Road Segmentation using Convolutional Neural Networks

Vincent Ballet, Paul Nicolet, Alexandre Rassinoux
École Polytechnique Fédérale de Lausanne
{vincent.ballet, paul.nicolet, alexandre.rassinoux}@epfl.ch

***Abstract*—This paper presents a technique performing satellite image segmentation by separating roads from what we call background, representing anything but roads. Among multiple methods, we choose the current most powerful one for the task, known as U-Net, a particular type of convolutional neural network which has already been adopted for biomedical image segmentation in the past. The final model consists in training such a neural network from cropped and augmented satellite images along with groundtruth road labels, achieving a pixel-wise F1-Score of 94.56%.**

I. Introduction

Image segmentation is a computer vision process which consists in partitioning an image into multiple sets of pixels called segments. Segmenting images can appear particularly useful for various domains where extracting given objects from an image is needed. In our case, we aim to segment satellite images and extract pixels which correspond to roads with high probability. We could imagine that such a process is useful for use cases like automatic world roads indexing and use this information to improve navigation tools. Figure 1 is an example of final segmentation on two test images, with our model, with the red overlay representing background zones.

Classical simple machine learning techniques are often not powerful enough for image classification as they can't easily represent complex models without really smart feature engineering. Additionally, learning from images is recognized as being a computationally heavy task, and fully connected neural networks do not scale well for such problems. We then describe in the rest of the document how we leverage new techniques particularly successful with images classifications: convolutional neural networks.

Our dataset consists in 100 satellite images and groundtruth labels pairs of size 400x400 pixels.

II. Methodology

In the context of object recognition, current approaches make essential use of machine learning methods. Also, among the different ones we tried for this classification problem figure simple models such as logistic regression or random forest. As we were not satisfied with their performance, we tried computationally more expensive models: neural networks.

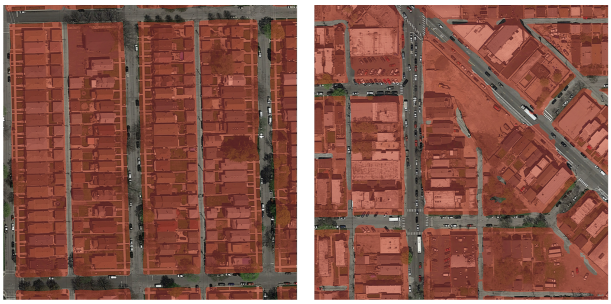

Figure 1. Pixel-wise prediction on test set

A. Convolutional Networks

While convolutional networks (CNN) have already existed for a long time, their success was limited due to the size of the available training sets and the size of the considered networks. Today, convolutional neural networks have been used with great success in the context of image recognition [1], [2]. It then appears natural to use this type of neural networks for our image classification task.

We implement a CNN which works at a patch-wise level as a first attempt. The structure of such model is the application of two convolutions (with variant padding), each followed by a rectified linear unit (ReLU) activation and a 2x2 max pooling operation. It has finally three fully connected layers.

However, current state of the art convolutional neural networks for image segmentation have been described by Ronneberger, Fischer, and Brox, winners of the EM segmentation challenge at ISBI 2012 [3], as the U-Net.

B. U-Net

The U-Net is the network we adopt as final model. Its architecture is built to work with a very few training images. The network does not have any fully connected layer and the output only consists in pixels corresponding to the input image. Hence, arbitrarily large images can be fed to the model. Zero-padding is used to allow border pixels prediction. We split our training images into patches because of GPUs limitations with large images.
The network architecture is illustrated in Figure 2. It consists of a contracting path (left side) and an expansive path (right side). The contracting path follows the typical architecture of a convolutional network. It consists of the repeated

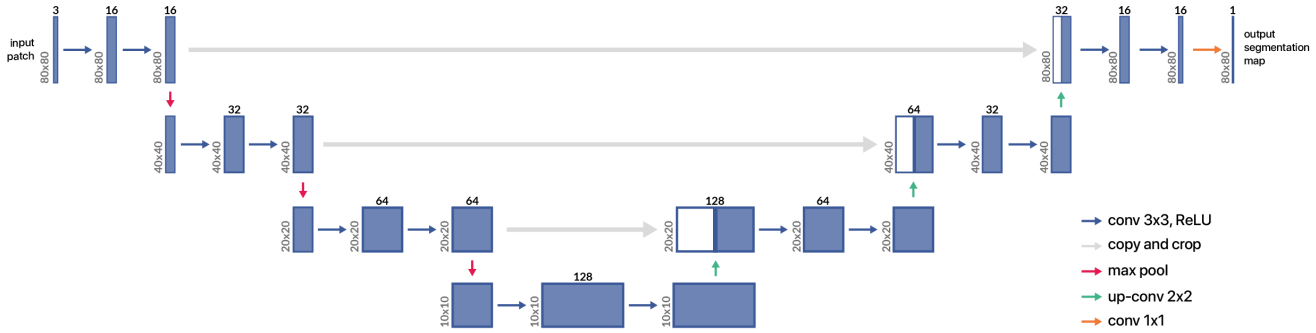

Figure 2. U-Net architecture

application of two 3x3 convolutions (zero-padded convolutions), each followed by a rectified linear unit (ReLU) as activation function and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step we double the number of feature channels. Every step in the expansive path 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 in order to bring back some original information, and two 3x3 convolutions, each followed by a ReLU. At the final layer a 1x1 convolution is used to map each 16-component features vector to the final output mapping each pixel to a probability of being a road. In total the network has 18 convolutional layers.

C. Data Pre-Processing

It is useful to perform image pre-processing mainly because our dataset in extremely small. Also, many kinds of pre-processing methods help avoid overfitting the training dataset. We describe the techniques and justify them in this section.

Training a deep learning model requires more than 100 data samples to get satisfying results. We decide to generate more training data by adding transformed version of the original images. First, each image is cropped in multiple small patches which we choose to be of size 80x80 pixels and overlapping of 20 pixels in our final model. Since roads are usually between 20 and 40 pixels wide, it is important for a patch to cover at least the width of a road. We have been able to see that very small patches imply bad results. On the other hand, large patches do not improve results, while decreasing the training set size. Each of these patches can then be rotated by α degrees and flipped vertically and horizontally to create even more images. By generating more data, we allow our model to be trained on a high number of different patches with multiple road orientations instead of a few entire images with very often parallel and perpendicular roads. We avoid overfitting the original training set and producing a final model way more robust to different road configurations.

Additionally, we tried to augment our images with new channels (in addition to the RGB channels provided) by concatenating to each image a specific version of it, like a blurred version, or a version displaying the detected edge with known computer vision algorithms with the *aleju/imgaug* library. It turns out that these augmentations do not help the model and do not make a significant improvement and even sometimes make it worse compared to training on pure patches. Actually, this makes sense as one of the roles of the CNN is to adapt convolutions to automatically extract the most appropriate image transformation for the task.

D. Model Selection

Even though the U-Net seems to be the best deep learning model for the segmentation task, there are many parameters and hyperparameters which can be optimized and improve the overall prediction by several percents.

The strategy followed is a combination of manual exhaustive search and empirical grid search on the following parameters: patch size, patch overlap, patch rotations and flips, channel augmentations, U-Net structure (overall depth and convolutions at each level) and post-processing, which will be further discussed later on.

Given that training a model takes between one and two hours, we cannot afford to cross-validate each of our attempts since this would multiply the training time by k for a k-fold cross validation. Thus a coarse-grained model selection is done by randomly splitting the training set into a training and validation set in order to gather validation statistics. It is then possible to perform a more fine-grained validation using cross-validation on the few remaining good models, and get the final set of pre-processing parameters (see Table I) and model parameters (see Figure 2) to get the best predictions.

E. Training

Training a deep neural network is not as straightforward as training standard machine learning models: it is more com-

Table I
PRE-PROCESSING OPTIMAL CONFIGURATION

Patch size	Patch overlap	Augmentation	Rotations	Flips
80x80px	20px	None	$i \cdot 45°$	None

putationally expensive, time consuming, and there are many choices to consider. We discuss those in this subsection.

1) Epochs: It is not possible for a deep neural network to reach convergence after a single pass on the full dataset. Hence the model is trained for multiple epochs. During our experiments, depending on the parameters, loss plateau was reached after 30 to 70 epochs.

2) Optimization: loss, optimizer and validation score: We use binary cross-entropy (BCE) as loss function to quantify difference between network outputs and target labels, and then back-propagate it in the U-Net. BCE aims at determining how close is the predicted distribution from the true distribution, for the binary case. BCE is computed as follows for two vectors $\vec{o}$ and $\vec{t}$ respectively representing the output and the target:

$$BCE(\vec{o}, \vec{t}) = -\frac{1}{n} \sum_i \vec{t_i} \cdot \log \vec{o_i} + (1 - \vec{t_i}) \cdot \log (1 - \vec{o_i})$$

As a stochastic optimizer, we choose adaptive moment estimation [4] (Adam), which combines two powerful techniques in order to make convergence faster. First, it uses momentum by using exponential decay on older gradients and adapt convergence pace. Additionally, instead of keeping track of a single learning rate, Adam uses one learning rate for each model parameter, with each of this individual rates upper bounded by the initialization learning rate. Using Adam optimizer instead of a classical stochastic gradient descent shows a significant improvement in the speed for our model to reach its optimization plateau. The base learning rate is chosen to be 0.001. Furthermore, we also use a dynamic learning rate adaptation in order to refine optimization when loss starts to stagnate: learning rate is divided by 10 each time the training loss stops decreasing for five epochs (Figure 3 shows losses as a function of epochs). Learning rate fine tuning is not necessary with Adam since internal learning rates for each parameter are adaptive.

Since labels are highly skewed with only around 25% road pixels, it is hard to evaluate the model using classical accuracy. Hence, we use F1-score, a harmonic mean of precision and recall, to mitigate the data imbalance.

3) Regularization and Overfitting: It is important to consider various regularization methods in order to avoid very complex models which might overfit our small dataset.

The first method to solve these problems is contained in our pre-processing steps. Since we use extensive patch generation with rotation, we avoid overfitting parallel roads as discussed in subsection II-C.

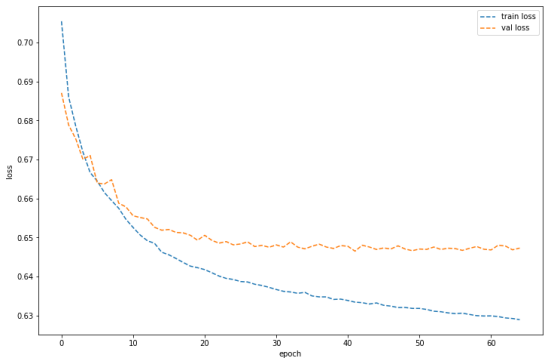

Figure 3. Training and validation losses through epochs

In addition to pre-processing, we use dropout. This technique consists in randomly deactivating certain units and their corresponding links by simply ignoring those during a forward or backward pass for a particular training phase. This is necessary because neurons develop co-dependency between each other, curbing each unit power and leading to overfitting [5]. In our model, each unit of the last contracting path layer is dropped with probability $p = 0.2$.

One last technique is used: batch normalization. Even though this technique has been first developed to accelerate deep neural networks training [6], it turns out that, coupled with dropout, this acts as a regularization tool, which allow us to speed up training, while dispensing with L2-regularization [7]. Batch normalization consists in applying classical normalization pre-processing to each batch while training the network.

4) Implementation details: The neural network is implemented using the Pytorch framework. As training a large CNN takes time, we use a GPU NVIDIA TITAN X in order to accelerate the process along with the CUDA version of Pytorch. Training would simply not be feasible on a simple CPU, as it takes around 2 hours on the GPU.

Table II
MODELS SCORES

Baseline	Random forest	CNN	U-Net	U-Net + pre-proc.
73.03%	78.76%	85.51%	92.32%	94.56%

III. RESULTS

A. Baseline Model

As the aerial images are composed of roughly 73.03% of background, this corresponds to the baseline accuracy when we predict each pixel to be a background pixel.

B. Classical Models

For machine learning classifiers, it is necessary to extract features from each patch to be classified, we keep those simple by using RGB color mean and variance.

A basic approach consists in using a logistic regression classifier trained on each 16x16 patch. Results are however similar to predicting all background. Another one is to use a random forest classifier. Performance is better than the logistic regression model but still not satisfying with 78.76%, even with performing a grid-search over hyperparameters.

C. CNN

The implemented CNN described in II-A achieves the first satisfying result with such a model: 85.51%.

D. U-Net

The U-Net described in section II is likely to be the most efficient model. We get a score of 92.32% without any image augmentation technique, and we reach our best score of 94.56% by using all the pre-processing steps described above.

IV. Discussion

After looking at the predicted images, it appears quickly that there is room for improvement to get very high accuracy. We read in related research [8] that some computer vision techniques can be used to detect roads with very high precision using road vectors data. In this direction, we try to extract road vectors from predicted images, aiming at refining those prediction by using similar techniques. A result of vector extraction can be seen in Figure 4. Even though we can clearly see roads being followed by vectors, those would need to be near perfection to be able to improve a prediction without degrading it.

Another post-processing technique, namely majority-voting, has surprisingly mediocre results. Indeed, we think that it is possible to use the fact that our U-Net learns road in any orientation, to predict a given test image rotated by $90°$, $180°$ and $270°$. Then, along with its original version, it is possible to decide for each pixel which label to assign based on a majority voting protocol on the 4 predictions. It turns out that even if the technique often visually improves predicted roads by filling holes for example, the prediction if usually lower than without performing any post-processing.

V. Conclusion

At the end, we are able to get a quite solid model with high score for segmenting satellite images by extracting road pixels from background using the U-Net. This shows that in the domain of deep learning, it is often the case that using already developed related work will bring best results than starting from scratch.

For multiple audio, text and image applications, people have already build the best model, and sometimes *reinventing the wheel* is definitely not the best option, as seen by using the U-Net.

In future work, we could implement more advanced techniques like *ensemble learning*, which consists in combining multiple set of good models together in order to improve stability and power of the prediction.

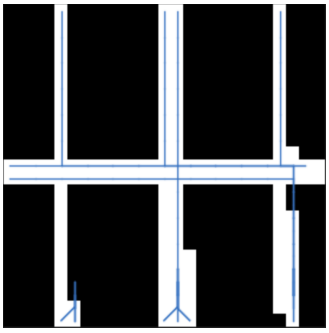

Figure 4. Road vector detection

Although there are many ways to improve those prediction, we reached a global accuracy that defies human's capacity to solve this kind of problems. Our models could be used anywhere in the world and provide real-time information that could help answer bigger problems. For instance, identifying nature disasters, in the cases of earthquakes or flood. We believe that neural networks is just the first step in the quest of a brighter future in which machines can work not for humans but with humans.

References

[1] F. H. Y. LeCun and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," The Courant Institute, New York University, Tech. Rep., 2004.

[2] I. S. A. Krizhevsky and G. Hinton, "Imagenet classification with deep convolutional neural networks," University of Toronto, Tech. Rep., 2012.

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

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[5] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012. [Online]. Available: http://arxiv.org/abs/1207.0580

[6] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[7] L. A. dos Santos. (2017) Batch norm layer. [Online]. Available: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/batch_norm_layer.html

[8] J. Yuan and A. M. Cheriyadat, "Road segmentation in aerial images by exploiting road vector data," in *2013 Fourth International Conference on Computing for Geospatial Research and Application*, July 2013, pp. 16–23.