Facade Parsing using Deep Learning

Semester Project - Swiss Data Science Center
supervised by Radhakrishna Achanta

Gregoire Clement
EPFL
gregoire.clement@epfl.ch

Abstract

Nowadays, thanks to the tremendous amount of data and computing power at our disposable, deep learning [4] have never been so popular. As a matter of fact, for most computer vision related tasks, older technologies just cannot keep up the pace, in term of performance, that deep learning and especially convolutional neural networks [5] enable. Hence, it seems natural to use this technology to solve the facade parsing problem. In this report we analyze how, given an image of a facade, we can automate the process of recognizing different elements of interest.

1. Introduction

1.1. Motivations

This project is part of collaboration between civil engineers and the Swiss Data Science Center (SDSC). Its goal is to help civil engineers automating the process of evaluating the damage on a building after an earthquake occurred. This report focuses on a sub-part of this task, which is identifying the building structure, where are the walls, windows, doors, etc.

1.2. Goals

Hence, the goals are both to:

1. Develop a well-performing method using today's latest technologies
2. Provide the tools to include it in another project or to extend it

The first goal includes exploring how similar tasks are being solved. Discuss how we could transfer theses technologies to this task. Implement them to get the best performing results.

The second goal, done in parallel with the first, is to make all the work done during this project easily understandable, customizable and extensible. It is done through a well-documented library called `facade_project` as well as code snippets and examples to demo each important part of the project.

2. Approach

2.1. Data

The dataset available is 418 photographs. Each being labeled via the labelme [11] tool.

2.2. Task Definition

First, we define in details what we expect form our model and then go through different approaches to design it. Given an image of the facade of a building as input, our model need to extract the following information:

- where the walls are
- where the windows are
- where the doors are

Hence, one approach is to define for each pixel of the input image whether it describes a wall, a window, a door or neither of these (which we call background). It lets us define the following four classes of interest `{background, wall, window, door}`. This task is well-known, in the literature we talk about semantic segmentation of an image, i.e. giving meaning to its pixels.

Also, another approach is possible, one can predict directly the locations of each object. Then, because walls, windows and doors are roughly rectangles, it only remains to predict width and height of each individual entity. The task becomes a regression task.

One should know that, though both approaches are different, they can be solved at the same time by a single neural network. This is called multi-task learning [1] and can result in improved learning efficiency and prediction accuracy for the task-specific models, when compared to training the models separately.

2.3. Semantic Segmentation

This task is simply doing classification for each single pixel of an input image. This means the output is of the same size and is called a mask, pictured in figure 1.

Figure 1. An example of an image, its mask, and the superposition of the two.

2.4. Heatmaps Regression

This task can be understood as doing regression for each single pixel of an input image. In this project, we regress on the center locations, widths and heights of windows and doors exclusively. We do so because windows and doors are not superposed, unlike walls and windows for example, and also because the dataset does not always distinguish neighboring walls as being different walls.

In order to be able to do regression, heatmaps are created for centers, widths and heights. To do so, each we apply for each polygon representing the windows or doors a gaussian like function centered at the centroid and normalized such that maximal values are 1. This gives us the center heatmap, to build the width and height heatmaps we simply multiply each individual functions by the respective width and height of the rectangular envelope of the polygon representing the window or door. Because the center heatmap has maximal value of 1, we multiply it by a constant factor such that its mean value is roughly equal the ones of the width and height heatmaps. Resulting target heatmaps are shown in figure 2.

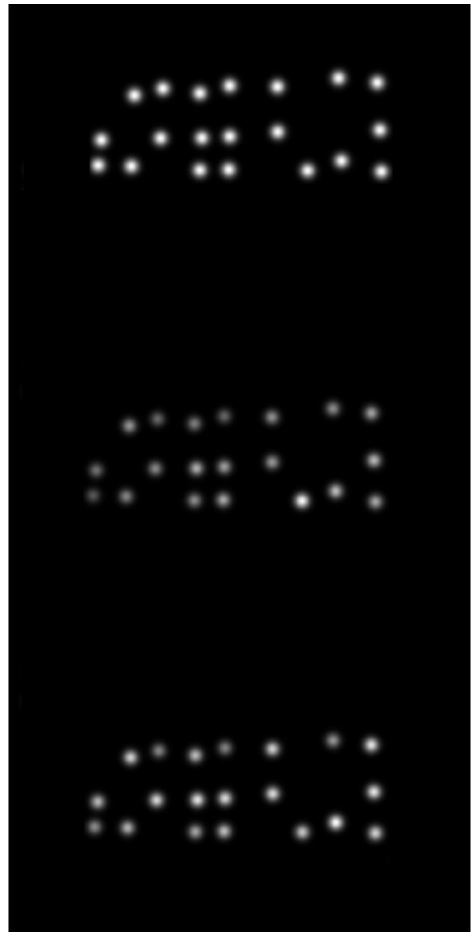

Figure 2. An example of the three heatmaps for the same image as figure 1. It only represents doors and windows. From top to bottom: center, width and height heatmaps.

3. Implementation

Data (inputs) and tasks (targets) being well defined, we can now think about the implementation of our deep learning model. In order to get good performance, one must usually carefully design each of the following steps: the data augmentation pipeline, the model architecture, the criterion (loss function).

3.1. Data Augmentation

Deep learning models are data hungry, the more you feed them data, the better they will usually performs. Though, we have a limited amount of data, one can easily increase it via data augmentation, especially with images. By applying transformations to images, we generate new data, although, the data is not entirely new, it will help the model generalize better. Most of the transformations are done randomly and when possible, on the fly, i.e. right before being fed to the network.

3.1.1 Rotation

Rotating the image of random angle. This must be done offline, as it requires quite some computation power, and is stored on disk for few angles. Only rotation of angles in between the range $[-10, 10]$ degrees are done because a building is not rotation invariant, e.g. a door is usually located at the bottom.

3.1.2 Random Crop

Cropping part of the image. This is particularly useful when we have images of different size and we want to construct batches for the network to train faster.

3.1.3 Random Flip

Randomly flipping the image horizontally and not vertically because of the door being generally at the bottom or the sky at the top.

3.1.4 Random Brightness and Contrast changes

Randomly change the brightness and/or the contrast of the input image, in order for example, to simulate changes of luminosity by the sun.

3.2. Model Architecture

In order to do image processing, one will for sure use convolutional neural networks [5]. But many architectures use this technology. We discuss here two architectures known to perform well on segmentation tasks.

3.2.1 U-Net

This is a convolutional neural network that was initially developed for biomedical image segmentation [7]. It is fully convolutional, this means it can handle inputs of any sizes. The network consists of a contracting path and an expansive path which gives its u-shaped architecture, see figure 3. This architecture is known to work great with limited amount of data.

3.2.2 Albunet

This architecture [9] is U-Net inspired. It uses pre-trained ResNet [3] as an encoder. It is different from U-Net in the fact that it adds skip-connections to the upsampling path,

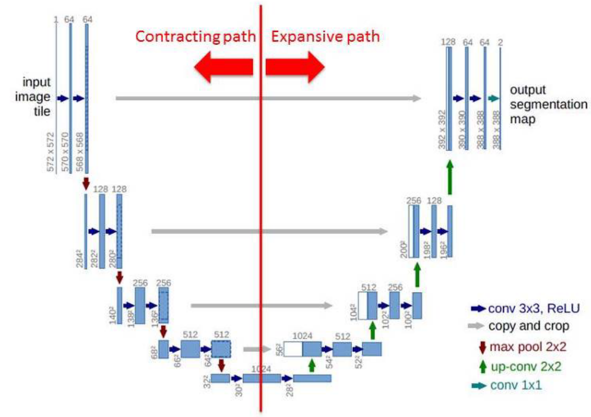

Figure 3. U-Net architecture

see figure 4. Pre-trained encoder enables the networks to re-use the features extractor capabilities of the ResNet network and, as a result, saves training time and increases performance.

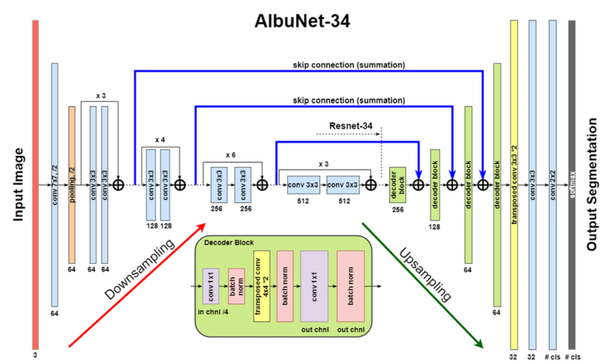

Figure 4. Albunet architecture

3.3. Criterion

The model is trained via stochastic gradient descent. Hence, we need to define a differentiable loss function which we will try to minimize. Before discussing few of them, we need to define Y being the tensor representing the target (or ground-truth) and $\hat{Y}$ being the tensor representing the output (prediction) of our model.

3.3.1 Mean Square Error

The mean square error function is used for the regression onto the heatmaps. It is simply defined as the mean of the difference squared between each output and target pixel.

$$L_{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(Y_i - \hat{Y}_i\right)^2$$

3.3.2 Cross-Entropy Loss

The cross-entropy measures the difference between two probability distributions. Hence, Minimizing the cross-entropy loss is equivalent to minimizing the difference between the probability distribution of our model and the true probability distribution given by our dataset. For this reason, this loss function is widely used in classification tasks. To compute it with tensors, we generally take the softmax of our (hot-encoded) output in order to have probabilities for each class, and then compute the negative log likelihood loss, see figure 5.

$$L_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} Y_{i,c}\log\left(\hat{Y}_{i,c}\right)$$

For further details about it, see [8].

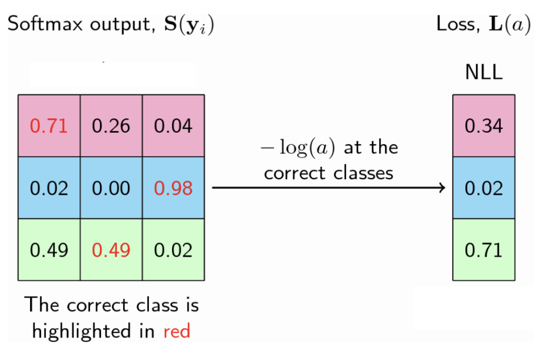

Figure 5. Cross Entropy Loss

3.3.3 Dice Loss

The dice loss is specifically designed for segmentation tasks. It measures for each class the ratio between the number of pixel predicted correctly and the sum of the number of predictions and the total number pixels belonging to the class. The dice loss can be expressed respectively for sets and tensors as following

$$L_D = \frac{2|A\cap B|}{|A|+|B|} = \frac{2\sum_i^N \hat{Y}_i Y_i}{\sum_i^N \hat{Y}_i^2 + \sum_i^N Y_i^2}$$

Compared to the CE loss, it naturally handles well unbalanced classes, where we would requires computing weights for the former. For further details about about it and especially how to make it more stable, see [10].

3.3.4 Facade Loss

Using the dice loss for the segmentation and the mean square error for the regression on the heatmaps, we can define the loss we use for our model.

$$L_F = \alpha L_D + \beta L_{MSE}$$

Where α and β are used to manage the scale of each loss and control how much attention is given to each task. Notice that one could replace the dice loss by a weighted cross-entropy loss.

4. Library

The library, named `facade_project`, we propose is well documented and modularized. Please note that it is entirely written with PyTorch [6] backend. In the following we present some of the main modules. Additionally, some scripts and Jupyter notebooks, showcasing uses of the library, are available.

4.1. `facade_project.data.*`

In this module, one can find multiples ready-to-use PyTorch dataset classes. Also, it includes a whole set of transformations to augment a dataset focused for this project and some nice wrappers around datasets offering features like caching.

4.2. `facade_project.geometry.*`

This module contains all the tools required to apply any sort of transformations on images, masks and even heatmaps. This includes for example cropping background from mask or reconstructing mask from heatmaps.

4.3. `facade_project.nn.*`

In this module, one can find the PyTorch implementation of the two networks architectures proposed. Also, it offers useful functions to compute losses or metrics. Finally, it contains a fully customizable training loop which given a model, a dataloader and criterion trains the model. The loop handles logging of loss and metric, tensorboard, automatic weights saving and can even be stopped and restarted later.

4.4. `facade_project.show.*`

This module contains many useful functions to display images, masks or heatmaps. It is generally very useful in order to verify, for example, that transformations are doing what it is supposed to and also to display results.

5. Results

We finally reach the most subtle part of any project, the results. Overall one can confidently say that the model is able classify the pixels with high accuracy. But accuracy is not sufficient, we need a better metric. Closely related to the dice loss, we are using the Jaccard index to assess performance of the segmentation part of the model for each class of interest.

$$J = \frac{|A\cap B|}{|A\cup B|} = \frac{|A\cap B|}{|A|+|B|-|A\cap B|}$$

In other words, it is simply the intersection over the union (iou), each set here being the pixels predicted and the ground-truth to a given class. The closer to one for each class, the better our model performs.

For the overall task in general, a metric could be the accuracy in terms of windows and doors detection. And that is where the heatmaps come into place. Using the predicted heatmaps we can construct rectangles and, we can assign its label thanks to the segmentation mask.

In the following, a validation set of 10% of the dataset is randomly selected and the network is never fed with this data. The criterion used is the facade loss. Fixing $\alpha = 1$ and playing β shows that both in terms of Jaccard index or rectangles accuracy, the best performing models happen to be the ones with a good compromise on each loss. This is what multi-task learning theory promised us. The model trained with $\beta = 0.002$ yields the best segmentation and $\beta = 0.005$ yields the best rectangles accuracy, as shown in figure 6.

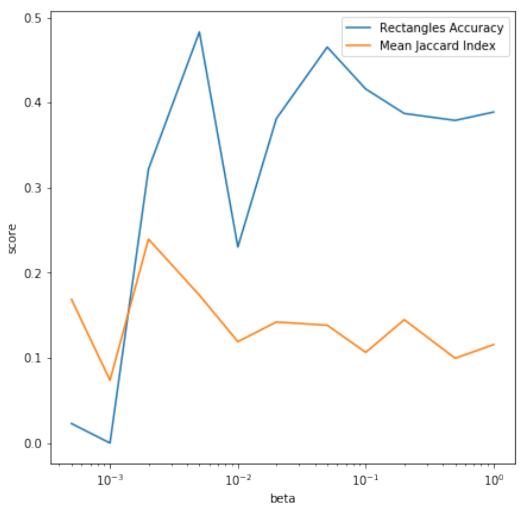

Figure 6. Best score per model with respect to its β. Mean Jaccard index was taken only for window and door classes.

Constructing back the mask from the heatmaps yields to the predictions shown in figure 7. The predictions are done with a Albunet trained for 25 epochs and a loss function defined with $\alpha = 1$, $\beta = 0.005$.

Overall, windows and doors are detected with an accuracy of 76%, but correctly labeled with an accuracy of respectively 75% and 22%. As a matter of fact, most doors are detected correctly but labeled as windows.

Note that no matter the metric or the hyperparameters, the Albunet architecture was performing better than the U-Net for the same number of parameters.

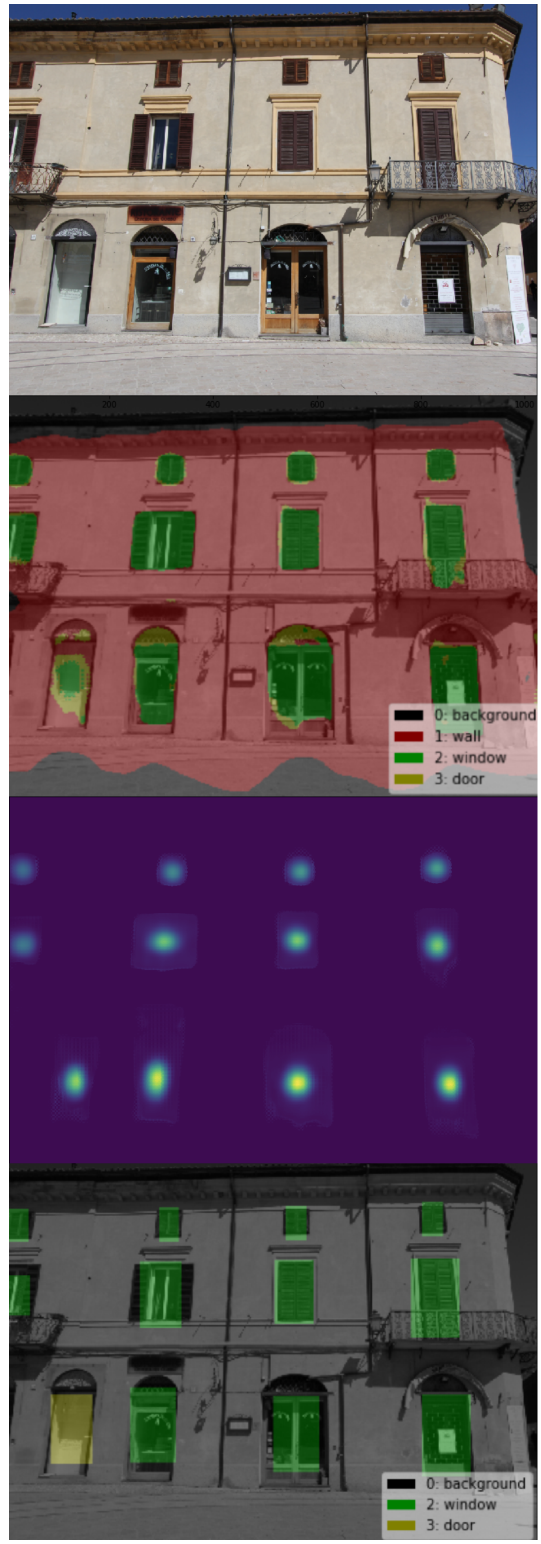

Figure 7. Input image, output mask, output center heatmap, and final predictions using the heatmaps to find rectangles and assigning labels using the mask.

6. Conclusion

In general, we can confidently say that deep learning is a great, well-performing tool to tackle this challenging facade parsing task. Indeed, most of the goals mentioned initially are met. The results are satisfactory, though they could be improved, especially for doors. And in fact, all the tools provided by the library enables easy extension and possible incremental improvements.

7. Further Improvements

Notice that we used heatmaps to construct the final masks as it enables simple reconstruction. Though, there exists few different technologies such Mask R-CNN [2] which also outputs rectangular mask and do the segmentation in a more sophisticated way. This should be explored in order to compare results of each architecture. It would also be able to handle rectangular masks for the walls, which we did not attempt in this project. Finally, extending the dataset would greatly help as 418 images is not much.

References

[1] R. Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[2] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[5] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256. IEEE, 2010.

[6] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[7] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[8] J. Shore and R. Johnson. Properties of cross-entropy minimization. *IEEE Transactions on Information Theory*, 27(4):472–482, 1981.

[9] A. Shvets, V. Iglovikov, A. Rakhlin, and A. Kalinin. Angiodysplasia detection and localization using deep convolutional neural networks. 04 2018.

[10] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 240–248. Springer, 2017.

[11] K. Wada. labelme: Image Polygonal Annotation with Python. `https://github.com/wkentaro/labelme`, 2016.