POLITECNICO DI TORINO

Tesi di Laurea Magistrale

Road Feature Extraction With Deep Learning Methods

Relatore
Prof. Barbara Caputo

Correlatore:
Prof. Aleksei Tepljakov

Laureando
Davide Liberato Manna
matricola: 253005

Dicembre 2019

Abstract

Deep Convolutional Neural Networks (CNNs) have recently become the subject of rigorous investigation especially due to their favourable properties in the field of computer vision and hence have been utilized in numerous related applications. Among these, image classification and semantic segmentation have acquired particular research interest. This work is a part of an applied project funded by Reach-U - a company specializing in geographic information systems, location based solutions and cartography - and is geared towards smart city applications. In particular, the topic of this thesis is focused on the detection of road features which includes both semantic segmentation and image classification. Towards accomplishing this goal, two ad-hoc datasets were produced containing specific road features in accordance with the needs of the company. Towards graphical scene processing, a novel lightweight CNN architecture was introduced in this thesis capable of performing adequate semantic segmentation. For the problem of image classification, a custom CNN was proposed and its performance compared to the state-of-the-art ResNet network, the latter fine-tuned towards solving the same task. Despite the low number of network parameters, it is expected that the proposed segmentation network will have better performance in the sense of shape detection even though its application results in lower accuracy levels. In contrast, the proposed classifier reveals competitive accuracy with respect to its counterpart. A graphical user interface has also been developed which uses the resulting CNN as backend. The complete solution is suitable to satisfy the end goal which is the construction of an appropriate 3D model based on 2D segmented images.

Acknowledgements

I would like to thank my Co-Supervisor, Dr. Aleksei Tepljakov. His constant support, his suggestions and our frequent conversations gifted me extremely enjoyable moments. Working alongside with him allowed me to learn a lot about my passions and about the pleasure of working in a friendly environment. In this regard, I would also like to thank the rest of the research team, Andri and René, for always making me feel at ease and for sharing their ideas with me, confronting each other. An other thanks should be given to prof. Barbara Caputo, for first introducing me to very interesting topics with her lessons and for then agreeing to be my supervisor.

I would like to thank my family for allowing me to undertake my path and for never stopping believing in me. A special thanks, however, should be given to my life partner, Serena. Her constant presence, despite the difficulty in bearing with me and my madnesses, her support and her precious suggestions made this work possible and kept me motivated in carrying it out till the end.

Thanks to Alberto, Stefano, Diego, Dade, Fede and all of my lifetime friends for always welcoming me back even when I started disappearing. And finally, thanks to Ale (Sax), Lorenzo, Pietro, Davide, Isa, Jeanpier, Giorgio, Andrea, Simona and all of the wonderful people I met during my years at Politecnico of Turin: their friendship has been a gift that this journey left me.

This research endeavor was partially supported by Archimedes Foundation and Reach-U Ltd. in the scope of the smart specialization research and development project #LEP19022: "Applied research for creating a cost-effective interchangeable 3D spatial data infrastructure with survey-grade accuracy".

Contents

List of Tables

List of Figures

Study hard what interests you the most in the most undisciplined, irreverent and original manner possible.
R. Feynmann

Chapter 1

Background

Automation has always been one of the main goals of computer science, aiming to improve everyday tasks, speed them up and possibly allow a higher ease of use. Following this baseline, first special purpose machines such as the Difference Engine [9] were built, incorporating all of the required components and hard-coded knowledge to solve the task they were designed for. When the idea of a general purpose machine sprout up however, researchers started to look forward towards the possibility that these machines would actually become intelligent [10] and, as time passed and programmable computers became reality, these ideas continued growing and new theories were developed stating that machines could actually develop their own intelligence [11].

Nowadays we know these ideas under the name of *Artificial Intelligence* (AI). AI attracted significant research interest and thus became a widespread research field and has had a continuingly increasing involvement in several different practical applications, starting from the medical field with medical imaging analysis [12,13,14,15] to autonomously driving cars [16,17] and robotics [18,19].

But what does Artificial Intelligence *mean*? The term was founded by John McCarthy in 1955 who, one year later, started the Dartmouth Summer Research Project on Artificial Intelligence, which marked AI as a standalone field. From that moment on several pieces of literature have been dedicated to what an artificial intelligence is and to whether a machine might be capable of *thinking*, however after over 65 years there is not an official reference definition yet and the term is often misused or misunderstood. A lot of the confusion, perhaps, comes from the ambiguity of what we consider to be *intelligent.* In any case, whichever its meanining might be, Turing stated that for a machine to become intelligent it is crucial to undergo an educational process which will allow it to build its own knowledge. Other approaches, such as that of instilling hard-coded logic into the so-called knowledge-driven systems, have been pursued by researchers with poor results [20]. Nevertheless, these results hightlighted the need for a machine to *extract* its own knowledge from raw data. This process has become known as *Machine Learning*. Although machine learning algorithms have had several different implementations, in recent years one of them

has been having outstanding outcomes and received a incredible boost in terms of research interest and fundings from companies: we are talking about *Deep Learning.*

Before addressing the core aspects of this work, this chapter will provide some general background information. We will address Machine Learning developments by considering some of the most important milestones in history. The most relevant theoretical topics will also be covered and expanded in mathematical terms in order to provide a solid base. An overview about this work arguments and its context will also be detailed in section 1.2.

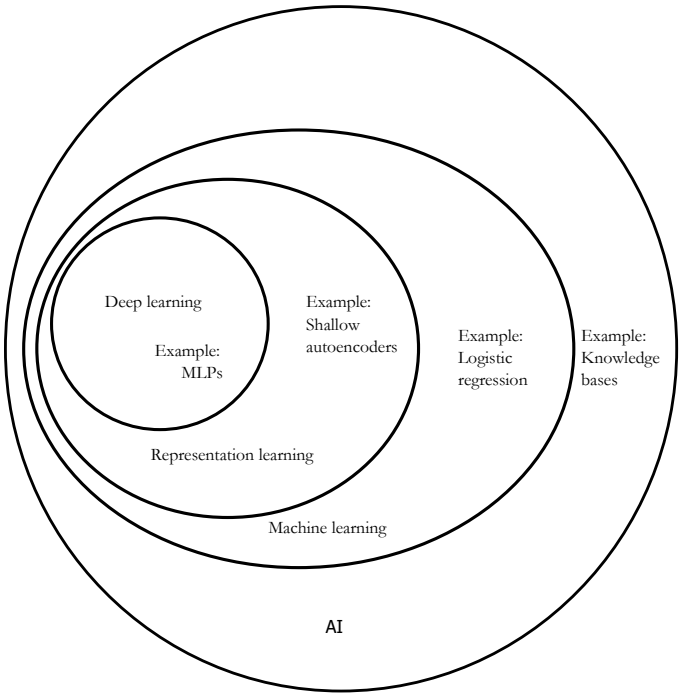


Figure 1.1: Relationships between applied artificial intelligence concepts, also referred to as "AI Circles" [1]

1.1 History and Theoretical Insights

Within this section we will provide a brief historical background regarding the topics developed in this script. We will discuss Machine Learning and Artificial Intelligence history, further lingering on the theoretical aspect that are most relevant to the addressed topics as soon as they come up on the historical line. Specifically we will explore from a neuroscientifical point of view how the concept of neural network was developed and how it evolved into a more statistical approach in modern solutions. Starting from 1943, we will discuss Neural Networks concept birth, introducing both mathematical and neuroscientifical aspects. We will then discuss the computer-brain analogy and move towards the implemetation of the first machine learning algorithms. Due to their extreme pairing with machine learning, we will also see and discuss computer vision techniques and how

they were employed within the learning algorithms. Finally, we will see *backpropagation,* a concept that stands as a base for modern deep learning methods, and some of the latest architectural implementations.

Neuron and Neural Networks (1943)

Warren McCulloch, a neurophysiologist, and the mathematician Walter Pitts wrote a paper about functioning of brain's neurons [21, 22]. In this paper, they described an approximation of the electrical neural signal displayed in presence of stimuli. Even if today the concept of neurons and how their work could seem obvious, it must be taken into consideration that Neuroscience was recognized as a standalone discipline only in mid-20th century. Before 1943 it was considered part of other disciplines and very little was known about brain and neuron functioning. The first empirical proof of how a neuron works was provided by Alan Lloyd Hodgkin and Andrew Huxley in 1952. They achieved to create the model of the signal transmission of squid neurons [23, 24]. In the light of the period we can consider this first attempt of neural modelling very surprising compared to the level of knowledge about the brain and its signal propagation process. In McCulloch and Pitts paper, a simple model of a neural network using an electrical circuit was also provided. Neuroscience discipline lately brought us the knowledge of how neurons and neural network works. We now know that brain is composed by more than 10 billion neurons, each connected with 10 thousand other neurons trough *dendrites* and *axons*. The connection within neurons are fundamental for the propagation of the chemical and eclectic signal. Neurons are characterized by an all or nothing threshold, it means that a neuron will fire if and only if the signal received is enough to overcome the minimum required [25]. In these terms, neurons seem very close to a programming language: in fact, the binary system employed by computer consists in 0 or 1. Zero for the absence and 1 for the presence of the characteristic.

When designing an artificial neural network, this activation is modelled by a mathematical function which *emits* or *fires* a value under certain conditions. Several types of activation functions exist and all provide different features, *sigmoid* function for example models the output in values between 0 and 1 while the *hyperbolic tangent* (or *tanh*) does the same between -1 and 1 (Figure 1.2a). In most of the cases anyway, the choice of the activation function would fall upon one which introduces a *non linearity* inside the network. Using a Linear function can in fact hamper the backpropagation system because its derivative would always be constant and not related to the input, moreover it would nullify any effort to build deeper networks: all the layers would collapse into a single one since a linear combination of linear functions still is a linear function.

The sigmoidal units described above provide a saturated output for most of their domain values (0 or -1 for very negative values and 1 for very positive) and susceptible to the input only when its value is close to zero. Such saturation might once more hamper most of the gradient based learnings, therefore an other activation unit has taken hold and is nowadays a de facto standard in CNN design: the *Rectified Linear Unit* or *ReLU*.

ReLu is a simple activation function which returns the input as is whenever greater than zero and returns 0 for every value below it:

$$f(x) = max(0, x). \tag{1.1}$$

The powerful aspect of ReLU resides in the fact that it introduces a non linearity in the system, while keeping linear properties for most of its values (Figure 1.2b). Other advantages are that it increases the grade of sparsity (all values < 0 are zero-ed) and diminishes the likelihood of having vanishing gradients, while being inexpensive to use. Generally speaking, deep networks using rectified linear are easier to train than those using units with a curvature like the *softplus* or with a double-sided saturation [26].

Finally, an other function worth noticing is the *softmax* activation function. Softmax can be though of as a generalization of the sigmoid when trasposed to a multiclass problem and is basically a normalized exponential function defined in the following way [27]:

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad \text{for } i = 1, ..., K \text{ and } x = (x_1, ..., x_K) \in \mathbb{R}^K.$$

In other words, we take the exponential of each element of the input x and normalize it by dividing it by the sum of the exponentials of all the other elements. As a result, we will obtain a set of values in [0, 1] summing to 1, which naturally resembles a probability distribution. For this reason, softmax is often used as the activation of function for the last layer in neural networks.

To conclude, several possibilities exist when it comes to the choice of the activation functions, however it has been prooven that rectified linear unit and its variants yield to better results when applied to the output of the hidden units of deep neural network, thus making them the default choice in most of the cases.

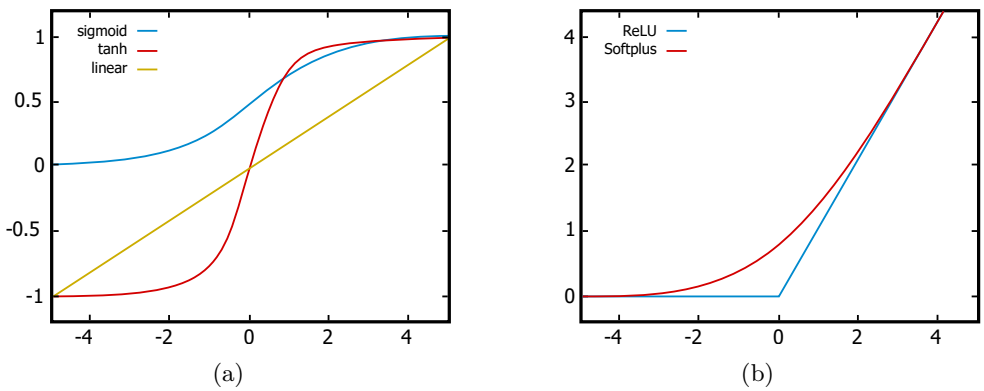


Figure 1.2: Examples of Activation Functions.

Machine Learning and Artificial Neural Networks (1949)

Donald Hebb presents his theories about neuron excitement and communication in his book "The Organization of Behavior" [28]. In his work, Hebb states:

> "When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell"

thus giving life to what is know as Hebbian Theory. Basing on brain neurons behaviors and interaction described by this theory, the first Machine Learning models were originated. As a matter of fact, these concepts were translated in computer science for the creation of artificial neural networks and artificial neurons. Artificial neural network (ANN) connection was based on two assumptions. The connections between neurons that are activated at the same time are defined Strong, those between neurons that are activated separately are defined as Weak. The type of connection between two neurons is defined by a "weight" and it is used to describe the relationships and nodes between neurons [29, 30]. Artificial neural networks are based on this model; however, they are still far from matching with their original competitor: the brain. ANN, in fact, are not able to reach the same level of complexity, even if they have shown good problem solving in tasks difficult for humans such as image recognition and prediction based on past knowledge.

Brain-Computer Analogy and Turing Machine (1950)

With the advance of the computer technology, the brain – computer analogy was brought to its extreme. From the technological and IT point of view, this analogy was interesting source of inspiration. The brain in fact is extremely powerful both in terms of computational power and efficiency since it is able to perform an incredibly high amount of operation in parallel using low energy. On one side, this analogy lead to the development of the simulation of a first hypothetical neural network. On the other side this metaphor was extremely damaging for the advances in Neuroscience research, which tried to organize and model brain functioning from computer behavior. In the same years Turing creates his "Turing test" [11]. Turing test consisted in discriminating whether the answer given to a participant's question came from a person or an AI (see Figure 1.3 for setting). To pass the test, the computer must be able to produce human-like answer such that the participant would not be able to tell the human and the computer apart. The most incredible results achieved by this test was that the participants were indeed not able to discriminate between the answers given by a real person and the ones produced by the machine.

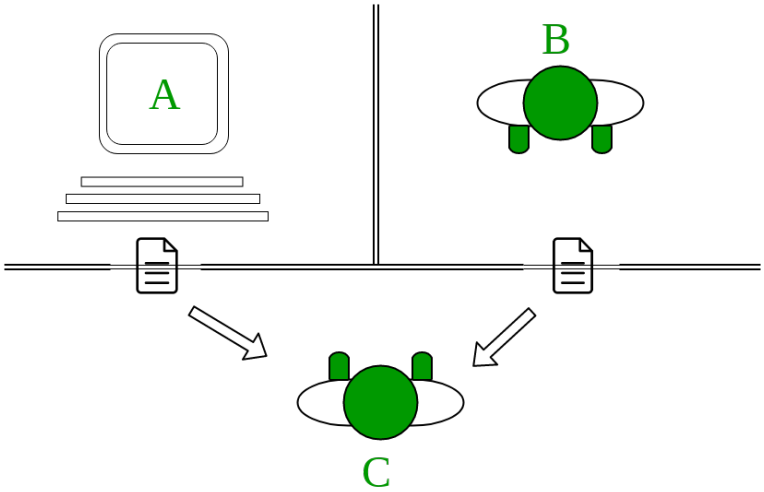


Figure 1.3: Turing Test Schema. The participant (C) has to discriminate whether the answer to the question comes from A (machine) or B (human).

The Birth of Machine Learning (1952)

After the modelling of neural networks for, Arthur Samuel wrote the first computer learning program. The program consisted in a game in which the computer, basing on past experience, had to develop strategies to win. From this program Arthur Samuel first came up with the term "Machine Learning". This is the first example of the application of machine learning techniques. Keith D. Foote defined machine learning (ML) as the use of algorithms to parse data, learn from it, and then make a determination or prediction about something in the world. The machine is trained on a large amount of data by means of these algorithms, thus giving it the ability to learn how to perform a given task [31]. The process of learning begins with the observations of data in order to find a characterizing, repeated pattern in it. Such patters are then used to build a mathematical model which will be used to better identify future data and perform better predictions. In other words, machine learning algorithms allow the computer to learn a representation from collected data and then use it make different decisions without being specifically programmed to make them. ML algorithms can be subdivided into to categories: Supervised and Unsupervised.

- Supervised algorithms. The starting point of these algorithms is always a training phase where the machine uses a known, labelled dataset to infer functions and models to make future predictions. The advantage given by a labelled dataset is that the output produced by the machine can be compared with the correct answer, thus allowing it to modify its behavior accordingly.
- Unsupervised algorithms. Contrary to supervised algorithm, unsupervised learning bases the training on an unknown dataset. This means that neither label nor previous classifications are present for comparison. Unsupervised learning studies how the system can infer functions, cluster and describe hidden structure from a dataset.

Machine learning also requires the analysis of massive quantities of data to learn a generalizing model. Consequently, the training phase usually requires a lot of time to reach its end. Combining ML with AI and cognitive technologies can make such information processing more efficient.

The Perceptron (1957)

In 1957, Frank Rosenblatt combined Donald Hebb's model of brain cell interaction with Arthur Samuel's Machine Learning efforts and created the Perceptron [32]. Perceptron is basically a mathematical model of a biological neuron. Biological neuron receives and emit electrical signal from axons and dendrites (Figure 1.4) In perceptron electric signals were represented with numerical values. At the synapses between the dendrite and axons, electrical signals are modulated by different weights. Moreover, biological neurons fires if and only if the received signal has a big enough amplitude to surpass a given threshold. In a similar way, in the perceptron the signal amplitude is obtained by the weighted sum of all the inputs. Such sum can be used as the argument of a properly set step function, which will act as a threshold and determine the output (Figure 1.5). In mathematical terms, this behavior can be described by the following formula [33]:

$$f(x) = g(w_0 + n \sum w_i x_i) = g(w_0 + \boldsymbol{w} \cdot \boldsymbol{x}), \tag{1.2}$$

where $g(\cdot)$ is an activation function, in this case a *step*, w_0 is the bias and $\boldsymbol{w} \cdot \boldsymbol{x}$ is the dot product between the input the weight vector.

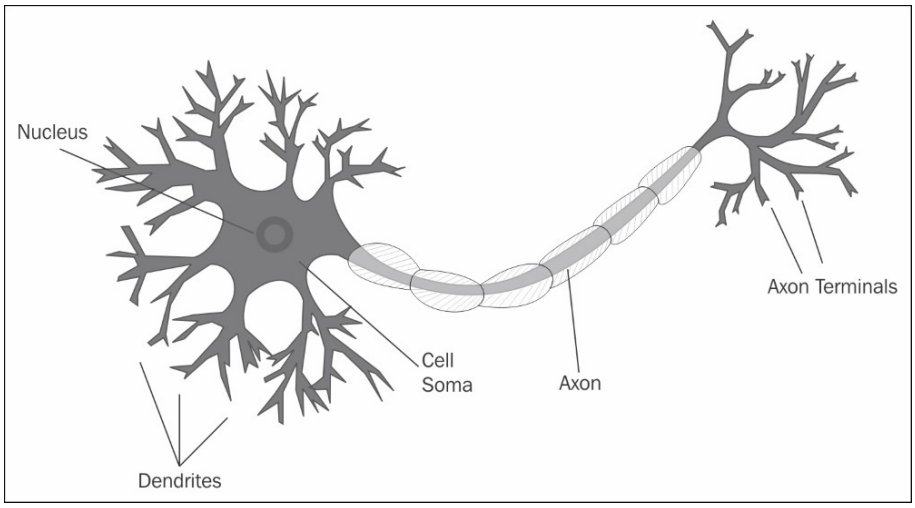


Figure 1.4: Scheme of a Brain Neuron. Densdrites and Axos represent connections between single neurons and layers through which the signals flow.

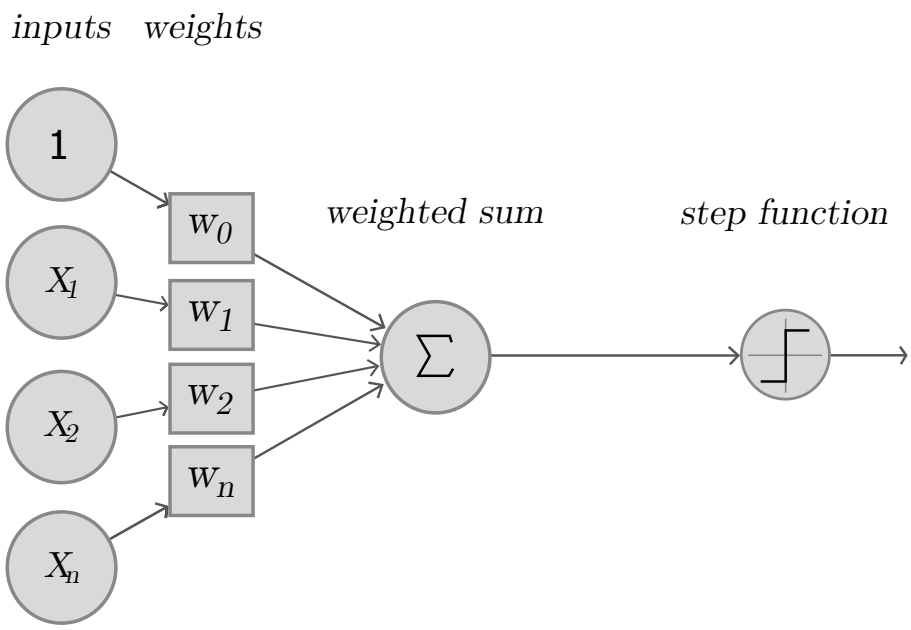


Figure 1.5: Scheme of the Single Layer Perceptron.

Equation 1.2 describes the behavior of a single perceptron and allows us to solve a binary classification problem by providing the probability of a given input of belonging to a certain class. In particular, the output can be either be 1, if the input belongs to the class, or 0 if not. To represent multi-class classification problems, instead, we need to use a number of perceptrons which depends on the number of classes to solve [34]. In this case, perceptrons are organized in a chain where each has its own weight vector and the final prediction is done with a one-vs-rest strategyIn this case, perceptrons are organized in a chain where each has its own weight vector and the final prediction is done with a one-vs-rest strategy [35].

In multi-layer perceptrons (MLPs), the layers between the input and the output are referred to as hidden layers and might be perceptrons themselves. In this case, we say that the model contains N nodes, where N is the total number of hidden layers. Each layer in a MLP is associated with a weight matrix and an activation function which is applied to its output (Figure 1.6).

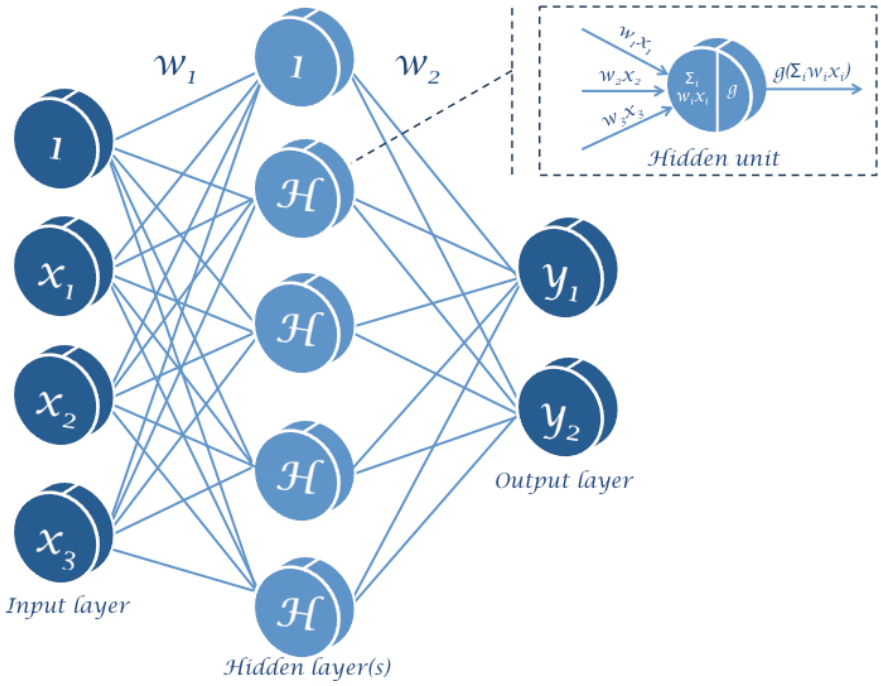


Figure 1.6: Scheme of a Multi-Layer Perceptron.

Suppose to have a l layers multi-layer perceptron, then $l-1$ transformations will take place between the layers. IfW_n,..., W_{l-1} are weight matrices, then an MPL can be described as:

$$f(x) = g(W_{l-1}g(W_{l-2}...x)).$$

This first perceptron and its multilayer version was nothing but an attempt to simulate the process occurring in the human brain. As previously mentioned, the single perceptron was in fact way far from reproducing the complexity of a neural network and from the modern complexity of multilayer neural network. This attempt to simulate the brain processes was however remarkably important and posed some of the bases for future research in this field.

ADALINE and MADALINE (1959)

Stanford, 1959, Bernard Widrow and Marcian Hoff developed ADALINE and MADALINE (Multiple Adaptive Linear Elements). The main objective of ADALINE - and its multi-layer version MADALINE - was that to recognize binary patterns [36]. This neural network was the first neural network applied to real world problem. By reading a stream of bits from a telephone line, it was able to predict the next bit using an adaptive filter that eliminates echoes on the lines. The main difference with the Perceptron was in the fact that ADALINE used the weighted sum of its input to update the layer's weight instead of using the output.

Learning Weights Approaches (1962)

Widrow and Hoff develop a learning procedure that analyzes the input value before before being adjusted by the neuron weights (and turning to either 0 or 1). This learning procedure used the rule Weight Change = (Pre-Weight line value) * (Error / (Number of Inputs)) [37] to adjust the weight values and distribute them to the remaining nodes. This theory is based on the idea that, if the error is correctly weighted, a perceptron can better modify and correct its learning. Despite the initial interest for neural network development, papers suggesting the non-extensibility of single-layer neural network to multiple layer ones were published in this period. In addition, researchers in this field were using shared weights across all the layers, which basically prevented them from learning different representations and features from data. These approaches resulted in failure when employed in real applications and consequently in a shortage of funding.

Birth of Computer Vision (1965)

In these years, computer scientists were also focusing on mimicking human vision on computers. First experiments were carried out with the aim of having a computer "see" objects in multimedia files (e.g. images and videos) and report such information. Before the birth of Computer Vision, image analysis was performed manually with the employment of x-ray and MRI. All-in-all, technology in these years was very rudimentary, however it posed a first base for future development, which significantly improved in the early 21st century with the introduction of more complex algorithms [38].

Nearest Neighbour Algorithms (1967)

This family of algorithms allowed a program to perform a very basic pattern recognition, thus marking the beginning of pattern recognition algorithms research. Nearest neighbor was firstly used for mapping routes and proved efficient in finding solutions to "Travelling Salesperson" problem [39]. The aim of the algorithm was to find the optimal route for a salesperson who had to program an efficient order to visit a set of potential customers.

Backpropagation (1970s – 1980s)

With the advent of multilayer neural network, researchers were trying to find solution for applying Widrow-Hoff rule to multilayer networks. Three independent groups from the Psychology Department of Stanford university were approaching possible solutions. The idea was to send a feedback to the previous stages and layers of the neural network to dynamically adjust the weights in order to have a more efficient learning. This solution was called backpropagation [40]. Backpropagation allows a network to adjust its hidden layer weight to adapt to new situation and data. Nowadays backpropagation is still in use to train deep neural networks, therefore we will provide a deeper insight on the topic.

When designing a feedforward neural network, what we create is a model that accepts an input x and produces an output $\hat{y}$. In this setting, information provided by x propagates to the hidden units of the first layer, then flows through all the others until the output unit is reached and a cost $J(\Theta)$ is calculated on the goodness of the prediction. In order to train the network and achieve better performance, the information from the cost is allowed to flow backwards throughout the network and weights are updated so that, on the next run, the cost would possibly be lower. The updates for the weights are obtained by calculating the gradient given a certain cost. This operation can become very expensive in terms of numeric computation, therefore we employ backpropagation algorithm as a mean to make it simpler and very inexpensive.

Back-propagation algorithm has its foundation on a rather simple mathematical property of derivatives, called the *chain rule of calculus.* By this property, the derivative of a function made by the composition of other functions is given by the product of other known derivates. Let $f(x) : \mathbb{R} \to \mathbb{R}$ and $y = g(x) : \mathbb{R} \to \mathbb{R}$, then the derivative of their composition $z = f(g(x)) = f(y)$ is given by:

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x}. \tag{1.3}$$

When we transpose this calculus from the scalar case to the multi-dimensional one we obtain partial derivatives of z with respect to all the elements of x. In particular, suppose $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$ and let $\boldsymbol{y} = g(x) : \mathbb{R}^n \to \mathbb{R}^m$ and $f : \mathbb{R}^m \to \mathbb{R}$, then the partial derivatives of $z = f(\boldsymbol{y})$ are given by:

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}. \tag{1.4}$$

If we write it in vector notation:

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^T \nabla_{\boldsymbol{y}} z = J_{\boldsymbol{y}}(\boldsymbol{x})\nabla_{\boldsymbol{y}} z. \tag{1.5}$$

Where $J_{\boldsymbol{y}}(\boldsymbol{x})$ is the Jacobian matrix of $\boldsymbol{y}$ with respect to $\boldsymbol{x}$ and it is a $m\ x\ n$ matrix. What this notation highlights is that the gradient of a variable $\boldsymbol{x}$ can be obtained by multiplying the Jacobian $J_{\boldsymbol{y}}(\boldsymbol{x})$ by the gradient of z with respect to $\boldsymbol{y}$. If we recursively apply this calculus for each node in the network, we have implement the back-propagation algorithm.

Gradient Descent and Optimizers

In most cases, the concept of back-propagation is misused or misunderstood. As we previously described, back-propagation algorithm provides an efficient way to calculate the gradient given a certain cost by recursively considering hidden units in a network. The algorithm, however, does not define a policy to update the weights using such gradient, instead this is done by the *optimizers.* The term refers to the fact that we are trying to

minimize (or maximize) a function $f(x)$ by varying x. Such function is typically referred to as the *objective function* and, given that we want to minimize it, we also call it *cost* or *loss* function (See 1.1).

Most of the optimizers rely on the concept of *Gradient Descent* [41], which takes into account the derivative of a function to determine the direction of the next step to take towards reaching a minimum (or a maximum) in such function. In particular, a minimum point might be local or global. It is called a local minimum if the sorrounding points have higher value, but there exists at least an other point with a lower one. It is called a global minimum if no other points have lower value (although there can be points with the same value, i.e. multiple global minima). A characteristic of local/global minima/maxima is that the derivative in such points is 0. An other type of points which have 0 derivative but are neither minima, nor maxima are the *saddle* points. In general, a point where the derivative is equal to 0 is called a critical or stationary point and it will not provide any information on which direction to take. In the context of Deep Learning it might be very difficult to reach a global minimum, since several local minima or other stationary points might be present and might halt the learning. We therefore seek solutions yielding acceptable performances even if not optimal. In Figure 1.7 we provide an example of how the derivate is used in gradient descent.

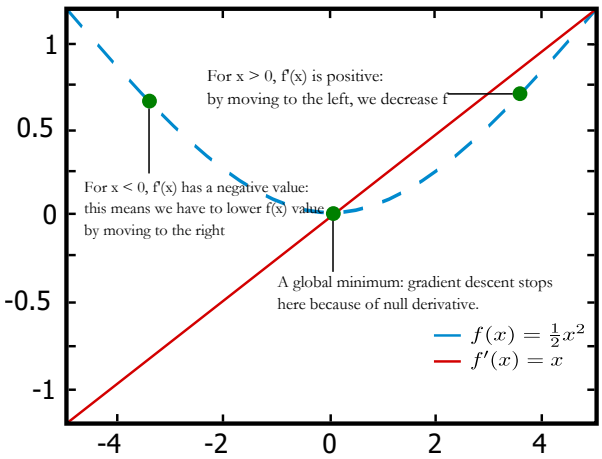


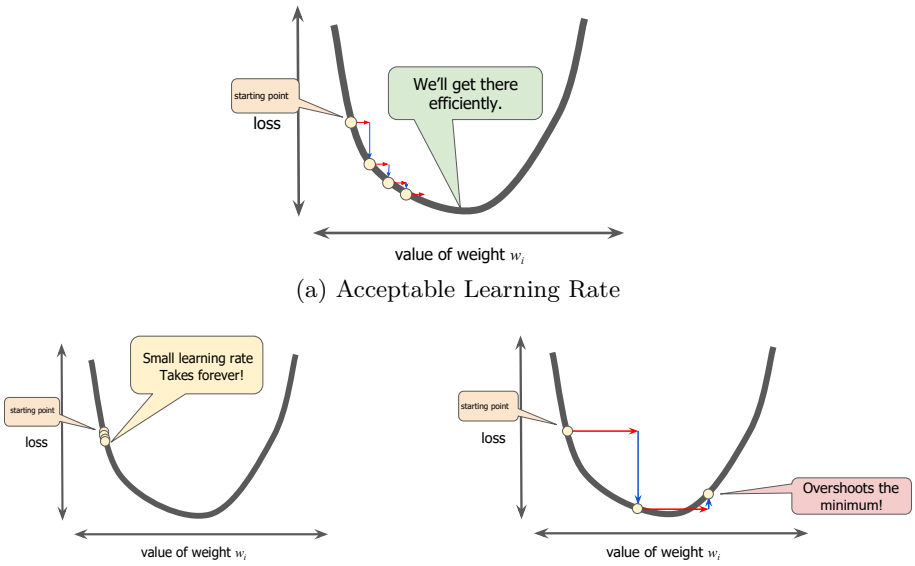
Figure 1.7: Gradient Descent Example. Given a certain position on $f(x)$, in order to reach the central point (a global minimum) we need to move in a direction opposite in sign with respect to the derivative calculated on the same position.

Usually, the goal is to minimize functions having multiple inputs and yet a single output (this is a requirement for the minimization process to make sense). Because of this, we leverage the concept of partial derivatives and of gradient, which is a generalization to a vector where the element i is the partial derivative of the function with respect to input x_i. Whenever all the elements in a gradient are zero there is a critical point. Since we want to find a minimum and the gradient determines the direction of the steepest increase of the

(cost) function, we want to move in the opposite direction. Therefore, Gradient Descent suggests to apply the following modification to the function input:

$$\boldsymbol{w} \leftarrow \mathbf{w} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}), \tag{1.6}$$

where ϵ is a constant — typically in the range $[10^{-1}, 10^{-6}]$ — called *learning rate*. The choice of the learning rate is often crucial and should be carefully taken since it determines the size of the next step. If it is too high we might miss a minimum and the loss might even diverge, if it is too low we might not make significant improvements and never reach a convergence (Figure 1.8).



(a) Acceptable Learning Rate

(b) Too Low Learning Rate

(c) Too High Learning Rate

Figure 1.8: Learning Rate Choice Examples [2]

If we write down the Taylor series approximation of the cost function and perform some calculations, we find out that the optimal learning rate is given by the inverse of the second order derivate. In vector calculus, the second order derivates take the shape of a Hessian matrix. Calculating the Hessian matrix and its inverse at every step during the training phase of a network is computationally very expensive, therefore what its typically done is to define the LR within a certain range and eventually perform some adjustments during the training iterations.

Stochastic Gradient Descent. The update mechanics explained so far refer to what is considered the *vanilla* Gradient Descent, also referred to as Batch Gradient Descent. In this setting, the algorithm computes the gradient on all the samples contained inside the training set before performing an update, however this is most often unfeasible or unwanted due to the high number of samples.

At the extreme opposite we find a variant which uses single samples randomly taken from the dataset: this is the Stochastic Gradient Descent or SGD. This optimizer performs an update on the weights for each sample encountered and, because of this, may cause very violent fluctuations on the cost function. This has both a positive and a negative side. On one hand the frequent updates and oscillations allow SGD to explore a wider solution space, therefore making it more probable to identify better local minima. On the other hand, SGD might keep on overshooting and eventually skipping an exact minimum, thus making convergence more difficult.

A third variant takes the best from the previous two ones and puts them together into randomly selected mini-batches, hence the name Mini-Batch Gradient Descent. Often referred to as SGD, this version of the algorithm estimates the gradient value by calculating the average gradient on m (mini-batch size) samples. In this way, since a wider range of samples is taken into consideration, a more accurate gradient estimate is achieved, while the update frequency remains high. This method might lead to a slow asymptotic convergence, however the rapid progress in the initial steps make it a valid choice with respect to its vanilla version. Other extensions of SGD use momentum or nesterov accelerations to achieve higher convergence rates, yet the underlying algorithm remains the same.

Adaptive Learning Rate Optimizers. Adagrad [42] is an other famous optimization algorithm, which relies on a simple idea: adjust the learning rate such that parameters relating to the most frequenty occurring features undergo lighter updates, while those relating to infrequent features are updated more heavily. Because of this setting, it is mainly suitable for very sparse datasets. Adadelta [43] is an extension of Adagrad which tries to reduce its aggressive, inesorable learning rate decrease.

Finally, an other adaptive learning rate optimization algorithm is Adam [44]. Adam stands for Adaptive Moment Estimation and it assigns individual learning rates for each parameter. The concept is similar to that of momentum in which older gradients are conveyed into an exponentially decaying average, however it also includes concepts from Adadelta and keeps a decaying average of the squared old gradients too. Its been shown by the authors that Adam performs very well compared to other adaptive learning algorithms and is nowadays among one of the most used ones.

Loss and Concept of Risk

Deep learning algorithm often operate in a supervised learning setup. This means that together with the training data we know all the corresponding labels, it is therefore possible to provide a measure of how good the algorithm is performing on a given pair (sample, true label). We therefore define a *Loss* or *Cost* function as the function providing such a quality measure. Given a dataset $D = \{(x_1, y_1), ..., (x_n, y_n\}$ where $x_i \in \boldsymbol{x} \subset \mathbb{R}^d \quad \forall i \in [1, n]$ represent data samples and $y_i \in \boldsymbol{y} \subset \mathbb{R} \quad \forall i \in [1, n]$ and a classification function $f(x)$ represent the corresponding label, several possible Loss functions could be defined. Some examples might be:

- *Classification Loss.* This is the simplest loss function. It returns 1 if the classification is wrong, 0 if it is right. using a synchronization software

$$L(y_i, f(x_i)) = \begin{cases} 1 & y_i \neq f(x_i) \\ 0 & y_i = f(x_i) \end{cases}. \tag{1.7}$$

- *L_1 Loss.* Also called *Least Absolute Deviation (LAD)* returns a loss value linear with respect to the errors.

$$L(y, f(x)) = \sum_{i=1}^{n} |y_i - f(x_i)|. \tag{1.8}$$

- *L_2 Loss or Least Square Errors.* This cost function considers the square of the difference between the true label and the predicted one. It is typcally preferred to L_1, however its measure quality might suffer if the dataset contains outliers.

$$L(y, f(x)) = \sum_{i=1}^{n} (y_i - f(x_i))^2. \tag{1.9}$$

- *Hinge Loss.* It is manily used in classifiers which define and maximize a *margin* such as Support Vector Machines. Hinge Loss penalizes all the predictions which are less than 1, i.e. those which fall within the margins.

$$L(y, f(x)) = \max(0{,}1 - y \cdot f(x)). \tag{1.10}$$

- *Log Loss* or *Cross-Entropy.* Perhaps the most common loss function, it is used to calculate the performance of a classification which produces class probability distributions as output. The mesured cost is higher as the probability diverges from the true label.

$$\begin{aligned} L(y, f(x)) &= -(y \cdot ln(f(x)) + (1-y)ln(1-f(x))) && \text{binary case} \\ L(y, f(x)) &= -\textstyle\sum_{c=1}^{C} y_c \cdot ln(f(x)_c), && \text{multi-class case} \end{aligned} \tag{1.11}$$

  where $c \in [1, C]$ is the class number and y_c is a binary operator stating if c is the correct label for the current observation.

The previous represent just a subset of the existing loss functions, altough they are among the most used ones. In general, choosing the right loss function is crucial when designing the learning algorithm for a classifier, also because a certain loss might require a different activation function as the output of a deep learning model.

Very close to the concept of Loss, comes that of *Risk.* The risk is defined as the expected value of the Loss function, therefore proving an estimate of how many mistakes will be made on the predictions. This is an extremely important concept, because if we have

a function determining how good the classification is going to be, then we can use such function as the objective of a minimization algorithm and obtain an optimal classifier. The formal definition of risk is given by the following:

$$R_{L,P}(f) = \int L(y, f(x))\mathrm{d}P(x, y), \tag{1.12}$$

where $R_{L,P}(f)$ is the Risk calculated on the loss function L over a classifier f given a certain probability distribution of data P. In a real case, data is not continuos but rather discrete, i.e. a fraction of the whole, therefore we take a frequentist approach and the integral becomes a sum:

$$R_{L,P}(f) = \frac{1}{n}\sum_{i=0}^{n} L(y_i, f(x_i)). \tag{1.13}$$

A minimum risk corresponds to a better classification performance, we are thus interested in minimizing it an obtaining an optimum:

$$f^* = \arg\min_f R_{L,P}(f). \tag{1.14}$$

In general, we define the *Bayesian Risk* as the expected value of the loss relatively to the classifier which minimizes it. In other words, it is the optimal risk and is denoted as:

$$R^*_{L,P} = R_{L,P}(f^*). \tag{1.15}$$

In the previous equations we do not take into account the data provenience, if either from training or from testing. We are, however, interested in knowing the performance estimate on test data since that would be the real application scenario. Nevertheless, test data distribution is not known in advance. Why not using the *Empirical Risk* $\hat{R}_n$, i.e. the risk of making mistakes on training data instead? Training data distribution is in fact known a priori and by the law of large numbers, the frequency value tends to the expected value when $n \to \infty$, therefore by taking a large number of training samples the empirical risk tends to the real risk, which can be minimized so to get close to the Bayesian Risk:

$$\begin{aligned} \lim_{n\to\infty} \hat{R}_n(f) &= R_{L,P}(f) \\ \inf_f \hat{R}_n(f) &= R^*_{L,P}(f) \qquad \text{for } n \to \infty. \end{aligned} \tag{1.16}$$

Although this seems to be the right path to follow, minimizing the empirical risk leads to having a classifier composed of a set of delta functions perfectly matching training samples and systematically mistaking on test ones. This problematic is commonly referred to as *overfitting*. Because of this, using the empirical risk in this way is very dangerous and perfect minimization is hence discouraged. Instead of this, we often adress an other minimization problem that is the *Structural Risk Minimization* [45, 46]. Structural risk introduces a regularizing unit which is in charge of controlling the trade-off between the empirical risk minimization and the training-vs-test error gap, therefore avoiding to fall

into overfitting.

Autoencoders (1987)

It is unclear, due to terminology differences which have slowly converged over time, if this is the correct date for autoencoders birth. Nevertheless, in this year the autoencoder concept was for the first time applied to ANN [1, 47]. The original idea was that to perform dimensionality reduction or feature learning. Nowadays however, they have been extensively applied to Generative Adversarial Models and as a basic structure for network architectures able to reconstruct the input. An autoencoder is in fact a neural network which is trained to "copy" its input to its output. A simple compy operation is of course meaningless, therefore autoencoders are in some way constrained to learning more general reconstructions.

In its core, an autoencoder can be subdivided into two modules, from which its name is derived. The first one, $h = f(x)$, performs a feature selection on the input or, in other words, it learns its *code* and is therefore called *encoder*. The second one $y = g(h)$ tries to reconstruct the input basing on the features - often the most important ones - extracted from the input by the encoder. Since in some way it reverts the job done by the encoder, this part is called *decoder*.

Generally speaking, h and y can be single-layer neural networks (and so they were in their original implementation) however, just like any other feedforward neural network, autoencoders can largely benefit from greater depths. In fact, most recent implementations exploit depth to learn more powerful data representations (Figure 1.9).

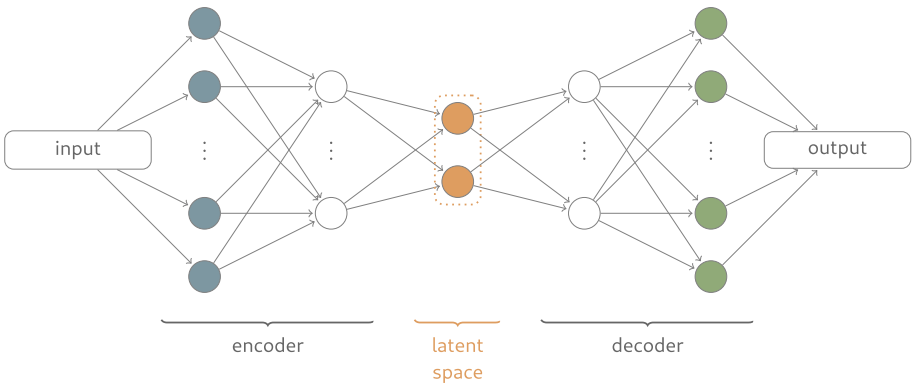


Figure 1.9: Deep Autoencoder Example. In the scheme, the latent space is also known as *bridge* and represents the interface between encoder and decoder [3].

Convolutional Layer (1989)

The concept of Convolutional Layer was first brought up by LeCun et al. [48] and was applied to build a neural network architecture able to recognize digits in a zip code. The

main idea was that of performing an *undersampling* of the input image by means of a grid-like unit called *kernel.* By being repeteadly applied on different points of the image, this unit enables the most important features to proceed to the next step in the network, while leaving the least important ones behind. Nowadays, we call these kind of architectures Convolutional Neural Networks or CNNs.

The name *convolution* comes from the operation employed by the network which mainly comes from mathemitcal and signal processing fields. Suppose f and w are two real and integrable functions, then their convolution, denoted by the symbol $*$, is defined as:

$$s(t) = (f * w)(t) = \int_{-\infty}^{+\infty} f(\tau)w(t-\tau)d(\tau). \tag{1.17}$$

In other words, a convolution is the integral of one function multipled by the inversed and shifted version of a second one. The result is a function s which provides a description of how the shape of one is varied by the other.

To make an example, imagine that we want to estimate the position of a moving car. Suppose that information about its current position is provided by a GPS system, which is somehow noisy: because of this, we want to obtain a cleaner estimation of the position. To do so, we perform a weighted average of the position measurements, also taking into consideration the fact that the latest measurements are more relevant than older ones for the estimate. If the car position is described by the input function $f(t)$ in time domain while the weighting function is $w(t)$, we can use equation 1.17 to obtain function $s(t)$ describing the weighted average car position. Note that, since it is inverted before the multiplication, $w(t)$ needs to be 0 for negative time values, otherwise the estimation will consider future car positions which is impossible.

In real use cases, however, data streams are not continuos but rather discrete. In this case, we switch the integral in equation 1.17 with a sum an obtain the following:

$$s(t) = (f * w)(t) = \sum_{a=-\infty}^{+\infty} f(a)w(t-a) \qquad \text{with } a \in \mathbb{Z}. \tag{1.18}$$

When applied to convolutional neural network, f is usually known as the *input*, w as the *kernel* and s as the *feature map.* Moreover, we often use convolutions on multiple axis simultaneously. When considering a 2D image (input) for example, we typically want to use a 2D kernel too. In this case we use a 2-dimensional version of equation 1.18 in which we also leverage the convolution commutativity property (which comes handy in practical implementations):

$$\begin{aligned} s(t) = (f * w)(i,j) &= \sum_m \sum_n f(m,n)w(i-m, j-n) \\ &= \sum_m \sum_n f(i-m, j-n)w(m,n). \end{aligned} \tag{1.19}$$

While the previous one is the mathematical definition of convolution, in machine learning

applications this term is often used to denote an other operation which is instead named *cross-correlation*. The main difference between convolution and cross-correlation is that the latter does not *flip* the kernel. Furthermore, as the name might suggest the cross-correlation is a measure of similarity between two signals and is commonly used to search if a short known signal is present into a longer one. Keeping this in mind, if we consider that kernels - or *filters* - in a convolutional network are trained to recognize specific features into bigger images then cross-correlation seems to be more adequate than the actual convolution. Cross-correlation operation is defined by the star symbol $\star$ and the formula for its discrete 2D version would then be:

$$s(t) = (f \star w)(i, j) = \sum_m \sum_n f(i + m, j + n)w(m, n). \tag{1.20}$$

In practice, a convolutional filter in a convolutional neural network is implemented by a window (the kernel) - often a 3x3 square matrix in image related applications - which is slided over the input. The values encountered in each step of the sliding are multiplied by the corresponding ones in the kernel and summed together to form a new value. The result of this operation is a set of feature maps (one for each convolutional filter) in which every cell contains the result of one convolution. In Figure 1.10 it is shown an example of the calculation. In the 2-dimensional case, we would have N x width x height feature maps stacked together to form a 3-dimensional matrix which we call *tensor*. This tensor typically contains a down-sampled version of the original input in which the features considered more important by the network are condensed and carried towards the next layer.

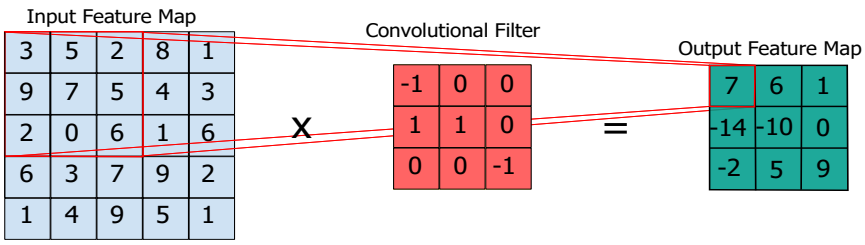


Figure 1.10: Computation Example in a Convolutional Layer. The cells in the red bounding box within the input feature map are multiplied by those contained in the convolutional filter, the values are then summed and form the number 7 in the output feature map. The rest of the output values are calculated in the same way by sliding the window horizontally and vertically on the input map.

Research Restore and Deep Learning Re-Branding (1990 - 2006)

In these years, we assist to a general approach-shift, which passed from a *knowledge driven,* where the "intelligence" came from hard coded knowledge, to a *data driven* one, where data became itself the source for new knowledge.

As for what concerns Neural Networks, research was still being carried out, but it was reduced to very few survivors. ANNs were not achieving good results and in seemed impossible to train deep multi-layer networks, whereas classical machine learning methods kept on achieving better performance. In particular, a paper about soft-margin SVMs (Support Vector Machines) - which are still a standard nowadays - was published in this period [49] and reported very low error rates on the standard MNIST character recognition dataset. SVM's success and ANN's continuos failures brought a depression period for the field: papers about Neural Networks were being rejected continuosly by journals, grants were nearly impossible to obtain and at some point it even started to not being consider a Machine Leaning branch anymore.

This depression period lasted until 2006, when Geoffrey Hinton et al. [50] — who managed to obtain fundings from the Canadian Institute for Advanced Research (CIFAR) — published a breakthrough paper and coined the term "Deep Learning". The term underlined the possibility, given by the new weight initialization system they presented, of training Deep networks in an effective manner. In particular, their solution was able to reach state-of-the-art performance on MNIST dataset.

Although this publication is considered as the most important for research restore, what made the difference was company's renewed interest in the field. In particular Google, IBM and Microsoft increased their investments in view of Deep Neural Networks being applied to speech recognition tasks.

Deep Learning Revolution On Computer Vision Tasks: AlexNet (2012)

Since 2006, research in deep learning field rocketed up. It is, however, not until 2012 that the so-called Deep Learning Revolution took place, when one of Hinton's PhD student — Alex Krizhevsky — presented his solution to a computer vision challenge which involved image recognition and classification: the ImageNet Large Scale Visual Recognition Challenge [51]. The solution presented by Krizhevsky consisted in a very deep convolutional neural network which was called AlexNet [52] (after its inventor) and was able to achieve a top-5 error of 15.3%, outperforming the second in ranking by over 10.8 percentage points. This outbreaking result was made possible thanks to the advances in Nvidia's GPU boards and CUDA API, which were used by Krizhevsky et al. to optimize (and actually make possible) the training speed of such a deep network (8 layers).

Although research in deep learning (DL) has extended to several other fields, most of its application still regard computer vision tasks, in which it has established as a de-facto standard by outperforming other classical approaches to the same problems. Nevertheless, the relation between computer vision and DL is not unilateral: while DL is being employed to solve classification, recognition, segmentation and other computer vision problems, several computer vision methodologies are used to aid CNNs learning by preprocessing data before feeding it in.

Image Preprocessing and Data Augmentation

When properly used, image preprocessing techniques can sensibly enhance the performance of a CNN (but also of other machine learning methods). Image processing is a sub-field of computer vision which focuses on 2D images by transforming them into other slighly-different ones. By applying some of these transformations to data samples before feeding them to a convolutional neural network we can increase the range of "seen" images and consequently enhance the network's ability to generalize its representation for the various class labels, as if more samples were provided in the first place: hence the term data augmentation. Other usages of image (pre-)processing regard the manipulation of source data to be considered as "original": it is possible to extract objects from a wider image or to simply mask some parts of it by means of filtering techniques. Some common image processing techniques are hereby presented, while a visual example is provided in Figure 1.11

- **Horizontal/Vertical Flipping.** As the name suggests, it consists in applying a vertical or horizontal "flip" to a 2D image, as if a mirror was put on a side or on top of it.
- **(Random) Cropping.** Crops are selected from the original image in a random way. This augmentation technique might be problematic since the crop might select an area of the image which is not significant to the semantic meaning of the class it should represent. It should therefore be used with care.
- **Blurring.** Many variants exists, such as GaussianBlur, LogaritmicBlur, LinearBlur, however the outcome is conceptually the same: a blurred version of the original image.
- **Noise Addition.** Trasforming an image by applying some random noise results in no changes for the human eye as long as the number of modified pixels remains low. This, however, does change a lot for a CNN, which see the images in terms of pixel values. In particular, recent studies report the vulnerability of deep learning-based systems to image noise injection or information loss due to compression [53, 54, 55] and report the regularizing effect of adding such noise during the training phase.
- **Saturation Shifts.** It consists in varying the saturation levels of an image so that the resulting one would still look like the original one, but with different color intensities.
- **Brightness Shifts.** As the name suggests, light values in the image receive (random) shifts.
- **Geometrical Transformations.** In general, an image might be rotated, inclined or distorted: if these tranformations are not destructive, they are good candidates for image augmentation.

The previous one was just a short list of possible image augmentation techniques, however they are among the most used ones. Augmenting the dataset is a key strategy to use when considering CNN training and, whenever possible, it should always be considered.

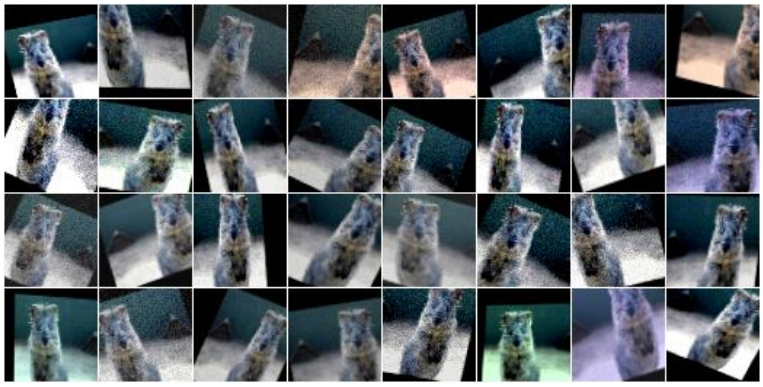


Figure 1.11: Image Augmentation Examples. The presented images were obtained by randomly applying multiple image transformations using ImgAug Python library [4]

ResNet and U-Net for Semantic Segmentation (2015)

ResNets [56] constitute one of the major advances in modern deep network architecture design. The problem addressed by ResNet was that adding extra layers to CNNs often resulted in a poorer outcome, which is somewhat counter-intuitive. As a matter of fact, the inclusion of extra layers usually led to higher training errors due to several related problems which restrained the network from converging. The solution proposed by He et al. was to add shortcut connections between the layers: such shortcuts allow to combine the input of one layer (i.e. tensors before they are processed) with its output [57]. In this way the networks retains some of the original input features while allowing an easier gradient flow during the backward pass. He et al. also address some architectural design issues and propose the so-called *bottleneck* convolution setting.

During the same period, an other CNN architecture - the U-Net [13] - specializing in in biomedical applications was developed. U-Net embeds an autoencoder structure, introducing an equal amount of down-sampling and up-sampling layers as well as the employment of skip connections. In particular, U-Net was dedicated to a specific computer vision task: Image Segmentation. Image segmentation is a technique used to understand what is inside a given image on a pixel-wise level. It differs from image recognition or object detection from the fact that the former assigns a label to the whole image, rather than to its pixels, and the latter draws bounding boxes around each object. By means of image segmentation we can decompose an image into meaningful parts and achieve more fine-grained understanding. Moreover, Image segmentation can be subdivided into two subtypes. *Semantic* Segmentation consists in dividing an image into different regions by singularly classifying each pixel as belonging to a certain class. Its concept is different to that of *Instance* Segmentation, where objects are identified as single entities over than from the semantic meaning they carry.

As for medical imaging analysis, semantic segmentation is particularly useful as a mean to make localization easier or automated. The output of these kind of networks is typically a

tensor matching the width and height of the original image and having N-classes channels, where every channel reports the probability that each pixel has to belong to such class. The final outcome, is a segmented image where each color corresponds to a different class (Figure 1.12). Medical imaging analysis in not the only application field for semantic segmentation. As a matter of fact, several efforts are being done into applying semantic segmentation to autonomous driving cars or scene understanding, as in one of this work topics (See Section 1.2).

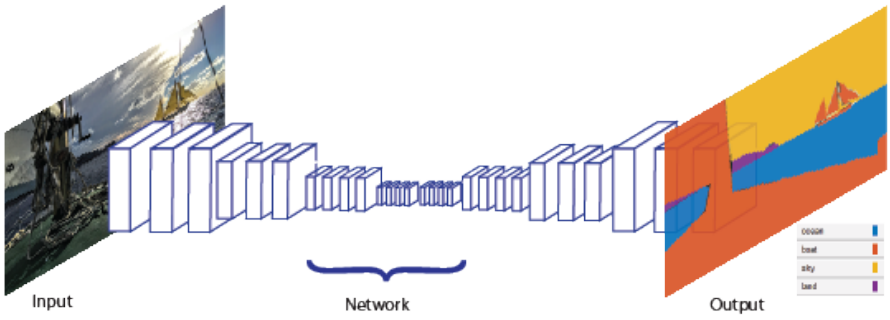


Figure 1.12: Semantic Segmentation with Autoencoder Example [5]. The input images is downsampled within the encoder and reconstructed when passing through the decoder.

1.2 Project Overview

In this chapter we will see an overview on the project this work belogns to. We will briefly talk about the entities supporting this project and see how the workload was split in order to achieve its specifications and goals. Finally, we will have a look at the equipment involved in the data aquisition phase.

1.2.1 Stakeholders

The whole research project was born from the collaboration of three main stakeholders: ReachU, Tallin Technical University (TalTech) and Archimedes Foundation. ReachU is a company specializing in geographic information systems, carthography and location based solutions for tourism, telecommunication services, security and smart cities. Archimedes Fundation is an independent entity established by the Estonian government and is in charge of imlementing and coordinating both national and internation projects in the fields of education, training and research. TalTech University Software Sciences Institute is the "performer" employing its research teams to carry out the development of new solutions towards the achievement of the projects objectives.

1.2.2 Description

In the last decade, spatial content market has not changed in its background. Most of data management processes still employe an orthogonal photo base for data generation, verification and usability. At the same time, the need for deatailed digital data has increased several times. With this research project, the main target is to influence the whole spatial data management process and produce a next-level 3D spatial base layer. In order to look at the whole picture, the research will include all the steps involved in the spatial data management process, i.e. data generation, processing and integration in a comprehensive platform.

Formally, the project is entitled as "Applied Research for Cost-Effective Compatible Geodetic Accuracy 3D Spatial Data Infrastructure" and has the aim of exploring and building a new infrastructure for spatial information to manage, standardize and integrate a wide range of data from different sources, while maintaining a high level of data accuracy. More specifically, the ultimate objective of the project is to create a Spatial Data Infrastructure platform able to serve several different applications by means of an API system. The SDI platform will provide 3D mappings of urban environments starting from multiple datasets which include pictures shot by a 360° camera system (composed by 6 HD cameras), LIDAR recordings and institutional databases. In order to achieve this, the project has been split into 5 distinct subjects.

- The first subject is oriented to the creation of a point cloud image dataset. Each of these images shall be classified and properly colored starting from data collected with MMS (Mobile Mapping System) and UAV (Unmanned Aerial Vehicles) systems. The whole process comprehends several sub-steps, among which the creation of an automatization tool for the coloring and classification tasks.

- The second subject regards information extraction from othogonal, panoramic and point cloud images. These images mostly contain road scenes in a rural (and sometimes urban) environment. The main interests would be object identification and other road attributes such as traffic signs, road signs, road area and size.

- The third subject is based on the dataset produced whithin the first subject. Its objective is that of creating varying-quality 3D urban models in an automatized way.

- The fourth subject will have as an output an integrated version of the first three subjects outputs with external databases. Such databases originate from national entities and contain topographic maps, zone maps, demographic maps and so on.

- The fifth and last subject aims to creating and describing an SDI platform with the achieved results. Access will be made possible by a thoroughly documented REST API system.

The whole project will have a three years duration, starting in April 2019 and ending in April 2022.

1.2.3 Data Aquisition Equipment

Most of the data used in the early stages of the project is directly collected by ReachU-employees. To do so, Reach-U developed a Mobile Mapping System (MMS), consisting in a car fleet (Figure 1.14) where each car is equipped with a LIDAR scanner, a 6-cameras system and two GNSS systems with INS sensors wrapped in a single device. All the sensors need to be mounted on a rigid frame composed by an iron rod having the camera system on top, the LIDAR scanner halfway up and the GNSS/INS on the base (Figure 1.13). Furthermore, these components need to be coordinated and time synchronized. An on-board terminal is in charge of doing so by means of a synchronization software developed by Regio/Reach-U; the whole process averagely generates 1TB of data each 100 km: each car of the fleet is able to generate up to 3TB during a single working day. Data collected in this way is already being used by the Estonian Road Administration by means of an online tool named EyeVi.

Figure 1.13: EyeVi Data Collection Equipment.

Figure 1.14: MMS Fleet. Data collection equipment is mounted and fixed on the car tops.

GNSS Positioning System The used Global Navigation Satellite System (GNSS) device model used for data collections is Advanced Navigation's Spatial Dual (Figure), which mounts 2 GNSS receivers and offers 10cm positioning accuracy. Car orientation, tilting angle and other information are given by a INS (Inertial Navigation System) sensor which is combined with the GNSS.

Figure 1.15: Spatial Dual by Advance Navigation [6].

LIDAR system LIDAR — also known as "laser scanning" or "3D scanning" — stands for LIght Detection And Ranging. This technology is widely used in the automotive industry, in UAV/MMS mapping, in robotics and other sectors, and makes use of eye-safe laser beams to monitor the surrounding environment and recreate it in a digital 3D one.

The device used in this case is the Velodyne VLP-16 sensor (Figure 1.16) which includes two LIDAR systems capable of measuring up to 100 meters with an accuracy of ± 3 cm. The scans are performed in a vertical field of view of 30 ° (+15° from a LIDAR system and –15° from the other). For this reason, the device was mounted with an inclination with respect to the horizontal line so that the most interesting part of the panorama, i.e. the road surface and most of the buildings, was completely covered. Finally, the horizontal / azimuth field of view covers 360 degrees, however the area covered has a blind point at the 180° angle which ranges by ± 5° given by the presence of the iron rod on which the system is fixed.

Figure 1.16: Velodyne VLP-16 ("Puck") LIDAR sensor [7].

Camera System The MMS employs Ladybug5+ as spherical imaging system (Figure 1.17). A 30 MP Resolution is achieved by the combined action of 6 high-sensitivity 5 Mp CCD global shutter sensors (Sony IMX264). Covered view is the 90% of a sphere and maximum height from the ground is 250 cm.

Figure 1.17: Ladybug 5+ image from flir.com website [8].

Data Collection and Synchronization During data collection process, GNSS and INS data are first combined in order to calculate the trajectory of the vehicle. This trajectory contains the 3D coordinates of the platform and the orientation/attitude angles along with timestamp and other supplementary information (e.g. velocity, acceleration, etc.). Raw LIDAR data (i.e. thedistance to an object measured with laser beams and angles at which the laser beam was sent out) is combined with GNSS/INS trajectory to calculate a point cloud. Point cloud contains the 3D coordinates of each point and the intensity value of surface reflectivity. Panoramic images are georeferenced with GNSS/INS data, thus allowing the placement of panoramas in their correct place on a map and view 360° images (by stitching them together). In Reach-U system, the panoramic images are also projected to the ground to create an orthogonal image of the road surface, which can be used to monitor and quantify defects on the road (e.g. pot holes, cracks, etc.) [58]. Camera and LIDAR data can be combined to create an RGB colored point cloud. For this purpose, the images are projected to the point cloud and the corresponding color extracted. Some software products able to do this exist, however it might prove difficult to integrate them in the automatic workflow. For creating 3D models and matching different datasets automatically, it will be necessary to detect, segment and classify objects (hence the topic of this work).

1.2.4 Team and Objectives

Within the whole project, our research team was in charge of semantic segmentation of both 2D images and 3D point cloud ones. Moreover, the team was appointed the identification of other road features, such as road width, road floor type and road signs. The workload was further subdivided between the team members (initially composed by 5 researchers)

for organizational purposes, such that the present work focus was set on pavement type detection and on the initialization of a semantic segmentation system for road features.

In particular, our team mostly concentrated on the utilization of Deep Learning methodologies - which are at the base of most Artificial Intelligence applications - to achieve said objectives, hence the name "Team AI". With respect to semantic segmentation, the particular features of interest have been defined and changed during the setup phase, with a consequent production of multiple datasets. In any case, the final concern regarded the detection of poles, traffic signs and informational signs (Section 2.2.1).

Chapter 2

Datasets

In the previous chapter we made an introduction and outlined a background for the topics addressed in this thesis. In particular, we talked about the project it resides in and presented the objectives to fulfil which are that of providing a Deep Learning based solution able to perform road surface type classification and that of setting up a research towards semantic segmentation of road features from given road scenes(section 1.2.4). In order to lay down a suitable work base, Reach-U provided a great amount of data collected from different sources and stored in a shared online environment. Most of the times, however, raw data cannot be used *as is* and an analytical process must be set up to extract the most useful pieces, perform adjustments and organize them in a way suitable to be employed for our Deep Learning solutions. Our starting point is always given by images, however the creation of two different types of dataset will be covered. In particular, we will see how the different nature of the objectives substantially influences the structure of the datasets, which thus require the use of different construction methods.

2.1 An Image Classification Dataset

Among the provided data, we found a set of images - and related pieces of information - to be particularly suited for the classification task. In particular, the following elements were included:

- A set of images, referred to as *orthoframes,* that are constructed from panoramic images collected from a drive, shot from the rear of the car. The orthoframes depict the road and the surrounding area from a point of view orthogonal to the ground (hence the name). The narrower the road the larger the surrounding area and vice versa. For each orthoframe, an U-shaped mask has been supplied and applied. The mask eliminates less relevant (unfocused, distorted) parts of the orthoframe image (Figure 2.1). The orthoframes are organized into directories named in format `yyyyMMdd_hhmmss_LD5` where the directory name indicates the start time of the individual drive. One drive can contain involve cruisings on different roads.

- Each othoframe is supplied with an accompanying VRT file. VRT files are used within Geospatial Data Abstraction Library (GDAL) and are essentially XML files containing metadata that describe various properties of the associated raster image. For present case, VRT files provide the geolocation coordinates of the upper left corner of the orthoframe as well as pixel dimension, making it possible to compute exact geolocation coordinate for each pixel in the orthoframe, if necessary.
- Geolocations of the spots where the orthoframes were shot (camera point locations). This information is contained in a set of files (`locations.*`) that can be parsed using Python Shapefile library.
- Geolocations of the points that constitute the road lines as reported by the authorities. This information is contained in similar shapefiles (`roadlines.*`) and, besides the road line point coordinates, include the road ID, name and other attributes.
- Pavement type information (no pavement, gravel road, pavement road, stone mosaic or unknown) is supplied with the road attribute file (`road_attributes.tsv`). This file contains the road ID that can be associated with the one found in the roadline files and road line segment beginning and end distances from the road line beginning for which the pavement type is valid.

Figure 2.1: Orthoframe (left) and the corresponding orthoframe mask (right)

2.1.1 Data Association and Mask Production for Road Surface

The next step is to produce the masks to extract the part of the road that can be further used for producing the valid image segments for CNN training. To ensure that the extracted image area depicts the road surface and that we have identified a proper pavement type from the records we must do the following:

- Select the camera points valid for current orthoframe. These points are directly used for producing the proper mask later on.

- Select the road ID, road line points and distances of each of these points from the point of origin of the road line valid for selected set of camera points.
- Determine the pavement type valid for the current orthoframe and roadline (must match the road ID) and roadline points distances determined above.

That of filtering the suitable camera points and roadline falling within the geolocational boundaries of a given orthoframe might seem a straightforward task, however some complications arise.

1. Road line points are available at uneven intervals, many of which are much longer than the slice of road depicted in the given orthoframe. To provide a continuous road line for the orthoframe, one needs to insert additional road line points at the edges of the orthoframe. These additional points are obtained by interpolating from the closest existing road line points outside ofthoframe boundaries. This is accomplished with the clipLine function of openCV. Of three road line points depicted in Figure 2.2, only the one at 344.87m, is obtained from the road line shapefile, the other two at the edges of the image are the inserted ones. The distance from the origin of the road line, however, is calculated cumulatively for each road line point, original or not.

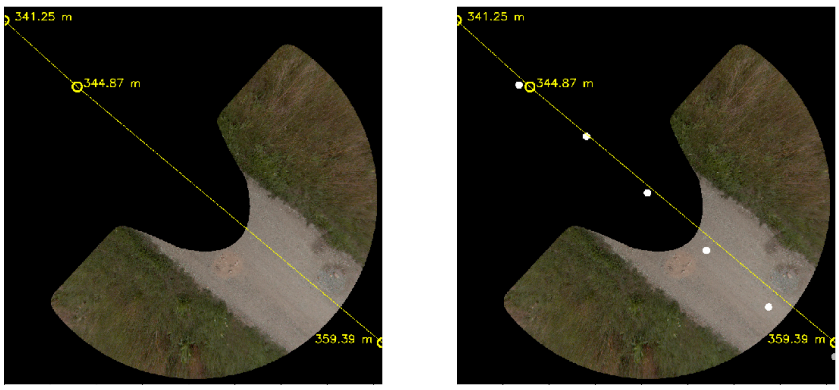


Figure 2.2: Road line alone (left) and road line with the camera points (right) valid for the given orthoframe. This is the simplest case with one road line and one set of camera points.

2. Not all pictured roads have a corresponding road line (at least not in the available shape file). In this case, the road attributes cannot be determined at all.
3. Camera points occur at approximately 3m intervals, thus it is highly probable that there is a number of camera points within the orthoframe. However, two artificial camera points are inserted at the edges of the orthoframe; the same way as the additional road line points were added.
4. There might be camera points originating from different drives (at crossing roads or because of repeated drives on the same roads; this more likely to happen on wider

roads). The camera points do not have a drive ID but they do have an ordering that makes it possible to split the camera points into sequences that originate from different drives and then identify and select the sequence that has a camera point nearest to the orthoframe center (the camera point where the given orthoframe was actually shot).

5. Similarly, there might be several roadlines in the same orthoframe (at crossings or just close roads), therefore, it is important to select the appropriate road line otherwise the road attributes will be improperly defined. If there is more than one road line that intersects the orthoframe, we need to find out which one of those was supposedly followed during the drive. We already have determined a set of camera points that is valid for the current orthoframe. Next we use a subset of them that fall within the orthoframe mask. Given this subset of camera points we browse through all road lines that intersect the orthoframe. For each camera point and the given road line, we choose two closest road line points to the camera point (Figure 2.3). Those three points form a triangle with sides a, b, c. Note that the closest pair of road line points is those for which angles A and B of the triangle do not exceed 90 degrees.

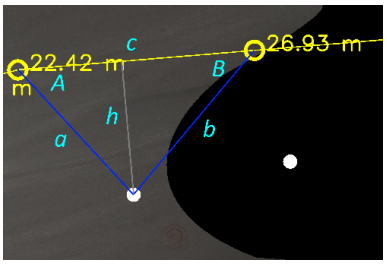


Figure 2.3: Determining the distance h between the camera point and two closest road line points

The distance of the camera point from the road line is given by

$$h = \sqrt{b - \left(\frac{b^2 + c^2 - a^2}{2c}\right)^2}.$$

Each road line can then be characterized by the averaged value of h computed over the valid set of camera points inside the orthoframe mask and most likely, the appropriate road line for the orthoframe is the one which provides the lowest value of this measure. Once all of the above is sorted out, a mask depicting the definite extract of the road pavement with a valid pavement type can be produced. The eventual mask is obtained by drawing a 500 pixel wide line (that corresponds to the width of the car) over the selected camera points. Figure 2.4 shows one such example and the produced mask.

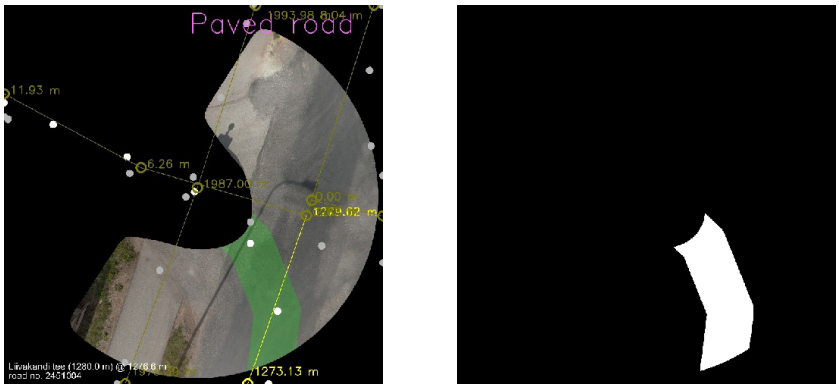


Figure 2.4: There are 5 different road lines and camera points originating from 4 individual drives within the boundaries of this particular orthoframe (left). The relevant camera points are associated with the appropriate road line so the correct pavement type can be identified and the pavement mask (right) is correctly produced.

2.1.2 Dataset Construction

Most of the orthoframes collected for this study, incidentally, represent gravel roads (62.20%), followed by paved roads (~33%). Unpaved roads are represented on 4.42% of the orthoframes, whereas the remaining two classes make up mere 0.39% (Figure 2.5).

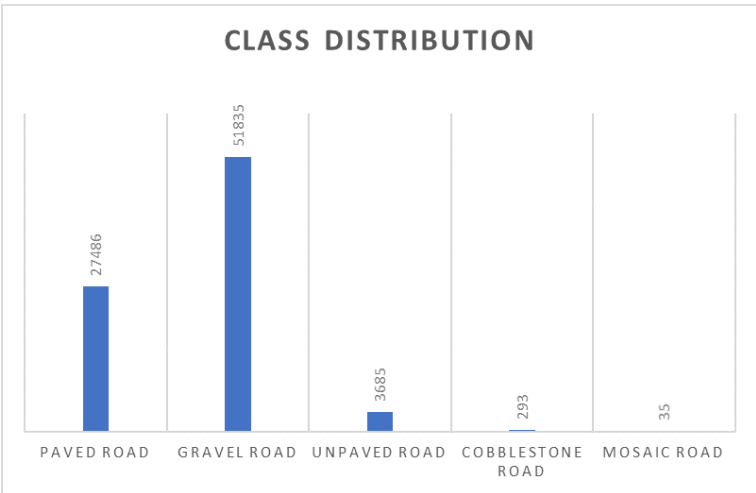


Figure 2.5: Class Distribution. Paved Road and Gravel Road images are preponderant. Unpaved road, Cobblestone road and Mosaic road images are respectively one, two and three orders of magnitude less with respect to the previous two ones.

The very unbalanced class distribution is addressed by leaving the cobblestone and mosaic roads out from the scope of the classification problem, the road surface segments are extracted only for first three classes in Figure 2.5. To produce the training and testing

samples for the CNN, we calculate all possible extractable segments from the masked area using the previously produced pavement masks. The coordinates of the segments are retrieved so that each segment overlaps with its neighbours by half of its own size. Segments that would contain more than 1% of black pixels (i.e. RGB(0, 0, 0)) are discarded. This is a rather costly operation as on average, it takes 1.35 seconds per orthoframe. We have set the segment size to 128x128 pixels. The smaller segment size allows us to obtain more samples from each image, while keeping each samples weight low in terms of memory. Larger segment sizes lead to fewer samples, that would intensify the problems with poorly represented classes. An example of calculated segments is shown in Figure 2.6.

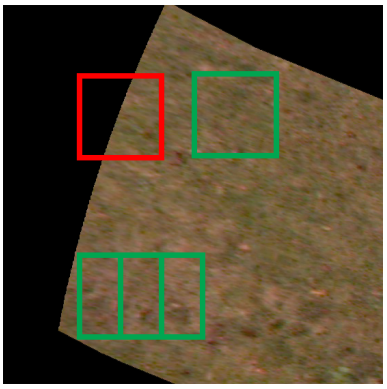


Figure 2.6: Segment Extraction Example. Green boxes represent valid segments, while the red one is an invalid segment that will be discarded.

In result, 10 000 segments for the least represented class – Unpaved Road – can be easily collected. It is also a sufficiently high number, so 10^4 is the limit of extractable segments for all classes. In order to promote the use of a wider range of images, we set a limit of 5 segments to be extracted from each image so the final dataset contains 30k samples equally distributed between Paved, Gravel and Unpaved Road classes. The last step is to split the overall dataset into training and validation datasets with 75%/25% ratio. The sample selection is image-based. This means that in order to decide which segments should be inserted in the training set and which should be in the validation one, we first retrieve all the original images from which the segments were extracted, shuffle the list and randomly select 75% of images. All the segments extracted from these images will be assigned to the training set, while the remaining ones go to the validation set. The reason behind this is that a totally random split into training and validation samples might bias the network towards a certain prediction. If during the training phase, the network processes the segments that originate from the same orthoframe than the validation samples and possibly even overlap with the validation samples, the validation segments would be too familiar to the network. The resulting dataset is named the primary dataset not to confuse this with other datasets we produce later on. One specific characteristic of the orthoframes is that segments further away from the centre of the image appear less focused, an example of such difference is provided in Figure 2.7. To find out if sharpness of a segment has any

effect on pavement detection quality, valid segments are sorted by Euclidean distance from the central point of the image, either in increasing or decreasing order, thus making it possible to simply select the first N (=5) ones. In case of mixed selection, the list is shuffled rather than sorted. Consequently, we obtain further three different datasets, which we call:

- Blurry dataset, containing far-from-the-centre segments
- Sharp dataset, with close-to-the-centre segments
- Mixed dataset, with randomly selected segments

Figure 2.7: Extracted Segments Example. The road segment on the left taken from the edge of the image is blurrier than the one on the right taken from the centre.

2.2 Semantic Segmentation Datasets

Our second task consists in performing semantic segmentation on road scene. To accomplish this task, we select a set of images originally collected by means of a spherical imaging system (Ladybug 5+), which provides 6 different panoramic images from the same scene, one per direction: up, down, left, right, front, back. Images orientated upwards show sky only, while those orientated downwards show almost only the car itself. Left and right oriented images do provide more informative scenes, but still not interesting for our work case. We finally select front and back oriented images, both depicting a road scene but providing different information since elements on the road (such as traffic signs) are presented from multiple perspectives and points of view.

2.2.1 Dataset Construction

Since we are in a supervised learning setup, out first need is to generate a label for each image in the dataset. In a normal image classification problem, where the aim is to assign a class from a given set to an image, a label is simply the name of the class which is representative of the image content. In our case we need to perform semantic segmentation over one image, which requires to have a label for every pixel contained in such an image. As a matter of fact, a label for an image to be semantically segmented is an image itself where every pixel is assigned to a specific value which is mapped to a given class. We call these label images masks or ground truth (GT) images. As a first attempt, we decide to build a simple two-classes problem by creating masks for 32 images. Ground Truth images

are produced by using a graphics software and dividing asphalt (white) from background (black), then they are divided into train and validation set. As a result, we obtain a very small and simple dataset which was used in the early steps of the development of a solution and as a proof of concept.

Figure 2.8: Road image (left) with its corresponding Ground Truth (right)

Further research brought to the discovery of so-called annotation tools which allow pixel annotation of images in a comfortable way. While several tools were available, we opted at last for Intel open source Computer Vision Annotation Tool (CVAT) [59] which provides a web-based interactive annotation system under MIT licence. As a next step, we decide the set of interest classes to be segmented by the network At first, the classes of interest are stated to be 7: road surface, road markings, curbs, cars; anything falling out of the interest area was agreed to be classified as background; as a result we have 7 distinct classes in total.

Along with the classes, we decide on pixel annotation guidelines to be followed while labelling images. Such guidelines are needed in order to ensure that different people performing the annotation would use the same criteria while labelling objects on the images and that the labelling style would as consistent as possible throughout the dataset, since it may vary from one to another [60]. The guidelines state that the operator should, using CVAT tool, zoom into the image 4 times while annotating pixels. Any object which is easily recognizable within this zoom amount should be annotated with one of the classes of interest. With easily, we mean that the object edges should not in any case be indistinguishable by whatever other object is surrounding it. Furthermore, any object appearing on the image borders should be annotated as belonging to one class only if it is unmistakeably distinguishable by the human operator.

Provided panoramic images have 4096x4096 resolution, which is very high when thinking that these should be provided as input to our networks. Moreover, the topmost part of the images mostly contain sky or other objects such as houses which we consider background information; because of this we decide to crop the images by half over the horizontal line, thus keeping the bottom part only. As a result, we obtain reduced size and easier to annotate images (Figure 2.9) in what we refer to as road dataset.

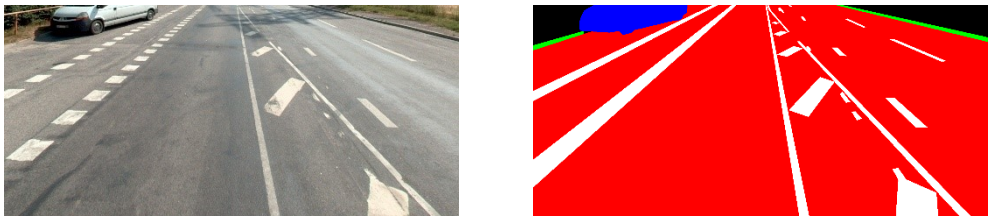


Figure 2.9: Image (left) and its correspondent Ground Truth (right). The topmost part of the original version has been cut out, making it result in a rectangular shaped image.

Due to interest changes, we later decide to reduce the classes to two, being them traffic poles and traffic signs. By counting the background, the total number of classes is now three. We keep the same annotation guidelines as for the previous dataset, however, in order to introduce further clarity, we define traffic poles in the following way:

- A traffic pole is a 4 to 16 meters high pole
- Traffic poles include streetlights, electricity poles and flag posts
- Traffic poles do not include traffic lights

We are gonna refer to this as pole dataset.

Finally, we consider a background-only image as not containing enough information, therefore we decide to discard any image which does not have at least one object different than background. This reduces the total dataset size by ~4.7%. An example of how images and ground truths appear in this dataset is provided in Figure 2.10.

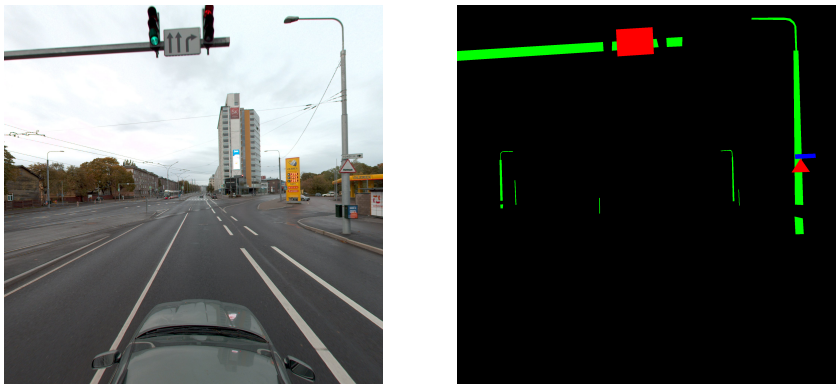


Figure 2.10: Second - and final - dataset example. Differently from the previous case, images in this dataset have kept their original resolution. Labelled objects in the GT (right) also differ. Note that the traffic lights are not considered part of the traffic pole.

Being a very long and time-consuming process, two persons were hired by the company to perform the actual annotation process, hence allowing a parallel research and development of an appropriate CNN. At the end of the annotation process, we obtain 220 images for

the former set of classes and 3482 for the latter set of classes. We collect working statistics over the second annotation process, which are reported in Table 2.1

Table 2.1: Image Annotation Statistics. The whole annotation process required ~39 working days.

Operator Number	Processed Images	Total Working Hours	Images/Hour rate	Images/Day rate
1	1336	112	11.93	~95 (95.44)
2	2274	197	11.57	~92 (92.56)

2.2.2 Berkeley Dataset

Given the need to test our network architectures, we firstly use our two-labels dataset. However, we soon discover the need for a more realistic, multi-label dataset which allows to infer better conclusions on our network performances. While several semantic segmentation datasets exist, many of them refer to different fields with respect to our work case. Although this might not be an issue for network testing and benchmark evaluation purposes, we choose to rely on a related field dataset. Among the existing ones, perhaps the most popular is CityScapes [61] followed by Camvid [60] and KITTI [62], however we finally opt for Berkeley DeepDrive100k (bdd100k) [17] dataset as our reference. BDD100k not only provides a large, diverse benchmark on which to train and test our networks, but also a good example to refer to when deciding the annotation guidelines for the pixel annotation team. Images contained in BDD100k cover several different scenarios and weather conditions along with being classified into 37 different classes: this provides a big challenge for our network, which, by achieving a good performance here, might end up having a better performance on the possibly easier future dataset.

2.2.3 Data Generator and One-Hot Encoding

A proper training of a DeepCNN requires a huge amount of data which needs to be fed to the network. Loading so many samples into memory is by far unfeasible and modern CNNs use mini-batches containing a fraction of them instead. Keras framework provides a useful tool wrapped in a class named Data Generator to enable loading batches of images directly from the directory to the network. In order to grant a correct flow of samples with the corresponding label, source data needs to be organised into a set of folders such that each folder contains only samples belonging to the same class. While this is straightforward in image classification it is not feasible for semantic segmentation.

As previously mentioned in Section 2.2.1 for our task the class is represented by a distinct image (not a simple name) where each pixel is colored with a value determining the class it belongs to: obviously, this file needs to be carried along with the corresponding image when being fed to the network so to provide a ground truth to compare the prediction to. For this reason, we implement an ad-hoc Data Generator able to serve our segmentation

task by providing both the image and its GT. What's more, the output of a network is a belief map in which a separate channel containing probability values between 0 and 1 for every pixel is present for each class. This means that the GT cannot be used as is, but needs to go through a transformation process called "One-hot encoding".

If `n_classes` defines the number of classes in the classification task, by means of this process we obtain `n_classes`-tensors from the 3-channels ground truths (Figure 2.11). Finally, in order to reduce the latency introduced by image retrieval and one-hot encoding, we implement a multi process system within the Data Generator which allows us to achieve up to 9x speed improvement with respect to its single-process version. Moreover, in order to avoid having the network learn unwanted sequences in data, we implement an Index Pool class shared among different processes and accessed in a mutual exclusion regime. The Index Pool prepares a shuffled list of indexes and returns $N = \text{batch\_size}$ of them in a sequence. Whenever the end is reached, the list is shuffled again.

Sample N.	Label
1	Red
2	Green
3	Blue
4	Red

One-Hot

Sample N.	Red	Green	Blue
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0

Figure 2.11: One-Hot Encoding Example. Each label has a dedicate channel where the cell corresponding to the sample is set to 1 if belonging to such class. 0 otherwise.

Chapter 3

Methodology

3.1 Semantic Segmentation Task

Several different convolutional neural network (CNN) architectures exist with respect to semantic segmentation. State of the art solutions are reported to score over 83% mean IoU (Intersection over Union) [63] on Cityscape and new solutions are researched continuously. We explore this subfield of Deep Learning by trying our own implementations and testing out methodologies taken from the literature.

A First Approach

In our first setup, we build a simple autoencoder network. An autoencoder consists in two macro modules, the first an encoder which learns a representation of the input data and possibly reduces its resolution at each step, the second a decoder which tries to reconstruct the image starting from the encoder output, i.e. from the data representation. In this first solution, the encoder is a VGG16-like network [64] which learns data representation and reduces its resolution to to 8x8. Note that the final fully connected layer has been removed from the original VGG16 architecture: by doing this we avoid the resolution from being further reduced, while at the same time we drastically decrease the amount of trainable parameters.

The decoder part consists in a series of blocks containing a first non-trainable up-sampling layer, followed by 3 or 2 convolutional layers which attempt to improve the up-sampling performed by the previous layer, without decreasing the feature maps resolution.

All the convolutional layers in the network use a 3x3 kernel and are followed by a batch normalization layer and a ReLU activation function; the only exception is the last convolutional layer, which is a 1x1 convolution with N-labels feature maps as an output which are then flattened and reshaped to obtain a 2D map with size width $\times$ height $\times N - labels$. This final layer is followed by a softmax activation function for prediction.

At this point, the network output consists of width × height prediction maps having size N-labels. Each map reports, for each pixel (which are now flattened on each channel), the probability of belonging to i^{th} class as predicted by the network.

Finally, we use pixel-wise cross-entropy as the loss function by comparing the network prediction and the one-hot encoding of the ground truths:

$$L(\theta) = -\frac{1}{N}\sum_{i=0}^{N}\sum_{c=0}^{M} y_{i,c}\ln(p_{i,c}).$$

Dilated Residual Network

According to [17], Dilated Residual Networks (DRN) [65] perform well on Cityscapes dataset, which is very similar to our work case. DRNs idea is based on preserving spatial resolution within a convolutional network. The reason behind this idea is that, despite providing a progressively increasing receptive field, image down-sampling causes the loss of spatial information which would instead be useful to achieve more accurate image classification and detailed scene understanding. Whenever the foreground object is not spatially dominant, such loss becomes extremely significant. In our segmentation task we need to segment varying size objects where road labelled pixels typically occupy most of the space in the image, whereas road markings, curbs or traffic lights are usually shaped in thin lines and occupy smaller areas. These considerations lead us to try out DRNs architectures, in particular we adopt its C-26 implementation.

DRN architecture code is provided under BSD-3 licence, however such code uses PyTorch framework therefore some adjustments and transposition are required before starting the actual experiments. Furthermore, the provided implementation of the DRN outputs feature maps with an incompatible size with respect to our ground truths. To overcome this issue, we design and append a decoder part to the original architecture so to perform the up-sampling of the encoder predicted output and match the initial input size. We design the decoding unit so that each up-sampling layer is followed by a pair of Conv3x3+BN+Relu block in charge of improving the reconstruction quality without decreasing its resolution. What's more, we implement skip connections between each block in the contracting part (decoder) and their correspondent in the expansion one (encoder) as described in [66]. The feature maps carried along these links are added to the up-scaled ones in order to help the original image reconstruction by retaining some of its features. Such addition is not always straight forward since residual units might have a different – typically smaller – resolution with respect to their up-sampled counterparts. The reason behind this phenomenon is that, when downsampling the input feature maps, convolutional layers tend to round up the resulting size to the closest integer. For instance, an image with size 225x225 would be downsampled to ceil(225/2) = ceil(112.5) = 113x113 when going through the encoder. When performing the up-sampling, the reconstructed resolution would thus be 226x226: this means that if we want to add the residual 225x225 feature map, we will face an inconsistency. Therefore, residual feature maps are adjusted by means of zero padding

on the top and on the right sides so to match the up-sampled features. After addition is performed, a cropping layer restores the original residual feature map resolution (Figure 3.1).

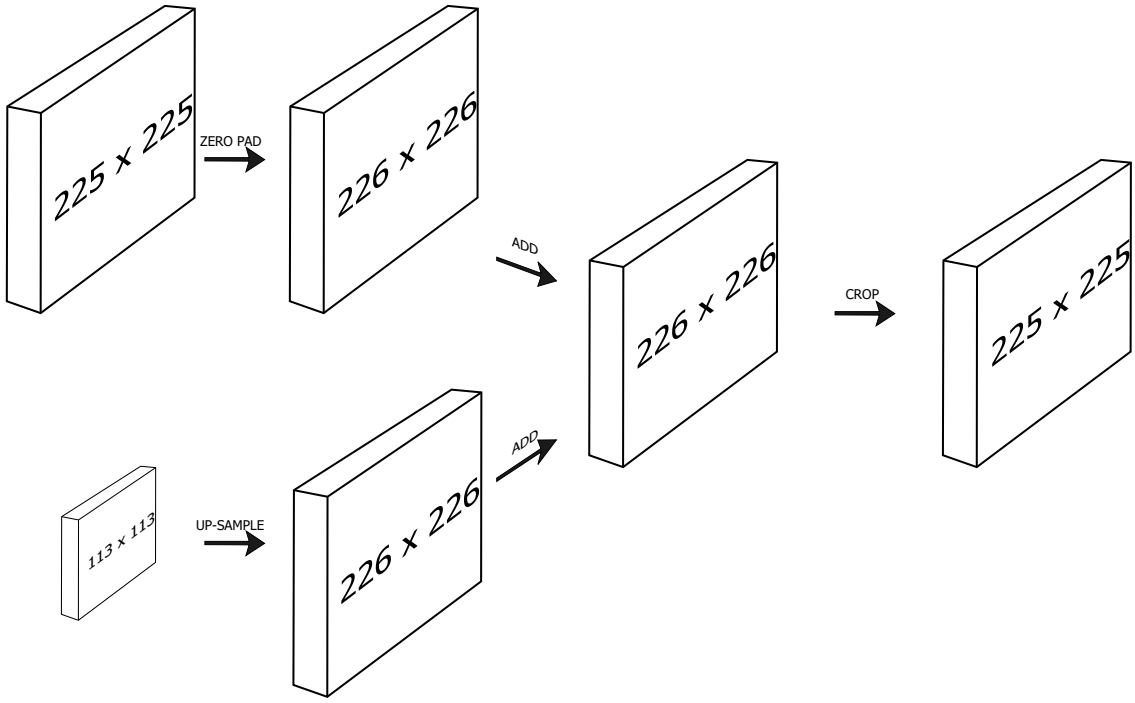


Figure 3.1: Residual Addition During Up-Sampling Phase. The residual (above) feature maps get zero padded before addition with the up-sampled (below) ones. The result is then cropped to the correct size.

Atrous Convolutions and Pyramidal Block

In order to explore other solutions and meet hardware constraints problems, we design a new, lightweight, segmentation network. In particular, our attention goes towards Atrous or dilated convolutions, which have recently been the subject of various researches and have been adopted in some of the newest architectures thanks to their properties. Among these, the aforementioned and tested DRN makes an extensive use of dilated convolutions inside of its down-sampling blocks as a mean to obtain a higher receptive field while preserving more spatial features with respect to the non-dilated counterpart. Other state-of-the-art networks like the one described in [67] make use of atrous convolutions both in parallel and cascade to handle multi-scale objects and to capture further contextual interactions, which have been proven to be beneficial when performing semantic segmentation on an image pixels [68, 69, 70].

In our setting, we design a basic block which exploits 3 parallel convolutions each with a different dilation rate (Figure 3.2). All of these convolutions use a 3x3 kernel to parse the

input feature maps and are followed by a Batch Normalization layer and a ReLU activation function. By using three different dilation rates, namely 1, 2 and 3, we want to be able to catch at the same time several contextual properties that each single pixel might be linked to. The resulting feature maps are then concatenated and passed forward to the next block in a cascade.

The idea is that at every pass a wider range of information (represented by the stack of concatenated layers from three differently dilated convolutions) is submitted to the next block. The block will then select, among these, the most relevant features, which might be given by context information (higher dilation) or by immediate interactions (lower dilation). We call this module Pyramidal Block because of the visual concept of these three differently dilated kernels and PyramidalNets the networks derived by the usage of this basic block.

To be concrete, we alternate blocks with stride 1, which preserve spatial acuity, and blocks with stride 2, which down-scale the feature maps by a factor of 2 on each dimension. At any down-scaling, we double the number or feature maps outputted by the convolutional layers in the block, in accordance to [56]. In order to avoid gridding artefacts experienced by Yu et al. [65], we adopt similar practices and append 2 convolutional layers at the end of the pyramidal blocks chain. As part of the artefact's avoidance, we do not use skip connections for these last layers as they might carry some of them along.

For what concerns the deconvolutional part of the architecture, we use a series of blocks which reconstruct the original resolution by doubling the size of each dimension at each step. A reconstruction block is composed of a first up-sampling layer followed by two convolutions. The up-sampling layer basically inserts a blank column and row between the input ones and applies an interpolation technique in order to fill the blanks in an appropriate way. In particular, we use a combination of nearest neighbour and bilinear interpolation. Convolutions within the decoding unit have the role of optimizing the up-sampling quality without reducing the feature map size. Each convolution is followed by BN and a ReLU activation function. Moreover, following the scheme described in Figure 3.1, we also add residual features at the end of each block. Finally, if `n_classes` is the number of target classes in the segmentation scope, class prediction is enabled by a last convolutional layer with `n_classes` feature maps as output and softmax activation function. We do not use any post-processing block such as DenseCRF to refine the segmentation quality.

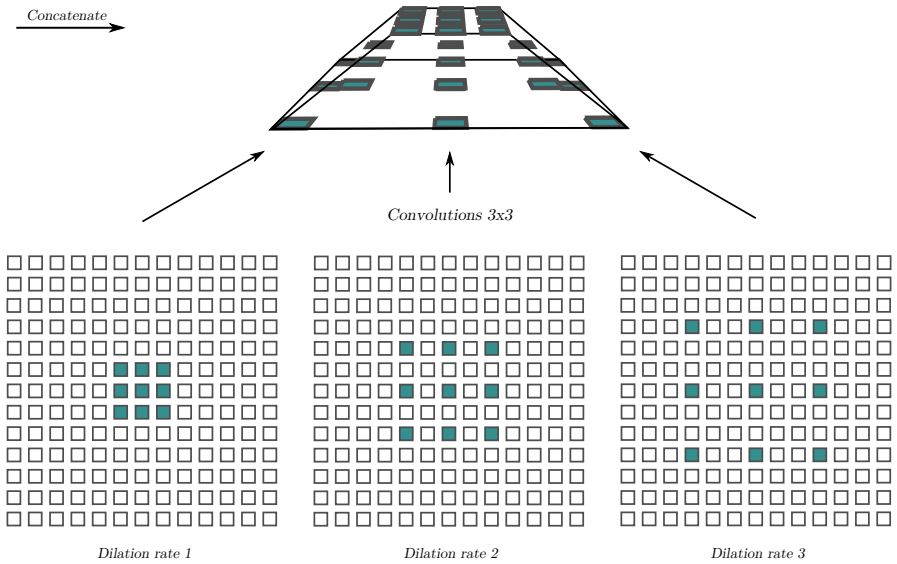


Figure 3.2: Pyramidal Block Concept. The same input feature map passes through three convolutions with differently dilated kernels at the same time. In each of the passes, the feature map reports a wider context information with respect to the same pixel, so that when they are concatenated the effect is that of a pyramidal point of view.

Experimental Results

Following company's needs and data availability, we run several experiments on different datasets in order to estimate our solutions' performances. Our first network is to be considered as a first approach to the field, aimed to better understand the requirements to build a functioning segmentation network, in particular the encoder-decoder pattern. The DRN and the PyramidalNet instead represent a more aimed effort towards a solution able to fulfill our requirements.

We train and evaluate all the CNNs on our firsthand-made dataset containing two classes. We use Stochastic Gradient Descent (SGD) optimization with LR = 0.001, momentum = 0.9 and decay = 0.0005 to update weights. We use our first dataset, which contains 24 training samples and 8 validation ones, to train the network on mini batches 5 images each for 50 epochs. To enhance the training, we use data augmentation techniques like horizontal flipping and rotation. The difference in the learnt representation among the different architectures is striking, especially between our first-attempt network and the other two (Figure 3.3). Although the first CNN seems to identify the main area in which the road is located, it fails to define a precise shape. As a matter of fact, the boundaries themselves are rough and characterized by a set of straight horizontal lines which overlap with the sorrounding vegetation. Such poor understanding of the scene is to be attributed to the network design, given the simplicity of the presented dataset. DRN and PyramidalNet, on the other hand, successfully draw a shape around the road area which, although not

smooth in its outline, also covers the farthest details in the image.

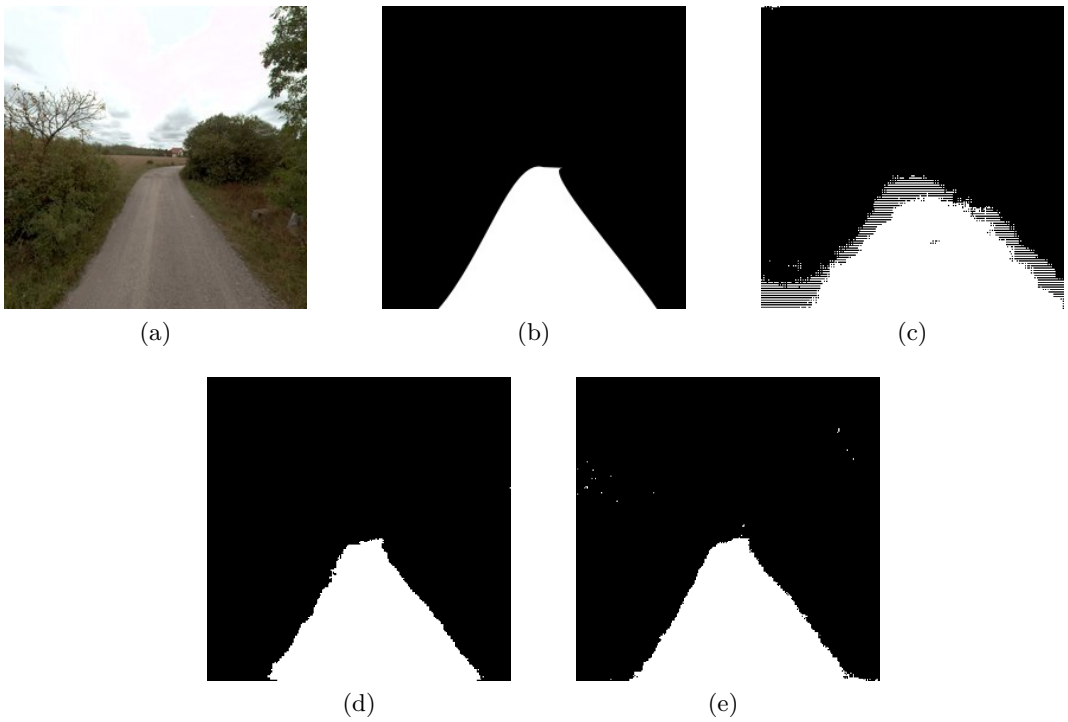
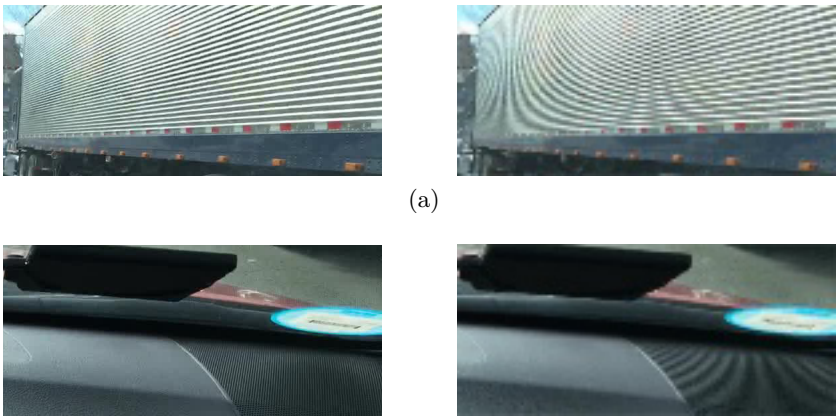


Figure 3.3: Segmentation Results our Firsthand Dataset. The original image (a), its ground truth (b) and segmentation results from the three different networks, respectively from the first CNN (c), from DRN (d) and from PyramidalNet (e).

Results obtained on the firsthand dataset, however, are not representative and serve as a first example of how our networks behave on the segmentation task only. In fact, such dataset is too small - both in terms of classes and samples - and too restricted in its depicted scenarios. For this reason, we test the CNNs on the previously mentioned BDD100k dataset (see Section 2.2.2). BDD100k provides a wider range of scenarios, which are present in a higher number of samples and with a higher number of classes, hence the need to perform some adjustments on the hyperparameters. In particular, we define a stepped Learning Rate (LR) schedule for the SGD optimizer: in this setup, the LR is initialized to a larger (0.1) value and decreased with step 10^{-1} up to reaching 10^{-5} as the epochs pass. In this way, we impose more aggressive updates in the earlier phases, whereas lighter updates guarantee slight refinements in the later ones. As for batch size, we experience tuning issues given by the fact that images coming from BDD100k dataset have a greater size in terms of memory with respect to our previous scenario. As a matter of fact, using a batch size greater than one with full size images would cause an out of memory error on our gpu, thus forcing us to use 1 as a batch size. To overcome this issue, we etablish a customized optimizer which allows gradient accumulation. We also try image resizing which results in

quality loss (Figure 3.4), however this allows us to use a bigger batch size therefore we test with this setting as well.



(a)

(b)

Figure 3.4: Image Resizing Example. On the left, some details extracted from the original image. On the right, the corresponding in the resized one: in these examples, line distortion produces fan-shaped artefacts.

Since the first of our datasets described in 2.2.1 contains a considerably lower amount of images with respect to BDD100k, we randomly select a subset of the latter and train DRN and PyramidalNet on this reduced version (which contains ~250 images splitted in train and validation). The results obtained with DRN lack in being precise (Figure 3.5c), however it still seems to identify the major objects and to classify them properly. Although missing thinner objects and still providing a slightly rough segmentation, PyramidalNet outperforms its counterpart and returns a nearly perfect segmentation (Figure 3.5d). It is to be noticed that in this case we use reduced-size images and batch size 2 instead of 1, however the achieved results are extremely promising.

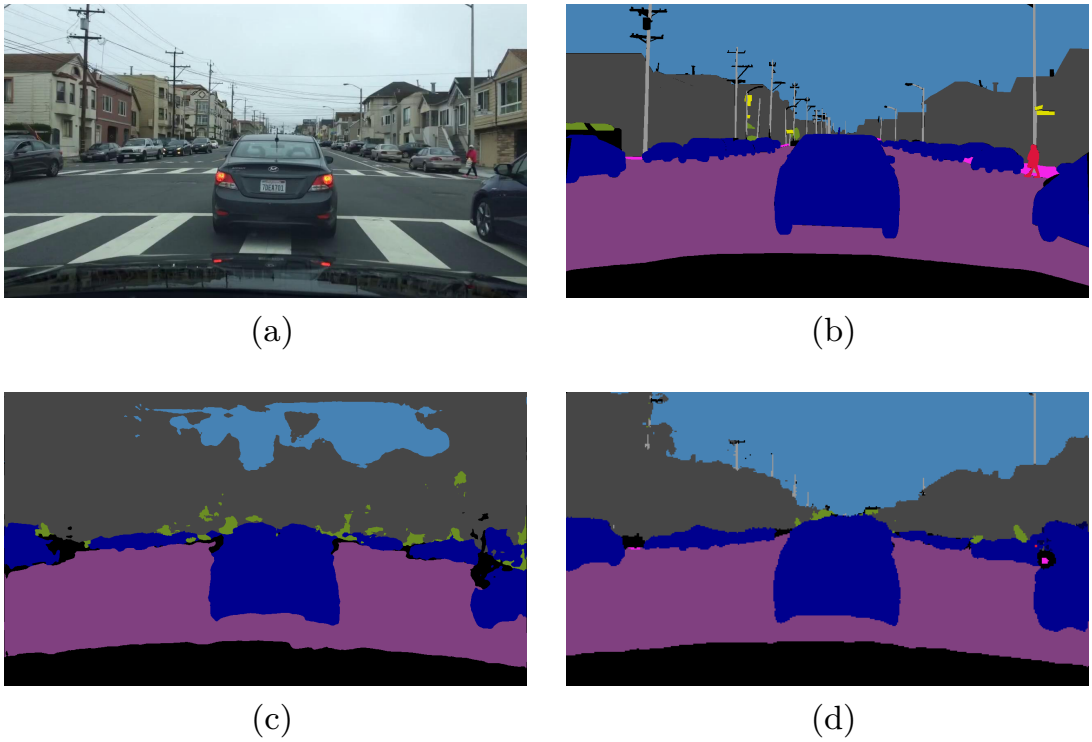
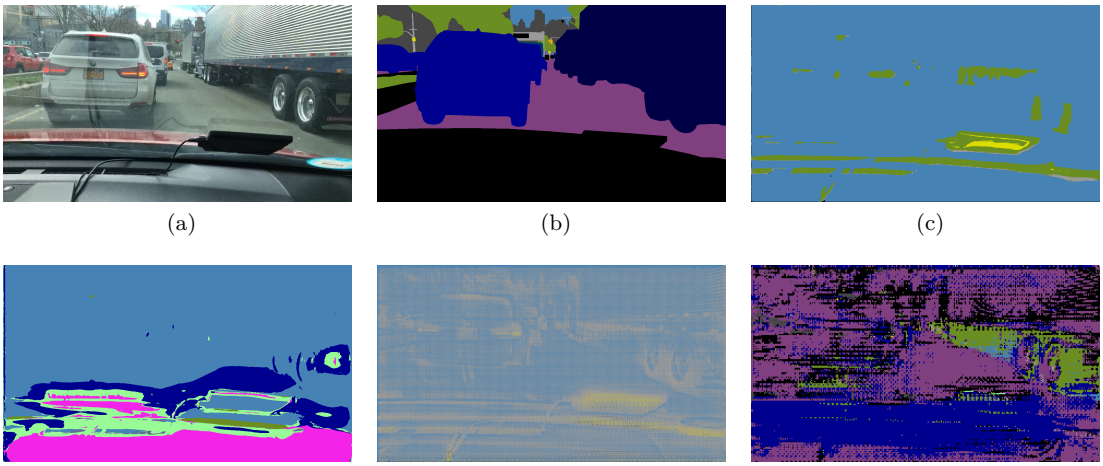


Figure 3.5: Reduced BDD100k Segmentation Output Example. The original image (a), its ground truth (b), prediction output from DRN (c) and from PyramidalNet (d) are reported.

The previous experiment contained just a small subset of the whole possible samples provided by BDD100k and was slightly biased towards certain types of scenarios. We therefore try to experiment training on th whole dataset, which contains 7000 images. In this case, segmentation obtained by both DRN and PyramidalNet are not satisfactory. When trained with batch size 1, DRN segments almost everything as sky (Figure 3.6c), which is one of the most recurring pixel labels. When the same is trained using a gradient accumulation factor equal to 32 it seems to be learning a wider range of possible labels, even though they are still applied in a rather confusing way and the most frequent ones are still being favorited over the others (Figure 3.6d). We try image resizing and a bigger batch size (8) to train PyramidalNet (Figure 3.6f). Also in this case however, segmentation quality is very poor and road class (violet color) - which is one of the most recurring ones - is favorited at the expenses of other less reccuring labels. We finally test PyramidalNet on full size images using an accumulation factor equal to 16 and train the network for 20 epochs (previous trainings were run for 100 epochs). Again, semantic segmentation does not provide any reliable result.



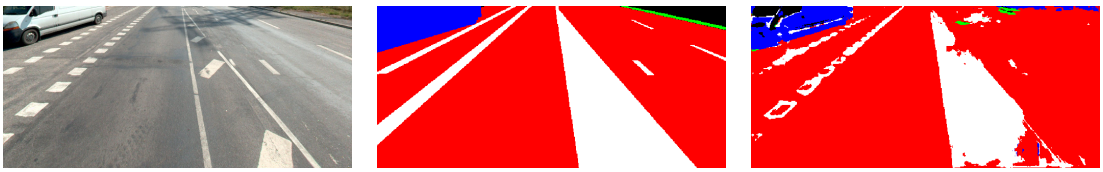
(a) (b) (c)

(d) (e) (f)

Figure 3.6: Segmentation results on BDD100k dataset for DRN and Pyramidal Network

Although results obtained on the the totality of BDD100k samples are not encouraging from a semantic segmentation quality point of view, those obtained from its reduced version seem to be a promise towards a good performance on easier datasets. It is also interesting to notice PyramidalNet tendency to faithfully segment elements' shapes (Figure 3.6e). Because of these considerations, we test PyramidalNets on the datasets mentioned in section 2.2.1, which might indeed be considered as simpler challenges with respect to BDD100k.

Firstly, we train the network using the road dataset, which contains labels such as road, road markings, cars, curbs and so on. As previously mentioned, this dataset is not considerably large and its upper part has been cut due to containing "not interesting" pieces of information. We set batch size 2 and half input resolution; usual image augmentation techniques are also used. On such set, our network performance reaches high values both in terms of precision (94%) and mean IoU (83%), however the predominance of road and road marking labels strongly affect the overal prediction and overwhelm other less recurring ones like curbs. In Figure 3.7 we provide an output example, here the original ground truth contained some noise too (imprecise labelling).

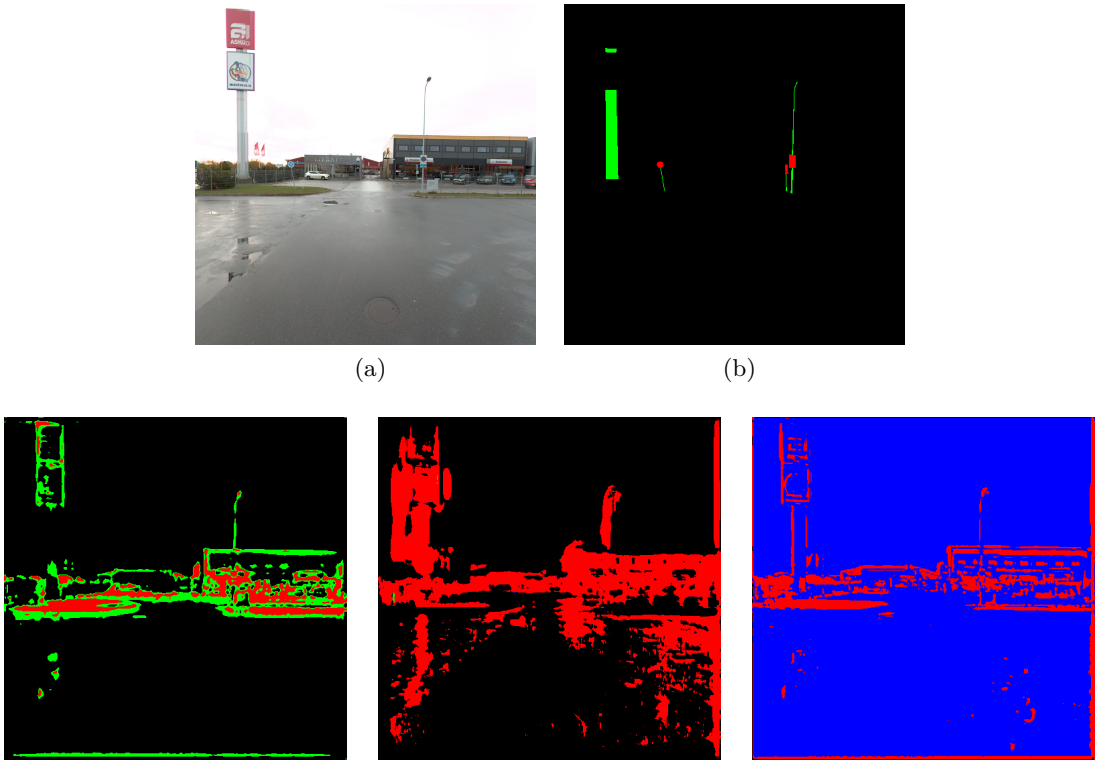


(a) (b) (c)

Figure 3.7: Segmention Example Results on Reach-U Road Dataset.

Secondly, we train our network on the second dataset - the pole dataset - mentioned in

section 2.2.1. In this dataset, poles, traffic signs and info signs are labelled as ground truths, while all the rest is considered background. Given the thin nature of such objects, this dataset is extremely imbalanced and background information predominance forces us to adopt some weight regularization. In particular, we implement a weighted version of the cross-entropy loss function and use it to perform several experiments. Performance is in this case poorer with respect to the road dataset, however the network still confirms its tendecy to isolate objects' shapes.



(a) (b) (c) (d) (e)

Figure 3.8: Segmentation Example Results on Reach-U Pole Dataset. Subfigure a and b are original and ground truth images respectively. The remaining ones represent some experimental results. In particular, figure c is the output of a network trained with slight loss weighting: although the prediction is incorrect, labels are all being used (instead of having a complete background predction). In figure d and e we experiment heavy weighting on labels "traffic sign" (red) and "info sign" (blue) given that they are the least recurring ones. Such a heavy weighting has the result of nullifying background information, which was supposedely several orders of magnitude more present.

We finally report some training values in numerical terms in Tables 3.1 and 3.2. We

evaluate our first experiments using a standard accuracy metric

$$\frac{TP}{TP + TN},$$

this however turns out to be imprecise when applied to segmentation tasks, especially when label distribution is imbalanced. For example, consider a black image with a single white pixel in its centre. If the segmentation output is a completely black image, accuracy measure would return 99.9% which is incorrect, given that the only informational point has been misclassified. Other types of measure such as precision and recall provide a more accurate estimate of the network performance and, for this reason, have been used in 3.2. Nontheless, for segmentation tasks Dice coefficient - or F1 score - and IoU (Intersection over Union) measure, both relying on the calculation of the area of intersection between the ground truth and the segmentation output, are more suitable. In our case we go for F1 score in our later experiments, however both methods are valid. In particular, we calculate the mean F1, which takes into account each label individually and then reports the average score: this provides even more accurate estimates since an overall version of the F1 would still be influenced by class imbalances.

Table 3.1: Training Results for Discusses Networks on BDD100k Datasets. Although accuracy reports fairly high values, actual segmentation results in poor quality. Mean F1 score (calculated on each label and averaged) reports this performance drop more accurately.

Network Configuration	Dataset	Epochs	Batch Size	Val. Accuracy (%)	Mean F1 score (%)
DRN	BDD100k Reduced	100	1	75.25	-
PyramidalNet	BDD100k Reduced	100	2	82.81	73.40
DRN	BDD100k	100	1	75.74	-
DRN, Cumulative Gradient (32)	BDD100k	100	1	75.25	-
PyramidalNet	BDD100k	100	8	79.14	38.40
PyramidalNet, Cumulative Gradient (16)	BDD100k	20	1	71.53	34.22

Table 3.2: Training Results for Discussed Networks on our Datasets. Once again, accuracy measure is unreliable: most of the pixels in pole dataset are background and the network mostly labels them in this way: this keeps the overall accuracy measure high. It is interesting to see how mean F1 score on road dataset is doubled even though accuracy is lower.

Network Configuration	Dataset	Loss Weights	Batch Size	Val. Accuracy (%)	Mean F1 score (%)
PyramidalNet	Road Dataset	None	4	93.96	83.02
PyramidalNet	Pole Dataset	Slight on all labels	2	99.53	46.25
PyramidalNet	Pole Dataset	Heavy on Traffic signs	2	99.48	41.09
PyramidalNet	Pole Dataset	Heavy of Traffic and Info signs	2	99.43	38.67

3.1.1 Conclusions

We set up a research base for semantic segmentation tasks. Our studies brought us to the testing and development of three different typologies of CNN architectures, all based on the autoencoder pattern but having a different design. Our first implementation consisted in a naïve approach to the problem and helped us moving the first steps towards this computer vision field. We secondly test Dilated Residual Networks, which employ dilated convolutions to improve output resolution without losing receptive field. In particular, we adopt DRN C-26 as described in [65] and customize it by appending a decoder composed of up-sampling and convolutional layers to improve reconstruction quality. Moreover, recent state-of-the-art implementations make extensive use of dilated convolutions in their architectures. Because of this consideration an of the accuracy levels obtained by DRN on our firsthand-made dataset and on a reduced version of BDD100k, we are induced to develop our own network employing dilated convolution as part of its building blocks, thus giving life to what we refer to as PyramidalNets. Due to out of memory issues experienced during previous testing with DRNs, our PyramidalNets are designed to not be the cause of too high loads on GPU memory.

Results obtained by the application of PyramidalNets to the different datasets reveal a tendency towards faithfully identifying shapes that outline object contours inside images. Even though the overall segmentation is inaccurate on more challenging tasks, this approach seems to have promising potential and will therefore be further investigated. A possible research direction might be that of increasing the network depth. In particular, deeper architectures employing Pyramidal blocks will be evaluated upon the arrival of new hardware capable of bearing them, whereas other training strategies employing adaptive learning rate optimizers will be subject of testing in the near future.

3.2 Image Classification Task

As previously mentioned, one of the objectives required by the project regards Image Classification. More specifically, we are requested to implement a method able to perform road type classification where the classes of interest are Paved road, Unpaved road, Gravel road, Cobblestone road and Mosaic Road. Furthermore, we develop a simple application providing a Graphical User Interface (GUI) to road type prediction.

3.2.1 Deep Learning Methods

State-of-the-art networks in image classification are able to obtain over 85% top-1 accuracy and over 97% top-5 accuracy on ImageNet dataset. These architectures, however, account for an extremely large number of classes (ImageNet counts 20000+ classes) of which they are required to learn the most important features in order to make a precise prediction. This often translates into extremely deep networks counting over 80 million parameters to train, which requires a big amount of GPU memory and computational time; some architectures involve several parallel networks performing independent predictions, which are then used to perform an election to decide which label is most likely to be true. In present case, our focus is limited to three labels and our final prediction embeds, we will see, an election-like system by itself. These considerations and further limitations imposed by the hardware, brought us to research simpler and lighter solutions in terms of memory usage, without, however, ignoring existing more complicated solutions.

Building a Convolutional Neural Network from scratch

As a first approach to the classification problem, we design a convolutional neural network (CNN) suitable for the task at hand. We proceed with a description of its main features and the analysis of its performance on the primary dataset. We then consider other datasets and perform further analysis.

Custom Network We first consider a small, simple convolutional neural network (CNN) composed by a series of convolutional layers, 8 in total. Each layer consists of a 3x3 kernel, is followed by a Batch Normalization (BN) layer [71] and a Leaky ReLU activation function. Downsampling is performed directly using the convolutional filters themselves by applying a stride of 2, the only exception is the first resolution reduction which is achieved by a max pooling with kernel 3x3 and stride 2. By the end of the convolutional stack, input resolution is reduced by a factor of 16, going from 128x128 to 8x8. This is followed by a pair of Fully Connected (FC) layers separated by a Dropout [72] layer. The first FC consists of 512 neurons and makes the transition from the last convolutional layer to the final predicting FC of 3 neurons smoother. All the layers in the network are initialized using the "standard" initialization technique described in [73]. As a result, we obtain a fairly shallow CNN with ~9.5M parameters which can easily fit in GPU memory.

We train all the layers of the network from scratch for 100 epochs on the whole training set with a batch size of 32 and backpropagate the cross-entropy loss calculated on the network's output. Learning rate is scheduled to be higher in the early steps of the training (0.1) and to slowly decrease throughout the training epochs, reducing to 0.0001. Input data is augmented by means of vertical/horizontal flips, scaled and standardized in a sample-wise manner.

In order to evaluate the performance of the network, we consider several accuracy measurements. As previously mentioned, our dataset is composed of several segments extracted from a set of original images, at most 5 per image. The segments are split into training and validation sets so that the segments that originate from the same image will be either in the former or in the latter set. Classification performance can be most easily evaluated on the basis of individual segments. More intuitive, however, would be an orthoframe-based prediction where a simple election system is arranged to find out the winning class among the segments collected from the given orthoframe (assuming that each orthoframe represents one road type only). Figure 3.9 depicts an example of the election scheme for orthoframe-based prediction. The performance measures for segment-based and orthoframe-based classification for the primary dataset are given in Tables 3.3 and 3.4, respectively.

Table 3.3: Segment-wide Confusion Matrix for Custom Network. Most of the mistaken predictions happen when Gravel Road is the true label.

Predicted / True	**Paved road**	**Unpaved road**	**Gravel road**	**TP**	**FP**	**TN**	**FN**	**Precision**	**Recall**
Paved road	2538	18	184	2538	202	4628	123	0.926	0.951
Unpaved road	10	2214	192	2214	202	4953	131	0.916	0.944
Gravel road	122	113	2109	2109	235	4780	376	0.899	0.849
Average								**0.914**	**0.914**

Table 3.4: Segment-wide Confusion Matrix for Custom Network. Paved road prediction quality decreases. However, better prediction for Gravel and Unpaved road help keeping up the overall accuracy.

Predicted / True	**Paved road**	**Unpaved road**	**Gravel road**	**TP**	**FP**	**TN**	**FN**	**Precision**	**Recall**
Paved road	512	5	29	512	34	932	22	0.938	0.959
Unpaved road	2	450	37	459	35	963	43	0.920	0.959
Gravel road	20	14	431	438	54	957	51	0.927	0.867
Average								**0.928**	**0.928**

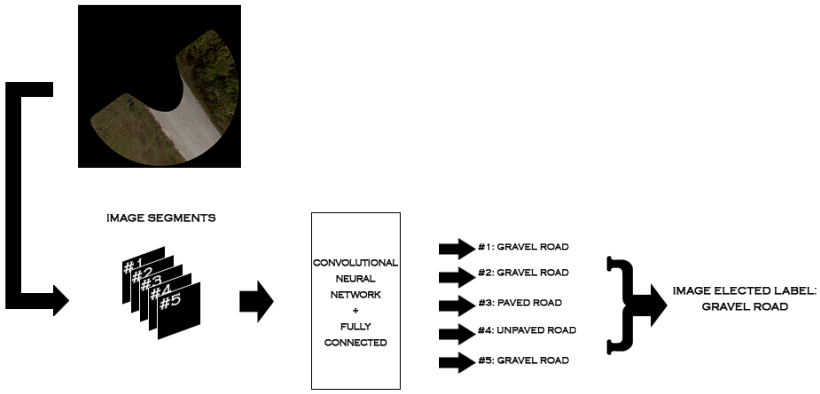


Figure 3.9: Election Process. Segments labels are predicted by the network. The image class is decided by the most frequent label for relative segments

The analysis of the confusion matrix relative to the segment-wise classification reveals good performances in classifying Paved road and Unpaved road data, which reach ~95% recall values. On the other hand, Gravel road seems to be a major cause of fail cases and its best recall performance settles at ~85%, 10 percentage points less with respect to the other cases. Consequently, the average recall resulting from these measurements drops to 91.4%. When it comes to orthoframe-based prediction, however, performances on all the labels receive a considerable improvement, especially on Gravel road data where precision climbs from 89.9% to 92.7% and recall from 84.9% to 86.7%, thus granting an average gain of 2.3%. On the average, we register a 1.4% improvement with respect to the segment-based prediction.

On the ReLU and Leaky ReLU Activation Functions In most of nowadays CNNs, the Rectified Linear Unit (ReLU) is the default recommended activation function to apply to the hidden units in the network. This very simple function, allows to avoid the output saturation by applying a simple max operation on it:

$$H_n = max(0, y_n).$$

Where y_n is the output of the nth layer. What's more, ReLU introduces a non-linearity inside of a network, thus allowing it to reach a higher approximation power. This, however, does not come free of any cost: using ReLU as an activation function may in fact cause other side problems, such as that of the "dying ReLU", which brings neurons inside of the network to never activate. To avoid this and to allow an easier flow of the gradient during the backpropagation phase, we experiment the use of the so-called Leaky ReLU [74]. This variation of the ReLU, instead of thresholding negative values to zero, basically multiplies

them by a small constant, which is typically set to 0.01, whenever encountering negative values instead of thresholding them to zero.

$$H_n^{'} = \begin{cases} y_n & \text{if } y_n \geq 0, \\ ay_n & \text{if } y_n < 0. \end{cases}$$

In this way, the first order derivative will not be zero for negative values, instead updates, even though small will still be made possible. Although the authors report negligible impacts on accuracy with respect to ReLU, our experiments show that Leaky ReLU activation functions averagely yield ~2% accuracy gains to final predictions, hence the adoption for our custom network.

Further Experiments on Datasets In section 2.1.2, we mentioned the creation of several different dataset apart from the original one, namely the Sharp, Blurry and Mixed datasets. Differently from the original dataset, these three ones contain segments which were extracted in a guided approach, by selecting respectively segments belonging to the central zone only, to the border zone only and anywhere in between for each image. In order to analyse whether the segments position within the image, thus the fact of being blurrier or sharper, has an impact on the prediction accuracy, we cross validate our network by training it on each of the aforementioned datasets, each time evaluating its performance on all of the validation sets of such datasets. In other words, we first train our network using the Blurry train set and validate it by evaluating its performance all the Blurry, Sharp and Mixed validation sets, then we do the same for the Sharp and Mixed train sets. As done for previous network evaluation, we collect data in confusion matrices and obtain Precision and Recall values, which we present in a condensed form in Tables 3.5 and 3.6.

Table 3.5: Precision and Recall values for segment-wide evaluation. Prediction improves when done on alike set of samples.

Validate On / Train On	**Blurry**		**Mixed**		**Sharp**	
	Precision	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*
Blurry	**0.903**	**0.903**	0.895	0.895	0.857	0.858
Mixed	0.893	0.892	**0.903**	**0.903**	0.889	0.889
Sharp	0.872	0.872	0.892	0.892	**0.903**	**0.903**

Table 3.6: Precision and Recall values for image-wide evaluation. Alike set of samples still obtain more accurate predictions, however Mixed-trained networks seem more resilient to changes.

Validate On / Train On	**Blurry**		**Mixed**		**Sharp**	
	Precision	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*
Blurry	**0.918**	**0.917**	0.911	0.911	0.884	0.883
Mixed	0.922	0.922	**0.928**	**0.927**	0.910	0.910
Sharp	0.884	0.884	0.911	0.910	**0.917**	**0.917**

As expected, networks trained on either Blurry or Sharp datasets seem to perform better when asked to predict labels for validation data belonging to the same type, slightly worse when working on Mixed data and even worse when working on opposite-type data. A Sharp-trained network will lose up to 5% accuracy when attempting to classify segments originally belonging to the Blurry validation set. On the other hand, networks trained on the Mixed dataset have higher mean accuracy levels across all of the validation sets they are tested on: a likely explanation for this behaviour might be that, given the mixed nature of the training set, the networks learn a better and more generalized representation of the data, therefore resulting into more stable and reliable predictors.

Adapting an Existing Solution

Deep convolutional neural networks [52] have brought great advances in the field of image classification. Each convolutional layer in a deep network learns progressively higher-level features by being staked onto previous ones. Moreover, previous studies demonstrate that network depth has a crucial role in the accuracy of the classification task [64]. Our first approach to the classification problem was to build a relatively shallow CNN with 8 layers of depth only, while state-of-the-art solutions often count over 50 layers. Simply adding layers to a convolutional network, however, does not guarantee an improvement in accuracy and side problems such as that of the vanishing/exploding gradients [73, 75] or the training accuracy degradation arise. While the vanishing gradient issue has largely been addressed and reduced by means of normalization systems, such as batch normalization, the training accuracy degradation has been faced in other ways. In particular, He et al. [56] have achieved to build very deep convolutional networks by including skip connections between the basic building blocks in the network architecture. Such connections allow to avoid degradation problems by introducing a direct link through which information propagates more easily in spite of network's depth [76].

Deep Residual Network Given the success that so-called Residual Networks (ResNets) have had on the classification task, we experiment with a 50 layer implementation (ResNet50) as described in [56]. Other implementations, such as those with 101 and 152 depth layers,

do not fit in GPU memory and cannot therefore be used. To start with, we initialize the network's weights to Imagenet, then we remove the final dense layer for two distinct reasons. Firstly, our input shape is 128x128, while in the original implementation it is 224x224. This means that the output from the last convolutional layer would have different resolution and would thus not be compatible with the FC layer, whose size had been statically computed basing on the feature map resolution resulting from the previous input size. Secondly, in the original implementation there was the need to classify 1000 different labels, whereas in our case we only need 3. We then proceed by adding a first FC layer with 512 neurons and ReLU activation function, followed by a Dropout layer with 50% probability and a second FC with 3 neurons. This last layer is provided with a SoftMax activation function to enable prediction and loss evaluation by cross entropy function. As a result, the network counts ~40M parameters in total, which is a considerable difference with respect to the previously employed network, however it is still low enough to not cause out of memory issues. We train all the layers of the network for 100 epochs on the whole training set using the same training configuration and data pre-processing as before. The CNN performance measures are collected and presented in Tables 3.7 and 3.8.

Table 3.7: Segment-wide Confusion Matrix. Along with the prediction frequencies, precision and recall values are reported. The network seems to have more difficulties when dealing with Gravel road class.

Predicted / True	**Paved road**	**Unpaved road**	**Gravel road**	**TP**	**FP**	**TN**	**FN**	**Precision**	**Recall**
Paved road	2599	9	134	2599	143	4687	71	0.947	0.731
Unpaved road	8	2261	157	2261	165	4990	84	0.931	0.964
Gravel road	63	75	2194	2194	138	4877	291	0.940	0.882
Average								**0.940**	**0.940**

Table 3.8: Image-wide Confusion Matrix. Along with the prediction frequencies, precision and recall values are reported. Overall performance increases.

Predicted / True	**Paved road**	**Unpaved road**	**Gravel road**	**TP**	**FP**	**TN**	**FN**	**Precision**	**Recall**
Paved road	525	1	18	525	19	947	9	0.965	0.983
Unpaved road	2	459	25	459	27	1004	10	0.944	0.978
Gravel road	7	9	454	454	16	987	43	0.965	0.913
Average								**0.958**	**0.958**

Although the number of fails case related to Gravel road is still higher when compared to Paved road and Unpaved road, the overall statistics receive a striking improvement with respect to the custom network and gain 2.6% points on the segment-based classification. When evaluating orthoframe-based performance, the overall precision and recall measures improve from 94% to 95.8%, therefore gaining 3% points compared to the custom network solution.

3.2.2 Conclusions

We applied and analysed two types of networks. The custom CNN reaches 90% validation accuracy levels in first 12 epochs, then starts a refinement phase in which learning progress has a much slower pace, reaching a final 91.4% accuracy after 100 epochs. ResNet50 saturates much faster - high accuracy values (~93.8%) only marginally different from the eventual 94% are also obtained in the first 12 epochs. Figure 3.10 provides a visualization of the learning curves.

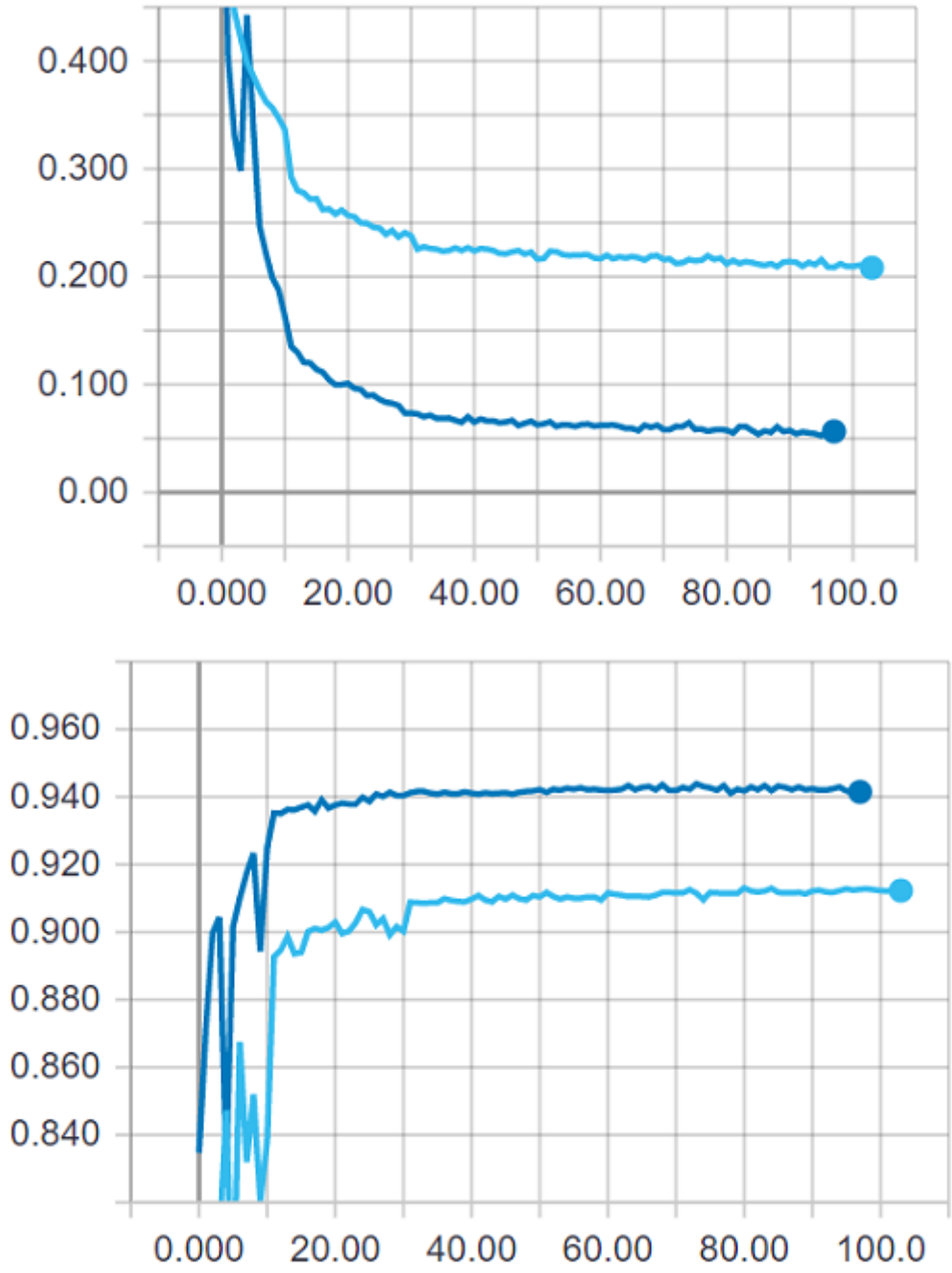


Figure 3.10: ResNet (blue) and our custom CNN (light-blue) compared. The graphs, show the training loss over epochs (left) and the validation accuracy(right). ResNet reaches higher accuracy values within fewer epochs.

The final step for the classification task consists in building a software implementing an easy interface towards road surface type prediction. We achieve this by using PyQt library with

Qt Designer, which allow us to quickly design a simple cross-platform GUI application. In particular, we subdivide its functionalities into two separate tabs: the first one allows single image predictions and its main role is to provide an illustration of how such prediction is taking place and how the network is predicting single segments from the image, the second one allows predictions on a batch of images organized in a structured folder and provides results in a comprehensive csv file for later analysis. Figure 3.12 depicts an example of a single-image tab, where the image, its mask and the CNN model must be selected before proceeding to the classification. It is also possible to adjust the number of segments that are used in the election process, although the value is restricted to odd ones, to avoid draws. An in-app console output keeps track of performed actions and finally displays the colors assigned to the classes, which are used to paint the segment boxes on the image itself. An example of multiple-image tab is provided in Figure 3.11. Here, images are classified in a batch and an image-by-image feedback is out of the scope. To sum it up, we have developed a simple GUI application which allows its users to perform predictions on single images or batches of them. Such predictions are made possible by a trained network, provided as hdf5 file within the application package. Studies over the networks were carried out by comparing the performances on the same datasets, with the same training parameters, of two CNN architectures. These studies revealed better classification accuracy are provided by ResNet50, which outperforms the shallow network by 2.6% percentage on a segment-based classification and by 3% on an orthoframe-based one, thus confirming the higher efficacy of deeper networks. Moreover, separate studies were aimed at understanding the impact of the image quality of the extracted segments for the overall performance. We found that networks trained on segments either closer to the border or closer to the centre of the image lead to an averagely poorer classification quality when applied on data which is different from the training one in terms of extraction position (image quality is in direct correlation with the segment position). In particular, the networks trained on Sharp dataset perform worse than others. On the other hand, a mixed location set of segments leads to better results over any type of validation data, which is probably due to the fact that a wider variance allows the network to learn a better generalization of data. Finally, while the custom solution provides relatively high (92.8%) accuracy results, its shallower nature imposes a limit on its performance. ResNet50, instead, embodies a greater learning power, which leads to better classification accuracy. The results, however, might still be improved and further experiments are to be carried out by using a wider range of augmentation techniques and a bigger dataset. In addition, as image-based prediction is more accurate than segment-based prediction, further generalization might be useful by considering a series of images instead of single images.

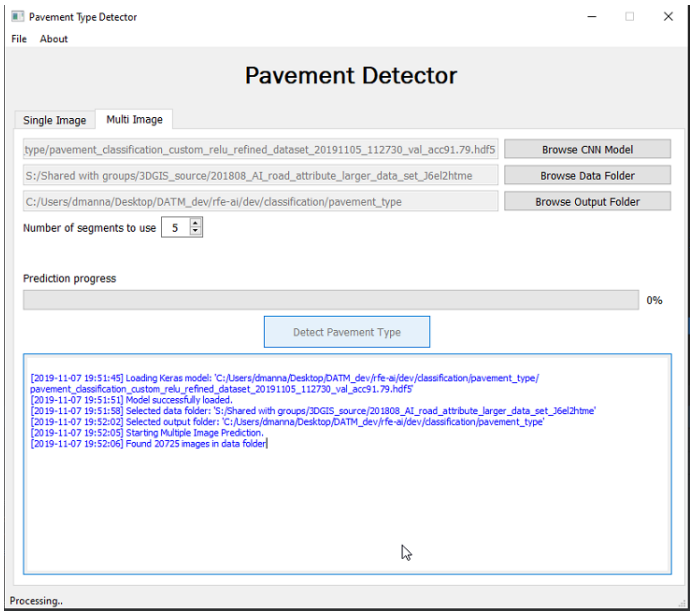


Figure 3.11: Pavement Detector Application - Multiple Image Mode The input data folder is scanned to have a specific structure, which matches the ones provided by Reach-U

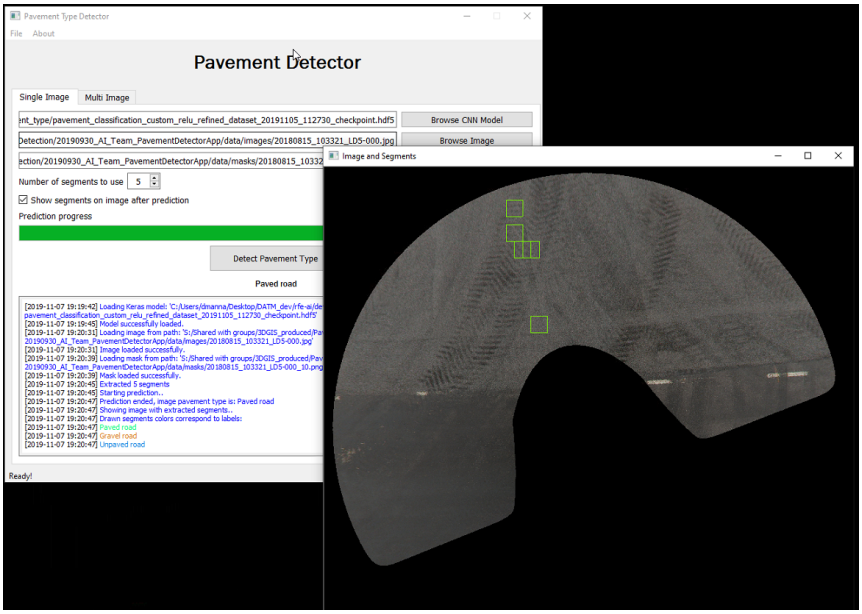


Figure 3.12: Pavement Detector Application - Single Image Mode The segments used for classification are reported on the image. The color indicates the predicted label for each segment

Chapter 4

Conclusions

In this work we focussed on the resolution of two main objectives. Firstly, semantic segmentation research was set up. Datasets for this task needed to be built in the first place, hence Berkeley Deep Drive dataset was used a reference and some guidelines were provided to hired persons in order to enable them to produce suitable ground truths. By the end of this process two datasets wer finally produced. The first road-related dataset contained 230 samples of cropped images were labels such as road, road markings and cars were highlighted in different colors. Given some interest changes, the second dataset contained over 3000 samples, however the labelled objects differed from the previous ones. Here, poles, traffic signs and information signs were labelled inside the ground truths. The nature of such objects lead to the cration of a very unbalanced dataset given that most of the pixels in the image represented background information.

Three distinct convolutional neural networks (CNNs) were mainly researched. The first one consisting in a simple implementation of an autoencoder architecture, the second one in a Dilated Residual Network implementation and the third one in a novel architecture combining several aspects in what we called a Pyramidal Network (PyramidalNet). PyramidalNet, in particular, was characterized by a novel building block — the pyramidal block — in which dilated convolutions are used in parallel and then concatenated. What's more, skip connections internally to the building blocks and among units in the encoder and the decoder parts are used. These networks have been studied by performing experiments on the aforementioned datasets and finally brought to the choice of the PyramidalNets as a subject for further studies.

Secondly, we addressed an image classification task in which different road types needed to be told apart. Since the outcome of this task will be used for other applications such as 2D to 3D transposition and since the original data source was very unbalanced, we built an image segments extraction system. This system allows the extraction of suitable segments containing road only image parts, thus allowing to build a balanced dataset and to train the networks on more class-specific features. Next, we researched two different CNNs following the hardware constraints which held us back from testing further deeper

architectures. Both the researched CNNs achieved remarkable results, however, at last, ResNet50 — a 50 layers-depth implementation of the Residual Networks — showed better performance by striking 95.8% both in recall and precision measures.

As a last step, a GUI application integrating our best CNN for a straight-forward use of the classification system. The application allowed to perform batch predictions by selecting image directories and to perform single-image predictions for investigation purposes: the used segments for image-wide prediction are displayed on the image itself.

The provided solutions were finally able to satisfy the end goals of this work, which are aimed to allow future research and development of deep learning systems to transpose 2D data to a 3D environment.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, available: http://www.deeplearningbook.org.

[2] G. Developers, "Reducing loss: Learning rate," 2019. [Online]. Available: https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate

[3] T. Spinner, J. Körner, J. Görtler, O. Deussen, "Towards an interpretable latent space," 2019. [Online]. Available: https://thilospinner.com/towards-an-interpretable-latent-space/

[4] Imgaug, "A simple and common augmentation sequence," 2019. [Online]. Available: https://imgaug.readthedocs.io/en/latest/source/examples_basics.html

[5] MathWorks, "Getting started with semantic segmentation using deep learning," 2019. [Online]. Available: https://uk.mathworks.com/help/vision/ug/getting-started-with-semantic-segmentation-using-deep-learning.html

[6] Advanced Navigation, "Spatial Dual," 2019. [Online]. Available: http://www.advancednavigation.com/product/spatial-dual

[7] VelodyneLidar, "Puck," 2019. [Online]. Available: https://velodynelidar.com/vlp-16.html

[8] Flir, "Ladybug5plus," 2019. [Online]. Available: https://www.flir.com/products/ladybug5plus/

[9] C. Babbage, *On the Application of Machinery to the Computation of Astronomical and Mathematical Tables*. Taylor, 1824.

[10] Ada Augusta Lovelace, *Scientific Memoirs*, 1983, ch. Sketch of the Analytical Engine invented by Charles Babbage... with notes by the translator, pp. 666–731.

[11] A. M. Turing, "I.—Computing Machinery And Intelligence," *Mind*, vol. LIX, no. 236, pp. 433–460, 1950.

[12] M. N. Wernick, Y. Yang, J. G. Brankov, G. Yourganov, and S. C. Strother, "Machine learning in medical imaging," *IEEE Signal Processing Magazine*, vol. 27, no. 4, pp. 25–38, 2010.

[13] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Lecture Notes in Computer Science*. Springer International Publishing, 2015, pp. 234–241.

[14] B. J. Erickson, P. Korfiatis, Z. Akkus, and T. L. Kline, "Machine learning for medical imaging," *RadioGraphics*, vol. 37, no. 2, pp. 505–515, 2017.

[15] M. L. Giger, "Machine learning in medical imaging," *Journal of the American College of Radiology*, vol. 15, no. 3, pp. 512–520, 2018.

[16] M. Kuderer, S. Gulati, and W. Burgard, "Learning driving styles for autonomous vehicles from demonstration," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2641–2646.

[17] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, "BDD100K: A diverse driving video database with scalable annotation tooling," arXiv:1805.04687.

[18] J. Kober and J. Peters, "Learning motor primitives for robotics," in *IEEE International Conference on Robotics and Automation*, 2009, pp. 2112–2118.

[19] A. Giusti, J. Guzzi, D. C. Cireşan, F. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016.

[20] D. B. Lenat and R. V. Guha, *Building Large Knowledge-Based Systems; Representation and Inference in the Cyc Project*, 1st ed. Addison-Wesley Longman Publishing Co. Inc., 1989.

[21] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[22] W. S. Mcculloch, "The brain computing machine," *Electrical Engineering*, vol. 68, no. 6, pp. 492–497, 1949.

[23] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 1952.

[24] A. Hodgkin and A. Huxley, "The dual effect of membrane potential on sodium conductance in giant squid axon of lolicon," *Journal of Physiology*, vol. 116, pp. 497–506, 1952.

[25] W. B. Connors, M.F. Bear, M.A. Paradiso, *Neuroscience: Exploring the Brain.* Lippincott Williams & Wilkins, 2007.

[26] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *AISTATS*, 2011.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[28] D. O. Hebb, *The Organization of Behavior.* Taylor & Francis Inc, 2002.

[29] A. Singer, "Implementations of artificial neural networks on the connection machine," *Parallel Computing*, vol. 14, no. 3, pp. 305–315, 1990.

[30] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity," *Parallel Computing*, vol. 14, no. 3, pp. 347–361, 1990.

[31] D. Michie, D. J. Spiegelhalter, C. Taylor *et al.*, "Machine learning," *Neural and Statistical Classification*, vol. 13, 1994.

[32] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[33] H. D. Block, "The perceptron: A model for brain functioning," *Rev. Mod. Phys.*, vol. 34, pp. 123–135, 1962.

[34] A. G. Parlos, K. T. Chong, and A. F. Atiya, "Application of the recurrent multilayer perceptron in modeling complex process dynamics," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 255–266, 1994.

[35] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[36] D. F. Specht, "Probabilistic neural networks and the polynomial adaline as complementary techniques for classification," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 111–121, 1990.

[37] B. Widrow and M. E. Hoff, "Adaptive switching circuits," Stanford Univ Ca Stanford Electronics Labs, Tech. Rep., 1960.

[38] J. L. Mundy, A. Zisserman *et al.*, *Geometric invariance in computer vision*. MIT press Cambridge, MA, 1992, vol. 92.

[39] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[41] A. CAUCHY, "Methode generale pour la resolution des systemes d'equations simultanees," *C.R. Acad. Sci. Paris*, vol. 25, pp. 536–538, 1847.

[42] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[43] M. D. Zeiler, "Adadelta: An adaptive learning rate method," 2012, arXiv:1212.5701.

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017, arXiv:1412.6980.

[45] V. N. Vapnik, *The Nature of Statistical Learning Theory*. Berlin, Heidelberg: Springer-Verlag, 1995.

[46] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278 – 2324, 1998.

[47] D. H. Ballard, "Modular learning in neural networks," in *Proceedings of the Sixth National Conference on Artificial Intelligence*, ser. AAAI'87, vol. 1. AAAI Press, 1987, pp. 279–284.

[48] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[49] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[50] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[51] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[52] A. Krizhevsky, "Learning multiple layers of features from tiny images," in *Technical Report*, vol. 1, no. 4, Citeseer. University of Toronto, 2009.

[53] A. Adate, R. Saxena, and Don.S, "Understanding how adversarial noise affects single image classification," in *Communications in Computer and Information Science*. Springer Singapore, 2017, pp. 287–295.

[54] F. Marra, D. Gragnaniello, and L. Verdoliva, "On the vulnerability of deep learning to adversarial attacks for camera model identification," *Signal Processing: Image Communication*, vol. 65, pp. 240–248, 2018.

[55] Z. You, J. Ye, K. Li, Z. Xu, and P. Wang, "Adversarial noise layer: Regularize neural network by adding noise," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019.

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, arXiv:1512.03385.

[57] L. Sun, "Resnet on tiny imagenet," Technical Report, Stanford University, 2017.

[58] A. Riid, R. Lõuk, R. Pihlak, A. Tepljakov, and K. Vassiljeva, "Pavement distress detection with deep learning using the orthoframes acquired by a mobile mapping system," *Applied Sciences*, vol. 9, no. 22, p. 4829, 2019.

[59] OpenCV, "Computer Vision Annotation Tool (CVAT)." [Online]. Available: https://github.com/opencv/cvat

[60] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009.

[61] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.

[62] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[63] Papers With Code, "Semantic Segmentation on Cityscapes test." [Online]. Available: https://paperswithcode.com/sota/semantic-segmentation-on-cityscapes

[64] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015, arXiv:1409.1556.

[65] F. Yu, V. Koltun, and T. Funkhouser, "Dilated residual networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[66] Z. Zhang, Q. Liu, and Y. Wang, "Road extraction by deep residual U-Net," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 749–753, 2018.

[67] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017, arXiv:1706.05587.

[68] X. He, R. Zemel, and M. Carreira-Perpinan, "Multiscale conditional random fields for image labeling," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR).*, 2004.

[69] D. Hoiem, A. A. Efros, and M. Hebert, "Putting objects in perspective," *International Journal of Computer Vision*, vol. 80, no. 1, pp. 3–15, 2008.

[70] L. Ladicky, C. Russell, P. Kohli, and P. H. Torr, "Associative hierarchical CRFs for object class image segmentation," in *IEEE 12th International Conference on Computer Vision*. IEEE, 2009.

[71] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, arXiv:1502.03167.

[72] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[73] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010.

[74] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[75] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[76] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," 2016, arXiv:1603.05027.