

# Road segmentation from satellite imagery using U-net architecture

Iris Defromont, Fredrik Nguyem, Marc Gombau Rotllan  
*CS-433 Machine Learning, EPFL, Switzerland*

**Abstract**—Road extraction is a relevant topic in the field of CNN’s (Convolutional Neural Networks), which has a very useful range of applications. Some of these are automated driving systems, by helping perceive drivable regions, and remote sensing, for example. The main challenges in this area lie on continuity and robustness, while in general CNN’s are computationally expensive and time consuming. We propose our U-net model to perform road segmentation on a given satellite imagery dataset. By using data augmentation techniques, implementing our U-net model and enhancing it further; we have created a classifier that predicts drivable regions from images of GoogleMaps

## I. INTRODUCTION

In computer vision, semantic image segmentation is a crucial task that involves classifying each pixel of an image into a specific category. As urban areas expand rapidly and road networks grow to accommodate increasing populations, traditional road mapping methods have become insufficient due to the massive volume of aerial imagery data captured.

This project focuses on developing a machine learning model that can separate road from background in an image primarily from urban areas. The challenge lies in the complexity of satellite images, where roads may be obscured by trees, cars, or other structures, and can differ significantly depending on their material or location. Deep learning techniques have proven to be highly effective for this problem, as they can learn to identify abstract relationships in the data, reducing the need for manual feature engineering. Some of the most state of art methods to tackle the problem is with CNNs such as U-net and transformer-models such as SegFormer.

In, this project we will explore the U-net architecture, with some different variations of the Unet and data processing to see how they compare. However, we will mostly be referring to the final (best) model in the report, when discussing data augmentation, pipeline and results etc.

## II. MODELS AND METHODS

### A. Model

Unet is currently one of the most widely used models for image segmentation [1]. This model is a particular type of Convolutional Neural Network (CNN) which is divided into two main parts: encoding or contraction and decoding or expansion. Encoding reduces the size of the image block by block, while increasing its depth, to obtain information about the general context of the image. Each block generally uses two successive convolution stages and one pooling stage. Decoding restores the original size by increasing the image

size block by block. In general, each block successively uses an upsampling step using a 2D convolution, then a concatenation step between the images obtained after upsampling and the images obtained after convolution of the encoding part corresponding to the same dimension, and finally a convolution step. In particular, the concatenation step makes it possible to compare, for the same dimension, the encoding image and the image after decoding in order to recover the information lost during decoding.

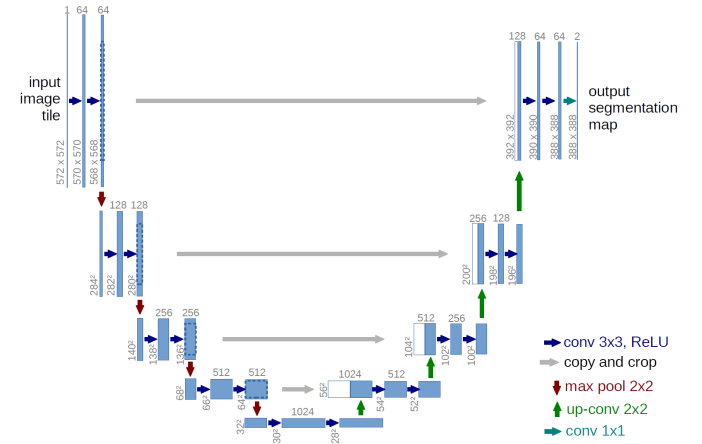


Fig. 1: U-net architecture

### B. Losses

It can be seen that the “background” and “foreground” (i.e. roads) classes are unbalanced in the images. Indeed, the background is generally predominant over the roads. To deal with this imbalance, we’ll be using two different types of loss capable of responding effectively to this problem, as well as a combination of the two: the dice loss and the binary focal loss.

- Dice loss - Measures the overlap between the predicted mask and the true mask as

$$Dice(A, B) = \frac{2|A \cap B|}{|A| + |B|}$$

Hence, encourages the predicted segmentation mask to overlap well with the true mask.

- Binary focal loss - Variation on the binary cross-entropy loss that addresses class imbalance well by down-weighting the easy examples such that their contribution

to the total loss is small. Hence, places more weight on hard-to-classify pixels

- Combination loss - Should leverage strength of both losses

### C. Metric

The metrics used is mainly f1-score, both for validation and for submission as it balances both precision and recall. Another metric that we considered is the Intersection over Union (IOU-score), as it just like dice-loss measures overlap between prediction and true mask which we want. However, as the ALCrowd verification measured using f1-score we opted for that as well.

$$F1 = \frac{2TruePositive}{2TruePositive + FalsePositive + FalseNegative}$$

### D. Data Processing

1) *Data Augmentation*: The training data consist of 100 satellite images and their corresponding ground truth, both 400x400 pixels. An example is shown in figure 2. To avoid over-fitting due to the small data size and to improve the robustness against varying settings in the images, we want to increasing our number of images to train on while also performing data augmentation on our new data. The augmentation consisted of series of random image manipulations with a certain probability ( $p$ ) such as:

- Random left or right rotations of max  $20^\circ$  with ( $p = 1$ )
- Mirroring images left to right or top to bottom ( $p = 0.2$ )
- Random zooming with 0.8 zoom rate with probability ( $p = 0.2$ )
- Random zooming with 0.8 zoom rate with probability ( $p = 0.5$ )
- Random left or right shears with max  $10^\circ$  ( $p = 0.5$ )
- Random distortions that can make roads appear 'wavy' ( $p = 0.5$ )

In total 1900 new images was created this way and added to the training set. To perform the data augmentation we made use of Python library 'Augmentor'<sup>1</sup> as it allows easily to load augmentation into memory without having to download the augmented images. This made it possible to run the model remote without having to upload a big amount of data each time we change augmentation. In hindsight it, might have been better to use dynamic augmentation in a dataset wrapper such as Albumentation<sup>2</sup>, as it does not take as much memory.

2) *Data preprocessing*: The model architecture for the final model requires a image size input divisible by 32, hence we resized the images to size 416x416. As the mask is gray scaled, we also have to make it black and white to represent 2 classes ('background' and 'road'). Finally, the images are processed and scaled using *efficientnetb0* backbone, which is one of the most efficient, optimized and accurate feature extractor<sup>3</sup>.

<sup>1</sup>Augmentor documentation

<sup>2</sup>Albumentation documentation

<sup>3</sup>Keras

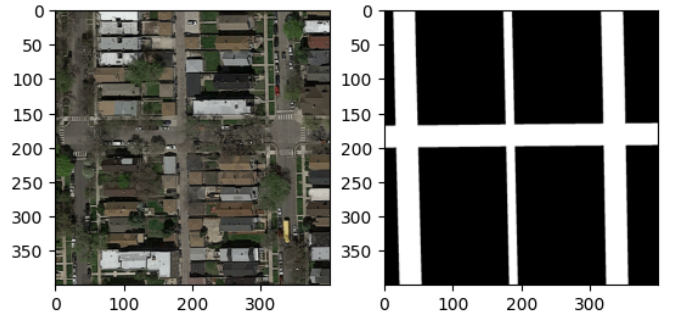


Fig. 2: Example image and groundtruth

### E. Training

For the training pipeline we used a Keras with Tensor Flow backend framework, which we drew as inspiration from examples given by authors of python library segmentation-models which can be found in the library's github<sup>4</sup>. The main components we used in the training was:

- *Adamw* optimizer with learning rate  $10^{-3}$  and weight-decay  $10^{-4}$
- *ReduceLROnPlateau* learning rate scheduler with factor of 0.5 and patience of 5 (i.e. it will drop learning rate by half if the validation loss Plateaus for 5 epochs)
- Batch size of 64 for training and 32 for validation was used, as it was the largest values without encountering memory issues.
- 80-20 training-validation split.
- Pre-trained weights from backbone *efficientnetb0* that has been trained on the dataset Imagenet.
- Early stopping with patience 10 with respect to validation loss.
- 50 max epochs without early stopping
- model checkpoints every epoch
- Combination loss, since as we will see in the next section, it will perform better than the other loss

As a result of long training time and time constraint, not all hyper-parameters were explored. In the next section we will see some of them.

Since the images from the data augmentation is fixed, we can refresh them by saving the model and rerun the whole model with the saved weights. In that way we can generate new images to train on and get better scores. This is kind of a workaround to not having a dynamic image augmentation.

## III. RESULTS

(Tables are below the summary, formatting issues) In table I we see the local f1-score and validation loss for different losses, and clearly the combination of both losses give the best result (highest f1-score). Hence, we will use it for further training. In table II we see that with (AdamW) or without the weight decay (Adam) they perform similarly. Similarly, in the same table we see that the model performs the same

<sup>4</sup>Segmentation model github

with  $10^{-3}$  and  $10^{-2}$ , which could be a result of the learning scheduler. Lower learning rate were not tried since it slows down the convergence of the loss. In table III and IV, we can see comparison between some different approach to the Unet model which we will discuss further in the next section.

#### IV. DISCUSSION

In this project, we developed and evaluated three different road segmentation models based on the Unet architecture, using different machine learning libraries, different data augmentation techniques and different loss functions. We compared the results obtained for each model on the basis of F1 score, accuracy and visual quality of predicted images.

1) *Model 1: Pytorch-based Unet with basic data augmentation:* Model 1 was mainly inspired by the exercises carried out during the semester, where a baseline self-built Unet architecture along with extra batch normalization blocks was implemented using PyTorch. We used cross-validation to compare training and validation losses and used the Adam optimizer with a learning rate of  $10^{-5}$  and a weight decay of  $10^{-8}$ . Data augmentation was carried out manually using `v2.transforms`, and in the end we generated 6400 images which we used to improve the training. This model produced an F1 score of 0.449 and an accuracy score of 0.763. However, as Table III shows, the visual results were far from good. Predicted images were heavily pixelated and road boundaries were not accurately detected. This suggests that Model 1 is unable to segment the image accurately and cannot distinguish road boundaries from other image elements.

2) *Model 2: Tensorflow-based Unet with ImageDataGenerator data augmentation:* For Model 2, we opted for TensorFlow, given its advantages for image processing. This model used ImageDataGenerator from keras [2] [3] for data augmentation, with transformations such as rotation, shift, shear and zoom. We used callbacks [4] such as ModelCheckpoint, EarlyStopping and ReduceLROnPlateau to avoid overfitting and optimize the training process. The model achieved a better F1 score of 0.479 and an accuracy of 0.713 compared to Model 1. Visually, the results were better than those of Model 1, with less pixelation and more accurate road boundaries. However, problems remained, as shown in Table III, notably with regard to vehicles visible in the road (for example, in image 9) and confusion between roads and railroads (for example, in image 43). Despite these improvements, the model still struggles to handle certain borderline cases, such as the distinction between roads and other structures, indicating a limitation in its ability to generalize.

3) *Model 3: Unet with Dice Loss and Binary Loss with Augmentor data augmentation:* The final model, used a combination of dice loss and binary focal loss, aimed at improving performance on unbalanced classes. The model consists on an Unet model from python library segmentation-models with pre-trained weights. We used the Augmentor library [5] for data augmentation and the Adam optimizer with ReduceLROnPlateau callback for learning rate adjustments. This model performed best, with an F1 score of 0.890 and

an accuracy of 0.937. The predicted images were significantly more accurate, with clear distinctions between roads and other features. As shown in Table III, the segmentation was not pixelated, vehicles were not detected on the roads, and the model successfully differentiated between roads and railroads. However, a few minor errors remained, such as slightly curved roads or minimal errors in detecting roads where there are none. These errors, although present, were much less frequent and less noticeable than in previous models. For this reason, we reserved this model for the final rendering and spent the rest of the time trying to improve it by finding the optimum parameters for learning. As seen in Table IV, the choice of loss function has a significant impact on both the f1 score and the validation loss at the end of the training process. Among the various loss functions tested, the Dice loss yielded the highest f1 score, indicating better overall segmentation performance. However, despite the improvement in the f1 score, the validation loss remained relatively high, suggesting that the model was not generalizing as effectively during the validation phase. So, to optimize the model, we chose to use focal loss to obtain a high f1 score and a low validation loss. And as seen in Table II, the highest f1 score, while maintaining a low validation loss, was achieved when using an AdamW optimizer with a learning rate of  $10^{-3}$ .

#### V. ETHICAL RISK

The ethical risk identified as most critical in the project was the risk of unfairness. The images used for training and testing in the code appear to come primarily from satellite images of developed countries. As a result, if the code is used with images that are not satellite-based or that feature different types of roads (e.g., unpaved roads, dirt roads, curved roads like roundabouts, or roads of different colors), the predictions obtained are likely to be inaccurate. This could make it more difficult for users in certain countries, particularly developing ones, to apply the code effectively.

This risk of unequal usage is difficult to address within the scope of the project. To reduce this bias, a more diverse set of images should have been used, including images from various countries and non-satellite sources. However, this would have posed challenges, as these images would need to be accompanied by their corresponding predictions (ground truth images), which would need to match the format required by the code. Nevertheless, efforts were made to mitigate this risk as much as possible through data augmentation. Various image transformations were applied, such as altering the color of the images, modifying the shape of the roads to make them less straight, and adjusting the brightness of the images.

#### VI. SUMMARY

In conclusion, the Unet architecture seems to be perform very well with a highest f1-score of 0.890, but requires some work such as pretrained model weights as well as comprehensive data augmentation and preprocessing. There

are many things that can still be improved for further studies, such as better hyper parameter selection, further loss function studies and speed optimization of the model.

loss type	f1-score (local)	val loss
Dice	0.936	0.0819
Focal	0.888	0.158
Combinator	0.963	0.0833

TABLE I: f1-score and validation loss for different loss types

optimizer	f1-score (local)	val loss
AdamW $lr = 10^{-3}$	0.96459	0.085906
AdamW $lr = 10^{-2}$	0.95795	0.085906
Adam $lr = 10^{-3}$	0.96459	0.16276

TABLE II: f1-score and validation loss for different optimizer and learning rate

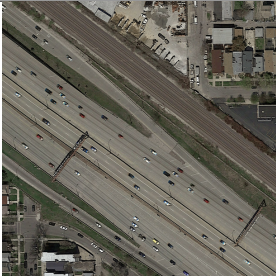

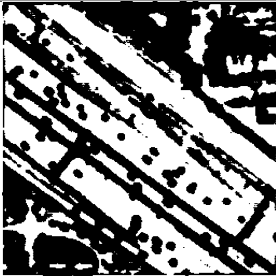



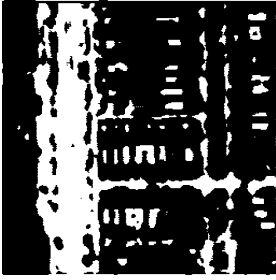
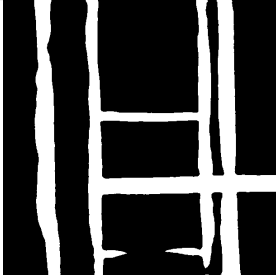
Images	Test image	Model 1	Model 2	Final Model
Test image 9				
Test image 43				

TABLE III: Comparison of the prediction obtained with the different models for some images

Models	ID in AICrowd	f1 score	accuracy score
Model1	277181	0.449	0.763
Model2	277980	0.479	0.713
Final Model	278443	0.887	0.935

TABLE IV: Comparison of the scores obtained for each models

# DIGITAL ETHICS CANVAS

## CONTEXT

Machine learning code for road segmentation

## SOLUTION

## BENEFITS

Help determine where roads are on a satellite image, which could be used to create a GPS, for example, or for urban planning studies

## WELFARE

### RISK

- Can the solution be used in harmful ways, in particular with regards to vulnerable populations?
- What kind of impacts can errors from the solution have?
- What type of protection does the solution have against attacks or misuse?

In its current state, the code does not pose any immediate danger to anyone. However, if our code were used as part of a larger system, such as a road detection code for a GPS or an autonomous vehicle navigation system, an error in our code could have much more significant consequences.



### MITIGATION

To prevent misuse of the code, it should warn the user to verify the predictions obtained before using them. This could be added to the README.



## FAIRNESS

### RISK

- How accessible is the solution?
- What kinds of biases may affect the results?
- Can the outcomes of the solution be different for different users or groups?
- Could the solution contribute to discrimination against people or groups?

The images used for training and testing all appear to come from satellite images of developed countries. This means that if someone uses our code with images that are not satellite images or images that show different types of roads (for example: non-concrete roads, dirt roads, non-straight roads such as roundabouts, roads of other colours) then the predictions obtained will surely be wrong. Some countries, particularly developing countries, will find it more difficult to use our code.



### MITIGATION

This risk of unequal usage is difficult to resolve within the context of the project. To reduce this bias, different images should have been used instead of the ones provided: images of roads from different countries and images that do not come from satellites. This would have been difficult because these images would need to be accompanied by their predictions, which would have to match the format used in our code. However, we tried to reduce this risk by augmenting our data with transformations that allowed us to change, for example, the color of the image and thus the roads, as well as altering the shape of the roads, making them less straight, and adjusting the brightness of the images.



## AUTONOMY

### RISK

- Can users understand how the solution works and what its limits are?
- Are users able to make choices (e.g. consent, settings) in their use of the solution and how?
- How does the solution affect user autonomy and agency?

Python code, and particularly machine learning code, can sometimes be difficult to understand for beginners in Python. Furthermore, the code uses many libraries and requires modifying certain values to be used outside the scope of the course, which can sometimes prove to be challenging.



### MITIGATION

By adding a README to our project that lists all the necessary installations for using the code (such as the libraries and the recommended graphics card), an explanation of the different files and how to use them, an explanation of the image format to be used, and examples of results obtained with the code. Additionally, the code should be well-commented so that any user with basic Python knowledge can read and understand the code and know where to change which constants to obtain the desired predictions.



## PRIVACY

### RISK

- What data does the solution collect
- Is it collecting personal or sensitive data
- Who has access to the data?
- How is the data protected?
- Could the solution disclose / be used to disclose private information?

Normally, only the user of the code has access to the images they use and the predictions they generate. Therefore, there should be no issues regarding data protection risks.



### MITIGATION

No issues regarding data protection risks.



## SUSTAINABILITY

### RISK

- What is the carbon footprint of the solution?
- What types of resources does it consume (e.g. water) -and produce (e.g. waste)?
- What type of human labor is involved?

From a sustainability perspective, the training part of the code requires a powerful computer or servers to process large datasets, which consumes a lot of energy. Servers, in general, have a very large carbon footprint because they require energy not only to operate but also to ensure proper cooling.



### MITIGATION

There aren't many solutions to this problem. To reduce the energy consumption of the code, we could improve its performance by using more powerful and specialized libraries to reduce execution time and memory usage. A basic example is using NumPy arrays for matrix calculations instead of relying on too many for loops.



## REFERENCES

- [1] A. I. Aramendia. The u-net : A complete guide. [Online]. Available: <https://medium.com/@alejandro.itoaramendia/decoding-the-u-net-a-complete-guide-810b1c6d56d8>
- [2] R. M. K. Most used TensorFlow keras callbacks: Explanation and implementation. [Online]. Available: <https://medium.com/@mkk.rakesh/most-used-tensorflow-keras-callbacks-explanation-and-implementation-a932c81fc4a6>
- [3] A complete guide to data augmentation. [Online]. Available: <https://www.datacamp.com/tutorial/complete-guide-data-augmentation>
- [4] K. Team. Keras documentation: Model training APIs. [Online]. Available: [https://keras.io/api/models/model\\_training\\_apis/](https://keras.io/api/models/model_training_apis/)
- [5] “Augmentor — Augmentor 0.2.12 documentation.” [Online]. Available: <https://augmentor.readthedocs.io/en/stable/>