ECE 285 Final Project Report: Semantic Segmentation Using U-Net

Patrick Youssef
Computer Science and Engineering
A13860272

Abstract

In this report we'll go over some foundational techniques used for semantic segmentation using Convolutional Neural Networks (CNNs). After covering some recent, we will implement a well-know model and present results and improvements. In particular we will implement a U-Net [4] model and improve the performance modern techniques such as batch norm and transfer learning. At the end of the paper I will train the model on the CityScapes dataset [1] and present the results.

1 Introduction

The utility of solving the semantic segmentation problem has a strong impact on the autonomous vehicle sector among other ones. At it's core, the problem is classifying individual pixels of an input image to define fine boundaries between object types. This can be useful in defining free space in a scene, develop an understanding of a scene, or other such tasks that require higher fidelity understanding of object position. When reviewing modern deep learning principles it is obvious that a convolutional neural network is the obvious tool to solve this, but that doesn't define the model.

Given the prior, I will implement a model based on the works done in "U-Net: Convolutional Networks for Biomedical Image Segmentation" [4] and train it on the CityScapes dataset [1] to see how well it performs. In addition, I will be modifying the base model with some more modern augmentations and see the performance implications. Due to hardware constraints I have made some dataset modifications which I detail later.

2 Related Work

There are multitudes of methods in the modern era for attempting the semantic segmentation problem but I want to focus on the relationship between U-Net and Fully Convolutional Networks (FCN) [3]. I consider the U-Net as an advancement of the base FCN model for a few reasons.

- **Multiple Decoders:** U-Net has a chain of decoders as opposed to the one decoder convolution in FCN.
- **Concatenation vs Summation:** In improved FCN variants with skip connections, prior feature layers are added to later layers which may maintain feature dimensions but does not maintain all the original information. In U-Net, prior feature maps are concatenated with decoder features to maintain all information.

From my reading, it seems like the space of semantic segmentation models has gotten much more complicated but generally keeps base inspiration from the methods of both of these models. As U-Net seems like a progression of FCN, I opted to implement that model.

3 Method

As mentioned prior I opted to implement a U-Net [4] as my base model for this problem. Let's first formalize the problem of semantic segmentation before we can discuss how the model solves it.

3.1 Problem Statement

Semantic segmentation is the problem of predicting classes for each pixel of an input image. Given an image of $[H, W, C]$ (where C is the number of channels) the goal is predict class scores over each pixel. The result is a matrix of size $[H, W, N_{classes}]$. To convert the class scores into a resulting segmentation of size $[H, W, 1]$ we assign the class with the highest score to each pixel using argmax along the class dimension. At this point we have a format that is similar to the ground truth for supervised learning. Generally speaking our model is a function such that:

$$f(\text{Image } [H, W, C]) \rightarrow \text{Class Scores } [H, W, N_{classes}]$$

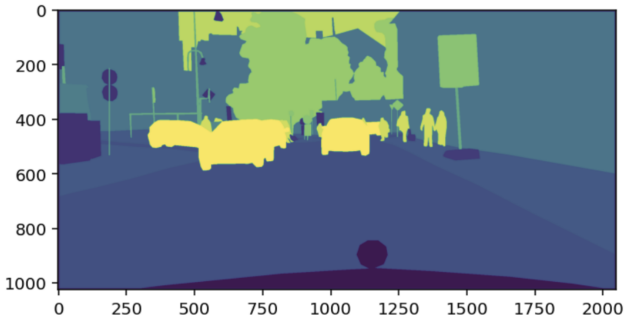

Figure 1: Example segmentation from the CityScapes dataset.

One of the difficulties of this problem when considering supervised learning is how to attain the ground truth segmentation. Unlike classification, obtaining the ground truth segmentation requires decently precise masks to be manually applied. This is one reason that segmentation datasets often can be smaller. That being said, when selecting a dataset we need to ensure that the quality of the segmentation masks provided are good.

3.2 Model Architecture

As mentioned prior I opted to implement a U-Net architecture. The skipped connections and multi-level feature maps heavily improved the performance of this model as compared to prior models. This model is by no means state of the art, but many of the core principles of the improved performance from this model have extended into modern models. Figure 2 shows the base architecture that I will implementing as from the original paper. We can consider the model as 2 distinct components:

3.2.1 Encoder

The purpose of the encoder in the U-Net is to encode the original image into multiple-level feature maps that are later used to compute the final segmentation. In particular, the U-Net has distinct blocks in the encoder, each with 2 convolutional layers. After each block, the representation is saved for the decoding step and is max-pooled for the next encoder block.

3.2.2 Decoder

The purpose of the decoder is to upsample the encoded features into the final segmentation. Similarly to the encoder, the decoder has distinct blocks composed of a series of convolutions that result in the final class scores. In particular, each decoder block starts with a transpose convolution that decreases the channel count but doubles the feature map dimensions which over time leads to the original image size. Next, prior features from the encoder are concatenated with the transposed results and passed

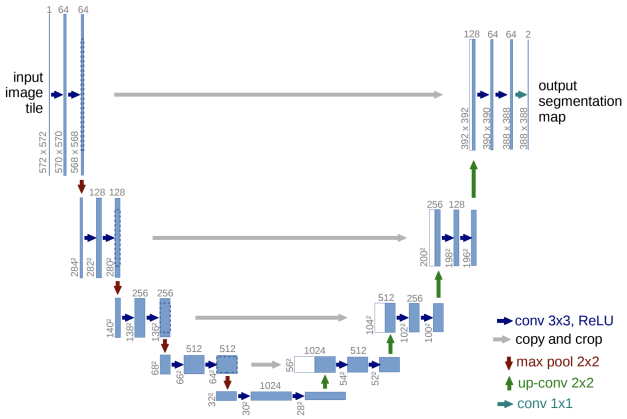

Figure 2: U-Net architecture from the original paper.

through 2 convolutional layers for further decoding. This is repeated through all prior features until we have a feature map of the same size as the original image, but with as many channels as there are classes. This can easily be reduced into the class predictions through an argmax.

3.2.3 Modifications

Although the original model clearly defines the layers used in all components, I opted to make some changes to simplicity and improvements. I will cover these in more detail later on but to mention them here:

- Modified the encoder architecture to implement VGG-19 pre-trained encoder layers.
- Opted to use padded convolutions rather than valid convolutions to maintain a consistent feature map size on each level.
- Added batch norms to improve training rate and performance [2].

The exact details of how the model is implemented is detailed in the code but I will give an overview of the layer composition for my modified U-Net. Note that the layer labeling in Table 1 follows the same scheme as from the original VGG paper [5] where the number after conv is the kernel size and the second value is the output channel count. All convolutions also have a batchnorm applied after so that is not listed here for the sake of brevity. Unless stated otherwise all other aspects of the base U-Net are as detailed in the original model. Feature map size is for a 128x128 image, and feature map size are for when the feature are saved (directly after convolution and ignoring pooling until further convs). Given that I am using padded convolutions to maintain consistent feature maps sizes on each level it's quite easy to infer the dimensionality of the decoder blocks. For this reason I will not be detailing them for the sake of brevity.

4 Experiments

In this section, I will cover my implementation, training, and results. Alongside the initial results I will also cover changes that I made to improve the performance and some data size testing. To start, I trained a base U-Net with a nearly identical structure to that of the original and a pre-trained variant that I detailed prior.

4.1 Dataset Used

As segmentation has seen a lot of utility in the autonomous vehicle sector I decided it would be fitting to use the CityScapes dataset [1] of typical driving scenes. The dataset contains frames from a car-mounted camera driving around in 6 different cities (Berlin, Bielefeld, Bonn, Leverkusen, Mainz, and Munich). Alongside the base images finely annotated object class images are also provided with 35 class descriptions. I did make some changes to the dataset which I will detail below.

Block Name	Layers	Pre-Trained	Feature Map Size
Encoder1	conv3-64	Yes	
	conv3-64	Yes	$[64, 128, 128]$
Encoder2	conv3-128	Yes	
	conv3-128	Yes	$[128, 64, 64]$
Encoder3	conv3-256	Yes	
	conv3-256	Yes	
	conv3-256	Yes	
	conv3-256	Yes	$[256, 32, 32]$
Encoder4	conv3-512	Yes	
	conv3-512	Yes	
	conv3-512	Yes	
	conv3-512	Yes	$[512, 16, 16]$
Encoder5	conv3-512	Yes	
	conv3-512	Yes	
	conv3-512	Yes	
	conv3-512	Yes	
	conv3-1024	No	$[1024, 8, 8]$

Table 1: Overview of pre-trained encoder structure.

4.1.1 Class Reduction

Table 2 details the class reduction that I implemented as the original 35 classes were not performing well.

New Classes	Included Base Classes
void	unlabeled, ego vehicle, rectification border, out of roi, static, dynamic, ground
flat	road, sidewalk, parking, rail track
construction	building, wall, fence, guard rail, bridge, tunnel
object	pole, polegroup, traffic light, traffic sign
nature	vegetation, terrain
sky	sky
human	person, rider
vehicle	car, truck, bus, caravan, trailer, train, motorcycle, bicycle, license plate

Table 2: Class reduction overview

4.1.2 Image Size Reduction

Due to memory constraints and training time on my hardware I opted to square crop and resize the input images to 128x128 with a batch size of 32.

4.1.3 Input Transforms

To help with training and to meet the needs of the pre-trained model I performed some transforms on the input data before training.

- **Crop and Resize:** To streamline cropping and resizing I made a custom transform using PIL that can be applied with the typical PyTorch loader transforms.
- **Scaling:** Input images were scaled to a range of $[0, 1]$
- **Mean and STD Centering:** The pre-trained model was trained on ImageNet so I need to match the mean and standard deviation of their dataset.

4.1.4 Ground Truth Transforms

Alongside the input transforms I also had some applied to the ground truth segmentations.

- **Crop and Resize:** Same crop and resize as with the input so that dimensions match.
- **Class Reduction:** The class relabeling was done as another custom transform.

4.2 Training

For training, a subset of 3000 images was used to train the model. To manage overfitting I compute the validation loss every few training batches and keep track of this on TensorBoard. If a validation loss outperforms the prior best a model checkpoint is saved and the best loss model is used for testing.

4.2.1 Optimizer

After a lot of testing I opted on using the Adam optimizer with no weight decay and a decaying learning rate as detailed in Figure 3. This decay was the result of a decay rate of 0.95 applied on each epoch. As we are trying to predict the correct class of each pixel, this is a large scale classification problem well suited to using cross entropy loss.

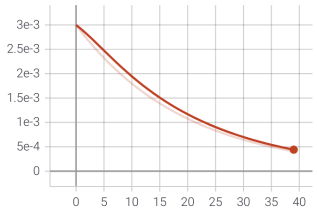

Figure 3: Learning rate vs Epoch

4.3 Initial Results

Here I will detail the results of my testing on the base model architecture with minor modifications as well as my VGG-19 pre-trained variant.

4.3.1 Losses

As I mentioned, I initially trained two variants of the U-Net (base and VGG-19 variants). The loss curves that you see in Figure 4 show the results for both the models with the pre-trained variant greatly outperforming the base. Note that the x axis is not indicative of anything in particular as it's merely the count of updates performed to each variable.

4.3.2 Metrics

Alongside the losses and images below I also opted to compute two valuable metrics for this type of problem: mean pixel accuracy and mean intersection over union.

Model	Metric	Performance
Base	Pixel Accuracy	0.635
	IoU	0.243
Pre-Train	Pixel Accuracy	0.857
	IoU	0.530

Table 3: Overview of metrics performance.

4.3.3 Example Segmentations

Below are example images that I selected to show the strengths and weaknesses of the performance of this model. Additionally I present the best results from the pre-trained variant as well as the base

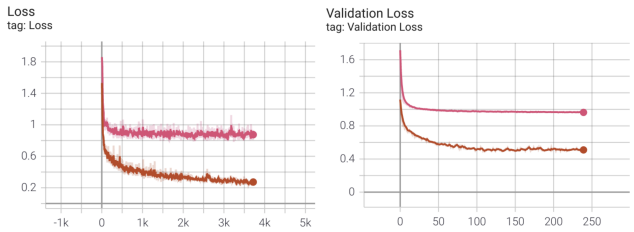

Figure 4: Pink: Base Model, Orange: Pre-Trained

variant. As you'll see the pre-trained variant qualitatively outperforms the base model. Figure 5 are the pre-trained results and Figure 6 are the base model results.

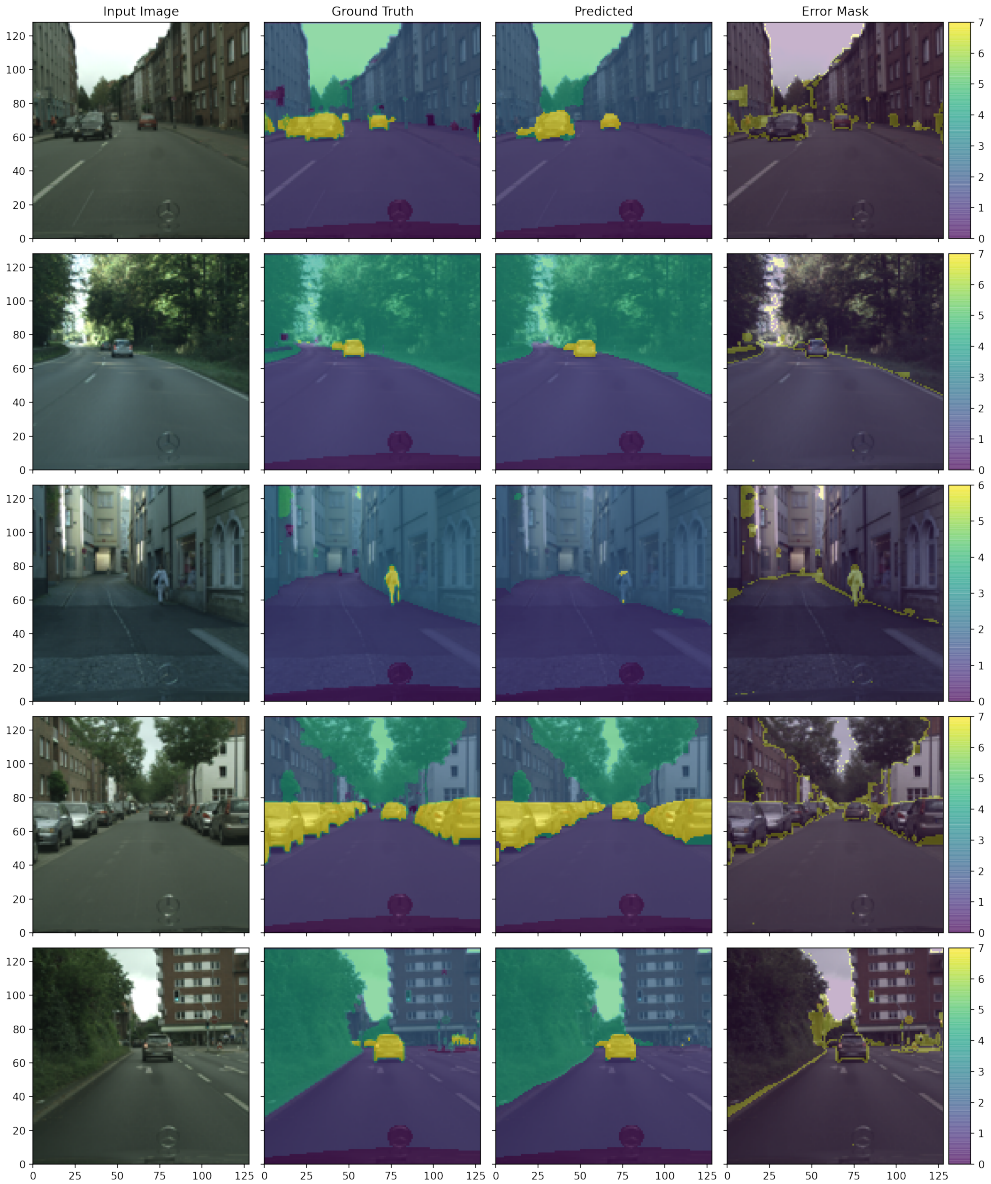

Figure 5: Best Pre-Trained Model Results

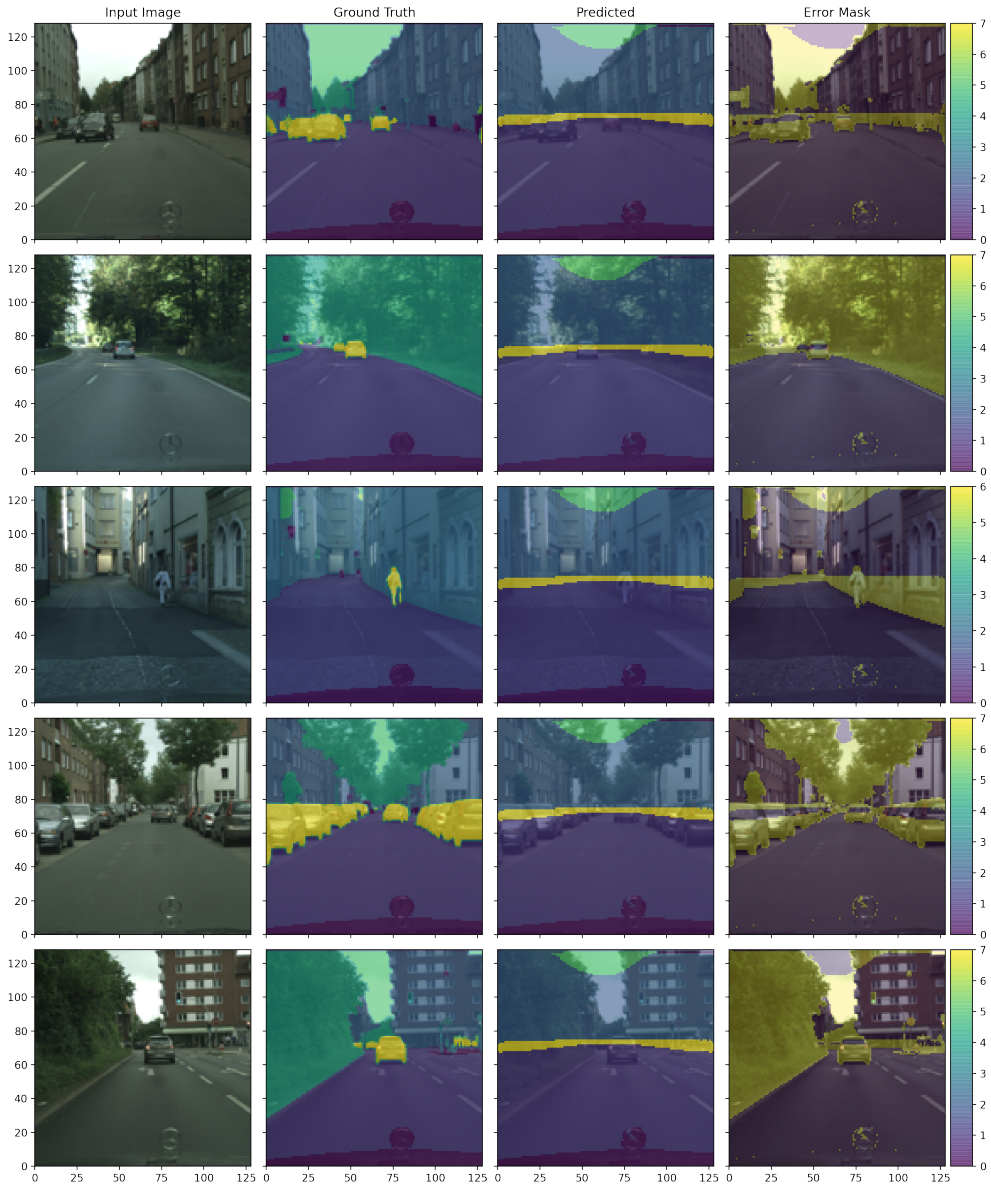

Figure 6: Best Base Model Results

4.3.4 Strength & Weaknesses

I want to discuss some of the strengths and weaknesses I've noticed from the grid of images. When I refer to performance, I am looking at the pre-trained results.

Strengths:

- In bulk groups there is very little pixel misclassification. When looking at the road for example the boundaries are relatively clean. The absolute position of boundaries is not perfect but spurious misclassifications are minimal.
- Vehicles, buildings, and roads are classified to a very usable degree.

Weaknesses:

- Small objects are almost entirely lost. Small cars, signs, and people are almost completely lost in the predictions and grouped into the surrounding class.

4.4 Dataset Size Testing

Alongside the base testing, I also opted to try training both of my models on a dataset half the size to see what the implications of a smaller dataset are. This is especially interesting as segmented data can be hard to gather, so if less data can give us similar results that would be great. As before I kept track of the training and validation loss throughout training and composed collages of the results.

4.4.1 Losses

When looking at the impact of the smaller dataset we can see that the loss impact seems less severe for the base model as compared to the pre-trained model. We can see an appreciable gap in both the training and validation loss for the pre-trained models in Figure 7.

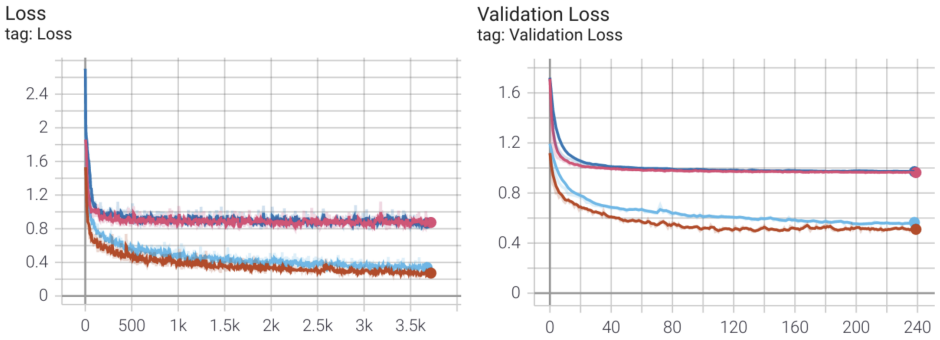

Figure 7: Pink (Base), Orange(Pre-Trained), Blue (Base Half Data), Cyan (Pre-Trained Half Data)

4.4.2 Metrics

As with the two prior models I computed the same metric on the half-data models as well. To my surprise the results are not too different between the different data amounts. The IoU of the base model on half of the data slightly outperforms the whole data set which is a surprise but it's within training margin of error in my opinion. Generally speaking it seems that the implications of a data set reduction of this degree are much less severe than originally expected.

Model	Metric	Performance
Base	Pixel Accuracy	0.634
	IoU	0.239
Base Half	Pixel Accuracy	0.635
	IoU	0.243
Pre-Train	Pixel Accuracy	0.857
	IoU	0.530
Pre-Train Half	Pixel Accuracy	0.832
	IoU	0.530

Table 4: Overview of metrics performance.

4.4.3 Example Segmentation

Although the metric performance is not too different, the qualitative performance is very different for the pre-trained model. We can see in the center image that the half model performs much worse than the prior results that we've seen. This indicates that either that image was an outlier in the general performance or that the metrics alone are not enough to explain the performance.

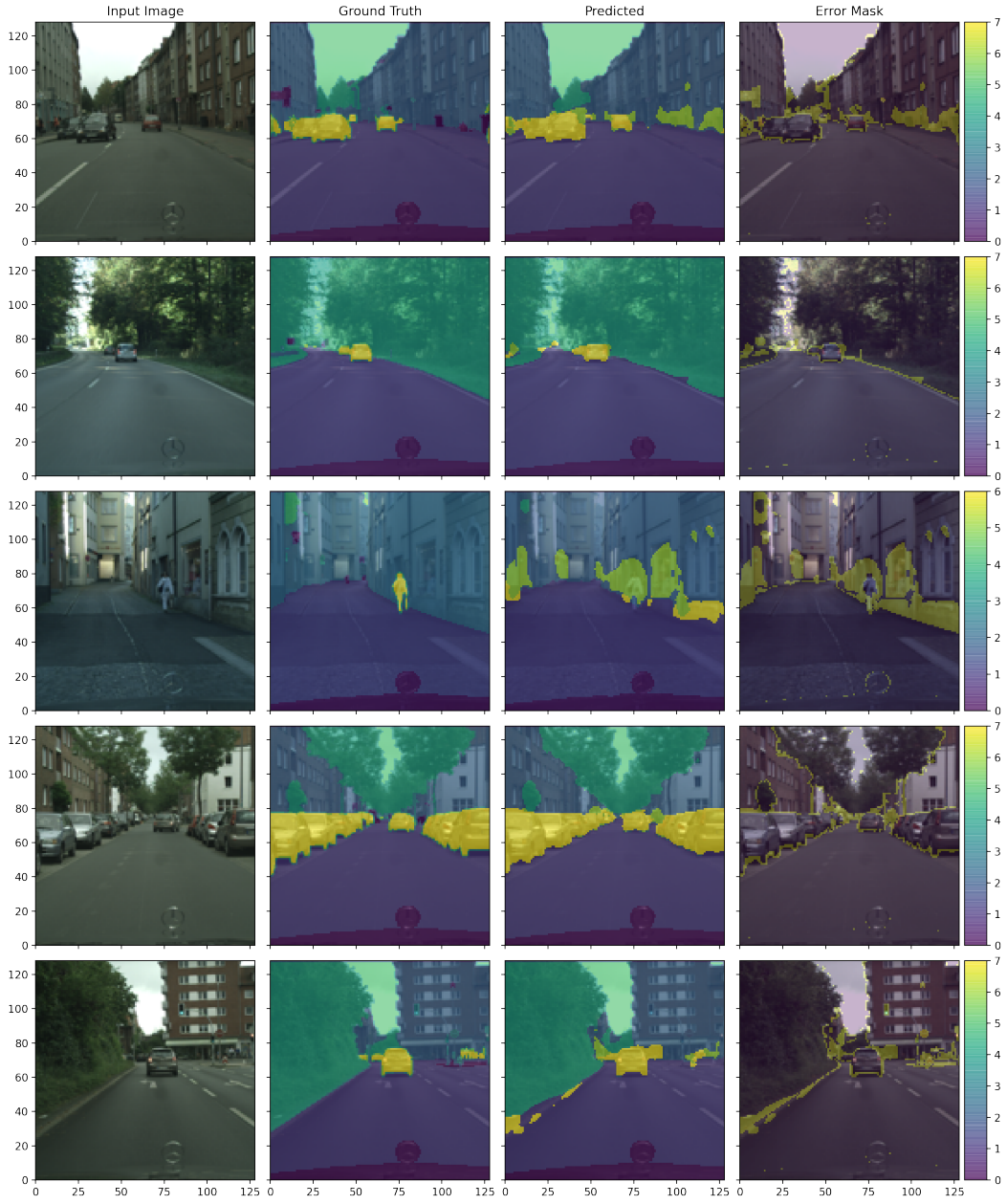

Figure 8: Best Pre-Trained Half Data Model Results

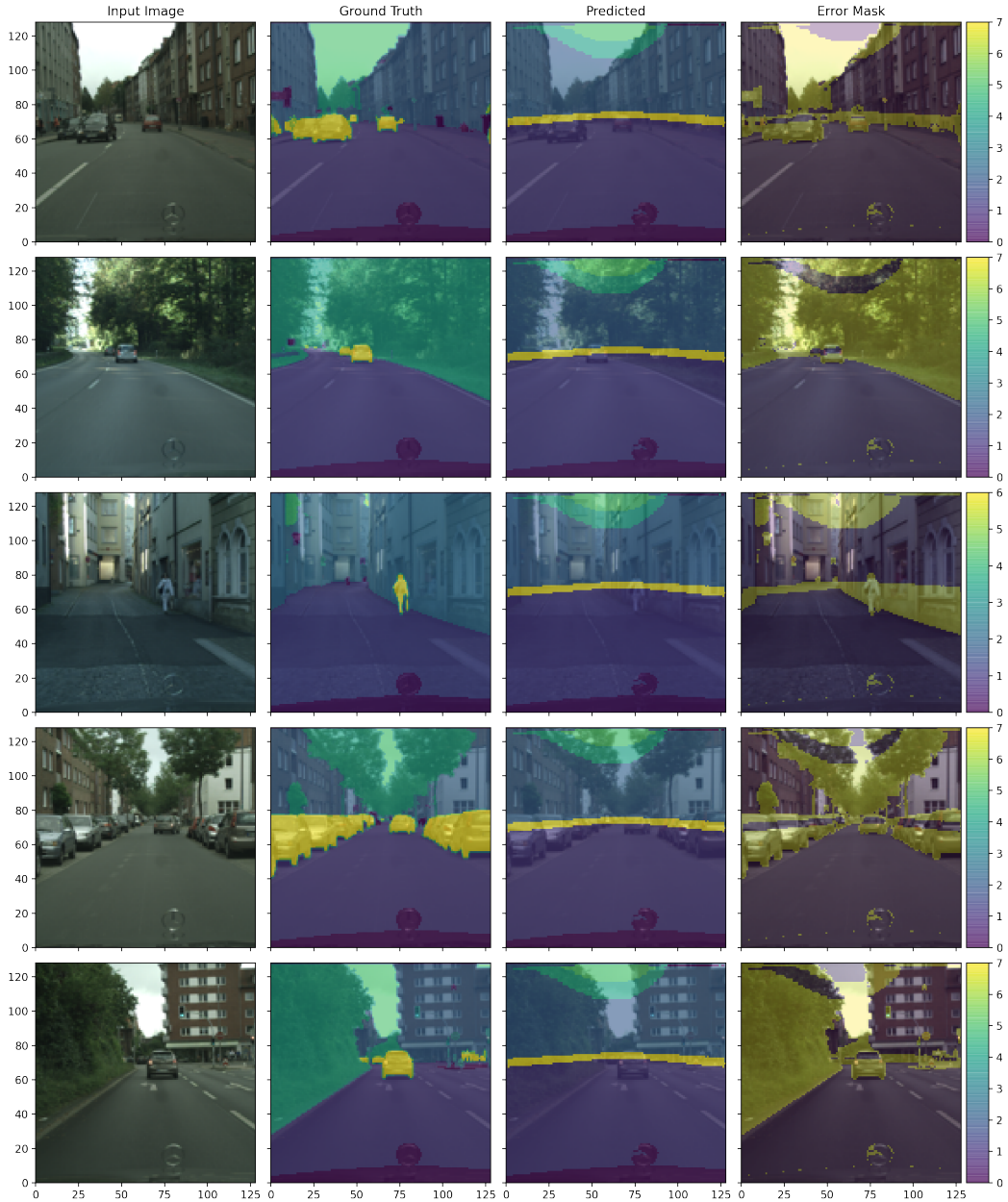

Figure 9: Best Base Half Data Model Results

4.5 Improvements

There is always more to be improved in these types of problems, and although a modern model will very likely improve the results there are changes that can be made to this work.

- **Training on larger images:** In my results the images were cropped to a fairly course 128x128 which can make it difficult to detect smaller features. I like to think about the convolution kernel scanning across and how many respective feature values would correlate to that object. For smaller objects this may only be 1-2 which can be difficult to keep track of through the model, even with the skipped connections.
- **Better Encoder:** Although the pre-trained VGG encoder helped immensely, an encoder similar to a ResNet-50 [5] would improve the feature representation.

References

[1] Marius Cordts et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[2] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[3] Jonathan Long, Evan Shelhamer, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2015. arXiv: 1411.4038 [cs.CV].

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[5] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].