CSI 6900
INTENSIVE GRADUATE PROJECT IN COMPUTER SCIENCE

Scene parsing using Machine Learning

Faculty of Graduate and Postdoctoral Studies (Master of Computer Science)

Name: Snehal Sudhir Bhole
Student Number: 300318105
Course Code: CSI 6900
Submission Date: 29-04-2024

Abstract

Recent years have seen incredible achievements in Computer Vision, expanding the limits of understanding the visual world using Machine learning. Scene parsing or scene segmentation is the center of these technological advancements. Scene Segmentation, commonly known as scene understanding involves labelling the pixel in an image into meaningful segments assigning each pixel label to which the object belongs. Scene parsing is predominantly used in various crucial applications such as Computer Vision, Medical Imaging, Augmented Reality, security, and surveillance, thus can be seen as the backbone for various real-time systems.

The report discusses various Scene parsing approaches, their advantages, and disadvantages while training U-Net and YOLO models on the ADE20K dataset. The paper also discusses data cleaning and preprocessing performed, and model performance using distinct evaluation metrics. In the end, all the models including ResNet, Xception, U-Net, YOLO and SAM are then integrated with a ReactJS-based front-end application with Flask backend, handling various API endpoints developed and tested on the image processing task.

Keywords

Scene Parsing, Semantic Scne Understanding, ADE20K Dataset, Object Detection, Classification, Semantic Segmentation, Instance Segmentation, Pixellib, ResNet101, Deeplab - Xception, YOLO, U-Net, SAM, Class Imbalance

Table of Contents

List of Figures

List of Tables

List of Equation

1. Introduction

The field of Computer Vision has witnessed remarkable progress in recent years, with Machine Learning techniques pushing the boundaries of how we understand the visual world. At the forefront of these advancements lies scene parsing, also known as scene segmentation. Understanding the visual content of an image is the fundamental challenge that involves labelling or categorizing each pixel in an image to represent the objects, regions, and their attributes within the scene. The goal is to parse an image into meaningful segments assigning each pixel label to which the object belongs. Scene parsing has a plethora of applications in various domains. It is crucial in self-driving Cars to analyze their surroundings. Robotics, medical imaging, augmented reality, environmental planning, surveillance, and security are some of the areas where Scene parsing is a backbone of technology. When we consider scene parsing, there are three modules involved namely Scene understanding, Positioning, and Navigating. In the case of Autonomous driving or an integrated system using scene parsing as its backbone, the final step is control execution or path planning [1].

In general, Scene parsing has 2 main tasks:

1. Object Detection: It deals with identifying and locating obstacles in the form of bounding boxes.
2. Scene Segmentation: It deals with assigning semantic category labels to each pixel (Refinement of object).

When we think of Scene segmentation for autonomous vehicles, there are three sub-streams of Scene segmentation: Full Scene segmentation, Object Instance Segmentation and Lane line segmentation as shown in Figure 1.

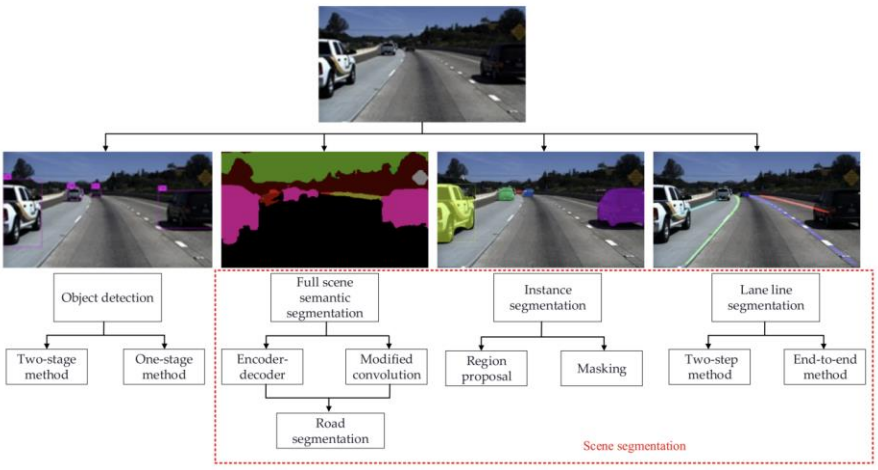


Figure 1: The overall scene segmentation task [1]

1.1. Objectives

This research project aims to delve into the realm of scene parsing through the development and implementation of cutting-edge deep learning algorithms. The project's objectives are meticulously crafted to address key challenges and advancements in the field of computer vision, particularly in the context of accurate and efficient scene parsing. The outlined objectives of the project are as follows:

1. Study and understand deep learning algorithms specifically tailored for scene parsing, including learning about segmentation modules, algorithm training and development, and critical analysis of current architecture.

2. Fine-tune algorithm parameters to minimize loss and achieve high accuracy in scene parsing tasks.

3. Optimize the trained models and evaluate the model's performance against existing architectures.

4. Employ metrics to interpret and comprehend the performance of the developed models, aiding in objective evaluation and comparison.

5. Develop a user-friendly Graphical User Interface (GUI)-based application integrated with the deep learning model. This application will facilitate testing and analysis of the model's performance by taking user inputs and analyzing images to detect various objects within the scene.

The project will primarily utilize Python and its libraries, leveraging development environments such as PyCharm or VSCode. TensorFlow and PyTorch, along with computer vision libraries and other Python modules as deemed necessary, will form the core technologies driving the project forward. Leveraging existing research on the topic will provide a solid foundation for innovation and advancement in scene-parsing algorithms within the scope of this academic endeavour.

1.2. Report Overview

The report illustrates the tasks completed during the winter term. The following sections are included in the report:

- Introduction
- Literature Review
- Dataset
- Methodology and Results
- Outcome and Evaluation
- Conclusions
- References

2. Literature Review

The field of scene parsing using machine learning has witnessed significant advancements in recent years. Scene parsing involves the pixel-wise labelling of objects and regions within an image, enabling machines to comprehend the visual content of a scene at a granular level. This literature review aims to explore and synthesize key research contributions, methodologies, challenges, and advancements in scene parsing using machine learning techniques.

Early approaches to scene parsing relied heavily on handcrafted features and traditional machine learning algorithms such as support vector machines (SVMs) and random forests. These methods often faced challenges in handling complex scene structures, intricate object boundaries, and variations in lighting conditions.

2.1. Object Detection

Object detection is defined as identifying and locating objects within images or videos. It goes beyond simply classifying an image, object detection shows the exact position of an object in an image, often by drawing a bounding box around it.

There are 2 approaches used for object detection, the two-stage method, and the one-stage method.

- Two-Stage Method
- One-Stage Method

The most popular deep learning technique used for image processing tasks is Convolutional Neural Networks. It is parameterized by its weights vector $\theta = [W, b]$ where W is weights controlling or modelling the non-linearity of data for interneural connections and b is a set of bias values, used to provide flexibility to adapt and modify concerning input and final labels [1]. When the neural network is being trained, the weights and biases are updated to model the non-linearity or linearity in the data. CNN typically uses local spatial correlations between image pixels and implements a discriminator layer, for high-level representation of objects. Deep learning based on CNN can automatically extract features and learn more complex abstract representations of the input. Owing to these benefits, CNN-based deep learning techniques have been applied extensively and successfully in the field of autonomous driving scene interpretation. The learning capability of deep learning increases exponentially with the depth of the model.

2.2. Two-stage Methods

A Region-Based Convolutional Neural Network (R-CNN) was proposed by Girshick et al. in 2014[2], which integrates CNN with region proposal extracts. This is one of the effective object detection techniques that use CNN to extract feature maps instead of block-wise orientation histograms. By employing selective searching, the approach produces many areas and extracts information from region proposals. Nevertheless, the R-CNN has the subsequent issues: (1) It takes a lot of time since each extracted region proposal must be processed separately; (2) It is not an end-to-end network model; and (3) The extracted region proposals need to be cropped or twisted to a fixed size, which leaves missing

data or a distorted image. Table 1 summarises two-stage algorithms used for the segmentation of images.

Network	Description	Advantage	Disadvantage
RCNN Region-based – CNN	→ Combination of region-based proposals + CNN to extract features classifying and refining the bounding boxes using a deep neural network. → Localization and classification are separate.	→ High detection accuracy.	→ Extracted region proposals must be cropped to a fixed size which leads to missing information or distorted image. → Time–consuming (region proposals are processed separately). → Not an end-to-end network model.
SPPnet Spatial Pyramid Pooling	→ Can generate fixed-length output irrespective of input size. → Robust to-object deformation. → Can extract information from various scales.	→ Can work with any image without resizing them. → Reduced the training and inference times.	→ Limitations to use deeper feature extractors. → May not be optimal for capturing context and information across a wide range of input image sizes as it uses a fixed set of pooling regions.
Fast R-CNN	→ Uses ROI Pooling layer+ hierarchical sampling+ optimizes the classifier and bounding box regressors with a multi- task loss instead of training separately.	→ Uses a robust L1 loss instead of L2 loss for regression.	→ Computationally expensive during training and inference. → Higher memory requirements. → Canbe sensitive to hyperparameter choices.
Faster RCNN	→ Region Proposal Networks (RPN).	→ The RPN model is trained separately from the Fast R-CNN classification pipeline. → RoI sizes can be determined. → The RPN model predicts the probability and location of an object of an anchor.	→ High Computational Cost. → Dependency on Anchor Boxes. → Difficulty in Handling Small Objects. → Limited Adaptability to Extreme Aspect Ratios.
Feature Pyramid Networks	→ Provides multi-scale feature representations that can be handy in object detection by supporting scale invariance. → Combines low-resolution, semantically strong features with high-resolution, semantically weak features via a top-down	→ Better Localization Accuracy. → Improved Semantic Information.	→ Larger time to compute features of multiple levels.

	pathway and lateral connections.		
Mask R-CNN	→ Solve a slightly different problem of instance segmentation.	→ Fixes the ROI Pooling layer by addressing the misalignments in slicing.	→ Not Real-Time. → Inability to Handle Overlapping Instances. → High Resource Requirements.

Table 1: Two-stage segmentation Algorithms [3]

2.3. One-Stage Methods

Redmon et al. proposed You Only Look Once-V1 (YOLO-V1) [4], a one-step detection approach, in 2015. It resolved the problem of detection speed, which plagued the two-stage detection techniques. The bounding boxes and corresponding class probabilities were geographically separated by the authors using object detection as a regression problem. Bounding boxes and class probabilities are directly predicted by a single neural network from complete images in a single assessment. The detection pipeline may be immediately tuned from start to finish for detection performance because it is a one-stage network. Both training and prediction involve the prediction of the entire picture, which efficiently utilizes context data to lower the false detection rate. Table 2 shown below summarizes two-stage methods used in image processing tasks.

Network	**Description**	**Advantage**	**Disadvantage**
YOLO	→ Detects bounding boxes at onceby dividing the input image into a grid, predicting bounding boxes and confident scores + non-maximum suppression to remove duplicate detections.	→ Fast Object detector. → Simplicity in architecture.	→ More localization errors than Fast R-CNN. → Can detect max of 2 objects of the same class in a grid. → Difficult to predict object data whose aspect ratios were not present in training data, May struggle with small object detection. → Learns from coarse object features due to the down-sampling process.
YOLO v2	→ Capable of identifying 9000 categories with multi-scale detection and object classification.	→ Improved Accuracy Better handling of Small Objects. Detect a Larger number ofObject Categories.	→ More Complex Architecture. → Longer Training Time. → Dependency on large-scale datasets.

Yolo V3	→ Uses logistic regression to assign objectness scores to anchor boxes. → Trains binary classifiers for giving multiple labels to the same box. → Use k-means to determine bounding box priors with three prior boxes for three sizes.	→ Improved accuracy. Support for various objectsizes. → Robust performance.	→ Increased computational complexity. → Longer training time. → Dependency on large-scale datasets.
YOLO V4	→ Real-time, open source, single shot, and DarkNet framework. → Find the optimal balance by experimenting with many changes categorized as bag-of-freebies and bag-bag-of specials.	→ High accuracy and improved speed. → Efficient handling of various object sizes and scales Integration of CSPDarknet-53 and PANetfor improved features.	→ Increased complexity and resource requirements Complex Architectures.

Table 2: One-Stage Methods (Different YOLO Variants)

2.4. Scene Segmentation

Scene semantic segmentation is segmenting object categories at the pixel level in a full image. The approaches for full-scene segmentation are divided into two categories: encoder-decoder structure models and modified convolution structure models. Traditional semantic segmentation methods generally rely on hand-crafted features usually tailored for specific tasks. These methods do not offer ideal performance when it comes to speed and accuracy.

2.5. Encoder-Decoder Structure Models

Encoder-Decoder structure is a commonly used architecture of semantic segmentation algorithms. The encoder of the model is a CNN network that processes input images and extracts high-level features. As the network goes deeper, the spatial resolution decreases while semantic information increases.
The decoder is responsible for generating segmentation masks by up-sampling the features extracted by the encoder to the original image size. It works with low-resolution feature maps and upscales them while incorporating skip connections from the encoder to preserve spatial information. Common techniques used in the decoder include transposed convolutions (also known as deconvolutions or up-sampling layers) and bilinear interpolation.

Skip Connections are vital for retaining fine-grained spatial information during up-sampling. It deals with concatenating feature maps from the encoder with the up-sampled feature maps in the decoder. This helps the model in localizing the objects to improve accuracy, especially small or multiple objects in an image.

2.6. Modified Convolution Structure Models

The Modified Convolution Structure image segmentation models are creative modifications to conventional convolutional neural networks, aimed to improve the precision and performance of segmentation tasks. To obtain more accurate and comprehensive segmentation results, these adjustments usually focus on important elements such as the convolutional layers, pooling layers, skip connections, and output layers.

Paper Title	**Algorithms used**	**Dataset**	**Limitations**
Pyramid Scene Parsing Network [5]	pyramidscene parsing network +FCRN	ADE20K dataset	Small Dataset
Image parsing with a wide range of classes and scene-level context [6]	Segmentation and Feature Extraction + Label Likelihood Estimation+ Fusing Classifier	SIFTflow and LMSun	Intensive Resources to train
SYGNet: A SVD-YOLO based GhostNet for Real-time Driving Scene Parsing [7]	SygNet, YOLO and Ghostnet	KITTI Dataset	Moderate Real-time performance
Semantic understanding of scenes through the Ade20k dataset [8]	External Annotators, SegNet, FCN-8s, DilatedVGG, Mask R-CNN, DilatedResNet, PSPNet with a dilated ResNet- 50	Densely annotated Dataset (Annotated with external annotator on ADE20K, COCO, Pascal datasets)	Need for densely annotated images as the model fails to detect the concepts in some images that have occlusions or require high-level context reasoning
Recurrent Scene Parsing with Perspective Understanding in the Loop [9]	MatConvNet Toolbox, trained using SGD with pre-trained ResNet50 and ResNet 101 with depth-predicting gateway	NYUD-depth-v2, Cityscapes, Stanford- 2D-3D, SUN-RGBD	Approach converges after a few iterations and performance saturates
Panoptic Segmentation [10]	Mask R-CNN+COCO, Megvii, G-RMI, PSPNet multi-scale	Cityscapes, ADE20k, Mapillary Vistas	Cannot have overlapping segments
OneFormer: One Transformer to Rule Universal Image Segmentation [11]	OneFormer model, Mask2Former, ConvNeXt, DiNAT with task guided queries	Cityscapes, ADE20k, COCO	Intensive Resources to train
Simple Open-Vocabulary Object Detection with Vision Transformers [12]	Vision Transformers	Objects 365, COCO, LVIS	Work better on large-scale data.

Table 3: Literature Review

Dilated convolutions are a frequently used tweak that allows the network to capture more context information without going overboard with parameter counts. By distributing the kernel elements using a dilation rate parameter, dilated convolutions preserve computational efficiency while giving the network a wider receptive field. The incorporation of skip connections, which are frequently

influenced by architectures like the U-Net, is another change. By creating direct links between the encoder and decoder layers, skip connections enable high-resolution data to be sent across the network. This aids in maintaining minute details throughout the segmentation process, which is especially helpful for jobs with complex object boundaries.

In addition, adjustments are made to the output layer structure to guarantee that segmentation goals are met. This might include using pixel-wise classification layers, in which the network extracts characteristics from each pixel and assigns a class label to it. Additionally, segmentation boundaries and overall quality can be enhanced by incorporating post-processing techniques like conditional random fields (CRFs).

The literature review briefed in Table 3, reveals a diverse landscape of segmentation models, each with unique strengths and considerations. ResNet with PixelLib offers simplicity and efficiency, ideal for tasks with moderate complexity. Xception coupled with DeepLab showcases detailed feature extraction capabilities, albeit with higher computational demands. YOLO stands out for real-time object detection, while UNet excels in medical imaging and general segmentation tasks. SAM's incorporation of spatial attention mechanisms enhances scene understanding. In deciding which model to use, task requirements such as speed, accuracy, and resource constraints play pivotal roles. For real-time applications, YOLO is a compelling choice, while U-Net shines in precise segmentation tasks. DeepLab and SAM offer advanced features but may require higher computational resources. Ultimately, the choice hinges on balancing task-specific needs with model capabilities and computational feasibility.

3. Dataset

The ADE20K semantic segmentation dataset [13] is a comprehensive collection comprising over 20,000 scene-centric images meticulously annotated with pixel-level labels for objects and their parts. Within this dataset, there exist 150 distinct semantic categories encompassing elements like sky, road, and grass, as well as discrete objects such as person, car, and bed. Additionally, the dataset encompasses scene-specific images, including those from industrial, indoor, and outdoor settings. Figure 2 below illustrates various categories alongside representative images associated with each category.

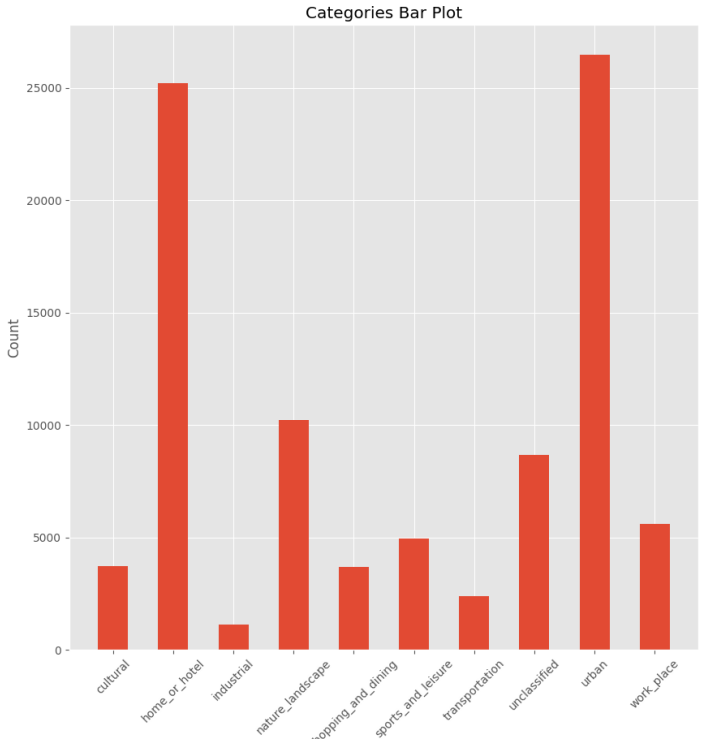


Figure 2: Categories Count Bar Plot

3.1. Exploratory Data Analysis

The ADE20k Dataset utilized in this project serves as a benchmark dataset renowned for its densely annotated nature, featuring coloured masks delineating various objects within a scene. These coloured data masks, representing labelled images, are systematically converted into NumPy arrays wherein distinct class values correspond to pixel values. Figure 3 shows the 150 classs present in the dataset. Subsequently, these masks are employed in the training phase of neural network models. Notably, the original dataset comprises coloured masks, yet for the neural network training process, grayscale annotations are employed. These annotations are obtained following the guidelines delineated on Gluon CV's data preparation page.[14]

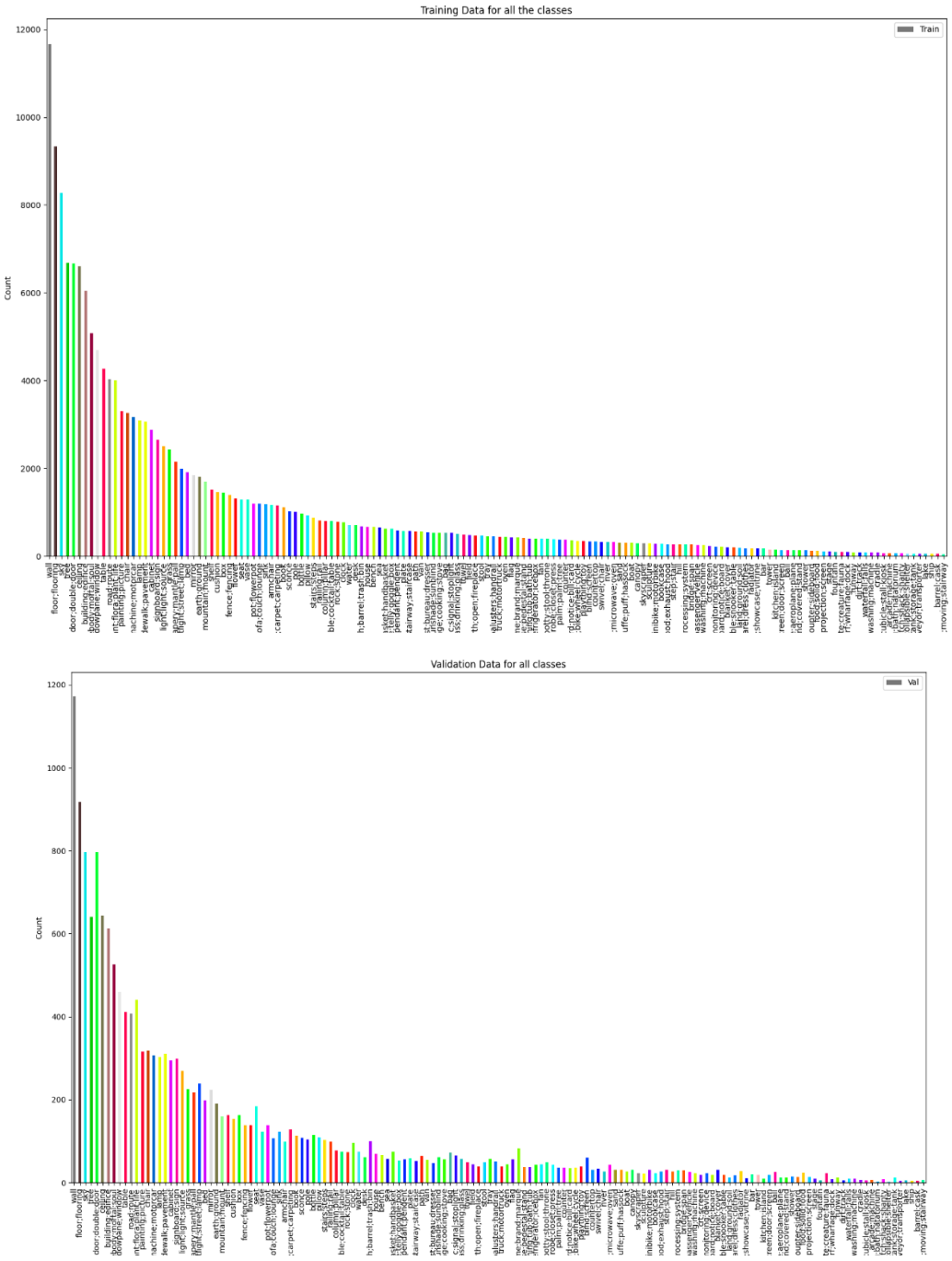


Figure 3: Training and Validation data for 150 classes.

The dataset which we are considering consists of a more detailed file structure, such as a list of attributes, objects, parts of objects, and masks. The mask is defined by polygons. This file is derived from [15] and it serves as the cornerstone of the initial data discovery. All the images, together with their annotations, are kept in the dedicated folder_name and thus lookups are performed by the index_ade20k.pkl file. Upon entering the respective folder_name, a given image, denoted as image_name (e.g., ADE_train_00016566.jpg), is accompanied by the following components: upon entering the respective folder_name, a given image, denoted as image_name is accompanied by the following components:

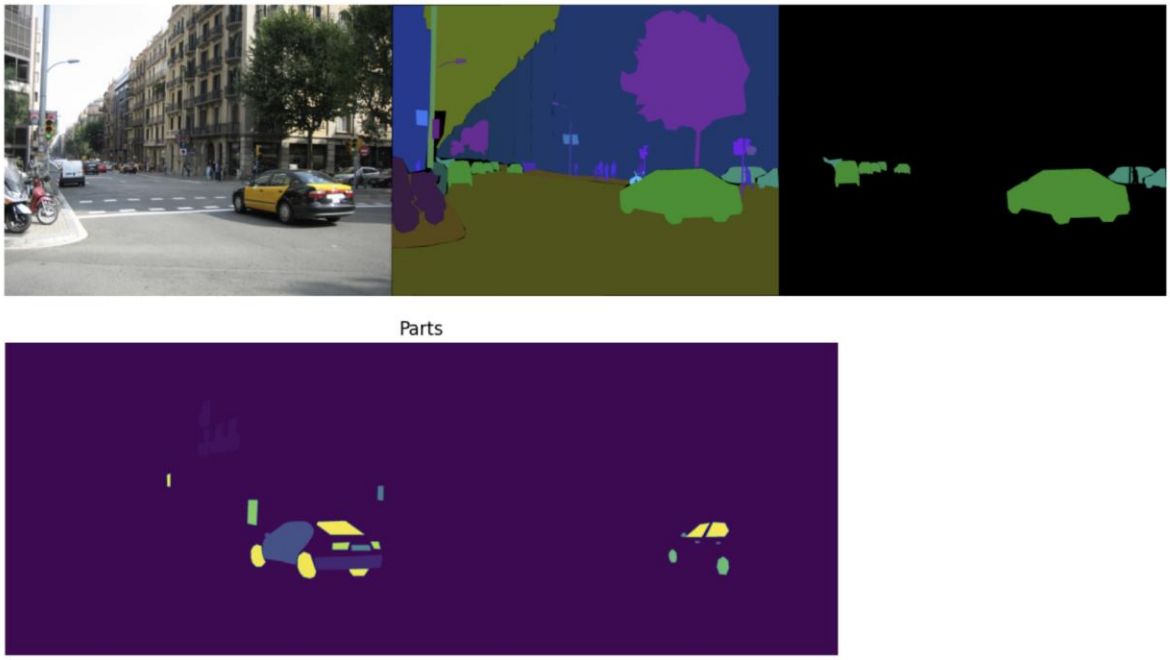


Figure 4: Example of class masks, objects and object parts in the image.

- image_name.jpg: This file includes the original image card however, there are covered license plates and blurred facial-recognition features (for example: ADE_train_00016566.jpg).
- image_name_seg.png: An example pixel-wise annotation file demonstrating object and instance (e.g. ADE_train_00016566_seg.png). The RGB channels express class information as well as any derived instances from it. Check out utils/utils_ade20k.py for a close look at how such files might be understood.
- image_name_parts_{i}.png: The detection of the object parts can be carried out at a level as the file named ADE_train_00016566_parts_1.png.
- image_name: A folder that contains all occurrences for this image, successfully saved as PNGs with mask bits set to ones, showing blocked objects (e.g., ADE_train_00016566.jpg).
- image_name.json: JSON file should be presented as the data source which has been responsible for keeping track of important information like the annotation time, annotated polygons, attribute annotations, etc. (e.g., ADE_train_00016566.json).

Figure 5: Polygons in images.

Through such a well-defined approach in data organization, the required image data and their annotations can easily be retrieved and used; they serve later for model training, analysis, or other purposes. The dataset contains 20000+ images, for model training purposes, I have chosen around 2000 images for YOLO and U-Net model training. Figure 4 shows the images, annotation mask, Class masks and Parts in the image using index_ade20k.pkl file and plots the polygons using Matplotlib. Similarly, Figure 5 shows the polygon of the segmentation mask on images.

3.2. Data Cleaning and pre-processing

The dataset contains 150 categories and initially, I trained all the models for all the 150 classes, but the models faced issues learning different boundaries and textures from the images and distinguishing 150 different classes thus for the optimal and accurate learning of models I chose a total of 14 classes. Figure 7 shows the masks generated for test images showing colour masks for the U-Net Model having 79% training accuracy. The masks shown in the images depict that the model isn't stable and cannot distinguish different classes, let alone the masks. All the other classes were considered as background classes thus class values were set to 0 and colour was set to (0,0,0).

Initially, the dataset had 150 distinct categories, and models were trained across all these categories. However, encountered challenges as the models struggled to effectively learn diverse boundaries, and textures associated with a vast array of 150 classes. The provided Figure 6 showcases test images, colour-coded training masks and masks generated after training with the U-Net model having a training accuracy of 79%. However, upon inspection of the masks depicted in the images, it becomes evident that the model's performance is unstable. It exhibits difficulty in distinguishing between different classes, let alone accurately delineating the masks.

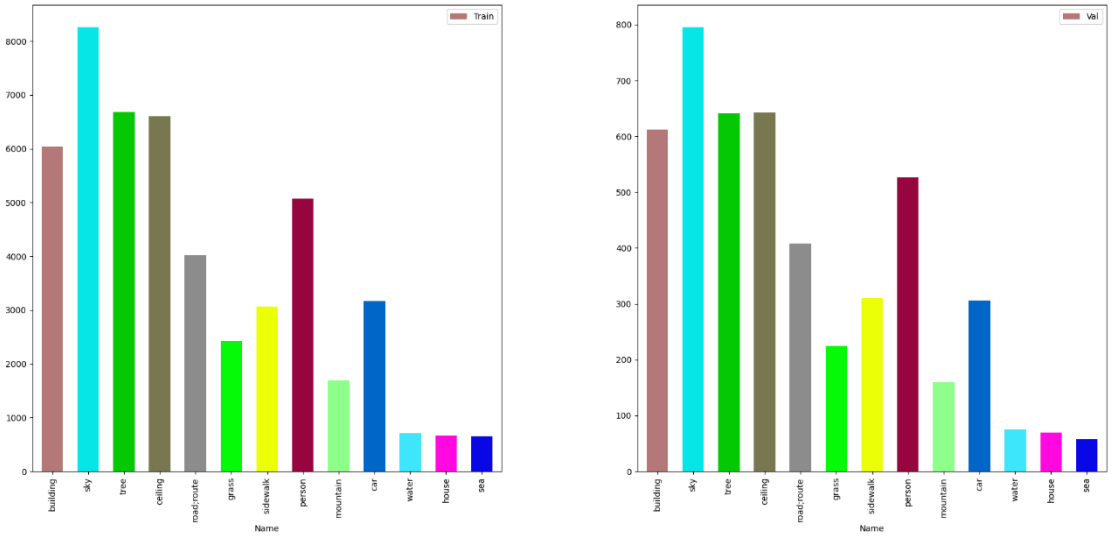


Figure 6: Training and Validation dataset for selected 14 classes.

Consequently, for optimal and precise model learning, a strategic decision was made to focus on a subset of 14 classes. To streamline the learning process, all other classes were designated as background classes, with class values set to 0 and their corresponding colour representation set to (0,0,0). This approach also helped to eliminate the images that don't contain relevant classes or images with background pixel percentages of 90% or more.

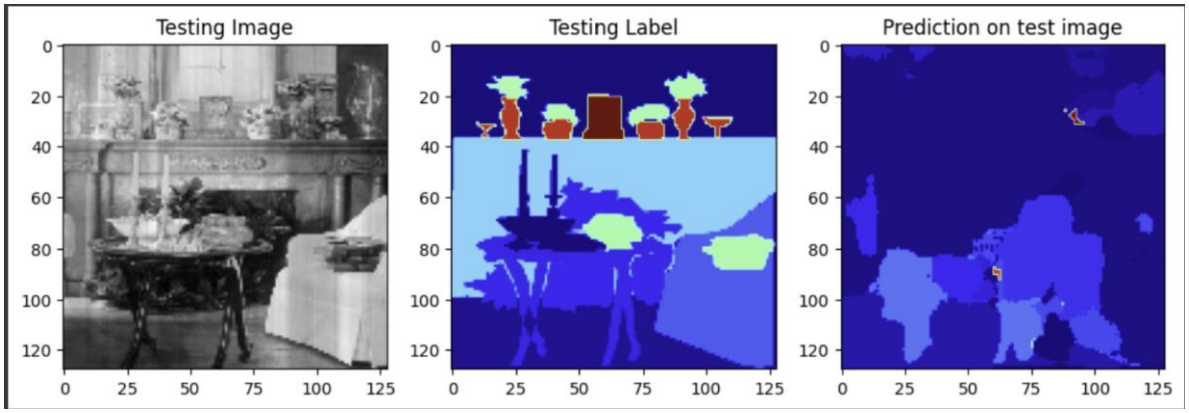


Figure 7: Training model for all 150 classes.

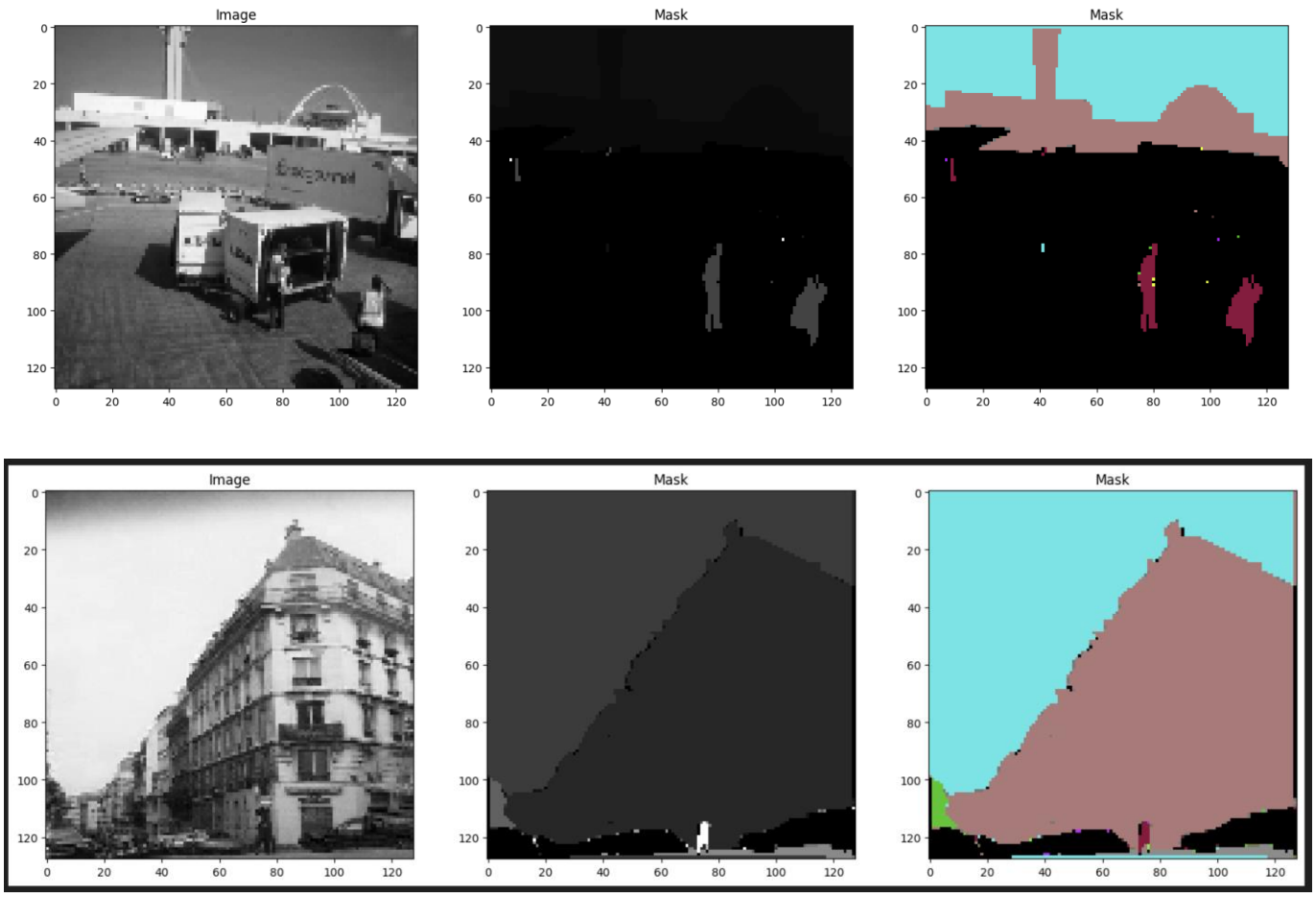
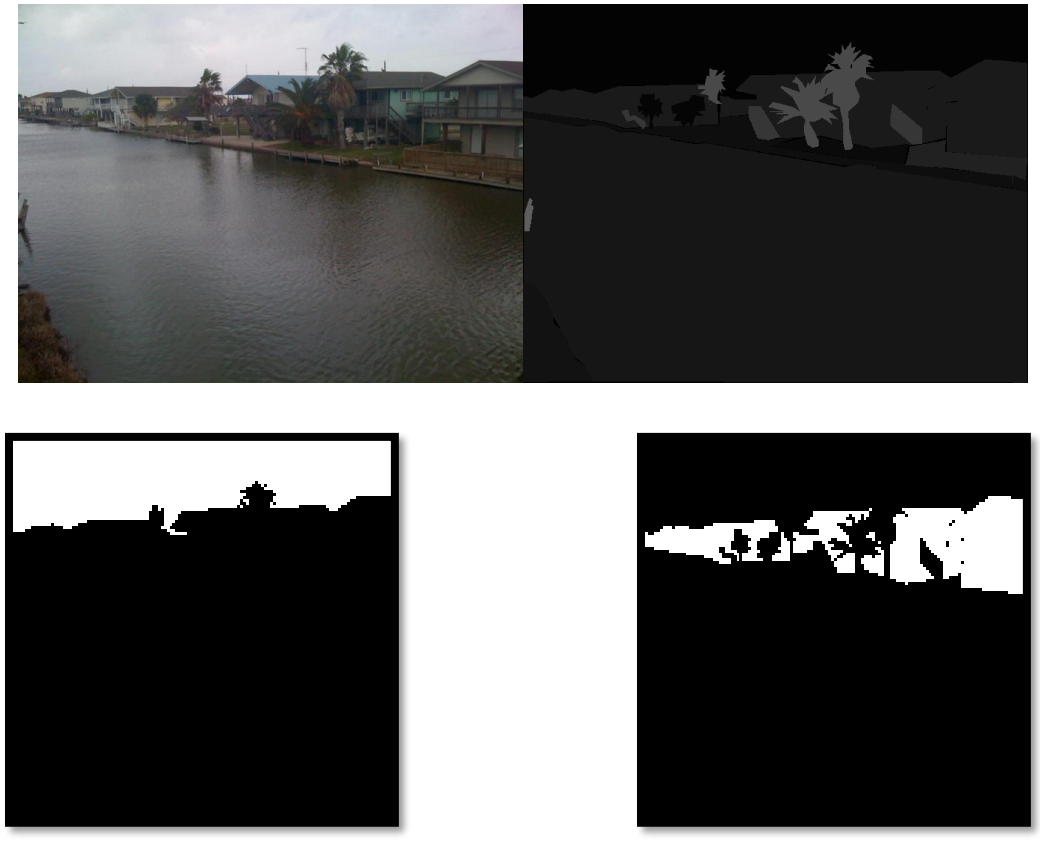


Figure 8: Images with its grayscale mask and coloured Maks with a black background.

When using different neural network architectures, input data preprocessing is customized to guarantee the best possible compatibility with the neural network that is being used. For this research, I collected photos with grayscale annotations and used a training-specific algorithm to process them. Figure 8 depicts examples of grayscale masks and coloured class masks used for training the U-Net model. All the images were resized to (128,128) and used with U-Net for predicting 14 distinct classes.

Grayscale binary masks were specifically used as the input data for the YOLO architecture, and a precisely constructed annotation file using the COCO JSON Format was produced. This file was created to comply with the specifications of the YOLO model and support grayscale masks that correspond to different classes in the database.

For the selected categories, label encoding is performed before splitting the data to represent the class labels in Numerical representation. Label encoding provides a straightforward way to represent categorical data as numbers. All the images are converted to NumPy arrays and then normalized to ensure model stability, improved convergence and model performance.



(a) Binary Class Mask for image ADE_train_00005005.jpg

a) Binary Class Mask for image ADE_train_00005032.jpg

Figure 9: Grayscale annotation images.

Figure 9 a) and b) give examples of binary class masks used for the YOLO model's annotation file. Figure 10 shows the polygon plot using an annotated file to validate the annotation JSON file.

Figure 10: Plotting the images from the YOLO Annotation file.

4. Methodology and Results

In the project, two distinct neural network architectures, namely U-Net and YOLO, are trained on the ADE20K dataset's subset. However, for other models, pre-trained models along with their weights are utilized. The training process was completed using Google Collab and Kaggle's GPU resources, ensuring efficient computation and model training. A subset of the dataset comprising 2000 randomly selected images are used for training the U-Net and YOLO models. Each model is evaluated based on various metrics, and their respective performances are thoroughly discussed within their dedicated sections in the project report. The decision to train U-Net and YOLO models on a subset of the dataset was likely made to manage computational resources while still obtaining meaningful insights into their performance.

The research attempted to utilize pre-existing information and weights to accelerate the training process and enhance the overall performance of the model by utilizing pre-trained models for different architectures.

4.1. Front-end of GUI Application

For developing the front-end application, I have used React JS. React JS is a great choice as the goal was to develop interactive and dynamic applications. React JS is A JavaScript-based library, prominently used to build user interfaces with component-based architecture facilitating modular and reusable code. The initial step is to install all the dependent libraries and Create React App (npx create-react-app my-app) . I developed a component to upload an image, display the processed images, and a component for all the buttons that can route to respective APIs. Upon creating all the required components, implement event handlers and callbacks to handle user interactions within your components.

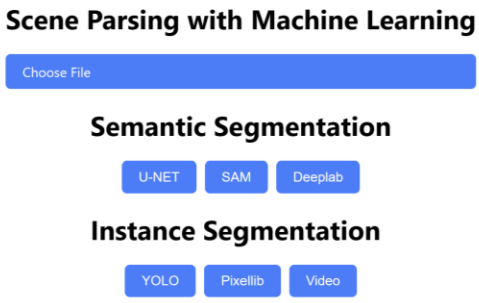


Figure 11: Home page.

Upon uploading the image, the user will get an option to select the operation to perform on the image. Once the image is uploaded the image element is displayed on the screen with the processed image adjacent to it. Figure 12 shows the placement of the two images on the webpage. The front-end application processes the uploaded file and sends or receives the file in the form of an Array Buffer which stores the uploaded files and sends the object to the backend.

4.2. Backend using Flask.

Creating a Flask backend to serve different APIs is a powerful way to handle various functionalities. Flask is a Python-based Micro-framework that is lightweight and well-suited for web applications and APIs. Developing APIs in Flask involves creating route handles that can process HTTP requests and respond with appropriate responses. To develop the backend, the initial step is to create a virtual environment and add all the dependent libraries. For this project, I have installed libraries like python-OpenCV, pandas, matplotlib, pixellib and other image segmentation-related libraries. All the required modules are imported at the beginning of the Python file and then create and initiate a Flask app. The next step is to define route handlers to create API endpoints. Each route corresponds to a specific URL pattern and HTTP method (e.g., GET, POST). These API handlers can handle different HTTP methods (GET, POST, PUT, DELETE, etc.) using the methods parameter in the route decorator.

API calls can access request parameters within route handlers using Flask's request object. Before using the APIs, I loaded all the weights and model architectures.

4.3. ResNet

PixelLib is a Python library[16], developed specifically for image and video segmentation operations. It provides an easier way to include these features in software projects. Its main characteristic is that it can perform segmentation with a few lines of code without training the model can be easily used by people who do not have expertise in using Machine learning models. Its straightforward API and minimalistic syntax enable developers to incorporate segmentation functionalities seamlessly. Moreover, it offers various parameters to customize segmentation tasks according to specific requirements.

PyTorch is a well-known deep-learning framework that PixelLib uses as its backend[17]. Faster and more precise object segmentation results from this integration's efficient segmentation model computation and optimization. PixelLib also makes use of the PointRend segmentation architecture, which presents a revolutionary semantic segmentation method. PointRend is a great option for real-time computer vision applications because of its remarkable segmentation accuracy, which is achieved by concentrating on using fewer computing resources.

When real-time image segmentation is considered, Mask RCNN and Point Red architectures are indeed two prominent architectures. Mask-RCNN is proven to be effective for image segmentation tasks, providing a proper balance between Accuracy and speed. However, in a real-time environment, it faces challenges with inference speed. On the other hand, PointRed provides more accurate, and faster results. Point Red is specifically designed to keep dynamic, constantly changing environmental conditions. It focuses on capturing fine-grained details and output boundaries. Thus, it becomes the first choice over Mask RCNN when considering real-time applications.

Pixellib provides comprehensive segmentation results, including object coordinates, class IDs, class Names, object counts, scores, and extracted objects that could be used for advanced analysis, object

manipulation, and further exploration of the segmented data. Figure 11 shows the results generated using the Pixellib library using ResNet101 and its weights. ResNet101 is a deep convolutional residual network that is designed to handle the vanishing gradients problem. The Reset architecture enables the network to learn multiple layers of features without getting stuck in local minima.

To keep the gradients from disappearing and to preserve the information flow across the network, ResNet adds residual connections to the network. The residual link serves as a workaround that enables data to get straight to the output by eschewing one or more network levels[18]. The network can learn the residual function and make minor parameter adjustments according to the residual connection, which speeds up convergence and improves performance. This helps the network converge more quickly and perform better by allowing it to learn residual functions. The premise behind the residual connection is that it is simpler to learn the simpler mapping between the inputs and the outputs rather than attempting to learn the intricate one. The figure shows the results generated using Pixellib with ResNet101. The resNet architecture was evaluated on the ImageNet2021 ImageNet classification dataset having 1.28 million images and 1000 classes.

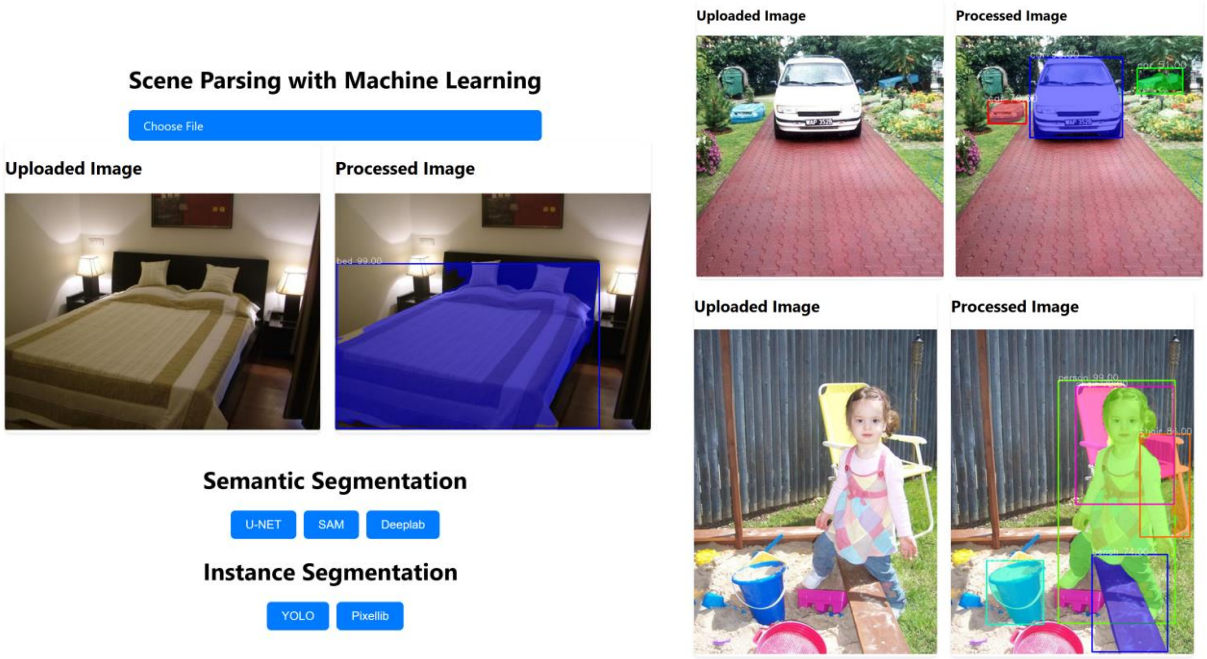


Figure 12: Semantic Segmentation using Pixellib library using Resnet101 weights

4.4. Deep Lab – Xception

The Xception network serves as the foundation for the Deep Lab model, which is an expansion of the DeepLabV3+ design [19]. Deeplabv3 is fully convolutional Neural Network model designed by Google researchers to improve semantic segmentation challenges. Its is an incremental update on its previous versions. For enhanced semantic segmentation performance, it combines the benefits of Deep Lab's atrous spatial pyramid pooling (ASPP) module with the depth-wise separable convolutions of the

Xception architecture**.** It uses ResNet models, trained on ImageNet as the backbone with ASPP.

One common strategy for building a segmentation model involves utilizing a pre-trained model trained on benchmark classification datasets like ImageNet. These models are designed to classify a wide range of objects, often up to 1000 classes, requiring them to learn generalizable features. They undergo extensive experimentation during development before being made publicly available. Consequently, pre-trained models are widely adopted for various downstream tasks such as object detection, segmentation, and more.

The intrinsic nature of deep convolutional neural networks (CNNs) is characterized by a reduction in feature resolution as the network progresses through consecutive pooling or stridden convolution layers, which aids in down sampling[20]. This mechanism is crucial for tasks like image classification, as it mitigates memory consumption while maintaining transformation invariance.

However, this down-sampling behaviour poses challenges in tasks such as segmentation, where spatial information holds paramount importance. Deeper layers within the network excel in encoding "what is in the image," yet they cannot retain precise information regarding "where in the image" a particular feature or object resides.

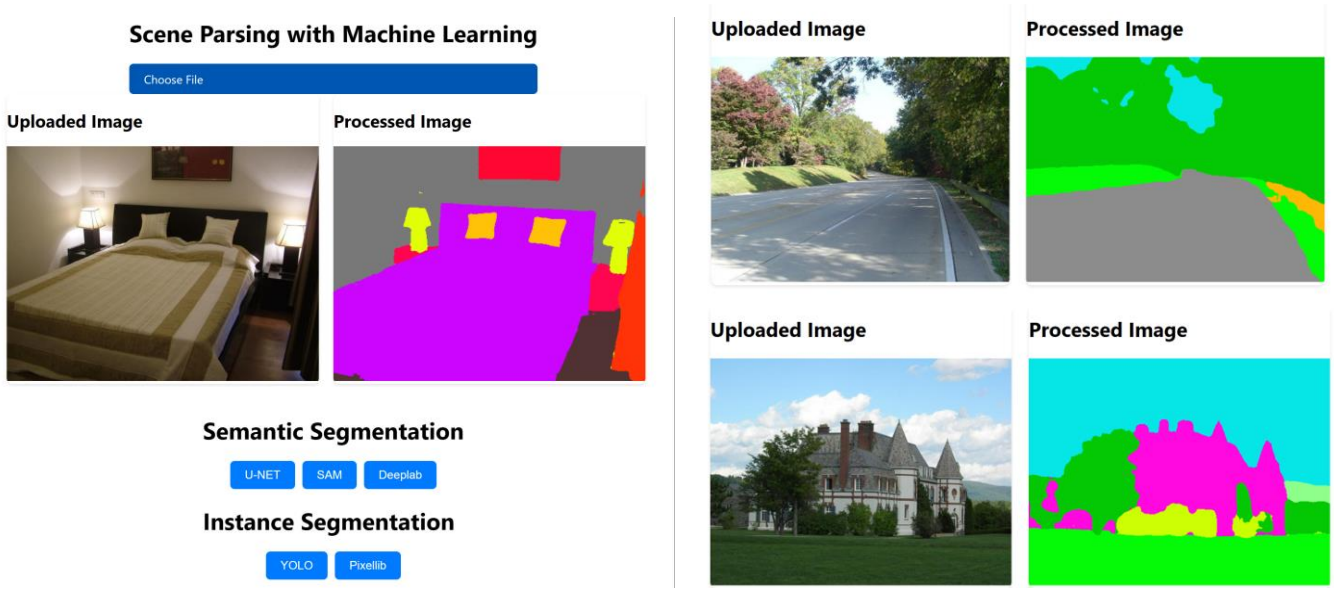


Figure 13: Semantic Segmentation using Deep Lab with Xception model.

Figure 13 shows the results generated for the input image using the Pixellib library and Deeplab Xception's pre-trained weights.

Furthermore, another significant challenge arises from the existence of objects at varying scales within an image. For instance, the "person" class may manifest in multiple resolutions within the same image. As the network delves deeper, its receptive field expands, enhancing its capability to detect larger objects effectively. Nevertheless, this improvement comes at the cost of potentially overlooking smaller-sized objects, thereby impacting segmentation accuracy.

To address the initial challenge, Deeplabv3 integrates "Atrous Convolution," coupled with adjustments to the backbone's pooling and convolutional striding elements. This approach allows the model to preserve spatial information effectively while navigating through deeper layers of the network. To tackle the second issue, Deeplabv3 employs a refined version of the "Atrous Spatial Pyramid Pooling" module. This module is specifically designed for multi-scale feature extraction, enabling the model to capture and analyze objects present at varying scales within an image. By leveraging this modified module, Deeplabv3 enhances its ability to accurately segment objects of different sizes and resolutions, contributing significantly to its overall performance in scene parsing tasks.

4.5. U-Net

Olaf Ronneberger and his team developed U-Net in 2015 for their work on biomedical images [21]. The U-Net architecture (as shown in Figure 14) derives its name from its distinctive "U" shape, which comprises convolutional layers and two interconnected networks: the encoder and the decoder. This architecture effectively addresses the fundamental questions in segmentation tasks: determining "what" objects are present in an image and precisely "where" these objects are located.

The encoder network, also known as the contracting network[22], plays a pivotal role in learning feature representations of the input image, thus addressing the first question of identifying objects. This process resembles traditional classification tasks performed using convolutional neural networks (CNNs), albeit with a crucial distinction: the absence of fully connected layers at the end. Instead, U-Net outputs a mask of the same size as the input image, facilitating precise segmentation.

The encoder network comprises four encoder blocks, each consisting of two convolutional layers with a 3x3 kernel size and valid padding, followed by a Rectified Linear Unit (ReLU) activation function. Subsequently, the output is fed into a max-pooling layer with a 2x2 kernel size, effectively reducing spatial dimensions and computational overhead.

The bottleneck layer, situated between the encoder and decoder networks, consists of two convolutional layers followed by a ReLU activation function, culminating in the final feature map representation.

What distinguishes U-Net's efficacy in image segmentation are the skip connections and the decoder network. These components deviate from conventional CNN architectures and significantly enhance segmentation accuracy.

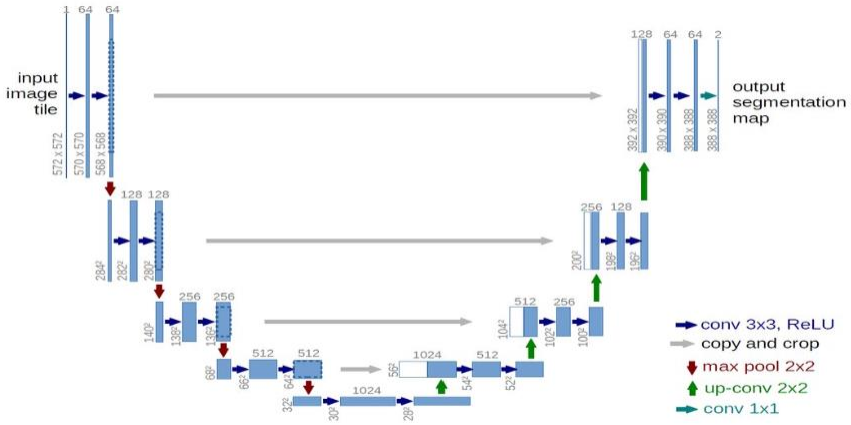


Figure 14: U-Net Architecture [21]

The decoder network, also termed the expansive network, aims to up-sample feature maps to match the input image's size. It achieves this by utilizing skip connections, which pass contextual feature information from encoder blocks to the decoder. The decoder network resolves the "where" aspect by generating a segmentation mask based on the learned feature representations.

Skip connections, denoted by grey arrows in the model architecture, harness contextual information from encoder blocks to refine the segmentation map. This interplay between high-resolution features and localization information enables U-Net to capture both detailed feature information and precise object localization.

Ultimately, a 1x1 convolutional layer with sigmoid activation produces the final segmentation mask, offering pixel-wise classification. This bidirectional flow of information from the contracting path to the expansive path ensures comprehensive feature understanding and accurate localization, making U-Net a robust solution for image segmentation tasks.

Figure 15 shows the results generated for the input image using U-Net architecture and training it on the ADE20K dataset. The U-Net model is trained using the Adam optimizer, which is known for its efficiency in optimizing deep learning models by adapting the learning rate for each parameter. The categorical cross-entropy loss function is employed to measure the difference between predicted and actual pixel values in multi-class segmentation tasks. Additionally, evaluation metrics such as accuracy, mean Intersection over Union (mean_iou), recall, and F1-score are utilized to assess the model's performance comprehensively. These metrics provide insights into different aspects of segmentation quality, including pixel-level accuracy, class-wise IoU, and overall precision and recall rates.

Moreover, the U-Net model implements a dynamic learning rate strategy where the learning rate can increase if the model reaches a plateau during training. This adaptive learning rate scheme helps the model overcome stagnation and explore different areas of the optimization landscape for better convergence. Furthermore, the implementation includes node dropout, a regularization technique that

randomly deactivates nodes during training to prevent overfitting and enhance model generalization. Additionally, model checkpointing is incorporated to save the best weights based on validation accuracy, ensuring that the model retains its best-performing configuration for deployment and further evaluation.

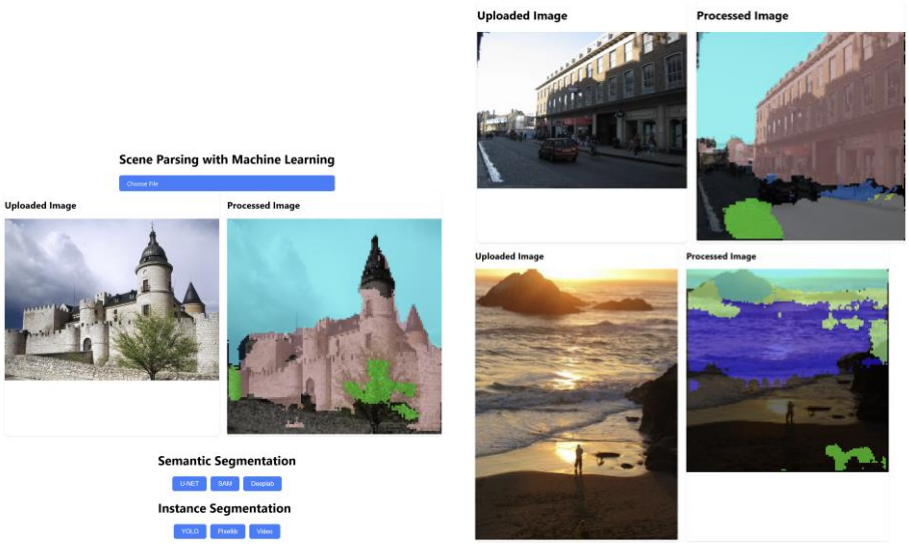


Figure 15: Semantic Segmentation using U-NET

4.6. YOLO

In 2015, the You Only Look Once (YOLO) algorithm made a significant impact in the realm of object detection, as introduced in a seminal research paper authored by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. YOLO revolutionized object detection by offering a single-shot approach, where a single neural network predicts bounding boxes and class probabilities in one pass, utilizing the entire image as input.

The latest iteration, YOLOv8, builds upon this foundation by allowing users to harness the power of pre-trained models, specifically trained on extensive datasets such as COCO (Common Objects in Context). This integration enables YOLOv8 to perform image segmentation, providing detailed pixel-level information about objects within an image, thereby enhancing analysis and comprehension of image content.

Despite the inherent computational cost associated with image segmentation, YOLOv8 seamlessly incorporates this technique into its neural network architecture, ensuring efficient and accurate object segmentation. The operational mechanism of YOLOv8 begins by dividing the input image into grid cells as shown in Figure 16. Leveraging these grid cells, YOLOv8 predicts bounding boxes (bbox) alongside class probabilities for detected objects.

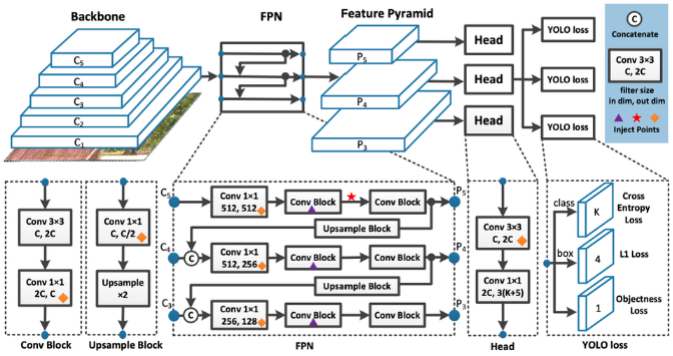


Figure 16: Architecture of YOLO

Furthermore, YOLOv8 employs the Non-Maximum Suppression (NMS) algorithm to address overlapping bounding boxes. For example, in scenarios where multiple instances of the same object (e.g., cars) are present, resulting in overlapping bounding boxes, the NMS algorithm efficiently reduces this overlap, leading to cleaner and more precise detections.

YOLOv8 is offered in three distinct variants: YOLOv8, YOLOv8-L, and YOLOv8-X, with varying backbone network sizes. The primary discrepancy among these variants lies in the size of the backbone network, with YOLOv8 featuring the smallest backbone network and YOLOv8-X boasting the largest. This variation in backbone network size provides users with flexibility, allowing them to choose a model variant that best suits their specific requirements and computational resources.

To train the YOLOv8 model, data.yaml file which contains the names of classes, training, validation, and testing data directories is used. This file is generated using the COCO style annotation labels converted to YOLO annotations for the dataset under consideration. The input images are processed and the instance segmentation of the image with bbox, masks and class probabilities are returned. The model uses various evaluation metrics such as mean average precision, precision, recall and loss.

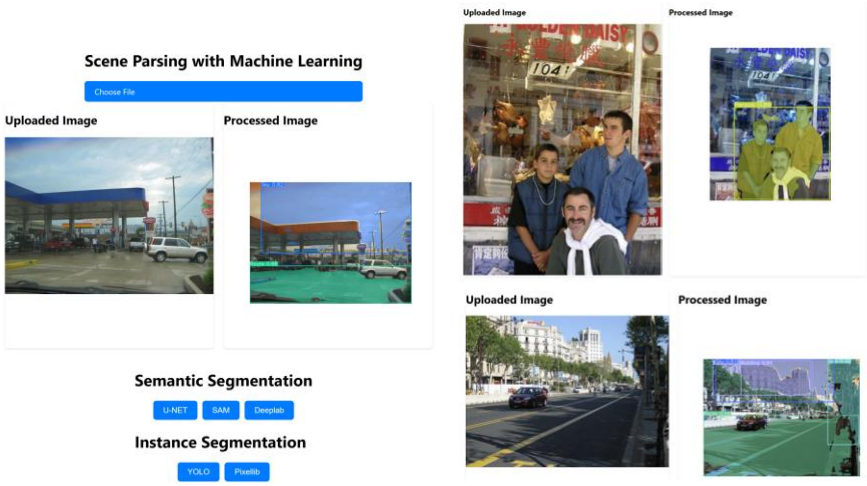


Figure 17: Instance Segmentation using YOLO.

Figure 17 shows the results generated for the input image using Yolo architecture and training it on the ADE20K dataset.

4.7. SAM

The Segment Anything Model (SAM)[24] represents a groundbreaking advancement in image segmentation technology, introducing promotable segmentation capabilities alongside real-time performance, thus establishing new benchmarks within the domain. SAM, at the core of the Segment Anything initiative, signifies a paradigm shift in image analysis by its innovative model, task framework, and dataset.

SAM's design is characterized by its ability to conduct promptable segmentation, enabling the generation of precise segmentation masks based on various prompts, including spatial or textual cues that delineate objects within images. This feature grants SAM unmatched versatility in addressing diverse image analysis tasks.Central to SAM's capabilities is its adaptability to novel image distributions and tasks without prior knowledge, known as zero-shot transfer learning. Leveraging the extensive SA-1B dataset, comprising over 1 billion masks distributed across 11 million meticulously curated images, SAM [25] demonstrates exceptional zero-shot performance, surpassing traditional fully supervised methods in numerous scenarios.

The architectural underpinnings of SAM include a robust image encoder, a prompt encoder, and an efficient mask decoder. This unique design fosters flexible prompting mechanisms, real-time mask generation, and inherent ambiguity handling, thereby enhancing SAM's efficacy in segmentation tasks.The SA-1B dataset, introduced alongside the Segment Anything project, represents a significant contribution, being the largest segmentation dataset to date. With its vast collection of masks and images, the dataset provides SAM with a rich and diverse training resource, bolstering its generalization capabilities.Figure 18 shows the results generated for the input image using SAM. Its prowess in zero-shot performance across varied segmentation tasks positions it as a readily deployable tool with minimal prompt engineering requirements, making it invaluable for a wide array of applications within the image analysis domain.

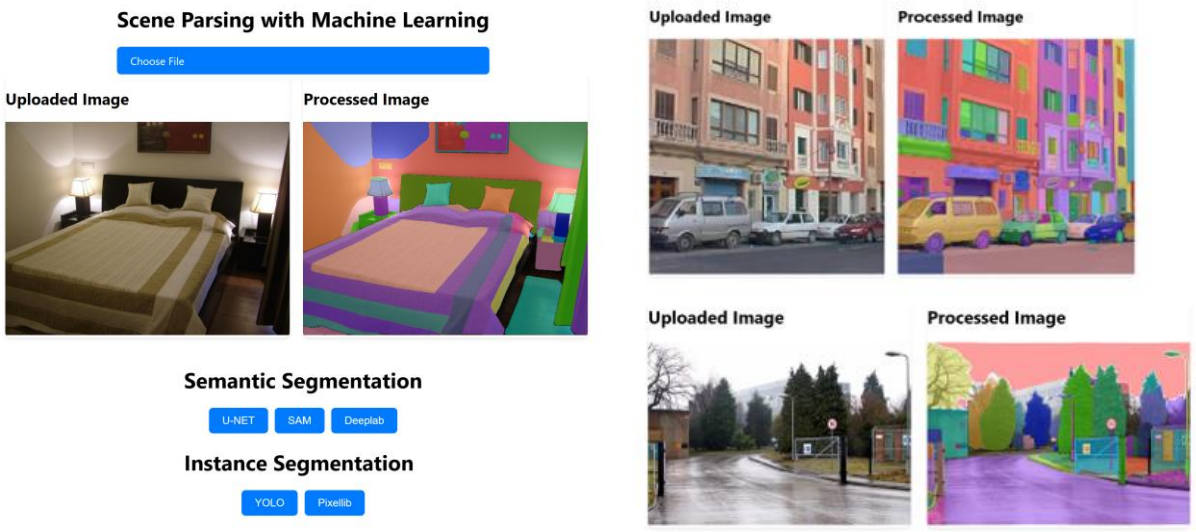


Figure 18: Semantic Segmentation using SAM.

5. Outcomes and Evaluation

The evaluation metrics used to interpret model performance are training and test dataset loss, precision, recall, mean Average Precision mAP, mAP50, and mAP-95.

Accuracy of the model is a common evaluation metric that measures the ratio of correctly predicted instances to the total number of Instances in the Dataset.

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Negative + False\ Negative + \ True\ Positive + False\ Positive}$$

$$Precision = \frac{TruePositive}{True\ Positive + False\ Positive} \quad Recall\ = \frac{TruePositive}{True\ Positive + False\ Negative}$$

$$F1 - Score = \frac{2*Precision}{Precision+Recall} \quad IoU = \frac{(Object \cap Detected\ Box)}{(Object \cup Detected\ Box)} \quad mAP = \frac{1}{N}(\sum APi)\ for\ \ i = 1\ to\ i = N$$

Equation 1: Evaluation Metrics

The loss is a measure of the model's error during the training and evaluation phase. It measures the extent to which the model's prediction matches with actual ground prediction. Precision is a measure of percentages of true positives among all the positive predictions made by the model. Similarly, Recall or sensitivity is measured as the proportion of true positive rate, which measures the proportion of true positive predictions among all actual positive instances in the dataset.

The F1 score is a metric that combines precision and recall into a single value. A high F1 score indicates a balance between precision and recall, meaning the model is good at both minimizing false positives and false negatives. Intersection over union (IoU) is a metric commonly used in object detection tasks to measure the overlap between predicted bounding boxes and ground truth boxes.

Mean Average Precision calculates the average precision (AP) for each class and then computes the mean of these AP values across all classes. AP measures the precision-recall curve's area, indicating how well the model ranks and detects objects across different confidence thresholds. A map50 calculates the mean Average Precision with an IoU (Intersection over Union) threshold of 0.5. Similarly, mAP50-95 measures model performance with a threshold of 0.9.

The U-Net model is specifically designed for semantic segmentation tasks making it well-suited for tasks like medical image segmentation where pixel-level accuracy is crucial. Its U-shaped architecture facilitates focusing on intricate patterns in data while preserving spatial information. But when it comes to memory or resource allocation, especially for large-scale datasets, it demands a significant number of computational resources. U-Net models can be very sensitive to hyperparameters to achieve optimal performance. While training the U-Net model, I faced multiple issues related to the exhaustion of resources and making the whole training process time-consuming.

.

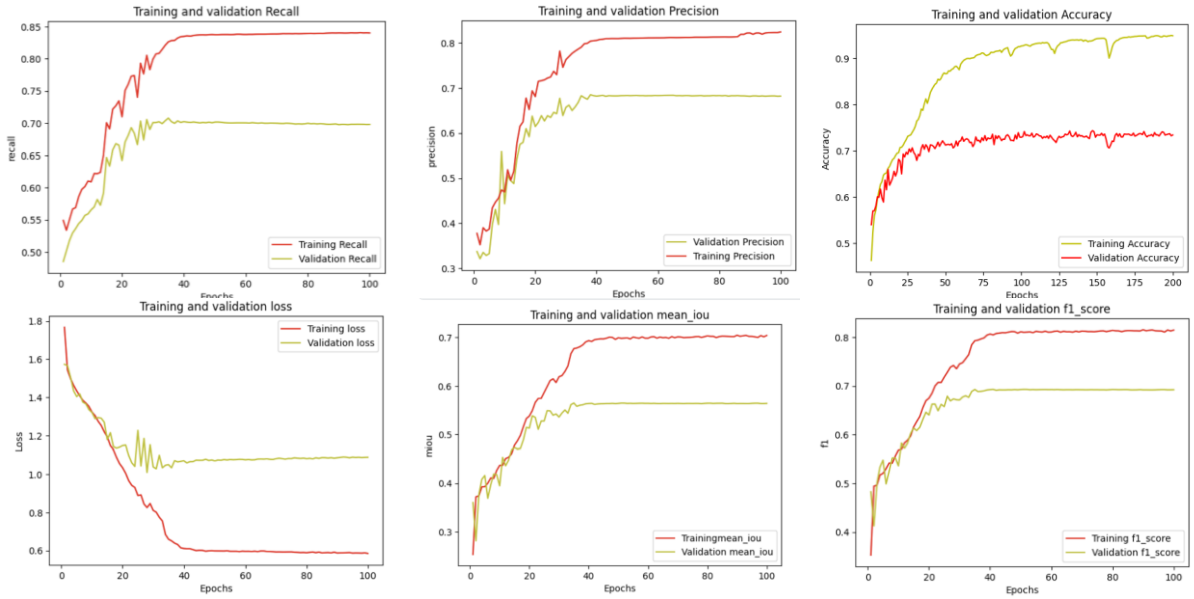


Figure 19: Evaluation Metrics Graphs generated on training U-Net for 200 Epochs.

Overall, the model is performing well on training data and can satisfactorily segment the unseen images. However, if you try to understand the reliability of the model, some limitations become apparent. Some results generation for the segmentation of distinct images are shown in Figure 20.

Figure 20: Results generated for UNET.

Figure 21 shows the graphs generated while training YOLO on the ADE20K dataset. The YOLO model is trained for 200 epochs using Google Colab Pro.

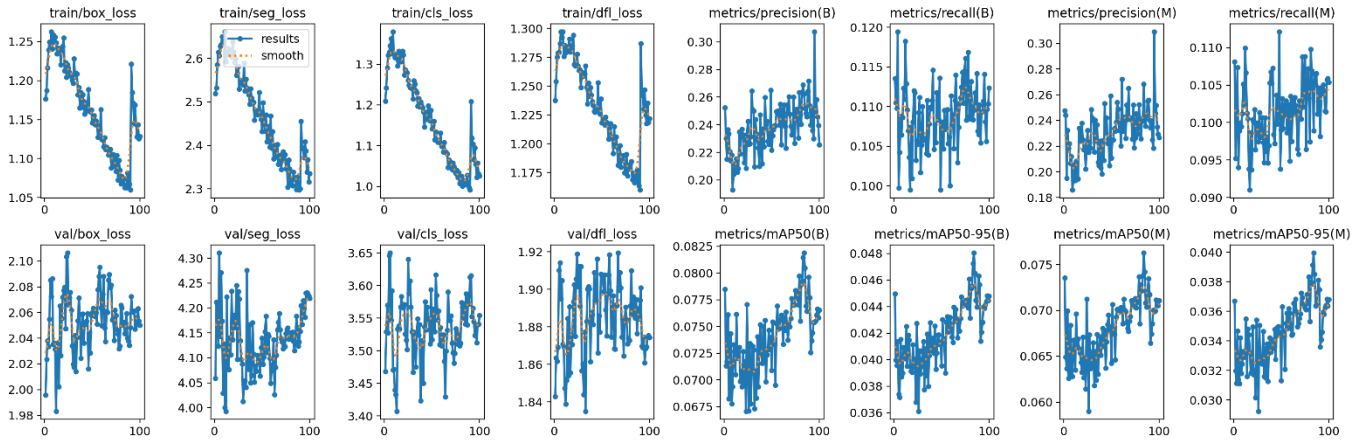


Figure 21: Evaluation Metrics graphs for YOLO for 200 epochs

YOLO models are single-stage detection approaches that allow faster interference compared to two-stage methods. This efficiency can be effectively used in real-time applications. Yolo model predicts bounding boxes and class probabilities directly. These models tend to have high localization accuracy even in the presence of class imbalance. If you look at the data preprocessing section, the data considered here is not balanced. The class imbalance can affect the training process when it comes to model convergence and performance. The model can face difficulties in learning minority classes leading to biased prediction and overall low performance. Minority classes may have lower accuracy. For class imbalance, the model may prioritize the majority class and exhibit higher false negatives.

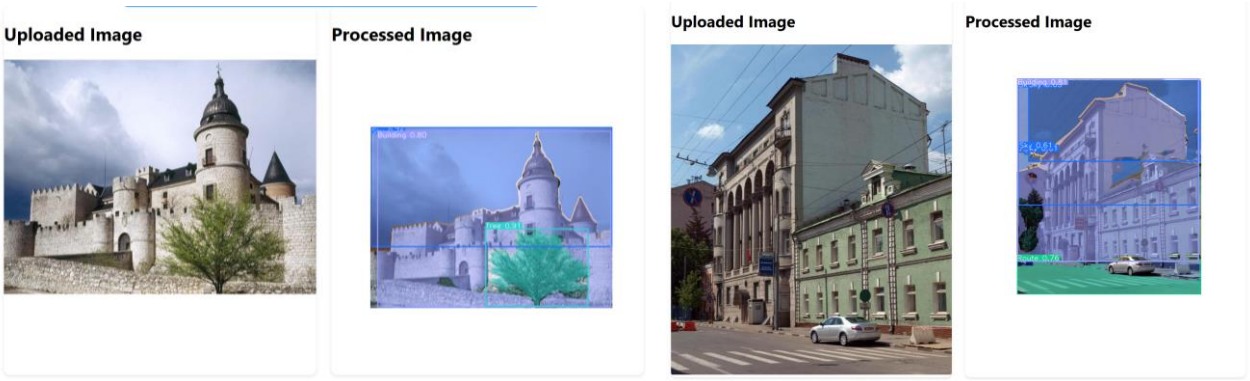


Figure 22: Results generated for YOLO.

Model training has been a resource-intensive and time-consuming process. For 200 epochs, the overall training time was 24 hours, which generated the results shown in Figure 22, at a glance, the results shown here are accurate but are showing results for a lower threshold, which is set to 60%. The precision, and recall values are around 0.3 and 0.12 respectively which are very low. Moreover, the model stability is less, but to improve these results, there is a need for strong highly efficient GPUs.

The ADE20k dataset is a challenging dataset commonly used for semantic segmentation tasks. Justifying lower mean Average Precision (mAP), recall, and precision values for the ADE20k dataset can be attributed to several factors inherent to the dataset and the nature of semantic segmentation tasks.

The ADE20k dataset contains a wide variety of indoor and outdoor scenes with complex structures, textures, and object classes. This diversity can make it challenging for a model to accurately segment objects and scenes with high precision and recall. Class Imbalance is another issue associated with semantic segmentation datasets. ADE20K includes fine-grained object boundaries and intricate object shapes, requiring precise localization and detailed segmentation. Models may face difficulties in achieving high recall and precision for objects with complex shapes or structures. Ambiguities in object semantics, varying object scales, occlusions, and scene clutter can contribute to lower mAP, recall, and precision values as the model grapples with these complexities. Training effective models for semantic segmentation on datasets like ADE20k requires sophisticated architectures, extensive data augmentation, fine-tuning, and careful hyperparameter tuning. Even with state-of-the-art models, achieving high recall and precision across all classes in the ADE20k dataset remains a challenging task.

Table 4 discusses the advantages and disadvantages of ResNet using PixelLib, Xception using DeepLab,

YOLO, UNet, and SAM models for the project. This table provides a concise overview of the strengths and weaknesses of each model, aiding in the decision-making process for selecting the most appropriate model based on project requirements, computational resources, task complexity, and desired outcomes.

Model	Pros	Cons
ResNet using Pixellib	Well-established, widely used backbone structure. Pixelib simplifies training and inference	May require additional tuning for specific tasks.
Xception using Deeplab	Deep learning network with Atrus CNN layer for dense predictions, efficient use of GPU Memory.	Complexity in training and configuration Resource-intensive for large data structures
YOLO	Single-stage generator with high inference speed Capable of detecting multiple classes	Limited to bounding box prediction with small object detection
U-Net	U-shaped Architecture helps to understand detailed features	Memory-intensive for high-resolution images
SAM	Incorporates Spatial attention Mechanism for better feature extraction, effective for complex scene understanding	Complexity in implementation Requires careful Hyper Parameter tuning

Table 4: Comparison of Models.

Parameter	ResNet (PixelLib)	Xception (DeepLab)	YOLO	U-Net	SAM
Architecture Complexity	Relatively straightforward architecture with residual blocks.	DeepLab uses a modified Xception backbone with atrous convolutions, resulting in a more complex architecture.	Compact architecture with convolutional layers and detection heads.	Symmetric architecture with encoder and decoder paths designed for image segmentation.	May have a more complex architecture due to the incorporation of spatial attention mechanisms.
Training Speed	Efficient training speed depending on dataset size and hardware resources.	Training can be slower due to the complexity of the Xception backbone and atrous convolutions.	Known for fast training due to its single-stage detection approach.	Training speed can vary based on image resolution and model depth.	Training speed can be moderate depending on the model architecture and attention mechanism.
Inference Speed	Typically, fast inference speed,	Inference speed can be slower	Known for real-time inference	Inference speed can vary but may	Inference speed can be moderate,

	especially with optimized implementations.	due to the complexity of the architecture and dense predictions.	speed, suitable for fast detection tasks.	be slower compared to YOLO for object detection tasks.	similar to other semantic segmentation models.
Memory Usage	Moderate memory usage, depending on the model size and batch size.	Can be memory-intensive, especially with larger input sizes and batch sizes.	Efficient memory usage due to its single-stage architecture.	Can be memory-intensive, especially with high-resolution images and deep architectures.	Memory usage can vary based on model complexity and attention mechanism.
Model Size	Model size depends on the specific ResNet variant used.	DeepLab models can have larger file sizes due to the complexity of the Xception backbone.	Relatively compact model sizes compared to some other object detection architectures.	Model size can be larger, especially with deeper architectures and additional layers.	Model size may vary based on architecture complexity and attention mechanism parameters.
Task Suitability	Suitable for semantic segmentation tasks with moderate complexity.	Well-suited for semantic segmentation tasks requiring detailed feature extraction.	Ideal for real-time object detection tasks where speed is crucial.	Particularly effective for medical image segmentation and general image segmentation tasks.	Effective for tasks requiring spatial attention mechanisms and complex scene understanding.

Table 5: comparison of ResNet using PixelLib, Xception using DeepLab, YOLO, U-Net, and SAM model

This comparison Table 5 highlights the key differences among ResNet using PixelLib, Xception using DeepLab, YOLO, UNet, and SAM models across various parameters crucial for computer vision tasks. ResNet with PixelLib offers an efficient training speed and moderate memory usage, making it suitable for semantic segmentation tasks. Xception with DeepLab, while complex and memory-intensive, excels in detailed feature extraction for semantic segmentation. YOLO stands out for its real-time inference speed in object detection tasks. UNet is preferred for image segmentation, especially in medical imaging. SAM incorporates spatial attention mechanisms, making it effective for tasks requiring complex scene understanding.

6. Conclusion

This project successfully achieved its learning objectives, providing valuable insights into the world of scene parsing and the challenges it presents. A comprehensive understanding of scene parsing techniques was gained through an in-depth study of segmentation modules, algorithms, and critical analysis of model architectures (including resNet, U-Net, YOLO, DeepLab's Xception, and SAM). The project focused on training YOLO and U-Net models on the ADE20K dataset. While successful training was achieved, a significant challenge was the extensive computational resources required for fine-tuning model performance and optimization. Training of YOLO and U-Net models was conducted, aiming to minimize loss and achieve high accuracy in scene parsing tasks. However, resource limitations hindered extensive fine-tuning for optimal performance. While a user-friendly GUI application was not fully developed within this project, the groundwork was laid for future integration of the trained models. This application would accept user input and analyze images for object detection within scenes.

Further research and development efforts could address the resource limitations encountered. By utilizing more powerful computational resources, optimization and fine-tuning or hyperparameter tuning for the optimal model performance of the models could be conducted, potentially leading to better and more reliable performance on the models trained from scratch. Additionally, the development of a user-friendly GUI application would provide a valuable tool for testing and analyzing the models in a practical setting.

Overall, this project provided a strong foundation for understanding and implementing scene parsing techniques. The project outcomes offer a valuable starting point for further research and development in this exciting field.

7. References

[1] *Guo Z, Huang Y, Hu X, Wei H, Zhao B. A Survey on Deep Learning Based Approaches for Scene Understanding in Autonomous Driving*. Electronics. 2021; 10(4):471. https://doi.org/10.3390/electronics10040471

[2] *Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, 20–23 June 2014; pp. 580–587.* *1311.2524.pdf (arxiv.org)*

[3] *A guide to Two-stage Object Detection: R-CNN, FPN, Mask R-CNN : Source*

[4] *J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.*

[5] *Pyramid Scene Parsing Network: H. Zhao, J. Shi, X. Qi, X. Wang and J. Jia, "Pyramid Scene Parsing Network," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 6230-6239, doi: 10.1109/CVPR.2017.660.*

[6] *Image parsing with a wide range of classes and scene-level context: M. George, "Image parsing with a wide range of classes and scene-level context," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 3622-3630, doi: 10.1109/CVPR.2015.7298985*

[7] *SYGNet: A SVD-YOLO based GhostNet for Real-time Driving Scene Parsing: H. Wang et al., "SYGNet: A SVD-YOLO based GhostNet for Real-time Driving Scene Parsing," 2022 IEEE International Conference on Image Processing (ICIP), Bordeaux, France, 2022, pp. 2701-2705, doi: 10.1109/ICIP46576.2022.9897534*

[8] *Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A., & Torralba, A. (2016). Semantic Understanding of Scenes through the ADE20K Dataset (Version 2). arXiv. https://doi.org/10.48550/ARXIV.1608.05442*

[9] *Kong, S., & Fowlkes, C. (2018). Recurrent Scene Parsing with Perspective Understanding in the Loop. In 2018 IEEE/CVF Conference on Computer Vision and*

Pattern Recognition. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE. https://doi.org/10.1109/cvpr.2018.00106

[10] *Kirillov, A., He, K., Girshick, R., Rother, C., & Dollár, P. (2018). Panoptic Segmentation (Version 3). arXiv. https://doi.org/10.48550/ARXIV.1801.00868*

[11] *Jain, J., Li, J., Chiu, M., Hassani, A., Orlov, N., & Shi, H. (2022). OneFormer: One Transformer to Rule Universal Image Segmentation (Version 2). arXiv. https://doi.org/10.48550/ARXIV.2211.06220*

[12] *Minderer, M., Gritsenko, A., Stone, A., Neumann, M., Weissenborn, D., Dosovitskiy, A., Mahendran, A., Arnab, A., Dehghani, M., Shen, Z., Wang, X., Zhai, X., Kipf, T., & Houlsby, N. (2022). Simple Open-Vocabulary Object Detection with Vision Transformers (Version 2). arXiv. https://doi.org/10.48550/ARXIV.2205.06230*

[13] *ADE20K Dataset : Source*

[14] *Prepare ADE20K Dataset: Source*

[15] *ADE20K Pickel file GitHub - CSAILVision/ADE20K: ADE20K Dataset*

[16] *Pixellib Documenatation: Source*

[17] *Implementing Real-Time Semantic Segmentation in Your Projects : Source*

[18] *He, K. Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. 1512.03385.pdf (arxiv.org)*

[19] *Chollet, F. (2016). Xception: Deep Learning with Depthwise Separable Convolutions*

[20] *DeepLabv3 & DeepLabv3: Source*

[21] *Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. https://doi.org/10.48550/arXiv.1505.04597*

[22] *Image Segmentation with U-Net: Source*

[23] *Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. : https://doi.org/10.48550/arXiv.1506.02640*

[24] *SAM : https://docs.ultralytics.com/models/sam/*

[25] *Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., & Girshick, R. B. (2023). Segment Anything. : https://doi.org/10.48550/arXiv.2304.02643*

Appendix

Summary of Technologies Used

- **IDE:** PyCharm Community, VS Code, Jupyter Notebook, Google Colab, Kaggle
- **Version Control Tools:** GitHub, Git bash
- **Programming Languages:** Python, JavaScript, HTML, CSS
- **Python Libraries:** Pandas, MatPlotLib, Keras,Tensorflow, Pytorch, Pixellib, Pillow, Scipy, Sk-Learn, Flask

The source code can be found on GitHub. - **GitHub**