Project 2 - group_acm
Road Segmentation

Maria Cervera de la Rosa, Clementine Aguet & Axel de la Harpe

Pattern classification and machine learning, EPFL, Switzerland

I. Introduction

Information extracted from aerial photographs is essential for geographic information systems development. Specifically, road extraction has a wide range of applications, such as traffic management, urban planning or car navigation fields [1]. Conventional road extraction is still performed manually, leading to a costly, time consuming and error prone procedure. The question addressed in this project is related to the development of system capable of automatically extracting roads from urban areas. A great advantage of this task is the large amount of satellite image available to train the model, which is not often the case in machine learning problems. Although such systems have already been implemented, one of the main challenges remains the complexity and the diversity of road textures and its surroundings. How to handle the presence of trees, cars or shadows?

The dataset provided for this project consists of 100 satellite images acquired from GoogleMaps, as well as the corresponding ground-truth images in which each pixel is label as road or background. The model is trained on those images in order to segment roads.

II. Models and Methods

To solve this problem, we use two methods. First, a Random Forest Classifier (RFC) upon which more features and complexity are sequentially added. Despite the addition of more and more elaborate features, the performance of such classifier seems to be bounded and the results are not as satisfying. For this reason, we explore as second approach Convolutional Neural Networks.

A. Random Forest Classifier

Building on the provided code snippets on python that uses *scikit-learn* library, we develop a Random Forest Classifier (RFC), which consists of a set of machine learning algorithms based on decisions trees. In order to avoid overfitting, several trees are trained simultaneously by training them on a random subset of the dataset. The output (binary classification here) then corresponds to the most common value across the trees. RFCs have two main characteristics that suit well road segmentation: robust to mislabeled data and easily parallelized [2]. For those reasons, they are frequently used in the literature with successful results [3].

The main parameters to tune in a RFC are the number of trained trees and the maximum tree depth. In our implementation, we decide to have a nearly unbounded tree depth (by setting a very large maximum tree depth at 10,000). The classifiers are built with different number of trees, namely 10, 40 and 70 in order to study potential differences in performance.

Since features in an image depend on several neighboring pixels and not on individual ones, we use patches and compute features in those patches. Our train and test images have between 400 and 600 pixels in each dimension. We start building patches of 16x16 and in a second place try working with patches of 24x24, as it has been suggested that small patches might not provide enough information to build useful features [2].

The critical part of this approach is the construction of the feature matrix to train the classifier. As mentioned, in this approach, we have several features *per patch*. We train the classifier with different combinations of the features and compared the performance. Our most basic features correspond to the mean and variance of the patch either for the grayscale image or for each of the RGB channels, yielding either 2 or 6 features per patch respectively.

As those features do not relate directly to the shapes contained in the patch, we build more informative features. First, the edges are computed using a Canny Edge Detector [4]. This black and white image containing the edges is then used to compute the Hough transform. Using this algorithm, points in an image are transformed to a parameter space where angle and distance are represented. Points aligned in the image space will generate sinusoids that all cluster at a given point in the parameter space. The presence of these clusters in the Hough transform matrix reflects the presence of straight lines in the image [5]. To detect these clusters, we calculate the standard deviation of the obtained accumulator matrix: the higher the standard deviation the higher the number of straight lines in the patch, which are particularly important in roads. So this feature might help us distinguish between road and non-road patches.

As a last step, the features are normalized, based on which the classification is performed at the patch level (0 meaning background, 1 meaning road).

B. Convolutional Neural Networks

The second approach followed is the use of convolutional neural networks (CNN). This approach is less constrained than RFC, in the sense that we do not need specify the important features to the model. Instead, the CNN learns them by itself. As done for RFC model, we divide each image into patches and classify each patches successively.

CNN is preferred to regular neural networks for this project as it is more adapted to classify images and deals with less parameters. Moreover, the spatial structure is a point considered in CNN, where distant or adjacent pixels will be handled in the different way [6]. Another important aspect of CNN is parameters sharing. Assuming that a feature relevant at a certain position should also be relevant at other position [6], neurons in the same activation map will share weights and biases.

Our input layer is 3 dimensional [width x height x depth] with the last dimension corresponding here to the three color channels (RGB).

The elementary unit of a CNN is the convolutional layer, where each neuron is connected to a small region of the previous volume. The receptive field (F) is the parameter representing its spatial extent. We set it to 3, which is commonly used and seems appropriate regarding the relatively small input size of 16x16 or 24x24. Connections are local along height and width (W and H) but cover the full depth of the input [7]. The receptive field is slid over the input volume and the dot product is computed, leading to a 2D activation map. After which, an activation function is applied. The Rectified Linear Unit (ReLU) is commonly employed and chosen here. The stride length (S) is the parameter describing by how much the filter is moved and is set to 1. Zero-padding ($P = \frac{F-1}{2}$) is also employed as it allows to control/preserve the spatial size of the output volume. It consists of adding zeros around the input. A set of such filters builds the convolutional layer, leading to a volume with a depth corresponding to the number of filters (K), which we vary between 32, 64 and 128 depending. The output size can be calculated as [($\frac{W-F+2P}{S} + 1$) x ($\frac{H-F+2P}{S} + 1$) x K] [7].

Pooling layers are also commonly added to CNN architecture after a convolutional layer. It corresponds to a downsampling operation that reduces the spatial size of each depth slice independently (fewer parameters). One of the most frequent and chosen form is the max-pooling of size 2x2 [7], where the maximum activation is taken out of a 2x2 input region.

Dropout is a good approach to avoid overfitting, which is a preeminent problem in machine learning and will lead to a less generalizable model. This regularization means that selected network units will not contribute to the activation of the following layer. The retained units are randomly chosen with probability p [8]. Dropping out 20 to 50% of the neurons gives relevant results and corresponds to what we use. Too low value leads to overfitting, while too large leads to underfitting.

The last element of the network architecture consists of a fully-connected layer[6]. This layer is connected to all neuron of the previous layer and reduces the input into a vector of class scores along the depth, leading to a final volume of [1 x 1 x 256] in our case. A $softmax$ or $sigmoid$ activation function are often applied to the output.

For this project, several CNN architectures are implemented with different patch sizes and classification approaches. We use the *Keras* deep learning library for our implementations, which is a high-level neural network library that can be run on top of Tensorflow or Theano (We use Tensorflow).

At first, the classification is performed for each patch treated as one element. The patch is defined as either road or background. Then a "pixel by pixel" prediction approach is realized, where each pixels of a patch is classified. This method implies that the fully connected layers at the end of the CNN outputs 256 values.

An input with similar dimensions than the output is first given to the neural model. The images are divided in patches of 16x16. But in order to let the CNN access the context of the pixel it is trying to predict, we give larger patches as input (24x24) than the predicted patches (16x16). The predicted patches are the centers of input patches. The model can thus make better prediction knowing the surrounding pixels.

Additionally, several CNN architectures are implemented. The key differences in those models are the order and the number of layers, the number of filters of each layer, as well as the addition and position of maxpooling or dropout layers.

The first model developed is simply composed of 3 convolutional layers each followed by a maxpooling layer. Due to the small input size of the patches (24x24), no more than 3 maxpooling layers can be added in our architecture. In order to overcome this drawback, we apply more convolutional layers before a maxpooling, approach based on VGG architecture [9]. Commonly, two convolutional layers are stacked before a maxpooling operation, allowing us to use 6 convolutional layers. We test this kind of architecture while varying parameters, such as the number of filters, as well as the addition and probability value of dropout layers.

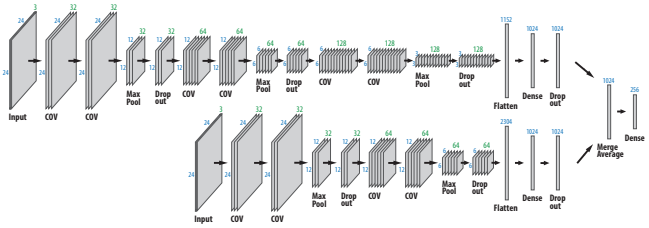

Figure 1. Final CNN architecture

The last and best architecture implemented consists of merging two different CNN. Each CNN is construct as a serie of two similar convolutional layers followed by maxpooling and dropout with increasing number of filters, 32, 64 and 128 filters for the first one. The second one is the same but only goes up to 64 filters. The two models are then merged by computing the tensor average. After which, a fully connected layer and a dropout gives the output of 256 predictions. The architecture can be visualized in figure 1.

Finally, all our CNN were trained on the full training set using mean squared error as loss function and a stochastic gradient descent optimizer. The number of epoch is defined regarding the time necessary to reach a convergence of the loss.

III. Results and Discussion

A. Random Forest Classifier

The scores obtained on Kaggle with the different versions of the random forest classifier can be found in Table I.

Classifier	Nb Trees	Patch size	Nb Features	Hough	Score
LogReg	-	16x16	6	no	0.535
RFC	70	16x16	6	no	0.784
RFC	10	16x16	6	no	0.770
RFC	70	24x24	6	no	0.767
RFC	70	16x16	2	no	0.712
RFC	70	16x16	6	yes	0.787

Table I
KAGGLE SCORES FOR DIFFERENT CLASSIFIER IMPLEMENTATIONS

Looking at those results, we can identify an important performance difference in the performance between a logistic regression classifier (0.535) and a random forest classifier (0.784), which confirms our hypothesis that RFCs are better suited for image segmentation. When optimizing the parameters of our RFC, we find that a forest with 70 trees (0.784) yielded a score higher than with 40 (0.781) or 10 (0.770). We thus keep 70 trees for subsequent analysis. However, it is important to note that not all parameters are optimized (for example classification threshold, maximum number of nodes) and for the remaining ones the optimization was very rough: other values obtained by a refined exploration of the classifier performance will most likely lead to improved results.

Based on RFCs with 70 trees and maximal depth of 10,000, our model is trained with different combinations of features. We find slightly better results using patches of 16x16 (0.784) than 24x24 (0.767), and keep the former for the following analysis. When comparing a classifier trained with 2 (mean and variance for grayscale image) versus 6 features (mean and variance for each RGB channel), the second one yields considerably higher results (0.712 vs 0.784 respectively). Finally, the results are even improved (0.787) when integrating Hough feature.

The steps involved in the construction of Hough feature are presented in Figure 2.

The object edges are accurately extracted from the Canny edge detector. Analyzing these edges using the Hough transform leads to very different results in the presence or not of roads. In particular, we see that in the presence of roads, some angles (near to 0 and 180^o in this case) accumulate many trajectories in one point, showing the presence of straight lines. Therefore in the case of roads, the standard deviation of the Hough transform is very different for angles where straight lines are present. For this reason, when taking the overall standard deviation, we see higher variations in the presence of roads than not (see annotated values in Figure 2).

Overall, a more systematic parameter optimization, combined with the construction of more sophisticated features could greatly improved the performance of our RFCs. However, this process is time-consuming and quite empirical. For this reason, we decide to explore CNN, which have the big advantage of learning the features by themselves.

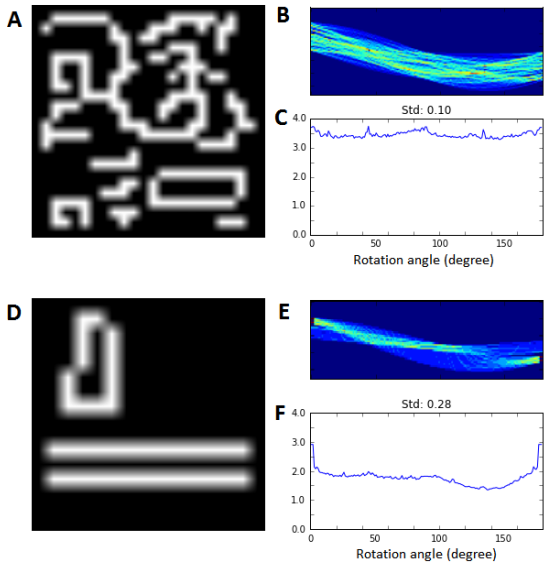

Figure 2. Feature engineering pipeline. A) Edges in an patch without roads from the canny edge detector. B) Hough transform and C) standard deviation of the hough transform for each angle. D) E) and F) represent the same for a patch where roads are present.

B. Convolutional Neural Network

The scores obtained on Kaggle with the different approaches and architectures of convolutional neural network can be found in table II. Overall CNN seems to be a relevant approach for the purpose of road segmentation task with scores above 80 percents, when trained with the whole image set.

Layers	Input	Maxpooling	dropout	epoch	Score
16, 16, 32, 32, 64, 64	16	Every two	yes	150	0.866
16, 32, 64	24	After each	no	70	0.871
16, 16, 32, 64	24	Every two	no	70	0.878
32, 32, 64, 64	24	Every two	no	70	0.880
64, 64, 128, 128	24	Every two	no	170	0.883
Final model	24	Every two	yes	80	0.893

Table II
KAGGLE SCORES FOR DIFFERENT CNN. FIRST, THE NUMBER OF CONV. LAYERS AND THEIR CORRESPONDING FILTER DEPTH. MAXPOOLING IS EITHER PLACED AFTER EACH CONV.LAYER OR EVERY TWO. DROPOUT IS EITHER USED AFTER THE CONV. LAYERS OR ONLY AFTER THE FULLY CONNECTED ONE. THE EPOCH NUMBER TO TRAIN THE MODEL CAN BE SPECIFY.

Although the outcomes are not present in table II, predictions made using a "pixel by pixel" instead of "patch by patch" approach lead to a better accuracy.

Now looking at Kaggle scores, the first important information is the improvement of the results when using inputs of size 24x24 rather than 16x16. Thus giving input patches larger than the output prediction seems to be a good point, as it brings knowledge about the surrounding pixels.

Furthermore, as said previously, applying maxpooling process after each convolutional layer restrains the extension of our network architecture. The scores are lower when only three convolutional layers are used, each followed by a downsampling operation, than with an increased number of convolutional layers and maxpooling every two. As well, the accuracy seems better with more filters per convolutional layer and more convolutional layer.

Building model based on the known VGG architecture, which consists of a sequence of units composed of convolutional layers followed by maxpooling and dropout operation with an increasing number of filters at each unit, seems to be a good approach for our task and giving rise to significant score. This type of architecture gives us the best results and is used for the final model, which additionally merge two models of the kind.

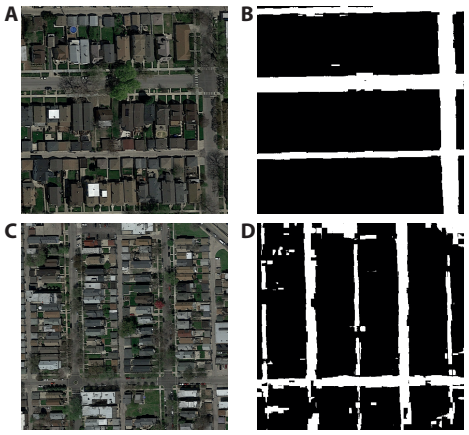

Figure 3. Training and testing prediction obtained with the final model

IV. Conclusion

Based on our work, we can retain that Convolutional Neural Network seems particularly well adapted for this type of image segmentation task, achieving very good results. This type of algorithms supplant Random Forest Classifier in performance from what we could see in this project. Neural networks have another appreciable advantage over the RFC, no effort on feature engineering is required to produce relevant results. While RFC need to be fed with manually constructed features in order to produce good results. The CNN learns by itself the pertinent filters to apply to the images. However, the attention should be paid on the architecture of the network and the parameters defining it. The results are not really surprising, given that CNN are praised for this kind of task in the scientific literature on the subject.

Nonetheless, other applications could have been done to improve the development of such model for the road segmentation task.

The main limitation encountered in this project was the computational time. A possible way to improve our model performance for road segmentation could have been to explore even larger model with more convolutional layers or multiple merging layers. But within the framework that we used to train the networks, it became hard to find others CNN architectures that will produce significantly better results without the need of extremely large computation time and it is clearly constraint.

Additionally, we could have tune more finely the value of numerous parameters of the CNN model by applying cross-validation, such as the dropout value, the size of the fully connected layers, the size of the filters, the loss function, the optimizer type and each of its subsequent parameters. The cross-validation could have also help us to get an estimation of the performance of the model before doing a submission on Kaggle and to increase the reliability of our accuracy test. But this process is fastidious and time consuming, which is principally due to the long computation time that prevented us from doing it (a standard CNN take between 2 to 12 hours to run over full image data set).

One of the best and easiest way to improve the outcomes of our model would be to use data augmentation in order to maximize the relativity small image dataset that we have. This process consists of shifting, shearing, flipping, rotating or applying other transformations to the satellite images and simultaneously to the corresponding ground-truth images. It could furthermore help reducing over-fitting by training the model on more data and showing the network more variable images and orientations of roads. Unfortunately, due to time constraint, we were not able to performs this task and analyses its potential.

The approach of incorporating reccurrent neural network (RNN) to the model was also inspected. RNN are similar to other neural network but with the incorporation of feedback loops, which adds the capacity to learn through not only separate patterns but an input sequence. At first sight, RNN seems more relevant for sequence processing tasks, such as the recognition of temporally extended patterns. But this type of networks can as well be applied to picture, like shown with the pixel RNN model developed by Google DeepMind [10]. This type of neural network might have been well suitable for modeling the continuity of the road to be segmented in an image. Thus a further step of our model development could have been to merge a CNN with a RNN in a same manner as done in our final model, expecting that these two type of neural network would have learned differently and merging them would improve the performance.

Finally, a post-processing treatment could have been performed on the prediction pictures, as when having a look at the outputs we can observe that some isolated or small group of pixels might be missclassified. This method will involve the process of removing the noise and improve the quality of the classified output.

References

[1] Y. Wang, Y. Tian, X.Tai and L. Shu *Extraction of Main Urban Roads from High Resolution Satellite Images by Machine Learning*, January 2006

[2] Volodymyr Mnih *Machine Learning for Aerial Image Labeling*, 2013

[3] Stefan Kluckner, Thomas Mauthner, Peter M. Roth, and Horst Bischof *Semantic classication in aerial imagery by integrating appearance and height information*, 2009

[4] Canny, John. *A computational approach to edge detection.*, 1986

[5] Hough, Paul VC *Method and means for recognizing complex patterns. No. US 3069654.*, 1962

[6] Michael Nielsen *Neural Networks and Deep Learning*, Jan 2016

[7] F. Li, A. Karpathy, J. Johnson *Notes related to the Convolutional Neural Networks for Visual Recognition course*, 2016

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, 2014

[9] Karen Simonyan and Andrew Zisserman *Very deep convolutional networks fo large-scale image recognition*, 2015

[10] A. van den Oord, N. Kalchbrenner and K. Kavukcuoglu *Pixel Recurrent Neural Networks*, August 2016