization ability of the model. To improve the training efficiency and reduce the risk of over-fitting, I use a transfer learning by fine-tuning a pre-trained CNN.

Generally, my approach represents a significant improvement over existing image segmentation methods and has potential to be applied in a wide range of computer vision applications such as medical imaging, autonomous driving and robotics. The goal of the image segmentation is to simplify and change the representation of an image into something that is more meaningful and easier to analyze. According to a human perception image segmentation is the process of dividing the image into non- overlapping meaningful regions. The main objective of image segmentation is to divide an image into many sections for the further analysis, so we can get the only necessary or a segment of information. The partitioning the image will be based on some image features like **color, texture, pixel intensity value** etc. There are several techniques of image segmentation like **thresholding method, region based method, edge based method, clustering methods and the watershed method**. In this paper we will see some segmentation methods and what are the necessary things we should know while doing segmentations. We will also check some papers and analyze which method is best for image segmentation. Keywords: perception, segmentation, texture, edge, clustering.

ACKNOWLEDGEMENT

First, I thank the Almighty God for the blessings of good health, protection and guidance through my studies and sufficient grace and love from Him and for wonderful creation He gave me during the study period that made me a good young mother.

Secondly, I would like to extend my gratitude to my colleagues, mentors, and advisors, whose guidance and support have been invaluable throughout my journey in studying and applying image segmentation techniques. Their expertise and encouragement have inspired me to delve deeper into this field and strive for excellence.

Lastly, I also thank my parents and siblings for their support through my stay here in the University and contributions and encouragement they give during the study and development of the system.

IMAGE PROCESSING OVERVIEW

The optic information is a vital information in the image recognize, refined and processed by the human brain. The half of the cortical part of the human brain is assigned to optics (which process the information). There are many applications of image processing. We use image processing in many fields such as medical field, image sharpening and restoring, robot vision, pattern recognition etc.

The image sharpening and restoring refers the image which we have captured by our digital after that we can manipulate it to get the desired results. There can be performed several operations like **zooming, blurring, sharpening, detecting edges, gray scale to color conversion**.

Another example we can take it in **self-driving cars**. It will detect hurdless. It is the usual operation for image processing by detecting the objects and then based on the image it will calculate the distance how far it's from here. It's not like directly it will predict, first the model will be trained based on that training data it will predict easily. The latency will be very low so it will recognize very fast.

There are several methods to detect object in an image such as points, lines and edges, curve. These methods can be implemented using several threshold techniques. These thresholds can be fixed based on the region-oriented segmentation approaches. The segment itself is a challenge since making a machine behave or act like human brain is challenging. The machine must be able to see and understand image as human being do.

DEFINITION OF TERMS

Image segmentation; is an image processing task in which the image is segmented or partitioned into multiple regions such that the pixels in the same region share common characteristics.

Classification; is a fundamental task in machine learning and data mining that involves categorizing or assigning predefined labels or classes to input data.

Clustering; is a technique used in unsupervised learning, where the goal is to group similar data points together based on their inherent patterns or similarities.

Segmentation; refers to the process of dividing an image or a dataset into multiple regions or segments based on certain criteria.

F1 score; is a metric commonly used in binary classification tasks to evaluate the performance of a classification model. It combines precision and recall into a single value to provide a balanced assessment of the model's effectiveness.

Labels; refer to the predefined categories or classes assigned to data instances.

Grabcut; is a graph-cut-based method that aims to automatically extract foreground objects from images.

Table of Contents

CHAPTER ONE

INTRODUCTION

Image segmentation is a most important part in the image processing in computer vision, it is used almost everywhere to process the images so our model should be able to recognize what's inside the image. The segmentation splits the image into many sections or objects. The level to which the splitting the image is being carried rely on the problem is which has been solved. When the object of an image has been segregated, segmentation should stop on that time.

The purpose of image segmentation is to simplify the image data into more meaningful and manageable information for further analysis.

For example we have an image so our aim is to recognize the objects into the image, for that we segment the image the details should be visible so our model work if there is a presence of an outliers or the paths aren't clear or broken. It's of no use to segment the image if it may not be able to identify those elements.

The base of an image segmentation is having two basic properties. Either it will be discontinuity or likewise.

- In this, the first division is the approach of an image partition is **based on sudden changes**. We can take an example of edges in an image.
- While in another category it is segmenting an **image based on region similarities according to predefined criteria.** Like thresholding, whether the region is growing or splitting or merging.

What is image segmentation.

Image segmentation is an image processing task in which the image is segmented or partitioned into multiple regions such that the pixels in the same region share common characteristics.

Image segmentation refers to the process of partitioning a digital image into multiple segments like set of pixels and the pixels in a region are similar according to some homogeneity criteria such as **color, intensity or texture**. It also defined as a process of partitioning an image into homogeneous groups. Each region is homogeneous according to similar characteristics.

There are two forms of image segmentation:

1. **Local segmentation** – It is concerned with a specific area or region of the image.
2. **Global segmentation** – It is concerned with segmenting the entire image.

Modes and types of segmentation

Image segmentation is divided into two groups, which include:

1. **Semantic segmentation** – This refers to the process of detecting class labels for every pixel. For example, in a picture with cars and people, cars will be segmented as one object and people as another object.
2. **Instance Segmentation** – This refers to the process of detecting an instance of an object for every pixel. For example, in a picture with cars and people, each car will be detected as an individual object as well as every person.
   In computer vision, this is useful because it can be applied in several areas including controlling traffic by counting the number of cars.

Levels of image analysis

There are three levels of image analysis;

- **Classification** – categorizing the entire image into a class such as people, animal, outdoors.
- **Object detection** – detecting objects within an image and drawing a rectangle around them, for example a person or sheep or cat etc.
- **Segmentation** – identifying parts of the image and understanding what object they belong to.

Segmentation lays the basis for performing object detection and classification.

Problem statement

Despite the significant progress in the image segmentation over the past few years, image segmentation has remains a challenging task in computer vision. The complexity of images, variation in lighting conditions and the presence of noise and occlusion to pose a significant challenge to image segmentation. Consequently, there is a need for more accurate and efficient image segmentation algorithms to improve the quality and reliability of computer vision applications like GrabCut algorithm which aims to solve is that of accurately segmenting an object of interest in an image by separating it from the background. This problem is particularly challenging when the object of interest has complex shape, texture, or color, and when the background has similar features. GrabCut algorithm addresses this problem by modeling the foreground and background color distribution using Gaussian mixture models

General objectives

To implement an algorithm that handle complex objects with irregular shapes, textures, and colors by incorporating spatial information and pixels correlations into the model and perform background and foreground feature extractions.

Specific Objective

- To analyze accurate segmentation of the object of interest by effectively modeling the foreground and background color distribution using Gaussian mixture models.
- To implement flexible framework that can be easily adapted to different image segmentation tasks such as video segmentation, medical image analysis or object recognition.
- To model the foreground and background color distribution using Gaussian mixture model based on the pixels inside and outside the initial rectangle/mask.

- To converge to a stable segmentation by defining a convergence criteria based on the change in segmentation between iterations
- To iteratively refine the segmentation by updating the foreground and background models based on the current segmentation and recomputing the probability map and segmentation.
- To subdivide an image into its constituent parts and extracts those parts of interest or objects and review the state-of-art image segmentation algorithms.

Research questions

1. How can we improve the accuracy and efficiency of image segmentation algorithms?
2. What are the limitation of current image segmentation techniques and how can they be addressed?
3. How can unsupervised learning methods be used for image segmentation without the need for manually labeled training data?
4. Can image segmentation be used in real-time application such as autonomous driving or robotics with high accuracy and low latency?
5. How can image segmentation be used for image restoration such as removing noise or artifact from the image?

Justification

Image segmentation is an important step in computer vision. Machines need to divide visual data into segments for segment-specific processing to take place which will contribute to the development of more and efficient image segmentation algorithms. Image segmentation thus will improve the quality and reliability of computer vision and finds its way in prominent field like robotics, medical imaging, autonomous vehicle and intelligent video analytics.

Scope and Limitations

This research proposal will focus on the development of an image segmentation algorithm that can handle complex images with varying lighting conditions, noise and occlusions. The proposed algorithm with be evaluated using benchmark datasets and compared against existing state-of-the-art algorithms.

Over segmentation when the image is noisy or has intensity variation, cannot distinguish the shading of the real images and power and time consuming hence classical segmentation cannot be applied hence a preprocessing step including color calibration and noise removal are highly recommended.

Segmentation also has its limitations as it needs to be implemented in the proper manner. As segmentation is one of the most important process in the marketing plan or for your business, you need to know the limitations of segmentation and what pitfalls lie ahead if you go wrong with your target market segment.

1) **Segments are too small** – if the chosen segment is too small then you will not have the proper turnover which in turn will affect the total margins and the viability of the business.

2) **Consumers are misinterpreted** – the right product to the wrong customers. What if your market research says that your customers want a new soap and you come out with a new facial cream. The concept is same, cleanliness. But the concept is completely different.

3) **Costing is not taken into consideration** – targeting a segment is ok but you also need to know how much you will have to spend to target a particular segment. If it is a Sec A segment and you do not have the budget to be present in the places the Sec A customer visits, then your segmentation strategy is a failure.

4) **Presence of noise and occlusion** – this affect the accuracy of the segmentation

5) **Computation complexity** – some of the existing algorithms limits image segmentation application in real-time systems.

6) T**here are too many brands** – Along with segmentation, you also need to check out the competition offered in the same segment from other products. Getting into a segment already saturated will mean higher costs and lesser profit margins.

7) **Consumer are confused** – If the consumer himself doesn't know whether he will be interested in a particular product or not, than that's a sign that you need to get out of that segment / product.

8) **Product is completely new** – If a product is completely new than there is no market research to base your segmentation on. You need to market it to the masses and as acceptance increases, only then will you be able to focus on one particular segment.

CHAPTER TWO

LITERATURE REVIEW

In this section, I provide a review of the existing literature on image segmentation.

2.1 Proposed algorithm

2.1.1 GrabCut algorithm

GrabCut is a popular image segmentation algorithm that was introduced by Carsten Rother, Vladimir Kolmogorov, and Andrew Blake in 2004. This algorithm is based on the idea of graph cuts and uses an interactive process to extract the foreground from the background of an image. In this literature review, we will discuss the key features, strengths, and weaknesses of the GrabCut algorithm, along with some of its applications in the field of computer vision.

GrabCut algorithm is used to partition an image into foreground and background regions. The algorithm works by iteratively refining an initial segmentation using combination of user-provided input and machine learning. The algorithm is based on the graph cut technique, where the image is represented as a graph and the foreground and background regions are separated using graph cuts.

The algorithm begins by taking a user-defined bounding box around the object of interest and then automatically initializes the foreground and background regions based on this bounding box. It then iteratively updates the probability distribution of the foreground and background region using a Gaussian Mixture Model (GMM) and computing the graph cut to separate the two regions. The GrabCut algorithm has been widely used for variety of application including object recognition and image editing.

One of the key advantages of the GrabCut algorithm is its interactive nature and its ability to accurately segment complex images. The user can refine the segmentation by providing additional input such as making pixels as foreground or background. This makes the algorithm particularly useful for applications where the object of interest is complex and difficult to separate from the background.

Another notable application of the GrabCut algorithm is in medical image analysis. For example, in a 2010 paper by Rui Li Et Al, the GrabCut algorithm was used for the segmentation of brain tumor images. The authors compared the performance of the algorithm with other segmentation techniques and found that it provided better results.

The GrabCut algorithm has also been extended to handle color images as describe in 2019 paper by Tingwei Wang. The author proposed an extension of the algorithm that incorporates color information in addition to spatial information. They showed that their algorithm outperformed the original GrabCut algorithm on a dataset of color images.

Another extension of the GrabCut algorithm was proposed in 2016 paper by Uday Kamath et Al. the author proposed a multi-scale GrabCut algorithm that operates at different resolution of the image. He showed that his algorithm outperformed the original GrabCut algorithm on a dataset of natural images.

One limitation of the GrabCut algorithm is that it requires user input to initialize the foreground and background regions. This can be time-consuming and may not always produce accurate results. However, several techniques have been proposed to automate this process, such as using saliency maps or deep learning-based methods.

Another weakness of GrabCut is its sensitivity to image noise and artifacts. Since the algorithm relies heavily on color and texture information, it can be easily influenced by small changes in the image, such as noise or compression artifacts. This can result in inaccurate segmentation results and may require additional processing steps to correct.

Therefore, GrabCut algorithm is a powerful technique for image segmentation with a wide range of applications. Its interactive nature, combined with its ability to handle complex images, makes it a popular choice for many researchers in computer vision and image processing.

2.2 Other Image segmentation Approaches

Image segmentation can be broadly classified into two categories

- Bottom-up approach
- Top-down approach

In bottom-up approach, image segmentation is performed by grouping pixels based on their similarity in color, texture, intensity or other visual features.

In top-down approach, image segmentation rely on prior knowledge or models to guide the segmentation process.

Commonly used algorithms also called **state-of-the-art algorithms** such as;

2.2.1 Clustering-based Algorithm

Clustering is the task of dividing the population (data points) into a number of groups, such that data points in the same groups are more similar to other data points in that same group than those in other groups. These groups are known as clusters.

a. K-means Clustering

K-means is a simple unsupervised machine learning algorithm. It classifies an image through a specific number of clusters. It starts the process by dividing the image space into k pixels that represent k group centroids.

K-means is a clustering algorithm that is used to group data points into clusters such that data points lying in the same group are very similar to each other in characteristics.
It is one of the most commonly used clustering algorithms. Here, the k represents the number of clusters determined by the user and the algorithm iteratively updates the cluster centroids until convergence. Let's understand how k-means works:

1. First, randomly select k initial clusters
2. Randomly assign each data point to any one of the k clusters
3. Calculate the centers of these clusters
4. Calculate the distance of all the points from the center of each cluster
5. Depending on this distance, the points are reassigned to the nearest cluster
6. Calculate the center of the newly formed clusters
7. Finally, repeat steps (4), (5) and (6) until either the center of the clusters does not change or we reach the set number of iterations

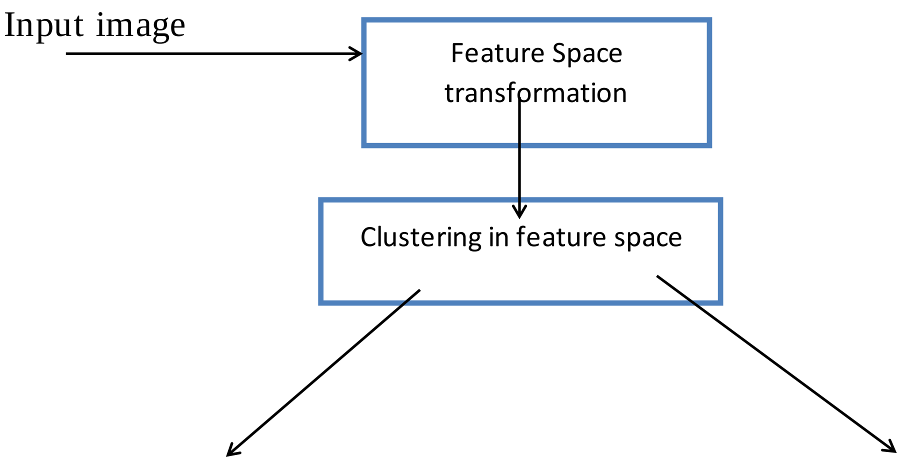
The key advantage of using k-means algorithm is that it is simple and easy to understand. We are assigning the points to the clusters which are closest to them.

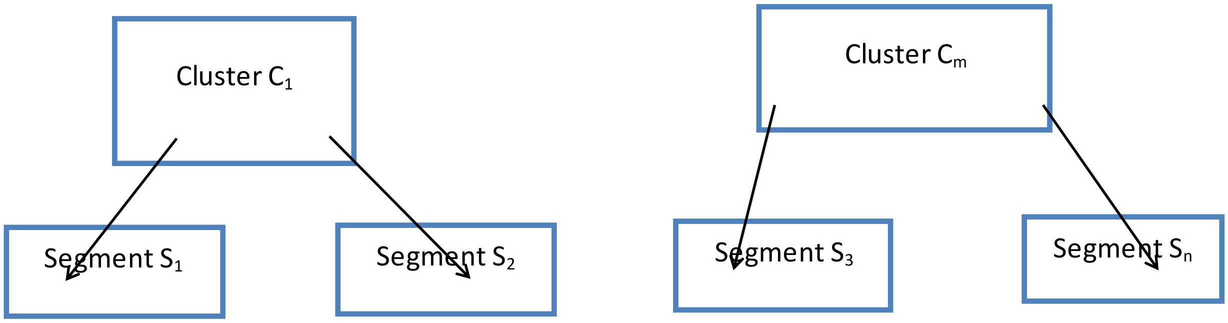
K-means works really well with small datasets. It can segment the objects in the image and give impressive results. But the algorithm hits a roadblock when applied on a large dataset (more number of images).

It looks at all the sample at every iteration, so the time taken is too high. Hence, it's also too expensive to implement. And since k-means is a distance-based algorithm, it is only applicable to convex datasets and is not suitable for clustering non-convex clusters.

b. Fuzzy C Means

With the fuzzy c-means clustering method, the pixels in the image can get clustered in multiple clusters. This means a pixel can belong to more than one cluster. However, every pixel would have varying levels of similarities with every cluster. The fuzzy c-means algorithm has an optimization function which affects the accuracy of your results.
Clustering algorithms can take care of most of your image segmentation needs.





The other approaches;

c. Contour Detection

Contours can be simply defined as curves/polygons formed by joining the pixels that are grouped together according to intensity or color values.

OpenCV provides us with inbuilt functions to detect these contours in images. Contour detection is generally applied on binary images (grayscale images) after edge detection or thresholding (or both) has been applied to them.

d. Color Detection

Detection and classification of colors by using their RGB colorspace values are known as color detection. For example:

```
        R   G   B

Red =    (255, 0, 0)

Green =  (0, 255, 0)

Blue =   (0, 0, 255)

Orange = (255, 165, 0)

Purple = (128, 0, 128)
```

2.2.2 Edge-based segmentation/discontinuity detection

This is a method of segmenting a picture into areas based on discontinuity. This is where **edge detection** comes in. Discontinuity in edges generated due to intensity is recognized and used to establish area borders. Examples: Histogram filtering and contour detection.

Edge-based segmentation is one of the most popular implementations of segmentation in image processing. It focuses on identifying the edges of different objects in an image. This is a crucial step as it helps you find the features of the various objects present in the image as edges contain a lot of information you can use.
Edge detection is widely popular because it helps you in removing unwanted and unnecessary information from the image. It reduces the image's size considerably, making it easier to analyses the same.
Algorithms used in edge-based segmentation identify edges in an image according to the differences in texture, contrast, grey level, colour, saturation, and other properties. You can improve the quality of your results by connecting all the edges into edge chains that match the image borders more accurately.
There are many edge-based segmentation methods available. We can divide them into two categories:

a. Search-Based Edge Detection

Search-based edge detection methods focus on computing a measure of edge strength and look for local directional maxima of the gradient magnitude through a computed estimate of the edge's local orientation.

b. Zero-Crossing Based Edge Detection

Zero-crossing based edge detection methods look for zero crossings in a derivative expression retrieved from the image to find the edges.
Typically, you'll have to pre-process the image to remove unwanted noise and make it easier to detect edges. Canny, Prewitt, Deriche, and Roberts cross are some of the most popular edge detection operators. They make it easier to detect discontinuities and find the edges.
In edge-based detection, your goal is to get a partial segmentation minimum where you can group all the local edges into a binary image. In your newly created binary image, the edge chains must match the existing components of the image in question.

2.2.3 Thresholding segmentation/similarity detection

A method of segmenting a picture into sections based on resemblance. **Thresholding, area expansion, and region splitting and merging** are all included in this methodology. All of them split the image into sections with comparable pixel counts. Based on established criteria, they divide the picture into a group of clusters with comparable features. Example: K-means, Color detection/ classification.

The simplest method for segmentation in image processing is the threshold method. It divides the pixels in an image by comparing the pixel's intensity with a specified value (threshold). It is useful when the required object has a higher intensity than the background (unnecessary parts).

You can consider the threshold value (T) to be a constant but it would only work if the image has very little noise (unnecessary information and data). You can keep the threshold value constant or dynamic according to your requirements.

The thresholding method converts a grey-scale image into a binary image by dividing it into two segments (required and not required sections).

According to the different threshold values, we can classify thresholding segmentation in the following categories:

a. Simple Thresholding

In this method, you replace the image's pixels with either white or black. Now, if the intensity of a pixel at a particular position is less than the threshold value, you'd replace it with black. On the other hand, if it's higher than the threshold, you'd replace it with white. This is simple thresholding and is particularly suitable for beginners in image segmentation.

b. Otsu's Binarization

In simple thresholding, you picked a constant threshold value and used it to perform image segmentation. However, how do you determine that the value you chose was the right one? While the straightforward method for this is to test different values and choose one, it is not the most efficient one.

Take an image with a histogram having two peaks, one for the foreground and one for the background. By using Otsu binarization, you can take the approximate value of the middle of those peaks as your threshold value.

In Otsu binarization, you calculate the threshold value from the image's histogram if the image is bimodal.

This process is quite popular for scanning documents, recognizing patterns, and removing unnecessary colors from a file. However, it has many limitations. You can't use it for images that are not bimodal (images whose histograms have multiple peaks).

c. Adaptive Thresholding

Having one constant threshold value might not be a suitable approach to take with every image. Different images have different backgrounds and conditions which affect their properties.

Thus, instead of using one constant threshold value for performing segmentation on the entire image, you can keep the threshold value variable. In this technique, you'll keep different threshold values for different sections of an image.

This method works well with images that have varying lighting conditions. You'll need to use an algorithm that segments the image into smaller sections and calculates the threshold value for each of them.

2.2.4 Neural Network for segmentation

For the goal of decision making, neural network-based segmentation algorithms replicate the learning techniques of the human brain. This approach is widely used in segmenting medical images and separating them from the background. A neural network is made up of a vast number of linked nodes, each with its own weight.
Perhaps you don't want to do everything by yourself. Perhaps you want to have an AI do most of your tasks, which you can certainly do with neural networks for image segmentation.
You'd use AI to analyze an image and identify its different components such as faces, objects, text. Convolutional Neural Networks are quite popular for image segmentation because they can identify and process image data much quickly and efficiently.

a. Mask R-CNN

Data scientists and researchers at Facebook AI Research (FAIR) pioneered a deep learning architecture, called Mask R-CNN, that can create a pixel-wise mask for each object in an image. This is a really cool concept so follow along closely!

Mask R-CNN is an extension of the popular Faster R-CNN object detection architecture. Mask R-CNN adds a branch to the already existing Faster R-CNN outputs. The Faster R-CNN method generates two things for each object in the image:

- Its class
- The bounding box coordinates

Mask R-CNN adds a third branch to this which outputs of the object mask as well. Take a look at the below image to get an intuition of how Mask R-CNN works on the inside:

The experts at Facebook AI Research (FAIR) created a deep learning architecture called Mask R-CNN which can make a pixel-wise mask for every object present in an image. It is an enhanced version of the Faster R-CNN object detection architecture. The Faster R-CNN uses two pieces of data for every object in an image, the bounding box coordinates and the class of the object. With Mask R-CNN, you get an additional section in this process. Mask R-CNN outputs the object mask after performing the segmentation.
In this process, you'd first pass the input image to the ConvNet which generates the feature map for the image. Then the system applies the region proposal network (RPN) on the feature maps and generates the object proposals with their objectness scores.

After that, the Roi pooling layer gets applied to the proposals to bring them down to one size. In the final stage, the system passes the proposals to the connected layer for classification and generates the output with the bounding boxes for every object.

2.2.5 Region-based segmentation

Region-based segmentation algorithms divide the image into sections with similar features. These regions are only a group of pixels and the algorithm find these groups by first locating a seed point which could be a small section or a large portion of the input image.
After finding the seed points, a region-based segmentation algorithm would either add more pixels to them or shrink them so it can merge them with other seed points.
Based on these two methods, we can classify region-based segmentation into the following categories:

a. Region Growing

In this method, you start with a small set of pixels and then start iteratively merging more pixels according to particular similarity conditions. A region growing algorithm would pick an arbitrary seed pixel in the image, compare it with the neighboring pixels and start increasing the region by finding matches to the seed point.
When a particular region can't grow further, the algorithm will pick another seed pixel which might not belong to any existing region. One region can have too many attributes causing it to take over most of the image. To avoid such an error, region growing algorithms grow multiple regions at the same time.
You should use region growing algorithms for images that have a lot of noise as the noise would make it difficult to find edges or use thresholding algorithms.

b. Region Splitting and Merging

As the name suggests, a region splitting and merging focused method would perform two actions together – splitting and merging portions of the image.
It would first the image into regions that have similar attributes and merge the adjacent portions which are similar to one another. In region splitting, the algorithm considers the entire image while in region growth, the algorithm would focus on a particular point.
The region splitting and merging method follows a divide and conquer methodology. It divides the image into different portions and then matches them according to its predetermined conditions. Another name for the algorithms that perform this task is split-merge algorithms.

2.2.6 Watershed segmentation

In image processing, a watershed is a transformation on a grayscale image. It refers to the geological watershed or a drainage divide. A watershed algorithm would handle the image as if it was a topographic map. It considers the brightness of a pixel as its height and finds the lines that run along the top of those ridges.

Watershed has many technical definitions and has several applications. Apart from identifying the ridges of the pixels, it focuses on defining basins (the opposite of ridges) and floods the basins with markers until they meet the watershed lines going through the ridges.

As basins have a lot of markers while the ridges don’t, the image gets divided into multiple regions according to the ‘height’ of every pixel.

The watershed method converts every image into a topographical map The watershed segmentation method would reflect the topography through the grey values of their pixels. Now, a landscape with valleys and ridges would certainly have three-dimensional aspects. The watershed would consider the three-dimensional representation of the image and create regions accordingly, which are called “catchment basins”.

It has many applications in the medical sector such as MRI, medical imaging, etc. Watershed segmentation is a prominent part of medical image segmentation so if you want to enter that sector, you should focus on learning this method for segmentation in image processing particularly.

Image segmentation frameworks & Tools

With recent advancements in computer vision, there are a number of both open source and closed source frameworks in different programming languages which allow you to take advantage of them and build an image segmentation model fast and efficiently.

Below are a few worth mentioning:

- **Programming Languages:** GrabCut image segmentation algorithm can be implemented using various programming languages such as Python, MATLAB, Opencv-is to provide a real-time computer vision library.
- **DeepMask and SharpMask** : DeepMask is an object proposal based on a convolutional neural network built by Facebook Research. Once the Algorithm is given an input patch, it will generate a class-agnostic mask and an associated score.
- **Image Processing Libraries:** Various image processing libraries such as OpenCV, scikit-image, and PIL can be used to load and manipulate images.
- **Machine Learning Libraries:** Machine learning libraries such as TensorFlow, Keras, and PyTorch can be used to train and validate deep learning models for image segmentation.
- **Integrated Development Environments (IDEs):** IDEs such as PyCharm, Spyder, and Jupyter Notebook can be used to write, run and debug the GrabCut image segmentation algorithm code.
- **Data Visualization Libraries**: Data visualization libraries such as Matplotlib and Seaborn can be used to visualize the input images, the ground truth masks, and the predicted segmentation masks.

Algorithm	**Description**	**Advantages**	**Disadvantages**
Region-Based Segmentation	Separates the objects into different regions based on some threshold value(s). it looks for similarities between two adjacent pixels, then similar pixels are grouped into a similar region.	• Simple calculations. • Fast operation speed. • When the object and background have high contrast, this method performs really well.	• When there is no significant grayscale difference or an overlap of the grayscale pixel values, it becomes very difficult to get accurate segments. • Performs poorly when the contrast between the object and background is lower.
Edge Detection Segmentation	Makes use of discontinuous local features of an image to detect edges and hence define a boundary of the object. Takes advantage of these sharp adjustments to detect boundaries of objects using filters and convolutions.	• It is good for images having better contrast between objects. • Simple to implement • Easy when searching for smooth edges	• Not suitable when there are too many edges in the image and if there is less contrast between objects. • Highly sensitive to noise • Limited accuracy
Segmentation based on Clustering	Divides the pixels of the image into homogeneous clusters. Splits an image into *k* different splits	Works really well on small datasets and generates excellent clusters.	• Computation time is too large and expensive. • k-means is a distance-based algorithm. It is not suitable for clustering non-convex clusters.
Mask R-CNN	Gives three outputs for each object in the image: its class, bounding box coordinates, and object mask.	• Simple, flexible and general approach. • It is also the current state-of-the-art for image segmentation.	

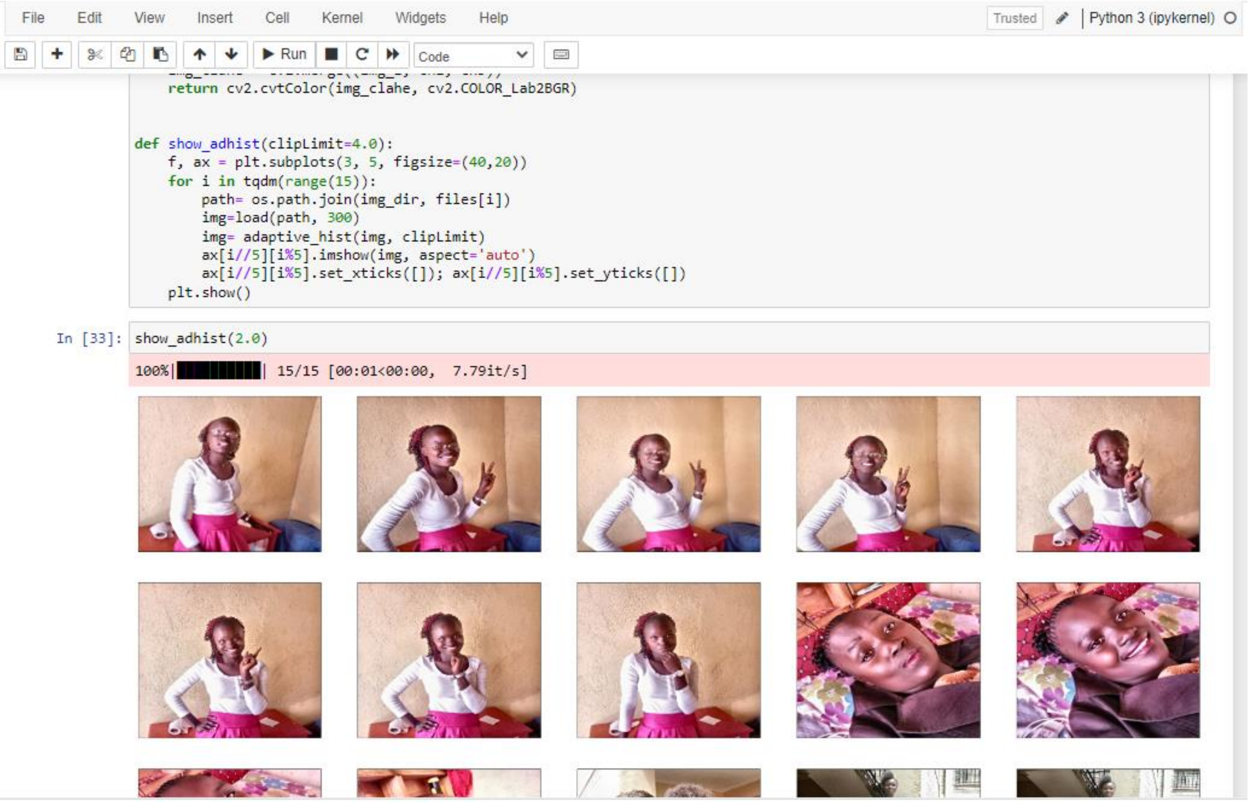
CHAPTER THREE

METHODOLOGY

3.1. Preprocessing

The first process in **G**rabcut image segmentation is preprocessing which involves enhancing the image quality, removing noise and normalizing the image intensity values **as shown in figures below.**

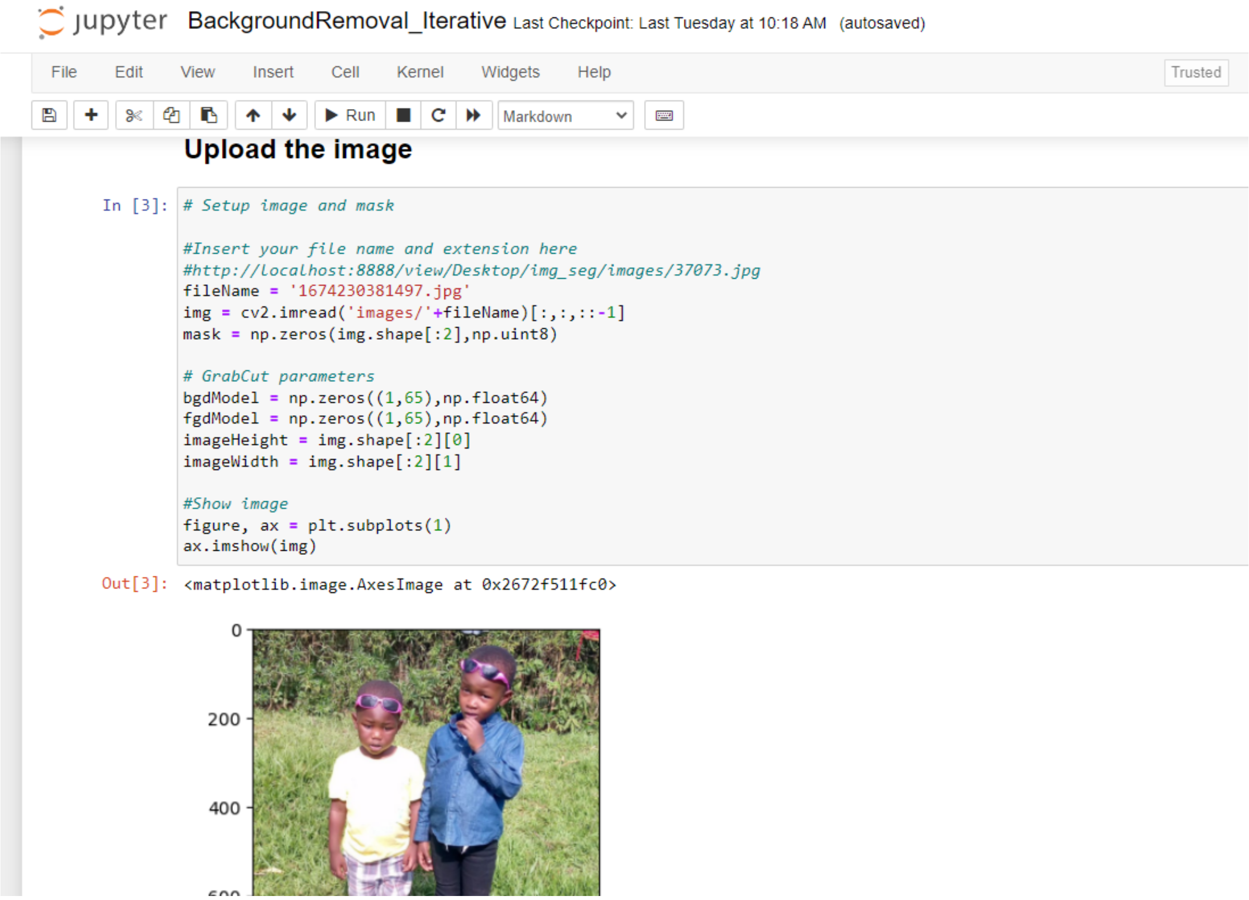
Figure 3.1a Enhancing Image quality.



```
    return cv2.cvtColor(img_clahe, cv2.COLOR_Lab2BGR)


def show_adhist(clipLimit=4.0):
    f, ax = plt.subplots(3, 5, figsize=(40,20))
    for i in tqdm(range(15)):
        path= os.path.join(img_dir, files[i])
        img=load(path, 300)
        img= adaptive_hist(img, clipLimit)
        ax[i//5][i%5].imshow(img, aspect='auto')
        ax[i//5][i%5].set_xticks([]); ax[i//5][i%5].set_yticks([])
    plt.show()
```

```
In [33]: show_adhist(2.0)
```

```
100%|██████████| 15/15 [00:01<00:00,  7.79it/s]
```


jupyter
BackgroundRemoval_Iterative Last Checkpoint: Last Tuesday at 10:18 AM (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Run
Markdown
0
200
400
600

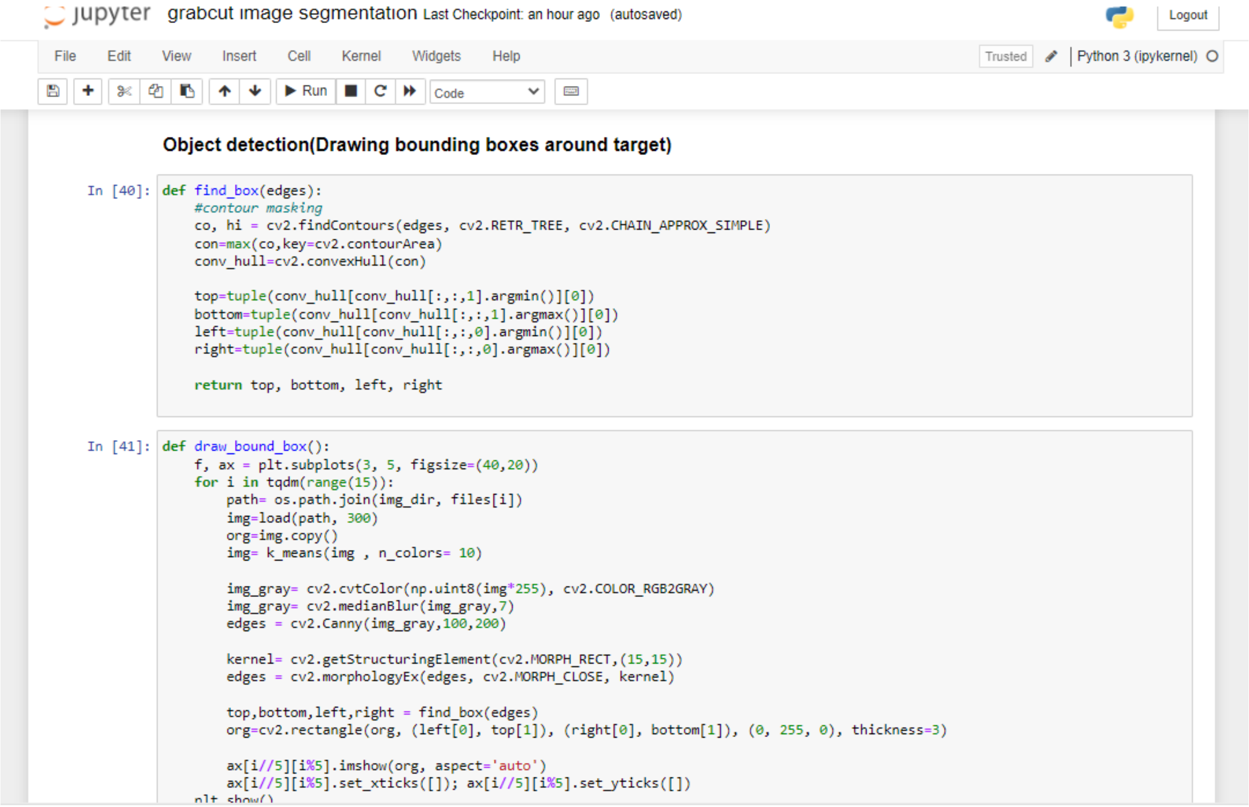
Upload the image

```
In [3]: # Setup image and mask

        #Insert your file name and extension here
        #http://localhost:8888/view/Desktop/img_seg/images/37073.jpg
        fileName = '1674230381497.jpg'
        img = cv2.imread('images/'+fileName)[:,:,::-1]
        mask = np.zeros(img.shape[:2],np.uint8)

        # GrabCut parameters
        bgdModel = np.zeros((1,65),np.float64)
        fgdModel = np.zeros((1,65),np.float64)
        imageHeight = img.shape[:2][0]
        imageWidth = img.shape[:2][1]

        #Show image
        figure, ax = plt.subplots(1)
        ax.imshow(img)

Out[3]: <matplotlib.image.AxesImage at 0x2672f511fc0>
```

Figure 3.1b Uploading the images

3.2 Feature extraction

The next step is to extract features from preprocessed image by identifying regions of interest such as edges, corners, color or texture. In grabcut we define the initial foreground and background regions by manually drawing a bounding box around the object of interest or selecting pixels in the image as shown in figures below.

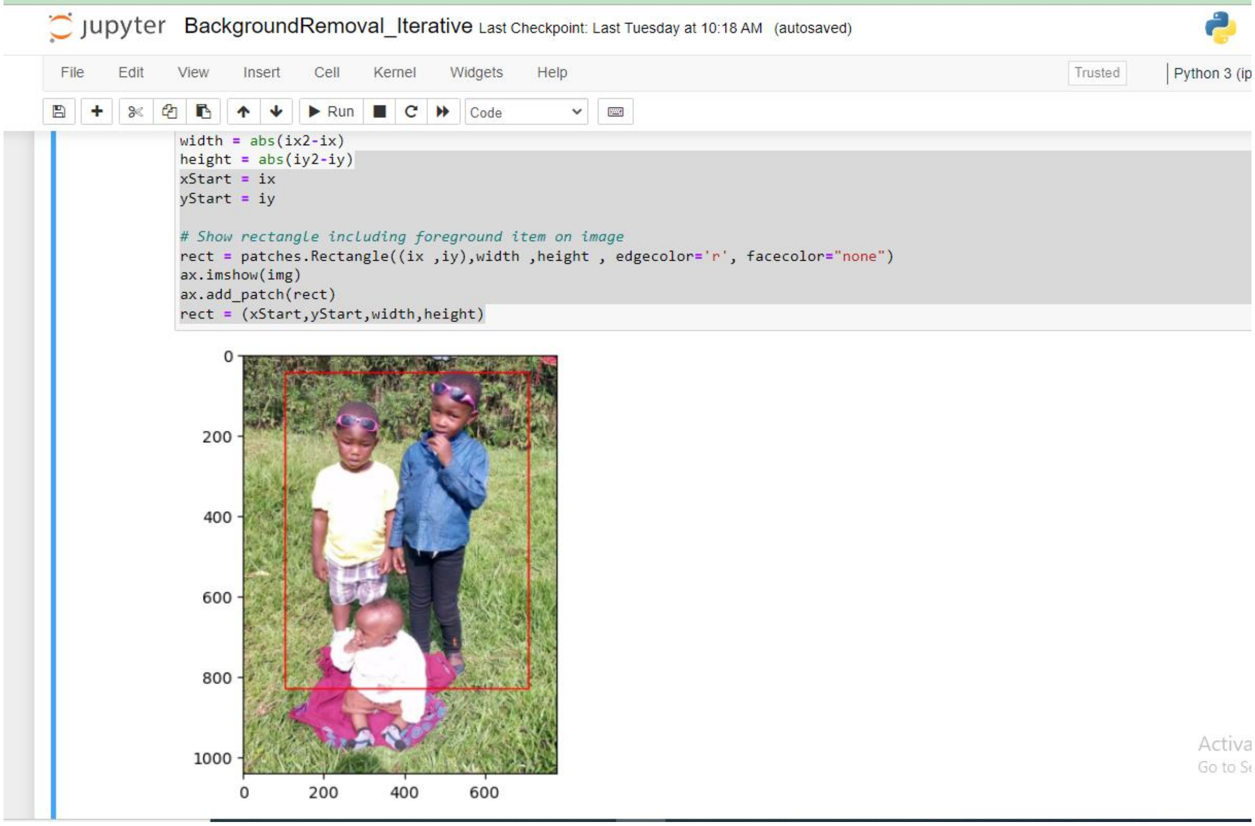


Object detection(Drawing bounding boxes around target)

In [40]:
```
def find_box(edges):
    #contour masking
    co, hi = cv2.findContours(edges, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    con=max(co,key=cv2.contourArea)
    conv_hull=cv2.convexHull(con)

    top=tuple(conv_hull[conv_hull[:,:,1].argmin()][0])
    bottom=tuple(conv_hull[conv_hull[:,:,1].argmax()][0])
    left=tuple(conv_hull[conv_hull[:,:,0].argmin()][0])
    right=tuple(conv_hull[conv_hull[:,:,0].argmax()][0])

    return top, bottom, left, right
```

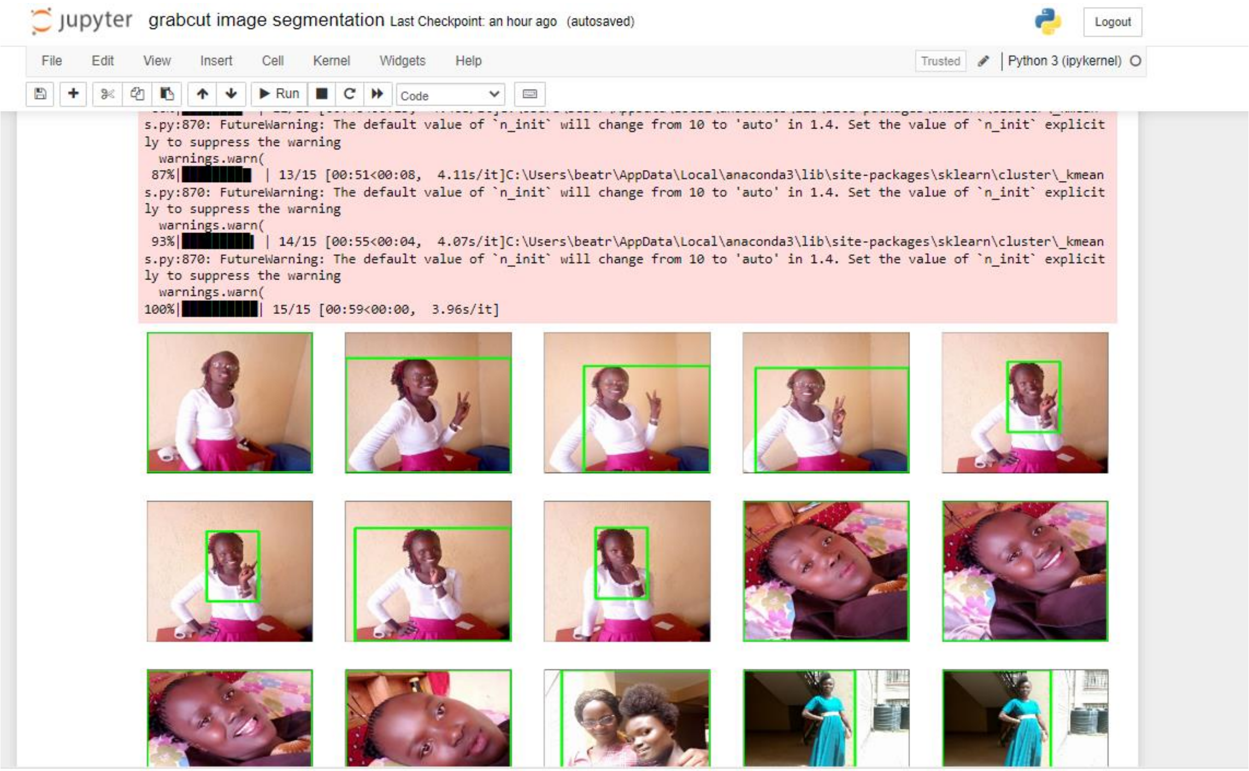
In [41]:
```
def draw_bound_box():
    f, ax = plt.subplots(3, 5, figsize=(40,20))
    for i in tqdm(range(15)):
        path= os.path.join(img_dir, files[i])
        img=load(path, 300)
        org=img.copy()
        img= k_means(img , n_colors= 10)

        img_gray= cv2.cvtColor(np.uint8(img*255), cv2.COLOR_RGB2GRAY)
        img_gray= cv2.medianBlur(img_gray,7)
        edges = cv2.Canny(img_gray,100,200)

        kernel= cv2.getStructuringElement(cv2.MORPH_RECT,(15,15))
        edges = cv2.morphologyEx(edges, cv2.MORPH_CLOSE, kernel)

        top,bottom,left,right = find_box(edges)
        org=cv2.rectangle(org, (left[0], top[1]), (right[0], bottom[1]), (0, 255, 0), thickness=3)

        ax[i//5][i%5].imshow(org, aspect='auto')
        ax[i//5][i%5].set_xticks([]); ax[i//5][i%5].set_yticks([])
    plt.show()
```

Figure 3.2a. Object detection code



```
width = abs(ix2-ix)
height = abs(iy2-iy)
xStart = ix
yStart = iy

# Show rectangle including foreground item on image
rect = patches.Rectangle((ix ,iy),width ,height , edgecolor='r', facecolor="none")
ax.imshow(img)
ax.add_patch(rect)
rect = (xStart,yStart,width,height)
```

Figure 3.2b Object detected by drawing rectangular box removing background regions.



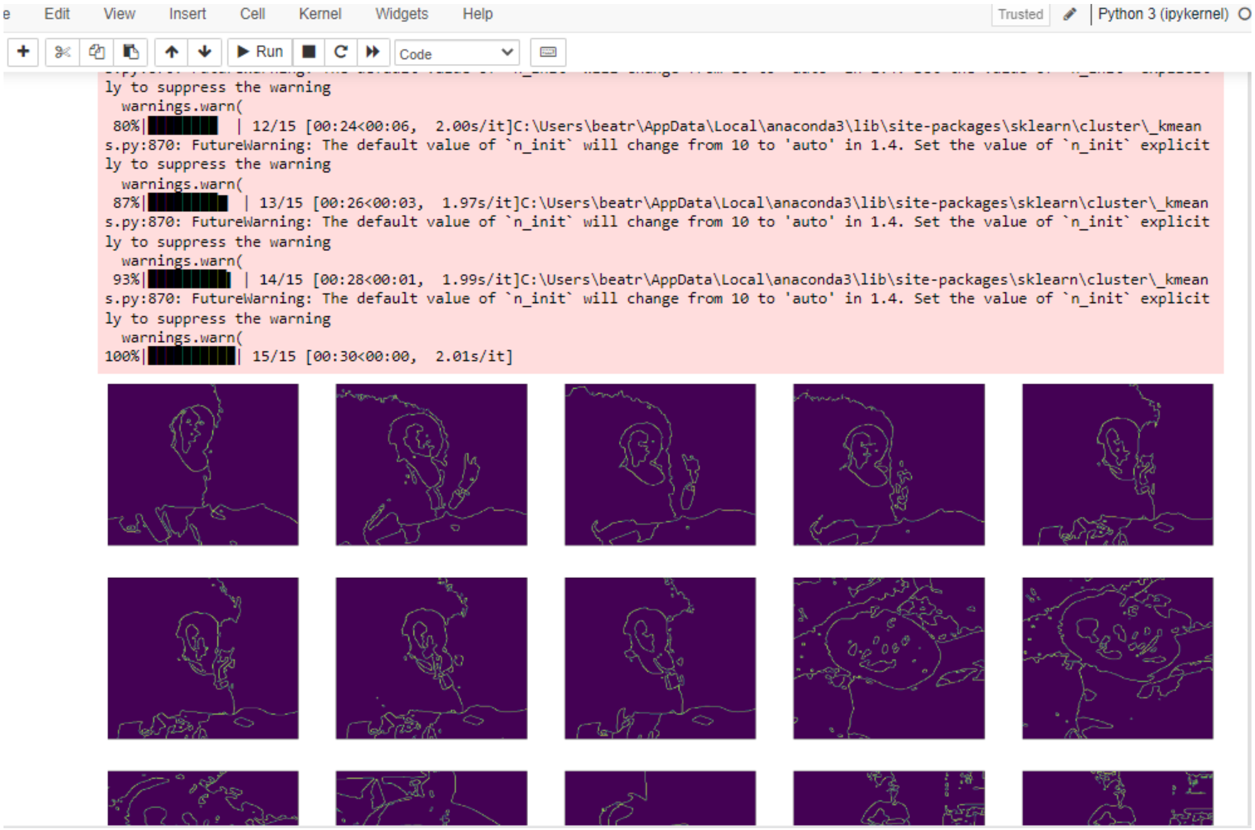
```
s.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicit
ly to suppress the warning
  warnings.warn(
 87%|████████▋ | 13/15 [00:51<00:08,  4.11s/it]C:\Users\beatr\AppData\Local\anaconda3\lib\site-packages\sklearn\cluster\_kmean
s.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicit
ly to suppress the warning
  warnings.warn(
 93%|█████████▎| 14/15 [00:55<00:04,  4.07s/it]C:\Users\beatr\AppData\Local\anaconda3\lib\site-packages\sklearn\cluster\_kmean
s.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicit
ly to suppress the warning
  warnings.warn(
100%|██████████| 15/15 [00:59<00:00,  3.96s/it]
```

Figure 3.2c. defining initial foreground

Deep learning-based segmentation method such as U-Net and Mask R-CNN do not require feature extraction and can directly learn the segmentation task from the raw image data.

3.3 Build the graph for the Segmentation

The next step is to build a graph representation of the image. This involves defining nodes and edges that represent the pixels in the image and their relationships with their neighboring pixels. The extracted features are now segmented into different regions based on similarity of features. The approaches for image segmentation involve threshold-based segmentation, region-based segmentation and edge-based segmentation.



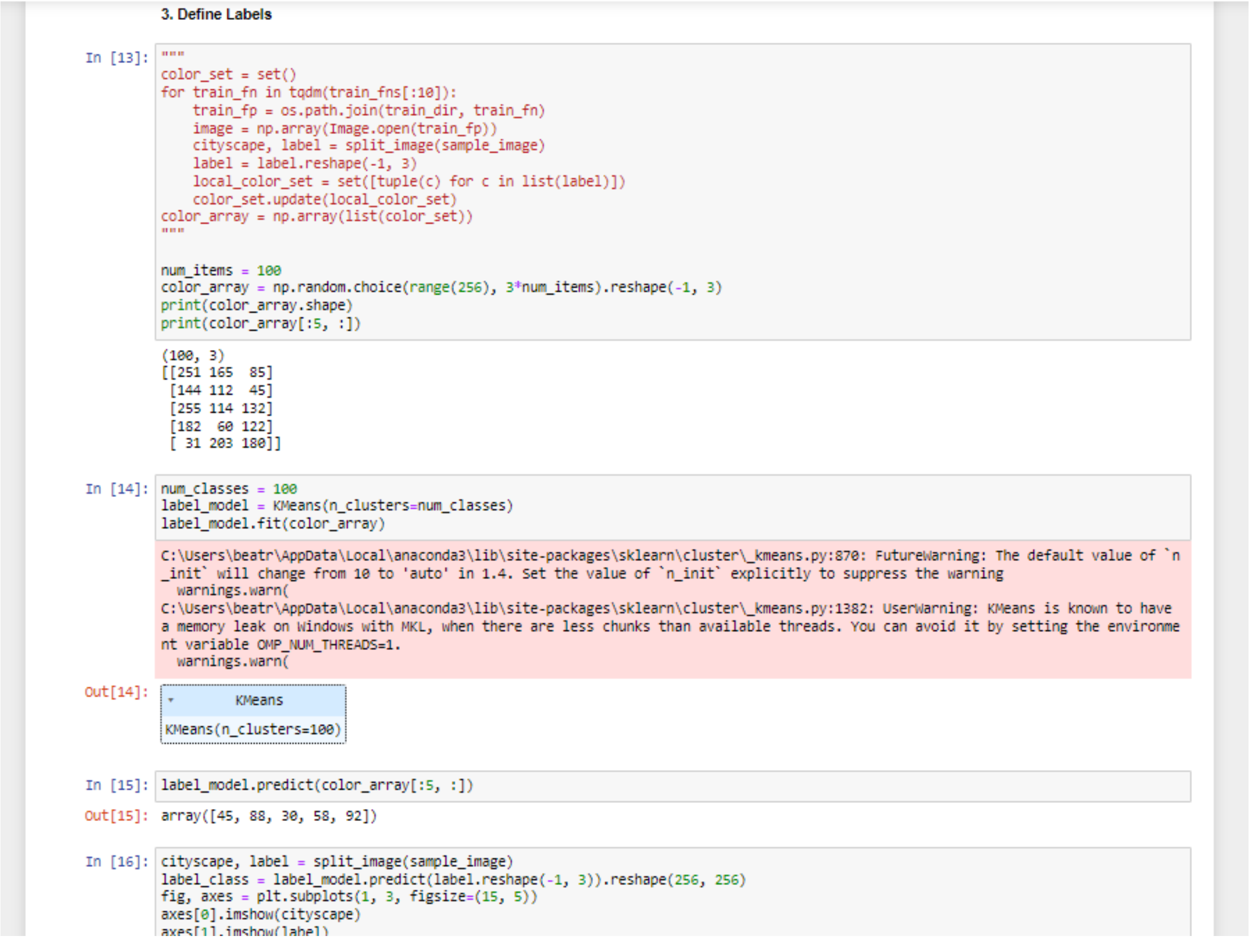
```
ly to suppress the warning
  warnings.warn(
 80%|████████  | 12/15 [00:24<00:06,  2.00s/it]C:\Users\beatr\AppData\Local\anaconda3\lib\site-packages\sklearn\cluster\_kmean
s.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicit
ly to suppress the warning
  warnings.warn(
 87%|████████▋ | 13/15 [00:26<00:03,  1.97s/it]C:\Users\beatr\AppData\Local\anaconda3\lib\site-packages\sklearn\cluster\_kmean
s.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicit
ly to suppress the warning
  warnings.warn(
 93%|█████████▎| 14/15 [00:28<00:01,  1.99s/it]C:\Users\beatr\AppData\Local\anaconda3\lib\site-packages\sklearn\cluster\_kmean
s.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicit
ly to suppress the warning
  warnings.warn(
100%|██████████| 15/15 [00:30<00:00,  2.01s/it]
```

Figure 3.3a. Building graph representation of image

3.4. Assign labels

The algorithm assigns labels to the pixels in the image based on their initial classification as foreground or background. This step is performed using a probabilistic model that considers color and texture information as shown below.

Figure 3.4a.



3. Define Labels

```
In [13]: """
         color_set = set()
         for train_fn in tqdm(train_fns[:10]):
             train_fp = os.path.join(train_dir, train_fn)
             image = np.array(Image.open(train_fp))
             cityscape, label = split_image(sample_image)
             label = label.reshape(-1, 3)
             local_color_set = set([tuple(c) for c in list(label)])
             color_set.update(local_color_set)
         color_array = np.array(list(color_set))
         """

         num_items = 100
         color_array = np.random.choice(range(256), 3*num_items).reshape(-1, 3)
         print(color_array.shape)
         print(color_array[:5, :])

         (100, 3)
         [[251 165  85]
          [144 112  45]
          [255 114 132]
          [182  60 122]
          [ 31 203 180]]
```

```
In [14]: num_classes = 100
         label_model = KMeans(n_clusters=num_classes)
         label_model.fit(color_array)

         C:\Users\beatr\AppData\Local\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n
         _init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
           warnings.warn(
         C:\Users\beatr\AppData\Local\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMeans is known to have
         a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environme
         nt variable OMP_NUM_THREADS=1.
           warnings.warn(

Out[14]: ▾        KMeans
         KMeans(n_clusters=100)
```

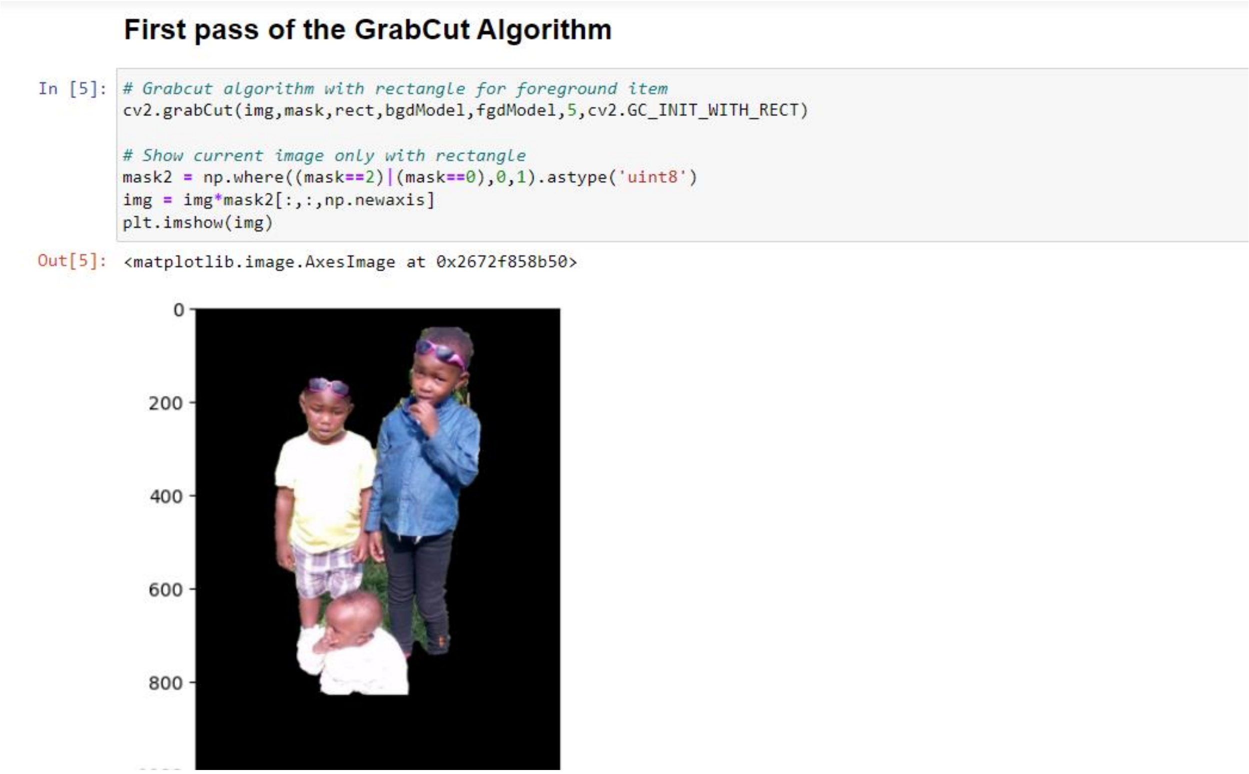
```
In [15]: label_model.predict(color_array[:5, :])

Out[15]: array([45, 88, 30, 58, 92])
```

```
In [16]: cityscape, label = split_image(sample_image)
         label_class = label_model.predict(label.reshape(-1, 3)).reshape(256, 256)
         fig, axes = plt.subplots(1, 3, figsize=(15, 5))
         axes[0].imshow(cityscape)
         axes[1].imshow(label)
```

3.5. Refine the segmentation

The segmentation results can be refined by iteratively updating the foreground and background models based on the current segmentation results. The algorithm can also incorporate user input to refine the segmentation results.

Figure 3.5a

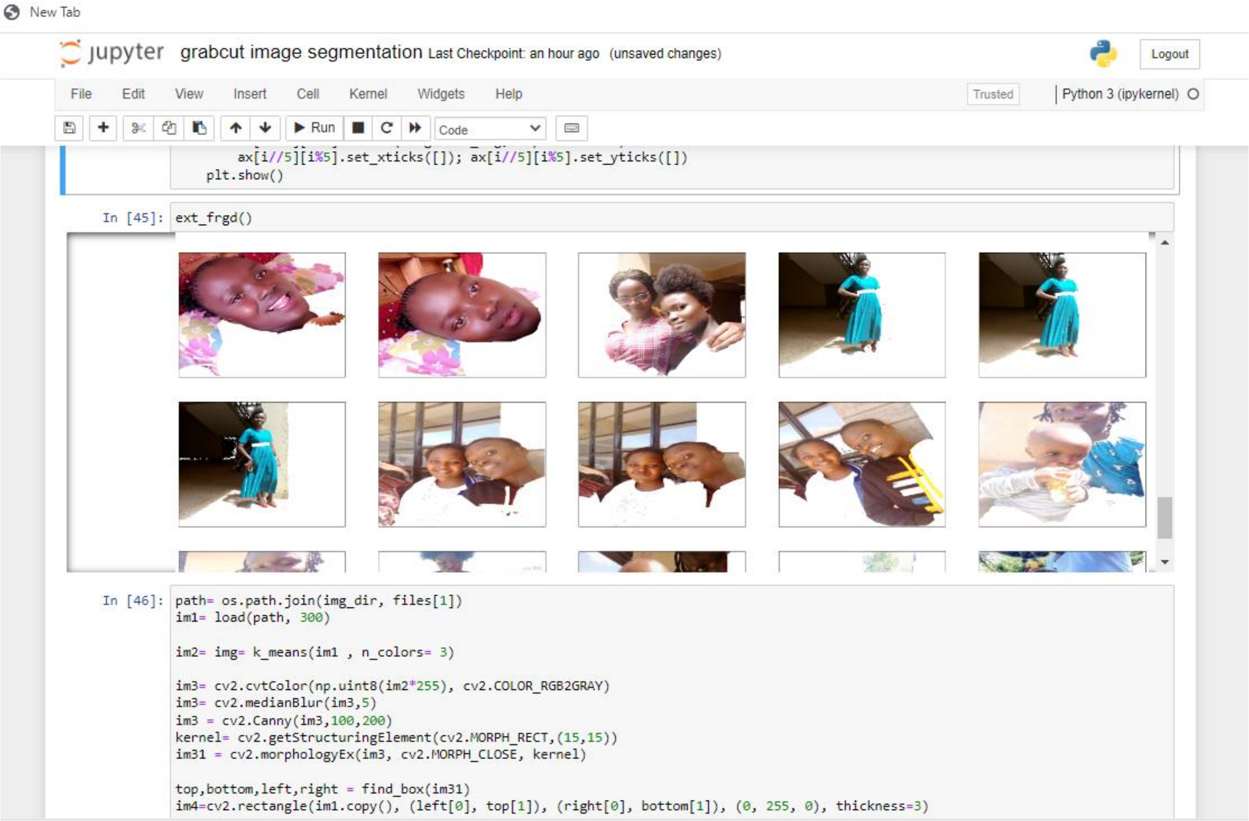


First pass of the GrabCut Algorithm

```
In [5]: # Grabcut algorithm with rectangle for foreground item
        cv2.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_RECT)

        # Show current image only with rectangle
        mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
        img = img*mask2[:,:,np.newaxis]
        plt.imshow(img)
```

```
Out[5]: <matplotlib.image.AxesImage at 0x2672f858b50>
```

Figure 3.5b

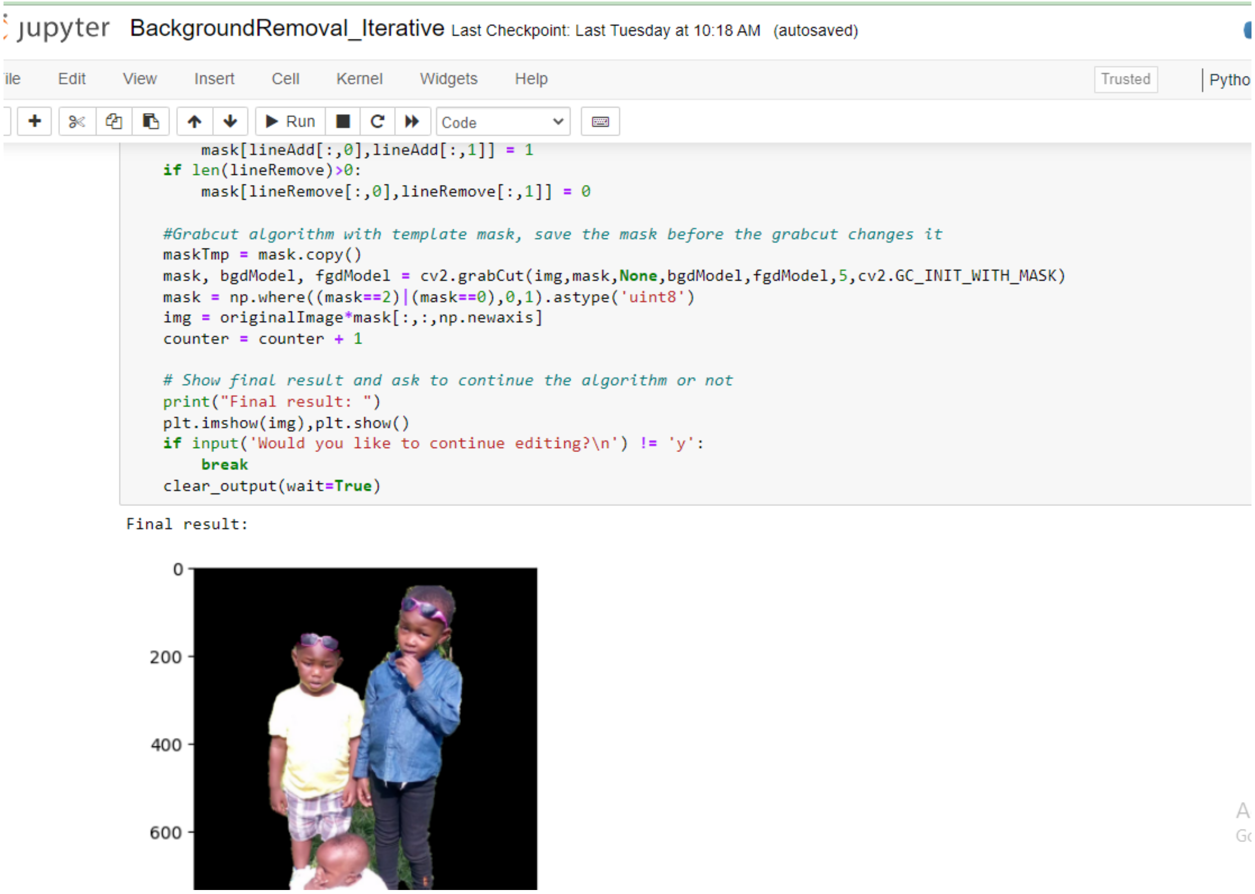


```
        ax[i//5][i%5].set_xticks([]); ax[i//5][i%5].set_yticks([])
    plt.show()
```

```
In [45]: ext_frgd()
```

```
In [46]: path= os.path.join(img_dir, files[1])
         im1= load(path, 300)

         im2= img= k_means(im1 , n_colors= 3)

         im3= cv2.cvtColor(np.uint8(im2*255), cv2.COLOR_RGB2GRAY)
         im3= cv2.medianBlur(im3,5)
         im3 = cv2.Canny(im3,100,200)
         kernel= cv2.getStructuringElement(cv2.MORPH_RECT,(15,15))
         im31 = cv2.morphologyEx(im3, cv2.MORPH_CLOSE, kernel)

         top,bottom,left,right = find_box(im31)
         im4=cv2.rectangle(im1.copy(), (left[0], top[1]), (right[0], bottom[1]), (0, 255, 0), thickness=3)
```

Figure 3.5c


jupyter
BackgroundRemoval_Iterative Last Checkpoint: Last Tuesday at 10:18 AM (autosaved)
ile
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Pytho
Run
Code
0
200
400
600
A
Gc

```
    mask[lineAdd[:,0],lineAdd[:,1]] = 1
if len(lineRemove)>0:
    mask[lineRemove[:,0],lineRemove[:,1]] = 0

#Grabcut algorithm with template mask, save the mask before the grabcut changes it
maskTmp = mask.copy()
mask, bgdModel, fgdModel = cv2.grabCut(img,mask,None,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_MASK)
mask = np.where((mask==2)|(mask==0),0,1).astype('uint8')
img = originalImage*mask[:,:,np.newaxis]
counter = counter + 1

# Show final result and ask to continue the algorithm or not
print("Final result: ")
plt.imshow(img),plt.show()
if input('Would you like to continue editing?\n') != 'y':
    break
clear_output(wait=True)
```

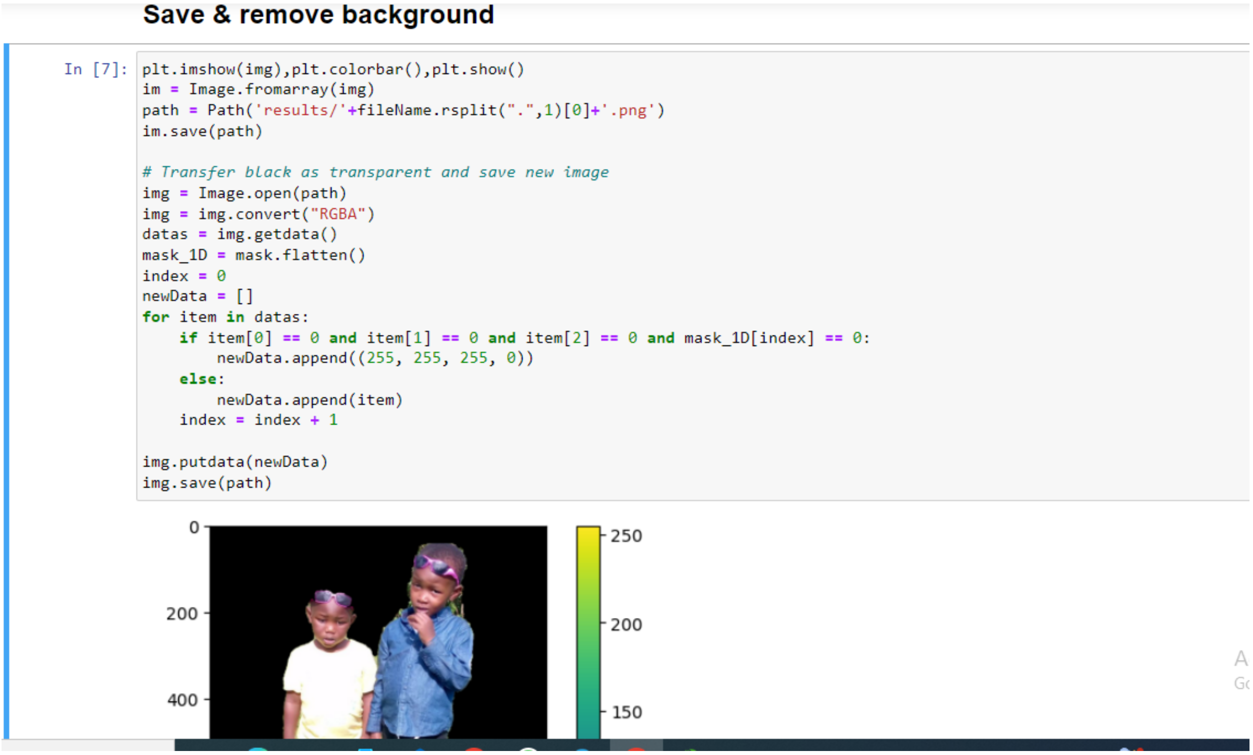
```
Final result:
```

3.6. Evaluation

The final output of the GrabCut algorithm is a binary image where the pixels belonging to the object of interest are marked as foreground, and the remaining pixels are marked as background.

The segmentation result is evaluated using metrics such as precision, recall and F1 score. If the segmentation result is not satisfactory, the method can be tweaked or a different segmentation approach can be used.

Figure 3.6a



Save & remove background

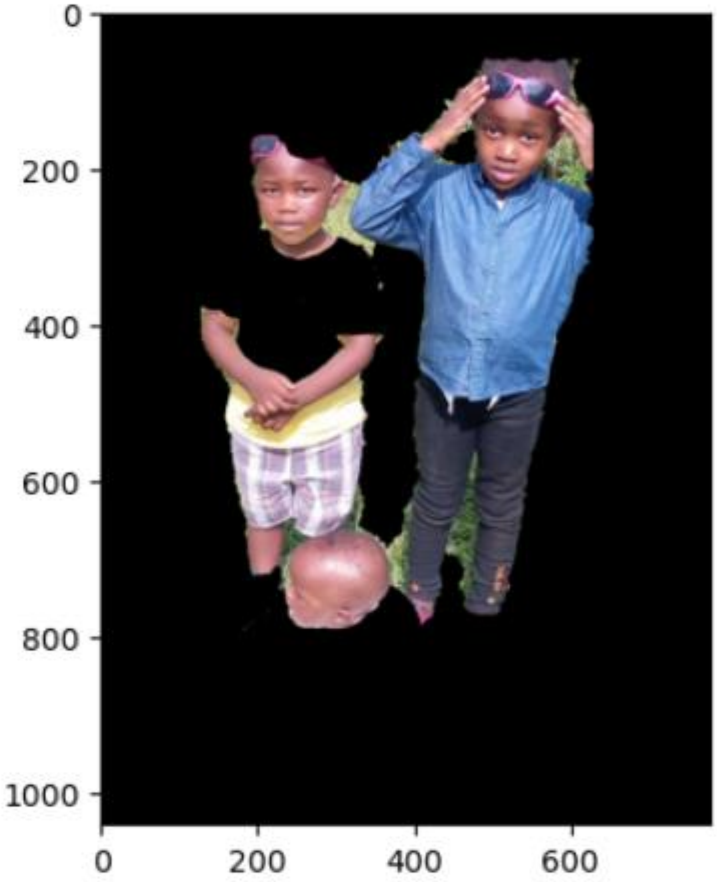
In [7]:
```
plt.imshow(img),plt.colorbar(),plt.show()
im = Image.fromarray(img)
path = Path('results/'+fileName.rsplit(".",1)[0]+'.png')
im.save(path)

# Transfer black as transparent and save new image
img = Image.open(path)
img = img.convert("RGBA")
datas = img.getdata()
mask_1D = mask.flatten()
index = 0
newData = []
for item in datas:
    if item[0] == 0 and item[1] == 0 and item[2] == 0 and mask_1D[index] == 0:
        newData.append((255, 255, 255, 0))
    else:
        newData.append(item)
    index = index + 1

img.putdata(newData)
img.save(path)
```

3.7. Post-processing

The segmented image can undergo additional post-processing steps to remove any noise or artifacts and to enhance the final results.

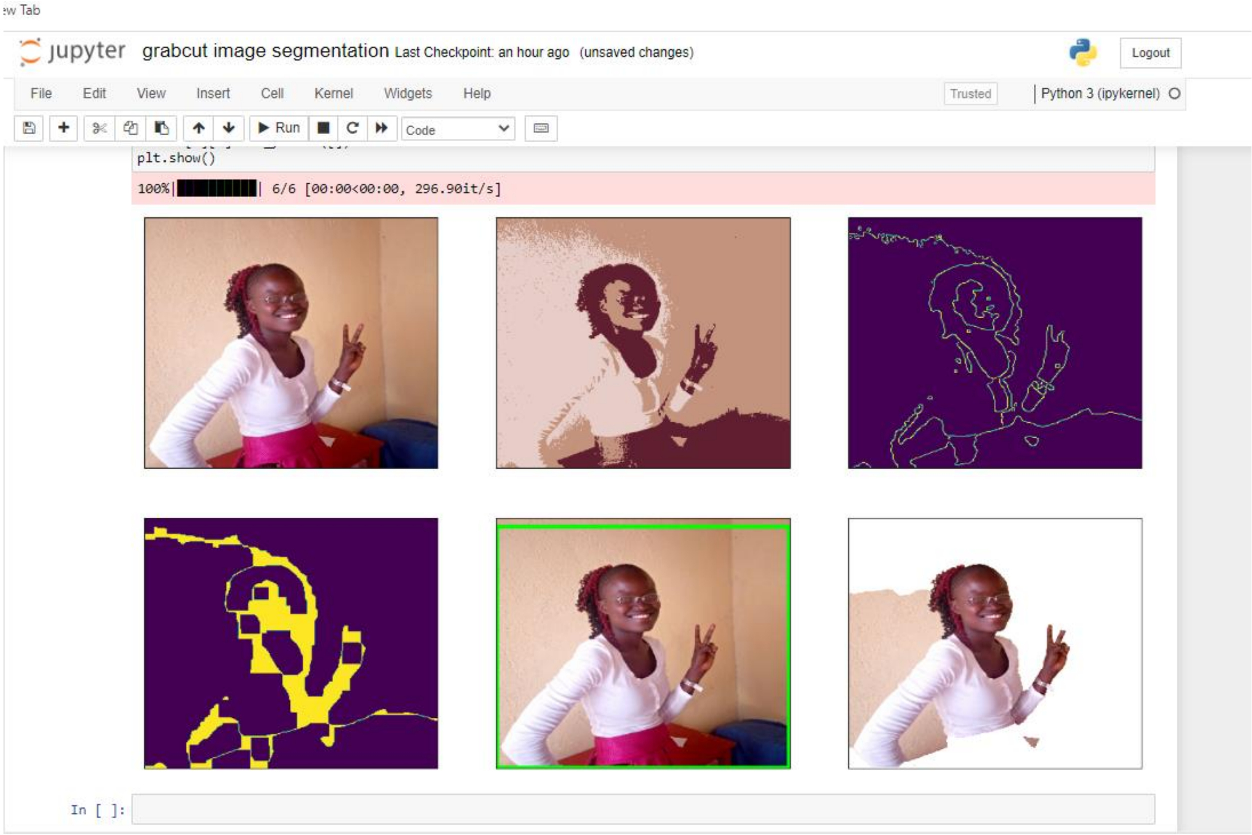
Figure 3.7a

```
Final result:
```



```
Would you like to continue editing?
yes
```

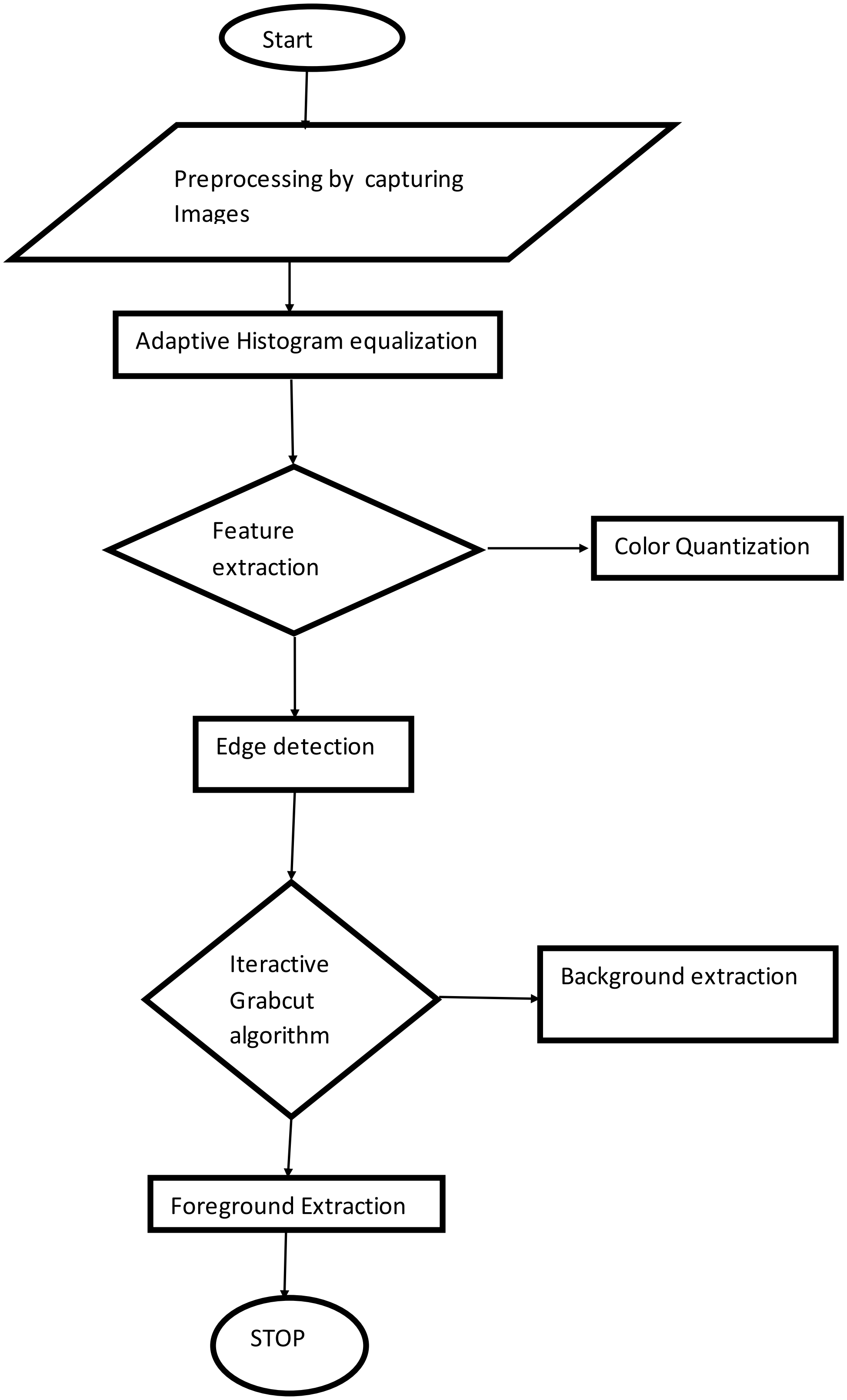
Figure 3.7b



```
plt.show()
```

```
100%|██████████| 6/6 [00:00<00:00, 296.90it/s]
```

Flowchart for Grabcut segmentation methodology



CHAPTER FOUR

SYSTEM IMPLEMENTATION, DESIGN AND ANALYSIS

4.1 System Analysis

This is to identify the requirements of Grabcut image segmentation the system. This includes the input and output data requirements, hardware and software requirements, and user interface requirements.

4.2 System Design

This involves selecting the appropriate hardware and software components, designing the user interface, and defining the algorithms and data structures needed for the system architecture.

4.2.1 Grabcut system design

1. **User interface design:**

The user interface of the system should be intuitive and easy to use. The user should be able to load the image, define the foreground and background regions, and visualize the segmentation results.

2. **Algorithm design:**

The GrabCut algorithm is a graph-based algorithm that uses a probabilistic model to assign labels to the pixels in the image. The algorithm should be designed to efficiently build the graph, assign labels to the pixels, and refine the segmentation results.

3. Data structures design:

The data structures used in the GrabCut algorithm include graphs, matrices, and vectors. The system design should incorporate these data structures efficiently to optimize the algorithm's performance.

4. **Software and hardware design:**

The GrabCut algorithm can be implemented using different programming languages, such as C++, Python, or MATLAB. The system design should select the appropriate software tools and hardware components based on the algorithm's requirements and performance needs.

5. **Testing and validation:**

Once the implementation is complete, the system should be tested and validated to ensure that it meets the requirements and performance criteria defined in the system analysis phase.

4.2.2 Requirement Specification

1. Functional requirement

i. **Image Loading**: The system should be able to load digital images in different formats, such as JPEG, PNG, and BMP.

ii. **User Input:** The system should provide an intuitive user interface for the user to input the initial foreground and background regions.

iii. **Image Segmentation:** The system should segment the image using the GrabCut algorithm and output a binary image where the pixels belonging to the object of interest are marked as foreground, and the remaining pixels are marked as background.

iv. **Segmentation Refinement:** The system should allow for iterative refinement of the segmentation results by updating the foreground and background models based on the current segmentation results.

v. **Post-processing:** The system should incorporate post-processing techniques to remove noise and artifacts from the segmented image.

2. Non-functional requirement

I. **Performance:** The system should be able to process images of different sizes and resolutions within a reasonable time frame.

II. **Accuracy:** The system should produce accurate segmentation results that are consistent with the user's input and expectations.

III. **Usability:** The system should provide an intuitive and user-friendly interface that allows users to easily load images, input foreground and background regions, and visualize the segmentation results.

IV. **Portability:** The system should be able to run on different platforms, such as Windows, Linux, and MacOS.

V. **Scalability:** The system should be able to scale to accommodate larger datasets and image volumes.

4.3 Budget

4.3.1 Hardware requirements

NAME	ESTIMATED COST
EliteBook laptop intel core	Ksh.23,000

i5, 500gb hard disk, 4gb RAM	
WIFI	Ksh.500
TOTAL	Ksh.30,500

4.3.2 Software requirement

Dataset	High quality
Python using libraries like OpenCV	Free open-source
Project management tools	
Jupyter	

4.4 Constraints

4.4.1Technical Constraints:

The system may have technical constraints such as limitations on the maximum image size and resolution that it can process or the maximum number of iterations for segmentation refinement.

4.4.2Operational Constraints:

The system may have operational constraints such as limitations on the availability of trained personnel to operate and maintain the system.

4.5 Project schedule

1. **One Week : project planning and Data collection**

Defining project scope and objectives

Identify the image segmentation task to be performed and required software and hardware resources.

Collect and preprocess the dataset for training the testing the model.

2. **Two Weeks : model design and Development**

Design the deep learning model architecture.

Implement the model using a suitable deep learning framework such as OpenCV..

Train the model using the preprocessed dataset.

3. One Week : model optimization

Evaluate the performance of the model on the validation

Apply techniques such as data augmentation and transfer learning to improve the model's performance.

Test the optimized model on the test set and evaluate its performance.

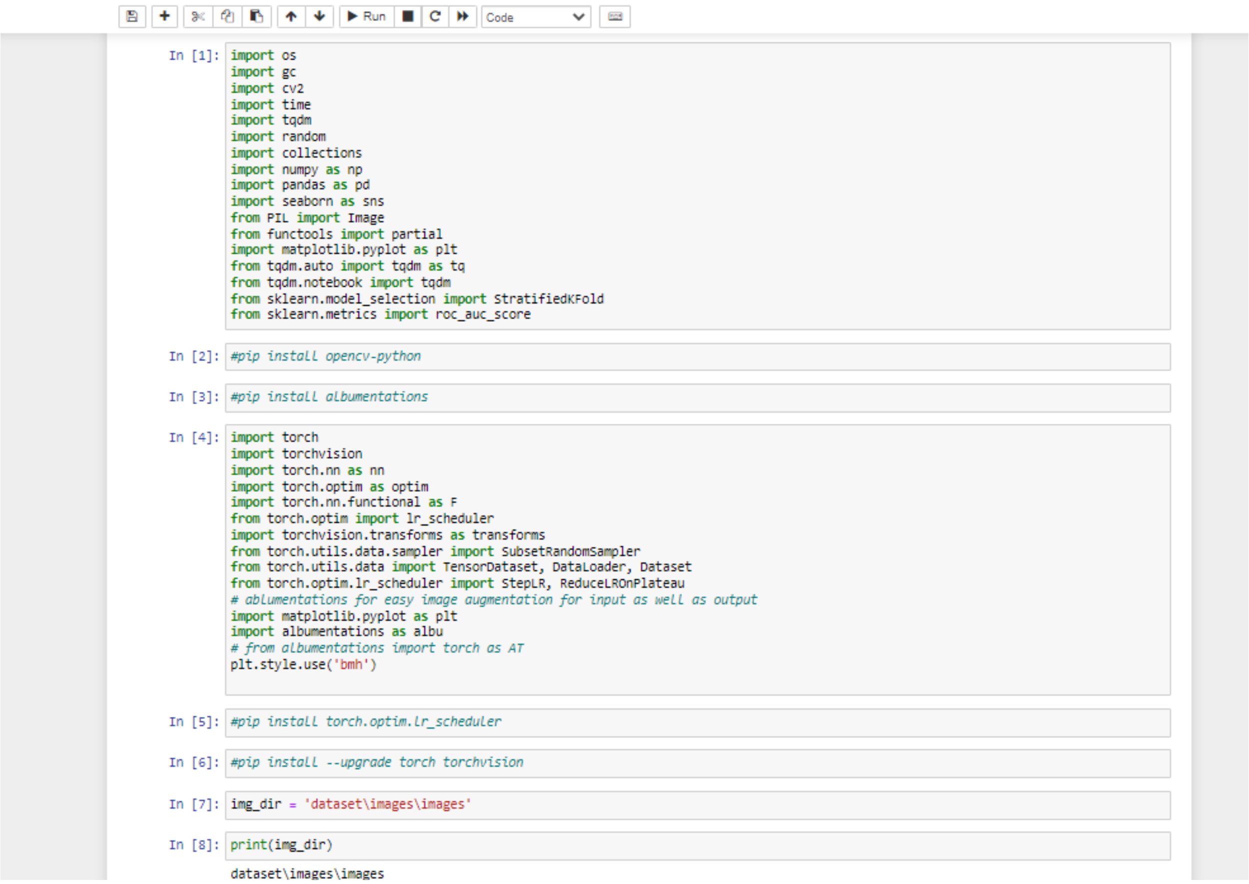
4. Two Weeks: Results analysis and reporting

Analyze the results of the model on the test set.

Compare the performance of the model with other state-of-the-art methods.

Present the report to the project stakeholder and seek feedback for future improvement.

The entire project is scheduled to take a maximum of 6 months. The first 3 months are for writing project proposal and the last 3 months for actual implementation of the project.



```
In [1]: import os
        import gc
        import cv2
        import time
        import tqdm
        import random
        import collections
        import numpy as np
        import pandas as pd
        import seaborn as sns
        from PIL import Image
        from functools import partial
        import matplotlib.pyplot as plt
        from tqdm.auto import tqdm as tq
        from tqdm.notebook import tqdm
        from sklearn.model_selection import StratifiedKFold
        from sklearn.metrics import roc_auc_score

In [2]: #pip install opencv-python

In [3]: #pip install albumentations

In [4]: import torch
        import torchvision
        import torch.nn as nn
        import torch.optim as optim
        import torch.nn.functional as F
        from torch.optim import lr_scheduler
        import torchvision.transforms as transforms
        from torch.utils.data.sampler import SubsetRandomSampler
        from torch.utils.data import TensorDataset, DataLoader, Dataset
        from torch.optim.lr_scheduler import StepLR, ReduceLROnPlateau
        # ablumentations for easy image augmentation for input as well as output
        import matplotlib.pyplot as plt
        import albumentations as albu
        # from albumentations import torch as AT
        plt.style.use('bmh')

In [5]: #pip install torch.optim.lr_scheduler

In [6]: #pip install --upgrade torch torchvision

In [7]: img_dir = 'dataset\images\images'

In [8]: print(img_dir)

        dataset\images\images
```

Figure 4.6a Imports

4.7 MODEL IMPLEMENTATION, TESTING AND VALIDATION

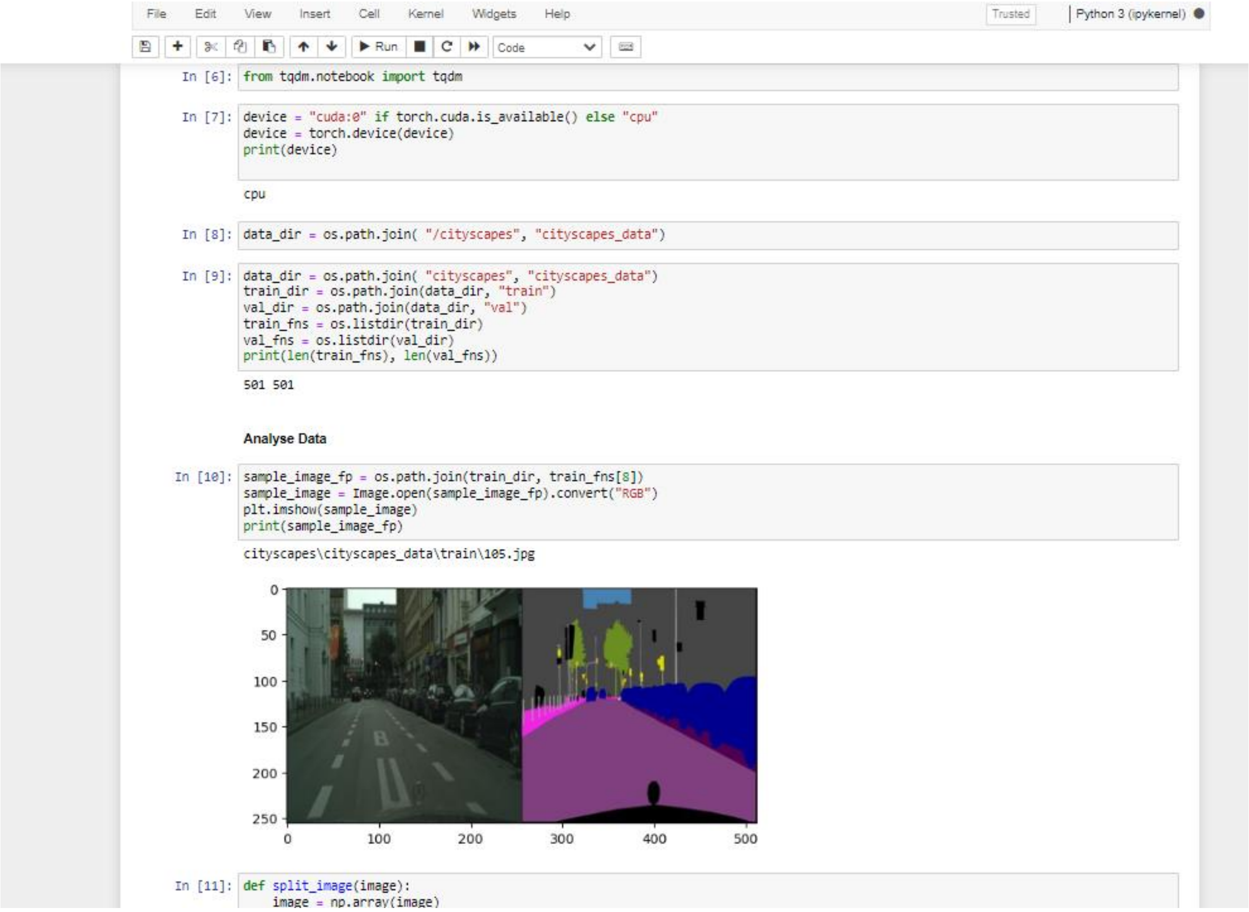
4.7.1 U-NET MODEL

The U-Net model is a deep learning architecture that has been used for semantic image segmentation tasks, including the GrabCut algorithm. The U-Net architecture is known for its ability to effectively capture contextual information while maintaining high-resolution features, making it well-suited for image segmentation tasks.

1. **Data Preparation:**

The first step is to prepare the data for training the U-Net model. This involves obtaining a dataset of images and corresponding segmented masks. For the GrabCut algorithm, the initial foreground and background regions can be used as the ground truth segmentation masks.

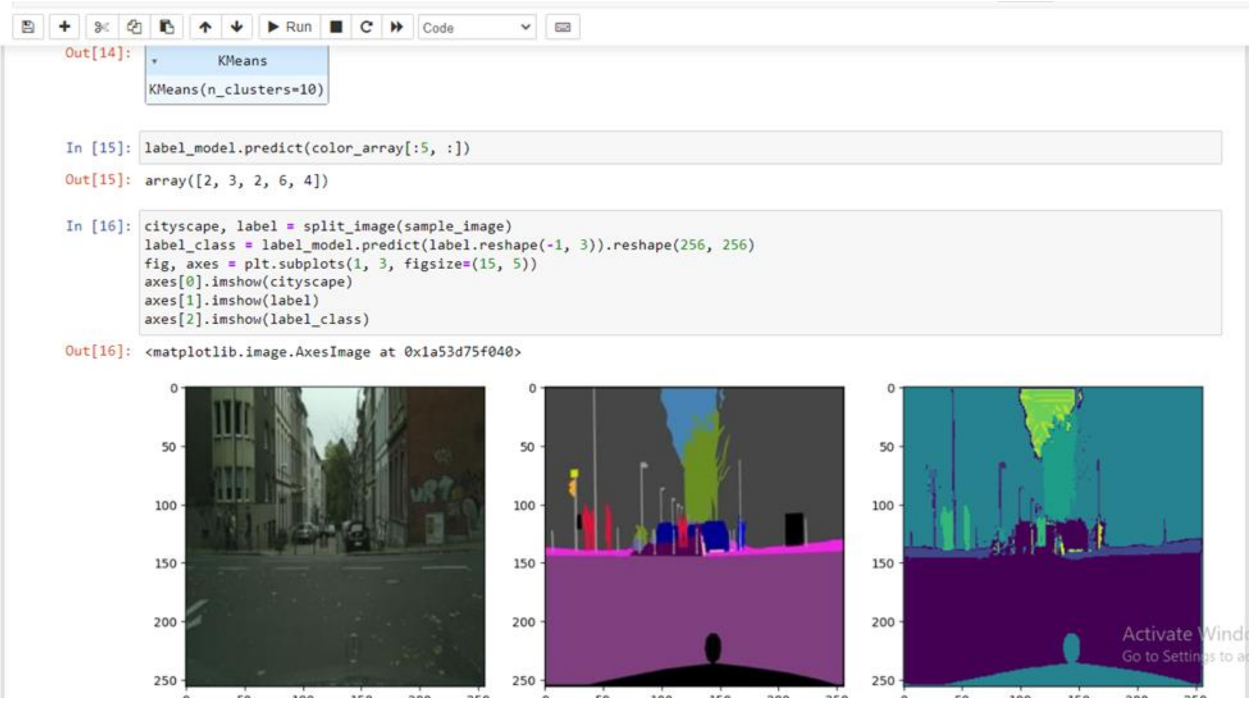
Figure 4.7a



```
In [6]: from tqdm.notebook import tqdm

In [7]: device = "cuda:0" if torch.cuda.is_available() else "cpu"
        device = torch.device(device)
        print(device)

        cpu

In [8]: data_dir = os.path.join( "/cityscapes", "cityscapes_data")

In [9]: data_dir = os.path.join( "cityscapes", "cityscapes_data")
        train_dir = os.path.join(data_dir, "train")
        val_dir = os.path.join(data_dir, "val")
        train_fns = os.listdir(train_dir)
        val_fns = os.listdir(val_dir)
        print(len(train_fns), len(val_fns))

        501 501
```

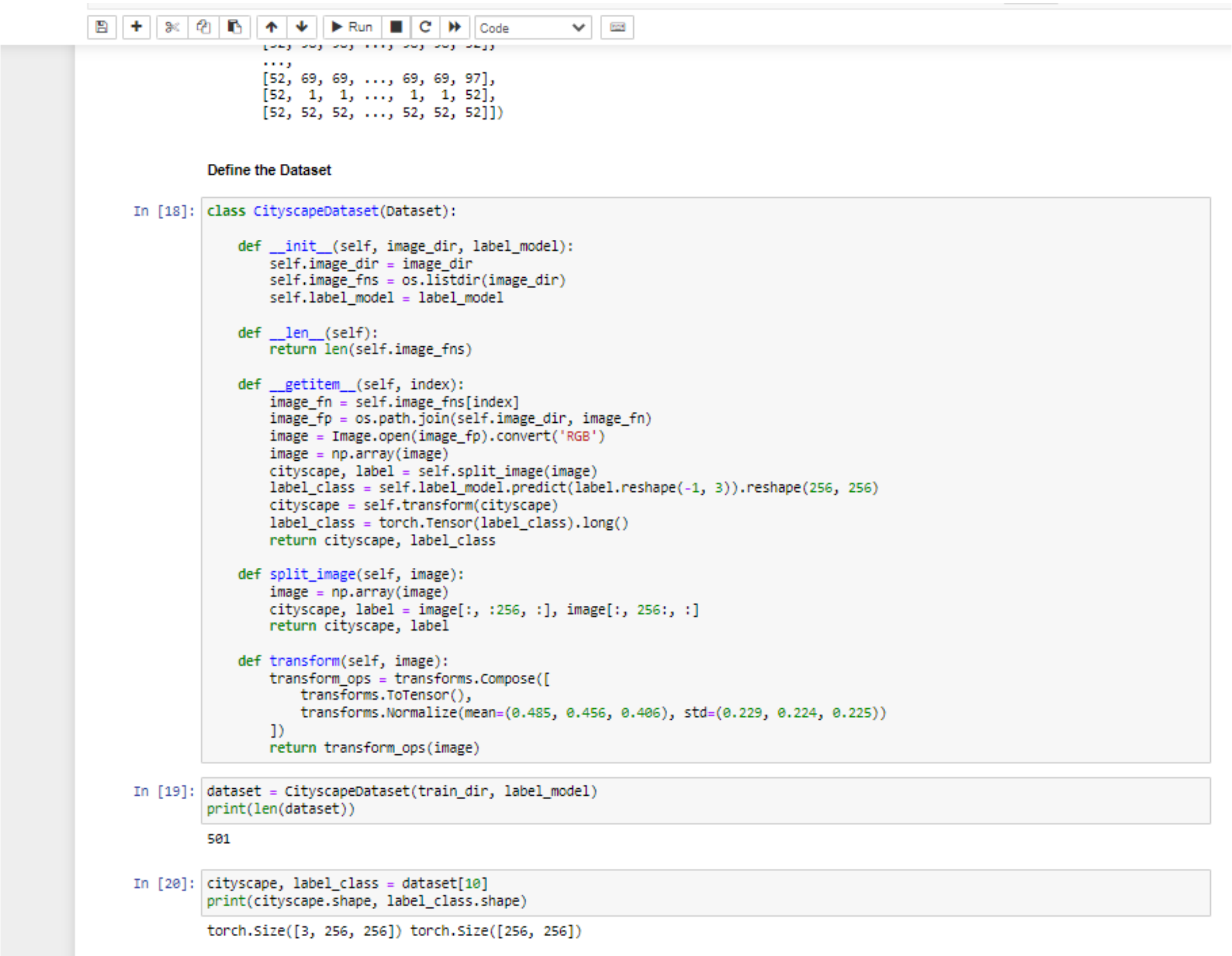
Analyse Data

```
In [10]: sample_image_fp = os.path.join(train_dir, train_fns[8])
         sample_image = Image.open(sample_image_fp).convert("RGB")
         plt.imshow(sample_image)
         print(sample_image_fp)

         cityscapes\cityscapes_data\train\105.jpg

In [11]: def split_image(image):
             image = np.array(image)
```



```
Out[14]: ▾ KMeans
         KMeans(n_clusters=10)

In [15]: label_model.predict(color_array[:5, :])

Out[15]: array([2, 3, 2, 6, 4])

In [16]: cityscape, label = split_image(sample_image)
         label_class = label_model.predict(label.reshape(-1, 3)).reshape(256, 256)
         fig, axes = plt.subplots(1, 3, figsize=(15, 5))
         axes[0].imshow(cityscape)
         axes[1].imshow(label)
         axes[2].imshow(label_class)

Out[16]: <matplotlib.image.AxesImage at 0x1a53d75f040>
```

Defining datasets in figure 4.7b below.



```
         ...,
         [52, 69, 69, ..., 69, 69, 97],
         [52,  1,  1, ...,  1,  1, 52],
         [52, 52, 52, ..., 52, 52, 52]])
```

Define the Dataset

```
In [18]: class CityscapeDataset(Dataset):

             def __init__(self, image_dir, label_model):
                 self.image_dir = image_dir
                 self.image_fns = os.listdir(image_dir)
                 self.label_model = label_model

             def __len__(self):
                 return len(self.image_fns)

             def __getitem__(self, index):
                 image_fn = self.image_fns[index]
                 image_fp = os.path.join(self.image_dir, image_fn)
                 image = Image.open(image_fp).convert('RGB')
                 image = np.array(image)
                 cityscape, label = self.split_image(image)
                 label_class = self.label_model.predict(label.reshape(-1, 3)).reshape(256, 256)
                 cityscape = self.transform(cityscape)
                 label_class = torch.Tensor(label_class).long()
                 return cityscape, label_class

             def split_image(self, image):
                 image = np.array(image)
                 cityscape, label = image[:, :256, :], image[:, 256:, :]
                 return cityscape, label

             def transform(self, image):
                 transform_ops = transforms.Compose([
                     transforms.ToTensor(),
                     transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
                 ])
                 return transform_ops(image)
```

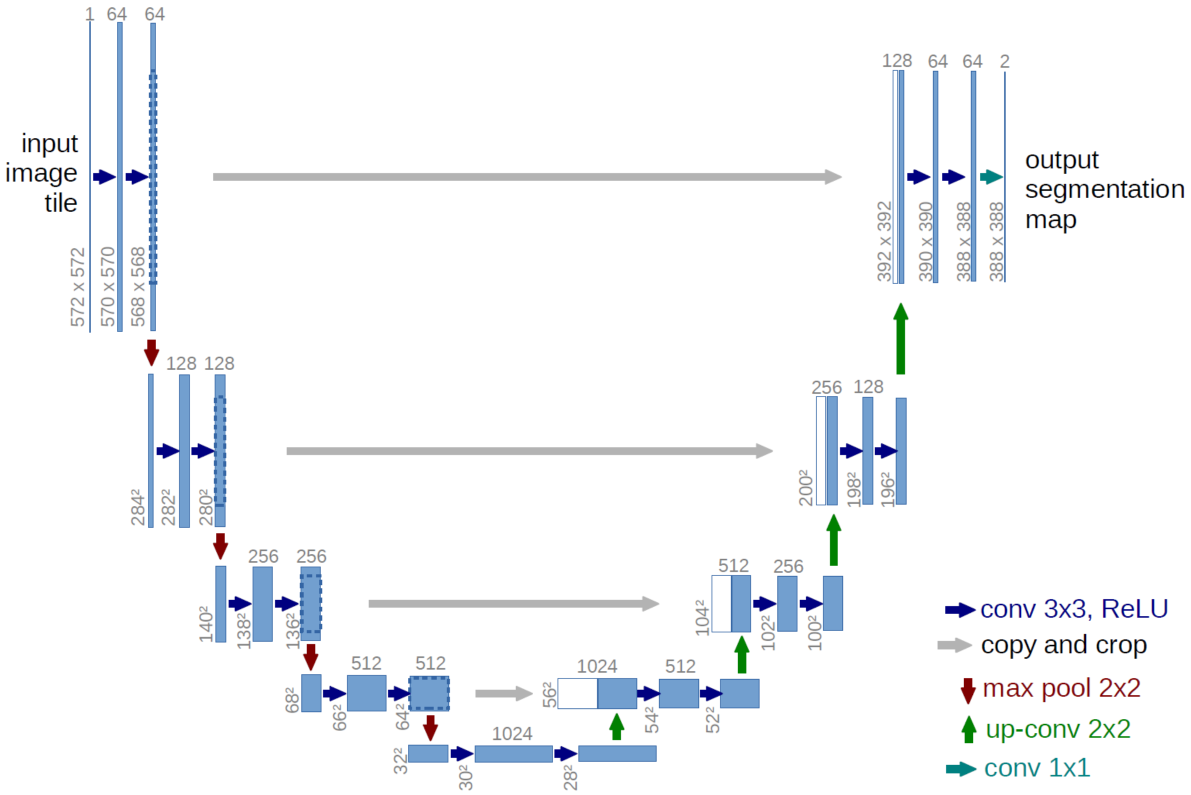
```
In [19]: dataset = CityscapeDataset(train_dir, label_model)
         print(len(dataset))

         501
```

```
In [20]: cityscape, label_class = dataset[10]
         print(cityscape.shape, label_class.shape)

         torch.Size([3, 256, 256]) torch.Size([256, 256])
```

2. Model Architecture:

The U-Net model consists of a contracting path, which captures the context of the input image, and an expanding path, which generates the segmented mask. The contracting path consists of several convolutional and pooling layers, while the expanding path consists of several transposed convolutional layers. Here is an example architecture of the U-Net model:

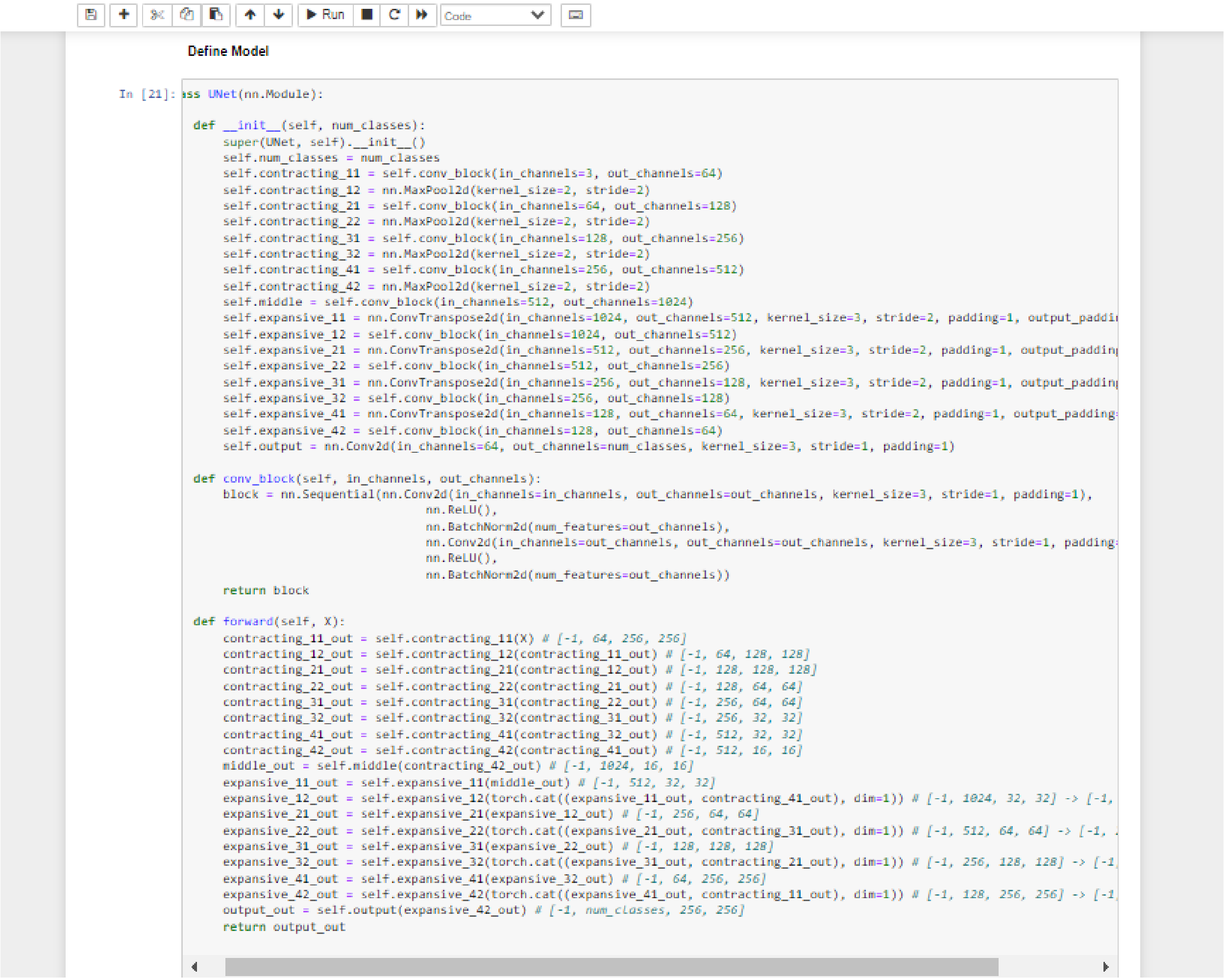
U-net Architecture



3. Model training

❖ **Define the U-Net Model:** Build the U-Net model architecture using TensorFlow. The U-Net model should consist of an encoder and decoder structure that allows for high-resolution segmentation maps.

Figure 4.**7c**



Define Model

```
In [21]: ıss UNet(nn.Module):

    def __init__(self, num_classes):
        super(UNet, self).__init__()
        self.num_classes = num_classes
        self.contracting_11 = self.conv_block(in_channels=3, out_channels=64)
        self.contracting_12 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.contracting_21 = self.conv_block(in_channels=64, out_channels=128)
        self.contracting_22 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.contracting_31 = self.conv_block(in_channels=128, out_channels=256)
        self.contracting_32 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.contracting_41 = self.conv_block(in_channels=256, out_channels=512)
        self.contracting_42 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.middle = self.conv_block(in_channels=512, out_channels=1024)
        self.expansive_11 = nn.ConvTranspose2d(in_channels=1024, out_channels=512, kernel_size=3, stride=2, padding=1, output_paddi
        self.expansive_12 = self.conv_block(in_channels=1024, out_channels=512)
        self.expansive_21 = nn.ConvTranspose2d(in_channels=512, out_channels=256, kernel_size=3, stride=2, padding=1, output_paddin
        self.expansive_22 = self.conv_block(in_channels=512, out_channels=256)
        self.expansive_31 = nn.ConvTranspose2d(in_channels=256, out_channels=128, kernel_size=3, stride=2, padding=1, output_paddin
        self.expansive_32 = self.conv_block(in_channels=256, out_channels=128)
        self.expansive_41 = nn.ConvTranspose2d(in_channels=128, out_channels=64, kernel_size=3, stride=2, padding=1, output_padding
        self.expansive_42 = self.conv_block(in_channels=128, out_channels=64)
        self.output = nn.Conv2d(in_channels=64, out_channels=num_classes, kernel_size=3, stride=1, padding=1)

    def conv_block(self, in_channels, out_channels):
        block = nn.Sequential(nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, stride=1, padding=1),
                              nn.ReLU(),
                              nn.BatchNorm2d(num_features=out_channels),
                              nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, padding
                              nn.ReLU(),
                              nn.BatchNorm2d(num_features=out_channels))
        return block

    def forward(self, X):
        contracting_11_out = self.contracting_11(X) # [-1, 64, 256, 256]
        contracting_12_out = self.contracting_12(contracting_11_out) # [-1, 64, 128, 128]
        contracting_21_out = self.contracting_21(contracting_12_out) # [-1, 128, 128, 128]
        contracting_22_out = self.contracting_22(contracting_21_out) # [-1, 128, 64, 64]
        contracting_31_out = self.contracting_31(contracting_22_out) # [-1, 256, 64, 64]
        contracting_32_out = self.contracting_32(contracting_31_out) # [-1, 256, 32, 32]
        contracting_41_out = self.contracting_41(contracting_32_out) # [-1, 512, 32, 32]
        contracting_42_out = self.contracting_42(contracting_41_out) # [-1, 512, 16, 16]
        middle_out = self.middle(contracting_42_out) # [-1, 1024, 16, 16]
        expansive_11_out = self.expansive_11(middle_out) # [-1, 512, 32, 32]
        expansive_12_out = self.expansive_12(torch.cat((expansive_11_out, contracting_41_out), dim=1)) # [-1, 1024, 32, 32] -> [-1,
        expansive_21_out = self.expansive_21(expansive_12_out) # [-1, 256, 64, 64]
        expansive_22_out = self.expansive_22(torch.cat((expansive_21_out, contracting_31_out), dim=1)) # [-1, 512, 64, 64] -> [-1,
        expansive_31_out = self.expansive_31(expansive_22_out) # [-1, 128, 128, 128]
        expansive_32_out = self.expansive_32(torch.cat((expansive_31_out, contracting_21_out), dim=1)) # [-1, 256, 128, 128] -> [-1
        expansive_41_out = self.expansive_41(expansive_32_out) # [-1, 64, 256, 256]
        expansive_42_out = self.expansive_42(torch.cat((expansive_41_out, contracting_11_out), dim=1)) # [-1, 128, 256, 256] -> [-1
        output_out = self.output(expansive_42_out) # [-1, num_classes, 256, 256]
        return output_out
```

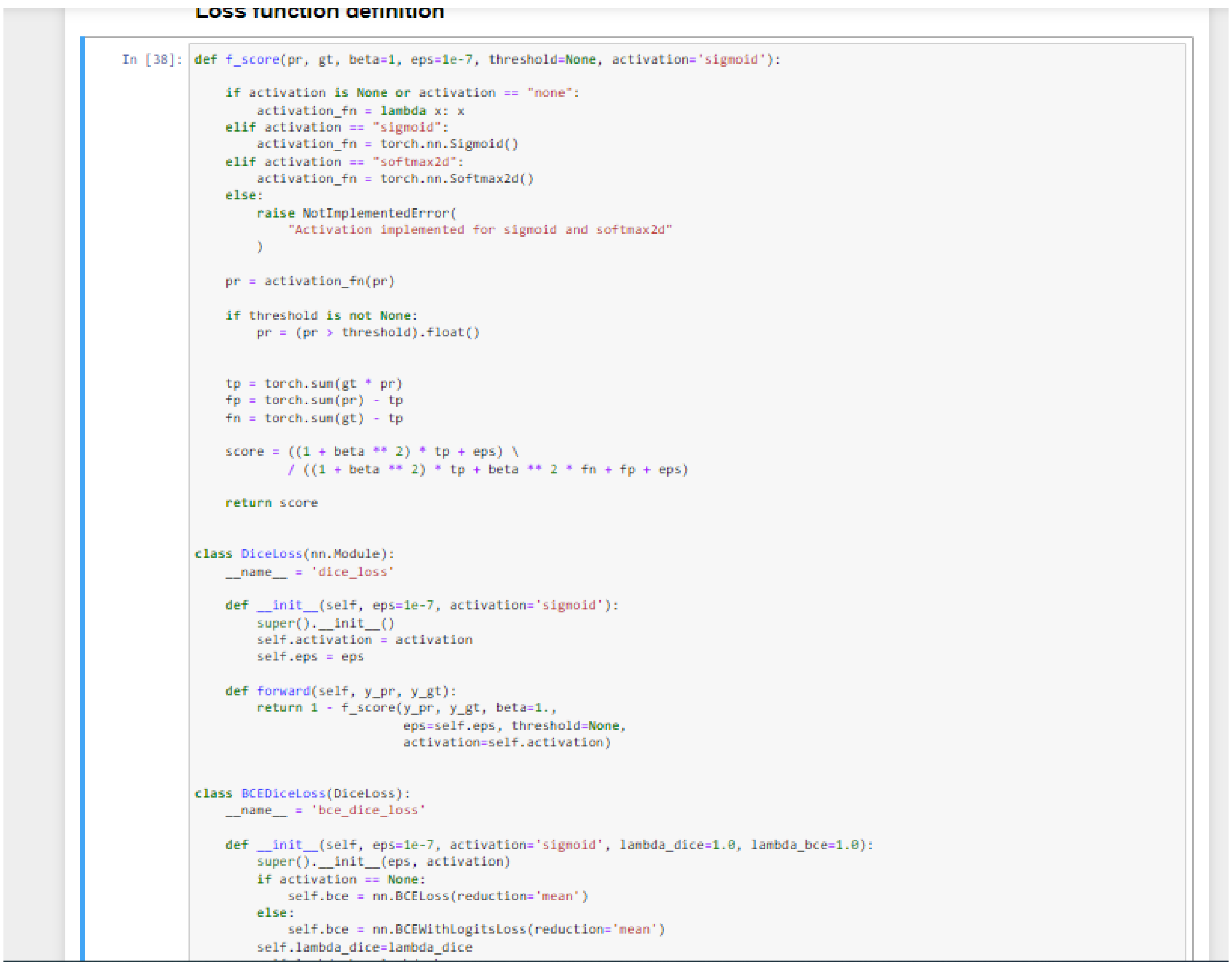
- **Compile the Model:** Define the loss function and optimization algorithm for the U-Net model. The loss function should be tailored for image segmentation, such as binary cross-entropy or dice coefficient loss.

Loss function definition

Defining loss function of the model using the code in the figure 4.7d below.

Figure 4.7d

Loss function definition

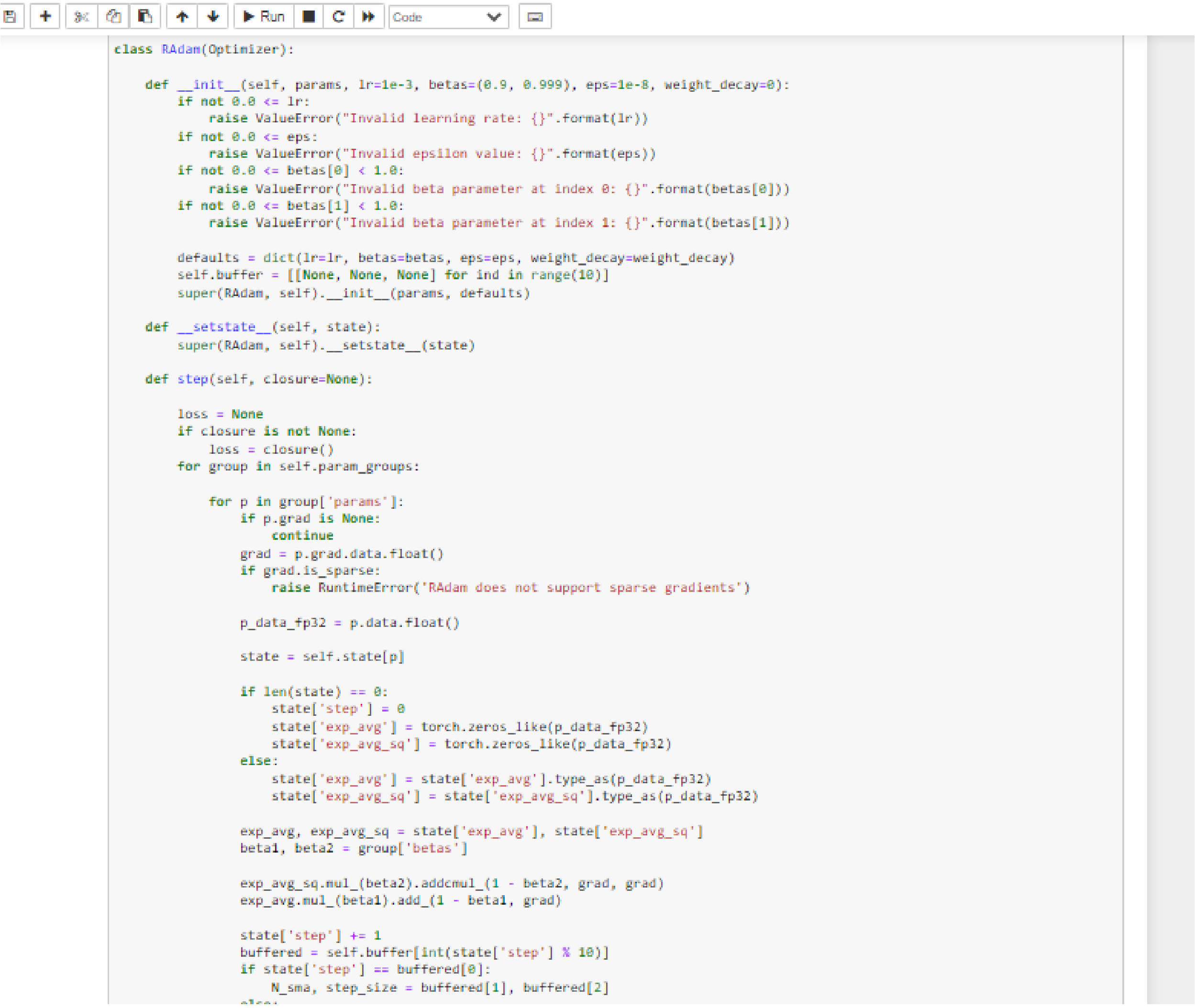


In [38]:

```
def f_score(pr, gt, beta=1, eps=1e-7, threshold=None, activation='sigmoid'):

    if activation is None or activation == "none":
        activation_fn = lambda x: x
    elif activation == "sigmoid":
        activation_fn = torch.nn.Sigmoid()
    elif activation == "softmax2d":
        activation_fn = torch.nn.Softmax2d()
    else:
        raise NotImplementedError(
            "Activation implemented for sigmoid and softmax2d"
        )

    pr = activation_fn(pr)

    if threshold is not None:
        pr = (pr > threshold).float()


    tp = torch.sum(gt * pr)
    fp = torch.sum(pr) - tp
    fn = torch.sum(gt) - tp

    score = ((1 + beta ** 2) * tp + eps) \
            / ((1 + beta ** 2) * tp + beta ** 2 * fn + fp + eps)

    return score


class DiceLoss(nn.Module):
    __name__ = 'dice_loss'

    def __init__(self, eps=1e-7, activation='sigmoid'):
        super().__init__()
        self.activation = activation
        self.eps = eps

    def forward(self, y_pr, y_gt):
        return 1 - f_score(y_pr, y_gt, beta=1.,
                           eps=self.eps, threshold=None,
                           activation=self.activation)


class BCEDiceLoss(DiceLoss):
    __name__ = 'bce_dice_loss'

    def __init__(self, eps=1e-7, activation='sigmoid', lambda_dice=1.0, lambda_bce=1.0):
        super().__init__(eps, activation)
        if activation == None:
            self.bce = nn.BCELoss(reduction='mean')
        else:
            self.bce = nn.BCEWithLogitsLoss(reduction='mean')
        self.lambda_dice=lambda_dice
```

Optimizer definition

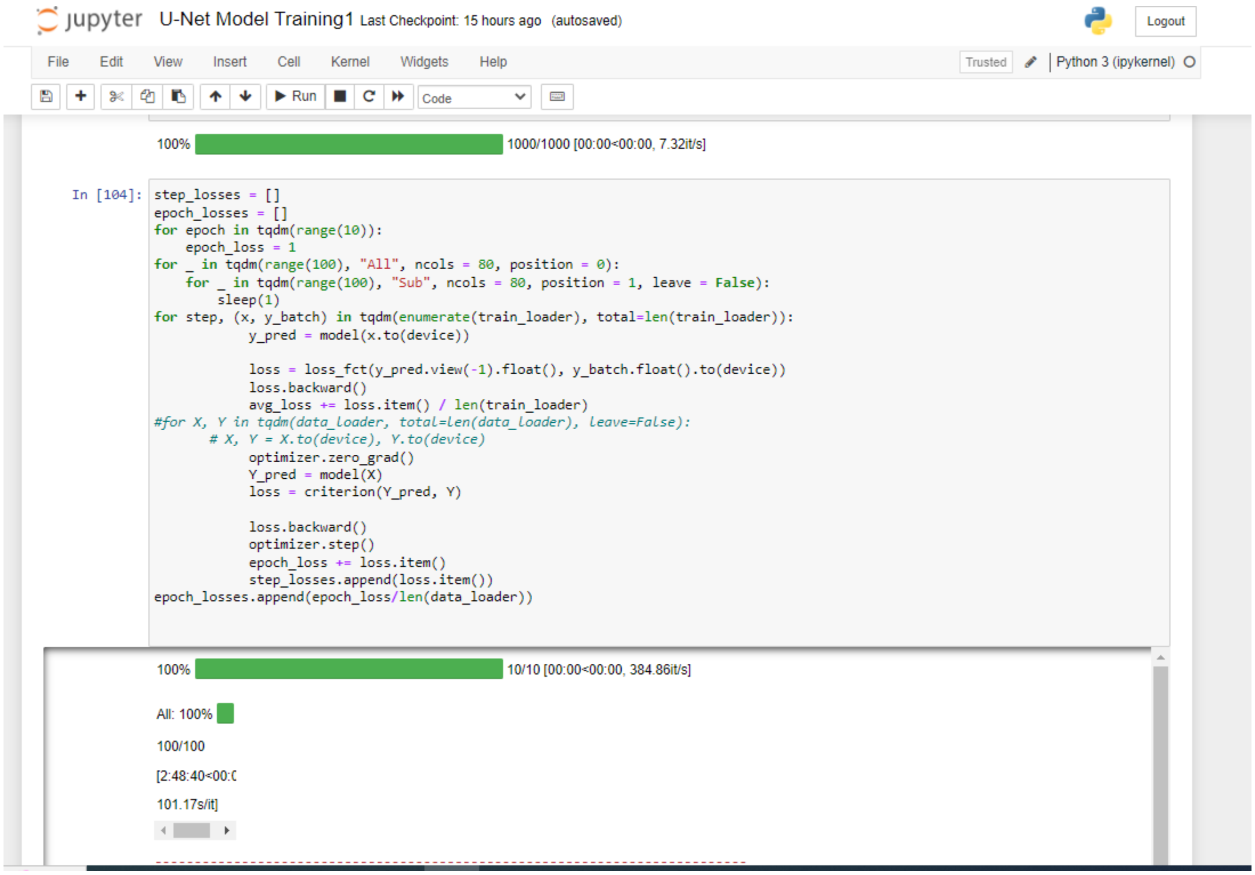
Figure 4.7e



```
class RAdam(Optimizer):

    def __init__(self, params, lr=1e-3, betas=(0.9, 0.999), eps=1e-8, weight_decay=0):
        if not 0.0 <= lr:
            raise ValueError("Invalid learning rate: {}".format(lr))
        if not 0.0 <= eps:
            raise ValueError("Invalid epsilon value: {}".format(eps))
        if not 0.0 <= betas[0] < 1.0:
            raise ValueError("Invalid beta parameter at index 0: {}".format(betas[0]))
        if not 0.0 <= betas[1] < 1.0:
            raise ValueError("Invalid beta parameter at index 1: {}".format(betas[1]))

        defaults = dict(lr=lr, betas=betas, eps=eps, weight_decay=weight_decay)
        self.buffer = [[None, None, None] for ind in range(10)]
        super(RAdam, self).__init__(params, defaults)

    def __setstate__(self, state):
        super(RAdam, self).__setstate__(state)

    def step(self, closure=None):

        loss = None
        if closure is not None:
            loss = closure()
        for group in self.param_groups:

            for p in group['params']:
                if p.grad is None:
                    continue
                grad = p.grad.data.float()
                if grad.is_sparse:
                    raise RuntimeError('RAdam does not support sparse gradients')

                p_data_fp32 = p.data.float()

                state = self.state[p]

                if len(state) == 0:
                    state['step'] = 0
                    state['exp_avg'] = torch.zeros_like(p_data_fp32)
                    state['exp_avg_sq'] = torch.zeros_like(p_data_fp32)
                else:
                    state['exp_avg'] = state['exp_avg'].type_as(p_data_fp32)
                    state['exp_avg_sq'] = state['exp_avg_sq'].type_as(p_data_fp32)

                exp_avg, exp_avg_sq = state['exp_avg'], state['exp_avg_sq']
                beta1, beta2 = group['betas']

                exp_avg_sq.mul_(beta2).addcmul_(1 - beta2, grad, grad)
                exp_avg.mul_(beta1).add_(1 - beta1, grad)

                state['step'] += 1
                buffered = self.buffer[int(state['step'] % 10)]
                if state['step'] == buffered[0]:
                    N_sma, step_size = buffered[1], buffered[2]
                else:
```

- ❖ **Train the Model:** Train the U-Net model on the training data using the compiled loss function and optimization algorithm. The training process should be monitored for validation loss and accuracy.

Figure 4.**7f**



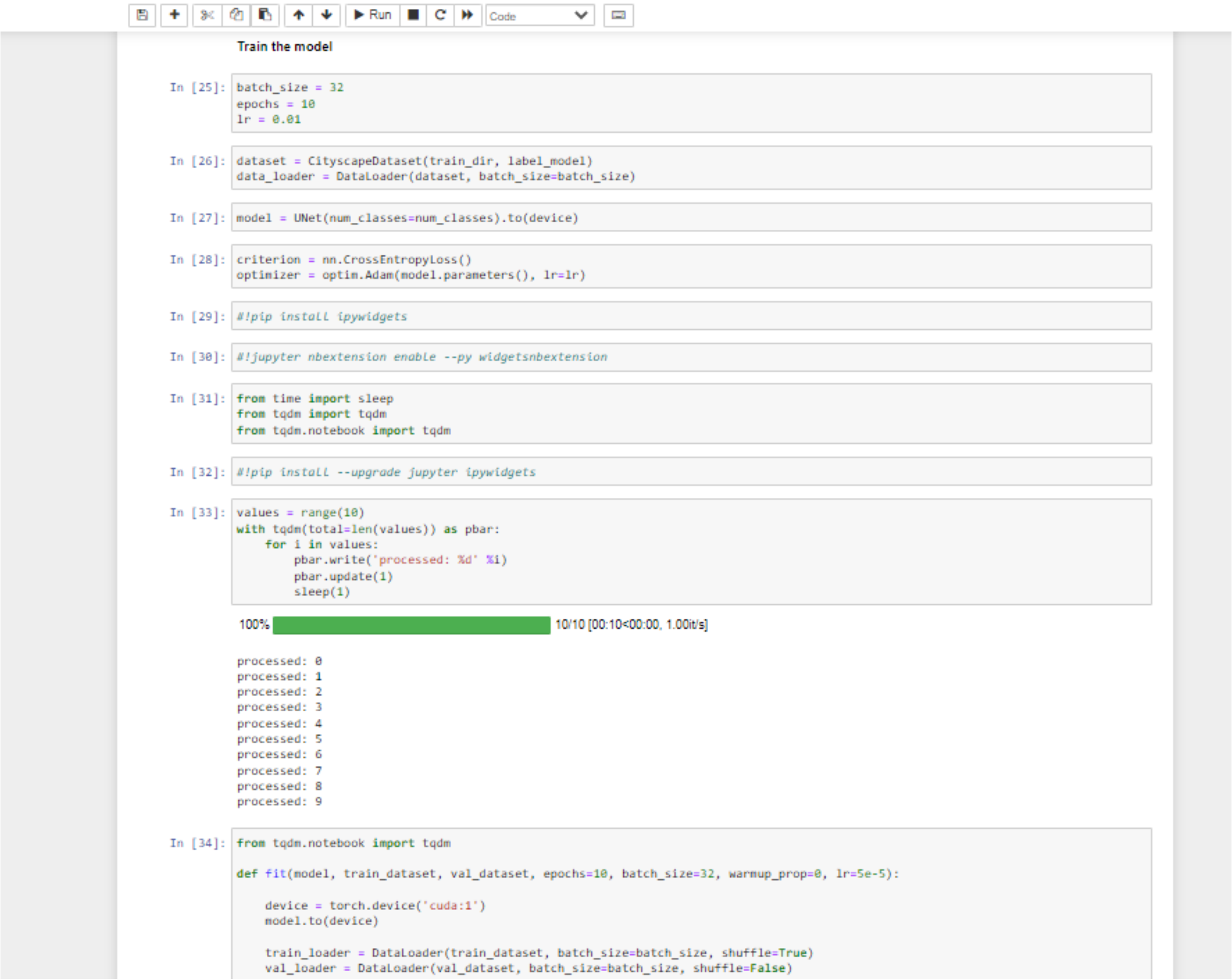
```
100% ██████████████████████████ 1000/1000 [00:00<00:00, 7.32it/s]
```

In [104]:

```
step_losses = []
epoch_losses = []
for epoch in tqdm(range(10)):
    epoch_loss = 1
for _ in tqdm(range(100), "All", ncols = 80, position = 0):
    for _ in tqdm(range(100), "Sub", ncols = 80, position = 1, leave = False):
        sleep(1)
for step, (x, y_batch) in tqdm(enumerate(train_loader), total=len(train_loader)):
            y_pred = model(x.to(device))

            loss = loss_fct(y_pred.view(-1).float(), y_batch.float().to(device))
            loss.backward()
            avg_loss += loss.item() / len(train_loader)
#for X, Y in tqdm(data_loader, total=len(data_loader), leave=False):
       # X, Y = X.to(device), Y.to(device)
            optimizer.zero_grad()
            Y_pred = model(X)
            loss = criterion(Y_pred, Y)

            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
            step_losses.append(loss.item())
epoch_losses.append(epoch_loss/len(data_loader))
```

```
100% ██████████████████████████ 10/10 [00:00<00:00, 384.86it/s]

All: 100% █

100/100

[2:48:40<00:(

101.17s/it]
```

Figure 4.7g



Train the model

```
In [25]: batch_size = 32
         epochs = 10
         lr = 0.01

In [26]: dataset = CityscapeDataset(train_dir, label_model)
         data_loader = DataLoader(dataset, batch_size=batch_size)

In [27]: model = UNet(num_classes=num_classes).to(device)

In [28]: criterion = nn.CrossEntropyLoss()
         optimizer = optim.Adam(model.parameters(), lr=lr)

In [29]: #!pip install ipywidgets

In [30]: #!jupyter nbextension enable --py widgetsnbextension

In [31]: from time import sleep
         from tqdm import tqdm
         from tqdm.notebook import tqdm

In [32]: #!pip install --upgrade jupyter ipywidgets

In [33]: values = range(10)
         with tqdm(total=len(values)) as pbar:
             for i in values:
                 pbar.write('processed: %d' %i)
                 pbar.update(1)
                 sleep(1)

         100% 10/10 [00:10<00:00, 1.00it/s]

         processed: 0
         processed: 1
         processed: 2
         processed: 3
         processed: 4
         processed: 5
         processed: 6
         processed: 7
         processed: 8
         processed: 9

In [34]: from tqdm.notebook import tqdm

         def fit(model, train_dataset, val_dataset, epochs=10, batch_size=32, warmup_prop=0, lr=5e-5):

             device = torch.device('cuda:1')
             model.to(device)

             train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
             val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
```

- **Evaluate the Model:** Evaluate the trained U-Net model on the test data to determine its accuracy and performance. The evaluation metrics should include precision, recall, and F1-score.

CHAPTER FIVE

ACHIEVEMENTS, RECOMMENDATIONS, CHALLENGES AND CONCLUSION

5.1Achievements

Through the project:

- GrabCut has proven to be a highly effective algorithm for object segmentation especially in scenarios where there is a clear contrast between background and foreground.

- Grabcut has opened up new possibilities for interactive image manipulation and object tracking in dynamic environment.
- Finding the optimal parameter settings using empirical tuning to some image characteristics.
- GrabCut provides a robust framework for foreground extraction, enabling users to easily separate objects from their backgrounds for various editing purposes.

What the developer learnt:

- I have gained a deeper understanding of segmentation algorithms, graph cuts, Gaussian mixture models and iteractive optimization processes.
- I have learnt how to handle user interactions effectively and designed user-friendly interfaces for providing the necessary inputs.
- I have learnt practical experience in implementing graph cut algorithms, understanding their intricacies and optimizing them for efficient and accurate segmentation.

5.2Challenges

1. selecting accurate bounding box by the user is challenging especially when dealing with irregular shape objects.
2. Grabcut relies on user-provided input in form of initial bounding box to guide the segmentation process
3. when there is no clear contrast between the foreground object and the background or similar colors,texture or pattern the algorithm struggle to accurately differentiate between foreground and background.
4. Large images affects the performance and computational complexity and also real-time applications may require additional optimization to achieve acceptable performance levels.
5. selecting appropriate parameter values can significantly impact the segmentation quality.

5.3Future recommendations

By implementing the following future recommendations, the GrabCut image segmentation algorithm using the U-Net model can achieve even higher accuracy and be applied to a wider range of applications.

1. Improving the U-Net model architecture: The performance of the U-Net model can be improved by modifying the model architecture. For example, using deeper networks, adding skip connections, and incorporating attention mechanisms can lead to better segmentation accuracy.
2. Increasing the size of the training dataset: The U-Net model requires a large training dataset to achieve high accuracy. Increasing the size of the training dataset can lead to better generalization performance and more robust segmentation.
3. Incorporating other deep learning models: The U-Net model can be combined with other deep learning models such as the Mask R-CNN to improve the accuracy of the

segmentation. This can be done by using the U-Net model to generate initial segmentation masks that are then refined by the Mask R-CNN.

4. Improving user interactions: The GrabCut algorithm relies on user interactions to refine the segmentation. Improving the user interactions by incorporating more intuitive and user-friendly interfaces can lead to better segmentation accuracy and user experience.
5. Expanding the applications of the algorithm: The GrabCut image segmentation algorithm using the U-Net model has numerous applications in various fields such as medical imaging, robotics, and computer vision. Further research can explore the application of the algorithm in other fields such as autonomous driving, agriculture, and surveillance.

5.4 Summary

The GrabCut image segmentation project involves developing an algorithm that can accurately segment an image into foreground and background regions. The algorithm is based on graph-cut optimization and involves iteratively refining an initial segmentation mask based on user-provided inputs.

The project can be implemented using various programming languages such as Python, MATLAB, and C++. Various image processing and machine learning libraries such as OpenCV, scikit-image, and TensorFlow can be used to load, manipulate and segment images. The algorithm can be trained and validated using a U-Net model, which is a deep learning model that has been shown to achieve state-of-the-art performance in image segmentation tasks.

The project requires a clear understanding of image processing concepts, graph theory, and machine learning techniques. The implementation of the project involves several steps, including image preprocessing, initial segmentation, user interaction, graph-cut optimization, and post-processing.

The success of the GrabCut image segmentation project depends on the accuracy of the initial segmentation, the effectiveness of the user interactions, the quality of the optimization algorithm, and the performance of the machine learning model. The project has numerous applications in various fields, including medical imaging, robotics, and computer vision.

5.5 Conclusion

In this paper we have surveyed some of the image segmentation approaches. These techniques are mostly used to detect the points, edges, line detection, patterns etc. As of now we have seen in some cases the author's has used SegNet encoding decoding technique, UNet++ in medical imaging, encoding decoding with Atrous etc. What we have noticed that each technique have been used based on the data and its quality. The techniques are mentioned above are is used in many domains like face identification, medical science to detect the cancerous cells from images, recognising the pattern, roads, satellite image classification etc. This survey may help the researcher's for the further research in the domain of image segmentation.

REFERENCES

1. Conference on Computer Vision and Image Processing (pp. 148-155). Zhou, Zongwei, et al. "Unet++: A nested u-net architecture for medical image segmentation." *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer, Cham, 2018. 3-11.
2. Gur, Shir, et al. "Unsupervised microvascular image segmentation using an active contours mimicking neural network." *Proceedings of the IEEE International Conference on Computer Vision*. 2019.
3. Badrinarayanan, Vijay, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation." *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017): 2481-2495.
4. Chen, Liang-Chieh, et al. "Encoder-decoder with atrous separable convolution for semantic image segmentation." *Proceedings of the European conference on computer vision (ECCV)*. 2018.
5. Senthilkumaran, N., and S. Vaithegi. "Image segmentation by using thresholding techniques for medical images." *Computer Science & Engineering: An International Journal* 6.1 (2016): 1-13.
6. https://cnvrg.io/image-segmentation/
7. Rother, C., Kolmogorov, V., & Blake, A. (2004). "GrabCut": Interactive foreground extraction using iterated graph cuts. In Proceedings of the ACM SIGGRAPH Conference on Computer Graphics (pp. 309-314).
8. Boykov, Y., Veksler, O., & Zabih, R. (2001). Fast approximate energy minimization via graph cuts. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(11), 1222-1239.
9. Li, Y., & Yu, J. (2018). Interactive image segmentation using GrabCut algorithm. In Proceedings of the IEEE Conference on Robotics and Automation (pp. 4644-4649).
10. Dubská, M., Kolingerová, I., & Sochor, J. (2017). GrabCut segmentation algorithm for image processing in surveillance systems. In Proceedings of the 41st International Conference on Telecommunications and Signal Processing (pp. 41-44).
11. Ries, J., & Demirci, M. F. (2019). Image segmentation using GrabCut algorithm and Markov random fields. In Proceedings of the International