Semester Project: Road Segmentation

Computational Intelligence Lab

Nicolas Küchler, Octavio Martínez, Sabina Fischlin,
group: *TheRoadSegmenters*,
Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—The road segmentation problem as given in the semester project description has the goal to label patches of 16x16 pixels as either *road* or *background*. In our solution we have trained two different models, one a *UNet* and the other a modified *SqueezeNet*, and have post-processed the obtained predictions on the test images using *conditional random fields (CRF)*. The two models chosen each were able to identify different features of a road. For our final results we have output the average of the post-processed prediction probabilities from both models. By combining the two solutions, we were able to unite their respective strengths. The use of CRF as a post-processing mechanism has successfully been able to remove many false predictions, in particular when a background pixel was falsely labelled as a *road*.**

In order to achieve more accurate results we have extended the given training data set with additional (self-labeled) images from *Google Earth* and have also run all images from the training set through an augmentation process.

We also present a SqueezeNet architecture adapted to an image segmentation problem.

I. Introduction

With their results clearly dominating over other models, neural networks have become the state of the art for image segmentation problems. Our first approach to the road segmentation challenge was therefore to implement a UNet. The UNet model was first proposed in [1] for biomedical image segmentation and is now one of the most successful [2] models for image segmentation in general.

We have found however, in our earliest test runs using only the UNet model, that while it did well on certain image areas (e.g. green areas and highways), it would perform badly on others (e.g. parking lots, roads with varying texture). What if a model with an entirely different architecture was able to perform better on the areas that the UNet was missing? In other words, we aimed to complement the results of one model with the results of another.

While deeper networks usually yield much better results their performance is also quite low. SqueezeNet was created in an attempt to use smaller networks with higher performance which could produce just as strong results as deep networks [3]. With SqueezeNet being a simpler model it is also much easier to control what the network learns. SqueezeNets are typically used for multi-label classification problems. With small adjustments to its classical architecture we were able to adapt it to the needs for the binary segmentation problem at hand.

We have also found that especially the UNet would make many false predictions (i.e. labeling a patch as *road* instead of *background*) and merely averaging over the prediction probabilities of both models would not be able to eliminate this problem. *Conditional random fields (CRF)* are used to create smoother results by maximizing the agreement on labels between similar pixels [4]. In other words, CRF have the ability to remove straying predictions.

In the next section we propose a combination of these techniques to improve the prediction precision.

II. Models and Methods

A. Data Augmentation

With the provided training data set only comprising 100 images, we opted to augment the data using the library *Augmentor* [5]. Although this would add another layer of non-determinism we decided to use online augmentation, i.e. re-augmenting the data at the beginning of every session, instead of augmenting the data only once. This has the advantage that the training data is as varying as possible. However, this meant that we did not have a fixed data set for the training of the models and therefore fixed the number of batches per epoch.

We have applied a combination of random basic transformations such as flips (horizontally and vertically), rotations (90°, 180° and 270°), random rotations by ± 25°, shears and zooms. Additionally we have applied color transformations, especially as a counter measure against the issues arising from differences in brightness and colors of roads. In particular brightness, contrast, and PCA color augmentation as described in [6] have been performed.

B. Training of the Models

For the training of both models we used a square loss function. In order to evaluate the models we set aside 10-15 images (dependent on the data set used, see *section II-E* for more information) from the training set to the validation set. These were selected according to their representation of different challenges with regards to predicting the presence of a road.

Our implementation uses *tensorflow* and employs its *Adam Optimizer*. We now describe the models and their architecture in detail.

1) UNet: A UNet consists of an encoder and a decoder — much like a normal autoencoder. What sets it apart is its U-like architecture, with the encoder and decoder part consisting of several levels. Each level contains an encoder and a decoder block (except for the first layer which includes special input and output blocks). A speciality of the UNet architecture are the skip connections between the blocks of the same level (see left side of *figure 1*). For our implementation of the UNet we have been inspired by its use in another Kaggle competition on satellite imagery feature detection, where its application was applied to a similar task [7].

The encoder starts with the input block, followed by four encoder blocks, each consisting of three runs of a sequence of batch normalization as regularization, 3x3 convolution and a ReLU activation layer, followed by a 2x2 max pool with stride 2 to half the width and height (as described in *figure 1*). After the second repetition of the sequence inside each encoder block there is a skip connection to the decoder block on the same level.

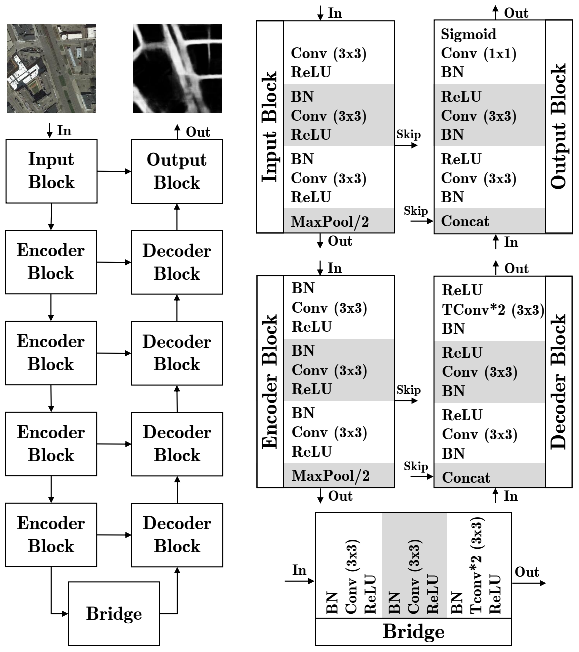

Fig. 1: UNet - Architecture
left: overall architecture, right: structure of each block

The decoder, analogously to the encoder, consists of consecutive decoder blocks, each of which first concatenates the skip connection with the input from the previous level, before three runs of a sequence of again batch normalization, 3x3 convolution and ReLU. With the exception of the last repetition which uses a 3x3 transpose convolution with stride 2 to reverse the halving of the width and height during the encoding (see *figure 1*). Throughout the network 64 filters are used except for the convolution right after the concatenation layers which uses 96 filters.

For the training we have used a patch-based approach to decrease uncertainty by extracting random patches of size 256x256 of a training image.

We have trained the UNet in 75 epochs, with a batch-size of 4 and 100 batches inside each epoch. The UNet uses an exponential learning rate, which we initially set to 0.005 with a decay of 0.95 and 1000 decay steps. For the validation we have used the same patch-size with stride 72.

2) SqueezeNet: Since the SqueezeNet was not originally designed for image segmentation we had to adjust its architecture from [3] to fit the needs of the task at hand. We used the original design (with the exclusion of the last two layers) to construct an encoder to bring the data to a smaller latent space. For the decoder we took the same structure but inversed it (see *figure 2*). This made it possible to retrieve a black and white image consisting of the prediction probabilities for each pixel (see *section II-C* for details on the output format). Inspired by the architecture of the UNet, we added skip connections between each layer implemented as additions of the outputs of one side to the other.

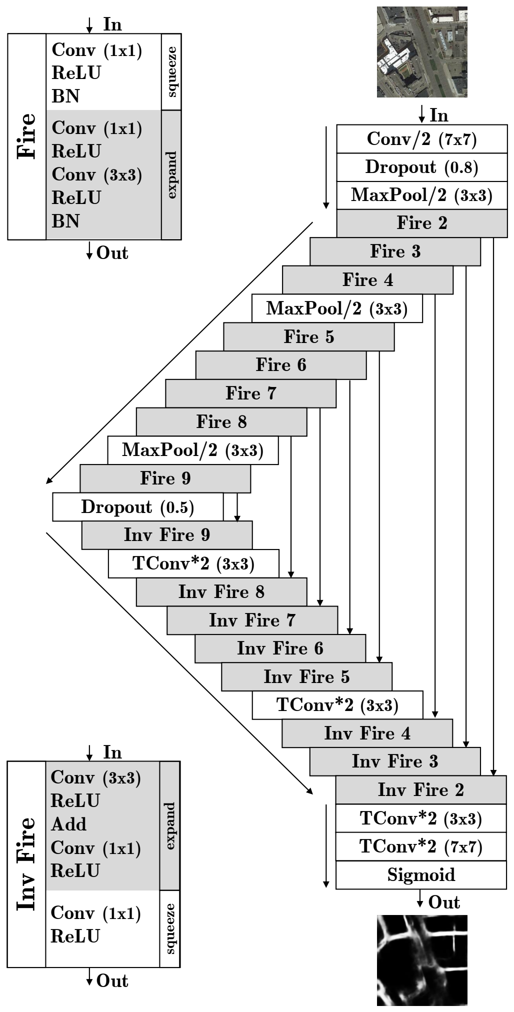

Fig. 2: SqueezeNet - Architecture

The main blocks in this model are called *fire modules*. Each fire module consists of a *squeeze* and *expand* part. We distinguish between a regular fire module and an inverse fire module, which appear in the decoder part and perform the squeeze and expand in reversed order. The squeeze part consists of a 1x1 convolution filter, ReLU and batch normalization, with the expand part consisting of a 1x1 convolution filter and

ReLU, followed by a 3x3 convolution filter another ReLU and batch normalization. The inverse fire module follows a similar structure (see *figure 2* for the structure of the respective modules).

The number of filters and the sizes correspond to the ones used in [3]. We have used a total of 8 fire modules. For the first two fire modules we begin with 16 filters for the squeeze component and 64 for the expand component. The number of filters used in the squeeze and expand modules increases by 16 and 64 every 2 fire modules. It is important to note that the same sizes are used for the inverse fire modules and correspond to the same numeration as in the fire modules.

As a further step for regularization and to increase the precision we included a *dropout layer*, which drops a channel from a pixel with a probability of 20%. This way the model is able to generalize better as parts of the image are not available.

The model was trained for 45 epochs, with 100 batches per epoch and a batch-size of 32. The used learning rate was 0.0001 and it was fixed for the entirety of the training process. As opposed to our patch-based approach for the UNet, we used the whole image for both training and validation.

C. Making Predictions

The output of both models are matrices of per pixel prediction probabilities, visualized as black and white images with white representing a road prediction of probability 1 and black probability 0 (i.e. $p_{road} = pixelencoding/255$).

For the UNet we have additionally applied an *ensemble* approach, i.e. we averaged over (0°, 90°, 180°, 270°) rotations with patch sizes 256x256 with stride 88 and the full image size 608x808.

From these per pixel predictions we have arrived at the 16x16 patch-wide prediction by calculating the mean of each patch and applying a 0.25 probability threshold to classify a patch as a road.

D. Post-processing using CRF and averaging

For the post-processing with CRF we used the inference algorithm described in [4] and provided by the library *pydensecrf* from [8], which enables efficient inference on a pixel basis as opposed to a region basis and could therefore easily be embedded in our overall architecture.

We used the prediction probabilities output by the models as the unary potential. For our results we ran the algorithm for 25 inference steps. We were able to obtain the best results by keeping the strength of the pair-wise Gaussian location potentials low. This would normally make the segmentation less smooth, but worked to our advantage in this case, since smoothness with regards to location only would remove many correct road predictions. The opposite was the case with regards to the bilateral potentials taking colors into account: by raising their strengths we were able to create smoother clusters of predictions based on colors, i.e. a stray prediction of a road on a background of a different color (e.g. a nearby grass patch) would be removed.

It is important to note that the CRF post-processing results in a strict $\{0,1\}$ per-pixel prediction matrix.

The averaging function we have implemented is a simple per pixel average of the probability of two prediction probability matrices. We have again applied the 0.25 probability threshold to classify a patch as a road.

In the case of averaging after the CRF post-processing (as has been done for our final result), it should be noted that the $\{0,1\}$ prediction output of the CRF means that the average results in a true overlay of the two solutions.

Figure 3 shows a visualization of the full computational pipeline used to arrive at the final prediction mask.

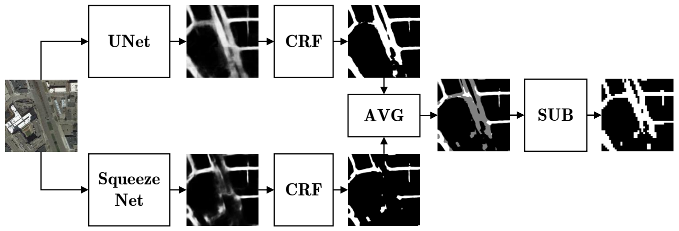

Fig. 3: Full computation pipeline

E. Dataset Extension

During several test runs of our models, we have noted that the dataset provided for the training only represents part of the test images well. Many images in the test set raise challenges which have no counterpart in the training set. Difficulties we have observed were slim roads, or roads obscured by trees and/or shadows of buildings, large groups of cars, concrete which has been patched together and has different texture. For some cases, i.e. when the image is of a green area, there were no corresponding training samples at all.

The data augmentation we applied was only able to partially erase these difficulties. To achieve more accurate predictions, we therefore decided to extend the training set with manually labeled satellite images which we obtained from *Google Earth* of the city of Berlin. We have chosen 46 images according to the challenges described above.

With this extension we were indeed able to get better results. In order to compare the different results easily we have added the choice of dataset (*default*: originally provided training set, *ext-half*: the original training set with half of the extended images, *ext-full*: the fully extended training set) as a parameter to the model training. The extended data sets can be found in the repository [9].

III. Results

All computations have been done on the *Leonhard cluster* provided by ETHZ. (In order to reproduce the results the instructions in the readme file should be followed which can be found in [9] or the supplementary files.)

The results are given in *Table I* in terms of the public and private scores assigned on the Kaggle competition site [10]. While for our final and best result we used the process as described in *figure 3*, we have also added the scores of each

Model	Training data set	Public score	Private score
Baseline 1 (linear regression)	default	0.39985	0.39115
Baseline 2 (simple conv and maxPool layers)	default	0.86640	0.85138
UNet (without CRF)	ext-full	0.87090	0.86028
SqueezeNet (without CRF)	ext-half	0.88373	0.87360
UNet (with CRF)	ext-full	0.89457	0.88433
SqueezeNet (with CRF)	ext-half	0.88809	0.87808
Prediction average of both models without CRF	(*mixed*)	0.88710	0.87912
Prediction average of both models with CRF	(*mixed*)	0.89931	0.89082

TABLE I: Private and public scores obtained on *Kaggle* showing the outputs produced by the various models and processing parameters.

individual step. For comparison two baseline solutions have also been added.

The first one uses the linear regression model with the pre-processing steps used in exercise 11 of the course.
The second baseline is a simple architecture using only convolution and max-pooling layers. It starts with two convolution layers with 32 and 64 channels correspondingly followed by a max-pooling layer. After that two blocks of convolution and max-pooling layer are applied. For the decoder part we use a convolution with 256 channels, 3 up-sampling layers which are refined by two convolution layers and a sigmoid activation.

Both baselines were trained using the default dataset provided for this project.

IV. Discussion

What becomes clear from *table I* is that all of our models, either individually or in combination, scored higher than the two baselines.

For both the UNet and the SqueezeNet predictions the CRF post-processing was shown to improve the prediction accuracy. Especially the scores of the UNet improved. This is due to the fact that the UNet has made many more false road classifications as the SqueezeNet, which shows the effectiveness of the CRF in this regard. *Figure 4* shows an example of the results of successful CRF post-processing on an image with many false predictions. While the CRF post-processing also removed some correct predictions, these were outweighed by the number of desired removals.

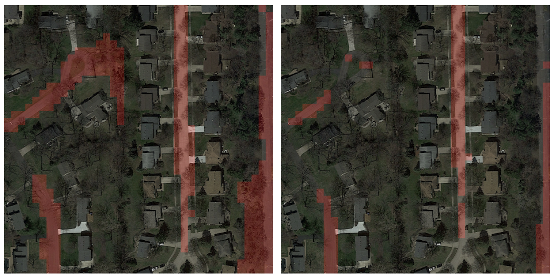
Fig. 4: Road predictions for *test_7.png* from UNet before (left) and after (right) post-processing with CRF

The UNet and SqueezeNet on their own were indeed able to predict different image areas. For many images the differences were quite significant as can be seen in *figure 5*. As hoped by combining the two solutions of the UNet and the SqueezeNet we were able to obtain better results. However, without the CRF post-processing the improvements were not as prominent.

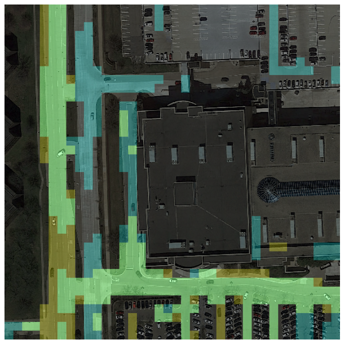
Fig. 5: Comparison of the UNet (yellow) and SqueezeNet (blue) road predictions for *test_10.png* (green: both predicted a road)

We could potentially have obtained even better results by combining more than two models. The choice of the prediction probability threshold also has proven to be quite tricky and further adjustments might also have an effect. Additionally, a modification in the post-processing pipeline order together with threshold adjustments could also increase the prediction accuracy.

Additionally it should be noted that we had initially trained the SqueezeNet model on the *ext-half* data set and for time reasons were not able to run it again on the *ext-full* as we had done for the UNet model. Doing this could have potentially yielded even better results.

V. Summary

As shown in our results, by combining the three separate techniques, the UNet model, SqueezeNet model and CRF post-processing, we were able to take advantage of their respective strengths and obtain better results than each on their own.

We were also able to design an adaptation of the SqueezeNet architecture for image segmentation.

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] Z. Zhang, Q. Liu, and Y. Wang, "Road extraction by deep residual u-net," vol. 15, no. 5, pp. 749–753, 2018, (available on https://arxiv.org/pdf/1711.10684.pdf).

[3] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *The Computing Research Repository (CoRR)*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[4] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 109–117. [Online]. Available: http://papers.nips.cc/paper/4296-efficient-inference-in-fully-connected-crfs-with-gaussian-edge-potentials.pdf

[5] "Augmentor library," https://github.com/mdbloice/Augmentor, accessed: 2018-07-05.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[7] "Dstl satellite imagery feature detection competition," https://deepsense.ai/deep-learning-for-satellite-imagery-via-image-segmentation/, accessed: 2018-07-05.

[8] "Pydensecrf library," https://github.com/lucasb-eyer/pydensecrf, accessed: 2018-07-05.

[9] N. Küchler, O. Martinez, and S. Fischlin, "Code repository on gitlab," https://gitlab.com/nicolas-kuechler/cil-road-segmentation-2018/tree/master, accessed: 2018-07-05.

[10] "Kaggle competition," https://www.kaggle.com/c/cil-road-segmentation-2018, accessed: 2018-07-05.