Parameters estimation from remote sensing data for the generation of virtual 3D city models

Jaume Alexandre Solé Gómez

Supervisors: Michel Roux, Philippe Salembier

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

A thesis submitted for the degree of

Telecommunications Technologies and Services Engineering

Paris, July 2018

Acknowledgements

I would like to express my very great appreciation to Michel Roux and Philippe Salembier, supervisors of this thesis, for their valuable and constructive suggestions during the planning and development of this research work. Their willingness to give their time so generously has been very much appreciated.

I would also like to thank to Alessandro Delmonte, Emanuele Dalsasso, Manuele Sabbadin and Xiangli Yang, their help was really useful for me every time I needed to solve any problem.

Finally, I wish to thank my parents for their support and encouragement throughout my study.

Abstract

Virtual 3D city generation is an essential element in various economical areas such as video games, virtual environments for the film industry, electro-magnetic propagation for the design of telecommunication infrastructures, pollution propagation estimation... Traditionally it must be constructed by hand in a 3D application by a human operator. While this approach gives the operator full control, it requires skills and a lot of time to complete an entire city. In the last years, several tools have been proposed for the automation of the process. However, many parameters such as the width and the length of the roads and the size and the height of the buildings are still set by the operator or randomly generated.

The purpose of this project is to develop new tools for the estimation of these parameters from the automated analysis of remote sensing data. High resolution aerial imagery with vertical and oblique viewing angle are convenient sources to estimate the distribution of morphological parameters related to the shape of the terrain, the size and the height of the buildings, and the structure of the road network.

In this document we will explain an approach for the estimation of some of these parameters and we will test if it's possible to use this approach also for aerial raw images.

Resum

La generació de ciutats 3D virtuals és un element essencial en diverses àrees econòmiques, com els videojocs, els entorns virtuals per a la indústria del cinema, la propagació electromagnètica per al disseny d'infraestructures de telecomunicacions, l'estimació de la propagació de la contaminació... Tradicionalment s'ha de construir a mà en una aplicació 3D per un operador humà. Si bé aquest enfocament li dóna a l'operador el control total, requereix habilitats i molt temps per completar una ciutat completa. En els últims anys, s'han proposat diverses eines per a l'automatització del procés. No obstant això, molts paràmetres com l'amplada i la longitud de les carreteres i la mida i l'altura dels edificis segueixen sent establerts per l'operador o generats aleatòriament.

L'objectiu d'aquest projecte és desenvolupar noves eines per a l'estimació d'aquests paràmetres a partir de l'anàlisi automàtica de dades de teledetecció. Les imatges aèries d'alta resolució amb un angle de visió vertical i obliquo són fonts convenients per estimar la distribució de paràmetres morfològics relacionats amb el terreny, la mida i l'altura dels edificis, i la xarxa de carreteres.

En aquest document explicarem un enfocament per a l'estimació d'alguns d'aquests paràmetres i anem a provar si és possible utilitzar aquest enfocament també per a imatges aèries sense processar.

Resumen

La generación de ciudades 3D virtuales es un elemento esencial en diversas áreas económicas, como los videojuegos, los entornos virtuales para la industria del cine, la propagación electromagnética para el diseño de infraestructuras de telecomunicaciones, la estimación de la propagación de la contaminación... Tradicionalmente debe construirse a mano en una aplicación 3D por un operador humano. Si bien este enfoque le da al operador el control total, requiere habilidades y mucho tiempo para completar una ciudad completa. En los últimos años, se han propuesto varias herramientas para la automatización del proceso. Sin embargo, muchos parámetros como el ancho y la longitud de las carreteras y el tamaño y la altura de los edificios siguen siendo establecidos por el operador o generados aleatoriamente.

El objetivo de este proyecto es desarrollar nuevas herramientas para la estimación de estos parámetros a partir del análisis automatizado de datos de teledetección. Imágenes aéreas de alta resolución con un ángulo de visión vertical y oblicuo son fuentes convenientes para estimar la distribución de parámetros morfológicos relacionados con la forma del terreno, el tamaño y la altura de los edificios, y la estructura de la red de carreteras.

En este documento explicaremos un enfoque para la estimación de algunos de estos parámetros y probaremos si es posible utilizar este enfoque también para imágenes aéreas sin procesar.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work has been performed in the research team Image, Modeling, Analysis, Geometry, Synthesis (IMAGES) of the department of Image, Data, Signal department [IDS] of Télécom Paristech in Paris (France).

The generation of virtual 3D cities is becoming increasingly popular and increasingly necessary for various fields of our society such as video games, virtual environments for the film industry, electro-magnetic propagation for the design of telecommunication infrastructures, pollution propagation estimation... In our case, Télécom Paristech is working on a virtual 3D city generation system to have a realistic model to perform simulations of the propagation of electromagnetic waves of an antenna in a specific city, explained in [19] and [3].

The main problem of these works is that we need to put the parameters manually and this process requires a lot of time, so our idea is to try to compute these parameters automatically. The parameters that we need to calculate are:

- **Mean perimeter of the typical block**: A typical block means the smallest area that is surrounded by streets. The typical blocks are the space for buildings within the street pattern of a city, and form the basic unit of a city's urban fabric
- **Anisotropy ratio**: According to [15], for two directions, the anisotropy ratio is the area of the parallelogram which is spanned by these two unit vectors. In other words, we have to estimate the distribution of the streets in the image.
- **Mean street width**: The average of the distances between one side and the other of the street.

- **Mean building height**: The average of the distances between the floor and the top of the building.
- **Mean length of a building facade**: The average of the length of the walls that are facing to the street of a building.

In this project, we also want to expand the idea proposed in [9] where segmentation and classification of immovable objects such as buildings or roads of ortho-corrected aerial images were made using a *CNN*. The Convolutional Neural Networks (*CNN*) is a type of neural network where the neurons correspond to receptive field, like the neurons in the primary visual cortex of a biological brain. This kind of networks is a variation of a multilayer perceptron but using two-dimensional matrices. They are very effective for artificial vision tasks, such as classification and segmentation of images, among other applications. Our idea is try to improve the network used in [9] and also try if it's possible to extend the range of the objects that the network can segment. In this paper, images are ortho-corrected, in other words, images have been processed so that all elements have the same scale, free of errors and deformations, with the same validity as a cartographic map. An additional goal of this work is to verify if we can use aerial images without this correction for the same *CNN*.

Our approach is to use convolutional neural networks (*CNN*) trained with ortho-corrected images to be able to estimate the previously mentioned parameters automatically and to verify if the same *CNN* can be used to extract parameters of aerial raw images (non ortho-corrected).

The project main goals are:

1. The estimation of the distribution of structural parameters.
2. The estimation of these parameters using both ortho-corrected and aerial raw (non ortho-corrected) images.
3. Implementation of the segmentation and the classification of the city using convolutional neural networks (*CNN*).

Chapter 2

State of the art of the technology used or applied in this thesis

There are several approaches that use convolutional neural networks to solve image segmentation problems like [7] [16] [2] [8] [12] [21]. They performs really well and achieves good results, but our approach is more focused on image segmentation of aerial images. In the field of aerial image segmentation using convolutional neural networks, we can see that there is some literature talking about that like [9] [14] [13] [10] [17].

Paper [9] talks about creating a dataset from *Google Maps* for the imagery and using *OpenStreetMap* as a ground truth and also presents an architecture to segment aerial images inspired by the *FCN 8's* architecture [12]. OpenStreet map is a collaborative project to create a free editable map of the world and the Fully Convolutional Networks (FCN) is a kind of Convolutional Neural Network for which the output is also an image instead of being only a numeric value. The paper also talks about training the network with a huge dataset of a lot of cities in order to learn what a city "look like" and after fine tuning using small dataset to improve the results.

For the parameters estimation, there is not any precise paper or method because this is a very specific task for that work. For calculating these parameters we used different tools like the *Histogram of Oriented Gradients (HOG)* [4], the *Euclidean Distance Map* [5] and some morphological transformations.

Chapter 3

Methodology

In this chapter, we will explain the entire procedure we have followed for the realization of this project, we will begin by explaining how we created our training dataset and we will continue to explain the *CNN* architectures we have used for our experiments. We will also talk about the experiments we have made with aerial raw images, the semantical resolution from our system and the procedure performed to estimate the parameters we need.

3.1 Generation of the training and validation datasets

For the generation of the training dataset we have used the method described by [9]. Using the QGis open source geographic software, we have synchronized perfectly the images of Google Maps with the maps of OpenStreetMap to use them as ground truth. Using this procedure, we have generated a small dataset of the city of Toronto (Canada) and we have also taken the dataset already made for the paper [9], this dataset consists of the cities of Chicago, Paris, Zurich and Berlin. For the Toronto dataset, we downsampled the image in order to match the resolution from the International Society for Photogrammetry and Remote Sensing (*ISPRS*) aerial raw images. We split our dataset using 70% for the training and 30% for the validation in order to avoid overfitting while training.

This dataset is labeled in such a way that the background is in white, roads in blue and buildings in red, as shown in figure 3.1. In the case of our Toronto dataset we have tagged using the same procedure.

For the aerial raw images we used the Toronto Dataset from the *ISPRS* to test what happens with our *CNN* trained with ortho-corrected images when the input is an aerial raw image. The International Society for Photogrammetry and Remote Sensing (*ISPRS*) is a non-governmental organization devoted to the development of

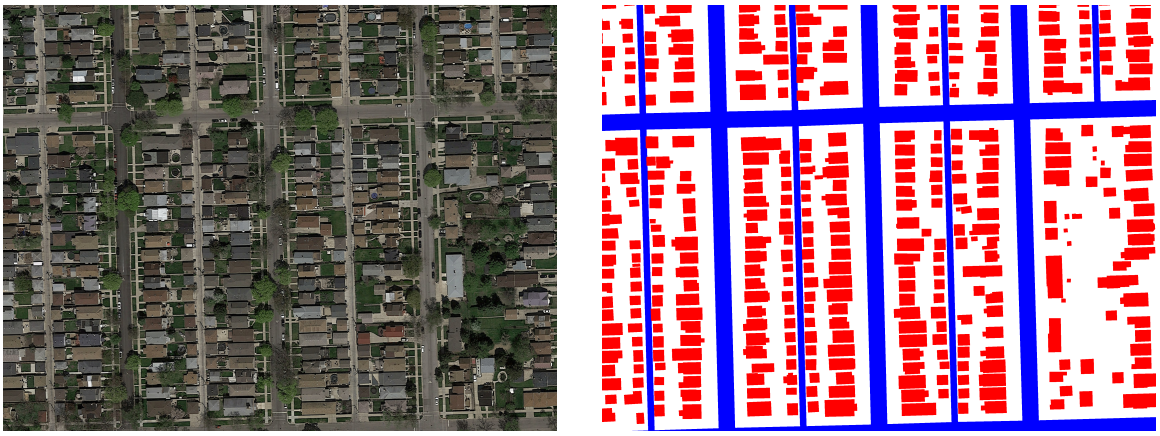


Figure 3.1: Example of a Chicago image from our dataset and the respective ground truth.

international cooperation for the advancement of photogrammetry and remote sensing and their applications. In the figure 3.2 we can see an example from the Toronto Dataset from the *ISPRS* and in the table 3.1 we can see the resolution, the Ground Sample Distance (*GSD*) and the number of images for each city. The GSD represents how many centimeters of the ground is represented by a pixel.

Figure 3.2: Example from the *ISPRS* Toronto Dataset.

	Resolution	GSD	Number of images
Chicago	3000 x 2500 pixels	11.1 cm	497
Paris	3000 x 3000 pixels	9.8 cm	625
Zurich	3000 x 2500 pixels	10.1 cm	375
Berlin	2500 x 2000 pixels	9.1 cm	200
Toronto	4000 x 4000 pixels	15.4 cm	8
Toronto ISPRS	11500 x 7500 pixels	15 cm	10

Table 3.1: The specifications of our dataset

3.2 Neural Network Implementation

In recent years, the world of neural networks has advanced a lot and is increasingly used.

As we have explained before, in this project we will focus on the convolutional neuronal networks (CNN) for the accomplishment of our task. CNN is a type of neuronal networks that simulate the primary visual cortex of the human brain by having a receptive field in each neuron instead of normal neurons. These neural networks are very useful for image processing tasks.

For our neuronal network we have used several types of layers:

- Convolutional layers are responsible for the convolution of all regions of the image by the different filters, these layers have a functional activation known as ReLU so that the resulting matrice does not have negative values.
- Pool layers are responsible for performing downsampling in the image.
- Dropout layers are responsible for losing a percentage of the values of the matrices to prevent the neuronal network from becoming too specialized in the task we are doing.
- Deconvolutional layers are responsible for carrying out the convolutions as well as the convolutional layers but also increasing the vision. Athough we call deconvolutional layers, this layers are not doing a mathematical deconvolution, so in many papers they call this layer transposed convolution or convolution with fractional strides.

- The Fuse layers are responsible for adding the values of the matrices of two different layers.
- The Skip layers are responsible of connect one output from one layer to the input of another layer skipping some layers between booth of them.
- The layer with the SoftMax function is responsible for assigning the final value to each pixel, depending on the probabilities of each class.

We did some research in order to see which was the best architecture for image segmentation and we realized that the one explained in [16], called U-NET, is one with the best results. We also realized that the network explained in [9] is similar to U-NET [16], that's why in our project we used an adaptation of what is used in [9] but with small variations in order to make [9] closer to the U-NET [16]. Considering that [9] was already an adaptation of FCN 8's [12], we decided to carry out various experiments with all the architectures to see if there was an improvement. In Appendix A you can see all the implemented networks that we tested. In our case, we used the VGG19 architecture for the initial layers instead of the VGG16 architecture that they used, both of them explained in [18]. Visual Geometry Group (*VGG*) is a department from the Oxford University that created this architecture that scored first place on the image localization task and second place on the image classification task in the Image Net Large Scale Visual Recognition Challenge (ILSVRC) in 2014. Localization is finding where in the image a certain object is, described by a bounding box. Classification is describing what the object in the image is. This predicts a category label, such as "cat" or "bookcase". Basically, the most important change between *VGG16* and *VGG19* is that *VGG19* adds 3 filters more to the architecture that help to extract better the features of the image. The VGG19 layers from our project were already pretrained.

Another important change is that in our implementation we used one more skip layer connected to Pool1, since we had observed that in other implementations as in the case of U-Net [16] that improved producing sharper edges and reducing the *saltpepper effect*. The *saltpepper effect* is an effect that simulates kind of noise similar to the salt and the pepper when the segmentation it's not good enough. In figure 3.5 we can see an example of this effect. All the Pool layers downsample the image by the factor of 2 and both Dropout layers have the probability of 0.5. At the end we added a SoftMax function in order to convert the class probability to the final

value of the pixel. In the figure 3.3 we can see the exactly architecture that we have implemented.

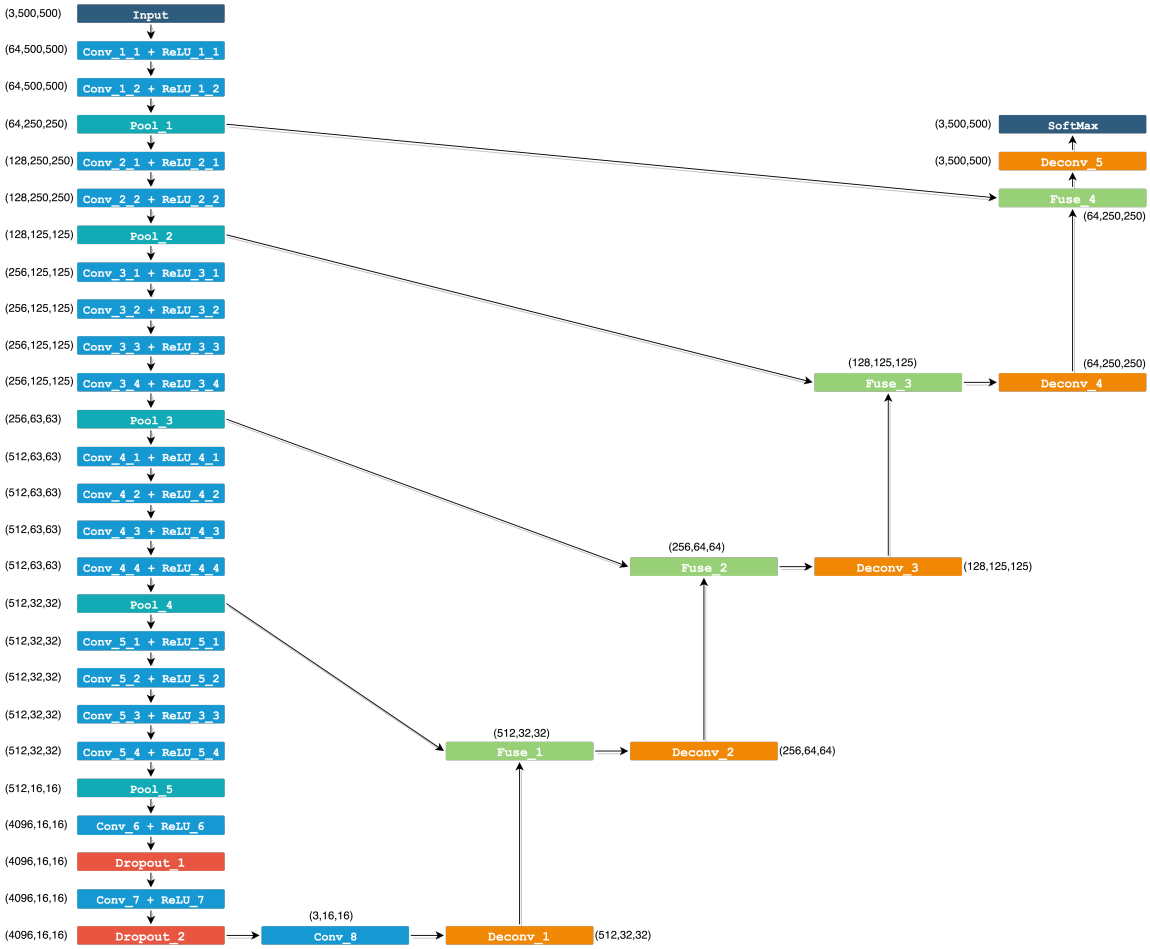


Figure 3.3: The *CNN* implemented for our project.

For the input, we normalized dividing by 255 our pixel values in order to have the values between zero and one, and we split the image in 500x500 pixel patches. We used the Cross-Entropy for the loss function and we evaluated the F1-score after each mini batch. The learning rate was set at $5 \cdot 10^{-5}$ and a batch size of 10 patches of $500x500$ pixels. In the literature we can find some optimizers such as Adadelta [20] and Adagrad [6] but for our implementation we used Adam Optimizer [11] because we saw that in the majority of the cases this optimizer achieves the bests results. The networks were trained until the Cross-Entropy and the F1-Score does not improve more, always checking that don't overfit with the validation dataset. The CNN has been implemented using the TensorFlow API for Python.

3.3 Aerial Raw Images

For the aerial raw images we trained the network with all the cities and when the loss functions stop improving we fine tuned with the small dataset that we make only of the city of Toronto from *OpenStreetMap* and *Google Maps*. We tested the trained network using the *ISPRS* from Toronto. We also tested to calculate the parameters using the aerial raw images to see if its possible. In the Results section we will talk about the results of this experiments.

3.4 Parameters Estimation

For the creation of the virtual 3D city model we need to estimate the five parameters mentioned before. In this section we will explain a method to estimate 4 of them because to estimate the mean height of the buildings is necessary to have some depth information that we don't have in the *Google Maps* images. For all the outputs we reconstructed the patches in order to have the same dimensions as the original images. All the parameters were estimated using the *OpenCV* library for *Python*.

3.4.1 Mean perimeter of the typical block

A typical blocks corresponds the smallest area that is surrounded by streets. The typical blocks are the space for buildings within the street pattern of a city, and form the basic unit of a city's urban fabric.

To calculate the average perimeter of the typical block we have taken the resulting image from our neural network and we have isolated only the areas surrounded by streets, as can be seen in fig.3.4. In order to reduce the effect of saltpepper and clean the image, we apply an opening and a closing with a kernel of 5x5 pixels, in fig. 3.5 we can see an area of our original image, the same area after the opening and the closing where we can notice the difference. Once the image is cleaned, the pixels are labeled using connected components with 8 connectivity, meaning that all pixels that are connected horizontally, vertically or diagonally to the pixel belong to the same region.

Figure 3.4: The resulting image from our neural network isolating only the areas surrounded by streets.

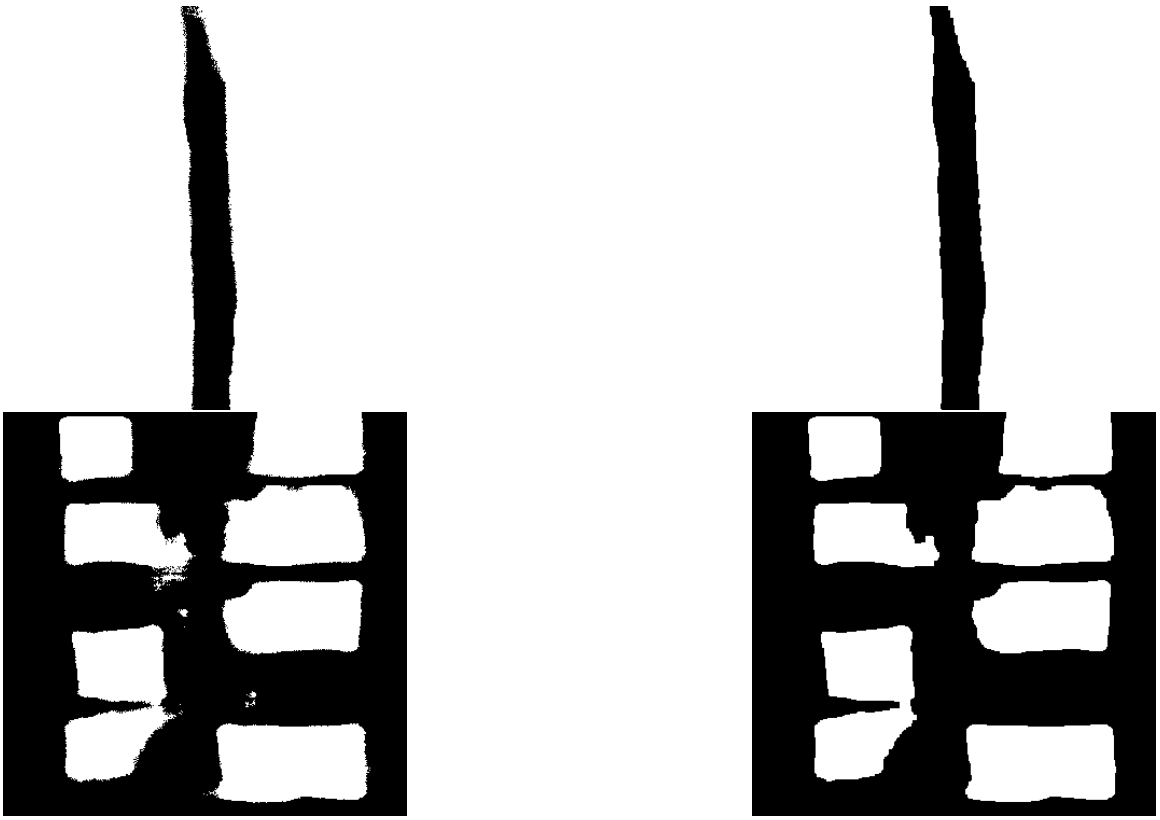


Figure 3.5: In the left column we can see some examples of a small regions of the output of our *CNN* and in the right column we can see some examples of the same images after the opening and the closing. As we can see, this process reduces a lot the saltpepper effect.

Once the image is correctly labeled, we calculate the average perimeter of the labeled regions. In order to obtain more information about the blocks, we have also calculated the entropy between all the perimeters in order to see if all the buildings are more-less of the same size or if all the buildings have a different sizes, and a histogram to see the dispersion between the perimeters of the different blocks. In figure 3.6 we can see an example of the histogram for the perimeters and the labeled regions.

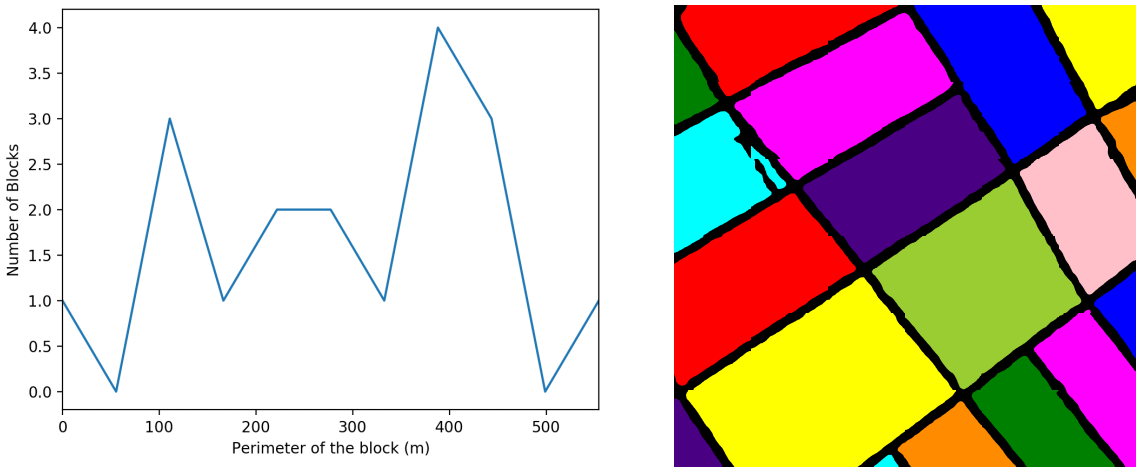


Figure 3.6: In the left we can see an example of the histogram for the perimeter of the blocks and in the right resulting labeled blocks.

3.4.2 Mean street width

In this section we need to calculate the mean between the width of the streets of the image. Similar to what we have done in the section dealing with the mean perimeters of a typical block, we take the reconstructed image and we perform an opening and a closing using a 5x5 kernel but this time isolating the streets instead the blocks, as we see in the figure 3.7.

With the resulting image we calculate an EDT (Euclidean Distance Transform), explained in [5]. Basically, the information that this map provides us is the euclidean distance from a certain pixel to the nearest black pixel. On the other hand, from the closing image we calculate the morphological skeleton of the streets, in other words, we calculate which pixels represents the center of the street. In the figure 3.8 we can see an example of the EDT and the skeleton.

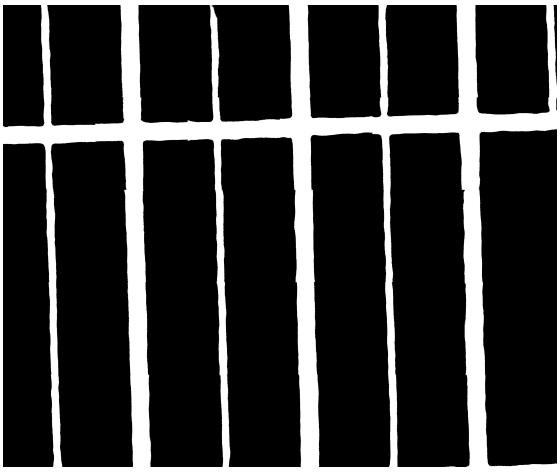


Figure 3.7: The resulting image from our neural network isolating only the the roads.

Figure 3.8: In the left we can see an example of the skeleton of the roads and in the right we can see the EDT of the roads of the same image. We inverted the colors of the images in order to make it easier for the reader.

Once we have the EDT and the skeleton, we look at the values of the EDT at the points where the skeleton indicates that that pixel is the center of the street and then we multiply by two, since we want the total width of the street, not only of center to the end. Following the same reasoning as before, we calculate the entropy and the histogram to see the dispersion between the different widths of the streets. In the figure 3.9 we can see an example of the histogram for the different widths of the streets.

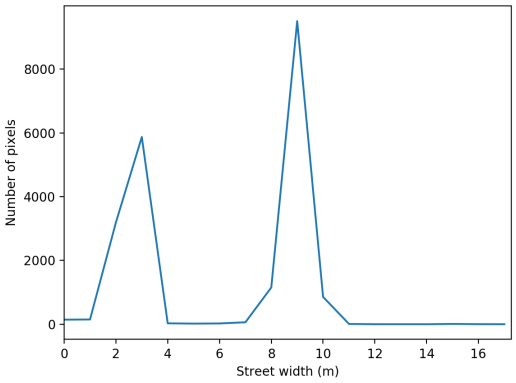


Figure 3.9: An example of the histogram for the width of the streets.

3.4.3 Anisotropy ratio

According to [15], for two directions, the anisotropy ratio is the area of the parallelogram which is spanned by these two unit vectors. In other words, we have to estimate the distribution of the streets in the image. In our case, we only need to compute the directions of the streets.

For computing this, we isolated the streets in the image and we also did an opening and a closing with a 5x5 pixel kernel for cleaning the image. Afterwards, we compute the morphological skeleton of the streets and then compute the gradients of the 8x8 pixel area for making and histogram of all the directions of the image. In the figure 3.10 we can see the histogram of the directions of the gradient.

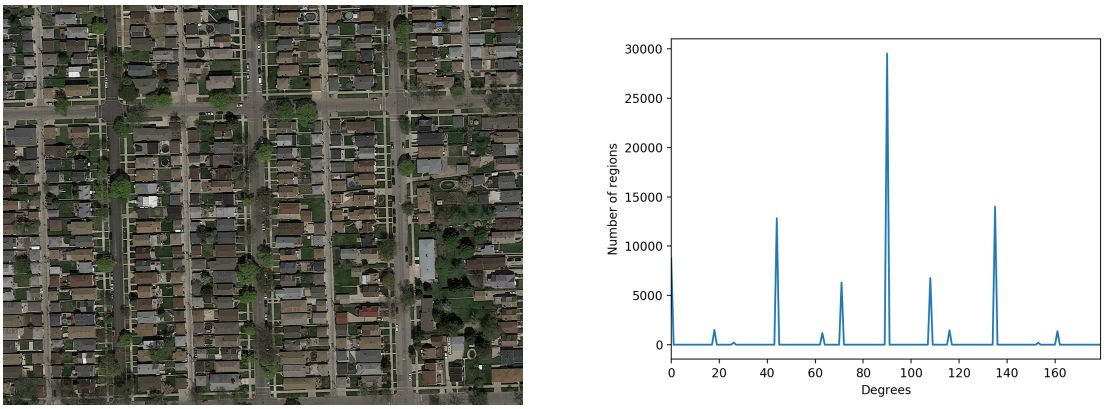


Figure 3.10: An example of the histogram for angles of the orientation of the streets.

3.4.4 Mean length of a building facade

A facade is, by extension, any outer surface of a building. Although by default, when speaking of the facade, reference is made to the front or main facade, indicating more data if not (back facade, north facade, etc.). In our case, we need to indicate all the walls that are facing outside and calculate the mean length. To calculate this parameter we have used some morphological transformations in *OpenCV* library for *Python*.

For our approach we have isolated the buildings of the output image of our neuronal network and we have also applied an opening and a closing with a kernel of 5x5 pixels to clean the image. With the resulting image we apply the algorithm Canny Edges Detection, explained in [1], to obtain the edges of the buildings. Figure 3.11 shows the image after performing the opening and closing and the image after applying the algorithm Canny Edges Detection.

Figure 3.11: Example of the isolated buildings after the opening and the closing and the same image after the Canny algorithm.

Taking advantage of the already labeled blocks that we have generated in the section to calculate the mean perimeter of the typical blocks, we eroded the blocks independently until they collide with the pixels of the Canny edge image. At the moment when one of the pixels goes out of what would be the region of the block, the system stop to erode and label the pixels that have gone out like facades. We realized that in some cases the distance between the road and the building was not the same for all the block, making that in some cases the algorithm only detect the facades of one side of the block but not in the other. For solving that, we made asymmetrical kernels with all the eight directions of the plane, as we can see in the table 3.2, and

then we erode the same block with each kernel independently and after we did a logical *OR* with all the detected facades of all the erosions. Figure 3.12 shows the detected facades using the different kernels and in the figure 3.13 the combination of all of them. In the figure 3.14

Figure 3.12: The facades detected using different kernels for the erosion.

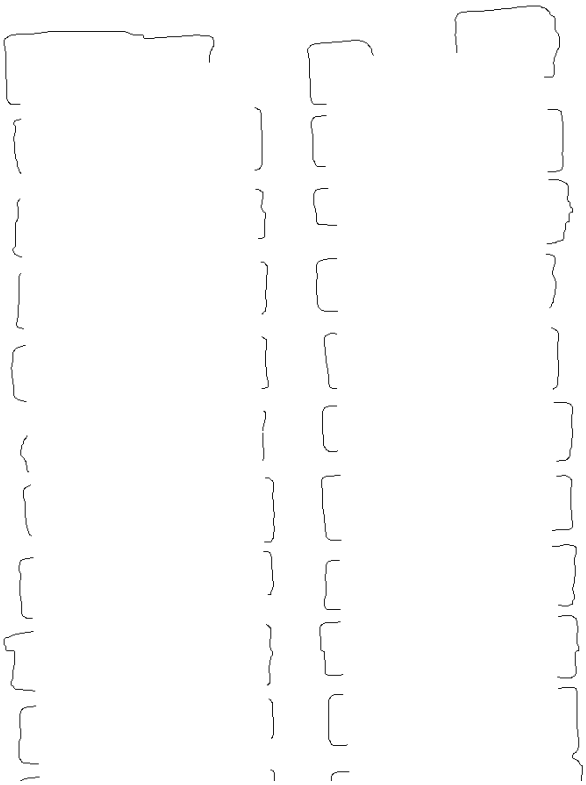


Figure 3.13: The result of the combination of all the erosions.

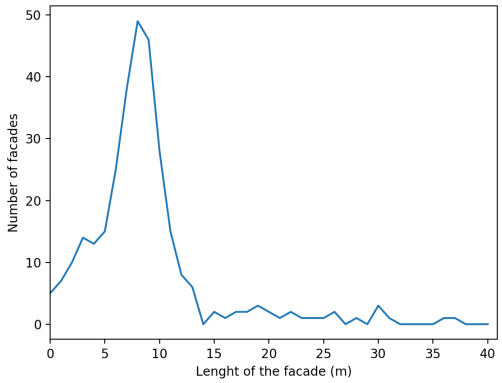


Figure 3.14: An example of the histogram for the length of the building facade.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Table 3.2: The different kernels used for the erosions.

Chapter 4

Results

In this section we will discuss the results obtained using our methodology. First of all we will discuss the results of our neural network and we will compare with the results of similar neural networks that we tested. Afterwards, we will discuss if our neural network trained with ortho-corrected images can be used to segment aerial raw images. To finish, we will discuss the results of our approach for estimate the parameters that we need.

4.1 Neural Network

For the neural networks we tested some experiments in order to compare them with our approach. First, we tested the FCN 8s explained in [12], then the network described in [9] and finally, our approach. All of the three networks were trained using our big dataset and after tested with our validation dataset. In the table 4.1 we can see the results of the validation dataset for our experiments.

	F1 Score	CrossEntropy
FCN 8s	70.32 %	0.57
FCN from [9]	85.49 %	0.42
Our FCN	91.98 %	0.095

Table 4.1: The results of the *CNN* that we tested in this project.

Checking the results, we can consider that adding this third skip layer to the network helps to improve the image segmentation, this skip reduces a lot the saltpepper effect and also makes the edges sharper. The training time of our neuronal network using our database was 80 hours using a cluster from the university with an nVidia Tesla K80 graphics card. Once the neuronal network converged, the resulting error function was 0.095. In the figure 4.1 we can see two examples of our dataset with the

reconstructed output overlay.

Another point we wanted to verify was whether our neuronal network could segment more classes than the two we needed to estimate our parameters. For the neuronal network, does not suppose any problem, but the network must be retrained each time you want to increase or reduce the number of classes. With regard to our database, OpenStreetMap has more tags than roads and buildings, such as trees, rivers, parks, etc. reason why using the same method mentioned previously but creating a ground truth with more classes.

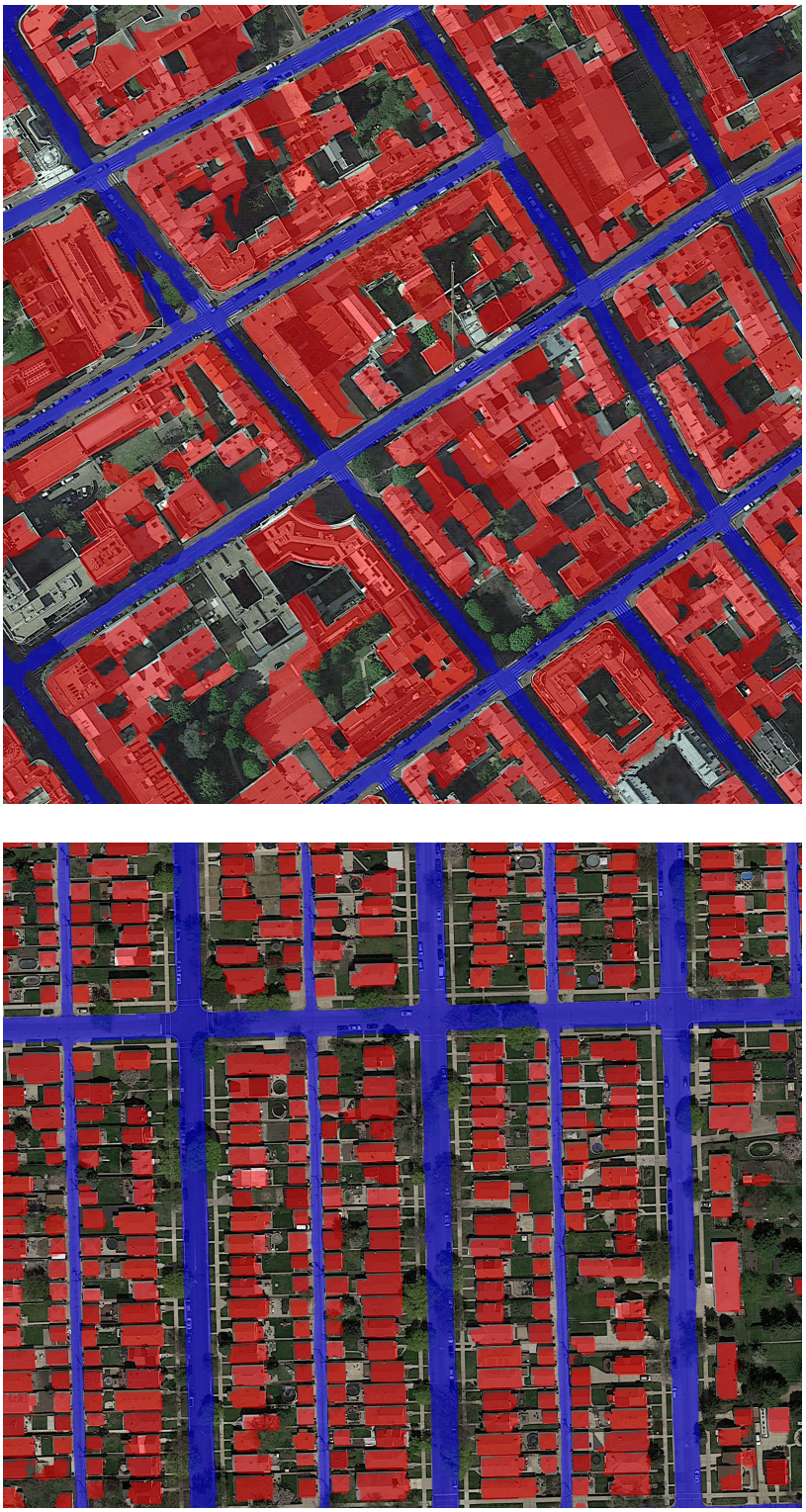


Figure 4.1: Example of two reconstructed images from the output of our neural network. The top image is from Paris and the below one from Chicago.

4.2 Aerial Raw Images

For the aerial raw images we wanted to test if was possible to estimate our parameters using non ortho-corrected and non labeled images given by *ISPRS*. For this case we first trained with all the dataset and after fine tuned with a small dataset of Toronto.

The main problem with the *ISPRS* dataset is that it is not labeled, so we cannot have a numeric result such as F1-Score or Accuracy of the segmentation. Apart from that, we can visually see if the network can segment properly but we have some problems with the dark areas or with the areas whose color and contrast are quite different to our dataset. We can also realize that we notice more the edges of the 500x500 pixel patches that we used for our network. In the figure 4.2 we can see an example of the output of our neural network using aerial raw images. We also saw that, for the segmentation of the aerial raw images, helps a lot to train with all the dataset and after fine tuning with small dataset of the city, but for ortho-corrected images we saw that you don't need to fine tuning, the network segments well with the training only with all the cities.

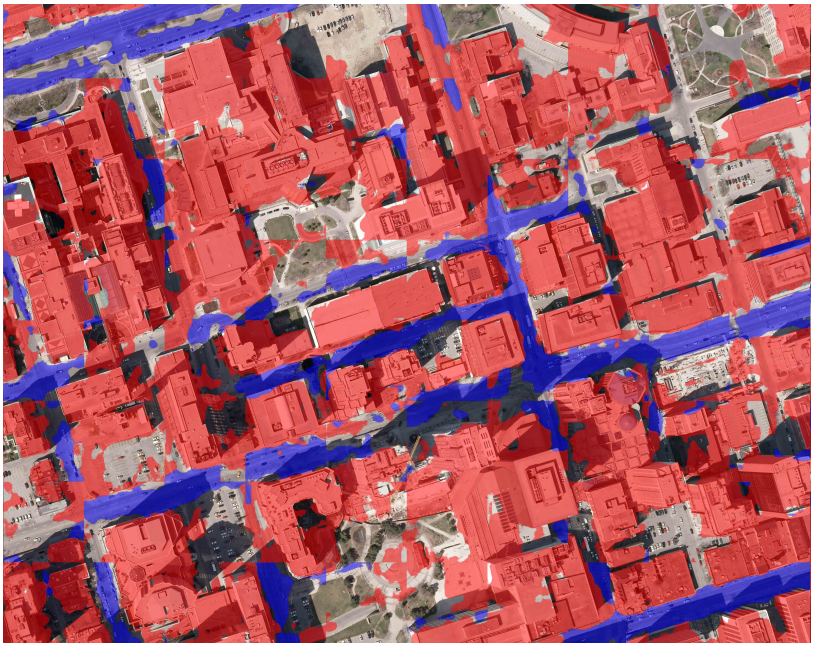


Figure 4.2: The output of our neural network using aerial raw images.

The estimation of the parameters is quite complicated because the segmented regions are not well as defined as in the ortho-corrected photos. For this reason, we can have problems for the detection of facades due to the fact that the segmented

region does not adapt perfectly to the facade of the image. Since we only need the average of our parameters, using this method we can obtain sufficiently good results, although we obtain better results and better estimations using ortho-corrected images.

4.3 Parameters Estimation

The fact that this is a very specific task for this work means that there is no other approaches to compare. If we visually observe the image and check the results obtained with our approach we can observe how the results are optimal. In some cases, such as the average of the width of the streets or the perimeter of the typical blocks, we have manually counted the pixels in the image to see if they correspond with those calculated with for our algorithm. We realized that estimator was really precise when we use the ground truth of the neural network for the input, so we used the ground truth to compare the quality of the estimation when the input of the estimator was the output of the neural network.

For checking the quality of our estimators we created a small dataset with ground truth images and the output images from our network and we computed the difference of the mean value of the parameters, the variance of the error and the minimum and maximum error. Also, we wanted to know how different where the histograms between themselves, so we computed the difference between the entropy of the parameters distribution, the correlation between histograms and also the distances between histograms using the Chi-Square method and the intersection method.

The Chi-Square method compute the distances between histograms using the following function:

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)} \tag{4.1}$$

And the intersection method compute the distances between histograms using the following function:

$$d(H_1, H_2) = \sum_I min(H_1(I), H_2(I)) \tag{4.2}$$

In order to understand the results we have to know the typical values for each parameter. For the block size, the typical value is around 300m, for the street width is around 5m and the facade length around 30m. For the direction of the streets it is

difficult to give a typical value, since the values oscillate between 0 and 180 degrees depending on the image.

In the table 4.2 we can observe the results of the comparison of the values and the table 4.3 we can observe the results of the comparison between histograms.

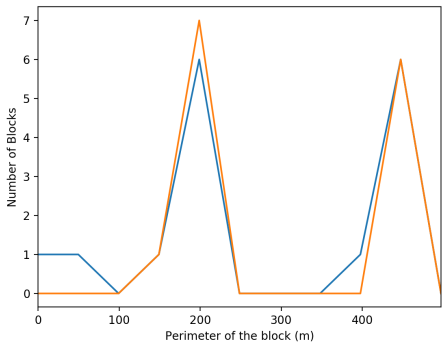
	Mean error	Max Error	Min Error	Variance
Block size	19.238 m	28.99 m	7.48 m	+/- 10.755 m
Street width	0.182 m	0.314 m	0.03 m	+/- 0.142 m
Street orientation	0.304°	0.69°	0.16°	+/- 0.265°
Facade length	5.215 m	10.32 m	0.1 m	+/- 5.11 m

Table 4.2: The error values for the estimation of the parameters.

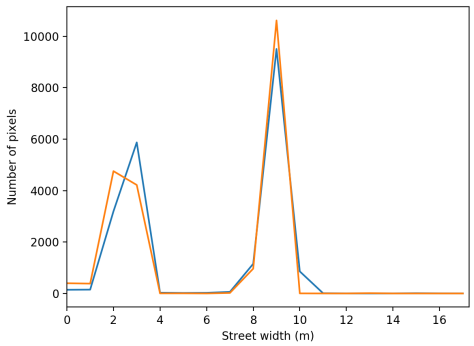
	Entropy error	Correlation	Chi-Square distance	Intersection distance
Block size	0.271	0.762	8.833	3.5
Street width	0.196	0.948	3167.078	17082.5
Street orientation	0.408	0.895	41665.558	58466
Facade length	0.09	0.862	55.926	138

Table 4.3: The results of the comparison of the histograms.

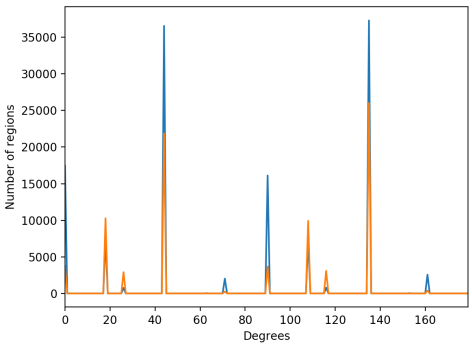
In the figure 4.3 we can see a comparison between the histograms of the parameters generated using the ground truth and the histograms of the parameters generated using the output of our network.



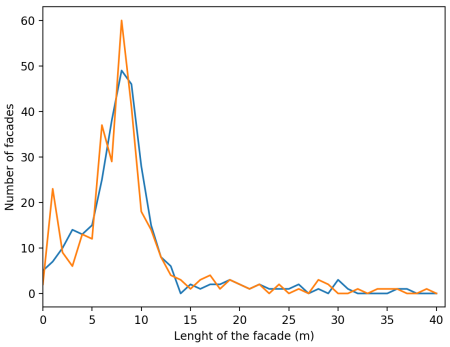
(a) The histograms for the block size.



(b) The histograms for the street width.



(c) The histograms for the angles of the streets.



(d) The histograms for the length of a building facade.

Figure 4.3: The different histograms distributions for the parameters. The orange line represents the ground truth estimation and the blue line represents the estimation using the output of our network.

If we look to the table 4.3 we can see that all the means and entropy error values are really low and all the correlation values are really high for all the parameters. If we take a look to the distance values we can see that in some cases are really high, in special for the street orientation but if we look tot the histograms in the figure 4.3 we can see that for the orientation histogram we have a lot of samples, so if we compare the error with the number of samples we still have a really low error. Otherwise, if we take a look to the figure 4.3 we can see that all the histograms are really close to the ground truth, that indicates that our estimator have a really good performance.

Notice that if we observe the histogram for the facades seems that we have good histogram approximation, but if we observe the resulting image, we can see that this where the most artifacts are, since in some cases the algorithm does not correctly detect the facades or in some others the algorithm detects them more than the fact that

the algorithm also detects the side walls of the building which should not be facades. In figure 4.4 we can observe the artifacts that we have just mentioned. Since what we need is the mean and the distribution of these lengths, the error that introduces does not imply a serious problem for our work, since the majority of the facades finds them well.

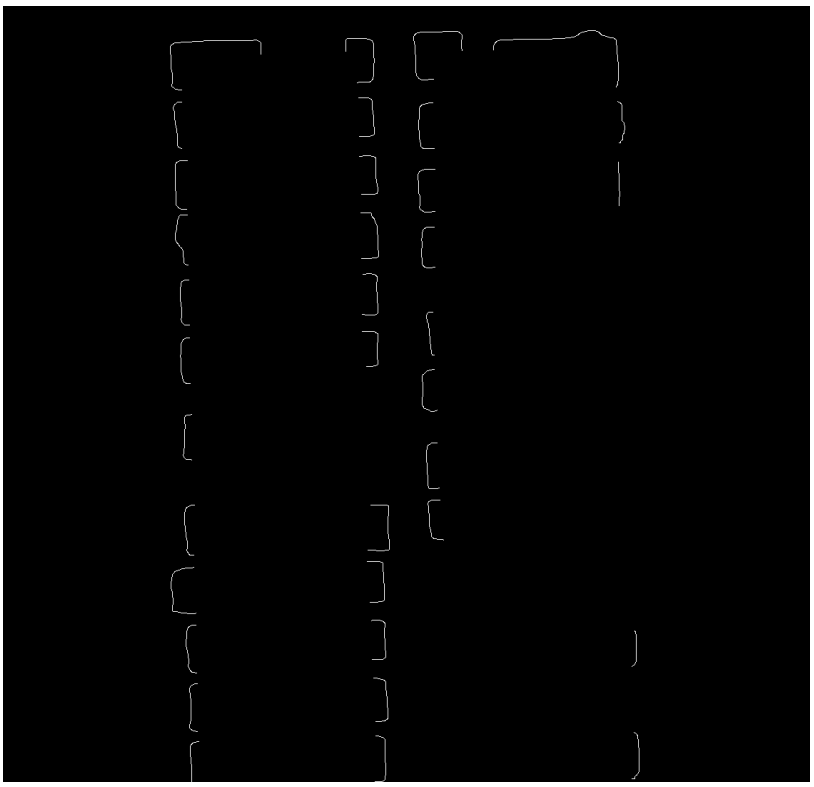


Figure 4.4: Example of the artifacts produced when calculating the facades. If we look at the row of houses on the right, we can see how some facades are not detected, and if we look at the facades facing the central street we can see how side walls of some of the buildings are shown.

Chapter 5

Conclusions and future development

As we told in the introduction, Télécom Paristech is working on a virtual 3D city generation system to have a realistic model to perform simulations of the propagation of electromagnetic waves of an antenna in a specific city, explained in [19] and [3].

For the process of the creation of the virtual 3D city generation is necessary to estimate some parameters in order to retrieve some information about the city. The parameters that we need to calculate are:

- **Mean perimeter of the typical block**: A typical block means the smallest area that is surrounded by streets. The typical blocks are the space for buildings within the street pattern of a city, and form the basic unit of a city's urban fabric
- **Anisotropy ratio**: According to [15], for two directions, the anisotropy ratio is the area of the parallelogram which is spanned by these two unit vectors. In other words, we have to estimate the distribution of the streets in the image.
- **Mean street width**: The average of the distances between one side and the other of the street.
- **Mean building height**: The average of the distances between the floor and the top of the building.
- **Mean length of a building facade**: The average of the length of the walls that are facing to the street of a building.

Traditionally, the estimation of these parameters had to be performed by hand using a human operator or randomly generated. With our system we have managed to simplify the process achieving a good estimation of this parameters, although the

height of the buildings we continue without being able to calculate it with this method because we don't have any deep information in our images. A possible solution would be to take advantage of the angles of the aerial raw images with respect to the ground to estimate the height of the buildings or to create a database where ground truth is the height of the buildings and train the neuronal network using ortho-corrected images in order to able to estimate the height of the buildings.

For the estimation of the parameters, the results are optimal and we obtain the precision necessary to carry out our task. The time it takes to calculate each parameter of the image is very low (approximately 10 seconds for each parameter, except for the detection of facades that can take up to 2 minutes), which implies that we can create a huge database in order to calculate the parameters of all the images and make the average and the distribution instead of using one image for compute the parameters.

With regard to the estimation of parameters using aerial images without horto-correction, although this methodology detects and segments the images, we believe that it still needs an improvement in performance in order to be able to have more precision. A possible solution would be to create manually labeled database with aerial raw images and to segment in the same way that we have done our database and train the neural network with the two databases so that the network has more examples of how the cities are.

Finally, we can ensure that our neuronal network presents a considerable improvement in performance compared to other neuronal networks proposed in [9] and [12], although adding more filters also increases the computational cost and the time of convergence. However, we believe that this computational cost is worth if we consider the improvement we get. Also, the fact that we have pretrained *VGG19* weights considerably reduces the computational cost. One important thing that we realized is that training with a huge dataset and after fine tuning with a small dataset really helps to segment the aerial raw images, but for the ortho-corrected images we realized that is not necessary, the network achieves good results only with the huge dataset.

Bibliography

[1] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986.

[2] Abhishek Chaurasia and Eugenio Culurciello. Linknet: Exploiting encoder representations for efficient semantic segmentation. *CoRR*, abs/1707.03718, 2017.

[3] Thomas Courtat, Laurent Decreusefond, and Philippe Martins. Stochastic simulation of urban environments. December 2014. working paper or preprint.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. 1:886–893 vol. 1, June 2005.

[5] Per-Erik Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14(3):227 – 248, 1980.

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[9] Pascal Kaiser, Jan Dirk Wegner, Aurélien Lucchi, Martin Jaggi, Thomas Hofmann, and Konrad Schindler. Learning aerial image segmentation from online maps. *CoRR*, abs/1707.06879, 2017.

[10] Ronald Kemker, Carl Salvaggio, and Christopher Kanan. Algorithms for semantic segmentation of multispectral remote sensing imagery using deep learning. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2018.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[12] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[13] Martin Längkvist, Andrey Kiselev, Marjan Alirezaie, and Amy Loutfi. Classification and segmentation of satellite orthoimagery using convolutional neural networks. *Remote Sensing*, 8(4), 2016.

[14] Dimitris Marmanis, J D. Wegner, Silvano Galliani, Konrad Schindler, Mihai Datcu, and Uwe Stilla. Semantic segmentation of aerial images with an ensemble of cnns. III-3:473–480, 06 2016.

[15] Werner Nagel and Viola Weiss. Mean values for homogeneous stit tessellations in 3d. *Image Analysis and Stereology*, 27(1):29–37, 2011.

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[17] Shunta Saito, Takayoshi Yamashita, and Yoshimitsu Aoki. Multiple object extraction from aerial imagery with convolutional neural networks. 2016:1–9, 02 2016.

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[19] Xiaoxing Yu, T. Courtat, P. Martins, L. Decreusefond, and J. M. Kelif. Crack stit tessellations for city modeling and impact of terrain topology on wireless propagation. pages 22–29, May 2014.

[20] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[21] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CoRR*, abs/1612.01105, 2016.

Appendix A

Other Neural Networks Implementations

In this section we can see the other Neural Network architectures that we used to compare our Neural Network. The only change that we made in both of the architectures respect from the original versions is adding the 3 more layers of the *VGG19*. We trained the *CNN* until the *CrossEntropy* and the F1-Score stop improving with the same learning rate, the same batch size and the same optimizer from our network.

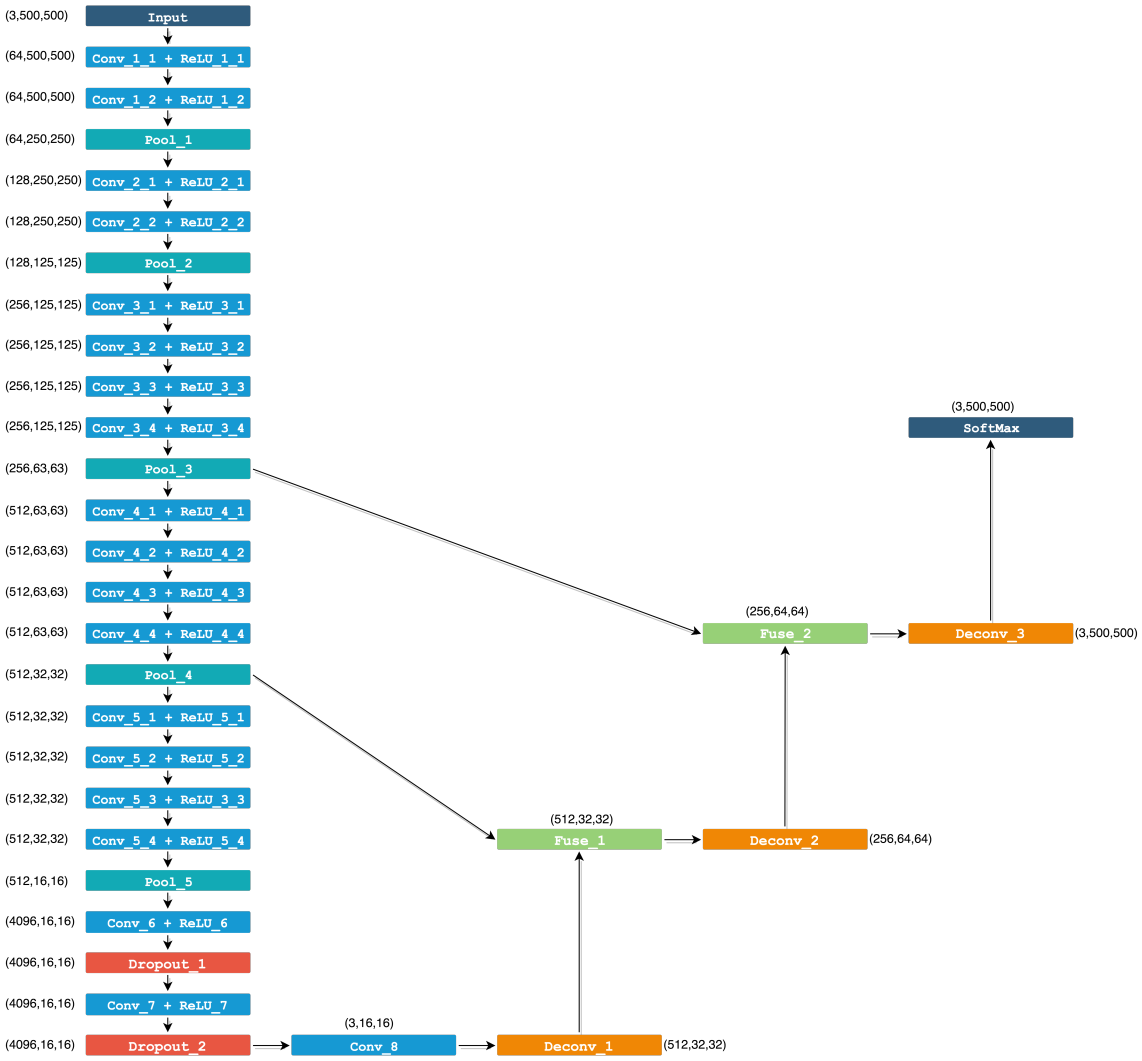


Figure A.1: The FCN 8s Neural Network, described in [12]

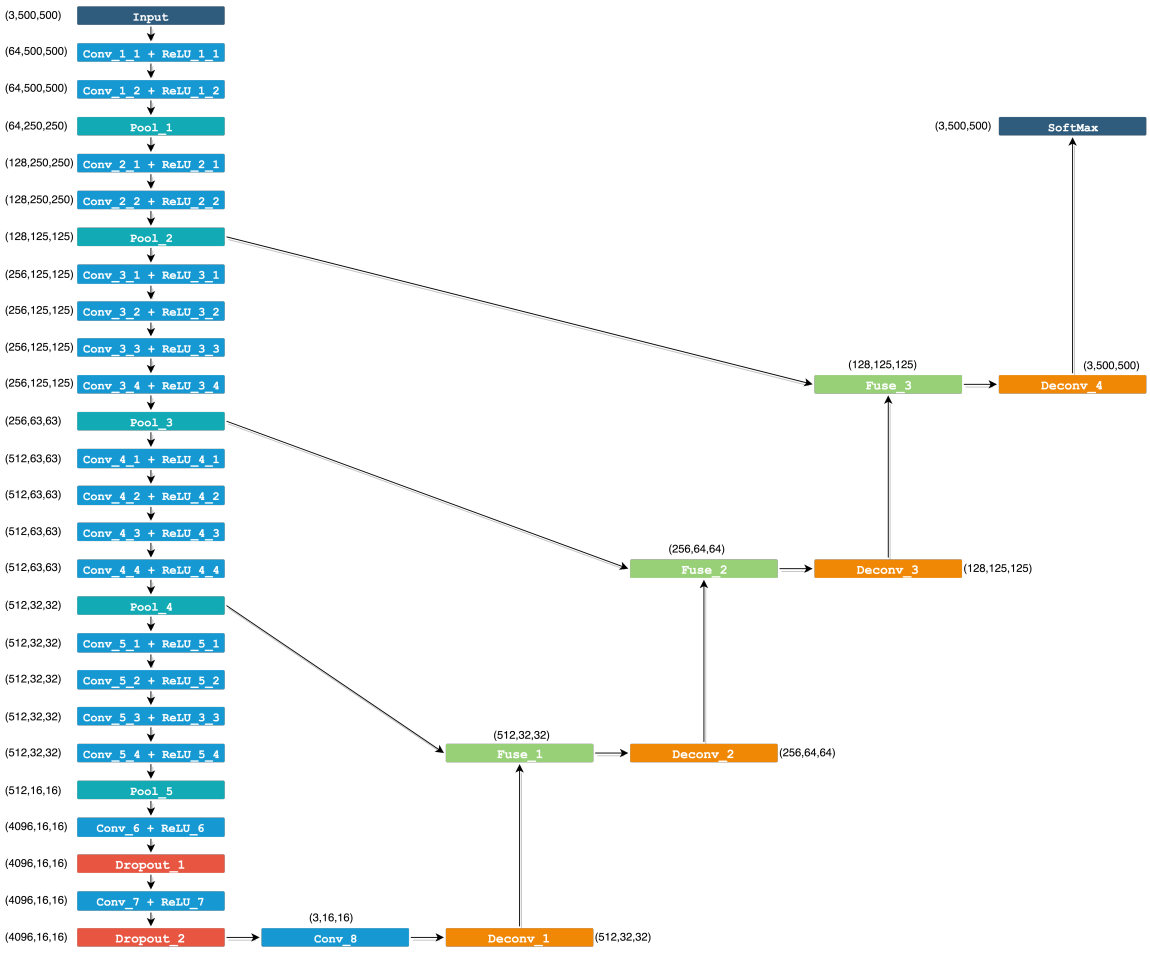


Figure A.2: The Neural Network architecture described in [9]