Aerial Mapping of Renewables

Nikila Govindarajan Saravanarajan

Master of Science

School of Informatics

University of Edinburgh

2023

Abstract

The modern world faces alarming levels of pollution, leading to significant health and environmental consequences. Industrial activities, powered predominantly by non-renewable energy sources, have been major contributors to this environmental degradation. Recognizing the pressing need for sustainable energy sources, solar energy has emerged as a viable solution. Solar panels, or photovoltaic (PV) cells, efficiently harness sunlight, converting it into energy even under cloudy conditions. This research project, in collaboration with OnGen - an Edinburgh-based company specializing in promoting green energy solutions, aims to automate the identification of potential sites for solar panel installation from aerial imagery. The primary objective is the detection of rooftops suitable for solar installation. The process involves: Identification of roof spaces from aerial datasets, Determination of vacant roof areas apt for solar panel placement, and Pinpointing the exact regions suitable for panel fitting.

To accomplish these objectives, rooftops were discerned from aerial photographs utilizing the advanced Mask R-CNN object detection and instance segmentation technique via Detectron2, achieving a Mean Average Precision (mAP) of 0.67 and Mean Average Recall (mAR) of 0.59. Subsequently, the YOLOv8 (You Only Look Once version 8) instance segmentation model was employed to determine vacant rooftop segments, registering a commendable F1 score of 0.59.

For the project's needs, datasets have been sourced from Roboflow, a renowned platform in the domain of Computer Vision. This automation not only facilitates quicker and more accurate site identification but also underscores the significance of transitioning to renewable energy sources in the fight against industrial pollution.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Nikila Govindarajan Saravanarajan*)

Acknowledgements

First and foremost, I wish to express my profound gratitude to my supervisor, Michael Herrmann. His consistent guidance, invaluable insights, and steadfast support have been instrumental in bringing this dissertation to fruition. My appreciation also extends to Emma and Chris from OnGen. Their generosity in affording me the opportunity to undertake this project, coupled with their expertise, played a crucial role in my journey.

At the heart of my support system are my parents and brother, whose unwavering belief in me served as a beacon during moments of doubt. Their encouragement and wisdom have been the pillars upon which this achievement stands.

I am also deeply thankful to my friends: Ruchira Sakalle, Sartaj Syed, Zain Amir, Arshia Shetty, and Ishika Bhansali. Their assistance during challenging phases and their unwavering belief in my capabilities helped navigate the intricacies of this project.

In closing, each individual mentioned has contributed uniquely to this endeavor, and for that, I am eternally grateful. It is on the foundation of their support and faith that this dissertation stands tall.

Table of Contents

Chapter 1

Introduction

1.1 Project Motivation

The current state of the world is characterized by widespread pollution and a decline in the health of living organisms. One of the primary drivers of pollution is industrial activity, which relies heavily on non-renewable energy sources to power machines and devices. The growth of industrialization has significantly contributed to the escalation of pollution levels across the globe [22]. According to a report by the World Health Organization (WHO), outdoor air pollution is responsible for an estimated seven million premature deaths each year, with industrial emissions being a significant contributor to this figure (WHO, 2018). The combustion of fossil fuels in industrial processes releases various harmful substances into the atmosphere, including nitrogen oxides, sulfur dioxide, and particulate matter, which can have severe impacts on human health (Environmental Defense Fund, n.d.). Moreover, a study by the United Nations Environment Programme (UNEP) reveals that industries are the largest source of hazardous waste globally, with over 400 million tons generated annually (UNEP, 2015). This waste often ends up in landfills or is released into the environment, polluting water and soil and posing significant health risks to humans and wildlife. Therefore, It is imperative to adopt cleaner and sustainable sources of energy and to implement stricter regulations to minimize industrial pollution and protect the environment and public health.

One sustainable source of energy or a renewable source of energy is solar energy which can be harvested with the use of solar panels. Renewable energy is the energy derived from natural sources that are replenished at a faster pace than they are consumed. Solar panels are photovoltaic (PV) cells fitted on the top of rooftops to absorb sunlight to convert it into solar energy. These solar panels harvest solar energy even in cloudy

weather conditions which make them very reliable.

OnGen is a company that has been founded in Edinburgh to help industries big or small to minimise their greenhouse gas emission. They specialize in using cutting-edge digital tools to analyze property-specific data. By doing so, they make it effortless for individuals and organizations to make informed decisions regarding energy efficiency, generating their own energy, or transitioning to greener tariffs.

The scope of this research project is to automatize the software that is being used by OnGen to detect spaces for the installations of renewable sources of energy such as solar panels within the properties of these organisations to make the process for the organisation hassle-free and less human intensive. On doing so, the identification of spaces will be faster thus delivering the clients with accurate results at a quicker pace.

1.2 Project Objectives

In this research project, the main objective would be to finding roofspaces on which solar panels can be successfully fitted in-order to harvest solar energy in an efficient way. Steps taken to do this are:

- Detecting roofspaces from the dataset obtained from aerial images.
- Detecting empty roof spaces on which solar panels can be installed by manually annotating the results obtained (Roof tops from aerial images of buildings) from the previous step to create a second dataset to identify empty roof spaces.
- Effectively find the region in which the solar panels can be fitted.

For the purpose of this project, the dataset has been collected from Roboflow which is a Computer Vision developer framework for better data collection to preprocessing, and model training techniques. Roboflow has public datasets readily available to users and has access for users to upload their own custom data also.

1.3 Project Structure

In this document, we will start by exploring the existing work, the related work and the methodologies what were used to complete project along with the evaluation methods which prove the efficiency of the work.

Chapter 2

Literature Review

2.1 Background

The primary focus of the project revolves around computer vision, particularly concerning models and image processing techniques employed for object detection. The objective is to detect rooftops from aerial images of buildings. Below, we will explore some widely used methodologies for object detection in this context.

2.1.1 Object Detection with Deep Learning

Object detection involves the accurate identification and localization of target objects within images. This task goes beyond mere object classification, as it encompasses not only categorizing object types but also determining their spatial orientation within the provided image [22]. Object detection in the recent years has been done mainly with the help of deep learning algorithms. The two types of commonly used algorithms are One-stage detectors and Two-Stage detectors. One-stage detectors are utilised primarily when speed is the key point and Two-Stage detectors are used when accuracy is our main concern. Let us look a little more in detail about these two algoritms.

- One-stage detectors

  One-stage Object Detection Models belong to a category of object detection models characterized by their single-stage approach. Unlike two-stage models, which involve a preliminary stage of region proposal followed by object detection, One-stage models directly perform detection on a densely sampled set of locations within an image. This streamlined process typically leads to quicker inference times.

In One-stage object detection, the model doesn't rely on a separate step for generating region proposals before detecting objects. Instead, it directly predicts object classes and bounding box coordinates in a single pass over the input image. This design simplifies the detection pipeline, making it computationally more efficient and suitable for real-time applications.

One-stage models are often known for their simplicity and speed, making them well-suited for scenarios where fast processing is crucial, such as in video analysis, robotics, and embedded systems. However, they may require more training data and fine-tuning to achieve competitive accuracy compared to their two-stage counterparts.

One-stage object detection models streamline the detection process by combining region proposal and object detection into a single step, resulting in faster inference times and making them particularly valuable for applications demanding real-time or high-speed processing [26, 32, 41].

Among the frequently employed algorithms for One-stage object detection, the You Only Look Once (YOLO) framework stands out prominently [17, 32]. YOLO has manifested itself in multiple iterations, each refining and advancing the concept further. These iterations include YOLOv1, YOLOv2, YOLOv3, and subsequent versions. YOLO is recognized for its characteristic approach of simultaneously performing object detection and classification in a single pass over an image, thus eliminating the need for a two-stage proposal process.

The YOLO models have undergone progressive enhancements in terms of architecture, accuracy, and speed. They have been trained on extensive datasets to effectively detect and classify objects within images. These pretrained deep learning models offer the advantage of rapid object detection, making them particularly suitable for scenarios where real-time or near-real-time analysis is crucial.

- Two-Stage detectors

  Two-stage object detectors are a class of object detection models that follow a two-step process to identify objects within an image. These detectors are known for their effectiveness in accurately localizing objects and are often used for complex or challenging detection tasks. The two-stage approach involves region proposal generation in the first stage, followed by object classification and bounding box refinement in the second stage.

The breakdown of these two stages are explained below:

- Region Proposal Generation In this stage, the model identifies potential regions of interest (ROIs) within the image that are likely to contain objects. These regions are proposed based on various techniques, such as selective search, region proposal networks (RPNs), or similar methods. The goal is to reduce the search space for objects and focus computational resources on relevant areas.
- Object Classification and Refinement Once the regions of interest are identified in the first stage, the second stage involves classifying the proposed regions and refining the bounding box predictions. The model assigns object labels to the proposed regions and fine-tunes the bounding box coordinates to accurately localize the objects within those regions.

Two-stage detectors are known for their accuracy and ability to handle objects of various sizes and orientations. They often achieve high precision and are particularly suitable for scenarios where precise object localization is crucial, such as medical imaging, autonomous driving, and aerial imagery analysis.

In the recent times, there have been multiple two-stage detectors which are used. The most prominent ones are listed below:

- Faster R-CNN

  Faster R-CNN introduced the concept of using a Region Proposal Network (RPN) to generate region proposals for potential objects. It combines the proposal generation and object detection processes into a single end-to-end network. The RPN generates object proposals based on anchor boxes and their associated scores, which are then used as candidate regions for object detection. The output of the RPN is fed into a classifier and a regressor to perform the final object detection [10].

- R-CNN

  R-CNN was one of the pioneering two-stage object detection methods. It introduced the idea of using selective search for generating region proposals and then using a Convolutional Neural Network (CNN) to classify objects within those regions. R-CNN uses a separate CNN for each proposed region to extract features and perform classification. Although R-CNN achieved

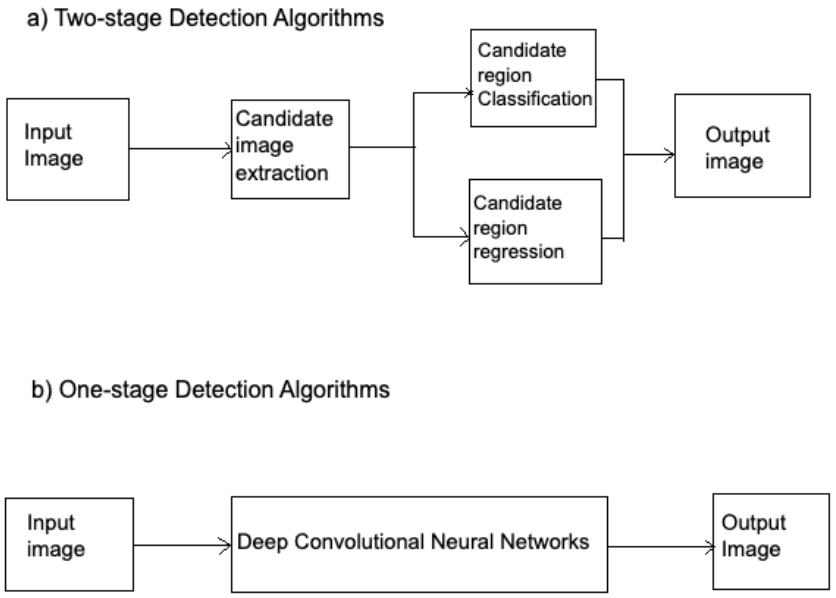


Figure 2.1: Object Detection algorithms flowchart [46]

good results, it had performance limitations due to the need to process each proposed region individually [11].

– Mask R-CNN

  Mask R-CNN is an extension of Faster R-CNN that goes beyond object detection to include instance segmentation. In addition to predicting object bounding boxes and classes, Mask R-CNN predicts object masks for each detected instance. It adds a mask branch to the Faster R-CNN architecture, enabling pixel-wise segmentation of objects within the proposed regions. This makes Mask R-CNN suitable for tasks that require not only object detection but also accurate delineation of object boundaries [13].

The image 2.1 shows the structure of One-stage and two-stage object detection algorithms.

2.1.2 Image Segmentation

Image Segmentation has gained great traction in the field of computer vision in the past few years. Image segmentation is the process of segmenting an image into different regions to better understand the objects or the types of regions present in the frame. It is the process of identifying boundaries in an image and categorise them according to the

type of pixels such that in the end the pixels of interest are classified under one category. There are multiple types of image segmentation techniques available but we will be looking at the two most commonly used algorithms and their properties [45].

- Semantic Segmentation

  Semantic segmentation is a computer vision technique that involves assigning a class label to each pixel in an image, effectively dividing the image into distinct regions corresponding to different object categories or structures. This fine-grained pixel-level labeling provides a detailed understanding of the composition and layout of objects within the scene.

  In semantic segmentation, the primary objective is to classify every pixel into one of several predefined classes, such as ”car,” ”tree,” ”building,” ”road,” ”sky,” and more. This process generates a segmentation map that highlights the boundaries and locations of different objects, enabling a comprehensive analysis of the image content.

  Semantic segmentation is a foundational technique in computer vision, enabling a detailed understanding of image content and supporting various applications such as scene understanding, autonomous driving, medical image analysis, and more [27].

- Instance Segmentation

  Instance segmentation is a computer vision task that combines object detection and semantic segmentation to provide a more detailed understanding of an image. In instance segmentation, the goal is to not only classify each pixel into specific object classes (as in semantic segmentation) but also distinguish between different instances of the same class, assigning a unique label to each individual object instance.

  In other words, instance segmentation goes beyond semantic segmentation by not only identifying objects but also segmenting each object instance separately. This allows for precise localization of object boundaries and provides a pixel-level mask for each instance, outlining the exact shape and position of each object.

  Instance segmentation is particularly useful in scenarios where multiple instances of the same object class appear in close proximity or overlap with each other. It enables a more detailed and accurate understanding of complex scenes containing multiple objects of interest.

Instance segmentation is a more challenging task compared to semantic segmentation due to the need to differentiate between individual instances of the same class. However, it provides richer and more detailed information about the objects in an image, making it a crucial technique for various advanced computer vision applications [39].

2.2 Related work

The endeavor to identify appropriate roof spaces for solar panel installations using aerial imagery is an emerging field with limited existing literature. While current studies primarily focus on distinguishing roofs with pre-installed solar panels from vacant ones, deep learning techniques, especially in instance segmentation and object detection, have become the cornerstone of such research endeavors. This nascent area has witnessed a stream of advancements in accurately detecting roof spaces optimal for solar panel deployments.

Over the past few years, the realms of computer vision and image analysis have experienced monumental progress, leading to the development of groundbreaking methodologies for processing and analyzing visual data. Techniques encompassing object identification, segmentation, and scene analysis have demonstrated significant efficacy, especially in healthcare applications [14].

The confluence of deep learning techniques has transformed the landscape of computer vision. It has enhanced the precision and efficiency of decision-making processes, paving the way for innovative approaches to diverse challenges. Renowned object detection frameworks like R-CNN, Fast R-CNN, and Faster R-CNN stand as testament to their robustness and proficiency [2]. Additionally, the synergy of deep learning with Convolutional Neural Networks (CNNs) has found applications across various sectors, notably in satellite image processing for tasks like LiDAR data interpretation, architectural delineation, and building categorization via segmentation [38]. Despite these advancements, a gap remains in employing deep learning-based CNNs to ascertain the existence or nonexistence of solar PV on rooftops through satellite imagery analysis.

Semantic segmentation, a cornerstone of computer vision, has unveiled unparalleled potential, extending from granular pixel identification to holistic scene interpretation, and has become indispensable in fields like autonomous navigation [35]. The capabilities of Deep Neural Networks (DNNs) in proficiently segmenting diverse objects within visuals have been well-documented in academic literature, underscoring their potential

for accurate object recognition following training phases [36].

As this research domain continues its evolution, our investigation seeks to pioneer a new trajectory by harnessing deep learning CNNs to evaluate the presence of solar PV structures on rooftops using satellite imagery. This venture not only capitalizes on recent technological strides but also ventures into a relatively uncharted territory of computer vision, enriching both the academic and practical horizons [20].

Chapter 3

Methodology

3.1 Dataset - Collection and Usage

3.1.1 Roboflow

The problem statement requires aerial dataset of buildings. For this, I have utilised freely available Roboflow dataset to train the proposed system. Roboflow is a platform designed to streamline and simplify the process of preparing, augmenting, and deploying computer vision datasets [9]. It aims to make the process of going from raw images to a trained model more manageable, irrespective of the size of the dataset. This dataset from Roboflow contains 339 training images, 30 validation images and 91 test images. The dataset contains 13 different categories of rooftops based on local visual features. These categories are: Building roof, Commercial flat roof, commercial slope roof, construction area, flat roof, playground, slope flat roof, slope roof, solar flat roof, solar panel ground, solar slope roof, tree shading slope roof and unknown shape roof. Some examples of these roofs are shown in Image 3.1 with a bounding box with the roof area shaded depicting the roof area that is available to us. These different categories prove to be advantageous to us in detecting the estimation of solar energy required and help us analyse the power consumption in each of these buildings. For example, an independent house or a flat will not consume as much power as required by a commercial building. Based on this on deploying this system to the OnGen website, the program will be able to detect the precise amount of energy that has to be generated to supply the building in question.

Figure 3.1: Example of labels for different types of rooftops

3.1.2 COCO (Common Objects in COntext) annotations

COCO (Common Objects in COntext) is a large-scale object detection, segmentation, and captioning dataset. When we talk about COCO annotations, we're referring to the way objects in this dataset are labeled and described. COCO annotations provide a standardized format for object detection, segmentation, and captioning, making it easier for researchers and developers to train and validate models on the dataset [25]. It is important to annotate the images of a dataset to understand the coordinates of the Region Of Interest (ROI) and to add labels to these images based on their unique IDs. 3.2.

```
{
    "info": {...},
    "licenses": [
        {
            "id": 1,
            "name": "Attribution-NonCommercial-ShareAlike License",
            "url": "http://creativecommons.org/licenses/by-nc-sa/2.0/",
        },
        ...
    ],
    "categories": [
        ...
        {
            "id": 2,
            "name": "cat",
            "supercategory": "animal",
            "keypoints": ["nose", "head", ...],
            "skeleton": [[12, 14], [14, 16], ...]
        },
        ...
    ],
    "images": [
        {
            "id": 1,
            "license": 1,
            "file_name": "<filename0>.<ext>",
            "height": 480,
            "width": 640,
            "date_captured": null
        },
        ...
    ],
    "annotations": [
        {
            "id": 1,
            "image_id": 1,
            "category_id": 2,
            "bbox": [260, 177, 231, 199],
            "segmentation": [...],
            "keypoints": [224, 226, 2, ...],
            "num_keypoints": 10,
            "score": 0.95,
            "area": 45969,
            "iscrowd": 0
        },
        ...
    ]
}
```

Figure 3.2: COCO Annotation format

3.2 Detecting roof tops

The aim of this section of the project is to detect rooftops from images of buildings while ignoring the background, such that only the roofs are segmented, cropped and stored for further processing. In order to achieve this, multiple pretrained object detection and segmentation algorithms are available. But for this project I have used Facebook's object detection algorithm, Detectron2 along with Mask R-CNN to identify and crop rooftops.

3.2.1 Model Selection

The most crucial step for any data science related project is choosing the most apt model for the problem statement. There are many factors which have to be taken into account while making this selection. For the problem statement at hand, the first step is to successfully detect rooftops from aerial images of buildings. In order to achieve this, two main algorithms are to be employed, namely, object detection and instance segmentation. Both of these algorithms have been explained in detail in Chapter 2. There are multiple state of the art algorithms available to us to achieve these two tasks. The important features that are to be considered for model selection are 1. Accuracy: This is the most important factor to be considered. The model must be able to provide the best accuracy depending on the dataset. 2. Speed: The model should be able to perform within a reasonable time-frame but this would also depend on the type of the dataset used and how many layers the chosen model is configured. 3. Robustness: The model must be able to handle different real life scenarios such as lightning conditions, occlusions and viewpoints. 4. Versatility: Since the model that has to be used in this problem statement needs to handle both object detection and instance segmentation together, this is a very important feature to be considered. 5. Fine tuning and transfer support: Since the dataset available to us is not vast, the model must be able to support transfer learning efficiently, allowing us to fine-tune pre-trained models on a small dataset. 6. Scalability: The dataset contains 9 different categories, therefore the chosen model must be highly scalable.

On keeping the above mentioned features in mind I have chosen Decetron2 with Mask R-CNN for object detection and instance segmentation. Mask R-CNN builds on the success of fast R-CNN by adding a branch to predict the segmentation mask at each region of interest (RoI). This means that it can perform both object detection and pixel-wise instance segmentation simultaneously. In benchmarks, it has shown superior

results. Moreover, Detectron2 offers more out-of-the box features such as panoptic segmentation and DensePose which makes it versatile for multiple image vision tasks. Since Detectron2 is a pre-trained model offered by Facebook AI Research (FAIR) ensures regular updates, new features, and a high-quality codebase. Apart from this, Detectron2 is built natively with PyTorch, one of the leading deep learning frameworks. This makes it easy to integrate with other PyTorch-based tools and libraries. Hence using this model will be advantageous to the reseach at hand.

3.2.2 Detectron2 with Mask R-CNN

Detectron2 is a pretrained deep learning framework backed by Fackebook AI Research (FAIR) which is widely being utilised in the cases of object detection with segmentation. Detection of rooftops from aerial images of buildings is classified under the task of object detection along with instance segmentation. Detectron2 uses a Mask R-CNN architecture that combines object detection and instance segmentation in a single framework. The Mask R-CNN model contains two components, a region proposal network (RPN) that generates candidate object locations, and a network that predicts the class, bounding box, and mask for each candidate region [7]. The basic architecture and components of Detectron2 are shown in image 3.3. The Detectron2 framework consists of two core components, a backbone network and a head network. The backbone network serves as the primary mechanism for feature extraction, using the capabilities of convolutional neural networks (CNNs) to generate feature maps from the input image. These processed feature maps are then passed to the subsequent ”head” network, whose responsibility extends to detection, segmentation, and even instance-specific tasks when employing architectures like Mask R-CNN.

Within this head network, two critical stages come into play: the region proposal network (RPN) and the detection segment. The RPN’s role is to formulate initial object proposals by scrutinizing the feature maps across diverse scales and dimensions. For every proposed region, RPN gauges the probability of the region encompassing an object and estimates the bounding box parameters for it. Then comes the detection segment, which not only refines these initial proposals but also categorizes them based on their visual traits. When using Mask R-CNN, this stage is further enhanced to produce pixel-wise masks for each object, providing a detailed instance segmentation.

Detectron2, equipped with a myriad of advanced techniques, aims for enhanced accuracy and operational efficiency. One of its notable strategies is the deployment of

feature pyramid networks (FPNs) that draw features over various scales and resolutions, ensuring precise detection of objects, irrespective of their dimensions [24]. Additionally, with the integration of anchor boxes, Detectron2 presents a foundational perspective about the expected shapes and sizes of objects within the image, thus refining the object proposal search. The combination of these elements, especially with the inclusion of Mask R-CNN, offers a robust approach to object detection and instance segmentation [13, 10].

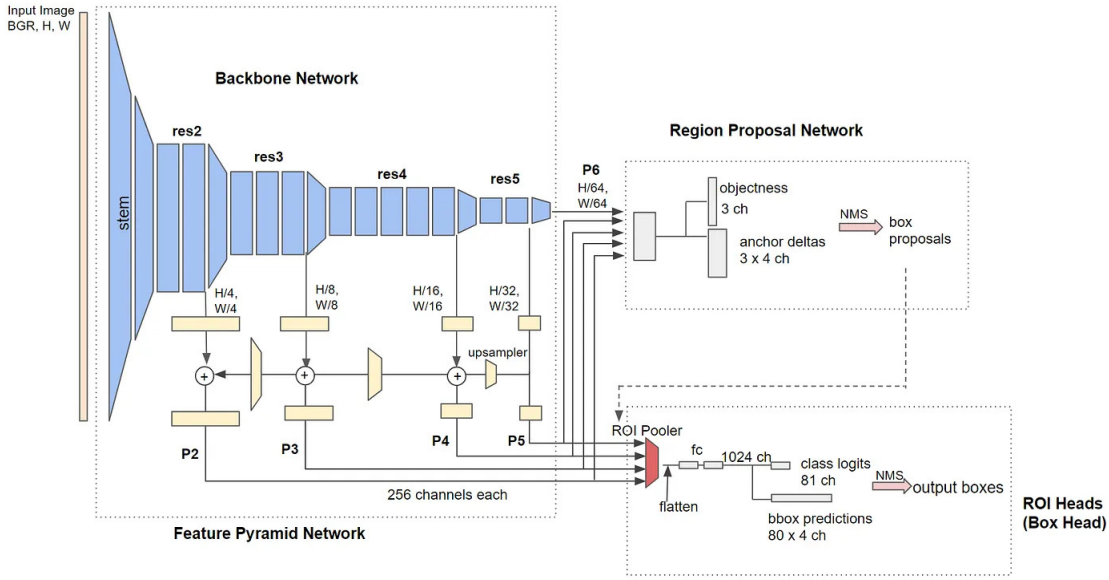


Figure 3.3: Architecture of Detectron2 with a baseline R-CNN model

3.2.2.1 Model Training loop

The training loop is a fundamental part of deep learning code, iterating through batches of data, feeding them into the model, and updating the model's weights based on the computed gradients. In the proposed system, the training loop not only updates the model but also includes features for periodic evaluation, logging, and early stopping. The Model training loop is depicted in the flowchart in Figure 3.4.

The training process for an object detection and instance segmentation model starts with the Initialization phase. During this phase, the model's architecture is defined, and systems for weight updating, like the optimizer and the learning rate scheduler, are set up. With the foundational elements in place, the model begins pulling in batches of images and annotations from the dataset, marking the start of the Training Loop. In each iteration of this loop, the model takes a batch of data, processes it, and produces

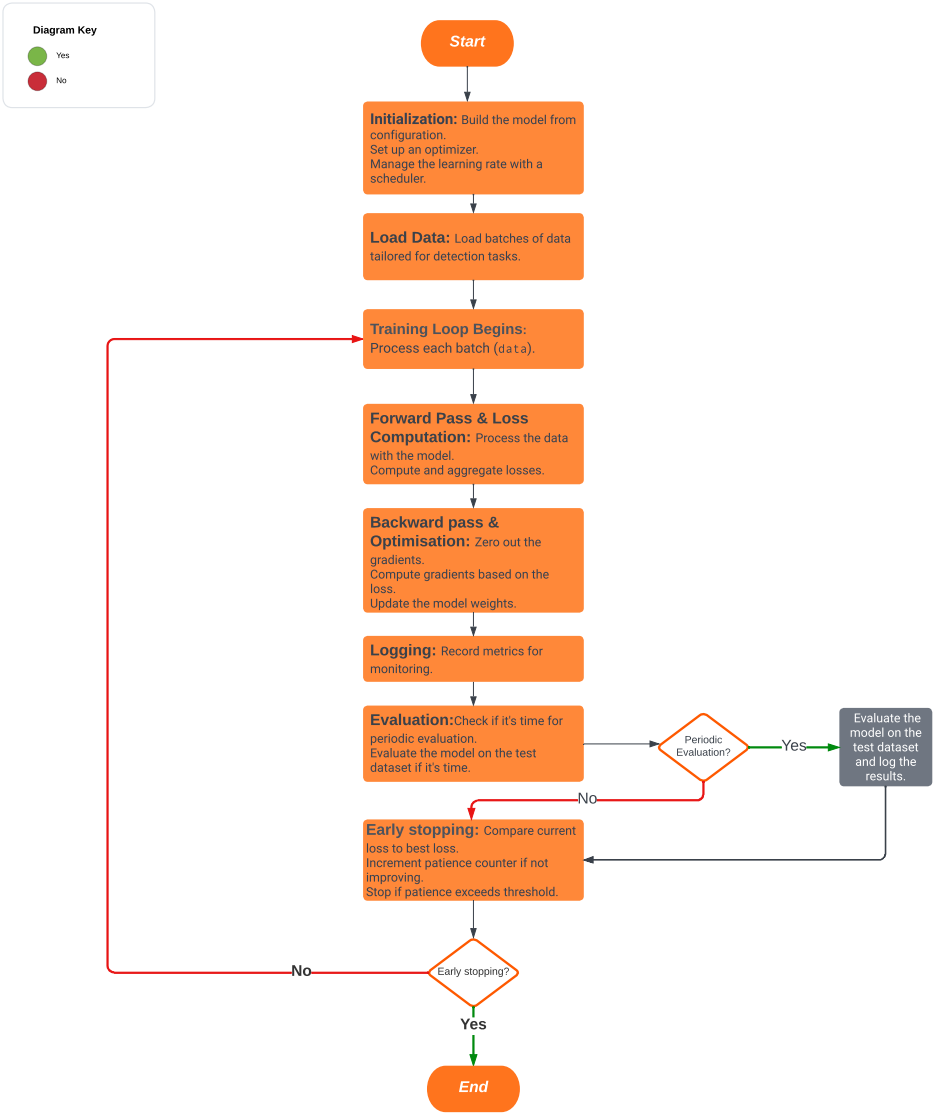


Figure 3.4: Detectron2 with Mask R-CNN Model Training Loop

predictions in what's termed the Forward Pass. These predictions are then compared to the actual labels, and a loss value is calculated to quantify the model's prediction errors. The importance of this loss value is profound; it guides the model's learning. Using this loss, in the Backward Pass, the model calculates gradients that describe how each weight should be adjusted to reduce prediction errors. The optimizer then steps in, leveraging these gradients to fine-tune the model's weights.

While the training loop continues, there are systematic checks in place to ensure the model's efficacy. Periodically, at predetermined intervals, the model undergoes

Evaluation on a separate test dataset. This is an insightful phase, revealing how the model is likely to perform on unseen data and ensuring it's learning correctly. However, to avoid the pitfall of overfitting, where the model becomes too tailored to the training data and loses generalization capabilities, there's an Early Stopping mechanism. If the model's performance doesn't show improvement over a specific number of iterations, termed as PATIENCE, the training is halted, ensuring resources aren't wasted and the model remains optimal. If neither evaluation nor early stopping intervenes, the training process gracefully moves to the next iteration, repeating the cycle until either all data batches are processed or the early stopping criterion is met. Thus, the journey from initialization to a trained model is both iterative and evaluative, ensuring not just learning but effective learning.

3.2.3 Advantages of combining Detectron2 with Mask R-CNN

Detectron2 is a popular computer vision library developed by Facebook AI Research, and it offers a flexible and efficient framework for training and deploying object detection and segmentation models. An important model architecture that Detectoron 2 supports is the mask R-CNN (region-based convolutional neural network). Mask R-CNN is a powerful and widely used architecture for example segmentation tasks, and its integration with Detectron2 brings several advantages:

- Instance Segmentation Capability Mask R-CNN is specifically designed for instance segmentation, which involves both object detection (identifying object bounding boxes) and pixel-level segmentation (segmenting individual object instances within those bounding boxes). Detectron2's integration of Mask R-CNN allows you to tackle instance segmentation tasks effectively.

- High Accuracy Mask R-CNN has demonstrated state-of-the-art performance on various instance segmentation benchmarks and challenges. By using Mask R-CNN in Detectron2, you can leverage its accuracy for complex tasks where distinguishing between different object instances is important.

- Easy Model Confguration Detectoron 2 provides a user-friendly configuration system that allows you to customize and fine-tune various aspects of the Mask R-CNN model, such as anchor size, input image size, and more. Hyperparameters. This flexibility enables you to adapt the model to your specific use case and data.

- Efficient Training Detectron2 is designed to efficiently use hardware resources (e.g., GPUs) during training, making it well suited for training complex models such as masked R-CNNs. It supports multi-GPU training and distributed training, enabling faster convergence and reducing training time.

- Scalability Detectron2 is built on top of PyTorch and benefits from its dynamic computation graph, making it easy to experiment with different model architectures and techniques. This scalability allows researchers and practitioners to innovate and experiment effectively.

- Wide Adoption and Community Support Detectoron 2 has gained significant popularity within the computer vision community. Its active development and large user base ensure that you can find resources, tutorials and solutions to common issues related to Mask R-CNN and other models.

- End-to-End Workflow Detectron 2 provides a seamless end-to-end workflow for object detection and instance segmentation tasks. This includes data loading, preprocessing, model training, evaluation, and estimation, streamlining the entire process.

- Transfer Learning Detectron2 allows you to start from pre-trained models on large benchmark datasets, such as COCO, and fine-tune them on your specific dataset. This transfer learning can significantly speed up convergence and improve model performance.

In summary, the integration of Detectron2's mask with R-CNN offers a powerful and efficient solution to deal with instance segmentation tasks. It combines the strengths of both library and architecture to provide a versatile and effective tool to detect rooftops in the current computer vision task [4]. The rationale for selecting Mask R-CNN over other R-CNN variants is elaborated upon in Appendix A.

3.3 Mapping empty spaces for solar panels on roof tops

The final output from the previous model provides the predicted rooftops with an accuracy score for each of the images. From this, I have cropped out just the instance segmented region of the result such that only the rooftop is available. The image 3.5 gives a representation of this. The image in 3.5a gives the accuracy of the type of roof

(a) Segmentation and bounding box

(b) Cropped segmented region

Figure 3.5: Example of segmentation and cropped imaged from the segmented region

that has been identified from the model with an accuracy of 85 and the shaded region depicts the roof region which is our region of interest (ROI). This ROI is cropped and shown in Image 3.5b. This cropped image will be further used for the final model to detect empty (void of obstacles) spaces on rooftops where solar panels can be successfully installed. This new dataset is formed from the test images that was used to predict the roof spaces and type of roof from the previous model.

3.3.1 Custom Annotations

After obtaining the cropped images from the aforementioned model, it's essential to annotate them to serve as training data for a subsequent model, which is specifically designed to detect empty roof spaces. In order to do this, Roboflow is used to manually annotate the images with a single label called "solar" to identify these empty regions. The image 3.6 shows an example of the manual annotations as motioned. Since the test dataset of the previous model is being used, there are 91 images available for the next dataset. Since the dataset size is very small, image augmentation techniques were utilised to enlarge the dataset. On applying rotations and blur to the images, 264 images were obtained from where they were split as 70% train, 10% validation and 20% test datasets.

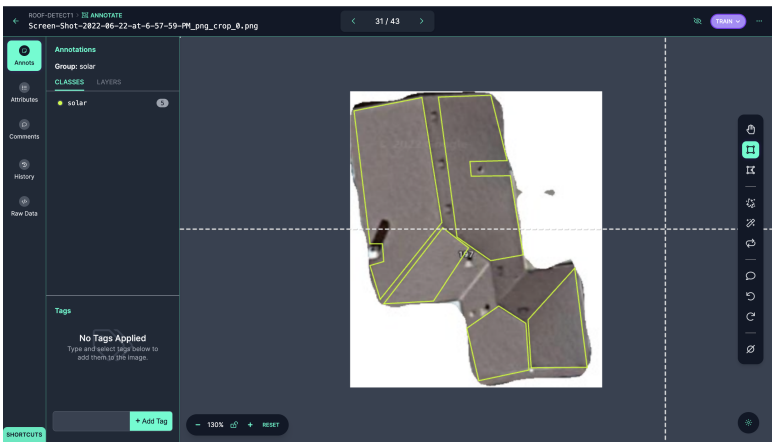


Figure 3.6: Manual annotations on Roboflow

3.3.2 YOLO (You Only Look Once) annotations

For this part of the problem statement, YOLOv8 model is being utilised for detecting empty spaces on the rooftops. For this model, the annotations are in the YOLO format. These annotations are different in comparision to the genric bounding-box representations. In the YOLO annotation format, for each object in the image, a line in the annotation file is made, containing: 1) Class index. 2) The x and y center coordinates of the bounding box, normalized by the image width and height respectively, such that they are in the range of 0 and 1. 3) The width and height of the bounding box are also normalized by the width and height of the image, respectively [33, 32]. *For example: " 1 0.716797 0.395833 0.216406 0.147222 " shows an instance of YOLO annotation. In this case, it represnts an object of class 1 and the normalized center coordinates respectively.*

3.3.3 YOLOv8

In the domain of computer vision, the YOLO (You Only Look Once) series stands out for its exceptional speed and efficiency. The 2023 release, YOLOv8, has surpassed previous versions, achieving record-breaking accuracy. YOLO's continuous advancements underscore its prowess in identifying objects within images and videos. As a One-stage detection method (detailed in Section 2), it's particularly suited for real-time video and image processing. Compared to two-stage detection techniques, YOLO's processing is significantly swifter, leading to reduced processing time and more immediate results in real-time scenarios [28]. For the case of detecting empty spaces on rooftops, the YOLOv8 model was primarily chosen for its speed due to its ability to detect objects in

a single forward pass of the network [12].

3.3.3.1 Architecture of YOLOv8

The architecture of YOLOv8 closely mirrors that of YOLOv5 [16] in its fundamental aspects, particularly within its backbone. The transition from the convolutional 3 (C3) module, a primary building block consisting of multiple convolutional layers, to the convolutional 2 fusion (C2f) module represents a significant advance. The development was inspired by the Cross Stage Hierarchical Network (CSPNet) design approach, where a conventional convolutional layer is split into a feature-rich part and a more lightweight part. This division facilitates improved efficiency without additional computational demands. The integration of Enhanced Learning to Augment Network (ELAN) technology, which emphasizes the refinement of feature learning, further improves the C2f module. As a result, YOLOv8 achieves a more comprehensive gradient flow, which is essential for accurate object detection, while maintaining its lightweight nature.

In the spinal cord termination segments, the well-known spatial pyramidal pooling fusion (SPPF) module remains intact. This module processes the data through three sequential maxpooling operations, each with a 5x5 dimension, before adding layers. This methodology ensures efficient identification of objects at a diverse range of scales, without contributing additional weight to the overall model.

For the backbone of its architecture, YOLOv8 employs the Pyramid Attention Network - Feature Pyramid Network (PAN-FPN) method, which optimizes the combination and utilization of feature layers at different scales. The model's creators cleverly combined two upsampling strategies, multiple C2f modules, and a state-of-the-art decoupled head structure to mold the throat module. Adopting the decoupled head concept from YOLOx [41] , YOLOv8 integrates object confidence scores with bounding box regression to achieve unprecedented accuracy levels.

One of the defining characteristics of YOLOv8 is its ability to be compatible with all previous versions of the YOLO series. Furthermore, its seamless compatibility between these versions and its expertise in different hardware platforms from central processing units (CPUs) to graphics processing units (GPUs) underline its dynamic flexibility. A visual representation of this architecture can be referred to in Figure 3.7, where CBS stands for a combination of convolution, batch normalization, and Scaled Exponential Linear Unit (SELU) activation functions [28, 34].

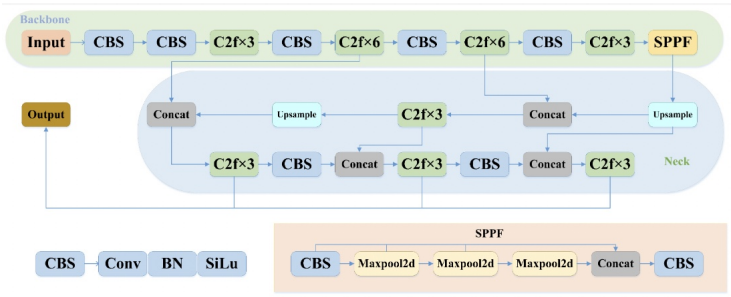


Figure 3.7: Architecture diagram of the YOLOv8 object detection model, showcasing the various layers, connections, and components essential for real-time object detection.

3.3.3.2 Proposed System

The system under discussion employs the YOLOv8 model to perform instance segmentation, specifically targeting vacant rooftop areas. Instead of juggling multiple categories, this system simplifies the process by only using a single label: ”solar” for the segmented regions. This streamlined approach is facilitated through the advanced capabilities of the YOLOv8-segmentation model. Once the dataset has been prepared after custom annotations as mentioned in Section 3.3.1. The ”data.yaml” file serves as a configuration file that provides the necessary details about the dataset being used for training or evaluation. This YAML (yet another markup language) file is essential for the model to correctly understand and process the dataset. Using this file, the segmentation model is trained on the training dataset [23].

In the proposed system, the YOLOv8 model is implemented along with early stopping to mitigate the issues of underfitting or overfitting [31]. The dataset is split into training and validation subsets. The purpose of this partition is twofold: the training set is used to train the model, while the validation set helps monitor the model’s performance on unseen data. This becomes important when implementing early stopping, a method designed to stop training when the model stops showing improvement, thus avoiding overfitting. The model itself, yolov8n-seg.pt, is loaded and initialized. A loss function (CrossEntropyLoss) and an optimizer (Adam) are specified for the training process. The training loop is started and runs for a predefined number of times set by the max_epochs variable. During each epoch, the model undergoes training using the training dataset. A forward pass is applied to obtain the predictions, and the loss is calculated by comparing these predictions to the true targets. This loss is then backpropagated through the network to update the model weights. Simultaneously, the performance of the model is evaluated on the validation dataset, and the validation loss is calculated. After each

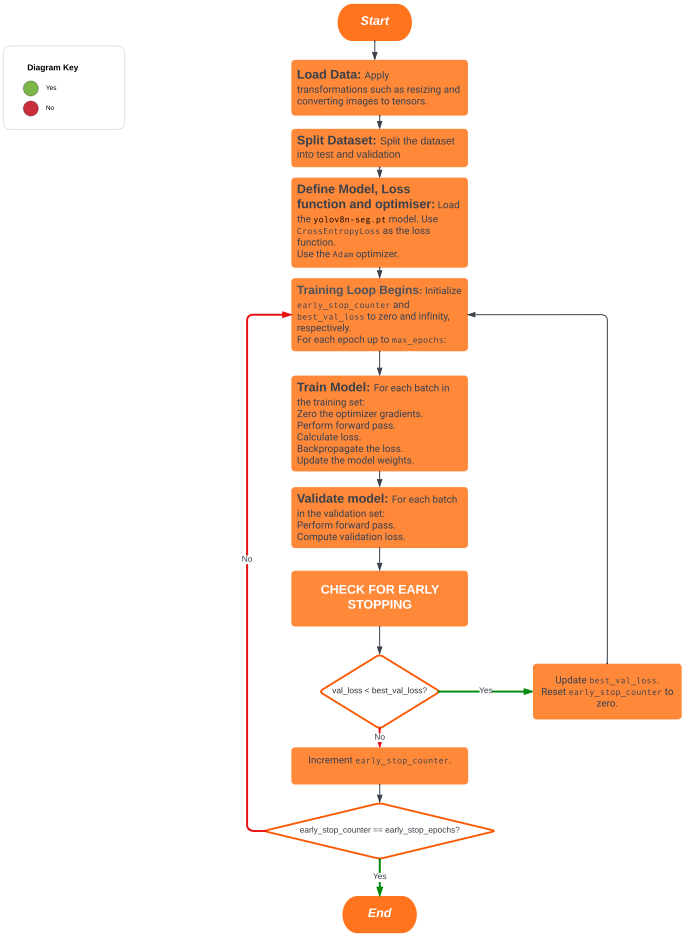


Figure 3.8: YOLOv8 proposed model training loop

epoch, the code checks whether the validation loss improves compared to the previous best validation loss [43]. If there is no validation loss for a predetermined number of consecutive epochs specified by early_stop_epochs, training is terminated, indicating that the initial stop method is active This ensures that the model will not continue training without any meaningful progress. Finally, once the training process, either reaching its maximum epochs or being halted by early stopping, concludes, the training and validation losses over the epochs are plotted. This visualization serves as a useful tool to gauge how well the model trained over time, showing its progression and helping in diagnosing potential issues like overfitting or underfitting. The flow of this process can be inferred from figure 3.8

3.4 Feedback loop

A feedback loop in the context of object detection and instance segmentation can be understood as a system where the output of a process (or model) is fed back into the system to refine, improve, or correct its subsequent output. This iterative process can greatly increase the performance and accuracy of object detection and segmentation algorithms [37, 30]. In this case, the input image provided by the user will be used to detect the rooftop using detectron2 and Mask R-CNN as mentioned in Section 3.2 after which the results of the cropped segmented area will be used to detect the empty roofspaces using the YOLOv8 model as mentioned in Section 3.3. The concept applies to these algorithms as follows:

- Error Rectification: After the initial run in the object detection algorithms in both the cases, the inaccuracies and wrongly labelled data will be identified. These errors are used to identify the errors to readjust the model parameters to better the results. After which, the adjusted data is rerun on the modified system to better the accuracy.

- Active Learning: To improve the accuracy of a model, experts manually label data about which the model is uncertain. This newly labeled data is added to the training set, and the model is trained again. By repeating this process, the model often performs better, even with less labeled data than conventional methods [5, 21].

- Iterative Refinement: Some models are designed with a built-in repeated feedback mechanism. For example, the model may output an initial segmentation mask, then adjust and refine this mask based on feedback from its own prior predictions [19].

- Post-processing Feedback: Following the initial detection or segmentation, certain heuristic post-processing techniques, such as non-maximum suppression in object detection, may be employed. These post-processing results can then guide adjustments to the model's raw predictions in subsequent iterations, completing the feedback cycle. This continuous refinement helps in enhancing the model's accuracy over time [18].

- Human-In-The-Loop (HITL): This is a more interactive form of feedback where humans intervene directly in the model's learning process. For instance, a human

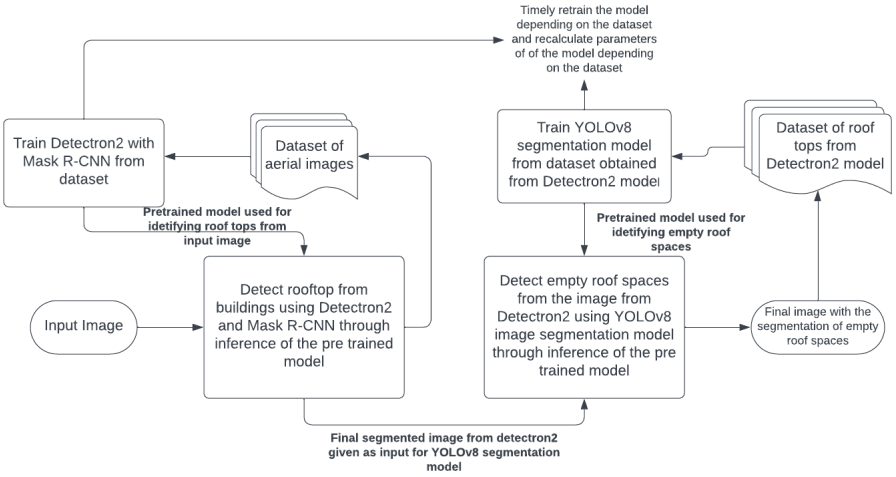


Figure 3.9: Representation of Feedback Loop

might correct a model's predictions, and these corrections are used to fine-tune the model.

For the problem statement at hand, I will be mostly utilising the Human-In-The-Loop feedback mechanism while integrating it with OnGen's existing system. In this situation the user will be able to modify the region in which they would prefer the solar panel to be fitted. Since client satisfaction is a prime factor in this system, the user will be allowed to manually bound the segmented region if it is not upto their liking. Upon such changes, the coordinates and the annotations of the modified segmented area will be recorded in the system and added to the dataset to increase accuracy. Since I have utilised two models for this system, the initial feedback loop will be used to detect only the roofspace from the aerial image. Upon successful detection, the user will be able to modify the segmented area. Once this is done, the next model will be run to identify the empty roof space area from the image modified by the user. Again, the user will be able to change the coordinates of this segmented area as well to add to the dataset if the YOLOv8 model. Hence, two feedback loops are implemented here to better the accuracy of the system. The flowchart of the system is given in Figure 3.9. The puesdocode of the same is dicussed more in detail in Appendix D.

One drawback of this system is that, the dataset size might increase overtime causing large processing times. In order to eliminate this, the previous data from the dataset must be discarded based on a threshold on the size of the dataset. This will make sure that the model retains its swift processing time while gradually increasing the accuracy.

Chapter 4

Evaluation & Discussion

In this section, we will first examine the outcomes achieved by the Detectron2 with the Mask R-CNN model in detecting roofspaces. Subsequently, we will delve into the results procured by the YOLOv8 model for segmenting unoccupied roof spaces, building upon the findings from the Detectron2 model.

4.1 Rooftop detection from aerial images

Detectron2, integrated with the Mask R-CNN architecture, represents one of the pinnacle achievements in the realm of computer vision, particularly for tasks involving bounding box detection and instance segmentation. Developed by Facebook AI Research (FAIR), Detectron2 is designed to identify objects within an image and encapsulate them within bounding boxes. In tandem with this, the Mask R-CNN framework, also a brainchild of FAIR, goes a step further by segmenting each individual instance within these bounding boxes, making it possible to distinguish between overlapping objects and providing a more granular view of the objects within an image.

The synergy between these two systems is quite evident. While Detectron2 lays down the groundwork by identifying and bounding potential objects, Mask R-CNN takes over to meticulously segment each detected instance, ensuring that even in densely packed images, every object is distinctly recognized [44, 13].

During the preliminary phases of this project, I experimented with employing Detectron2 in conjunction with Faster R-CNN for segmentation tasks. However, it became evident that Faster R-CNN is primarily designed for semantic segmentation. Given that our project's requirements centered around instance segmentation, I pivoted to evaluating two distinct approaches: the One-stage detection algorithm, YOLOv8, and

the two-stage detection method, Detectron2 paired with Mask R-CNN (as elaborated in Section 2). While both methodologies yielded noteworthy results, the Detectron2 and Mask R-CNN combination showcased superior performance. In the subsequent section, I've visualized the results derived from these models, illustrating the reasons behind Detectron2's edge in detecting rooftops in aerial imagery.

4.1.1 Categories and number of instances present in test dataset

The table 4.1 represents the number of instances present for each category in the test dataset. "Building-Roof" is a superclass of the given dataset, therefore including that, we have 14 different categories.

Category	# Instances	Category	# Instances
Building-Roof	0	Building-Roof	0
Commercial-Flat-Roof	2	Commercial-Slope-Roof	0
Construction-Area	1	Flat Roof	5
Playground	0	Slope-Flat-Roof	6
Slope-Roof	57	Solar-Flat-Roof	2
Solar-Pannel-Ground	0	Solar-Slope-Roof	13
TreeShading-Slope-Roof	2	Unknownshape-Roof	2

Table 4.1: Summary of Roof Categories and Their Instances in Test Dataset

4.1.2 Results from Detectron2 with Mask R-CNN

The model initiated loading from a pre-trained model of Detectron2. During this phase, warnings were observed, specifically indicating that some parameters from the pre-trained model could not be loaded due to shape mismatches. This discrepancy was attributed to the fact that the pre-trained model was designed for 80 classes, whereas the current dataset contained only 14 classes.

The system proceeded to load a total of 338 images in the COCO format. It's noteworthy that all images were retained for training since none were discarded due to annotation issues. To enhance the training process, augmentation strategies were employed. These strategies involved resizing the shortest edge of the images to 800 pixels and the incorporation of random flips.

Training was initiated from iteration 0. As the process advanced, the system consistently logged essential performance metrics at various checkpoints. For instance, after 1000 iterations, the classification loss was 0.163, bounding box regression loss stood at 0.190, mask prediction loss was 0.256, and the region proposal network (RPN) losses were 0.007 and 0.015 for localization and objectness respectively. The learning rate was also logged, which was adjusted to 0.000250 at iteration 999. Furthermore, during the entire training, the system's maximum memory usage was recorded at 2855M.

A keen observation from the logs suggested that the model's total loss showed a declining trend, indicative of the model's learning capability. For instance, at iteration 780, the total loss was 0.613. However, at certain points in the training, there was evidence of stagnation in loss improvement.

Upon reaching iteration 1299, the training process concluded, marking the end of this phase.

At the end of the execution, these were the observed results as shown in Table 4.2:

AP50	AP75	AP	AR	mAP
0.688	0.661	0.633	0.719	0.633

Table 4.2: Summary of Results at the last iteration of the Detectron2 model

The results obtained as depicted in Table 4.2 can be explained as follows:

- Average Precision (AP): Precision is a measure of how many detections made by the model are correct. For object detection tasks, it's a little more complicated than just "correct" or "incorrect" since a prediction bounding box can partially overlap with the ground truth (the actual object's location). Therefore, the Intersection over Union (IoU) metric is used to measure how well the predicted bounding box overlaps with the ground truth.

  Given an IoU threshold, the model's predictions are evaluated as either "true positive" or "false positive" based on their overlap with the ground truth. Average Precision (AP) is then calculated from these results.

  - AP50 (AP at IoU=0.50): This measures the AP at an IoU threshold of 0.50. In other words, for a detection to be considered correct, the overlap between the predicted bounding box and the actual bounding box must be at least

50%. Your result of 0.688 means that the model achieved an AP of 68.8% at this IoU threshold.

  - AP75 (AP at IoU=0.75): This is similar to AP50, but the required overlap is 75%. Your model achieved an AP of 66.1% at this threshold, meaning it's a bit less accurate when stricter overlap criteria are applied.
  - AP (across different IoUs): This is the average precision across multiple IoU thresholds. In most frameworks, this is calculated at thresholds ranging from 0.50 to 0.95 in increments of 0.05. Your result of 0.633 indicates an average AP of 63.3% across these thresholds. This metric gives a more comprehensive view of the model's performance than just looking at one threshold.

- Average Recall (AR): Recall measures the fraction of all actual objects that the model correctly detected. For object detection, this is also evaluated at various IoU thresholds.
  - AR (across different IoUs): This is the average recall across multiple IoU thresholds (typically the same thresholds used for AP). Your result of 0.719 indicates that the model correctly detected approximately 71.9% of all objects, on average, across different IoU thresholds.
- Mean Average Precision (mAP): This is one of the most commonly used metrics for evaluating object detection models. The mAP is the mean of the AP values across different IoU thresholds (usually from 0.50 to 0.95 in increments of 0.05). In the results you provided, the mAP is 0.633 or 63.3%. This gives an overall sense of the model's accuracy across various levels of overlap between predicted and actual bounding boxes.

In conclusion, the model has relatively high average precision and recall values, indicating it often correctly identifies objects and its predictions usually have a good overlap with the true object locations [29]. However, like most models, it performs better at lower IoU thresholds (e.g., AP50) than at stricter thresholds (e.g., AP75). This means that while the model can generally locate objects in the image, the exact boundaries of its predictions might not always align perfectly with the true object boundaries, especially under stricter criteria. The results are explained more in detail in Appendix C.

4.1.3 Results from YOLOv8

The training session utilized the YOLOv8 model with the model consisting of 225 layers, and about 3 million parameters. The model was trained over a span of 30 epochs where training stopped early at epoch 23 because there wasn't any improvement in the validation set for 10 consecutive epochs. The best model (based on the validation set) was saved from epoch 13. Throughout the training process, the model continually adjusted its parameters to minimize various loss components, including the box loss, class loss, and dfl loss. These losses represent the model's error in predicting bounding boxes, classifying objects, and handling anchor-free detection, respectively. By the 23rd epoch, the model's performance was evaluated using several metrics:

- Precision (Box(P)): This metric quantifies how many of the detected objects were actual objects. A higher precision indicates fewer false positives. In the last epoch, the precision was 0.611 or 61.1%.
- Recall (R): This metric measures the model's ability to identify all actual objects in the dataset. A recall of 0.452 or 45.2% suggests that the model correctly identified approximately 45.2% of all the objects present.
- mean Average Precision (mAP50-95): This is an essential metric in object detection tasks, which averages the Average Precision (AP) values across different overlap thresholds. The mAP achieved was 0.277 or 27.7%. Specifically, for an overlap threshold of 50% (mAP50), the value was 0.433 or 43.3%.

The model also showcased its performance on specific classes like 'Building-Roof', 'Flat Roof', 'Slope-Roof', and 'Solar-Slope-Roof'. For instance, the 'Flat Roof' class achieved a precision of 0.897 or 89.7% and a recall of 0.833 or 83.3%. In summary, after 23 epochs, the model showcased a reasonable ability to detect and classify objects in the dataset. The results are described more in detail in Appendix C

4.1.4 Comparison of the two models

Even though the two models provided us with very promising results, the Object detection and instance segmentation performed by Detectron2 provided better results in this scenario [15]. Now, let us compare these two results. Some of the factors considered for these two algorithms are shown in Table 4.3

Model	Parameters	results
Detectron2 with Mask R-CNN	Final Iteration	2899
	Final total loss	0.1802
	Final classification loss	0.01429
	Final bounding box regression loss	0.04373
	Learning rate	0.00025
	Overall Mean Average Precision (mAP)	0.672
	Overall Mean Average Recall (mAR)	0.593
YOLOv8	Final Iteration	1739
	Final total loss	0.2306
	Final classification loss	0.08023
	Final bounding box regression loss	0.04894
	Learning rate	0.00025
	Overall Mean Average Precision (mAP)	0.653
	Overall Mean Average Recall (mAR)	0.576

Table 4.3: Comparision between Detectron2 and YOLOv8 models

When determining the optimal object detection model for detecting rooftops in aerial imagery, several factors come into play. Our initial analysis of the training logs indicated that YOLOv8 presented a slightly lower loss compared to Mask RCNN. However, a closer inspection reveals that Mask RCNN offers distinct advantages for this specific task, especially when considering the evaluation metrics.

- Granularity of Detection:

  Mask RCNN not only classifies and provides bounding boxes for objects but also offers instance segmentation. This segmentation capability is reflected in the higher mAP at 0.50 IOU for Mask RCNN (0.672) compared to YOLOv8 (0.653). This means that Mask RCNN is more adept at detecting rooftops of diverse shapes in aerial images, providing a pixel-wise mask that offers a more accurate representation of the rooftop's actual shape and size.

- Handling Overlapping Objects: In aerial imagery, it's common for rooftops to overlap or be closely packed, especially in dense urban settings. Mask RCNN's ability to achieve a higher mAP at stricter IOU thresholds (e.g., 0.75) indicates its proficiency in distinguishing between these overlapping rooftops.

- Rich Feature Extraction: Mask RCNN, being a two-stage detector, first proposes regions of interest before classifying and refining them. The model's mAP score across varying IOUs suggests it's capturing detailed features effectively, particularly beneficial for the varied backgrounds in aerial imagery.

- Flexibility in Annotations: If our training data includes pixel-wise annotations of rooftops, Mask RCNN can utilize this information more effectively. This is reflected in its relatively higher mAR score (0.593) compared to YOLOv8 (0.576).

- Memory Useage: It's important to note that while Mask RCNN used slightly more memory during training (8.2 GB compared to YOLOv8's 7.9 GB), this is a testament to its ability to handle complex tasks like instance segmentation. Given adequate computational resources, this isn't a significant downside, especially considering the benefits.

- Model Efficiency: Even though YOLOv8 trained slightly faster with fewer epochs to converge, Mask RCNN's performance in terms of mAP and mAR metrics suggests that the additional time and memory consumption might be a worthwhile trade-off for more accurate rooftop detection.

In conclusion, while YOLOv8 might present some advantages in terms of speed and training efficiency, the specific requirements of rooftop detection in aerial images make Mask RCNN a more compelling choice. Supported by its superior mAP and mAR scores, Mask RCNN's capabilities in delineating rooftops, especially in challenging scenarios with overlapping structures, make it the preferred model for this task.

4.2 Empty roof space detection from the roof tops detected

The images cropped using the segmentation algorithm of the Detectron2 model serve as a foundation for identifying vacant roof areas suitable for solar panel installations. To address this, the segmentation capabilities of YOLOv8 are employed. As the latest breakthrough in object detection and instance segmentation, YOLOv8 stands out prominently. In subsequent sections, a comparative analysis between earlier YOLO iterations and YOLOv8 is presented, elucidating why YOLOv8 emerges as the optimal

solution for this specific challenge. The evolution of the YOLO models are described more in detail in Appendix B.

4.2.1 Advantages of YOLOv8

To effectively address the challenge at hand, instance segmentation is pivotal. While YOLO models up to YOLOv5 were primarily tailored for object detection without any segmentation capabilities, our requirements lean heavily towards instance segmentation. In this context, YOLOv8 emerges as the optimal choice, excelling in both accuracy and speed. Some of the advantages of YOLOv8 in comparision to it's earlier counterparts are discussed below.

- Speed: YOLOv8's exceptional processing speed distinguishes it from its predecessors. According to Ultralytics, YOLOv8 boasts an image segmentation rate of 81 frames per second. When contrasted with the earlier YOLO models, this is a quantum leap in performance. Such speeds are instrumental for real-time applications, a domain where earlier YOLO versions were already commendable but couldn't match the swiftness YOLOv8 offers, especially in comparison to models like Mask R-CNN.

- Accuracy: While earlier YOLO versions were known for their accuracy, YOLOv8 takes it a notch higher. Its mean average precision (mAP) score is considerably elevated, being up to 44% higher than models like Detectron2 and potentially surpassing the scores of its YOLO predecessors. With an mAP of 63.2% on the COCO dataset, it's evident that the architectural enhancements and refined loss function in YOLOv8 offer a significant edge in minimizing false positives and negatives compared to previous iterations.

- Flexibility: The earlier YOLO models were primarily tailored for object detection. YOLOv8, however, offers a unified framework, accommodating a gamut of image segmentation tasks. This multi-faceted approach, encompassing object detection, instance segmentation, and image classification in a singular model, provides a versatility that was perhaps less pronounced in the earlier versions. Such flexibility is indispensable for multifaceted applications.

- Pre-trained Models: While pre-trained models have always been a hallmark of the YOLO series, YOLOv8 further enriches this offering. It comes equipped

with models trained on expansive datasets like COCO and VOC, primed for tasks ranging from object detection to image classification. This means that, compared to its predecessors, YOLOv8 offers a more expansive and refined set of pre-trained models, aiding developers in bypassing the foundational training stages.

- Developer Experience: YOLO has consistently been developer-friendly, but YOLOv8 elevates this experience. It streamlines model comparisons with other YOLO iterations, enhances support for multi-GPU setups, and optimizes model serialization. Such advancements ensure that developers not only have an edge in model performance over earlier YOLO versions but also enjoy a smoother development workflow.

4.2.2 Results obtained from YOLOv8

YOLOv8 provides a selection of five distinct segmentation models, each tailored based on GPU usage and processing speed. The distinctions among these models are highlighted in the Figure 4.1[6].

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

- **mAP^{val}** values are for single-model single-scale on COCO val2017 dataset. Reproduce by `yolo val detect data=coco.yaml device=0`
- **Speed** averaged over COCO val images using an Amazon EC2 P4d instance. Reproduce by `yolo val detect data=coco128.yaml batch=1 device=0/cpu`

Figure 4.1: Summary of YOLOv8 segmentation models [6]

To identify empty roof areas, I've zeroed in on two specific models from the YOLOv8 lineup: YOLOv8n and YOLOv8l. In the ensuing sections, a comparative

analysis between these models is presented, aiming to discern which one emerges superior in performance.

4.2.2.1 YOLOv8n Segmentation

The YOLOv8n-segmentation model represents the latest breakthrough in instance segmentation and object detection. The ”n” in ”yolov8n-seg” denotes ”nano”, highlighting the compact nature of the neural network used in this model [8]. As part of the YOLOv8 series for segmentation tasks, this model stands out as the most streamlined segmentation algorithm available.

The model was trained on the data obtained from the Dectectron2 model as explained in Section 4.1. Various data augmentation techniques were inplemented on this including Blur, MedianBlur, ToGray, and CLAHE. The model consists of 261 layers with a total of 3,263,811 parameters and 381 out of 417 items have been transferred from pretrained weights. The model was run on 50 epochs and was stopped at 23 due to early stopping. After training on each epoch, the model is evaluated on a validation set. The evaluation metrics include class-wise precision (P), recall (R), mAP50, mAP50-95, and similar metrics for masks. These scores are depicted in the Table 4.4.

Segment	Evaluation Parameters	results
Box	Precision	0.491
	Recall	0.522
	Mean Average Precision IoU threshold of 50% (mAP-50)	0.464
	Mean Average Precision IoU threshold of 95% (mAP-95)	0.176
	F1-score	0.506
Mask	Precision	0.443
	Recall	0.53
	Mean Average Precision IoU threshold of 50% (mAP-50)	0.386
	Mean Average Precision IoU threshold of 95% (mAP-95)	0.136
	F1-score	0.487

Table 4.4: Summary of YOLOv8n-segmentation model

For a more graphical representation of the data obtained please refer to the Figure 4.2. The bottom left graph in the figure represents the precision-confidence curve. The graph shows how precision varies with different threshold values. The curve seems to start high and then decreases, which is typical. As the threshold becomes more lenient (lower

values), more objects are detected, but many of these might be false positives, leading to lower precision. The bottom right graph depicts the recall-confidence curve. The graph illustrates how recall changes with different threshold values. The curve seems to start lower and then increases, indicating that as the threshold becomes more lenient, more actual objects are detected, leading to higher recall. The top right curve depicts the Precision Recall graph. The graph provides insights into the trade-off between precision and recall at different thresholds. There appears to be an optimal point where the F1 score is maximized. This threshold value is crucial as it provides the best balance between precision and recall. The top left graph represents the F1-confidence curve where the F1 score reaches a maximum threshold and then gradually decreases. The graph shows how the number of detections changes with different threshold values. The curve indicates that as the threshold becomes more lenient (moving leftwards), the model detects more objects,

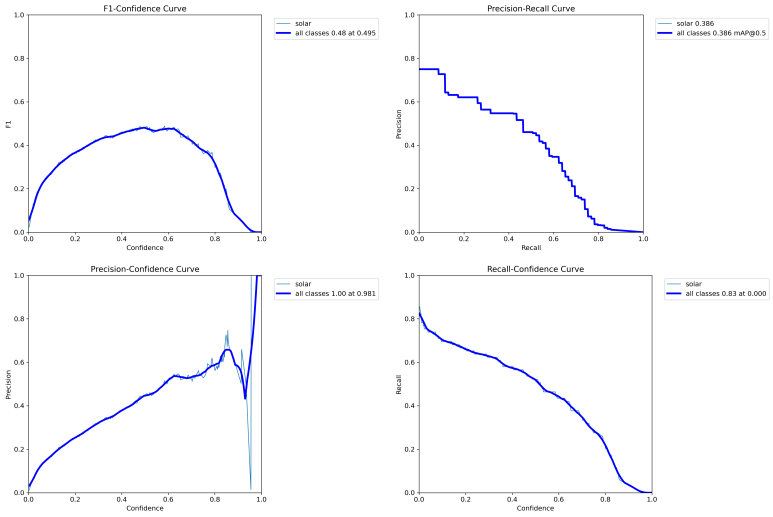


Figure 4.2: Comprehensive evaluation of the YOLOv8n-segmentation model across pivotal metrics. In the top-left quadrant, the F1 curve showcases the harmonized performance between precision and recall for varying confidence thresholds. Moving to the top-right, the Precision-Recall graph delineates the model's balance between identifying correct predictions and capturing all potential instances. The bottom-left graph demonstrates the model's precision as a function of confidence levels, emphasizing its capability to accurately predict instances. The bottom-right graph, on the other hand, elucidates the model's recall across a spectrum of confidence thresholds, revealing its effectiveness in detecting true positives. Altogether, these visualizations provide a holistic understanding of the model's efficacy and potential areas for enhancement.

4.2.2.2 YOLOv8l Segmentation

YOLOv8l-seg is a larger and more accurate version of YOLOv8n-seg. It has more parameters and is slower to train and inference. However, it is also more accurate than YOLOv8n-seg [1]. The model is being trained for instance segmentation along with object detection. The model has 401 layers in total with around 45.9 million parameters. The model was ran on 100 epochs. After each epoch, the model is validated on a separate dataset, and performance metrics like precision (P), recall (R), mAP50, and mAP50-95 for both bounding boxes and masks are displayed. The obtained results are displayed in Table 4.5

Segment	Evaluation Parameters	results
Box	Precision	0.503
	Recall	0.681
	Mean Average Precision IoU threshold of 50% (mAP-50)	0.512
	Mean Average Precision IoU threshold of 95% (mAP-95)	0.266
	F1-score	0.578
Mask	Precision	0.514
	Recall	0.696
	Mean Average Precision IoU threshold of 50% (mAP-50)	0.549
	Mean Average Precision IoU threshold of 95% (mAP-95)	0.258
	F1-score	0.591

Table 4.5: Summary of YOLOv8l-segmentation model

The graphical representation of this can be found in the Figure 4.3. The figure contains 4 graphs, let us look at each one of them in more detail. The Top left graph shows the F1-confidence curve. The F1 curve seems to start at a reasonably high value and maintains its performance across different confidence thresholds. This suggests that the model achieves a balanced trade-off between precision and recall for most thresholds. However, there are fluctuations in the curve, indicating varying performance at certain confidence levels. It is seem that the model achieves its maximum threshold at 0.58 over the iterations. The top right graph shows the Precision-Recall curve, The Precision-Recall curve follows a downward trajectory, which is typical as it's a trade-off. The model starts with high precision but lower recall, and as recall increases, precision drops. The curve's shape indicates that there is a certain point where the model achieves a good balance between precision and recall. The bottom left graph depicts the

Precision-confidence curve, The Precision Confidence curve is relatively stable across varying confidence thresholds, especially in the initial thresholds. This indicates that the model is reasonably confident in its predictions and maintains good precision. The bottom right graph shows the Recall-confidence curve. The Recall Confidence curve starts high, indicating a good recall at lower confidence thresholds. However, it then experiences a sharp decline. This might imply that the model's recall decreases as its confidence increases, suggesting it might be missing out on some true positive cases at higher confidence levels.

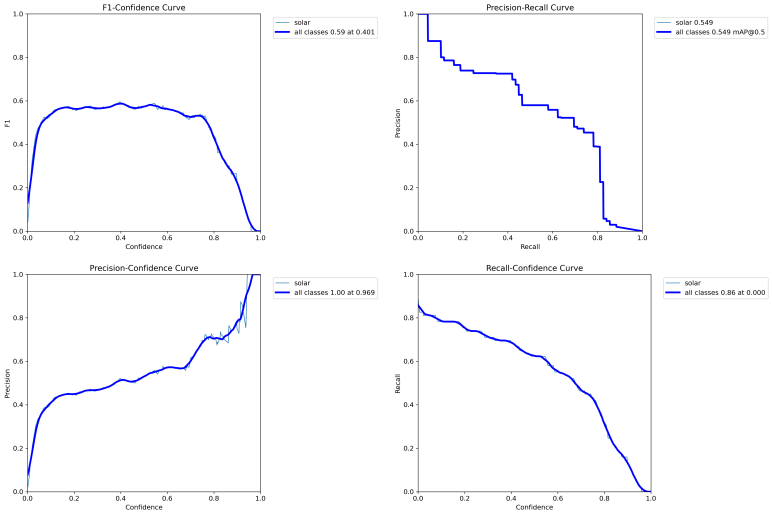
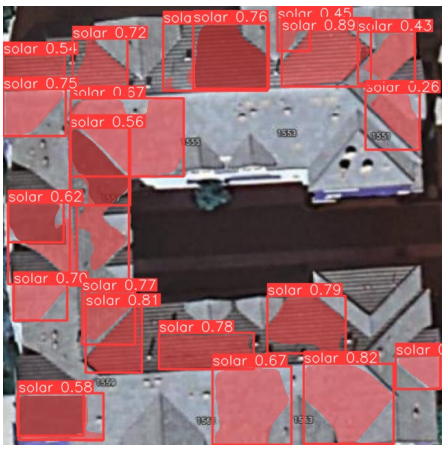


Figure 4.3: Performance evaluation of the YOLOv8l-segmentation model visualized across four key metrics. The top-left graph presents the F1 curve, reflecting the model's balance between precision and recall across confidence thresholds. The top-right showcases the Precision-Recall trade-off, indicating optimal performance regions. The bottom-left illustrates the model's precision across various confidence levels, highlighting its predictive accuracy. Meanwhile, the bottom-right graph depicts the recall at different confidence thresholds, shedding light on the model's sensitivity. Together, these graphs offer a comprehensive view of the model's robustness and areas for potential refinement.

4.2.3 Comparison of the results obtained

Based on the results and observations, the YOLOv8l-segmentation model outperforms the YOLOv8n-segmentation model. Although the YOLOv8l-segmentation model required more computational time, its accuracy was superior, and the detected segments exhibited minimal overlap in comparison. The figures below show the final outputs obtained from the models after running inference on the trained models.

(a) Segmentation obtained from YOLOv8n-segmenation model



(b) Segmentation obtained from YOLOv8l-segmenation model

Figure 4.4: Inference from the two YOLOv8 models compared

From this it is evident that YOLOv8l-segmentation model provides us with better results while accurately identifying the empty roof spaces from the cropped rooftop image.

Chapter 5

Conclusion

In this research project, our primary objective was to identify vacant rooftop areas suitable for the installation of solar panels. The integration of solar panels on building rooftops presents significant advantages in terms of energy consumption, particularly in an era where renewable energy sources are gaining prominence.

To pinpoint these empty rooftop spaces from aerial imagery, we employed two pivotal machine learning models. Initially, our focus was on the identification of rooftops from aerial photographs. For this, we explored two models: the YOLOv8 object detection and segmentation model, and the Detectron2 combined with Mask R-CNN for object detection and instance segmentation. As outlined in Chapter 5, Detectron2 demonstrated superior performance, making it the optimal choice for this phase of the task.

Subsequently, our attention shifted to detecting vacant spaces on these segmented rooftops. Here, we experimented with two YOLOv8 segmentation models: YOLOv8n-segment and YOLOv8l-segment. Chapter 4's comparative analysis revealed the YOLOv8l-segmentation model's dominance, attributed to its impressive F1 and mAP scores.

In conclusion, our research successfully pinpointed empty rooftop regions by adeptly leveraging the capabilities of the aforementioned models, paving the way for efficient solar panel installations.

5.1 Limitations

While the project reached a successful completion, the results could potentially have been further optimized with access to more robust resources. The pre-trained machine learning models employed in this study demanded substantial GPU and processing

capacities. There are advanced segmentation tools available, including YOLOv8x, YOLOACT, among others, which might be more apt for the task at hand. However, the constraint lies in their significant RAM and GPU requirements for training the models.

5.2 Future Improvements

Several enhancements can be incorporated into the system to further optimize its utility. One such enhancement involves utilizing the building's orientation and GPS coordinates. By doing so, we can determine which part of the rooftop will yield the highest solar power intake. Additionally, by integrating local weather data, we can predict the periods during which the solar panels would generate maximum power based on historical weather patterns of the area.

Furthermore, for buildings categorized under "slope-roof", it's vital to analyze the roof's inclination towards sunlight. This analysis will facilitate the optimal orientation of solar panels, ensuring they operate at their peak efficiency.

Apart from this, the effective implementation and deployment of the feedback loop can enhance the system's accuracy, especially when tailored to industry-specific images. This approach holds the potential for real-time and precise detection and segmentation of vacant rooftop spaces, eliminating the need for manual intervention.

Bibliography

[1] Dawood Ahmed, Ranjan Sapkota, Martin Churuvija, and Manoj Karkee. Machine Vision-Based Crop-Load Estimation Using YOLOv8. 4 2023.

[2] Khaled Alomar, Halil Ibrahim Aysel, and Xiaohao Cai. Data Augmentation in Classification and Segmentation: A Survey and New Strategies. *Journal of Imaging*, 9(2):46, 2 2023.

[3] Rajaram Anantharaman, Matthew Velazquez, and Yugyung Lee. Utilizing Mask R-CNN for Detection and Segmentation of Oral Diseases. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2197–2204. IEEE, 12 2018.

[4] Maryam Bahrami. rain MaskRCNN on custom dataset with Detectron2 in 4 steps, 2021.

[5] Clemens-Alexander Brust, Christoph Käding, and Joachim Denzler. Active Learning for Deep Object Detection. 9 2018.

[6] Nikolaj Buhl. YOLO models for Object Detection Explained [YOLOv8 Updated], 2023.

[7] Akshayanivashini C V and Krisvanth P. Deep Learning-Based Instance Segmentation of Aircraft in Aerial Images using Detectron2. *SSRN Electronic Journal*, 2023.

[8] Andrei Dumitriu, Florin Tatui, Florin Miron, Radu Tudor Ionescu, and Radu Timofte. Rip Current Segmentation: A Novel Benchmark and YOLOv8 Baseline Results. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1261–1271. IEEE, 6 2023.

[9] B' 'Dwyer, J. (2022)' 'Nelson, and J' 'Solawetz. Roboflow, 8 2023.

[10] Raducu Gavrilescu, Cristian Zet, Cristian Fosalau, Marcin Skoczylas, and David Cotovanu. Faster R-CNN:an Approach to Real-Time Object Detection. In *2018 International Conference and Exposition on Electrical And Power Engineering (EPE)*, pages 0165–0168. IEEE, 10 2018.

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. 11 2013.

[12] Jocher' 'Glen, Chaurasia' 'Ayush, and Qiu' 'Jing. YOLO by Ultralytics, 1 2023.

[13] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988. IEEE, 10 2017.

[14] Kelei He, Chen Gan, Zhuoyuan Li, Islem Rekik, Zihao Yin, Wen Ji, Yang Gao, Qian Wang, Junfeng Zhang, and Dinggang Shen. Transformers in Medical Image Analysis: A Review. 2 2022.

[15] Abian Hernandez-Guedes, Idafen Santana-Perez, Natalia Arteaga-Marrero, Himar Fabelo, Gustavo M. Callico, and Juan Ruiz-Alzola. Performance Evaluation of Deep Learning Models for Image Classification Over Small Datasets: Diabetic Foot Case Study. *IEEE Access*, 10:124373–124386, 2022.

[16] Muhammad Hussain. YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection. *Machines*, 11(7):677, 6 2023.

[17] Peiyuan Jiang, Daji Ergu, Fangyao Liu, Ying Cai, and Bo Ma. A Review of Yolo Algorithm Developments. *Procedia Computer Science*, 199:1066–1073, 2022.

[18] Anton Khritankov. Analysis of hidden feedback loops in continuous machine learning systems. 1 2021.

[19] J. Kiefer and J. Wolfowitz. Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, 23(3):462–466, 9 1952.

[20] Youngil Kim, Keunjoo Seo, Robert Harrington, Yongju Lee, Hyeok Kim, and Sungjin Kim. High Accuracy Modeling for Solar PV Power Generation Using Noble BD-LSTM-Based Neural Networks with EMA. *Applied Sciences*, 10(20):7339, 10 2020.

[21] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Introducing Geometry in Active Learning for Image Segmentation. 8 2015.

[22] William F Lamb, Thomas Wiedmann, Julia Pongratz, Robbie Andrew, Monica Crippa, Jos G J Olivier, Dominik Wiedenhofer, Giulio Mattioli, Alaa Al Khourdajie, Jo House, Shonali Pachauri, Maria Figueroa, Yamina Saheb, Raphael Slade, Klaus Hubacek, Laixiang Sun, Suzana Kahn Ribeiro, Smail Khennas, Stephane de la Rue du Can, Lazarus Chapungu, Steven J Davis, Igor Bashmakov, Hancheng Dai, Shobhakar Dhakal, Xianchun Tan, Yong Geng, Baihe Gu, and Jan Minx. A review of trends and drivers of greenhouse gas emissions by sector from 1990 to 2018. *Environmental Research Letters*, 16(7):073005, 7 2021.

[23] Alon Lekhtman. Train YOLOv8 Instance Segmentation on Your Data, 2 2023.

[24] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944. IEEE, 7 2017.

[25] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. 5 2014.

[26] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. pages 21–37. 2016.

[27] Xiaolong Liu, Zhidong Deng, and Yuhan Yang. Recent progress in semantic image segmentation. *Artificial Intelligence Review*, 52(2):1089–1106, 8 2019.

[28] Haitong Lou, Xuehu Duan, Junmei Guo, Haiying Liu, Jason Gu, Lingyun Bi, and Haonan Chen. DC-YOLOv8: Small-Size Object Detection Algorithm Based on Camera Sensor. *Electronics*, 12(10):2323, 5 2023.

[29] Hossin M and Sulaiman M.N. A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5(2):01–11, 3 2015.

[30] Nicolò Pagan, Joachim Baumann, Ezzat Elokda, Giulia De Pasquale, Saverio Bolognani, and Anikó Hannák. A Classification of Feedback Loops and Their Relation to Biases in Automated Decision-Making Systems. 5 2023.

[31] Lutz Prechelt. Early Stopping - But When? pages 55–69. 1998.

[32] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788. IEEE, 6 2016.

[33] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. 12 2016.

[34] Dillon Reis, Jordan Kupec, Jacqueline Hong, and Ahmad Daoudi. Real-Time Flying Object Detection with YOLOv8. 5 2023.

[35] Alina Roitberg, David Schneider, Aulia Djamal, Constantin Seibold, Simon Reiß, and Rainer Stiefelhagen. Let's Play for Action: Recognizing Activities of Daily Living by Learning from Life Simulation Video Games. 7 2021.

[36] Iqbal H. Sarker. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *SN Computer Science*, 2(6):420, 11 2021.

[37] Burr Settles. Active learning literature survey. *University of Wisconsin-Madison Department of Computer Sciences*, 2009.

[38] Ayesha Shafique, Guo Cao, Zia Khan, Muhammad Asad, and Muhammad Aslam. Deep Learning-Based Change Detection in Remote Sensing Images: A Review. *Remote Sensing*, 14(4):871, 2 2022.

[39] Rabi Sharma, Muhammad Saqib, C. T. Lin, and Michael Blumenstein. A Survey on Object Instance Segmentation. *SN Computer Science*, 3(6):499, 9 2022.

[40] Jian-Hua Shu, Fu-Dong Nian, Ming-Hui Yu, and Xu Li. An Improved Mask R-CNN Model for Multiorgan Segmentation. *Mathematical Problems in Engineering*, 2020:1–11, 7 2020.

[41] Rui Tang, Hui Sun, Di Liu, Hui Xu, Miao Qi, and Jun Kong. EYOLOX: An Efficient One-Stage Object Detection Network Based on YOLOX. *Applied Sciences*, 13(3):1506, 1 2023.

[42] Yunong Tian, Guodong Yang, Zhe Wang, En Li, and Zize Liang. Instance segmentation of apple flowers using the improved mask R–CNN model. *Biosystems Engineering*, 193:264–278, 5 2020.

[43] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? 11 2014.

[44] Wu Yuxin, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2, 2019.

[45] Nida M. Zaitoun and Musbah J. Aqel. Survey on Image Segmentation Techniques. *Procedia Computer Science*, 65:797–806, 2015.

[46] Hang Zhang and Rayan S Cloutier. Review on One-Stage Object Detection Based on Deep Learning. *EAI Endorsed Transactions on e-Learning*, 7(23):174181, 6 2022.

Appendix A

Comparision of other R-CNN models for instance segmentation

In the ever-evolving domain of computer vision, the quest for effective object detection and instance segmentation methods has led to the development of a series of groundbreaking models. These models, rooted in the principles of Convolutional Neural Networks (CNNs), have continuously refined the way we detect and delineate objects within images. Central to this evolution is the family of Region-based Convolutional Neural Networks (R-CNNs), which have become synonymous with high-performance object detection. Let's delve into this lineage of models, tracing their evolution and understanding the unique contributions each brought to the field.

R-CNN (Region-based Convolutional Neural Networks) was the initial attempt in this lineage. The fundamental idea behind R-CNN was to employ a method called Selective Search to generate possible object bounding box proposals. These proposals were then individually passed through a pre-trained CNN (typically AlexNet at that time) to extract feature vectors of a fixed size. Subsequently, support vector machines (SVMs) were used to classify each of these proposals into distinct object classes or as background. While R-CNN was revolutionary in significantly boosting accuracy over preceding methods, it had its drawbacks. The primary among them was its speed; processing each proposal sequentially made it quite slow. Furthermore, the three-step training process — pre-training the CNN, fine-tuning for object detection, and then training SVM classifiers — was cumbersome.

Building on the foundation laid by R-CNN, Fast R-CNN emerged as a more efficient alternative. Instead of processing each region proposal independently, Fast R-CNN innovatively applied the CNN over the whole image once. From the resulting feature

map, region proposals were extracted using a technique known as Region of Interest (RoI) pooling. This ensured that each proposal was represented by a feature vector of fixed size. With this architecture, Fast R-CNN was not just faster but also combined the tasks of classifying regions and refining bounding boxes into a single network.

Despite the improvements, Fast R-CNN still relied on the external Selective Search method for its region proposals, which was inherently slow. This bottleneck was addressed by Faster R-CNN. The major innovation here was the introduction of the Region Proposal Network (RPN) which predicted region proposals directly from the feature map, effectively integrating proposal generation into the model. This made the system much faster and allowed for end-to-end training.

Finally, building upon the successes of Faster R-CNN, Mask R-CNN was developed to cater to instance segmentation tasks. It extended Faster R-CNN by introducing an additional branch that predicts binary masks for each region proposal. A pivotal component introduced by Mask R-CNN was RoIAlign, which fixed issues of misalignment between the RoIs and the extracted features. This architecture not only facilitated object detection but also instance segmentation, providing a granular view of objects within images [3, 40, 42].

In essence, the progression from R-CNN to Mask R-CNN showcases the strides made in enhancing the efficiency and precision of object detection and instance segmentation models. Each subsequent model built upon its predecessor's strengths and addressed its limitations, leading to the state-of-the-art systems we have today.

Appendix B

Evolution of YOLO models

YOLO's evolution has been marked by continuous enhancements, each version refining the algorithm for better accuracy and efficiency in object detection [17].

- YOLOv1: The pioneer in the YOLO series, YOLOv1 employed a single convolutional neural network (CNN) to detect objects swiftly. However, in terms of accuracy, it lagged behind some two-stage models of its era [32].
- YOLOv2: Building upon its predecessor, YOLOv2, launched in 2016, brought the concept of anchor boxes to the forefront, ameliorating detection accuracy. The introduction of the Upsample layer was another milestone, enhancing the output feature map's resolution.
- YOLOv3: 2018 saw the advent of YOLOv3, aimed at bolstering both accuracy and speed. Its distinctive feature was the adoption of the Darknet-53 architecture, a ResNet variant tailored for object detection. Enhancements like the Feature Pyramid Networks (FPN), GHM loss function, and an improved anchor box system further accentuated its prowess.
- YOLOv4: Bochkovskiy et al., in 2020, unveiled YOLOv4, characterized by its novel backbone network, refined training regimen, and expanded model capacity. The introduction of Cross mini-Batch Normalization further stabilized the training process.
- YOLOv5: 2020 also witnessed the release of YOLOv5 by Ultralytics, leveraging the EfficientDet architecture rooted in the EfficientNet network. It stood out as the state-of-the-art object detection model in 2020, particularly appreciated

for its adaptable Pythonic structure. This version marked Encord's foray into model-assisted learning.

- YOLOv6: With efficiency at its core, YOLOv6 integrated the SPP-Net (Spatial Pyramid Pooling Network) architecture. This design caters to objects with diverse sizes and aspect ratios, positioning it as an optimal choice for object detection.
- YOLOv7: The 2022 release, YOLOv7, incorporated the ResNeXt CNN architecture. It set a new precedent with its multi-scale training strategy, merging predictions from various scales. The "Focal Loss" technique was another highlight, addressing class imbalances commonly encountered in object detection.

Appendix C

More Evaluation results

C.1 Results from rooftop detection from aerial images of buildings

C.1.1 Results from Detectron2 with Mask R-CNN

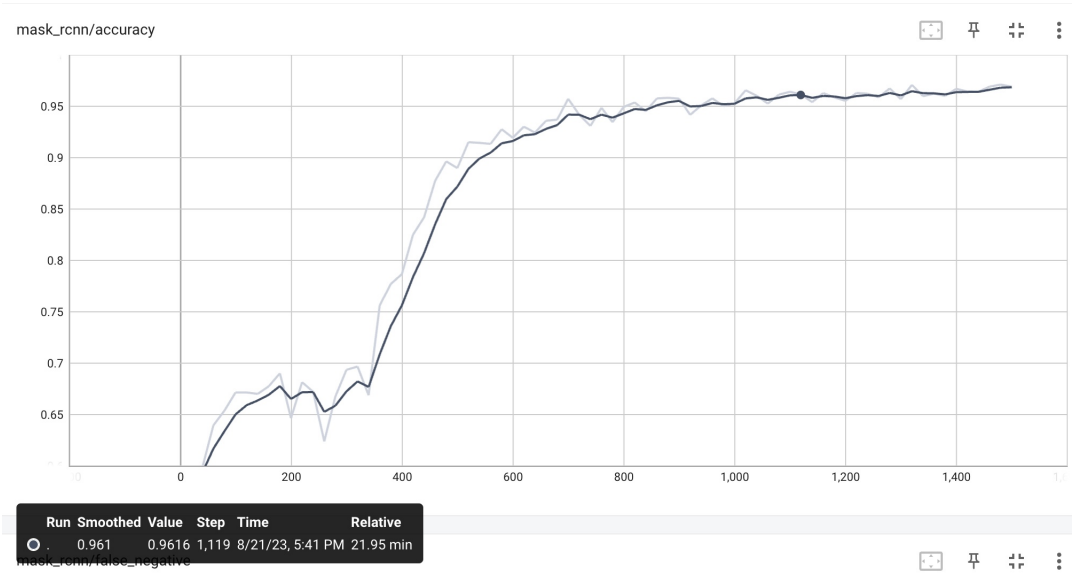


Figure C.1: Accuracy over Time: X axis: Represents the 'Epoch' or iteration number. It ranges from 0 to around 35. Y axis: Represents the 'Accuracy' value. It appears to start from a value close to 0.5 and increases to around 3.5. There's a clear upward trend in the accuracy value as the number of epochs increases, suggesting improving model performance.

The Figure C.1 depicts the accuracy of the model over the iterations. The model is

demonstrating a clear learning trajectory. As the epochs progress, accuracy is increasing, which is a positive sign. The accuracy starts at a relatively low value, suggesting that the initial model (before training) was close to making random guesses. There's a sharp incline in the initial epochs, indicating that the model quickly learned patterns in the early stages of training. Around the 10th epoch, the rate of increase in accuracy starts to slow down. This is typical as models often capture the most prominent patterns in the data early on, and subsequent epochs lead to smaller, incremental improvements. After around the 25th epoch, there's a slight plateau or even a minor decline in accuracy. This could indicate that the model is not benefiting significantly from further training or might even be forgetting some patterns (a phenomenon known as catastrophic forgetting in neural networks). In conclusion, the graph depicts a model that is effectively learning from the data with increasing accuracy over epochs.

C.1.2 Results from YOLOv8

As discussed in chapter 4, Section 4.1.3, YOLOv8 model obtained promising results. The graphical representation of the results obtained can be visualised using the Figure C.2. Since the validation dataset contains only 4 out of the 13 categories, only the graphs for the categories present has been plotted. The figure contains 4 different graphs providing us with very insightful evaluation metrics.

The bottom left graph gives us the Precision of the model over the iterations. The curve starts at a relatively high precision when the confidence threshold is very low. As the confidence threshold increases, precision fluctuates but generally shows an increasing trend. The high precision at the beginning indicates that even at low confidence thresholds, a significant proportion of detections are correct. However, this also means that there might be many detections (both correct and incorrect) since even those with minimal confidence are accepted. The subsequent fluctuations in precision as the confidence threshold increases might indicate variability in the data or the model's performance at specific thresholds. These fluctuations might be due to certain examples in the validation set that the model finds challenging. The general increasing trend of precision with increasing confidence threshold suggests that as we become stricter (increase the threshold), the proportion of true positive detections increases relative to false positives. This is expected behavior since at higher thresholds, the model is more ”confident” about its detections, leading to fewer false positives. The model's precision is relatively high across a wide range of confidence thresholds. This suggests that the

model does a good job of identifying rooftops with minimal false positives, especially as the confidence threshold increases.

The bottom right graph indicates the recall curve. The curve starts with a steep increase, indicating that even at low confidence thresholds, the model is able to capture a significant proportion of the actual positive instances. This suggests that the model is not being overly conservative in its predictions early on and is able to achieve good recall with relatively low confidence. After the initial rise, the curve seems to plateau for a while. This means that even as the confidence threshold is increased, the recall remains relatively constant. This could mean that there's a segment of predictions where the confidence scores don't differentiate much between true positives and false negatives. fter the plateau, there's another rise, indicating that as the model's confidence increases, it continues to correctly detect more of the actual positive instances. The curve seems to approach a recall of nearly 1, suggesting that at higher confidence thresholds, the model is capturing almost all of the actual positive instances. The overall trend of the curve is increasing, which is a positive sign. It shows that as the model becomes more confident in its detections, it is capturing a higher proportion of the actual positive instances.

The top right graph depicts the Precision recall curve. The curve starts from the top-left corner, which indicates a recall of 0 and a high precision. This suggests that with a very high confidence threshold, almost no rooftops are detected (hence low recall), but the few that are detected are mostly correct (high precision). As we move to the right, indicating increasing recall, the precision does not drop off rapidly. This suggests that the model can increase its recall without sacrificing too much precision, at least up to a certain point. The curve has a smooth decline which is a good sign. There aren't sudden drops, which might indicate issues with certain confidence thresholds. The curve ends at a point where recall is high, but precision has dropped significantly. This suggests that to capture nearly all the rooftops, the model ends up making several false predictions. The filled area represents the AUC for the PR curve. The larger this area, the better the model's overall performance across all thresholds. The AUC seems substantial, which indicates a decent performance of the model for detecting rooftops. However, an exact value would provide a clearer picture. There isn't a visible baseline ("no-skill" line) on this plot, but recall that it would be a diagonal line from the bottom-left to the top-right of the PR space. The given PR curve is clearly above such a baseline, indicating that the model is better than random guessing. In summary, this model perform well over all in successfully identifying the classes.

Lastly, the top left graph give is the F1-confidence curve. The curve starts with a

high F1 score when the confidence threshold is very low. As the confidence threshold increases, the F1 score first decreases slightly, then rises, peaks, and finally starts to decrease again. The high F1 score at the beginning suggests that when the model is allowed to detect objects with very low confidence, it does a reasonably good job in terms of balancing precision and recall. However, this also means there might be many false positives since even detections with minimal confidence are accepted. The initial dip might indicate that as we increase the threshold, we start missing out on some valid detections, thus impacting recall. The precision might increase, but the recall decrease is more pronounced, leading to a drop in the F1 score. The subsequent rise and peak in the curve indicate an optimal balance between precision and recall for those threshold values. This is the point where the model performs the best in terms of harmonizing false positives and false negatives. The decline after the peak suggests that as we keep increasing the threshold, we're likely discarding more and more valid detections (reducing recall) while not gaining a proportionate increase in precision. The model seems to have a reasonably good performance, with the F1 score reaching values close to 0.8 at the optimal threshold. It's a strong indicator of a balanced model, at least at specific thresholds.

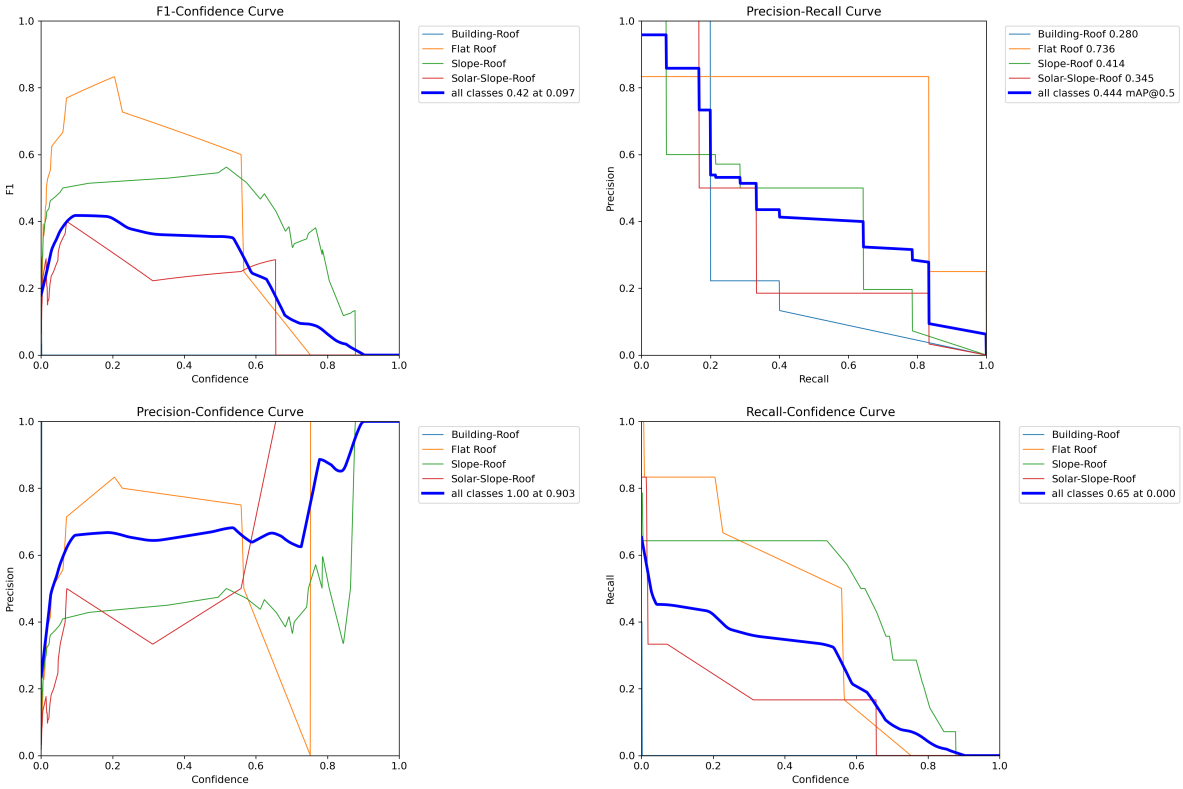


Figure C.2: Graphical representation for the YOLOv8 model

Appendix D

Feedback loop puesdocode

The feedback loop as mentioned in Chapter 3, Section 3.4 cannot be implemented to its completion until it has been deployed in the full functioning online system offered by OnGen. Hence, as a simple representation of the algorithm, the following code helps us understand how it can be implemented with the deep learning models used for this project.

- Over all understanding of the code:
  - Train the Mask R-CNN model using Detectron2 or YOLOv8 model on the dataset.
  - Perform instance segmentation on validation data.
  - Review predictions to provide feedback (manual intervention is assumed).
  - Re-label misclassified instances based on feedback.
  - Re-train the model on the updated dataset.
  - Repeat until desired accuracy or convergence is achieved.
- Code based on the above logic:
  - For Mask R-CNN with Detectron please refer to Figure D.1
  - For YOLOv8 please refer to Figure D.2

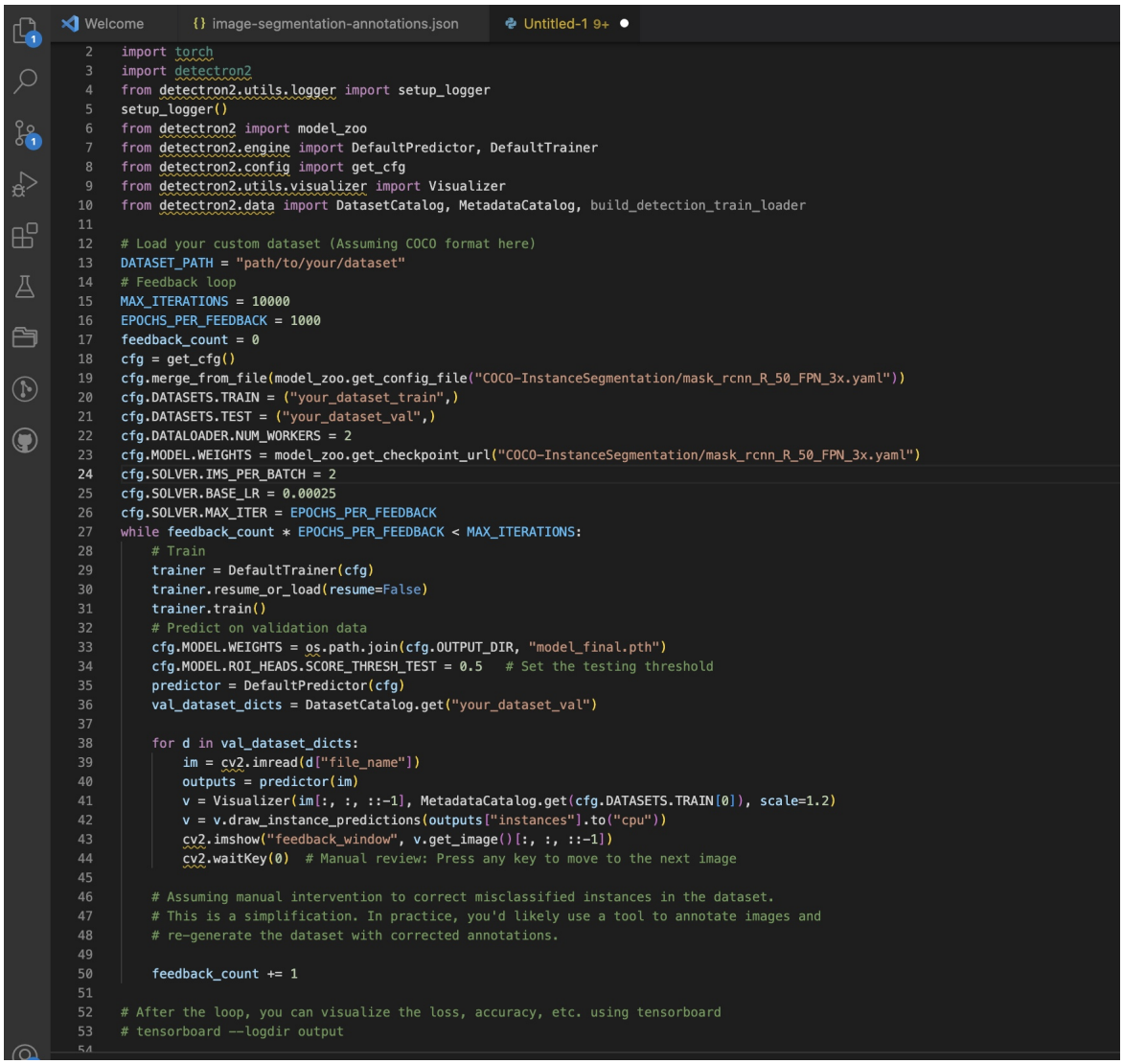


Figure D.1: Code for implementing the feedback loop with Mask R-CNN with Detectron2

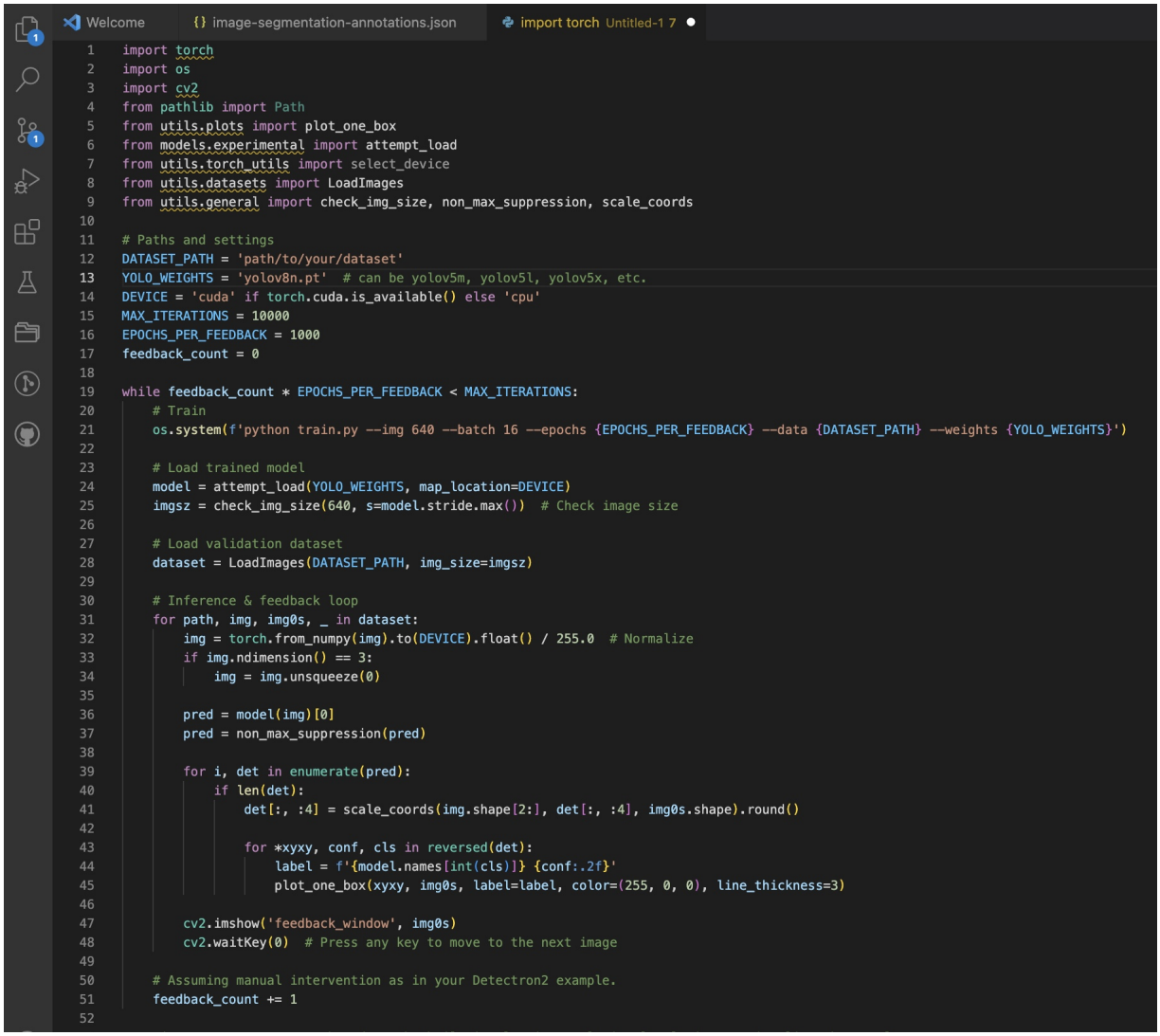


Figure D.2: Code for implementing the feedback loop with YOLOv8