ETHZ CIL 2023 : Road Segmentation

Timothé Laborie Fisnik Mustafa Elif Emanet Shubham Chowdhary

timothelaborie@gmail.com, mustafaf@student.ethz.ch, emanete@student.ethz.ch, schowdhary@student.ethz.ch
Group: Hard Baseline, *Department of Computer Science, ETH Zurich*

Abstract

In this work, we tackle the problem of road segmentation from aerial imagery. We leverage deep learning methods, specifically the Segformer B5 (Xie et al., 2021), UperNet-ConvNext-XLarge (Xiao et al., 2018)(Liu et al., 2022), and OneFormer (Jain et al., 2022) models, and train them on a new large dataset of satellite images from Google Maps. We use geometric image transformations like random flips to augment the data. We adopt a multi-stage training procedure to prevent overfitting and preserve pre-trained weights. We test different ensemble methods to optimize the final predictions. Our approach achieves a public F1 score of 0.94186 on Kaggle, beating last year's previous best record of 0.94037 and demonstrating the effectiveness of the selected models for the task. However, we note that challenges remain, such as the segmentation of parking lots and obstructed roads. We also present a new operator called PaSSIV to improve the performance of models in low-data settings.

1 Introduction

1.1 What is road segmentation

Road segmentation is a computer vision task that specifically aims to distinguish road pixels from non-road pixels in an image. The complexity of this task lies in the diverse and intricate patterns of roads, which can drastically vary across different regions, weather conditions, and lighting conditions. Further complications arise from the presence of occlusions such as cars, trees, pedestrians and buildings.

1.2 Related work

Road segmentation has been an area of active research in the computer vision community for many years. Early approaches primarily relied on traditional machine learning methods and handcrafted features. For instance, (Mokhtarzade & Zoej, 2007) utilized an approach based on the pixel intensity and texture features. However, these methods tend to struggle with the complexity and diversity of road layouts, weather conditions, and occlusions.

The advent of deep learning has brought significant improvements to the task of road segmentation. Convolutional Neural Networks (CNNs), have demonstrated their effectiveness in many segmentation tasks, including road segmentation. For instance, (Hinton, 2010) used CNNs for road extraction from aerial imagery.

In recent years, transformer-based architectures have become increasingly popular in computer vision tasks. Recent models such as Segformer, Oneformer and Upernet get strong performances on segmentation tasks. These models, however, weren't specifically trained for road segmentation.

2 Data

2.1 Official dataset

The official dataset provided consists of 144 train and 144 test images, each with a resolution of 400x400 pixels. Each image, taken from an aerial view, is paired with a corresponding road mask. These images are primarily from urban areas in the USA.

2.2 Data collection

Using the Google Maps static API, we obtained an additional 21466 images along with their road masks. We removed those with less than 2 percent of road presence, resulting in a total of 18297 usable images. All these images are 400x400 pixels, with a zoom level of 18, which yields a car size of about 12x7 pixels, matching the official dataset. 8968 of the images are from Boston, the rest are from Los Angeles (LA). We chose these cities because they look similar to the official dataset. We collected these images by defining a bottom-left and top-right corner, then iterated over latitude/longitude increments of 0.002 (Boston) and 0.0027 (LA) to get every image within the rectangle. We used a larger step size for LA as it is a larger city and we wanted the images to cover a larger part of it.

Before collecting this dataset, we initially collected a smaller dataset of only 1500 images to test the performance gains of using Google Maps data. After seeing large performance improvements, we decided to collect more images.

3 Training setup

Each model was trained using a pixel-wise binary cross-entropy (BCE) loss function. This allows us to treat each pixel as an independent binary classification problem. We trained on a combined dataset of the official and Google Maps images, keeping 10% of the official dataset for validation.

3.1 Training procedure

The training procedure consists of a multi-stage approach. In the first stage, we train only the model's classification head, the decoder and the last encoder layer for two epochs on the small Google Maps dataset with a learning rate of 1e-4. This is done for weight initialization purposes. We then unfreeze the remaining layers, excluding the patch embeddings. After a warmup epoch for the AdamW (Loshchilov & Hutter, 2017) optimizer to learn the normalization factors, we proceed to train for three epochs with a learning rate of 1e-4 on the full dataset, followed by one epoch with a learning rate of 1e-5. We then fine-tune the model by training it on the official dataset for 5 more epochs. This process is designed to preserve the pre-trained weights these models come with. The fine-tuning step at the end allows the model to adjust to the slightly different input distribution of the official data.

We chose these values for the learning rates and epoch counts so that the learning rate gets reduced right as the train loss stops decreasing, and also verified the effectiveness by checking the validation score.

3.2 Image augmentation

The image augmentations we used are horizontal flip, vertical flip, and random 90-degree rotation, each with a probability of 0.5. Through some testing, we found that they slightly improve the validation score.

4 Models

4.1 Baselines

To evaluate the effectiveness of our chosen models and methods, we compared their performance against several baselines.

4.1.1 Patch-wise classifier

Our first baseline is a simple model that classifies each 16x16 patch of pixels independently from each other. The model first extracts 8 values for each patch and does the classification using a linear combination of them. It obtains a score of 0.60248.

4.1.2 SEGFORMER B5 with no external dataset

To test how much our Google Maps dataset improves performance, we trained a Segformer b5 model using only the official dataset. This approach achieved a public Kaggle score of 0.91584, much lower than the score of 0.93575 it gets when additionally trained on external data.

4.1.3 SEGNET with no external dataset, trained from scratch

SegNet (Badrinarayanan et al., 2015) is a deep learning architecture for pixel-wise semantic segmentation. Our implementation of SegNet did not use any pre-trained weights. We trained it on the official dataset only. The idea was to see what performance could be achieved without any external resources. As expected, SegNet performed considerably worse compared to other models, achieving a score of only 0.85486.

4.2 Models we used

The models we used are Segformer b5, UperNet-ConvNext-XLarge, and Oneformer. We tested many of the models available on HuggingFace and we chose these models because they performed better than others in our testing.

4.2.1 SEGFORMER B5

Segformer is a modern architecture for semantic segmentation. It combines the advantages of transformers and multi-scale feature fusion, making it effective for high-resolution semantic segmentation tasks. The b5 variant we used has 84 million parameters.

The model's classification head contains a convolutional layer that outputs 768 features per group of 4x4 pixel. We modify the model by adding a bilinear upsampling layer followed by a 1x1 convolution to output the road masks.

4.2.2 UPERNET-CONVNEXT-XLARGE

UperNet is a type of semantic segmentation network that utilizes pyramid pooling and global pooling to deal with objects of different sizes. In the UperNet-ConvNext-XLarge model, ConvNext, a new type of convolutional neural network, is used as the encoder. This model has 391 million parameters.

Model	public score
Segformer	0.93575
Upernet-convnext	0.93884
Oneformer	0.93742
majority voting	0.94060
averaged logits	**0.94186**

Table 1: Scores of the models we used for the final submission.

To make the model usable for road segmentation, we re-define the final 1x1 convolutional layer to have only one output channel.

4.2.3 ONEFORMER

OneFormer is a transformer-based universal image segmentation framework that surpasses conventional frameworks in semantic, instance, and panoptic segmentation tasks, utilizing a single universal architecture. This model has 222 million parameters.

For our work, we only need semantic segmentation. We modify the model's architecture as follows:

1. We remove the Hungarian matcher from the end of the model, resulting in an output size of (250, 100, 100). These values are the outputs of an MLP.
2. We apply bilinear upsampling to get an output size of (250, 400, 400)
3. We add a 1x1 convolution to get an output size of (1, 400, 400)

5 Methods

5.1 Ensemble

We tested four different ensemble methods.

5.1.1 Majority voting

We first averaged the predictions for each 16x16 image patch across the three models, considering a patch to be a road if the average prediction was above a threshold of 0.25. Then, we performed majority voting among the three models for each patch. This achieved a score of 0.94060.

5.1.2 Weighted patch-wise predictions

For this method, we keep track of the average prediction per patch and per model, then use a weighted average where the best model has a higher weight. We then threshold the resulting prediction at 0.25. Regardless of the weights, this method did not result in any improvements over the score that the best model already gets on its own.

5.1.3 Averaging logits

For each pixel we average the logits of the three models and pass the result into the sigmoid function, then treat a patch as a road if the average prediction is above 0.25. This method achieved a score of 0.94186.

5.1.4 Averaging logits with weights

This method is like the previous one, except we additionally reduce the weight of the worst model (Segformer) when averaging the logits. This always resulted in a slightly worse score.

6 Results

A detailed summary of the results for each model can be found on Table 1.

7 Experiments

7.1 Benefits of increasing dataset size

We checked the performance of Segformer and Upernet with both the small and the full Google Maps dataset to see how much the full dataset increases the score. Both models were trained until the validation score stopped increasing. For Segformer, changing from the small dataset to the full one increased the score from 0.92744 to 0.93575. For Upernet, the score went from 0.93355 to 0.93884.

7.2 Changes to the training procedures

7.2.1 Training the whole model right away

We observed that our custom training procedure increases the score by about 0.005 compared to the score obtained when training the whole model right away.

7.2.2 Training the patch embeddings

We found that unfreezing the patch embeddings of the models does not help the performance.

7.2.3 Fewer epochs

The 4th epoch, which is done at a lower learning rate of 1e-5, increases the public score by 0.009. Fine-tuning for 5 epochs on the official dataset further increases the score by 0.0025.

7.3 Photometric transformations

We tried to augment the data using photometric transformations such as random brightness and contrast adjustment, Contrast Limited Adaptive Histogram Equalization (CLAHE), Hue-Saturation-Value (HSV) adjustment, and Gaussian noise. Through some testing with Segformer and Segnet we found that they usually made the performance worse, so we decided not to use them.

7.4 Different prediction procedure

Initial attempts at prediction involved generating predictions 10 times for each test set image, each time with a different image augmentation, and averaging the logits. However, this approach did not yield a performance gain, possibly because the models were trained to ignore these augmentations. As a result, we opted for a simpler approach of generating predictions just once for each image.

7.5 Exploiting Road Geometry: PaSSIV

PaSSIV, **Pa**th **S**earch via **S**emantic **I**nterpolation in **V**iew, is a novel attempt at improving the performance achievable by transformer models in low data regimes, such as when only using official data. The idea is to introduce a trainable inductive-bias on how roads look like typically. PaSSIV works in two steps: (1) it learns how road geometry looks like from the labels, and then (2) it augments the predictions of the backbone model by adjusting the predictions on potential road segments, especially the thin and occluded ones. This method **resulted in improved generalization score on Kaggle test over a vanilla OneFormer** in low-data regime. More details are available in section D.4.

8 Discussion

8.1 Failure cases

One common issue we observed across the models was the difficulty in handling parking lots. The models would often create many small roads that lead to nowhere. We attribute this to the fact that in the training data, the parking lots often had small, unpredictable ”road” labels going through them. In Figure 6 we can see that for this particular test set image, the model is not very confident with its labeling.

For images where roads were completely obstructed by large buildings, the models were unable to find the roads.

In Figure 1 we can also see that the shadow of a large building is making the predictions look weird in the bottom left corner.

8.2 Further improvements

In this section we discuss other ideas that could potentially increase the score further, at a cost of making the training process much longer or more complicated.

8.2.1 More dataset cleaning

Despite removing the images that are less than 2% roads, the Boston images in the Google Maps dataset contain a larger ratio of rural versus urban images compared to the official dataset. Removing some of the rural images could have potentially improved the score by making the train set distribution closer to the test set distribution.

8.2.2 Separating Boston and Los Angeles

Using the Google Maps dataset, it would be possible to train a classifier to detect the city of a test set image. A separate road segmentation ensemble could then be trained for both cities.

8.2.3 Image retrieval + fine-tuning

The procedure would be as follows:

1. Train the model as normal, on the full Google Maps dataset
2. Find the 100 nearest images for the first test image using a pre-trained vision model
3. Train the model for one epoch on those 100 images, then predict the labels for the test image
4. Revert the model back to the old weights, repeat for all other test images

This idea effectively creates an ensemble of expert models. Each model would be specialized for the particular type of image that it needs to predict.

9 Conclusion

In conclusion, road segmentation from aerial imagery remains a challenging problem despite the advances in deep learning and the availability of large-scale datasets. We present an ensemble of models that out-performs the previous best score, introduce a new operator, and present some ideas for potential future performance improvements.

References

Badrinarayanan, V., Kendall, A., and Cipolla, R. Segnet: A deep convolutional encoder-decoder architecture for image segmentation, 2015.

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

Demir, I., Koperski, K., Lindenbaum, D., Pang, G., Huang, J., Basu, S., Hughes, F., Tuia, D., and Raskar, R. Deepglobe 2018: A challenge to parse the earth through satellite images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

Hinton, V. M. . G. E. Learning to detect roads in high-resolution aerial images, 2010.

Jain, J., Li, J., Chiu, M., Hassani, A., Orlov, N., and Shi, H. Oneformer: One transformer to rule universal image segmentation, 2022.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. A convnet for the 2020s, 2022.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization, 2017.

Mnih, V. *Machine Learning for Aerial Image Labeling*. PhD thesis, University of Toronto, 2013.

Mokhtarzade and Zoej, V. Road detection from high resolution satellite images using artificial neural networks, 2007.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation, 2015.

Xiao, T., Liu, Y., Zhou, B., Jiang, Y., and Sun, J. Unified perceptual parsing for scene understanding, 2018.

Xie, E., Wang, W., Yu, Z., Anandkumar, A., Alvarez, J. M., and Luo, P. Segformer: Simple and efficient design for semantic segmentation with transformers, 2021.

A Failure cases (val set)

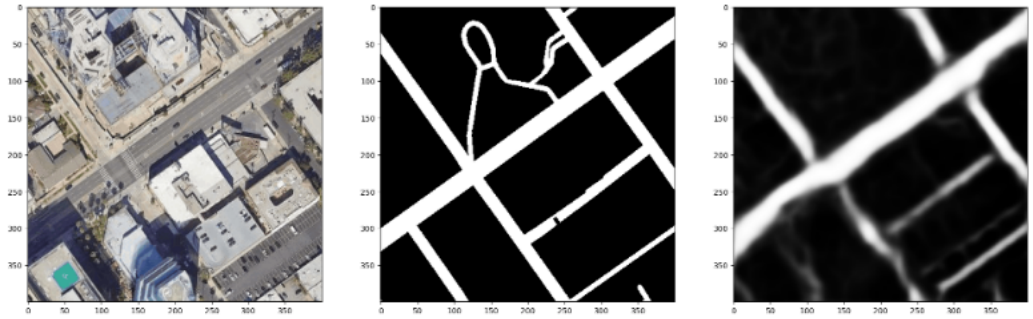


Figure 1: The model cannot predict the obstructed road, and the shadows cause the predictions at the bottom to be messy. (middle image is the label, right is prediction)

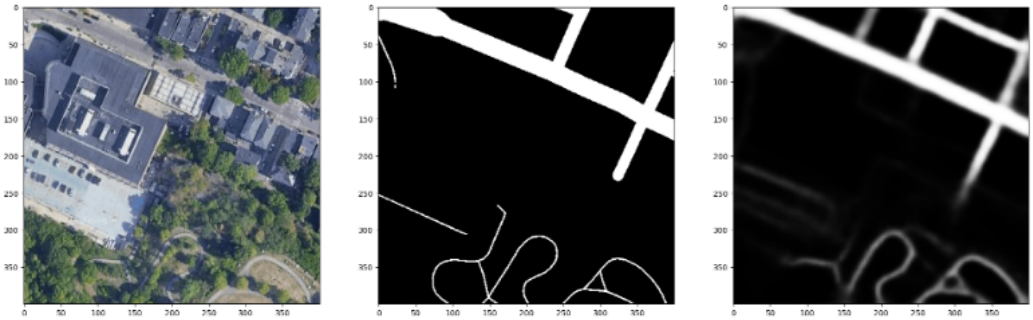


Figure 2: The model creates a road that doesn't exist at the top

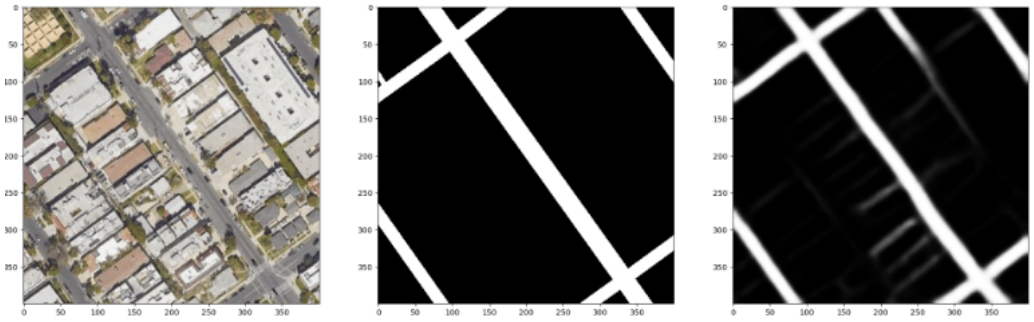


Figure 3: The model thinks that there are roads between the houses

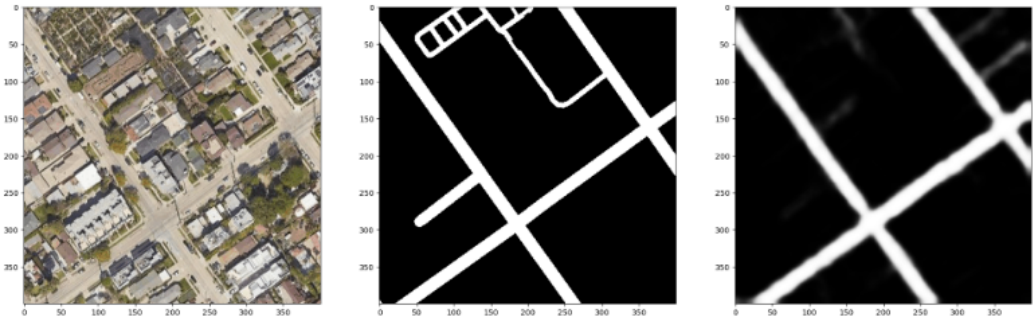


Figure 4: The model fails to detect many roads

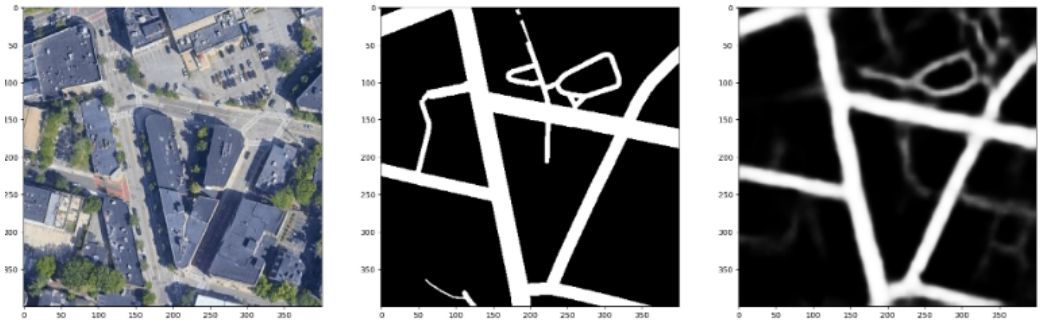


Figure 5: This type of image is very difficult to label correctly

B Failure cases (test set)

Figure 6: When seeing parking lots, the models get confused.

C Comparisons between the models on some test set images

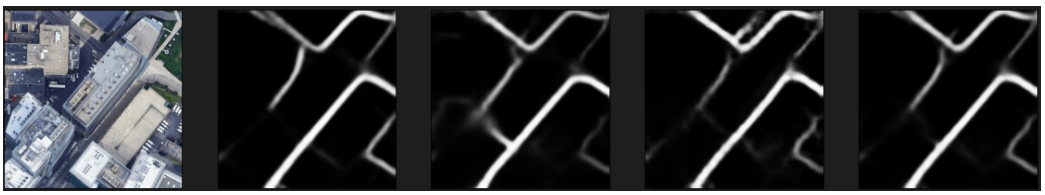


Figure 7: Comparison between the models. In order: oneformer, upernet, segformer, averaged logits

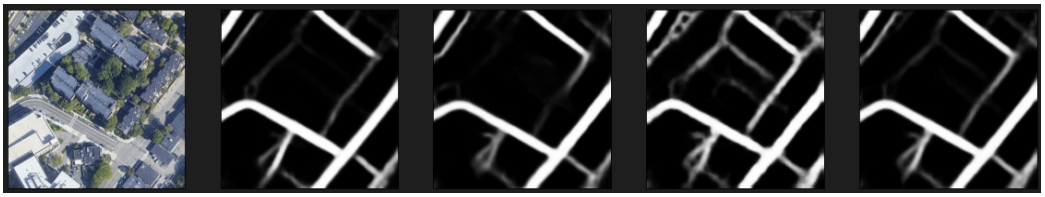


Figure 8: same order as above

D Bonus content

These sections were removed from the experiments section to make the report fit within 4 pages.

D.1 Choosing the prediction threshold

Through some testing we found that setting the prediction threshold at 0.3 instead of the default value of 0.25 frequently achieves a slightly better score on the public leaderboards. This is highly suspicious, however, for two reasons: First, when we setup a local validation loop to test the F1 score achieved by different thresholds, we found that 0.25 is almost always better. Secondly, the public leaderboard score is calculated using only 72 images, which means there is a high chance that some of these images are tricking the models into predicting roads that do not exist, which would explain why a higher threshold results in a slightly better public score. We therefore decided to use a threshold of 0.25, as this is apparently what was used to convert the original labels into patches.

D.2 Performance of alternative models

Besides the three models from our ensemble, we also tested other models. The models were tested using the small Google Maps dataset as it takes a very long time to train on the full size one. To make the comparison with the models in the ensemble fairer and to counteract the fact that the following models get fewer updates due to the smaller dataset, we double the number of epochs for the stages where the full Google Maps dataset was used in the normal training procedure (Section 3.1). The rest of the training procedure is unchanged.

D.2.1 U-NET

U-Net (Ronneberger et al., 2015) is a convolutional network architecture for segmentation of images, initially designed for biomedical image segmentation. U-Net achieved a score of 0.90446. We think this might be because the architecture is very old and cannot compete well with more modern architectures.

D.2.2 DEEPLABV3+

DeepLabv3+ (Chen et al., 2018) is a model for semantic image segmentation. It combines the strengths of Deep Convolutional Neural Networks (DCNNs) and Atrous Convolution to segment objects at multiple scales, as the encoder module and a simple yet effective decoder module is used to obtain sharper segmentations. DeepLabv3+ achieved a score of 0.90862. Upon examination, we found that it was incapable of detecting small roads. The architecture likely isn't designed for this task.

D.2.3 UPERNET-SWIN

This model is like UperNet-ConvNext-XLarge, but it uses the Swin Vision Transformer as the encoder instead of ConvNext. This model has 234 million parameters. Despite having over twice as many parameters as Segformer, it performs significantly worse and gets a score of just 0.92440. This suggests that the choice of the encoder in the UperNet model has a significant impact on the performance.

D.3 Alternative datasets

Before downloading a dataset from the Google Maps API, we tried to find and use existing datasets. Due to their divergence from the official dataset, we found it to be far more effective to simply use the Google Maps dataset instead.

D.3.1 MASSACHUSETTS ROADS

This dataset consists of 1500x1500 pixel satellite images of Massachusetts (Mnih, 2013). Due to the lower zoom level, we randomly cropped 400x400 pixel patches for consistency with the official dataset. Some images in this dataset were corrupted, being partially masked and having mismatched labels. We eliminated those images where more than 10 percent of the pixels were white.

D.3.2 DEEPGLOBE

The Deepglobe dataset (Demir et al., 2018) comprises 1024x1024 pixel images with a similar zoom level to the official dataset . It contains many low-quality, rural images.

D.4 PaSSIV: Modelling, Experiments and Inferences

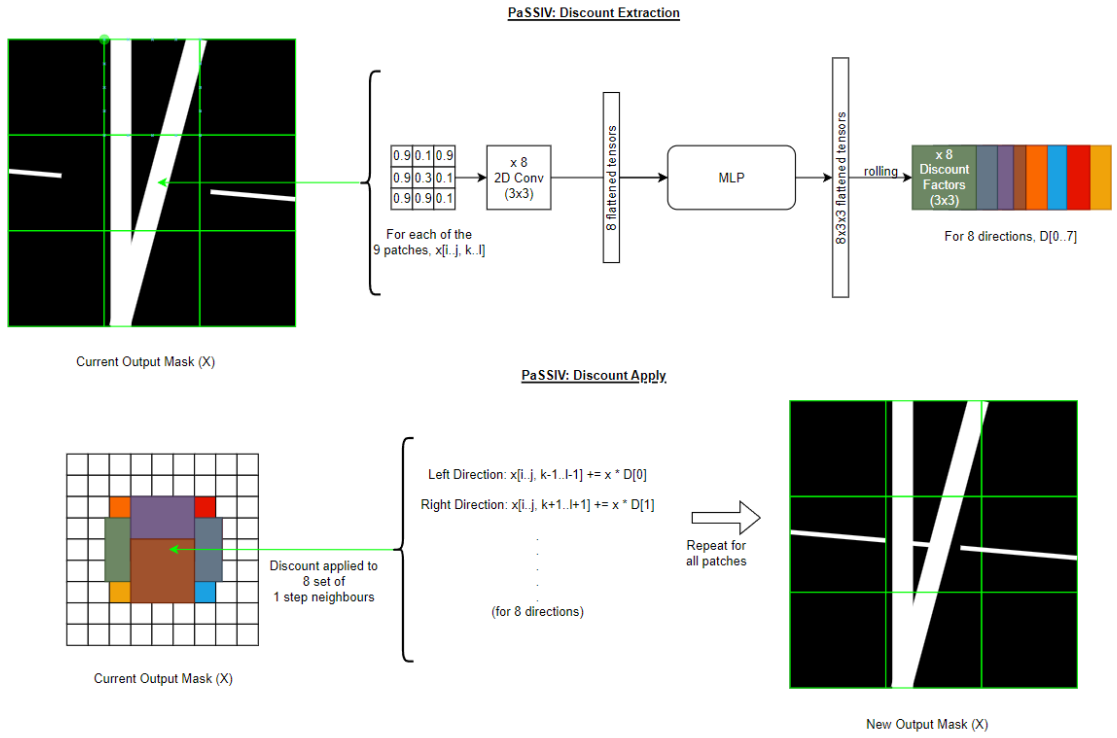


Figure 9: Operations in PaSSIV. PaSSIV takes the prediction masks of a transformer as an input, and divides it into several focus patches. The journey of one such focus patch is shown in 'Discount Extraction' process above. Every focus patch generates 8 directional discount patches. Every discount patch, D_i creates a weighted connection between the focus patch and its directional neighbour as shown in 'Discount Apply'. When the discount patches are pre-trained on road geometries, the PaSSIV operation results in a new prediction mask as shown.

D.4.1 How PaSSIV works

Transformers hold minimal inductive bias while attending to input features, and therefore perform well in abundance of data only. PaSSIV aims to improve the performance of transformer models in a low-data regime, such as when using only official data. The idea is to introduce a **trainable inductive-bias leveraging road geometry** to assist the training. Expanding on section 7.5, it works in two stages:

1. learns the road geometry (eg., directional continuity, branching) from the mask labels in pre-training phase
2. in the fine-tuning phase, augments the weaker transformer confidence on road pixels with adjusted confidence on **potential road pixels**, especially, representing thin, occluded, and non-trivial roads. The 2-step training regime is illustrated in Figure 10.

As illustrated in Figure 10, PaSSIV is a block that adds to a backbone model, a transformer here. PaSSIV divides every output of the backbone model, a prediction mask, into several patches called `focus_patches`. Each of these patches are

operated on by convolution and feed-forward neural layers to generate 8 (corresponding to 8 2-D directions) additional patches called Discount patches ($D_i, i \in [8]$) of the same size.

Every discount patch D_i that corresponding to a focus patch, acts as a weight (0-1) for the residual connection between the focus patch and the neighbouring pixels in that direction. `field_view` decides how far in that direction does the residual connection exist with the neighbouring pixels. Pre-training allows these Discount patches, D, to capture oblique road connections and continuity. Then in the fine-tuning phase, if a backbone model predicts only part of the road, the residual connections help activate other possible pixels that represents roads. In this way, PaSSIV augments the weaker road predictions of the backbone model.

D.4.2 Hyper-Parameters and Design constants for reproducibility

The PaSSIV block is motivated specifically for low data regime. This is because when there is scarcity of data, knowing the geometric properties of roads can help in making predictions. But when large amounts of data already exist, this intuition on road structures become marginally useful.

We use OneFormer as a baseline to test the performance of PaSSIV. We perform tests between (1) **OneFormer**: OneFormer trained on official dataset, and (2) **PaSSIV OneFormer**: OneFormer combined with PaSSIV, with the following hyper-parameters.

Quantity	**Values experimented with**
Learning Rate	10^{-4}
Batch Size	2 images (1250 16×16 patches)
Pre-Training Epochs	$100 - 200$ (until convergence)
Fine-Tuning Epochs	50
`focus_patch` size	48×48 pixels (9 16×16 patches), 80×80 pixels
`field_view` (measured in number of patches)	1, 3

Table 2: Experimental constants

D.4.3 Search for optimal threshold for 16×16 patch classification

While searching for an optimal threshold to classify every 16×16 patch as a road patch or not, we tracked validation scores across thresholds $[0.15, 0.20, 0.25, 0.3, 0.35]$ for both vanilla OneFormer and OneFormer with PaSSIV. The plots in Figure 11 and 12 clearly suggest optimal scores with $[0.20, 0.25, 0.3]$, with usually the best performance with 0.25. We also note that vanilla OneFormer is clearly able to show separation in performance between $[0.20, 0.25, 0.3]$ and $[0.15, 0.35]$ threshold values. Whereas, introduction of pixel-wise PaSSIV-based inference keeps this separation low. This suggests PaSSIV inference can lead to noisier fine-tuning. Based on the inferences here, all of our comparisons ahead use threshold amongst $[0.20, 0.25, 0.3]$.

D.4.4 Ablation Study

Figure 13 compares the trained PaSSIV OneFormer against OneFormer without the PaSSIV block. The F1 scores improves only marginally against the baseline until 12 epochs. But as epochs increase the performance is **statistically indistinguishable**. This suggests that more iterations on the dataset makes explicit geometry induction into the transformers redundant.

D.4.5 Effects of changing size of Focus Patches

A focus patch (`focus_patch`) defines the size of the patch a PaSSIV block operates on at once. It can neither be too small (due to pixelated golden label masks in chunks of 16×16), nor too big (fails to capture local geometry). Since every golden mask is uniform up until a pixel dimension of 16×16, to capture the local view of multiple golden patches, we chose `focus_patch` size as 48×48 pixels (3 16×16 pixels on each side), and compared it with `focus_patch` size of 80×80 pixels (5 16×16 pixels on each side). Figure 14 shows marginal improvements in **keeping the focus patch size small**.

D.4.6 Effects of changing length of field of view of neighbor patches

Discount patches generated from a single focus patch can be applied to nearest neighbors, next nearest neighbours, and so on. `field_view` controls this behaviour. A value of 1 means the Discount patch is applied to pixels present up to a stride of 1 full length of the focus patch. Figure 15 shows a negligible difference in performance, when we shift to farther neighbours. This is also intuitive because of diminishing weights applied to neighbours that are farther (see Section D.4.7). This also implies that with larger focus patch size, there are no improvements in increasing the `field_view`.

D.4.7 Effects of rate of decay of Discount Patches with distant neighbors

Currently, for a given direction, PaSSIV uses a Discount patch D_i to establish a weighted connection between the immediate neighbors and the current focus patch with a weight of value D_i. The next closest neighbors are connected by weight D_i^2, and so on. We decided to decay this relationship faster across distant neighbours, for example, by using D_i for the immediate neighbour, D_i^3 for the next nearest neighbor and so on. But this change implicitly translates to keeping a small field of view, so the results mimic the effects in Section D.4.6.

D.4.8 PaSSIV in Large Data regime

When a large dataset is available, we found that **OneFormer always outperformed PaSSIV-OneFormer** in terms of faster convergence. Given sufficient epochs until convergence, both achieved **similar performance**. We infer from this finding that in the presence of large data, geometric interpretation of roads **is better modelled by the attention blocks than PaSSIV**.

D.4.9 Some test set predictions: Qualitative and Quantitative test set analysis

Qualitative analysis of the PaSSIV-OneFormer outputs vs vanilla OneFormer outputs can be found in figure 18, 21. These are predictions from the official test set. We can see that the PaSSIV-based predictions are more consistent on continuous and practical road predictions compared to a vanilla Oneformer. Even then, we would like to highlight that the hallucinations highlighted in Section 8.1 aren't necessarily eliminated, and they can still appear.

Nonetheless, to evaluate generalization and possible distribution shifts between PaSSIV-OneFormer and vanilla OneFormer, we evaluate Kaggle scores on both. **PaSSIV-OneFormer beats vanilla OneFormer** with a score of 0.92475 against 0.9244. But since the analysis is on a low-data regime, the scores are still inferior to our Kaggle scores on large datasets. Nevertheless, this study provides a novel method to improve generalization scores over backbone image segmentation models by leveraging inductive bias of the domain, when the data available is limited.

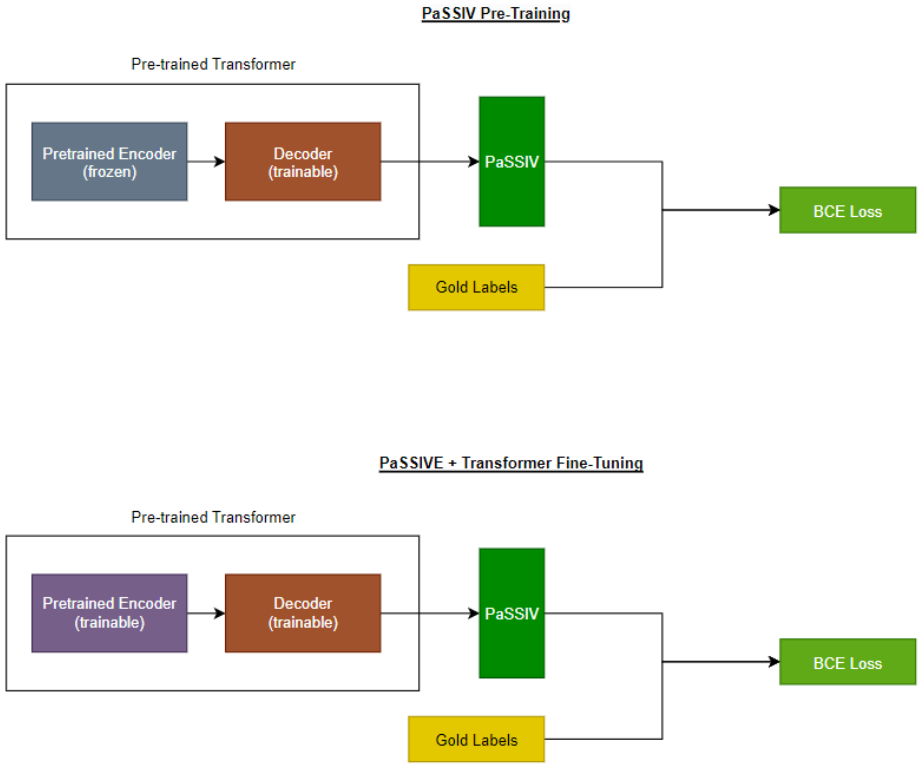


Figure 10: PaSSIV pre-training and fine-tuning setup against a backbone transformer. Pre-training involves freezing the pre-trained weights of the backbone model, and only training the decoder and PaSSIV block. Fine-tuning involves training all the trainable parameters end-to-end post pre-training. To keep the comparisons fair, we also trained vanilla OneFormer this way, without the PaSSIV block.

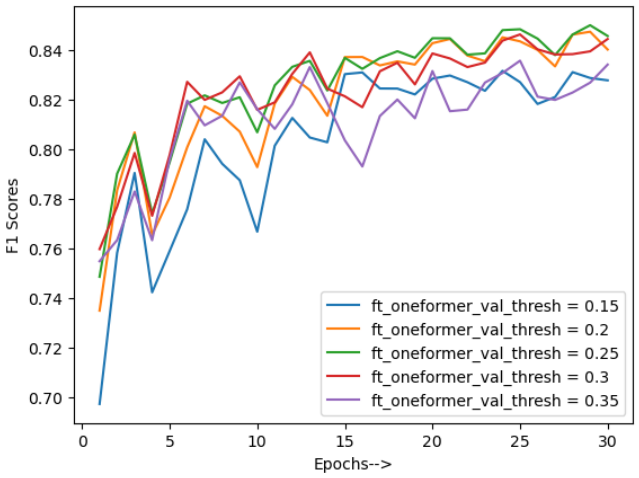


Figure 11: OneFormer: Validation performance across different thresholds

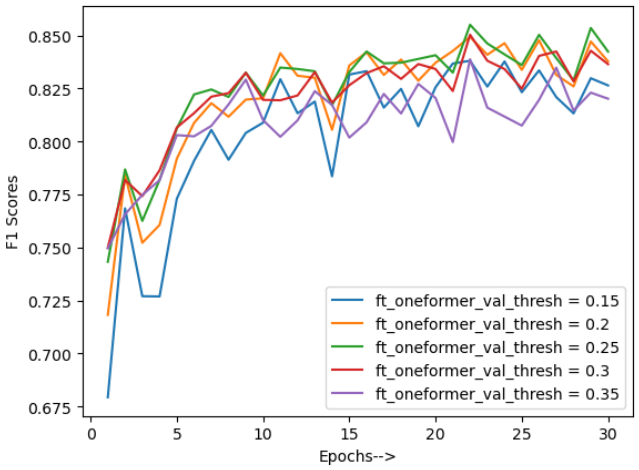


Figure 12: PaSSIV OneFormer: Validation performance across different threshholds

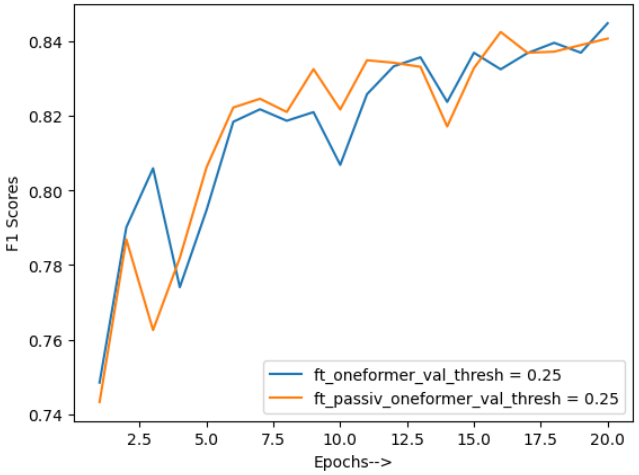


Figure 13: OneFormer vs OneFormer+PaSSIV in Low-Data Regime. PaSSIV improves the scores in the initial epochs ($\sim$ 12), but the added intuition on road-geometry becomes redundant with more epochs.

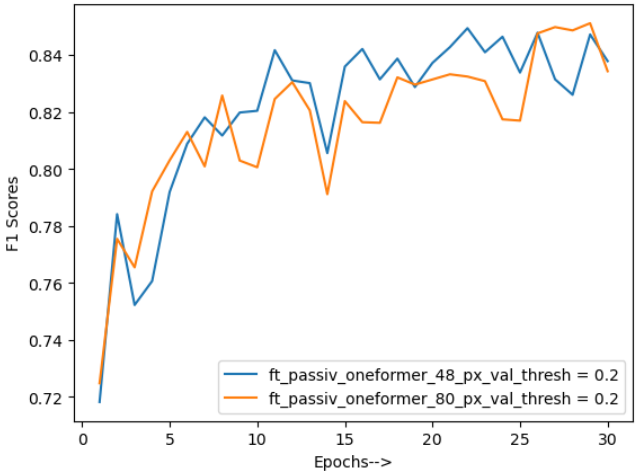


Figure 14: Effect of increasing size of focus patch

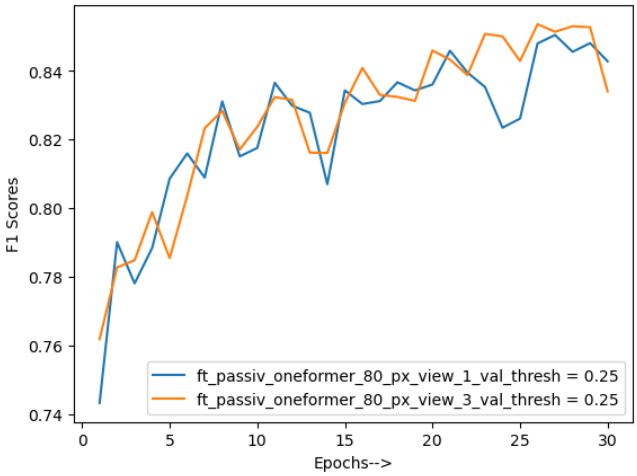
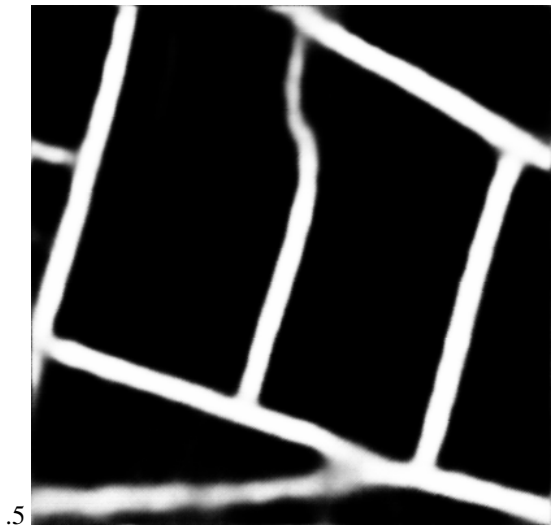
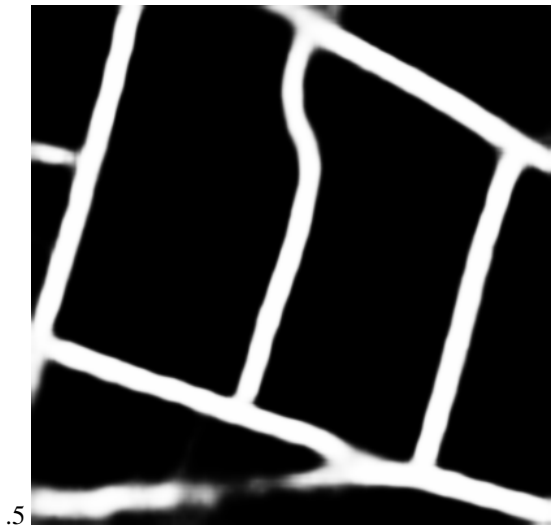


Figure 15: Effect of applying Discount patches to farther neighbours in the same direction.



.5

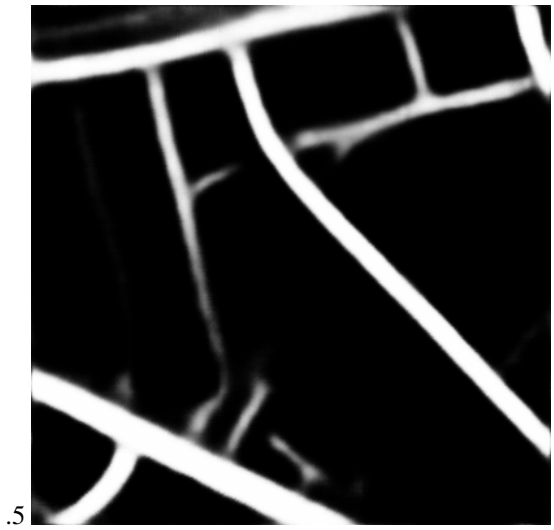
Figure 16: PaSSIV-OneFormer Prediction



.5

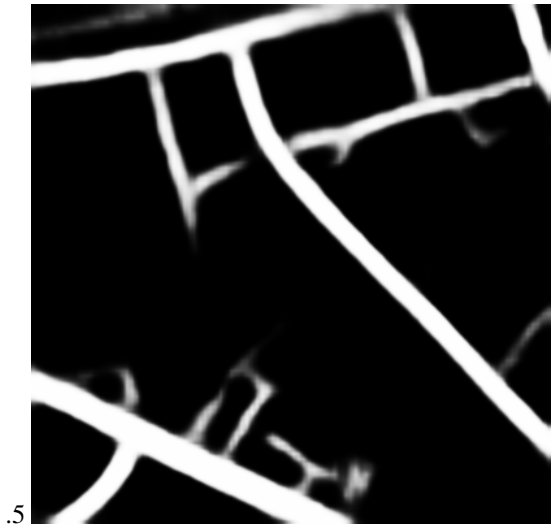
Figure 17: OneFormer Prediction

Figure 18: Example 1: Difference in PaSSIV and vanilla-based OneFormer predictions



.5

Figure 19: PaSSIV-OneFormer Prediction



.5

Figure 20: OneFormer Prediction

Figure 21: Example 2: Difference in PaSSIV and vanilla-based OneFormer predictions