Technical University of Crete
School of Electrical and Computer Engineering

Deep Learning Techniques for Detecting Crossroads in Satellite Images

DIPLOMA THESIS

Author
Theodoros Papadopoulos

Chania 2018

Πολυτεχνείο Κρήτης
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τεχνικές Μάθησης σε Βάθος για Ανίχνευση Διασταυρώσεων σε Δορυφορικές Εικόνες

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Θεόδωρου Παπαδόπουλου

Εξεταστική Επιτροπή

Αναπληρωτής Καθηγητής Μιχαήλ Γ. Λαγουδάκης (Επιβλέπων)
Καθηγητής Μιχάλης Ζερβάκης
Επίκουρος Καθηγητής Παναγιώτης Παρτσινέβελος (Σχολή ΜΗΧΟΠ)

Χανιά 2018

Abstract

The ultimate goal of machine learning and artificial intelligence in general is and has always been to try and create intelligent machines that mimic the way the human brain thinks. Deep Neural networks are a very promising step towards that direction. The downside is they require huge amounts of computational power and vastly large datasets to be trained correctly and achieve their full potential. In this thesis, we study the application of deep learning methods to the analysis of satellite earth images for automated detection of crossroads. To overcome the training problem, we decided to use a technique called Transfer Learning. With transfer learning we take an already trained deep neural network, extract the acquired knowledge (the weights of the neurons) and re-train the final layers of it with our very own dataset. We apply this massive force of object detection on satellite imagery due to potential applications in many activities, such as urban planning, crop and forest management, disaster relief, and more. We decided to focus on a single type of objects, namely crossroads, because they are intriguing objects with a lot of variability, considering that crossroads never have the same shape or color and their view may be obstructed by trees or buildings. We created a training data set using Google Maps for input images and OpenStreetMaps for identifying points of ground truth. We then re-trained a deep network using the TensorFlow Objection Detection package. The final crossreoad detector performs quite well on a variety of satellite images from different cities around the world. Finally, we created a user-friendly, web application, to deliver a platform where even inexperience users can navigate to any desired area on Google maps, perform crossroad recognition using our detector, and displaying the results on top of the input image. Our work can be easily integrated into other applications, but more importantly also provides a guideline on building detectors for other kinds of objects of interest in satellite images.

Περίληψη

Απώτερος στόχος της μηχανικής μάθησης, αλλά και της τεχνητής νοημοσύνης γενικότερα, ήταν πάντα η προσπάθεια να δημιουργηθούν έξυπνα μηχανήματα, που μιμούνται τον τρόπο με τον οποίο σκέφτεται ο ανθρώπινος εγκέφαλος. Τα βαθιά νευρωνικά δίκτυα είναι ένα πολλά υποσχόμενο βήμα προς αυτή την κατεύθυνση. Το μειονέκτημά τους είναι ότι απαιτούν τεράστια υπολογιστική ισχύ, αλλά και ένα γιγαντιαίο όγκο δεδομένων για να εκπαιδευτούν σωστά και να αξιοποιήσουν στο έπακρο το τεράστιο δυναμικό τους. Στην παρούσα διπλωματική εργασία μελετάμε την εφαρμογή μεθόδων μάθησης σε βάθος στην ανάλυση δορυφορικών εικόνων με σκοπό την αυτόματη ανίχνευση διασταυρώσεων σε οδικά δίκτυα. Για να ξεπεραστεί το πρόβλημα της εκπαίδευσης, αποφασίσαμε να χρησιμοποιήσουμε μια τεχνική που ονομάζεται Transfer Learning (Μεταφορά της Μάθησης). Για να εφαρμοστεί η τεχνική αυτή, παίρνουμε ένα ήδη εκπαιδευμένο δίκτυο, εξάγουμε την ήδη αποκτηθείσα γνώση (τα βάρη των νευρώνων του δικτύου) και επανεκπαιδεύουμε τα τελευταία επίπεδα του δικτύου με τα δικά μας δεδομένα. Έπειτα μελετάμε τεχνικές ανίχνευσης αντικειμένων σε δορυφορικές εικόνες, αποβλέποντας στην πιθανή εφαρμογή τους σε πολλές δραστηριότητες, όπως αποστολές έρευνας και διάσωσης, πολεοδομικού σχεδιασμού, διαχείρισης καλλιεργειών και δασών, πρόβλεψης καιρού, ανακούφισης πληγέντων από καταστροφές, κλιματικές αλλαγές και πολλά αλλά. Αποφασίσαμε να εστιάσουμε σε ένα μόνο είδος αντικειμένου, τις διασταυρώσεις, επειδή από πλευράς αναγνώρισης παρουσιάζουν μεγάλο ενδιαφέρον, λόγω της ποικιλομορφίας τους ως προς το σχήμα και το χρώμα, όπως και το γεγονός ότι η ορατότητά τους μπορεί να είναι περιορισμένη εξαιτίας δέντρων, κτιρίων, κλπ. Κατασκευάσαμε το σύνολο των δεδομένων εκπαίδευσης χρησιμοποιώντας το Google Maps για την λήψη εικόνων εισόδου και το OpenStreetMaps για να εντοπίσουμε τις ακριβείς συντεταγμένες πραγματικών διασταυρώσεων (ground truth) που μας ενδιαφέρουν. Στη συνέχεια επανεκπαιδεύσαμε ένα δίκτυο κάνοντας χρήση του πακέτου Tensorflow Object Detection. Το τελικό αποτέλεσμα αποδίδει αρκετά καλά σε μια μεγάλη ποικιλία δορυφορικών εικόνων από διάφορες πόλεις σε όλο τον κόσμο. Τέλος, δημιουργήσαμε μια φιλική προς το χρήστη διαδικτυακή εφαρμογή, για να προσφέρουμε μια πλατφόρμα, όπου ακόμα και αρχάριοι χρήστες μπορούν να πλοηγηθούν σε οποιαδήποτε επιθυμητή περιοχή στους χάρτες της Google, να πραγματοποιήσουν αναγνώριση διασταυρώσεων κάνοντας χρήση του ανιχνευτή μας, και να δουν τα αποτελέσματα πάνω στην εικόνα που επέλεξαν ως είσοδο. Η δουλειά μας μπορεί εύκολα να ενσωματωθεί σε άλλες εφαρμογές, αλλά πιο σημαντικό είναι το γεγονός ότι παρέχει τις κατευθυντήριες γραμμές για την κατασκευή ανιχνευτών για οποιοδήποτε άλλο αντικείμενο ενδιαφέροντος σε δορυφορικές εικόνες.

Table of Contents

Table of Figures

Chapter 1 - Introduction

Machine learning is one of the fastest-growing and most exciting fields in modern science and engineering. Deep learning represents the cutting edge of machine learning and has become increasingly important in recent years. Google Research has already made huge progress in the field of Deep Learning not only with the Google Brain project, but also with the acquisition of the London-based startup DeepMind. Moreover, research results have shown that deep learning methods outperform traditional machine learning approaches in several domains, including the domain of Computer Vision, and by any metric.

In practice though, very few people train an entire Deep Neural Network, for example a Convolutional Network, from scratch with random initialization, mainly for two reasons. First, it is very expensive to train deep neural networks. Second, they require huge amounts of data to be able to achieve their full potential. To overcome such hurdles, a well-known technique commonly used in deep learning is known as "Transfer Learning". The idea of transfer learning is to first obtain a pre-trained model (a deep network), which has already been trained on a large dataset; the network may have been trained on a different task than the one we care about, with the same input though, but with different output. Then, the next step is to find layers within the pre-trained network, which output reusable features that capture important patterns in the data. The last step is to use the output of those layers as input features to a much smaller neural network (corresponding to a small number of parameters) and train this new network with the target data set. This smaller network, which is added on the top of the pre-trained network, only needs to learn the relations for the new problem over the features and patterns in the data that have already been learned before and are already available in the pre-trained network.

In this thesis, we apply the above techniques into a complex problem of machine vision, the analysis of satellite imagery. Aerial image analysis is the process of processing aerial images for the purposes of identifying objects of interest and determining various properties of the identified objects within the images. The process of aerial image analysis originated during the First World War; photos taken from airplanes were closely examined for reconnaissance. Since then, in past century or so, aerial image analysis has found many applications in several diverse areas, such as search and rescue missions, urban planning, crop and forest management, weather prediction, disaster relief, climate modeling, etc.

In our times, when technology made it possible for drones to be available and relatively cheap for anyone to acquire, we believe that fast and accurate processing of such images may have a huge impact and may find use in even more applications.

Finding crossroads in a satellite image is a pretty difficult task using the traditional methods, due to the nature of the images we examine, which contain occlusions, shadows, and a wide variety of non-road objects. We propose a solution using deep learning neural networks in order to speed up the process while at the same time produce pretty accurate results in the vast majority of cases.

1.1 Thesis Outline

Chapter 2 describes in brief the theoretical concepts required for this thesis, such as Image Classification, Localization, Object Detection, Neural Networks, Deep Learning, Transfer Learning, Neural Network training techniques. Also, we mention useful tools and APIs we used such as OpenStreetMap, the open source software library Tensorflow and the Tensorflow Object Detection API.

In Chapter 3 we present our first attempts to solve this problem. We re-train an image classifier using the Inception V3 model provided by Tensorflow and then we apply it on images using the sliding window and image scaling technique.

In Chapter 4 we describe our final and official approach for retraining a deep neural network using the transfer learning technique. We demonstrate the way we constructed our dataset, the process of preparing our data in order to feed them to the network for training, and also the training process itself. Finally, we discuss the tools we used to construct our web application, explain the rationale by which our server works and present some early results.

In Chapter 5 we present our final results. We examine the cases where our model performed well, we compare them with the cases where it did not, and we try to understand the reasons behind any misdetection.

Finally, in Chapter 6 we give an evaluation of the whole thesis, what we learnt, the results we achieved, and suggest future improvements.

Chapter 2 - Background

2.1 Image Classification

The task of Image Classification refers to the process of assigning a label from a fixed set of categories to an input image. However simple it may sound at first, this is one of the core challenges in Computer Vision and has a large variety of practical applications. Moreover, there are many other Computer Vision tasks that may seem to be different (such as object detection, segmentation), but they can be easily traced back to Image Classification. Figure 1 shows a widely-known example of image classification.

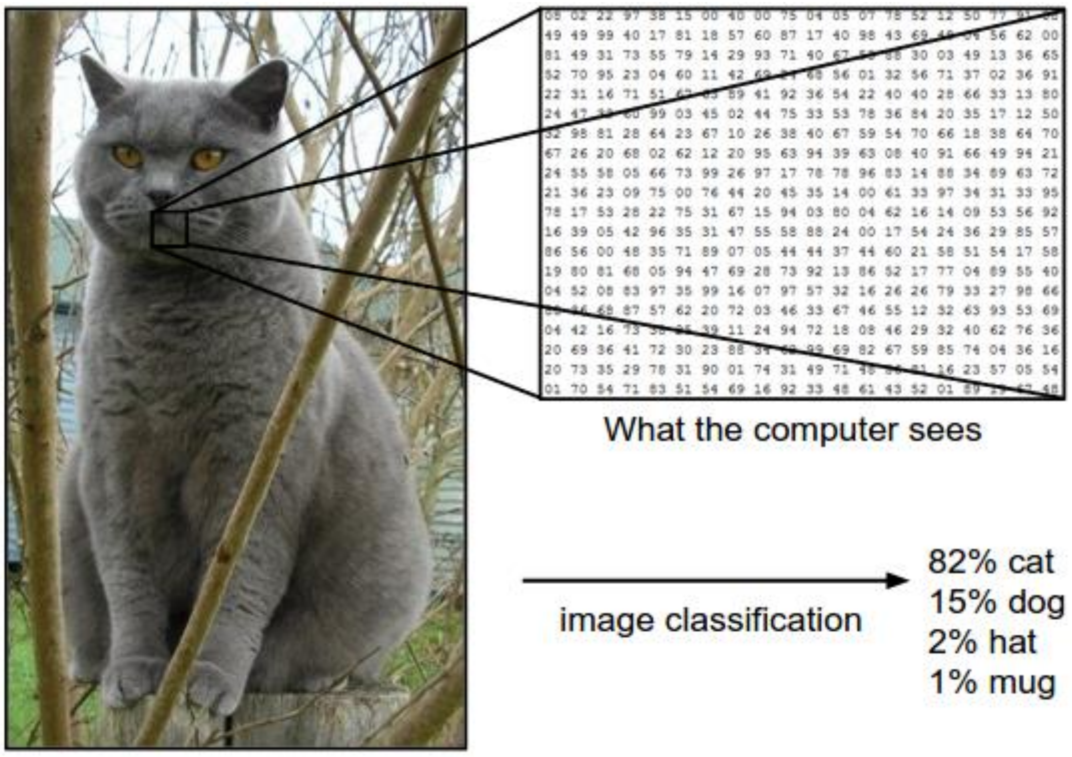


Figure 1 - Image Classification [1]

Since the task of recognizing a visual concept (e.g. cat) is relatively straightforward for a human to perform, it is worth considering the challenges involved for a Computer Vision algorithm, such as viewpoint variation, scale variation, different light conditions, background confusion, deformation, and many others. An efficient image classification model must be invariant to all these factors mentioned before, while at the same time maintain sensitivity to any other class-irrelevant variations.

2.2 Localization

Localization is the process of locating a single object within a given image. Figure 2 shows a simple example of localizing a cat within an image.

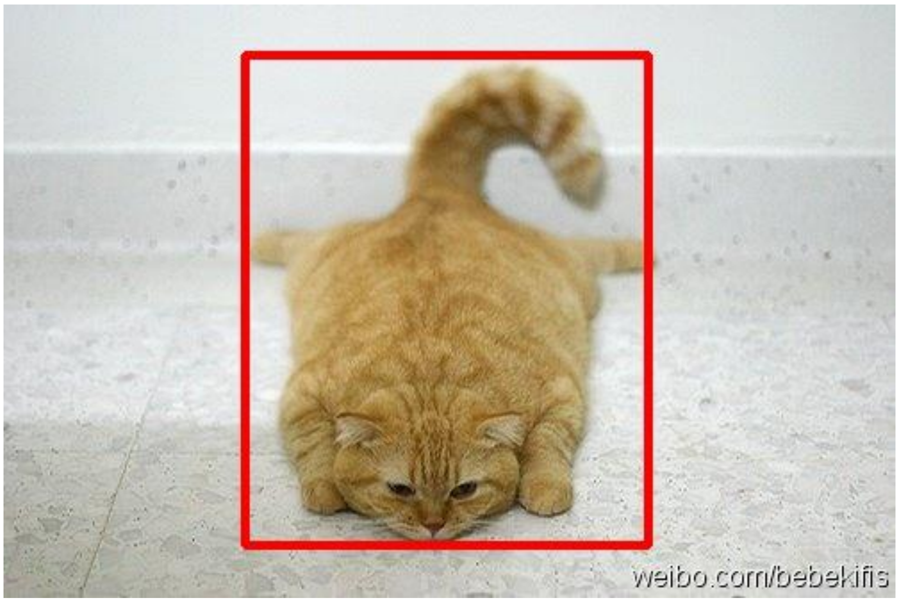


Figure 2 – Localization [2]

2.3 Object Detection

Combining the tasks of classification and localization, we realize the necessity for detecting and classifying objects simultaneously. Object detection is the process of finding and assigning a label to a number of objects within a given image, most of the times involving several classes (more than one). Figure 3 depicts an example of detecting three types of objects (cat, dog, bird).

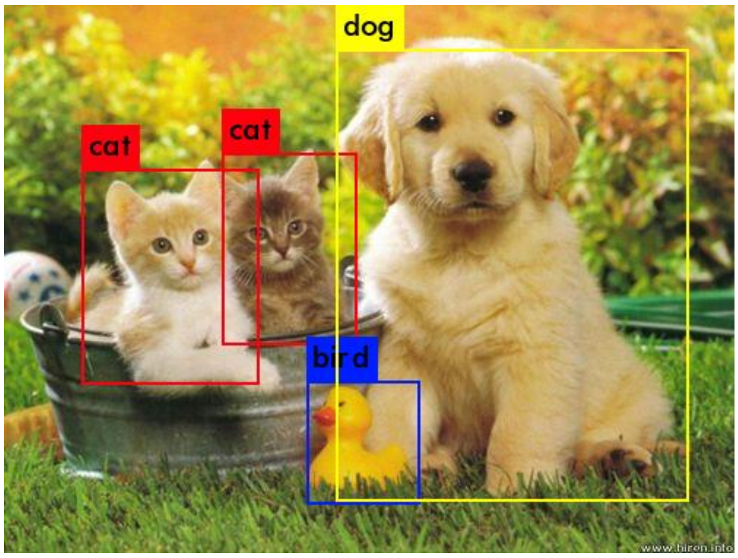


Figure 3 - Object Detection [3]

Traditional methods of detection involved using a block-wise orientation histogram (SIFT or HOG) feature which could not achieve high accuracy in standard datasets, such as PASCAL VOC. These methods encode very low-level characteristics of the objects and therefore are not able to distinguish well among the different labels. In our days, Deep learning methods, such as learning with convolutional networks, have become the state of the art in object detection in an image.

2.4 Neural Networks

The ultimate goal of machine learning and artificial intelligence in general is and has always been to try and create intelligent machines that mimic the way the human brain thinks. Neural networks are the first very promising step towards that direction.

We like to think that Neural Networks are machines that weigh the evidence they receive and take decisions based on them. Similar to the human brain, they consist of a large number of neurons, all of which are interconnected to each other and collectively form a powerful system capable of great things.

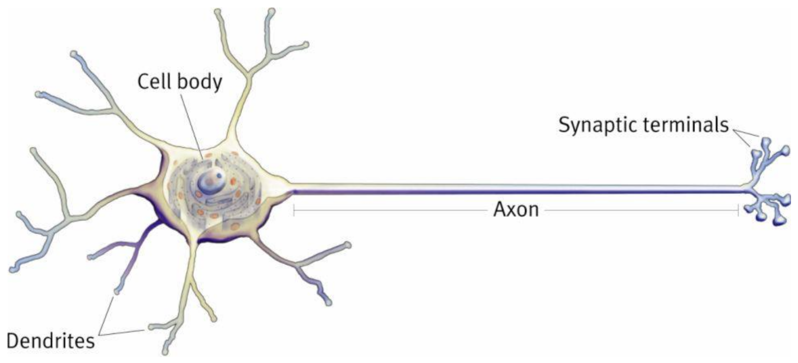


Figure 4 - Natural Neuron [4]

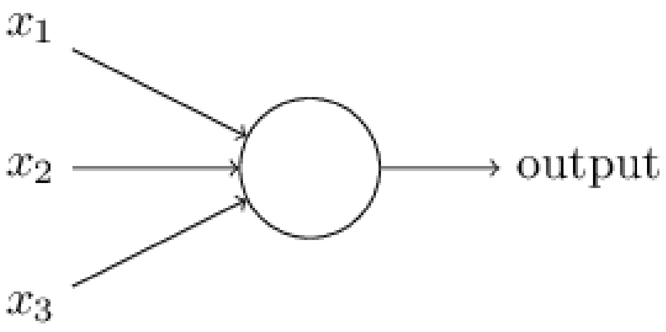


Figure 5 - Artificial Neuron [5]

In Figures 4 and 5, we see a natural neuron and an artificial neuron respectively. They share many similarities; they both have many inputs, but only a few outputs which continue to distribute information to the rest of the neurons in the network.

Every artificial neuron completes a specific order of tasks, which are:

- Takes the x_1, x_2, x_3 … x_n values as inputs, weighs them according to the corresponding w_1, w_2, w_3 … w_n weights, sums them up, and produces the final value as outcome.
- The above outcome is then passed through the activation (or transfer) function to produce the final output Y.
- A bias is added in case we want to control the values at which the neuron fires.

Having said the above, a more accurate representation of the artificial neuron looks like the one in Figure 6.

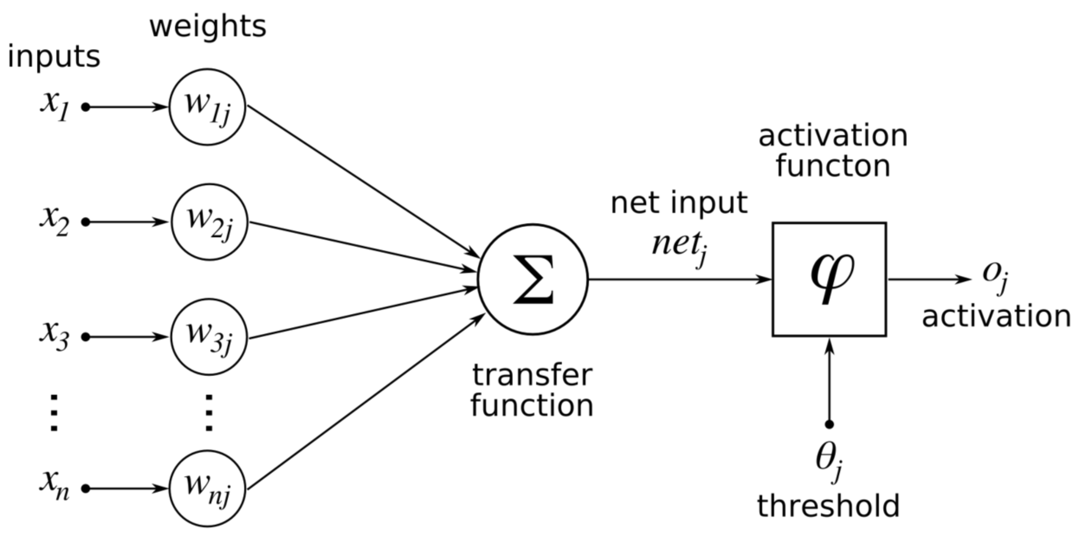


Figure 6 - More accurate artificial neuron [6]

More formally, the output computed by each neuron *j* can be described as a function:

$$o_j = \varphi\left(\sum_{k=1}^{n} w_{kj}x_k + \theta_j\right)$$

where φ is the activation function or transfer function. While there are many activation functions the most commonly used is the Sigmoid function which is graphed in Figure 7.

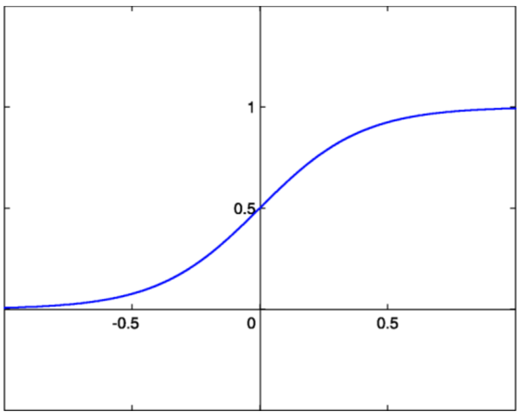


Figure 7 - Sigmoid function [6]

The Sigmoid function is mathematically computed as:

$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

A simple neural network consisting of several neurons is presented in Figure 8.

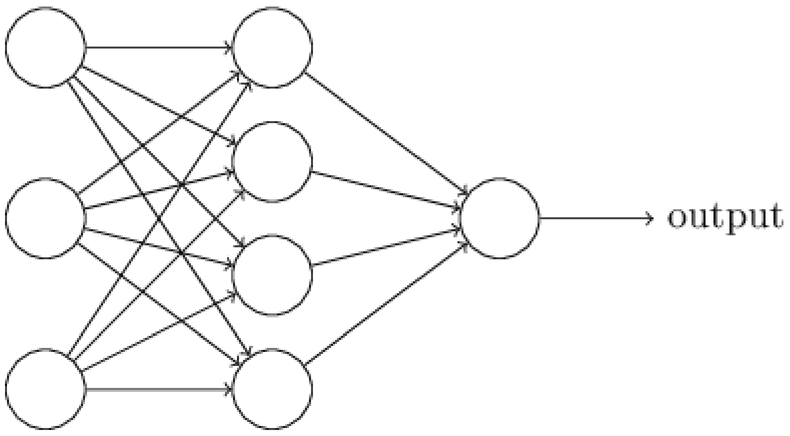


Figure 8 - Simple neural network [5]

Neural networks similar to the one in Figure 8 can learn to perform tasks. The main advantage is that they can be programmed without the need for a programmer to take action every time new data arrives; they can be trained or retrained using the new data, and therefore they can grow, evolve and learn on their own. Every time new data set is presented to a network, it learns from it; it adjusts and reconfigures the weights of every neuron and the outcome is predicted using all, past and present, experiences.

2.5 Deep Learning

Previously in this chapter we described what neural networks are and how they operate, but we have not mentioned anything about deep learning just yet. The reason for this is quite simple; deep learning is simply a very large neural network. It is a neural network that consists of many “hidden” layers and therefore its output is produced from a deep pipeline of input processing. Figure 9 shows an example of a relatively small deep network consisting of three hidden layers.

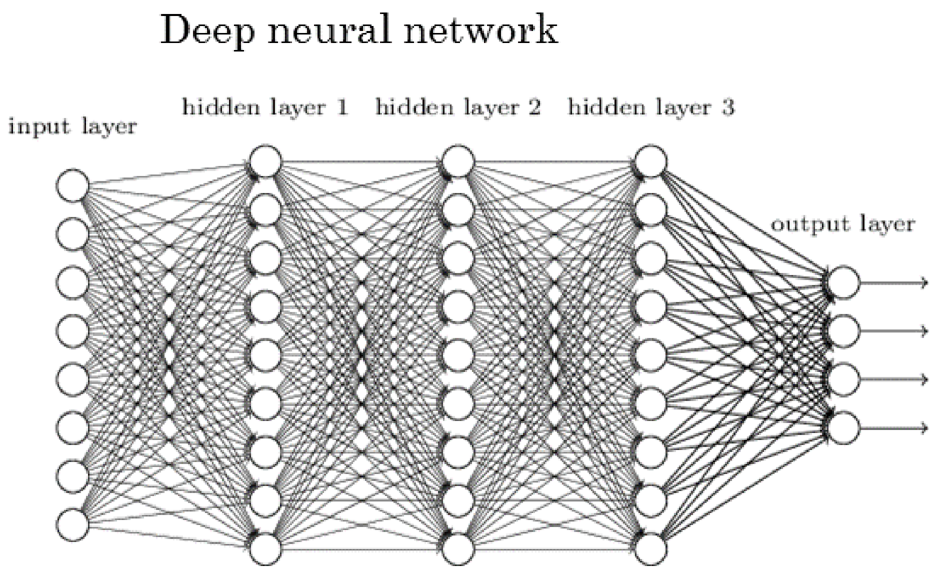


Figure 9 - Deep neural network [5]

Up until recently we were unable to manipulate such networks, especially the training part, but the difference nowadays is that we have developed powerful enough computers and possess large data sets, to be able to exploit their full potential.

One more important point about deep learning neural networks is their scalability. The larger the networks we construct and the more data we train them with, the more their performances increases dramatically. Generally this is different to other machine learning techniques that after a certain point they reach a ceiling in performance. In Figure 10 we see a diagram that illustrates this concept; this illustration was delivered by Andrew Ng, Chief Scientist at Baidu Research, who formally founded Google Brain, and eventually resulted in the popularization of deep learning technologies across a large number of Google services.

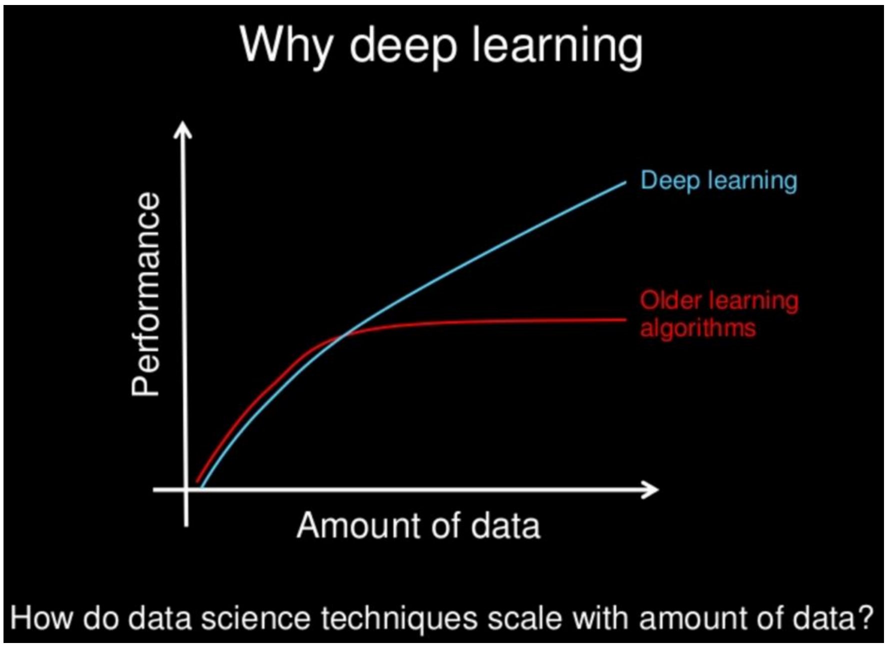


Figure 10 - Deep learning scale [7]

Another great benefit of deep learning models, besides scalability, is their ability to extract features from raw data all by themselves. This is called feature learning and is the process by which deep networks compose feature hierarchies, which enables them to build complex high-level features by combining simpler ones. This groundbreaking ability allows them to learn advanced functions that link the input to the output of any data, without depending on human intervention.

If we desire to design a network graph, depicting how the output concepts are built on top of each other, the graph is going to be deep and with many layers. For this reason, we call this approach deep learning. In Figures 11, 12 and 13 we can see a visualization of how this concept of feature extraction and synthesis works.

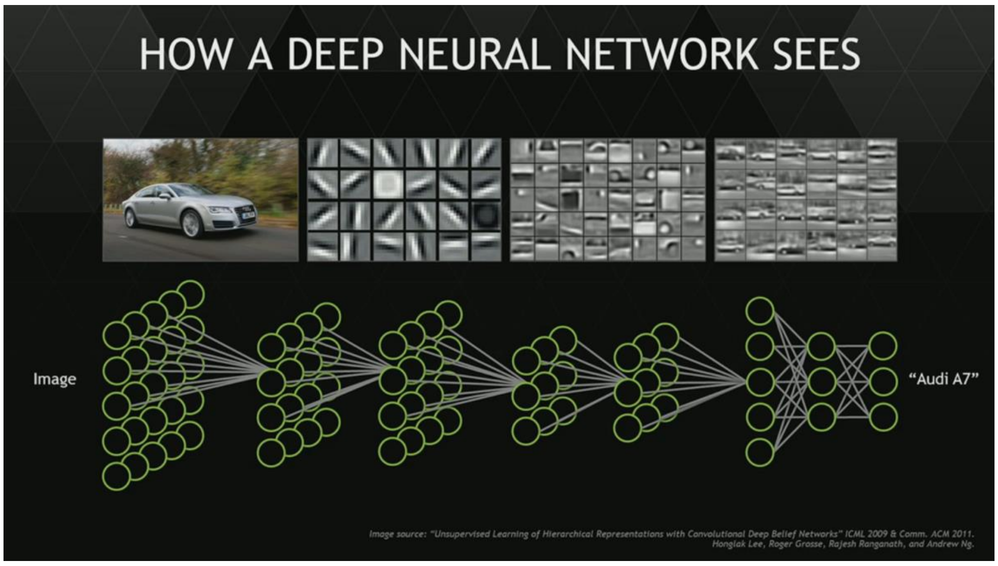

HOW A DEEP NEURAL NETWORK SEES
Image
"Audi A7"
Image source: "Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks" ICML 2009 & Comm. ACM 2011.
Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Ng.

Figure 11 - Feature extraction for car detection in images [8]

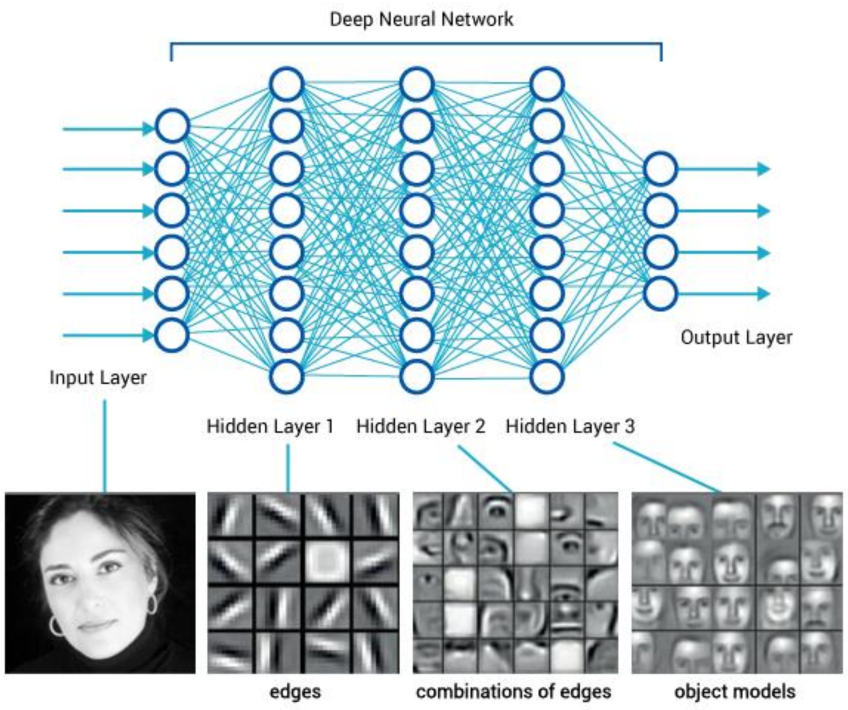


Figure 12 - Feature extraction for face detection in images [9]

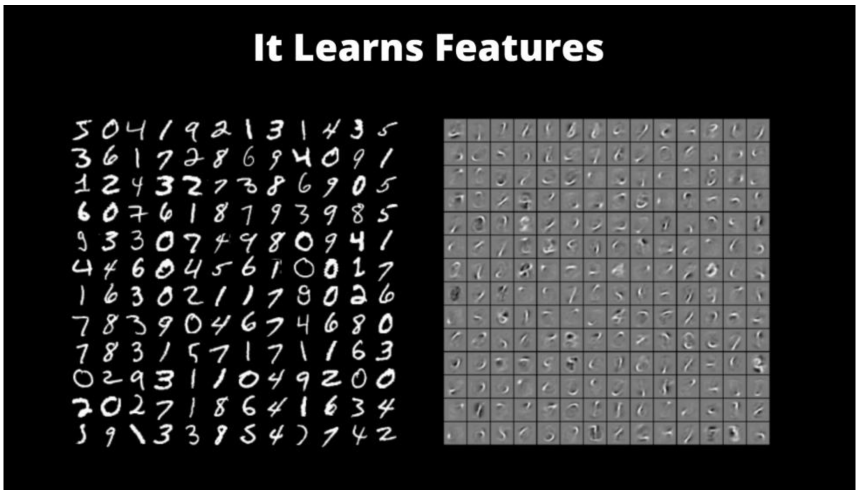


Figure 13 - Feature extraction for digit detection in images [10]

Taking in consideration all the above characteristics, deep learning techniques seem to be a perfect fit for solving problems in machine vision, which is the topic of our thesis.

2.6 Transfer Learning

"I think AI is akin to building a rocket ship. You need a huge engine and a lot of fuel. If you have a large engine and a tiny amount of fuel, you won't make it to orbit. If you have a tiny engine and a ton of fuel, you can't even lift off. To build a rocket you need a huge engine and a lot of fuel.

The analogy to deep learning is that the rocket engine is the deep learning models and the fuel is the huge amounts of data we can feed to these algorithms."— Andrew Ng [7]

The most common type of deep neural networks used nowadays for object recognition, are the Convolutional Neural Networks. In machine learning, a Convolutional neural network (CNN or ConvNet) is a class of deep, feed-forward artificial neural networks. They are designed to require minimal preprocessing and by preprocessing we mean data cleaning, instance selection, normalization, transformation, feature extraction and selection, etc.

As we already mentioned in the introduction, very few people train a large Convolutional Network from scratch. Assume you are faced with a research problem, such as identifying cancerous moles in photographs. Furthermore, let's say you managed to gather a dataset of 10,000 labeled images. While at first this might seem a lot, it is next to nothing in comparison with the enormous datasets on which deep learning achieves its biggest success. In addition, you will probably need hundreds of GBs of RAM and hundreds of GBs of VRAM on GPUs to attack a complex supervised machine learning problem.

That means that we have to find resourceful ways to solve Deep Learning problems. That's especially true, when we deal with complex real life problems in areas, such as image and voice recognition. Once you already have some number of hidden layers in your model, adding another one would need vast resources to retrain the network. For these reasons, with the exception of some very large companies like Google, Facebook, IBM, and Microsoft, it is very difficult to accrue the data and the computational machines required for training strong deep learning models.

In such a situation, transfer learning comes to rescue. Before we continue, we must form a sense of what transfer learning is. This becomes easier to comprehend by comparing it to a teacher – student analogy. A teacher has a lot of experience from all the years he/she is related to the particular topic. All this knowledge and information that is acquired is then lectured to the students, as a brief and comprehensive overview of that topic. And so it can be seen as a "transfer" of valuable information from the instructor to the trainee (see Figure 14).

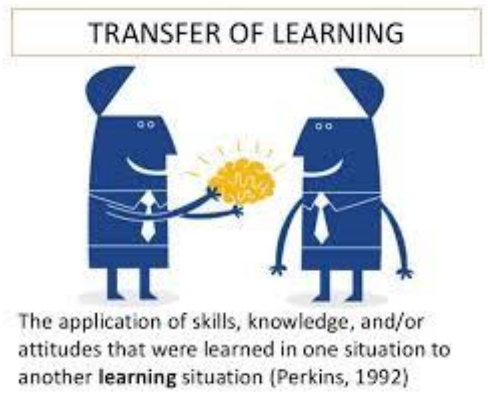

TRANSFER OF LEARNING
The application of skills, knowledge, and/or attitudes that were learned in one situation to another **learning** situation (Perkins, 1992)

Figure 12 - Transfer learning illustration [11]

With this analogy in mind, we compare this to neural network which is trained on some data. That particular network gains knowledge from this data, which is translated to "weights" of the network. These weights can then be extracted and transferred to any other neural network. Instead of training a new network from scratch, we can "transfer" the already learned features and focus on learning any additional features that may be required.

This technique helps tremendously considering that there exists many pre-trained models, trained on large amounts of data, which have been made available publically, along with the values of billions of parameters. One can exploit pre-trained models and rely on transfer learning to build models for a new specific case. We have to keep in mind though that a pre-trained model may not be 100% accurate in a new application, but it saves a lot of effort.

Finally, let's have a look at Figures 15 and 16 below in order to visualize how this procedure works.

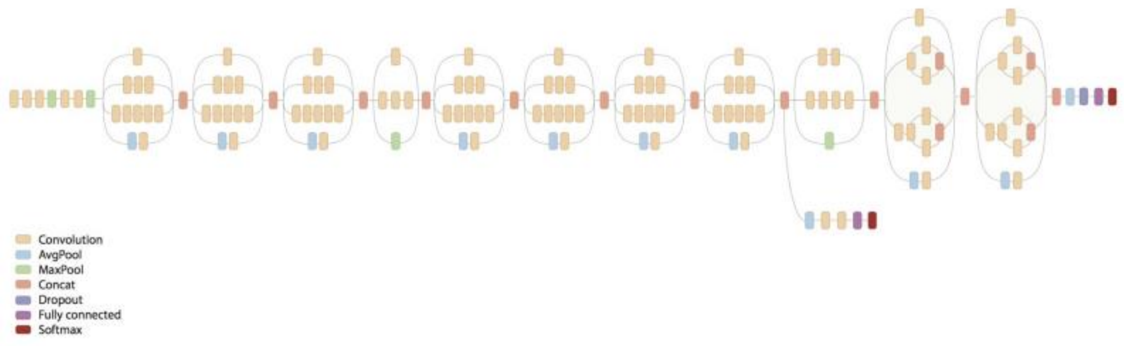


Figure 13 - Inception V3 CNN i [34]

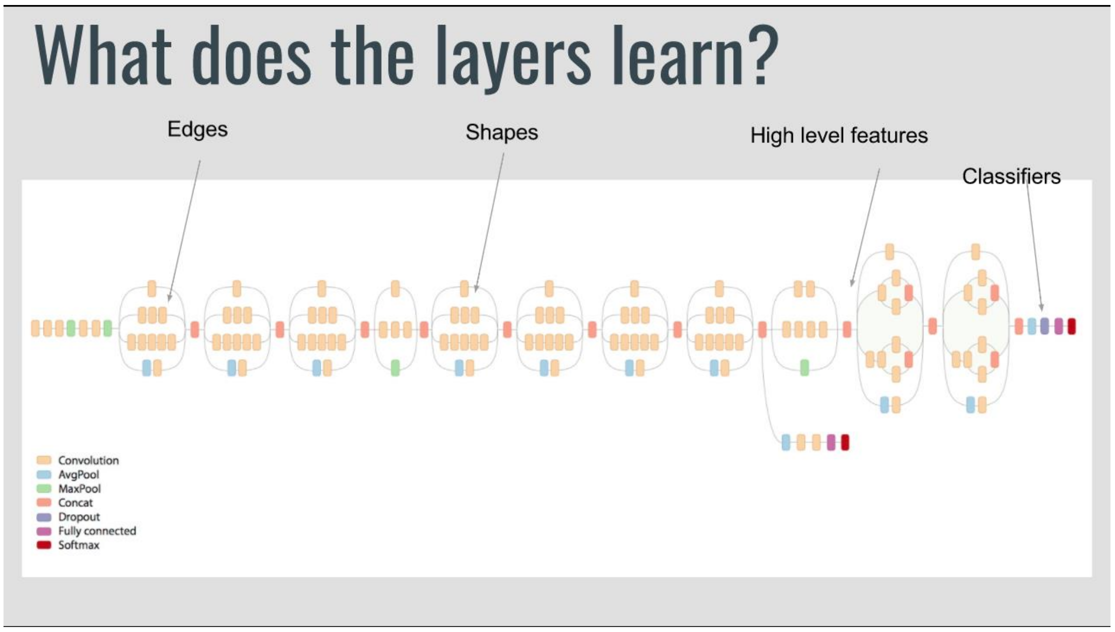
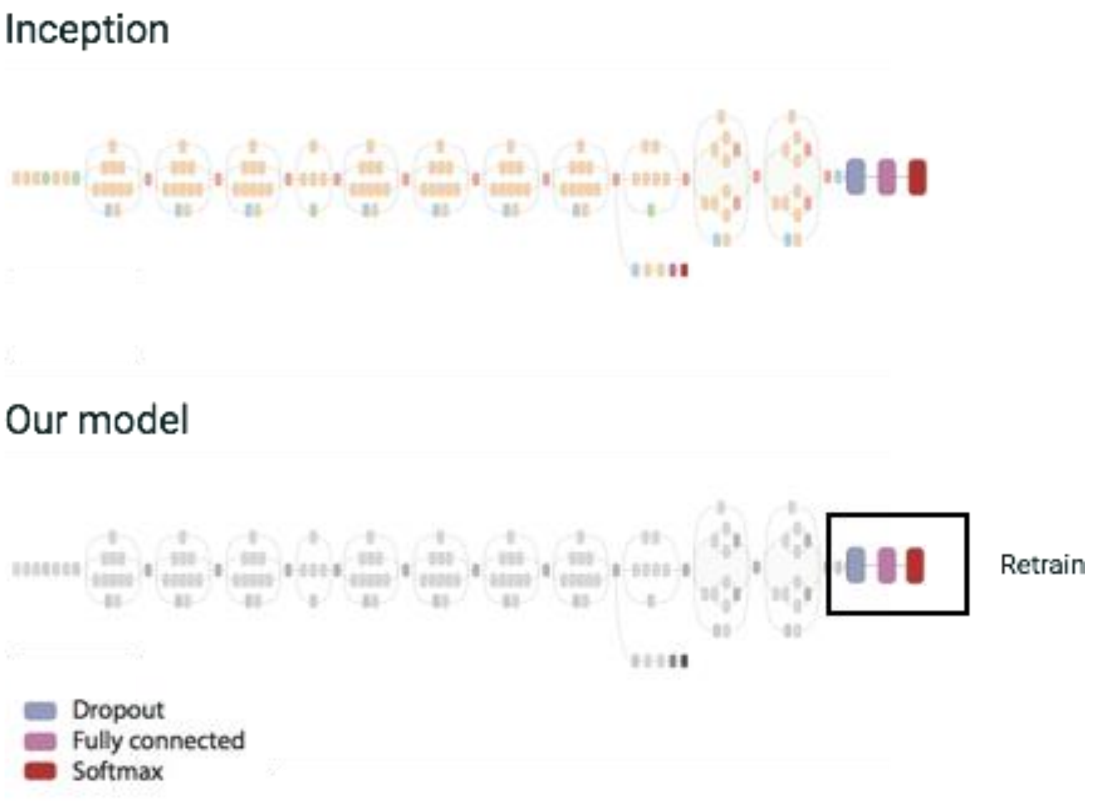


Figure 14 - Inception V3 CNN ii [12]

In order to apply the transfer learning technique, we must take this model and re-train only the last part of it that produces the output on our dataset as we can see in Figure 17.



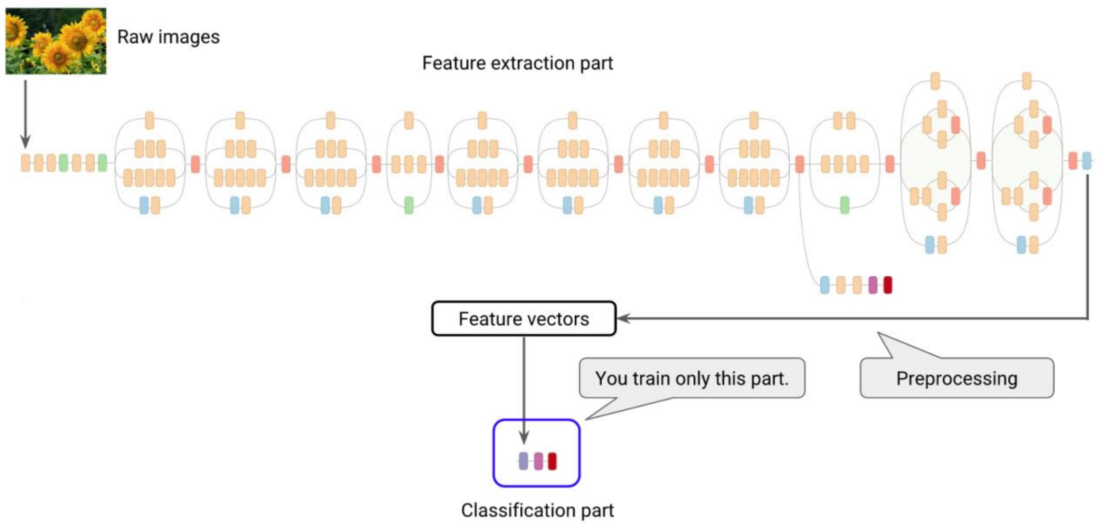


Figure 15 - Retrain last layers [13]

Finally, Figure 18 shows more diagram, again by Andrew Ng, illustrating his opinion concerning transfer learning.

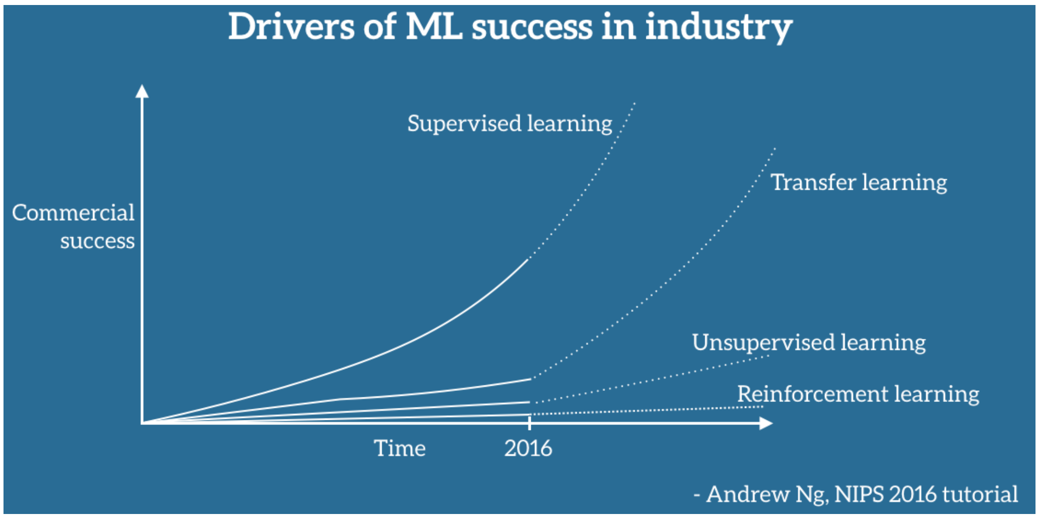


Figure 16 - Transfer learning prediction chart [14]

2.7 OpenStreetMap

OpenStreetMap (OSM) is a collaborative project to create a free editable map of the world. The creation and growth of OSM has been motivated by restrictions on use or availability of map information across much of the world, and the advent of inexpensive portable satellite navigation devices. OSM is considered a prominent example of volunteered geographic information [15].

In their official site (www.openstreetmap.org), one can see a classic interactive map that displays the classic map tiles rendered by the OpenStreetMap data as an example of its use and what one is able to do with it. The real power though comes from the possibility to access every single data behind this simple and default rendering. And that is the main difference behind the other famous commercial competitors, like Google and Yahoo maps. OpenStreetMaps' purpose is to offer a thorough database of every street, city, road etc. on the planet and not just display map tiles.

2.8 Tensorflow

Tensorflow is an open source software library for numerical computation using data flow graphs. Tensorflow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well [16].

Chapter 3 – Problem Statement

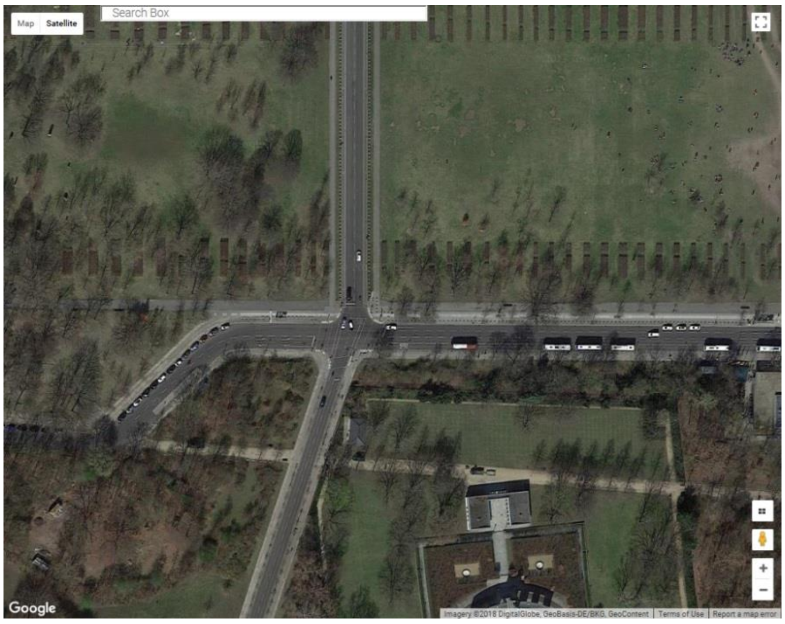
The goal of this thesis is to create a fast and reliable object detection system, using state-of-the-art technology. Undeniably, in the last few years Deep Learning Networks achieve the greatest results in machine vision, so we knew from the beginning we wanted to experiment with them and try to get a glimpse of their true power.

The obstacle we will have to overcome in order to train a Deep Neural Network is their complex training process. These large networks require enormous computational power and vastly large datasets in order to achieve their potential, which is probably off the limits for most candidates who wish to train one from scratch. To overcome this problem, we will use a new technique called Transfer Learning, where we take an already trained network, extract the features it has already learnt and train its last layers with our own new data.

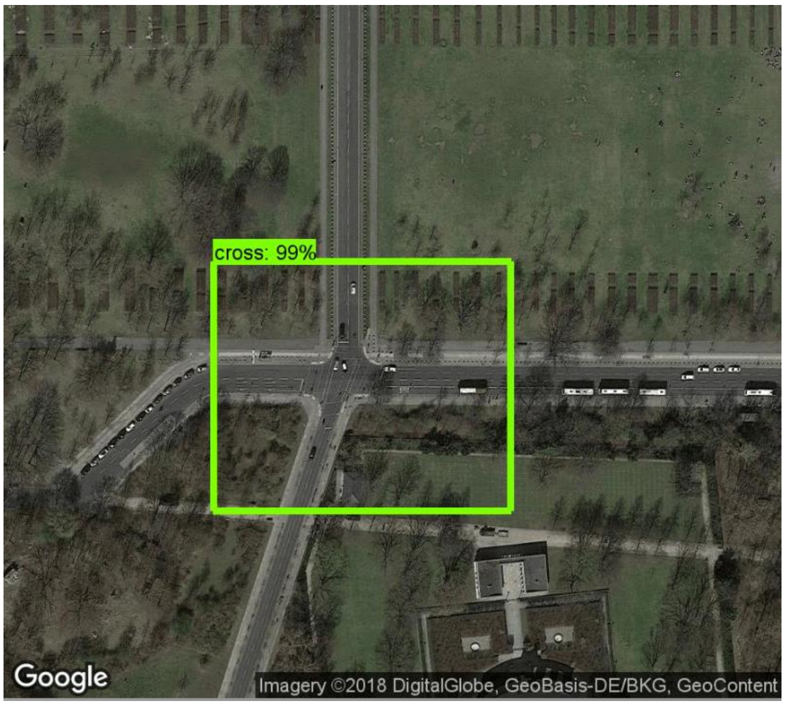
Another field we feel really excited about is the field of satellite imagery analysis. It is a complicated machine vision problem and if resolved correctly it can provide solutions and a great number of information related to our everyday world. For example, as we mentioned before in Chapter 1, it can assist in search and rescue missions, urban planning, crop and forest management, weather prediction, disaster relief, climate modeling, etc.

Finally, we want our network to operate on a web platform, in order to make it easy and simple for anyone interested in our work to examine our results, even though they may lack any previous knowledge of this topic. To this end, we will make use of an interactive Google map, on which the potential user will be able to choose the area for detection, press a button and the result will be displayed back when available.

In short we would like our system to receive a satellite image like the one in the figure below, analyze it, and produce as outcome the picture with the location of the detected crossroad (if any) marked on it.


Map
Satellite
Search Box
Google
Imagery ©2018 DigitalGlobe, GeoBasis-DE/BKG, GeoContent | Terms of Use | Report a map error

Ideally, the result of our application should look as in the figure below.



3.1 Related Work

One of the most innovative and inspiring works in satellite image analysis is the Ph.D. Thesis of Volodymyr Mnih [17] under the supervision of Prof. Geoffrey E. Hinton. Many other great projects referenced this paper in the years that followed. In his paper, Volodymyr presented a novel approach to automatically detect and label roads in aerial imagery using deep neural networks.

During our literature review, we found a great e-book about neural networks and deep learning from Michael Nielsen [5]. Although the book is an introduction to the field and we did not really use any of the techniques inside, it is a great read for beginners and helps tremendously to get a basic understanding of the field.

A great Github repository that also helped a lot during our background research is the one maintained by Andrew L. Johnson [18]. His work is similar to that of Volodymyr Mnih and we found his bibliography very helpful.

A very interesting web application we found online, is the one of finding features of interest in cities, and is called Terrapattern [19]. Their work is on discovering “patterns of interest” in unlabeled satellite imagery within a selected set of cities, as they mention on their website. In their web application, the user can click on an interesting spot on the map and the application will find other locations that look similar to it within the same city.

Another interesting application of satellite image analysis is the work of Adam Van Etten [20]. In his work he detects boats in satellite images. Although he does not use neural networks for his classifier, we found his work very interesting, since he has some great ideas about object detection and we found his implementation well-documented, well-explained and well-constructed.

Chapter 4 – First Approach

Before we talk about our final solution, it is worth discussing the technique we applied and experimented with in the early stages of our implementation.

We started solving this problem by retraining a new image classifier. To do so, we decided to retrain the final layers of Inception V3, an already trained model provided by Google via Tensorflow [21], a CNN that was trained for 2-3 weeks on multiple GPUs on ImageNet, provided by Google [22]. The ImageNet project is a large visual database designed for use in visual object recognition software research. As of 2016, over ten million URLs of images have been hand-annotated by ImageNet to indicate what objects are pictured; in at least one million of the images, bounding boxes around the objects of interest are also provided. The training was accomplished with the help of Tensorflow which was described in Chapter 2.

4.1 Training our First Classifier

We split our dataset into two major categories: "yes" and "no" representing whether a satellite image contains a crossroad in it or not. In order to retrain the network we download the pre-trained model, add a new final layer, and train that layer on the crossroad photos we gathered. We trained for the maximum default number of iterations, which is 4,000.

The first phase analyzes all the images and calculates the bottleneck values for each of them.

Inception V3 [23] is made up of many stacked layers on top of each other as we can see in Figure 19. These layers are already trained and are very useful at finding and summarizing features and information that already help us classify many images. In our project we train explicitly the last layer (`final_training_ops` as shown in Figure 20) while all other layers maintain the same trained state.

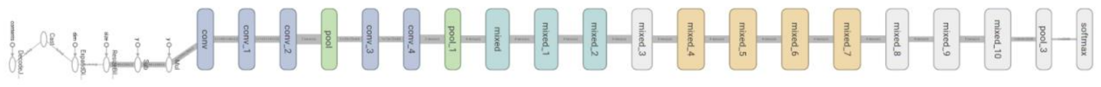


Figure 17 - Inception V3 layers

In the Figure 20 the node labeled "softmax", on the left side, is the output layer of the original model. While all the nodes to the right of the "softmax" were added by the retraining script.

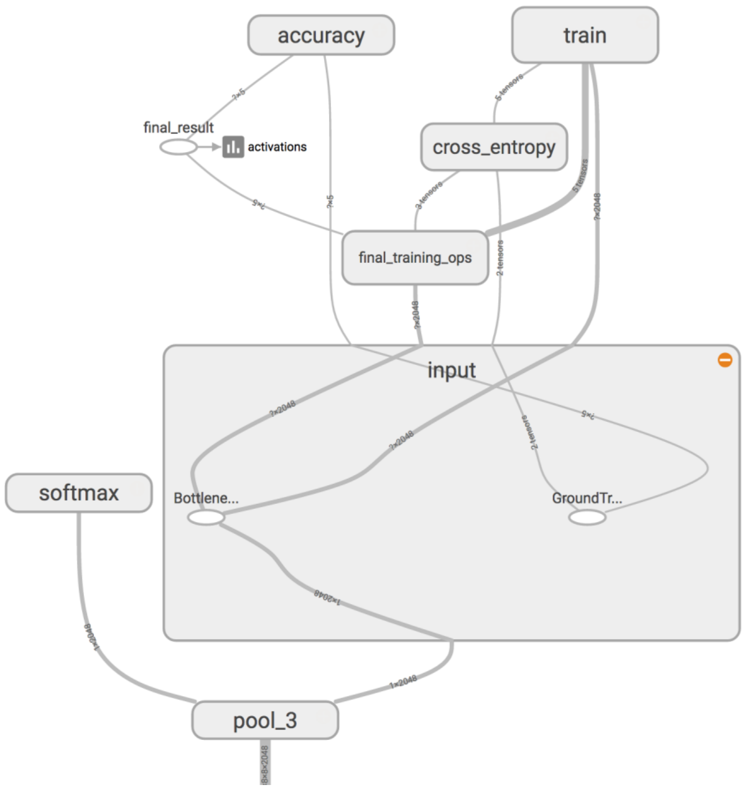


Figure 18 - Retraining nodes

A bottleneck is an informal term that is often used for the layer just before the final output layer that actually does the classification. "Bottleneck" is not used to imply that the layer is slowing down the network. We use the term bottleneck because, near the output, the representation is much more compact compared to that in the main body of the network.

All images are reused multiple times during training. Calculating the layers behind the bottleneck for each image takes a significant amount of time. Since these lower layers of the network are not being modified, their outputs can be cached and reused. So, the script runs the constant part of the network, everything below the node labeled Bottleneck in the figure above, caching the results. Once the script finishes generating all the bottleneck files, the actual training of the final layer of the network begins.

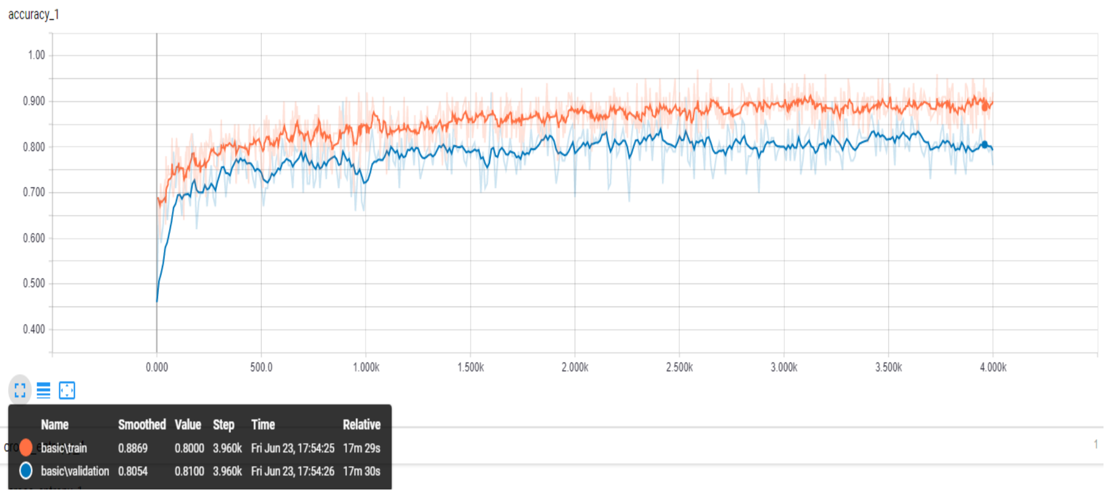
The training operates efficiently by feeding the cached value for each image into the Bottleneck layer. The true label for each image is also fed into the node labeled

GroundTruth. Just these two pieces of data are enough to calculate the classification probabilities, training updates, and the various performance metrics.

As training progresses, we can see a series of step outputs, each one showing training accuracy, validation accuracy, and the cross entropy:

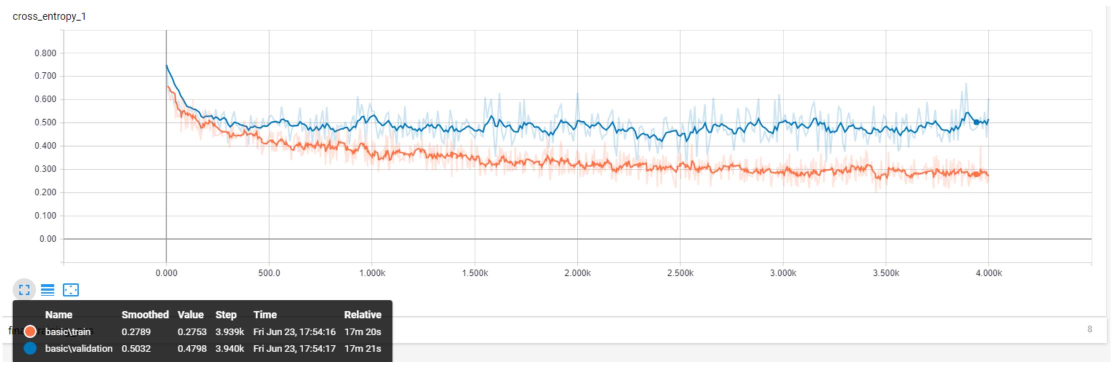
- The **training accuracy** shows the percentage of the images used in the current training batch that were labeled with the correct class.
- **Validation accuracy**: The validation accuracy is the precision (percentage of correctly-labeled images) on a randomly-selected group of images from a different data set that has never been seen by the network.
- **Cross entropy** is a loss function that gives a glimpse into how well the learning process is progressing (the lower the numbers, the better in this case).

It is possible to monitor the progress of the model's accuracy and cross entropy as it is being trained. If the model has finished generating the bottleneck files, one can check the model's progress by opening the TensorBoard [24]. TensorBoard is a browser based application that helps visualize the training parameters (such as weights and biases), metrics (such as loss), hyper parameters or any statistics. Figures 21 and 22 are important figures as they capture the progress of our training process.



	Name	Smoothed	Value	Step	Time	Relative
●	basic\train	0.8869	0.8000	3.960k	Fri Jun 23, 17:54:25	17m 29s
●	basic\validation	0.8054	0.8100	3.960k	Fri Jun 23, 17:54:26	17m 30s

Figure 19 - Training accuracy



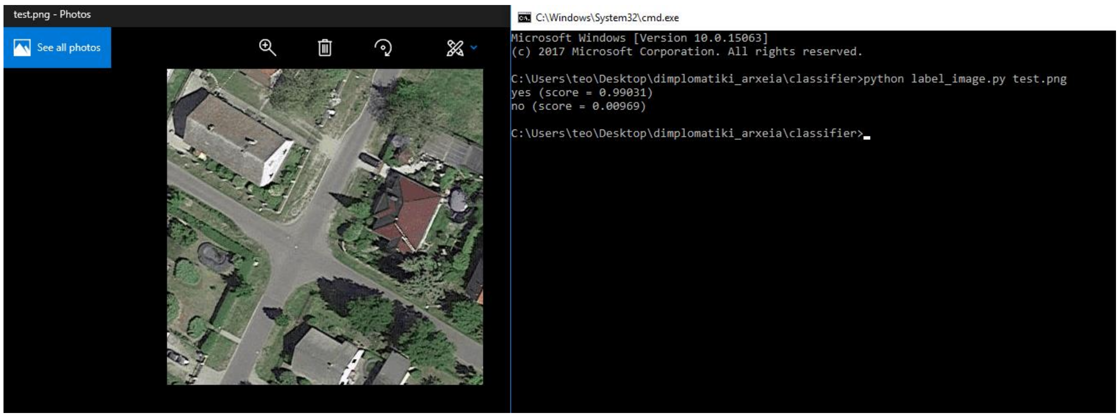
	Name	Smoothed	Value	Step	Time	Relative
●	basic\train	0.2789	0.2753	3.939k	Fri Jun 23, 17:54:16	17m 20s
●	basic\validation	0.5032	0.4798	3.940k	Fri Jun 23, 17:54:17	17m 21s

Figure 20 - Training cross entropy

A true measure of the performance of the network is to measure its performance on a data set which is not in the training data. This performance is measured using the validation accuracy. If the training accuracy is high, but the validation accuracy remains low, it means that the network is overfitting; in particular, the network is memorizing very specific features in the training images that do not help it classify images more generally. The training's objective is to make the cross entropy as small as possible, so you can tell if the learning is working by keeping an eye on whether the loss keeps trending downwards, ignoring the short-term noise.

As we mentioned earlier, by default this script runs for 4,000 training steps. Each step chooses 10 images at random from the training set, finds their bottlenecks from the cache, and feeds them into the final layer to get predictions. Those predictions are then compared against the actual labels to update the final layer's weights through a backpropagation process.

Now that we have our classifier we can use it on new images for classification using the **`label_image.py`** script, passing as argument an image file. In Figure 23 we can see an example of our classifier applied to a never-seen-before image.



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\teo\Desktop\dimplomatiki_arxeia\classifier>python label_image.py test.png
yes (score = 0.99031)
no (score = 0.00969)

C:\Users\teo\Desktop\dimplomatiki_arxeia\classifier>
```

Figure 21 - Classifier example

Having obtained a classifier with satisfying results, we now move on to build our object detection unit. The technique we decided to use was to apply a sliding window in combination with an image pyramid.

4.2 Image Pyramid

An "image pyramid" is a representation of an image on many scales. Sometimes the object we wish to detect surpasses the boundaries of our classifier's window of pixels. This technique helps us find objects of different sizes in a given image.

In the bottom layer of the pyramid we have the original image at its original size, in terms of width and height. In every higher layer, the image is resized and usually smoothed via Gaussian blurring. The image is consecutively resized as many times as we want; this normally terminates, when a minimum size has been reached and no further subsampling needs to take place.

In Figure 24 we can see an example of image pyramid and in Figure 25 we can see this technique applied to one of our own images.

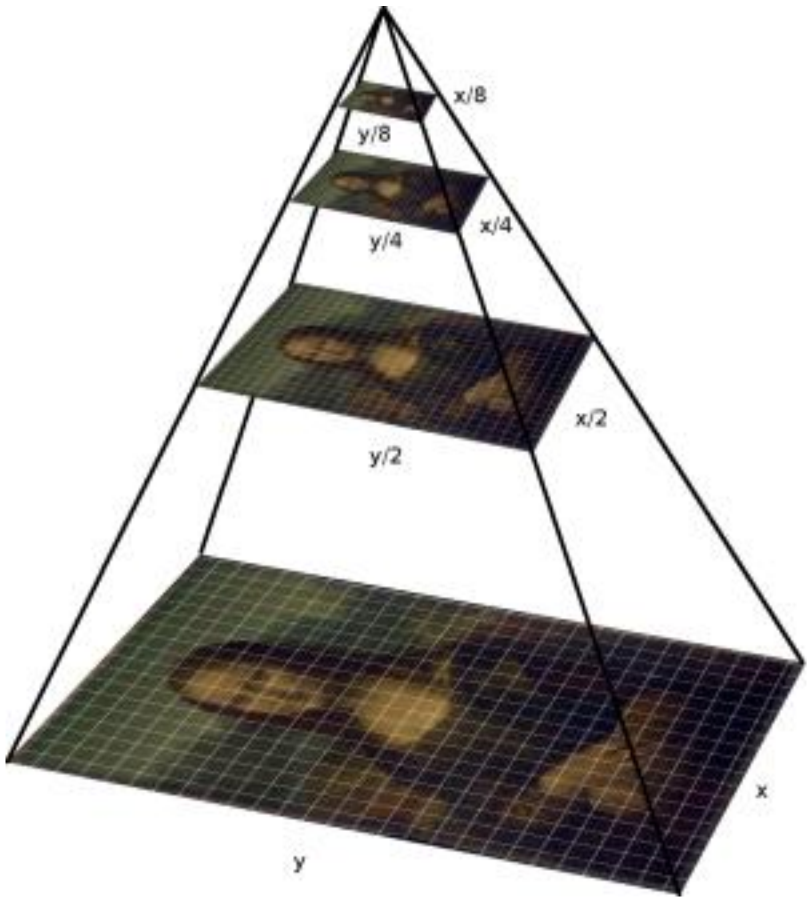


Figure 22 - Pyramid example [25]

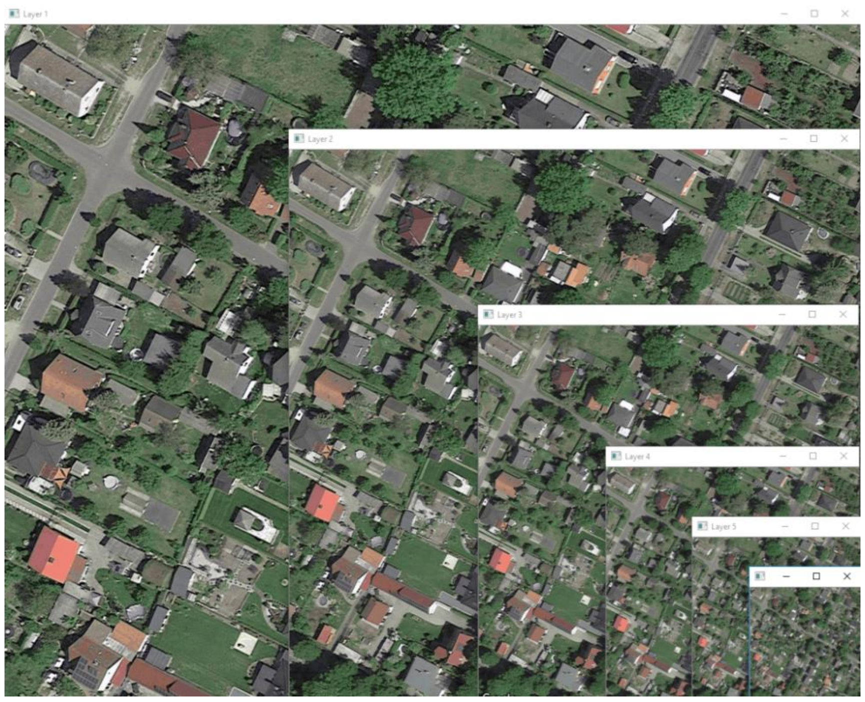
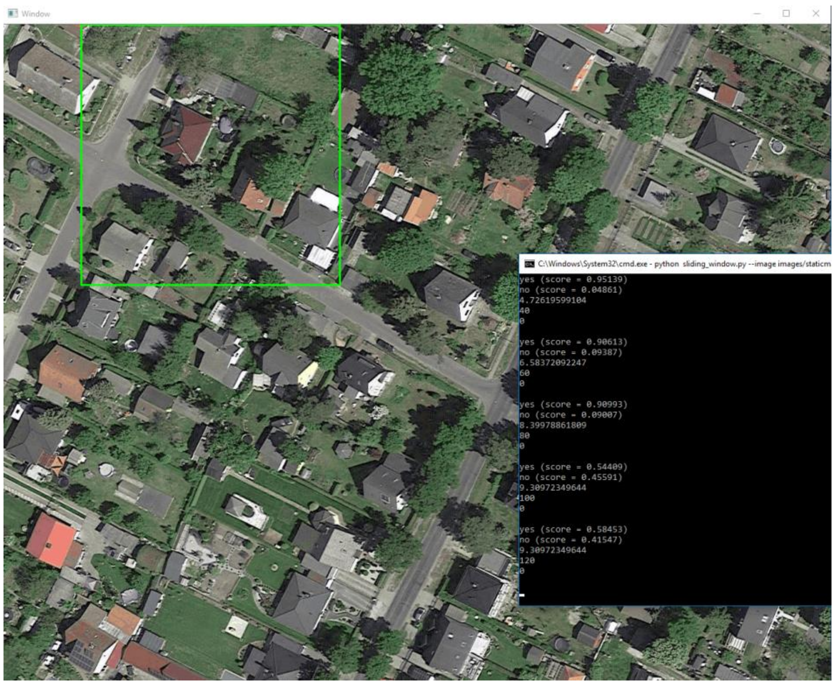


Figure 23 - Pyramid on our satellite image

4.3 Sliding Window

As the name indicates, a sliding window is a box region of fixed width and height that "slides" horizontally and vertically on an image, in a fixed step of pixels. Every time this window is applied on a part of the image, we apply on that area our image classifier to determine whether the window contains a crossroad. We repeat this process for every scale of our image.

In Figure 26 we see this technique applied to one of our images and with our own classifier.


Window
C:\Windows\System32\cmd.exe - python sliding_window.py --image images/staticm

```
yes (score = 0.95139)
no (score = 0.04861)
4.72619599104
40
0

yes (score = 0.90613)
no (score = 0.09387)
6.58372092247
60
0

yes (score = 0.90993)
no (score = 0.09007)
8.39978861809
80
0

yes (score = 0.54409)
no (score = 0.45591)
9.30972349644
100
0

yes (score = 0.58453)
no (score = 0.41547)
9.30972349644
120
0
```

Figure 24 - Sliding window

4.4 Preliminary Results

We used images of 1200x1200 pixels and a window of 400x400 pixels and a step of 100 pixels for experimentation, but these numbers are easy to change. Every time our classifier finds a crossroad in the given window it stores the confidence value into the Results Array, given the value is greater than a threshold we established. The Results Array is a 12x12 array, each pair of values representing a box of 100x100 pixels in our actual image.

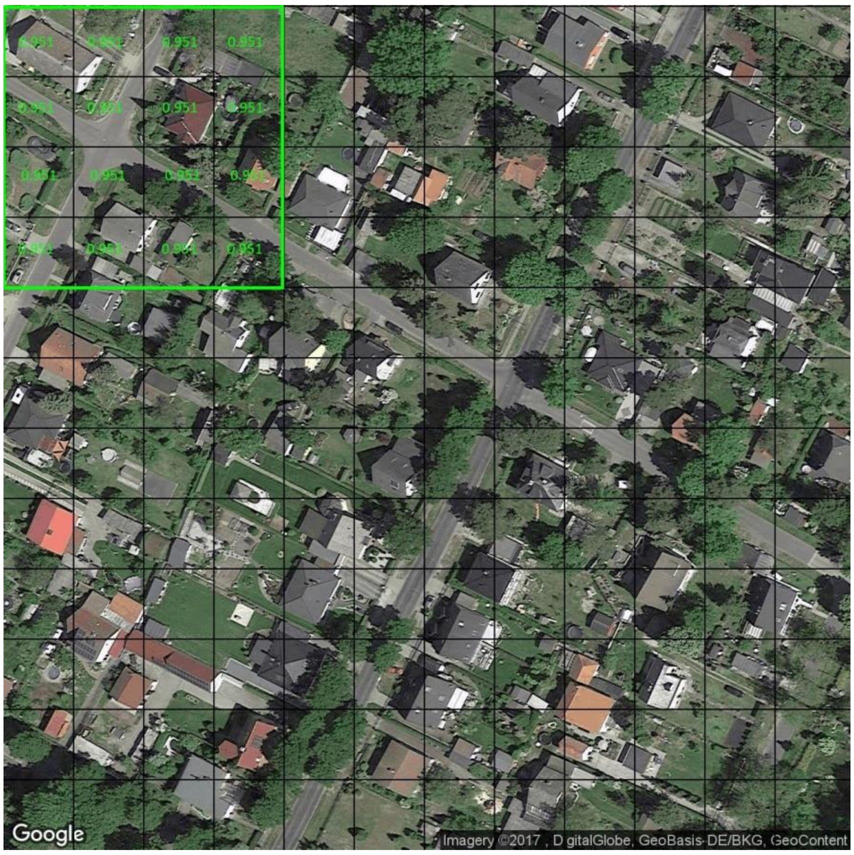
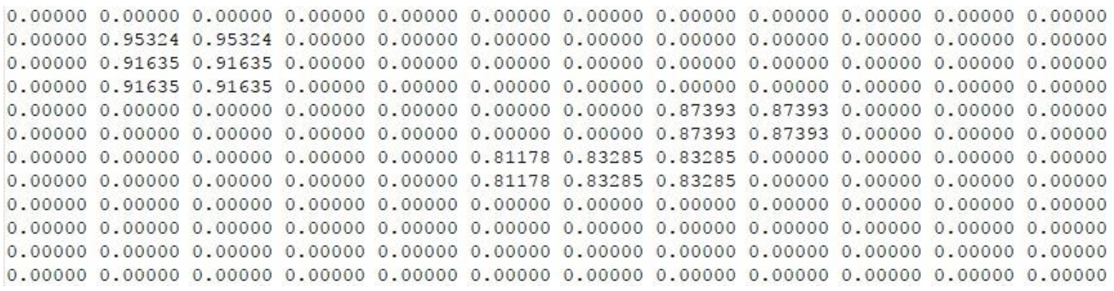


Figure 25 - Results visualization

When the whole operation is done, we take the sum of each box, divide it by the times we added a value to it and we can find which of the boxes are part of a crossroad. In Figure 28, we can see how our final array looks like.



```
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.95324 0.95324 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.91635 0.91635 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.91635 0.91635 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.87393 0.87393 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.87393 0.87393 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.81178 0.83285 0.83285 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.81178 0.83285 0.83285 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

Figure 26 - Results array

These numbers suggest which blocks of pixels contain a crossroad and the confidence value of it. Figure 29 show how these values translate to areas within our image.

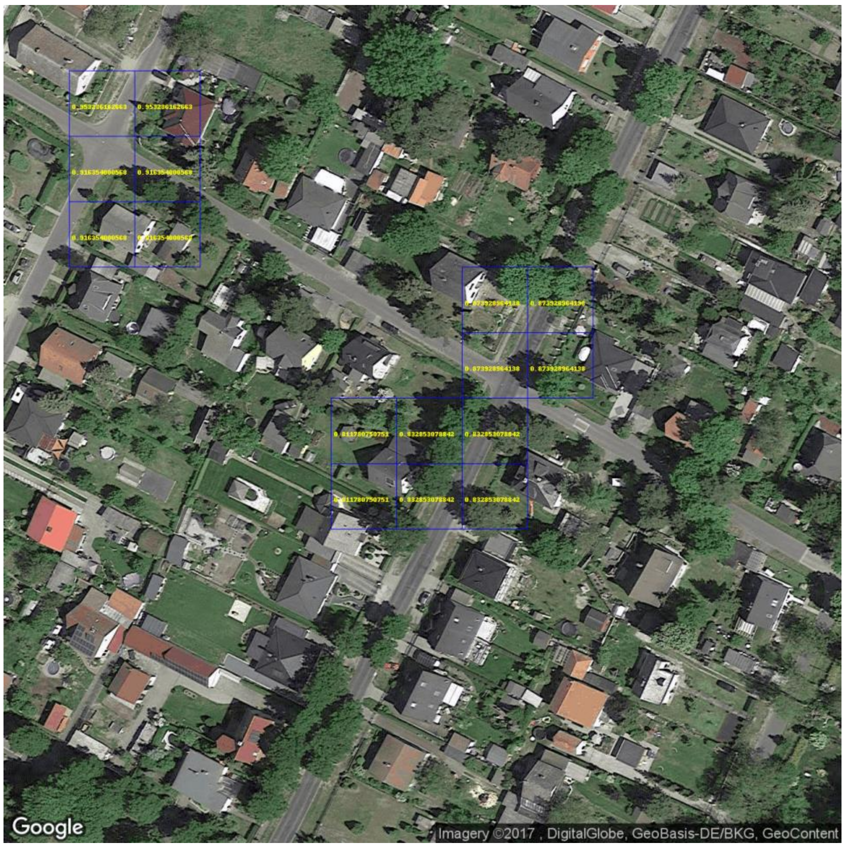


Figure 27 - Results on Image examples

As we can see it does a decent job finding the centers of the crossroads; after this step, we could draw a rectangle around these center values. These results can get a little better or a little worse with a few modifications, but this technique does not guarantee a global and reliable enough solution. So we had to find a way to improve our detection.

4.5 Tensorflow Object Detection API

We conducted a broad research and read many papers on techniques and algorithms one can adopt to improve his results, but most of them were either somewhat hard to implement or too target-specific that may not work well in our case. One interesting example we found is a paper [20], which used background removal and then applied the image classifier to find boats on satellite images with great results. However, that idea can be a much more complicated task to perform on images that depict cities, since the background does not consist of only one element and does not have the same color consistency (the sea in the particular example of that paper).

So, while investigating ways to further improve our own object detection results, we found a tool that proved to be a catalyst in our project. Google recently released a new open source framework that makes it easier for developers and researchers to identify objects within images and it is called “Tensorflow Object Detection API” [26]. This is an open source framework built on top of Tensorflow that makes it easy to construct, train and deploy object detection models.

4.6 COCO dataset

It is important to note that the models in Tensorflow Object Detection were trained on COCO (Common Objects in Context Dataset), which is a new image recognition, segmentation, and captioning dataset. COCO has several features:

- Object segmentation
- Recognition in Context
- Multiple objects per image
- More than 300,000 images
- More than 2 Million instances
- 80 object categories
- 5 captions per image
- Keypoints on 100,000 people

Figure 30 shows two examples from COCO.

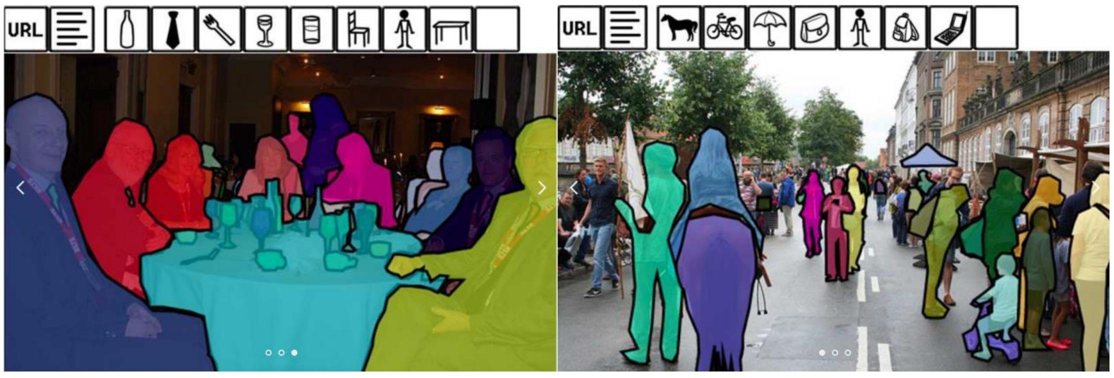


Figure 28 - COCO dataset

Having all the above in mind, we move on to the next chapter where we explain our steps and general process with greater detail [27].

Chapter 5 - Our Approach

The approach we propose in this thesis is summarized in Figure 31. The first stage illustrates the process of creating our dataset. The second stage shows the steps we made to prepare our data for training. Stage 3 is about the retrain process of our network and finally in the fourth and last stage we can see the way our web application works.

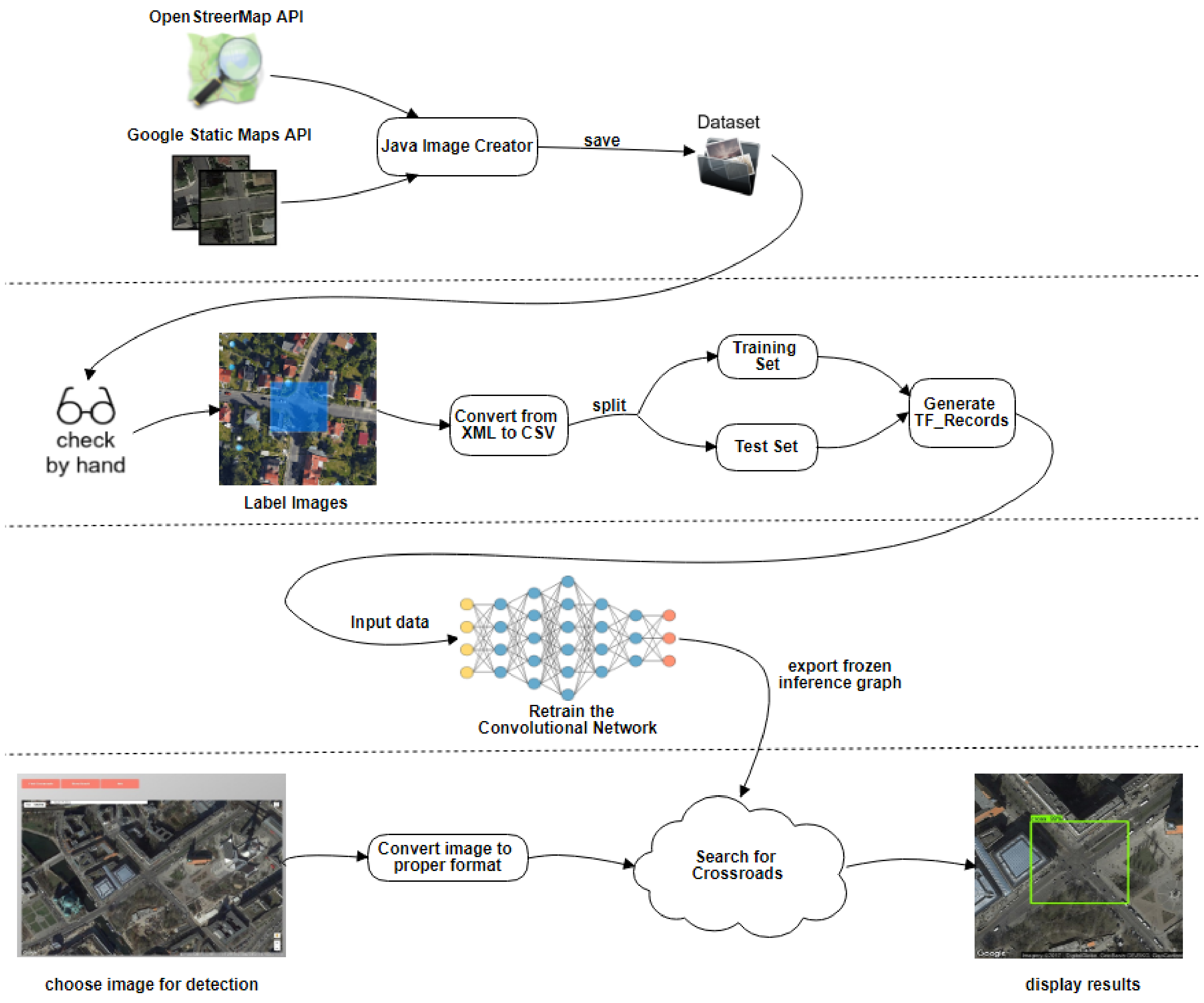


Figure 29 - Project Layout

5.1 Creation of Dataset

One of the hardest problems to solve in deep learning has nothing to do with neural nets: it’s the problem of getting the right data in the right format and in sufficient volume.

Deep learning and machine learning more generally, needs a good training set to work properly. Collecting and constructing the training set – a sizable body of known data – takes time and domain-specific knowledge of where and how to gather relevant information. The training set acts as the benchmark against which deep-learning nets are trained. That is what they learn to reconstruct before they’re unleashed on data they haven’t seen before.

Specifically in our case, we were supposed to find a large volume of satellite images, at least several hundreds of them, and, in addition, these images must contain the object we want our neural net to be trained on, namely crossroads.

Since such database does not already exist as far as we know, we must create one. For this reason, we implemented the “dataset creator” application. This is a java application that automates the procedure of constructing the desired dataset. This is what the application performs in summary:

- Using Overpass API (the API of OpenStreetMaps) we search for roads that cross each other and as a result, we get a list of geographical nodes (longitude , latitude) that meet such criteria.
- For every node in our list, we find the corresponding satellite image on Google Maps and using Google’s Static Maps API we download each image with each selected node as center.
- These two steps are repeated; it runs the two APIs consecutively and in that order: get the image, process it, and save it in the hard disk.

After we have collected a satisfying number of data, around 2000, we must check them by hand to see which are fit for our dataset and which are not. The reason for doing so is that we could not find a fully automated way to download images that contain exclusively crossroads, which means we also downloaded simple road junctions or in some cases even noise data. Figures 32 and 33 show positive and negative examples respectively.

Crossroads:

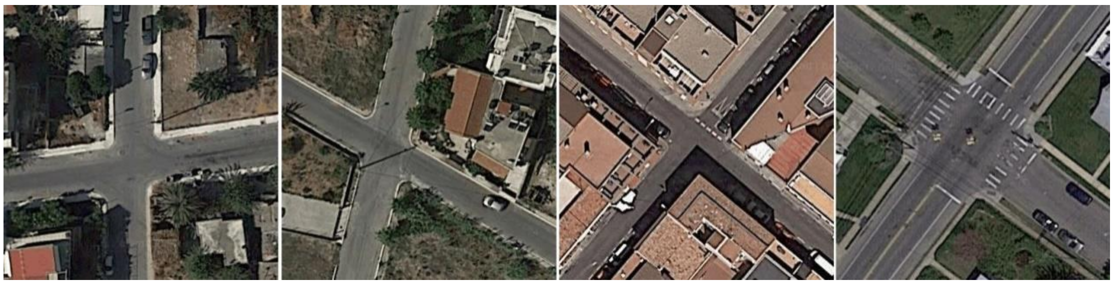


Figure 30 - Positive examples dataset

Not Crossroads:

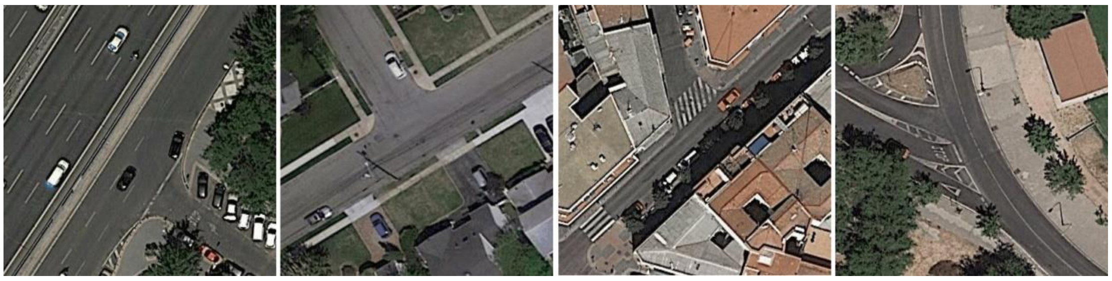


Figure 31 - Negative examples dataset

5.1.1 Overpass API

In a previous chapter, we explained the reason we decided to use OpenStreetMaps for our project. To access the OpenStreetMaps information one must use their official API, which is named Overpass API. This is a read-only API that serves up custom selected parts of the OpenStreetMaps map data. It acts as a database over the web: the client sends a query to the API and gets back the data set that corresponds to the query.

Overpass API has a powerful query language with many features available for use. This language is called "Overpass XML" and is an XML-based language, which shares a lot of similarities, but also a few differences with XML. There is a whole manual for it on Wikipedia [28].

The query we created runs for a specific bounding box over the map. The logic behind the code is pretty simple:

- We assign in two node variables every road (motorway, residential road etc.) contained in the specific bounding box
- Check where these two nodes intersect

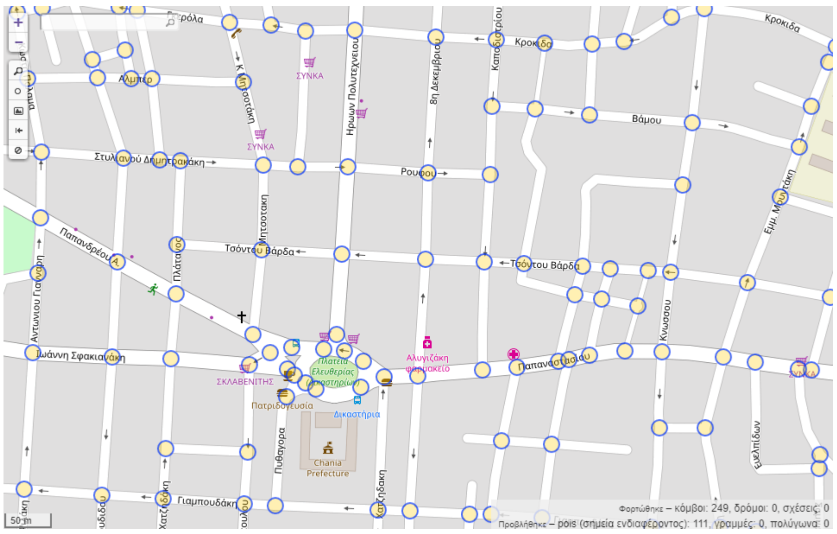

Φορτώθηκε – κόμβοι: 249, δρόμοι: 0, σχέσεις: 0
Προβλήθηκε – pois (σημεία ενδιαφέροντος): 111, γραμμές: 0, πολύγωνα: 0
50 m

Figure 32 - Bounding box

Figure 34 shows is a visualization of the results we get for the centers every time we run our query for a specific bounding box on OpenStreetMaps data.

A bounding box is a box of coordinates that is supposed to change during the whole process of dataset gathering in order to add diversity to our data. Downloading the entire dataset from the same bounding box would not be wise for various reasons.

No matter what problem you are trying to solve in object detection, the items you want to detect should be presented to the network under many variations (lighting, camera angle, etc.). The same rule applies here and, in addition, we must consider that every country and, by extend, every city has its own characteristics as far as town planning and street layout is concerned. According to that rationale, if our dataset contained pictures only from Chania for example, then the results on other cities would be compromised. The more you allow your neural network to “see”, the more prepared it will be for new data.

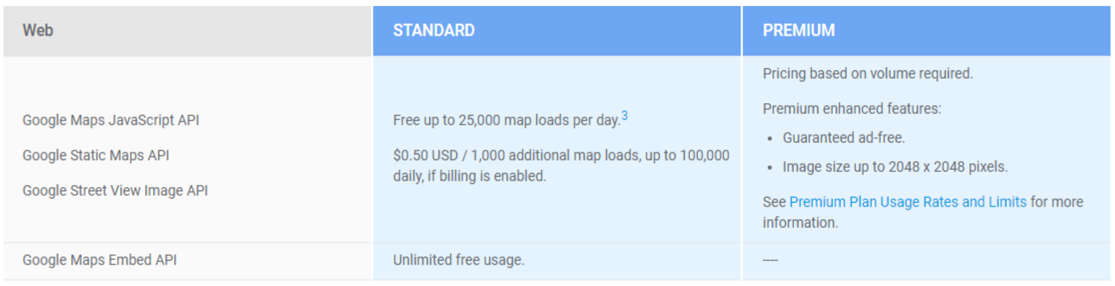
For the above reasons, we had to change the bounding box during every run of our dataset creator. This procedure was done a great many times and on many different cities, e.g. Chania, Athens, Milan, Berlin, Madrid, New York, etc. To be honest, if we wanted to achieve perfection on our results, a better approach to our implementation would be to add a restriction to run our final application exclusively at the cities it is trained on. Another implementation would be to have a large dataset containing data from every country of the world, but that sounds, and probably is, impossible. Despite all that, our current results are already very satisfying, even on places it is not trained on, but more details on this subject will be given in the results.

As mentioned earlier, not all nodes are crossroads, so we have to sort the results out by hand.

5.1.2 Google Static Maps API

After we managed to have in our possession a list of nodes containing coordinates of crossroad candidates, we have to download the corresponding satellite image. For this task, we decided to use Google's Static Maps API. [29]

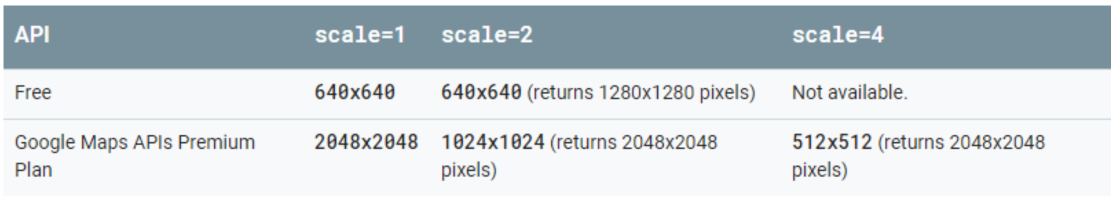
Following the instructions from their web page, first of all we had to create a new license API_KEY. There are a few differences between the free and premium version and we had to figure out which one is best for us. As one can see in Figure 35, the most noticeable limitation is the map loads per day. Since we don't expect to surpass that number, it is safe to decide to use the free version.



Web	STANDARD	PREMIUM
Google Maps JavaScript API Google Static Maps API Google Street View Image API	Free up to 25,000 map loads per day.3 $0.50 USD / 1,000 additional map loads, up to 100,000 daily, if billing is enabled.	Pricing based on volume required. Premium enhanced features: • Guaranteed ad-free. • Image size up to 2048 x 2048 pixels. See Premium Plan Usage Rates and Limits for more information.
Google Maps Embed API	Unlimited free usage.	—

Figure 33 - Google license keys

One other limitation is the maximum size you can download per request which is shown below:



API	scale=1	scale=2	scale=4
Free	640x640	640x640 (returns 1280x1280 pixels)	Not available.
Google Maps APIs Premium Plan	2048x2048	1024x1024 (returns 2048x2048 pixels)	512x512 (returns 2048x2048 pixels)

As we can see if we use the scale=2 option we get in return a 1280x1280 pixels image which is actually pretty okay for us, since we the final image size we use for our application is 800x800. By final image size we mean the size of the image that is sent back to the server for detection, when someone presses the “Find Crossroads” button.

In our dataset images we used images of different sizes, usually smaller than 1280x1280, so in conclusion the free API_KEY was fine for us.

Afterwards, to download the desired satellite image, one has to construct a URL which contains all the necessary information about the target image, or as they call it to set the “URL parameters”. These parameters contain a lot of options, such as the center of image, the zoom level, the size, the format, etc.

Therefore, we set our Java “dataset creator” to adjust our desired parameters and apply this API for every node of coordinates we got from our OSM API. Just before downloading each image, we applied one last modification; we removed the Google logo from the bottom of every image, in fear that our network might consider it as an important aspect of recognition during learning. This procedure was done by downloading every image a little bit taller, e.g. 800x900px and then removing 50 pixels from the top and 50 pixels from the bottom. Figure 36 shows an example.

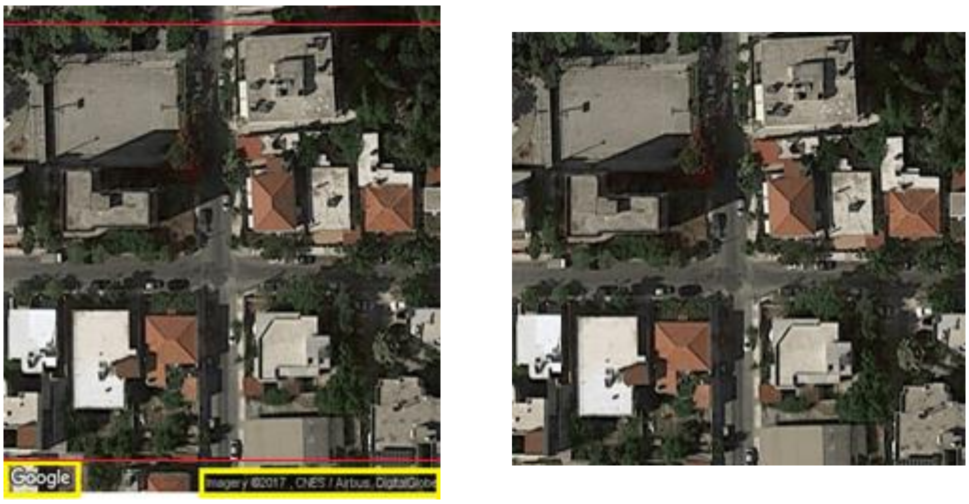


Figure 34 - Google logo removal

5.2 Data Preparation

Before we feed our data in the network for training, we had to process them a bit. First, and most important, since we use supervised learning, we have to label our images. This process is basically drawing boxes around the crossroads in the selected images. This is one of the most time consuming tasks, while preparing our data.

For this task, we used a label maker, available publicly online, called **LabelImg** [30]. When we are done with all the labeling, the LabelImg will automatically create an XML file that records the locations of the selected objects in the pictures. A visualization of the labeling process is shown in Figure 37.

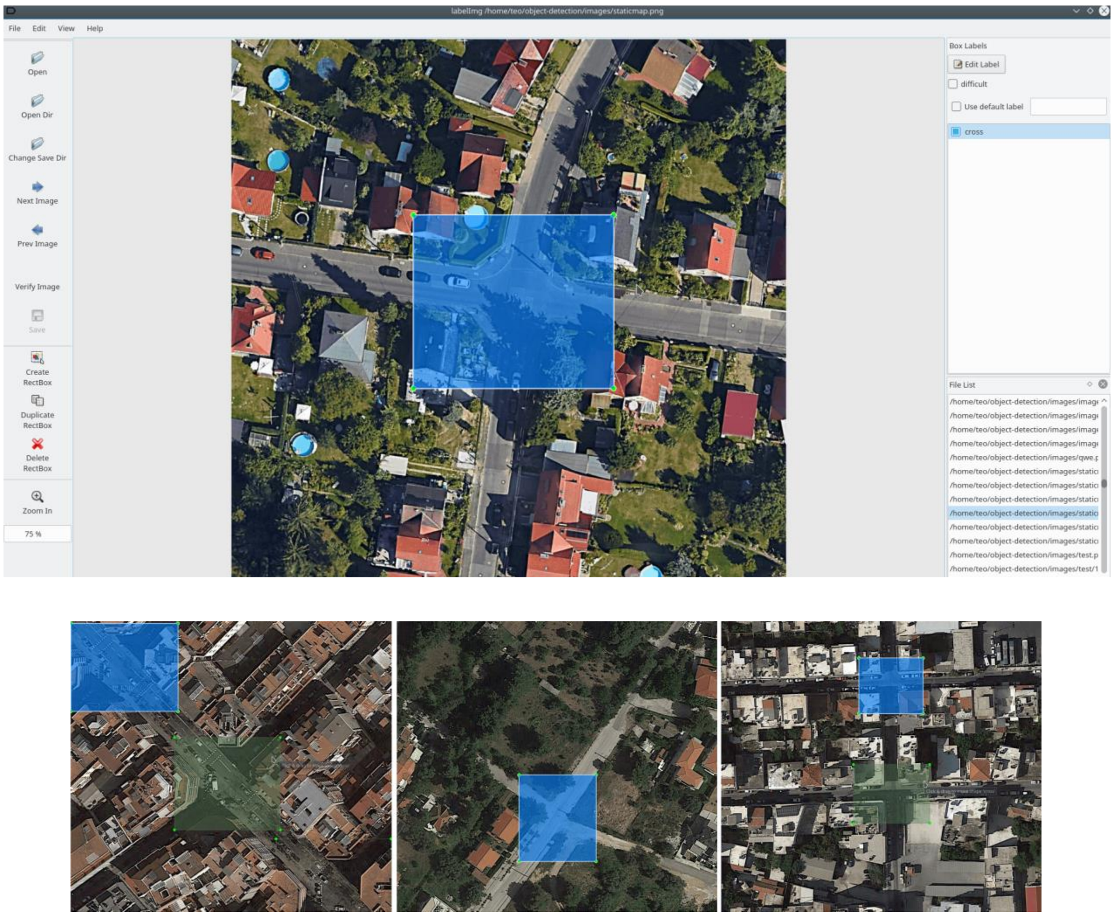


Figure 35 - Image labeling

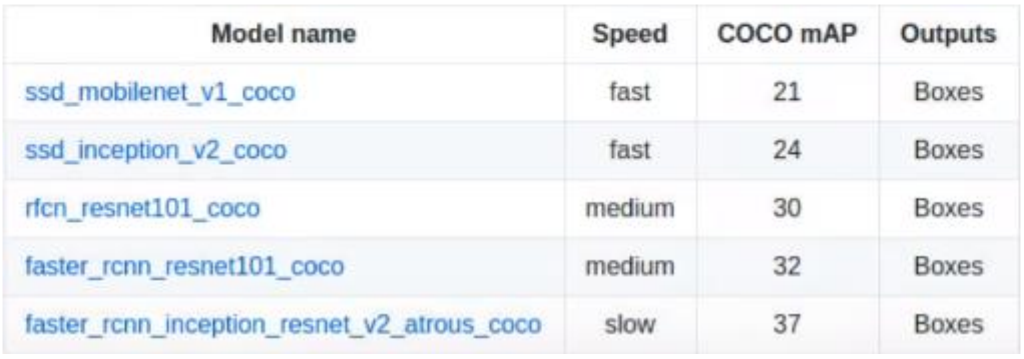
Finally, we must convert the XML outcome of LabelImg to CSV format required for training the model. This conversion is done by the python script **`xml_to_csv.py`** provided with the Tensorflow Object Detection API.

After being done with labeling our images, we move on to the next task, which is splitting our test data. A general practice is to split the labeled data into a training and a test set. The model is trained/tuned with the training set and its performance is tested on how well it generalizes to data it has never seen before with the test set. The model's performance on the test set provides insights on how the model is performing and allows sorting out issues, such as bias vs. variance trade-offs. A general rule of thumb is to deploy 90% of your data into the training set and the rest 10% for test in a random way, and so we did.

Finally, we must generate the “tf records”, required by the Tensorflow Object Detection API, which contain information about the number and the parameters of the labels we are going to use. In our case, we only have one label we want to use. We accomplish this with the python script called **`generate_tfrecords.py`**, provided with the API.

5.3 Training our Network

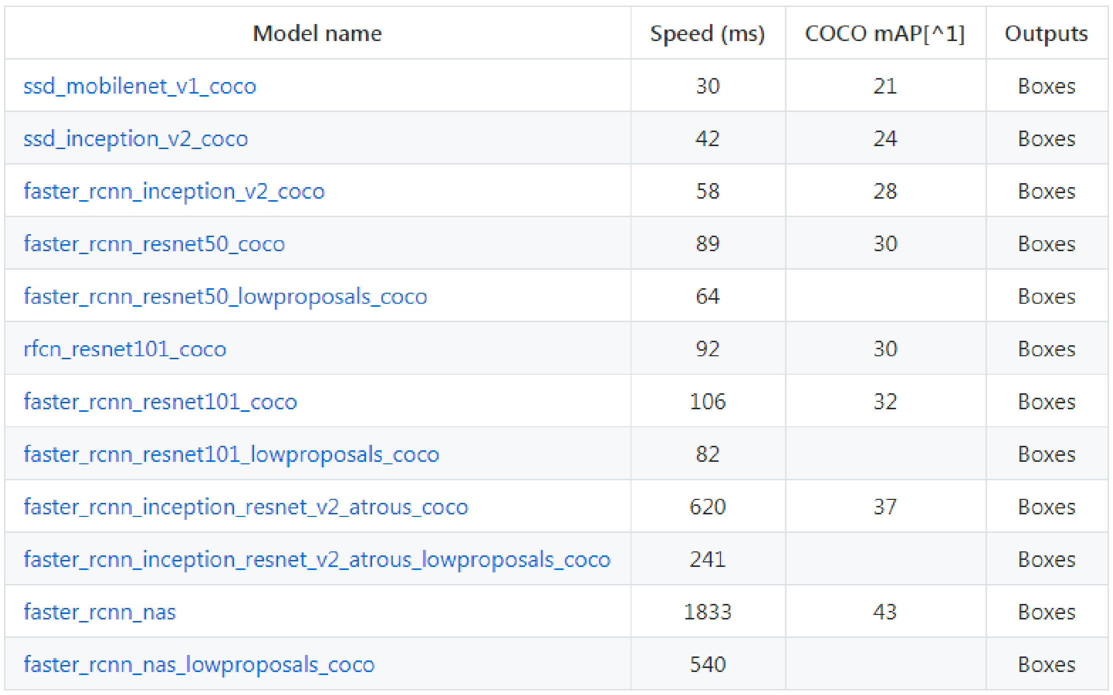
As mentioned in Chapter 3, the API we decided to use is the “Tensorflow Object Detection API”. In order to continue, we must choose the model we are going to train to use in our project. In Figure 38 (mAP stands for mean average precision) we can see the pre-trained models that were available while we were still building our project.



Model name	Speed	COCO mAP	Outputs
ssd_mobilenet_v1_coco	fast	21	Boxes
ssd_inception_v2_coco	fast	24	Boxes
rfcn_resnet101_coco	medium	30	Boxes
faster_rcnn_resnet101_coco	medium	32	Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	slow	37	Boxes

Figure 36 - Pre-trained models

One other great thing about this API is that it is constantly updated and renewed. As an example, by the time we are writing our report, Google has already added seven more models, as shown in Figure 39, which provide better scalability and more options.



Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes

Figure 37 - New available pre-trained models

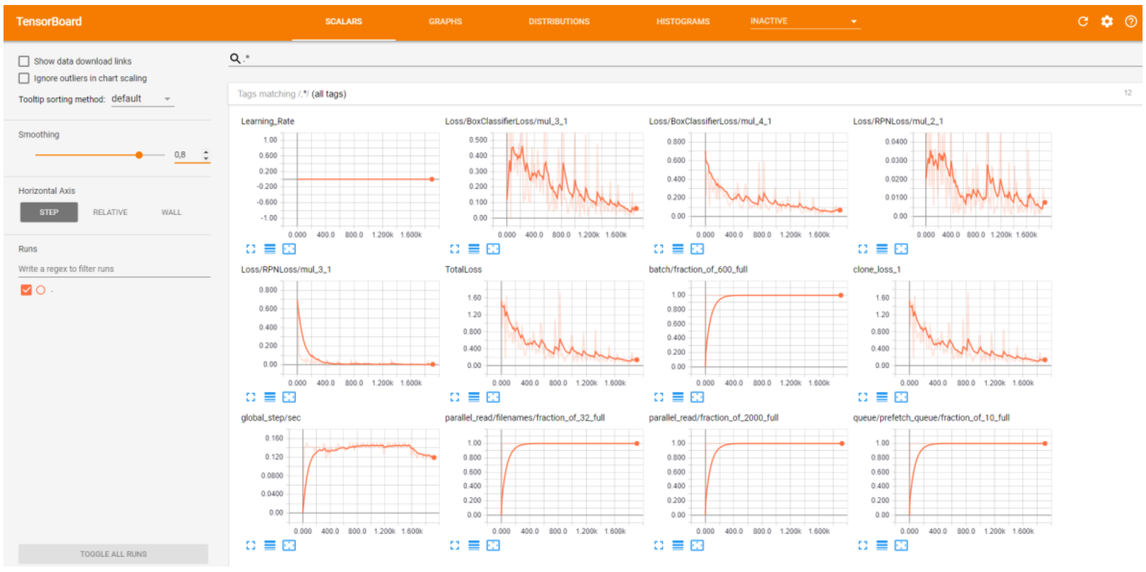
We chose to work with **`rfcn_resnet101_coco`** out of our five options for two main reasons:

1. The first is based on the obvious observation that the faster a network is, the less accurate it is. Since in our task we are going to work with images and not with video, we have the luxury to trade some speed for better performance.

2. The second reason is that these networks are trained on COCO, which is not specialized for satellite images (as we noted in Chapter 3) and we expected to see a decrease in performance to begin with, so it sounds more reasonable to choose a more accurate model than a fast one.

Having chosen the pre-trained model we prefer, we must download and edit the corresponding configuration file from the API's repository.

Now, we are ready to retrain the network with our own data. The whole training procedure lasted about four hours on a conventional desktop computer. At some point we stopped the process by hand, as we were checking out our results with the help of Tensorboard.


Tooltip sorting method: default

Smoothing 0,8

Horizontal Axis STEP

Tags matching /.*/ (all tags)

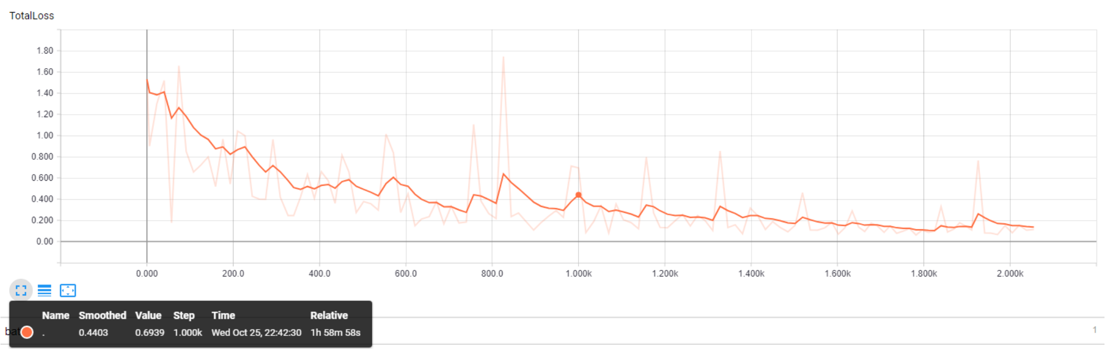
TensorBoard
SCALARS
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
INACTIVE
Show data download links
Ignore outliers in chart scaling
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
Learning_Rate
Loss/BoxClassifierLoss/mul_3_1
Loss/BoxClassifierLoss/mul_4_1
Loss/RPNLoss/mul_2_1
Loss/RPNLoss/mul_3_1
TotalLoss
batch/fraction_of_600_full
clone_loss_1
global_step/sec
parallel_read/filenames/fraction_of_32_full
parallel_read/fraction_of_2000_full
queue/prefetch_queue/fraction_of_10_full

Figure 38 - Tensorflow results

Figure 40 shows a first look on our final results. As we can see, it provides a lot of useful information about our training procedure, but probably the most important and worth discussing metric is the one of Total Loss. The lower the Loss, the better a model performs. The loss is calculated on both the training and the testing sets and its interpretation is how well the model performs on these two sets. Loss is not in percentage as opposed to accuracy and it is a summation of the errors made for each example in training or testing sets.

In the case of neural networks, the loss is usually the negative log-likelihood (basically, cross entropy) or the residual sum of squares (or sum of squared errors of prediction) for classification and regression respectively. Then naturally, the main objective in a learning model is to reduce (minimize) the loss function's value with respect to the model's parameters by changing the weight vector values through different optimization methods, such as backpropagation in neural networks. The loss value implies how well or bad a certain model behaves after each iteration of optimization. Ideally, one would expect the reduction of loss after each, or several, iteration(s).

The accuracy of a model is usually determined after the model parameters are learned and fixed and no learning is taking place. Then, the test examples are fed to the model and the number of mistakes (zero-one loss) the model makes is recorded, after comparison to the true targets. Then, the percentage of misclassification is calculated.



Name	Smoothed	Value	Step	Time	Relative
.	0.4403	0.6939	1.000k	Wed Oct 25, 22:42:30	1h 58m 58s

Figure 39 - Total Loss function

Figure 41 is our own Total Loss function over training steps after some smoothing. Keep in mind that with our hardware (without use of the GPU), 1000 training steps took about two hours to complete and correspondingly 2000 training steps took about four hours. As we can see, the majority of training is completed after three hours and in the last hour there was no real improvement, so we decided to stop the learning process.

Once we are satisfied with the learning procedure, we stop the process. Then, we must find the checkpoint file that was created the time we stopped training and use it to export the frozen inference graph, which is actually the network we are going to use for our detection. We do this using the **`export_inference_graph.py`** script.

We are almost ready to go. To test our network, we use one last python script which loads the necessary libraries, our frozen inference graph and the files we want to use for detection and present the outcome. Figure 42 shows a first result.

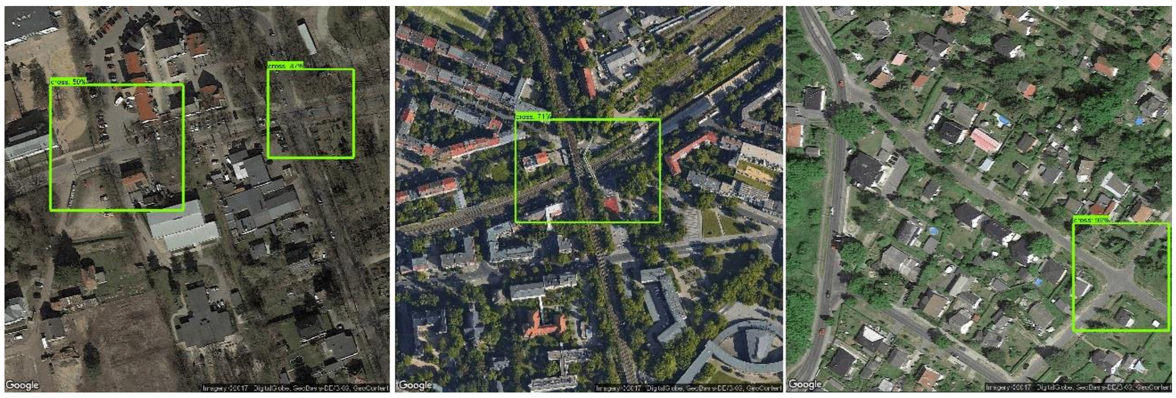


Figure 40 - First results

5.4 Setting up the Server

Since we have been working a lot with python, and also our last and very important script, the one that takes the image and produces the results is also in python, we figured it would be appropriate to set up our web server in python too. Using a browser front end simplifies the process of obtaining the target image, running it through our detector and displaying it back with the results in the browser. For our web server, we decided to use an open framework called Flask.

Flask [31] is a Python framework, based on Werkzeug, Jinja2 and inspired by the Sinatra Ruby framework, available under BSD license. Although Flask is rather young compared to most Python frameworks, it holds a great promise and has already gained popularity among Python web developers. Flask was designed to be easy to use and extend. The idea behind Flask is to build a solid foundation for web applications of different complexity. One is free to plug in any extensions needed and build own modules.

So, we set up our web server, which consists of the following parts:

- The **`main.py`**, the back-end of our server, that handles which **`html`** files are displayed each time, receives the image the user wants to process, feeds it to the neural network through a "POST" function, gets the result and saves it for later display.

- The **`.html`** files which are the front-end part of our server. The main page consists of a Google map, with a search box attached to it so the user is able to go to the desired location she likes and start looking for images to send back to the network for processing. This is done using an Ajax function, which is called whenever the "Find Crossroads" button is pressed. After the detection is completed, the image is displayed to the results part of the page.

- The **`html`** files include a **`.css`** file which adds styling to our application.

Figure 43 shows the general layout of our web interface before the query and Figure 44 shows the layout with the results displayed on the right.

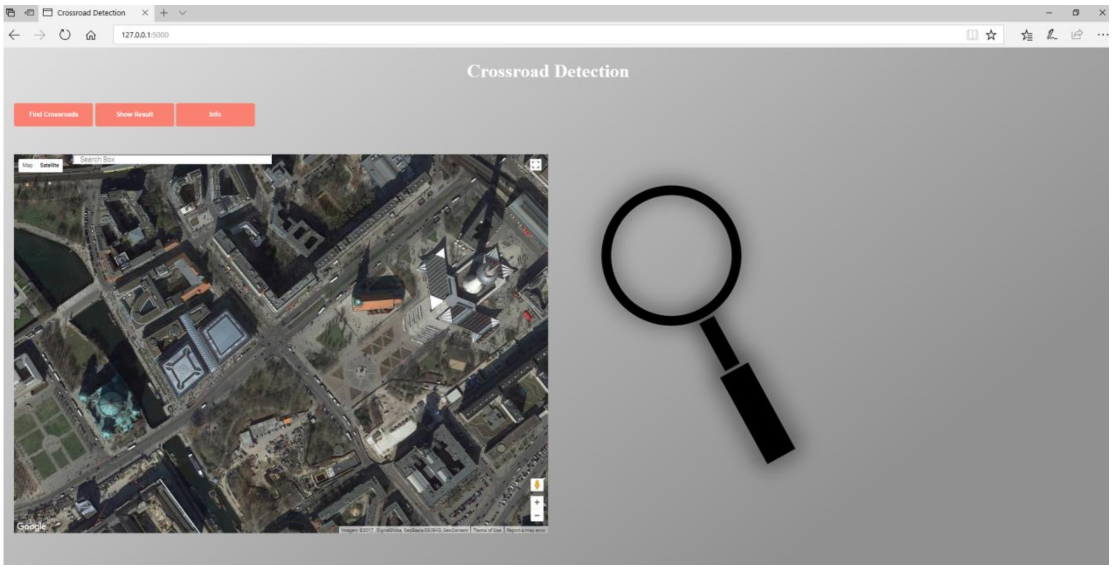


Figure 41 - Application home page

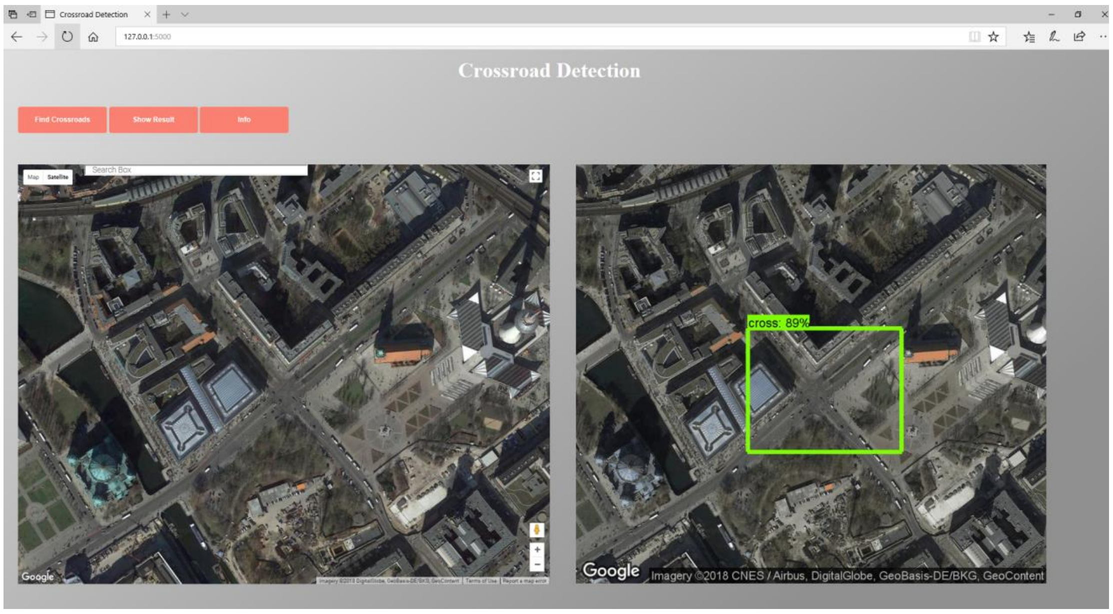
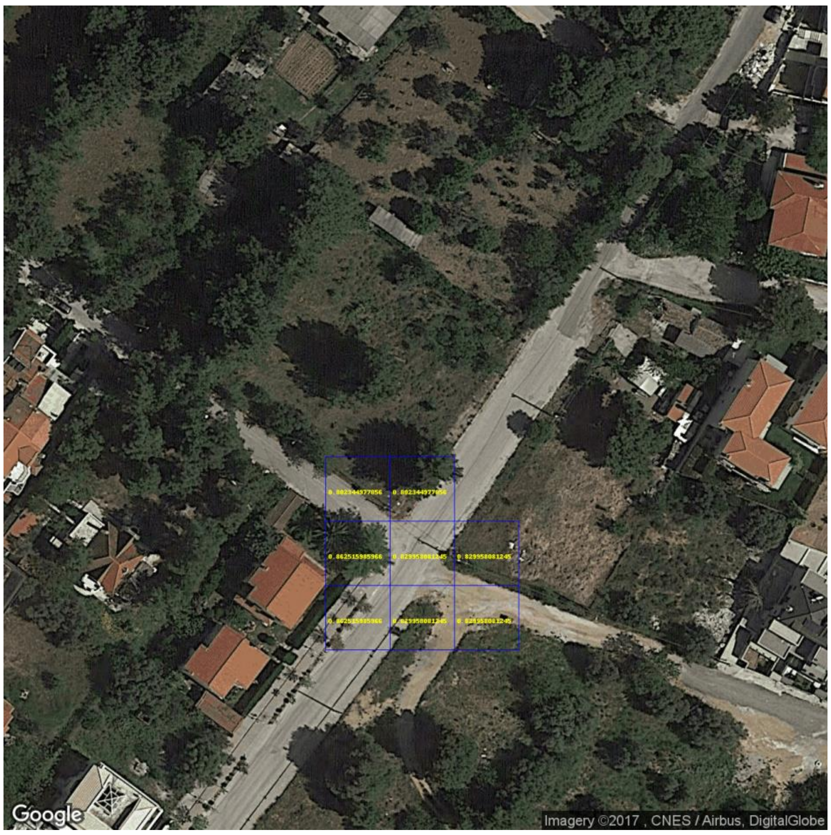


Figure 42 - Homepage with results

Chapter 6 – Results

Going back to Chapter 3, in our first implementation, we can see that the results were somewhat accurate, but not quite as good as we wish them to be. Figure 45 shows some of these early results.



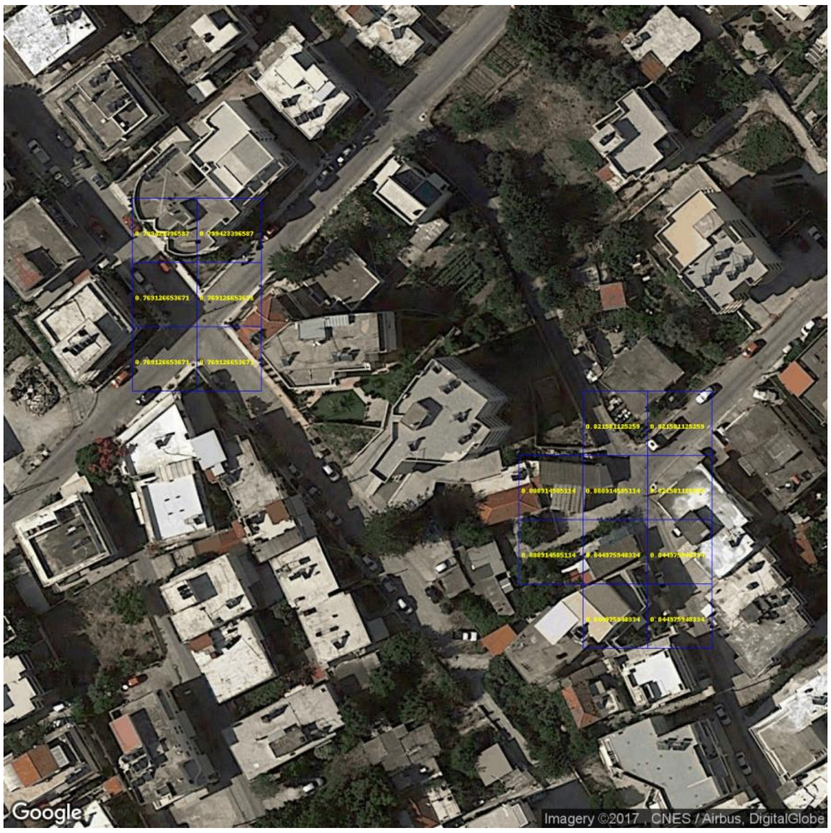


Figure 43 - Early results

We did not proceed on the final step, to draw rectangles around the crossroads that we found, because we knew we had to improve these initial findings, which nevertheless gave us a good taste of classification and localization accuracy.

Another factor worth examining is computational time. The detection process lasted about two seconds per window iteration, which means that, depending on the image size, window size, and window step, the whole image analysis lasted a couple of minutes. Comparing these times to our final implementation, where the whole process of detecting an 800x800px image takes a little less than 10 seconds, we noted a pretty significant improvement. The original time for the final implementation was a bit higher, about 13-14 seconds, but we made a few modifications to our back-end code to drop it down, the most important of which was loading the initial detection graph only once, when the server is initially booted.

As far as accuracy is concerned, the results are also improved dramatically as we can see in Figures 46, 47, 48, and 49 below, which demonstrate different cases from four cities.

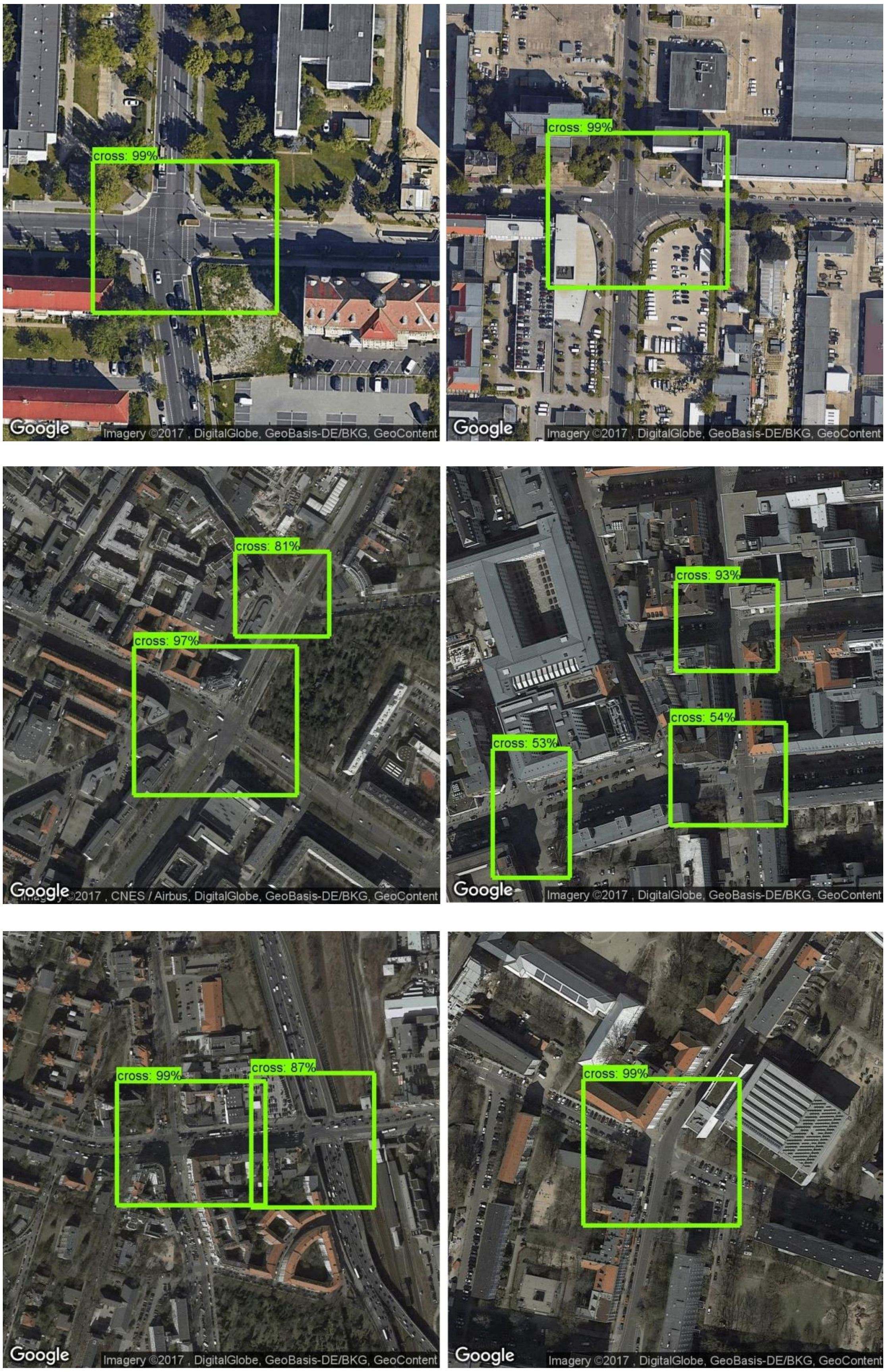

cross: 99%
Google
Imagery ©2017 , DigitalGlobe, GeoBasis-DE/BKG, GeoContent
cross: 99%
Google
Imagery ©2017 , DigitalGlobe, GeoBasis-DE/BKG, GeoContent
cross: 81%
cross: 97%
Google
©2017 , CNES / Airbus, DigitalGlobe, GeoBasis-DE/BKG, GeoContent
cross: 93%
cross: 54%
cross: 53%
Google
Imagery ©2017 , DigitalGlobe, GeoBasis-DE/BKG, GeoContent
cross: 99%
cross: 87%
Google
Imagery ©2017 , DigitalGlobe, GeoBasis-DE/BKG, GeoContent
cross: 99%
Google
Imagery ©2017 , DigitalGlobe, GeoBasis-DE/BKG, GeoContent

Figure 44 - Berlin results

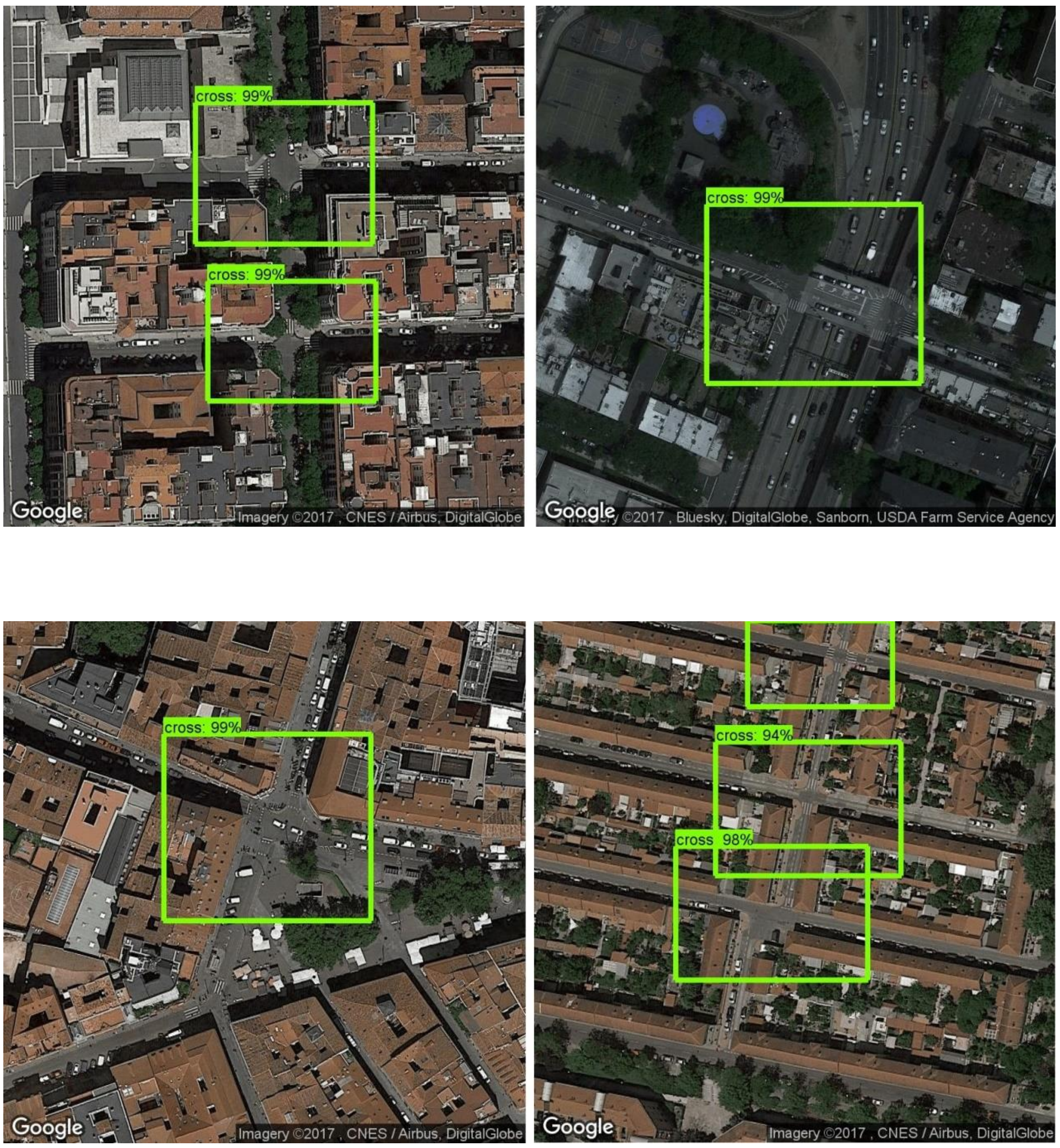

cross: 99%
cross: 99%
Google
Imagery ©2017 , CNES / Airbus, DigitalGlobe
cross: 99%
Google
Imagery ©2017 , Bluesky, DigitalGlobe, Sanborn, USDA Farm Service Agency
cross: 99%
Google
Imagery ©2017 , CNES / Airbus, DigitalGlobe
cross: 94%
cross: 98%
Google
Imagery ©2017 , CNES / Airbus, DigitalGlobe

Figure 45 - Madrid results

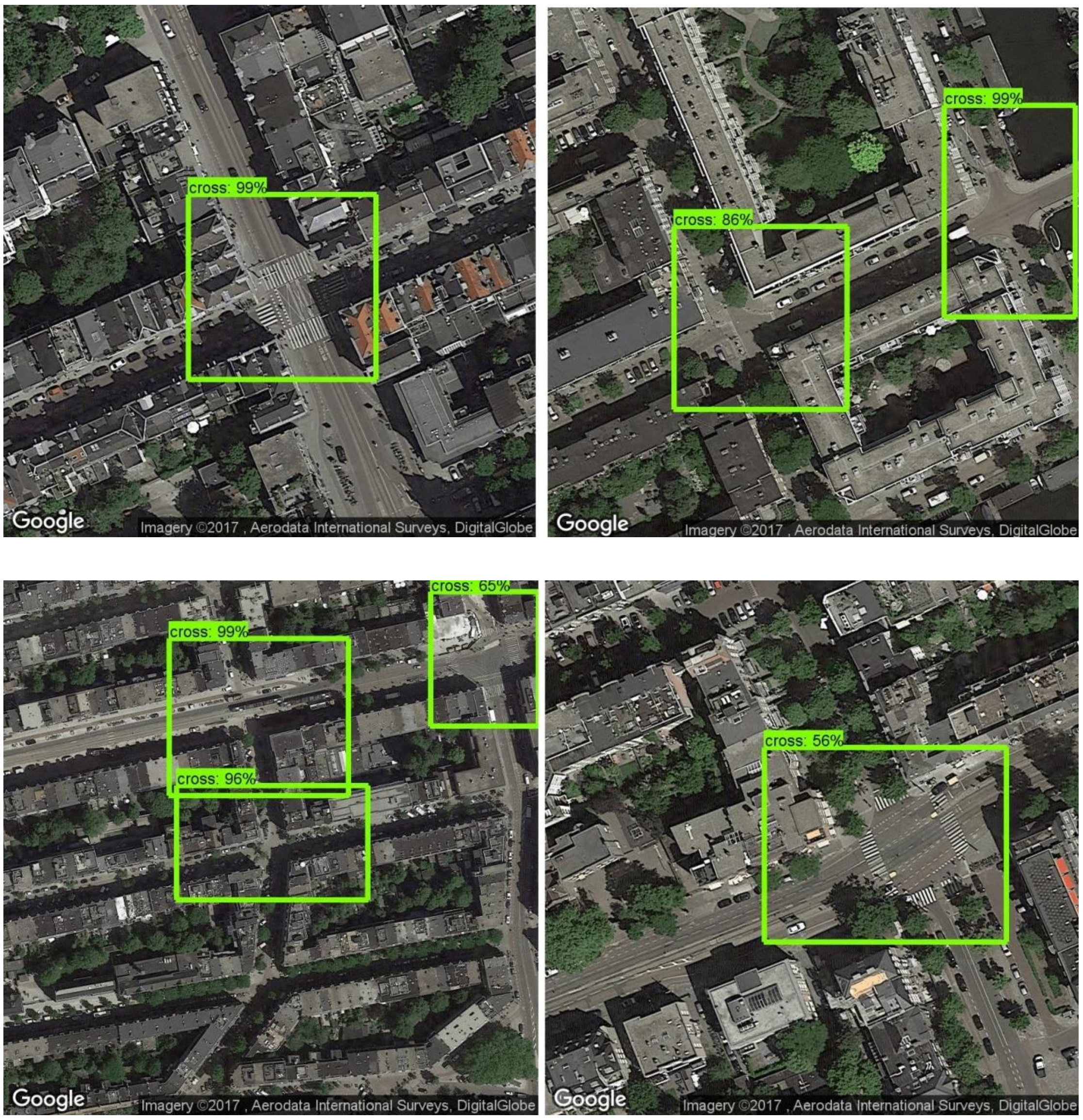

cross: 99%
Google
Imagery ©2017 , Aerodata International Surveys, DigitalGlobe
cross: 99%
cross: 86%
Google
Imagery ©2017 , Aerodata International Surveys, DigitalGlobe
cross: 65%
cross: 99%
cross: 96%
Google
Imagery ©2017 , Aerodata International Surveys, DigitalGlobe
cross: 56%
Google
Imagery ©2017 , Aerodata International Surveys, DigitalGlobe

Figure 46 - Amsterdam results

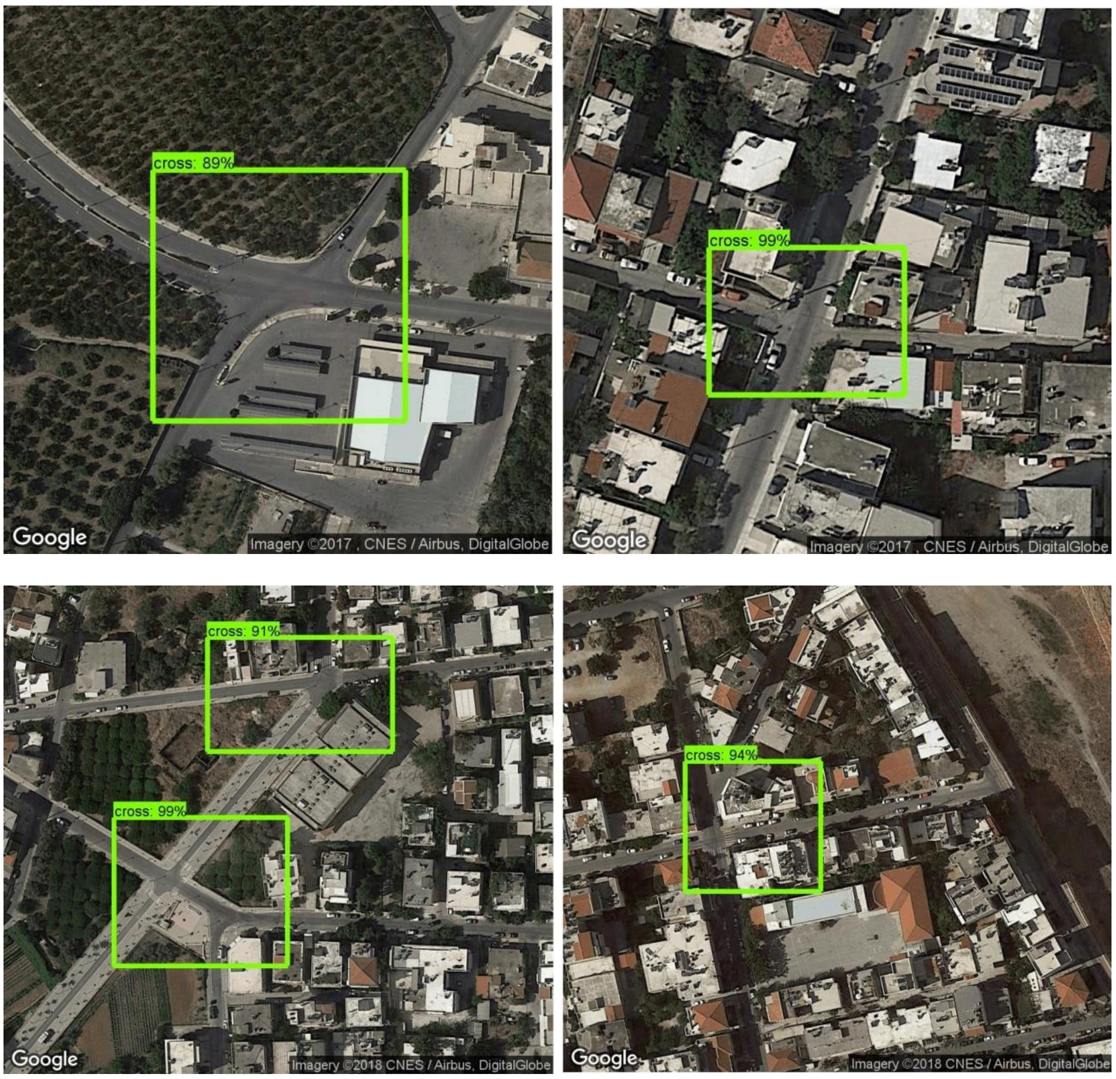

cross: 89%
Google
Imagery ©2017 , CNES / Airbus, DigitalGlobe
cross: 99%
Google
Imagery ©2017 , CNES / Airbus, DigitalGlobe
cross: 91%
cross: 99%
Google
Imagery ©2018 CNES / Airbus, DigitalGlobe
cross: 94%
Google
Imagery ©2018 CNES / Airbus, DigitalGlobe

Figure 47 - Chania results

As we can see, our network identifies correctly all the crossroads and in the majority of the cases with high confidence.

In the figures below, we present some more difficult cases. In many of them our network responds well, in others there is room for improvement.

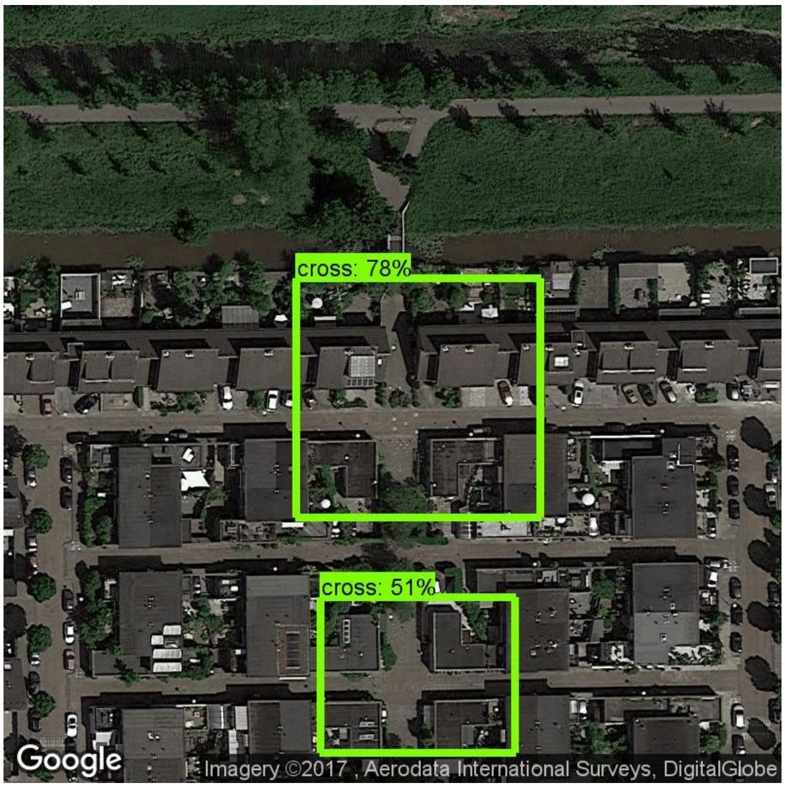

cross: 78%
cross: 51%
Google
Imagery ©2017 , Aerodata International Surveys, DigitalGlobe

Figure 48 - Difficult case 1

Here, in Figure 50, there are many parts in the image where the network could be fooled. It responded well, but we dropped down a bit in confidence value.

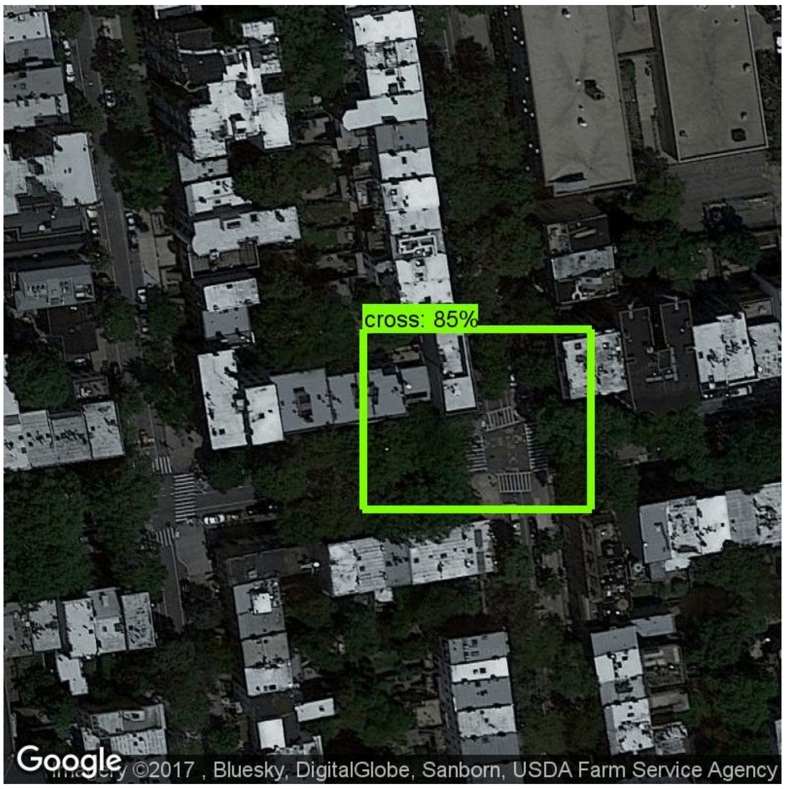

cross: 85%
Google
Imagery ©2017 , Bluesky, DigitalGlobe, Sanborn, USDA Farm Service Agency

Figure 49 - Difficult case 2

In Figure 51 both crossroads are hidden behind trees and make it hard even for a human eye to be absolutely sure about the result. Our network finds one of them, but misses the second one on the left.

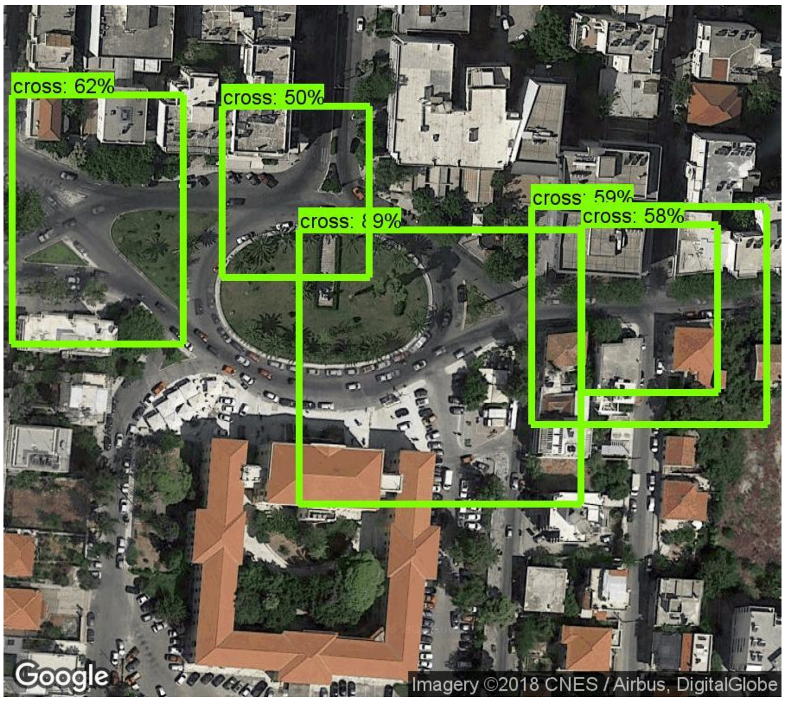


Figure 50 - Difficult case 3

In the scenario of Figure 52, we deal with a very complicated situation. We see many crossroads very close to each other and in addition most of them are not the classic “cross” style. Nevertheless, we can say that our network does a great job identifying almost all crossroads. The only drawback is that it classified one of them twice and missed one on the left side of the roundabout.

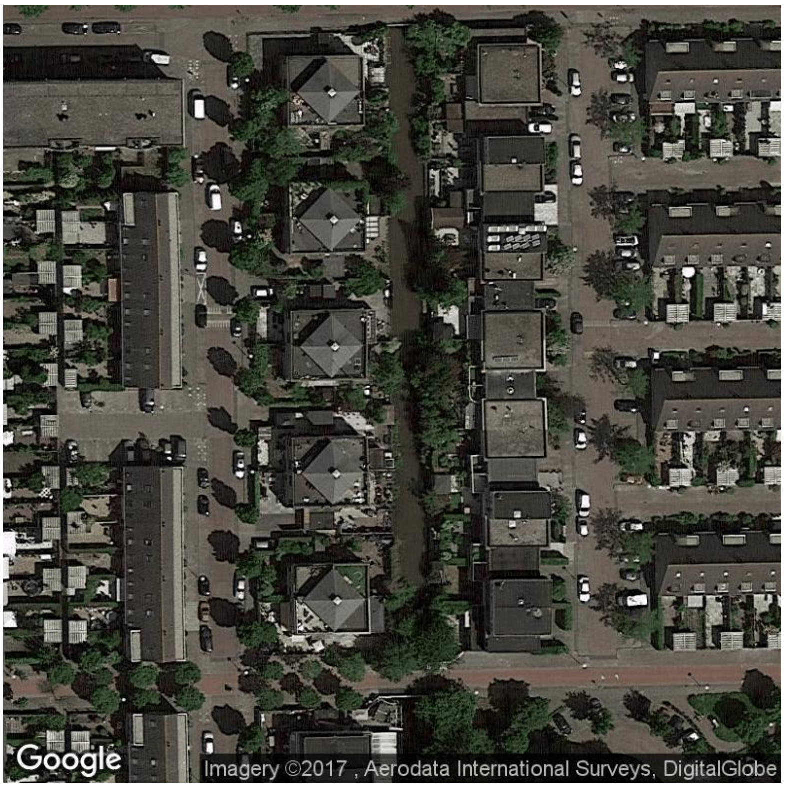

Google
Imagery ©2017 , Aerodata International Surveys, DigitalGlobe

Figure 51 - Difficult case 4

The scenario in Figure 53 above was a “trick” search. All the intersections in this image are “T-junctions” and they don’t form any regular crossroads, although a human eye may argue that there is one hiding on the left bottom part, under the trees.

In Figure 54 below, we can see a few results on which our network did not respond with success. Unsuccessful attempts include missing some crossroads, misclassifying some roads into crossroads, while they are not, or detecting some of them twice. We discuss a little bit more the double detection scenario in the last chapter.

A final thought before we see the above results is that we have to keep in mind that no object detection technique/implementation is ever 100% accurate. In our case we see correct results in more than 80-85% of the cases.

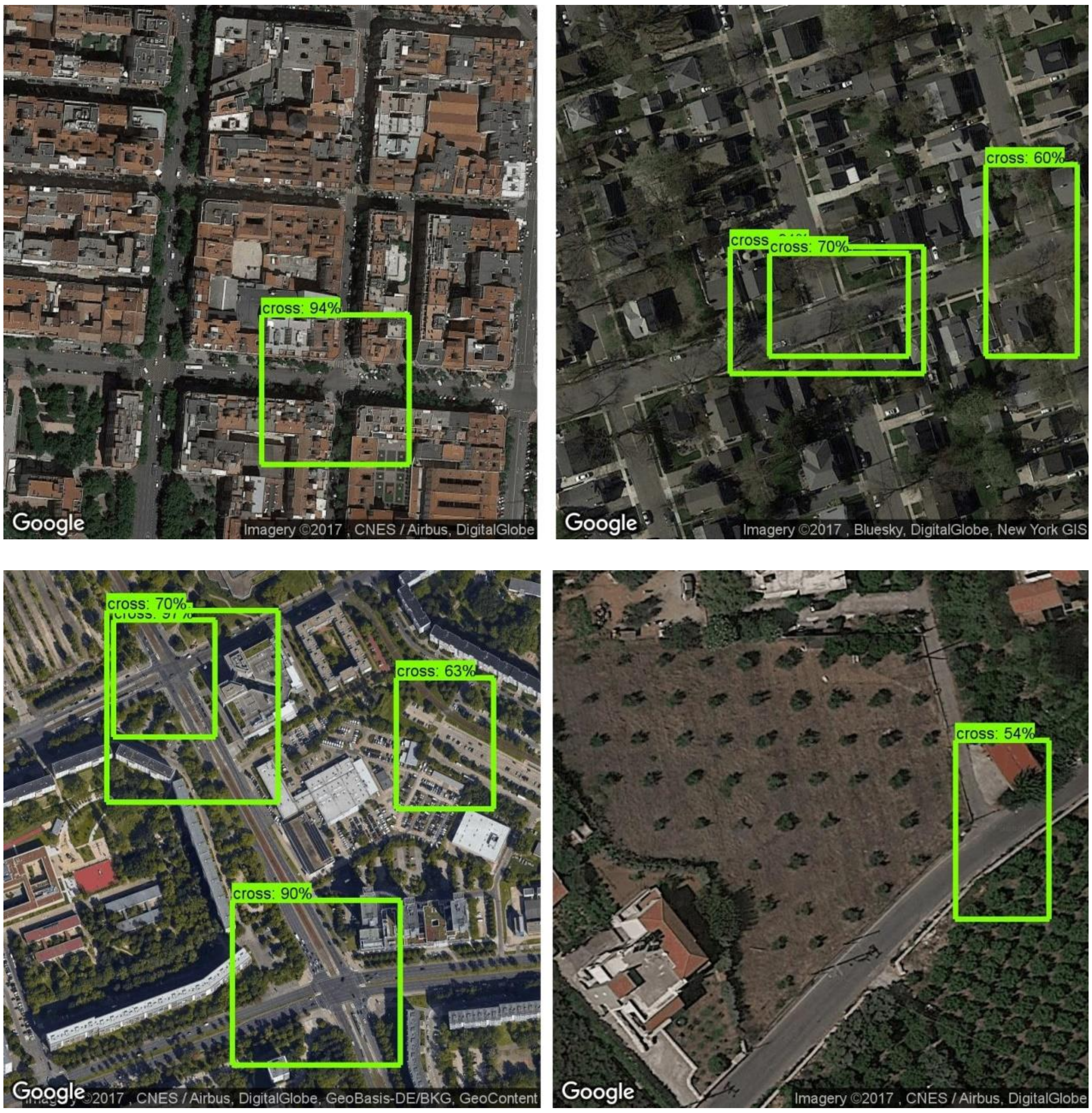

cross: 94%
Google
Imagery ©2017 , CNES / Airbus, DigitalGlobe
cross: 60%
cross
cross: 70%
Google
Imagery ©2017 , Bluesky, DigitalGlobe, New York GIS
cross: 70%
cross: 63%
cross: 90%
Google
©2017 , CNES / Airbus, DigitalGlobe, GeoBasis-DE/BKG, GeoContent
cross: 54%
Google
Imagery ©2017 , CNES / Airbus, DigitalGlobe

Figure 52 - Defective results

Chapter 7 – Conclusion and Future Work

Satellite images in machine vision are a pretty challenging task, especially if we try to apply the pre-existing methods. In this thesis, we suggested a solution using Deep Neural Networks and more specifically we trained them using the Transfer Learning technique. Finally, we constructed a web application to make it easier for anyone to test our work and also assist in any future applications our work may contribute.

Finally, we would like to revisit some of our results one last time. We believe that one of the main reasons we sometimes get less accurate results, is the nature of crossroads themselves as entities. In contrast to other objects, like animals, people, cars etc., in crossroads there is no solid definition on their specific outline. For this reason, it is a bit harder to say with confidence where it starts and where it ends. In Figure 55 we see an example of such a case. It is worth mentioning that on the up and right part of the image there is a bridge over a road, which is technically not a crossroad, and so it was correctly left out.

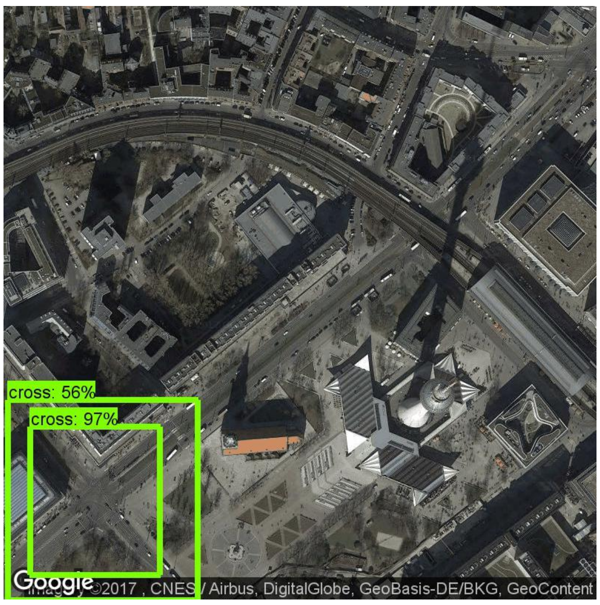

cross: 56%
cross: 97%
Google
Imagery ©2017 , CNES / Airbus, DigitalGlobe, GeoBasis-DE/BKG, GeoContent

Figure 53 - Double detection

This result does not occur so often and is not entirely wrong, since both squares indicate a crossroad, but with different borders. Also, there is no pre-trained model we are aware of, that was trained on detecting objects in satellite imagery. The pre-trained models we had to choose from were all trained on common everyday objects. Keeping in mind the above, it is definitely a step towards improvement, if we even study such problems.

Here are some things worth revisiting in the future for additional improvements:

- Further increase the training dataset five or even ten times and try to have even more diversity.
- As far as the zoom level is concerned, we decided to leave it up to the user to decide the desired zoom level at which the detection is going to take place and not set it to a fixed value. We might have better results with a fixed zoom value, especially if we set it to the value on which our network graph was trained on. This means that at lower zoom level, we may see a drastic decrease in accuracy. That problem can be dealt in two ways:
    - Train our network on more zoom levels
    - Take the original image and split it down to a number of images with higher zoom level, close to the level on which our network was trained on, perform the detection, and then reassemble the original photo from the previous pieces along with the results.
- Experiment with training more network model graphs in order to reduce time or increase accuracy, depending on future application needs. Some of the models in the Tensorflow Object detection API are recommended for video, so it would be interesting to test our techniques on video streams.
- Add more objects for detection, such as water pools, houses, automobiles, etc.

Bibliography

[1] S. C. class, 2017. [Online]. Available: http://cs231n.github.io/classification/.

[2] [Online]. Available: http://www.cbsr.ia.ac.cn/users/ynyu/detection.html.

[3] "https://stackoverflow.com/," [Online]. Available: https://stackoverflow.com/questions/45035831/how-can-i-detect-and-localize-object-using-tensorflow-and-convolutional-neural-n.

[4] C. U. B. Service, "http://blogs.cornell.edu/," 2015. [Online]. Available: http://blogs.cornell.edu/info2040/2015/09/08/neural-networks-and-machine-learning/.

[5] M. Nielsen, "http://neuralnetworksanddeeplearning.com/," 2017. [Online]. Available: http://neuralnetworksanddeeplearning.com/.

[6] "https://en.wikibooks.org," [Online]. Available: https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Activation_Functions.

[7] J. Brownlee, 2016. [Online]. Available: https://machinelearningmastery.com/what-is-deep-learning/.

[8] B. BRENNER, 2017. [Online]. Available: https://news.sophos.com/en-us/2017/06/02/were-taking-a-quantum-leap-over-traditional-machine-learning/.

[9] [Online]. Available: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/neural_networks.html.

[10] "https://www.datarobot.com/blog/a-primer-on-deep-learning/," [Online].

[11] [Online]. Available: https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/05/31130754/transfer-learning.jpeg.

[12] [Online]. Available: https://towardsdatascience.com/transfer-learning-using-keras-d804b2e04ef8.

[13] [Online]. Available: https://codelabs.developers.google.com/codelabs/cpb102-txf-learning/index.html?index=..%2F..%2Findex#5.

[14] A. Ng. [Online]. Available: https://yashk2810.github.io/Transfer-Learning/.

[15] "https://en.wikipedia.org/wiki/OpenStreetMap," [Online].

[16] "https://en.wikipedia.org/wiki/TensorFlow," [Online].

[17] V. Mnih, "Machine Learning for Aerial Image Labeling," 2013.

[18] A. L. Johnson, "DeepOSM," p. https://github.com/trailbehind/DeepOSM.

[19] G. Levin, D. Newbury and K. McDonald, "http://www.terrapattern.com/team," [Online].

[20] A. V. Etten, "Object Detection in Satellite Imagery, a Low Overhead Approach," pp. https://medium.com/the-downlinq/object-detection-in-satellite-imagery-a-low-overhead-approach-part-i-cbd96154a1b7, 2016.

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "https://github.com/tensorflow/models/tree/master/research/inception," [Online].

[22] "https://en.wikipedia.org/wiki/ImageNet," [Online].

[23] "https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0," [Online].

[24] "Tensorboard," Google, [Online]. Available: https://www.tensorflow.org/get_started/summaries_and_tensorboard.

[25] "http://iipimage.sourceforge.net/documentation/images/," [Online].

[26] "https://github.com/tensorflow/models/tree/master/research/object_detection," [Online].

[27] "http://cocodataset.org/#home," [Online].

[28] "https://wiki.openstreetmap.org/wiki/Overpass_API," [Online].

[29] "https://developers.google.com/maps/documentation/static-maps/," [Online].

[30] "https://github.com/tzutalin/labelImg," [Online].

[31] "http://flask.pocoo.org/," [Online].

[33] [Online]. Available: https://4.bp.blogspot.com/-TMOLlkJBxms/Vt3HQXpE2cI/AAAAAAAAA8E/7X7XRFOY6Xo/s1600/image03.png.

[34] [Online]. Available: https://towardsdatascience.com/transfer-learning-using-keras-d804b2e04ef8.