Convolutional Neural Networks for Road Segmentation

Clément Nussbaumer, Leandro Kieliger, Quentin Bacuet
Department of Computer Science, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

***Abstract*—In this project we train a convolutional neural network to perform image segmentation on aerial images of urban areas. The objective of the model is to be able to identify and separate roads from their surrounding environment. The network comprises four convolutional layers of increasing depth followed by a fully connected one. Dropout and image augmentation have been used to improve the performance of the model as well as diminish overfitting. However, more than the exact architecture of the network, the features fed to the model have shown to be of utmost importance in improving its accuracy.**

I. Introduction

Image segmentation is the process of separating the pixels of an image into distinct groups which share common properties. It has numerous applications in various fields such as biomedical imaging, computer vision and recognition tasks [2][3]. There exists a plethora of different techniques for performing image segmentation but convolutional neural networks have shown to be very efficient for this kind of task [4]. Indeed, simpler linear classifiers usually require smart image preprocessing to produce convincing results. Convolutional models on the other hand have the major advantage that, when properly trained, the network is able to automatically extract useful features.

In this segmentation task we slice satellite images in patches of 16 by 16 pixels with the objective to determine whether the patch under scrutiny represents a road or not. Park places and sidewalks are not considered roads. Section IV introduces a reference model and a basic convolutional network whereas V discusses the weaknesses of the initial implementation. Several solutions are then proposed to improve its accuracy. The final results are summarized and compared in section VI.

II. Exploratory analysis

The dataset is a collection of 100 images of 400 by 400 pixels depicting aerial views of urban areas. A dual set of 100 black and white binary images is also provided as a ground truth to train the model. A sample from the training set is presented in figure 1. In the ground truth set, pixels corresponding to roads are white whereas other pixels, which we will refer to as background, are black. These two colors directly correspond to the classes of our segmentation task. Because the images are sliced in patches of 16 by 16 pixels prior classification and that the ground truth has pixel-level precision, we define the following simple rule for attributing the class label to each ground truth patch:

If more than 25% of pixels in the ground truth image represent roads (and thus are white), the whole patch is considered to be a road.

The patch distribution into those two classes turns out to be as follows: 74.09% background and 25.9% roads. This ratio hints that any model obtaining less than 74.09% accuracy is actually worse than a naive deterministic classifier that would output the background class on any given input.

(a) Aerial view of an urban area (b) The corresponding ground truth image

Fig. 1. Input dataset sample

III. Metrics for performance evaluation

Throughout this report we will use two different metrics. The first one is the accuracy of the model. It measures the proportion of cases in which the model agrees with the ground truth and it will be used as the standard metric to rank the different models. However, as previously mentioned, accuracy does not tell the whole story. Indeed, even a deterministic model would receive honorable accuracy scores. This is why we introduce a second metric called the F1-score. It is defined as follows:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Note that in this case, a deterministic model would not even have a properly defined F1 score. This is because its precision cannot be computed since it never makes any positive predictions.

When it comes to validating models, we usually perform k-fold cross-validation. However, in the case of neural

networks, training can easily take hours, even on powerful configurations. This is why we opt instead for a train/validation/test split of the data. At the beginning of the training process, we randomly partition the initial training set into three separate sets containing respectively $70\%, 15\%$ and 15% of the data. Because the original training set is tiny and some images exhibit unique features such as rivers and sports fields, the downside of this approach is that we take the risk of having either only "simple" images on the test set and therefore overestimating the performance of our model on unseen data or having too little "special" images in the training set and observing the inverse effect. To circumvent this issue we initially implemented an alternative split in which we distributed individual patches across the train/validation/test sets with the intention of introducing more diversity in each of those sets. However, this approach proved to systematically overestimate the accuracy of our models and was therefore discarded. Our interpretation is that when "difficult" unseen *images* are fed to the model, they are consistently "difficult" across their patches. On the other hand, when testing with unseen *patches*, chances are that a very similar patch, coming from the same image, was used for training thus making the model effectively overfit the test data.

IV. Models

We now introduce the models used for the segmentation task. It is important when comparing the performance of models to consider a baseline performance which will serve as a benchmark for the other models. We fix ground zero to be the accuracy of the deterministic classifier. In I, we argue that convolutional neural networks are superior to common linear models on image processing tasks. The rationale is that the latter are unable to capture the inherent underlying structure of images without preprocessing steps whereas convolutional layers are very efficient in finding those patterns. To verify this affirmation we trained a linear model and compared its performance to the convolutional network.

A. Reference model

The linear model is a logistic regression to which we feed the mean and variance of each color channel of the patch as input features. The feature basis is augmented up to degree 4, including interactions. For the implementation, we use Scikit's logistic regression model with $\lambda = 10^5$ as the regularization strength. After 4-fold cross-validation, this model gives $70.666\% \pm 2.352$ mean accuracy and $57.915\% \pm 2.411$ mean F1-score which regarding accuracy is less than a naive deterministic model. As claimed in I, smarter feature preprocessing is necessary to obtain convincing results with this kind of models.

B. A basic convolutional network

The baseline convolutional model is a slight modification of the example provided as starting point for this project. Throughout this text we will refer to this model as CNN_A. It consists of two 5×5 convolutional layers of depth 32 and 64, each followed by a max-pooling layer of size 2×2. Follows a fully connected hidden layer of 512 neurons and the final output layer to which a softmax is applied. The modification we brought to this initial model was to use an Adam optimizer with initial step size 0.001 instead of the original momentum optimizer. The latter was unable to make the model overfit the train data while the former converged to almost 100% after a few epochs for the same validation accuracy. CNN_A reached convergence after 50 epochs, for an accuracy of 81.3% and 64.1% F1-score. It is already a significant improvement over the reference model.

V. Performance improvement methods

In the following subsections we identify possible issues with the initial model and propose solutions for improving its performances.

A. Regularization

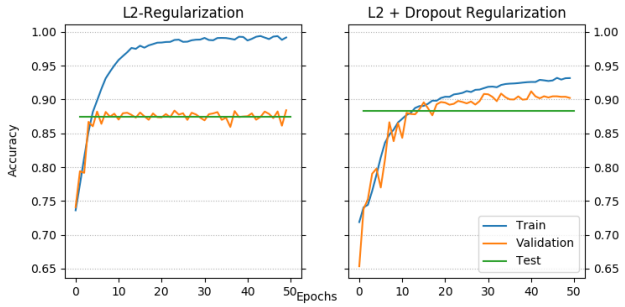

Fig. 2. L2 regularization with $\lambda = 5 \cdot 10^{-4}$ on the left, additional dropout regularization with $p = 0.5$ on the right for a model using a context size of 16 pixels.

Figure 2 shows the accuracy score of the classifier at successive epochs. It is clear that starting from epoch 5, the model massively overfits the training data. Indeed, while the training accuracy keeps increasing, the validation accuracy tops at around 88%. Because the model has almost no error to correct for, we cannot hope for score improvements on the validation set. We mitigate this overfitting in two different ways.

1) L2 regularization: Just like for linear classifiers, we penalize weights tensors with large components. This encourages the model to use all its inputs instead of relying on a few ones. In our model we use $\lambda = 5 \cdot 10^{-4}$. During our experiments we determined this kind of regularization to be only mildly effective. It can indeed bend the learning curve a bit but as shown in figure 2, it does not prevent the model from overfitting.

2) Dropout regularization: Dropout is another regularization technique in which we only keep a neuron activated with a probability p for each training pass of the model. When a neuron is chosen to be deactivated, its output is set to be zero. As seen in figure 2, applying dropout significantly reduces overfitting and produces better results. The downside of this

technique is that the time needed for the model to converge is substantially increased. In our case, by more than a factor two.

B. Patch context

(a) Patch as seen by the classifier (b) Patch with its environment

Fig. 3. Incorrect classification for ambiguous situations

As presented in I, predictions are made for every patch in the image, each one being then fed sequentially to the model. This makes the implicit assumption that patches are actually independent. Indeed, at any given training pass, the model is only given access to a single patch. We argue that this constitutes the major weakness of the model. When roads are occluded by objects such as trees or cars, the model tends to make wrong predictions. Figure 3 shows an example of such as case. It is clear that in that situation, even humans would have a hard time deciding whether the cars depicted on subfigure 3(a) are on a road or stationed on a parking lot (which for this task is not considered a road). In subfigure 3(b) the situation is less ambiguous as the neighboring pixels provide valuable information about the environment. Addressing this issue does not require any changes to the structure of the model. It suffices to feed as input not only the patches to be classified but also their neighboring region. In the implementation, patches near the edges have their neighborhood completed with a mirrored version of the image in order to simplify the patch extraction process. In theory, the bigger the neighboring region, the more information is provided to the model and the better the predictions. In the next chapters we will refer to the neighboring region and its size to respectively the patch context and the patch context size. Table I summarizes the effect of context on the accuracy of the classifier on the test set. The results are obtained by training CNN_A with dropout $p = 0.5$ after each hidden layer.

TABLE I
SCORES FOR VARIOUS CONTEXT SIZES

Pixel border	0px	8px	16px	24px	32px
Accuracy	0.800	0.861	0.884	0.890	0.909
F1-score	0.672	0.756	0.794	0.799	0.833

The results are impressive, we were able to increase the accuracy by several percentage points. As expected, the bigger the patch context size, the higher the accuracy. However, it also heavily impacts the performance of the model as the input size increases as the square of the context size, making each training pass significantly longer.

C. Dataset augmentation

A key observation when training convolutional neural networks on images is that it is possible to apply transformations that preserve their morphology while modifying completely their binary representation. Indeed, a patch representing a road should be classified as such, regardless of the exact orientation of the road. The subsections below explain which transformations were applied to the training set. Note that we chose not to perform any scaling as the size of the infrastructures gives useful hints about whether they are sidewalks or roads.

1) Rotating: Most of the training images exhibit vertical or horizontal roads, with only a few of them being diagonal. By adding 90, 180 and 270 degrees rotations, we can effectively quadruple the size of the original training set.

Because of the aforementioned grid-like structure of roads in the training images, our model performed poorly on some of the diagonal roads of the validation set. Adding 45 and 135-degrees rotated versions of the original training set allowed the model the produce much cleaner predictions on diagonal roads as can be seen in figure 4. We purposefully kept more right angle rotations than acute ones in order not to modify their proportions too much.

Performing acute angle rotations raises the question of how to cope with "missing" image data at the corners. Just like in the case of patch contexts, our approach was to complete the image by mirroring the edges. This can of course produce artifacts but as can be seen in 4, the model performed so much better that we deemed those artifacts to be insignificant.

2) Flipping: Building on the same idea as rotations, adding mirrored versions of the training images allows us to further add twice the size of the original training set.

(a) CNN_A trained with only right angles rotations or flip augmented dataset (b) CNN_A trained with fully augmentations

Fig. 4. Effect of 45-degrees rotations on prediction of diagonal roads

Augmenting the training set introduces the challenge of resources management. Indeed, for each possible transformation we introduce N training samples where N is the cardinality of our original training set. It is therefore not possible anymore to store all the training samples in memory. This issue is aggravated by the use of a context for each patch

to be classified. We circumvent this issue by generating the transformations on-the-fly at each epoch. More specifically, the training set remains untouched until performing the forward pass. Then, one transformation among the set of all possible transformations is applied to every patch of training set. Of course, we define an identity transformation to also include original, untouched, training samples.

D. Complexifying the architecture

The initial architecture proposed is rather simple. This has the advantage of allowing for faster prototyping and performance assessment for the various steps described above. It is possible however to use deeper networks with more parameters. Building neural net architectures is hard and at the time of writing, still often requires a lengthy process of trial and error. Nevertheless, experience has shown that some kind of architectures work better in practice than others[1]. This is why we chose a structure which has proven to be efficient in other image classification tasks such as the ImageNet challenge[5]. The new architecture is based on VGGNet configuration A[6]. Even though it is already a simplified version of the model used for the ImageNet challenge, it is too computationally expensive for us to train. Therefore, we simplify the model by trimming most of the last fully connected neurons. It is arguably a sensitive thing to do since fully connected layers are usually the ones performing the classification work and that we only have two classes from which to chose instead of a thousand. Table II illustrates the architecture of the final convolutional network that we will denote by CNN_B.

TABLE II
FINAL CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE

Layer type	**Parameters**
Input	$80 \times 80 \times 3$
Convolution, ReLU	64 layers 3×3
Max-Pool	2×2
Dropout	$p = 0.8$
Convolution, ReLU	128 layers 3×3
Max-Pool	2×2
Dropout	$p = 0.8$
Convolution, ReLU	256 layers 3×3
Dropout	$p = 0.8$
Convolution, ReLU	256 layers 3×3
Max-Pool	2×2
Dropout	$p = 0.8$
Full, ReLu	256 neurons
Dropout	$p = 0.8$
Softmax	

We found quite difficult to get this model to converge for dropout rates lower than 0.8. Also, other parameters such as larger convolution filter sizes were tested but none showed significant superiority over the simpler 3×3 stack architecture. The same observation was made after adding more fully connected hidden layers, arguably confirming our intuition explained just above.

VI. Results and comparison

The final results are summarized in the table below. It is worth noting that CNN_A has roughly the same performance as CNN_B as long as the input remains individual patches of 16 pixels. However, the deeper structure of CNN_B seems to make it respond much better to the introduction of patch context. Indeed, the shallower CNN_A seems limited in its expressiveness as its accuracy plateaued at around 91% even though we added more training samples. In both cases, it is the addition of patch context which resulted in the biggest leap in accuracy score.

TABLE III
MODELS PERFORMANCE ON TEST SET

Model	**Accuracy in %**	**F1-score in %**
Deterministic model	74.09	undefined
Logistic regression	71.03	58.45
CNN_A	81.3	64.1
CNN_A + $\text{D}_{.8}$	84.0	72.0
CNN_A + $\text{D}_{.8}$ + C_{32}	90.9	83.3
CNN_A + $\text{D}_{.8}$ + C_{32} + A	91.0	82.5
CNN_B	79.7	62.1
CNN_B + $\text{D}_{.8}$	84.1	72.4
CNN_B + $\text{D}_{.8}$ + C_{32}	91.4	82.7
CNN_B + $\text{D}_{.8}$ + C_{32} + A	92.3	84.9

C_N: Context of size N
D_p: Dropout regularization
A: Image augmentation

VII. Conclusion

In this project we trained a convolutional neural network which obtained a final accuracy of 92.3% on our test set, beating as expected and by a large margin the simpler linear model. To improve its performance and reduce overfitting, dropout with $p = 0.8$ and image augmentation were used. Image augmentation comprised mirroring and rotations of all right-angles as well as 45 and 135 degrees. Introducing the concept of patch context was certainly the most important enhancement as it increased the accuracy of the model by several percentage points. Finally, deepening the structure of the network allowed it to capture more information about the neighborhood of a patch. The resulting predictions on a sample from the test set can be seen in the figure below.

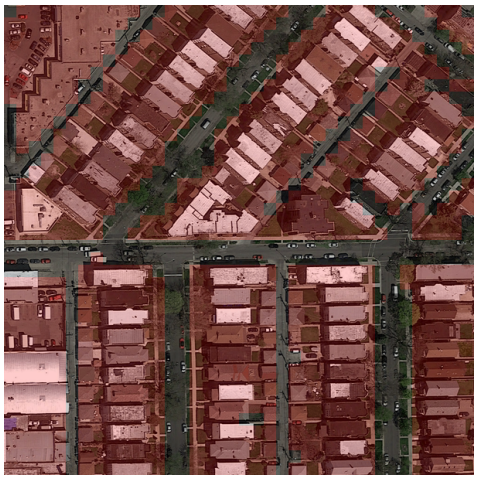

Fig. 5. Predictions of CNN_B on an image of the final test set

REFERENCES

[1] Stanford University CS231n, Convolutional Network Architectues. Fei-Fei Li, Justin Johnson, Serena Yeung; [accessed December 11, 2017]. *http://cs231n.github.io/convolutional-networks/#convnet-architectures*.

[2] Pim Moeskops, Max A. Viergever, Adriënne M. Mendrik, Linda S. de Vries, Manon J. N. L. Benders, and Ivana Išgum, Automatic Segmentation of MR Brain Images With a Convolutional Neural Network, IEEE Transactions on medical imaging, vol. 35, no. 5, MAY 2016, 2014. *http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7444155*.

[3] Liang-Chieh Chen, George Papandreou, Senior Member, IEEE, Iasonas Kokkinos, Member, IEEE, Kevin Murphy, and Alan L. Yuille, Fellow, IEEE, DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs, 12 May 2017. *https://arxiv.org/pdf/1606.00915.pdf*.

[4] Jonathan Long, Evan Shelhamer and Trevor Darrell, Fully Convolutional Networks for Semantic Segmentation. *https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf*.

[5] Jonathan Long, Evan Shelhamer and Trevor Darrell, Fully Convolutional Networks for Semantic Segmentation. *http://image-net.org/challenges/LSVRC/2017/index#comp*.

[6] ImageNet: Main Challenges, 2017. *http://www.robots.ox.ac.uk/~vgg/research/very_deep/*.