Machine Learning: Project 2
Road segmentation

Quentin Deschamps, Emilien Seiler, Louis Le Guillouzic
School of Computer and Communication Sciences EPF Lausanne, Switzerland

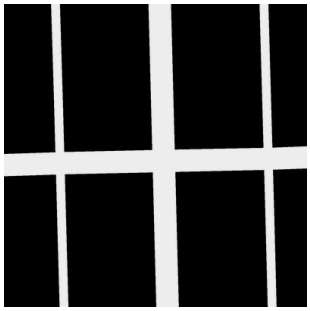
***Abstract*—This paper presents machine learning models and techniques performing satellite image segmentation. The goal is to classify each pixel of an image in road or what we call background, representing anything but not roads. Among multiple models, we choose the current most powerful one for this task, known as U-Net. This is a particular type of convolutional neural network which has already been adopted for biomedical image segmentation in the past. The final model consists in training such a neural network on a set of 1800 augmented satellite images along with ground truth road labels. Our method achieves a pixelwise F1 score of 90.1%.**

I. Introduction

Image segmentation is a sub-domain of computer vision and digital image processing which aims at grouping similar regions or segments of an image under their respective class labels. This task finds its way in prominent fields like Robotics, Medical Imaging, Autonomous Vehicles, and Intelligent Video Analytics. Apart from these applications, Image segmentation is also used by satellites on aerial imagery to segment roads.

In our case, we aim to do road segmentation on satellite images. It corresponds to a binary classification task on pixels, to know which of these correspond to roads. We could imagine that such a process is useful for use cases like automatic world roads indexing. For example, this information can be used to improve navigation tools.

(a) Satellite image 1



(b) Ground truth image 1

Fig. 1: Example of satellite image / ground truth pair

Our training dataset consists of 100 satellite images and ground truth labels pairs of size 400x400 pixels. The figure 1 shows an example. The goal is to train a classifier to segment roads using these images, i.e. assign a label (road=1, background=0) to each pixel, and predict these labels for 50 other satellite images of size 608x608.

Simple machine learning techniques are often not powerful enough for image segmentation as they cannot easily represent complex models without smart feature engineering. Additionally, learning from images is recognized as being a computationally heavy task, and fully connected neural networks do not scale well for such problems. That is why, we decide to use convolutional neural networks.

II. Methodology

In this part, we explain the method used to perform the task of road segmentation. This method combines machine learning techniques by the usage of convolutional neural networks, and image processing with data augmentation techniques and post-processing strategies.

A. Models

Today, convolutional neural networks (CNNs) are very successful for computer vision tasks such as segmentation. A caracteristic of our task is that the size of the input and the output must be variable. Indeed, the training images tensors have a shape of 400x400x3 (3 channels), whereas testing images tensors have a shape of 608x608x3. Moreover, we expect an output of 400x400x1 for the first case, and 608x608x1 for the second.

To handle this type of inputs and outputs, we can design an architecture with two major components. The first component is an *encoder*: it takes a variable-length sequence as the input and transforms it into a state with a fixed shape. The second component is a *decoder*: it maps the encoded state of a fixed shape to a variable-length sequence. This is called an *encoder-decoder* architecture, which is depicted in figure 2.

Fig. 2: Encoder-Decoder architecture

We select three convolutional neural networks following an encoder-decoder architecture, mainly used in the domain of image segmentation:

- U-Net
- SegNet
- Nested U-Net

1) U-Net: The U-Net [1] is the network we select as our final model. This is a CNN developed for biomedical image segmentation. It takes in input an image and outputs the label of each pixel.

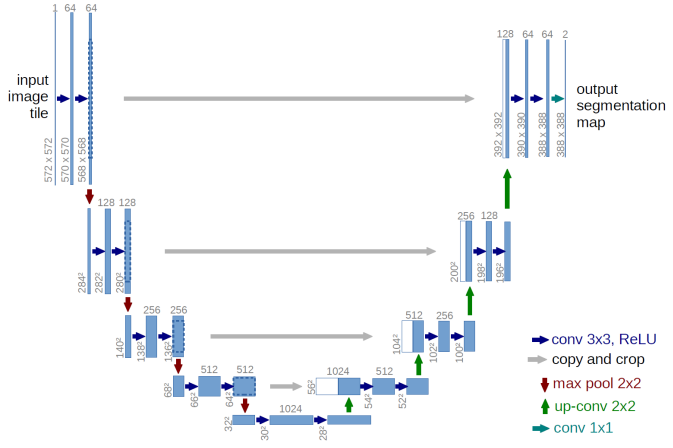


Fig. 3: U-Net architecture [1]

The architecture of the network is illustrated by the figure 3. It consists of a contracting path (left side) which corresponds to the encoder, and an expansive path (right side) corresponding to the decoder. The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step, we double the number of feature channels.

Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution ("up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution.

At the final layer, a 1x1 convolution is used to map each component feature vector to the desired number of classes. In total, the network has 23 convolutional layers.

2) SegNet: As the U-Net, the SegNet [2] is a CNN architecture used for semantic pixelwise segmentation. It consists of an encoder network of 13 convolutional layers, a corresponding decoder network of 13 convolutional layers followed by a pixel-wise classification layer (soft-max). The architecture is illustrated by the figure 4.

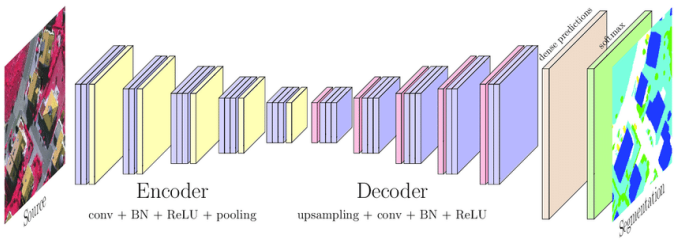


Fig. 4: SegNet architecture [2]

3) Nested U-Net: Finally, the Nested U-Net [3], or also called U-Net++, is a new, more powerful architecture for medical image segmentation. This architecture is illustrated by the figure 5. It consists of an encoder and decoder that are connected through a series of nested dense convolutional blocks. The main idea behind UNet++ is to bridge the semantic gap between the feature maps of the encoder and decoder prior to fusion.

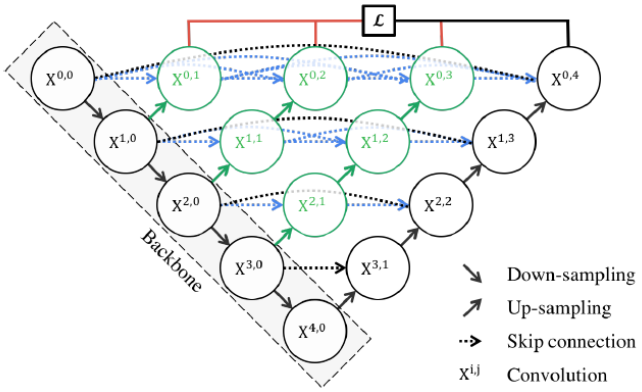


Fig. 5: Nested U-Net architecture [3]

B. Data augmentation

To train a deep learning model, it is important to have a consistent database to obtain relevant results. The database initially contains 100 images, which is not enough for a U-Net or a SegNet network. By applying identical transformations to the images and their respective mask, it is possible to increase the size of the training dataset.

Many data augmentation methods are possible, but not all of them are relevant to the given problem. An appropriate pre-processing must prevent our model from overfitting by reinforcing the corner cases, but must not confuse and bring bad data either. The techniques used are explained and justified below.

1) Cropping: First, by observing the images available to train the model, we can notice that most of the streets are parallel and perpendicular to each other, in addition to being of similar size. So, to obtain different road widths, we decide to crop the given images into the four corners and the central crop plus the horizontal flipped version of these. The size of the cropped images is 256x256 pixels. We recall that the size of the original images is 400x400 pixels.

2) Rotations: Then, to get different road orientations, we take the cropped center of rotated images, with seven different angles of rotation: 45°, 90°, 135°, 180°, 225°, 270°, 315°. The aim of this transformation is to make the model consistent on roads of all possible angles. The size of these images is still 256x256 pixels, to have the same size for all images.

3) Resizing: Finally, we keep the initial images in the augmented dataset. To have images of identical size to train our model, we resize the initial images to 256x256 pixels.

4) Summary: The table I summarizes the transformations. With the initial images, the ten crops, and the seven rotations, we multiply the size of our initial data by 18, and obtain a data set of 1800 images. By generating more data, we

avoid overfitting and produce a final model more robust to the different roads patterns.

	Cropping	Rotations	Resizing	Total
Number of images	100×10	100×7	100×1	1800

TABLE I: Data augmentation

C. Training

After choosing the architectures of the neural networks and augmenting the training set, we can train the networks. This step is computationally expensive and time consuming. In this part, we discuss about the different choices made for training.

1) Epochs: We fix a limit of 100 epochs for the training. Nevertheless, training our architectures on 100 epochs leads to overfitting. To avoid this phenomenon, we use a technique called "early stopping". It is a form of regularization technique consisting of stopping the learning when the loss starts to go up. Using this method, the training of a network takes from 40 up to 60 epochs.

2) Loss function: The loss function used to quantify the difference between predictions and target labels and back-propagate is the Sørensen–Dice coefficient. The Dice coefficient is a common metric for pixel segmentation. It is essentially a measure of overlap between two samples. This measure ranges from 0 to 1 where a Dice coefficient of 1 denotes perfect and complete overlap. The Dice coefficient was originally developed for binary data, and can be calculated as:

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|} \tag{1}$$

Where $|X \cap Y|$ represents the common elements between sets X and Y, and $|X|$ represents the number of elements in set $|X|$ (and likewise for set Y).

For the case of evaluating a Dice coefficient on predicted segmentation masks, we can approximate $|X \cap Y|$ as the element-wise multiplication between the prediction and target mask, and then sum the resulting matrix.

3) Optimizer: As a stochastic optimizer, we choose *Adam* [4]. This method is computationally efficient, has little memory requirements and is well suited for problems that are large in terms of data and/or parameters. We use a learning rate of 10^{-4}.

In addition, we use weight decay, or L_2 regularization, to regularize the weights of the neural network. We add to the loss function the following penalty:

$$\lambda \sum_i w_i^2 \tag{2}$$

Where the $\{w_i\}$ are the weights and λ is a small coefficient. For our case, we use $\lambda = 10^{-4}$.

4) Metrics: Additionally to the dice loss, we track two other metrics during the training phase:

- The **F1 score**, which is the harmonic mean of the precision and recall:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{3}$$

- The **Accuracy**, which is the proportion of correct predictions (both true positives and true negatives) among the total number of cases examined:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{4}$$

The figure 6 illustrates the tracking of the loss on the left, and the F1 score and the Accuracy on the right.

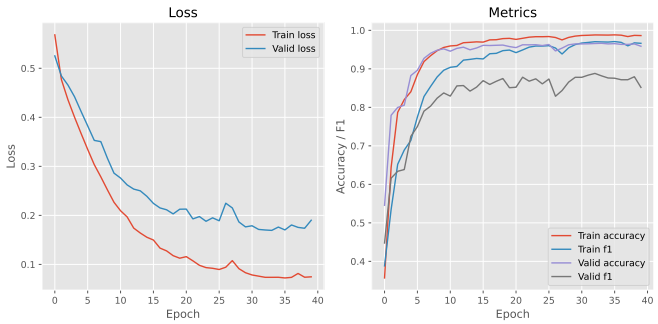


Fig. 6: Training and validation results for U-Net

For training, we split the dataset of 1800 images into two subsets: a training set (80%) and a validating set (20%). Our trainer saves the weights of the model when the F1 score of the validation if better. It allows to prevent overfitting.

5) Dropout: In addition to pre-processing, we try dropout. This technique consists in randomly drop units (along with their connections) from the neural network during training. This is an additional method to prevent overfitting. We use this technique only on U-Net: we add a dropout at the end of each block from figure 3 with probability $p = 0.2$. Nevertheless, in our case, it does not improve the final results. So, we can ignore this technique.

D. Post-processing

After looking at the predicted masks, it appears that they can be improved. We tried different creative post-processing methods in order to increase these predictions. We created a clustering method, consisting of separating the predicted road segments into clusters, and removing the smallest ones. Indeed, we supposed that if a cluster is too small and far away from other clusters, it is probably a mistake from the classifier.

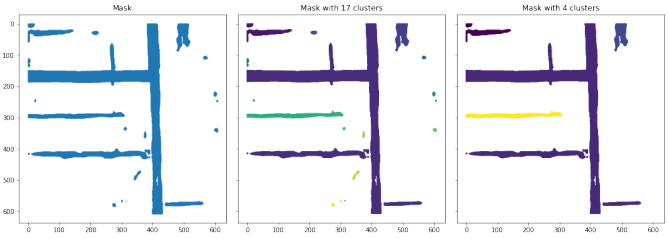


Fig. 7: Improvement of predictions using clustering

The figure 7 illustrates this method. Here, it is applied on the image 12 of the testing set.

- On the left, we have the prediction of the classifier. We see that there is some noise.
- On the middle, we detect the clusters using DBSCAN[1]. This method is well appropriated here since it can find arbitrarily-shaped clusters. The algorithm finds 17 clusters.
- On the right, we remove the clusters of less than 500 points. There are 4 clusters remaining.

So, this method allows to drop isolated pixels which cannot correspond to roads.

E. Implementation

The neural networks are implemented using the PyTorch [5] framework, associated with Torchvision. The training procedure is performed using a GPU on Google Colab, since it is too heavy for a CPU. Indeed, the training of U-Net takes about 4 hours on a GPU. You can find our full implementation on GitHub on the following link:

https://github.com/CS-433/ml-project-2-qelbis

F. Results

The table II shows the performances of the different models. The scores correspond to the output of submissions on *AIcrowd*. The U-Net architecture is the most efficient model.

Model	Data Augment.	Post-proc.	F1	Accuracy
U-Net	Yes	Yes	0.901	0.946
U-Net	Yes	No	0.900	0.945
Nested U-Net	Yes	No	0.896	0.943
SegNet	Yes	No	0.895	0.944
U-Net	No	No	0.853	0.922

TABLE II: Models scores

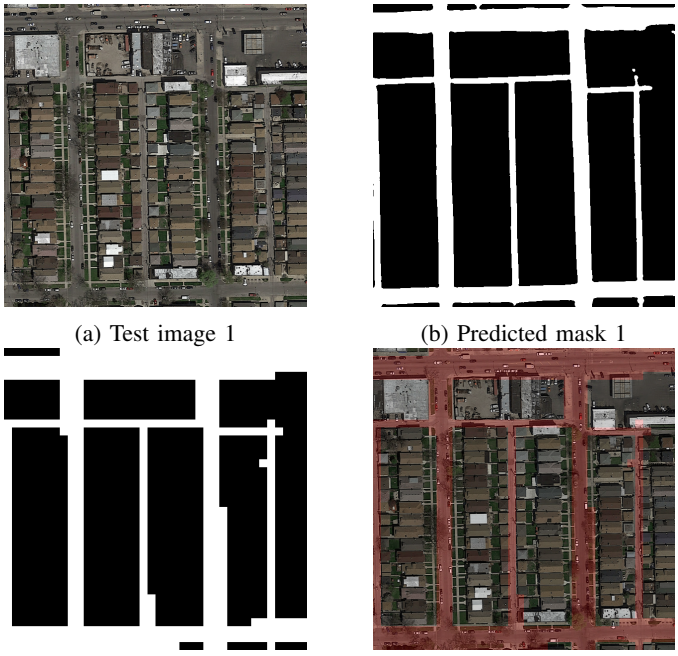
We see that data augmentation improves the performance of the model: we gain around 5 points for F1 score using U-Net with augmentation. The post-processing method is not very effective but it increases the F1 score and the accuracy of 0.1%. The figure 8 shows an example of a prediction using U-Net. We see that the mask fits well the roads of the image.

III. Conclusion

To conclude, U-Net is a strong neural network architecture for segmenting satellite images by extracting road pixels from background. Moreover, data augmentation brings to better results. Using several machine learning techniques to strengthen the learning process, our final model achieves a F1 score of 90.1%[2] on the testing data set.

[1]Density-based spatial clustering of applications with noise

[2]Submission link



(a) Test image 1 (b) Predicted mask 1

(c) Predicted mask patched 1 (d) Overlay 1

Fig. 8: Example of a prediction by U-Net

References

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

[2] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *CoRR*, vol. abs/1511.00561, 2015. [Online]. Available: http://arxiv.org/abs/1511.00561

[3] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "Unet++: A nested u-net architecture for medical image segmentation," *CoRR*, vol. abs/1807.10165, 2018. [Online]. Available: http://arxiv.org/abs/1807.10165

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/pdf/1412.6980

[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *CoRR*, vol. abs/1912.01703, 2019. [Online]. Available: http://arxiv.org/abs/1912.01703