Machine Learning CS-433: Road Segmentation

Matekalo Nicolas, Pena–Albert Awen, Faivre Bastien
Department of Computer Science, EPFL, Switzerland

***Abstract*—This is the second project of the Machine Learning course (CS-433) at EPFL. Among the 3 topic options available, we decided to work on one of the two predefined challenges: Road Segmentation. Several methods seen in the course and structures found by reading previous papers on image segmentation are used and tested to extract roads from satellite images. The objective is to obtain the best F1 score possible. Our final model, which is an improved UNet with a VGGNet architecture and residual connections, got an F1 score of 0.915.**

I. Introduction

Image segmentation is the process of dividing an image into multiple segments or regions, each of which corresponds to a different object or background. The goal of image segmentation is to partition an image into meaningful parts that correspond to different objects or regions in the image. In our case, the goal is to extract all the roads from satellite images. Therefore, our model should be able to classify each pixel in an input image as either a road or a background element, and then group them into batches of size 16x16 for submission to the AICrowd platform. To this end, we have used different deep learning techniques, some seen in class and others found by reading papers and tuned them to be the most efficient for our case study.

In order to get the best results possible, we tried to optimize each part of the algorithm. This includes the data, the model, and the prediction. First, we will dig into the data set.

II. Dataset Improvement

One of the key factors in the success of a deep learning model is the quality of the training dataset. The more diverse and representative the dataset is, the better the model can generalize to new situations and make accurate predictions. This is especially important when the dataset is small, which is our case since we are working with only 100 images for both the train and test sets.

In such a situation, it is even more important to carefully curate and preprocess the data to maximize its usefulness. This might involve augmenting the dataset with additional images, balancing the distribution of classes, or removing any irrelevant or noisy data.

We tried 4 methods to improve the dataset and accuracy:

1) **Add new images to improve the general accuracy**
2) **Resize existing images**
3) **Add new images to reduce error rate**
4) **Use image transformations**

A. Add new images to improve general accuracy

We explained before that adding new images to the dataset has a good chance to improve the accuracy of the model.

We attempted to improve the model's accuracy by adding new images of labeled roads from an online dataset from the University of Toronto [1]. However, we encountered challenges that made it difficult to improve the model.

The satellite images in the dataset were taken from different angles, making it hard for the model to learn robust representations of roads. Additionally, the dataset included images of twisting rural roads, which did not align well with our dataset of straight urban roads.

Fig. 1. Overlaying of a satellite image from the Massachusetts dataset with its associated groundtruth in white

The most significant problem we encountered was that the Massachusetts roads dataset was not labeled in the same way as our own dataset, leading to a large number of errors during training that made it difficult for the model to learn from the new data.

Due to the accuracy loss, we experienced when using the images from the new dataset, we decided not to use them in the future.

B. Resize existing images

We modified the dataset by resizing the input images to fit the model's required dimensions. Specifically, the images in the dataset were 400x400 pixels, while the model required inputs of 608x608 pixels.

We chose a new size of 304x304 pixels, which is a multiple of 608 that allowed us to partition the 400x400 images into smaller images of 304x304 pixels each.

To use these resized images as inputs to the model, we further partitioned the 608x608 images into 4 smaller 304x304 images. This allowed us to feed these smaller images into the model as separate inputs, which could then be combined to produce the final output prediction.

However, due to the redundancy introduced by this process, which caused the model to overfit the training data and resulted in poor generalization to new images, we decided to keep the initial size of the images in the dataset.

Hence, the model is trained to process images that are 400x400 pixels wide and to produce output predictions of the same size. If you would like to learn more about how we managed the 608x608 testing images, please refer to section IV for more information.

C. Add new images to reduce error rate

One issue we identified was that the model was struggling to accurately classify parking lots, frequently misclassifying them as roads. To address this issue, we decided to add more images of parking lots to the dataset.

Fig. 2. One of the added parking images with its associated groundtruth image on the right

The added benefit of this approach was that we did not have to manually label these images, as the model could learn from them without explicit supervision. This is the key difference from the first method that made it more feasible to improve the model in this way.

Unfortunately, the addition of these images did not result in a significant improvement in the model's performance as we had hoped. Please refer to the section V for a discussion of potential ways to improve this approach.

D. Use image transformations

Image transformations modify the appearance of the images in some way, such as by flipping, scaling, or rotating them. By exposing the model to a wider range of variations during training, these transformations can help it to learn more robust and generalizable features.

Because the dataset is small, the model is only trained on images of roads with a specific orientation, lighting, and environment.

Some of the image transformations we applied to the dataset include horizontal flip, vertical flip, transpose, and rotate (or more complex ones like Gaussian noise or channel dropout). We applied these transformations randomly at each epoch, in order to expose the model to a wide range of variations during training. This helped the model to learn more resilient and transferable features.

The use of random transformations has had a major impact on the accuracy of the model, particularly due to the limited size of the dataset. Please refer to section V to compare the performance of a model with and without image transformations. As a result, we have chosen to continue using this technique to further improve the model's performance.

E. Final dataset

The final dataset we ended up using consists of the original 100 images and the images of parking lots that we have added in section II-C. We did not use the methods described in sections II-A and II-B. The model is designed to handle input images of 400x400 pixels and to produce output predictions of the same size. During training, we apply the image transformations described in section II-D on the fly at each epoch.

III. Model and optimization

When it comes to image segmentation, several classic methods can be used such as thresholding or region-based. Another classic method that is the one we used is the deep learning approach. This method typically involves training a convolutional neural network (CNN) on the data set. The baseline we used for our model is a type of CNN typically used for image segmentation called "UNet".

A. UNet

UNet (cf figure 3) consists of a contracting path, the encoder, and an expanding path, the decoder. These two are connected by skip connections. Their main benefit is that they can help to improve the gradient flow through the network and reduce the vanishing gradient problem. On the one hand, the contracting path consists of a series of convolutional and pooling layers that reduce the spatial dimensions of the input image, hence reducing the computational complexity as well. On the other hand, the expanding path consists of a series of upsampling and convolutional layers that increase the spatial dimensions of the output back to the original size of the input image.

To implement our UNet baseline [3], we based our work on a paper on the UNet model from Olaf Ronneberger, Philipp Fischer, and Thomas Brox [2]. Here is the picture representing the architecture of a UNet (cf figure 3):

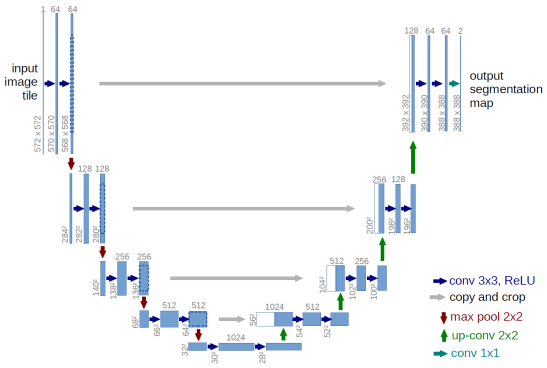

Fig. 3. Architecture of a UNet

B. Dropout

The first optimization we applied to our baseline is adding dropout to it. This regularization technique reduces overfitting in deep neural networks like ours. This way, our model is more likely to make accurate predictions on new, unseen data. It also reduces the complexity of the model since it reduces the number of parameters it needs to learn. This was important to us because training the model takes a long time due to its depth and the number of images and transformations in the dataset. We set a starting probability of $p = 0.5$ for the dropouts, and we applied them on the bottom part of the UNet called the *bottleneck*.

C. Encoder/Decoder tuning

The next changes we applied concern the architecture of the layers. We tuned the encoder (left part of the figure) and the decoder (right part of the figure). The original UNet is composed of an encoder and a decoder of 4 double convolutional layers and a bottom part that is a simple double convolutional block. To get more precision, our idea was to add more layers for both the encoder and decoder parts.

1) VGGNet: Another classic CNN architecture is the VGGNet developed by the Visual Geometry Group (VGG) at the University of Oxford [4]. This architecture consists of a series of convolutional and pooling layers, with a small number of filters in each layer. It is characterized by a deep network structure with up to 19 layers. The configuration with 19 layers called *VGG19* is the one we implemented for the encoder and decoder parts. Basically, it consists of 2 blocks of double convolution and 3 blocks of quadruple convolution separated by pooling operations (max-pooling in our case).

2) Residual connections: As our architecture is now deeper, the vanishing gradient problem becomes an important concern. In order to deal with this problem, we added residual connections to each block. Residual connections are basically shortcutted connections that bypass one layer. These shortcut connections allow the gradients to bypass the layers in the block and connect their input directly to their output by summing them. If the output has a different size than the input, the latter is either downsampled or upsampled. Such connections help to improve the gradient flow through the network and reduce the vanishing gradient problem. This way, our model got improved convergence and generalizes better to new data.

D. Training

To train our final model on the dataset, we used the binary cross-entropy loss with logits. This loss function is commonly used in classification tasks where the goal is to predict a binary outcome. This is exactly what our model is supposed to do, i.e. predict if a pixel is a part of the road ('1') or part of the background ('0'). The BCE with logits loss function is defined as follows:

$$loss = -(y * log(p) + (1 - y) * log(1 - p))$$

where y is the true label and p is the predicted probability of the positive class (the probability of a pixel being a road in our case).

To update the weights of our neural network during the training process, we used the AdamW optimizer. For the learning rate and the weight decay, we tried several values to find the best configuration for our model. It turned out that this best configuration is reached for a $1e^{-4}$ the learning rate and a $1e^{-5}$ the weight decay is performing best for our model. The latter is used to prevent overfitting. It is based on the idea of adding a penalty term to the loss function that encourages the weights of the network to be small.

To adjust the learning rate during the training, we used a learning rate scheduler. The type of scheduler we have chosen is the cosine one. Such a scheduler reduces the learning rate using a cosine function over the course of training. As the cosine function is a smooth and continuous function, the learning rate can decrease more gradually and avoid sudden drops or spikes. The learning rate scheduler is helpful as it reduces the need for manual hyperparameter tuning. Indeed, the learning rate is automatically adjusted over the course of training.

Please refer to the section V to evaluate the F1-score of the different models.

IV. Prediction phase

As we mentioned in section II-E, the model produces output predictions that are 400x400 pixels wide. However, as we mentioned in section II-B, the testing images are 608x608 pixels wide.

To solve the problem of the testing images being larger than the size that the model was designed to handle, we divided the testing images into 4 smaller 400x400 pixels wide images at the corners. Prior to inputting these smaller images into the model individually, we applied various transformations to them (e.g. rotations, flips). Hence, when wanting to predict a specific image, the model is actually predicting all transformed image parts that are obtained by splitting the original one. The results of these predictions are then gathered to obtain a single final prediction. This allowed the model to make predictions on a set of modified images that each emphasized different features. It helped the model to improve its accuracy by averaging a wider range of variations.

Then, the workflow for the prediction phase is as follows:

1) The original 608x608 image is split into four 400x400 images that fit the input size of our model.
2) Several transformations are applied to these four images, which gives us about thirty images (7 images per part of the input).
3) The model predicts each of these 400x400 images.
4) If needed, apply de-transformation on the resulting images (e.g. unflip, rotate back).
5) Collect the images in batches of 8 describing the same part of the entry. Average the result for each batch as follows: for a pixel to be classified as a road, at least half

of the images from the same batch has to classify as a road, otherwise it is classified as part of the background.

6) At this stage, we are left with the four parts of the 608x608 image. Group them back to get the correct size. For the pixels that overlap on several parts, average their values just like for the batch of 7 images.

V. Results and Discussion

Our final model along with several pre and post-processing optimizations got an F1-score of 0.915 on the test predictions, according to AICrowd. This result was obtained by applying different post-processing techniques to the predictions made by our trained model. Please find below a comparison of the F1-score of the different models we discussed in section III:

UNet	0.888
VGGResUNet without image transformations	0.903
VGGUNet	0.906
VGGResUnet	0.910
VGGResUNet+	**0.915**
VGGResUNet+ with parking images	**0.915**

The VGGResUNet+ model refers to the VGGResUNet model with the prediction process described in IV.

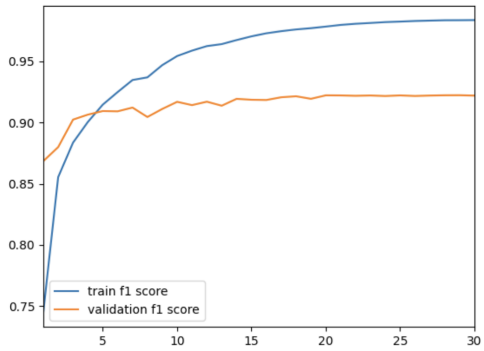

Fig. 4. Evolution of the F1 score of our final model over epochs

To begin discussing potential ameliorations, please see the following example of a misclassified prediction using our model:

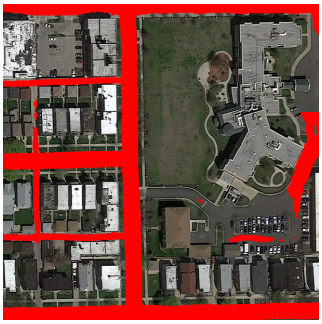

Fig. 5. Overlaying of a test image with its corresponding predicted groundtruth (in red) using our model

There is still room for improvement, as several ideas were not implemented throughout this project:

1) *Adding images with complex roads:* One idea would be to add images with complex roads (e.g. twisted roads, highways, traffic circles). The complexity of this task is to produce accurate groundtruths. One should carefully classify each pixel. However, this would certainly improve the accuracy of our model as the richer and more complex is the dataset, the better the model will generalize to new or complex data.
2) *Dilatation:* In the context of convolutional neural networks, dilatation is about dilating or spacing out the filter used for convolution operations. It helps to increase the receptive field of the network without increasing the number of parameters. This can help to reduce overfitting and simplify the model, while also allowing it to extract features from the input at various scales and resolutions for improved accuracy.
3) *Adding complex parking images:* Despite adding images of parking lots, the model still struggles to distinguish roads from parking lots when a road is connected directly to a parking lot. This is because the images we added do not include roads, so the mask is a simple black square. To improve the model's performance in identifying the separation between roads and parking lots when they are connected, we could add images of roads leading to parking lots and carefully label their groundtruth by hand.

VI. Summary

This project required us to implement deep-learning algorithms to extract roads from satellite images. To get the best F1-score possible, we focused on each part of the algorithm, including the data, the model, and the prediction. We augmented the data set by adding images on which the initial model was struggling and by transforming the images to get a richer dataset. We started from a UNet baseline and changed its architecture to get better precision. Finally, we applied some transformations on the test dataset and we averaged the predictions obtained for each image. All of this led us to get a result of 0.915, which is well-ranked on AICrowd. Ultimately, this project gave us the opportunity to strengthen our knowledge of deep learning and neural networks. It also gave us a taste of what research looks like in this domain.

References

[1] Volodymyr Mnih. "Machine Learning for Aerial Image Labeling". PhD thesis. University of Toronto, 2013.

[2] Philipp Fischer Olaf Ronneberger and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". PhD thesis. University of Freiburg, Germany, 2015. URL: https://arxiv.org/pdf/1505.04597.pdf.

[3] Aladdin Persson. *PyTorch Image Segmentation Tutorial with U-NET: everything from scratch baby*. 2021. URL: https://youtu.be/IHq1t7NxS8k.

[4] Karen Simonyan Andrew Zisserman. "Very deep convolutional netowrks for large-scale image recognition". PhD thesis. University of Oxford, England, 2015. URL: https://arxiv.org/pdf/1409.1556.pdf.