Road segmentation

Zeineb Ayadi, Selim Fekih, Nizar Ghandri
Ecole Polytechnique Federale de Lausanne, Switzerland

***Abstract*—This project is part of the ML course at EPFL. The goal is to develop a classifier that successfully classifies roads and background segments in satellite images. We perform data preprocessing and develop two models, a Convolutional Neural Networks (CNN) model and a U-Net model then compare them. An F1-score of 0.850 was achieved using the best U-net model on the train set and a F1-score of 0.886 on the test set.**

I. Introduction

The data-set that has been provided contains urban areas images picked from Google Earth. Each image is associated to a grey-scale ground-truth image where roads are colored in white and background colored in black.
The objective is to build a classification model to detect roads in given satellite images.
First, we explored the data-set and preprocessed it. Then, we built the CNN and the U-Net models, ran them on the preprocessed data, using Google Colab with GPU for faster computing, and compared their performances. Finally, we took the model with the best accuracy and F1-score and used it to make the predictions over the test set.
We provide an example of the given dataset:

(a) Image (b) Groundtruth

Fig. 1: Satellite image and it's Groundtruth

II. Data Preprocessing

A. Analysing the data

We are provided a training and a test set from CrowdAI. The training set consists of 100 RGB satellite images, of size 400x400 pixels with their corresponding ground-truth images of the same size in black and white. The test set contains 50 images RGB satellite image but of different size, 608x608 pixels. The ground-truth images for the test set are not provided, the performance is tested on CrowdAI by submitting the predictions. The main test metric is the F1 score.

B. Data Augmentation

We are provided with only 100 images, which is not enough to train a good model. To deal with this problem, we performed some data augmentation to generate new images from the given ones. Below are the operations we performed on the images. Each new image is generated by a random mix of these operations.

1) Rotation by 90° or 180°: we chose only rotation angles that are multiple of 90° to avoid problems with image dimensions.
2) Vertical and Horizontal Flips: these two operations differ from the ones above, they apply mirror transformations to the images.
3) Horizontal and vertical shifts: where the shift factor follows a uniform law U(0,0.05)
4) Zoom on the image: the zoom range follows a uniform law U(0,0.2)
5) Change image brightness: make it darker or lighter. The brightness factor follows a uniform law U(0.5,1.5)
6) Add some salt and pepper noise in order to randomly change some pixels values. This helps our models learn over noisy images.

The brightness change and the noise are only performed on the satellite images, the ground-truth images are kept the same. All other operations are performed both on the images and their ground-truth.
The data augmentation is done using an Image Data generator combined with some methods that we defined.

After running our models and checking the predictions we noticed that the models are less efficient in classifying inclined roads than the vertical and horizontal roads. Thus, we decided to generate new rotated images with angles randomly chosen from the range [0°, 45°] while still keeping the previous rotations. These last rotation are done with 'reflection' which means that the input is extended by reflecting about the edge of the last pixel to fill the empty parts in the new image.

III. Models and methods

A. Models

Image segmentation is a critical process in computer vision, and Neural Networks showed good performance on these types of tasks. Thus for the purpose of this project, we chose to fully concentrate on neural networks based techniques to try to get good results. First we implemented a traditional CNN model because it is known for being well suited for

image segmentation tasks. Then, in order to try and improve our results, we implemented a U-Net, which is an improved version of CNN.

1) **CNN:**
The CNNs are feed forward neural networks that use convolution operations in order to create fully connected layers. They have the property of using patches. This will help us reduce the number of parameters while training our model without loosing much information. The input images are cropped into patches of 16x16 pixels and fed to the network.
The model consists of 4 convolutional layers and 3 dense layers. We adapted the parameters in order to have the best test accuracy. We also use batch normalization after each convolutional layer in order to avoid over-fitting. The architecture of our CNN model is described in the following table:

TABLE I: CNN Architecture with Layers

Layer	Characteristics
Input	16x16 RGB
Convolutional Block	64 5x5 filters
Activation function	Leaky-Relu
Max Pooling	(2,2), 'same'
Dropout	0.25
Convolutional Block	128 3x3 filters
Activation function	Leaky-Relu
Max Pooling	(2,2), 'same'
Dropout	0.25
Convolutional Block	256 3x3 filters
Activation function	Leaky-Relu
Max Pooling	(2,2), 'same'
Dropout	0.25
Convolutional Block	256 3x3 filters
Activation function	Leaky-Relu
Max Pooling	(2,2), 'same'
Dropout	0.25
Fully connected	128 units
Activation function	Leaky-Relu
Fully connected	64 units
Activation function	Leaky-Relu
Dropout	0.5
Fully connected	2 units
Activation function	Softmax

2) **U-Net :**
The Unet is a neural network architecture mainly used for segmenting images. This architecture works well even when the training set is small. The input image is fed to the network, then the data is propagated through the network to finally get the output. It only contains Convolutional layers and does not contain any Dense layer, unlike our previous model.

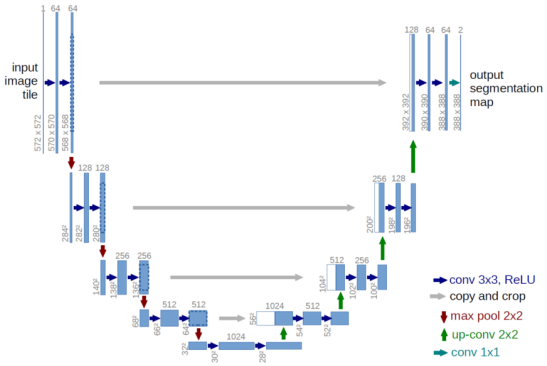


Fig. 2: Example of a Unet architecture for a 572x572 pixels input

The architecture of our Unet model consists of:

- **Contraction path (Encoder):**
  It has the architecture of a typical Convolutional Network. It consists of the repeated application of two 3x3 convolutions, each followed by an activation function (Leaky-ReLU or Relu) and then a down-sampling consisting of a (2x2) max pooling operation with stride 2. At each down-sampling step we double the number of features starting with 16 features. Each convolution step is followed by a Batch normalization and then a Dropout.
- **Bottleneck:**
  It consists of two convolutional layers each followed by an activation function (Leaky-ReLU or Relu)
- **Expansion path (Decoder):**
  It consists of repeated up-sampling of the feature map. The up-sampling performs a 2x2 transposed convolution ("up-convolution") that halves the number of feature channels then concatenates it with the corresponding cropped feature map from the contracting path. Then two 3x3 convolutions are applied on each step followed by a Batch normalization, an activation function (Leaky-ReLU or Relu) and a Dropout. At the final layer a 1x1 convolution is used to map each component feature vector to the two desired classes. In total the network has 23 convolutional layers.

The strength of the Unet model resides in the fact that it uses the same feature maps that are used for contraction to expand a vector to a segmented image. This preserves the structural integrity of the image and reduces distortion enormously.

B. Functions

We describe here the functions that are used in our models:
a) Activation functions[3] :
In the convolutional layers, we used the activation functions 'ReLU' and 'Leaky ReLU' to increase the

non-linearity in our images. They both allow back-propagation, however Leaky-Relu has the advantage of avoiding the vanishing gradient problem encountered by the Relu. Thus we prefer to use the Leaky-Relu. In the final dense layer, we used the 'Softmax' activation function that converts the output to a probability distribution that we use for predictions.

b) Loss function :

We used the 'binary crossentropy' as a loss function. Crossentropy loss functions have the advantage of being independent for each vector component (class), which is useful in multi-label classification. We use the binary one since we have only two classes (road and background)

c) kernel regularizer (CNN):

We used L2 regularizer in the dense layers of the CNN model. The L2 regularization forces the weights to be small but does not make them zero and provides non sparse solution avoiding the risk of Over-fitting.

d) Batch Normalization :

It standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

e) Dropout:

It is a regularization technique that "drops out" or "deactivates" few neurons in the neural network randomly in order to avoid the problem of Over-fitting.

f) Max pooling:

It reduces the dimensionality by computing the maximum value in each patch of each feature map. It allows to highlight the most present feature in the patch. Max pooling works better in practice than average pooling for computer vision tasks like image segmentation, that's why we chose it here.

g) Optimizer[4] :

We used the Adam optimizer to compile the models. The Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. It combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

C. Implementation

The train set contains 400*400 pixels images, while the test set contains 608*608 pixel images. we made several attempts to deal with this problem and get the best results:

- First, we split all the images (train and test) to patches of size 16*16, using the fact that gcd(400,608)=16. the CNN model made a good performance so we decided to keep training it with this input shape. However, the Unet model gave bad results, the train f1-score couldn't do better than 0.576 and the algorithm was also pretty slow. Thus further modifications were made to improve the performance of the Unet model.
- In the second attempt, we decided to keep training the Unet model on images of size 400*400 pixels, thus we feed it the entire images while training. However, for the test set, we split the image of size 608*608 pixels into four images of size 400*400 pixels and feed them to the model. Then we assemble the predictions into an image of 608*608 pixels. To combine the results of the overlapping parts of the images we defined two methods. In the first method we only keep the last prediction for that part of the image and ignore the others. While, in the second method we take the average of the predictions to get the final predicted image. This attempt gave better results and we reached a train F1-score of 0.914.
- Finally, we trained our model on images of size 608*608 pixels. In order to do that, we expand the dimensions of the images in our training set by applying a symmetric crop to the borders. This method also produced good results: F1-score of 0.886 on Aicrowd for the test set.

  The third attempt gave the best results thus the submission was made with it.

D. Callbacks

We use Keras callbacks to implement:

- Learning rate decay when loss has stopped improving for 3 epochs
- Early stopping of the training process when the F1 metric has stopped improving for 5 epochs

E. Metrics

We use the following metrics to evaluate our models:

- **Precision:** The ratio of correctly predicted positive observations to the total predicted positive observations
- **Recall:** The ratio of correctly predicted positive observations to the all observations in actual class
- **F1-score:** The harmonic mean of Precision and Recall
- **Accuracy:** The ratio of correctly predicted observation to the total observations.

F1-score is a better metric when there are imbalanced classes as in our case. In fact, we have more background than roads in the dataset.

IV. Results

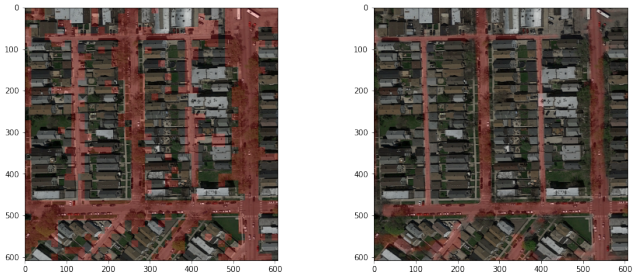
We note in the table below the results we got with the different models. The models were run with a sufficient number of epochs that allowed them to converge. The CNN models were trained with 200 epochs. The Unet models were trained with 300 epochs and took around 60 min to train. Both models were run on Google Colab using GPU.

TABLE II: Models results

Model	Activation Function	Input size	Batch size	Train F1-score	Test F1-score
CNN (old aug)	Leaky Relu (α_1= 0.1, α_2= 0.01)	16*16	16	0.826	0.698
CNN (new aug)	Relu	16*16	16	0.741	0.425
CNN (new aug)	Leaky Relu (α_1= 0.1, α_2= 0.01)	16*16	16	0.795	0.676
Unet (old aug)	Leaky Relu (α= 0.3)	400*400	16	0.914	0.863
Unet (new aug)	Leaky Relu (α=0.3)	400*400	4	0.874	0.880
Unet (new aug)	Leaky Relu (α=0.3)	608*608	16	0.771	0.839
Unet (new aug)	Leaky Relu (α=0.3)	608*608	4	0.850	0.886

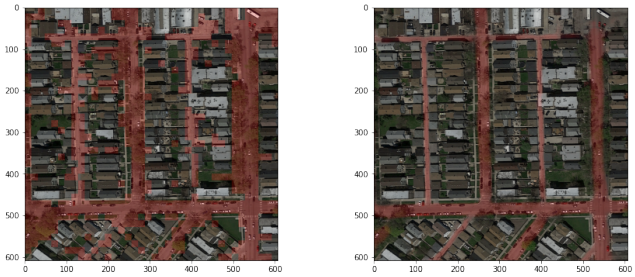
Note: α_1 is for the convolutional layer, α_2 is for the dense layer of the CNN
Note: 'old aug' and 'new aug' represent the data augmentation before and after the additional rotations respectively

We present here pictures of the predictions of the CNN and Unet models respectively on a test image to compare their results. The two models we used are with data augmentation and Leaky-Relu activation function. The Unet model is the one with input size 600*600 with batch size 4. The red parts are the parts that the model classified as road. The pictures on top are those we got before adding the extra rotations in the data augmentation and the pictures above are after this addition:



(a) CNN (b) Unet

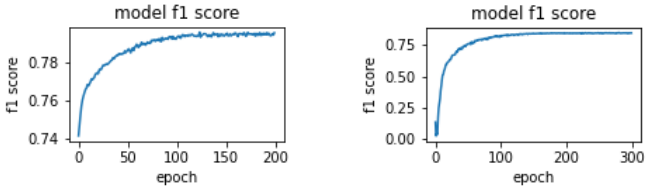
Fig. 3: prediction results on the test set before additional data augmentation



(a) CNN (b) Unet

Fig. 4: prediction results on the test set after additional data augmentation

As we can see the Unet model performs better in both cases, it has a higher capacity of detecting the borders of the roads than the CNN model. We can also see the improvement of the two models at detecting inclined roads after adding the additional data augmentation with the various rotations. The Unet model is also better when dealing with inclined roads.



(a) CNN (b) Unet

Fig. 5: F1-score evolution for one CNN model and one Unet model

As we can see from these plots, the models converge easily with the given number of epochs.

V. Discussion

In the models we treated, Unet gave us better results. However, CNN took less time and memory to run.
The performance of our models can be improved even more if we provide a larger training set, by taking images from google maps and labeling them, but this task is quite time consuming. More hardware resources, like GPUs, would of been very useful to perform more computations on the training and the data augmentation.

Thus for choosing a model and the methods for preprocessing, we shouldn't just consider the model's performance but also the computation cost, the available resources and the time taken by the model.

In our project, we chose to focus on just two neural network models but there were other options we could have explored. For example, we could have tried using some linear classification models for our predictions (logistic regression, random forests). Or some clustering techniques to perform this task (k-means). Finally, there exists other neural networks architectures with good performance like Fully Convolutional network.

VI. Conclusion

As we saw from the predictions, Unet is better at image segmentation than CNN, specially in detecting the borders of the roads. The reason it is able to localise and distinguish borders is that it performs classification on every pixel, so the input and output share the same size. However, the CNN model assigns labels to patches of the image which makes it less precise around the borders.
We also note the importance of a good data augmentation at improving the predictions of the models, thus one should be very careful and try to generalize the train set to all the situations as much as possible.

REFERENCES

[1]https://arxiv.org/pdf/1505.04597.pdf
[2]https://analyticsindiamag.com/convolutional-neural-network-image-classification-overview/
[3]https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/
[4]https://keras.io/api/optimizers/adam/
[5]https://www.aicrowd.com/challenges/epfl-ml-road-segmentation