Report project 2: Road Segmentation

Gregoire Clement, Charles Gallay, Thomas Batschelet
Department of Computer Science, EPFL Lausanne, Switzerland

***Abstract*—Image segmentation is a process, whose implementation consists in partitioning an image into sets of pixels in order to be able to distinguish elements of interest. This challenge can be tackled with the help of some machine learning techniques such as CNN (convolutional neural network). In this report, we present a strategy which is based on the well known U-net architecture [1] on which we build a more adapted pipeline allowing us to achieve high performance in predicting roads in a satellite image.**

I. Introduction

When it comes to the domain of computer vision and more specifically on challenges like image classification or image segmentation, Convolutional Neural Networks (CNN) have proven to be highly efficient. They can achieve decision making performance close to a human and even achieve better results on some particular tasks.

Given a satellite image (e.g. from Google Map), the task of identifying roads can be seen as a pixel-wise classification problem where our goal is to label each pixel in the image as either background or road.

This problem is a subclass of the well-known topic called Image Segmentation, which is not restricted to binary classification, we can also classify each pixel with a particular label among more than one class.

This technique is extremely useful when implementing systems that are trained in detecting objects in a scene e.g., autonomous cars that have to be aware of pedestrians or infrastructures on the road.

Therefore, for this project, we choose to implement a model based on a CNN. We have, at disposition, a set of 100 images to train a model, each of size 400x400 pixels with their corresponding ground truth images. On the other hand, the test set is composed of 50 images, but this time, of a different size, namely 608x608 pixels. For each 16x16 patch of the test images, we have to make a prediction of being either foreground (road) or background. For a patch to be considered as a road, at least 25% of the pixels should really be roads pixels.

The difference in size between the train and the test sets imposes us some constraints. To deal with it, we experiment with two different techniques discussed later.

II. Models

A. Baseline model

In this section, we describe what is our baseline model. We break the problem into a much simpler one. The idea is to classify each 16x16 patch as either foreground (road) or background. This classification is performed by a CNN consisting of 2 convolutional layers and 2 max-pooling layers followed then by 2 fully connected layers. To train our model we split the images into non-overlapping patches of size 16x16 and assign them a prediction. To construct a full prediction, we predict for each patch and concatenate the results.

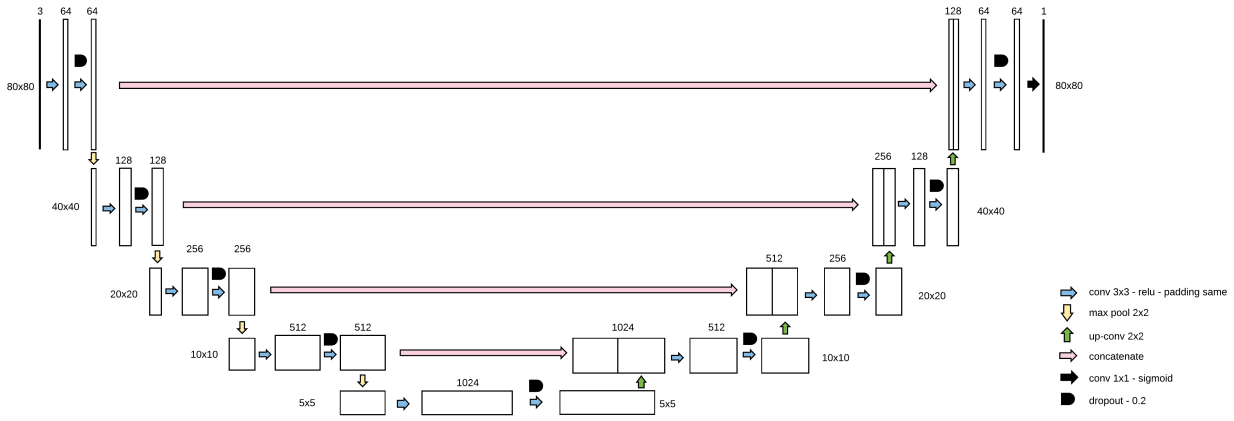

Figure 1. U-net based model

B. U-Net based model

The model we implemented is a slight modification (mainly for implementation simplicity) of U-net. It is a kind of Fully Convolutional Neural Network (FCNN), which means it has the property of taking any size of input. As a matter of fact, one can easily convince itself that if the network is only composed of convolutional layers, the size of the output is directly tied to the size of the input.

The main difference between our network and U-net is how we apply our convolution. In the paper, they use a VALID padding while our implementation uses a SAME padding strategy. The advantage of such an approach is that the size of the predicted image is exactly the same as the input. On the other hand, we have to accept some errors on the image borders, as when computing the convolution the image is extended with blacks pixels.

To help our network to generalize we add some Dropout layers after each block (we define a block as being a succession of two convolutional layers and a max pooling operation) (Figure 1) for where to put dropout layers.

We built two versions of this model, one with filters going from 32 to 512 (as we go deeper), and one going from 64 to 1024. The former having the advantage of being faster (3x) and the latter of having better results (Table I).

C. Model optimization

1) Data Augmentation: Some argue that CNN are translation invariant due to the sliding technique and/or max-pooling layers. While it is not yet well-established why they perform reasonably well with translation, they fail at being highly efficient in detecting rotation, scaling, distortion, etc. In fact, such a transformation on a pattern would need to be (re-)learned independently from the ones previously seen during training.

When looking into our training set, we realize that roads are mainly depicted by vertical and horizontal shapes but there are less diagonal roads. As a result, when training our model using only those images, we obtain good results for vertical and horizontal roads, but as expected, our model performs poorly at detecting diagonal shaped roads (Figure 2).

To overcome this, we need to give our model the opportunity to learn new filters. This is our motivation in performing data augmentation on the training set. We enrich it with images on which we applied rotations which increases our data set and gives our network the opportunity to learn diagonals.

Another augmentation we perform is dividing our image into overlapping patches. We chose a patch of size 80x80 that we slid by a stride of 16. On an image of size 400x400 that creates 441 patches. By doing so it helps our network to gain better translation invariance [2].

Not only these augmentations will help our model to detect diagonal shaped roads but interestingly will also help with the overfitting issues [3].

2) Dropout: Because the fully connected layers takes up all parameters, overfitting becomes a real issue. A solution consists in selecting only a subset of nodes from the previous layer by using a technique called dropout [4]. It reduces the network by dropping each individual node with a certain probability, which is in our case 0.2 as we do not want to lose too much information. Not only does it prevent overfitting but it also reduces significantly the training process time.

3) Early stopping: Another method to prevent overfitting is to simply not allow the network to train too long. In order to find the optimal time to stop, we start by splitting the labelled dataset into a training and validation set (80%-20% randomly chosen images with their corresponding ground truths). The motivation is to use a metric that warns us

Figure 2. Without and with data augmentation on test image 3

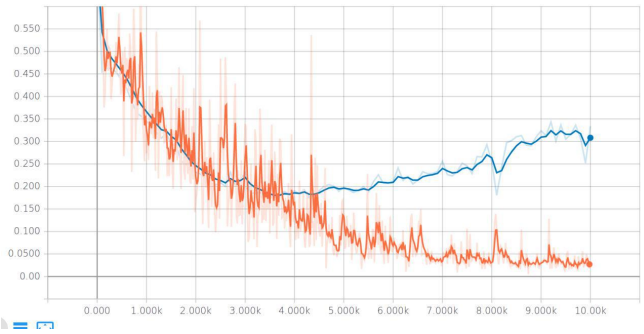

Figure 3. Loss on validation set in blue and train set in orange (graph performed using Tensorboard)

when our network starts over-fitting. During the training of the network, we constantly compute the loss on the validation set and directly stop the training process when it starts increasing. This technique is known in the literature as early stopping (Figure 3).

4) Size invariant model: To be able to make a prediction for any input sized images, we implement two solutions. For the first one, we make use of a FCNN. In fact, when faced with a convolutional neural network, we can always change it into an FCNN with a trick [5] on a dense layer (fully connected layer). The FCNN, then, allows us to take as input, for instance, our 608x608 image and predict its roads (ground truth) as a 608x608 image.

The other way to go consists in predicting, for a given set of fixed size patches their corresponding ground truth images. The overall method will patchify an image by sliding a fixed sized frame resulting in a chunk of patches. Depending on the chosen sliding size, partially overlapping images may be included. For each patch, the model will then predict the ground truth and based on them, reconstruct a final prediction.

5) Activation function: We used the rectified linear units (ReLUs) activation function as it is a state of the art technique for deep learning neural networks. ReLUs improves those by speeding up training as the gradient computation is very simple. The function has output 0 if the input is less than 0, and raw output otherwise. Moreover it reduced likelihood of the gradient to vanish.

6) Loss function: The objective function we chose to minimize for this problem is binary cross entropy on each pixel.

$$\ell(y, \hat{y}) = -\hat{y} * log(y) - (1 - \hat{y}) * log(1 - y)$$

which penalizes miss classification on pixel. $\hat{y} \in \{0, 1\}$ is the correct labelling will $y \in [0, 1]$ is the prediction of the network.

In order to find the optimal weights for our model we chose the Adam optimizer which has the advantage of typically requiring little tuning.

III. Pipeline

A. Preprocessing

1) CLAHE & Gamma correction: In order to help our model reducing the number of features that it has to learn, we do some preprocessing on our images before feeding them to the model. This step sets itself apart from data augmentation, as the preprocessing mechanism performed on the training set must be identically applied to the test set before being fed to the network.

We apply two preprocessing methods[6]; Contrast-limiting adaptive histogram equalization (Figure 4) and Gamma correction also know as Gamma adjustment (GA). CLAHE mainly allows us to improve the contrast, which consequently helps the network to detect more easily

Figure 4. Without and with Gamma and CLAHE enhanced images

boundaries. This is essential when detecting shape as roads as it will naturally learn to identify more easily the road's edges. GA, on the other hand, works on the luminance of the image, depending on the parameter gamma, the dark region on the images gets darker ($\gamma > 1$) or lighter ($\gamma < 1$). We apply a γ of $1/1.2$ to the image to lighten the darker part of it, this process allows the model to more easily see details in the image.

2) Patchifying: For each image, patches of size 80x80 will be extracted with a stride of 16 pixels.The main reason behind this choice is that we have different sizes of training and testing set images. Choosing those parameters allows us to construct a set of patches that cover entirely the image regardless of whether it comes from the training or testing set. Moreover, it also corresponds to the patch size that is used for the F1 score prediction.

B. Post-processing

To compute our final prediction, we need to undergo several post-processing steps. As each image is constituted of a set of overlapping patches, we reconstruct the final prediction by stacking all the patches and compute the mean prediction of each pixel.

Secondly, as we did for data augmentation, we used a similar idea by applying a rotation of 90, 180, and 270 on each test set image and compute, for each of them, the mean prediction by stacking and rotating back their corresponding 4 sub-images. This has proven to be efficient as we allow our model to extract more information from the same image by looking at it from several different perspectives.

Finally, we assign a prediction for each 16x16 (non-overlapping) patches by computing the mean pixel prediction within a patch and label it as road if it is greater than 0.25, otherwise background. This last steps is shown on figure5.

IV. Results & Discussion

Model	F1-score
Baseline	0.79060
U-Net Based (without patch)	0.87327
U-Net Based on 80x80 patches	0.90843
U-Net Based on 80x80 patches + Gamma&CLAHE	0.91015
U-Net Based on 80x80 patches + rotations	0.91468
U-Net (32) Based 80x80 patches + Gamma&CLAHE + rot	0.92330
U-Net (64) Based 80x80 patches + Gamma&CLAHE + rot	0.92941

Table I
SCORES OBTAINED FOR EACH MODEL ON THE VALIDATION SET (METRIC IS PIXEL-WISE F1-SCORE)

Observing our results we find that to perform well we obviously need a good model, but most of all, it is really important to train it well using techniques like data augmentation, which helps to detect diagonal roads in our case as lot of data often outperform better models. Training on patch of size 80x80 really helps us to have more data on whitch to train our model.

Combining result of multiple prediction of patch (overlapping and rotation patches) allow us greatly improve the our prediction as for each pixel in the center of the images we have to aggregate the result of $(80/16) * (80/16) * 4 = 100$ predictions (Stride on x, Stride on y, rotations)

V. Further work

To improve further our model, we could apply 45 degree rotations on the training set images in order to create new filters that would enhance the performance at detecting diagonal roads. Indeed, although the 90 degree rotations already increases the data with images depicting diagonal roads, we still see room for improvement as the ratio still remains low.

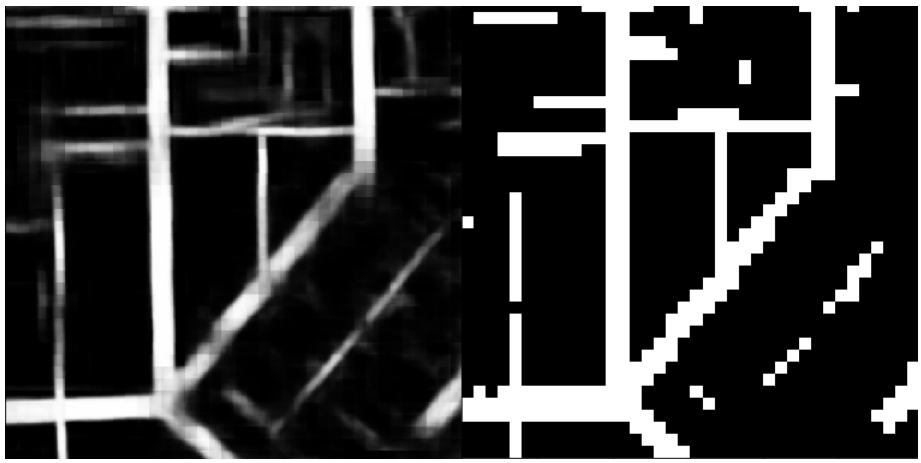

Figure 5. Pixel-wise and patch-wise prediction (as done for the submissions)

REFERENCES

[1] P. F. Olaf Ronneberger and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[2] E. Kauderer-Abrams, "Quantifying translation-invariance in convolutional neural networks," 2016.

[3] V. S. M. D. M. Sebastien C. Wong, Adam Gatt, "Understanding data augmentation for classification: when to warp?" 2015.

[4] G. Hinton, "Dropout: A simple way to prevent neural networks from overfitting," 2014.

[5] J. L. Evan Shelhamer and T. Darrell, "Fully convolutional networks for semantic segmentation," 2016.

[6] Z. S. H. B. D. M. J. E. M. K. B. P. Pisano, Etta and S. Pizer, "Contrast limited adaptive histogram equalization image processing to improve the detection of simulated spiculations in dense mammograms," 1998.