AUTOMATED ANALYSIS OF OVERHEAD IMAGERY FOR HABITAT SEGMENTATION

Approved by:

Dr. Pradalier, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. AlRegib, Co-Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Owen
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Voss
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: December 7, 2017
A great power comes with a great responsibility

*Ben Parker*
ACKNOWLEDGEMENTS

I would like to thanks my advisor: Cedric Pradalier for sharing his knowledge and the precious insight he gave me. I would like to thanks Professor Voss for lending us his GPUs without which we could not have carry out this project. I sincerely, thanks Quentin Chipot form AgroParisTech without whom we would never had a DataSet. I would like to thanks my girlfriend and my parents for their unswerving support and love. And finally, I would like to thanks the other students of the DREAM lab for the laughs and chats.
# TABLE OF CONTENTS

Acknowledgments ................................................................. v

List of Tables ......................................................................... ix

List of Figures ......................................................................... xi

## Chapter 1: Introduction and Background ............................... 1

1.1 Motivation ................................................................. 3

1.1.1 A Limited Toolbox ..................................................... 3

1.1.2 Perspective Open by Automation ................................... 7

1.2 Case of Study ............................................................. 8

1.2.1 Territory Overview ..................................................... 8

1.3 Research Axes ............................................................ 9

## Chapter 2: Training Data Generation ................................. 10

2.1 Overhead-imagery & Classes ........................................... 10

2.1.1 Overhead-Imagery Properties ...................................... 10

2.1.2 Class Definition ....................................................... 18

2.2 Visual Confusion: From a Human Classification to a Machine DataSet ... 20

2.2.1 Intra-Class Confusion: Class Gathering ......................... 21
2.2.2 Inter-Classes Confusion: Visual Confusion .......................... 21
2.2.3 Results ................................................................. 24
2.3 Human errors and Irregularities ........................................... 27
2.4 Large Scale features and Unbalanced DataSet ......................... 27
  2.4.1 Large Scale Feature ................................................. 29
  2.4.2 Unbalanced Classes ............................................... 30
2.5 Summary ................................................................. 33

Chapter 3: Segmentation on State Of the Art Imagery ....................... 36
  3.1 Technical Approaches .................................................. 36
    3.1.1 SegNet ............................................................. 36
    3.1.2 DeepLab .......................................................... 40
    3.1.3 SegNet improvement: FlowNet2 Inspired Network ............... 45
  3.2 Results ................................................................. 47
  3.3 Discussion .............................................................. 54

Chapter 4: Segmentation and noise filtering .................................. 58
  4.1 Noise correction ........................................................ 58
    4.1.1 Conditional Random Fields ..................................... 58
    4.1.2 Local Corrections ................................................. 61
  4.2 Class Enhancement ..................................................... 68
    4.2.1 Extracting New classes From Expert Experience ............... 68
  4.3 Discussion .............................................................. 69
## LIST OF TABLES

2.1 Class definition table with class label ........................................... 19

2.2 Class level table ............................................................................ 26

2.3 Dataset Composition and Specification. ........................................... 27

2.4 Count of class appearance(App) in thousands, Before Augmentation (BA), and After Augmentation (AA) ................................................................. 31

2.5 Metrics after(AA) and before(BA) dataset augmentation after 150k iteration. 31

3.1 Segmentation scores for the different architectures. At 150,000 iterations. *DeepLab was tested on a smaller test set due to network limitation; it is enough to extract a tendency but those results are not entirely accurate. . . . 50

3.2 Per class mIoU score at 150k iterations. *DeepLab was tested on a smaller test set due to network limitation; it is enough to extract a tendency but those results are not entirely accurate. ............................... 50

3.3 Evolution of the segmentation scores over time. ................................. 52

3.4 Computational performance evaluation. ............................................. 53

3.5 Segmentation scores showing the border effect impact on the classification. Results at 150k iterations without data augmentation. ........................................ 54

4.1 Segmentation scores using CRFs as segmentation enhancement scheme. . . . 61

5.1 Segmentation scores on 1955 for the different training set. After 100k iterations. ......................................................................................... 77
5.2 Evolution of the segmentation scores over time. Note that, due to the loss of 2 classes in 1955, the last convolution of the decoder had to be learned from scratch. The metrics presented here are the actual score there was no error in the label or any mismatch between groundtruth images and predictions. Note that 1,000 iterations is equivalent to 1 epoch.

B.1 Benchmark of neural networks on the kitti dataset[41]
LIST OF FIGURES

1.1 Human made segmentation of the natural habitats. Best viewed in colors. . 2

1.2 ZNIEFF map with faunistic and floristic areas of interest in green. Region Poitou from Jorde et Al [4]. Best viewed in colors. . . . . . . . . . . . . . 5

1.3 This is an example of what Corine Land Cover looks like at a low scale. Blue: forest, Red: fields, transparent: fields nearby cities, purple: cities. 2015, Corine program, Lorraine, France. Best viewed in colors. . . . . . . . 6

1.4 The Orne drainage Basin. Best viewed in colors. . . . . . . . . . . . . . . 9

2.1 On the left before preprocessing, on the right after preprocessing [14]. . . 11

2.2 Normalized Difference Vegetation Index map. From IGN maps, 2015, lor-
raine, France. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.3 On the top: RGB image, on the bottom: grey scale image after shadow removal process. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.4 Zoom on 1976 imagery . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

2.5 Zoom on 1955 imagery . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

2.6 Zoom on 2015 RGB imagery . . . . . . . . . . . . . . . . . . . . . . . . 17

2.7 Zoom on 2015 fake infrared imagery . . . . . . . . . . . . . . . . . . . . 17

2.8 The 3 types of forest. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

2.9 The football fields and meadows are visually alike except for their shape. . . 23

2.10 Yellow: temporary meadow, Red: permanent meadow, green: broad leaves forest, Orange: hedge, Purple: hem, Blue: riparian grove . . . . . . . . . . 24
3.13 Reconstructed segmentation map after 150k iterations with a 85k images dataset using SegNet. .................................................. 56
3.14 Evaluation of the border effect impact on classification. ............... 57
4.1 Conditional Random Field on the last feature map [45] ..................... 59
4.2 Confusion Map, for DeepLab with and without CRFs ..................... 61
4.3 Conditional Random Field on the last feature map [45] ..................... 62
4.4 Results of the filtering process. On the left the segmentation of the forest without filtering, on the right segmentation map with filtering .......... 63
4.5 Original Full SegNet’s segmentation map. .................................. 65
4.6 Full SegNet’s segmentation map after filtering. .......................... 66
4.7 Full SegNet’s segmentation map after error correction. .................. 67
5.1 Left RGB, center gray scale conversion, right objective. ................. 71
5.2 Left before, center after blurring, right objective. ........................ 72
5.3 Left before, center after first color correction, right objective. .......... 73
5.4 Left before, center after grain application, right objective. .............. 74
5.5 left before color correction, center final result, right objective. ....... 75
5.6 Left fake 1955 image, right real 1955 image. ............................. 76
5.7 Confusion matrices for the different training set. They were all tested on the same test set. .................................................. 78
SUMMARY

Reliable habitat maps are an important component of any long-term landscape planning initiatives relying on current and past land use. We propose a computational approach to identify natural habitats based on the automated analysis of overhead imagery. Ultimately, this approach could be used to assist environment scientists in determining the quality of waters bodies.

The overall objective of our project is to be able to classify the evolution of land usage since the advent of aerial imagery. In practice our aim is to bring automatic habitat classification to the level achieved by a human expert performing a fine-scale classification of habitat at resolutions covering from hedges and lake to fields, pastures or forest. Relying on the recent progresses in machine learning algorithm and in particular convolutional neural networks trained using deep learning (e.g. SegNet, DeepLab), our approach trains a machine to segment an overhead imagery into a dozen of expert-specified land use classes.

At the time of this writing, the training is performed using data from a hand-labelled high-resolution (0.5m/pixel) database around the Orne River (Moselle, France). Data are available for two time periods: 2015 and 1955. In addition, we also generated artificial 1955 data from 2015 imagery and used them as learning base for the 1955 imagery as the data available in 2015 provides more quantity and more diversity.

The following discusses the creation of the dataset required to learn land usage: which layers to use as training images and how to deal labeled data that were not intended to be used as learning material for machine learning algorithms. Moreover, it highlights the performances of these state-of- the-art machine learning algorithms for habitat recognition and highlight their potential in the context of studies in environment sciences and environmental decisions. The automatic approach might present an alternative for manual fine scale habitat classification, which is labor intensive and time consuming.

It also illustrates the potential benefits of generating artificial imagery and to be used as
learning base instead of using a learning base which is not correlated to our problem like ImageNet. This approach may prove useful for time periods where there is no labeled data.
CHAPTER 1
INTRODUCTION AND BACKGROUND

The following document discuss of computational approaches to identify habitats and land using patterns at regional scale, based on automated analysis of overhead imagery. Identifying and mapping habitats are of major importance for achieving today's sustainable management goals at regional scale. The EU biodiversity strategy to 2020 calls upon its members to map ecosystems and their ecosystem services (European Union, 2011 doi: 10.2779/39229). However, this approach presents multiple methodological challenges. To better understand the challenges at hand, it is key to know that an habitat is not visually consistent and strongly relies on context. For instance, a permanent meadow is similar to a temporary meadow except its shape and size. In addition to that, a permanent meadow may have some sparse trees on it or small man-made objects. Similarly, a park in a city from a purely visual point of view is identical to a meadow, yet, the fact that it is surrounded by buildings tells us, human, that it is a city park and not a meadow. This is to say that this problem can in fact be seen as two sub-problems, an image segmentation problem where we identify visual environment, and some sort of scene recognition problem where local features are gathered to identify the proper natural habitat. Figure 1.1 shows what the final segmentation should look like.

To tackle this problem, we chose an hybrid approach binding machine learning and standard computer vision:

- Deep Neural Networks are used to perform the visual-habitats classification as they have proved to be an outstanding tool for image recognition [1] [2] [3].

- Graphs are applied on the visual-habitats classification to extract the natural-habitat classification.
Yet, before we get into how it works, let’s see what is available today and what we can get from being able to automatically identify natural-habitats. So let us start with a little story which demonstrates that different land-uses and therefore different habitats present different ecosystem-services and human benefits in a regional context.
Motivation

The story takes place in the eighties in eastern France close by the city of Nancy. At that time people cared little about the adverse effects of pesticides on the environment and farmers used more and more of them each year. Those behaviors were particularly true in the surroundings of the Vittel’s mineral water, and thermal spring. In the nineties reports led Nestle Wasters France, the owner of the spring, to create AGRIVAIR in collaboration with the French National Institute of Agronomic Research (I.N.R.A.). Within 20 years, AGRIVAIR managed to restore the quality of water by convincing the neighboring farms to change the crops they grew, to completely stop using pesticide, and to reduce their livestock density. In other word they reshaped the land use of the drainage basin to increase the quality of the services given by the different habitats. In doing this, Nestle received the first "Biodiversity Progress” label from the Veritas office rewarding their effort but more importantly making the Vittel mineral water much more valuable. However, the tools available to perform such restoration are at the time of the writing limited.

Limited Toolbox

In the case of Nestle they had to analyze the land uses around the spring, find where the farms were, what they were cultivating, which one were responsible for polluting the soils, where were the forests, the rivers, urban areas, wetlands, wastelands, meadows and so on... This required a fine analysis of the neighboring region so let’s see the tools that are currently available to do that.

Human Expert Knowledge

Today, the best way to analyze habitats and land occupation is a human expert who analyzes overhead-imagery coupled to a ground expedition. Yet this solution is expensive: an expert costs about 2600euros a month and suffers from major drawbacks. First, the analysis is
slow: it takes about two days to label 25 square kilometers of land, and a field trip takes at least half a day, when our network can currently process 50 square kilometer per hour on a single slow GPU. In addition to that, our expert told us that: “human made classification isn’t consistent through the day.”¹ Indeed, human attention tends to fall as the day goes: the segmentation made by an expert in the morning may slightly differ from the one made in the afternoon.

*Land Use Maps: ZNIEFFs*

As the previous solution is way too expensive for municipalities, they often turn to maps of Natural Area of Faunistic and Floristic Interest also called ZNIEFFs, those maps which are made by the ministry of ecology help them identify ecosystemic features within their territory and take decisions to protect and/or improve them. Unfortunately, those maps are not exempt of faults. The first one is that they are made and published by the government which make their development slow, and thus they are not updated frequently and most of the time are not up to date. Then comes the scale issue: the listed features are large: they often are wider than a kilometer. This makes them hardly usable for small territories. Indeed, communes for instances, could be interested in finding the hedges in their agricultural areas to monitor their health, and this can’t be seen with a ZNIEFF. To that we have to add that the amount of data available is limited and not accurate. This is mostly due to the fact that they started to list the environmental features in the early eighties. Thus some of them rely on old ecological concept, and old territorial data that may have changed since then. All those issues makes them a poor asset and reduce the capacity of the communities to properly take advantage of their environmental heritage. Figure 1.2 shows the lack of data and the scale issues. We can clearly see that the scope of the information provided by those maps remain too broad to be used at a low scale and that the coverage is very limited. To illustrates that, a hedge, a pond or a meadow which provide insightful information

¹Quote from QUENTIN CHIPOT geomatics expert at AgroParisTech.
about the well being of the ecosystem are a few meters large and can’t be found using those maps which instead have a look at larger environmental features. This make them unusable for ecosystemic rehabilitation. The shade of green correspond to the level in the ZNIEFF classification.

Figure 1.2: ZNIEFF map with faunistic and floristic areas of interest in green. Region Poitou from Jorde et Al [4]. Best viewed in colors.

Land Use Maps: Corine Land Cover

In response to the need of more efficient tools, the European environment agency created the Corine program [5] which aimed at analyzing the land cover of Europe in its whole. This led at the beginning of the century to the creation of Corine Land Cover [6] a 3 level classification of the environment gathering as much as 42 classes\(^2\). Yet, according to the

\[^2\text{Here, a class is a category of land sharing similar properties from an ecosystemic point of view. See subsection 2.1.2 for more information.}\]
biologist this tool is of limited interest in term of environmental analysis. Indeed, the precision of the classification of a hundred meter makes it inapplicable to the identification of ecosystemic services\(^3\) and even less applicable to determine the quality of those. Furthermore, the classes extracted by the Corine program are not the ones one needs to identify key features in the environment. Finally, this survey isn’t updated that frequently: its last update is from 2012 which can create important classification errors. Figure 1.3 illustrates well the issues of scale and temporal delays. We can clearly see that the labels do not match the image properly: in purple the city isn’t labeled correctly and we can clearly see some meadows inside the forest which aren’t detected by CLC. In addition to all this the fields delimitation, in red, looks uncertain which explains why such data cannot be used at the scale of a town.

![Figure 1.3: This is an example of what Corine Land Cover looks like at a low scale. Blue: forest, Red: fields, transparent: fields nearby cities, purple: cities. 2015, Corine program, Lorraine, France. Best viewed in colors.](image)

\(^3\)Ecosystem services are the many and varied benefits that humans freely gain from the natural environment and from properly-functioning ecosystems. Such ecosystems include, for example, agroecosystems, forest ecosystems, grassland ecosystems and aquatic ecosystems. Collectively, these benefits are becoming known as ‘ecosystem services’, and are often integral to the provisioning of clean drinking water, the decomposition of wastes, and the natural pollination of crops and other plants. [7]
All in all, it seems clear that today the only tool fitted to help biologist to better analyze their territory is not a tool but a human expert. This calls for the creation of a new tool which recent breakthrough in machine learning computer vision and open data makes now, more than ever, possible. So let see what perspective such a tool would allow.

Perspective Open by Automation

Earlier, we discussed of the example of Nestle and how they managed to tune the surroundings of their water spring to increase its quality, but it is not the only application that an automatic classification tools would make possible. Making such a tool could be a total game changer as it could be able to classify data not only from today but also from the past. So one may wonder as to why this would be of any interest when we want to reshape today’s landscapes. Being able to look back in time helps understanding the functioning and the resilience of the actual habitats. For instance, a forest or a meadow that is hundreds years old have a much more important resilience towards perturbation and climate change than temporary meadows and forests. Accordingly, an ecosystem that changes all the time does not have time to balance it self which weakens its viability and functioning towards its direct neighboring environment. This is why we will investigate classification on old analog photography from the fifties and seventies. An other application of this toolsustainable management perspective of this approach could be the identification of green corridors and biodiversity. Mapping of green and blue corridors is of great importance in today’s sustainable french management at regional scale since the environmental policies of 2009 and 2010 (Loi Grenelle I et II). A "corridor" whether it is green or blue is a patchwork of habitats presenting an ecological interest. The blue being marine areas and the green being terrestrial areas, the automation tool could help significantly reduce the amount of time needed to identify the area by assisting the cartographers and biologists in providing them with highlighted areas of interest that a human would just have to validate, reducing the long and painful process of analyzing whole maps and extracting features by hand. It
would also provide a more unified classification if the system was to be used nation wide. If our project focuses on analyzing the quality of the water based on its surroundings there are tremendous amounts of potential applications for such a tool. The increasing number of recent studies [8] [9] [10] [11] focused on the monetary value one could can get from the ecosystem, demonstrates the need for large scale automation tools such as the one we propose. In addition to that it could also be used to monitor environmental changes over the years.

Case of Study

As said earlier, this study takes place in a global project aiming at monitoring the evolution of the land use and its impact on the water quality at the scale of the drainage basin. It is inspired by two previous studies that were carried out in Haute Normandie a northern French region and within the European union [12].

Territory Overview

The study focuses on the basin of the Orne water stream in the Grand Est region, in the north east of France. The Orne is river that originate from Ctes de Meuse and joins the Moselle water stream nearby Richemont a local town. The river is 85 kilometers long for a total basin surface of 1268 square kilometers as shown on figure 1.4.

The west of the basin distinguish itself by a very dense forest cover, located in the hillsides of the Cte de Meuses. The center west of the territory is rural with small towns nested in the great plains of Openfields. To the east, we can find the principals cities of the area: Jarny, Homcourt, Joeuf and Amnville. The topography is characterized by numerous wedged hills.
Research Axes

In order to make this automation tool we will address two distinct problems:

- **Classification:** The classification will first be made using Auto-Encoders (Deep Neural Networks able to perform image segmentation). Then some filtering will be performed, and graph analytics based on human expertise will be used to improve the classification results.

- **Going back in time:** To get back in time a fake dataset in grey scale will be generated from state of the art RGB imagery and some noise or effects will be applied to them in order for them to match the style of the target time period. Then if some data is available we will use it to finetune the so trained network.
CHAPTER 2
TRAINING DATA GENERATION

This chapter will explain how the dataset used to train the Neural Networks was made. It will highlight the issues encountered when we tried to generate the data required to train the Deep Learning algorithms and how they were overcome. It will explain how to make a machine understandable dataset based on segmentation classes that were intended to be used by humans and thus with a high level interpretation. The original class label and the images matching those were kindly provided by our partner the research institute Laboratoire d'Etude des Ressources Forêt-Bois AgroParisTech-INRA. AgroParisTech is a French university-level institution in life science and agronomy, also known as a "Grande École". INRA is the French National Institute of Agronomic Research. When making the dataset three main issues were encountered:

- Confusion created by lookalike classes.
- Human interpretation, errors, and irregularities.
- Strongly unbalanced class representation.

But first, see the data that were used in the context of the study.

**Overhead-imagery & Classes**

This section details the different data that our partners gave us. It goes through the images that we worked with and the labeled data.

**Overhead-Imagery Properties**

Images from three different time period were provided by the French national Institute of Geographical and forestry Information. (IGN): 2015, 1976 and 1955. All of them with a
spatial resolution of 2 pixels per meter or 1.5 feet per pixel. Because the task is to find small details in the environment, the choice was made to keep this resolution to train our network. As for the quality of the images handed to us, they are all 10,000x10,000 pixels ortho-images meaning that a post processing was applied to them in order to remove the terrain relief as shown in figure 2.1. Because of our Teslas k20c only have 5 gigabits of RAM we couldn’t process the whole images; so the choice was made to feed the networks with cropped images of 300x300 pixels. This allows to make batches of three 4-channels images using SegNet [13]. To obtain those images we extract 300x300pixels patches of the original ortho-photos with a stride of 150 pixels. However, this means that the scope of an image is 150 meters or a bit more than a pixel of the Corine [5] (100 meter per pixel). This creates issues as the network can only see objects smaller than 150meters when some of them such as a mineral extraction site can be 1 kilometer wide. Section 2.4 shows solutions to overcome this problem.

Figure 2.1: On the left before preprocessing, on the right after preprocessing [14].

2015 State of the Art Imagery

For the 2015 period we were given two type of images: RGB images, and RG-NearInfraRed (RG-NIR)images also known as fake color infrared images. From those we can generate an extra layer called NDVI [15] layer. We recall that NDVI stands for Normalized Difference
Vegetation Index, and that NDVI maps are obtained using the following formulae:

\[
NDVI = \frac{NIR - RED}{NIR + RED}
\]  

(2.1)

It shows how the near infrared spectrum is absorbed in comparison to the red one. This can be used to discriminate the vegetation [16] which usually has high infrared absorption. Note that this layer is not a linear combination of two channels; thus, it is harder to express using a single layer. Unfortunately, all the pictures were taken at different moment in the year, with different lighting, and weather conditions. This made unreliable the use of an NDVI layer as such. As can be see in figure 2.2 there is a huge difference in term of intensity between the trees at the top and the bottom of the image this could be explained by the fact that this picture is an assembly of multiple pictures: some were taken in spring, some at the beginning of the fall. Because of this, the infrared channel is unbalanced: in summer there is less water so the land tend to be dry, when during spring or fall there are more precipitations which leads to an increase in soils’ humidity. The increased humidity in the soils impact the mean of the infrared channel. For these reasons, the NDVI channel was not integrated as an input channel.

When experimenting with the classification it appeared that sometimes the network would miss-interpret shadows for water. So we thought to add a grey scale image on which the shadows would have been removed. To do so we used the method described by Corke et al [17]. The idea is that a shadow is only illuminated by the light reflection of the sky and its surrounding objects. If we consider the reflections from surrounding objects to be neglectable then only the sky illuminates the shadows. In our case it is even more true because we cannot see shadows that are not illuminated by the sky. The optimal parameters to transform the image in a shadow invariant image can be found through an optimization process. This process tries to minimize the variance between two images in the temperature color space: one in the shade, and one illuminated by the sun. The
parameters found help us to project our image in the temperature color space where our image is shadow invariant. The image is then converted back to grey scale color space and we have our shadow invariant image. Yet, as figure 2.3 illustrates, the resulting image has an important Signal to Noise Ratio or SNR which smooth the textures. This is problematic as the network strongly relies on textures to perform the segmentation. In [17] it is stated that the resulting images are noisy but their images are not as noisy as ours. The various preprocessing applied by the IGN on the images along with the compression of the images may have decreased the quality of the images but we cannot directly tie this to a higher noise level in our images.

All in all the images used to train the network on the 2015’s data is based on 4 channels images Blue Green Red and Near-Infrared (BGR+NIR). Details can be found in Table 2.3.
Figure 2.3: On the top: RGB image, on the bottom: grey scale image after shadow removal process.
1976 & 1955 Imagery

The picture from 1976 are analog black and white photography that have been digitized and corrected. They yield a similar resolution to the one of 2015. However, the details are not as sharp: they are slightly blurred, and a typical analog photography grain can be seen. Also the contrast and luminosity of the image is poor. Figure 2.4 illustrates those issues: the edges of the houses are hardly distinguishable from roads because of the poor contrast; the textures in the trees are completely blurred, and the grain can be seen in the meadows and fields, also some rare artifacts can be found. Those images were taken in grey scale and the digitized version covers 255 levels of grey.

Similarly the images from 1955 have been scanned and corrected. However, the quality of those images is worse: those, contain large artifacts and some of them were even torn and stuck back together. In addition to that, the details within the images are heavily blurred. Figure 2.6, 2.7, 2.4 and 2.5 show the different image types that we were provided. As for those time period the training images have a single channel.
Figure 2.4: Zoom on 1976 imagery

Figure 2.5: Zoom on 1955 imagery
Figure 2.6: Zoom on 2015 RGB imagery

Figure 2.7: Zoom on 2015 fake infrared imagery
Class Definition

Our partner AgroParisTech provided us with 25 classes to retrieve as described in table 2.1. This classification is based on an interpretation key with 6 main classes: Built areas, Agricultural lands, Moor and Bushes, Forest areas, and Hydrographical areas. All the classes are declined in subclasses with up to 4 hierarchical levels. This key was built to study the impact of the land use on the water quality, and put the emphasis on the most important areas in that regard. Its most notably true with the presence of Hems and riparian grove in the classification.
Table 2.1: Class definition table with class label

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Level 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built areas, industrial sites, and other artificial habitats</td>
<td>Residential and equipment areas, recreational facilities, and transport infrastructures</td>
<td>Residential and equipment areas</td>
<td>Recreational facilities</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>111</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>Parks and home gardens</td>
<td>Large urban parks</td>
<td>Home gardens</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>121</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>Industrial or agricultural areas</td>
<td>Industrial areas</td>
<td>Commercial areas</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>131</td>
<td>132</td>
</tr>
<tr>
<td></td>
<td>Mines, drop-off areas and dumps</td>
<td>Agricultural exploitation</td>
<td>Mineral extraction</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>133</td>
<td>141</td>
</tr>
<tr>
<td>Agricultural Lands</td>
<td>Arable land, and cultures</td>
<td>Vine yards</td>
<td>Other cultures</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>21</td>
<td>211</td>
</tr>
<tr>
<td></td>
<td>Meadows</td>
<td>Temporary meadows</td>
<td>Permanent meadows</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>221</td>
<td>222</td>
</tr>
<tr>
<td>Moor, bushes, and herbaceous vegetation</td>
<td>Hedges</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Marshes, swamps, and riparian groves</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Moor, heath and, bushes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Forest areas</td>
<td>Continuous vegetation</td>
<td>Broad leaves trees</td>
<td>Forest</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>51</td>
<td>511</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Tree farm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5112</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Coniferous trees</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fruits orchards</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5113</td>
</tr>
<tr>
<td></td>
<td>Chopping areas</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hydrographical areas</td>
<td>Lakes and marres</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Watercourses</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>62</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Continental waters shores</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>63</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Based on this class definitions, AgroParisTech classified 250 square kilometers in 2015, 125 square kilometers in 1955 and did not classify data from 1976. All of those classes give different ecosytemic services and our aim is to find them all. Yet, as explained in section 2.2 this classification was largely reinterpreted to make it more machine learning friendly.

**Visual Confusion: From a Human Classification to a Machine Data Set**

As briefly discussed earlier, this section is dedicated to the reinterpretation of the human classification to ease the learning of the different classes by a program. Indeed when we tried to learn the classification as is, a few issues came up. Some classes, like the Mixed trees class were confusing the network. This particular class is a combination of broad leaves tree and coniferous tree which both have a class of their own. For a human expert it is easy: if he sees sparse patches of conifers within a broad leaves trees’ forest then he will assume that it is actually a mixed tree forest. But for the machine with a very limited scope (150meters) it is not. Another example of issue are the hems, they are visually consistent with meadows, but they tend to have a higher length to width ratio and are often found at the edge of forests or fields. This definition, even for a human is not that precise and it strongly relies on a context which is hard to get with our scope. Similarly, for the hedges which definition cannot be easily interpreted by a neural network. They are defined as follow: to be an hedge, the trees must form a line thinner than 10 meters, and longer than 50 meters with interruptions smaller than 1.5 meters. A lot of other classes poses similar issues which is why we had to rethink the way the environment was classified. To deal with this issues 2 cases will be taken into consideration:

- Classes integrating other classes at a larger scale.

- Classes visually similar to each others that can only be distinguished by context, scale, or shape.

The idea is to shrink the amount of classes and re-expend them once the classification is
done using contextual information.

---

**Intra-Class Confusion: Class Gathering**

The intra-class confusion is a confusion that occurs when a class gathers main features from two or more classes. In our case, only the mixed tree class enters this category. As Figure 2.8 illustrates this class enters in conflict with the broadleaf trees, and the coniferous trees. As we can see, the mixed tree forest contains patches of the two others classes. In addition to that we can see a large patch of broadleaf trees at the bottom left of the image. This means that when we put the mixed tree label on this class, we actually classify patches of conifer and deciduous trees as mixed trees and not only small ones. Thus, in doing this, the neural network gets confused by the class and is having issue learning it. It leads to a degradation of its performances as the classes enter in conflict. Figure 2.11 illustrates this by showing how the presence of the mixed tree class impacts the indecision of the network on the two other classes and on itself. Those results are obtained using SegNet [13], it was trained on two different dataset. A dataset with all the classes present and a dataset with a reduced number of classes based on the previous observation. From there the confusion matrices are computed and used to monitor where the network is confused.

In order to overcome this, we count on the network ability to correctly differentiate coniferous and deciduous trees. If so, we should be able to detect mixed tree forests using post processing algorithms: they would be indicated by the presence of conifer patches within deciduous forest more about that in section 4.2. So the mixed tree class is removed from the learning phase.

---

**Inter-Classes Confusion: Visual Confusion**

The inter-class confusion is a confusion that occurs when a class is visually consistent with another one. For instance figure 2.9 shows that meadows and football field are very similar from a strict textural point of view. The context help us to differentiate them but
they are alike. Moreover, the low frequency of apparition of the football fields make them hard to learn for the network. Figure 2.10 is a perfect example of how hard it can be to differentiate the classes. On this picture we can see a hem, a permanent meadow and a temporary meadow which are alike visually. We can also see riparian groves, deciduous trees and a hedge which are also similar visually: they are all broad leaves trees. The thing that helps us distinguish the trees is their shape or the context. For instance, the expert
knows that the trees in blue are riparian groves as there is a river flowing in between them. The trees in orange and blue form a line longer than 50 meters without interruption so the expert knows that there is an hedge. The trees in green are marked as forest as they are packed together, as for the meadows, the expert distinguish them because of the slight texture marks, the one with the deepest texture, in yellow is the temporary meadow, when the red one is the permanent meadow. We can also see that there is a tree in the middle of the red area: they are often found on permanent meadow and almost never on temporary ones. The hem is differentiated from the meadows by its shape. As we can see they are often thin and long but they can be larger than the one shown in purple here.

(a) Football fields, class n112.  
(b) Permanent meadow, class n222

Figure 2.9: The football fields and meadows are visually alike except for their shape.

This leads to the merging of the permanent and temporary meadows with the hems, and to the merging of the deciduous forest with the hedges and Riparian groves. In addition to that we also merged: the extraction sites and dumps, the lakes and ponds and watercourses for the same reasons. Also the large urban parks, home gardens and Residential equipment areas and recreational facilities, were removed from the training as they entered in conflicts with the meadows. Section 4.2 will go through the class expansion process.
Figure 2.10: Yellow: temporary meadow, Red: permanent meadow, green: broad leaves forest, Orange: hedge, Purple: hem, Blue: riparian grove

Results

All in all, the classification is reshaped as can be seen in table 2.2.
Figure 2.11: Segmentation results using original classification
<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Level 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Built areas, industrial sites, and other artificial habitats</strong></td>
<td>Residential and equipment areas, recreational facilities, and transport infrastructures + Industrial or agricultural areas</td>
<td><strong>Mines, drop-off areas and dumps</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Agricultural Lands</strong></td>
<td>Arable land, and cultures</td>
<td><strong>Vine yards</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Meadows</strong></td>
<td><strong>Other cultures</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Hem</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Moor, bushes, and herbaceous vegetation</strong></td>
<td>Moor, heath and, bushes</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Forest areas</strong></td>
<td>Continuous vegetation</td>
<td><strong>Broad leaves trees</strong></td>
<td><strong>Forest</strong> + <strong>Hedge</strong> + <strong>Marshes, swamps, and riparian groves</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Tree farm</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Fruits orchards</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Coniferous trees</strong></td>
</tr>
<tr>
<td><strong>Hydrographical areas</strong></td>
<td>Lakes and marres + Watercourses</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Continental waters shores</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2.3: Dataset Composition and Specification.

<table>
<thead>
<tr>
<th></th>
<th>2015</th>
<th>1976</th>
<th>1955</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>300x300pxl</td>
<td>300x300pxl</td>
<td>300x300pxl</td>
</tr>
<tr>
<td>Channels</td>
<td>4 RGB+NIR</td>
<td>1 GreyScale</td>
<td>1 GreyScale</td>
</tr>
<tr>
<td>Resolution</td>
<td>0.5m/pxl</td>
<td>0.5m/pxl</td>
<td>0.5m/pxl</td>
</tr>
<tr>
<td>Labeled Surface</td>
<td>250km</td>
<td>0km</td>
<td>125km</td>
</tr>
</tbody>
</table>

Finally, the specification of each datasets used to feed the networks can be found in table 2.3.

**Human errors and Irregularities**

This short section is here to outline that the labeled data contains mistakes. AgroParisTech claims an error rate below 10% but it is in fact most probably under 5%. On figure 2.12 we can see that the expert forgot to label the river and that the hem contains small bushes. In addition to that we can see that the tree in the middle of the meadows was considered as a meadow this is not an error from the point of view of the expert but it is still some information loss.

**Large Scale features and Unbalanced DataSet**

Finally, one of the biggest issue that needs to be addressed is the scale variation of the features we want to be able to detect. Our partners want to be able to find features smaller than 5 meters or 10 pixels yet, they also want us to find features as large as 1 kilometer or about 3000pixels. Also the dataset that they provided us was largely unbalanced with some classes only seen 250 times when others comes up 50,000 times on 85,000 images dataset as can be seen in table 2.4. This representation gap led to difficulty for the network to learn the under-represented classes.
Figure 2.12: Yellow: temporary meadow, Red: permanent meadow, green: broad leaves forest, Orange: hedge, Purple: hem, Blue: riparian grove
As evoked earlier in subsection 2.1.1, our networks only have a scope of 300 pixels; this is an issue as some features can be very large when others are significantly smaller. Also, some classes like urban parks solely rely on context, and small images often lack the range to grasp this context. At the time of this writing, we did not have time to test an implementation for this issue, yet, if we had had more time we would have tried to add supplementary context layers. Those layers could be made by taking the original image position and extracting larger images centered on it. Then the extracted images would be converted in grey scale: to reduce the size of the layers, and down-sampled to the size of our original image. Figure 2.13 shows the process designed to create those layers. This process is inspired by what Herranz et al. [18] did when they tried to develop new approaches to perform scene recognition, except that we are doing the opposite of what they did, we take larger images to get the context when they extract small patches. In our case, we have images of 300x300 pixels, so we could take images of 600x600 pixels, 1200x1200 pixels, and 2400x2400 pixels which would provide a range of 1.2 kilometers. This pyramidal approach should help the network to make abstraction of insignificant details depending on the size of features of interest. In regards of this, to use these layers, we believe that residual networks like RESNET-101 [19] should be used as they are able to skip unnecessary features and adjust their depth properly [20].
Unbalanced Classes

To overcome the unbalance in the class representation, some data augmentation was applied to the classes which needed it the most. However, because we had a very specific study we couldn't perform as much augmentation as we can usually do when we lack data. Here the issue is that the scale of our data is invariant in the way that we will always have a spatial resolution of 0.5 pixel/meter so it is not relevant to crop or upscale. Following this logic we cannot modify an image as context matters. This leaves us with two [21] [22] [23] things to play with: color tweaking and isometric transformations like rotations. Because some features are color sensitive and as we did not had enough time to investigates further in that direction, we chose to perform usual transformations that do not change the appearance of the image. In the end we can generate a total of 7 new images from the original image as shown in figure 2.14. Only those angles were used as they fit perfectly.

In doing so we managed to significantly reduce the representation gap between classes. Even if there is room for improvement as showed in table 2.4, the classes representation is more homogeneous. This is outlined by the drastic reduction of the max to min ratio, and a light diminution in the standard deviation. Figure 2.15 shows the loss variation on SegNet.
Figure 2.14: Transformations performed on a sample image

Table 2.4: Count of class appearance(App) in thousands, Before Augmentation (BA), and After Augmentation (AA)

<table>
<thead>
<tr>
<th>Class</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>Std Var</th>
<th>Max/Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA</td>
<td>1.7</td>
<td>66</td>
<td>0.63</td>
<td>1.6</td>
<td>0.14</td>
<td>9.3</td>
<td>4.7</td>
<td>1.5</td>
<td>4.4</td>
<td>51</td>
<td>48</td>
<td>4.1</td>
<td>0.78</td>
<td>19.3</td>
<td>12.2</td>
<td>0.2092</td>
<td>452</td>
</tr>
<tr>
<td>AA</td>
<td>12</td>
<td>63</td>
<td>3.2</td>
<td>5.5</td>
<td>0.7</td>
<td>12</td>
<td>5.5</td>
<td>7</td>
<td>3.8</td>
<td>54</td>
<td>49</td>
<td>7.2</td>
<td>2.5</td>
<td>21.5</td>
<td>16</td>
<td>0.1989</td>
<td>81</td>
</tr>
</tbody>
</table>

with and without dataset augmentation. As we could have foreseen, the loss is going down faster when training on the augmented dataset. This shows that the data augmentation efficiently improves the training without having to use sophisticated augmentation methods.

Table 2.5: Metrics after(AA) and before(BA) dataset augmentation after 150k iteration.

<table>
<thead>
<tr>
<th>Method</th>
<th>mIoU</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA</td>
<td>21.7</td>
<td>59.31</td>
<td>34.58</td>
</tr>
<tr>
<td>AA</td>
<td>21.36</td>
<td>47.42</td>
<td>37.0</td>
</tr>
</tbody>
</table>

The confusion matrices clearly show a performance improvement on lowly represented classes such as 141, 5111, 61, 5222, 5114 achieving +40% on the class 141. Unfortunately they also depict a performance loss on the classes on which the network used to be good at before augmentation. From previous observation with a 4 times smaller dataset we
remarked that it required at least 50 epoch to achieve convincing results in distinguishing the trees and the bushes for instance. Figure 2.17 shows the results obtained in that previous observation.

We could have expected better results [22] yet, it achieved its objective in the way that it helped the network reach better results on lowly represented classes. Also, it is possible that to learn the new classes the network uses more neurons than before and that we reach the limit of what SegNet can achieve. Deeper convolutional layers should be investigated. More details about the metrics used in 1.

As can be seen in table 2.5 the results are mitigated, it seemed that the data augmentation had a negative impact on the overall classification results with a decrease in the Global accuracy (G score) and the mean intersection over union (mIoU). Yet it seems that the mean class accuracy (C score: mean of the diagonal of the confusion matrices) was raised. This may indicates that the network with dataset augmentation may have needed more training time to reach the same number of epoch as the one without augmentation. The network

1More details about the metrics used here can be found in section 3.2
Figure 2.16: Confusion matrices at 150k iteration. Network trained without data augmentation at the top and with data augmentation at the bottom.

may also have overfitted, however, on figure 2.15 we do not see an overfitting behavior with divergence.

Summary

In the end, we have 2 datasets: one for the 1955 period, and one for 2015. The 1976 period doesn’t have any labeled data, so we will have to make our own (see chapter 5 for more information). The classes have been largely modified but are more suited to be learned by
This emphasis that we cannot always simply take some labeled data and to feed a neuronal network with it. The dataset must be carefully designed. Making sure to carefully balance the classes and maybe using some more aggressive form of data augmentation like GANs generated images or noised images.

Also we noted that a larger dataset does not obviously mean better results.

At that we can add that neural networks can’t do everything and sometimes other computer vision approaches may be better suited. Among the thing that should be explored, adding more input layers are the most interesting thing to do. Here are some of the layers that should be worth investigating in:

- High resolution Digital Elevation Model (DEM).
- High resolution Lidar imagery.
- Proper NDVI layer.
- Pyramidal Context Layers(HCL) as described in subsection 2.4.1.
Using both DEM and Lidar Imagery layer should prove extremely efficient to discriminate classes. About the labeled data, a even more machine oriented classification would be nice to have in a future work.
CHAPTER 3
SEGMENTATION ON STATE OF THE ART IMAGERY

The following explains how the image segmentation was performed and the issues at hand when trying to segment natural habitat.

Technical Approaches

To perform pixelwise labelisation, neural network were chosen as they are intensively used for this tasks nowadays [24] [25] [26] [27] [28] [29] [30] and outperform previous attempt at doing it [31] [30]. Multiple framework were tested: Caffe [32], TensorFlow [33] and Torch [34], and at the end, Caffe was chosen as it was simpler to use with a very large amount of models available. Moreover, even though it lacks the flexibility than can be found in Tensorflow, Caffe was designed to make classifiers so it was not necessary to consider something more complex. Also most of the auto-encoder implementations available in the literature were made with Caffe so it appeared as the best choice.

SegNet

SegNet [13] [35] was the first auto-encoder that we tried. Its simple architecture is an advantage as it can easily fit on our GPUs memory without sacrificing the input size too much. In comparison DeepLab v2 based on residual networks [19] will be seen in subsection 3.1.2 is taking 8.1Gb of ram for a single 300x300 pixels image per batch when we can fit 3 images like that on 4.7Gb using SegNet. SegNet also has the advantage to use a simple unpooling scheme which doesn’t rely on learned parameters. Indeed to perform the unpooling the network stores in memory the indexes used when it performs the Max-Pooling, as can be seen in figure 3.1[36] This is possible because SegNet’s architecture is symetrical as you can see on figure3.2. So when the network wants to perform the last
unpooling it is going to take the pooling indexes from the first pooling layer and so on. Furthermore, given the fact that the problem at hand requires to segment texture [3] [24] a ResNet [37] [38]-like encoder may have been a bit excessive [24] even more than VGG-16 [39] that is used within SegNet. Moreover, SegNet uses weighted losses which helps learning under-represented classes by assigning them strong weights and avoids overfitting by assigning lower weights to omnipresent classes [40]. Finally, the fact that it was well documented and came with a comprehensive guide made it our first choice despite its poor performances in comparison to other networks B.1 on standard dataset like Kitti [41]. However, studies similar to ours pointed out that at that time, state of the art approaches like deeplab do not produce better results than SegNet [42].

To train SegNet, we first added a test layer which was not present in the implementation provided online. This layer is critical, as it is used to know when the network starts to overfit and thus which are the most optimal weights. To know if the network overfits we monitor the value of the loss function on the test set. If it goes down it means that the network learns features applicable everywhere, if it goes up it means that the network’s performance is decreasing on the test set: It is no longer learning general features, but learns the training
set by heart. Ideally, to check if we are still training properly, we should process the whole test set, yet, because of the massive amount of time it would take, we chose to test on 600 images taken randomly in the test set. This does not ensure that all the classes will be present, but we noticed that it is usually enough to get all the classes in the batch and to have a good estimation of the network’s performance. To perform the training the default parameters of SegNet were left unchanged. The solver was left to Stochastic Gradient Decent (SGD) with a learning rate of 1e-3, the optimization runs for a total of 86 epochs with a step size of 28 epochs. After each step the learning rate is divided by 10. Batches of 8 images were simulated using the ”iter_size” parameter in Caffe. We had to simulate them as our GPUs RAM limitations impose that if we want to test and train in the same network we can only fit 1 image per batch on both the train set and test set. The size of the batches was set to 8 as it helps grabs more diversity from our dataset. With a single image per batch we have a low probability of getting a wide diversity of classes, thus the training is not optimal. Yet, we did not want to have to perform too many batches before performing the backpropagation as it may degrade the overall efficiency of the training and thus make
the training last forever. The following citation from Ian Goodfellow\textsuperscript{1} summarize well the issues at hand when trying to determine the proper size of a batch:

The size of the learning rate is limited mostly by factors like how curved the cost function is. You can think of gradient descent as making a linear approximation to the cost function, then moving downhill along that approximate cost. If the cost function is highly non-linear (highly curved) then the approximation will not be very good for very far, so only small step sizes are safe. You can read more about this in Chapter 4 of the deep learning textbook, on numerical computation \cite{43} When you put \( m \) examples in a minibatch, you need to do \( O(m) \) computation and use \( O(m) \) memory, but you reduce the amount of uncertainty in the gradient by a factor of only \( O(\sqrt{m}) \). In other words, there are diminishing marginal returns to putting more examples in the minibatch. You can read more about this in Chapter 8 of the deep learning textbook, on optimization algorithms for deep learning \cite{43}

The weights for the loss of our dataset were computed and we did not remove the mean of the training set to the train images as it was not done in the original implementation of SegNet. They recompute the weights of the batch norm at the end of the training in a way that the batch norm weights take into account the mean of train set. At the end, the network was trained with and without finetuning from a VGG-16 pretrained on ImageNet \cite{44}. The weights were downloaded from the visual geometry group’s website at Oxford university. Results can be seen in section 3.2. We did not have time to finetune from a VGG-16 trained on a more relevant dataset. A dataset more focussed on textures and large scale feature should most definitely be considered in further studies. Indeed, SegNet’s output is quite noisy, chapter 4 details on how to correct it.

\textsuperscript{1}From the Quora thread that can be find at: https://www.quora.com/In-deep-learning-why-dont-we-use-the-whole-training-set-to-compute-the-gradient
DeepLab

Even though SegNet is a state-of-art architecture for semantic segmentation, it carries CNN properties that are not relevant for segmentation such as translation invariance and poor resolution outputs. The first characteristic comes from the invariance of the convolution operator and results in poor segmentation borders: if the output is the same after a small input translation, the segmentation borders can not be accurate. The second is the result of successive convolutions and pooling which reduce the output size by a factor up to 32 for the most recent networks. The output resolution can not provide accurate segmentation results.

Both of these CNN properties result in poor segmentation accuracy. One way to address this issue is to upsample the segmentation output. Several techniques ranging from simple bilinear interpolation [40] to 'deconvolutional networks' [28] produces a segmentation output at the same resolution as the image. The most popular technique is the use of a symmetric decoder to the convolutional part of the architecture as in SegNet [13]. The architecture can be seen in figure 3.2. These additional layers learn the best convolutional filter parameter to upsample the low resolution segmentation output. This is a way to solve the low resolution output problem but not the translation invariance issue. A solution to this issue is to introduce context information from the convolutional part of the network to the deconvolutional one using skips (Figure 3.3). This allows to use the localization information in the feature map before the pooling layers erase them.

Another method is to fuse segmentation results from different scale processing as in the deeplab architecture. DeepLab is the current top segmentation architecture on the Pascal-VOC segmentation challenge [45]. It embeds several method into classic network architecture to get rid of the CNN properties that are not relevant for segmentation. Their contribution lies in the combination of the following tools:

- Convolution a trous [46](from the french: "with holes"): it computes responses
on layers at any resolution by convoluting the filters with pixels from the layers spaced by a rate \( r \) instead of convoluting contiguous pixels (Figure 3.3). When \( r=1 \), convolution a trous is the same as the classic convolution.

- Pyramidal pooling: the value of one pixel of a feature layers is inferred from the fusion of multiple convolution a trous. This enables to use information at different scale to correlate them. (Figure 3.4)

- Multi-scale processing: the latest version of DeepLab run three ResNet [19] on the same input image at different scales (0.5, 0.75, 1) in parallel and also fuse the output of each network to produce a final output. Each of these four outputs is compared to the ground truth segmentation and the four resulting loss are backpropagated. The loss on the scaled output are backpropagated through their respective network and the loss on the fused output is backpropagated through all the networks.

- Conditional Random Fields (CRF): the model integrates a long-range fully connected CRF over the segmentation image pixels to improve localization information based on the segment context. (Figure 4.1)

**Convolutions a trous**

This convolutional layers allow to correlate spatially remote features of the input while maintaining the same spatial dimensions. The correlation between remote features is necessary when the image holds multiple-scale elements such as trees or fields, or when there exist a strong dependence between elements. For example urban trees are defined by trees near an urban area. Figure 3.3 shows an example of atrous convolution in one dimension. Let \( x, y, w \) be the input, output and filter of size \( K \) respectively. Classic convolution showed
in Figure 3.3-a is defined as:

\[ y[i] = \sum_{k=1}^{K} x[i + k]w[k] \]

where the filter processes contiguous features of the input.

Atrous convolution showed in Figure 3.3-a is defined as:

\[ y[i] = \sum_{k=1}^{K} x[i + k \ast r]w[k] \]

where the filter processes input features spaced by \( r \). This convolution allows to compute correlation between features spaced by \( r \) units and the output has the same spatial dimensions as the input.

This solves the problem of losing spatial resolution and localization information induced by the classic convolutional-pooling model. In classic DCNNs, the correlation of non contiguous regions of an image occurs at the upper layers of the networks. This results from the increasing field of view as the depth increases. For example, after a pooling layer of size 2 with a stride of 2, the field of view if doubled. The same goes for convolutional layers depending on the convolution parameters. These layers reduces the spatial
dimensions of the layers while keeping the most informative features. This means that convolution at the upper layers of the network, filters information processed from a larger field of view on the input image than the first layers.

However, two issues results from this method. The first one is that localization information about the learned features is lost during spatial dimensionality reduction. At the upper layers, there is no information on where a feature came from on the input image. The second issue is that the output has a lower spatial resolution up to a factor of 32 in the most recent architecture. In semantic segmentation, this is an issue as we want the segmentation to map the initial image dimension.

The challenge is to process spatially remote features while maintaining the spatial dimension of the layers. As shown above, atrous convolution solve this problem.

**Atrous Spatial Pyramid Pooling**

Atrous Spatial Pyramid Pooling (ASPP) fuses the results of atrous convolution at different rates to infer the relations between multiple scale features.

Figure 3.4 shows that ASPP computes several atrous convolutions at different rate to process the input features at different scales. This produces several intermediary maps which are fused to produce a final feature map that contains multiple scale information on the input.

This allows to fuse multiple scale processing on a feature in a lighter way than the state-of-the-art methods such as image pyramid and encoder-decoder architecture as in SegNet.

Most of the methods to deal with multiple scale elements in the input require either to run the network several times on the same input or to maintain local information along the network which size is the order of several layers size. For example Image Pyramid, also used in the latest version of DeepLab [45] computes several scale of the input image, produce a segmentation map for each scale and then fuse them to produce a segmentation map for the initial spatial dimension. This method is still used but is memory hungry as
it requires a copy of the network for each scale. A less hungry approach to keep scaling information is to fuse feature maps with high resolution from lower layers of the network with high resolution and the upper feature maps with low resolution. This approach has not be implemented for semantic segmentation networks but is inspired from other networks such as FlowNet [47]. The first part of the network is an encoder that produces a feature vector in a reduced dimension. The second part of the network is a decoder that produces a segmentation map from the feature vector. The decoder is made of ’deconvolutional’ layers, one for each convolutional layer in the encoder. For each pair of such layers, the feature map from the deconvolution is concatenated with the feature map from the corresponding convolution and fed to the next deconvolutional module. The convolutional feature map holds resolution and localization information from the that the following layers may have
lost through feature dimension reduction. This method is lighter than Image Pyramid but still suffer from a memory overhead due to the concatenation of feature maps.

Spatial Pyramidal Pooling process input features at different scales in a way that is local to the layers. It does not require network duplication nor feature map concatenation which makes it lighter than the previous approaches.

**Implementation**

The SGD optimization is run with latest DeepLab model on a NVIDIA GT-1080 with a batch size of 10. The network is made of three parallel ResNet networks that process different scale of the input image. The segmentation is obtained after fine-tuning from the DeepLab segmentation model on Pascal-VOC dataset[48] for 150 000 iterations.

---

**SegNet improvement: FlowNet2 Inspired Network**

This Network was created as we felt that SegNet failed to grab context at large scale due to its architectural limitations. As it can be seen in figure 3.2, when the network performs upsampling it only relies on the features map that have already been upscaled. This creates two main issues:

- When the networks reaches the unpooling layers it is very deep: for instance in deconv2.2 the data went through 22 3x3 convolutions, 5 maxpooling and 3 unpooling. This may prevent it from assimilating simple features.

- As we upscale features maps from maps that were already upscaled we build upon potentially wrong features maps. This means we don’t perform as good as we could as we do not try to correct the error after each upsampling step.

The solutions implemented to solve those issues are inspired by what was done in FlowNet2 [47]:
Residual unpooling approach

One way to reduce the deepness of the network is to do a residual network like approach but applied to the unpooling part. To do that the best way it to concatenate the opposite feature map to the freshly upscaled map. In doing so, we provide the Network with a map of the same size: so that it doesn’t need to be reshaped, and we give him higher scope interpretation: For instance if we take the deconv2.2, the range of the features that are being concatenated are 15 pixels (2 conv3x3 + 1 maxpool3x3 + 2 conv3x3). The implementation of this architecture can be seen in figure 3.5.

Multi-loss training & intermediary segmentation maps

In order to correct the error at each upsampling step, intermediary loss have been added to the network. Before each upsampling + convolutions a SoftmaxWithLoss layer was added. After each of this layer, Caffe performs the backpropagation which means that it adjusts the weights before performing the upsampling and thus corrects the errors.

In addition to that, the softmax layer output is concatenated to the output of the convolution placed after the upsampling layer. This provides the network with its current estimation
of the segmentation map which also help reducing the depth of the network. Figure 3.6 details the correction process: to perform the correction, the results from the last convolution before the upsampling is taken and upsampled using a deconvolution layer. The result of this is then compared in the "softmax-with-loss" layer to the ground truth which has been resized to the proper scale by the downsampling layer. This not only gives us the current estimation of the labels by the network but it also perform the backpropagation on all the layers before the "softmax-with-loss". The intermediary estimation of the label is then concatenated to the upsampling results after it has been convoluted.

Figure 3.6: New detailed upsampling method to correct error at intermediary steps

The complete architecture can be seen on figure 3.7.

Finally, the two first solutions were merged which gives us the architecture showed in figure 3.8:

The training of this architecture is similar to the one of SegNet except that we have 5 more losses. The test layer was also added. Like SegNet, we do not substract the mean of the train set and we finetune from VGG-16 trained on ImageNet. The results can be found in the section 3.2.

Results

This section details the results of the above architectures. As could have been foreseen, DeepLab outperformed both SegNet and our architecture. It also seems that SegNet with
its current learning parameters takes more time to converge than DeepLab.

To evaluate the quality of the segmentation, 4 metrics have been chosen based on the recommendation formulated in [49], the confusion matrices for each class are also gener-
ated as suggested by [50]:

- Confusion matrices: Those matrices help us understand the confusion within the network by displaying how each class is classified by the network. Note that they do not reflect the indecision of the network but more wrongly learned features; to grasp the indecision of the network we would have to measure the entropy before the softmax.

- The Global average accuracy or G score: It is the mean of the diagonal terms of a confusion matrix.

- The Class average accuracy or C score: This scores is the percentage of properly classified pixels.

- The mean Intersection over Union or mIoU: This score which is also know as the Jackard Index is a commonly used in segmentation benchmarking. This score comes in complement of the previous scores as it penalizes the false positive.

- The BF score: It is used as a complement to the mIoU score as according to Csurka [49] and Long [40], it is not always conform to human judgement on segmentation quality. So they introduced the BF score which is a semantic contour measure. The addition of both the Jackard Index and BF score gives a good evaluation of the segmentation quality [49].

Also, we could have considered the BJ score [50] but as it is not much used yet, it was judged not relevant to used is here. Indeed it would give little more insight than the mIoU and BF score as it is meant to replace them. Also the two previous scores help us evaluate the quality of the tested architecture and their relevance in our study as we can compare their scores to the one they achieved on other datasets. Table 3.1 summarize the best score achieved for each networks, 3.2 list the class accuracy for each networks and figure 3.9 shows the confusion matrices for each networks. Finally, we did not consider the BF score
Table 3.1: Segmentation scores for the different architectures. At 150,000 iterations. *DeepLab was tested on a smaller test set due to network limitation; it is enough to extract a tendency but those results are not entirely accurate.

<table>
<thead>
<tr>
<th>Method</th>
<th>mIoU</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>SegNet</td>
<td>21.37</td>
<td>37.01</td>
<td>47.43</td>
</tr>
<tr>
<td>DeepLabV2 ResNet x3</td>
<td>45.49*</td>
<td>52.52*</td>
<td>88.2*</td>
</tr>
<tr>
<td>SegNetMod (ours)</td>
<td>10.2</td>
<td>21.41</td>
<td>43.75</td>
</tr>
</tbody>
</table>

Table 3.2: Per class mIoU score at 150k iterations. *DeepLab was tested on a smaller test set due to network limitation; it is enough to extract a tendency but those results are not entirely accurate.

<table>
<thead>
<tr>
<th>Method</th>
<th>5114</th>
<th>212</th>
<th>5113</th>
<th>63</th>
<th>211</th>
<th>33</th>
<th>5222</th>
<th>52</th>
<th>32</th>
<th>5111</th>
<th>221</th>
<th>61</th>
<th>141</th>
<th>111</th>
<th>Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>SegNet</td>
<td>0.5</td>
<td>53.7</td>
<td>0.1</td>
<td>6.99</td>
<td>0</td>
<td>6.95</td>
<td>19.69</td>
<td>1.64</td>
<td>1.66</td>
<td>19.4</td>
<td>30.44</td>
<td>25.48</td>
<td>16.80</td>
<td>43.44</td>
<td>93.7</td>
</tr>
<tr>
<td>DeepLabV2 ResNet x3</td>
<td>31.06*</td>
<td>91.81*</td>
<td>50.19*</td>
<td>4.81*</td>
<td>0*</td>
<td>17.18*</td>
<td>23.34*</td>
<td>71.89*</td>
<td>0*</td>
<td>82.6*</td>
<td>67.5*</td>
<td>70.0*</td>
<td>32.3*</td>
<td>60.3*</td>
<td>77.73*</td>
</tr>
<tr>
<td>SegNetMod (ours)</td>
<td>1.63</td>
<td>64.5</td>
<td>0</td>
<td>2.3</td>
<td>0</td>
<td>7.25</td>
<td>0.3</td>
<td>0.94</td>
<td>0.14</td>
<td>8.4</td>
<td>12.4</td>
<td>7.5</td>
<td>3.36</td>
<td>0</td>
<td>23.0</td>
</tr>
</tbody>
</table>

As you can see, only DeepLab v2 managed to get good scores on the dataset. Both SegNet and our modification of it performs poorly. It is most probably due to a learning issue, Segnet uses weighted losses which may have slowed the learning of the most represented classes. The confusion matrices 3.9 tend to confirm this hypothesis: the prediction of the class 33 (the bushes) gathers all the tree related classes (5111, 52, 63, 5114, 5222). Those classes are visually hard to differentiate and DeepLab atrous convolution technique may help it grab more context than the one of SegNet. Or it could be that the encoder in VGG reaches its limits or need more time to train: in earlier observations on a previous iteration of the dataset with less data we had much better results for SegNet. A weird observation is that DeepLab and our modification of SegNet have issues identifying the mask, when SegNet score 93%. Adding a texture dedicated 4 layers network could be a good way to boost the performance of the encoders on trivial textures.

As for the performance of our modification of SegNet, it seems that it is very noisy, this
Figure 3.9: Confusion matrices for the 3 architectures.
Table 3.3: Evolution of the segmentation scores over time.

<table>
<thead>
<tr>
<th>Method</th>
<th>60k iter</th>
<th></th>
<th></th>
<th>90k iter</th>
<th></th>
<th></th>
<th>120k iter</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mIoU</td>
<td>G</td>
<td>C</td>
<td>mIoU</td>
<td>G</td>
<td>C</td>
<td>mIoU</td>
<td>G</td>
<td>C</td>
</tr>
<tr>
<td>SegNet</td>
<td>17.92</td>
<td>31.60</td>
<td>50.56</td>
<td>16.97</td>
<td>31.87</td>
<td>45.77</td>
<td>20.17</td>
<td>32.31</td>
<td>46.22</td>
</tr>
<tr>
<td>SegNetMod (ours)</td>
<td>2.59</td>
<td>7.54</td>
<td>15.47</td>
<td>6.74</td>
<td>16.69</td>
<td>42.17</td>
<td>7.69</td>
<td>21.41</td>
<td>32.86</td>
</tr>
</tbody>
</table>

(a) Accuracy at training time on the 600 random images from our test set.

(b) Loss at training time on the 600 random images from our test set.

Figure 3.10: Accuracy and loss of the training of SegNet (blue) and our modification of SegNet (green).

could be due to a lack of training or the fact that we chose to concatenate the layer instead of fusing them.

We also evaluated the evolution of the training in table 3.3 or how fast the network learns. Every 30,000 iterations, or 3 epochs the networks were tested and the evaluation metrics computed. This helps us evaluate the performance of each network as they train. Additionally, it spots the network that requires less computations to be operational.

Those results are not consistent with what the plot of the loss and the accuracy on the test set as can bee seen in figure 3.10 our modified version of Segnet should be more accurate than the standard SegNet. It could be due to some issues with the “compute_bn_normalization.py” script provided with SegNet that does not perform as it should on our architecture. It could also be tied to some problem with the frequency balancing of the classes.

Note that initially we planned on doing 90 epochs yet, in the end we were not able to it as it would have taken 1 month and a half to do it on our Nvidia k20c.
Finally, we also evaluated the computational performances of the network as listed in table 3.4. We took in consideration the time required to perform 1 batch at training time, the memory required to train and test and the size of the models. This was done to help consider the segmentation performance in regards to the computational costs. Those results were obtained using a Nvidia GeForce 1080Ti.

<table>
<thead>
<tr>
<th></th>
<th>batch time</th>
<th>memory train</th>
<th>memory test</th>
<th>model size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SegNet</td>
<td>1.2s</td>
<td>1.5gb</td>
<td>0.794gb</td>
<td>113Mb</td>
</tr>
<tr>
<td>DeepLabV2 ResNet $\times 3$</td>
<td>3.6s</td>
<td>7.9gb</td>
<td>4.8gb</td>
<td>501Mb</td>
</tr>
<tr>
<td>SegNetMod(ours)</td>
<td>1.4s</td>
<td>1.7gb</td>
<td>1.05gb</td>
<td>147Mb</td>
</tr>
</tbody>
</table>

Table 3.4: Computational performance evaluation.

Figure 3.11: Segmentation results: at 150k iterations.

All in all it seems that SegNet and our modification of it lack of training, to reach their full potential on the dataset. A previous attempt with a smaller dataset seems to corroborate
Table 3.5: Segmentation scores showing the border effect impact on the classification. Results at 150k iterations without data augmentation.

<table>
<thead>
<tr>
<th></th>
<th>mIoU</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>SegNet Full</td>
<td>23.2</td>
<td>34.6</td>
<td>59.31</td>
</tr>
<tr>
<td>SegNet Center</td>
<td>22.7</td>
<td>32</td>
<td>70.16</td>
</tr>
</tbody>
</table>

the idea that 14 epochs is not enough. Figure 3.12 shows what a more accurate classification map looks like. It was acquired after 50 epochs with SegNet. Figure 3.13 shows what we have at the moment with SegNet. At the time of this writing we were not able to compute the reconstructed segmentation map using DeepLab due to hardware limitations here at GeorgiaTechLorraine. As for DeepLab it performs well with a limited training time but it is much slower than SegNet when inferring: On the partner’s school GPU it took about 9 hours to segment 14,000 images when SegNet can infer the same amount of images under half an hour.

**Discussion**

Visually, the classification results can be noisy on certain type of classes. If the tree related classes are precise and sharp it is not always the case with the others. Indeed when the networks gets confused between meadows and fields for instance it creates some high frequence noise as can be seen on the segmentation maps in figure 3.11. The removal and correction of this noise are addressed in chapter 4. Also we can see that most of the classification errors occur on the side of the images. If we only take the center of the images the estimation of the label by the networks are better as shown in table 3.5 and figure 3.14. This correlates with the need for the network to have more context than what it currently has, as explained in subsection 2.4.1.

As we can see when we remove the border the C score is higher meaning that a higher number of pixels are properly classified. Also we can see that the G and mIoU score remain almost unchanged.
Figure 3.12: Reconstructed segmentation map after 500k iterations with a 24k images dataset using SegNet.
Figure 3.13: Reconstructed segmentation map after 150k iterations with a 85k images dataset using SegNet.
Figure 3.14: Evaluation of the border effect impact on classification.
As said in the previous chapter the raw segmentation results tend to be noisy and contains mistakes that can be easily solved using expert’s experience. For instance, the network often have a hard time differentiating meadows and farm land, and can sometime classify a small patch of pixels as meadow in the middle of a field. In a case like that it is fairly easy to detect this and correct the classification of the patch. This chapter will also discuss the expansion of classes based on context. As said in section 2.2 the number of classes have been reduced as they were inducing indecision in the network.

Noise correction

To solve the noise issues two approaches have been tested:

- Low pass filter and Classification correction based on expert experience.
- Application of Conditional Random Fields from DeepLab.

Conditional Random Fields

A Conditional Random Field (CRF) is a standalone graphical model over the image used for region-wise or pixel-wise segmentation. Here, it is used as a post-processing tools over the last feature map computed by the network to produce a pixel-wise segmentation output instead of using the classic softmax layer. It let us erase noisy segmentation results and exploit localization information to make the segmentation more accurate and recover object boundaries with a higher level of details. The post processing is done with the CRF implementation of [51] which proved to be the most efficient for a fully connected CRF. Their contributions lies in the use of convolutional gaussian filter to implement the message
passing step of the algorithm which complexity is in \( O(N^2) \) with \( N \) the number of pixels of the image. This method reduce the complexity to \( O(N) \) by observing that the message passing step can be expressed as a convolution with a Gaussian kernel.

The CRF is defined by \((I, X)\) with \( I \) a random field defined over a set of variables \( \{I_1, \ldots, I_N\} \) and \( X \) a random field defined over a set of variables \( \{X_1, \ldots, X_N\} \). Let \( \mathcal{L} = \{l_1, \ldots, l_k\} \) set of label which is the domain of each \( X_i \). The CRF \((I, X)\) is characterized by a Gibbs distribution with the form \( P(X|I) \propto \exp(-E(X)) \) with \( E(X) \) an energy function over the pixels defined as follow:

\[
E(X) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j)
\]

The maximum a posterior labeling of the random field is \( X^* = \text{argmax}_{X \in \mathcal{L}^N} P(X|I) \) which is equivalent to minimizing the energy \( E(X) \).

\( E(X) \) is made of a potential over individual pixel features \( \psi_u \) and pairwise potential \( \psi_p \) between all pairs of pixels of the image. Features can incorporate shape, texture, location and color descriptors. The post-processing uses the feature descriptors of the last feature.
map of the network. The unary potential is simply

$$\psi_u(x_i) = -\log(p_{\text{network}}(x_i))$$

with $p_{\text{network}}(x_i)$ is the label assignment probability at pixel $i$ as computed by the network. The potential decrease as the network’s certainty of the labeling increase.

The pairwise potential is made of two gaussian $G_1$ and $G_2$:

$$\psi_p(x_i, x_j) \propto G_1 + G_2 \propto w(1) \exp\left(-\frac{|p_i - p_j|^2}{2\theta_\alpha^2}\right) + \exp\left(-\frac{|I_i - I_j|^2}{2\theta_\beta^2}\right) w(2) \exp\left(-\frac{|p_i - p_j|^2}{2\theta_\gamma^2}\right)$$

The proportionality coefficient between the pairwise coefficient and the gaussians is label compatibility function $\mu(x_i, x_j)$ that introduces a prior of the knowledge of the labels relations in the image. A simple example is given by the Potts model, $\mu(x_i, x_j) = [x_i \neq x_j]$. It means that only potentials between pixels labeled differently are taken into account into the potential. $G_1$ is the appearance kernel and models the observation that nearby pixels with similar color are likely to be in the same class. The degree of nearness and similarity is controlled by the meta parameters $\theta_\alpha, \theta_\beta$ chosen with grid search. The potential value is added to the energy function only for pixels labeled differently. This can be interpreted this way: if pixels are labeled differently while being nearby and having the same color, the potential is high. The optimization would tend to label these pixels with the same label going with the intuition that nearby label with the same color are likely to belong to the same class. If pixels are far away and hold different colors, the negative exponential leads to a low potential going with the intuition that they probably have different label. It appears that this label compatibility function is not the best as a way to minimize the the pairwise potential would be to set all the pixels with the same label. However, this effect is balanced by the unitary potential of each pixel. The second gaussian $G_2$ controls the smoothness of the segmentation map to erase spurious segmentation: nearby pixels have a high chance to
Table 4.1: Segmentation scores using CRFs as segmentation enhancement scheme.

As can be see on figure 4.2 and table 4.1 the CRFs successfully improve the classification results. All in all it seems that if we use DeepLab + CRFs we will not need to use filtering algorithms as the output is smooth.

**Local Corrections**

The following approaches try to improve the classification results based on human experience and using reconstructed classification results. Reconstructed classification results are obtained after scrapping the test map classifying them and reconstructing them.
Noise Removal

The aim here is only to remove the high frequency noise generated by the classification. Figure 4.3 shows the kind of noise that we can find on the meadow channel of the classification for instance. To perform the filtering all the classification maps are being treated separately.

![Figure 4.3: Conditional Random Field on the last feature map](image)

On each map a custom made gaussian kernel is applied to dim the high frequencies: it acts like a low pass filter and weakens the value of isolated pixels. The map obtained is then normalized and a thresholding is applied: it put all the value below 0.5 to 0 and the remaining to 1. The results is then masked by the original classification maps: this is done so that the filter applied at the beginning does not make the classification leak on other classes. After that we compute the contours of the classification map and look for contour with an area below 5 square meters: we consider that below 5 square meter it has
to be a classification error as we do not look for feature that are that small. Thus, if an
countour is below 5 square meters its shape is replace by 0. Once the previous is done on
each classification map we rebind the classes between each others. If there is a conflict
i.e. a pixel have multiple classes assigned: the highest value is kept. Finally, the previous
operations may have left unclassified areas in the image: to correct them a mask growth
is performed. A neighboring graph of the unclassified is made, based on that the highest
value of their neighbors is applied to them. The previous operation is iterated until there
is no pixel left unclassified. The whole filtering operation is performed in approximately a
minute on a 9000x9000 pixels matrix. Figure 4.4 shows the image results before and after
filtering for a single map.

![Forest map before filtering](image1.png) ![Forest map after filtering](image2.png)

Figure 4.4: Results of the filtering process. On the left the segmentation of the forest
without filtering, on the right segmentation map with filtering

**Error Correction**

The error correction is performed after the previously described algorithm. The idea is that
this algorithm relies on feedback that our expert gave us. When the expert looked at our
map, he noticed that sometimes it would classify patches of pixels under 25 square meters
as fields in a the middle of a meadow and vice versa. The same behavior can be seen in
pond that sometimes have small patches of meadow in the middle of them. This behavior
can be easily corrected in some case using graphs.

To correct the misclassification, the layout of the classification map is transcribed into a graph. To do this an algorithm parses all the pixels of the image and creates blobs of continuous pixels of the same value. In this graph the vertexes are the number of the blobs and the edges links blobs that are neighbors. The vertex contains additional information such as the class of the blob, its list of pixels and its surface size. Using this data we are able to discriminate small isolated blobs (blobs with a single neighbor below 25 square meter) and based on a set of rule say whether or not they are a classification error.

Results of this can be seen in table 4.1. Obviously, with the previous approach we cannot correct classification errors on the side of the fields for instance, but this is not meant to replace the classification in it self but correct things that we are sure are mistakes.

**Noise Reduction Results**

The results are purely qualitative, as the output maps from SegNet were not good enough which means the metrics would not be relevant. Figure 4.6 shows the results obtained by the filtering algorithm: the high frequency noise is removed without destroying the edge of the segmentation. Also, the small patch were not erased out, this means that the filtering does not create data loss. Figure 4.7 illustrates the error correction algorithm. Here the rule that was set is: every isolated label below 100 square meter is replaced by its surrounding label. In the future more complex rule should be implemented.

As discussed earlier because of Network limitation we could not recreate a complete segmentation map using DeepLab and thus test those algorithm on proper results. Also, because DeepLab shows little to no noise in its segmentation the filtering algorithm may be useless, yet the error correction one may be of use.
Figure 4.5: Original Full SegNet’s segmentation map.
Figure 4.6: Full SegNet’s segmentation map after filtering.
Figure 4.7: Full SegNet’s segmentation map after error correction.
Class Enhancement

Building on the neighborhood graph that was made in subsection 4.1.2 it is also possible to extend the classes as they were shrunk in section 2.2. There are two ways of doing it using the context provided by the previous graph, or using the raw definition of the classes.

Extracting New classes From Expert Experience

This technique rely on the strict definition of a class. It can be used to extract hedges, water flows, lakes, and mixed trees. As mentioned earlier, a hedge is a line of trees with a width smaller than 10 meters and a length longer than 50 meters with interruption smaller than 1.5 meter. Yet, we cannot apply this definition ”as-is” for multiple reasons: the first one is that the classification may leak a bit around the trees so instead of using 20 pixels (0.5pxl/meter) as width threshold we will use 30. For the interruption the threshold will be set to 5 pixels and the length threshold will be set to 100pixels. Also, this is a general rule when the expert classifies the land use, s/he does not take a rule and makes sure that it is exactly under 10 meters or longer than 50meters which is why we will keep higher threshold that the one indicated. Rivers, lakes and ponds can be discriminated based on their shape. A river is a water body with a high width to length ratio. Mixed trees can be identified by analyzing the coniferous tree density within a deciduous forest.

Only the hedge detection algorithm was made because of hardware limitations and time constraints. To perform the identification of the hedges, the coniferous forest, deciduous forest and bushes classes are merged. Then a succession of morphological transformations are applied: the map is eroded of half of the width threshold(15pxl) this map is then inverted and masked by the original map. This give us all the tree patches in the map with a width below 30meters, but it also keeps scattered trees. To remove them the map is eroded of 5 pixels and dilated back of 10 pixels. The image is then masked by the original image to prevent leaks. Note that the dilatation is two times larger than the erosion to extend the
reach of those patches as the first extraction process truncated them a bit. After that the skeleton of the image is extracted. From there the remaining pixels’ positions are stored in a list and a length calculation algorithm is used to find lines longer than 100 pixels with interruption shorter than 5 pixels.

Discussion

At the time of this writing, the computer vision enhancement is still in development. The CRFs allowed to increase the classifications results when used with DeepLab. Yet, their impact is non negligible with a performance jump. The filtering algorithms displayed good looking results. Once applied to correct feature map and used with a proper set of rules, they should be able to reproduce behavior similar to the one of an expert.
CHAPTER 5
LEARNING ON FAKE DATA

This chapter will focus on the gray scale image analysis. More precisely, it explores the generation of mock data from state of the art imagery to help networks get higher accuracy when finetuning from them. The main reason why we want to do that is that in our study has a limited amount of labeled data from the 1955 and we cannot train and test from such a small amount of data. To do so, RGB images will be modified using Photoshop scripts to make them look like 1955 images. Doing this gives us a larger dataset to train the network on. The previously trained network will then be finetuned on the 1955 data. Comparing the approach with and without finetuning will provide insights on how efficient our method is. Such method already showed promising results, the group developing TrimBot [52], made 3D rendering of plants using blender to extract the ground truth easily. They uses the obtained images to pre-train their networks and it seems to produce satisfying results.

Technical Approaches

Creating fake images

To create images matching the desired time period, a script was made using Adobe Photoshop.

Making 1976 imagery

Each transformation steps from RGB imagery to 1976 gray scale imagery are detailed below. They are important as they help understand the differences between the two. The first step is to convert the image into gray scale image. As can be seen in figure 5.1 the tons balance are completely different between the 2 periods it needs some fixing. Also the
details are too sharp on the 2015 image and there is no grain. To do that standard Photoshop filters are applied. To reduce the precision of the image two types of blur are applied: a Gaussian blur with a kernel of 3, and a crystallization filter is applied to a duplicate of the original image is applied whose opacity is then turned down to 10%. The Gaussian blur smooths the textures while the crystallization layer creates some small deformations on the buildings. Results can be seen in figure 5.2. To help reestablishing the color balance a first curve filter is applied as we can see on the 1976 picture the light tones are much more present and the dark tones are weaker. So we set the curve effect to do that as illustrates figure 5.3. Other filter will be applied later on to correct the tones but this one had to be applied before the grain effect as it had a negative impact on them. Multiple 7 percent noise filter are then applied to an duplicate of the original image whose opacity is the turned down to 50percent. A slight gaussian blur then is applied on the noise to make it look bigger and

![Figure 5.1: Left RGB, center gray scale conversion, right objective.](image)

Gaussian blur with a kernel of 3, and a crystallization filter is applied to a duplicate of the original image is applied whose opacity is then turned down to 10%. The Gaussian blur smooths the textures while the crystallization layer creates some small deformations on the buildings. Results can be seen in figure 5.2. To help reestablishing the color balance a first curve filter is applied as we can see on the 1976 picture the light tones are much more present and the dark tones are weaker. So we set the curve effect to do that as illustrates figure 5.3. Other filter will be applied later on to correct the tones but this one had to be applied before the grain effect as it had a negative impact on them. Multiple 7 percent noise filter are then applied to an duplicate of the original image whose opacity is the turned down to 50percent. A slight gaussian blur then is applied on the noise to make it look bigger and
less sharp as can bee seen in figure 5.4 Finally the color tones are modified using a level filter to restrain the spectrum of the image in the dark tones and amplify them in the light ones. A curve filter is applied to saturate the tones and a brightness and contrast set at max contrast is used to make the light tone brighter. Figure 5.5 shows the final results. As we can see, the tones are not perfectly matching and the grain is too thin yet we did not managed to get better results with Photoshop. More tweaking is most probably needed to make the images more 1976 alike but it should be sufficient to train our network. After all if it was perfect we would not need to finetune. Note that the difference of tones and contrast in the small forest sample is due to the fact the the trees were young and small in 1976 and casted almost no shadow.
Figure 5.3: Left before, center after first color correction, right objective.
Figure 5.4: Left before, center after grain application, right objective.
Figure 5.5: left before color correction, center final result, right objective.
Making 1955 imagery

A similar process was applied to create fake 1955 images: figure 5.6 shows the result.

![Figure 5.6: Left fake 1955 image, right real 1955 image.](image)

Training

Seven different networks were trained to evaluate the impact of using finetuning from heavily modified data:

- **(1)** SegNet 1955: Training only on a limited dataset from 1955.

- **(2)** SegNet BW: SegNet trained only a simple black and white conversion of the 2015 dataset.

- **(3)** SegNet fake55: SegNet trained only on the processed 2015 images to make them look like 1955 images.

- **(4)** SegNet fake76: SegNet trained only on the processed 2015 images to make them look like 1976 images.


•(7) SegNet fake76+ft+1955: Finetuning from SegNet fake76 on the limited data set from 1955 used in SegNet 1955.

The results from (1) will be used as benchmark for the other networks. With (2), (3) and (4) we investigate if the network is able to apply what he learned on fake images to real images. It will also show the importance of having realistic fake images. (5), (6) and (7) aims to demonstrate that learning from pre-trained weights helps the network perform better after finetuning on the proper period imagery. It also aims to show the relevance of having realistic fake images as pre-trained weights.

Note that if not specified, the networks’ encoders were finetuned from imagenet. To perform the training due to material limitation we used a partner school GPU farm and trained all those networks on 1080Ti for a 150,000 iterations.

Results

Table 5.1 displays the metrics for each network and figure 5.7 displays the confusion matrix of each network. We can see that, SegNet when trained on the real 1955 images performs relatively well. However, the others fails completely. This is most probably due to the

<table>
<thead>
<tr>
<th></th>
<th>mIoU</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>SegNet 1955</td>
<td>21.72</td>
<td>38.9</td>
<td>77.8</td>
</tr>
<tr>
<td>SegNet BW</td>
<td>2.5</td>
<td>12.2</td>
<td>9.18</td>
</tr>
<tr>
<td>SegNet fake55</td>
<td>1.55</td>
<td>10.71</td>
<td>2.79</td>
</tr>
<tr>
<td>SegNet fake76</td>
<td>4.42</td>
<td>14.76</td>
<td>10.76</td>
</tr>
</tbody>
</table>

Table 5.1: Segmentation scores on 1955 for the different training set. After 100k iterations.
Confusion matrix for the network trained on real 1955.

Confusion matrix for the network trained on fake 1955.

Confusion matrix for the network trained on fake 1976.

Confusion matrix for the network trained on fake Black and White.

Figure 5.7: Confusion matrices for the different training set. They were all tested on the same test set.

learning issues discussed in chapter 3. The DataSet of the real 1955 images contained 12,000 images when the other contained 65,000 images; this means that the network trained on real 1955 images had 100 epochs to learn when the others only had 18. This could explain why the other networks perform poorly on the train set but we cannot know for sure. In future experiment we will reproduce this experience but this time using DeepLab and see if it works or not.

As we can see in table 5.2 fine-tuning from the fake images of 1955, 1976 and black and white give better results than simply using the encoder from ImageNet. This shows that fine-tuning from fake images is worth investigating. Here we can suppose that the fact that the network had to relearn the last convolution layer dissipated of the impact of this method. In further study it would be nice to have similar datasets for both periods. In addition to that, with so few training iterations it is hard to say if to conclude on which of the fake data was the most effective as it is very noisy. This is an early conclusion as we did not had the computational resources required to train all those network in time. A complete training of

78
Table 5.2: Evolution of the segmentation scores over time. Note that, due to the loss of 2 classes in 1955, the last convolution of the decoder had to be learned from scratch. The metrics presented here are the actual score there was no error in the label or any mismatch between groundtruth images and predictions. Note that 1,000 iterations is equivalent to 1 epoch.

<table>
<thead>
<tr>
<th>Method</th>
<th>10k iter</th>
<th>15k iter</th>
<th>20k iter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mIoU</td>
<td>G</td>
<td>C</td>
</tr>
<tr>
<td>SegNet 1955+ImNet Enc</td>
<td>0.550</td>
<td>3.93</td>
<td>1.76</td>
</tr>
<tr>
<td>SegNet 1955+F55</td>
<td>0.82</td>
<td>4.19</td>
<td>1.78</td>
</tr>
<tr>
<td>SegNet 1955+F76</td>
<td>2.09</td>
<td>11.33</td>
<td>3.24</td>
</tr>
<tr>
<td>SegNet 1955+BW2</td>
<td>0.63</td>
<td>9.04</td>
<td>3.32</td>
</tr>
</tbody>
</table>

A minimum of 100 epochs should be considered here the fine-tuning stopped at 20 epochs which is a bit too premature to draw a definitive conclusion. The poor performances of the networks when put in regard of the performance of SegNet using RGB-NIR images is somewhat understandable. Indeed, we ask those networks to be able to recognize all the classes with a single channel when in 2015 they are having trouble with 4. Our expert also told us that classifying those images was not an easy task for a human and maybe we should consider reducing the resolution of our images. Also the dataset from 1955 is much smaller: 12,000 images when for 2015 we have up to 85,000 images.

Summary

The second set of experiment with the SegNet network seems to indicate that pre-training on fake data is a relevant technique when the dataset is limited. It could mean that problem related fine-tuning weights may be better suited than commonly used weights such as ImageNet. This experiments should be completed with a better architecture such as DeepLab to get rid off any bias the poor performances of SegNet may have induced. As discussed earlier, our GPUs have 4.7gb of RAM which is why we did not used DeepLab in the first place (DeepLab requires 8.1Gb of RAM at training time and 4.8Gb of RAM at testing time).

Indeed, we cannot get much from the first experiment except that it does not work with those training parameters. Thus, we must further study the relation between the fake data...
generation quality and the pre-training performance: maybe our images were not generated properly. In further work we could teach a network to generate such images as they did with fast neural style [53]. Furthermore, to evaluate the quality of the generated images, we propose to use our experiment. A network is trained and tested on the real data; then, the metrics extracted from the test are used as references. A second network is trained on the fake images and tested on the real images: if the fake images are close to the real one then the metrics will be similar, if not they are not, the metrics will lower.

Finally, those networks rely solely on images’ textures to differentiate the classes. This property could make them an excellent starting point for our RGB-NIR imagery network that, based on our observations, tend to rely too much on colors to distinguish classes. Because our project is constrained in time we could not carry further in that direction, however, in further studies, fine-tuning the RGB-NIR network from gray scale imagery should be investigated.
CHAPTER 6
CONCLUSION

This early result shows that machine learning is a promising approach to perform fine-scale segmentation of habitats. Some approaches provided solid results; even though they were only working on a subset of classes, the results obtained could already proved helpful for biologists. Indeed, even if they do not provide a perfectly accurate segmentation map, their outputs could greatly assist cartographers and biologists as it outlines the major features in the environment almost flawlessly.

When we made the dataset we learned multiple things:

- Neural networks cannot yet distinguish everything that the human can. In our case the network needed two things in order to recognize most of the classes: high resolution texture, and a large scope. With current architectures, feeding the network with large images is memory-intensive. To overcome this issue, we proposed a new data layer called Pyramidal Context Layers. Those layers provide the network with multi-scope contextual information while keeping the original image at the same size, thus preserving the resolution needed to extract information from the textures.

- Furthermore, we learned that adding more training data into the dataset does not necessarily increase the overall classification results. To the contrary, adding too much information slows the learning process as the network sees multiple times images that are relatively similar to the others. In our case, only DeepLab and 3 residual networks managed to learn correctly the dataset in a relatively short time (1 week). To prevent further issues the images used in the dataset will be more carefully selected.
When we evaluated the segmentation performances of current state of the art architectures DeepLab achieved the best results.

- As could have been foreseen, its 3 ResNets performed 88% of correct predictions using the complete dataset. It not only outperformed SegNet in term of raw segmentation performances but it also learned much faster.

- Additionally, we tried to make our own network: it did not achieved the results we could have hoped for. Yet, based on the previously drawn conclusion we may have to retrain this architecture with a lighter dataset. An other improvement could be to fuse the residual unpooling instead of concatenating them.

- All in all, despite that it is memory-intensive and computationally heavy, DeepLabv2 3×ResNet is by far the best architecture that we tested.

Furthermore, DeepLab gets even better when coupled with CRFs with which it reaches 91.3% pixel wise labeling accuracy. To improve the classification we also proposed to implement classical computer vision techniques based on a rule set defined by our expert.

Finally, we obtained encouraging results when we used fake images as learning base for other networks: Our sets of experiments seems to indicates that pre-training on fake data is a relevant technique. Those results should be consolidated with a more effective architecture such as DeepLab to get rid of bias induced by SegNet.

In further study different architectures should be tested. Taking advantage of the scale invariance of our image we could not only reduce the depth of the networks but also make them larger. This could be done by making small parallel networks dedicated to a unique task and then fusing their outputs. We could imagine having a texture dedicated network trained on the KTH-TIPS2 dataset [54], fused to a contextual network based only on pyramidal atrous convolution to extract the context. Moreover, we should consider using the graph analysis we made to extract high level context and mimic human behavior and thus, be able to classify more classes.
Appendices
Our experimental set up was composed of 3 Dell servers equipped with 16Gb of RAM, 12 cores Intel Xeons and a Nvidia K20c GPU.
### APPENDIX B

**KITTI BENCHMARK**

<table>
<thead>
<tr>
<th>Method</th>
<th>Setting</th>
<th>MaxF</th>
<th>AP</th>
<th>PRE</th>
<th>REC</th>
<th>FPR</th>
<th>FNR</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNV</td>
<td></td>
<td>96.69 %</td>
<td>92.41 %</td>
<td>97.38 %</td>
<td>96.01 %</td>
<td>1.18 %</td>
<td>3.99 %</td>
<td>1.2 s / GPU</td>
</tr>
<tr>
<td>RPP</td>
<td></td>
<td>96.04 %</td>
<td>89.77 %</td>
<td>95.61 %</td>
<td>96.48 %</td>
<td>2.02 %</td>
<td>3.52 %</td>
<td>0.16 s /</td>
</tr>
<tr>
<td>KRSF</td>
<td></td>
<td>96.02 %</td>
<td>93.60 %</td>
<td>95.61 %</td>
<td>96.44 %</td>
<td>2.02 %</td>
<td>3.56 %</td>
<td>0.3 s / GPU</td>
</tr>
<tr>
<td>KRS</td>
<td></td>
<td>95.89 %</td>
<td>93.51 %</td>
<td>95.79 %</td>
<td>95.99 %</td>
<td>1.92 %</td>
<td>4.01 %</td>
<td>0.3 s / GPU</td>
</tr>
<tr>
<td>YhY</td>
<td></td>
<td>95.80 %</td>
<td>89.11 %</td>
<td>94.89 %</td>
<td>96.73 %</td>
<td>2.38 %</td>
<td>3.27 %</td>
<td>0.4 s / 1 core</td>
</tr>
<tr>
<td>SAIT</td>
<td></td>
<td>95.65 %</td>
<td>90.17 %</td>
<td>96.06 %</td>
<td>95.25 %</td>
<td>1.78 %</td>
<td>4.75 %</td>
<td>0.04 s / GPU</td>
</tr>
<tr>
<td>TuSimple</td>
<td></td>
<td>95.64 %</td>
<td>93.50 %</td>
<td>95.46 %</td>
<td>95.83 %</td>
<td>2.08 %</td>
<td>4.17 %</td>
<td>0.2s / GPU</td>
</tr>
<tr>
<td>DFFA</td>
<td></td>
<td>95.58 %</td>
<td>89.30 %</td>
<td>95.10 %</td>
<td>96.06 %</td>
<td>2.25 %</td>
<td>3.94 %</td>
<td>0.4 s / GPU</td>
</tr>
<tr>
<td>MVNet</td>
<td>la</td>
<td>95.45 %</td>
<td>91.49 %</td>
<td>97.51 %</td>
<td>93.49 %</td>
<td>1.09 %</td>
<td>6.51 %</td>
<td>0.04 s / 1 core</td>
</tr>
<tr>
<td>RSNet</td>
<td></td>
<td>95.28 %</td>
<td>92.43 %</td>
<td>95.22 %</td>
<td>95.35 %</td>
<td>2.18 %</td>
<td>4.65 %</td>
<td>0.09 s / GPU</td>
</tr>
<tr>
<td>DCCN</td>
<td>la</td>
<td>95.18 %</td>
<td>92.44 %</td>
<td>94.69 %</td>
<td>95.68 %</td>
<td>2.45 %</td>
<td>4.32 %</td>
<td>25 ms / GPU</td>
</tr>
<tr>
<td>SAIT</td>
<td></td>
<td>95.09 %</td>
<td>89.58 %</td>
<td>95.41 %</td>
<td>94.76 %</td>
<td>2.08 %</td>
<td>5.24 %</td>
<td>0.04 s / GPU</td>
</tr>
<tr>
<td>TDCac1 CNN</td>
<td></td>
<td>94.86 %</td>
<td>89.62 %</td>
<td>95.45 %</td>
<td>94.28 %</td>
<td>2.05 %</td>
<td>5.72 %</td>
<td>0.093 s / 1 core</td>
</tr>
<tr>
<td>RSNet-</td>
<td></td>
<td>94.84 %</td>
<td>92.83 %</td>
<td>94.32 %</td>
<td>95.37 %</td>
<td>2.62 %</td>
<td>4.63 %</td>
<td>0.07 s / GPU</td>
</tr>
<tr>
<td>baseline</td>
<td></td>
<td>94.80 %</td>
<td>92.80 %</td>
<td>94.35 %</td>
<td>95.25 %</td>
<td>2.60 %</td>
<td>4.75 %</td>
<td>0.05 s / 1 core</td>
</tr>
<tr>
<td>RBNet</td>
<td></td>
<td>94.77 %</td>
<td>91.42 %</td>
<td>95.16 %</td>
<td>94.37 %</td>
<td>2.19 %</td>
<td>5.63 %</td>
<td>0.18 s / GPU</td>
</tr>
<tr>
<td>WSLGAN</td>
<td></td>
<td>94.73 %</td>
<td>89.22 %</td>
<td>95.01 %</td>
<td>94.45 %</td>
<td>2.26 %</td>
<td>5.55 %</td>
<td>800ms / GPU</td>
</tr>
<tr>
<td>MMN</td>
<td></td>
<td>94.72 %</td>
<td>92.51 %</td>
<td>94.84 %</td>
<td>94.60 %</td>
<td>2.34 %</td>
<td>5.40 %</td>
<td>0.1 s / GPU</td>
</tr>
<tr>
<td>KRS</td>
<td></td>
<td>94.69 %</td>
<td>93.40 %</td>
<td>94.72 %</td>
<td>94.67 %</td>
<td>2.41 %</td>
<td>5.33 %</td>
<td>1 s / GPU</td>
</tr>
<tr>
<td>RSNet2</td>
<td></td>
<td>94.65 %</td>
<td>92.54 %</td>
<td>94.45 %</td>
<td>94.85 %</td>
<td>2.54 %</td>
<td>5.15 %</td>
<td>0.07 s / GPU</td>
</tr>
<tr>
<td>SSLGAN</td>
<td></td>
<td>94.62 %</td>
<td>89.50 %</td>
<td>95.32 %</td>
<td>93.93 %</td>
<td>2.10 %</td>
<td>6.07 %</td>
<td>700ms / GPU</td>
</tr>
<tr>
<td>FNETMS</td>
<td></td>
<td>94.51 %</td>
<td>92.72 %</td>
<td>94.92 %</td>
<td>94.11 %</td>
<td>2.30 %</td>
<td>5.89 %</td>
<td>0.04 s / GPU</td>
</tr>
<tr>
<td>DEEP-DIG</td>
<td>st</td>
<td>94.16 %</td>
<td>93.41 %</td>
<td>95.02 %</td>
<td>93.32 %</td>
<td>2.23 %</td>
<td>6.68 %</td>
<td>0.14 s / GPU</td>
</tr>
<tr>
<td>FusionNet</td>
<td></td>
<td>94.15 %</td>
<td>92.26 %</td>
<td>95.18 %</td>
<td>93.14 %</td>
<td>2.15 %</td>
<td>6.86 %</td>
<td>0.3 s / GPU</td>
</tr>
<tr>
<td>StixelNet II</td>
<td></td>
<td>94.05 %</td>
<td>85.85 %</td>
<td>91.30 %</td>
<td>96.98 %</td>
<td>4.21 %</td>
<td>3.02 %</td>
<td>1.2 s / 1 core</td>
</tr>
<tr>
<td>MultiNet</td>
<td></td>
<td>93.99 %</td>
<td>93.24 %</td>
<td>94.51 %</td>
<td>93.48 %</td>
<td>2.47 %</td>
<td>6.52 %</td>
<td>0.17 s / GPU</td>
</tr>
<tr>
<td>Model</td>
<td>SSL</td>
<td>RDBN</td>
<td>CoDNN</td>
<td>AXN</td>
<td>FuseNet</td>
<td>FCN-GCBs</td>
<td>ResNetPK</td>
<td>MBN</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td></td>
<td>93.94 %</td>
<td>93.90 %</td>
<td>93.90 %</td>
<td>93.86 %</td>
<td>93.86 %</td>
<td>93.86 %</td>
<td>93.78 %</td>
<td>93.77 %</td>
</tr>
<tr>
<td></td>
<td>89.36 %</td>
<td>91.23 %</td>
<td>92.86 %</td>
<td>92.63 %</td>
<td>93.49 %</td>
<td>86.62 %</td>
<td>89.01 %</td>
<td>85.59 %</td>
</tr>
<tr>
<td></td>
<td>94.78 %</td>
<td>94.13 %</td>
<td>94.51 %</td>
<td>94.09 %</td>
<td>94.40 %</td>
<td>92.15 %</td>
<td>94.78 %</td>
<td>91.02 %</td>
</tr>
<tr>
<td></td>
<td>93.11 %</td>
<td>93.67 %</td>
<td>93.29 %</td>
<td>93.63 %</td>
<td>93.23 %</td>
<td>95.62 %</td>
<td>92.81 %</td>
<td>96.69 %</td>
</tr>
<tr>
<td></td>
<td>2.34 %</td>
<td>2.66 %</td>
<td>2.47 %</td>
<td>2.68 %</td>
<td>2.48 %</td>
<td>3.71 %</td>
<td>2.33 %</td>
<td>4.34 %</td>
</tr>
<tr>
<td></td>
<td>6.89 %</td>
<td>6.33 %</td>
<td>6.71 %</td>
<td>6.37 %</td>
<td>6.77 %</td>
<td>3.17 %</td>
<td>7.19 %</td>
<td>3.31 %</td>
</tr>
<tr>
<td></td>
<td>0.05 s / GPU</td>
<td>0.25 s / GPU</td>
<td>0.01 s / 1 core</td>
<td>0.06 s / GPU</td>
<td>0.2 s / GPU</td>
<td>0.08 s / GPU</td>
<td>0.4s / GPU</td>
<td>0.1 s / GPU</td>
</tr>
<tr>
<td>Method</td>
<td>Precision</td>
<td>Recall</td>
<td>F1 Score</td>
<td>Accuracy</td>
<td>Speed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>--------</td>
<td>----------</td>
<td>----------</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LWDS</td>
<td>89.50 %</td>
<td>86.11 %</td>
<td>90.59 %</td>
<td>88.44 %</td>
<td>4.19 %</td>
<td>11.56 %</td>
<td>0.07 s / GPU</td>
<td></td>
</tr>
<tr>
<td>BNV</td>
<td>89.42 %</td>
<td>83.13 %</td>
<td>88.31 %</td>
<td>90.55 %</td>
<td>5.46 %</td>
<td>9.45 %</td>
<td>3 s / 2 cores</td>
<td></td>
</tr>
<tr>
<td>FCN-LC</td>
<td>89.36 %</td>
<td>78.80 %</td>
<td>89.35 %</td>
<td>89.37 %</td>
<td>4.85 %</td>
<td>10.63 %</td>
<td>0.03 s /</td>
<td></td>
</tr>
<tr>
<td>CB</td>
<td>88.89 %</td>
<td>82.17 %</td>
<td>87.26 %</td>
<td>90.58 %</td>
<td>6.03 %</td>
<td>9.42 %</td>
<td>2 s / 1 core</td>
<td></td>
</tr>
<tr>
<td>VPD</td>
<td>88.87 %</td>
<td>82.04 %</td>
<td>93.31 %</td>
<td>84.83 %</td>
<td>2.77 %</td>
<td>15.17 %</td>
<td>4 s / 1 core</td>
<td></td>
</tr>
<tr>
<td>HFM</td>
<td>88.83 %</td>
<td>80.33 %</td>
<td>85.23 %</td>
<td>92.75 %</td>
<td>7.32 %</td>
<td>7.25 %</td>
<td>5 s / 2 cores</td>
<td></td>
</tr>
<tr>
<td>DVFCN</td>
<td>88.64 %</td>
<td>91.37 %</td>
<td>89.20 %</td>
<td>88.10 %</td>
<td>4.86 %</td>
<td>11.90 %</td>
<td>0.07 s / GPU</td>
<td></td>
</tr>
<tr>
<td>ResAXN</td>
<td>88.46 %</td>
<td>91.06 %</td>
<td>90.10 %</td>
<td>86.88 %</td>
<td>4.35 %</td>
<td>13.12 %</td>
<td>0.06 s / GPU</td>
<td></td>
</tr>
<tr>
<td>SPRAY</td>
<td>88.14 %</td>
<td>91.24 %</td>
<td>88.60 %</td>
<td>87.68 %</td>
<td>5.14 %</td>
<td>12.32 %</td>
<td>45 ms /</td>
<td></td>
</tr>
<tr>
<td>FCNB</td>
<td>87.98 %</td>
<td>77.50 %</td>
<td>87.77 %</td>
<td>88.18 %</td>
<td>5.60 %</td>
<td>11.82 %</td>
<td>0.2 s / 1 core</td>
<td></td>
</tr>
<tr>
<td>ProbBoost</td>
<td>87.48 %</td>
<td>80.13 %</td>
<td>85.02 %</td>
<td>90.09 %</td>
<td>7.23 %</td>
<td>9.91 %</td>
<td>2.5 min / 8 cores</td>
<td></td>
</tr>
<tr>
<td>LWD</td>
<td>87.39 %</td>
<td>88.85 %</td>
<td>84.99 %</td>
<td>89.92 %</td>
<td>7.24 %</td>
<td>10.08 %</td>
<td>0.07 s / GPU</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>87.33 %</td>
<td>89.62 %</td>
<td>85.77 %</td>
<td>88.95 %</td>
<td>6.73 %</td>
<td>11.05 %</td>
<td>0.28 s /</td>
<td></td>
</tr>
<tr>
<td>Multimodal</td>
<td>87.28 %</td>
<td>90.82 %</td>
<td>87.12 %</td>
<td>87.44 %</td>
<td>5.89 %</td>
<td>12.56 %</td>
<td>0.3 s / 4 cores</td>
<td></td>
</tr>
<tr>
<td>CN24</td>
<td>86.32 %</td>
<td>89.19 %</td>
<td>87.80 %</td>
<td>84.89 %</td>
<td>5.37 %</td>
<td>15.11 %</td>
<td>30 s / 8 cores</td>
<td></td>
</tr>
<tr>
<td>GRES3D+VELO</td>
<td>la</td>
<td>85.43 %</td>
<td>83.04 %</td>
<td>82.69 %</td>
<td>88.37 %</td>
<td>8.43 %</td>
<td>11.63 %</td>
<td>60 ms / 4 cores</td>
</tr>
<tr>
<td>StixelNet</td>
<td>85.33 %</td>
<td>72.14 %</td>
<td>81.21 %</td>
<td>89.89 %</td>
<td>9.48 %</td>
<td>10.11 %</td>
<td>1 s / GPU</td>
<td></td>
</tr>
<tr>
<td>SPPlane + BL</td>
<td>st</td>
<td>85.23 %</td>
<td>88.66 %</td>
<td>83.43 %</td>
<td>87.12 %</td>
<td>7.89 %</td>
<td>12.88 %</td>
<td>2 s / 1 core</td>
</tr>
<tr>
<td>geo+gpr+crf</td>
<td>st</td>
<td>85.13 %</td>
<td>72.24 %</td>
<td>81.33 %</td>
<td>89.29 %</td>
<td>9.34 %</td>
<td>10.71 %</td>
<td>30 s / 1 core</td>
</tr>
<tr>
<td>RES3D-Velo</td>
<td>la</td>
<td>83.81 %</td>
<td>73.95 %</td>
<td>78.56 %</td>
<td>89.80 %</td>
<td>11.16 %</td>
<td>10.20 %</td>
<td>0.36 s / 1 core</td>
</tr>
<tr>
<td>SCRFFPFHGSP</td>
<td>st</td>
<td>83.73 %</td>
<td>72.89 %</td>
<td>82.13 %</td>
<td>85.39 %</td>
<td>8.47 %</td>
<td>14.61 %</td>
<td>5 s / 8 cores</td>
</tr>
<tr>
<td>RNF</td>
<td>83.71 %</td>
<td>72.64 %</td>
<td>88.14 %</td>
<td>79.71 %</td>
<td>4.89 %</td>
<td>20.29 %</td>
<td>.5 s / 1 core</td>
<td></td>
</tr>
<tr>
<td>GRES3D+SELAS</td>
<td>st</td>
<td>83.69 %</td>
<td>84.61 %</td>
<td>78.31 %</td>
<td>89.88 %</td>
<td>11.35 %</td>
<td>10.12 %</td>
<td>110 ms / 4 core</td>
</tr>
<tr>
<td>HistonBoost</td>
<td>st</td>
<td>83.68 %</td>
<td>72.79 %</td>
<td>82.01 %</td>
<td>85.42 %</td>
<td>8.54 %</td>
<td>14.58 %</td>
<td>2.5 min / 8 cores</td>
</tr>
<tr>
<td>FRS_SP</td>
<td>83.22 %</td>
<td>72.94 %</td>
<td>77.11 %</td>
<td>90.39 %</td>
<td>12.23 %</td>
<td>9.61 %</td>
<td>0.21 s / 4 cores</td>
<td></td>
</tr>
<tr>
<td>PFH+HSV</td>
<td>82.99 %</td>
<td>74.81 %</td>
<td>84.48 %</td>
<td>81.56 %</td>
<td>6.83 %</td>
<td>18.44 %</td>
<td>0.1 s / 1 core</td>
<td></td>
</tr>
<tr>
<td>FCNS</td>
<td>82.97 %</td>
<td>87.01 %</td>
<td>80.84 %</td>
<td>85.21 %</td>
<td>9.20 %</td>
<td>14.79 %</td>
<td>6 s / 1 core</td>
<td></td>
</tr>
<tr>
<td>INM</td>
<td>82.56 %</td>
<td>74.66 %</td>
<td>90.92 %</td>
<td>75.61 %</td>
<td>3.44 %</td>
<td>24.39 %</td>
<td>5 s / 1 core</td>
<td></td>
</tr>
<tr>
<td>SegNet</td>
<td>82.17 %</td>
<td>76.46 %</td>
<td>84.03 %</td>
<td>80.40 %</td>
<td>6.97 %</td>
<td>19.60 %</td>
<td>0.01 s / 8 cores</td>
<td></td>
</tr>
<tr>
<td>LKW</td>
<td>82.01 %</td>
<td>85.26 %</td>
<td>78.83 %</td>
<td>85.46 %</td>
<td>10.46 %</td>
<td>14.54 %</td>
<td>0.1 s / 1 core</td>
<td></td>
</tr>
<tr>
<td>SP-SS</td>
<td>81.60 %</td>
<td>69.62 %</td>
<td>78.13 %</td>
<td>85.40 %</td>
<td>10.89 %</td>
<td>14.60 %</td>
<td>0.01 s / 4 cores</td>
<td></td>
</tr>
<tr>
<td>PGM-ARS</td>
<td>80.97 %</td>
<td>69.11 %</td>
<td>77.51 %</td>
<td>84.76 %</td>
<td>11.21 %</td>
<td>15.24 %</td>
<td>0.05 s / i74700MQ</td>
<td></td>
</tr>
<tr>
<td>Pos-ex</td>
<td>79.47 %</td>
<td>85.99 %</td>
<td>78.78 %</td>
<td>80.17 %</td>
<td>9.84 %</td>
<td>19.83 %</td>
<td>120 ms / GPU(K20)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>RES3D-Stereo</td>
<td>st</td>
<td>78.98 %</td>
<td>80.06 %</td>
<td>75.94 %</td>
<td>82.27 %</td>
<td>11.88 %</td>
<td>17.73 %</td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>st</td>
<td>78.90 %</td>
<td>66.06 %</td>
<td>69.53 %</td>
<td>91.19 %</td>
<td>18.21 %</td>
<td>8.81 %</td>
<td></td>
</tr>
<tr>
<td>SPlane</td>
<td>st</td>
<td>78.19 %</td>
<td>76.41 %</td>
<td>72.03 %</td>
<td>85.50 %</td>
<td>15.13 %</td>
<td>14.50 %</td>
<td></td>
</tr>
<tr>
<td>SRF</td>
<td></td>
<td>76.43 %</td>
<td>83.24 %</td>
<td>75.53 %</td>
<td>77.35 %</td>
<td>11.42 %</td>
<td>22.65 %</td>
<td></td>
</tr>
<tr>
<td>CN24</td>
<td></td>
<td>76.28 %</td>
<td>79.29 %</td>
<td>72.44 %</td>
<td>80.55 %</td>
<td>13.96 %</td>
<td>19.45 %</td>
<td></td>
</tr>
<tr>
<td>RFH</td>
<td></td>
<td>76.18 %</td>
<td>61.64 %</td>
<td>68.38 %</td>
<td>86.00 %</td>
<td>18.12 %</td>
<td>14.00 %</td>
<td></td>
</tr>
<tr>
<td>CN</td>
<td></td>
<td>73.69 %</td>
<td>76.68 %</td>
<td>69.18 %</td>
<td>78.83 %</td>
<td>16.00 %</td>
<td>21.17 %</td>
<td></td>
</tr>
<tr>
<td>NSV</td>
<td></td>
<td>73.15 %</td>
<td>69.75 %</td>
<td>61.48 %</td>
<td>90.29 %</td>
<td>25.77 %</td>
<td>9.71 %</td>
<td></td>
</tr>
<tr>
<td>ARSL-AMI</td>
<td></td>
<td>71.97 %</td>
<td>61.04 %</td>
<td>78.03 %</td>
<td>66.79 %</td>
<td>8.57 %</td>
<td>33.21 %</td>
<td></td>
</tr>
<tr>
<td>ANN</td>
<td>st</td>
<td>62.83 %</td>
<td>46.77 %</td>
<td>50.21 %</td>
<td>83.91 %</td>
<td>37.91 %</td>
<td>16.09 %</td>
<td></td>
</tr>
<tr>
<td>VAP</td>
<td></td>
<td>59.23 %</td>
<td>42.05 %</td>
<td>44.44 %</td>
<td>88.75 %</td>
<td>50.55 %</td>
<td>11.25 %</td>
<td></td>
</tr>
</tbody>
</table>

Table B.1: Benchmark of neural networks on the kitti dataset[41]
REFERENCES


[14] W. Commons. (2010). Orthographic views project at a right angle to the data plane. perspective views project from the surface onto the datum plane from a fixed location. File:OrthoPerspective.svg.


