Neural Network Architectures for Road Segmentation

Satelliterz Group
Hadi Hammoud, Marin Cornuot, Antoine Roger
Machine Learning Course, Martin Jaggi and Nicolas Flammarion, EPFL

***Abstract*—This paper addresses the challenge of image segmentation using machine learning. The goal is to achieve the best possible accuracy for road segmentation, determining whether each pixel from a set of satellite images is part of a road or not. In order to achieve this task, deep learning models and more precisely Convolutional Neural Networks are a very promising technique we used to build generalized and robust models. One of the challenges is also to create accurate models from a very limited data-set. We address these challenges through several data-augmentation techniques and the testing of different Neural Network models and architectures described here. This allowed us to achieve an accuracy of 95% and a F1 score of 0.906 on the Aicrowd competition platform.**

I. Introduction

Image segmentation is one of the main domains of computer vision. The goal is to classify each pixel of an image into a set of predetermined categories. Applications are wide, ranging from autonomous driving, medical image analysis, and aerial image processing. For the last application, creating a map of all roads on Earth from satellite images would require a tremendous amount of manual work and generally a highly repetitive process, especially since new roads are built and modified everyday all over the world. One way to account for this problem is to automate it through machine learning techniques and more specifically the training of Deep Learning models. More specifically we will discuss how different types of models and architectures of the very well known Convolutional Neural Network perform. We will use a 'naive' CNN as a baseline and from there, we will investigate the performance of the U-net architecture which is an advanced technique for visual computing with promising results. We will also discuss some improvements of this model through the implementation of the ResNet variation and finally we will look into VGG16 which is a very recent technique for data segmentation through the usage of very deep learning models (with 16 layers or more).

II. Dataset

The training dataset we are considering is composed of 100 RGB satellite images from Google Maps, with resolution of 400x400 pixels. The testing dataset contains 50 of the same kind of images with resolution of 608x608 pixels.
For every image in the training dataset, we have a corresponding groundtruth mask indicating where the roads are. This mask is a matrix of the same size as the image filled with 1s for each pixel part of a road on the training image and 0s elsewhere as illustrated in Fig. 1.

This mask is what our models aims to recreate for the 50 images in the testing dataset.

All images from both the training and testing data-set look alike, they represent similar kind of residential area. All images have a similar color palette and similar dark tones. Since the images dimensions are different, the zoom level is a little different between the training and the testing images, which might be a hurdle to overcome for the models. This data-set represents several challenges, especially since there is the presence of obstacles on the roads (trees, cars, buildings) and there are also some variations in shapes and colors of the roads as well as tricky part on some images with the presence of rail tracks as well as parking lots which are a significant challenge for road segmentation by machine learning.

Fig. 1. Image and mask from the Training Data-set

III. Data Augmentation

The provided data-set is rather small (100 annotated images with their mask). To increase the performance of a deep learning model it is necessary to get as much data as possible. We use a wide variety of techniques to create new training samples by applying different transformations to the original training data-set. By running some tests, some transformations have proved very successful to improve the accuracy of our models while others where discarded. We also had to take into account the trade-off between accuracy and performance since more training data means longer training and this is especially true with deeper and wider Neural Net we have used. However,

U-nets tend to perform quite well even with relatively small data-set so this was not too much of a limiting factor.

Combinations of the following transformations were applied to the AIcrowd training data-set:

- ***Flipping:*** The images and masks were flipped both horizontally and vertically.
- ***Rotations:*** The images and masks were rotated by 0°, 90°, 180° and 270°
- ***Contrast:*** Contrast of the images was adjusted by a random factor among [0.8,1.2]
- ***Brightness:*** Brightness was enhanced by a random value within $+/-30\%$
- ***Hue:*** Hue was adjusted by a random factor among [-0.1,0.1]
- ***Cropping:*** A centered "zoom" was applied on the images, cropping 5% on all sides.
- ***Saturation:*** Saturation was increased randomly by somewhere between 0% and 50%

We also tried using some colors filtering as well as adding Gaussian noise but these techniques turned out to be inefficient and complexified our model without bringing any accuracy gain. This is probably because the training data-set is quite homogenous as we have seen and very similar to the images in the testing data-set. We reach 1400 images in our expanded data-set after fine tuning our data augmentation with the techniques described above. Not only did we gain several F1-Score and Accuracy points, these techniques also increase the robustness of our model and help reduce over-fitting.

IV. Models and Techniques

The use of Deep Learning Neural Networks is particularly adapted to this sort of computer vision tasks and are capable of achieving high accuracies, as apposed to standard linear classifiers. This sections describes the different models we used and crafted. Describing a naive Convolution Neural Network as a baseline and then the more efficient U-net Architecture as well as a ResU-net variation. We'll also discuss our tests with a variation of the VGG Neural Net.

A. Baseline: Fully Convolutional Neural Network

For our baseline, we choose a fully convolutional network to compare the results with. Motivated by the post-processing step, instead of predicting the class of each pixel in the image, this model predicts the class of each 16x16 patch. The idea behind its architecture is to first reduce the "image of pixels" into "image of patches" (so each patch will be associated with 1 pixel, resulting in an image size of (H/16, W/16)), and then retrieve the original image size by expanding the patches into 16x16 pixels with the same intensity(no decoders are used). The first step is achieved by applying consecutive convolutional layers with stride equal to filter size, and with product of filter sizes equal to 16. Since this can be done in different ways, we trained different models depending on the depth required to reach the "image of patches". In the CNN4 architecture, a filter of size 4 was applied first, followed by 2 convolution layers with filter sizes equal to 2 (hence resulting in image size reduction by factor of 4x2x2=16). The second step can be done using 1x1 filters to reduce the channel dimensions before applying the expand function which augments the image with the 16x16 pixels of same intensity as the patch pixel,

B. U-net

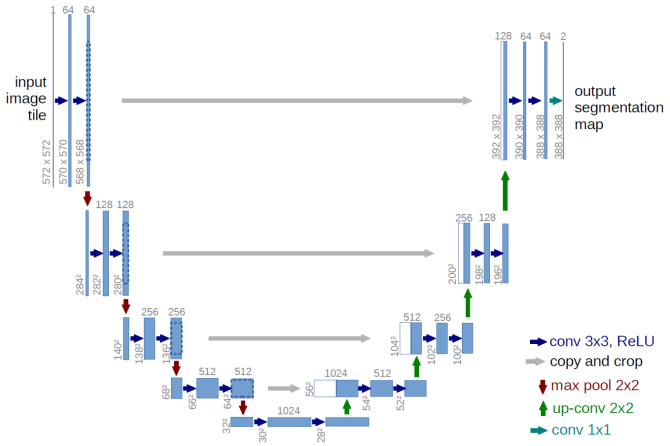

Fig. 2. U-NET architecture described in the original U-Net paper

Our main limitation with CNNs is our dataset's size. U-Nets are fully convolutional neural networks designed to work on relatively small image datasets. This is great for our study since we have a relatively small data set.

A U-Net is made of 3 different parts : the **encoder** part, the **decoder** and the **bridge** between the encoder and the decoder.

Encoder:
The encoder consists of 2 sequential blocks with a convolution and an activation function followed by a max-pooling reducing the spatial dimensions of the feature maps by half. This part of the network simplifies the image inputs to extract and learn their main features. **Decoder:**
The decoder begins with a transpose-convolution before being concatenated with the connection of the corresponding encoder layer. Those connections are the grey arrows in the U-Net figure: they transmit data from the encoder side to help the decoder reconstruct a full resolution mask.
The end acts as the opposite of the encoder block: it consists of 2 sequential blocks with and an activation function followed by a max-pooling reducing the spatial dimensions of the feature maps by half. This part of the network simplifies the image inputs to extract and learn their main features.
To connect both parts of the network, a simple **bridge** is needed consisting of a similar pattern as before: 2 sequential blocks with a convolution and an activation function. This bridge, also called the **bottleneck**, forces the model to learn the required information to reconstruct a full resolution mask on the way up.

Regarding the specificities of our U-Net implementation, we chose a 4 layers down U-Net since more layers started degrading the result due to overfitting.

C. Residual U-net

As we have seen, the main problem our model is facing is finding a balance between not having enough layers down and losing accuracy or having too many and overfitting: matching the training set too closely and not adapting well to unknown data.

One of the ways to fix this issue is **Residual Networks**. Residual Networks give the model the ability to skip layers and thus gives us the possibility to define more layers while making sure the model will not overfit since regularisation will skip unnecessary layers. The difference between the connections seen in UNet and these connection is that in UNet, connections are not added, but concatenated.

This allows us to use a 5 layers down Residual Network which gave us our best performance at 95% accuracy and an F1-score of 0,906.

D. VGG16 NN

VGG is a variation of a Convolutional Neural Net. It's main specification is the depth which is pushed to 16 to 19 layers. The increase in depth is done through the use of very small convolution filters (size 3x3). VGG models have proven to be very accurate for large-scale image recognition/classification. The ConvNet takes as input a fixed-size RGB 224 pixels wide squared image (3,224,224) which is a challenge since the training image are 400 pixels wide and this could cause some information loss. Pre-processing includes loading the image in the correct size and substracting the mean RGB value.
The image is passed through a stack of convolutional layers, with the use of filters and very small receptive fields (3x3) to capture direction informations. Spatial padding, spatial pooling and max pooling are then all performed. The input is then passed through a stack of convolutional layers followed by Fully-Connected layers with 4096 channels each. ReLU is used extensively used in the hidden layers. Interestingly, normalization is said not to increase performance but instead cause longer computation times. The architecture of the VGG16 model can be found here 3.

By training our dataset on the VGG16 Neural Net form pytorchvision we achieved a F1-score of 90.3% after 30 epochs with a batch size of 2 and a learning rate of $10e-4$ which was promising since very close to our best ResNet model. Unfortunately training these model required too much time for the loss to converge and too much computing power as we were hitting the credit limit on google colab (and did not have enough GPU power at our disposal). This is definitely something that should be looked further into and it is likely that with a deeper 19 deep VGG network as well as with more training cycles a better accuracy could have been reached. A residual extension could also be added for better performance and accuracy.

Fig. 3. Hidden layers of the VGG 16 Neural Net model

V. Training

Training was a big challenge has it takes a significant amount of epochs and thus computing power for the loss to converge. We used Google Colab to have more computing power but where not able to perform all the tests we wanted with sufficient epochs due to hardware limitations.
Furthermore we decided not to use a validation dataset since the training dataset was already small. Including a very small one would have not been very informative and the reduced training data-set would have affected a bit the performance of our models. For tuning the hyperparameters and help us detect overfitting we also had to deal with the 5 submission per day limitation of the AIcrowd platform.
Here 4 is the convergence graphs for the loss for the different models. We can see that we did not have enough time and power to let the VGG16 model converge properly, also the Naive CNN perform very poorly compared to the more advanced U-nets and VGG. Nonetheless, we observe that the more layers the CNN has, the lower the loss is (by a tiny margin).

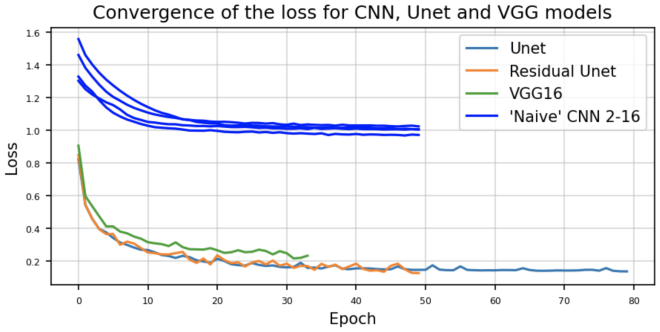

Fig. 4. Loss convergence following epoch training for different models

A. Loss

To choose the loss function to use in the model's training, we looked at the most adapted loss functions for semantic segmentation and boundary detection. Our problem can be reduced to a binary problem: for each pixel in the input image we want to output a binary output of 1 for a road and 0 otherwise.
This means that we can use the common Binary Cross Entropy formula for the model's loss:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1-y_i) \cdot log(1-p(y_i)$$

Binary Cross Entropy loss interprets the output as a probability of returning 0 and 1. This works in our model by interpreting the percentage of white pixels in a given patch as a probability to output one value or the other.

A usually more adapted solution to boundary detection is computing the loss with the Sørensen–Dice coefficient:

$$D = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2}$$

Dice loss is based on the overlap of prediction (p_i) and groundtruth (g_i) values. The denominator takes care of the loss on a global scale by computing the intersection between the 2 sets of values while the numerator computes loss on a local scale for each set individually. These 2 separate scales make for a more accurate representation of the loss and ultimately a faster convergence for the model.

Since both loss functions performed well on our model, we decided to keep both by adding a new type of loss made of the sum of both the BCE and the Dice loss.

B. Transfer learning

To speed up running times whenever possible, we used transfer learning between models with the same weights shape. For example, our 4 layers down U-Net and 4 layers down ResNet have the same weights shape since the only difference between the 2 models is the forward pass allowing the Residual Network to skip layers.

This means that by training one model, we give another model a good basis to start learning from. This can be extended to using weights from a different semantic segmentation problem to our specific problem but since our training took only around 30 to maximum 50 epochs to converge, we did not need to use external weights as a basis for our own.

VI. Data Post-processing

The VGG model takes as input images of size 224 by 244. This adds a constraint to our pipeline since the test images have a resolution of 608 by 608. To avoid scaling down our inputs and risk losing information and reducing the model's performance, we created a script to split the test inputs in 4 subdivisions. Then we run the model on these smaller parts. Finally, we recreate the full size output mask by combining the 4 smaller masks created.
Using this principle, we also decided to apply this principle to our convolutional neural networks to make sure we did not lose any information when running a 608 by 608 input on them.

A binary label $\{0,1\}$ is therefore assigned to each pixel based on the model's prediction of whether it represents a road or not. In order to make a submission on the platform we need to merge this data to make predictions for every 16x16 pixels patch. It is decided that over a threshold of $t = 25\%$ of white pixels in a given patch, it is labelled as "road". The transition from the Test image, to a model prediction and onto the 16x16 pixels patch mapping is illustrated Fig. 5. This technique is reasonable since all roads are in similar pixel regions and following rather straight perpendicular lines in general.

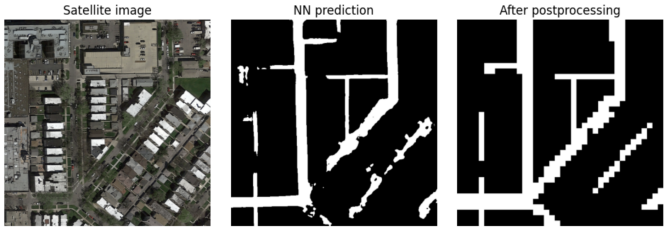

Fig. 5. Process from Test image, to Model prediction to 16x16 patch mapping

VII. Results

Each model we use is an improvement over the previous ones. Using only CNNs, we are not able to achieve accuracy above 80% due to the limitations of the models.
As expected, our U-Net is much more fitted to the task: its bottleneck solves standard CNNs difficulty of identifying the main features. It allows us to get a 94.1% accuracy and a F1-score of 0.899.
The next step in this evolution is introducing Residual Blocks. It builds upon our U-Net and gives it the ability to skip unnecessary layers which is great because it allows us to give it more layers initially and not worry about overfitting the model early. This is our best result with a 95% accuracy and a F1-score of 0.906, slightly above pytorch's VGG-16 network, despite not having enough computing power to experiment with hyperparameters and ways to optimize it. VII displays the performance of each model.
Overall, all models performed as expected with accuracies and F1-scores close to the best results we were challenging against.

Model	F1 Score	Accuracy
CNN16	0.560	0.781
CNN8	0.554	0.800
CNN4	0.591	0.803
CNN2	0.624	0.816
U-Net	0.889	0.941
Res-U-Net	0.906	0.950
VGG-16	0.903	0.949

VIII. Conclusion

In this paper, we investigated the performance of modern neural networks in solving the road segmentation task. We observed how certain architectures can be combined to build more powerful models. This was the case for Res-UNet, which combined the FCNN's architecture as encoders and the Residual blocks as skip connections, and ended being the best model in the list. Moreover, we applied several data augmentation techniques to further improve the result. Finally, with the Res-UNet, we were able to achieve an accuracy of 95% and a F1 score of 0.906 on the Aicrowd competition platform.

ACKNOWLEDGEMENTS

The team would like to offer special thanks to Martin Jaggi, Nicolas Flammarion, and the CS-433 ML staff for this project and for their valuable and constructive suggestions during the development of this research work.

REFERENCES

- U-Net architecture [Fig. 2]: https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/
- VGG article: https://arxiv.org/pdf/1409.1556.pdf (Very Deep Convolutional Networks For Large-Scale Image Recognition)
- VGG16 image architecture [Fig. 3]: https://datacorner.fr/vgg-transfer-learning/
- Dice loss study: https://medium.com/ai-salon/understanding-dice-loss-for-crisp-boundary-detection-bb30c2e5f62b
- Binary Cross Entropy study: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a
- U-Net model: https://github.com/nikhilroxtomar/Retina-Blood-Vessel-Segmentation-in-PyTorch/tree/main/UNET
Deep Learning in Neural Networks: An Overview: https://arxiv.org/pdf/1404.7828.pdf
U-Net: Convolutional Networks for Biomedical Image Segmentation: https://arxiv.org/pdf/1505.04597.pdf