Class Project 2 in Machine Learning (CS-433)
Road Segmentation on satellite images

Florian Grötschla[1], Adrien Bertaud[2], and Maximilian Wessendorf[1]

[1]IN-H, EPFL
[2]AUDIT-H, EPFL

***Abstract*—The goal of this project is to perform road segmentation on satellite images from google maps. We use the U-Net convolutional network architecture and optimize it regarding the complexity of the model and the usage of dropouts as we observed a tendency for overfitting during initial experiments. Against our initial expectations we find that having no dropouts and a more complex model gives us the best performance. We finally use this insight to build our final model that scores a F1-score of 0.876 in the AIcrowd competition.**

I. Introduction

With an ever increasing amount of satellite image data resulting from advances in private space economy it becomes more and more important to extract meaningful information from this kind of data automatically. In this road segmentation challenge the goal is to find and highlight all the roads on such image data automatically.

II. Data set

The provided data set contains a set of 100 satellite images acquired from GoogleMaps. It also provides ground-truth images where each pixel is labeled as road or background. Those images are 400x400 pixel in size each. Besides that there are 50 target images of size 608x608 for the final predictions that have to be submitted to the AICrowd ranking system. The images provided with ground-truth and those provided for submission have similar zoom factor, brightness and color.

III. Preprocessing

To enhance the performance of the actual model used for road segmentation we apply pre-processing to our data set. As the set of the 100 images used for training and testing is quite small one important thing to do is to increase the amount of training data. To achieve that we use the ImageDataGenerator[1] provided by Keras. The ImageDataGenerator generates the augmented images on the fly that are than used in the training process.

We use it to flip the images horizontally and vertically, shift their heights and widths and rotate them. We set the shift ranges for both width and height to 0.1 and the range for rotation to 20 degrees.

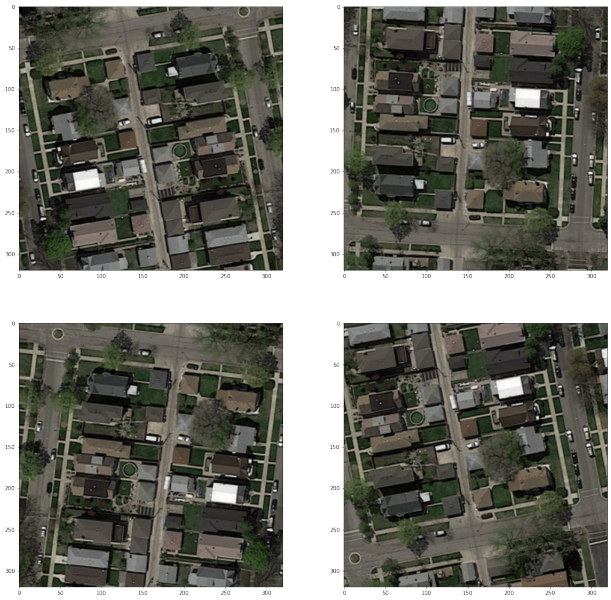

Figure 1. Four versions of the same image with different augmentations we applied to the images in the data set.

We don't use other augmentation steps like zooming or change of brightness and color because the target images have the same zoom factor and also are similar in regards of color and brightness.

To ensure that there are no non existent information from the shifting or rotation presented in the used images, we crop the augmented images at a size of 320x320 from the center. Figure 1 shows four randomly generated images with the described preprocessing pipeline.

IV. Method

To do state of the art road segmentation we utilize U-Net [1] which is a convolutional network architecture for fast and precise segmentation of images.

U-Net is a network architecture that consist of an encoder (contracting path) and a decoder (expansive path) part that are symmetrical and are connected by skip connections,

[1]https://keras.io/api/preprocessing/image/

leading to an ”U” like shape as shown in Figure 2 from [2]. The encoder part follows the typical architecture of a convolutional network. It consists of the repeated application of:

1) a first 3x3 convolution layer (unpadded)
2) followed by an activation function, we used Exponential Linear Unit (ELU)
3) followed by a batch normalization
4) a 2x2 max pooling operation with stride 2 for down-sampling
5) a second 3x3 convolution layer (still unpadded)
6) followed by an ELU
7) followed by a batch normalization

We had to choose one activation function between ReLU, Leaky ReLU, SELU, ELU, GELU, etc [2]. We have chosen ELU, that is commonly used in U-Net, because it presents several advantages[3]:

- it tends to converge fastly (better than ReLU, because mean ELU activations are closer to zero)
- it has good generalization performance (better than ReLU)
- it is fully continuous
- it is fully differentiable
- it does not have a vanishing gradients problem
- it does not have an exploding gradients problem
- it is faster to compute (better than SeLU [4])

At each down sampling step we double the number of feature channels, as it is prescribed in U-Net state of the art. Every step in the decoder part 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. [1]

Skip connections between the encoder and decoder side are used to pass features from the encoder path to the decoder path in order to recover spatial information lost during down sampling. The skip connections enable feature reusability and stabilize training and convergence.

We defined the number of skip connections as a parameter of our network, so as to tesk different sizes.

V. Post-processing

For the training we used 320x320 images (cropped from augmented 400x400 training images) as input. The test set contains 608x608 images, so we crop them into four overlapping 320x320 images. We perform the segmentation on each of those four images and stitch them back together by taking the mean probability of being a road per pixel for the overlapping parts. An example of this stitching can be seen in Figure 3.

[2]https://mlfromscratch.com/activation-functions-explained/
[3]https://closeheat.com/blog/elu-activation-function
[4]https://www.hardikp.com/2017/07/24/SELU-vs-RELU/

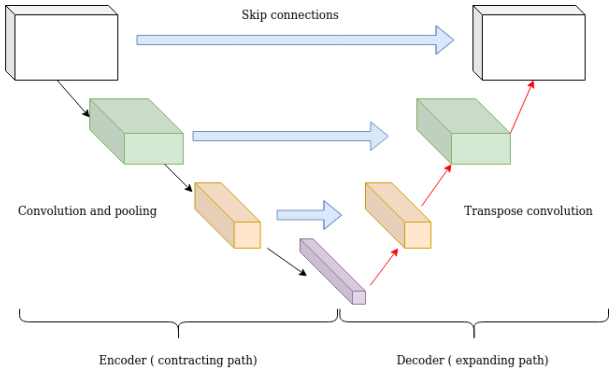

Figure 2. Rough structure of a U-Net architecture

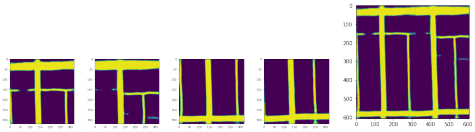

Figure 3. Reconstructed 608x608 image from 320x320 images.

VI. Experimental setup

When we were running preliminary experiments we noticed a strong tendency of overfitting. It became evident from the fact that the validation loss is significantly higher than the training loss and the difference becomes bigger with more epochs that were run. This problem still remained although we augment the input data to create more variations to trains with. We therefore applied two additional techniques to fight this problem and tested different configurations in an experimental evaluation. We first introduce the two techniques we used, namely changing the complexity and structure of the convolutional network and secondly using dropout during the training. The presentation of the results follows in the next section.

A. Network complexity

The first approach we tried was to change the complexity of the convolutional network to fit our data. We measure this complexity in terms of trainable parameters of the network and the number of layers that it has. As introduced before, the U-net has a certain number of skip-connections that connect layers of the encoder part to their pendant in the decoder part. When we introduce more layers to our network, we add layers to the encoder and decoder and connect them with a skip connection. This way, the number of skip connections can be used to express how many layers the network will have in our construction. Adding more skip connections increases the number of nodes and trainable parameters. Another way to change these numbers is to play with the number of feature maps for the convolutions. Higher feature maps result in bigger and more complex networks. For the tests, we played with the number of skip connections

and feature maps and settled for three configurations of the network: One with 3 skip-connections and a total of 483,441 trainable parameters (called low_complexity), one with 4 skip-connections and 1,944,049 trainable parameters (called mid_complexity) and a last one with 6 skip-connections and 7,785,465 trainable parameters (called high_complexity).

B. Dropout

Using dropouts is a common regularization technique that works well for a wide range of neural networks [3]. During the training, we randomly select a percentage of the nodes and don't use them and all their incident connections for one step of the training, thereby dropping them. The important parameter to set for this method is the percentage of nodes that we want to drop. Literature indicates that it can make sense to choose different values of the dropout for the input layer and the hidden layers. In this scenario, the dropout for the input layer is usually set a bit lower while the dropout for the hidden layers is higher. For general neural networks we found that using values up to 0.5, meaning that half of the nodes are randomly dropped, works well in practice [3]. However, for convolutional networks, smaller values up to 0.3 proved to be more effective [4]. To find a good choice for our convolutional U-net, we tested three different dropout configurations and trained all three networks introduced in the section before (with different complexities) with them. The configurations are:

- Dropout for input: 0, Dropout for hidden layers: 0 (to have a baseline without any dropout)
- Dropout for input: 0.1, Dropout for hidden layers: 0.15
- Dropout for input: 0.1, Dropout for hidden layers: 0.3

We call them no_dropout, mid_dropout and high_dropout.

VII. Results

To evaluate the performance of the applied methods we performed a grid search over the described configurations for dropouts and model complexities. For each run we observed the development of the loss and the f1-score over the epochs.

A. Network complexity

In Figure 4 we show the validation loss for different complexity configurations averaged over models with different dropout rates. We can see that the models with the highest complexity perform best in our experiments, followed by the ones with mid complexity. The models with the lowest complexity performed worst. The lowest average loss per group were 0.327 for the low complexity in the 97th epoch, 0.267 for the mid complexity in the 73rd epoch and 0.225 for the high complexity in the 61st epoch.

B. Dropout

In Figure 5 we show the validation loss for different dropout configurations averaged over models with different complexities. We can see that the models with no dropout

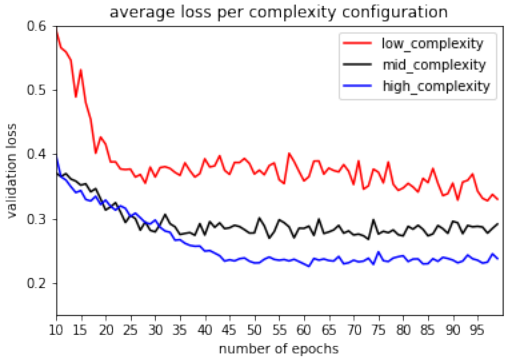

Figure 4. The average loss per complexity configuration. The average is calculated over models with different dropout rates but the same complexity.

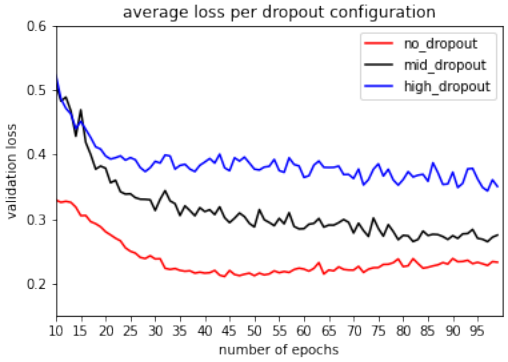

Figure 5. The average loss per dropout configuration. The average is calculated over models with different complexities but the same dropout rate.

clearly outperformed those with dropouts. The models with mid dropout also outperform those with high dropout. This evaluation shows that to train a high performing model we should have a very small dropout.

C. Best model

After our experimental evaluation it turned out that the methods we employed to fight over-fitting did not help to solve the problem. Quite the contrary, the convolutional networks with a higher complexity and no dropout performed better than the other versions. So, maybe the over-fitting is not coming from a network that is too complex or with not enough regularization, but from the fact that we do not have enough training data to work with. We tried to lessen this impact with the augmentation of the training images, but even with these methods it turned out to not be sufficient.

The model that performed best in the tests was the one with six skip connections and no dropout as displayed in Table I. To get to our final model that we submitted on AIcrowd, we used this setup and trained the model for 150 epochs on all of our training data, not using any data for the validation. It scored a F1-score of 0.876 and an accuracy

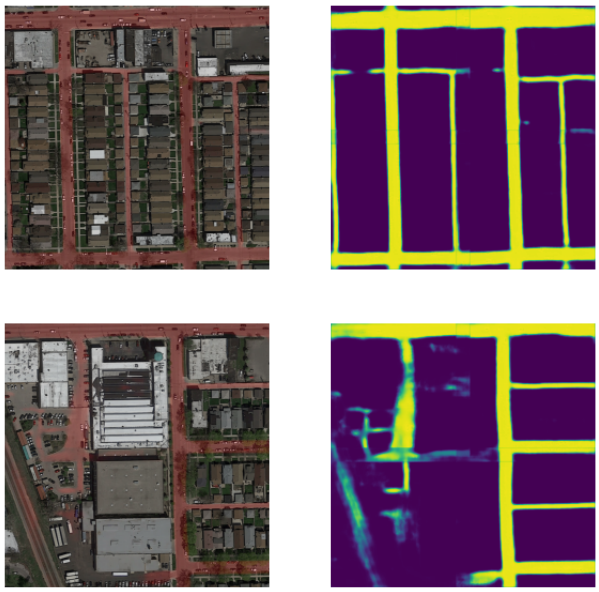

Figure 6. Predictions of our classifier, overlayed over the original image and the prediction mask on the right

	low_complexity	mid_complexity	high_complexity
no_dropout	0.82 (e72)	0.84 (e53)	**0.86 (e99)**
mid_dropout	0.78 (e99)	0.82 (e86)	0.85 (e86)
high_dropout	0.67 (e91)	0.78 (e99)	0.81 (e91)

Table I
HIGHEST F1 SCORE FOR ALL CONFIGURATIONS RUN IN THE GRID SEARCH WITH INDICATION IN WHICH EPOCH THE F1 SCORE OCCURRED.

of 0.935. To see where the problems of our predictions lie, we investigate Figure 6, which depicts two of the unlabeled images with our prediction as an overlay in red and only the prediction in the second column. In the first image we see that while the predictions look good in general, there are slight problems keeping streets connected when they are overshadowed by buildings. However, trees do not pose this problem this heavily as we can see in the second image. Another interesting observation that can be made in the second image is the classification of a parking lot as street. Even to the human eye, the parking lot looks like street and is almost indistinguishable from it without having the context information that it is used for parking. This may explain why our classifier runs into problems there. Another classification-error can be observed for the train tracks in the second image. They tend to be classified as street although they are not. This may be because they look very similar in appearance to streets.

The problems mentioned above could be lessened by training on more input, while the results already look very usable.

VIII. Discussion

In terms of complexities, we sometimes reached the limits of number of parameters that Google Colab (on GPU) allowed us to perform (it ran out of memory) and had to limit it in our grid search. The time of training for our final model is about 5 hours. The limit imposed by Google Colab is 12 hours, so we would still be able to train for more epochs.

We could gain learning time, making transfer learning, using pre-trained models for image recognition, like VGG, LeNet, ResNet or EfficientNet.

Another advantage of those pre-trained network is that they have a number of parameter really bigger than the limitation we reached with Keras on Google Colab. For example VGG19 has 143 million parameters, Inceptionv3 (GoogLeNet) has 24 million, ResNet50 has 26 million and EfficientNet-B7 has around 75 million. Letting those parameters fixed, we would only train a custom layer added to a pre-trained model.

IX. Summary

In conclusion, using a U-Net convolutional network architecture allowed us to reach a F1-score of 0.876 and an accuracy of 0.935. We made out the main problem for our training to be the lack of training data, however we tried to make the best out of the data that was provided to us. We therefore applied input augmentation and used an experimental evaluation to find good parameters. The experiments included the testing of several number of layers, feature maps values and the dropout factor. It turned out that our techniques to fight over-fitting did not result in better models, which is an interesting observation and shows once again that the given data was not sufficient.

References

[1] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, ser. LNCS, vol. 9351. Springer, 2015, pp. 234–241, (available on arXiv:1505.04597 [cs.CV]). [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a

[2] N. Adaloglou, "Intuitive explanation of skip connections in deep learning," *https://theaisummer.com/*, 2020. [Online]. Available: https://theaisummer.com/skip-connections/

[3] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[4] S. Park and N. Kwak, "Analysis on the dropout effect in convolutional neural networks," in *Asian conference on computer vision*. Springer, 2016, pp. 189–204.