Road Segmentation Machine Learning challenge

Vignoud Julien, Lepeytre Hugo, Benhaim Julien

***Abstract*—In this project we address an image segmentation problem that consists in detecting roads within satellite images. For that purpose, we implemented different models and architectures. Specifically, we compare the use of the U-Net architecture [1] to segment whole images and a CNN model which maps each image to a 2-d array of labels in a patch-wise manner. Data augmentation coupled with a fine-tuned Tversky loss improve the performance of our models on metrics such as the rotation invariance or overfitting. In the end we find that the U-Net model is the best model with an F1-score of 90%.**

I. Introduction

Image segmentation is an important problem in computer vision, in which each pixel or group of pixels of an image has to be labeled as belonging to one of several classes. In our case, road detection, the goal is to build a model that separates patches in two classes: road and background. Patches are squares of 16×16 pixels that do not overlap. Our performance metric is the F1 score rather than the accuracy; this is because classes are unbalanced and so it is very easy to have a high accuracy by only predicting the dominant class.

On the practical side, we have a training set composed of 100 satellite images of size 400×400 and their respective ground truths which are masks where each pixel is either 1 (road) or 0 (background). This will be subdivided into 3 sets: train, validation and test.
We then have another set of 50 satellite images of size 608×608. It is labeled, but we are not given access to those labels. Our task is to make predictions on the set and submit them on AICrowd to have them checked against the real labeling. As a result of this we are only given our accuracy and F1 score, as well as the best results of the other participants.

In Section II we introduce the different models and motivate our design choices. We also detail our improvements. Section III compares the different models and describes our numerical achievements. Finally, we present our contributions in Section IV.

II. Models and Methods

A. Baseline model

As a baseline we use the provided Convolutional Neural Network made up of 2 convolutional layers separated by max pooling layers. The two last layers are fully connected and the activation function is a ReLU. This model labels whole patches directly instead of predicting individual pixels and averaging over their values to label a patch. It is trained using a binary cross-entropy loss and a stochastic gradient descent. It attained an F1 score of 0.653[1].

B. U-net

Now we choose to explore the U-Net [1] architecture as it is designed specifically for image segmentation and is supposed to outperform other existing models such as the sliding-window convolutional network [2]. We reproduce the original architecture illustrated in Figure 1, that is composed of a contracting path, a bottleneck and an expanding path where inputs from the contracting path are concatenated with the current layer. Finally a 1×1 convolutional layer maps the feature vectors to a class probability. That is to say, the model outputs a matrix M of size 400×400 where

$$M_{ij} = p(\text{pixel}_{ij} \text{ is a road pixel})$$

The activation function is always the ReLU function except for the last layer that uses the sigmoid function in order to have a valid probability value. This basic architecture yields an F1 score of 0.836.

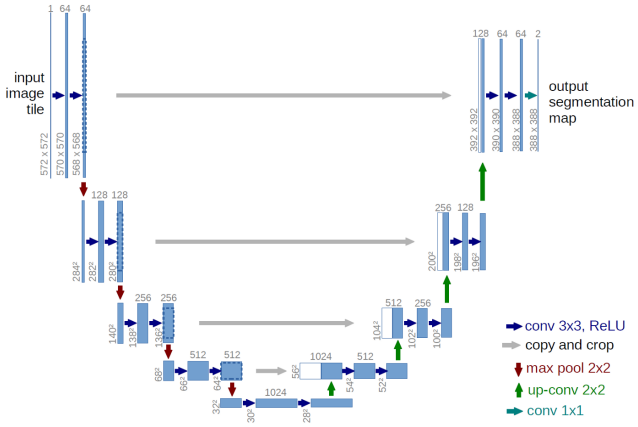

Figure 1. U-Net architecture.

On top of that we add batch normalization,i.e., scaling and shifting the feature vectors at each layer to set a zero mean and unit standard variation, in order to decrease the training time and act as regularization [3]. This does not improve performance but the training time reduces significantly. While batch normalization is supposed to help avoid the use of dropout layers, there is still an improvement when using them after normalization with a rate of 0.15. The different scores are represented in Table I.

We also implement dilated convolution [4] in the expanding path to widen the context window taken of each

[1]All F1 score measures are taken on data unused during the training phase, the validation set, which is also different from the testing set that is verified by AICrowd).

Table I
F1 SCORE WITH RESPECT TO THE DROPOUT RATE

Dropout rate	0	0.15	0.25	0.4
F1 score	0.842	**0.859**	0.800	0.752

convolution so that we can mitigate issues such as trees hiding the road. The dilated convolution is illustrated in Figure 2 and works as a regular convolution but takes as inputs, points that are spread apart and not adjacent. However, it does not significantly improve performance, and thus is not used in the final model.

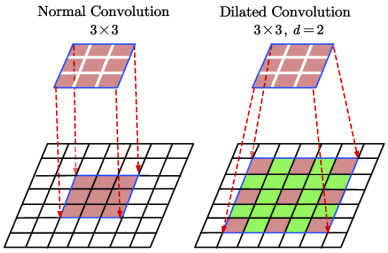

Figure 2. Regular convolution and a dilated convolution with a dilation rate $d = 2$. Source: [5]

Regarding optimizers we compare SGD, Adam, Adagrad and Nadam because they implement different learning rate decays and momentums. [6]. All of them converge towards the same optimum, except Adagrad which converges to a lower F1 score (0.847 compared to around 0.86 for the others). In the end, we chose the fastest optimizer, Adam [7].

Finally, we try replacing the ReLU activation function by the LeakyReLu [8], which aims at mitigating the vanishing gradient problem. However, we observe no significant difference so we decide to keep the original ReLU.

C. Patch-wise CNN

The second model architecture we explore is a fully convolutional network. Like the baseline, it outputs class probabilities for patches instead of individual pixels. The idea is to map directly a 400×400 image to the corresponding grid of patches of dimension 25×25. Thus, the architecture consists of convolutions separated by 4 max pooling layers with a kernel size of (2×2). This creates an output dimension of $400/2^4 = 25$. The convolutional layers use padding so as to keep the same dimensions from input to output.

The advantage of this convolutional architecture is that it restricts the context used to small windows for each patch predicted, as opposed to a fully connected architecture which would take into account every pixel in the image. This would not be relevant in the context of the problem and even worse, those fully connected layers would have much more parameters to train. It is also a much simpler architecture than the U-Net, with a more reasonable training time, even with a lot of data. The loss function used with this CNN is the binary cross-entropy. The need for the Tversky loss in the previous model was justified by the highly imbalanced distribution of pixels. This is not so much the case when we reason with patches, because 25

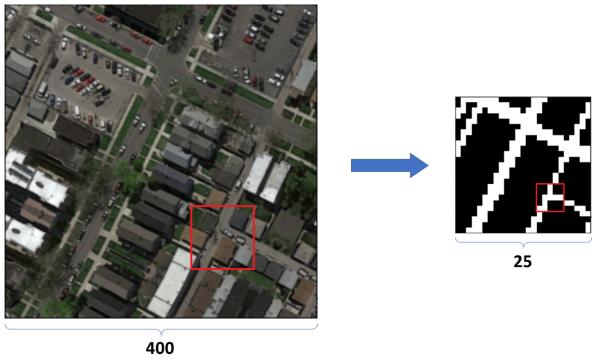

Figure 3. Output of the patch-wise CNN model

Table II
TEST SET METRICS FOR DIFFERENT ARCHITECTURE OF PATCH-WISE CNN

Architecture	1.	2.	3.	4.
F1 Score (test set)	0,88	0,885	0,885	0,88
Accuracy	0,938	0,941	0,941	0,938
Precision	0,893	0,911	0,902	0,898
Recall	0,868	0,86	0,87	0,863

percent of road pixel in a patch is enough to classify it as road. We use the same optimizer as before, i.e., the Adam optimizer. We also use dropout regularization with a rate of 0.2 to prevent overfitting.

We try out 4 different CNN architectures:

1) A simple Neural Net with 5 convolution layers; filter sizes from 64 to 128; kernel size of 3×3; LeakyRelu activations (rate 0.05).
2) Another one with increased filter sizes (128 to 256) and thus increase expressive power of the model.
3) A deeper architecture (two convolutions before each max pooling) with low filter size (64) for a total of 8 convolution layers. The aim is again to increase the representative power of the model, while avoid a drastic increase in training time.
4) Finally, the same deep architecture with increased filter sizes as well as a kernel size of 7×7 for the first convolution in order to increase the context windows.

Unfortunately, all these variations of architecture have little impact on performance, as showed in table II. The score of these models on AICrowd varies between 0.860 and 0.865.

Although we observe no significant variations, we are satisfied with the performance of this architecture. The best improvement for the model is to use data augmentation, presented in section II-E. We also change classical dropout with 2D spatial dropout layers. Dropout layers are less efficient in CNNs because the dropped pixels on an image can be easily inferred from the context. Spatial dropout layers address this problem by dropping entire regions from a feature map. [9]. Spatial dropout combined with a greater dataset (due to data augmentation) enables the model to generalize better: the validation loss is closer to the training loss. In the end, we improve the performance from 0.865 to 0.882.

D. Loss function

A challenging issue of road segmentation is the class imbalance; There is much more background than road. Therefore maximizing the accuracy is not equivalent to maximizing the F1 score. Instead of using the binary cross-entropy, a solution we have found is to use the dice loss [10], which is based on the F_1 score itself (also called the Sørensen–Dice coefficient hence the name of the loss function), as it weighs equally precision and recall.

$$Dice\ loss = 1 - \frac{2TP}{2TP + FP + FN}$$

As a result, the score increased from 0.843 to 0.851.

Following this line of reasoning, we study the Tversky loss [11], based on the Tversky index, that lets us weigh precision and recall, useful for highly imbalanced data. The α and $\beta = 1 - \alpha$ parameters control the magnitude of penalties for FPs and FNs:

$$Tversky\ loss = 1 - \frac{2TP}{2TP + \alpha FP + \beta FN}$$

Therefore increasing α penalizes false positives, i.e., wrongly classifying background as road. The α parameter fine tuning is reported in Table III. Notice that for $\alpha = 0.5$, the Tversky loss is equivalent to the dice loss and is equal to $1 - F_1$ where F_1 is the F1 score.

Table III
F1 SCORE WITH RESPECT TO THE TVERSKY α PARAMETER

α	0.4	0.45	0.5	0.6	0.7
F1 score	0.843	0.848	0.851	**0.857**	0.848

Finally we implement the Focal Tversky loss [12], parametrized by γ, a non-linear generalization of the Tversky loss:

$$Focal\ Tversky\ loss = (Tversky\ loss)^{\gamma}$$

The non-linear nature of the loss gives us control over how the loss behaves for different values of the Tversky loss.

$$\nabla FTL = \gamma (TL)^{\gamma - 1} \nabla (TL)$$

For images where the loss is high, increasing γ results in a greater gradient magnitude, therefore making the model focus on harder images. The F1 score for different γ values are reported in Table IV. The Focal Tversky loss with $\gamma = 1$ is equivalent to the Tversky loss.

Table IV
F1 SCORE WITH RESPECT TO THE FOCAL TVERSKY γ PARAMETER

γ	0.6	0.75	1	1.2
F1 score	0.853	**0.861**	0.857	0.848

E. Data augmentation

As our training set contains only 100 images and the U-Net architecture relies on the strong use of data augmentation [1], we implement a pipeline to artificially create new images from existing ones by using transformations such as rotations, horizontal and vertical flips and horizontal and vertical shifts. Rotations are performed up to an angle of 45° and at most half of the image can be shifted. In order to fill missing parts of the resulting transformations and make use of the whole image size, we mirror bordering parts as shown in Figure 4. Naturally, the same transformations are applied to the respective ground truths. We restrain ourselves to these transformations because others such as ZCA whitening, blurring or shearing may not reflect reality and create images that would never appear in real cases which is not useful for the model to learn.

We observe significant improvement, especially for diagonal roads which were underrepresented in the training set and thus often misclassified. But we notice that some images are still hard to classify, such as parking lots and wide motorways. Therefore, we hand pick a selection of these from the training set, and we create more augmented images based on this hard set in order to increase their representation. 32 images presenting prediction issues are selected during this procedure. These could be either parking lots, messy roads or trees.
In the end, training the model with 900 images with an increased representation of hard images makes the F1 score on the test set go from 0.861 to 0.9.

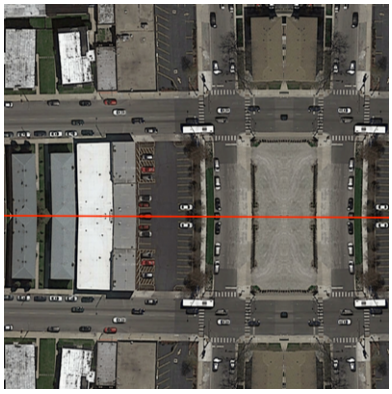

Figure 4. Augmented image created by shifting the original image and mirroring to fill the missing part. The red line denotes the symmetry axis.

F. Feature augmentation

As an exploratory procedure, we develop another augmentation idea: Instead of feeding a single RBG image per input to the model (which is an array of shape $400 \times 400 \times 3$) we give it different versions of the same image at the same time. This differs from data augmentation in a significant way because in this case we still have the same number of training samples, it is only the number of features given to the model that increases. So this procedure is closer to feature augmentation than data augmentation

Spatial modifications of the image were relevant for data augmentation but not so much here: since our models are convolutional they take spatial context into account and so if we add more channels to our image it has to have the same spatial layout. On the other hand this time it would not be a problem to create images that will not come up in real

life because for every real-life image we will recreate those modification and they with always be analysed through the context of the original image.

With these things in mind, we know that we can perform any kind of modification to our image as long as spatial context is not modified. So the ones we try are: making the images brighter, dimmer, and various kinds of whitening transformations such as ZCA whitening. Unfortunately, the leads to no significant impact on the performance of the model and will not be retained in the final architecture.

G. Variable image size prediction

In order to segment images from the AICrowd test set we have to adapt our model as it is trained on 400×400 images while the test set contains 608×608 images. We considered different possibilities in order to train a variable input size model. However, existing solutions such as Spatial Pooling Layers [13] were not adapted or not performing well in our case. We choose instead to make predictions in the following way: first splitting each 608×608 image into 4 400×400 images, and then predicting their masks and merging them back into a single 608×608 mask, averaging probabilities of overlapping pixels. Finally we binarize each pixel value between 0 and 1 and we label each patch according to its proportion of road pixels. The steps are illustrated in Figure 5. That way, the model is trained on 400×400 images and there is no need for any adaptation or compromise to segment 608×608 images.

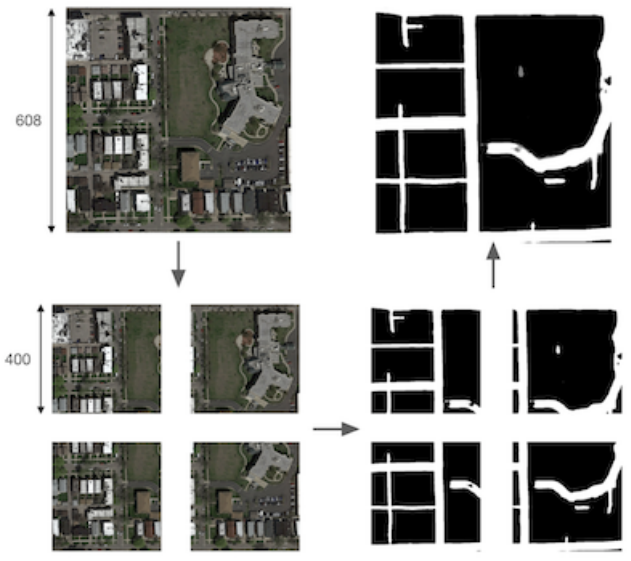

Figure 5. Different steps performed to segment 608×608 images. Notice that some of the parts of the image are redundant among the smaller masks. Moreover we can also notice the averaging of predictions as the small white circle predicted in only one mask becomes gray (less certain) after the averaging.

H. Hardware and software

We used Tensorflow version 2.3.0 with Keras API to build and train our models. Google Colab notebooks, providing a free Nvidia Tesla V100 GPU has been used for the training.

III. Results

The best score we achieved on this competition was a F1 score of 0.90 with an accuracy of 0.947. This as been obtained with the U-Net architecture. We trained it with the Adam optimizer (learning rate 0.001), the Focal Tversky loss with parameters $\alpha = 0.6$ and $\gamma = 0.75$. The training dataset was composed of 900 images augmented with flip, rotations, and an over-representation of hard-to-classify images. See table V for model comparison.

Table V
F1 score comparison of models, on the competition dataset

Model	Baseline	Patch-wise CNN	U-Net
F1 score	0.653	0.882	**0.90**

IV. Discussion

Some final thoughts on the whole project:
Our most significant improvements during the research process were without contest the use of a U-Net architecture and the data augmentation. Most of the other ideas explored hardly made any significant difference. A big limitation encountered was that we had a very limited amount of images we could load in the RAM before Google Colab crashed so we could not explore the data augmentation further. Finally, another idea that we did not have time to try was to generate data on the fly during training using Keras in order to avoid overfitting and augment generalization of the model.

Figure 6. Some masks predicted on the submission set with interesting insights. In the top-left image, we can see that the model predicts a road behind the white building even though we cannot see it: that shows that predicting on whole images rather than on patches yields more context. Similarly, in the top-right image we can see that trees don't impede the model, thanks to the context and the U-Net architecture, in which we concatenate features from the contracting and expanding paths. In the bottom-left image we can see that heavy data augmentation resulted in a proficiency in recognizing diagonal roads. However, the bottom-right image shows that the model still has issues with parking lots or big concrete areas, even after increasing the representation of such images in the training set.

REFERENCES

[1] P. F. Olaf Ronneberger and T. Brox, "U-net: Convolutional networks for biomedical image segmentation."

[2] L. M. G. J. S. Dan C. Cires, Alessandro Giusti, "Deep neural networks segment neuronal membranes in electron microscopy images."

[3] C. S. Sergey Ioffe, "Batch normalization: Accelerating deep network training by reducing internal covariate shift."

[4] X. H. Xinyu Lei, Hongguang Pan, "A dilated cnn model for image classification."

[5] Y. L. Z. Z. Z. H. A. J. JINGLONG DU, LULU WANG, "Brain mri super-resolution using 3d dilated convolutional encoder–decoder network."

[6] S. Ruder, "An overview of gradient descent optimization algorithms."

[7] J. L. B. Diederik P. Kingma, "Adam: A method for stochastic optimization."

[8] T. C. M. L. Bing Xu, Naiyan Wang, "Empirical evaluation of rectified activations in convolutional network."

[9] Q. V. L. Golnaz Ghiasi, Tsung-Yi Lin, "Dropblock: A regularization method for convolutional networks."

[10] S.-A. A. Fausto Milletari, Nassir Navab, "V-net: Fully convolutional neural networks for volumetric medical image segmentation."

[11] D. E. Seyed Sadegh Mohseni Salehi and A. Gholipour, "Tversky loss function for image segmentation using 3d fully convolutional deep networks."

[12] N. M. K. Nabila Abraham, "A novel focal tversky loss function with improved attention u-net for lesion segmentation."

[13] S. R. J. S. Kaiming He, Xiangyu Zhang, "Spatial pyramid pooling in deep convolutional networks for visual recognition."