CS-433 Project 2: Road Segmentation

Olle Ottander, Paolo Celada and Gustav Karlbom
Department of Computer Science, EPFL Lausanne, Switzerland

***Abstract*—Image semantic segmentation is the process of assigning a label to each pixel in an image. It has applications in various different fields, from *medical images* to *object detection* and *task recognition*. Using a set of aerial images from urban areas, the aim of this project is to build a classifier able to perform such semantic segmentation, by assigning a label (road=1, background=0) to each pixel. After initial research, where multiple models were considered, a CNN network architecture called U-net was chosen, since it was the best suitable choice for the task. The U-net model is supported by preprocessing, specifically image augmentation, normalization and post-processing. The training phases considered multiple parameters and implementation choices. Once a valid model is setup, predictions on a test set are calculated and submitted to the online platform *AIcrowd* [1], which calculates both F1 score and accuracy.**

I. Introduction

With the evolvement of computer science, image classification is becoming increasingly useful. To build a model for image classification, convolutional neural networks (CNN) can be a very effective method. Furthermore, CNN:s has applications in video recognition, image segmentation and other areas. Starting with implementing a baseline model, the goal is to implement a CNN model specifically designed for the task. In this project image segmentation is used to make a classification between road and background by analyzing the image given each pixel. The segmentation does not need to be a binary classification, it can also classify the data into more than two groups. Here, a pixel which is road is labeled as one and a pixel which is background is labeled as zero.

To train the model a set of 100 satellite images is provided, each of size 400x400 pixels with corresponding ground-truth images (as shown in Figure 1). Each pixel in the grid has its own RGB value associated with a specific color. The test dataset instead consists of 50 images, without corresponding ground-truth, which is to be predicted. The size is of 608x608 pixels, and consequently they need to be re-scaled to the training size.

II. Models and methods

A. Baseline Models

As a first step in building a model, two different baseline implementation provided as examples are analyzed and studied. The first method is a classic logistic regression model, and the second is a simple convolutional neural network characterized by two sequential concatenations of convolutional layers and pooling layers with softmax cross entropy as loss function. As expected, the logistic regression model did not execute the task sufficently. The CNN model performed a bit better with an F1-score of 0.656 and accuracy of 0.780. The resulting metrics are taken predicting on the test set provided.

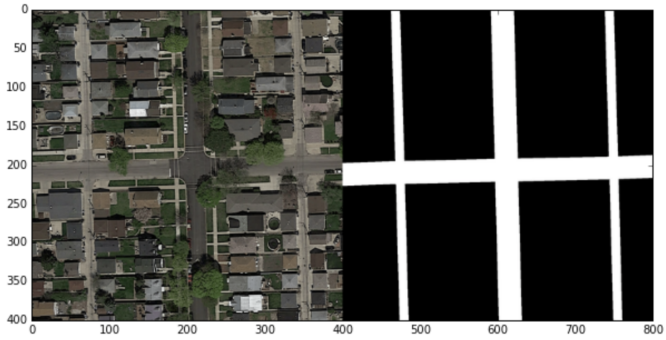

Fig. 1. Example of training data: aerial image on the left and corresponding ground-truth on the right.

B. U-net model

The model used for the road segmentation is U-net. It is a convolutional network architecture for fast and precise segmentation of images. The model was originally proposed by Olaf Ronneberger, Philipp Fisher, Thomas Brox in the paper *U-Net: Convolutional Neural Nets for Biomedical Image Segmentation* [2]. Upon introduction in 2015, it was the best performing model in a series of challenges from the IEEE International Symposium on Biomedical Imaging (ISBI), such as the segmentation of neuronal structures in electron microscopic stacks.

1) Contracting Phase: the first phase of the network consists of 4 steps aiming to gradually contract the image. More specifically, each step consists of two 3x3 convolutions with ReLU activation followed by a 2x2 max-pooling. With each of the four steps, the resolution of the image halves while the number of filters doubles. As resolution decreases, filters are able to see larger contexts, i.e. have a larger receptive field. Furthermore, the increasing number of filters enables that more complex features can be captured. The aim of this is increasing the knowledge of ”what” and decreasing that of ”where” in the image, as put by Ronneberger.

2) Expansive Phase: after the contracting phase two more 3x3 convolutions are performed, after which the expansion phase of the network begins. This phase also consists of 4 steps. In each step, first a 2x2 up-convolution is performed to double the resolution of the feature maps. Then, a concatenation with the high resolution feature map from the corresponding contraction step is performed, which allows pre-contraction information to be restored. Lastly, each step ends with two 3x3 convolutions with ReLU activation. Through each step, the image is gradually expanded to create a high

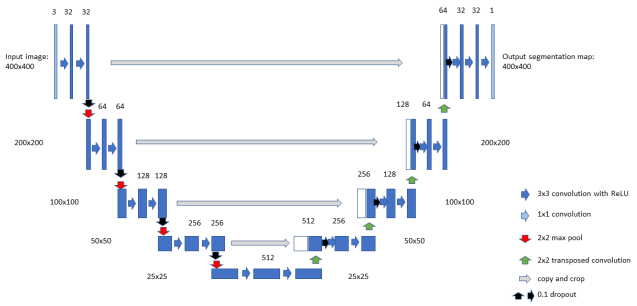

Fig. 2. The implemented U-net based model.

resolution segmentation image. In other words, this phase of the network restores the information of "where" that is lost when contracting the image, expanding back to our original image resolution. Finally, the expanding phase is followed by a 1x1 convolution which maps the feature vectors to the output class, road or not.

3) Parameters: For the up-convolution, transposed convolution is used. It is a method for up-convolution with learnable parameters. In the implemented model a dropout rate of 0.1 is used in each contraction and expansion step, with the goal of preventing overfitting. Also, the number of features was initially chosen to go from 32→512 instead of 64→1064 to increase computational speed. Further differences to the original model scheme shown in Figure 2 is that the implemented model has 3 input features, the RGB values, instead of 1. Also, as stated earlier, the output has one class in this case.

The model implemented is shown in Figure 2. Later, the number of features used, here starting at 32, was experimented with as well, as detailed in the results section.

III. Preprocessing

A. Image augmentation

The available training data consists of 100 aerial images with corresponding ground truth. Neural networks are known to have better predictive capabilities when trained on a large set of training data. Thus, data augmentation is one possible solution to deal with this problem. To be more precise, image data augmentation is used to artificially expand the training set available, by adding slightly modified copies of already existing samples. Doing this allows the model to train on a bigger data set, and consequently it achieves higher accuracy. Multiple transformations can be applied, from random rotation to horizontal/vertical flips, random brightness change, horizontal/vertical shifts, random zoom variation and many more. The different transformations are displayed in Figure 3

Among the transformations shown, only a subset are used for the actual model. Careful analysis of the target, predicting test images, lead to discarding brightness and zoom variations.

Fig. 3. Example of different data augmentation transformations.

The images always present a fixed light and have been taken by a fixed height, and as such those transformations would have only tricked the model forcing it to learn misleading information.

Applying random rotation and shift imply dealing with empty corners or sides. Two possible techniques can be applied to remedy this: *nearest*, in which the nearest pixels color is propagated, and *fixed*, in which a fixed predefined value is assigned. Results for both cases are presented in the result section.

The initial implemented data augmentation generated an addition of about 1200 new images. Among these, 300 were generated by random rotations, 300 by random flips and 300 by random zoom adjustments. When testing the augmented images on a working model, it was found that the augmentation caused unnecessary repetitive segmentation of the same images. Furthermore, it was also seen that the model was not learning enough about askew images. To solve the first problem duplicates were removed, while for the second the number of rotated augmented images increased. Therefore, for every single image in the training set the following are added:

- **3 images** representing all possible combinations of both **horizontal and vertical flips**. By setting the correct seed value in the generator, image duplicates are avoided;
- **12 images** generated mainly by **rotation**, with horizontal and vertical flip to increase randomness;
- **7 images** generated by **horizontal and vertical shifts**, again with the support or random flips to increase diverseness.

Hence, the total number of training images is exactly 2300.

It is important to notice that a higher number of training images would have improved the model performance. However, given the scarce GPU availability on *Google Colab*, it

would have increased the training time drastically. *Google Colab* provides only *Nvidia Tesla K80s*, *T4s*, and *P100* GPU's, with a limited lifetime. All image data augmentation is done through *Keras API*.

IV. Model Training

This is the core of the project. Model training is critical to achieve optimal performance, and it is where hyper-parameters and model configurations must be chosen carefully. Results of different configurations are shown in the results section.

A. Optimizer

An optimization algorithm is used to change attributes in the neural network, such as weights and learning rate. These attributes are all finalized towards reducing the overall loss and improve the accuracy.
In this project three different optimizer are tested:

- a classic SGD;
- RMSprop, first proposed by Geoff Hinton [3];
- Adam optimizer, an adaptive learning-method algorithm well known for image segmentation [4].

The best F1 score and accuracy is found using the Adam optimizer, with a learning rate starting at e^{-3} and which decreases by a factor 10 if the loss does not improve in two concurrent epochs.

B. Loss

The loss is a very import component in a Neural Network. Since it measures the difference between the prediction and the actual value, its gradients can be used to update the weights properly. As it is such a crucial component, lots of research have been conducted regarding the matter, and even specifically for image segmentation [5]. After careful consideration, the number came down to three. For all following losses, $\hat{y}$ is the predicted value while y the true value.

The first one is *Mean squared logarithmic error* measures the average difference between the log-transformed true and predicted value. Naturally the error can range from 0 to ∞. A downside with the means squared error is that a cost occur if the desired value is not obtained and the loss is symmetric around this target value. It is defined as:

$$L_{MSL}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(\log(y_i + 1) - \log(\hat{y}_i + 1))^2$$

The second one is *Binary Cross-Entropy* which was proposed in the original U-net paper and is widely used for classification objectives. It uses the *sigmoid* function to assign a label to each pixel. Here a cost always occurs. However, the loss is asymmetric and becomes smaller the further we are to the right and the other way around. It is defined as follows:

$$L_{BCE}(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

The last one used is *Focall Loss* [6] which is a version of binary cross-entropy. The difference is that *focal loss* applies a modulating term to the cross entropy loss to put higher weight in learning from hard negative examples. It is dynamically scaled and the scaling goes to zero as the prediction goes towards one. It is defined as

$$L_{Focal}(p_t) = -\alpha_t(1 - p_t)^\gamma \log p_t$$

where

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise} \end{cases}$$

and *p* is the model probability of road label, estimated by the model.

C. Validation

In order to evaluate the predictive performances of different models, a validation set extracted from the training set ($\sim 10\%$) is used while training the model. Validation loss is used by *Keras* to perform early stopping if no improvement is detected.

V. Submission Technique

Each model developed during the project is trained considering as, input a 400x400 pixel image. The images provided for testing and submitting predictions to the online platform instead are of size 608x608 pixels. To deal with this variation, the test image is cropped in 4 sub-images of each 400x400 pixels (considering the biggest image available starting from each corner). Then, each group of 4 is submitted to the model and the final prediction is reconstructed using each group outputs. This method allowed the use of the original model.

VI. Post Processing

At this point, a working model is available. Considering that improving it by increasing the number of training data is not feasible, a different approach is considered and tested in practice. Starting from a predicted mask as in Figure 4, it's clearly noticeable that occasionally where the presence of a road is evident (ie. a straight sequence of pixels going edge-to-edge), the model wrongfully miss some pixels. To tackle this problem a simple algorithm is developed, which consider the known fact that roads mostly delimit clear shapes. The 400x400 px image is divided in 25x25 patches of 12x12 px (where each patch has the same label). The algorithm then simply scans the image for all patches, and changes label of a *background* patch to *road* if it's vertically or horizontally in-between 2 *road* patches. More easily, it fills subsequent "holes" of max. length 2. Figure 4.

VII. Results

The results for 10 different set-ups are presented in Table I. First, three different losses were compared, where it was concluded that MSL performed the best.

Second, the threshold for assigning a patch as a road was evaluated by comparing 25% and 50%. Here it was concluded

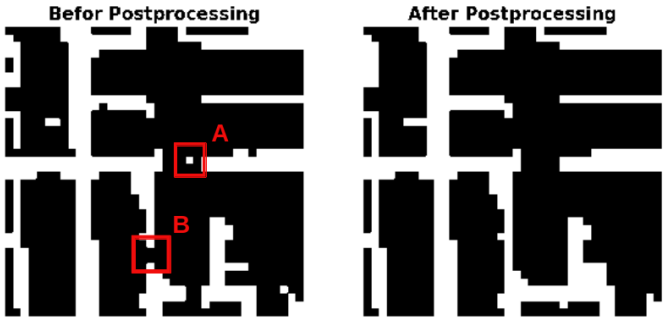

Fig. 4. Example of post processing conditions.

Loss	Threshold	Features	Post-process	F1-score	Accuracy
Focal	0.5	32		0.586	0.841
BCE	0.5	32		0.658	0.724
MSL	0.5	32		0.863	0.922
MSL	**0.25**	32		0.821	0.888
MSL	0.5	**64**		0.862	0.923
MSL	0.5	**16**		0.868	0.926
MSL	0.5	**8**		0.876	0.930
MSL	0.5	**4**		0.849	0.912
MSL	0.5	8	**Remove+fill**	0.862	0.920
MSL	0.5	8	**Remove**	0.876	0.930

TABLE I
RESULTS OF THE SIX DIFFERENT SET UPS.

that having a lower threshold worsened the score due to more false positives.

Third, the number of features used throughout the contracting and expanding phase was explored. $32 \rightarrow 512$ was the original choice used, inspired by the original U-net model with $64 \rightarrow 1024$ parameters, but halved to increase computational speed. The original U-net set-up of $64 \rightarrow 1024$ was also tested, but this performed worse than the reduced model. Therefore, further reductions were explored, concluding with optimal number of parameters being $8 \rightarrow 128$.

Lastly, post-processing was evaluated. First, both filling and removing road patches was conducted, but this instead decreased the performance of the predictions. Then, only the removing part was conducted, which by inspection did improve some selected predictions, but not enough to improve the score of the predictions.

To conclude, the best performing set up was U-net inspired model with number of features ranging from $8 \rightarrow 128$, using the MSE loss function, with a patch classification threshold of 0.5, and optionally with a post-processing procedure of removing isolated road patches. This set-up obtained a F1-score of 0.876 and an accuracy of 0.930.

VIII. Discussion

From the results section, it is concluded that MSL was the best performing loss function of those evaluated. This goes against the authors prior expectations, as it is the simplest loss function evaluated, and also the one least adapted to classification tasks. The conclusion we can draw is that simpler can be better. A second conclusion from the observed results is that a sufficiently large threshold for patch classification must be chosen to prevent too many false positives. For this project, 50% was deemed sufficient.

It was also observed that decreasing the number of features used in the U-net model not only improved computational speed, it also improved F1-score and accuracy of the predictions up to a certain point. The number of features were allowed to decrease from the $64 \rightarrow 1024$ in the original U-net to $8 \rightarrow 128$ before the predictions score did not improve further. This reduction improved F1-score from 0.862 to 0.876 and computational time from 2.3 hours to 0.3 hours. Our explanation for the performance improvement is that too many parameters resulted in overfitting. The original U-net model was developed for segmentation of rather complex structures such as neuronal structures in electron microscopic stacks, which may be deemed as relatively complex compared to satellite images of roads. Thus, the conclusion is that for this project, the large number of parameters in the original U-net model is not only necessary but also counterproductive.

Lastly, the model could most likely also have been improved further by using a larger dataset, which would have given the model more data to train on. Cross validation could have also been used to further fine tune the hyperparameters. Due to lack of time and the scope of the project the model was unfortunately not developed further.

IX. Summary

By first training the model and then using it on the test data it can be concluded that the U-net achieves a sufficient result in the image segmentation task provided. This is dependent on image augmentation, a proper loss function, post processing and fine tuned hyper parameters. The best F1-score generated by the model was 0.876 and was achieved using the setup MSL loss, with a patch classification threshold of 0.5, and number of parameters in the U-net network going from $8 \rightarrow 128$, with post-processing not needed.

References

[1] M. EPFL. (2021) Class project 2 — road segmentation. [Online]. Available: https://www.aicrowd.com/challenges/epfl-ml-road-segmentation/

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015. [Online]. Available: https://arxiv.org/abs/1505.04597

[3] G. H. Tijmen Tieleman, "Divide the gradient by a running average of its recent magnitude," 2012. [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[5] S. Jadon, "A survey of loss functions for semantic segmentation," *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, Oct 2020. [Online]. Available: http://dx.doi.org/10.1109/CIBCB48159.2020.9277638

[6] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," 2018. [Online]. Available: https://arxiv.org/abs/1708.02002v2