The Road Segmentation Problem : Machine Learning Project 2

Bastien Golomer, Bastien Le Lan, Arthur Brousse
Machine Learning (CS-433), School of Computer Science, EPF Lausanne, Switzerland

***Abstract*—The problem solved in this paper consists in a semantic segmentation of roads on satellite images. We experimented with a double U-Net, accompanied with various innovative solutions, such as Squeeze & Excite blocks, the use of pretrained models and Atrous Spatial Pyramidal Pooling. After a bespoke data augmentation and some hyperparameter tuning, we obtain the following performances : F1-score of 0.844 and accuracy 0.916.**

I. Introduction

We are given a training dataset of a 100 colored images of size 400×400 and their corresponding masks (that is, the ground truths of road placement in these images). The goal is to correctly classify the pixels of 50 test images of size 608×608, by assigning labels 1 to pixels belonging to roads and labels 0 to any other pixel. This is commonly known as semantic segmentation. This paper presents a novel approach based on the work of [1], a DoubleU-Net. In the following, we describe this architecture as well as the different state-of-the-art optimisations and additional modules implemented to increase performances. Hyperparameters are discussed as well as several ways to improve the current architecture for better results.

II. Models and methods

A. U-Net

For semantic segmentation tasks, U-Net are a type of architecture, based on Convolutional Neural Networks, trained end-to-end for pixel-wise prediction. The U-Net architecture consists of two parts, namely, analysis path and synthesis path. In the analysis path, deep features are learned, whereas in the synthesis path, segmentation is performed based on the learned features. Additionally, U-Net uses skip connections operation. The skip connection allows propagating dense feature maps from the analysis path to the corresponding layers in the synthesis part. In this way, the spatial information is applied to the deeper layer, which produces a significantly more accurate output segmentation map. Thus, adding more layers to the U-Net will allow the network to learn more representative features leading to better output segmentation masks.

B. The Double U-Net Architecture

1) General Concept: A novel approach to improve the performance of U-Nets is called DoubleU-Net, which is a combination of two U-Net architectures stacked on top of each other, as seen in fig. 1. The first U-Net uses a pre-trained VGG-19 as the encoder (described further down), which has already learned features from ImageNet and can be transferred to another task easily. The squeeze-and-excite block is used in the encoder of NETWORK 1 and decoder blocks of NETWORK 1 and NETWORK 2. An element-wise multiplication is performed between the output of NETWORK 1 with the input of the same network, and acts as an input for the second modified U-Net that produces another mask (Output2). The whole concept is that we assume that the produced output feature map from NETWORK 1 can still be improved by fetching the input image and its corresponding mask again. NETWORK 2 is used to capture more semantic information efficiently.

This kind of novel architecture is based on the work of [1], which was initially applied to Lesion boundary segmentation datasets.

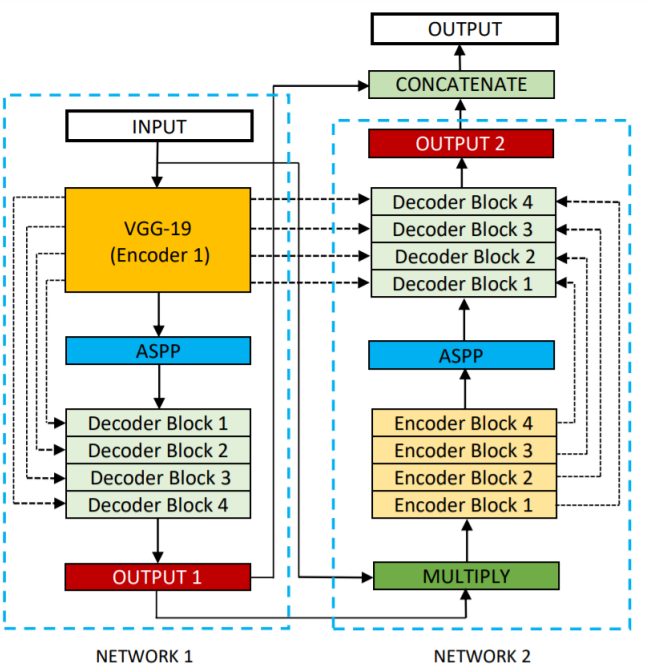


Fig. 1: Block diagram of the DoubleU-Net architecture. The arrows linking encoders and decoders represent the transfer of skip connections

2) Transfer learning: Pre-trained ImageNet models have significantly improved the performance of CNN architectures. They have become common practice as first approach for solving classification problems. Integrating them to new

architectures with related tasks generally allows sparing a serious amount of time building and training a model from scratch (already trained on massive datasets and parameter optimisation already done). This is method is called transfer learning, removing the need for a large set of labelled training data for every new model. As done in [1], we are using VGG-19 convolutional neural network in our architecture. The main reasons for using the VGG network are: (1) VGG-19 is a lightweight model as compared to other pre-trained models, (2) the architecture of VGG-19 is similar to U-Net, making it easy to concatenate with U-Net, and (3) it allows much deeper networks for producing better output segmentation mask.[2]

3) Encoder: The first encoder in the DoubleU-Net uses VGG-19, whereas the second encoder is built from scratch. Each encoder block in the encoder 2 performs two 3×3 convolution operation, each followed by a batch normalization. The batch normalization reduces the internal co-variant shift and regularizes the model. A Rectified Linear Unit (ReLU) activation function is used, which introduces non-linearity into the model. This is followed by a squeeze-and-excitation block (see after), which enhances the quality of the feature maps. After that, max-pooling is performed with a 2×2 window and stride 2 to reduce the spatial dimension of the feature maps.

4) Decoder: Each block in the decoders performs a 2×2 bi-linear up-sampling on the input feature, which doubles the dimension of the input feature maps. Now, we concatenate the appropriate skip connections (see after) feature maps from the encoder to the output feature maps. In the first decoder, we only use skip connection from the first encoder, but in the second decoder, we use skip connection from both the encoders, which maintains the spatial resolution and enhance the quality of the output feature maps. After concatenation, we again perform two 3×3 convolution operation, each of which is followed by batch normalization. After that, we use a squeeze and excitation block. At last, we apply a convolution layer with a sigmoid activation function, which is used to generate the mask.

C. Optimisation methods

1) Skip connections: Skip connections in deep architectures, as the name suggests, skip some layers in the neural network and feeds the output of one layer as the input to the next layer. By introducing skip connections in the encoder-decoder architecture, fine-grained details can be recovered in the prediction, which is suitable in our case. Even though there is no theoretical justification, symmetrical skip connections are generally effective in dense prediction tasks [3]. More generally, it is experimentally validated that these additional paths are often beneficial for the model convergence.[4]

2) Atrous Spatial Pyramid Pooling (ASPP): ASPP is a combination of atrous convolution and spatial pyramid pooling. It was created to capture the contextual information at multiple scales for a more accurate classification[5]. We employed the same ASPP module architecture as developed in DeepLabv3. Specifically, the ASPP module consists of (a) one 1×1 convolution and three parallel 3×3 convolutions with rates of 6, 12, and 18, respectively and (b) an image-level feature that is produced by global average pooling. The resulting features from all of the branches are bilinearly upsampled to the input size and then concatenated and passed through another 1×1 convolution, see fig. 3. ASPP is applied to the feature map produced by the encoder part, and the resulting feature map is fed into the decoder part.

3) Squeeze Excite (SE) Block: The squeeze and-excite block in the proposed network reduces the redundant information and passes the most relevant information [6]. The role this operation performs at different depths differs throughout the network. In earlier layers, it excites informative features independently from classes of the segmentation, strengthening the shared low-level representations. In later layers, the SE blocks become increasingly specialised, and respond to different inputs in a highly class-specific manner.

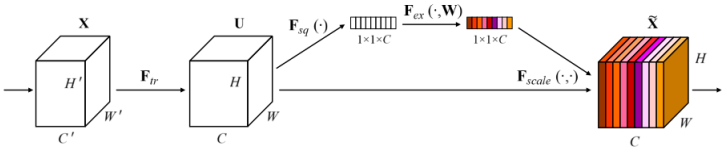


Fig. 2: Squeeze-and-Excitation block [6]. Inputs have C channels, Height H and width W.

Here are the steps pictured in fig. 2

Squeeze:
Global Average Pooling operation ($\mathbf{F}_{sq}$) to reduce the C×H×W image to C×1×1 to get a global statistic for each channel. Made simple, all we do is that we take the mean of each channel across H×W spatial dimensions and build an array called descriptor.

Excite:
The excitation operator maps the descriptor into a set of channel weights ($\mathbf{F}_{ex}$, a composition of ReLu and sigmoid) which is then applied to the original image ($\mathbf{F}_{scale}$). In this regard, SE blocks intrinsically introduce dynamics conditioned on the input, but generalised to the relationships in between the input's channels contrary to convolution filters, applied to channels independently.

4) Adam Optimizer: Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments [7]. This method was created to be computationally efficient, with little memory requirement and well suited for problems that are large in terms of data/parameters [8], so it was integrated in the architecture. Another optimizer named Nadam was also an option but was much slower and therefore discarded promptly.

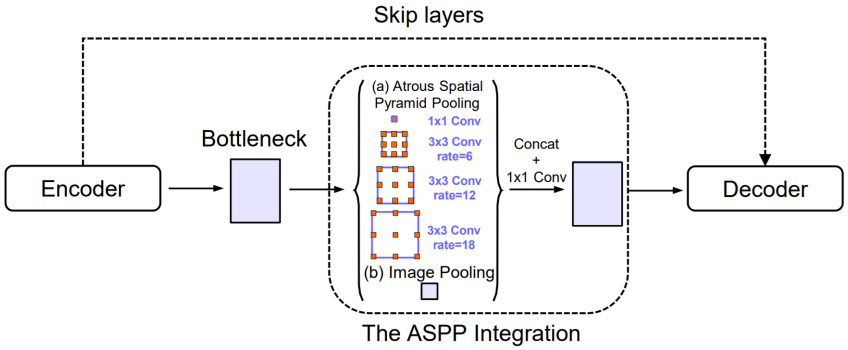


Fig. 3: Atrous spatial pyramid pooling (ASPP) integration in the Double U-Net [5]

D. Metrics

DoubleU-Net is evaluated on the basis of Sørensen dice coefficient (DSC), mean Intersection over Union (mIoU), Precision, and Recall. However, we compare and emphasize more on the evaluation metrics that were used in the challenge, in our case, the evaluation metrics for the road segmentation challenge is F1 score, a metric computed from and proportionally to Recall and Precision. In all of the datasets, we used 90% of dataset for training, 10% for validation. We chose dice loss and Adam optimizer as they performed slightly better, the batch size is set to 7 and the learning rate to $1 \cdot 10^{-4}$. The models are trained for 8 to 12 epochs, depending on how much GPU we managed to have (more on that later).

E. Data augmentation

The dataset comprises one hundred satellite images taken from Google Maps, of size 400×400. By doing a visual inspection of the images, we see that many of them are lookalikes, so they probably will not provide enough information to correctly segment the images from the test set. To correct this, we proceed to a serious augmentation of the data, by applying different transformations to our images. Common transformations applied to satellite images are color transformations (random changes in brightness, contrast and saturation) and spatial transformations (random flips, rotations). Here are the transformations we performed :

1) 90 *degrees rotations :* Each image is randomly rotated by a multiple of 90 degrees.
2) *Other rotations :*Additionally, we randomly rotate each image and paste it on a background made of a multiple of a 90 degrees rotation of the same image. All these rotations are implemented to make our network more robust to curves. Indeed, a visual inspection of the training dataset revealed that the images contained mainly straight roads.
3) 90 *degrees patches rotations :* patches representing a quarter of the image are randomly rotated by 90 degrees and stuck together. The idea behind this transformation is to make the network able to distinguish roads in photos that lack structure and geometry (dead ends, blocks of houses of various shapes, etc.)
4) *Translations :* The images are translated in their diagonal and vertically, the parts leaving the frame are added on the opposite side.
5) *Grey scale :* transform images to black and white nuances and apply vertically and horizontally. Such augmentation aims to make the network more sensitive and make the distinction between roads and buildings' roofs or other elements.
6) *Image grid :* Make a 3×3 or 4×4 grid using the basic image as unit element (lattice of the image). The goal with this augmentation is to produce more complex images with thinner roads and more developed networks, as the given training dataset displays simpler networks compared to the test one.

We also added several other augmentations that we later discarded, because they did not yield significant improvements in the performances, and increased the risk of overfitting. These discarded augmentations included lightness and saturation, rotating and zooming in, among others. There are fully implemented and available on the GitHub Folder of our code.

F. Hyper parameters selection

The batch size and number of epochs were chosen to be the maximum possible with the RAM at disposal (12 GB with Google Colab). Hence throughout our trainings, we always used a batch size of 7 and a number of epochs of 10. While this might seem low, we reached acceptable levels of precision and F1-score with these numbers, which can be explained by the architecture of our network. The learning rate was chosen to be **1e-4** after making simulations with several tangible values, as shown in table I.

The loss we chose was the Dice Loss, computed as :

$$DL = 1 - \frac{2\sum_i y_i f(x_i)}{\sum_i (y_i^2 + f(x_i)^2)}$$

Indeed, it was proven a decent loss for semantic segmentation problems with the advantage of not depending on any parameters to tune. [9]

	Wide grid search				
LR	1e-5	2.5e-4	5e-4	7.5e-4	1e-3
val prec	0.7998	0.7808	0.7659	0.7867	0.7811
val rec	0.7914	0.7167	0.6926	0.6587	0.6106
val loss	0.5187	0.3344	0.2956	0.2938	0.3155

	GS around 1e-5		GS around 1e-4	
LR	0.9e-5	1.1e-5	0.9e-4	1.1 e-4
val prec	0.6418	0.6447	0.6437	0.5112
val rec	0.6954	0.7223	0.6236	0.4360
val loss	0.5850	0.5695	0.5176	0.6214

TABLE I: **learning rate tuning:** Selection of appropriate learning rate(LR) with a search over a linear range from 1e-5 to 1e-3, then around 1e-4 and 1e-5.The models were trained using 800 images, 6 epochs per parameter and a batch size of 7.

G. Post-processing

A simple post processing step was added to the pipeline. While producing the submission file, an algorithm detects pixels which are predicted as background, even though their neighbourhood is mostly white (holes in the roads). These pixels are turned white, under the exact condition that at least 5 batches of pixels in the neighbourhood are white colored. Pixels in angles of road connections are excluded from this treatment to keep sharpness.

III. Results

After applying all the described steps, we obtained a fully trained model. The best score we obtained on the AIcrowd platform was a F1-score of **0.844** and a precision of **0.916**. This result was obtained by training our model for 10 epochs. We tried to run it for longer, but were limited by the Google Colab environment.

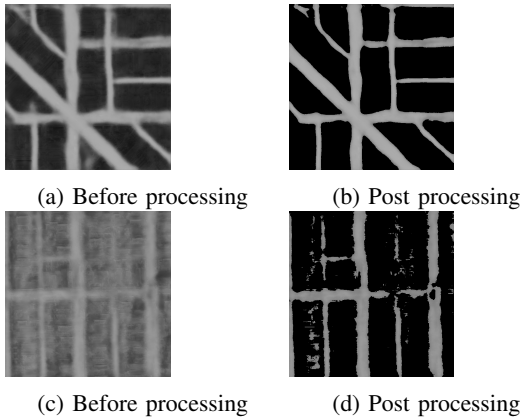
IV. Discussion

A. On the model

The limited computation power that we had at our disposal was a major problem for us throughout the realisation of this project. For instance, we've always been limited to a maximum batch size of 7, as for a bigger size we would invariably get an *Out Of Memory* error from Colab. Similarly, we were always disconnected from Colab after running between 10 to 15 epochs. Even though we managed to train our model to a correct level of precision and F1-score, we feel much better could have been achieved with more computation power. Indeed, we could think of many improvements but couldn't implement them, due to time and computation capacity constraints. A simpler model would probably have been easier to train, but we wanted to try something new, as the architecture we used was published in 2020. While the outcome does not reach our expectations, we still have learnt a tremendous quantity of knowledge on neural networks, and how they should be built and optimized.

B. Possible improvements

We would have liked to perform better hyperparameters tuning, in the sense that we had to do it on a limited number of simulations due to the constraints we faced, and thus it could only give us insights and no clear direction.
It would also have been interesting to perform more cross-validation on the pixel foreground threshold, i.e. the percentage of pixels required to assign a foreground label to a patch.
Ideally, we would also have liked to train our models for a longer time, since we never willingly stopped our simulations : it was always because of our computational capacity constraints.
Another thing we tried but couldn't push far enough was to change the loss to a focal loss, which is known to penalize better imbalanced dataset such as the one we had. We had a model that achieved a F1 score of **0.828**, which was convincing since the model was trained on a limited number of epochs. This value was only reached through post-processing the results, applying a threshold to the shades of grey. There was a large discrepancy between the pre-processed results, as shown in fig. 4. We thus believe that further training and more intelligent and adaptive post-processing could lead to better results than what was obtained with Dice loss.



(a) Before processing (b) Post processing

(c) Before processing (d) Post processing

Fig. 4: Predictions for test image #13 and test image # 20 using a Focal Loss.

V. Conclusion

We estimate that given all the constraints we faced, we came up with an acceptable model for the given task of road segmentation, in the sense that most roads are correctly classified. Its main drawbacks are the classification of curves and of shades on the roads. This paper presents an attempt to build a network that would combine efficiency and scientific novelty. To this extent, we came up with the idea of the Double UNet with the help of a pretrained model, which we decide to push further. However, this may have caused us more bad than good, since this model was maybe too complex and heavy for the resources at our disposal.

REFERENCES

[1] D. Jha, M. A. Riegler, D. Johansen, P. Halvorsen, and H. D. Johansen, "Doubleu-net: A deep convolutional neural network for medical image segmentation," in *2020 IEEE 33rd International Symposium on Computer-Based Medical Systems (CBMS)*, pp. 558–564, 2020.

[2] K. Team, "Keras documentation: Keras applications."

[3] N. Adaloglou, "Deep learning in medical imaging - 3d medical image segmentation with pytorch," Apr 2020.

[4] N. Adaloglou, "Intuitive explanation of skip connections in deep learning," Mar 2020.

[5] H. He, D. Yang, S. Wang, S. Wang, and Y. Li, "Road extraction by using atrous spatial pyramid pooling integrated encoder-decoder network and structural similarity loss," *Remote Sensing*, vol. 11, p. 1015, 04 2019.

[6] "Squeeze and excitation networks explained with pytorch implementation," Jul 2020.

[7] K. Team, "Keras documentation: Keras optimizers."

[8] J. Brownlee, "Gentle introduction to the adam optimization algorithm for deep learning," Jan 2021.

[9] S. Jadon, "A survey of loss functions for semantic segmentation," 2020.