Road extraction from satellite images

Marijana Peti, Loïc Wisniewski, Christophe Minutolo
Department of Computer Science, EPFL, Switzerland

***Abstract*—Image processing is an important field in machine learning that allows many tasks to be automated. Diseases identification, self-driving cars or classifying cats or dogs, the application domain of such algorithms are various. In this project we use some of these methods to identify roads in satellite images. To do so, we use convolutional neural network, one of the best methods to have satisfying results in segmentation problems. Used u-net model is described and the way predictions were improved.**

I. Introduction

This project is a student project for the machine learning course at the EPFL in 2018. It tries to answer the image segmentation problem that is one of the project proposed for the second half of the semester.

Image segmentation is an image processing operation that attempts to collect pixels among them according to predefined criteria. The pixels are divided into regions, which form a paving or partition of the image. The aim of this project is to build a model that is able to perform the segmentation of satellite images from Google Maps. In our case, the segmentation consists in detecting which parts of the images are roads, and which parts are background (e.g. buildings, fields, water ...).

This paper presents different methods to implement a classifier that has to detect roads and background.

II. Data exploration

As a first step, data exploration is important to produce machine learning models. It helps to find anomaly in data that could reduce performance of the machine learning algorithm. The dataset we used is available on crowdAI [1]. Here are grouped the noticeable information on the given data:

- The training dataset is composed of 100 satellite images of 400x400 pixels. They represent urban areas and the corresponding ground truth masks that indicates where the road is in the original picture. White pixels represent roads (foreground) and black pixels represent the rest (background).
- The test dataset is composed of 50 satellite images of 608x608 pixels.The task is to classify satellite image. The label associated with each block corresponds to 1, 0 otherwise.

By taking a look at the images of the dataset we can see how it could be hard in certain case to know if it's a road or background. For instance the special case of a parking that is not always labelled as a road or some roads are covered by trees. Our model should analyze these cases very carefully to avoid mistakes.

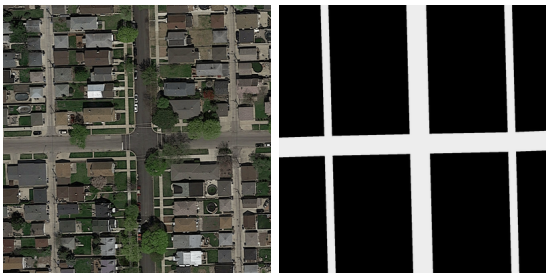


Fig. 1: Example of satellite image and ground truth

III. models

We have decided to use U-Net Model because it performs better with segmentation tasks [2]. The main qualities of this model are:

- It can localize well to provide high resolution segmentation masks.
- It works well with a small number of images (as in our case).
- Relatively robust against over-fitting.

A. Build the model

The U-Net model is built with an encoder and decoder functions. The encoder portion is composed by Convolution, Batch Normalization, and ReLU operations followed by a Max Pooling. Each Max Pooling will reduce the spatial resolution of our feature map by a factor of 2. The Decoder portion is composed of Convolution, Batch Normalization, and ReLU.

Figure 2 shows the complete structure of the proposed neural network, which is the result of various experiments. In the convolutional layer, we define filters of size $N * N$ that slide over the image and we compute the dot product (between the entries of the filter and the input) in order to create our "feature map", a 2-dimensional activation map of that filter. The CNN will learn the value of each filter but we have to give parameters (filter size, number of filters). We use ReLU for the activation function which removes negative values from an activation map by setting them to zero. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer. After that, we use pooling in order to reduce dimensionality of our feature map while maintaining the most important information. The main asset of pooling is that it reduces the number of parameters while controlling overfitting. Note that pooling is used to each feature map separately. In our case, images used for training have shape of 400*400. We use pooling layer with filters of size $2 * 2$ to

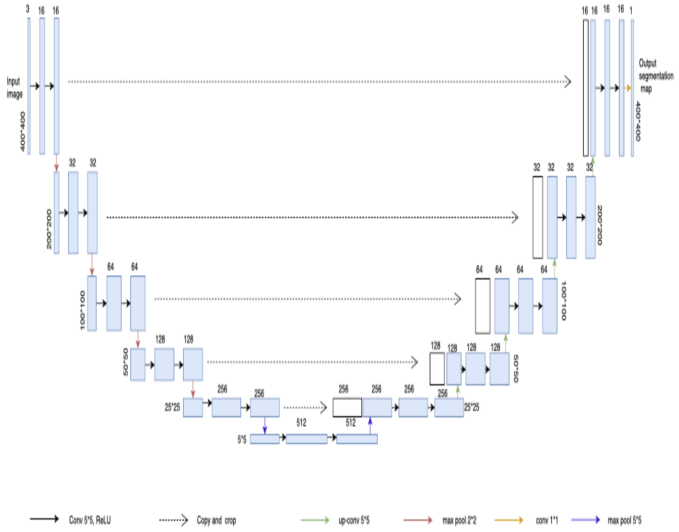


Fig. 2: Neural network architecture

have $200 * 200$ then $100 * 100$, $50 * 50$ and $25 * 25$. At this step , we cannot use anymore this filter so we decide to use a pooling layer with filters of size $5 * 5$ (smaller divisor).

B. Custom metrics

We decided to use custom metrics to keep track of the performance of our model:

- Dice coefficient: Metric that measures overlap.
- Accuracy: Metric that gives a measure of right predicted pixels.
- F1 Score: Metric that combine recall and precision. (Evaluation metric)

C. Baseline model

In order to evaluate our model, we have to define a baseline model that will be used as a comparison for the problem of classification. Our baseline model consists of using a CNN without data augmentation taken from this tutorial of image segmentation with Keras [3]. For this model we use the following parameters:

TABLE I: Baseline models

Parameters	model
Epochs	50
Convolution size	3
Numbers of filters	[32, 64, 128, 256, 512, 1024]
Batch size	2

For this model, our f1 score was 0.8007. Note that we had to adapt the model in order to apply it on our dataset (image with size 400*400).

IV. Hyperparameters optimization

A. Convolution size

Convolution neural networks are based on convolution layers. Theses layers have an important parameter that is the convolution size. It changes the size of the patch applied by the convolution layers. It means that each pixel of the next filters would gather data of $N * N$ pixel, N is the size of the convolution layers. See Table II for the results.

B. Loss function

The lost function is an important choice in a model. Different loss function can be used for segmentations task. Dice loss is one of the most common one [4]. BCE loss is also another loss function that can be considered [5]. We decided to compare them and their impact on F1 score. Moreover, we decided to use a custom loss function that would combine both of the previous loss function.

$$\mathcal{L}_{bce\_dice} = \mathcal{L}_{dice} + \mathcal{L}_{bce}$$

See Table III for the results.

C. Optimizer

Thanks to the optimizer, we get to the optimal weights for our model. On these model several optimizer were tested with default parameters: sgd, adam and adagrad. In Tab V it is possible to see the results.

D. Number of filter

If we take again our baseline model but we decide to use a different number of filters : [32, 64, 128, 256, 512, 1024], we notice that our f1 score is improved. Which means that if we take less filter, it simplifies the model because we have fewer parameters. The model is less complex, and easier to train. So we get a better model.

V. Data augmentation

The key to build a good model is big amount of useful data. Since we are only given 100 pictures, we had to made it bigger. First idea was to find similar data set on the internet, but similar one couldn't be found. Then we tried to apply simple transformations rotations by 90, 180, 270 degrees and after horizontal flips were applied on each set so the number of pictures for training was enlarged 6 times.

Next we added rotations by 45 and 135 degrees and their flipped versions. So this additionally increased the size of data 4 times. Rotating pictures by 45 and 135 causes black borders in it, which we could treat as background, but we used mirroring technique, and by doing that we achieved continuous roads in the picture. The steps of how full rotation was done can be seen on 3. First vertically flipped images were concatenated on the top and bottom of image, then on horizontally flipped images were concatenated zero matrices. After that the whole picture was rotated, it was cropped from the coordinates of width and height of the original image to size of test image. All this was also applied on the ground truth and it can be seen that it has continuous roads.

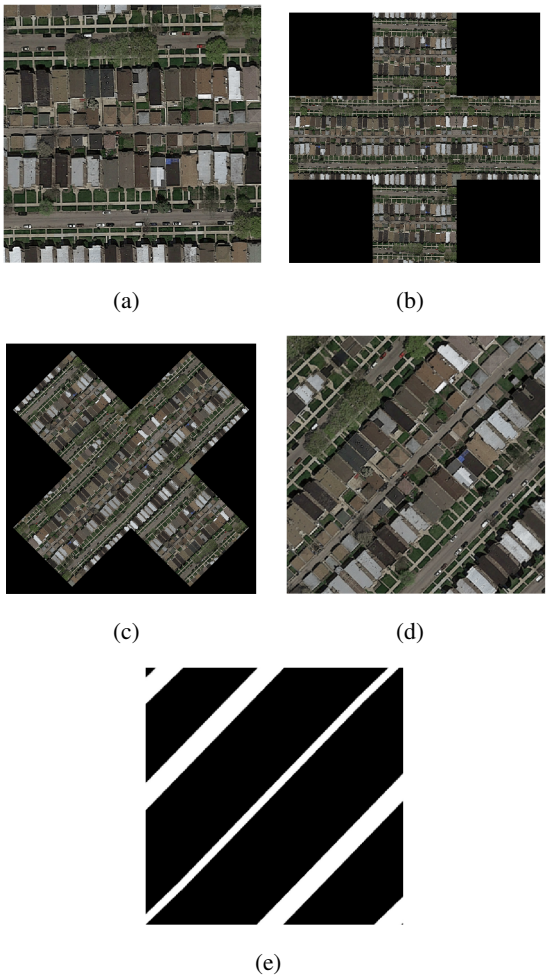


Fig. 3: Steps of rotation: (a) Image from training dataset, (b) original image concatenated with flipped versions and black background,(c) Concatenated picture rotated by 45 degrees, (d) Final rotated image, (e) Ground truth after rotation

VI. PREDICTIONS

Convolutional neural networks are sensitive to image size due to the convolutions layers that reduce the size of the image through the step in the network. In classical UNet, input image have to be of a square, with a size that is dividable by 16. As our neural network is adapted to 400x400 images we had to find out how to predict the 608x608 image that compose our test data set. Moreover, our network returns for each pixel a probability that this pixel would be a road whereas the required file for submission is a CSV file that contains, for each patch of 16x16 that compose the image, a boolean that indicates whether or not it is mainly a road.

A. Prediction on bigger images

To predict all pixels of the image without losing precision due to a potential down-scaling, we decided to predict 4 parts of the image for each test image and then combine them to create the final prediction. The overlapping pixels are replaced by the next patch applied. Patches are apply in the order presented on the Fig.4

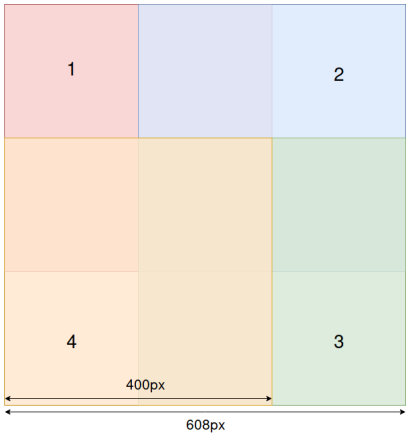


Fig. 4: Prediction of big images

B. Generation of the submission file

As the submission file does not correspond to the output of our composed prediction, we have to adapt our results. We have to convert probabilities for each pixels into boolean. For each patch we compute the mean of pixels thanks a threshold, we obtain 1 or 0 for each patch. This threshold (0.25) was chosen empirically by comparing real groundtruth to the patched groundtruth.

VII. RESULTS

Results of the tests we have done to find our best model are presented here. The final model is also available on the github repository. [6]

A. Convolution size

Here are the results for different convolution size with adam optimizer, a batch size of 2, for 50 epochs:

TABLE II: Comparison between convolution sizes

Convolution size	F1 Score
2	0.78012
3	0.80317
5	0.83162

We didn't try bigger convolution sizes as the memory required to run such models explode by increasing this parameter. In this model, bigger convolution size seems to performs better. It can be explained by the number of information needed to know if it's a road or not. There is often some noise like trees or cars that would need border information to predicate the good label.

B. Loss function

Here are the results for different loss functions. This results used the adam optimizer, a batch size of 2, for 30 epochs with a convolution size of 5:

TABLE III: Comparison among loss functions

Loss function	F1 Score
BCE & Dice	0.77888
BCE	0.7715
Dice	0.75384

C. Optimizer

Here are the results for different optimizer. This results used the BCE & Dice loss, a batch size of 2, for 60 epochs with a convolution size of 5:

TABLE IV: Comparison among optimizer

Optimizer	F1 Score
Adam	0.83162
SGD	0.764
Adagrad	0.769

D. Number of filters

Results for different number of filters. This results used the BCE & Dice loss, a batch size of 2, for 60 epochs with a convolution size of 5 and adam optimizer:

TABLE V: Comparison among different number of filters

Number of filters	F1 Score
32, 64, 128, 256,512	0.83162
16, 32, 64, 128, 256	0.877

E. Data Augmentation and the best model

Here are given the results before and after data augmentation.

TABLE VI: Comparison among models with different ways of data augmentation, rot = rotation, hf = horizontal flip, F1 score is score obtained on Crowd AI

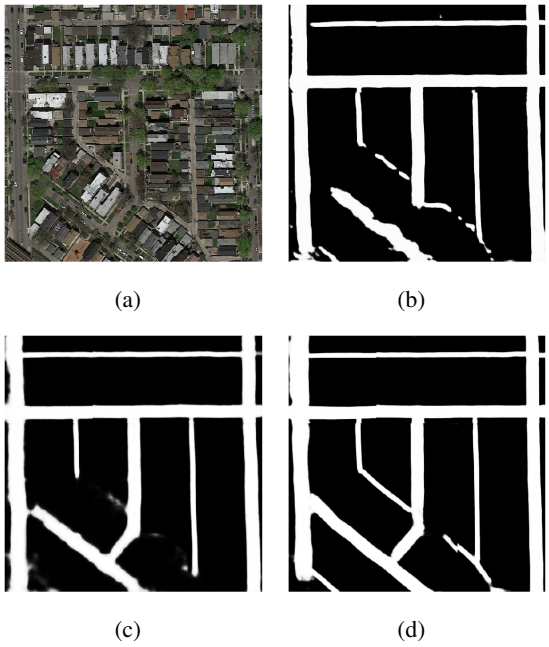
Model	F1 Score
without data augmentation	0.84
rot 90, 180, 270 and hf	0.88
rot by 90, 180, 270, 45 and 135 degrees and hf	0.897

Comparing Fig.5b and Fig.5c it can be seen that after data augmentation roads seem to be better connected, and on Fig,5d our model started to recognize diagonal streets with better accuracy, which makes sense, since we added rotated images, therefore more diagonal streets. Also the edges of roads seem to be more distinguished than in previous models.

Also since we used hyper-parameters for which we've got the best results and data augmentation our final F1 score was 0.897.

VIII. Other researches

In the previous section we've presented the techniques we succeed to make works but a part of researches that did not improved the results are also interesting. We tried to use transfer learning. This method stands on the observation that the first layers of a neural networks tends to learn general patterns while last layers learn dataset specifics patterns. It appeared that transfer learning would improve results of UNet [7].



(a) (b) (c) (d)

Fig. 5: Results: (a) Image from testing dataset, (b) prediction without data augmentation,(c) Prediction with basic data augmentation, (d) Prediction with basic data augmentation and rotations by 45 and 135 degrees and flips

Based on this researched we tried to implement Ternaus net [7] and LinkNet [8] thanks the pre-trained models included in Keras. A draft of their implementation is provided in our github repository. We didn't get good results with this models, but here are reasons that could explained the poor performance and would give ideas of later exploration:

- The preprocess functions we used are not the same as the pretrained model used
- Pretrained weights does not suit our use case
- We didn't succeed to find good hyper parameters to fine tune our model
- Problems in implementations of the top layers

IX. Discussion

In conclusion the performance of our final model is satisfying. Convolutional networks achieve the target in this problem of classification.Choosing the right hyperparameters will improved the model but the biggest improvement was achieved by increasing the data size, which again has shown that the big amount of data is the key to get the best model.

X. Summary

Machine learning can help to accomplish complex but repetitive tasks as road segmentation that can be useful for autonomous cars or to generate maps. This reports contains steps to follow to create a preforming model in such cases. Results of our best model are exposed and explanation of each our choices are explained here.

REFERENCES

[1] "Project 2: Road extraction from satellite images," https://www.crowdai.org/challenges/epfl-ml-road-segmentation/leaderboards, 2018.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[3] "Image Segmentation tutorial Keras," https://github.com/tensorflow/models/blob/master/samples/outreach/blogs/segmentation_blogpost/image_segmentation.ipynb, 2018.

[4] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation," in *3D Vision (3DV), 2016 Fourth International Conference on*. IEEE, 2016, pp. 565–571.

[5] P. Luc, C. Couprie, S. Chintala, and J. Verbeek, "Semantic segmentation using adversarial networks," *arXiv preprint arXiv:1611.08408*, 2016.

[6] "Ml world cup2 github," https://github.com/lpwisniewski/ml_worldcup_2, accessed: 2018-12-20.

[7] V. Iglovikov and A. Shvets, "Ternausnet: U-net with vgg11 encoder pre-trained on imagenet for image segmentation," *arXiv preprint arXiv:1801.05746*, 2018.

[8] A. Chaurasia and E. Culurciello, "Linknet: Exploiting encoder representations for efficient semantic segmentation," in *Visual Communications and Image Processing (VCIP), 2017 IEEE*. IEEE, 2017, pp. 1–4.