Capstone 3 Final Report:
Semantic Segmentation of Aerial Cityscapes

1.0 Problem Statement

Self-driving cars and flying taxis are no longer just a thing of science fiction. In fact, companies like Joby Aviation and Archer Aviation are already making strides towards certification and hope to have commercial services up and running by 2025 - that's three years ahead of the FAA's goal! But, safety is the top priority and the FAA won't compromise on that. That's where autonomous flight comes in. By using semantic segmentation of live video, these vehicles can understand their surroundings and potential obstacles, making them even safer. And it's not just people who will benefit from this technology. Retailers, restaurants, and delivery companies are already testing drone deliveries to speed up delivery times and reach rural communities. But, for these drones to work in urban settings, they need to be able to navigate through ever-changing environments. That's where semantic segmentation comes in again, aiding in autonomous package delivery services.

As we move towards a future of autonomous vehicles, it's crucial that we expand the abilities of computer vision, especially in the urban landscape. This study aims to train a deep neural network using aerial cityscape and urban images captured from drone views. The objective is to develop a model that can perform semantic segmentation of imagery taken from an aerial vehicle in an urban setting. The ultimate goal is to enhance the safety and efficiency of the transportation system, whether it be for humans or packages.

2.0 The Datasets

To accomplish this goal, I've chosen to work with two datasets: the *Varied Drone Dataset for Semantic Segmentation* (VDD) and the *Semantic Drone Dataset* (Christian Mostegel). The Varied Drone Dataset for Semantic Segmentation is from here-on-out referred to as the VDD dataset and the Semantic Drone Dataset as the SDD dataset.

The VDD dataset contains 400 pixel-level annotated images featuring verified scenes, camera angles (30, 60, and 90 (straight-down) degrees), and weather/light conditions of UAV images. The types of scenes available in this dataset are municipal residential zones, villas, school and college buildings, hospitals, highways and roads, facilities like gyms and libraries, rivers, lakes and mountains, villages and farm fields.

The SDD dataset contains 400 pixel-level annotated images of urban scenes from a 90 degree (straight-down) angle. This dataset contains images closer to the ground than the VDD dataset and captures images of buildings, people, cars, roads, and vegetation, further defined into more specific categories.

3.0 Data Wrangling

The VDD dataset requires manual download from their website and has images separated into training, validation, and testing folders – each with their respective source and ground truth images (raw image and segmented masks). It is important to note that the VDD

ground truth images are masked with integer values and are not RGB images, but rather single-channel images (grayscale). However, the source images are RGB. Viewing thumbnails through something like Windows' File Explorer will likely produce what appear to be all-black images due to the grayscale values being closer to 0 than 255. The class ID information can be found on their website and is shown below, in Table 1: VDD Class IDs:

Table 1: VDD Class IDs

VDD Class ID	VDD Class
0	other
1	wall
2	road
3	vegetation
4	vehicle
5	roof
6	water

The SDD dataset requires manual download from their website as well. These images are separated into only source and ground truth folders, but have a class id dictionary supplied in the download as a .csv file. Both the source and ground truth images are in RGB for this dataset. Their class ID information is shown below, in Table 2: SDD Class IDs:

Table 2: SDD Class IDs

SDD Class ID	SDD Class
0	unlabeled
1	paved-area
2	dirt
3	grass
4	gravel
5	water
6	rocks
7	pool
8	vegetation
9	roof
10	wall
11	window
12	door
13	fence
14	fence-pole
15	person
16	dog
17	car
18	bicycle
19	tree
20	bald-tree
21	ar-marker
22	obstacle
23	conflicting

4.0 Exploratory Data Analysis

For each dataset I verified the following:

1. The number of channels for each image and mask.
2. The size of each image and mask.
3. The observed class labels for each mask.
4. The distribution of class labels for each dataset.

Table 3: VDD and SDD Image Information, below, quickly compares image info between the datasets:

Table 3: VDD and SDD Image Information

	VDD		SDD	
	Images	Masks	Images	Masks
# of Channels	3	1	3	3
Resolution (height by width)	4000x3000	4000x3000	6000x4000	6000x4000

All 7 classes were observed in the VDD dataset and their distribution can be seen in Figure 1: VDD Class Distribution, below:

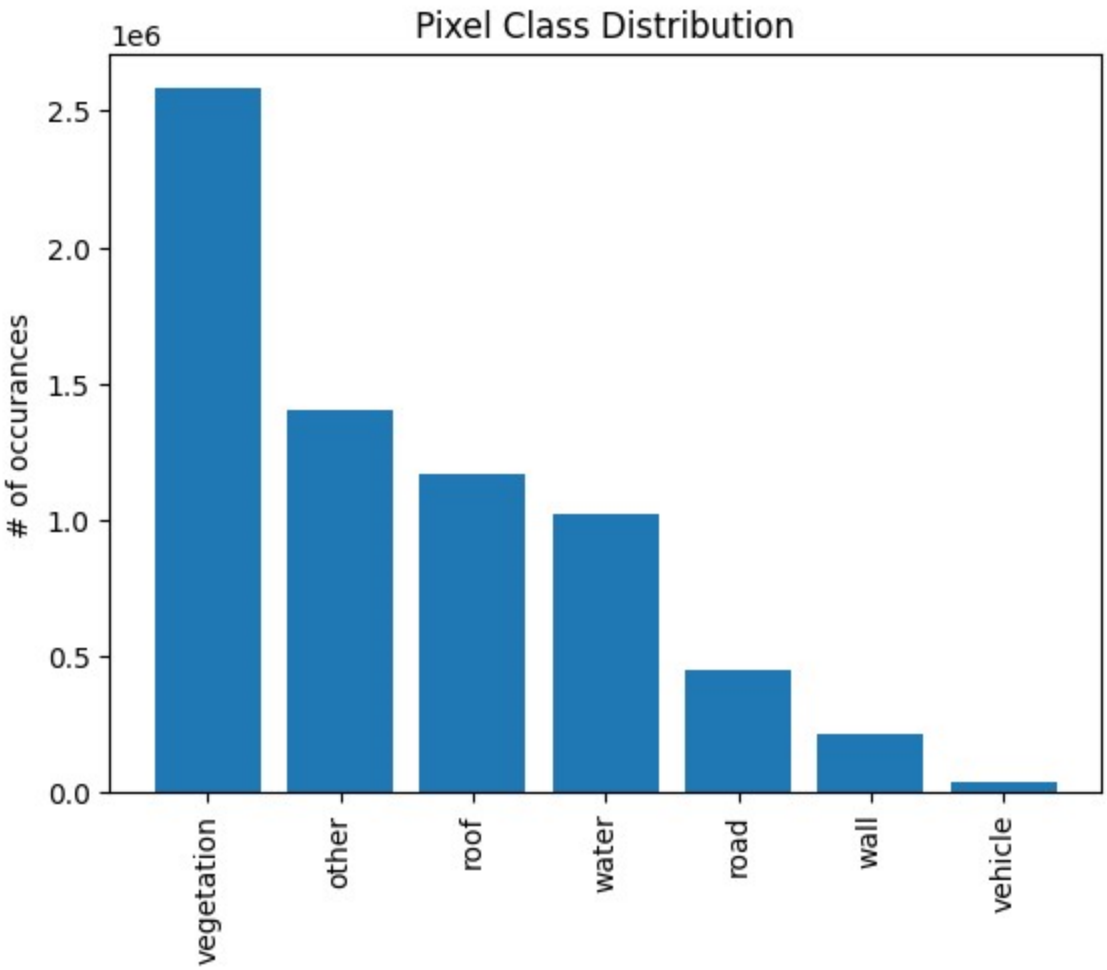

Figure 1: VDD Class Distribution

There is observed class imbalance, however this can be accounted for in training by weighting the less represented classes higher than those that are more represented.

All but one class were observed in the SDD dataset: the *conflicting* class. However, this class is not a class that we want represented, as it sounds like it does not represent any physical object, but rather a problem with the mask. The distribution of classes can be seen in Figure 2: SDD Class Distribution, below:

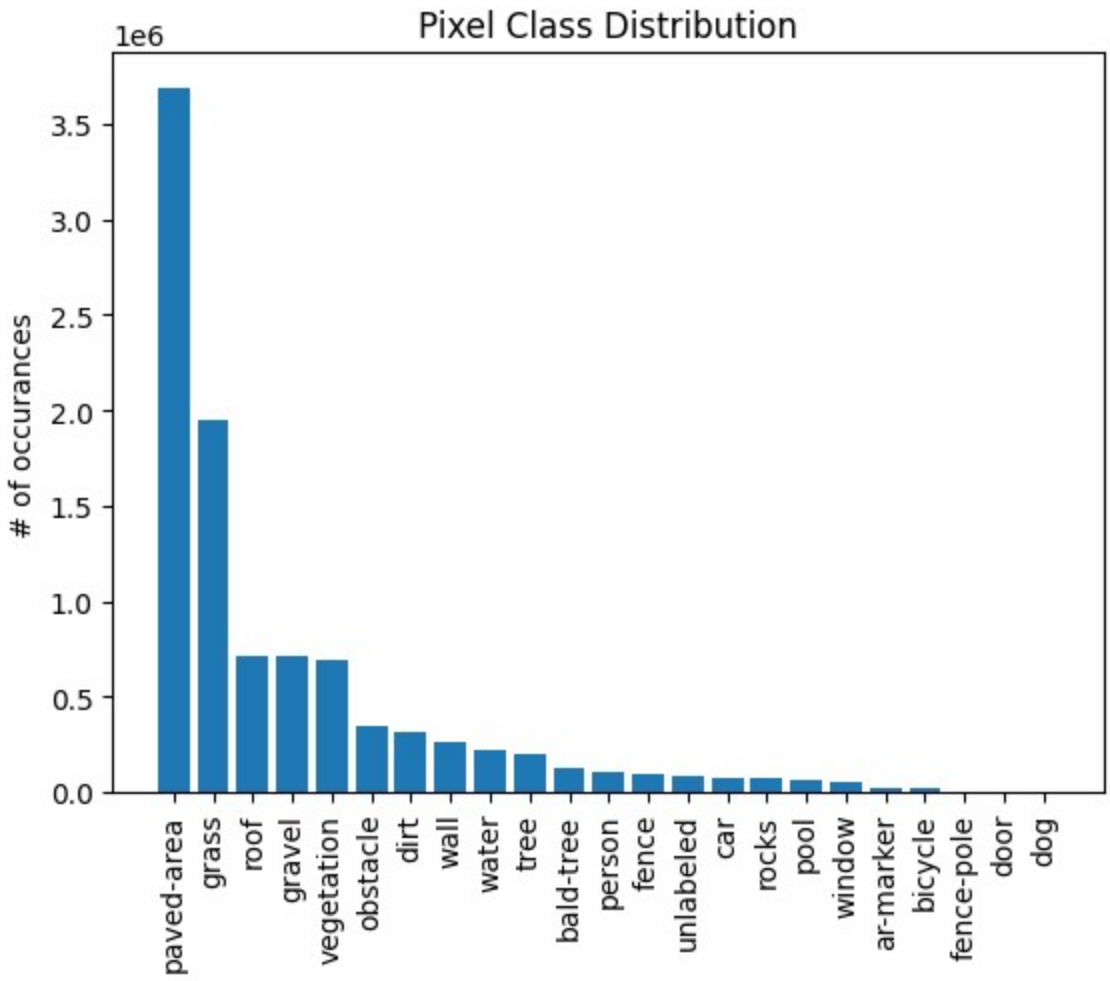

Figure 2: SDD Class Distribution

A very small portion of the dataset does appear to be *unlabeled*, but this will be left as is. Once again class imbalance is very notable in this dataset, given the nature of the image content. Class weights will be used in the modeling stage.

5.0 Preprocessing and Training Data Development

The preprocessing and training data development step includes remasking the ground truth images of both datasets into a single masking scheme and then saving the new masks and their corresponding images into a final training and testing folder.

In order to use both datasets for training, a common masking scheme needed to be defined. This common scheme is being referred to as the Master mask. The remasking scheme can be summarized in the following table, Table 3: Remasking Scheme:

Table 3: Remasking Scheme

Master Class ID (Class)	VDD Class ID (Class)	SDD Class ID (Class)
0 (unlabeled)	0 (others)	0 (unlabeled) 2 (dirt) 3 (grass) 4 (gravel) 6 (rocks) 7 (pool) 11 (window) 12 (door) 13 (fence) 14 (fence-pole) 16 (dog) 18 (bicycle) 21 (ar-marker) 22 (obstacle)
1 (wall)	1 (wall)	10 (wall)
2 (roof)	5 (roof)	9 (roof)
3 (road)	2 (road)	1 (paved-area)
4 (water)	6 (water)	5 (water)
5 (car)	4 (vehicle)	17 (car)
6 (vegetation)	3 (vegetation)	8 (vegetation) 19 (tree) 20 (bald-tree)
7 (person)		15 (person)

After each ground truth mask has been remapped into the Master mask, they are saved into separate training and testing folders. Since the VDD and SDD datasets capture aerial imagery from different angles and altitudes, an equal proportion of each dataset was dedicated to the training and testing splits. Those splits are 75% training and 25% testing. A random state seed of 42 was used when creating the train-test splits to allow for reproducibility.

After preprocessing, examples of the final input image and mask are shown below in Figure 3: VDD Image and Mask Pair Sample and Figure 4: SDD Image and Mask Pair Sample:

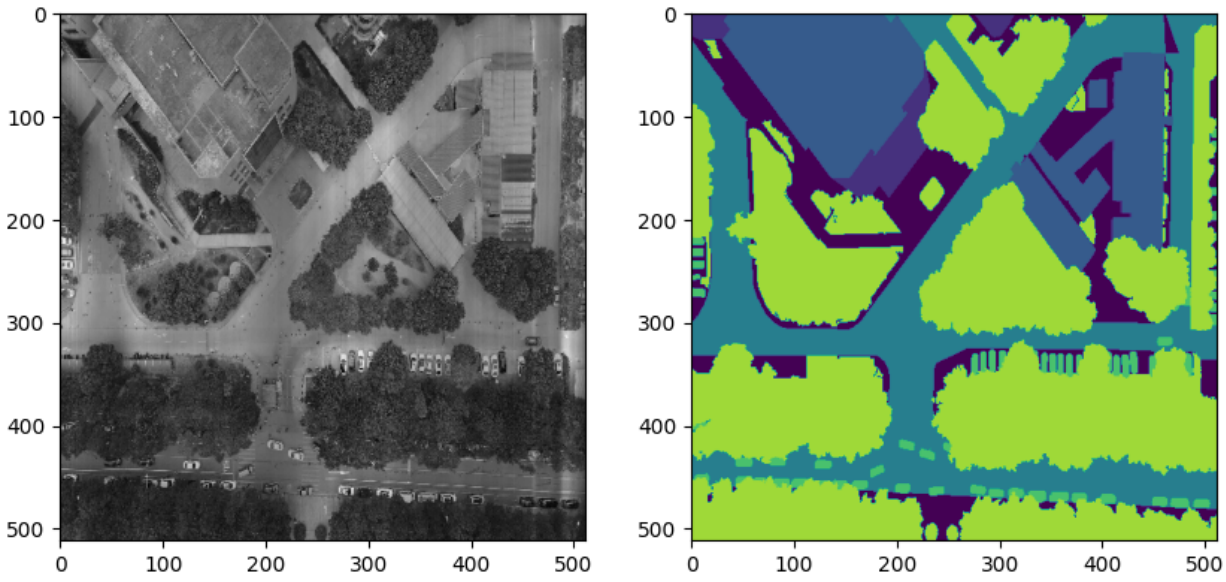

Figure 3: VDD Image and Mask Pair Sample

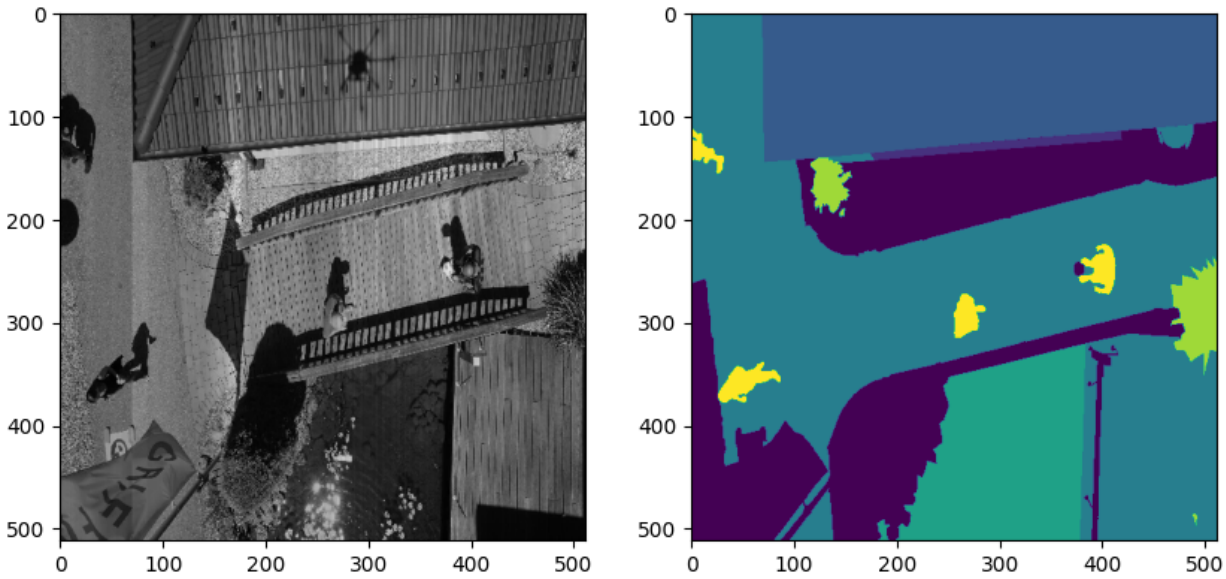

Figure 4: SDD Image and Mask Pair Sample

It can be seen in these sample images that the two datasets have images taken from different altitudes, but that the class values are uniform across the masks (right).

6.0 Modeling Architectures

Deep learning techniques provide a solution for complex problems such as pixel-level classification in an image. Among methods for performing semantic image segmentation (labeling the class of each pixel), the U-Net model architecture has become a foundational building block for many high performing model architectures. For this project, the Keras API was chosen for implementing the model.

6.1 U-Net

The U-Net model architecture for image segmentation was originally introduced in a paper published in 2015 (Olaf Ronneberger). The architecture consists of a contracting path (the encoder) to capture features through the use of convolutions and a symmetric expanding path that locates these features in an output image that is eventually the size of (or a similar size to) the input image. Across each symmetrical block in the architecture is a skip connection, which is a concatenation that further aids in localizing the features in the output image. The U-Net name comes from the U-like shape of the architecture, as seen below in Figure 5: U-Net Architecture:

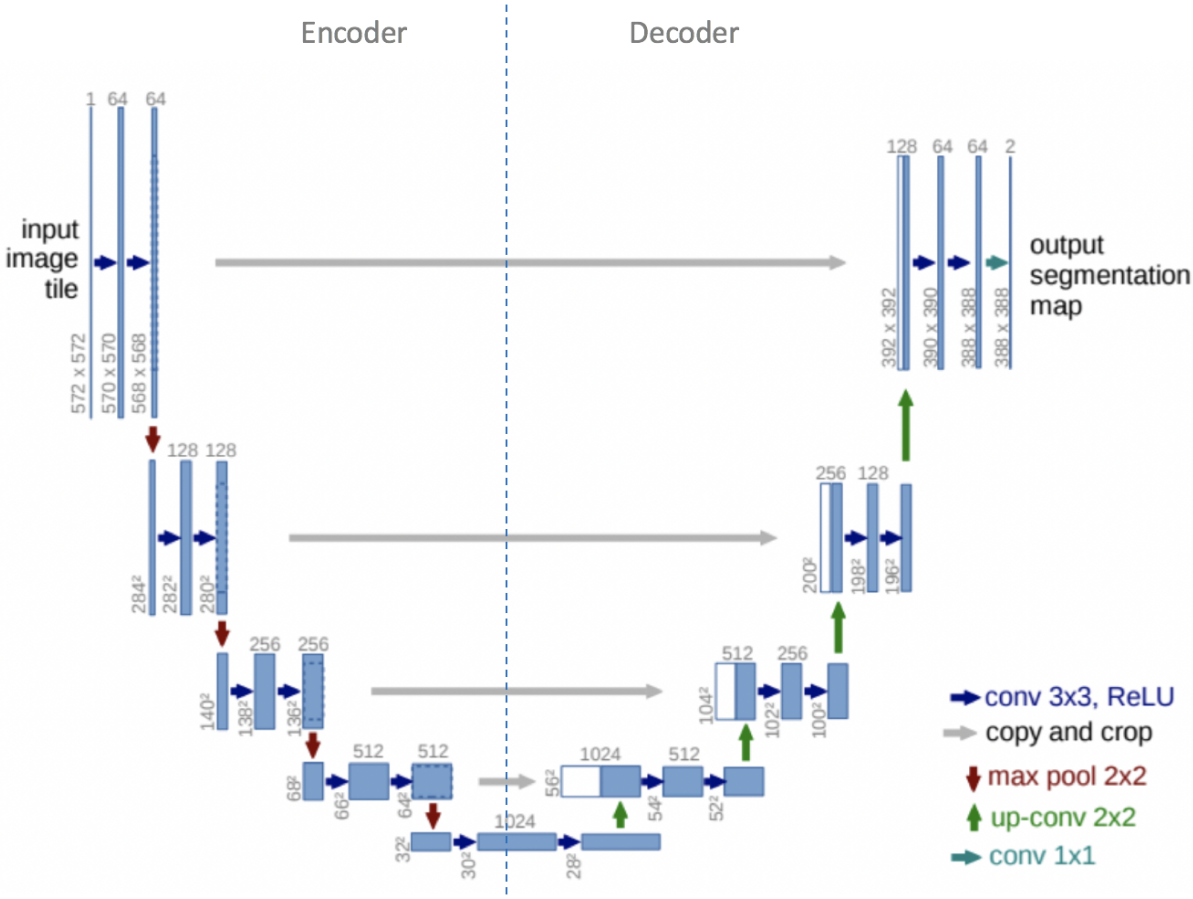

Figure 5: U-Net Architecture

The encoder is a series of four blocks consisting of two convolutions followed by ReLU activation. This is where the model creates features found in the input image. However, the convolutions in the encoder cause precise location in the original image to be lost. This is where the decoder comes in, by upsampling the image features back into the original resolution with the help of skip connections. The skip connections are points in the modeling process where concatenation occurs. This allows the features discovered in the encoder to be located in the decoded image.

However a key difference between the original U-Net architecture and the modified U-Net architecture used in this project is that the images are zero-padded to preserve image size through the convolutions. The tradeoff here is an increase in computation time for an output image that ends up as the size of the input image. Another change is the introduction of a dropout layer, which randomly forces a percentage of the layer weights to 0, to prevent overfitting to the training dataset. A visual representation of the modified U-Net architecture used in this project can be seen in Figure 6: Modified U-Net Architecture, below:

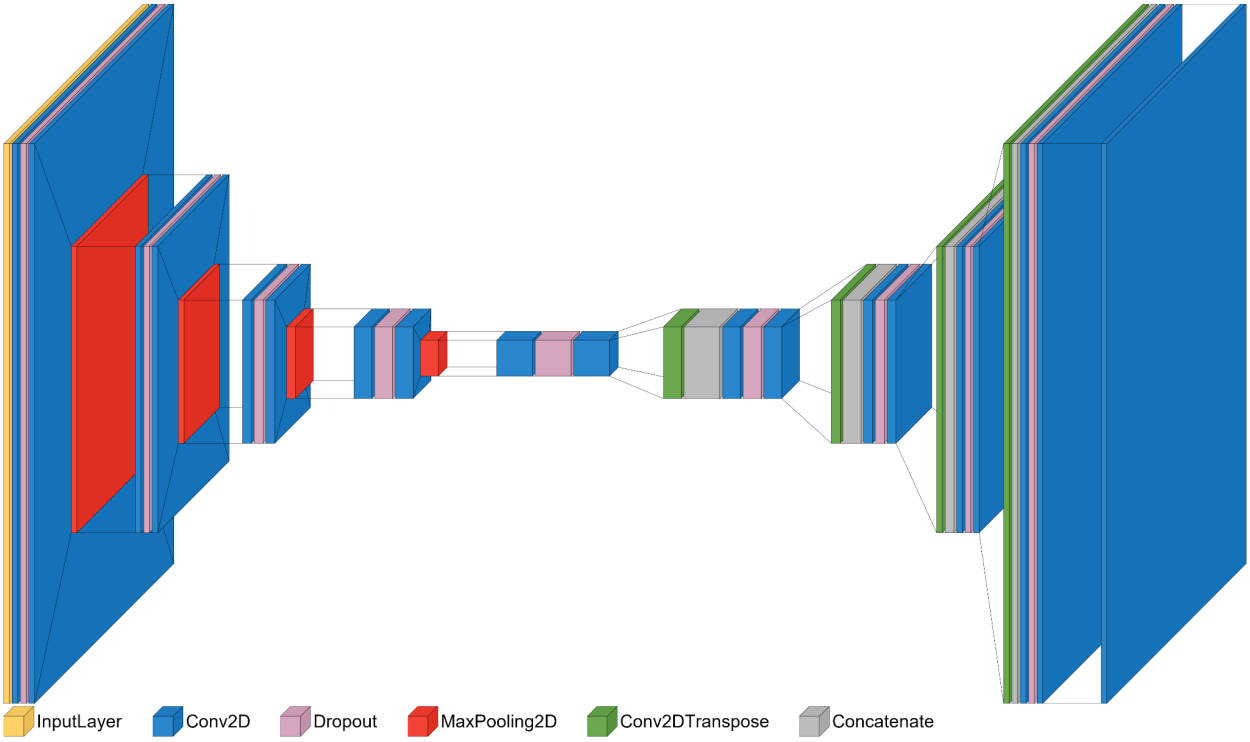

Figure 6: Modified U-Net Architecture

Furthermore, Table 4: Modified U-Net Architecture tabulates the model architecture and Table: 5 Modified U-Net Model Size tabulates the number of parameters, below:

Table 4: Modified U-Net Architecture

#	Layer (*)	Input Shape	Output Shape
0	InputLayer	(None, 512, 512, 1)	(None, 512, 512, 1)
1	Conv2D (3x3, same, ReLU)	(None, 512, 512, 1)	(None, 512, 512, 16)
2	Dropout (10%)	(None, 512, 512, 16)	(None, 512, 512, 16)
3	Conv2D (3x3, Same, ReLU)	(None, 512, 512, 16)	(None, 512, 512, 16)
4	MaxPooling2D (2x2)	(None, 512, 512, 16)	(None, 256, 256, 16)

5	Conv2D (3x3, Same, ReLU)	(None, 256, 256, 16)	(None, 256, 256, 32)
6	Dropout (10%)	(None, 256, 256, 32)	(None, 256, 256, 32)
7	Conv2D (3x3, Same, ReLU)	(None, 256, 256, 32)	(None, 256, 256, 32)
8	MaxPooling2D (2x2)	(None, 256, 256, 32)	(None, 128, 128, 32)
9	Conv2D (3x3, Same, ReLU)	(None, 256, 256, 32)	(None, 128, 128, 64)
10	Dropout (20%)	(None, 256, 256, 64)	(None, 128, 128, 64)
11	Conv2D (3x3, Same, ReLU)	(None, 256, 256, 64)	(None, 128, 128, 64)
12	MaxPooling2D (2x2)	(None, 256, 256, 64)	(None, 64, 64, 64)
13	Conv2D (3x3, Same, ReLU)	(None, 256, 256, 64)	(None, 64, 64, 128)
14	Dropout (20%)	(None, 64, 64, 128)	(None, 64, 64, 128)
15	Conv2D (3x3, Same, ReLU)	(None, 64, 64, 128)	(None, 64, 64, 128)
16	MaxPooling2D (2x2)	(None, 64, 64, 128)	(None, 32, 32, 128)
17	Conv2D (3x3, Same, ReLU)	(None, 32, 32, 128)	(None, 32, 32, 256)
18	Dropout (30%)	(None, 32, 32, 256)	(None, 32, 32, 256)
19	Conv2D (3x3, Same, ReLU)	(None, 32, 32, 256)	(None, 32, 32, 256)
20	Conv2DTranspose (2x2, 2x2, Same)	(None, 32, 32, 256)	(None, 64, 64, 128)
21	Concatenate (15)	(None, 64, 64, 128), (None, 64, 64, 128)	(None, 64, 64, 256)
22	Conv2D (3x3, Same, ReLU)	(None, 64, 64, 256)	(None, 64, 64, 128)
23	Dropout (20%)	(None, 64, 64, 128)	(None, 64, 64, 128)
24	Conv2D (3x3, Same, ReLU)	(None, 64, 64, 128)	(None, 64, 64, 128)
25	Conv2DTranspose (2x2, 2x2, Same)	(None, 64, 64, 128)	(None, 128, 128, 64)
26	Concatenate (11)	(None, 128, 128, 64), (None, 128, 128, 64)	(None, 128, 128, 128)
27	Conv2D (3x3, Same, ReLU)	(None, 128, 128, 128)	(None, 128, 128, 64)
28	Dropout (20%)	(None, 128, 128, 64)	(None, 128, 128, 64)
29	Conv2D (3x3, Same, ReLU)	(None, 128, 128, 64)	(None, 128, 128, 64)
30	Conv2DTranspose (2x2, 2x2, Same)	(None, 128, 128, 64)	(None, 256, 256, 32)

31	Concatenate (7)	(None, 256, 256, 32), (None, 256, 256, 32)	(None, 256, 256, 64)
32	Conv2D (3x3, Same, ReLU)	(None, 256, 256, 64)	(None, 256, 256, 32)
33	Dropout (10%)	(None, 256, 256, 32)	(None, 256, 256, 32)
34	Conv2D (3x3, Same, ReLU)	(None, 256, 256, 32)	(None, 256, 256, 32)
35	Conv2DTranspose (2x2, 2x2, Same)	(None, 256, 256, 32)	(None, 512, 512, 16)
36	Concatenate (3)	(None, 512, 512, 16), (None, 512, 512, 16)	(None, 512, 512, 32)
37	Conv2D (3x3, Same, ReLU)	(None, 512, 512, 32)	(None, 512, 512, 16)
38	Dropout (10%)	(None, 512, 512, 16)	(None, 512, 512, 16)
39	Conv2D (3x3, Same, ReLU)	(None, 512, 512, 16)	(None, 512, 512, 16)
40	Conv2D (1x1, Softmax)	(None, 512, 512, 16)	(None, 512, 512, 8)

*** The information with the parentheses are specific for each layer type when implemented in Keras:**

- **For Conv2D layers: (kernel size, padding option, activation function)**
- **For Dropout layers: (rate)**
- **For MaxPooling2D layers: (pool size)**
- **For Conv2DTranspose layers: (kernel size, strides, padding option)**
- **For Concatenate layers: (layer # to concatenate with)**

Table 5: Modified U-Net Model Size

Total Parameters	Trainable Parameters	Non-Trainable Parameters
1,940,936 (7.40 MB)	1,940,936 (7.40 MB)	0 (0.00 Byte)

When compiling this model in Keras, two arguments are required: a loss function and an optimizer function. The chosen loss function was *Categorical Cross Entropy* because there are more than two label classes and the image preprocessing provides labels in a one-hot representation. The chosen optimizer function was the *Adam* optimizer because, according to Kingma et al., 2014, the method is "computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters".

When training this Keras model, the batch size is a hyperparameter that needs to be chosen. The batch size is the number of images that are passed to the model before their error is analyzed by the optimizer and used to iteratively update the parameter weights to minimize the model error. A coarse batch size study was conducted using three-fold cross-validation to

test batch sizes of 8, 16, 32, and 64 up to 100 epochs. The three-fold cross-validation used even folds of the training data where one fold is held out as the validation dataset for scoring the model predictions during training. The results of this study can be visualized in Figure 7: Batch Size Study Results, below:

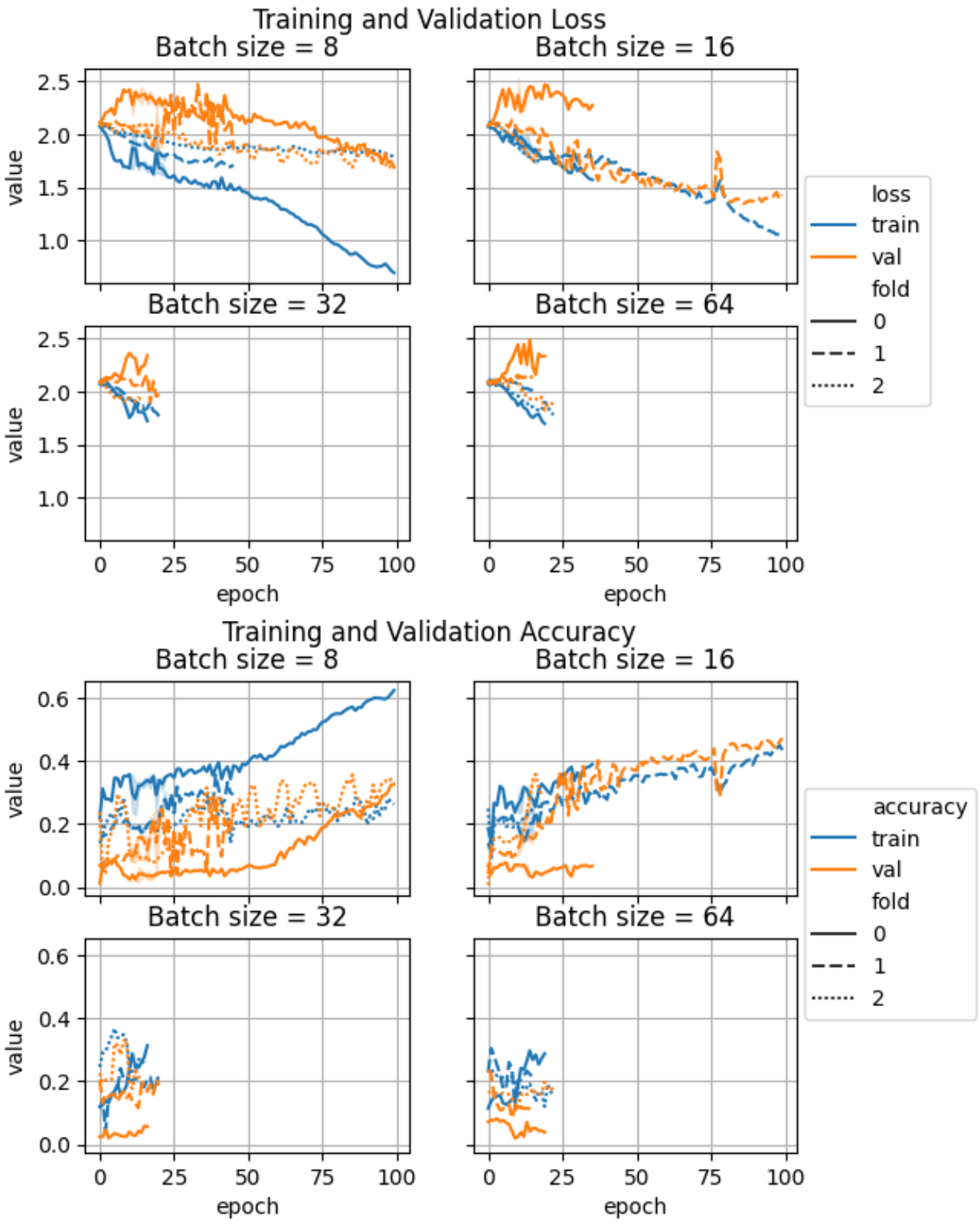

Figure 7: Batch Size Study Results

Note that training runs where the loss fails to improve fast enough results in a hard stop to the training. Although the training accuracy for a batch size of 8 was higher than that of 16

near 100 epochs, the validation accuracy was far below. This is a sign of training data bias as the model appears to overfit to the training data and perform poorly on the validation data. However, when the batch size was 16 images, the training and validation accuracy are close. This study concludes that this model trains the best with batch size of 16 images at a time.

Another hyperparameter that needs to be considered is the learning rate used by the Adam optimizer. The learning rate is the rate that the optimizer updates the parameter values during training. A coarse learning rate study was conducted using three-fold cross-validation as well for the learning rate values of 1e-4, 1e-3, 1e-2 and 1e-1 up to 100 epochs. The results of this study can be visualized in Figure 8: Learning Rate Study Results, below:

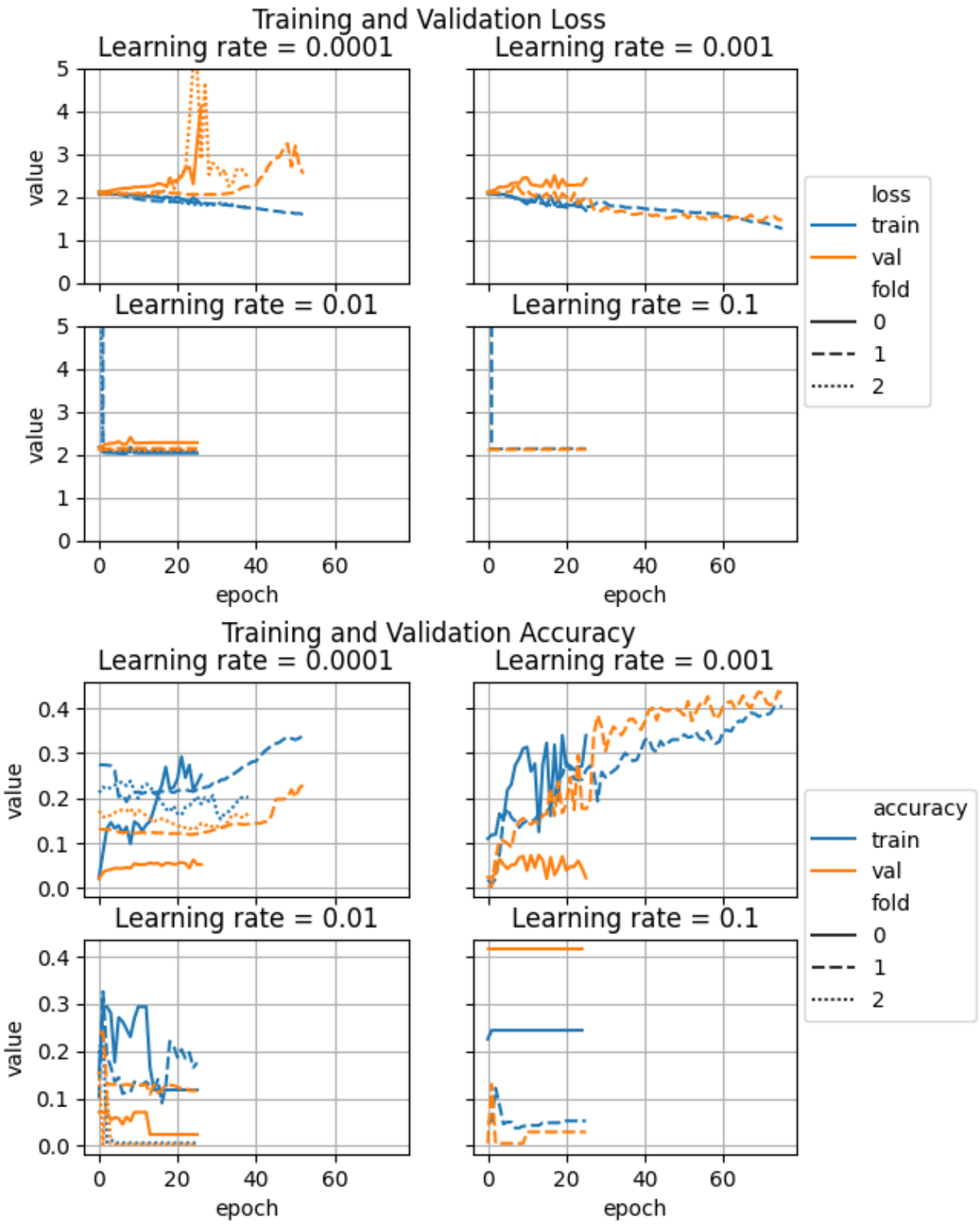

Figure 8: Learning Rate Study Results

Once again, training runs where the loss fails to improve fast enough results in a hard stop to the training. Additionally, when training the model with a learning rate of 0.001, the power

to the computer was interrupted, resulting in the training being stopped early. However, the results of the study were unaffected and a learning rate of 0.001 for the Adam optimizer resulted in the best model.

From the hyperparameter tuning studies, it was determined that the best values for the model's hyperparameters are a batch size of 16 and a learning rate of 0.001. Together, the furthest trained model that has been saved during this project has been one that was trained up to 50 epochs. This is the model that will be used to review its prediction accuracies.

At this point, the model performance has only been analyzed by its overall accuracy on its pixel classifications. But since there are 8 classes being predicted, this metric is not enough to tell the whole story. An excellent metric to score model performance with multi-classification is *intersection-over-union (IoU)*. For semantic segmentation, IoU can be defined as such:

$$IoU_i = \frac{True\ Positives_i}{\left(True\ Positives_i + False\ Positives_i + False\ Negatives_i\right)}$$

where *i* denotes a particular class. For the given class *i*, true positives are pixels correctly labeled class *i*, false positives are pixels labeled class *i* when they do not belong to it, and false negatives are pixels not labeled class *i* when they should be. The best model's predictions on the test dataset result in the following IoU scores, shown in Table 6: IoU Scores:

Table 6: IoU Scores

Class	IoU Score (%)
0 (unlabeled)	16.64
1 (wall)	10.25
2 (roof)	0.13
3 (road)	31.48
4 (water)	47.88
5 (car)	4.65
6 (vegetation)	32.06
7 (person)	2.80
Mean	18.24

At 50 epochs, these scores are not great. Considering that the model did continue to improve past 50 epochs during the hyperparameter studies, these IoU scores are expected to improve with further model training, until the loss converges. However, at this point it can be

pointed out that the model is appearing to struggle with the roof, person, and car classes the most. This can be viewed in the chart below, Figure 9: IoU Scores:

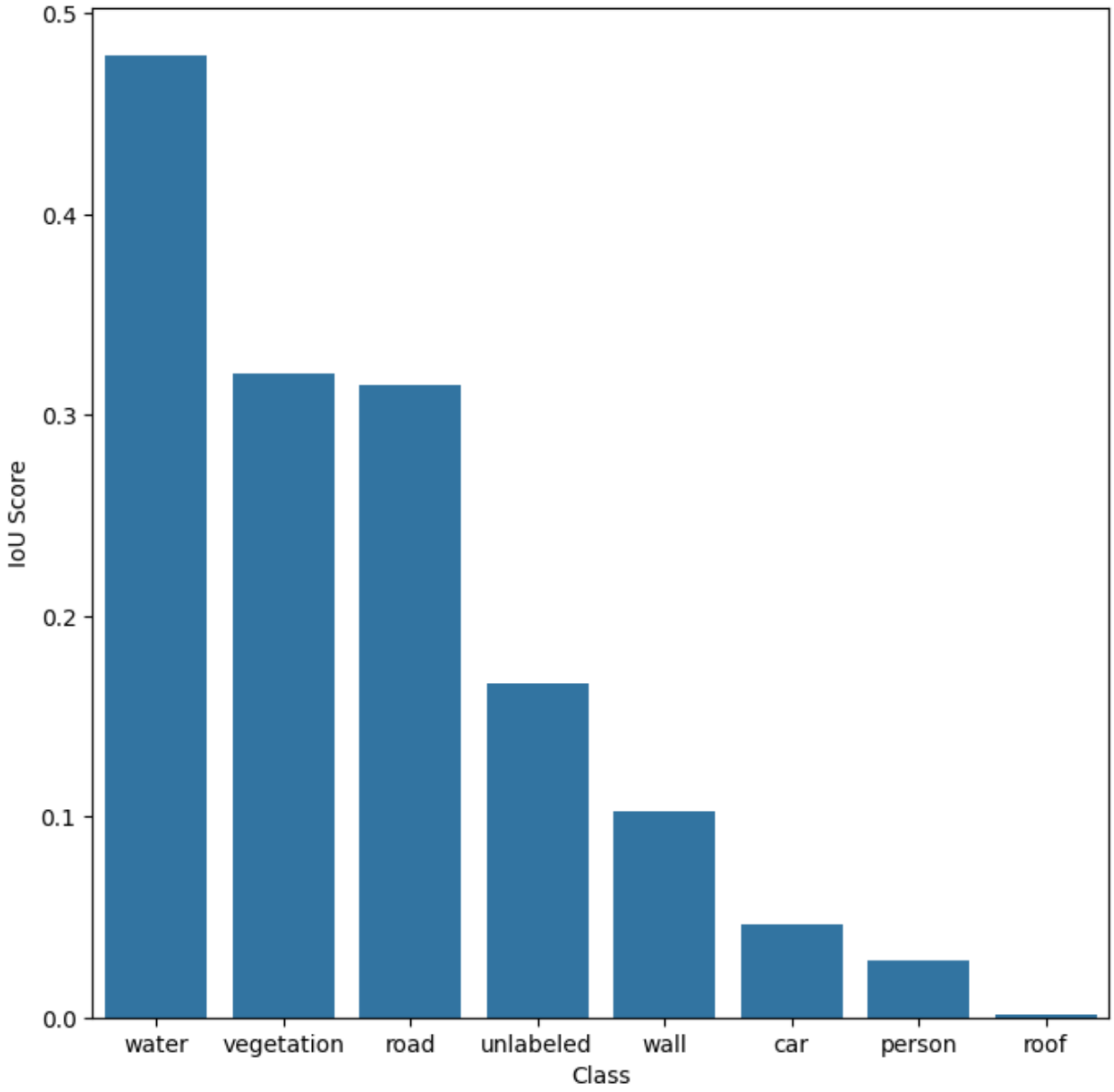

Figure 9: IoU Scores

Compared to the other classes, *roof* had a higher representation in the training images, and yet, it is the worst performing class, thus far. Plotting the predicted masks will show us why. Figure 10: Model Test Prediction Samples, below, shows a few test image and predicted mask pairs for the three least accurate classes:

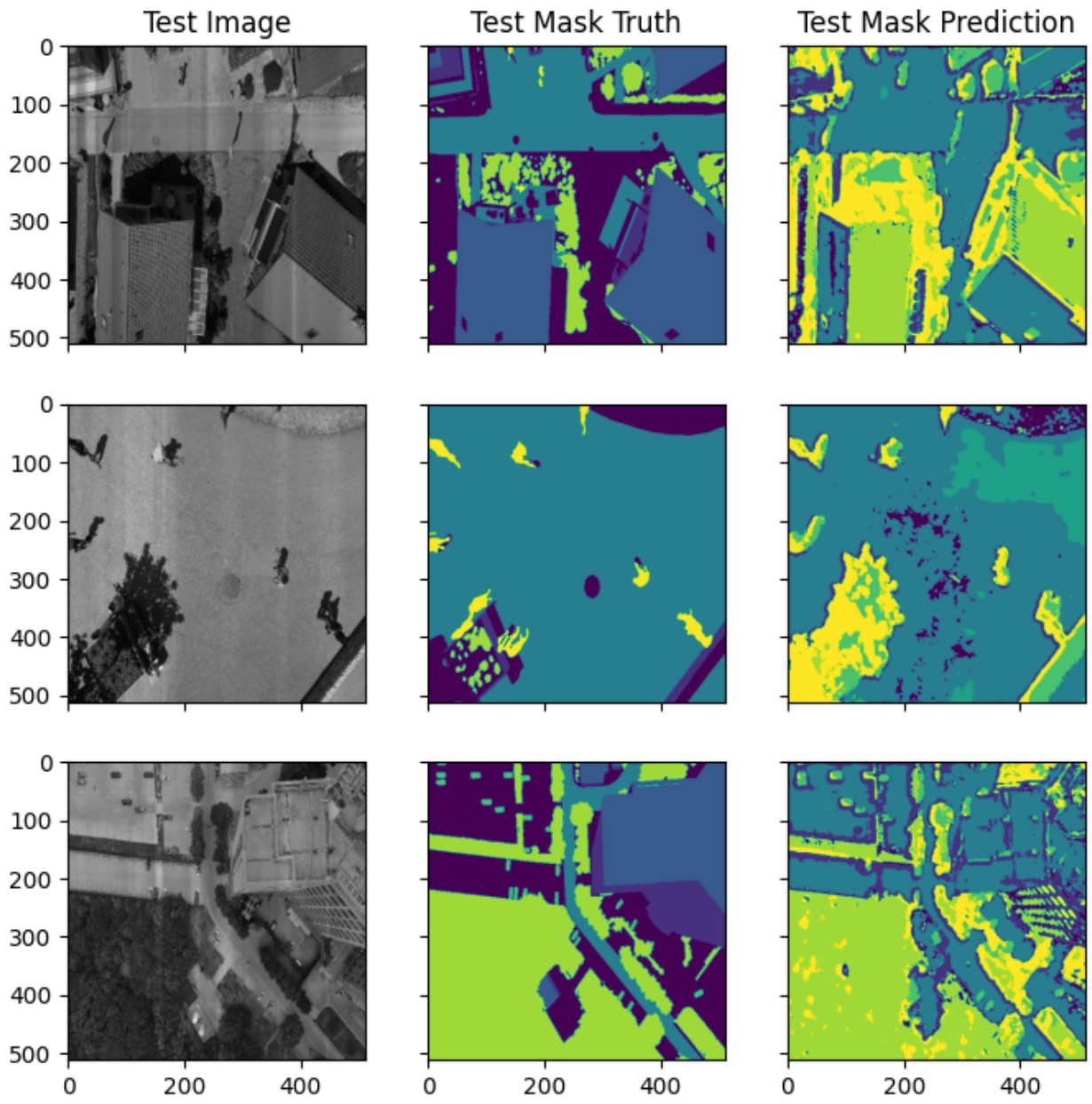

Figure 10: Model Test Prediction Samples

The first row shows an image with a few roofs. It can be seen in the predicted mask that the model is having trouble discriminating between the ground and roof as they are both being masked as the same value, whereas the ground truth mask shows them as different masks. The second row shows an image with several people. The model is able to predict that there are people, although their outlines are a little messy. However, it appears that the model is performing poorly in this class because it is also predicting pixels as persons when they are in fact not persons, such as the tree and its shadow in this case. The third row shows an image with quite a few cars parked in parking lots, however the model is once again predicting a lot

more than just cars as cars. This can be seen more clearly in the following plot, Figure 11: Model Test Prediction Samples (Highlighted):

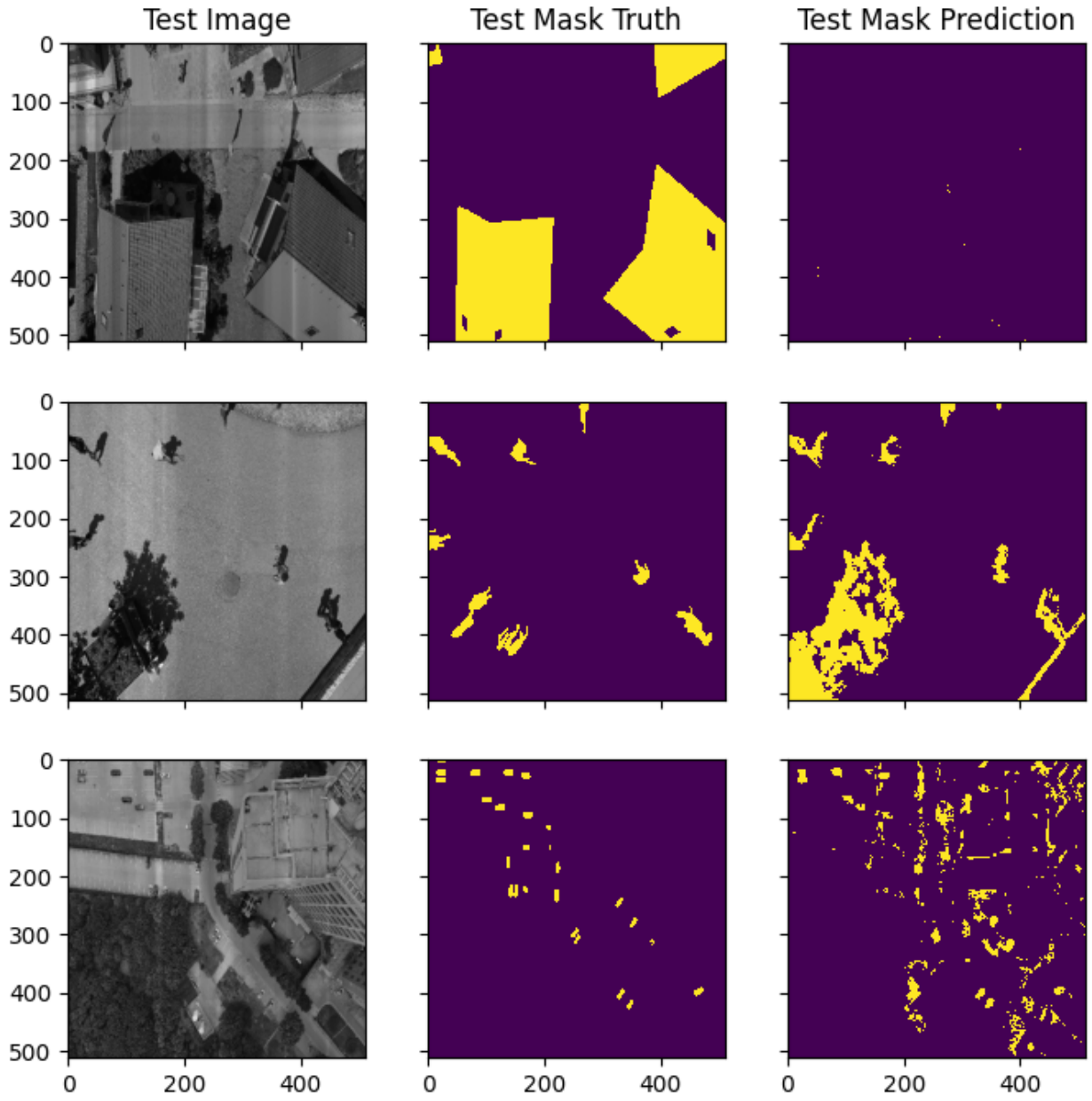

Figure 11: Model Test Prediction Samples (Highlighted)

Now, the same images from Figure 10 are shown again, but this time only the class of interest is highlighted. The first row shows that essentially none of the roof is being predicted as the roof (as it was all predicted to be the same class as the road. The second row shows that both people and other objects such as some trees and shadows are being predicted to be of the person class. The third row shows that less-intuitive features in the images are being labeled as the car class.

6.2 MobileNet

As aerial drones are designed to avoid excess weight, on-board computers are likely limited to only contain what is necessary to perform their necessary tasks. MobileNets are a class of efficient models created for mobile and embedded vision applications (Andrew G. Howard). MobileNetV2 was created as an improvement and can be combined with a reduced form of DeepLabV3 to perform mobile semantic segmentation models (Mark Sandler). For the application of such a model as discussed in this project, this type of architecture is an important one to note.

7.0 Conclusions and recommendations

The modified U-Net architecture used for training this semantic segmentation model trained best with a batch size of 16 images and a learning rate of 0.001 for the Adam optimizer. The performance of this model was analyzed after 50 epochs of training, although it is noted that the model had not yet converged at 50 epochs.

The predicted class accuracy (intersection over union) is very poor for classes 6 (car), 8 (person), and 3 (roof) -- all below 5% accuracy. The best predicted class was water (label 4) with an IoU score of 47.88%. Some classes, such as class 8 for persons failed to perform well because the model labeled more than just people as this class. Furthermore, the model does not seem to be able to discern the difference between roofs (class 2) and roads (class 3) well, as the model tends to predict them both as roads.

For continued work on this project, there are a few recommendations. First, the current modified U-Net model was trained until 50 epochs, but was not yet converged, as the loss was still decreasing and both training and validation accuracies were rising. It is recommended to extend the training of this model to convergence to investigate the model's potential performance improvements. Second, something to explore could be to combine classes that tend to be confused with one another such as persons and vegetation into something such as "obstacles". Third, is to train the model on individual datasets first before combining datasets together. This would allow a baseline performance for each dataset to be established to understand how the introduction of new data with new perspectives could be affecting the model's individual class prediction accuracies. And fourth, is to train a MobileNetV2 model combined with DeepLabV3 to test how using different model architectures compare for this use case.

8.0 Bibliography

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications; https://arxiv.org/abs/1704.04861.

Christian Mostegel, Michael Maurer, Nikolaus Heran, Jesus Pestana Puerta, Friedrich Fraundorfer: Semantic Drone Dataset; http://dronedataset.icg.tugraz.at/.

Diederik P. Kingma, Jimmy Ba: Adam: A Method for Stochastic Optimization, 2014; https://arxiv.org/abs/1412.6980.

Kingma, Diederik P., Ba, Jimmy: Adam: A Method for Stochastic Optimization; https://arxiv.org/abs/1412.6980.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen: MobileNetV2: Inverted Residuals and Linear Bottlenecks, 2019; https://arxiv.org/abs/1801.04381.

Olaf Ronneberger, Philipp Fischer, Thomas Brox: U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015; https://arxiv.org/abs/1505.04597.

VDD: Varied Drone Dataset for Semantic Segmentation; https://vddvdd.com/.