Road segmentation: project 2 for Machine Learning (CS-433)

Elia Fantini, Anastasiia Filippova, Narek Alvandian
Department of Data Science, EPFL, Switzerland

***Abstract*—Image segmentation is the process of partitioning a digital image into multiple sets of pixels, also known as image objects. In this task, such a process is applied to aerial images acquired from Google Maps in order to recognize roads out of background pixels. For this purpose, we used different Convolutional Neural Networks used every day in many state-of-the-art applications, such as U-Net, DeepLabV3, and DeepLabV3+. Furthermore, we tried different pre and post-processing techniques and we found the best pipeline to highly improve the performance of our machine learning model: our work resulted in a final f1 score of 0.91 on AICrowd.**

I. Introduction

Nowadays more and more computer vision applications are being used in all industry fields. Among them, image segmentation is particularly popular, as it is helpful to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Among different use cases, there are autonomous vehicles that use machine learning to recognize and elaborate surrounding objects and buildings. Convolutional Neural Networks have proven to perform extremely well on complex problems and image segmentation is one of them. Hence, we decided to experiment with different CNN architectures to address our binary classification challenge: given satellite pictures from Google Maps - identify and label each image's pixel, as either the road or anything else. Dataset we were given to train our model on, is a set of 100 images with a resolution of 400x400 RGB pixels, whereas the 50 testing images provided have a size of 608x608. Even though our neural networks were trained to perform pixel-wise predictions, ground truth data had predictions applied to groups of 16x16 pixels, referred to as patches. To be more specific, a patch is classified as a road if at least 25% of its pixels are labeled as road. The difference between the two prediction methods is shown in figure 1.

II. Data preprocessing

To improve the accuracy and F1 score of our models we tried different preprocessing techniques. Using the UNet[6] with constant parameters, we measured the difference between the performance, when using and not using the preprocessing methods during 50 epoch training. This way we can asses the effectiveness of a particular method. In the end, not all of the suggested preprocessing techniques proved to be useful.

A. Data augmentation

Since CNNs usually need a lot of data to perform well and we were only given 100 images, we decided to augment our data. We performed different transformations to each image. We did horizontal, vertical flips and random rotations. Pictures from the training set mostly had roads parallel to axes, models had difficulties in recognizing diagonal roads. This is why, we added a rotation of each image by 90, 180, and 270 degrees, as well as rotations with randomly generated angle values. After this process we saw a significant improvement in our metrics, Top-1 F1 score and accuracy went from 0.8372 and 0.9283, to 0.8836 and 0.9471, respectively. As we expected, UNet performs much better with augmented data. Not only did the final scores improve, the loss decreased faster. At the same time, F1 score increased much more rapidly (see plots 3).

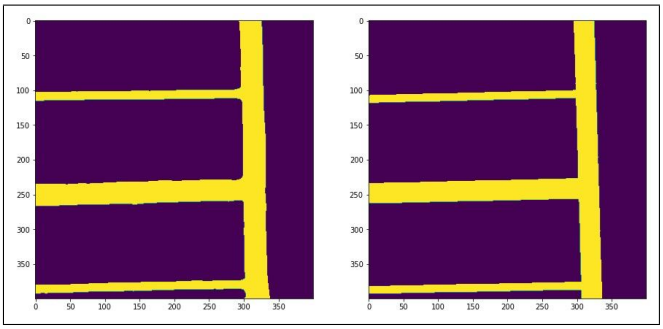

Fig. 1. Difference between pixel-wise predictions (left) and patch-wise ones (right)

B. Patches division

Another solution for augmenting the number of entries for the CNN was to divide an image into more sub-images. So, we divided each 400x400 picture into 25 patches of 80x80 pixels. Another benefit of this method is that we shuffle the data when we provide it to the CNN, hence dividing the images improves randomization. This, combined with the fact that smaller CNN entries require less RAM usage, let us use a bigger batch size, improving the gradient descent of the optimizer. After the experiments, this method didn't improve the performance noticeably, but rather made the model less stable, as we saw frequent drops/peaks in F1/loss during the training. For this reason we decided not to use patches division in our pipeline.

C. CLAHE and Gamma correction

Contrast enhancement techniques are useful to improve images' appearance and might increase the performance of our model highlighting the road for those images where it

doesn't stand out by default. Among different algorithms we chose Contrast Limited Adaptive Histogram Equalization (CLAHE) since it modifies histogram modifications locally, solving global modification's problems. We also tried to apply gamma correction to see if it had any positive effect on the CNN performance. Metrics did not improve significantly but F1 score grew much faster, it also resulted into training being more stable (see plots 3). On the other hand, gamma correction dropped the performance significantly. Hence, we decided to add only CLAHE to our preprocessing pipeline.

D. Data standardization

Neural networks process inputs using small weight values, thereby inputs with large integer values can derange or slow down the learning process. For this reason it is always a good practice to scale the pixel values so that they have values between 0 and 1, instead of 0 and 255. Such scaling was made for all of out training procedures. We also tried to perform standardization on every channel of the image, subtracting the channel-wise pixels' mean and dividing everything by standard deviation. Unfortunately, this last method decreased the performance, as shown in the table I. Hence, we did not include it in the submission pipeline.

III. Models and Methods

As mentioned earlier, Convolutional Neural Networks proved to be the best choice for this kind of application, hence we tried to experiment with different famous architectures. We started with a U-Net [6]. Subsequently, we moved to some more domain-specific pre-trained models. As a baseline, we used a simple Logistic Regression.

A. Baseline with Logistic Regression

We first wanted to try using Logistic Regression, as it is one of the most common statistical models for simple binary problems. For this baseline we used a different preprocessing pipeline, dividing the images into 16x16 pixels patches. Then, for each patch's RGB channel we computed the mean and variance, creating 6-dimensional features. Such a technique achieved an accuracy of 0.58 and an F1 score of 0.46, which are relatively low but this is something to be expected with such a simple model.

B. U-Net

U-Net is a convolutional neural network originally used for biomedical image segmentation. This choice was made because this CNN, was specifically made to perform well even when there is not much training data. In the original paper [6], authors outperformed prior state-of-the-art methods which utilized a sliding-window convolutional network. The architecture consists of a first contracting path to learn context and a symmetric expanding path for precise localization [6]. The first module of the network is composed of 2D convolution, batch normalization, and activation function, everything repeated twice. Such a module is called Double Convolution. Then, the contracting path is made by five consecutive layers of 2D Max Pooling and Double Convolution modules. The number of features doubles each time we go deeper in the network, starting from 32. Subsequently, there are five upsampling layers, which either do bilinear interpolation or transpose convolutions, followed by a concatenation of a feature map from contracting path, followed by double 3x3 convolution. Finally, a 2D convolutional layer with unitary kernel size outputs the predictions. For the activation function, we chose the rectified linear unit (RELU). A big benefit of this function is that it doesn't activate all nodes at the same time and is simple and fast to compute since it converts all negative inputs to zero and the node does not get activated. If the input is positive, it's not modified in any way. As a drawback, when a node isn't activated, the gradient is equal to zero and during backpropagation, all the weights will not be updated. As previously said, without any kind of preprocessing we reached an F1 score and and accuracy of 0.8372 and 0.9283, and they became 0.8836 and 0.9471 respectively after data augmentation. The improvement over the Logistic Regression is noticeable and was quite expected.

C. DeepLabv3

DeepLabv3 is a pre-trained model from [2]. The main feature of this architecture is the employment of atrous convolution in cascade or in parallel to capture multi-scale context, thanks to the use of multiple dilation rates. Dilation rate is a convolution parameter that defines a spacing between the values in a kernel. This way the field of view is wider but the computational cost remains the same. Another architectural key element is the Spatial Pyramid Pooling (SPP), a pooling layer that lets the network process any image, without any constraint on size. The SPP layer pools the features and generates fixed-length outputs, solving the problem of the different resolutions between train and testing data. As shown in table I, we obtained noteworthy improvement from UNet.

D. DeepLabv3+

After having received good results with DeepLabv3, we tried using its updated version, DeepLabv3+ [1]. This new architecture adds to the previous one a decoder module to improve general predictions, especially along object boundaries. They applied a depth-wise separable convolution both to the decoder and the ASPP (Atrous Spatial Pyramid Pooling) modules. Such convolution splits a kernel into two separate kernels that do two convolutions: the depth-wise and the point-wise convolutions. Compared to the separable one, depth-wise separable convolution needs much fewer multiplications to be computed. This results into faster and stronger encoder-decoder network. With only data augmentation as preprocessing, we achieved 0.8969 for F1 score and 0.953 for accuracy at the 50-th epoch, outperforming DeepLabV3 as shown in table I. Furthermore, the convergence rate was faster, plots were more stable and metrics reached even higher peaks than the scores just mentioned (see plots 3). Hence, we decided to use this pretrained model for our submission.

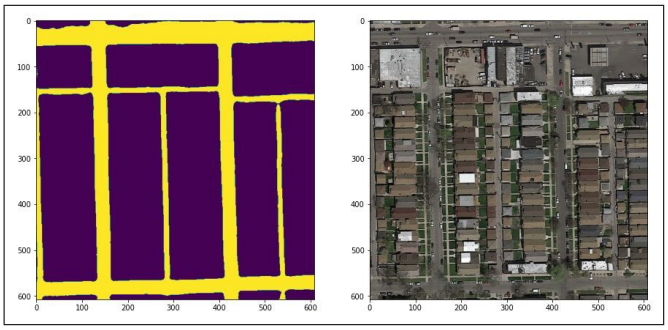

Fig. 2. Final model: example of predictions (left) and original image (right)

TABLE I
LOSS, ACCURACY AND F1 AFTER 50 EPOCHS

	Loss	Acc	F1
Logistic Regression		0.588	0.461
Base:UNet + BCELoss	0.150	0.928	0.837
Base + patches	0.272	0.927	0.833
Base + augm	0.206	0.947	0.883
Base + augm + clahe	0.219	0.944	0.879
Base + augm + gamma	0.206	0.947	0.883
Base + augm + gamma + clahe	0.219	0.944	0.879
Base + augm + norm	0.347	0.9226	0.829
DeepLabV3 + Dice + augm	0.165	0.952	0.894
DeepLabV3+ + Dice + augm	0.151	0.953	**0.896**

E. Optimizations

Once the architecture of the neural network was chosen, we had to choose other components. Good choice of the loss function, and optimizer are crucial for good performance.

1) Loss function: We experimented with two objective functions to be minimized: binary cross entropy loss (BCE) and dice loss. In both cases the loss was calculated on each pixel, not just on patches. BCE loss is defined the following way:

$$l(x,y) = \{l_1, \dots, l_N\}^T, l_n = -w_n[y_n log x_n + (1-y_n) log(1-x_n)]$$

where N is the size of the batch. Subsequently, the mean of the vector's values is computed. This aspect may be a drawback, because the per-pixel loss is calculated discretely, without knowing whether its adjacent pixels are boundaries or not. On the other hand, Dice loss considers the loss of information both locally and globally, which is critical for high accuracy. The equation for dice loss is the following:

$$DL = 1 - \frac{2\sum_{n=1}^{N} p_n r_n}{\sum_{n=1}^{N} p_n^2 + \sum_{n=1}^{N} r_n^2}$$

In this case p_n is the predicted probability for the positive class and $r_n \in [0, 1]$ is the ground truth of each pixel $n \in N$. In our experiments we achieved better results with Dice Loss compared to BCE. After some epochs, we could observe the validation loss started to increase, even though accuracy and F1 increased as well (see plots 3). This happens because BCE is sensitive to unbalanced dataset and our dataset contains a lot more background pixels.

2) Optimizer: Optimizers are another key element of the training process. Among many different algorithms, the most popular is by far Adam, which usually requires little tuning on the learning rate hyperparameter. For all our experiments we used a learning rate of 0.01, whereas with pretrained models we used a ten times slower one, in order not to change the pretrained weights rapidly.

3) Postprocessing: Data augmentation applied during the preprocessing pipeline had also the purpose of improving the invariance of the model to different transformations. Hence, we decided to make the most out of this property by flipping and rotating test images as well, allowing the CNN to analyze it from different points of view and extract as much information as possible. The final patch label's prediction is computed by averaging the resulting predictions of the same patch from all the transformations.

IV. RESULTS AND CONCLUSIONS

In table I we are reporting our results. Among all different preprocessing methods, data augmentation was the key for improving the performance. It as expected, since convolutional neural networks require a lot of data. Other techniques instead showed no noticeable improvement, and in some cases also a decreased the metrics.The only one that showed a slight improvement in the stability of the model during training was CLAHE. Between the two loss functions, Dice Loss proves to be better since it is more robust to our unbalanced dataset. The BCE Loss instead showed a raising validation loss over epochs. Finally, between all models DeepLabV3+ gave us the best result, probably because it was already pretrained. In order to not change the pretrained weights rapidly, we put the learning rate equal to 0.0001 , which is 10 times less than the one we used when training models from scratch. The last key element was the postprocessing. This technique was applied to all experiments and showed a good improvement in metrics, because of the strategy of averaging over flips and rotation predictions. All methods combined we achieved an F1 score of 0.91 on AICrowd platform and the result on a sample image is shown in figure 2.

V. FUTURE WORK

Many different adaptations, tests, and experiments have been left for the future due to lack of time. The following ideas could be tested in the future:

1) Add extra data to increase model's generalization ability. For example, we could take this dataset from a research thesis of the University of Toronto [4].
2) Use the combination of DiceLoss and BCELoss inspired by the paper "A survey of loss functions for semantic segmentation" by Shruti Jadon [3].
3) Test different optimizers. In fact, according to the paper "On the Convergence of Adam and Beyond" Adam optimizers does not always converge [5] and a possible solution is using AmsGrad optimizers. Other researchers implemented a new optimizer, called Yogi, which proved to outperform Adam in many applications[7].

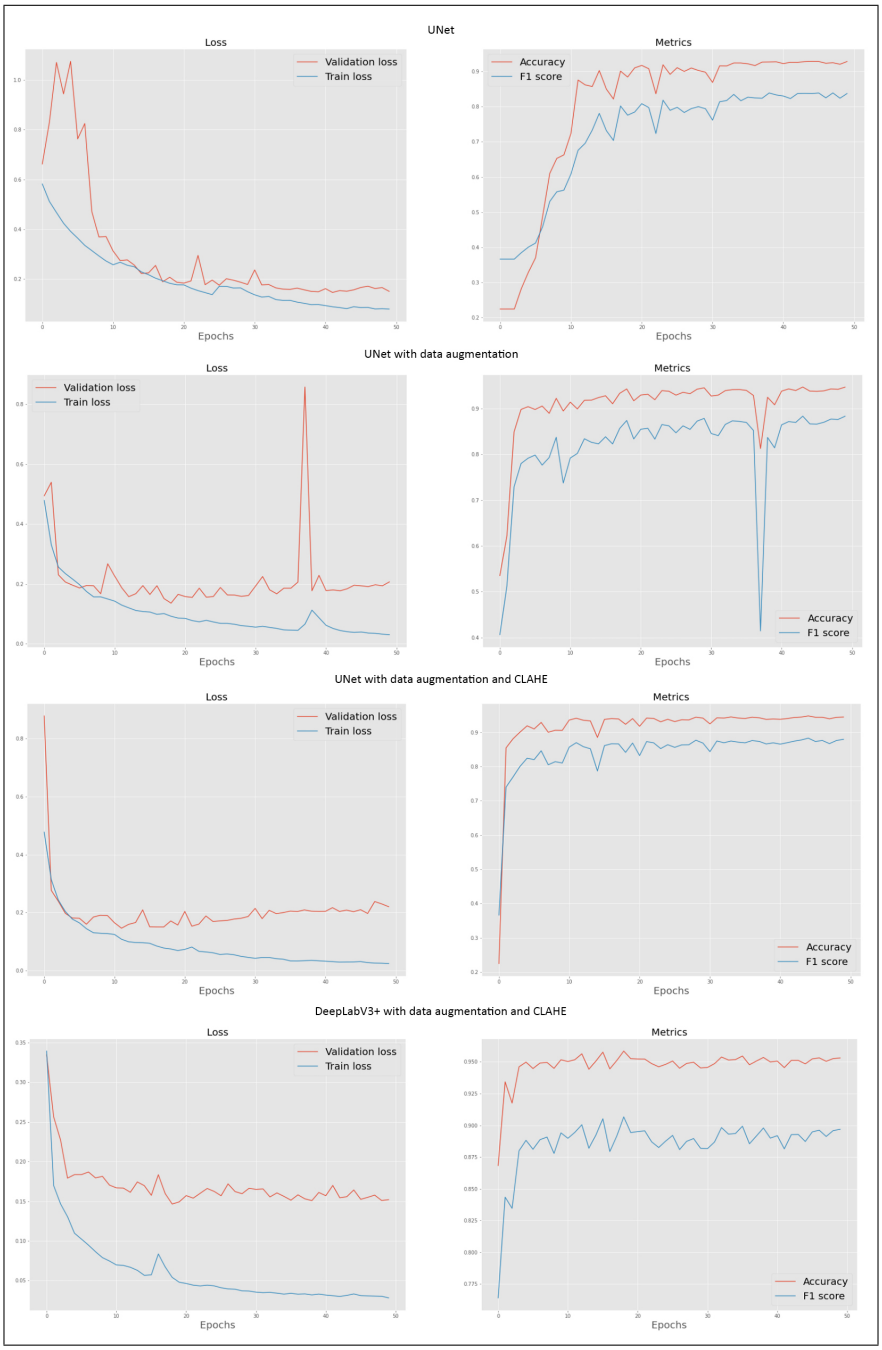

Fig. 3. Evolution of losses (left) and accuracy/f1 (right) over 50 epochs

4) To avoid overfitting and improve generalization, different ensemble machine learning algorithms may be tried, such as Stacked Generalization. To implement such a technique would require training a lot of models, called base models, and then training a meta-model, that learns how to best combine the predictions of the base models. All that would take a lot of computational power and time.

REFERENCES

[1] Liang-Chieh Chen et al. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: (Feb. 2018).

[2] Liang-Chieh Chen et al. "Rethinking Atrous Convolution for Semantic Image Segmentation". In: (June 2017).

[3] Shruti Jadon. "A survey of loss functions for semantic segmentation". In: *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)* (Oct. 2020). DOI: 10.1109/cibcb48159.2020.9277638. URL: http://dx.doi.org/10.1109/CIBCB48159.2020.9277638.

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

[5] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. *On the Convergence of Adam and Beyond*. 2019. arXiv: 1904.09237 [cs.LG].

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: (2015). Ed. by Nassir Navab et al., pp. 234–241.

[7] Manzil Zaheer et al. "Adaptive Methods for Nonconvex Optimization". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/90365351ccc7437a1309dc64e4db32a3-Paper.pdf.