Final Report

Gage Fleming

SN: 220103602

University of London

CM3070 Computer Science Final project

March 30th, 2025

Repository Link: https://github.com/Gage-Fleming/final_project

Word Count: 8128

1 Introduction (Word Count 824)

1.1 Project Concept

At Riteway Signs, our drivers install signs for our realtor clients throughout the day. These signs include a white post on a metal peg inserted into the ground, with the realtor's sign hung on the post. The driver must take a photo of every installation to validate that the job has been completed and is up to our standards.

A photo example of an acceptable installation.

The office staff review these photos as they come in and confirm they meet our company's standards. We review several points for every installation:

- The post is straight.
- The metal hardware is not rusted.
- The hardware is present (nylon screws, hooks, etc).
- The hooks are closed.
- The post is not scuffed or marked in any way.
- The realtor signage is correct and not damaged.
- The optional add-ons are straight and installed correctly.

A photo example of a dirty post.

This project aims to automate portions of this task using the skills I have learned throughout the degree. I want to apply machine learning techniques involving image processing, machine learning, and neural networks to develop a system that detects the relevant areas of an installation photo.

The first step involved narrowing the scope of this project. Since my current skill set aligns with image object detection tasks, this project focused on object detection of the following items to allow further automation to be built on top.

- Post
- Sign
- Top and lower plates

The second step involved curating a dataset as the project template calls for. Here, we will create a multi-class dataset. This dataset will contain images in which bounding boxes outline the relevant parts of an installation image. Tight bounding boxes will be drawn around the items listed above using the open-source image annotation tool CVAT [8].

For the midterm, a small dataset with 100 images was created. Seventy images were used for training, twenty for validation, and ten for testing. The two prior sets will have bounding boxes, and the test set will not. The final dataset featured approximately two hundred images with diverse backgrounds, angles, and installation items. The 70/20/10 split was maintained no matter the size of the total dataset.

The third step involved training multiple models on the dataset to see if an algorithm could accurately identify the installation items. The models were evaluated using industry-standard metrics laid out later on in the report. Future work will include building on this object detection to pass the objects to image anomaly detection algorithms. First, we must ensure that object detection of this nature is feasible using the current state-of-the-art methods.

1.2 Project Motivation

As mentioned, the current validation process involves humans manually and tediously reviewing each installation photo throughout the day. With 200-300 installations per day and approximately thirty seconds to review each, that is 1.5 - 2.5 hours of labour per day. This labour is spent doing repetitive tasks that are prone to human error. It is hard for a person to catch all the anomalies, like cracks on the bottom of the post or missing hardware, especially when other tasks are also trying to distract. Automating this process will significantly reduce the labour hours required daily and improve the quality of results.

We also face the problem of training new staff to follow our standards. The latest office staff need to be trained and taught the correct processes for validating installs and contacting the drivers if something is wrong. This is another drain on the company's resources.

An AI algorithm would have a significant upfront cost, especially for complicated tasks. However, breaking down the problem into several smaller steps should allow for developing an efficient image anomaly detection algorithm over time. If the algorithm efficiently captured anomalies, the next logical step would be implementing it into the website for instant driver feedback. If a photo does not meet our standard, then the driver could be made instantly aware with an indication of where the problematic area is. An algorithm providing feedback also gives the driver less leeway to try and argue about the situation. We also would remove scenarios where the driver has already left the area, and the office staff has to ask them to return, reducing some tension between the office and drivers.

In summary, the critical motivating factor for this project is efficiency. My workplace has a tedious, repetitive task that no one particularly enjoys. Automating the validation of installation photos will lead to more efficient drivers as they get instant feedback on whether their images are good quality. It will also free up the office staff to perform tasks of more substance and value relative to this task. This task starts with object detection, and this starting point is the highlight of this project.

1.3 Project Template

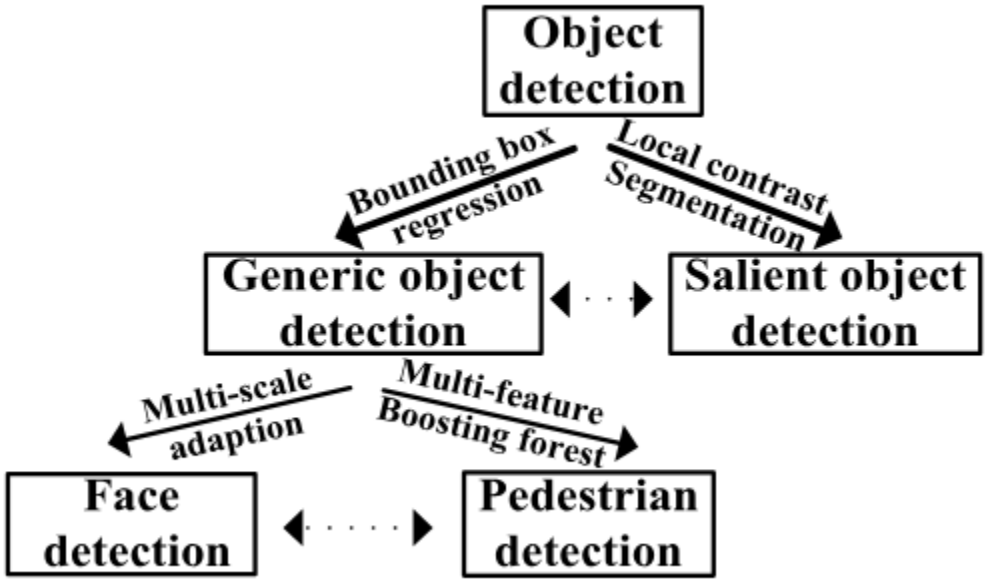
Template number two (Gather your own dataset) has been chosen from the CM3015 Machine Learning and Neural Networks for this project.

2 Literature Review (Word Count 2106)

The literature review portion of this paper will review six papers within the object detection/image segmentation research area and similar fields; this is a trending and complex area of computer vision in which deep learning is used to detect objects within images. These images usually contain various noise and differing variations of the objects in question. The literature review will validate that this project is feasible and situate it within the current body of work to set it up for success.

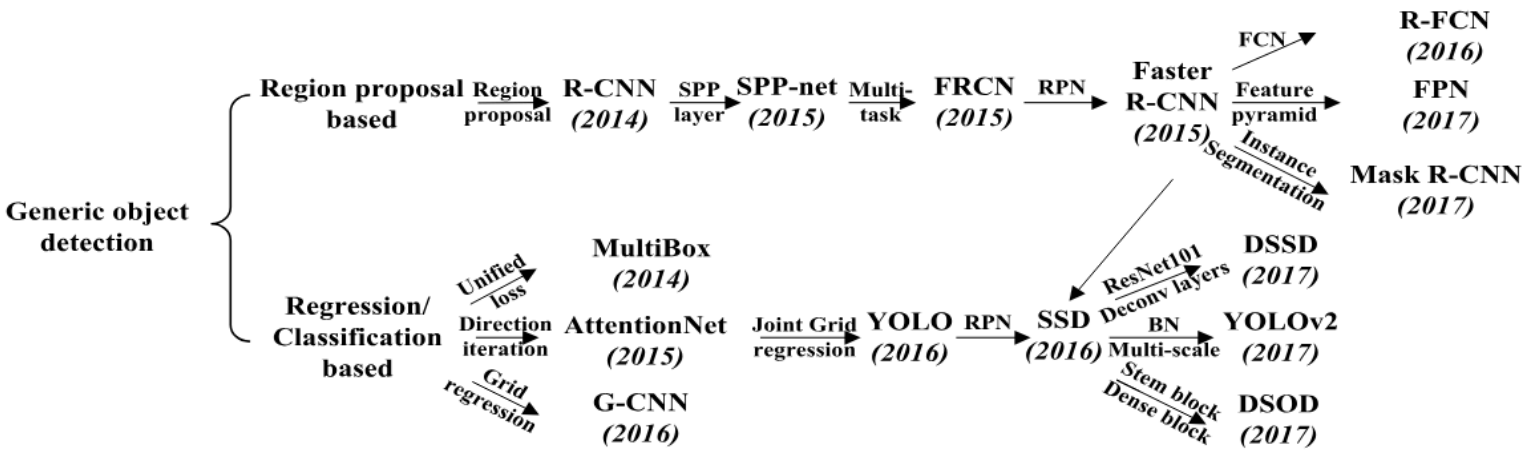
2.1 Object Detection with Deep Learning: A Review (Zhong-Qiu Zhao et al.)

This research paper presents a complete overview of deep learning-based object detection. First, it walks the reader through more traditional methods in the field, particularly highlighting their limitations due to their tedious nature and the weak architectures upon which they are built.



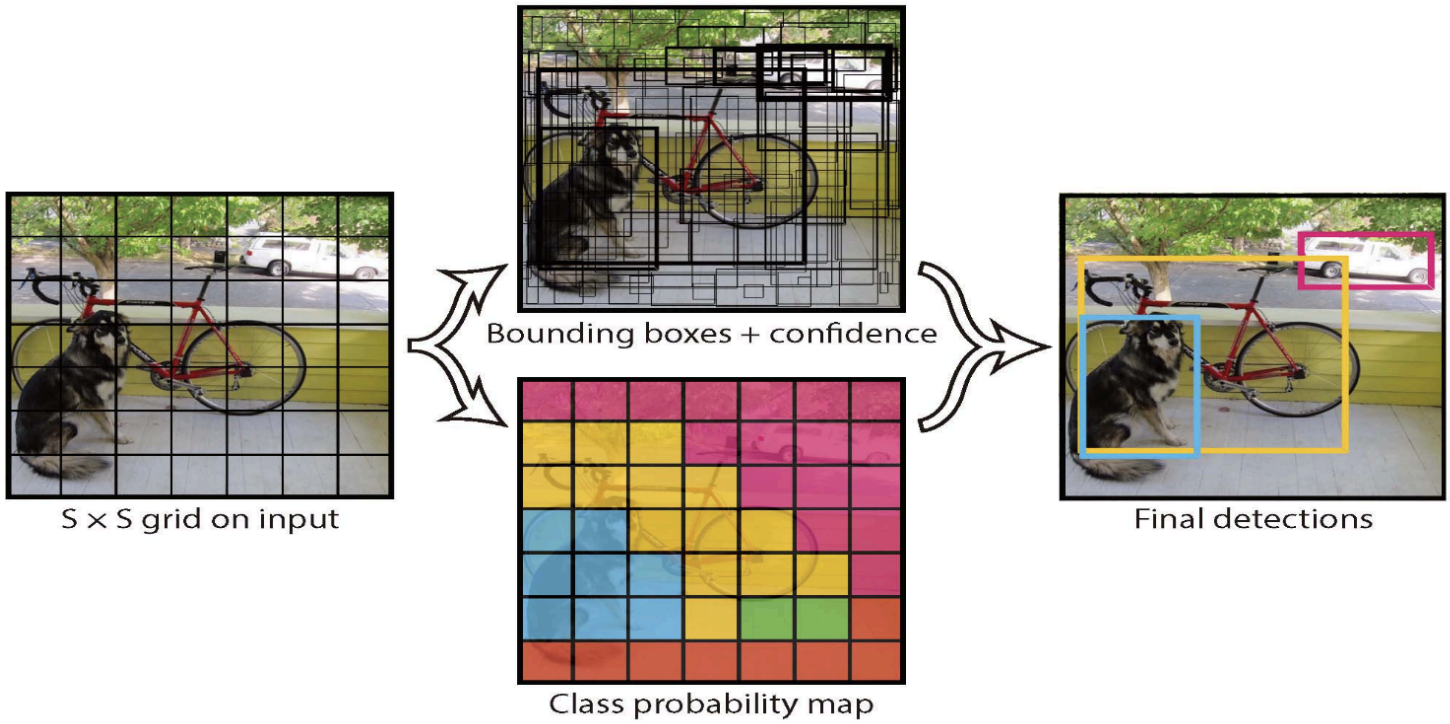
The application domains of object detection.

The paper then highlights the effect of deep learning and the advancements this technology has pushed within the object detection field. Convolutional neural networks, in particular, have led to significant improvement within the field as they excel in learning critical features within image datasets. The paper then goes on to categorize and review state-of-the-art detection methodologies.



The two main frameworks within object detection and associated methodologies.

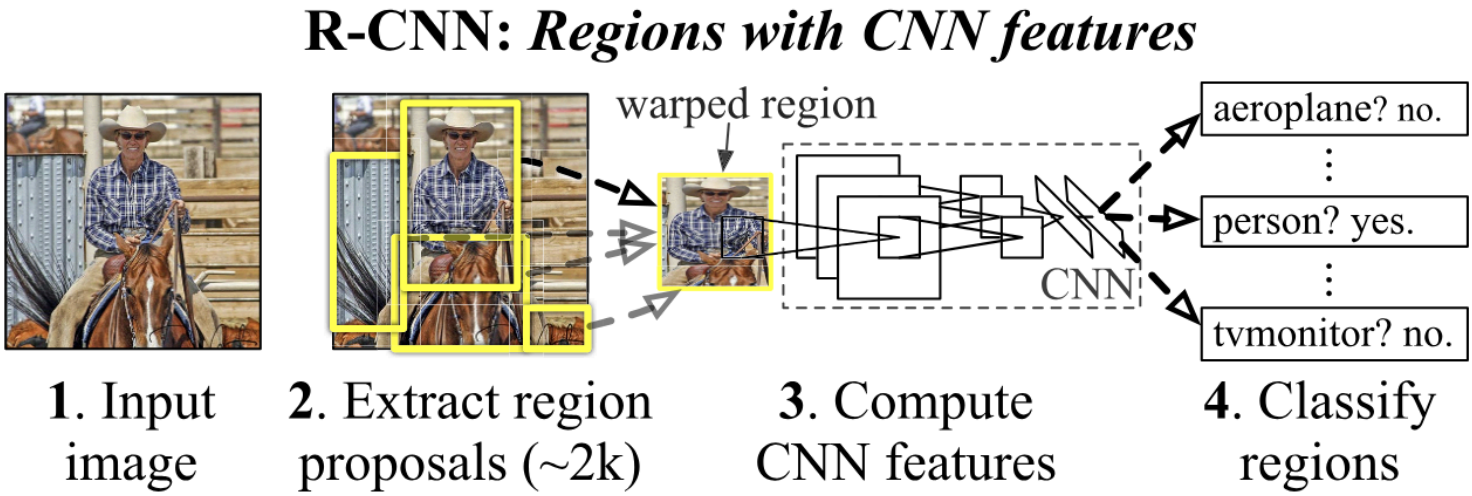
These are classified into proposal-based methods and regression/classification-based methods. The former contains methods such as R-CNN and SPP-Net, while the latter includes models such as YOLO and SSD. YOLO, in particular, appears in many current research papers.



The YOLO methodology at work.

Following this, a detailed comparison of the individual methodologies, architectures, and use cases is presented, and relevant performance metrics are compared.

Zhong-Qiu Zhao et al. then situate these approaches within specific tasks, including salient object detection, face detection and pedestrian detection. These tasks share similarities with the project's use case of identifying varying objects within a complex photo environment.



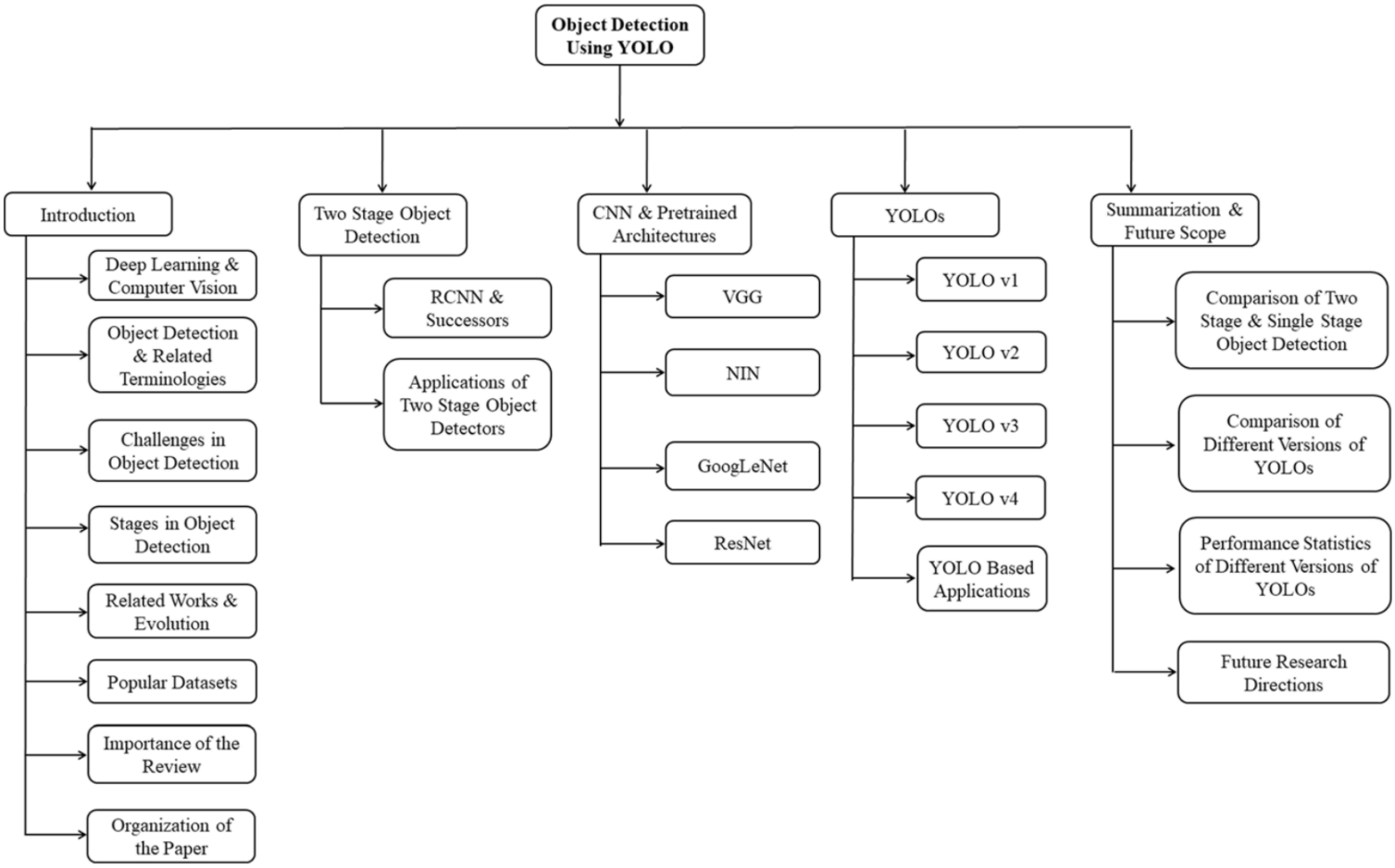
A flowchart showing the progression of the R-CCC methodology.

Finally, the paper identifies relevant problems within the field and proposes some solutions to these problems.

This paper provides a quality foundational understanding of the object detection field in this project's context. This includes the particular object detection algorithms and their differences in their respective strengths and weaknesses. The insights provided will guide the selection of model selection performance metrics and serve as a guide toward more in-depth research papers that walk the project through a more complete understanding of the chosen models and metrics. The paper also guides us through the general workflow of these algorithms, which will be fundamental to the project's workflow as a whole.

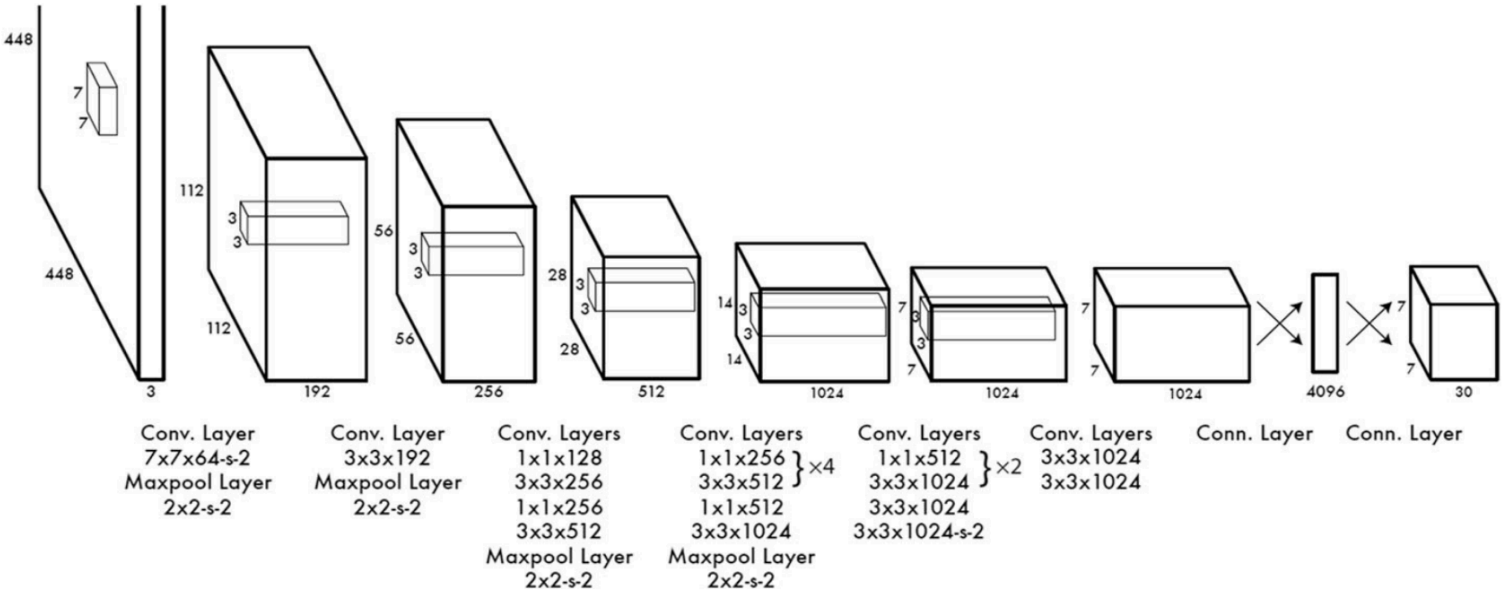
2.2 Object detection using YOLO: challenges, architectural successors, datasets and applications (Tausif Diwan et al.)

This research paper, written by Tausif Diwan et al., comprehensively reviews object detection history, techniques, and other talking points. However, it focuses on the you only look once (YOLO) methodology and how it fits into the field.



The layout of the paper.

The paper walks us through the transition from traditional approaches like R-CNN to the faster but less accurate YOLO-based methods. YOLO looks at object detection as a regression-type problem, which allows it to predict bounding boxes, class probabilities and segmentation in a single pass. Like all single-pass methods, these are suitable for real-time applications where speed is critical—something our project does need to emphasize. The paper emphasizes this difference between single-stage and two-stage object detection algorithms. The former prioritizes speed and simplicity, while the latter is usually more accurate but slower.



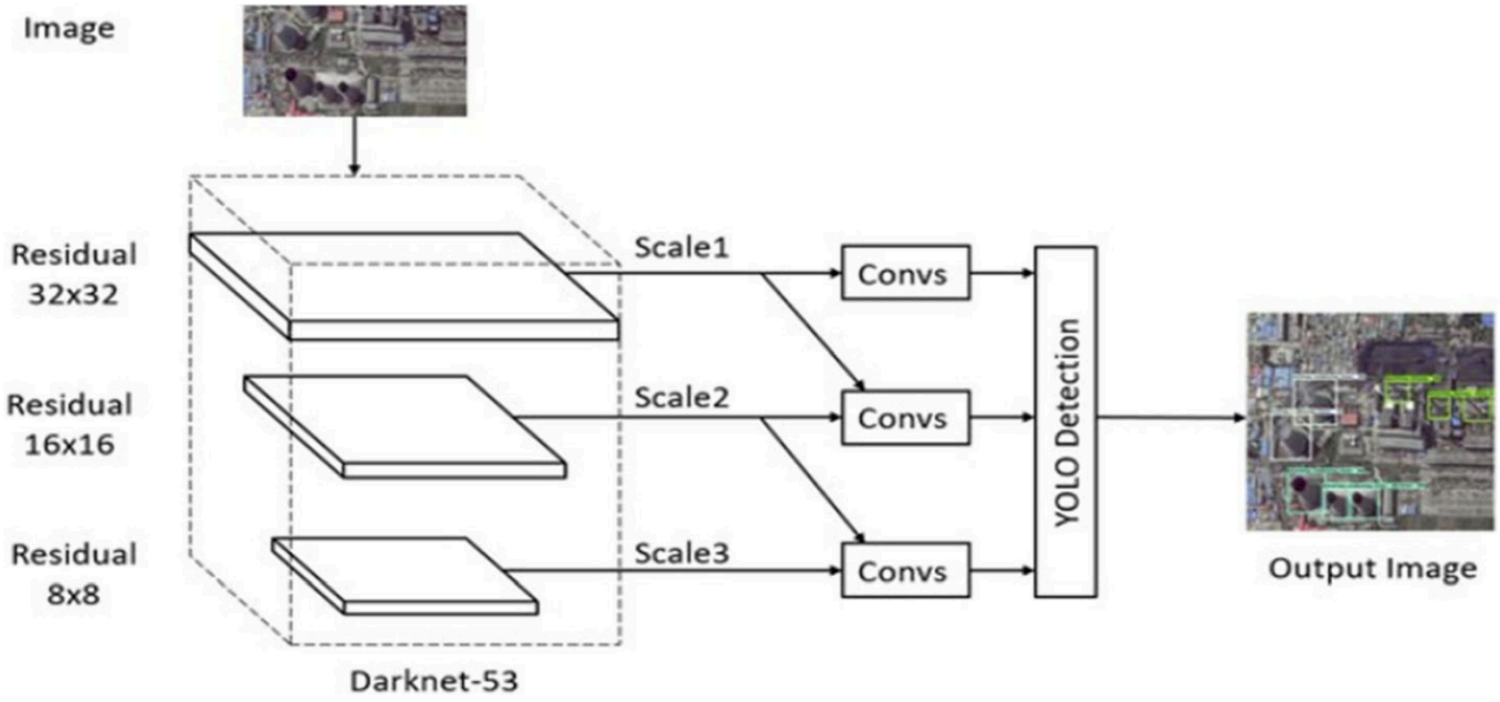
The standard YOLO (v1) architecture.

Tausif Diwan et al. then extensively break down YOLO and the progression of this particular methodology throughout the years, mapping the advancements in architecture and efficiency to improvements stemming from multi-scale training, anchor box regression and feature pyramid networks.

Yolo version	Architecture (Backbone)	Neck	Anchor boxes	Features
Yolo(v1)	Darknet (GoogLeNet 24 layers)	–	Handpicked	1 × 1 convolutions, global average pooling, linear activation and leaky ReLU.
Yolo(v2)	Darknet-19	–	5 anchor boxes using K-means clustering.	Batch Normalization, high resolution classifier, convolution with anchor boxes, dimension clusters, direct location prediction, fine-grained features and multi-scale training.
Yolo(v3)	Darknet-53	Feature Pyramid Networks	9 anchor boxes using K-means clustering.	Independent Logistic classifiers, multi-scale training and predictions.
Yolo(v4)	CSP Darknet-53	Path Aggregation Network	9 anchor boxes using K-means clustering.	Spatial Pyramid Pooling (SPP), DropBlock regularization, Mish activation, ReLU6, Class label smoothing and Cross Mini-Batch Normalization.

Comparison of YOLO architectures throughout successions.

Finally, the paper points to challenging areas within object detection, such as small object detection and noisiness in photos. It also highlights where YOLO can struggle, particularly as a generalized solution; it requires a large dataset to identify new objects efficiently.



YOLO (v4) architecture.

When considering this paper within the project context, a clear pathway was laid out for an efficient model to guide one of the project's model implementations. As laid out in a later section, speed is key for this project, and when the algorithm is complete, we want drivers to have instant feedback. Therefore, due to their speed, a single-pass detector will likely segment the note items in the installation photos. Of course, this assumption will be tested and validated by implementing several different models compared with performance metrics in this paper and others.

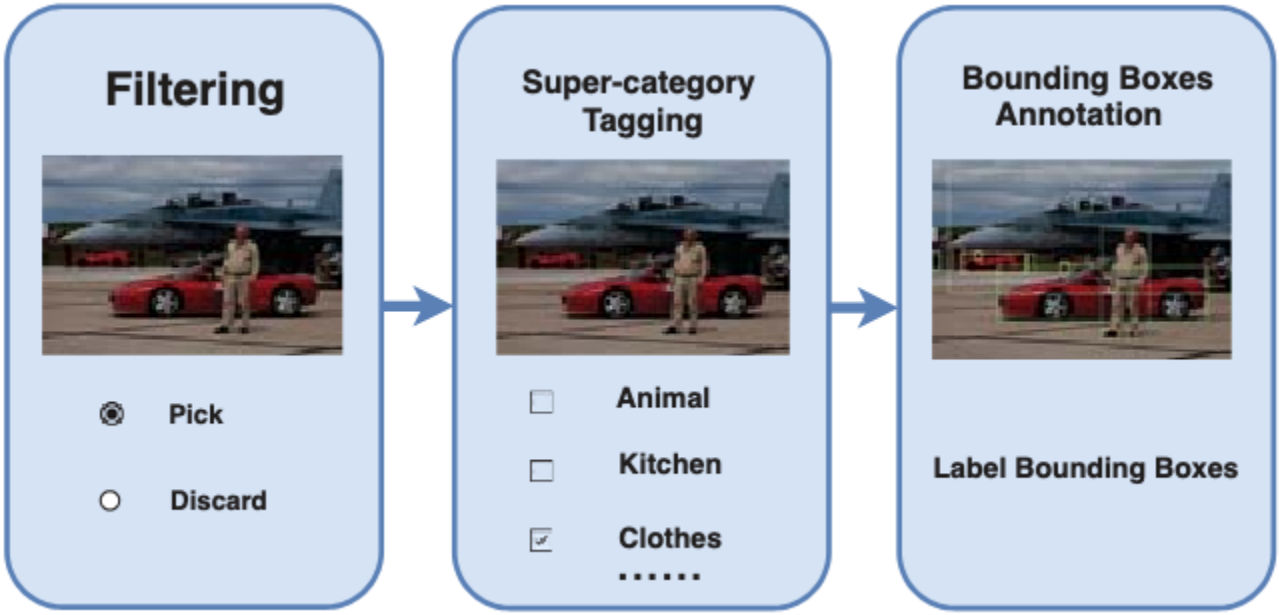
2.3 Objects365: A Large-scale, High-quality Dataset for Object Detection (Shuai Shao et al.)

Shuai Shao et al. present a large-scale, high-quality dataset designed for training object detection algorithms in this paper. The dataset contains 365 categories (hence the name), over 600,000 training images, and over 10,000 bounding boxes, making it significantly more extensive and diverse than pre-existing benchmark datasets like COCO.

Dataset	Images	Boxes	Categories	Boxes/img	Fully Annotated
Pascal VOC	11.5k	27k	20	2.4	Yes
ImageNet All	477k	534k	200	1.1	Yes
ImageNet Dense	80k	186k	200	2.3	Yes
COCO	123k	896k	80	7.3	Yes
OpenImages	1,515k	14,815k	600	9.8	Partial
Objects365	**638k**	**10,101k**	**365**	**15.8**	Yes

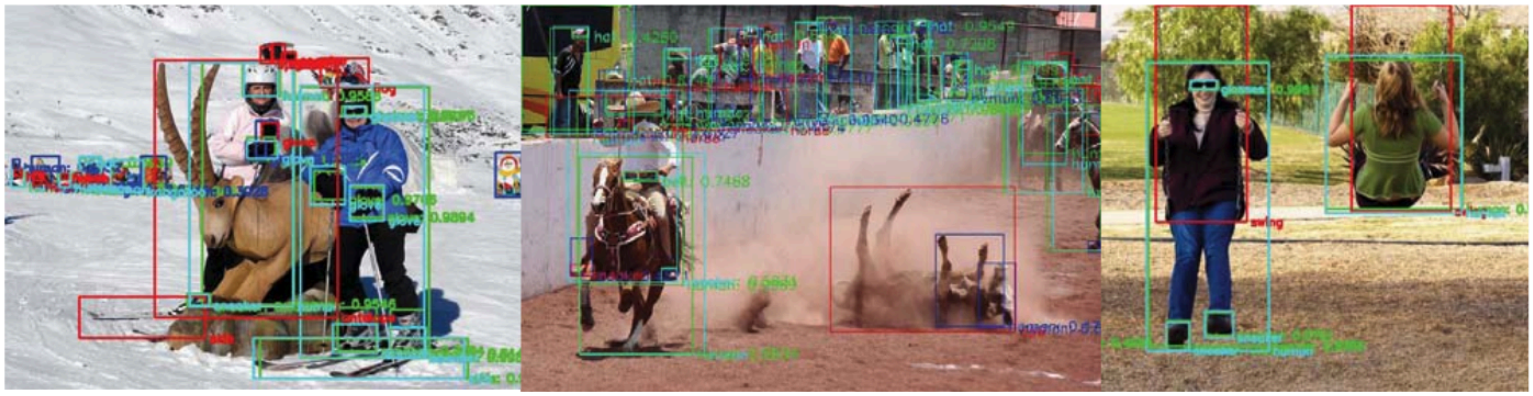
Statistical comparison of pre-existing datasets with Objects365

The paper then goes on to highlight how the dataset was created. Including a structured annotation pipeline that included annotators, inspectors, and examiners. Ensuring a robust redundant pipeline ensured a highly accurate and consistent dataset.



A simple walkthrough for the annotation pipeline.

Shuai Shao et al. highlight the success models have when pre-trained using Objects365. They then directly compare with ImageNet pre-trained models, and the mean average precision across multiple tasks is higher with the models trained on Objects365. These tasks include general object detection and segmentation, which will be employed in this project. The paper also points out how the dataset allows for faster fine-tuning, significantly reducing training time while maintaining comparable or better accuracy to other datasets.



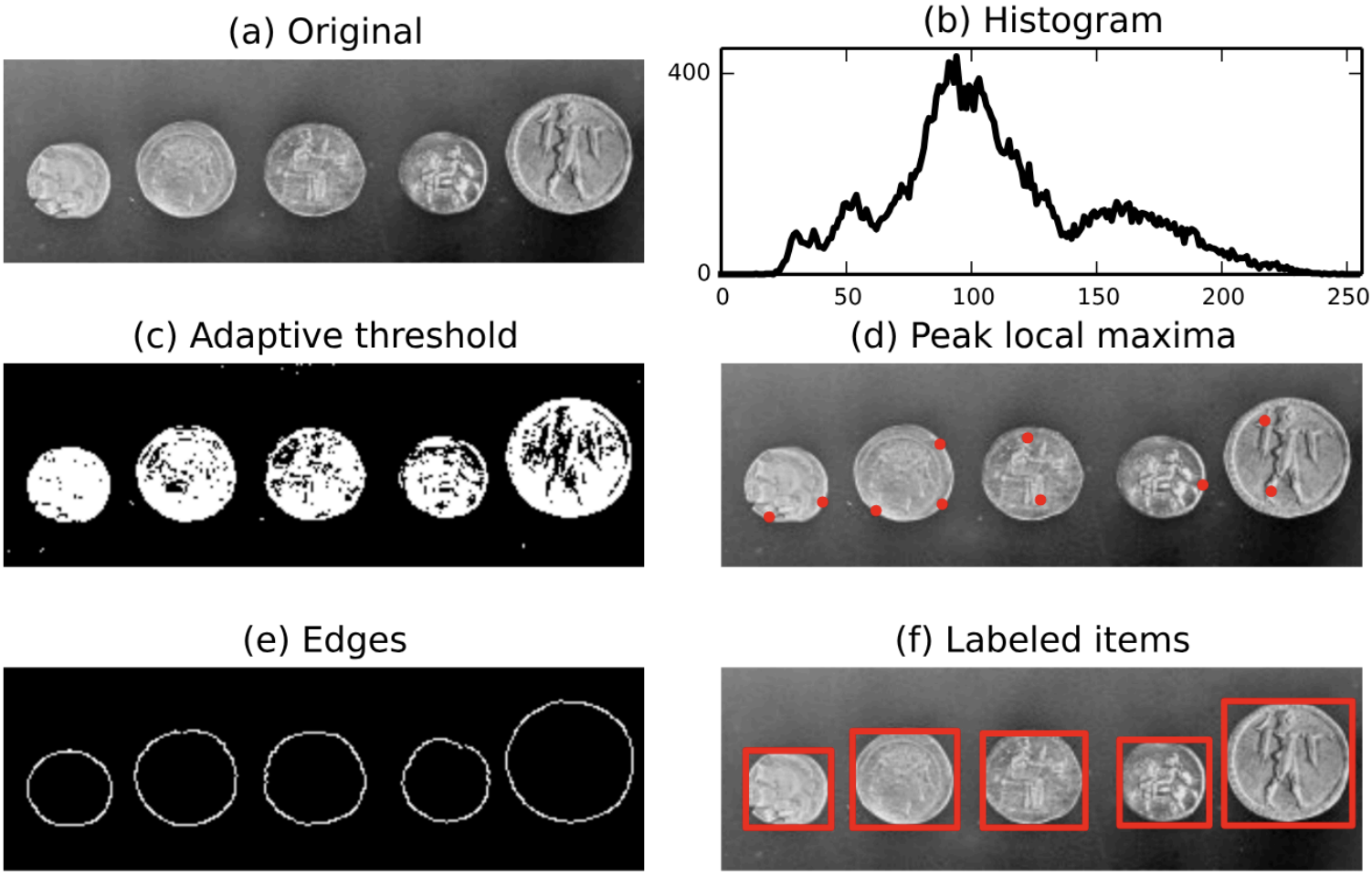
An illustration of some predictions on the Object365 dataset. Green boxes are ground truth, and red and blue are false negatives and false positives.

When looking at this paper within the context of this project, it is clear that the principles outlined in this dataset can inform the creation of our dataset. Notably, the emphasis on quality annotation processes and maintaining consistent labelling throughout the dataset are essential to this project. This will ensure accurate and reliable data labelling that sets the training of models up for success. In addition, the paper highlights the importance of using pre-trained models trained on reliable datasets. The outcome of the small dataset the project will provide heavily depends on the quality of the training the models come in with.

Overall, the insights from this paper will help establish best practices within this project and paint a road map toward what success in creating a dataset for object detection looks like.

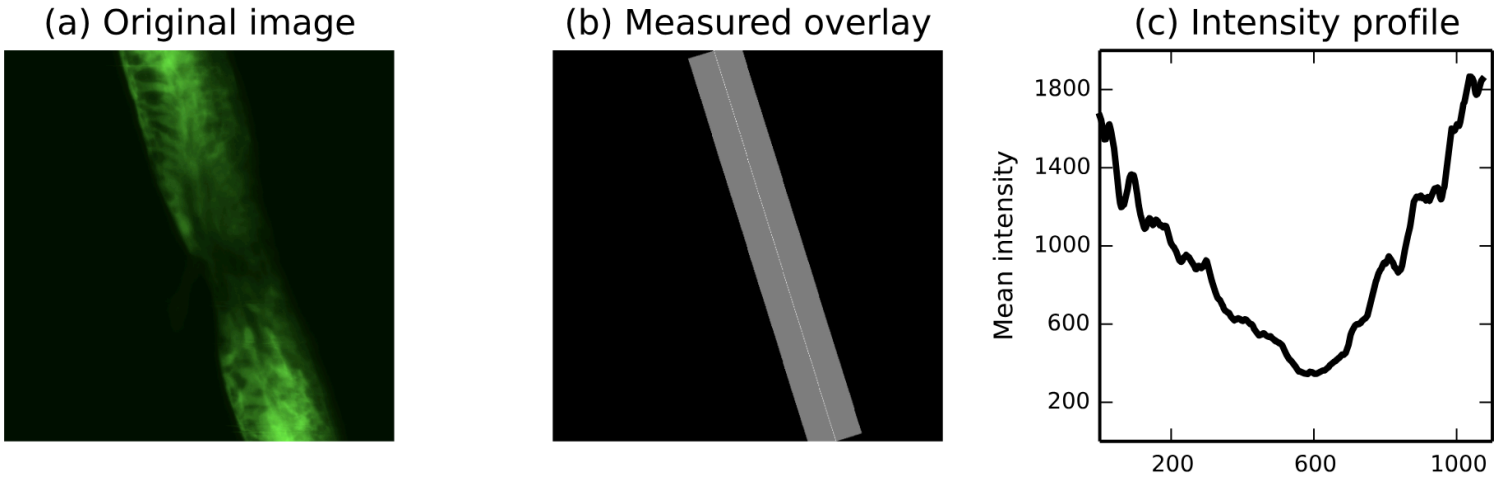
2.4 Scikit-Image: Image Processing in Python (Stéfan Van der Walt et al.)

This paper discusses scikit-image, an open-source Python library that provides a complete repository of image-processing tools. The robust library contains well-documented algorithms for many image-processing tasks, including edge detection, segmentation, feature extraction, and filtering.



Examples of several functions available in scikit-image.

Like most Python libraries, scikit-learn fits in well with a standard Python workflow, making it accessible to most Python users. The paper walks us through the library's applications in various domains, ranging from biology to astronomy, demonstrating the library's versatility and ease of use for beginners and advanced users.



An example of tracking recovery in spinal cord injuries with the library.

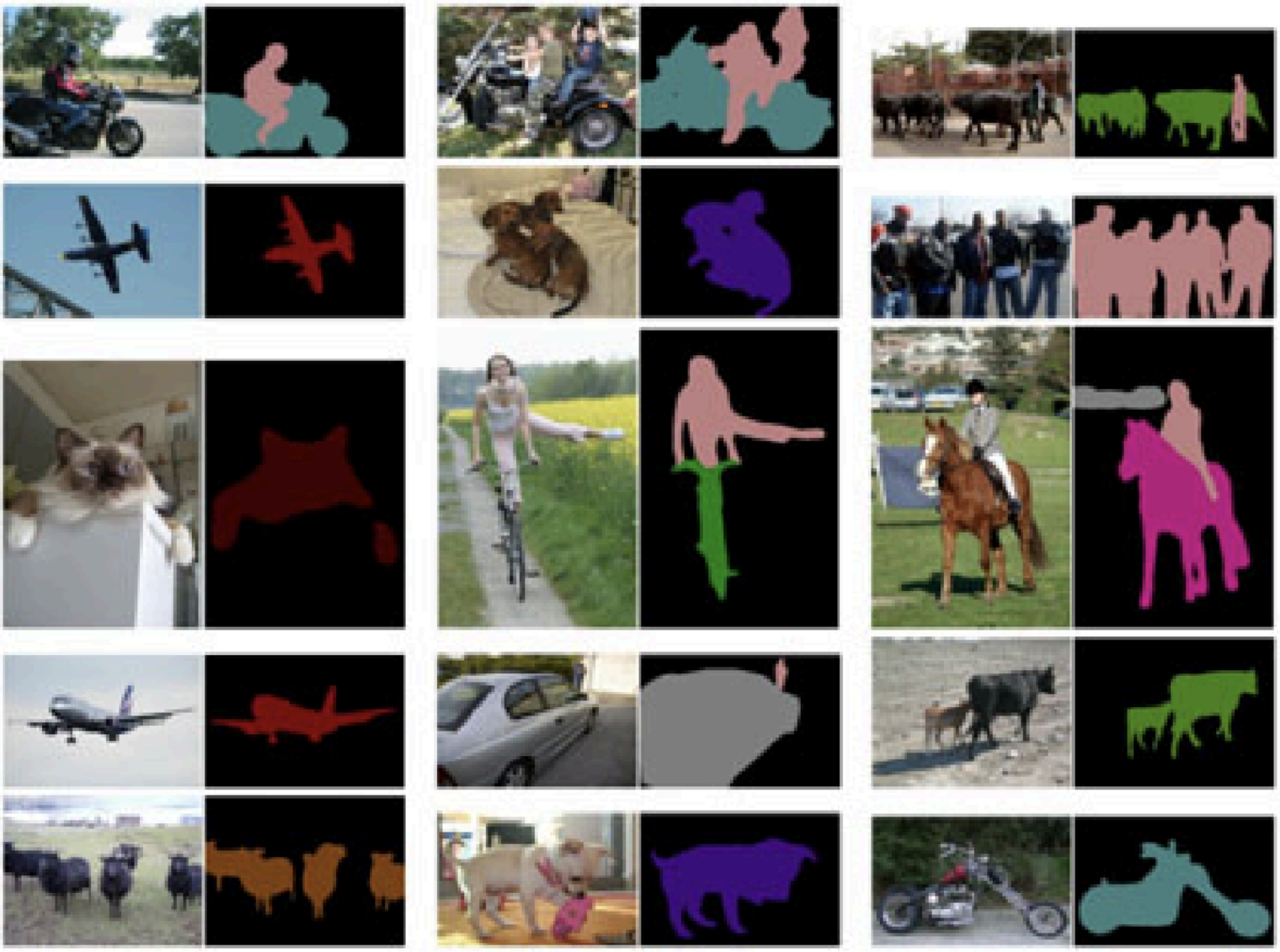
Stéfan Van der Walt et al. then indicate that scikit-image is an effective tool for reproducible research because its open-source nature allows one to inspect, modify, and share code for inspection and transparency. Finally, they look toward the library's future and goals.

When looking at this paper within the context of this project, we can easily see why it is relevant. This project will be based in a Python environment, and scikit-image fits perfectly for preprocessing, feature extraction and visualization tasks. Combining this with its open-source license makes the library a practical choice for building reproducible workflows. This will allow us to create a project with high-quality tools that meet research expectations.

In summary, by utilizing scikit-image, the project will benefit from access to well-optimized algorithms that reduce coding time and increase code transparency, issues that are all too frequent in machine learning projects. This source highlights the importance of using accessible, open-source libraries like scikit-image to streamline complex tasks like image processing to achieve consistent, reliable results. This paper and the library will be invaluable for preprocessing images before sending them to neural networks to help the machine learning algorithm process what is essential to this project.

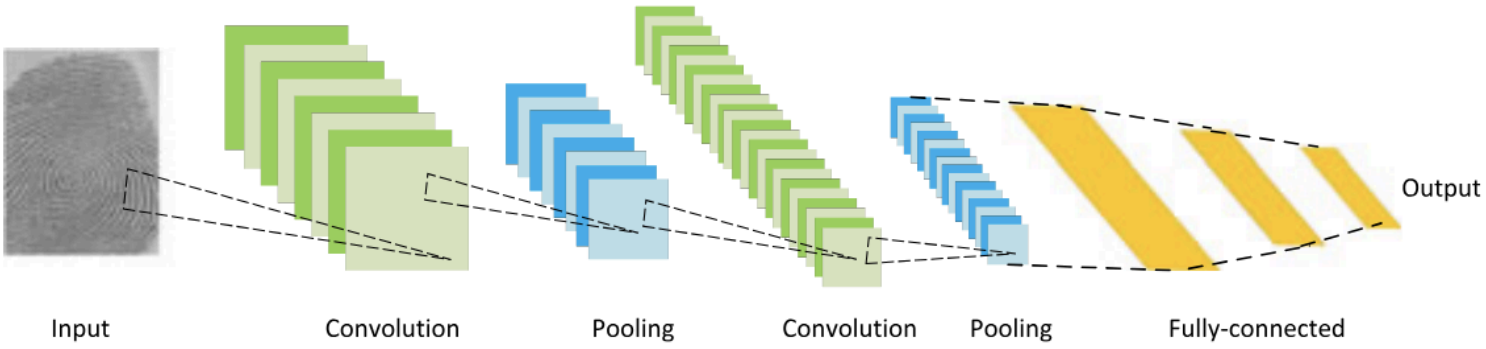
2.5 Image Segmentation Using Deep Learning: A Survey (Shervin Minaee et al.)

This research survey provides a complete exploration of deep-learning techniques related to image segmentation. Image segmentation is a more accurate object detection algorithm in which objects are outlined with pixel accuracy. We should employ this rather than bounding box detection to remove as much noise as possible from the image to be passed on to future algorithms outside this project's scope.



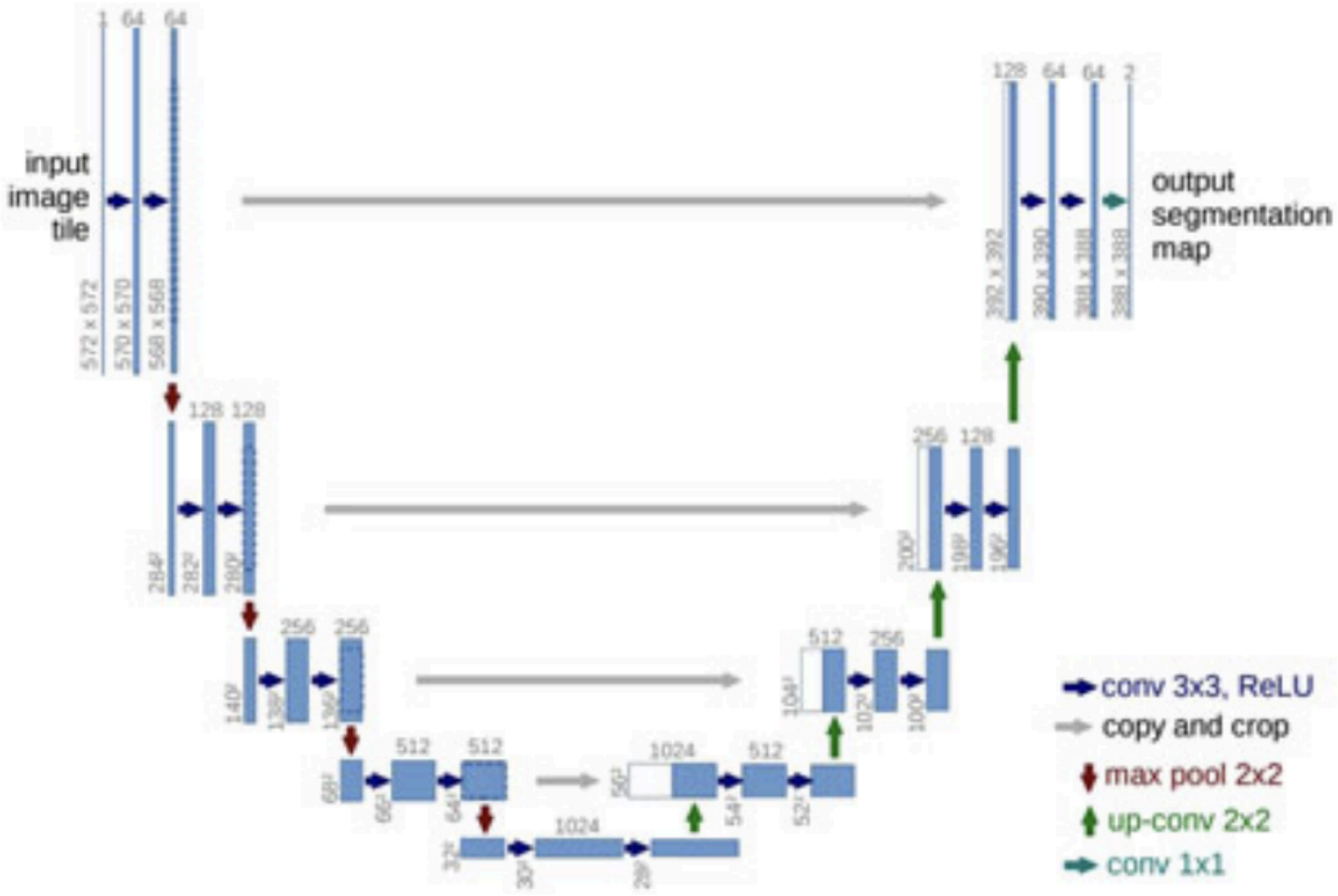
Visual examples of image segmentation at work.

Shervin Minaee et al. discuss recent advancements in this field, including fully convolutional networks, encoder-decoder models, and multiscale approaches. They then examine some state-of-the-art methods, including attention mechanisms and generative models. The paper is an excellent guide for the methodologies one can employ within this area.



The architecture of CNNs.

The paper then focuses on encoder-decoder-based models. These models achieve pixel-wise accuracy and are great for tasks requiring detailed object segmentation, such as the ones in this project. Models like U-NET and other variants are highlighted for their ability to capture the image's global context and the unique local details. Another highlight is the multiscale approach, which utilizes feature pyramid networks. These work great in identifying objects of varying sizes, which would also aid in determining the smaller plates next to the more prominent posts in this dataset.



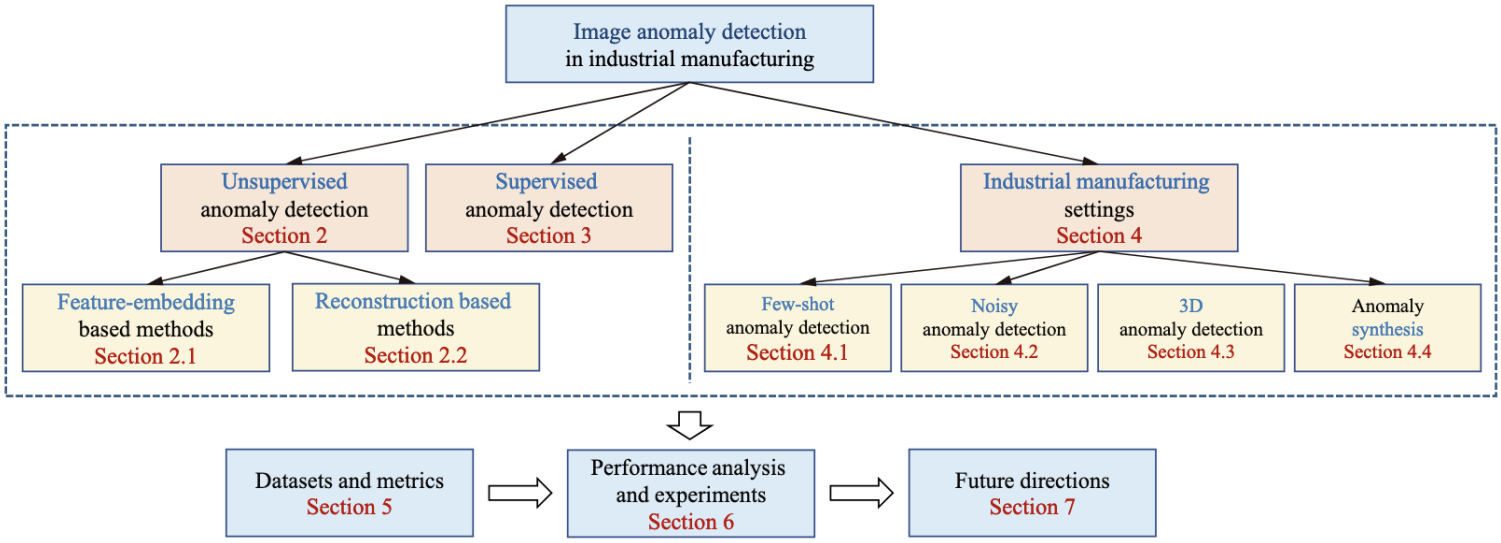
Architecture of U-Net models.

In this project's context, this paper will serve as another foundational guide to its success. Methodologies have been outlined, which will serve as some of the baselines within the project and as the actual models to be tested against each other on the full dataset. The evaluation metrics are also similar to past papers and have confirmed an industry-standard testing methodology, which will be adhered to to ensure our results are accurate to the current industry standards. Finally, the challenges outlined are also highly relevant to the high-noise photos we will use in our dataset. This will provide further insight as the project's code progresses.

2.6 Deep Industrial Image Anomaly Detection: A Survey (Jiaqi Liu et al.)

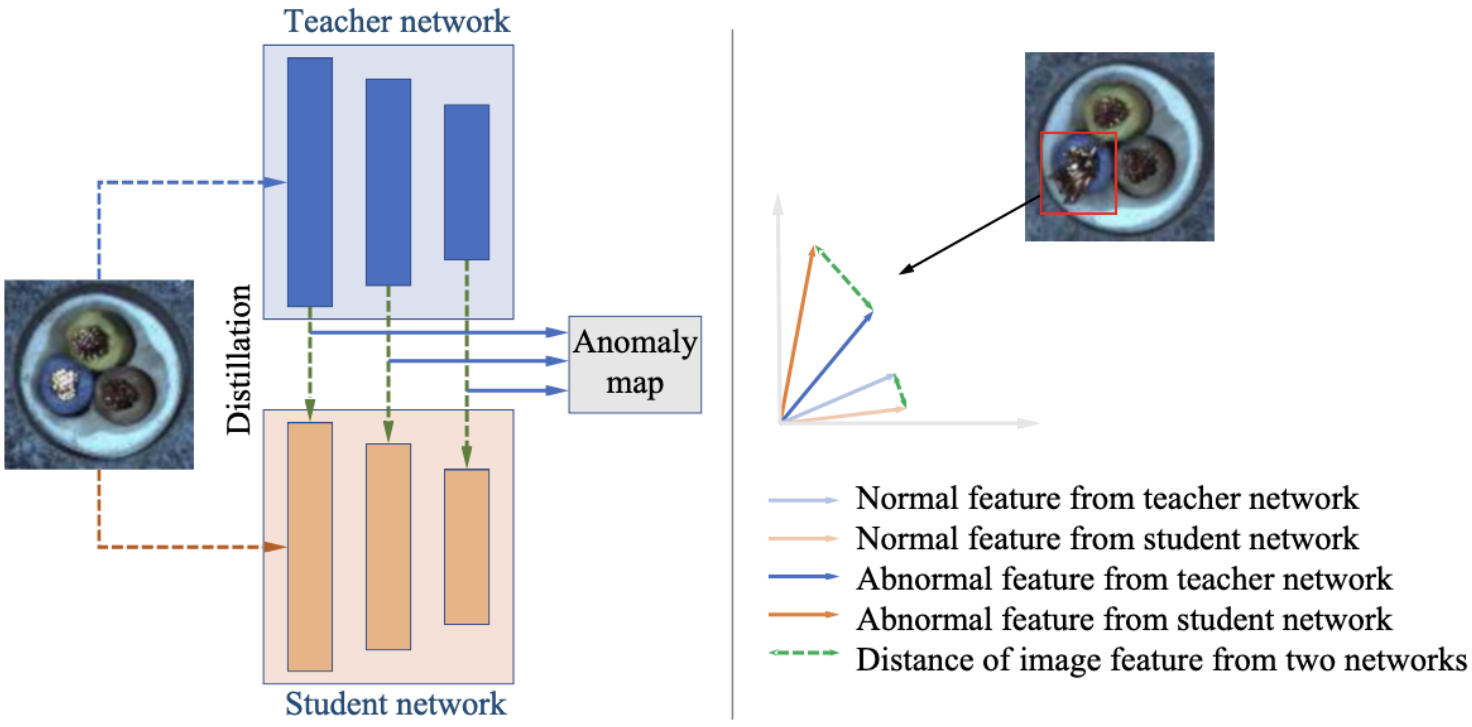
This research paper comprehensively reviews current deep learning-based image anomaly detection techniques. It covers architecture types, supervision levels, loss functions, datasets, and performance metrics within this context.

Jiaqi Liu et al. highlight that IAD research driven by advances in deep learning has significantly improved productivity and quality in defect detection.



The framework of the survey.

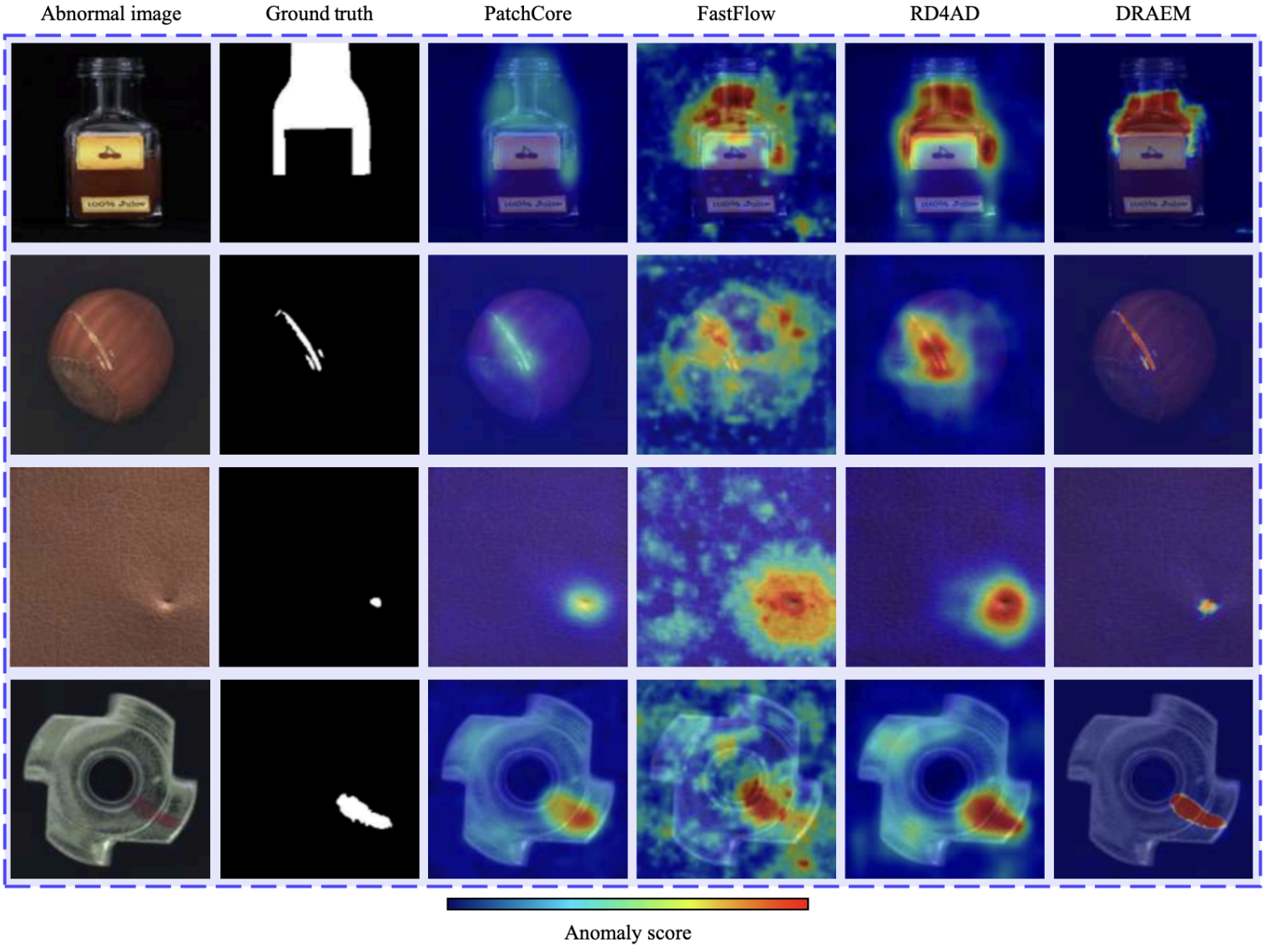
Jiaqi Liu et al. also identify several challenges within real-world applications of IAD, including the lack of labelled datasets, limitations of current metrics, and the need for robust, adaptable architectures. It also touches upon essential IAD methodologies, including teacher-student networks, memory banks, and reconstruction-based techniques.



Architecture of teacher-student model.

Finally, it points to areas of future research, such as enhancing loss functions for better feature extraction from anomalous samples and developing performance metrics for image and pixel-level anomaly detection.

When viewed within the context of this project, it is clear that this paper validates the project's future work. The paper pinpoints the latest IAD methodologies and categorizes them according to their strengths and application areas. It details a variety of architectures and techniques within IAD and thus will serve as a foundational guide as the project works through this area of machine learning. Once the image segmentation is successful, the project will implement several of the papers' models and metrics, which will be compared and contrasted with the performance metrics to see if a suitable solution exists for the project's focus.



Visualizations of varying results from different models.

Furthermore, the emphasis on teacher-student networks and reconstruction-based models highlight a probable path toward success in the anomaly detection of real estate post-installations, especially with the proposed dataset being single-class. Lastly, Jiaqu Liu et al. Highlight effective metrics and robust data preprocessing techniques to ensure the different models compete fairly.

Implementing an image anomaly detection algorithm within this project would be ideal. The project's primary focus will be on developing a quality object detection/segmentation algorithm to pass items on to an image anomaly detection algorithm. Exploring this within this project would be a great use of time.

3 Design (Word Count 1746)

3.1 Project Overview

This project aims to automate the validation process for real estate sign post installations using machine learning techniques. Currently, drivers upload photos of installations, which are manually reviewed by office staff to ensure they meet the company's standards. The goal of the review process is to ensure no anomalies with the installation—some possible anomalies are listed below.

- The post is straight.
- The metal hardware is not rusted.
- The hardware is present (nylon screws, hooks, etc).
- The hooks are closed.
- The post is not scuffed or marked in any way.
- The realtor signage is correct and not damaged.
- The optional add-ons are straight and installed correctly.

With 200-300 photos coming in daily, it is easy to see how this repetitive task is time-consuming, prone to error, and inefficient use of office staff time. Therefore, this project proposes a machine learning-based solution to automate the validation of installation photos.

Completely automating this process is outside of the scope of this project. Within it, we will focus on image segmentation, notably isolating the important parts of the photos. Future work will involve passing the isolated objects to further processing algorithms. As confirmed by the literature review, these detection/segmentation algorithms consistently show near or above human-level accuracy in these tasks. Thus, training a model to detect the posts, signs, and plates is the first step in automating this complex quality control task. The complete automation of this task will allow the office staff to work toward more meaningful work. It will hopefully start an automation overhaul of the company's more redundant and tedious tasks.

3.2 Project Template

Template number two (Gather your own dataset) has been chosen from the CM3015 Machine Learning and Neural Networks for this project.

3.3 Domain and Users

3.3.1 Domain

The project's domain is quality control, particularly identifying and highlighting real estate signage in installation photos. The domain of artificial intelligence being used is primarily related to image segmentation/object detection, which is a subset of computer vision. Of course, these domains do not exist within a vacuum, and many different data science/ML tasks will be employed within the project.

3.3.2 Users

- Drivers
    - The company employs several drivers to install the real estate signage. These drivers will use the complete automation tool to receive instant feedback on their sign installs, eliminating the time delay between the office staff review and the communication of potential issues found by office staff to drivers.
- Office Staff
    - The office staff will periodically refine the tool and check in with the drivers to ensure it increases driver efficiency and does not create unintended ill effects on the overall product.
- University staff
    - The university staff also needs to be considered, as we will ultimately be graded within an academic scope, and this project needs to be correctly situated within current research and educational standards.

3.4 Justification

The domain and users will ultimately guide several design choices throughout the project. The following subsections walk us through significant design decisions and how they relate to the domain and users.

3.4.1 Image Outline Dataset

This design decision is directly related to the company and the nature of the real estate sign installation business. When our drivers install signage, the office staff receive a photo, and the customer gets the same image. Therefore, we ask the drivers to take every photo diligently to ensure it meets our standards. Thus, we have a constant stream of images related to this dataset with various backgrounds and signage items. For segmentation, we will highlight the post, signage, and plates in the installation photos.

Should efficient models be trained, these outlined items will be the entry point to a more extensive pipeline within a future work plan. We want to use these segmented masks to pass on to image anomaly detections to evaluate the quality of the installation.

An example of a sign outlined using CVAT.

3.4.2 Photo Processing

Image machine learning tasks often go through several layers of image processing to present the image to the algorithms in a way that highlights the essential information. The algorithms seen in the machine learning course frequently reduce the pixel density and strip out colours to pass to the game-playing AIs. Depending on the success of varying algorithms, image processing may need to highlight the differences between the items the algorithm should care about and others. This will need to be tested in an iterative approach. Due to the nature of the photos, the background will be pretty noisy, especially in winter, when the white background may blend in with the post.

Therefore, edge detection, filters and processing should all be kept in mind to reduce the noise and add important information to the algorithms.

3.4.3 Evaluation Metrics

The project models will be evaluated against the standard machine learning valuation metrics, which include accuracy, precision, recall, and F1 scores. These metrics were taught throughout the machine-learning-based courses offered by this program. These metrics will guide the office staff in choosing a model they believe in and ensure the installed photos meet our standards.

To round this out, we also need to look at the speed of the models. Drivers should receive near-instantaneous feedback, and if a model takes several minutes or longer to segment a photo, it will not be of much use out in the field. The company and drivers would be better off sticking to the current solution to the problem.

3.5 Structure

The project will follow the below steps to completion.

1. Data collection
    a. A collection of installation photos will be compiled from recent installation photos. These photos should be of installations with a wide variation of information between them to depict the changes in signage and backgrounds accurately. Various backgrounds should be present to ensure a robust image processing pipeline is built that can handle the many different landscapes in the greater Calgary area. The final dataset should contain at least 180 training and 20 test images. The midterm dataset had seventy training, twenty validation, and ten test images.
2. Image preprocessing
    a. With a reasonable dataset collected, a possible image preprocessing algorithm will need to be developed to highlight the installation's essential parts for the algorithms to train and work with. To ensure success in the project, we will focus on the image segmentation portion of this

proposed workflow. This will limit the scope of the images and the training algorithm to help it focus on a singular task. Therefore, the items below could pose hurdles to this project's success and should be kept in mind as hyperparameters and the smaller items are adjusted.

i. Background: The background can significantly hinder algorithms looking to segment images, especially in winter. It can be hard to distinguish between the posts and the snow on the ground.

ii. Sign panel: This dataset will involve many different sign panels, and the design variance may cause issues with a segmentation algorithm trained on such little data.

iii. Plates: The same issue as above is present here. Will the variance in design be too much for the algorithm, and is there image preprocessing we can do to solve this should it arise?

iv. Neural network preprocessing: Several similar studies indicate images are processed in a manner that can include reducing pixel density or adjusting colours. Several options may be explored here depending on the results from different algorithms and parameters.

3. Model Training

a. With a valid dataset and an image preprocessing algorithm, image object segmentation algorithms must be trained on the processed dataset. The models below will be explored and compared with the evaluation metrics. We must also remember that the segmentation does not need to be perfect. We will be passing off these images for further processing, and as long as most of the noise is removed, it is okay if some of the background is left in these images. We must also keep our expectations within reality to ensure this project is finished and we are okay with decent results.

i. SSDMobileNetV2: This will serve as our baseline model as the setup time is quick and efficient.

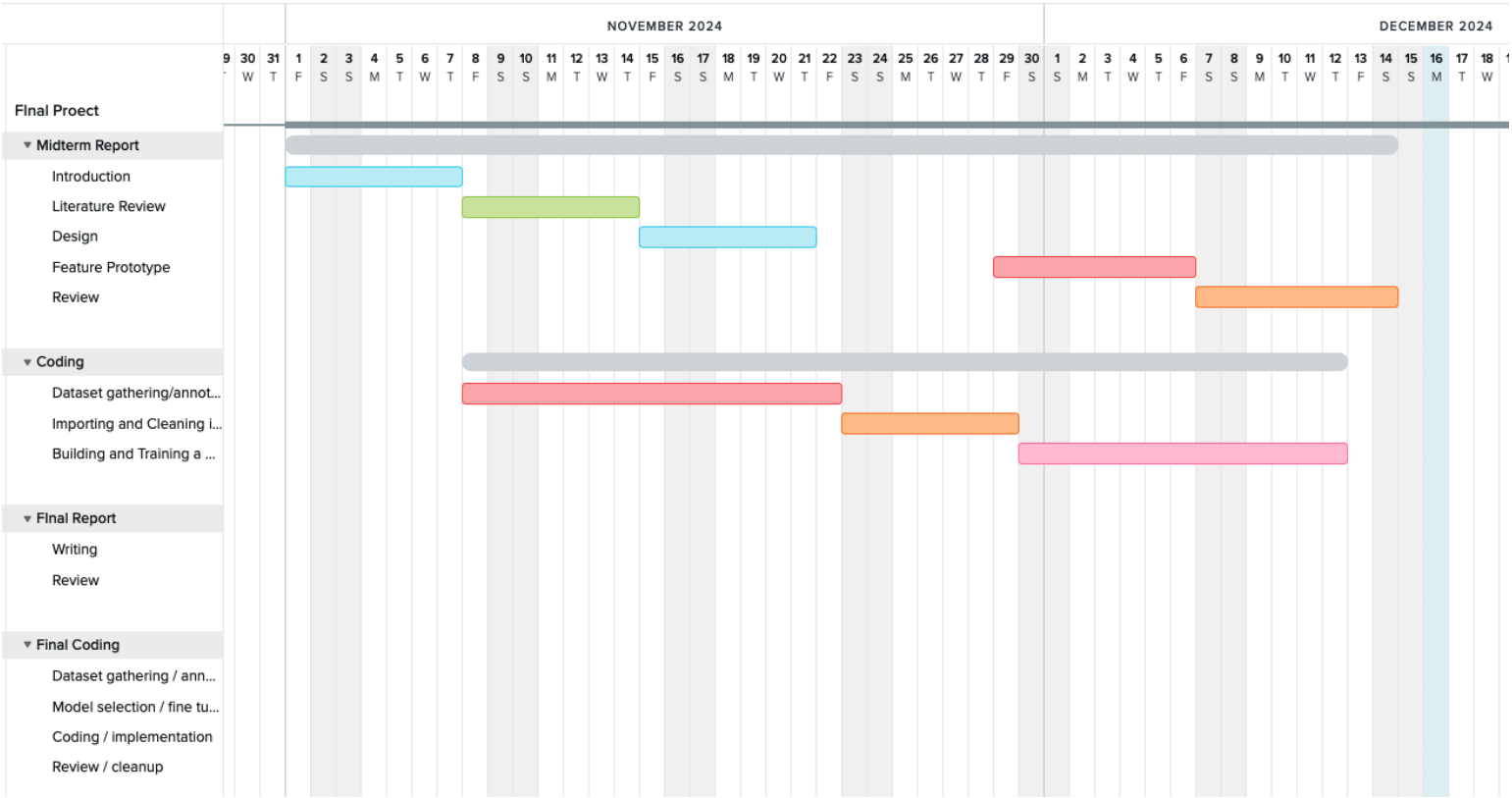
   ii. YOLOv5: A YOLO model will be employed, as this model is referred to in a primary reference. It is well-suited for detecting multiple objects in real time. Time is essential to the project, and this model seems quick to predict and set up time.
   iii. FASTER R-CNN: This is a two-stage detector opposite the single-stage detectors above. It should be more curated but at a cost to its speed. We can see if the accuracy gains are worth the time loss.
   iv. DeepLabV3+: A state-of-the-art model that should give us extreme pixel-wise accuracy for these objects at the cost of speed.

4. Model Testing
   a. After the model is trained, we want to evaluate it against the evaluation metrics to see how well it performs. We also want to see how it presents the image segmentation. Does the overall shape of the images match our expectations? User testing would be a valid source of information to confirm if the algorithm is accurate and quick enough for the company's needs.
5. Iterative development
   a. With the models trained and evaluated, we can decide on potential areas for adjustment throughout the several algorithms to improve the project's performance. This can include adjusting the image processing algorithm, the hyperparameters for the neural network, or even the dataset.
6. Extra work
   a. Should time permit, once we have a well-performing model, we can build further processing tools to round out the image validation. This will include:
      i. Confirming the correct sign is used with text extraction.
      ii. Attempt to pass the posts to a baseline image anomaly detection algorithm.
7. Deployment

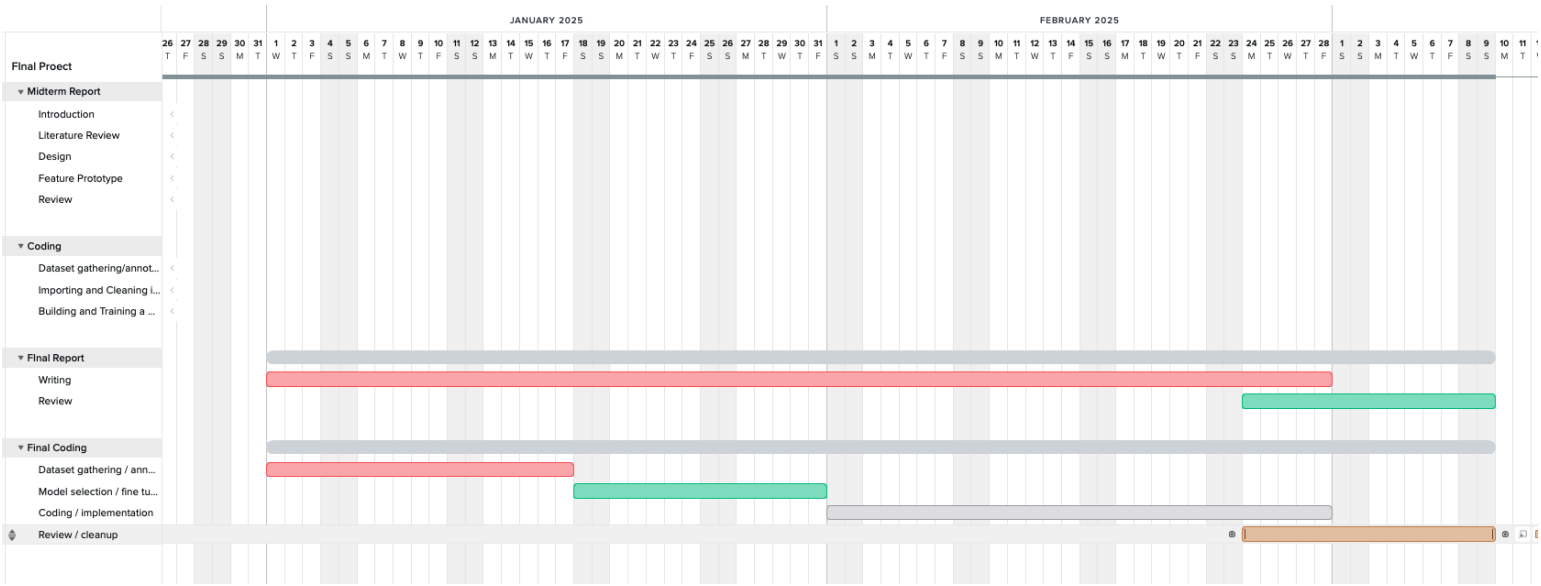
a. Future work will involve building out the rest of this pipeline, including passing the segmented portions of the photos to their respective pipelines. Posts will be sent for image anomaly detection, and the signs/plates should be confirmed as correct for that realtor.

3.5.1 Workflow Timeline

The Gantt charts below outline the ideal workflow during the midterm. As the marker can see in the Git repository, much of the final work was completed closer to the deadline due to unforeseen family issues. Unfortunately, life events do happen and often when they are inconvenient. However, projects of this nature must go on, and given the tighter deadline of closer to a month of actual work, I still believe a quality final submission was produced to round out this degree.



GANTT chart for the midterm portion of the course.



GANTT chart for the final portion of the course.

4 Implementation (Word count 1649)

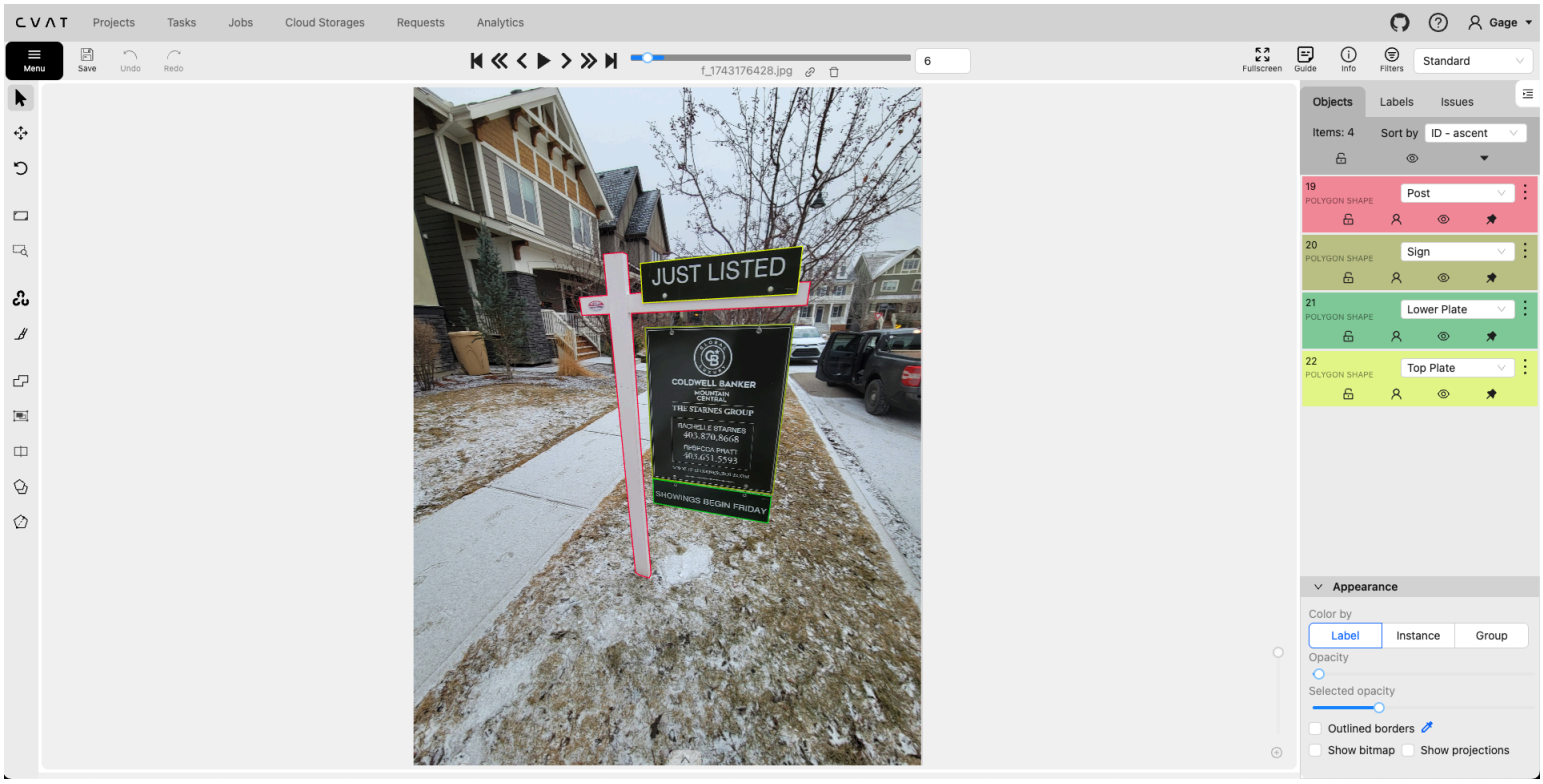
4.1 Template Statement

Template number two (Gather your own dataset) has been chosen from the CM3015 Machine Learning and Neural Networks for this project.

4.2 Project Overview

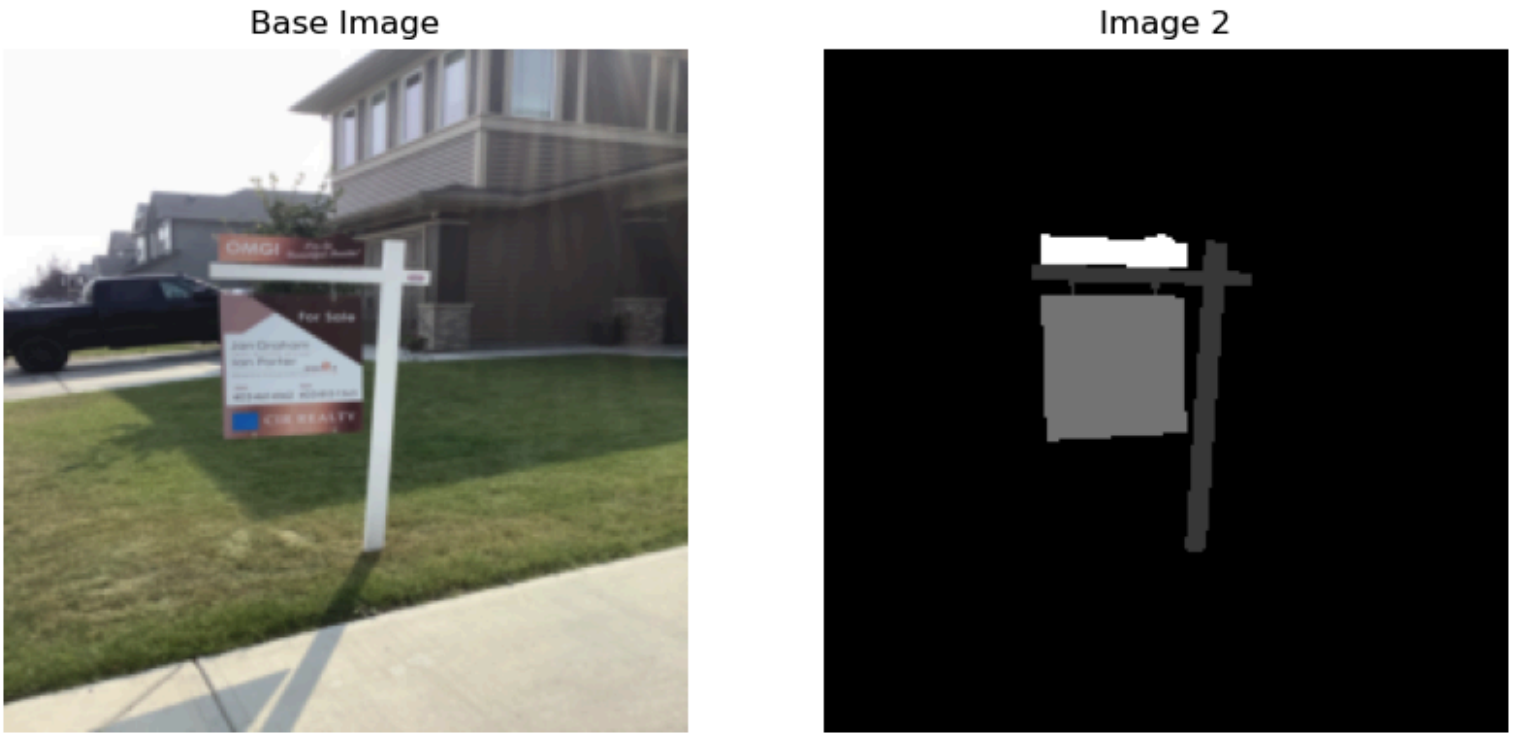
The project focuses on building a functional image segmentation base model to detect key components—posts, signs, and plates—from real estate installation photos. The prototype uses a curated dataset annotated using CVAT, a MobileNetV2-based encoder-decoder model, and various preprocessing and evaluation techniques. A broader picture is painted below.

This project created a custom dataset of around 200 images in which real estate signs, posts, and peripherals were annotated using CVAT. The photos were iterated through and manually annotated using the tools provided in the open-source program.



Example of an annotated photo within CVAT.

The images and annotations were then imported into Python and adjusted to create a unique mask containing the polygons stored in a data container with the photos. Simple data preprocessing, such as image resizing and tensor adjustment, also took place to send the data through neural networks.



An image with its associated mask.

The masks were created by assigning a pixel value based on the class the pixel belonged to. The class structure is broken down below.

- Class 0 - Background
- Class 1 - Post
- Class 2 - Sign
- Class 3 - Lower Plate
- Class 4 - Top plate

As one can see, even the slight difference in Pixel values leads to a noticeable difference on the mask. This is a byproduct, as we only need the neural network to know how to output the information.

This information was then split into appropriate amounts for training, and a machine learning model based on reference 25 was built to serve as the base model. This base model was built using an encoder/decoder model in which MobileNetV2 served as the encoder, and an example decoder was used from the Tensorflow example.

The model was then trained on the data, and analytical updates were provided as batches were completed. This, in essence, walks one through the project's significant steps.

4.3 Project Features

4.3.1 Custom Dataset

The custom dataset is the heart of the project. In it, I annotated approximately 170 images using the open-source tool CVAT. Pixel-wise outlines were drawn around each item class in a photo, and then the pictures and annotations were exported. The images were exported in their native format, while the annotations were exported in XML format for easy use within Python.

```
  <image id="0" name="f_1698528979.jpg" subset="default" task_id="6" width="4032" height="3024">
    <polygon label="Lower Plate" source="semi-auto" occluded="0"
points="1286.40,1612.80;1287.60,1830.80;2182.90,1830.80;2184.20,1613.60" z_order="0">
    </polygon>
    <polygon label="Sign" source="semi-auto" occluded="0" points="1241.40,638.80;2187.60,652.10;2176.50,1579.30;1268.70,1585.60"
z_order="0">
    </polygon>
    <polygon label="Post" source="semi-auto" occluded="0"
points="1173.38,407.53;1414.40,412.50;1417.70,370.70;1438.20,369.80;1445.20,413.00;1972.60,423.30;1976.50,384.60;1995.40,384.60;2000.9
0,424.70;2345.10,434.10;2346.75,401.62;2346.75,304.17;2350.69,259.88;2353.80,158.90;2497.00,164.30;2488.00,436.90;2676.30,441.40;2669.
60,559.30;2665.20,566.90;2552.70,565.10;2482.40,563.30;2413.80,2090.70;2372.10,2790.40;2254.60,2788.50;2269.20,2454.80;2330.50,933.20;
2342.50,559.10;2015.50,554.40;2015.10,646.10;1964.80,644.70;1963.80,551.80;1455.10,540.30;1451.90,640.50;1421.30,641.10;1410.70,541.60
;1177.90,535.00" z_order="0">
    </polygon>
```

An example XML datapoint for an image and its associated annotations.

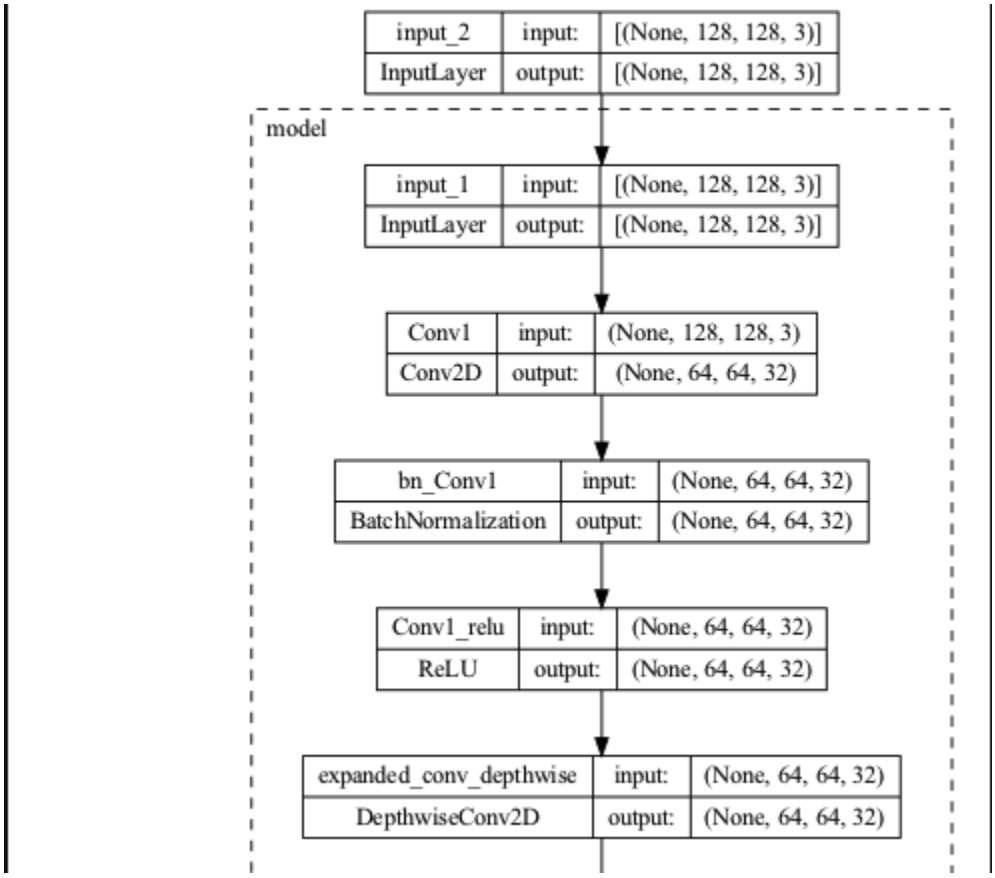
4.3.2 Image Preprocessing

This part of the project was more complicated than anticipated. When I wrote the test, I had yet to find a way to resize the images and annotations into consistent sizes efficiently. Through research, I built a pipeline that imported the photos and annotations and resized them to create a resized original photo and a resized mask, which mapped perfectly to the new image. The pictures and annotations are imported into Python; then, the image is resized through skimage.

The annotations are imported using xmltodict and then resized based on a new desired size. The annotations are then overlaid on a simple mask representing photo data.

4.3.3 Machine Learning

This feature involves creating, training and evaluating a base model on this dataset. The basemdoel in question is a U-net model built on MobileNetV2 and an example decoder provided in reference 25. Most of the code for the model was taken from this reference. Adapting it to my dataset was a complex undertaking. The dataset in the example code was already split into folders imported using Tensorflow datasets. My dataset was already imported in a Python dictionary, and it took some time to figure out how to adapt my implementation.



The first several layers of the base model.

The template for this project called for evaluating multiple models. However, the complexity of the dataset and mapping different models proved too much for my current skill set, so only a base model was trained on this data.

4.4 Project Algorithms, Techniques and Methods

4.4.1 Dataset Preparation

Images were downloaded from an AWS server, which stores our installer photo. These photos were then uploaded to CVAT for manual annotation and then exported in COCO format to an XML file for upload to the Python project. This file was then imported using xmltodict and transformed into pixel-wise masks using skimage.draw.polygon and Numpy. The masks of each item were overlaid on a blank mask to depict the unique classed by the pixel representation associated with said class.

An example mask in 128x128 size.

The data was then split into 70% training, 20% validation, and 10% testing splits using Tensorflow's data processing pipeline, which is explored in the following section.

4.4.2 Model Architecture

A simple, encoded decoder neural network was built to train and predict image segments for the base model. The decoder portion was built on a decoder presented in reference 26 and also referenced in reference 25/27. This allowed the neural network to upscale the image as it passed into the final layers. The encoder was built on a pre-trained MobileNetV2 model referenced in the literature review portion of this project. The model was directly

imported from Tensorflow and is the central processor to predict the segments. It cannot be stressed enough how integral references 25 and 27 were in getting this project to work. Much of the code used in creating the neural network was taken from this reference. It was the most user-friendly source I could find and is the only reason the project I am submitting functions to a reasonable degree. The input shape was (128, 128, 3), representing a simple input RGB image. Then, the output was a (128, 128, 5) array representing the five possible classes the model could select.***4.4.2***

4.4.3 Loss and Metrics

Several metrics were chosen to compare the models in the original project proposal. Each metric was tested against, and the results came back abysmal. This will be touched on in the evaluation portion of the project.

	precision	recall	f1-score	support
0	0.9023	0.9961	0.9469	499078
1	0.0000	0.0000	0.0000	17874
2	0.6154	0.1149	0.1936	32764
3	0.0000	0.0000	0.0000	3376
4	0.0000	0.0000	0.0000	3964
accuracy			0.8992	557056
macro avg	0.3036	0.2222	0.2281	557056
weighted avg	0.8446	0.8992	0.8597	557056

The evaluation metrics.

The complexity of the dataset and models brought on an oversight. The loss function was sparse categorical cross-entropy from the Tensorflow library and again chosen by the critical references. “Since this is a multiclass classification problem, use the tf.keras.losses.SparseCategoricalCrossentropy loss function with the from_logits argument set to True, since the labels are scalar integers instead of vectors of scores for each pixel of every class." - reference 25.

4.5 Code Explanation

4.5.1 Data Processing

The below code imports and sets up the data for processing, using helper functions to resize the images and create a mask based on the associated annotation data. Finally, a tensor is created, which the neural network uses to process the image data.

```python
# Create a storage container.
dataset = {}

# Parse the XML file
# Code reference: [15]
with open("./object_detection/annotations.xml", "r") as f:
    # Import the annotations in a dictionary format.
    annotations = xmltodict.parse(f.read())

# Loop through every image object in the annotations and store the image.
for image in annotations["annotations"]["image"]:
    # Get image name.
    image_name = image["@name"]

    # Load the image with scikit learn.
    photo = ski.io.imread("./object_detection/annotated_images" + '/' + image_name)

    # Create a resized image (224x224) in the storage container.
    # We'll need to store the images in 224x224 to standardize them for the neural networks
    resized_image = ski.transform.resize(photo, (128, 128), anti_aliasing=True)

    # Resize and standardize the polygon points
    resized_points = resize_points(image['polygon'], photo.shape[1], photo.shape[0])

    # Get the mask.
    mask = get_mask(resized_points)

    # Store the resized image and all associated masks.
    dataset[image_name] = {'image': resized_image,
                           'tensor': normalize(resized_image),
                           'mask': mask}
```

Data import code block.

4.5.2 resize_points

Resize_points is a helper function used in the code block below. However, it deserves its highlight. This function resizes the annotations based on a new desired size. This function is primarily based on code from before the midterm. As I later learned, I could have simply created the full-quality mask and resized that image from there, so the below is over-engineered. However, it broke me out of one of this project's main challenges: resizing the pictures for consistent use in neural network inputs.

It works through the string data from the XML import and splits the point data into neat lists, which are then manipulated based on a scalar value to adjust the data to the new desired size.

```python
def resize_points(polygons, width, height):
    """
    Resizes polygon point coordinates to fit a 512x512 image resolution.

    Parameters:
        polygons (list of dict): A list of polygon annotations, each containing:
            - '@label' (str): The polygon's name (e.g., 'Post', 'Sign').
            - '@points' (str): A semicolon-separated string of x,y coordinates.
        width (int): The original image width.
        height (int): The original image height.

    Returns:
        dict: A dictionary with labels as keys ('Post', 'Sign', etc.) and their corresponding resized points as NumPy arrays. Missing labels are set to None.
    """
    # Create a container and set zero values for potential missing items.
    resized_polygons = {'Post': None, # Class 1
                        'Sign': None, # Class 2
                        'Lower Plate': None, # Class 3
                        'Top Plate': None} # Class 4

    # Loop through each polygon.
    for polygon in polygons:
        # Run within a try block as some odd exporting polygons exist in the annotation file.
        try:
            # Create storage for the related points in this polygon.
            points = polygon['@points']

            # Get the image scale based on the x and y factors.
            scale_x = 128 / width
            scale_y = 128 / height

            # Parse the polygon points.
            points = points.split(';')
            # Split the points and convert to float format. Reference 22 helped with this task.
            points = np.array([list(map(float, point.split(','))) for point in points])

            # Scale the points to the new resolution.
            scaled_points = []
            for point in points:
                scaled_points.append([point[0] * scale_x, point[1] * scale_y])

            # Store the scaled points in the container.
            resized_polygons[polygon['@label']] = np.array(scaled_points)
        except:
            print('Invalid polygon provided', polygon, '\n')

    # Return stored points
    return resized_polygons
```

The resize_points function.

4.5.3 U-Net model Code

Below is the code to build the U-NET model. Again, this is taken directly from references 25 and 27, but it's an integral part of this project. We can see how the model takes an input array and sends it through the neural network by downsampling it through each layer and then upsampling it back to its prediction, which is then output into a generated mask.

```python
# Model code was taken directly from reference 25.
def unet_model(output_channels:int):
  inputs = tf.keras.layers.Input(shape=[128, 128, 3])

  # Downsampling through the model
  skips = down_stack(inputs)
  x = skips[-1]
  skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    concat = tf.keras.layers.Concatenate()
    x = concat([x, skip])

  # This is the last layer of the model
  last = tf.keras.layers.Conv2DTranspose(
      filters=output_channels, kernel_size=3, strides=2,
      padding='same')  #64x64 -> 128x128

  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)
```

U-NET model code.

4.6 Visual Representation of the Project

All meaningful visualizations can be found in the Jupyter notebook.

4.7 Project Evaluation

As presented now, the final project is a well-built system that relies on heavy research and quality code implementations. It is an objective failure in that light when compared to what I initially set out to do. However, that will be left for the evaluation section.

Regarding the presented items, the curated dataset and the loading of the dataset into a base model are of a quality nature. The dataset took many man-hours to produce in terms of sourcing and properly annotating the photos. Any more hours than originally planned for as the midterm proposal indicated we wanted 500 annotated images, and we are well short of that in the 170 range. This highlights the complexity of larger projects with 1000s of annotated pictures and just how many man hours go into setting up a project of this nature.

The images were then imported using quality Python practices and code. Then, the neural network training was built on several hours of research, during which learning and understanding how models of this nature receive photos and process them took up many hours. Passing this kind of data was not as simple as I thought and proved another major hurdle in this project. Neural networks, while excellent at finding patterns within data, can be quite finicky on how they process data, and in my personal experience, finding quality, easy-to-digest resources on image segmentation is quite tricky. I was navigating a dark area with a lousy flashlight for much of this project. I am happy with what I achieved, but it is clear I did not meet my original expectations due to my many struggles in a project of this nature.

5 Evaluation (Word count 1123)

To evaluate the project, we will examine it from three perspectives: the template it follows, the project proposal, and the metrics used to measure the accuracy of the base model.

5.1 Project Template

- **Goal one** (The aim of this project is to gather a dataset of a moderate size (100s), including both training and testing sets.)
    - The project completed this goal with a dataset of approximately 200 images.
- **Goal two** (You should label the images and train a machine learning model, based on a pre-trained model, on the dataset.)
    - The project successfully labelled the test said with annotations using CVAT and split the dataset into four classes (post, sign, lower plat and top plate).
    - The project successfully trained a pre-trained model on the dataset as the base model.
- **Goal three** (You should try a number of variant models, from simple to complex and using different pre-trained models if they exist, and compare the results. Iterative development should include both improvements to the model and dataset.)
    - The project failed to train variant models on the dataset and thus did not have any results to compare.
    - The project also failed to display the model's iterative development. However, it did show improvements to the dataset as its size grew over time and the annotation process was simplified.

Overall, the project mostly met the criteria for the project template. It did falter toward the end, as no further models were trained past the baseline model, and thus, no comparative results could be displayed. However, most criteria were met well based on many research, planning, and implementation hours.

The submitted project is adequate when viewed through this lens. There are apparent faults, such as the lack of model comparison and the objective failure of the base model to predict anything of meaning. Clear pluses are also

present, mainly in terms of the quality of the dataset. The project's ambition was overpromised, and image segmentation is a very complex field of machine learning that I underestimated.

5.2 Project Proposal

From this perspective, the project was a failure. I initially embarked on a journey to create a complete end-to-end image anomaly detection algorithm. Luckily, I realized quickly that this was too ambitious for a project of this nature. The project then pivoted to something aligned with what I delivered at the end, in which an image segmentation algorithm would be built that can automatically create masks that highlight pixel-wise the post, sign, and plates in a given install photo.

The project proposal indicated that 500 images were to be annotated by the due date of the final project. The final submission failed to reach this goal with only 170 annotated images. It also aimed to have several different trained models based on the research in the literature review portion but failed to provide more than one. A pattern emerges here in that many of the original goals in the project proposal were not met.

This highlights two things. Number one was that the original proposal was overly ambitious when situating it within the project template. The dataset is abstract and unique, while the image segmentation task is within a complex image processing field. I underestimated both of these in terms of complexity. This project is not developing an algorithm to identify stop signs; it is creating an algorithm to identify pixel-wise accurate masks for complex photos with lots of noise. The number of references for similar projects was lacking, and leading figures in machine learning produced the ones I could find. These guides proved difficult, if not impossible, for me to understand with my current skill set.

The nature of the project highlights how easy it is to overestimate your skillset and overpromise on a project, especially when trying to produce a project that meets and exceeds the expectations laid out by the project guidelines. In this case, I luckily overshot the requirements by a large margin and still produced a reasonable project.

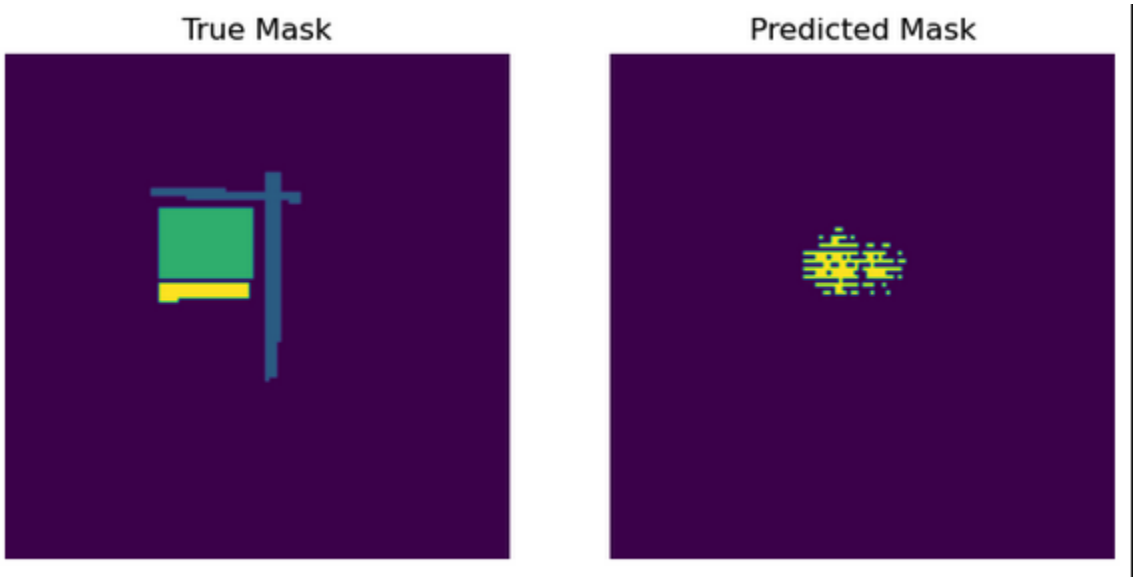
5.3 Model Evaluation

The base model, over its training epochs, obtained an accuracy in the high 80s at every epoch. The main issue with this training is the size of the dataset. Most projects of this nature have 1000s of data points to train on, and this project only had around 200. Due to the size, we had to add repeat to the training data; this caused the algorithm to repeat over the same data and thus arbitrarily inflate the accuracy.

```
Epoch 1/20
4/4 [==============================] - 3s 618ms/step - loss: 0.3393 - accuracy: 0.8919
Epoch 2/20
4/4 [==============================] - 2s 650ms/step - loss: 0.3376 - accuracy: 0.8928
Epoch 3/20
4/4 [==============================] - 2s 551ms/step - loss: 0.3366 - accuracy: 0.8911
Epoch 4/20
4/4 [==============================] - 2s 541ms/step - loss: 0.3397 - accuracy: 0.8933
Epoch 5/20
4/4 [==============================] - 2s 497ms/step - loss: 0.3365 - accuracy: 0.8919
Epoch 6/20
4/4 [==============================] - 2s 593ms/step - loss: 0.3357 - accuracy: 0.8933
Epoch 7/20
4/4 [==============================] - 2s 592ms/step - loss: 0.3291 - accuracy: 0.8939
Epoch 8/20
4/4 [==============================] - 3s 694ms/step - loss: 0.3293 - accuracy: 0.8931
Epoch 9/20
4/4 [==============================] - 2s 581ms/step - loss: 0.3349 - accuracy: 0.8939
Epoch 10/20
4/4 [==============================] - 2s 602ms/step - loss: 0.3359 - accuracy: 0.8912
Epoch 11/20
4/4 [==============================] - 3s 655ms/step - loss: 0.3276 - accuracy: 0.8944
Epoch 12/20
4/4 [==============================] - 2s 624ms/step - loss: 0.3403 - accuracy: 0.8899
Epoch 13/20
4/4 [==============================] - 3s 621ms/step - loss: 0.3295 - accuracy: 0.8932
Epoch 14/20
4/4 [==============================] - 2s 608ms/step - loss: 0.3223 - accuracy: 0.8955
Epoch 15/20
4/4 [==============================] - 2s 526ms/step - loss: 0.3237 - accuracy: 0.8941
Epoch 16/20
4/4 [==============================] - 2s 613ms/step - loss: 0.3233 - accuracy: 0.8936
Epoch 17/20
4/4 [==============================] - 2s 629ms/step - loss: 0.3328 - accuracy: 0.8915
Epoch 18/20
4/4 [==============================] - 2s 573ms/step - loss: 0.3206 - accuracy: 0.8944
Epoch 19/20
4/4 [==============================] - 2s 602ms/step - loss: 0.3207 - accuracy: 0.8968
Epoch 20/20
4/4 [==============================] - 2s 540ms/step - loss: 0.3245 - accuracy: 0.8920
```

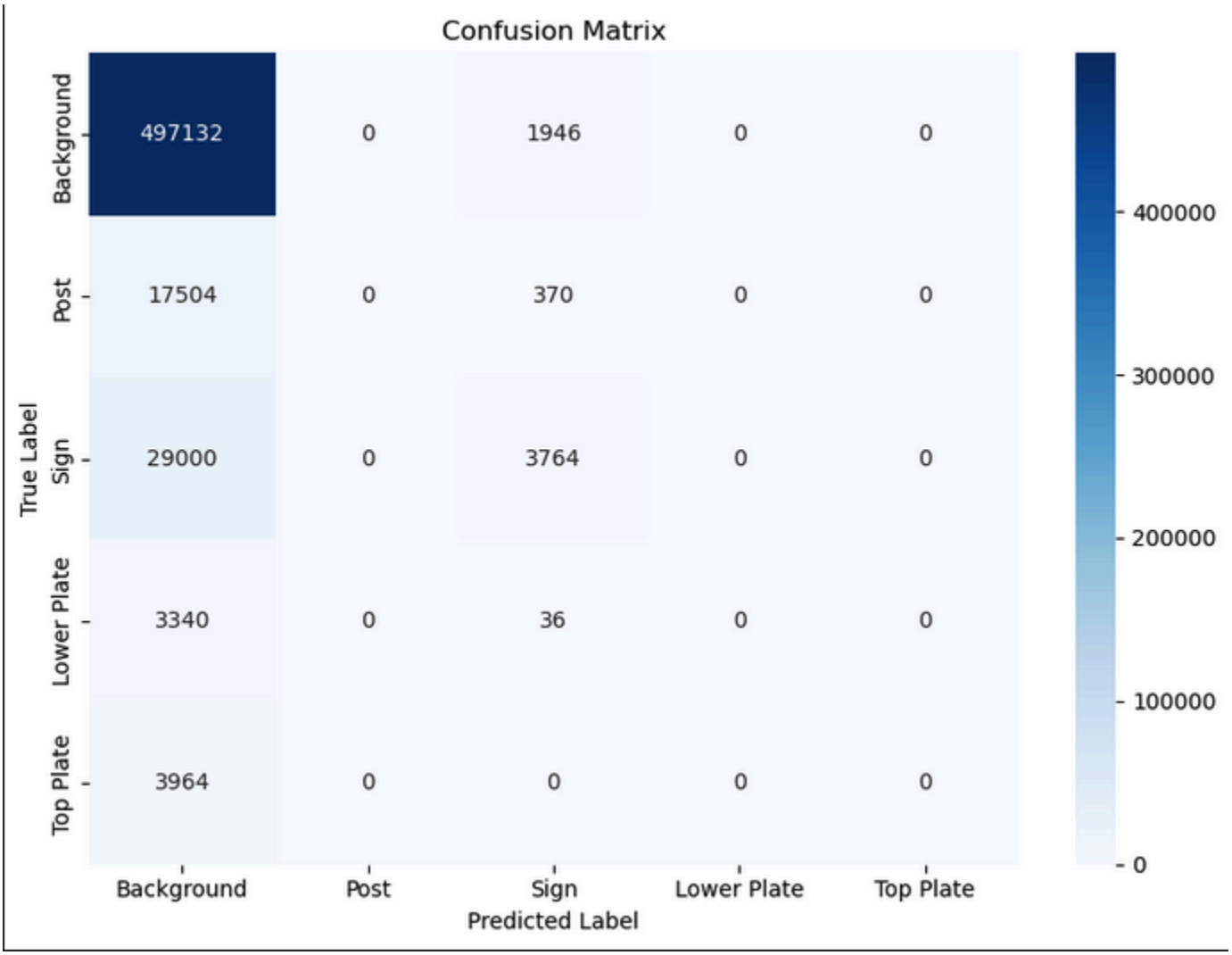
The training of the base model.

I think more accurate measures would be looking at the predicted mask on the test set, which shows abysmal results. The base model is unable to predict even the simplest mask in the photos, and again, I think the main reason is the dataset size. Just under 200 images is not enough to train any kind of machine learning algorithm properly.



An example of a predicted mask.

Finally, some other metrics were graphed, again showing abysmal results. The confusion matrix paints a very grim picture in which little information was properly allotted.



An example of a predicted mask.

Overall, a pretty abysmal picture is painted when evaluating the base model. It does not accurately predict next to anything correctly. The same, if not similar model was built and accurately predicted a differing dataset with 1000s of images in references 25 and 27. The only conclusion one can draw is that the dataset is insufficient, especially with this niche area on which to train the image segmentation model.

When looking at the accuracy per class, we can see that the two optional classes have no prediction, most likely due to the class imbalance. We already have a small dataset, but the top and lower plates were not present in every photo, which would lower the training data even further for these items, again limiting the success of the base model.

	precision	recall	f1-score	support
0	0.9023	0.9961	0.9469	499078
1	0.0000	0.0000	0.0000	17874
2	0.6154	0.1149	0.1936	32764
3	0.0000	0.0000	0.0000	3376
4	0.0000	0.0000	0.0000	3964
accuracy			0.8992	557056
macro avg	0.3036	0.2222	0.2281	557056
weighted avg	0.8446	0.8992	0.8597	557056

The evaluation metrics.

In this aspect, the project is an objective failure due to the dataset's size and the metrics produced by the base model. To flesh out this project, more images would need to be annotated, and different models would need to be explored.

5.4 User Feedback

The project as it sits was shown to coworkers on an informal basis. While it does not have much use in its current form, my coworkers provided support and were enthusiastic about the steps I have taken. My boss indicated this could become a legitimate work project should it be continued and a viable prototype be built. Of course, the current project does not solve any problems in the automation pipeline, so they could not provide much more past enthusiasm and indication of a potential future resource allotment depending on a successful fully-working prototype.

6 Conclusion (Word count 652)

This project represents a lot of what I have learned throughout this degree. From project management to software development, my skill set has been tested in a real-world setting, and it is clear I still have many areas to grow and develop as a professional. The marker can see clear relations to coursework throughout the degree, including agile software projects, software designs and development, data science, machine learning and neural networks, and artificial intelligence. Parts of what I have learned have been combined with my research skills, leading to this project's culmination.

The project started as an overambitious idea involving an entire end-to-end quality control pipeline using an image anomaly detection algorithm on real estate installation photos. After heavy scrutiny and research, it was refined into a more plausible project based on image segmentation. This involved collecting a quality dataset and annotating over 100 images using CVAT, then importing and processing the images within Python to be passed to a neural network for training and testing.

The project largely accomplished the requirements in the project template but failed to meet many of the original goals in the proposal. Highlighting how the project's success depends on the viewpoint from which you approach it.

There are still many successes and highlights to be taken away from this project, particularly the dataset and code quality, which I will pride myself on. Understanding the pre-processing required to get a neural network to train on this type of problem and data was highly time-consuming and challenging. Nevertheless, the data could be passed into the U-NET model, resulting in decent metrics per the model evaluation. In addition to this, the project successfully:

- Created a large quality dataset on a niche topic related to real estate post-installation photos.
- Imported and preprocessed the dataset to create consistent data for neural networks to work on.

- Trained a base model on the dataset to predict masks on data it has not yet been trained on.
- I relayed this project's journey and my struggles in getting to this final submission.

Throughout this journey, the project faced many setbacks. In particular, issues with time management. Unfortunately, several family events occurred during this project and tied me up for a large portion of the second half of the semester. This ultimately tested my ability to balance work, school and personal aspects. This compounded the over-ambitious nature of this project in which researching and understanding exactly how image segmentation algorithms work proved extremely difficult.

Upon review, the algorithm provided in the midterm was largely erroneous. Several items needed to be adjusted, and the proof of concept I presented was incorrect. The final project has significantly different code, which more accurately aligns with industry-standard applications. This required substantial reworking, which also took up the limited time I provided myself.

One learns that projects are not set in stone. They are fluid and subject to issues from areas you could not plan for. Especially with a single-person project, it’s easy to see how your limitations rear their heads in unfortunate manners when you don’t have help to accommodate for that. For example, in my agile software course, I had an individual who was an active software developer. They helped guide me through the problematic code areas where I struggled as a student. In this, I only had the internet and intuition to guide me.

Nevertheless, I am happy I put together something that adequately meets the project template outline for this project. Unfortunately, this project did not meet my usual standards. However, I am happy this semester caps off my journey throughout this degree.

To end this report, I want to thank the University of London and the many individuals who guided me and my peers throughout this journey. This degree will forever shape who I am and has been an integral part in setting me up for success, and for that, I am forever grateful.

7 References

[1] Liu, J., Xie, G., Wang, J., Li, S., Wang, C., Zheng, F., & Jin, Y. (2024). Deep industrial image anomaly detection: A survey. *Machine Intelligence Research*, *21*(1), 104–135. https://doi.org/10.1007/s11633-023-1459-z

[2] Chow, J. K., Su, Z., Wu, J., Tan, P. S., Mao, X., & Wang, Y. H. (2020). Anomaly detection of defects on concrete structures with the convolutional autoencoder. Advanced Engineering Informatics, 45, 101105. https://doi.org/10.1016/j.aei.2020.101105

[3] Haselmann, M., Gruber, D. P., & Tabatabai, P. (2018). Anomaly detection using deep learning based image completion. 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 1237–1242. https://doi.org/10.1109/ICMLA.2018.00201

[4] Walt, S. van der, Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., & Yu, T. (2014). Scikit-image: Image processing in Python. PeerJ, 2, e453. https://doi.org/10.7717/peerj.453

[5] Bergmann, P., Batzner, K., Fauser, M., Sattlegger, D., & Steger, C. (2021). The mvtec anomaly detection dataset: A comprehensive real-world dataset for unsupervised anomaly detection. International Journal of Computer Vision, 129(4), 1038–1059. https://doi.org/10.1007/s11263-020-01400-4

[6] R, Pawan & S, Shreyas & ., Mohana. (2024). Anomaly Detection for Real-World Industry Manufactured Products Using Computer Vision. 981-987. 10.1109/IDCIoT59759.2024.10468044.

[7] Scikit-image's documentation—Skimage 0. 24. 0 documentation. (n.d.). Retrieved 21 November 2024, from https://scikit-image.org/docs/stable/

[8] Documentation. (n.d.). CVAT. Retrieved 2 December 2024, from https://docs.cvat.ai/docs/

[9] Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object detection with deep learning: A review. IEEE Transactions on Neural Networks and Learning Systems, 30(11), 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865

[10] Diwan, T., Anirudh, G., & Tembhurne, J. V. (2023). Object detection using YOLO: Challenges, architectural successors, datasets and applications. Multimedia Tools and Applications, 82(6), 9243–9275. https://doi.org/10.1007/s11042-022-13644-y

[11] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector (No. arXiv:1512.02325). arXiv. https://doi.org/10.48550/arXiv.1512.02325

[12] Balanca, P. (2024). Balancap/ssd-tensorflow [Jupyter Notebook]. https://github.com/balancap/SSD-Tensorflow (Original work published 2017)

[13] Gupta, Anurag & Yadav, Darshan & Raj, Akash & Pathak, Ayushman. (2023). Real-Time Object Detection Using SSD MobileNet Model of Machine Learning. International Journal of Engineering and Computer Science. 12. 25729-25734. 10.18535/ijecs/v12i05.4735.

[14] Ramadan, L. (2016, July 15). Answer to 'How big should batch size and number of epochs be when fitting a model?' [Online post]. Stack Overflow. https://stackoverflow.com/a/38405970

[15] Raj, A. (2020, December 5). Xmltodict module in python: A practical reference - askpython. https://www.askpython.com/python-modules/xmltodict-module

[16] 4. A crash course on NumPy for images—Skimage 0.26.0rc0.dev0 documentation. (n.d.). Retrieved 15 December 2024, from https://scikit-image.org/docs/dev/user_guide/numpy_images.html

[17] Rescale, resize, and downscale—Skimage 0.25.0 documentation. (n.d.). Retrieved 15 December 2024, from https://scikit-image.org/docs/stable/auto_examples/transform/plot_rescale.html

[18] Tf. Keras. Applications. Mobilenet | tensorflow v2. 16. 1. (n.d.). TensorFlow. Retrieved 15 December 2024, from https://www.tensorflow.org/api_docs/python/tf/keras/applications/MobileNet

[19] Matplotlib. Patches. Polygon—Matplotlib 3. 9. 3 documentation. (n.d.). Retrieved 15 December 2024, from https://matplotlib.org/stable/api/_as_gen/matplotlib.patches.Polygon.html#examples-using-matplotlib-patches-polygon

[20] Python | os.listdir() method. (2019, May 17). GeeksforGeeks. https://www.geeksforgeeks.org/python-os-listdir-method/

[21] https://scikit-image.org/docs/stable/api/skimage.transform.html

[22] rain, superb. (2020, August 15). Answer to 'Efficient way to split and map float function on a string of points' [Online post]. Stack Overflow. https://stackoverflow.com/a/63428433

[23] Skimage. Draw—Skimage 0. 25. 2 documentation. (n.d.). Retrieved 26 March 2025, from https://scikit-image.org/docs/stable/api/skimage.draw.html#skimage.draw.polygon

[24] Kibe, K. (2023, March 17). Performing image segmentation using tensorflow. *Medium*. https://keviinkibe.medium.com/performing-image-segmentation-using-tensorflow-1c82608d2233

[25] *Image segmentation using tensorflow*. (2022, July 27). GeeksforGeeks. https://www.geeksforgeeks.org/image-segmentation-using-tensorflow/

[26] *Pix2pix: Image-to-image translation with a conditional gan | tensorflow core*. (n.d.). TensorFlow. Retrieved 31 March 2025, from https://www.tensorflow.org/tutorials/generative/pix2pix

[27] *Image segmentation | tensorflow core*. (n.d.). TensorFlow. Retrieved 31 March 2025, from https://www.tensorflow.org/tutorials/images/segmentation

[28] T, D. (2019, July 25). Confusion matrix visualization. *Medium*. https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea

[29] *Visualise the test images after training the model on segmentation task*. (2019, September 23). PyTorch Forums. https://discuss.pytorch.org/t/visualise-the-test-images-after-training-the-model-on-segmentation-task/56615

8 Appendix

1.0 Library Imports

Below we import the libraries we will be using in this notebook.

Marker note, references will be made if code was taken from or inspired by a source. The references can be found in the report.

In [1]:
```
%pip install tensorflow
%pip install os
%pip install numpy
%pip install skimage
%pip install sklearn
%pip install matplotlib
%pip install xmltodict
%pip install pix2pix
%pip install git+https://github.com/tensorflow/examples.git
```

```
Requirement already satisfied: tensorflow in /opt/anaconda3/envs/london/lib/python3.11/si
te-packages (2.12.0)
Requirement already satisfied: absl-py>=1.0.0 in /opt/anaconda3/envs/london/lib/python3.1
1/site-packages (from tensorflow) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from tensorflow) (2.0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /opt/anaconda3/envs/london/lib/pyth
on3.11/site-packages (from tensorflow) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /opt/anaconda3/envs/london/lib/pyth
on3.11/site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /opt/anaconda3/envs/london/lib/python3.11/s
ite-packages (from tensorflow) (3.11.0)
Requirement already satisfied: jax>=0.3.15 in /opt/anaconda3/envs/london/lib/python3.11/s
ite-packages (from tensorflow) (0.4.30)
Requirement already satisfied: libclang>=13.0.0 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from tensorflow) (18.1.1)
Requirement already satisfied: numpy<1.24,>=1.22 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from tensorflow) (1.23.5)
Requirement already satisfied: opt-einsum>=2.3.2 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /opt/anaconda3/envs/london/lib/python3.11/sit
e-packages (from tensorflow) (23.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.2
1.5,<5.0.0dev,>=3.20.3 in /opt/anaconda3/envs/london/lib/python3.11/site-packages (from t
ensorflow) (3.20.3)
Requirement already satisfied: setuptools in /opt/anaconda3/envs/london/lib/python3.11/si
te-packages (from tensorflow) (69.5.1)
Requirement already satisfied: six>=1.12.0 in /opt/anaconda3/envs/london/lib/python3.11/s
ite-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from tensorflow) (2.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /opt/anaconda3/envs/london/li
b/python3.11/site-packages (from tensorflow) (4.11.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /opt/anaconda3/envs/london/lib/pyth
on3.11/site-packages (from tensorflow) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /opt/anaconda3/envs/london/lib/pyth
on3.11/site-packages (from tensorflow) (1.48.2)
Requirement already satisfied: tensorboard<2.13,>=2.12 in /opt/anaconda3/envs/london/lib/
python3.11/site-packages (from tensorflow) (2.12.1)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /opt/anaconda3/envs/
london/lib/python3.11/site-packages (from tensorflow) (2.12.0)
Requirement already satisfied: keras<2.13,>=2.12.0 in /opt/anaconda3/envs/london/lib/pyth
on3.11/site-packages (from tensorflow) (2.12.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /opt/anaconda3/envs/london/lib/pytho
n3.11/site-packages (from astunparse>=1.6.0->tensorflow) (0.35.1)
Requirement already satisfied: jaxlib<=0.4.30,>=0.4.27 in /opt/anaconda3/envs/london/lib/
python3.11/site-packages (from jax>=0.3.15->tensorflow) (0.4.30)
Requirement already satisfied: ml-dtypes>=0.2.0 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from jax>=0.3.15->tensorflow) (0.5.1)
Requirement already satisfied: scipy>=1.9 in /opt/anaconda3/envs/london/lib/python3.11/si
te-packages (from jax>=0.3.15->tensorflow) (1.13.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in /opt/anaconda3/envs/london/lib/py
thon3.11/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (2.29.0)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /opt/anaconda3/envs/lond
on/lib/python3.11/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (0.5.2)
Requirement already satisfied: markdown>=2.6.8 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (3.4.1)
Requirement already satisfied: requests<3,>=2.21.0 in /opt/anaconda3/envs/london/lib/pyth
on3.11/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (2.32.2)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /opt/anaconda3/en
```

```
vs/london/lib/python3.11/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (0.7.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /opt/anaconda3/envs/londo
n/lib/python3.11/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (1.8.1)
Requirement already satisfied: werkzeug>=1.0.1 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (3.0.3)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /opt/anaconda3/envs/london/lib/p
ython3.11/site-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow)
(5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /opt/anaconda3/envs/london/lib/py
thon3.11/site-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow)
(0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4 in /opt/anaconda3/envs/london/lib/python3.1
1/site-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow) (4.7.2)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /opt/anaconda3/envs/london/li
b/python3.11/site-packages (from google-auth-oauthlib<1.1,>=0.5->tensorboard<2.13,>=2.12-
>tensorflow) (1.3.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anaconda3/envs/london/li
b/python3.11/site-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflo
w) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/envs/london/lib/python3.11/
site-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/envs/london/lib/pytho
n3.11/site-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow)
(2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/envs/london/lib/pytho
n3.11/site-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow) (202
5.1.31)
Requirement already satisfied: MarkupSafe>=2.1.1 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from werkzeug>=1.0.1->tensorboard<2.13,>=2.12->tensorflow) (2.1.3)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /opt/anaconda3/envs/london/lib/pyt
hon3.11/site-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.1
3,>=2.12->tensorflow) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<1.1,>=0.5->tensor
board<2.13,>=2.12->tensorflow) (3.2.2)
Note: you may need to restart the kernel to use updated packages.
ERROR: Could not find a version that satisfies the requirement os (from versions: none)
ERROR: No matching distribution found for os
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: numpy in /opt/anaconda3/envs/london/lib/python3.11/site-pa
ckages (1.23.5)
Note: you may need to restart the kernel to use updated packages.
Collecting skimage
  Using cached skimage-0.0.tar.gz (757 bytes)
  Preparing metadata (setup.py) ... error
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  └─> [3 lines of output]

      *** Please install the `scikit-image` package (instead of `skimage`) ***

      [end of output]

  note: This error originates from a subprocess, and is likely not a problem with pip.
error: metadata-generation-failed

× Encountered error while generating package metadata.
└─> See above for output.

note: This is an issue with the package mentioned above, not pip.
```

```
hint: See above for details.
Note: you may need to restart the kernel to use updated packages.
Collecting sklearn
  Using cached sklearn-0.0.post12.tar.gz (2.6 kB)
  Preparing metadata (setup.py) ... error
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  ╰─> [15 lines of output]
      The 'sklearn' PyPI package is deprecated, use 'scikit-learn'
      rather than 'sklearn' for pip commands.

      Here is how to fix this error in the main use cases:
      - use 'pip install scikit-learn' rather than 'pip install sklearn'
      - replace 'sklearn' by 'scikit-learn' in your pip requirements files
        (requirements.txt, setup.py, setup.cfg, Pipfile, etc ...)
      - if the 'sklearn' package is used by one of your dependencies,
        it would be great if you take some time to track which package uses
        'sklearn' instead of 'scikit-learn' and report it to their issue tracker
      - as a last resort, set the environment variable
        SKLEARN_ALLOW_DEPRECATED_SKLEARN_PACKAGE_INSTALL=True to avoid this error

      More information is available at
      https://github.com/scikit-learn/sklearn-pypi-package
      [end of output]

  note: This error originates from a subprocess, and is likely not a problem with pip.
error: metadata-generation-failed

× Encountered error while generating package metadata.
╰─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: matplotlib in /opt/anaconda3/envs/london/lib/python3.11/si
te-packages (3.8.4)
Requirement already satisfied: contourpy>=1.0.1 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /opt/anaconda3/envs/london/lib/python3.11/
site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.21 in /opt/anaconda3/envs/london/lib/python3.11/s
ite-packages (from matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=8 in /opt/anaconda3/envs/london/lib/python3.11/sit
e-packages (from matplotlib) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/anaconda3/envs/london/lib/python
3.11/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /opt/anaconda3/envs/london/lib/pyt
hon3.11/site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/envs/london/lib/python3.11/sit
e-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: xmltodict in /opt/anaconda3/envs/london/lib/python3.11/sit
e-packages (0.13.0)
Note: you may need to restart the kernel to use updated packages.
```

```
ERROR: Could not find a version that satisfies the requirement pix2pix (from versions: no
ne)
ERROR: No matching distribution found for pix2pix
Note: you may need to restart the kernel to use updated packages.
Collecting git+https://github.com/tensorflow/examples.git
  Cloning https://github.com/tensorflow/examples.git to /private/var/folders/r4/mvngz2z96
txg8fqvpmt13pr40000gn/T/pip-req-build-ge_1uicb
  Running command git clone --filter=blob:none --quiet https://github.com/tensorflow/exam
ples.git /private/var/folders/r4/mvngz2z96txg8fqvpmt13pr40000gn/T/pip-req-build-ge_1uicb
  Resolved https://github.com/tensorflow/examples.git to commit fed63294c10c71b7da028e0e7
5de9ff68ac56d17
  Preparing metadata (setup.py) ... done
Requirement already satisfied: absl-py in /opt/anaconda3/envs/london/lib/python3.11/site-
packages (from tensorflow-examples==0.1741282361.145486042149443330204603306743204048930
0359736599) (2.1.0)
Requirement already satisfied: six in /opt/anaconda3/envs/london/lib/python3.11/site-pack
ages (from tensorflow-examples==0.1741282361.145486042149443330204603306743204048930035
97 36599) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```python
# Tensorflow imports.
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatena
from tensorflow_examples.models.pix2pix import pix2pix

# OS import to access files.
import os

# Numpy Import for array normalizing.
import numpy as np

# skimage import for photo processing.
import skimage as ski
import sklearn as skl
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# matplot import for plotting.
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
from matplotlib.collections import PatchCollection
import seaborn as sns

# xmltodict import is used to parse the annotation file.
import xmltodict
```

2.0 Load Dataset with Annotations

Below we need to load in the dataset with the corresponding annotations. We will also be resizing the dataset to 512x512 images to ensure we have consistency for loading in the neural networks.

2.1 Define resize_points function

In [3]:
```python
def resize_points(polygons, width, height):
    """
    Resizes polygon point coordinates to fit a 512x512 image resolution.
```

```
    Parameters:
        polygons (list of dict): A list of polygon annotations, each containing:
            - '@label' (str): The polygon's name (e.g., 'Post', 'Sign').
            - '@points' (str): A semicolon-separated string of x,y coordinates.
        width (int): The original image width.
        height (int): The original image height.

    Returns:
        dict: A dictionary with labels as keys ('Post', 'Sign', etc.) and their correspo
    """
    # Create a container and set zero values for potential missing items.
    resized_polygons = {'Post': None, # Class 1
                        'Sign': None, # Class 2
                        'Lower Plate': None, # Class 3
                        'Top Plate': None} # Class 4

    # Loop through each polygon.
    for polygon in polygons:
        # Run within a try block as some odd exporting polygons exist in the annotation
        try:
            # Create storage for the related points in this polygon.
            points = polygon['@points']

            # Get the image scale based on the x and y factors.
            scale_x = 128 / width
            scale_y = 128 / height

            # Parse the polygon points.
            points = points.split(';')
            # Split the points and convert to float format. Reference 22 helped with thi
            points = np.array([list(map(float, point.split(','))) for point in points])

            # Scale the points to the new resolution.
            scaled_points = []
            for point in points:
                scaled_points.append([point[0] * scale_x, point[1] * scale_y])

            # Store the scaled points in the container.
            resized_polygons[polygon['@label']] = np.array(scaled_points)
        except:
            print('Invalid polygon provided', polygon, '\n')

    # Return stored points
    return resized_polygons
```

2.2 Define get_mask function

In [4]:

```
def get_mask(points):
    """
    Generates a segmentation mask from polygon annotations.

    Parameters:
        points (dict): A dictionary where keys are object classes ('Post', 'Sign', 'Lowe
        If a value is None, it is skipped.

    Returns:
        np.ndarray: A 128x128 numpy array mask where:
                    0 = background
                    1 = post
                    2 = sign
```

```
                    3 = lower plate
                    4 = top plate
    """
    # Create mask array.
    mask = np.zeros((128, 128), dtype = np.uint8)

    for key in points:
        polygon = points[key]
        # Continue if polygon is None.
        if polygon is None:
            continue

        # Get mask based on polygon.
        # Code from references 23 and 25. 25 was instrumental in getting this properly s
        rr, cc = ski.draw.polygon(polygon[:, 1], polygon[:, 0], mask.shape)

        # Overlay mask with pixel value depending on the class of polygon.
        # Class 1 = post
        # Class 2 = sign
        # Class 3 = Lower Plate
        # Class 4 = Top Plate
        if key == 'Post':
            mask[rr, cc] = 1
        elif key == 'Sign':
            mask[rr, cc] = 2
        elif key == 'Lower Plate':
            mask[rr, cc] = 3
        elif key == 'Top Plate':
            mask[rr, cc] = 4

    # Return the mask
    return mask
```

2.3 Define normalize function to store image as tensor.

In [5]:
```
def normalize(input_image):
    """
    Normalizes an image's pixel values to the range [0, 1].

    Parameters:
        input_image (tf.Tensor or np.ndarray): The input image with pixel values in the

    Returns:
        tf.Tensor: The normalized image with pixel values scaled to [0, 1].
    """
    # Normalize the pixel range values between [0:1]
    # Taken from reference 25.
    image = tf.cast(input_image, dtype = tf.float32) / 255.0
    return image
```

2.3 Load and Resize the Images/Masks

Below we will load in the images, resize them and load in and reszie the associated masks. This essentially sets up the dataset for use in machine learning algorithms.

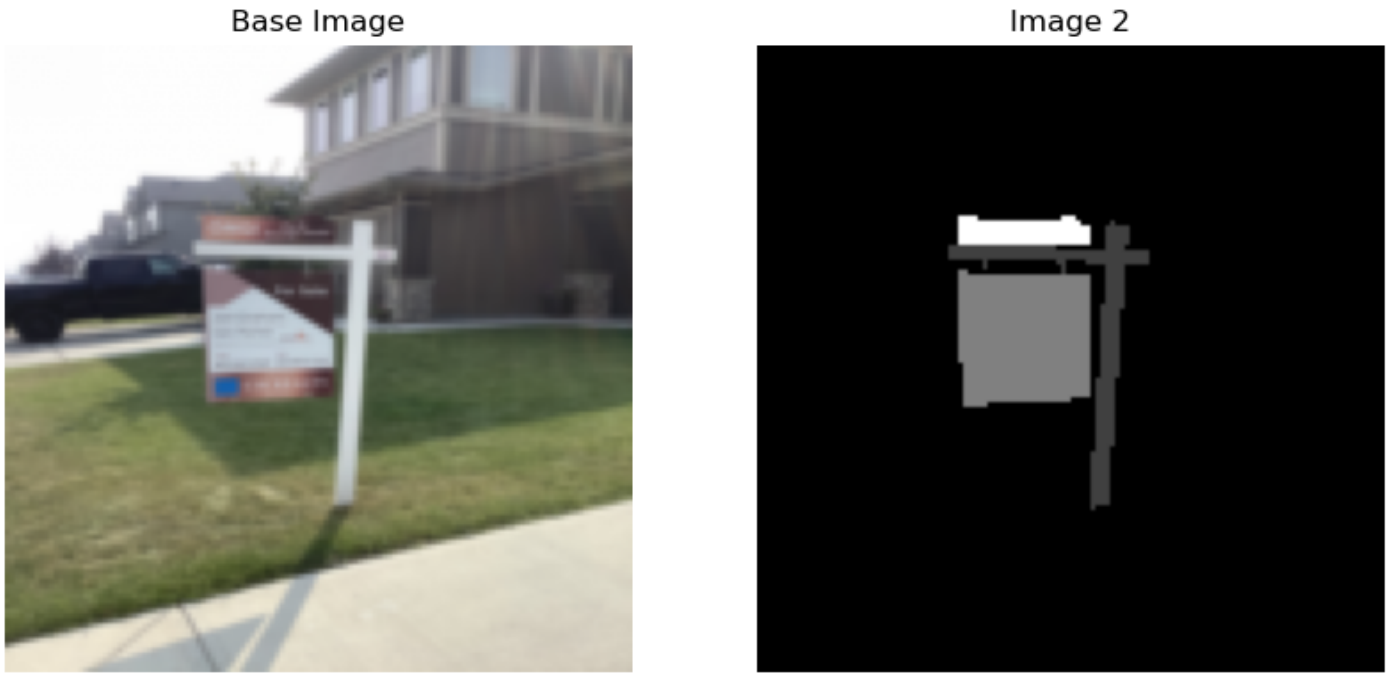
In [6]:
```
# Create a storage containers.
dataset = {}
annotations = []
```

```python
# Parse the XML file
# Code reference: [15]
with open("./object_detection/annotations.xml", "r") as f:
    # Import the annotations in a dictionary format.
    annotations = xmltodict.parse(f.read())

# Loop through every image object in the annotations and store the image.
for image in annotations["annotations"]["image"]:
    # Get image name.
    image_name = image["@name"]

    # Load the image with scikit learn.
    photo = ski.io.imread("./object_detection/annotated_images" + '/' + image_name)

    # Create a resized image (224x224) in the storage container.
    # We'll need to store the images in 224x224 to standardize them for the neural netwo
    resized_image = ski.transform.resize(photo, (128, 128), anti_aliasing=True)

    # Resize and standardize the polygon points
    resized_points = resize_points(image['polygon'], photo.shape[1], photo.shape[0])

    # Get the mask.
    mask = get_mask(resized_points)

    # Store the resized image and all associated masks.
    dataset[image_name] = {'image': resized_image,
                           'tensor': normalize(resized_image),
                           'mask': mask}
```

```
Invalid polygon provided @label

Invalid polygon provided @source

Invalid polygon provided @occluded

Invalid polygon provided @points

Invalid polygon provided @z_order

Invalid polygon provided @label

Invalid polygon provided @source

Invalid polygon provided @occluded

Invalid polygon provided @points

Invalid polygon provided @z_order

Invalid polygon provided @label

Invalid polygon provided @source

Invalid polygon provided @occluded

Invalid polygon provided @points

Invalid polygon provided @z_order
```

2.4 Confirm Masks Conform to Resized Image

In [7]:
```
# Choose a random image
image_info = dataset['f_1723741777.jpg']

# Load photos.
photo = image_info['image']
photo_mask = photo.copy()

# Layer masks over photo.
mask = image_info['mask']

# Create plot.
fig, axes = plt.subplots(1, 2, figsize = (10, 5))

# Show base image
axes[0].imshow(photo, cmap='gray')
axes[0].set_title("Base Image")
axes[0].axis('off')

# Show image with mask.
axes[1].imshow(mask, cmap='gray')
axes[1].set_title("Image 2")
axes[1].axis('off')

# Show plot.
plt.show()
```



Above we can see the mask does align with the corresponding image and highlights the items we want to detect in a pixel-wise manner.

2.5 Convert data to Tensors Sets

In [8]:
```
# Convert dictionary values to lists
images = [v['tensor'] for v in dataset.values()]
masks = [v['mask'] for v in dataset.values()]
```

```
# Convert to numpy arrays
images = np.array(images, dtype=np.float32)
masks = np.array(masks, dtype=np.int32)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(images, masks, test_size=0.1, random
```

In [9]:
```
BATCH_SIZE = 32
# Get the train dataset. We set it to repeat due to the small size of the dataset.
train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train)).shuffle(1000).batch(BA

# Get the test dataset.
test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test)).batch(BATCH_SIZE).repeat(

# Set
EPOCHS = 20
steps_per_epoch = len(X_train) // BATCH_SIZE
validation_steps = len(X_test) // BATCH_SIZE
```

3.0 Set up Base Model Architecture

Below we will set up the base model and run it individually on the different masks. Most of the code below was heavily inspired or taken from reference 23.

In [10]:
```
# Load in the base model.
base_model = tf.keras.applications.MobileNetV2(input_shape=[128, 128, 3], include_top=Fa

# Use the activations of these layers
layer_names = [
    'block_1_expand_relu',   # 64x64
    'block_3_expand_relu',   # 32x32
    'block_6_expand_relu',   # 16x16
    'block_13_expand_relu',  # 8x8
    'block_16_project',      # 4x4
]

base_model_outputs = [base_model.get_layer(name).output for name in layer_names]

# Create the feature extraction model
down_stack = tf.keras.Model(inputs=base_model.input, outputs=base_model_outputs)

down_stack.trainable = False
```

In [11]:
```
# Load in stacks for decoder.
up_stack = [
    pix2pix.upsample(512, 3),  # 4x4 -> 8x8
    pix2pix.upsample(256, 3),  # 8x8 -> 16x16
    pix2pix.upsample(128, 3),  # 16x16 -> 32x32
    pix2pix.upsample(64, 3),   # 32x32 -> 64x64
]
```

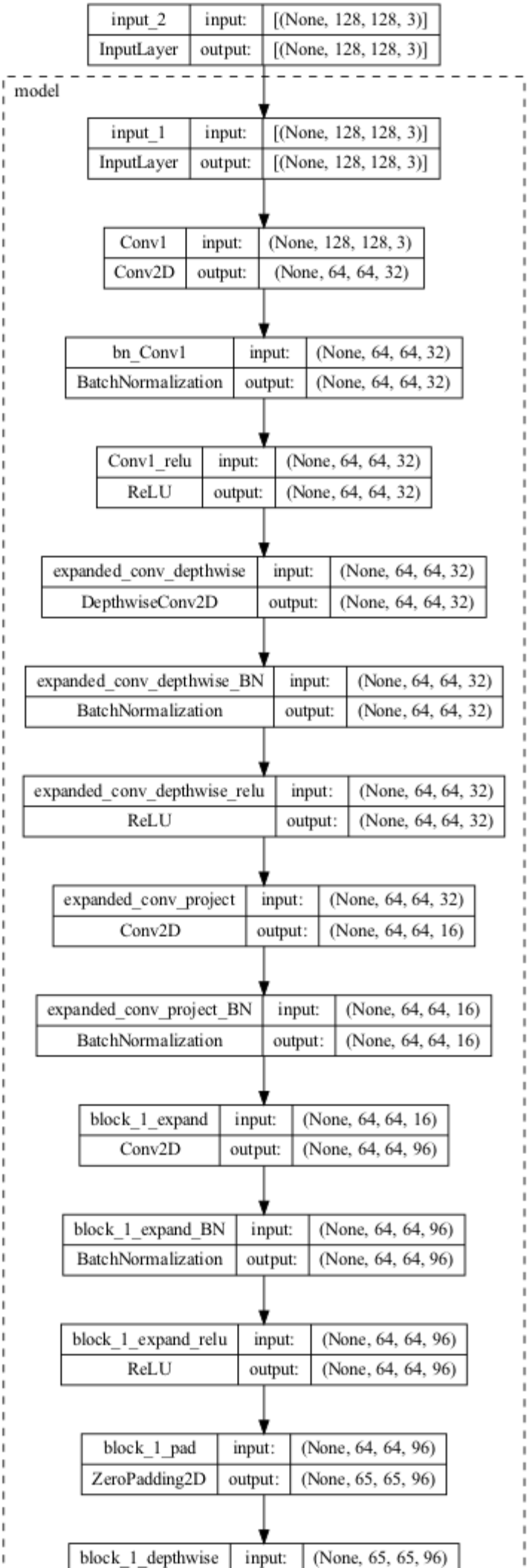
In [12]:
```
# Model code was taken directly from reference 27.
def unet_model(output_channels:int):
  inputs = tf.keras.layers.Input(shape=[128, 128, 3])

  # Downsampling through the model
  skips = down_stack(inputs)
  x = skips[-1]
```

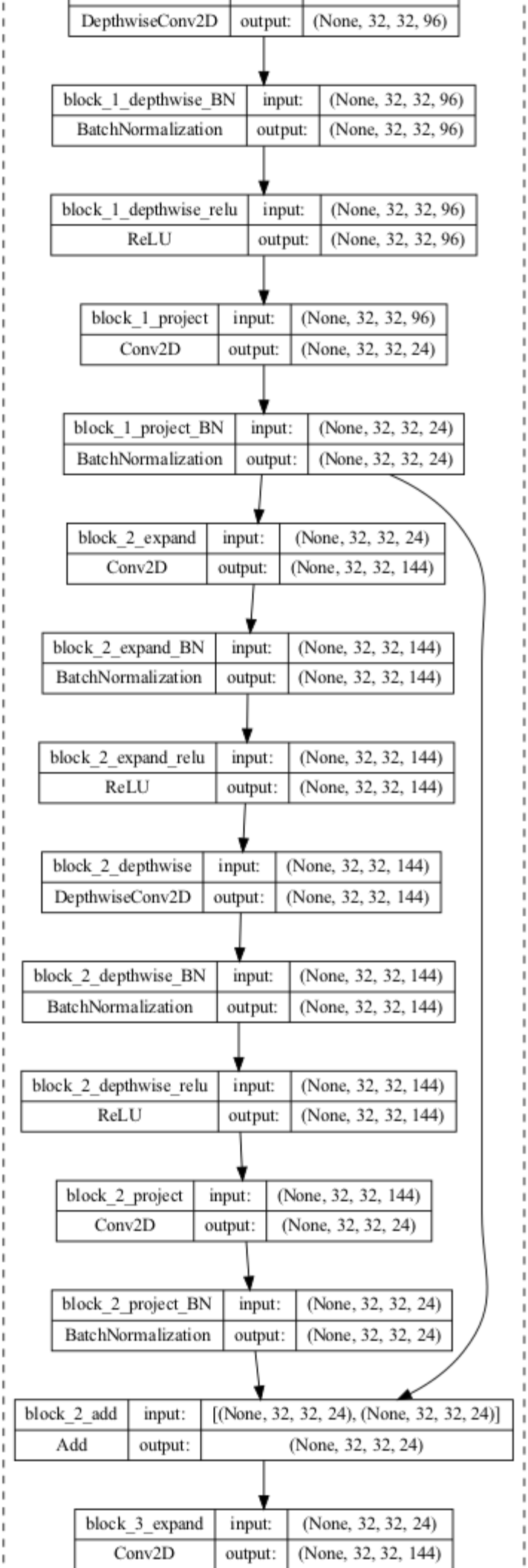
```
  skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    concat = tf.keras.layers.Concatenate()
    x = concat([x, skip])

  # This is the last layer of the model
  last = tf.keras.layers.Conv2DTranspose(
      filters=output_channels, kernel_size=3, strides=2,
      padding='same')  #64x64 -> 128x128

  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)
```

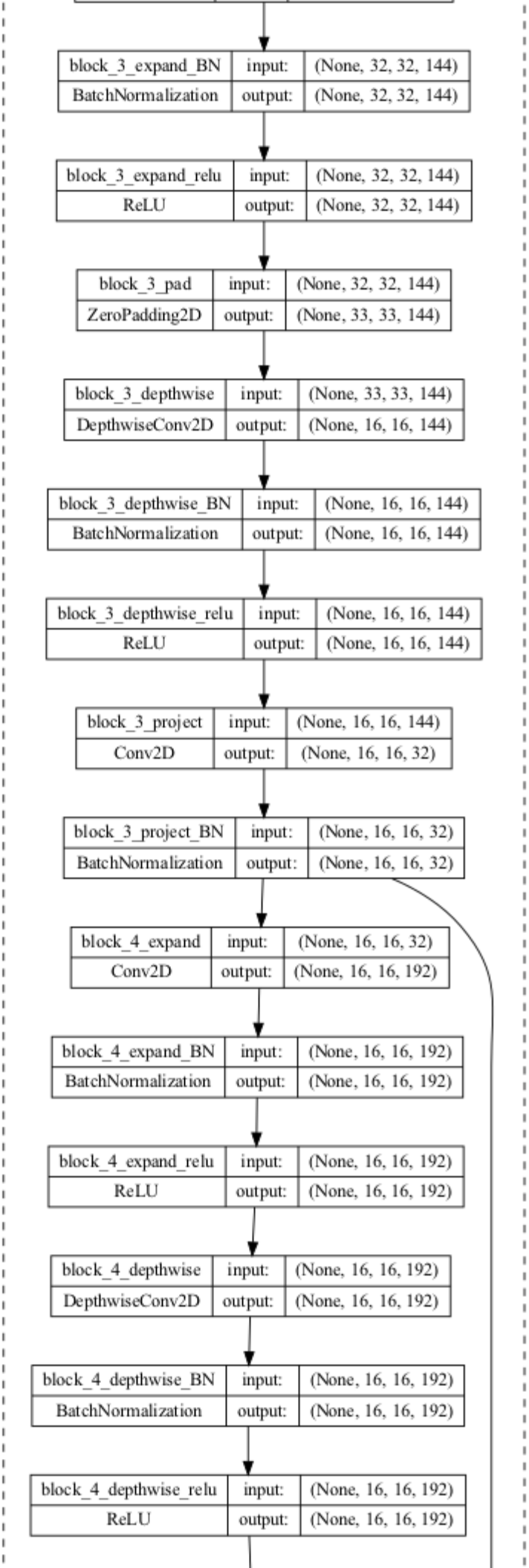
In [13]:

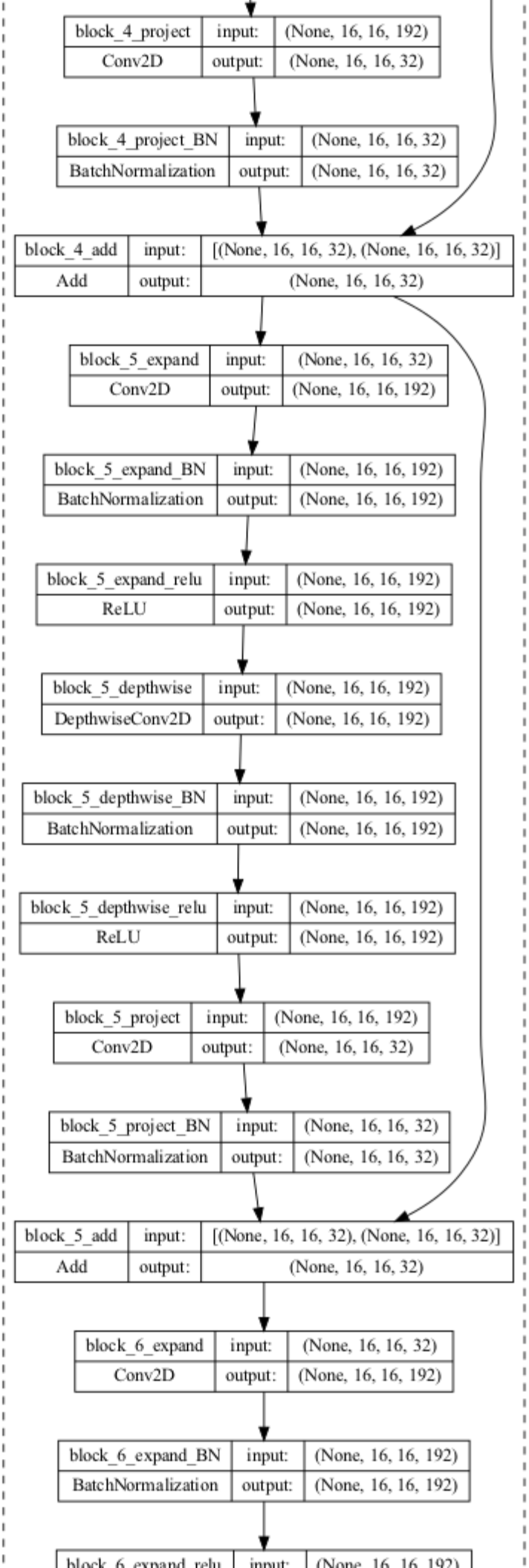
```
# Compile the model.
model = unet_model(output_channels=5)
model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_

# Plot the model.
tf.keras.utils.plot_model(model, show_shapes=True, expand_nested=True, dpi=64)
```

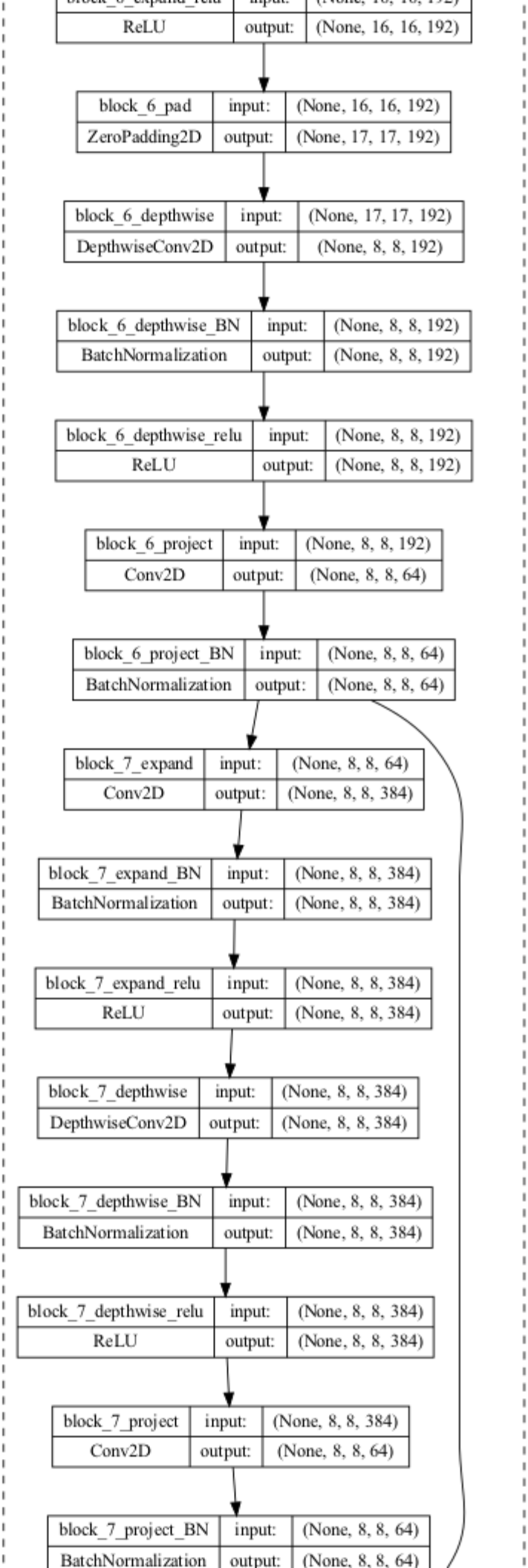
Out[13]:

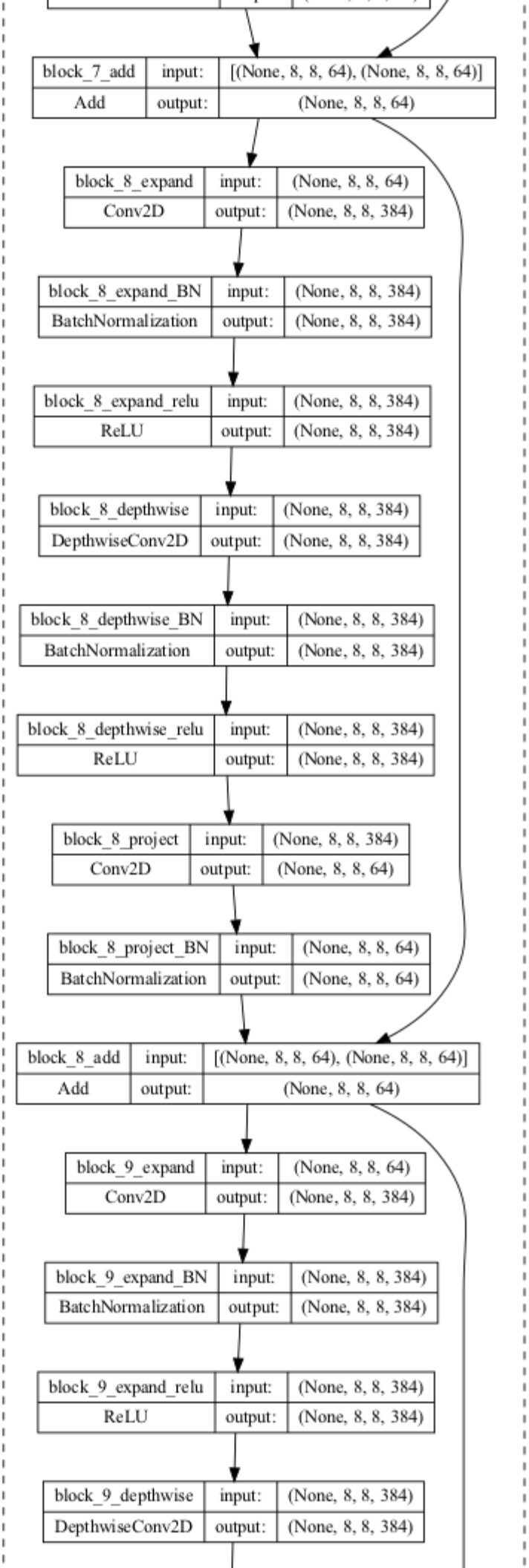

input_2 | input: | [(None, 128, 128, 3)]
InputLayer | output: | [(None, 128, 128, 3)]
model
input_1 | input: | [(None, 128, 128, 3)]
InputLayer | output: | [(None, 128, 128, 3)]
Conv1 | input: | (None, 128, 128, 3)
Conv2D | output: | (None, 64, 64, 32)
bn_Conv1 | input: | (None, 64, 64, 32)
BatchNormalization | output: | (None, 64, 64, 32)
Conv1_relu | input: | (None, 64, 64, 32)
ReLU | output: | (None, 64, 64, 32)
expanded_conv_depthwise | input: | (None, 64, 64, 32)
DepthwiseConv2D | output: | (None, 64, 64, 32)
expanded_conv_depthwise_BN | input: | (None, 64, 64, 32)
BatchNormalization | output: | (None, 64, 64, 32)
expanded_conv_depthwise_relu | input: | (None, 64, 64, 32)
ReLU | output: | (None, 64, 64, 32)
expanded_conv_project | input: | (None, 64, 64, 32)
Conv2D | output: | (None, 64, 64, 16)
expanded_conv_project_BN | input: | (None, 64, 64, 16)
BatchNormalization | output: | (None, 64, 64, 16)
block_1_expand | input: | (None, 64, 64, 16)
Conv2D | output: | (None, 64, 64, 96)
block_1_expand_BN | input: | (None, 64, 64, 96)
BatchNormalization | output: | (None, 64, 64, 96)
block_1_expand_relu | input: | (None, 64, 64, 96)
ReLU | output: | (None, 64, 64, 96)
block_1_pad | input: | (None, 64, 64, 96)
ZeroPadding2D | output: | (None, 65, 65, 96)
block_1_depthwise | input: | (None, 65, 65, 96)

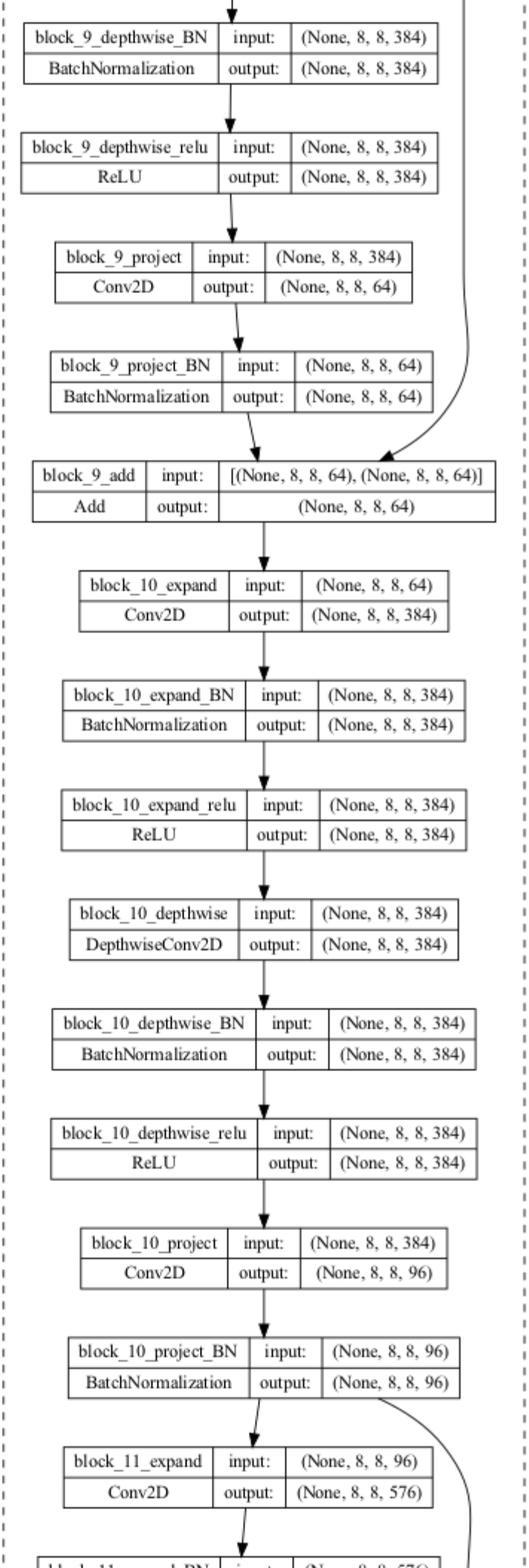

DepthwiseConv2D | output: | (None, 32, 32, 96)
block_1_depthwise_BN | input: | (None, 32, 32, 96)
BatchNormalization | output: | (None, 32, 32, 96)
block_1_depthwise_relu | input: | (None, 32, 32, 96)
ReLU | output: | (None, 32, 32, 96)
block_1_project | input: | (None, 32, 32, 96)
Conv2D | output: | (None, 32, 32, 24)
block_1_project_BN | input: | (None, 32, 32, 24)
BatchNormalization | output: | (None, 32, 32, 24)
block_2_expand | input: | (None, 32, 32, 24)
Conv2D | output: | (None, 32, 32, 144)
block_2_expand_BN | input: | (None, 32, 32, 144)
BatchNormalization | output: | (None, 32, 32, 144)
block_2_expand_relu | input: | (None, 32, 32, 144)
ReLU | output: | (None, 32, 32, 144)
block_2_depthwise | input: | (None, 32, 32, 144)
DepthwiseConv2D | output: | (None, 32, 32, 144)
block_2_depthwise_BN | input: | (None, 32, 32, 144)
BatchNormalization | output: | (None, 32, 32, 144)
block_2_depthwise_relu | input: | (None, 32, 32, 144)
ReLU | output: | (None, 32, 32, 144)
block_2_project | input: | (None, 32, 32, 144)
Conv2D | output: | (None, 32, 32, 24)
block_2_project_BN | input: | (None, 32, 32, 24)
BatchNormalization | output: | (None, 32, 32, 24)
block_2_add | input: | [(None, 32, 32, 24), (None, 32, 32, 24)]
Add | output: | (None, 32, 32, 24)
block_3_expand | input: | (None, 32, 32, 24)
Conv2D | output: | (None, 32, 32, 144)

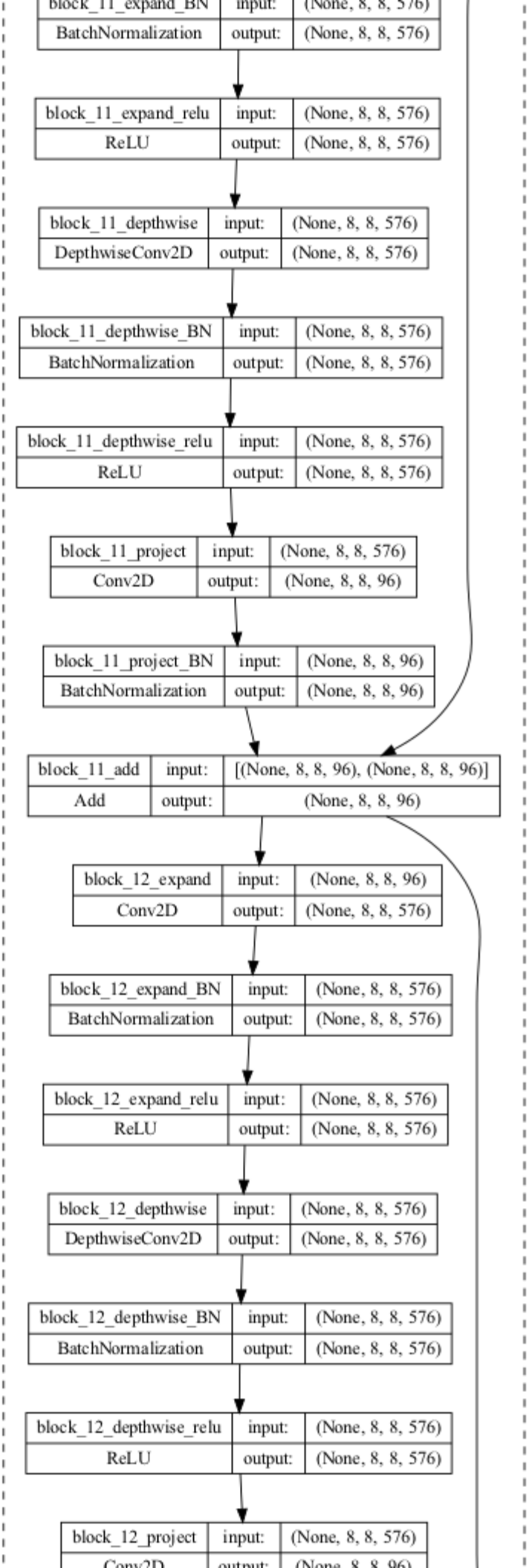

block_3_expand_BN | input: | (None, 32, 32, 144)
BatchNormalization | output: | (None, 32, 32, 144)
block_3_expand_relu | input: | (None, 32, 32, 144)
ReLU | output: | (None, 32, 32, 144)
block_3_pad | input: | (None, 32, 32, 144)
ZeroPadding2D | output: | (None, 33, 33, 144)
block_3_depthwise | input: | (None, 33, 33, 144)
DepthwiseConv2D | output: | (None, 16, 16, 144)
block_3_depthwise_BN | input: | (None, 16, 16, 144)
BatchNormalization | output: | (None, 16, 16, 144)
block_3_depthwise_relu | input: | (None, 16, 16, 144)
ReLU | output: | (None, 16, 16, 144)
block_3_project | input: | (None, 16, 16, 144)
Conv2D | output: | (None, 16, 16, 32)
block_3_project_BN | input: | (None, 16, 16, 32)
BatchNormalization | output: | (None, 16, 16, 32)
block_4_expand | input: | (None, 16, 16, 32)
Conv2D | output: | (None, 16, 16, 192)
block_4_expand_BN | input: | (None, 16, 16, 192)
BatchNormalization | output: | (None, 16, 16, 192)
block_4_expand_relu | input: | (None, 16, 16, 192)
ReLU | output: | (None, 16, 16, 192)
block_4_depthwise | input: | (None, 16, 16, 192)
DepthwiseConv2D | output: | (None, 16, 16, 192)
block_4_depthwise_BN | input: | (None, 16, 16, 192)
BatchNormalization | output: | (None, 16, 16, 192)
block_4_depthwise_relu | input: | (None, 16, 16, 192)
ReLU | output: | (None, 16, 16, 192)

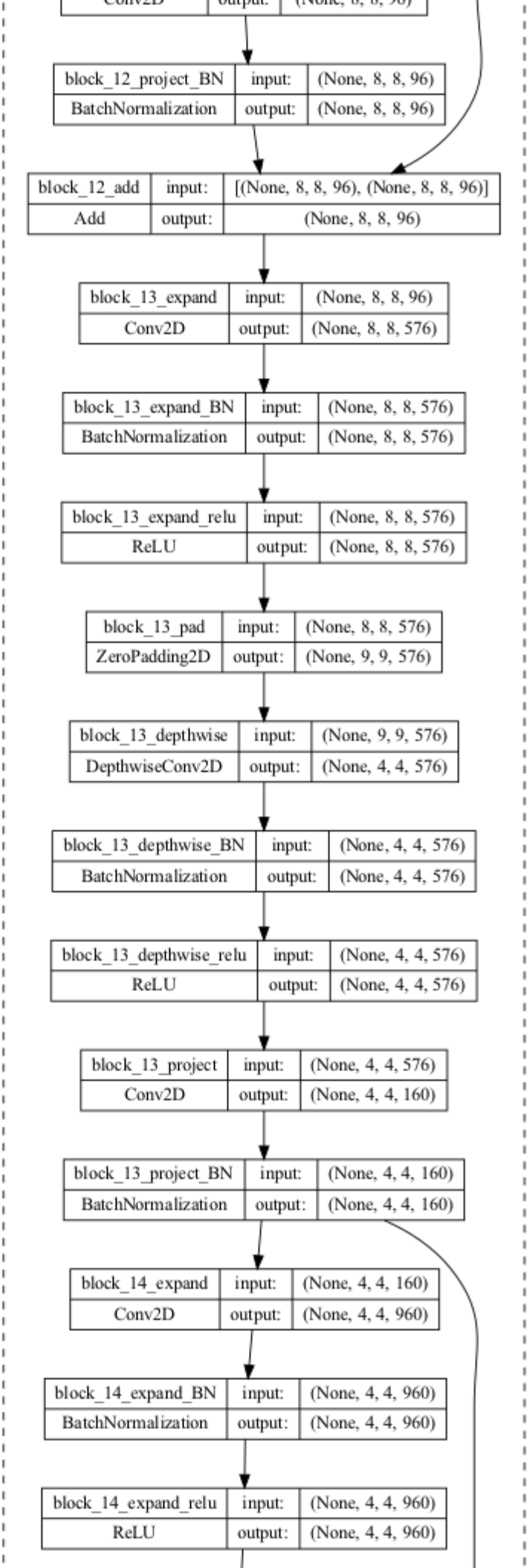

block_4_project | input: | (None, 16, 16, 192)
Conv2D | output: | (None, 16, 16, 32)
block_4_project_BN | input: | (None, 16, 16, 32)
BatchNormalization | output: | (None, 16, 16, 32)
block_4_add | input: | [(None, 16, 16, 32), (None, 16, 16, 32)]
Add | output: | (None, 16, 16, 32)
block_5_expand | input: | (None, 16, 16, 32)
Conv2D | output: | (None, 16, 16, 192)
block_5_expand_BN | input: | (None, 16, 16, 192)
BatchNormalization | output: | (None, 16, 16, 192)
block_5_expand_relu | input: | (None, 16, 16, 192)
ReLU | output: | (None, 16, 16, 192)
block_5_depthwise | input: | (None, 16, 16, 192)
DepthwiseConv2D | output: | (None, 16, 16, 192)
block_5_depthwise_BN | input: | (None, 16, 16, 192)
BatchNormalization | output: | (None, 16, 16, 192)
block_5_depthwise_relu | input: | (None, 16, 16, 192)
ReLU | output: | (None, 16, 16, 192)
block_5_project | input: | (None, 16, 16, 192)
Conv2D | output: | (None, 16, 16, 32)
block_5_project_BN | input: | (None, 16, 16, 32)
BatchNormalization | output: | (None, 16, 16, 32)
block_5_add | input: | [(None, 16, 16, 32), (None, 16, 16, 32)]
Add | output: | (None, 16, 16, 32)
block_6_expand | input: | (None, 16, 16, 32)
Conv2D | output: | (None, 16, 16, 192)
block_6_expand_BN | input: | (None, 16, 16, 192)
BatchNormalization | output: | (None, 16, 16, 192)

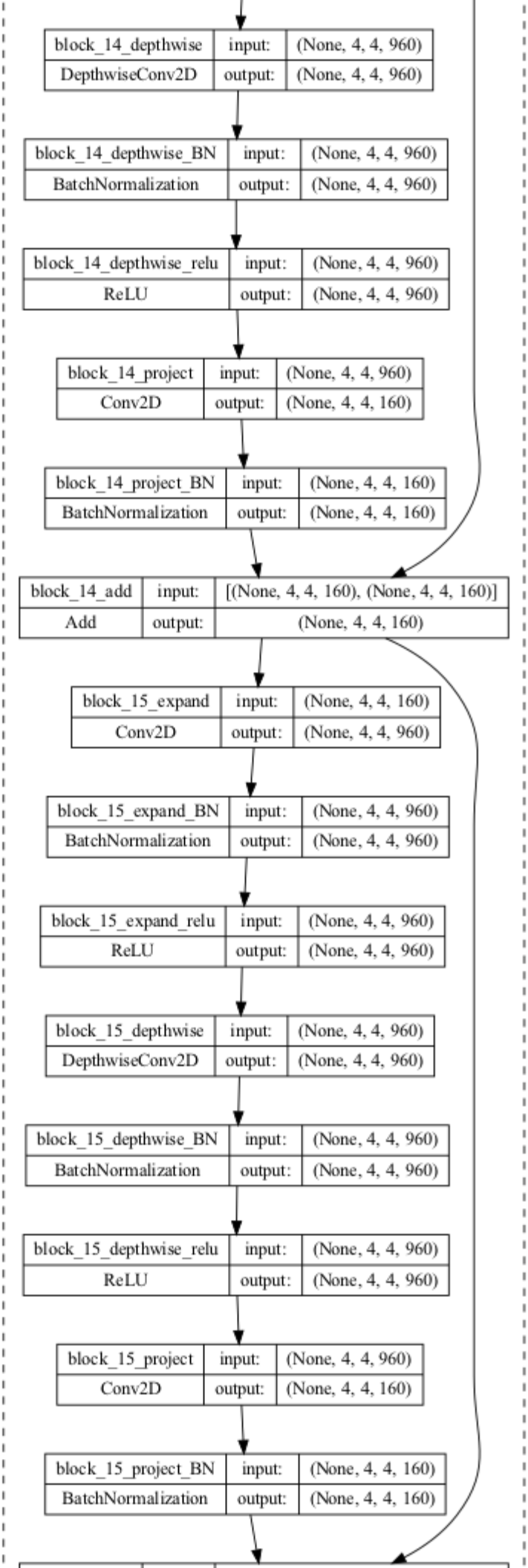

ReLU output: (None, 16, 16, 192)
block_6_pad input: (None, 16, 16, 192)
ZeroPadding2D output: (None, 17, 17, 192)
block_6_depthwise input: (None, 17, 17, 192)
DepthwiseConv2D output: (None, 8, 8, 192)
block_6_depthwise_BN input: (None, 8, 8, 192)
BatchNormalization output: (None, 8, 8, 192)
block_6_depthwise_relu input: (None, 8, 8, 192)
ReLU output: (None, 8, 8, 192)
block_6_project input: (None, 8, 8, 192)
Conv2D output: (None, 8, 8, 64)
block_6_project_BN input: (None, 8, 8, 64)
BatchNormalization output: (None, 8, 8, 64)
block_7_expand input: (None, 8, 8, 64)
Conv2D output: (None, 8, 8, 384)
block_7_expand_BN input: (None, 8, 8, 384)
BatchNormalization output: (None, 8, 8, 384)
block_7_expand_relu input: (None, 8, 8, 384)
ReLU output: (None, 8, 8, 384)
block_7_depthwise input: (None, 8, 8, 384)
DepthwiseConv2D output: (None, 8, 8, 384)
block_7_depthwise_BN input: (None, 8, 8, 384)
BatchNormalization output: (None, 8, 8, 384)
block_7_depthwise_relu input: (None, 8, 8, 384)
ReLU output: (None, 8, 8, 384)
block_7_project input: (None, 8, 8, 384)
Conv2D output: (None, 8, 8, 64)
block_7_project_BN input: (None, 8, 8, 64)
BatchNormalization output: (None, 8, 8, 64)

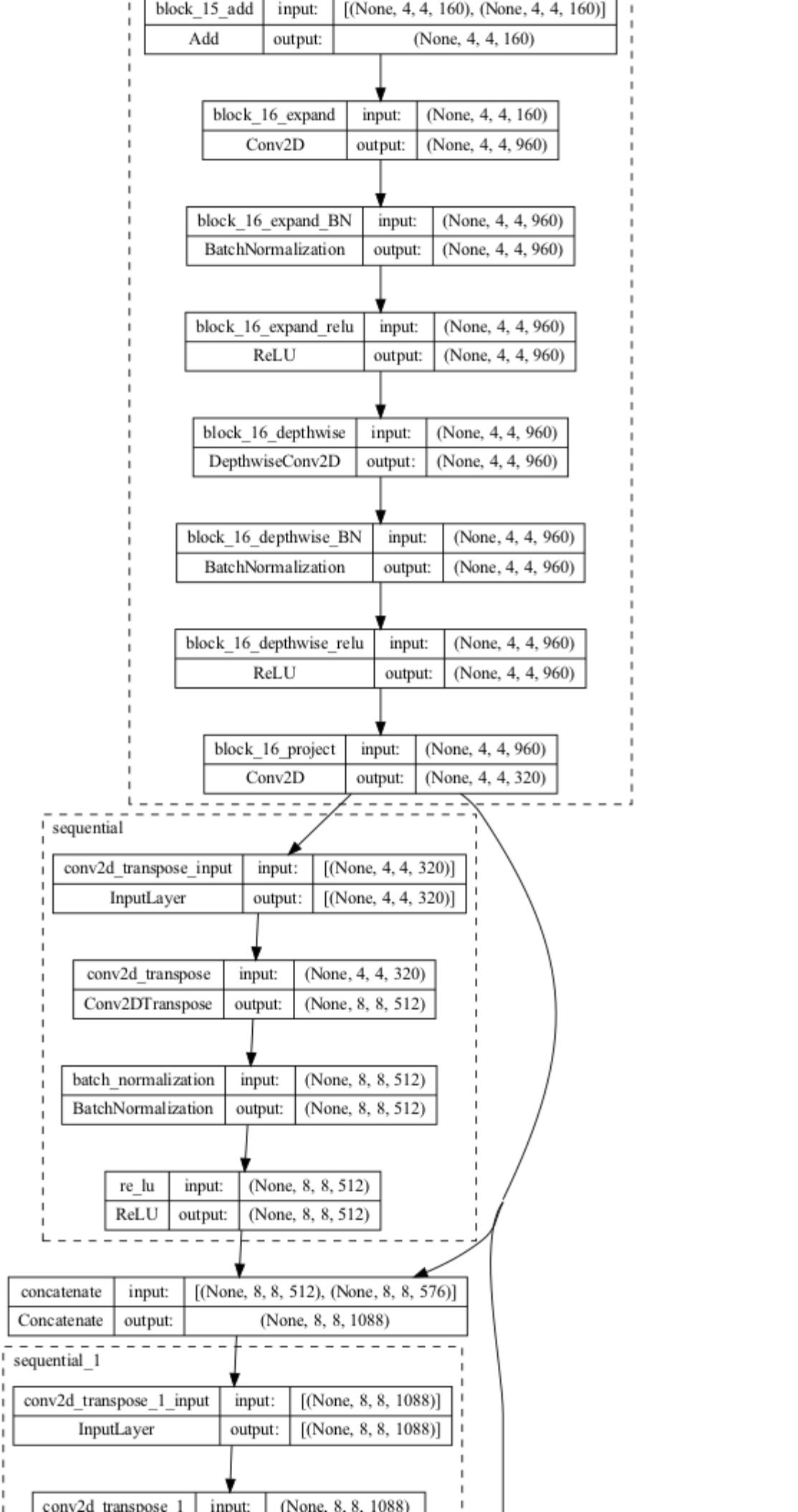

block_7_add | input: | [(None, 8, 8, 64), (None, 8, 8, 64)]
Add | output: | (None, 8, 8, 64)
block_8_expand | input: | (None, 8, 8, 64)
Conv2D | output: | (None, 8, 8, 384)
block_8_expand_BN | input: | (None, 8, 8, 384)
BatchNormalization | output: | (None, 8, 8, 384)
block_8_expand_relu | input: | (None, 8, 8, 384)
ReLU | output: | (None, 8, 8, 384)
block_8_depthwise | input: | (None, 8, 8, 384)
DepthwiseConv2D | output: | (None, 8, 8, 384)
block_8_depthwise_BN | input: | (None, 8, 8, 384)
BatchNormalization | output: | (None, 8, 8, 384)
block_8_depthwise_relu | input: | (None, 8, 8, 384)
ReLU | output: | (None, 8, 8, 384)
block_8_project | input: | (None, 8, 8, 384)
Conv2D | output: | (None, 8, 8, 64)
block_8_project_BN | input: | (None, 8, 8, 64)
BatchNormalization | output: | (None, 8, 8, 64)
block_8_add | input: | [(None, 8, 8, 64), (None, 8, 8, 64)]
Add | output: | (None, 8, 8, 64)
block_9_expand | input: | (None, 8, 8, 64)
Conv2D | output: | (None, 8, 8, 384)
block_9_expand_BN | input: | (None, 8, 8, 384)
BatchNormalization | output: | (None, 8, 8, 384)
block_9_expand_relu | input: | (None, 8, 8, 384)
ReLU | output: | (None, 8, 8, 384)
block_9_depthwise | input: | (None, 8, 8, 384)
DepthwiseConv2D | output: | (None, 8, 8, 384)

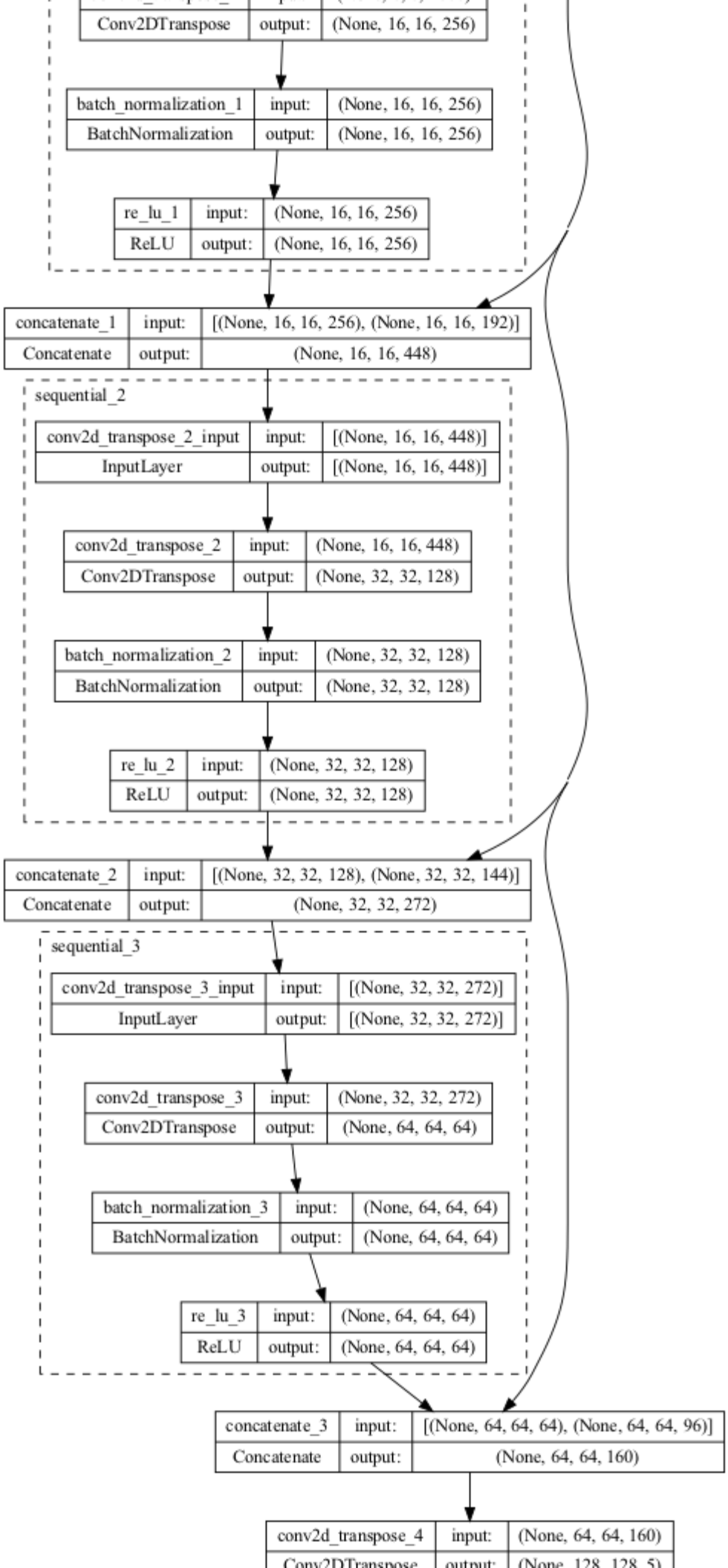

block_9_depthwise_BN | input: | (None, 8, 8, 384)
BatchNormalization | output: | (None, 8, 8, 384)
block_9_depthwise_relu | input: | (None, 8, 8, 384)
ReLU | output: | (None, 8, 8, 384)
block_9_project | input: | (None, 8, 8, 384)
Conv2D | output: | (None, 8, 8, 64)
block_9_project_BN | input: | (None, 8, 8, 64)
BatchNormalization | output: | (None, 8, 8, 64)
block_9_add | input: | [(None, 8, 8, 64), (None, 8, 8, 64)]
Add | output: | (None, 8, 8, 64)
block_10_expand | input: | (None, 8, 8, 64)
Conv2D | output: | (None, 8, 8, 384)
block_10_expand_BN | input: | (None, 8, 8, 384)
BatchNormalization | output: | (None, 8, 8, 384)
block_10_expand_relu | input: | (None, 8, 8, 384)
ReLU | output: | (None, 8, 8, 384)
block_10_depthwise | input: | (None, 8, 8, 384)
DepthwiseConv2D | output: | (None, 8, 8, 384)
block_10_depthwise_BN | input: | (None, 8, 8, 384)
BatchNormalization | output: | (None, 8, 8, 384)
block_10_depthwise_relu | input: | (None, 8, 8, 384)
ReLU | output: | (None, 8, 8, 384)
block_10_project | input: | (None, 8, 8, 384)
Conv2D | output: | (None, 8, 8, 96)
block_10_project_BN | input: | (None, 8, 8, 96)
BatchNormalization | output: | (None, 8, 8, 96)
block_11_expand | input: | (None, 8, 8, 96)
Conv2D | output: | (None, 8, 8, 576)


block_11_expand_BN | input: | (None, 8, 8, 576)
BatchNormalization | output: | (None, 8, 8, 576)
block_11_expand_relu | input: | (None, 8, 8, 576)
ReLU | output: | (None, 8, 8, 576)
block_11_depthwise | input: | (None, 8, 8, 576)
DepthwiseConv2D | output: | (None, 8, 8, 576)
block_11_depthwise_BN | input: | (None, 8, 8, 576)
BatchNormalization | output: | (None, 8, 8, 576)
block_11_depthwise_relu | input: | (None, 8, 8, 576)
ReLU | output: | (None, 8, 8, 576)
block_11_project | input: | (None, 8, 8, 576)
Conv2D | output: | (None, 8, 8, 96)
block_11_project_BN | input: | (None, 8, 8, 96)
BatchNormalization | output: | (None, 8, 8, 96)
block_11_add | input: | [(None, 8, 8, 96), (None, 8, 8, 96)]
Add | output: | (None, 8, 8, 96)
block_12_expand | input: | (None, 8, 8, 96)
Conv2D | output: | (None, 8, 8, 576)
block_12_expand_BN | input: | (None, 8, 8, 576)
BatchNormalization | output: | (None, 8, 8, 576)
block_12_expand_relu | input: | (None, 8, 8, 576)
ReLU | output: | (None, 8, 8, 576)
block_12_depthwise | input: | (None, 8, 8, 576)
DepthwiseConv2D | output: | (None, 8, 8, 576)
block_12_depthwise_BN | input: | (None, 8, 8, 576)
BatchNormalization | output: | (None, 8, 8, 576)
block_12_depthwise_relu | input: | (None, 8, 8, 576)
ReLU | output: | (None, 8, 8, 576)
block_12_project | input: | (None, 8, 8, 576)


block_12_project_BN | input: | (None, 8, 8, 96)
BatchNormalization | output: | (None, 8, 8, 96)
block_12_add | input: | [(None, 8, 8, 96), (None, 8, 8, 96)]
Add | output: | (None, 8, 8, 96)
block_13_expand | input: | (None, 8, 8, 96)
Conv2D | output: | (None, 8, 8, 576)
block_13_expand_BN | input: | (None, 8, 8, 576)
BatchNormalization | output: | (None, 8, 8, 576)
block_13_expand_relu | input: | (None, 8, 8, 576)
ReLU | output: | (None, 8, 8, 576)
block_13_pad | input: | (None, 8, 8, 576)
ZeroPadding2D | output: | (None, 9, 9, 576)
block_13_depthwise | input: | (None, 9, 9, 576)
DepthwiseConv2D | output: | (None, 4, 4, 576)
block_13_depthwise_BN | input: | (None, 4, 4, 576)
BatchNormalization | output: | (None, 4, 4, 576)
block_13_depthwise_relu | input: | (None, 4, 4, 576)
ReLU | output: | (None, 4, 4, 576)
block_13_project | input: | (None, 4, 4, 576)
Conv2D | output: | (None, 4, 4, 160)
block_13_project_BN | input: | (None, 4, 4, 160)
BatchNormalization | output: | (None, 4, 4, 160)
block_14_expand | input: | (None, 4, 4, 160)
Conv2D | output: | (None, 4, 4, 960)
block_14_expand_BN | input: | (None, 4, 4, 960)
BatchNormalization | output: | (None, 4, 4, 960)
block_14_expand_relu | input: | (None, 4, 4, 960)
ReLU | output: | (None, 4, 4, 960)


block_14_depthwise | input: | (None, 4, 4, 960)
DepthwiseConv2D | output: | (None, 4, 4, 960)
block_14_depthwise_BN | input: | (None, 4, 4, 960)
BatchNormalization | output: | (None, 4, 4, 960)
block_14_depthwise_relu | input: | (None, 4, 4, 960)
ReLU | output: | (None, 4, 4, 960)
block_14_project | input: | (None, 4, 4, 960)
Conv2D | output: | (None, 4, 4, 160)
block_14_project_BN | input: | (None, 4, 4, 160)
BatchNormalization | output: | (None, 4, 4, 160)
block_14_add | input: | [(None, 4, 4, 160), (None, 4, 4, 160)]
Add | output: | (None, 4, 4, 160)
block_15_expand | input: | (None, 4, 4, 160)
Conv2D | output: | (None, 4, 4, 960)
block_15_expand_BN | input: | (None, 4, 4, 960)
BatchNormalization | output: | (None, 4, 4, 960)
block_15_expand_relu | input: | (None, 4, 4, 960)
ReLU | output: | (None, 4, 4, 960)
block_15_depthwise | input: | (None, 4, 4, 960)
DepthwiseConv2D | output: | (None, 4, 4, 960)
block_15_depthwise_BN | input: | (None, 4, 4, 960)
BatchNormalization | output: | (None, 4, 4, 960)
block_15_depthwise_relu | input: | (None, 4, 4, 960)
ReLU | output: | (None, 4, 4, 960)
block_15_project | input: | (None, 4, 4, 960)
Conv2D | output: | (None, 4, 4, 160)
block_15_project_BN | input: | (None, 4, 4, 160)
BatchNormalization | output: | (None, 4, 4, 160)


block_15_add | input: | [(None, 4, 4, 160), (None, 4, 4, 160)]
Add | output: | (None, 4, 4, 160)
block_16_expand | input: | (None, 4, 4, 160)
Conv2D | output: | (None, 4, 4, 960)
block_16_expand_BN | input: | (None, 4, 4, 960)
BatchNormalization | output: | (None, 4, 4, 960)
block_16_expand_relu | input: | (None, 4, 4, 960)
ReLU | output: | (None, 4, 4, 960)
block_16_depthwise | input: | (None, 4, 4, 960)
DepthwiseConv2D | output: | (None, 4, 4, 960)
block_16_depthwise_BN | input: | (None, 4, 4, 960)
BatchNormalization | output: | (None, 4, 4, 960)
block_16_depthwise_relu | input: | (None, 4, 4, 960)
ReLU | output: | (None, 4, 4, 960)
block_16_project | input: | (None, 4, 4, 960)
Conv2D | output: | (None, 4, 4, 320)
sequential
conv2d_transpose_input | input: | [(None, 4, 4, 320)]
InputLayer | output: | [(None, 4, 4, 320)]
conv2d_transpose | input: | (None, 4, 4, 320)
Conv2DTranspose | output: | (None, 8, 8, 512)
batch_normalization | input: | (None, 8, 8, 512)
BatchNormalization | output: | (None, 8, 8, 512)
re_lu | input: | (None, 8, 8, 512)
ReLU | output: | (None, 8, 8, 512)
concatenate | input: | [(None, 8, 8, 512), (None, 8, 8, 576)]
Concatenate | output: | (None, 8, 8, 1088)
sequential_1
conv2d_transpose_1_input | input: | [(None, 8, 8, 1088)]
InputLayer | output: | [(None, 8, 8, 1088)]
conv2d_transpose_1 | input: | (None, 8, 8, 1088)


Conv2DTranspose output: (None, 16, 16, 256)
batch_normalization_1 input: (None, 16, 16, 256)
BatchNormalization output: (None, 16, 16, 256)
re_lu_1 input: (None, 16, 16, 256)
ReLU output: (None, 16, 16, 256)
concatenate_1 input: [(None, 16, 16, 256), (None, 16, 16, 192)]
Concatenate output: (None, 16, 16, 448)
sequential_2
conv2d_transpose_2_input input: [(None, 16, 16, 448)]
InputLayer output: [(None, 16, 16, 448)]
conv2d_transpose_2 input: (None, 16, 16, 448)
Conv2DTranspose output: (None, 32, 32, 128)
batch_normalization_2 input: (None, 32, 32, 128)
BatchNormalization output: (None, 32, 32, 128)
re_lu_2 input: (None, 32, 32, 128)
ReLU output: (None, 32, 32, 128)
concatenate_2 input: [(None, 32, 32, 128), (None, 32, 32, 144)]
Concatenate output: (None, 32, 32, 272)
sequential_3
conv2d_transpose_3_input input: [(None, 32, 32, 272)]
InputLayer output: [(None, 32, 32, 272)]
conv2d_transpose_3 input: (None, 32, 32, 272)
Conv2DTranspose output: (None, 64, 64, 64)
batch_normalization_3 input: (None, 64, 64, 64)
BatchNormalization output: (None, 64, 64, 64)
re_lu_3 input: (None, 64, 64, 64)
ReLU output: (None, 64, 64, 64)
concatenate_3 input: [(None, 64, 64, 64), (None, 64, 64, 96)]
Concatenate output: (None, 64, 64, 160)
conv2d_transpose_4 input: (None, 64, 64, 160)

Conv2D Transpose	Output:	(None, 128, 128, 3)

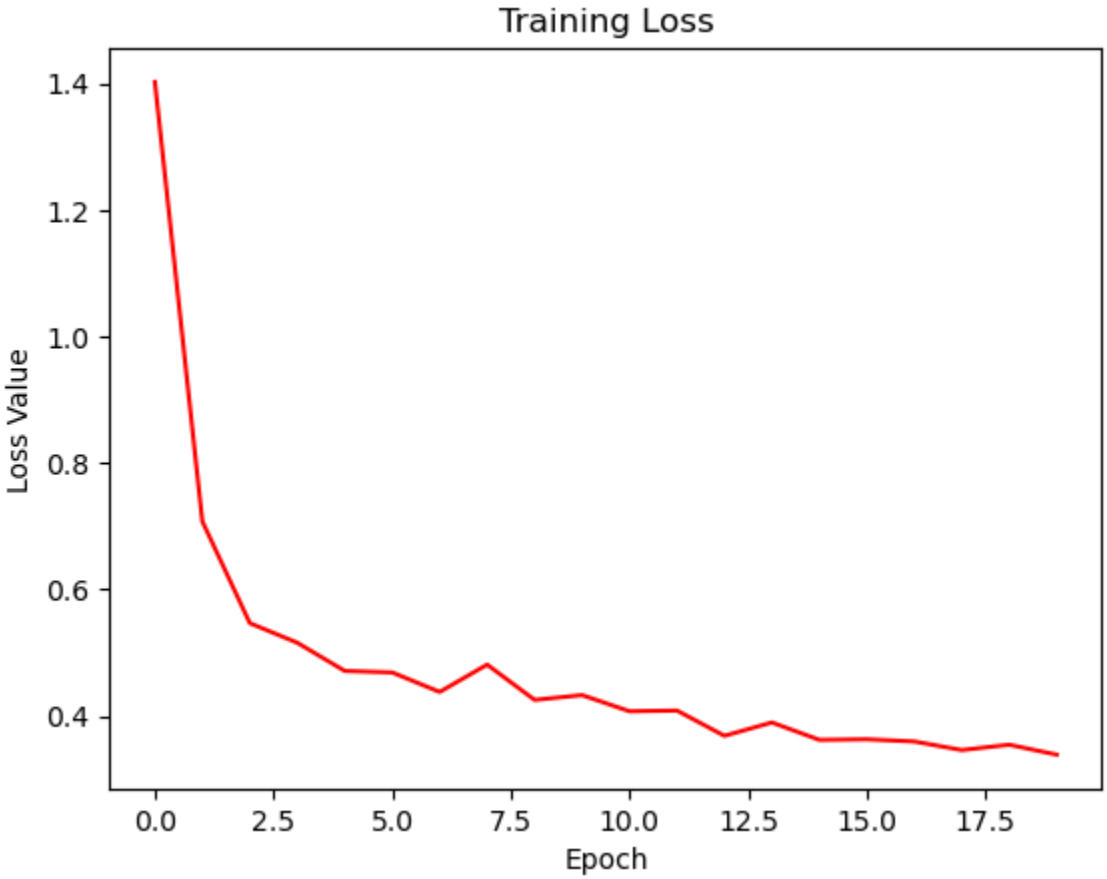
In [14]:
```
# Train the model.
model_history = model.fit(
    train_ds,
    validation_data=test_ds,
    epochs=EPOCHS,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps
)
```

```
Epoch 1/20
2025-03-31 05:48:23.931704: W tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Fai
led to get CPU frequency: 0 Hz
4/4 [==============================] - 4s 688ms/step - loss: 1.4026 - accuracy: 0.3716
Epoch 2/20
4/4 [==============================] - 3s 797ms/step - loss: 0.7076 - accuracy: 0.8350
Epoch 3/20
4/4 [==============================] - 3s 773ms/step - loss: 0.5467 - accuracy: 0.8859
Epoch 4/20
4/4 [==============================] - 3s 720ms/step - loss: 0.5157 - accuracy: 0.8843
Epoch 5/20
4/4 [==============================] - 2s 603ms/step - loss: 0.4711 - accuracy: 0.8896
Epoch 6/20
4/4 [==============================] - 3s 760ms/step - loss: 0.4687 - accuracy: 0.8865
Epoch 7/20
4/4 [==============================] - 3s 725ms/step - loss: 0.4378 - accuracy: 0.8926
Epoch 8/20
4/4 [==============================] - 3s 683ms/step - loss: 0.4811 - accuracy: 0.8772
Epoch 9/20
4/4 [==============================] - 3s 651ms/step - loss: 0.4253 - accuracy: 0.8921
Epoch 10/20
4/4 [==============================] - 3s 615ms/step - loss: 0.4329 - accuracy: 0.8864
Epoch 11/20
4/4 [==============================] - 3s 758ms/step - loss: 0.4073 - accuracy: 0.8887
Epoch 12/20
4/4 [==============================] - 2s 656ms/step - loss: 0.4084 - accuracy: 0.8824
Epoch 13/20
4/4 [==============================] - 3s 761ms/step - loss: 0.3684 - accuracy: 0.8930
Epoch 14/20
4/4 [==============================] - 3s 650ms/step - loss: 0.3896 - accuracy: 0.8815
Epoch 15/20
4/4 [==============================] - 3s 626ms/step - loss: 0.3621 - accuracy: 0.8890
Epoch 16/20
4/4 [==============================] - 3s 749ms/step - loss: 0.3630 - accuracy: 0.8863
Epoch 17/20
4/4 [==============================] - 3s 799ms/step - loss: 0.3595 - accuracy: 0.8866
Epoch 18/20
4/4 [==============================] - 3s 698ms/step - loss: 0.3459 - accuracy: 0.8890
Epoch 19/20
4/4 [==============================] - 3s 663ms/step - loss: 0.3544 - accuracy: 0.8876
Epoch 20/20
4/4 [==============================] - 3s 659ms/step - loss: 0.3387 - accuracy: 0.8905
```

4.0 Evaluate Model

4.1 Plot model loss.

In [15]:
```
# Code taken from reference 27.
# Get the loss history.
loss = model_history.history['loss']

# Plot the loss history.
plt.figure()
plt.plot(model_history.epoch, loss, 'r', label='Training loss')
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss Value')
plt.show()
```

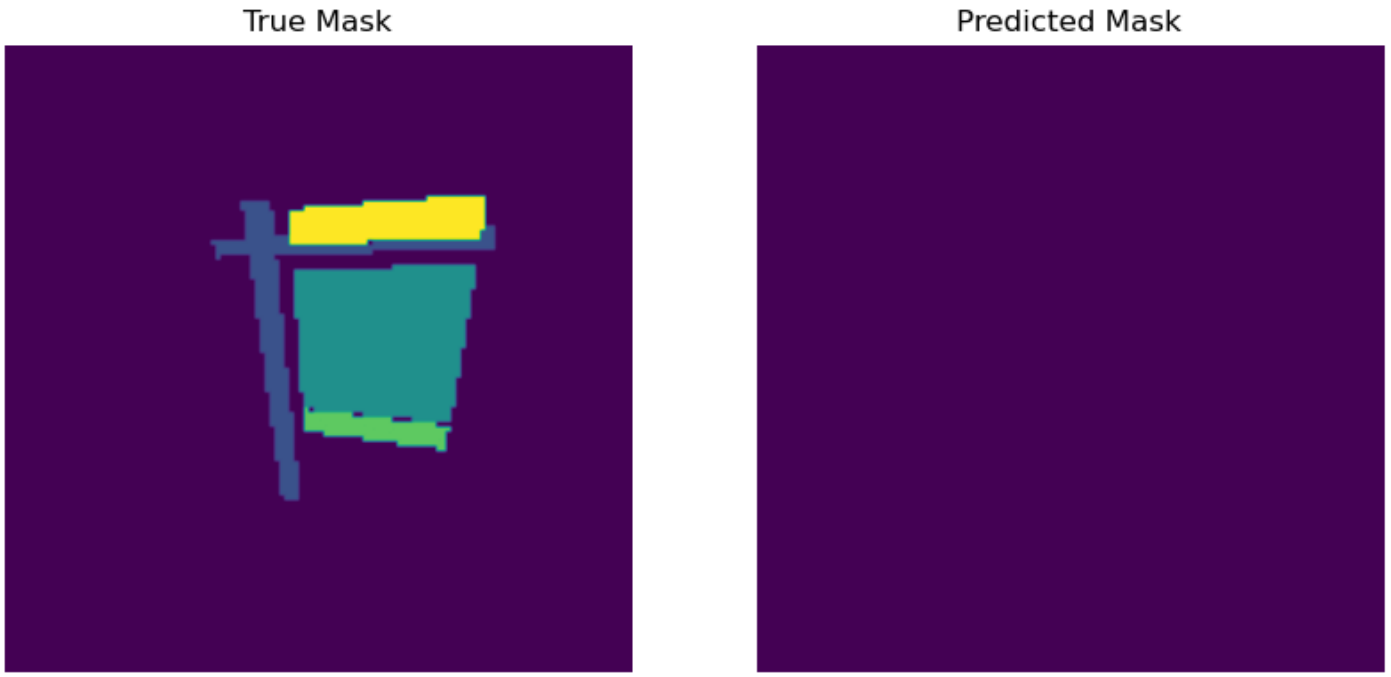


4.2 Plot metrics per class

In [23]:
```
# Guidance for this code provided by reference 29.
# Flatten masks and predictions
y_true_all = []
y_predictions_all = []

# Loop through the test data and predict the masks with the model.
for images, masks in test_ds.take(1):
    predictions = model.predict(images)
    predictions = tf.argmax(predictions, axis=-1).numpy()
    masks = masks.numpy()
    y_true_all.append(masks)
    y_predictions_all.append(predictions)

    # Plot a predicted mask vs an accurate mask

    # Create plot.
    fig, axes = plt.subplots(1, 2, figsize = (10, 5))
```

```
    # Show base image
    axes[0].imshow(masks[4])
    axes[0].set_title("True Mask")
    axes[0].axis('off')

    # Show image with mask.
    axes[1].imshow(predictions[4])
    axes[1].set_title("Predicted Mask")
    axes[1].axis('off')

    plt.show()

# Flatten all masks and predictions
y_true_all = np.concatenate(y_true_all).flatten()
y_predictions_all = np.concatenate(y_predictions_all).flatten()

# Per-Class Accuracy + Precision/Recall
print("Per-Class Metrics:")
print(classification_report(y_true_all, y_predictions_all))
```

```
1/1 [==============================] - 0s 251ms/step
```



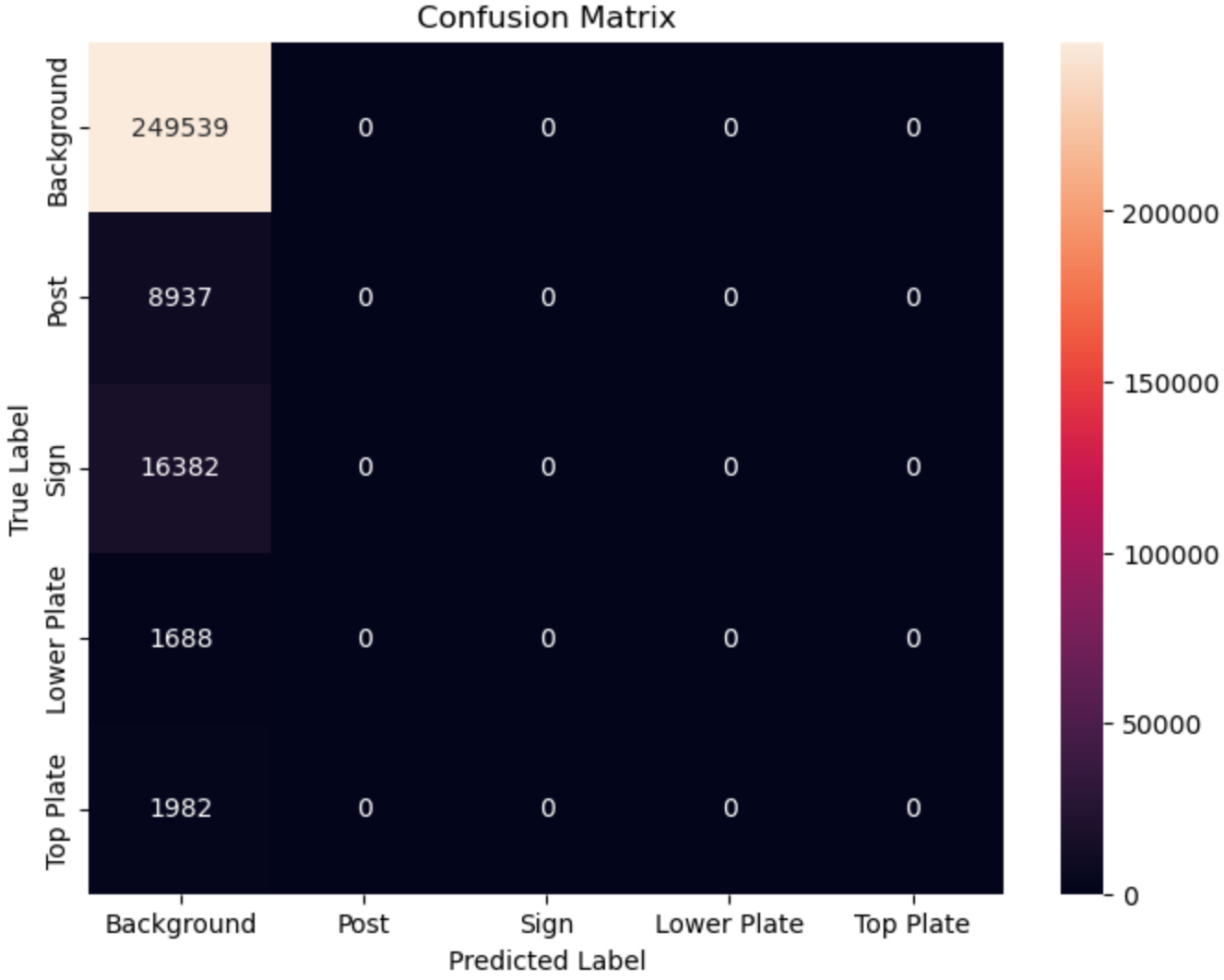

```
Per-Class Metrics:
              precision    recall  f1-score   support

           0       0.90      1.00      0.95    249539
           1       0.00      0.00      0.00      8937
           2       0.00      0.00      0.00     16382
           3       0.00      0.00      0.00      1688
           4       0.00      0.00      0.00      1982

    accuracy                           0.90    278528
   macro avg       0.18      0.20      0.19    278528
weighted avg       0.80      0.90      0.85    278528
```

```
/opt/anaconda3/envs/london/lib/python3.11/site-packages/sklearn/metrics/_classification.p
y:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels w
ith no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/anaconda3/envs/london/lib/python3.11/site-packages/sklearn/metrics/_classification.p
y:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels w
ith no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/anaconda3/envs/london/lib/python3.11/site-packages/sklearn/metrics/_classification.p
y:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels w
ith no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

4.3 Plot the confusion matrix

In [21]:

```
# Code from reference 28.
# Compute confusion matrix
cm = confusion_matrix(y_true_all, y_predictions_all)

# Define class labels.
class_names = ['Background', 'Post', 'Sign', 'Lower Plate', 'Top Plate']

# Plot the confusion matrix.
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```


Confusion Matrix
Background
Post
Sign
Lower Plate
Top Plate
True Label
249539
0
0
0
0
8937
0
0
0
0
16382
0
0
0
0
1688
0
0
0
0
1982
0
0
0
0
Background
Post
Sign
Lower Plate
Top Plate
Predicted Label
200000
150000
100000
50000
0