UNIVERSITAT DE BARCELONA

FUNDAMENTALS OF DATA SCIENCE MASTER'S THESIS

Using Deep learning and Open Street Maps to find features in aerial images

Author:
Marc BELTRÁN
Albert COMPANYS

Supervisor:
Dr. Santi SEGUÍ
Dr. Jordi VITRIÀ

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamentals of Data Science

in the

Facultat de Matemàtiques i Informàtica

July 3, 2018

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc in Fundamentals of Data Science

Using Deep learning and Open Street Maps to find features in aerial images

by Marc BELTRÁN
Albert COMPANYS

A great amount of the interesting information captured by aerial imagery is still not being used given how labour intensive the processing and annotation of these images is. Despite this, improvements in technology and advancements in the computer vision field have made available tools and techniques that can help make this process semi-automatized. In this project we focus on the use case of extracting roads from aerial imagery. For this purpose, we will study and compare models based on image segmentation using deep learning and *RoadTracer*, a revolutionary model proposed recently.

Acknowledgements

To our families and friends for putting up with us during this last stressful months. Many thanks!

To our project advisors, Dr. Santi Seguí and Dr. Jordi Vitrià, thank you for always encouraging us.

Contents

Chapter 1

Introduction

1.1 Context

A great amount of the interesting information captured by aerial imagery is still not used, even though it could help to enrich maps and improve navigation. For this information to be made available, objects such as buildings or roads need to be recognized in images. This is laborious to do manually, but non-trivial to perform computationally. Despite this, improvements in technology and new developments in the image processing and computer vision scene have provided tools to help with this computational problem that was previously very hard to solve.

FIGURE 1.1: An example of aerial imagery.

Aerial photography refers to taking photographs from an aircraft or another type of flying object such as drones. There are several types of images that can be taken using these technologies but for the purpose of mapping *orthophotos* are used. Orthophotos are geometrically corrected vertical photos that resemble the perspective of a map. Another way to understand orthophotos is to interpret them as photographs taken from an infinite distance looking straight down to the planet, having the perspective removed and some geometric transformations done to correct for variations in terrain. Figure 1.2 represents the difference between the perspective in an orhtophoto and a regular vertical photograph.

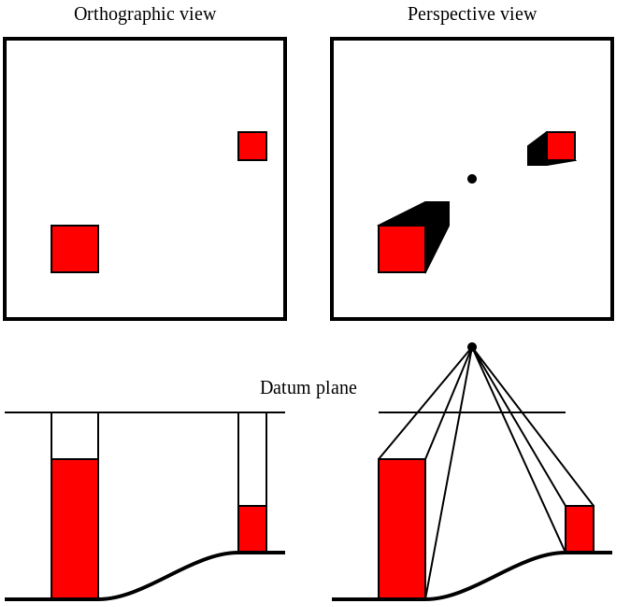


FIGURE 1.2: Orthoperspective compared to normal perspective.

1.2 Objectives

The objective of this project is to develop an automated method for detecting objects of a chosen class (i.e. pedestrian crosswalks, isolated buildings, roundabouts, etc.) on orthophotos, a method which can be adapted for various classes of objects.

The ideal model would consist of a model which, given an orthophoto and the types of objects it wants to detect, provides an output which shows for each part of the image which object it represents. This is called semantic segmentation and will be further explained in chapter 2.

In practical terms, the objective is to study the state of the art of segmentation of aerial imagery and other technologies that can be used for this purpose. To test and further understand the solutions, we will implement and analyze results of different techniques that we can use to solve this problem. As a use case, the objective is to perform tests of segmentation on road detection from aerial images. We will be limiting the scope of the models to this specific case for simplicity, but we will acknowledge how it could be expanded for a more general application.

Furthermore, we want to understand and apply the *RoadTracer* (chapter 5) approach recently published and be able to perform a simple comparison and evaluation of its performance with the classical and the methods based on convolutional neural networks (chapter 3).

1.3 Structure of the document

The following document is separated into two main blocks, one which deals with the classical segmentation problem and one which analyzes and applies the *RoadTracer* approach for road extraction in aerial imagery.

Finally, we include a discussion on the programming languages, frameworks and tools used to develop the project, and the conclusions and proposed future work to end the document.

1.3.1 Semantic Segmentation

The theoretical aspects of this block are discussed in chapters 2, 3 and 4. The theory follows a logical order: In the first section we will discuss the basics and general concepts of semantic segmentation on images. Later, we will go over the concepts of neural networks and how they can be applied for this purpose. Finally in this block, we look at the general pipeline of the process of performing segmentation using the mentioned techniques.

The practical and implementation details can be found in chapter 6 together with the results obtained using these methods.

1.3.2 Road Tracer

The theory for the understanding of the *RoadTracer* method can be found in chapter 5. This chapter is self-contained and does not follow the previous chapters.

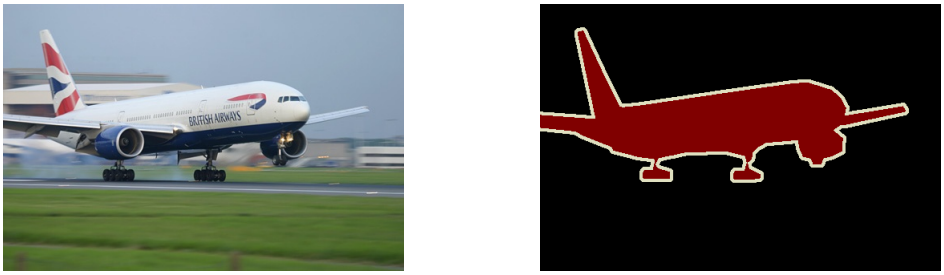
The details of the implementation of the pipeline and the results obtained can be found in chapter 7.

Chapter 2

Semantic Segmentation

2.1 Introduction

Image segmentation is the process of partitioning a digital image into meaningful object regions. In the field of computer vision, this is one of the most challenging problems related to image processing and analysis. Segmentation is the first step for other high level processing tasks related to images such as image interpretation, analysis, diagnosis, etc. The goal of segmentation is to change the representation of the image into something more meaningful that is easier to analyze. More precisely, we can define image segmentation as the process of assigning to each pixel of an image a label such that pixels with the same label in the image and other images share the same characteristics. The result of this process is a set of segments that together cover the whole image.



(A) Original image (B) Segmented image

FIGURE 2.1: An example of image segmentation [1].

2.2 Semantic Segmentation

Semantic segmentation is the task of clustering parts of an image that belong to the same class. In contrast, image segmentation only clusters pixels together based on general characteristics, which makes this task not defined and would in theory accept multiple valid segmentations. Semantic segmentation has many utilities in different fields such as detecting road signs [33], land use and land cover classification [8], or the medical field with use cases like detecting tumors [27] or colon crypts segmentation [26]. It is important to note that semantic segmentation differs from object detection in that it does not distinguish between several instances of the same type of object. This is caused by both neighboring pixels belonging to the same class but to difference instances of an object and the opposite case, where there could be a same object with some part not connected to the rest of it.

2.2.1 Algorithm criteria

Semantic segmentation is a classification task in which each pixel of the input image is assigned a class with a given probability. These algorithms can be separated into groups depending on several variables.

Classes

The possible classes this may include are decided beforehand and the models are trained with these classes in mind. Supervised segmentation algorithms can work on binary classes or with multiple classes. Furthermore, some algorithms can be developed to recognize when they don not know a class [31] and others can be unsupervised and don not distinguish classes at all.

Relationship between the pixels

We can distinguish between single class affiliation and multiple class affiliation. In the former only a single class can be assigned to a pixel, but in multiple class several classes can be assigned to the same pixel. This can be helpful with tasks with overlapping or transparent objects.

Data

The structure and type of the input images can change how the algorithms work and which ones are more effective. We can classify the types of input images following the following criteria:

- Grayscale or colored: Colored images are commonly used with photographic images when they are available. This provides color information that can be used by dissecting the image into color layers. Despite this being the preferable option, some use cases rely on grayscale images due to the means of obtaining them such as magnetic resonances in the medical field.

- Depth data: When we think of images we imagine a static image with no information on depth. Due to new cameras and sensors developed images can be extracted with layers of depth attached to them. We call this images RGB-D and they have been used in different segmentation applications incorporating the features derived from the depth data [3]. Despite this, the most common applications still work on images with no depth due to either the depth parameters not being necessary for the application or the increased difficulty of obtaining them.

- Number of images: Although single image segmentation is the most common approach, attempts have been made with multiple images. For instance, using stereo images has been used to try to infer the depth parameters by simulating what a human would see with two eyes [36]. Moreover a technique knows as co-segmentation [9], [24], has also been used to find a consistent segmentation along multiple images.

- Number of dimensions: We have been referring to a single unit of an image by the name pixel. However, this only applies to 2D images. Segmentation can be also applied to 3D images to create 3D segments by assigning a category

to each single unit of a 3D image which we call a voxel. Examples of 3D segmentation take us again to the medical field where for example segmentation techniques jave been used with volumetric x-ray images [32].

Active or passive

The classifying entity can be either an active protagonist or a passive one. By active we understand that the entity can move around to get other perspectives or interact with them [14]. Static models are those in which the received image can not be influenced by the requirements.

2.2.2 Traditional approaches

Before convolutional neural networks [23] were introduced and applied to segmentation problems, other traditional approaches were being used that relied on other techniques and made heavy use of domain knowledge. In this section, we will cover a brief overview of different techniques and approaches that were the backbones of the traditional methods. Given that we will not be applying any of these during the development of the project, we will not go into detail into each of the proposed models. Despite this, it is interesting to learn how the segmentation problems were approached before neural networks took over.

Preprocessing and feature selection

In contrast to approaches based on neural networks, image segmentation algorithms that rely on traditional approaches rely heavily on an adequate preprocessing of the input images and precise feature selection influenced heavily by domain knowledge. Models based on neural network techniques are, in general, better at finding the important features from a less preprocessed input. Below we detail some of the most commonly used processing techniques on the input data. These ideas can also be applied to models based on neural networks.

- Color: Images have to be transformed into some sort of numerical codification so that the algorithms can interpret them. Every pixel of the image is transformed into a set of numerical values, which vary in length and meaning depending on the codification used. Apart from the commonly used RGB for color images and grayscale codification for grayscale images, other color spaces can be used, such as YcBcr or HSI. Depending on the application one color space could yield better results than another, and it has been proven that no color space is superior to the others [17].

- Histogram of oriented gradients: A histogram of oriented gradients or HOG is defined as a discrete function that maps the position (x, y) in the image to a color. The partial derivatives of x and y are the gradients, and so the original image is now transformed into features that represent the gradient. A histogram of these maps is used to calculate directions for each patch. This was first introduced in [15].

- Dimensionality Reduction: Images are high dimensional data with a very high correlation between neighboring pixels. This means that if we use a feature for each pixel we could encounter problems with performance. One approach

relies simply on reducing the size of the image by compressing it. Other approaches take advantage of the correlation between pixels and so dimensionality reduction techniques are used to map the images to a lower dimension space while still retaining the most important information in the features. A commonly used technique for this is Principal Component Analysis (PCA) [35].

Supervised methods

Supervised segmentation methods rely on the input images which have each pixel assigned to one of the classes we want to predict. This is commonly known as a mask, where each segment would be represented by a codification.

- Random Decision Forests: First proposed in 1995 [20], this classifier consists of multiple classifiers trained and a combination of the resulting hypothesis is used. This is referred to as ensemble learning. The main advantage of this model is that the scale of measure of the input features can be arbitrary.

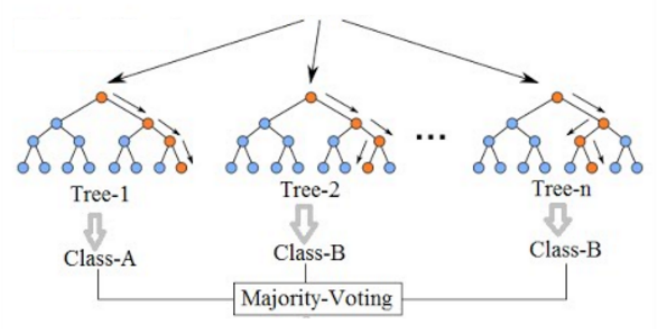


FIGURE 2.2: Random forest structure

- Support Vector Machines: SVMs [7] are binary classifiers that are based on transforming the input data into a vector space in which the classes are linearly separable by a hyperplane. Despite SVMs only separating between two classes, they can be expanded to be multi-class classifiers using the one-to-all technique in which a model is trained for each class against all the others.

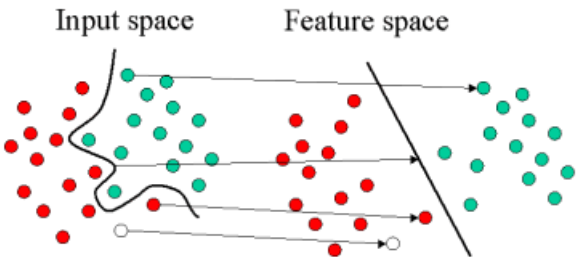


FIGURE 2.3: Visual representation of a SVM

- Markov Random Fields: MRFs [2] are unidirected probabilistical graphical distributions which assign to each feature and to each pixel a random variable.

Unsupervised methods

Unsupervised methods can be used for to help supervised methods as another source of information or a way to infer possible features to be used. In contrast to supervised methods, unsupervised models can be used to detect boundaries and delimit regions while not knowing the assigned classes. This set of models can be grouped into the following categories.

- Clustering algorithms: Clustering algorithms are based on finding the optimum class separation such that the distance between each pixel or feature to the center of its class is minimized. An example of this is k-means which has been used to aid in segmentation of medical imagery [10].
- Graph based segmentation: The idea is to model the images by representing pixels as nodes and encoding the dissimilarity of the pixels with their neighbours in the vertexes. The objective is to cut the graph in such a way that it keeps the connection between the pixels in the same segment and cuts the rest.
- Active Contour Models: ACMs are based on classical computer vision techniques that detect edges and segment the images along the found borders to create the segments. They have been used, for instance, to refine a supervised segmentation implementation in brain MR images [4].

Chapter 3

Neural Networks for Semantic Segmentation

3.1 Introduction to Convolutional Neural Networks

Convolutional Neural Networks (or CNNs) were first introduced in 1998 [22] as a new architecture based on existing neural network models. Despite this, it was not until 2012's ImageNet Computer Vision competition that CNNs were brought to the public used after being used to achieve image classification tasks with a small error when compared to the models available at the time. Since then, CNNs have been refined and upgraded and can achieve accuracy scores that surpass performances by humans in some cases [18].

Convolutional neural networks share the same base structure than neural networks. They consist of neurons that have weights and biases that can be learned depending on the given task. Neurons are interconnected and structured in layers, having each neuron receive an input, perform a calculation, and output a result to be used by other neurons. The difference with regular neural networks is that CNNs use the fact that the input is going to be an image to their advantage and so can use the properties of images such as correlation between neighboring pixels to optimize the network. Instead of having each neuron fully connected to neurons in the previous layer the architecture can be constrained in such a way that they are only connected to a small region of the layer.

Layers found in convolutional neural networks are based on a convolutional layer, a pooling layer and a fully-connected layer.

3.1.1 Convolutional Layer

A filter is a spatially small window that we can slide (or convolve) along the width and height of the input image and perform matrix based computations between the filter and a region of the image. Convolutional layers operate by calculating the optimum weights and biases assigned to these filters. During the training process, the network learns which filters to represent and which ones get activated given the established goal.

3.1.2 Pooling Layer

Downsampling is used to reduce the redundancy in the input. Pooling layers are used to reduce the size of the representation between layers to drive down computational times and prevent overfitting. These layers are introduced between convolutional layers and transform nxn windows of pixels into a single value to reduce the size of the input to the next layer. This new value is commonly calculated by

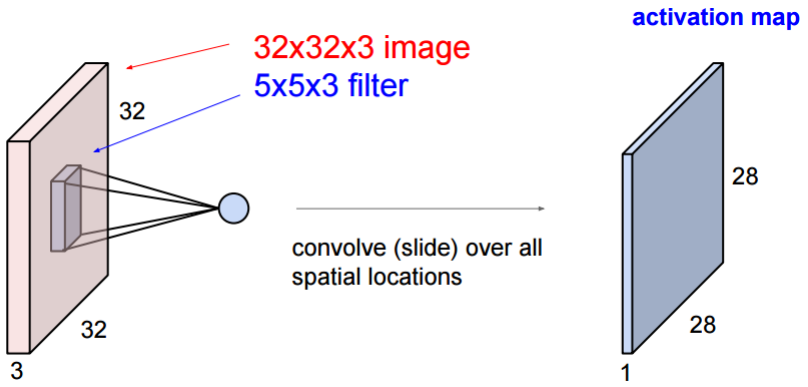


FIGURE 3.1: Visual representation of a convolution

taking the maximum of the values in the window (maxiumum pooling) or by taking the average (average pooling).

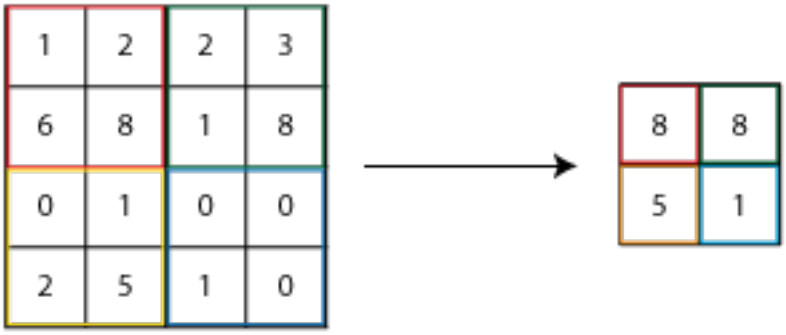


FIGURE 3.2: An example of pooling with a window of size 2.

3.1.3 Fully Connected Layer

In the fully connected layer, the input to this layer is flattened into a one-dimensional vector and used as an input to a neural network to perform a prediction or classification. This layer takes the output of a series of previous convolutional layers and acts as the final step of the process. Note that a fully connected layer can be expressed as a convolutional layer with a convolution of size $1x1$.

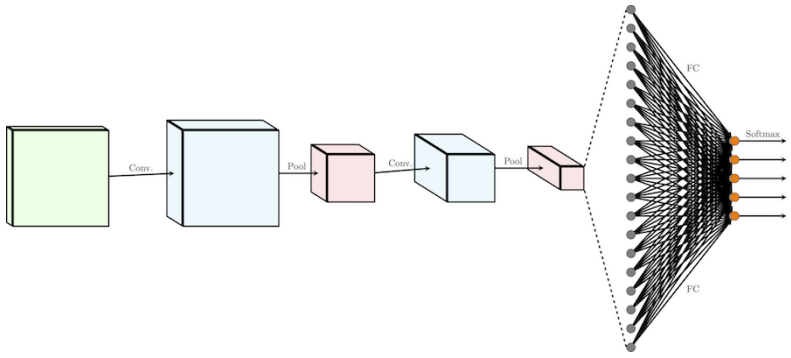


FIGURE 3.3: A sample structure of a Convolutional Neural Network

3.2 Semantic Segmentation

3.2.1 Fully Convolutional Networks

The first major milestone for semantic segmentation as a part of the computer vision and machine learning community was the PASCAL Visual Object Classes Challenge in 2007 [VOC7]. Despite this, the major breakthroughs came later after convolutional neural networks were first successfully used to process and classify images. Fully convoltuional networks (or FCNs) were first used for semantic segmentation in 2014 [23] and since many different approaches and techniques have been used based in this architecture. These networks worked without fully connected layers which made it possible to generate segmentation maps from images of any size. Another difference with regular CNNs was the omision of pooling layers. This layers help aggregate the content of parts of the image but lose the spatial position which is needed for segmentation tasks. Figure 3.4 shows a visual representation of a sample FCN.

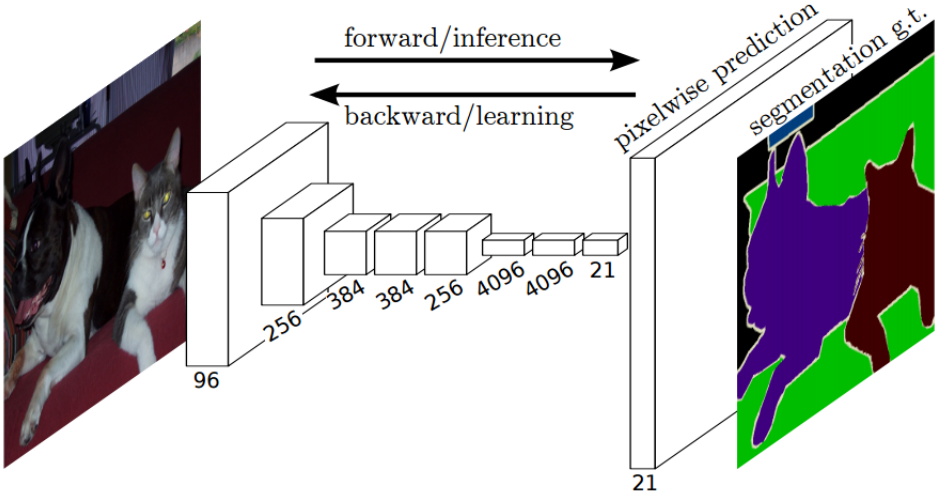


FIGURE 3.4: Fully Convolutional Network [23]

We will explore the main categories of architectures that arose to handle the issues presented by regular convolutional neural networks and the fully convolutional variant.

3.2.2 Encoder-Decoder Architectures

The general concept of these types of architectures is to take an existing network purposed for classification such as VGG16 [34] removing the fully connected layers at the end of the process and turning the final features into a segmentation map. We refer to the part of the network extracted from the classification network as the *encoding*, and the second part which takes the learned representation and transforms it into a map as the *decoding*. The *encoder* creates low dimensional representations of the input features of the image and the problem lies on how to decode the features. Different approaches to the *decoder* is what separated the main networks that use this architecture.

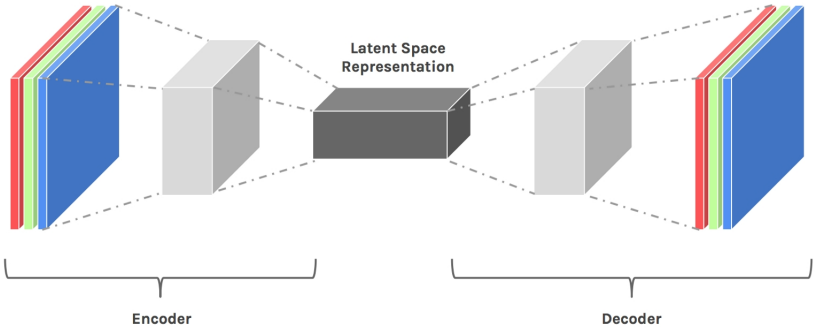


FIGURE 3.5: Encoder-Decoder architecture

3.2.3 Dilated Convolutions

Dilated convolutions [38] were introduced as an alternative way to perform the pooling layers of the encoding stage to avoid reducing the resolution of the input features. These convolutions are able to aggregate multiscale contextual information but without the drawback of losing resolution. The purpose of these convolutions is to be able to integrate knowledge of the wider context surrounding the pixel without increasing the cost. Figure 3.6 shows a visual representation on how a dilated convolution can be used to keep knowledge of a wider context without having to use a bigger convolutional filter.

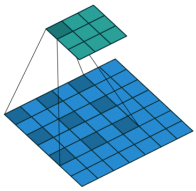


FIGURE 3.6: Visual representation of a dilated convolution

3.2.4 Conditional Random Fields

To increase the limited spatial accuracy that semantic segmentation models have due to the spatial information lost in parts of the network, a technique called conditional random fields [25] (or CRFs) was introduced. Its purpose is to act as a post-processing stage to capture fine-grained details and to model the segmented regions more accurately. CRFs combine the relations between pixels with models that perform pixel-by-pixel prediction scores to fine tune the output of the segmented regions.

(A) Original image (B) Segmented image (C) Segmented image after using CRF

FIGURE 3.7: An example of using a fully connected CRF

3.3 Models

Many different network models have been designed specifically to work on semantic segmentation since the first introduction of fully convolutional networks. In this section, we will take a look at some of the most important and better performing architectures that have been published.

3.3.1 SegNet

The approach of SegNet [5] to the decoding step relies on using the pooling indices computed in each pooling layer of the encoder to perform a non-linear upsampling process. This technique is know as *unpooling* and helps keep high frequency details intact in the segmentation instead of them being lost due to the encoding process. Despite this, the method still loses information on the neighboring pixels of the stored indices. Given that this techique doesn't rely on fully connected layers to perform the segmentation it requires less parameters to train. Figure 3.8 shows a visual representation of the structure of the network.

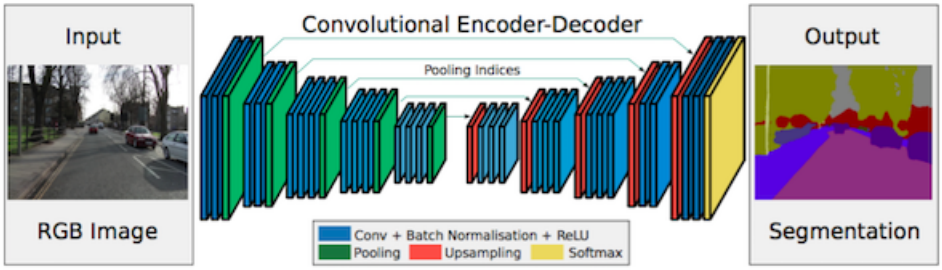


FIGURE 3.8: SegNet architecture [5]

3.3.2 U-Net

The U-Net [30] models the encoder-decoder architecture in a way parallel to that in ladder networks [29]. This is done in such a way that feature maps from the encoder are concatenated to upsampled feature maps from the decoder in every stage, creating a structure resembling a ladder. Every concatenation allows the network to *remember* back relevant features that were lost back in the encoding stage. The strong point of the U-Net is its ability to learn from a small sample of input images, making it a good candidate to be used with medical imagery.

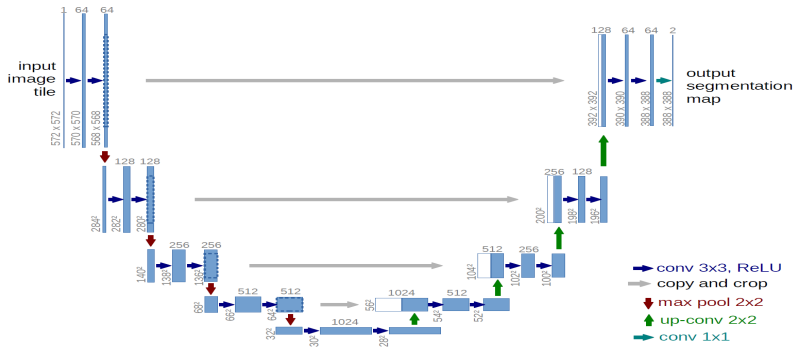


FIGURE 3.9: U-Net architecture

3.3.3 DeepLab

DeepLab v1 [13] and v2 [11] make use of diluted convolutions to increase the field of view of the convolutions without requiring an increase in the number of parameters. CRFs were also used to improve the final result of the segmented images. This models were the first to propose atrous (diluted) spatial pyramid pooling (or ASPP) to provide multiscale processing.

DeepLab v3 [12] was later published to improve on the previous versions by improving the ASPP module and to incorporate batch normalization [21] to help encode multi scale content.

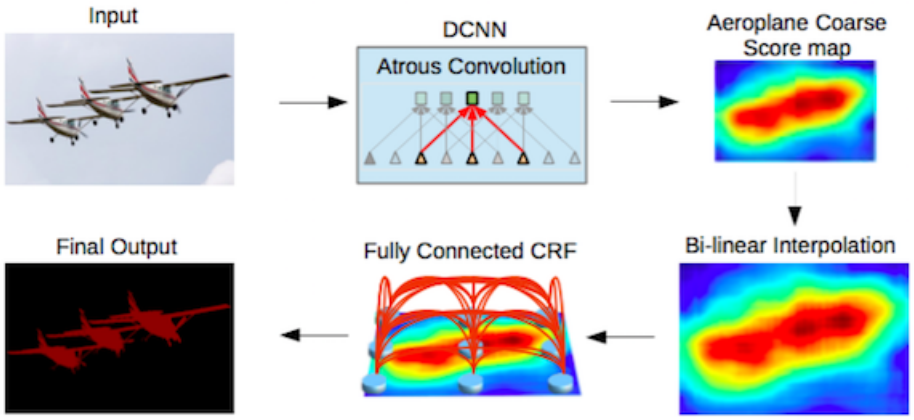


FIGURE 3.10: DeepLab v2 pipeline

3.3.4 PSPNet

The PSPNet [39] model proposes the use of dilated convolutions on the classification network ResNet [19] in the shape of pyramid pooling layers. A concatenation is created between the outsampled ouput of the pooling features with different kernel sizes. This is a way to provide information of the greater scene and give clues on the distribution of the segmentation classes. An auxiliary loss, or intermediate supervision, is used to help optimize the training process.

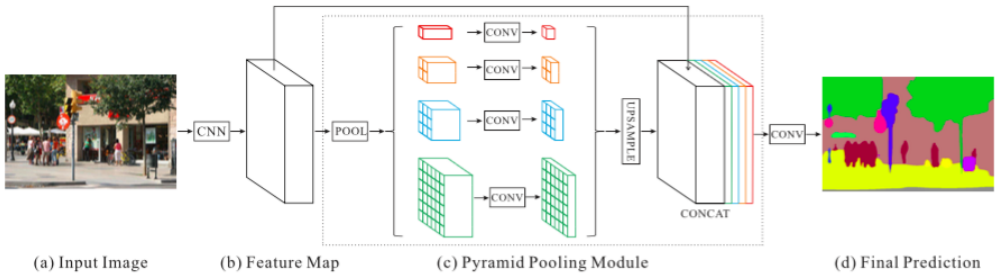


FIGURE 3.11: PSPNet architecture

3.4 Transfer learning

3.4.1 Definition

Transfer learning is an approach in machine learning model that consists of using pre-trained models for usually general tasks as starting points of another model created for a more specific task. Starting at a pre-trained point instead of a random state in the training process of a model allows faster progress and optimization when modeling the second task. Given the high computational costs of training deep neural networks this allows the main network to be trained with less resources, by using pre-trained models in sometimes very challenging classification problems which would require an amount of resources not available to everyone. It also improves generalization and can prevent overfitting to the main task, as long as the features learned from the pre-trained model are general and applicable to the task.

In the concept of deep learning this is called *inductive transfer* which works by narrowing the scope of possible models by using a model purposed for a general task as the starting point, as long as they are both related. Transfer learning in deep neural networks has been proven to considerably improve results and efficiency [37]. In figure 3.12 we can see a visual representation of this concept.

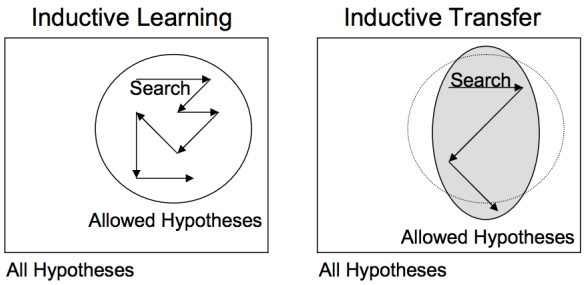


FIGURE 3.12: Visual representation of the narrowing of the possible models using inductive transfer

The main benefits to this technique include a higher performance at the start of the training process, a steeper slope in performance in relation to the training scale and a better final result when the performance curve becomes asymptotic. This can be visually seen in the graph in figure 3.13 [37].

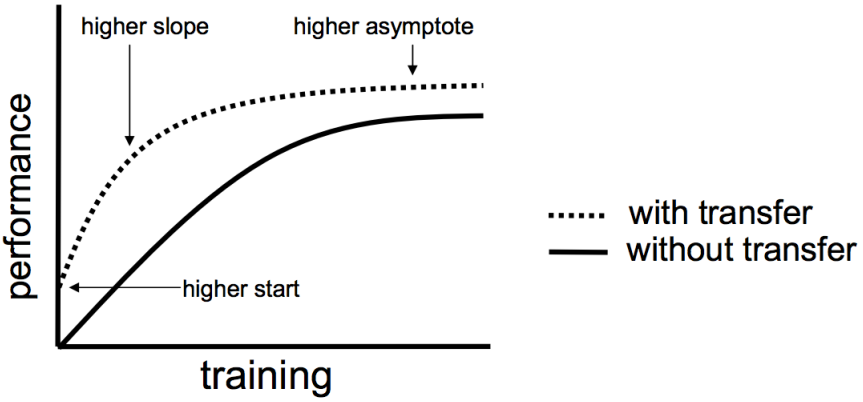


FIGURE 3.13: How transfer learning affects the performance

Transfer learning can be done using pre-trained models for very generic tasks computed by external sources or by developing a trained model with a set of data heavily related to the main task.

3.4.2 Using a pre-trained model

Many research institutions develop and release models trained on large and challenging datasets which would be impossible to do with a small number of resources. These models form a pool of candidate models and can be chosen depending on the main task. The models can be simply used as a starting point of the new model by reusing the whole model or parts of it. In some cases, the model can be modified and adapted for better results in the main task.

When dealing with image data, it is very common to use transfer learning with pre-trained models given the high computational cost of working with input images due to their high dimensions. When developing models based on images there are many pre-trained models developed for large image classification tasks and competitions that are released for reuse. These models can be directly downloaded and incorporated in the pipeline as a starting point. Examples of these types of models include Oxford's VGG Model [34] or Microsoft's ResNet Model [19].

3.4.3 Developing a general model

In some cases it can be possible to develop a generic model with data heavily related to the main task but with which would still provide a general baseline to which to start from. This can be done when the amount of data is very high and there is a strict relationship with the data to be used in the main model. Ideally the developed model would give a better starting point to the main model due to the focus and the input data used being related to the main task. Despite this, developed models could still give worse starting points than the pre-trained models discussed before due to the fewer resources that are available to train them compared with the models released by large institutions. The specifics of the task and the data available plays a very important role in deciding which approach to use.

Chapter 4

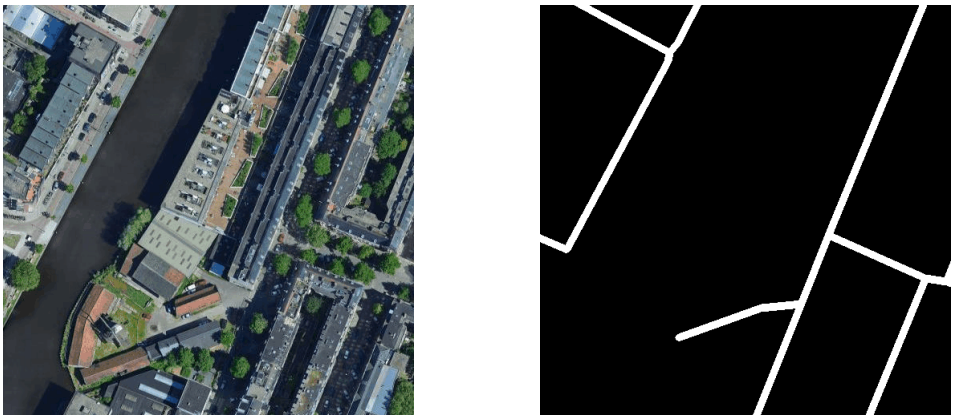
Using Semantic Segmentation on Aerial Imagery

4.1 Objective

As described in the introduction of the document, the objective of the project was to help to enrich maps and improve navigation by recognizing objects such as buildings or roads from orthophotos using automated methods which could be adapted to work on different types of classes.

One way to approach this problem is to use the aforementioned semantic segmentation techniques to create a semantic map of the image with the different classes we want to recognize. This would allow us to, given an input image, output an map which would assign to each pixel of the input orthophoto a category based on the predefined classes.

In figure 4.1 we can see an example on how the output would be of an model which segmented roads from the rest of the environment given the orthophoto.



(A) Original image (B) Segmented image

FIGURE 4.1: An example of segmentation on an orthophoto

In this example and throughout the project we will focus on the case of road detection as a case study of the different techniques we can try to achieve recognition and detection of features. This same techniques could be used to perform the same type of study for other features such as roundabouts or cross-walks as mentioned in the introduction to the document.

4.2 The Segmentation Pipeline

The process of generating the output map from the orthophoto does not only consist of training a model. There are certain steps and decisions that have to be made before even the model enters the equation. In this section we will describe how the full pipeline works in general terms for a segmentation pipeline and how we can adapt it for our task.

4.2.1 Data gathering and preparation

The first step to any machine learning pipeline consists of defining the format of the input data the model will be fed and how to obtain it. In a semantic segmentation problem we need both the images we want to segment and some sort of classification of what classes are present in the image and where. This is commonly known as a mask, which mimics the size of the input image and encodes what class each pixel belongs to using a different color for each class. We need this mask when defining a supervised model as we rely on the ground truth to perform the training stage of the modelling. We have seen examples of these masks in previous figures such as in figure 3.7. In the case of an unsupervised segmentation, we would only work with the input image.

In the specific case of segmentation of aerial images, we need to both find the orthophotos of different places of the planet and find a way to acquire the theoretical ground truth of where each feature we want to detect is. This can be done using several existing maps and mapping applications that we will specify in our implementation of the problem.

Once the images are collected and the masks acquired the next step is to decide how the data will be formatted as input. When dealing with images, there are several codifications they can be transformed to in such a way that they are possible to be fed to a network. These codifications are mathematical structures with numerical values, and some of them have been discussed in previous chapters.

In the case of orthophotos, technology nowadays allows us to access color images which we can transform to a color encoding such as RBG which creates a matrix of the size of the image with three layers of depth to represent the RGB spectrum.

4.2.2 Data augmentation

Image classification and segmentation rely on several data augmentation techniques to improve and refine models. By data augmentation we understand the process of creating variations of the existing images that will be fed as more input data. This process can be very useful to prevent overfitting when the number of training images is relatively small and reduce inconsistencies within the data. This creates a more general input to be used when training the network. Data augmentation techniques have been proven to be effective at boosting the performance of deep learning models [AUGMEN].

Different techniques can be used depending on the nature of the task, some of which we will explore in this section.

Traditional Transformations

The most basic augmentation techniques consist of creating affine transformations of the input images. Some of the most common examples are rotating the image,

mirroring it, zooming in and out or performing distortions. All of these techniques rely on simple mathematical operations on the image matrix.

For our task we would be most interested in images rotations to improve the detection and segmentation of roads going in different directions and using different hues in the image to help classification in areas with different climates.

Generative Adversarial Networks

Generative adversarial networks (or GANs) can be used to convert an image to a certain style using a technique known as style transfer. This converts the image to different styles to create new versions of the image and is heavily used in classification techniques. The effectiveness of this method for classification tasks has been shown to be significant [GAN].

When dealing with aerial images this technique is not as useful as others as in general orthophotos are very similar in style and there is not much to be gained by using the same image with different styles.

Learning the Augmentation

The most recent data augmentation technique consists in letting a model learn the augmentation we want to use. This can be done in classification tasks by creating an input to a model which consists of two images of the same class and receiving a layer the same size of the input images as an output. This layer can then be used as an input of a classification network and analyzing the loss of this model helps refine and create a new image which is a combination of the two inputs.

In general, this technique is only useful for classification tasks as in a segmentation task we would lose the mask of the inputs. Despite this and given how many segmentation models rely on pre-trained classification models, this is a very important technique that can be explored for specific tasks.

4.2.3 Model Selection

Once the input data to be used by the model is prepared, the next step in every pipeline consists of feeding the data into the models. The data that will be fed consists of the augmented data using the techniques described previously. Many models can be selected and refined depending on the task in hand. This section of the pipeline is based on testing different approaches to solve the problem. Different architectures can be tested and different parameters for the same architectures can also make a very big difference on the performance. To chose a model, the models are tested on a subset of data that has not been used in the training process to evaluate the performance. Different metrics can be used to evaluate this performance depending on the task. This evaluation task is done on regular raw data that has not been augmented as its purpose is not to improve the model, rather to test its performance.

Different performance metrics can be used depending on the task but for semantic segmentation problems the most simple one would be a pixel by pixel comparison of the predicted mask and the ground truth.

Chapter 5

The Roadtracer Approach To Road Detection

5.1 Segmentation Based Road Detection

Semantic segmentation techniques explained in previous chapters offer a very versatile method when it comes to aerial images interpretation. They can be trained for different purposes, whether it is detecting soil usage, pedestrian crosswalks, isolated buildings or roundabouts. As it has been explained earlier, semantic segmentation is about assigning a category to each pixel in an image. The focus of this project is set in road detection, a more concrete matter than just feature extraction of aerial images. Even though advanced CNN based methods such as DeepLab or PSPNet can obtain very good results for road detection, they are still using the approach of assigning every pixel a binary value based on the image alone and leaving all the decision power to the neural network. Therefore, this network is forced to learn properties that are intrinsic to roads, such as connectivity or continuity. In the following section we will introduce a completely different method created by a research team from the MIT that uses this properties in order to improve the performance of CNN based techniques for road detection.

5.2 Roadtracer: Iterative Graph Construction Method

In contrast to the techniques explained before, this approach consists of a search algorithm, guided by a decision function (implemented with a CNN). The search starts at a position known to be on the road and walks along setting vertices and edges. The decision function is invoked at each step and determines what to do: add a new edge to the network or backtrack to the last position with another open branch to explore. We will do a high level explanation of this approach coming next, but further details are perfectly explained in the original paper by Bastani et al [6].

5.2.1 Search Algorithm

It all starts with a region, defined as (v_0, B) where v_0 is the starting location (which we know is in a road) and B is a box containing the area we will work with. The algorithm keeps track of a graph G and a stack of vertices S, both initialized with one vertex v_0. S_{top}, the vertex at the top of the stack, is the current location of the algorithm. At each step, the decision function takes G, S_{top}, and a section of the aerial image centered at S_{top}. The possible outcomes of this function are either to walk a fixed distance forward from S_{top} along a direction, or to stop and return to the vertex before S_{top} in S. When advancing, the decision function selects the direction from a

set of angles in $[0, 2\pi)$ uniformly distributed. Then, a new vertex u is added to the graph and pushed into S (and the search is moved to u).

On the other hand, if the decision function decides to stop at a given step, S_{top} gets popped out of S. Stopping means there are no more paths to explore adjacent to S_{top} and, therefore, the vertex we stopped at will never be visited again.

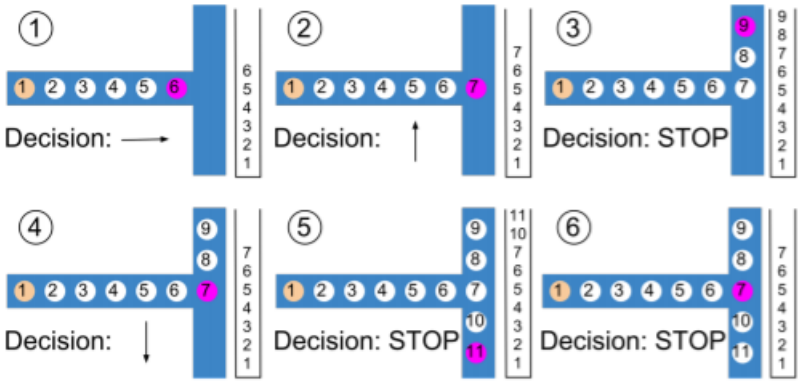


FIGURE 5.1: Exploring an intersection during the search process. Circles are nodes in the graph, with S_{top} painted in pink. Blue area represents the road, and the rightmost column represents the status of S at each step. [6]

Figure 5.1 displays an example of how the algorithm will proceed at an intersection. When the intersection is reached, we first follow the upper branch, and when the end of the branch is reached, the decision process decides to stop and the search algorithm returns to the last open path, which was the intersection. Then, the lower branch is selected. Once the end of this other branch is reached, the decision function will again select the stop action repeatedly until we get back to the starting position and S becomes empty. At this point, we will consider that the construction of the road network is complete.

Other problems may appear during the construction of the network. Since there are cycles in road networks, it is possible to arrive at a node that has already been explored. To deal with this, a node is only pushed onto the stack if it is not very close to an already explored node in G. This prevents the creation of small loops, for instance when a road forks in two at a thin angle. In 1 there is a pseudocode specification of the search algorithm.

5.2.2 CNN Decision Function

The decision function mentioned before, implemented via a CNN, is a key component of the RoadTracer algorithm. The input layer consists of a dxd window centered at S_{top}. The window is formed by four channels: the first three are the RGB values of the window, and the fourth channel is a render of the graph constructed at this point, G. This allows the CNN to understand which roads have been explored earlier in the search, and it also provides the CNN with context useful, for instance, to detect roads occluded by tall buildings or trees as shown in 5.2

The output layer is built with two components: an action component and an angle component that tells us which direction to walk in. The action component is a softmax layer with two outputs, one for each action, and the angle component is a sigmoid layer with as many neurons as possible angles. Then a threshold T is used

Algorithm 1 Iterative Graph Construction

Input: A starting location v_0 and the bounding box B
Initialize graph G and vertex stack S with v_0
loop while S not empty
    action, α := decision_func(G,S_{top}, Image)
    $u := S_{top} + (Dcos\alpha, Dsin\alpha)$
    if action = stop or u is outside of B **then**
        pop S_{top} from S
    else
        add vertex u to G
        add an edge (S_{top}, u) to G
        push u onto S
    end if
end loop

FIGURE 5.2: Sometimes shadows, trees, or tall buildings make it hard to determine wether there is a road or not, even for humans.

to decide between walking or not. If $O_{walk} \leq T$, then walk in the angle whose neuron had the maximum value. Otherwise, stop.

This gives the CNN the capability of producing a road network graph straight ahead, with no post-processing needed.

Network specific implementation in Tensorflow can be found in 7.4

5.2.3 CNN Training

Assuming we have the ground truth graph G^* from OpenStreetMap, training this particular CNN is non-trivial, as it takes a partial graph G and outputs the probability of walking at various angles, but we only have the mentioned ground truth map.

To properly train the network, the training examples are dynamically generated by running the search algorithm with the CNN as the decision function. As the CNN model gets better, new training samples are generated. Given a pair (a region) (v_0, B), the first step is to initialize an instance of the search algorithm (G, S), where G is the graph (initially containing only the starting position) and S is the vertex

stack. On each step, the CNN is given the input to decide on an action, based on the output layer, and update G and S based on that action.

Furthermore, the optimal decision is also determined (based on G^*) and the CNN is trained with that.

The training strategy consists of, on each step, based on G^*, we first identify the set of angles R where there exist unexplored roads from S_{top}. After that, R is converted into a target output vector O^*. If R is empty, then $o^*_{stop} = 1$. Otherwise, $o^*_{walk} = 1$ and for each angle $\theta \in R$ we have $o^*_i = 1$, where i is the closest walkable angle to θ. In the end, a loss function is computed between O and O^* and back-propagation is run to update the CNN parameters.

A very critical step in this process is how to decide where to start the walk in G^* to pick the next vertex. It may seem logical to start the walk at the closest location in G^* to S_{top}, but as shown in the example in figure 5.3, this approach can direct the algorithm to the wrong pathway when G differs slightly from $G*$.

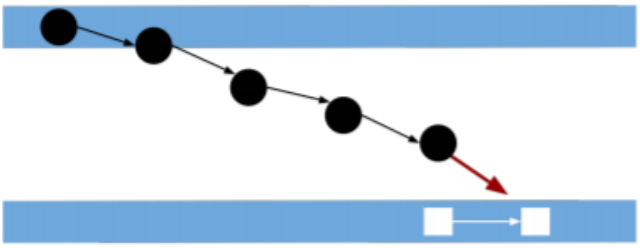


FIGURE 5.3: This is what can happen when you simply match S_{top} the closest location on G^*. In this examples, the black circles represent nodes of G and the blue areas represent actual roads. [6]

To deal with this problem, the authors decided to apply a map-matching algorithm to find a path in G^* that is most similar to a path in G ending at S_{top}. To obtain the path in G, a random walk in G is performed starting at S_{top}, and it is stopped when a certain number of vertices have been traversed or when there are no more vertices adjacent to current one that haven't already been visited in the walk.

Finally, the algorithm maintains a set E containing edges of G^* that have been already explored during the walk. E is initially empty and is filled after map-matching. Then, when performing the walk in G^*, the walk avoids traversing edges that are in E.

5.3 Results

The results for RoadTracer in urban areas are promising according to the authors of the original papers. We will try to reproduce those results later on, but in figure 5.4 there is a visualization of what to expect.

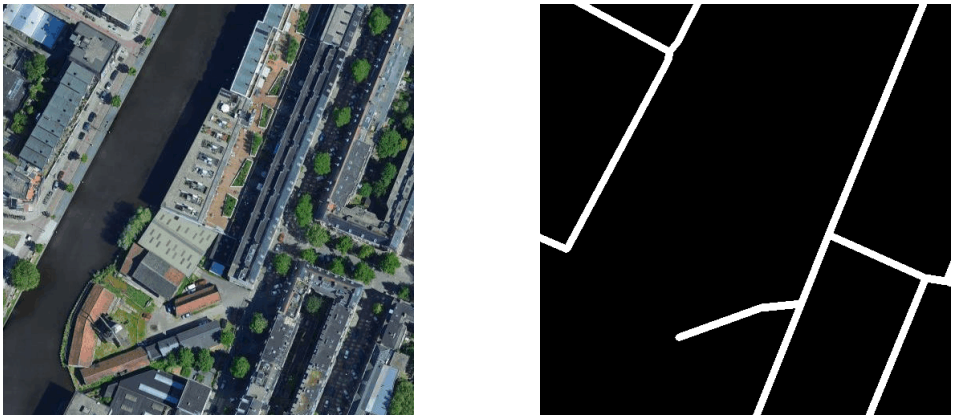
FIGURE 5.4: The results obtained in four different cities by F. Bastani et al. Left column is the result of a CNN semantic segmentation based approach, and right column is the result of applying RoadTracer to the same areas.

Chapter 6

Implementation of Semantic Segmentation for Road Detection

6.1 Introduction

We will be setting our focus in detecting roads from orthophotos using supervised semantic segmentation techniques. This approach can be mirrored and modeled to work on different features and types of terrain. The optimal outcome of the pipeline will consist of using an orthophoto as an input and receiving a mask as the output which locates the roads present in the image. As a reminder we will refer again to an example image as shown in previous chapters (figure 6.1).



(A) Original image (B) Segmented image

FIGURE 6.1: An example of segmentation on an orthophoto

In this section we will describe the pipeline we have implemented to tackle this problem and justify the decisions we have taken given the data and the resources available. Finally we showcase examples of the results the different tested models obtained and a comparison between the performance.

Furthermore, the technologies used for the implementation and the different sources for the models will be described in this and the following chapter.

6.2 Data gathering and preparation

For this task we require both aerial images of terrain and the masks indicating the location of the roads in the images. For this purpose, the idea was to use specific imagery from the Institut Cartogràfic i Geològic de Catalunya and use crowdsourced geographic information from the OpenStreetMap (OSM) project [28] to extract information on the roads. Unfortunately this data was not readily available for the development of the project so we had to look for other sources.

The processing and preparation routines and early models were created and tested using data from the DeepGlobe CVPR18 Challenge [16] which consisted of aerial images with accompanying binary maps assigning the corresponding category to each pixel in the orthophoto.

(A) Orthophoto (B) Provided mask

FIGURE 6.2: A sample of the DeepGlobe dataset

Other than the images provided with the challenge we also used later images from selected regions of our choice. To do this, we had to find and automated way to acquire them as well as information on the roads present in them. The orthophotos were extracted from the Google Maps API and the corresponding road information was extracted and converted into masks suitable for our task from the OpenStreetMaps project.

6.3 Data augmentation

When dealing with segmentation of roads the most important data augmentation principle we explored are simple transformations to the images. Using rotations of different angles helps with finding roads that are not in the common directions. For instance, if all the training examples had only vertical and horizontal roads it would be hard to locate diagonal or slightly angled roads in test images.

As for texture and color modifications, the training data had a variety of different terrains such as cities, green areas and dry areas which already provided the necessary types of textures the images could be so no modifications had to be made to account for other possible climates.

6.4 Model Pipeline

The pipeline for the implementation of the model is heavily dependent on the objective at task. Given the nature of our data, the idea is to first develop a general model using a variety of images from all around the globe with different climates and image tones. This model will serve as the baseline model from which we will retrain the it to perform segmentation for specific purposes. As described in the *Transfer Learning* section, this is beneficial over starting from a random state.

In our case, we will retrain the model using the images and masks from a specific section we want to explore. This can be useful when some of the existing roads or features are known and annotated but others are not. This would provide the model with extra information on specifically this area to help tune the general model and this way find features that the current annotations did not have. Both these models are trained with augmented data to help find, in this case, roads by using translations and rotations of existing roads as input features.

As for specific models, after reading on a variety of different models we ended up testing a classical encoder-decoder architecture by implementing the U-Net model and an architecture which uses diluted convolutions such as DeepLabv3. Given the data available, both approaches resulted in results which were very similar visually. Given this fact, we focused on understanding why certain segmentations were being done correctly and why some results were not accurate, to understand how the models were transforming the input images and treating the extracted features. In the following section we will look at visual examples of results extracted from a test set of sample images of the same DeepGlobe dataset used to train the model.

6.5 Results

Different models and parameters were tested with sample images from the DeepGlobe dataset [16]. In this section we will showcase some examples of how the best models performed in different circumstances and our interpretation on why they succeeded or failed depending on the characteristics of the image.

The first two examples we showcase (figures 6.3, 6.4, 6.5 and 6.6) show two examples of what are probably the easiest roads to detect, those where the color of the image clearly differentiates the roads from the rest of the image. In both examples, we can see how the predicted mask from the model returns a very close road map to that of the ground truth. These types of situations can be seen in areas with vegetation but with wide enough roads so that they are clearly visible from the orthophoto.

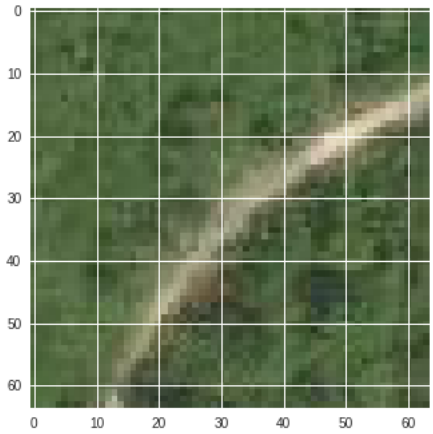
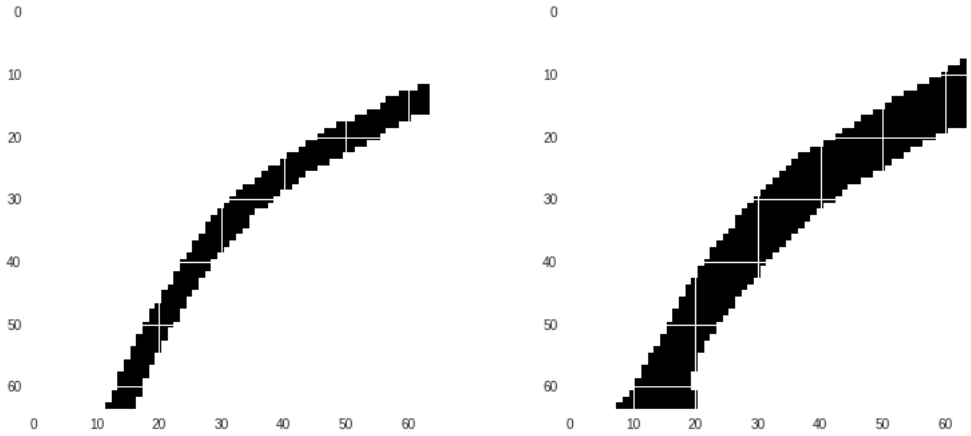


FIGURE 6.3: Original image



(A) Ground truth (B) Predicted segmentation

FIGURE 6.4

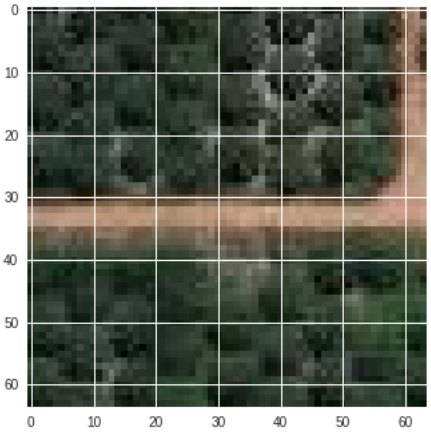
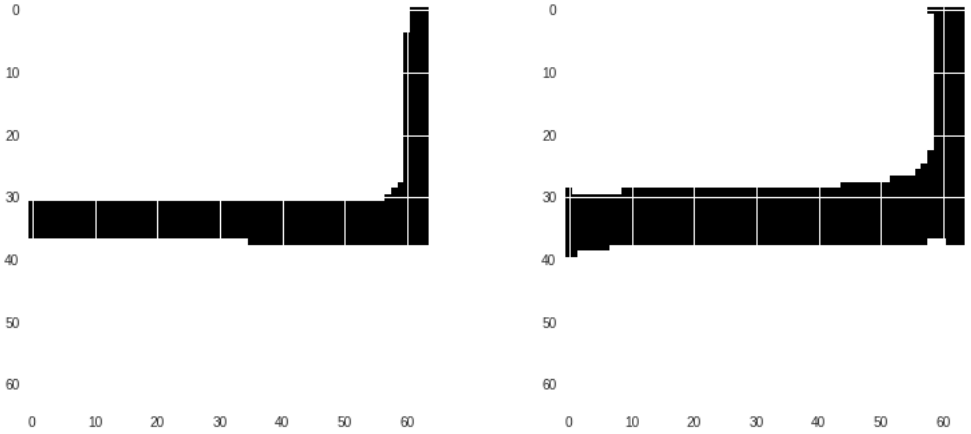


FIGURE 6.5: Original image



(A) Ground truth (B) Predicted segmentation

FIGURE 6.6

The following case we want to showcase is the phenomenon we see happening in figures 6.7, 6.8, 6.9 and 6.10. In cases like these, we have the situation where there are more than just two color tones in the image (road and non-road terrain). This makes the model segment some sections of the image as a road if it holds characteristics similar to those of a road, despite it maybe not being one. At least in the two examples showcased, the ground truth does not include a second section of road while the model does predict one. These *non-roads* look very close to roads to the naked eye and could be classified as roads if there is no more context to the image. Examples like these solve one of the objectives we wanted to complete, to automatically find what could possible be new roads that have not been labeled yet.

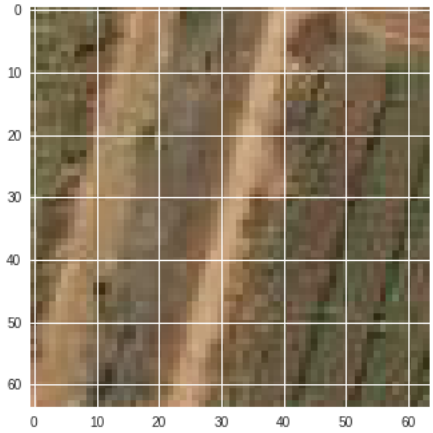
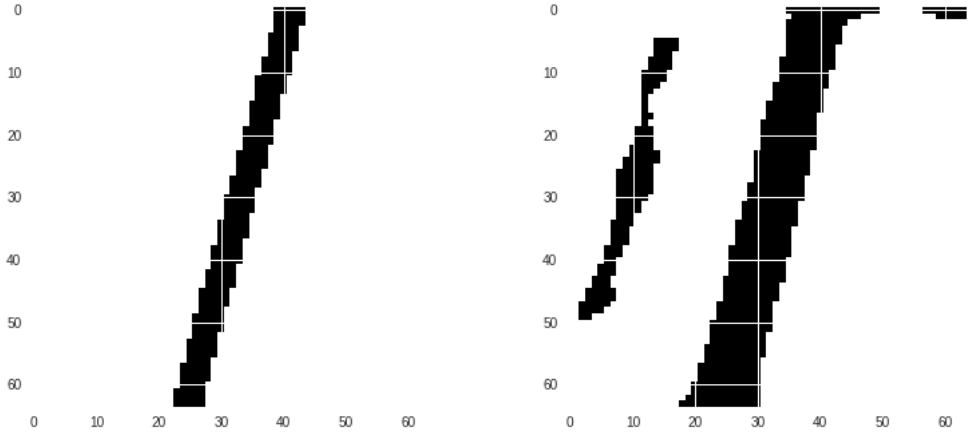


FIGURE 6.7: Original image



(A) Ground truth (B) Predicted segmentation

FIGURE 6.8

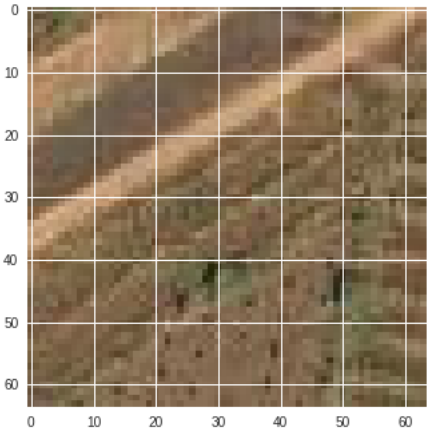
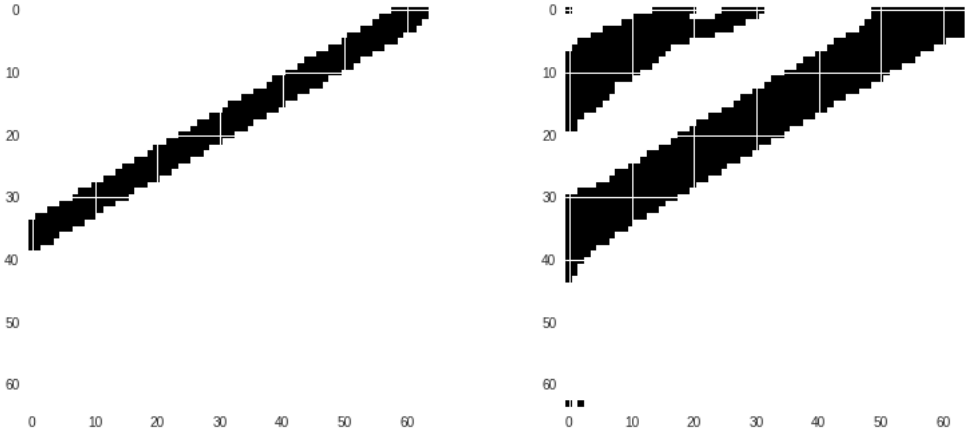


FIGURE 6.9: Original image



(A) Ground truth (B) Predicted segmentation

FIGURE 6.10

We have seen what could be false positives, or true positives that the ground truth had not found. Having too many of these though would be problematic as it would defeat the purpose of the task, to semi-automatize the process of finding roads or other features that had not been located before. If the model has a tendency to segment too many *non-roads* as roads then the manual review process would still be very labour intensive. Here we show a small path of forest which does not have any roads, and how the model correctly interprets that none of the pixels in the image correspond to a road.

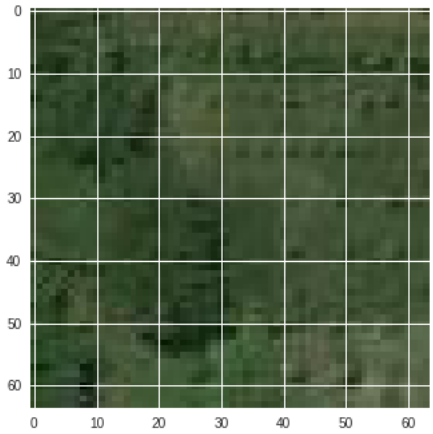
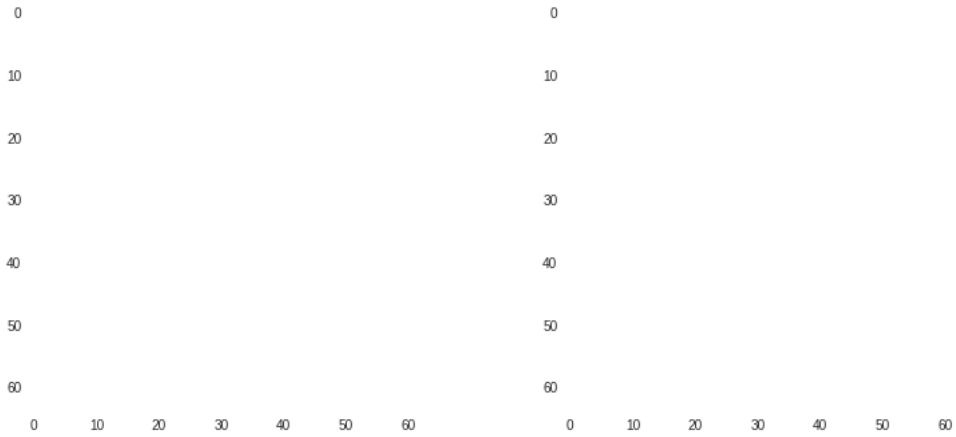


FIGURE 6.11: Original image



(A) Ground truth (B) Predicted segmentation

FIGURE 6.12

Finally, we want to discuss the issues present with using these segmentation techniques in urban areas. In contrast with the examples from rural areas we have showcased, urban areas proved to be much more complicated to map. This is partially due to the low contrast in colors between the buildings and the roads (many share the same tones of colors) and the problems that arose when shadows from the buildings were involved, which made it much harder for the models to distinguish between the classes.

To solve this issue, we expect that having a bigger training set consisting of urban areas to train on top of the existing models could improve performance. Furthermore and more specifically for roads, other approaches can be used which use the characteristics of roads to their advantage such as the *RoadTracer* model discussed previously.

Chapter 7

RoadTracer for Road Detection in Urban and Semi-urban Areas

7.1 Introduction

RoadTracer is a road detection algorithm developed by an MIT research team. In Chapter 5 we explained how it works at a theoretical level, iteratively building a graph that becomes the road network we draw in the end. In this chapter we will dig a bit deeper into the implementation of such algorithm and the particular pipeline we have followed in order to replicate the results for new regions.

7.2 Data gathering

As explained in Chapter 6, the data used for this experiment is a set of images obtained via the Google Maps API and the road network information obtained through OpenStreetMaps API. For that purpose we have used two GO scripts we found in the RoadTracer public repository, which after defining a bounding box (a pair of latitude and longitude points) and a region name, download the imagery desired and the network information in a graph format (a set of longitude and latitude pairs).

To define such bounding box coordinates, we selected the desired points and checked them in google maps. For instance, in figure 7.1 we can see the result of setting up the coordinates to obtain Barcelona imagery and road network data with the following code.

```
var regionMap map[string][4]float64 = map[string][4]float64{
    "barcelona": [4]float64{41.395419, 2.132807, 41.380222, 2.176477},
    "floresta":  [4]float64{41.434098, 2.054152, 41.454994, 2.090476}
}
```

7.3 Data preparation

This data gathering step gives us the image desired, but the graph file obtained is a list of points in (latitude, longitude) format. We need to convert those points to pixel coordinates if we want to be able to draw the network. For the conversion, another GO script is used. It converts the geographic coordinates to pixel position values just by taking the top left corner coordinates as origin and computing the relative position of each point in the graph from the origin and transforming the geographical coordinates to flat coordinates to use in the image.

For computational purposes, the images are split in several tiles of a fixed size and a starting location has to be set for each of the tile sets. This starting location has

FIGURE 7.1: Barcelona image for the bounding box defined in the code

to be a point placed in a road, and it will be where the iterative algorithm will start at.

7.4 Implementation of the decision function

In 5.2.2 we talked about the decision function that implemented the step of choosing which direction to build the road network to, or to stop and walk backwards to the last crossroad if there was no more graph to be explored from that node.

Even though we mentioned its input layer and its output layer, the best way of showing exactly how the network works is by looking at the Tensorflow implementation. Therefore, here is the code for the network:

First things first, the definition of the inputs and the target variables as Tensorflow placeholders.

```
inputs = tf.placeholder(tf.float32, [None, 256, 256, input_channels])
angle_targets = tf.placeholder(tf.float32, [None, 64])
action_targets = tf.placeholder(tf.float32, [None, 2])
detect_targets = tf.placeholder(tf.float32, [None, 64, 64, 1])
learning_rate = tf.placeholder(tf.float32)
```

The definition of the network layers is a composition of convolutional layers with an increasing size.

```
layer1 = _conv_layer('layer1', inputs, 2, input_channels, 128)
layer2 = _conv_layer('layer2', layer1, 1, 128, 128)
layer3 = _conv_layer('layer3', layer2, 2, 128, 256)
layer4 = _conv_layer('layer4', layer3, 1, 256, 256)
layer5 = _conv_layer('layer5', layer4, 1, 256, 256)
layer6 = _conv_layer('layer6', layer5, 1, 256, 256)
layer7 = _conv_layer('layer7', layer6, 2, 256, 512)
```

```
layer8 = _conv_layer('layer8', layer7, 1, 512, 512)
layer9 = _conv_layer('layer9', layer8, 2, 512, 512)
layer10 = _conv_layer('layer10', layer9, 1, 512, 512)
layer11 = _conv_layer('layer11', layer10, 2, 512, 512)
layer12 = _conv_layer('layer12', layer11, 1, 512, 512)
layer13 = _conv_layer('layer13', layer12, 1, 512, 512)
layer14 = _conv_layer('layer14', layer13, 2, 512, 512)
layer15 = _conv_layer('layer15', layer14, 1, 512, 512)
layer16 = _conv_layer('layer16', layer15, 1, 512, 512)
layer17 = _conv_layer('layer17', layer16, 2, 512, 512)
```

This composition is passed through one last output convolutional layer and a softmax function and then lost functions are defined.

```
detect_pre_outputs = _conv_layer('detect_pre_outputs', layer17, 1, 256, 2)
detect_outputs = tf.nn.softmax(detect_pre_outputs)[:, :, :, 0:1]
action_pre_outputs = _conv_layer('action_pre_outputs', layer17, 2, 512, 2,
    {'activation': 'none'})[:, 0, 0, :]
action_outputs = tf.nn.softmax(action_pre_outputs)
angle_outputs = _conv_layer('angle_outputs', layer17, 2, 512, 64,
    {'activation': 'sigmoid'})[:, 0, 0, :]

detect_loss = tf.reduce_mean(tf.square(detect_targets - detect_outputs))
angle_loss = tf.reduce_mean(
        tf.reduce_mean(tf.square(angle_targets - angle_outputs), axis=1)
        *
        action_targets[:, 0]
)
action_loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    labels=self.action_targets, logits=self.action_pre_outputs))
loss = self.angle_loss * 50 + self.action_loss + self.detect_loss * 5
```

We optimize with respect to the loss function, which is a composition between the three loss functions for angle, action, and detection.

7.5 Inferring a network on a new region

Once we have defined the neural network that will implement the decision process, gathered, and prepared the data, we are ready to infer a road network on the imagery we have downloaded.

Notice there is no training step, since we have not been able to train the road-tracer model ourselves. Google Colab (8.2) uses Google Drive as its storage system, and Google Drive free layer has 15 gigabytes only. It is quite obvious that the amount of high detail aerial images it takes to train a model of this characteristics exceeds the free storage Google Drive offers.

On the other hand, our particular computers and laptops with tensorflow-gpu installed lack of the computational power to train such a network in reasonable time. Therefore, we trained for 36 hours and managed to obtain partial results of the training process. In figure 7.2 we can see a step of the training iterative method. The yellow rings indicate the possible angles to take, and the intensity of the color identifies the probability of taking that angle. The green line is the ground truth

graph, with the red line indicating the already inferred pathway and the two dots representing the current point and the potential vertex to add to the graph.

FIGURE 7.2: A representation of a training step.

Because of the reasons mentioned, we have used a previously trained model the authors made available at their website and inferred a road network over the city of Barcelona and the semi-urban area of La Floresta near Barcelona.

7.6 Data post-processing

Since the model output is a set of nodes in pixel coordinates, we will need to do some post-processing on the data in order to achieve readable results. With that purpose we used a script that reads the graph file created by the inference step and generates an SVG visual representation of the graph. After converting that SVG to a png file and turning the background transparent by using the open source tool GIMP, we have programmed a python scripts that let us overlap the model output (or the ground truth visual graph) with the images of the city inferred, giving out a qualitative notion of the model performance. We also have created another python script that allows us to blend both the truth and the model visualizations for the sake of comparison.

7.7 Results

Even though we would really like to have a lot of result tables comparing the performance of traditional segmentation based models and this new approach that RoadTracer brings to the table focusing in metrics, there is one major obstacle we have been unable to surpass and therefore all we have are qualitative results. Semantic segmentation outputs a black and white image that has the same dimension as the

image it worked on, with white pixels corresponding to roads and black pixels corresponding to all other features. This implies that the notion of road thickness is intrinsic to the model's output.

On the other hand, the outcome of RoadTracer is an array of pixel coordinates that represents each node of the solution graph for the input image. In the post-processing stage we drew a visual representation of the graph by joining the nodes through a line, but since RoadTracer is more focused in the topological properties of the road instead of the metric ones, all we can get is a fixed width line joining the road nodes, which is not comparable at a quantitative level to the output of semantic segmentation.

Hence, we have only been able to produce qualitative results.

7.7.1 Semi-urban areas

The results shown in the original paper were promising enough for us to assume (although we reproduced them as well) that we would obtain a decent outcome when applying roadtracer to Barcelona. Nevertheless, we wondered if it could obtain equally decent results in semi-urban areas.

Figure 7.3 shows the ground truth graph overlapped with the image of the region La Floresta.

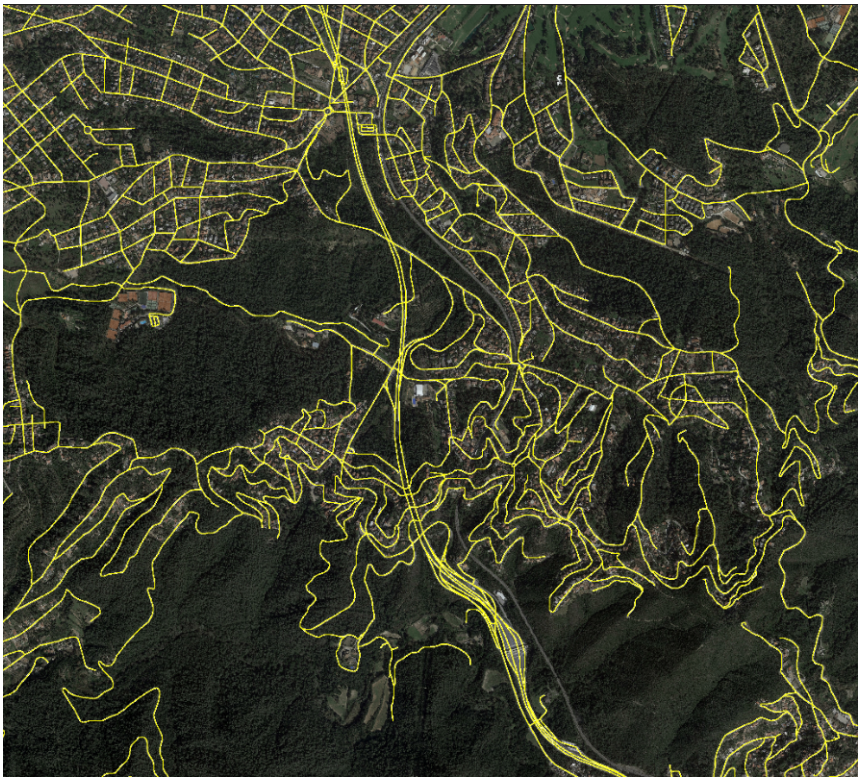


FIGURE 7.3: Truth graph visualization for the semi-urban area of La Floresta

After running inference in this region with the default parameters, we obtained the results seen in figure 7.4, which compared to the real network in 7.3, is obviously missing a lot of roads, conveniently coinciding with the less urban-like areas.

In order to try to improve the quality of the inference, we tweaked the code and changed the threshold value for which the decision function decides whether or

FIGURE 7.4: Model output visualization for the semi-urban area of La Floresta

not to draw a road. The default parameter was 0.4, and figure 7.5 shows the result obtained for $T = 0.3$. For this value it is more likely to draw a road, and we can see it captures some things it did not before. But it also invents a lot of roads through the forest that simply do not exist, creating a lot of false positives that we consider to be more critical than just missing roads. Hence, we consider the result of the default parameters to be better.

Finally in figure 7.6 we drew the difference between the real network and the first we inferred in 7.4.

7.7.2 Urban areas

After trying out semi-urban regions, we also tried to infer the road network within a crowded city such as Barcelona. Just as in the previous region, figure 7.7 shows the real network, while figures 7.8 and 7.9 show the model output for thresholds 0.3 and 0.4. Finally, figure 7.10 shows the difference between reality and inference.

FIGURE 7.5: Model output visualization for the semi-urban area of La Floresta for a different threshold of the decision function.

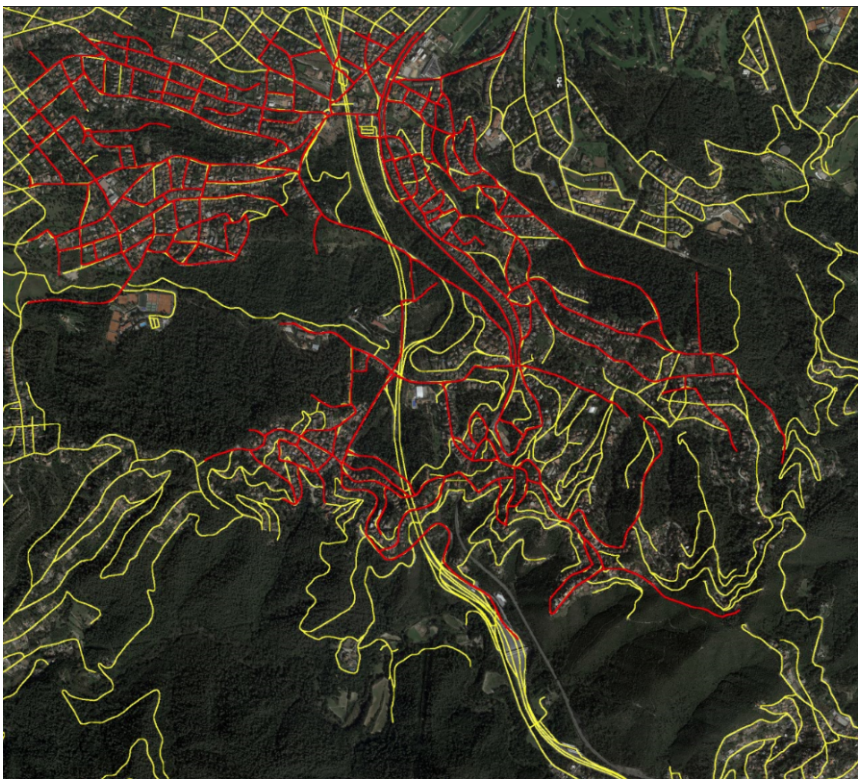


FIGURE 7.6: Difference between reality and the model output. Yellow line is the real network and red is the model output.

FIGURE 7.7: Ground truth graph visualization for Barcelona.

FIGURE 7.8: Model output graph visualization for Barcelona.

FIGURE 7.9: Model output graph visualization for Barcelona with $T = 0.3$.

FIGURE 7.10: Difference between reality and the model output. Yellow line is the real network and red is the model output.

Chapter 8

Technologies Used

8.1 Deep Learning Frameworks

8.1.1 Tensorflow

TensorFlow is an open source software library for high performance numerical computation. It is designed to allow deployment in a variety of platforms such as CPUs or GPUs and to allow scalability from laptops to clusters of servers. Originally developed by Google AI, it has great support for machine learning and deep learning, and we have used it to run the RoadTracer implementation by Bastani et Al.

8.1.2 Keras

For this project we have also used Keras. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. Developed with a focus on enabling fast experimentation, it is designed to achieve very fast results with very few lines of code. The semantic segmentation approaches to feature detection in this project are all developed in Keras.

8.2 Google Colaboratory

Google Colaboratory (Colab) is a research project by Google created to easily share research content among team members and also to be used with teaching purposes. It is essentialy a Jupyter Notebook environment that is available in the cloud and does not need to be configured to use AI tools such as Keras or Tensorflow.

It allows for free GPU computation power and uses Google Drive as its storage backend. All of our semantic segmentation models were developed and trained in Colab.

8.3 The Go programming language

Although we have not really programmed in Go for this project, all of the RoadTracer preprocessing scripts were in that language, hence we have had to learn its basics in order to understand and tweak the code.

Go is an open source programming language developed by a team at Google that is gaining popularity in the last years for its easy to use implementation of multithreading.

8.4 OpenStreetMap

OpenStreetMap is a community driven platform that gathers geographical data and uses it to build and offer free maps of very diverse properties. From train networks, roads or vegetation, they all are accessible via their API, which we have used to obtain road networks of the selected regions.

8.5 Google Maps

Similar to OpenStreetMaps, Google Maps offers data and aerial imagery (taken from several altitude levels) to the average user. Unlike OpenStreetMaps, it is not free to access programmatically, but there is a free layer that can be used to a certain extent, and that is what we have used to obtain aerial images of the regions we worked on.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

We started this project with the intention of implementing an automatic way of extracting map features through aerial images. Within the duration of the project, the appearance of RoadTracer made us want to take a more scientific approach into the project and turn it into a study of image segmentation techniques and how they can be applied to road detection, and more important, how do they perform compared to this new promising approach. For that purpose, and given that RoadTracer is specific to road detection, we used road detection as a study case for the rest of the project.

After conducting several experiments and trials, we realized the results were not comparable at a quantitative level since the outputs of both approaches would require a massive post-processing and even after doing it, we could not guarantee the reliability of the results.

On the other hand, we were able to produce and compare qualitative results for both approaches and confirm the hypothesis we had made before the practical tests were even made, just by assuming the techniques would work according to their theoretical design. Even though there is a high diversity of CNN based semantic segmentation techniques and they can be trained with several types of imagery to obtain quite good results, they all lower their performance in urban areas due to one issue: continuity. Road continuity is an assumption that segmentation methods do not make beforehand, hence in urban areas, with lots of trees and high buildings, it is quite easy to miss pieces of road.

RoadTracer crushes segmentation results by using an approach that explicitly considers road continuity, but at the cost of performing very poorly in semi-urban or rural areas, where it tends to miss pathways or infer new ones where there is nothing.

We think the application of this approach to automatically find new roads will save a lot of work to map makers in the future, although it needs deeper quantitative validation to be sure the results are reliable.

9.2 Future Work

After struggling without success trying to achieve a solid comparison between the two methods mentioned through the document, we strongly believe that the immediate step to take would be to implement a thorough data post-processing that made it viable to compare a segmentation approach to RoadTracer in the same region in terms of statistical terms such as accuracy or recall.

Another improvement that could be done is to work on the RoadTracer implementation to somehow make the decision making CNN infer the width of the road. That could save the so mentioned post processing required in order to properly compare results.

Appendix A

Team members contribution

During the first months of the project, since we didn't have the data we were supposed to work on, both of us worked on understanding and developing baseline semantic segmentation models such as uNet and also creating python libraries to fulfill our specific needs in the preprocessing stage.

After deciding we were focusing on road segmentation with the appearance of RoadTracer, the work was divided as follows.

1. **Marc Beltrán**: Implementation in Google Colab of most of the semantic segmentation algorithms in this project and development of data gathering, data preprocessing and postprocessing libraries and utilities.

2. **Albert Companys**: Analysis and reproduction of the RoadTracer paper results, porting the code to python 3. Also development of data gathering, data processing and postprocessing libraries and utilities.

Appendix B

Code Repository

All the code used to develop this project can be found in the public repository at https://github.com/MarcBeltran/TFM

Bibliography

[1] In: (). URL: http://host.robots.ox.ac.uk/pascal/VOC/voc2007/index.html.

[2] P. Kohli A. Blake and C. Rother. "Markov random fields for vision and image processing". In: *Mit Press* (2011).

[3] X. Jiang P. J. Flynn-H. Bunke D. B. Goldgof K. Bowyer D. W. Eggert A. Fitzgibbon A. Hoover G. Jean-Baptiste and R. B. Fisher. "An experimental comparison of range image segmentation algorithms". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 18, no. 7, pp. 673–689* (Jul. 1996.). URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=506791.

[4] M. S. Atkins and B. T. Mackiewich. "Fully automatic segmentation of the brain in mri". In: *Medical Imaging, IEEE Transactions on, vol. 17, no. 1, pp. 98–107* (Feb. 1998). URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=668699.

[5] V. Badrinarayanan, A. Kendall, and R. Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *ArXiv e-prints* (Nov. 2015). arXiv: 1511.00561 [cs.CV].

[6] Favyen Bastani et al. "RoadTracer: Automatic Extraction of Road Networks from Aerial Images". In: *CVPR* abs/1802.03680 (2018). arXiv: 1802.03680. URL: https://arxiv.org/abs/1802.03680.

[7] C. J. Burges. "A tutorial on support vector machines for pattern recognition". In: *Data mining and knowledge discovery, vol. 2, no. 2, pp. 121–167* (1998).

[8] L. Davis C. Huang and J. Townshend. "An assessment of support vector machines for land cover classification". In: *International Journal of remote sensing vol. 23, no. 4, pp. 725–749* (2002).

[9] A. Blake C. Rother T. Minka and V. Kolmogorov. "Cosegmentation of image pairs by histogram matching - incorporating a global constraint into mrfs". In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on, vol. 1, pp. 993– 1000* (June 2006). URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1640859.

[10] J. Luo C. W. Chen and K. J. Parker. "Image segmentation via adaptive k-mean clustering and knowledge-based morphological operations with biomedical applications". In: *Image Processing, IEEE Transactions on, vol. 7, no. 12, pp. 1673–1683* (Dec. 12 1998). URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=730379.

[11] Liang-Chieh Chen et al. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: *CoRR* abs/1606.00915 (2016). arXiv: 1606.00915. URL: http://arxiv.org/abs/1606.00915.

[12] Liang-Chieh Chen et al. “Rethinking Atrous Convolution for Semantic Image Segmentation”. In: *CoRR* abs/1706.05587 (2017). arXiv: 1706.05587. URL: http://arxiv.org/abs/1706.05587.

[13] Liang-Chieh Chen et al. “Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs”. In: *CoRR* abs/1412.7062 (2014). arXiv: 1412.7062. URL: http://arxiv.org/abs/1412.7062.

[14] J. Morimotot T. Asfour D. Schiebener A. Ude and R. Dillmann. “Segmentation and learning of unknown objects through physical interaction”. In: *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on. IEEE, 2011, pp. 500–506* (2011). URL: http://ieeexplore.ieee.org/ielx5/6086637/6100798/06100843.pdf.

[15] N. Dalal and B. Triggs. “Histograms of oriented gradients for human detection”. In: *Computer Vision and Pattern Recognition. CVPR 2005. IEEE Computer Society Conference* (vol. 1, June 2005, pp. 886–893 vol. 1.). URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1467360.

[16] “DeepGlobe Challenge”. In: (). URL: http://deepglobe.org/index.html.

[17] Y. Sun H.-D. Cheng X. Jiang and J. Wang. “Color image segmentation: advances and prospects”. In: *Pattern recognition, vol. 34, no. 12, pp. 2259–2281,* (2001).

[18] et al. He Kaiming. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.” In: *2015 IEEE International Conference on Computer Vision (ICCV), doi:10.1109/iccv.2015.123.* (2015).

[19] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[20] T. K. Ho. “Random decision forests”. In: *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on, vol. 1. IEEE, 1995, pp. 278–282.* (1995). URL: http://ect.bell-labs.com/who/tkh/publications/papers/odt.pdf.

[21] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[22] et al. Lecun Y. “Gradient-Based Learning Applied to Document Recognition.” In: *Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324., doi:10.1109/5.726791.* (1998).

[23] J. Long, E. Shelhamer, and T. Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *ArXiv e-prints* (Nov. 2014). arXiv: 1411.4038 [cs.CV].

[24] L. Grady M. D. Collins J. Xu and V. Singh. “Random walks based multi-image segmentation: Quasiconvexity results and gpu-based solutions”. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. IEEE, 2012, pp. 1656–1663.* (2012). URL: http://pages.cs.wisc.edu/~jiaxu/pub/rwcoseg.pdf.

[25] Faisal Mahmood and Nicholas J. Durr. “Deep Learning and Conditional Random Fields-based Depth Estimation and Topographical Reconstruction from Conventional Endoscopy”. In: *CoRR* abs/1710.11216 (2017). arXiv: 1710.11216. URL: http://arxiv.org/abs/1710.11216.

[26] "Memory based active contour algorithm using pixel-level classified images for colon crypt segmentation". In: *Computerized Medical Imaging and Graphics* (Nov. 2014). URL: http://mis.haifa.ac.il/~ishimshoni/SegmentCrypt/Active\%20contour\%20based\%20on\%20pixellevel\%20classified\%20image\%20for\%20colon\%20crypts\%20segmentation.pdf.

[27] K. Van Leemput N. Moon E. Bullitt and G. Gerig. "Automatic brain and tumor segmentation". In: *Medical Image Computing and Computer-Assisted Intervention—MICCAI 2002. Springer, 2002, pp. 372–379.* (2002).

[28] "OpenStreetMap". In: (). URL: https://www.openstreetmap.org/.

[29] Antti Rasmus et al. "Semi-Supervised Learning with Ladder Network". In: *CoRR* abs/1507.02672 (2015). arXiv: 1507.02672. URL: http://arxiv.org/abs/1507.02672.

[30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[31] D. Cohen G. Elidan S. Gould J. Rodgers and D. Koller. "Multi-class segmentation with relative location prior". In: *International Journal of Computer Vision, vol. 80, no. 3, pp. 300–316* (Apr. 2008).

[32] E. Hoffman S. Hu and J. Reinhardt. "Automatic lung segmentation for accurate quantitation of volumetric x-ray ct images". In: *Medical Imaging, IEEE 13 Transactions on, vol. 20, no. 6, pp. 490–498* (Jun. 2001).

[33] P. GilJimenez H. Gomez-Moreno S. Maldonado-Bascon S. Lafuente-Arroyo and F. LopezFerreras. "Road-sign detection and recognition based on support vector machines," Intelligent Transportation Systems". In: *Intelligent Transportation Systems, IEEE Transactions on, vol. 8, no. 2, pp. 264–278, Jun. 2007. [Online].* (). URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4220659.

[34] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[35] L. I. Smith. "A tutorial on principal components analysis". In: *Cornell University, USA, vol. 51, p. 52* (2002).

[36] O. Veksler Y. Boykov and R. Zabih. "Fast approximate energy minimization via graph cuts". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 23, no. 11, pp. 1222–1239* (2001). URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=969114.

[37] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *CoRR* abs/1411.1792 (2014). arXiv: 1411.1792. URL: http://arxiv.org/abs/1411.1792.

[38] Fisher Yu and Vladlen Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions". In: *CoRR* abs/1511.07122 (2015). arXiv: 1511.07122. URL: http://arxiv.org/abs/1511.07122.

[39] Hengshuang Zhao et al. "Pyramid Scene Parsing Network". In: *CoRR* abs/1612.01105 (2016). arXiv: 1612.01105. URL: http://arxiv.org/abs/1612.01105.