Machine Learning Road Segmentation Project

Abiola Adeye - Abdeslam Guessous - Aya Rahmoun
Ecole Polytechnique Fédérale de Lausanne, Switzerland

***Abstract*—The aim of this project is to create a road segmentation classifier by differentiating roads from background on satellite images. In this report we show how a Convolutional Neural Network, in particular the U-Net architecture can be used to perform this classification, whether the model is created from scratch or whether we opt for a pretrained model and adapt it using transfer learning.**

I Introduction

In a time where mapping tools like Open Street Maps are becoming an integral part of our society, where autonomous cars are no longer a future dream, but a present reality, sophisticated imagery of our roads becomes vital. Road segmentation, which is the process of identifying roads on satellite imagery, becomes fundamental.
Traditionally, road segmentation was done by hand during volunteer events called Mapathons or by specialized organizations. This process is not only time consuming and potentially expensive but can also be prone to human error.
Nowadays, this human interaction is only needed for providing ground truth images that can be then used by machine learning algorithms to make predictions over millions of images.
In this report we present a specific set of algorithms, named Convolutional Neural Networks (CNNs), which seem to be a perfect match for this task.
We started with a baseline model using a small CNN before implementing our own U-Net for which we tried to optimize the hyperparameters. We finished by using various pretrained models like VGG16 or ResNet and adapted them to our task using transfer learning. We then compared all the different models and concluded by stating which model is best suited for this specific task.

II Data Analysis and Augmentation

A. Data Analysis

Our training data set consists of 100 RGB images made of 400x400 pixels each. Each image corresponds to a satellite visualization of the streets and is associated to a ground truth gray scale image which represents roads in white and other structures in black.
The goal is to perform a binary classification for all the pixels in the images, i.e. output a 400x400 pixel mask where each pixel is predicted white if it is part of a road, and black if it isn't.
Implementing an efficient road segmentation task can be quite challenging. First of all, we have infrastructure like parking lots and train tracks that look very similar to roads. Furthermore, we can have objects like trees that obstruct the satellite view, which could fool our model into making a wrong prediction.
You can see a sample image from our training set and its associated ground truth in Figure 1.

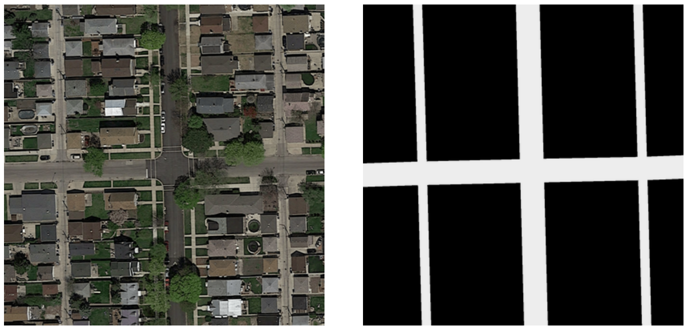
Fig. 1. Image Satellite (L) and its ground truth associated (R).

B. Data Augmentation

As we only had 100 samples in our training set, we chose to perform data augmentation in order to increase the amount of data and introduce variability in the data set. Indeed, if a data set in a machine learning model is rich and sufficient, the model performs better and more accurate.
This has been done by adding slightly modified copies of our existing data. We used various methods such as rotating, flipping and cropping. Each image and its ground truth is rotated 8 times by multiples of 45°, and mirrored with respect to the horizontal and vertical axis. We treated the parts of the rotated images outside of the size limits as background in order to keep all images in the same format. We also considered generating some images with noise, namely "salt and pepper" noise, by changing some random pixel values to black or white. Lighting condition was also a variable that could allow us to generate more variable data. Unfortunately, adding noise and changing lighting conditions did not improve our accuracy so we decided to remove it.
This data augmentation lead us to obtain 2400 images. This helped us represent situations from our test set that were not initially present in the training set. Finally, in order to load images into numpy arrays quicker we stored them in h5 files.

III Models and Methods

Different metrics were available for evaluating a model. The choice of the metric is relevant and has to be done according to the specificities of the tasks. Here, the data exploration showed that a large part of the image (around 75%) was background.

Model	Loss Function	Hyperparameters	Activation Function	F1 Score (%)
Baseline with dense CNN	Binary Cross Entropy	Sample Size = 100, Batch Size = 10	ReLU	65
U-Net	Binary Cross Entropy	Sample Size = 2400, Batch Size = 10	ReLU	72
U-Net	Dice Loss with IoU coeff	Sample Size = 800, Batch Size = 10	Leaky ReLU	70
U-Net	Dice Loss with IoU coeff	Sample Size = 1600, Batch Size = 10	Leaky ReLU	73
U-Net	Dice Loss with IoU coeff	Sample Size = 2400, Batch Size = 10	Leaky ReLU	75
ResNet	Dice Loss with IoU coeff	Sample Size = 2400, Batch Size = 20	Leaky ReLU	77
ResNet	Dice Loss with IoU coeff	Sample Size = 2400, Batch Size = 40	Leaky ReLU	78
VGG16	Dice Loss with IoU coeff	Sample Size = 2400, Batch Size = 32	Leaky ReLU	77
VGG16	Dice Loss with IoU coeff	Sample Size = 2400, Batch Size = 64	Leaky ReLU	78
ResNext	Dice Loss with IoU coeff	Sample Size = 2400, Batch Size = 32	Leaky ReLU	77
ResNext	Dice Loss with IoU coeff	Sample Size = 2400, Batch Size = 64	Leaky ReLU	78

TABLE I
HYPERPARAMETERS TUNING RESULT.

As a result, a model predicting only black will have a 75% accuracy, even if it performs very bad in reality. This shows that accuracy is not an appropriate metric. That's why we turned to F1 score metric. It is defined as the harmonic mean of the precision and recall [1]. The highest possible value of an F1 score is 1 and the worst possible value is 0. We chose to express it in % for simplicity.

A. Baseline

Image segmentation can be done with traditional neural networks that only have dense (i.e fully) connected layers. The issue with traditional dense neural networks is that they do not preserve spatial correlations. Spatial correlation is a property of data that defines that data points that are in a unit, a correlated group of neighbor objects, cannot be separated without modifying the integrity of the unit. In an image, pixels that are close to each other have high probability of being correlated. Therefore, as traditional neural networks separate pixels by putting images in one dimensional vector, it does not yield optimal results.

We know that in Convolutional Neural Networks, correlated filters are used to preserve spacial correlation between pixels. We therefore chose a CNN as our baseline, whose architecture was based on a dense convolutional network.
It can be split in two parts as shown above, the encoder and the decoder. The encoder (feature extractor) was made of two convolutions that were each followed by a (2x2) max-pooling layer that was feeding into a Relu activation function. The output of the encoder was then given to a 512 neurons fully connected dense network with 3 layers.
This implementation made predictions by patches of pixels. However, predictions over individual pixels yield higher results, and this will be shown in the next part. Our results are presented in Table I.

We looked for further ways to improve our accuracy and after a thorough research, we found out that the current state-of-the-art technology that is best suited for image segmentation is a special type of CNN called U-Net.

B. U-Net

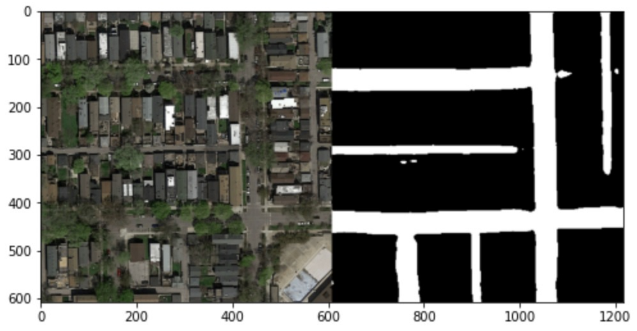

Fig. 3. Prediction Result of Our U-Net

A U-Net is based on the fully convolutional network architecture. This means that its encoder is made of several convolution blocks and its decoder is made of several deconvolution blocks. There is no fully connected layer in this type of network. Between the deconvolution and the convolution blocks, skip connections have been added in order to get a more fine-grained prediction. The U-Net learns segmentation in an end-to-end setting: we input a raw image and get a segmentation map as the output. It performs classification on every pixel, so that the input and output share the same size.
In practice, this turns out to give a better accuracy than predicting patches of pixels.

To implement our U-Net, we used Keras which comes with the Tensorflow backend. Keras is an API that makes it easy to build CNNs from scratch, layer by layer. The architecture of our network can be seen in Figure 2. It takes as input a 3-channel (RGB) 400x400 pixel image. At each level of our network, convolutional filters are used to increase the number of channels and to extract the spatially correlated features. The output of our convolutions then goes into a 2x2 max-pooling layer which is followed by a Relu activation function. For the convolutions, we used 3x3 filters and ”same padding” in order to conserve image dimensions. Once we reached the lowest level of our network, we started the

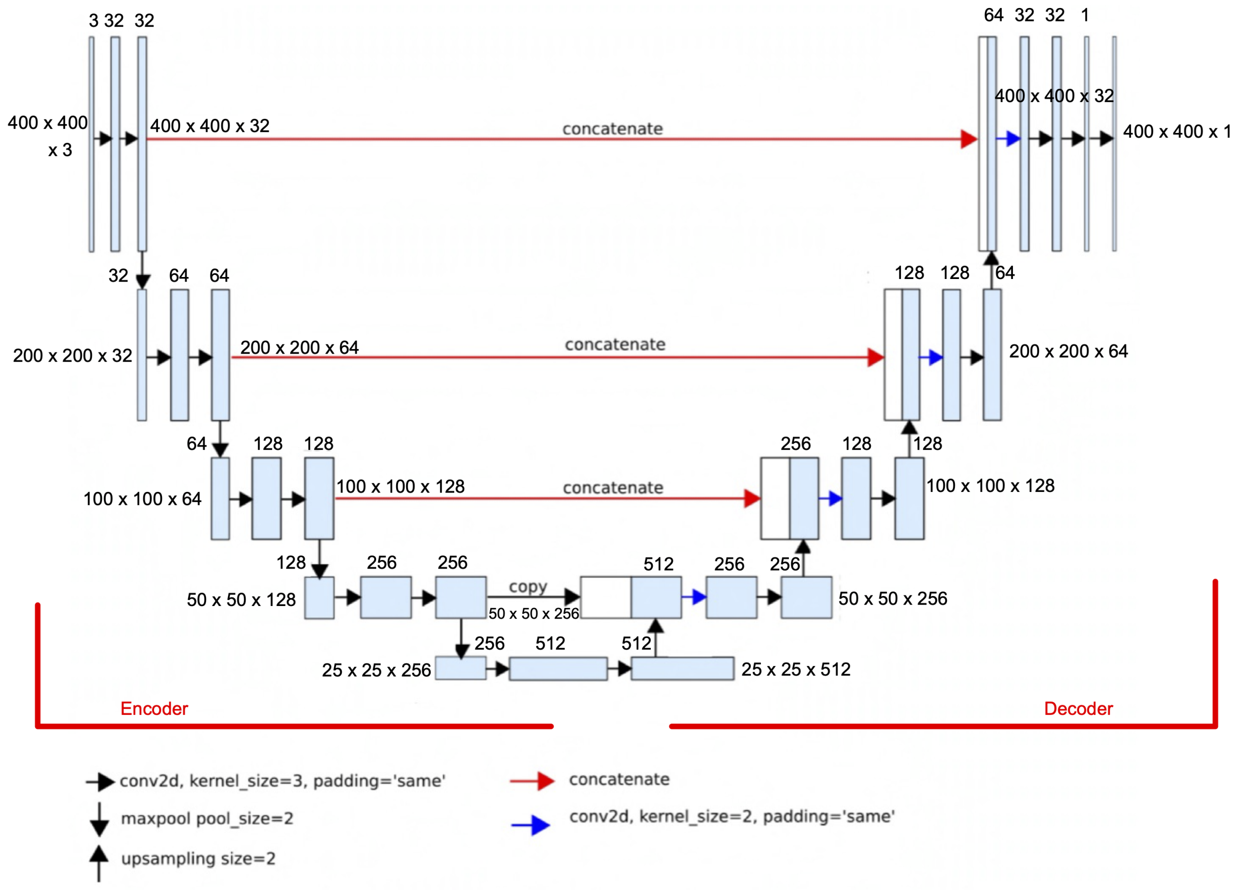

Fig. 2. Our U-Net architecture [7]

upsampling (decoding) operation. This allows us to get back to our initial image dimensions. Each deconvolution block uses 3x3 filters with ”same padding” and is followed by classical convolutional filters like in the encoding blocks.

In order to get a more fine-grained predictor, skip connections were added between the decoding and encoding blocks at each level. We also proceeded to hyperparameters tuning (cf. Table I).

The hyperparameters we tuned to improve our model are the following :

- *Activation Function:* It defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network [2].
  We trained our model using ReLU and Leaky ReLU (cf. Figure 4). We used ReLU in the beginning but we faced the ”dying ReLU problem: a ReLU neuron is “dead” if it’s stuck in the negative side and always outputs 0 [3]. After this, we trained our models using Leaky ReLU on our models. It fixed the “dying ReLU” problem, as it didn’t have zero-slope parts and therefore, speeded up training.

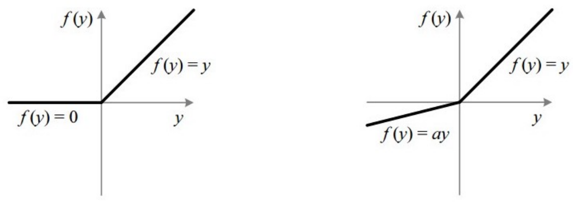

Fig. 4. ReLU (L) and Leaky ReLU (R).

- *Dropout Rate:* It is a technique used to prevent a model from overfitting. During the training, some layer outputs are randomly dropped. In some situations, different dropouts rates can be used for hidden units and for visible units. For our algorithm, we tried different values, going from 15% to 50%, but we finally saw that 20% was the best compromise between keeping enough neurons, and not overfitting.

C. Pretrained Models

After hitting a plateau in accuracy improvements with our self-built U-Net architecture, we were looking for a way to improve our results.
One of the most popular approaches that came up during our

research was to replace our encoder (the part of our network that learnt the features) by a pretrained encoder like VGG16 (cf. Figure 5), VGG19, Inception or ResNet.
Indeed, a pretrained model is already trained on a much bigger dataset and created by someone else to solve a similar problem. The idea is to use knowledge leveraged from a source task to improve our learning on the current task. This method is called transfer learning. [5].
This is very useful as pretrained models have been trained on millions and millions of samples like Imagenet, and can contain more than 10 million parameters.
Creating such a model ourselves with the resources that are available to us, in a reasonable time frame would have been quasi-infeasible. Surely, this allowed us to extract fine-grained details of our images.
Because our encoder and decoder have to be symmetric in a U-Net, we used a library called ”Segmentation Models” [6]. This library allows to automatically build a symmetric decoder for all of the pretrained models that it supports.

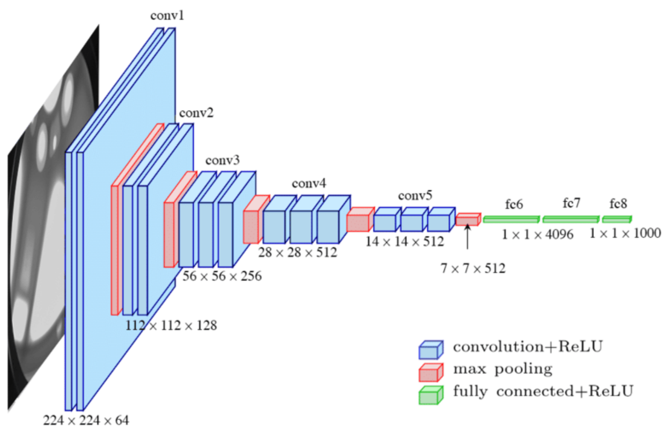

Fig. 5. A standard VGG16 architecture

D. Results

Model	Activation Function	F1 Score (%)
Baseline	ReLu	65
U-Net	LeakyReLu	75
VGG16	LeakyReLu	78
ResNet	LeakyReLu	78
ResNext	LeakyReLu	78

TABLE II
BEST RESULTS FOR OUR MODELS.

IV Summary

To create a classifier for road segmentation, we started by doing a data augmentation on our provided data set composed of 100 images and their ground truth. This allowed to train our model on 2400 images.
Then, we naturally turned to CNN, as it is well appropriated for images. After this, we decided to use U-Net, a particular type of CNN developed for image segmentation. We then tried different hyperparameters optimization methods like drop out, changing batch size, and used various activation function.
Finally, we decided to use pretrained models as road segmentation is a very common problem. We used famous models: VGG16, ResNet, ResNext, and Transfer Learning has been performed in order to adapt it better to our task.
Unfortunately, our model didn’t performed as good as we expected. Therefore, we could still improve it by changing the hyperparameters or some part of the architecture of the U-Net.

References

[1] https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/
[2] https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/
[3] https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7
[4] https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
[5] https://www.kaggle.com/basu369victor/transferlearning-and-unet-to-segment-rocks-on-moon
[6] https://segmentation-models.readthedocs.io/en/latest/tutorial.html
[7] Inspiration drawn from: https://helloml.org/u-net-convolution-neural-network-for-image-segmentation/
[8] https://www.tensorflow.org/