Report of Machine Learning Project 2: Road Segmentation

Hanqi Lu (326798), Zhiyan Liao (395790), N'Zian Cédric Koffi (345346)
Neuro-X Section, EPFL, Lausanne, Switzerland

***Abstract*—Road segmentation consists in automatically classifying, pixelwise, aerial landscape images, with applications in navigation, urban planification, or geographical analysis. In this report, we explore several approaches and explain our findings. We end up choosing a tuned UNET as our final approach. The achieved performance is 0.887 and 94 % as F1 score and accuracy. We also report an ethical concern, related to the dangers of a false positive in road detection.**

I. Introduction

The main challenge of road segmentation is the need for precise spatial localization and delimitation of the recognized features. This means that the output has the same dimension as the input (except for the channels). While conventional Convolutional Neural Networks (CNNs) are highly effective at recognizing features and are well-suited for tasks such as classification, where the output is typically a category label, they have limitations in preserving the spatial information. At each layer, as the feature-related information increases, the spatial information progressively lost. Therefore, the final output has a lower dimension than the input image, making it suitable for classification but less effective for segmentation tasks.

Here, we explore two well-known CNN models to tackle this issue: UNET [1] and Hypercolumn [2]. They have been widely-used for image segmentation, especially road segmentation tasks. To achieve the best performance, we also experiment with the preprocessing step, the model choice, its architecture and its hyperparameters.

II. UNET

The UNET consists of two parts. The first half acts as an encoding network that trades spatial information for feature-related information. Each stage is a double convolution followed by a max pooling that reduces the image dimension and enables the recognition of features. At each stage, the output is preserved in skip connections that will contribute to the spatial information reconstitution. The second half stage starts with an upsampling operation followed by concatenation with the skip connection output. Then a double convolution combines the two to localize the features. After the decoding half, the output contains the segmentation. The UNET gets its name from the decrease, then increase in image spatial dimension.

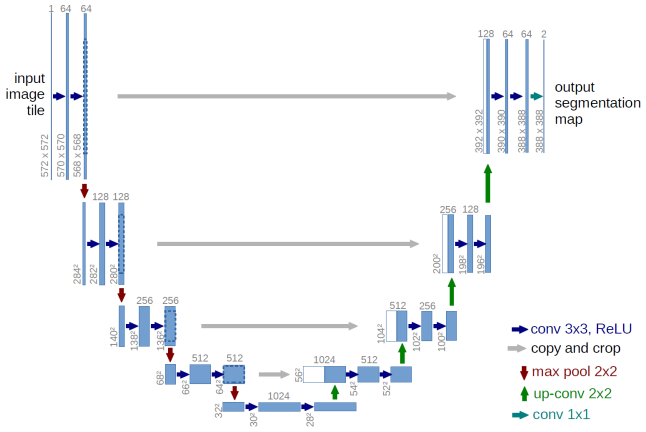

Figure 1. UNET architecture, with its encoding and decoding parts [1]

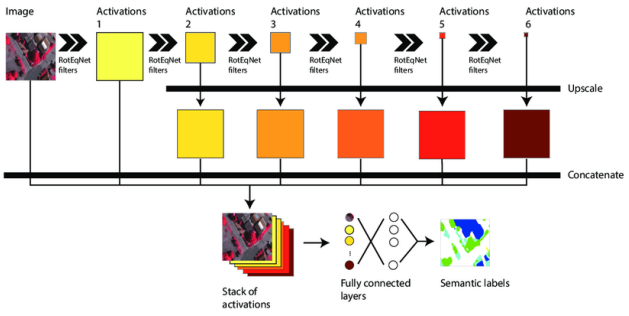

Figure 2. Hypercolumn architecture [3]

III. Hypercolumn

The Hypercolumn originates from the idea that the top layer sensitive to category-level semantic information is not the optimal representation of segmentation, while only the intermediate layers keep the information critical for accurate measurements. As shown in Figure 2, Hypercolumn perform downsampling through convolutions, and keep the intermediate output at the same time. Later these intermediate output are upsampled to the original scale and stacked into a large tensor (so called hypercolumn) to input a fully-connected layers and give pixel-wise classification.

IV. Experiments

A. UNET with small patches

We first started with preprocessing step involving cutting our image in small 16 x 16 patches. We assigned a label

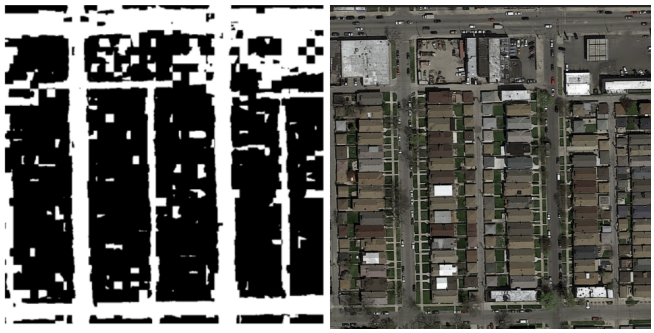

Figure 3. Poor spatial context exploitation

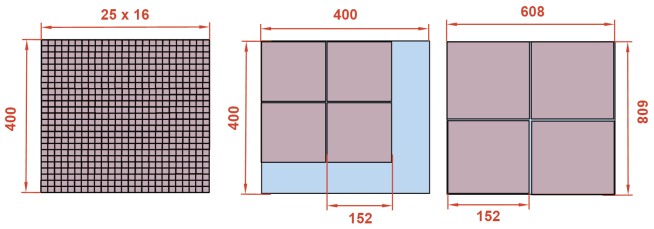

Figure 4. Image division into patches

based on a threshold on the proportion of bright pixel: above 25 % are road patches, and below are background patches. We then used this class to balance our dataset. Moreover, the UNET was trained to perform road segmentation on these patches independently, then we stitched these patches together to form the final prediction. This yielded an acceptable F1 score and accuracy: 0.717 and 71.37 %. But as we can see in Figure 3, the algorithm doesn't exploit the spatial context enough: enclosed road spaces obviously can't be.

B. UNET with larger patches and random rotation

We wanted to use larger patches, but the problem was that the sizes of the train and test sets were different: 400 x 400 and 608 x 608. The greatest possible patch size was the greatest common divider of 400 and 608: 16. We determined that only the test set would need an integer number of patches. We thus used 152 x 152 patches. This led to a data loss: 400 x 400 images, as shown in Figure 4. The problem could be solved by creating overlapping patches, but we decided that the data loss was tolerable. As a result, the algorithm was much better at exploiting spatial context.

But the algorithm was bad at recognizing inclined roads. We solved this problem through data augmentation: we added randomly rotated pictures (angle between -45° and +45°) to the dataset to train the model to segment these. These two modifications combined yielded far better results: $F1 = 0.797$ and $accuracy = 88.8\%$ on aircrowd, as shown Figure 5.

Finally, we took this one step further and stopped creating patches completely. The model was trained with full images. We done so in an attempt to exploit spatial context even more, and to reduce the data loss created by non-integer patches. This yielded slightly worse results: $F1 = 0.789$ and $accuracy = 89\%$ on aircrowd, meaning that patches are still of some importance for leaning. We could also notice that the model is robust to image dimension changes, since the performance was not significantly different on the test set with larger images: $F1 = 0.732$ on local validation set (400 x 400) vs $F1 = 0.789$ on the test set (608 x 608).

Figure 5. Better rotation recognition and spatial context exploitation (Test image 11)

C. Trials with a deeper network architecture

While the results were better than the first 16 x 16 patches version, the prediction still had defects. Mostly, some road segments were not recognized by the algorithm: there were many false negatives, but not as much false positives. This is visible on Figure 5 where many road segments are not identified. We hypothesized that if localization was the problem, then the road segments would get delocalized, and we would see more false positives. On the other hand, if road identification (feature extraction) was not powerful enough, then we would see more false negatives, which is what we observed. Since the encoding part is in charge of feature recognition, while the decoding part is in charge of feature localization, we decided to make the encoding part deeper. Given that both half work in tandem, we also made the decoding part deeper for symmetry considerations.

As shown Figure 6 new stage architecture, the encoding stages now perform four convolutions instead of two. To avoid spatial information loss, a skip connection was added between the second convolution output and the stage output.

Unfortunately, this yielded lower results than that of the shallow UNET, in addition to make training even more computationally intensive. After seven hours of training (about 100 epochs), a plateau was reached and the performances were: $F1 = 0.709$ and $accuracy = 74.33\%$ on a local validation set. The output was not graphically different (Figure 7). The reason is unclear; a complete explaination would require more experimentation, which was difficult due to the training duration. We do not think this is due to overfitting, as the loss on the validation set and train set were compared at each epoch. We did not observe an increase on

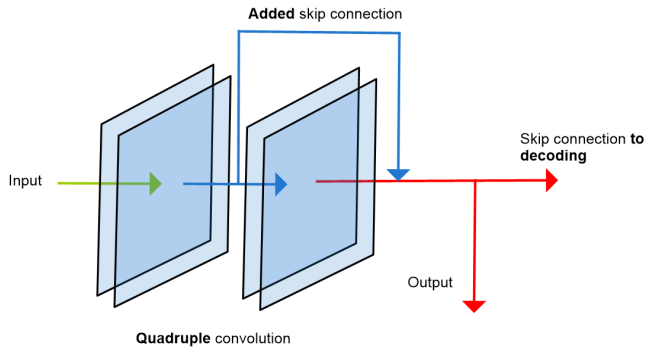

Figure 6. Quadruple convolution stage with added (blue) skip connection

Figure 7. Quadruple convolution graphic output (Test image 11)

the validation set while the train set loss kept decreasing. Instead, both losses plateaued. This suggests that the model could not learn effectively. The learning rate may be too high, but this is unlikely as a *Reduce on Plateau* scheduler combined with gradient clipping was used, and the learning rate $\gamma = 0.001$ was not too high. The leading explanation is vanishing gradient, which may happen when the network architecture is made deeper: gradients become negligeable as they get backpropagated through the layers. This is mitigated by skip connections, batch normalization, and the use of non-saturating activation functions such as ReLU, which were both used, but this is still a good explanation given the loss behavior: the loss barely diminished across epochs.

D. Dropout

This led us to abandon the deeper UNET to focus on optimizing its classic version. We added dropout layers to avoid overfitting, and this provoqued a leap in performance: $F1 = 0.887$ and $accuracy = 94\%$. This is the model we chose in the end. The performances of the different approaches are summarized in this table.

Approaches	F1 score	Accuracy
16 x 16 patches UNET	0.717	71.37 %
152 x 152 patches UNET	0.797	88.8 %
Full image UNET	0.789	89 %
Quadruple Conv UNET	0.709	74.33 %
UNET with dropout	0.887	94 %
Hypercolumn	0.56	N/A

Table I
RESULTS OF THE DIFFERENT APPROACHES

V. FINAL MODEL

A. Preprocessing

The dataset is divided into three sets: 55 images for training, 35 for validation, and 10 for testing. Given the relatively small size of the training set, we applied data augmentation to increase the number of training samples. The augmentation consists of geometric transformations such as horizontal flips, vertical flips, and rotations, and non-geometric transformations like color jitter, grayscale, and Gaussian blur.

When applying geometric transformations, it is essential to apply the same transformation to both the image and its corresponding mask to maintain their alignment. In total, we generated 100 augmented images, adding them to the original 55 training images, resulting in a total of 155 training images. Consequently, the validation/training ratio is 35/155, which is approximately 22%.

B. The model

We used a U-Net model following this architecture: Encoder (Downsampling Path):

Block 1: Input: RGB image with 3 channels. Two convolutional layers with 64 filters each, followed by batch normalization, ReLU activation, and optional dropout (0.1 probability). Max pooling with a kernel size of 2 reduces the spatial dimensions by half.

Block 2: Input: Output from the previous pooling layer. Two convolutional layers with 128 filters each, similar to the first block. Max pooling with a kernel size of 2.

Block 3: Input: Output from the previous pooling layer. Two convolutional layers with 256 filters each. Max pooling with a kernel size of 2.

Block 4: Input: Output from the previous pooling layer. Two convolutional layers with 512 filters each. Max pooling with a kernel size of 2. Bottleneck:

Two convolutional layers with 1024 filters each. Batch normalization and ReLU activation are used after each convolution. Dropout with a rate of 0.4 (to prevent overfitting).

Decoder (Upsampling Path):

Block 4: Input: Bottleneck output. Transpose convolution upsamples the feature map to the spatial size of the encoder's fourth block. Skip connection: The upsampled feature map is concatenated with the output from the encoder's fourth block. Two convolutional layers with 512 filters each. Block 3:

Input: Output from the previous decoder block. Transpose convolution upsamples the feature map to the spatial size of the encoder's third block. Skip connection: The upsampled

feature map is concatenated with the output from the encoder's third block. Two convolutional layers with 256 filters each.

Block 2:

Input: Output from the previous decoder block. Transpose convolution upsamples the feature map to the spatial size of the encoder's second block. Skip connection: The upsampled feature map is concatenated with the output from the encoder's second block. Two convolutional layers with 128 filters each. Block 1:

Input: Output from the previous decoder block. Transpose convolution upsamples the feature map to the spatial size of the encoder's first block. Skip connection: The upsampled feature map is concatenated with the output from the encoder's first block. Two convolutional layers with 64 filters each.

Final Output:

A single convolutional layer with a kernel size of 1 reduces the depth of the feature map to 1. The output represents a segmentation map, where each pixel corresponds to the binary segmentation.

C. Optimization

As we are doing binary classification, we decide to use Binary Cross-Entropy Loss with Logits (BCEWithLogitsLoss) as loss criterion, and Adam as optimizer. We also tried SGD with momentum, another commonly used optimiser, but it showed lower performance than Adam. We also added a scheduler to reduce the learning rate by 0.7 every 25 epochs. This provides finer adjustments and avoids oscillations when the training plateaus

D. Comparison with other approaches

As we can see in Figures 8 and 9, the UNET with dropout performs better in terms of loss and F1 score, and learns faster than the other approaches (for clarity, the UNET with 152x152 patches was not plotted separately due to its high similarity with the full image UNET).

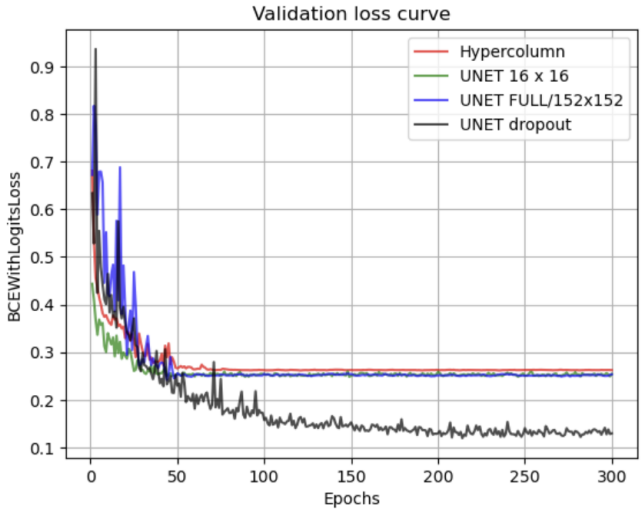

Figure 8. Validation loss curves for different models

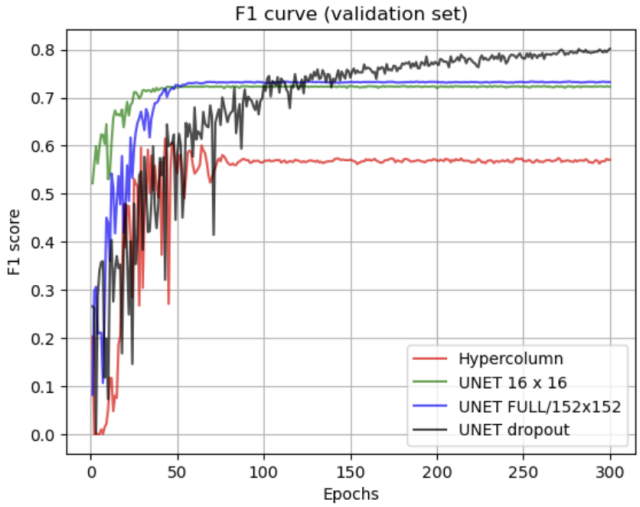

Figure 9. F1 curves for different models

VI. Ethical concern

The ethical concern we find is about the risk related to false positive or false negative depending on the application. We assume the segmentation mask is used with very little modification (enclosed road spaces are removed). In the case of navigation, mistaking background for a road can have severe consequences. There were several accidents, some fatal, caused by Google Maps indicating an impracticable road to drivers. In July 2024 [4], a driver found himself stranded on a Utah mountain after following a shortcut indicated by the software. He was rescued 3 hours later. In 2023 [5], several drivers found themselves stranded in the desert after being mislead by Google Maps. On the other hand, false negatives can also have serious consequences. For example, an ambulance could fail to reach a victim on time because a faster route is ignored by the road segmentation. There are also specific cases, such as geographical analysis after a natural disaster, where the goal is precisely to find practicable roads, and where finding them is critical. Thus, we think that depending on the application, there should be more emphasis on precision or recall, which an evaluation based solely on F1 score may overlook. We can tune it by shifting, after training, the threshold in the sigmoid function (usually 0.5). We can increase it up until the precision is high enough, or we can lower it up until recall is high enough for the application. Or we can use a custom loss function that penalizes more one or the other aspect. In any case, there will be a trad-off between the two.

REFERENCES

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015. [Online]. Available: https://arxiv.org/abs/1505.04597

[2] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Hypercolumns for object segmentation and fine-grained localization," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 447–456.

[3] D. Marcos, M. Volpi, B. Kellenberger, and D. Tuia, "Land cover mapping at very high resolution with rotation equivariant cnns: Towards small yet accurate models," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 145, pp. 96–107, 2018, deep Learning RS Data. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271618300261

[4] D. Jennings, "Driver following "shortcut" recommended by google ends up stranded on mountain for 3 hours," Jul. 2024. [Online]. Available: https://people.com/driver-following-shortcut-recommended-by-google-stranded-mountain-3-hours-8675533?utm_source=chatgpt.com

[5] M. Jha, "When google maps directions sent these drivers to a dangerous road," *The Hindustan Times*, Dec. 2023. [Online]. Available: https://tech.hindustantimes.com/tech/news/when-google-maps-directions-sent-these-drivers-to-a-dangerous-road-71701400684043.html?utm_source=chatgpt.com