SAPIENZA
UNIVERSITÀ DI ROMA

Deep Learning based approach for Semantic Segmentation of Aerial Images

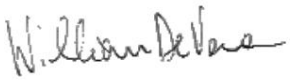
Faculty of Information Engineering, Computer Science and Statistics

Degree Course in Computer Science

Candidate

William De Vena

Serial number 1862820



Speaker

Prof. Danilo Avola

Co-rapporteurs

Prof. Luigi Cinque Dott.

Alessio Fagioli

Academic Year 2021-2022

Deep Learning Approach for Semantic Segmentation of Aerial Images. Thesis. Sapienza - University of Rome

This thesis was composed with LATEX and the Sapthesis class.

Author's email: william98wdv@gmail.com

Index

1. Introduction 4
1.1 Context of application. 4
1.2 State of the art. 6
1.3 Contribution and Outline. 11

2 Image segmentation 2.1 Image 12
segmentation in humans. 15
2.2 Classical approaches. 16
2.2.1 Threshold methods. 16
2.2.2 Methods of divisions and funds. 17
2.2.3 Methods based on graphs. 17
2.2.4 Method of active contours. 19
2.2.5 Clustering methods. 22
2.2.6 Supervised Algorithms. 27
2.3 Deep Learning Approaches. 27
2.3.1 Context-based architectures. 27
2.3.2 Feature-enhacement-based architectures. 28
2.3.3 Architectures based on deconvolution. 28
2.3.4 GAN-based architectures. 29

3 Deep Learning 31
3.1 Perceptron. 32
3.2 Multilayer Perceptron. 33
3.3 Activation functions. 34
3.3.1 Sigmoid function. 35
3.3.2 Hyperbolic tangent. 35
3.3.3 Rectified Linear Activation Function. 35
3.3.4 Softmax. 36
3.4 Learning. 37
3.4.1 Learning paradigms. 38
3.4.2 Loss functions. 38
3.4.3 Back propagation of the error. 39
3.5 Regularization. 40
3.5.1 Data augmentation. 42
3.6 Convolutional Neural Networks. 43
3.6.1 Advanced convolutional neural networks. 46

4 Architecture and work methods **52**
4.1 Difficulties addressed in the dataset. 52
4.2 Cleaning the dataset and Data Augmentation. 54
4.3 DeepLabV3. 57
4.3.1 Total architecture. 59
4.3.2 Dilated Convolution. 60
4.3.3 Residual Neural Networks. 61
4.3.4 Atrous Spatial Pyramid Pooling. 66

5 Experiments and Results **68**
5.1 Computational resources. 68
5.2 Technologies used. 69
5.3 FloodNet dataset. 69
5.4 Experiments. 70
5.5 Comparison with other works. 74

6 Conclusions **77**
6.1 Future developments. 78

Chapter 1

Introduction

1.1 Context of application

In the twenty years from 1995 to 2015, about 90% of the disasters that hit the world population were caused by meteorological phenomena such as floods, storms, heat waves and others. During this period, the EM-DAT (Emergency Events Database), one of the most important databases of disastrous events in the world, recorded 6457 disasters caused by meteorological events, which caused 606,000 victims (an average of 30,000 per year) and all '' approximately 4.1 billion total people affected (between 1995 and 2015) including injured, homeless people and people in need of assistance. Of these events, floods are among the most frequent and serious, in fact they represent 43% of the total events (Figure 1.1) and caused approximately 157,000 deaths plus 2.3 billion people affected (between 1995 and 2015) (Figure 1.2). Floods are among the most serious types also from the point of view of economic damage, in fact, in the United States alone there were 662 billion dollars of damage caused by floods (1995-2015) and in the European continent approximately 262 billion (1994-2015). Furthermore, the occurrences of disastrous events caused by meteorological phenomena are increasing, in particular, in the period between 2005 and 2014 there was an average of 335 events per year, or 14% more than in the period between 1995-2004 and almost double those recorded between 1985 and 1994. The same trend can be seen in the floods, which in the same period (2005-2014) reached a peak of 171 events per year, compared to 127 previous decades [1].

In this context, the management of these disastrous events is essential. In particular, the three phases of flood management are fundamental: the phase preceding the event, or the prevention phase; the phase during the event itself, in which it is essential to have knowledge of the affected areas and the spread of the flood, information that is often not easy to find; and finally the post-flood phase, in which the damage is assessed. In recent years, in the field of natural disaster management, the use of UAVs (Unmanned Aerial Vehicles), also called drones, has proved very useful. One of the first uses was in 2005 in the United States where, for the first time, drones were used by the CRASAR (Center for Robot-Assisted Search and Rescue) of the A&M University of Texas to search and rescue survivors of the events caused. from Hurricane Katrina. Subsequently, in 2015, CRASAR collaborated with Measure UAV consulting firm and American Red

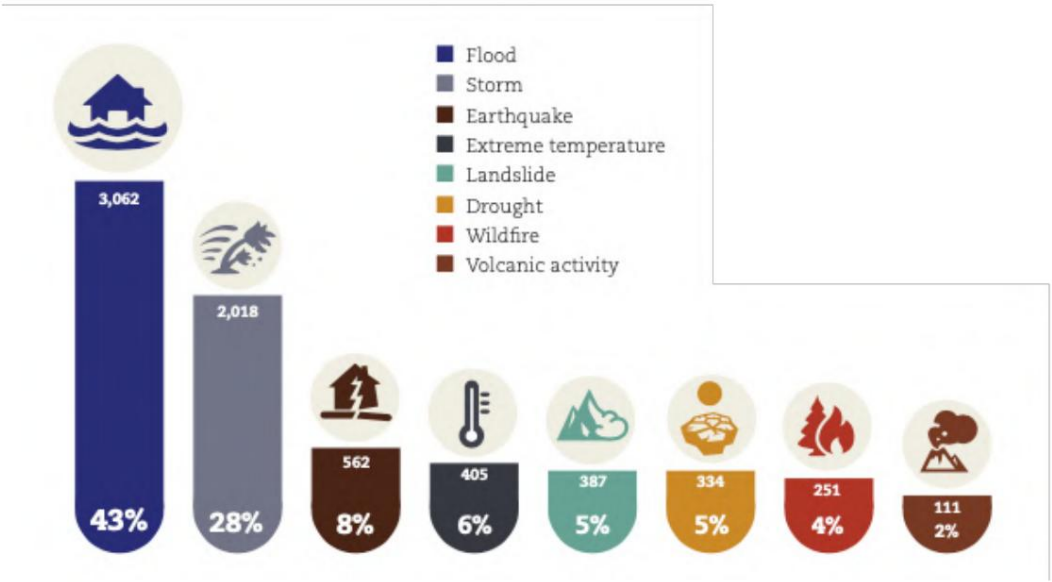


Figure 1.1. Percentage of natural disaster events by type of disaster (1995-2015) [1].

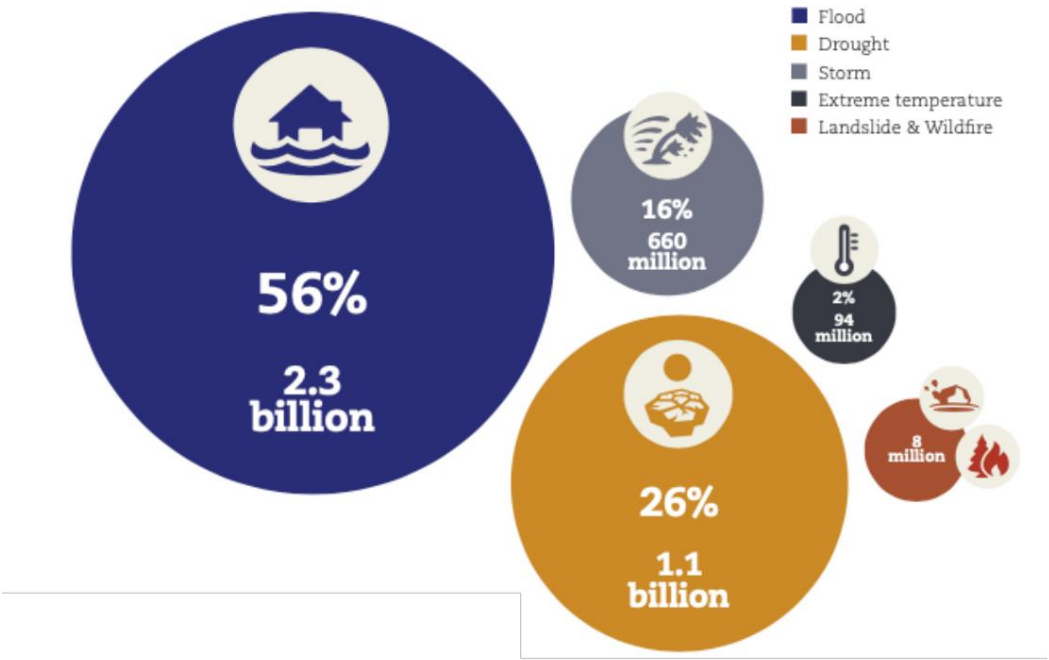
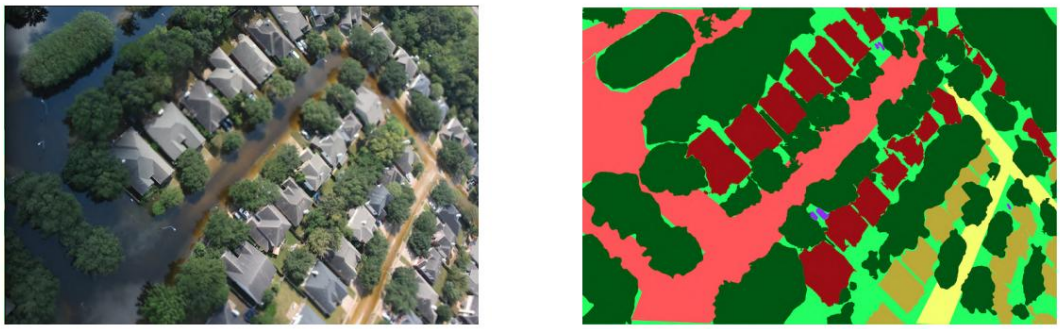


Figure 1.2. Number of people affected by meteorological disasters (1995-2015) (NB: i deaths are excluded from the total number of victims) [1].

Cross (ARC) to test the capabilities of UAVs in emergency situations and disastrous events, the following report described drones as a technology that brought "immediate benefits" to both civilians and rescuers. To date, drones have been used in many such events, including the 2008 Wenchuan earthquake, the 2011 Fukushimi Daiichi crisis and the handling of the Typhoon Haiyan disaster in the Philippines in 2013. to date, many organizations, mainly humanitarian, have launched programs that aim to integrate UAVs in the management operations of these events, including: the WFP (UN World Food Program), the World Bank, UNHCR (UN High Commissioner for Refugees). In particular, drones have found utility in all three phases mentioned above: in the prevention phase they are often used to monitor river beds or other basins; during floods they are instead used to map the affected areas, for search and rescue operations, for the transport of loads in areas otherwise unreachable, to provide real-time information from above on the state of the roads to guide rescue teams, but also to provide information on the most affected buildings to prioritize operations; finally, in the post-flood for the assessment of damage [2]. One of the main advantages of drones lies in the speed of their use and in the high resolution of the images they can capture. In particular, with them it is possible to obtain higher resolution images of the affected areas and in shorter times than other resources, such as satellite images [3]. Furthermore, with the processing of these images and thanks to their high resolution, it is possible to extrapolate very important information in this area. For example, among the various uses, buildings and roads that are flooded can be distinguished from those that are not flooded, some objects of interest such as people or vehicles can be identified, or the level of damage of a building can be classified. In fact, this work concerns semantic segmentation, one of the types of processing that is often applied to images, to provide very useful information, especially in the phase during the flood [4]. Semantic segmentation is nothing more than the classification of each single pixel of the image in one of the predetermined classes (Figure 1.3) and is distinguished from the classification task, as the latter concerns the assignment of a single class to the entire image . In particular, the model of interest of this work deals with the assignment to each pixel of the image of one of the following 9 classes: flooded building, non-flooded building, flooded street, non-flooded street, water, tree, vehicle and lawn . Of these, especially the first four classes are of great interest in the management of flood events, as distinguishing which buildings are flooded from which not, can be fundamental to guide and prioritize rescue operations, while identifying flooded roads it can instead be useful to distinguish which roads are passable by land.

1.2 State of the art

As we will see later, neural networks, compared to more traditional methods, are able to grasp information of a higher level. In particular, their complex architecture allows them to approximate very complex patterns, which traditional methods fail to grasp. In fact, as we will see in this paragraph,



(to) **(b)**

Figure 1.3. An example of semantic segmentation. (A) shows the input image e
la (b) the model output, where each color corresponds to a class: flooded road
(light red), flooded building (dark red), non-flooded street (light yellow), building
not flooded (dark yellow), tree (dark green), lawn (light green), vehicle (purple).

in the state of the art of semantic segmentation of aerial images, the majority
part of the methodologies used are Deep Learning. One of the architectures
most present in the literature and used in this field is the UNet. In several works
[5, 6, 7] the classic version proposed in [8] is used, while in others they are
use variations. For example, in [9] the version with the added attention mechanism, called
AttentionUnet [10], is used; in [11] comes
used in conjunction with SegNet [12] to build such an architecture
encoder-decoder (Figure 1.4); finally it is resumed in [13], in which the
MobileNet network [14] as a backbone, ie as the first part of the UNet (the enumeration der)
responsible for the extraction of the features. Returning instead to [9], the dataset
RescueNet is used as a benchmark for several models including, in addition to the
AttentionUnet, the PSPNet [15], the DeepLabV3 + [16] and the ENet [17]. In other works
instead, ensemble architectures made up of multiple models are used. In particular, in [18]
a CNN ensemble is tested on the S-PRS Vaihingen Dataset. In
in this case the ensemble is made up of several versions of the same model, namely the
same architecture trained several times on the same dataset but with initializations
different.

The authors' idea is that, being the space of the loss function in which the
extremely non-convex model, with many dimensions and with many local minima,
initializing a model with different parameters is almost guaranteed that
it will converge at different points. Consequently, their approach consists in training these
different versions of the model and then, in the inference phase, taking the
average of their predictions. A similar approach is also used in [19], in
which the ensemble is composed of the same model but taken in different stages of the
training. In particular, the authors highlight how, with this approach, it is
It is possible to train an ensemble without adding more time than to train a single model. As
in [9], they are also used in [20, 21]
models based on the attention mechanism [22]. In particular, the authors
of [20] show how a hybrid attention mechanism, that is the union of different types of
attention (on channels, on space and on classrooms) is able to capture
long-range dependencies, thus helping the model to produce feature maps

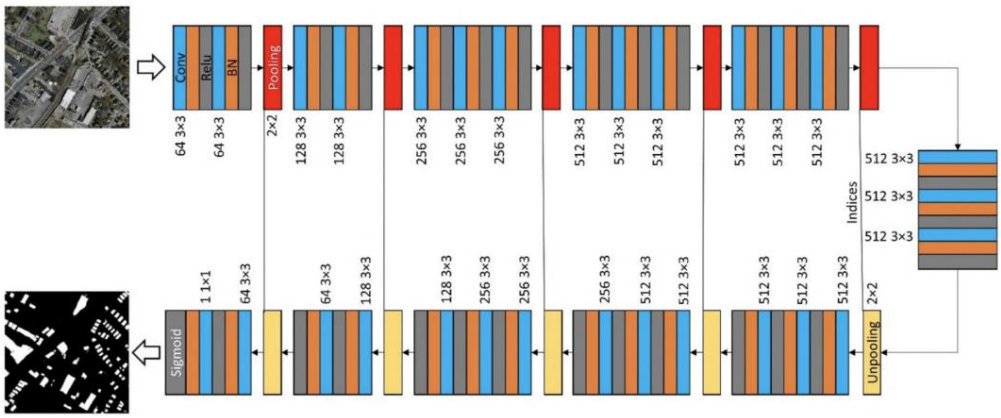


Figure 1.4. Architecture of the Seg-Unet model proposed in [11] with a combination of the components of Segnet (pooling indices) and those of Unet (skip connection).

quality. On the other hand, in [21] the authors, in addition to proposing a model based on the structure of an FCN with the addition of the *channel attention mechanism,* propose an approach based on the use of two types of data paid by two types of backbone. In particular, the first part of the model is composed of two ResNet101, one that takes the image as input and one that instead takes auxiliary data as input such as the NDVI (Normal zed Difference Vegetation Index) and others. In the second part, however, the two feature maps are concatenated and passed to the *channel attention module,* and then finally pass into the upsample part to produce the final masks (Figure 1.5).

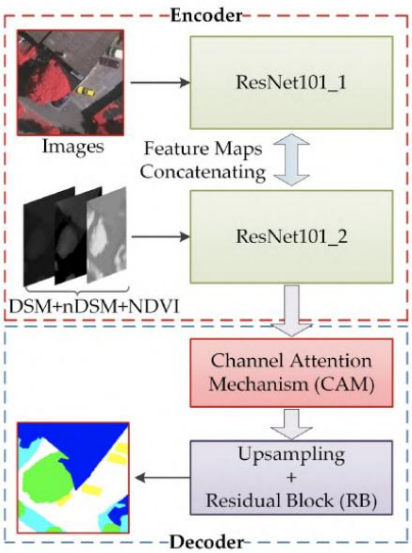


Figure 1.5. Model architecture based on the use of the *channel attention mechanism* proposed in [21].

In [23] the masks within the dataset, before being used to ad-

right their model, they undergo a first phase of processing in which they are transformed into *distance maps.* In particular, for each mask and for each class a binary mask is produced, whose pixel values represent the distance of that pixel of that particular class from the edge of the object to which it belongs (Figure 1.6).

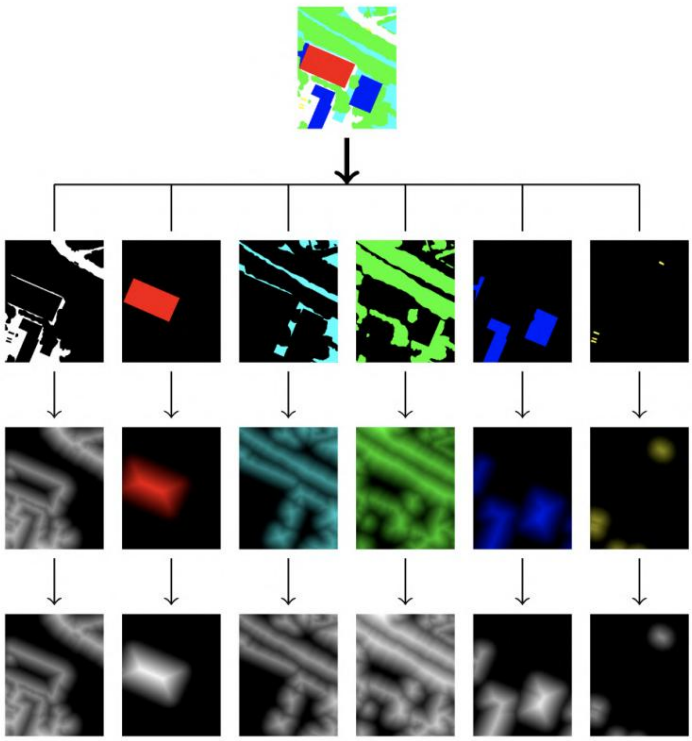
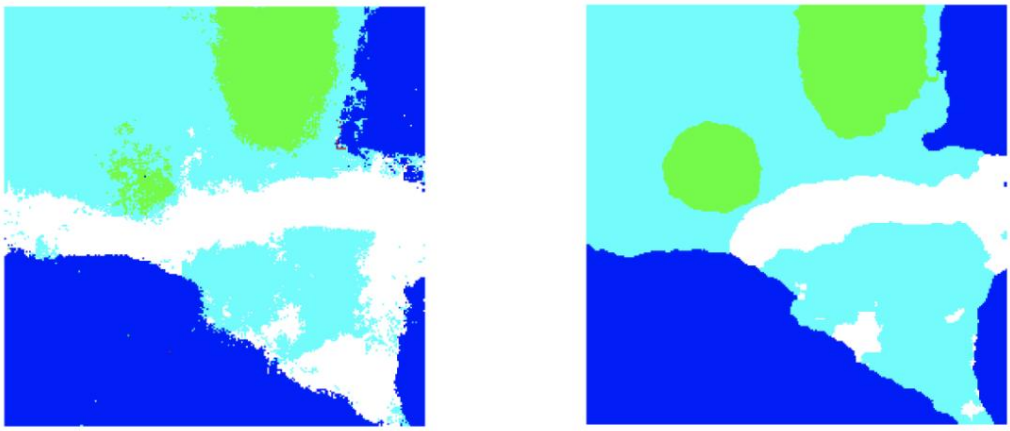


Figure 1.6. Illustration of the production of a *distance map* starting from a mask [23].

The reason for using distance maps is that, thanks to the fact that they encode more spatial information than simple masks, the outputs of the model obtained from their use in training result in less noise, with more spatial coherence and the edges of objects appear more defined (Figure 1.7).

A variant of semantic segmentation is the one addressed in [24], that is the open-set semantic segmentation, which consists in the task of semantic segmentation, with the addition of the fact that the total number of classes is unknown. Consequently, open-set semantic segmentation can be described as the task in which each pixel can be tagged as belonging to one of the classes learned during training, or as an unknown class. The difficulties of this type of task highlighted by the authors are mainly: the diversity of patterns in the unknown class and the similarity between the patterns of the known classes and those of the unknown classes. The approach proposed by the authors consists in segmenting the image with a CNN, which classifies pixel by pixel, and determining a threshold for which if the CNN softmax output does not exceed it, that pixel is classified as an unknown class. . In addition, they also propose a further



(to) **(b)**

Figure 1.7. The figure shows the result of the model without the use of *distance maps* (a) and the result of their use (b) [23].

step to improve the quality of the masks produced and decrease the number of pixels of the unknown class. In particular, the proposed method, called *Morphological Filtering,* consists in reassigning each pixel to the most present class in its neighborhood and, in practice, the effect can be seen as the erosion of the regions classified as an unknown class (Figure 1.8).

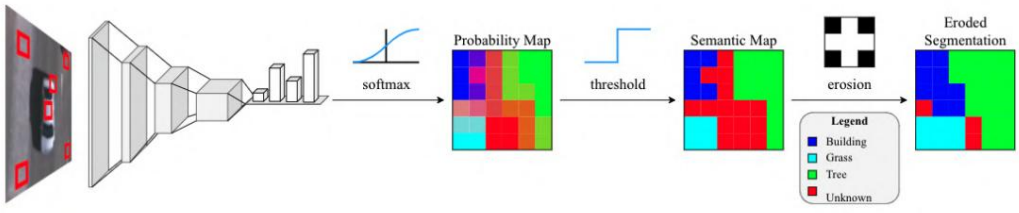


Figure 1.8. Illustration of the method proposed in [24].

Other jobs, such as [25], use segmentation as an intermediate step for other tasks. In particular, in the latter work cited, the masks produced are used for estimating the depth of flood waters and the model used is a version of the FCN [26] with stride 8 (FCN-8s) and VGG-16 as backbone. In [27] different versions of UNet [8] and LinkNet [28] are tested, combining different models for the backbone and the decoder part.

Furthermore, the authors showed that on average on almost all aspects, the models that had the pre-trained backbone on ImageNet performed better.
In some works, however, such as [29], in addition to Deep Learning architectures, more traditional Machine Learning algorithms are also tested. Their work concerns the task of segmenting the different types of vegetation and, in particular, their approach was to test different classifiers of ML, including Random Forest, decision trees and K-nearest neighbors. On the other hand, as regards the

the Deep Learning reading used by them, it is the SegNet [12]. Finally, in [30] the same specific task of this work is addressed, namely the FloodNet dataset.
In particular, the authors' approach consists in testing various architectures, including ENet [17], PSPNet [15] and DeepLabV3 + [16]. Specifically, the authors highlight how the best performing architectures are context-based ones and among these, the best was the PSPNet.

1.3 Contribution and Outline

In this work the task of semantic segmentation of aerial images of the FloodNet dataset is explored, through Deep Learning approaches. Specifically, a new approach is proposed based on the resolution of the main difficulties identified in the dataset. In particular, the main contribution of this work can be summarized in the following points:

- a *data cleaning phase,* to remedy the substantial presence of errors in the dataset masks.
- an offline data augmentation phase, useful for dealing with a strong imbalance of the dataset towards some classes and the presence of others to a much lesser extent.
- use of a context-based architecture, never used on this dataset, aimed at obviating the intrinsic difficulties of particular classes, whose semantics are strongly based on their context.

As for the structure of the paper, it is structured as follows.
In Chapter 2, the general problem of segmentation will be explored by dealing with the various existing methodologies, both traditional and Deep Learning. Later, in Chapter 3, we will deal with the main concepts of Deep Learning, also examining some of the most well-known architectures. In Chapter 4, on the other hand, we will go into the details of the specific task of this work and will show, in addition to the architecture, also the main methodologies used. Subsequently, in Chapter 5 we will illustrate the various experiments carried out throughout the work and their results. Finally, in Chapter 6 the conclusions of the work will be drawn.

Chapter 2

Image segmentation

Segmentation is one of the most popular tasks in the field of Deep Learning applied to images and, in general, in the field of Computer Vision. In particular, it consists in producing a mask of the same size as the input image, where the value of each pixel represents the class to which it has been assigned, or in other words, partitioning an image into regions with a very precise semantic value (Figure 2.1).

Thanks to the spatial information that can be obtained with segmentation, many fields benefit from it, including the field of autonomous vehicles, pedestrian detection, computer-assisted medical diagnosis and many others [4].

In the literature, segmentation has been applied to a large variety of data. Some of the most used types are:

- **grayscale:** a type of image that has only one channel and the pixel value essentially represents the amount of light in the pixel.
- **RGB (red, green and blue):** probably the best known type, it has three channels that respectively represent the amount of red, green and blue of the pixel.

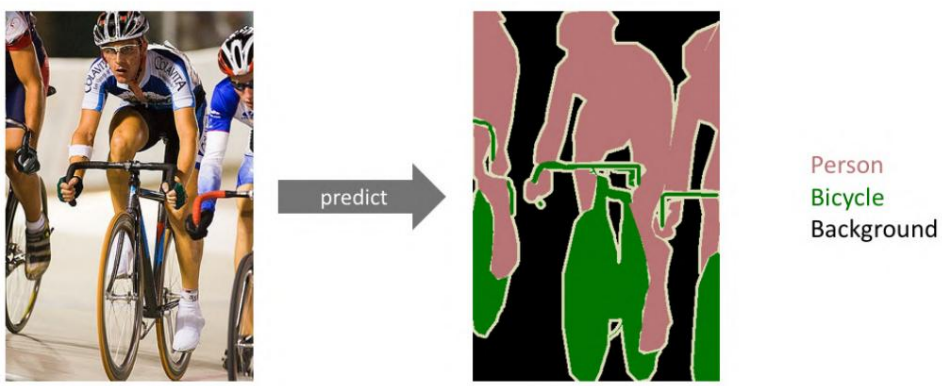


Figure 2.1. An example of the result of segmenting an image.

- **RGB-D:** a type of image that, in addition to having the three RGB channels, also has a fourth channel that represents the depth of that pixel, or the distance from the camera, information that is often very useful in the field of semantic segmentation [31].

- **3D:** a type of data often present in medical literature, such as in the field of CT (computed tomography), where images are captured with X-ray instruments [32].

As mentioned above, segmentation is one of the most popular tasks and over the years in the research field, many steps forward have been made, developing increasingly performing models and algorithms. In particular, the performance of these models was measured by testing them on datasets, which were used as benchmarks to compare the various methods. Among the best known and used benchmark datasets there
I am:

- **PASCAL-VOC** (PASCAL Visual Object Classes) [33]
- **Cityscapes dataset** [34]
- **ADE20K** [35]

In the field of computer vision, segmentation is approached with different methods. In particular, there are traditional methods, which do not make use of Machine Learning techniques; Machine Learning methods; and finally Deep Learning methods, which instead exploit architectures which, as we will see later, are able to learn very complex patterns that traditional and Machine Learning methods often fail to grasp. As for the traditional methods, the techniques are varied: some are based on the extraction of features and others on the preprocessing of the input image; instead, as regards Machine Learning methods, they are based on transforming the image pixels into a representation that can then be used by various Machine Learning algorithms to classify them, such as K-Means or SVM (Support Vector Machines). Instead, Deep Learning architectures are different and among the most used are UNet [8], DeepLabV3 [36], PSPNet [15], FCN [26] and others. We will see later the Deep Learning concepts on which these architectures are based. Beyond the methodologies used, a very important aspect of developing an algorithm, an architecture, or using an existing one, is the choice of the evaluation metric. There are several metrics:

- **Accuracy:** This is probably the simplest and most intuitive, ie it measures the number of correctly classified pixels, clearly in proportion to the total number of pixels. This metric is generally one of the most used in the field of Machine Learning. It is defined as:

$$Accuracy = \frac{\sum_{i=1}^{k} n_{ii}}{\sum_{i=1}^{k} t_i}. \quad (2.1)$$

  Where k is the number of classes, n_{ij} is the number of pixels of class i and classified as of class j and t_i is the total number of pixels of class i, that is:

$$ti = \sum_{j=1}^{k} nij \,. \tag{2.2}$$

Also, apart from general accuracy, which basically measures the accuracy of the model, but disregarding individual classes, we can also use the accuracy of a class. In particular, the accuracy of a single class *i* is defined as the ratio between the number of pixels of that class correctly classified by the model and the total number of pixels of that class:

$$Accuracyi = \frac{nii}{you} \,. \tag{2.3}$$

The main disadvantage of using accuracy is that in many tasks some classes prevail over others, therefore using accuracy can lead to a high value, but that is actually caused by the predominance of that class. For example, if 80% of the pixels in an image are of a class, a model that classifies all the pixels in the image as belonging to that class would obtain an accuracy of 80%.

- **Average Accuracy:** This and other metrics address the aforementioned problem of accuracy. This metric, in particular, represents the average of the accuracy of the single classes and solves the problem by normalizing the accuracy with respect to the total number of pixels:

$$MeanAccuracy = k \frac{1}{-} \sum_{i=1}^{k} Accuracyi = k \frac{1}{-} \sum_{i=1}^{k} \frac{nii}{you} \,. \tag{2.4}$$

- **Mean intersection over union (mIoU):** also called *Jaccard index,* it is the average of the *intersection over union (IoU)* of the *k* classes, where the IoU of a class is essentially the ratio between the intersection and the union of two sets: the set of pixels of that class in the mask and the set of pixels of that class in the prediction, that is:

$$IoU = \frac{target \ddot{y} prediction}{target \ddot{y} prediction.} \tag{2.5}$$

also definable as

$$IoU = \frac{nii}{ti - nii + Pk \, j = 1 \, nij} \,. \tag{2.6}$$

and consequently the mIoU is defined as:

$$mIoU = k \frac{1}{-} \sum_{i=1}^{k} IUi = \frac{1}{k} \sum_{i=1}^{k} \frac{nii}{ti - nii + Pk \, j = 1 \, nij} \,. \tag{2.7}$$

- **Frequency weighted intersection over union (FWIoU):** a variant of the mIoU, which instead of simply calculating the average of the IoUs, calculates a weighted average with respect to the number of pixels of the classes (frequency), that is:

$$FW\ IoU = (\mathrm{X}\ ti)\ \ddot{y}1\mathrm{X}\ tiIoUi = (\mathrm{X}\ ti)\ \ddot{y}1\mathrm{X}\ \frac{nii}{-\ nii + Pk\ j = 1\ nij}\ .\ (2.8)\ ti\ ti$$

- **Confusion matrix:** it is not a real metric, but it is a widely used method also and above all in the field of classification. The matrix of confusion is very useful in particular when you want to investigate the nature of the errors of your method. Specifically, it shows us for each class *i* the number of correctly predicted pixels *nii* but, even more importantly, for each class it shows us the subdivision of the errors in the other classes, that is *nij* for *j* ÿ [1, *2, ..., i - 1, i + 1, ...,* k]. Consequently, we can represent formally the confusion matrix as:

$$ConfusionM\ atrix = [nij\ ].\qquad (2.9)$$

  where *i, j* ÿ [1, *2, ...,* k].

Furthermore, in the field of Computer Vision, as well as in the algorithmic world, the methods are evaluated according to two further criteria: the temporal complexity, very important especially in some fields of application where the time necessary to process an image and produce its segmentation it must not be greater than a certain threshold, such as the field of autonomous vehicles; and spatial complexity, that is the amount of memory that the algorithm needs, a very important metric especially in fields where processing takes place on devices that do not have an abundance of memory, such as smartphones and cameras.

2.1 The segmentation of images in humans

Starting from how humans perceive an image and are able to naturally and intuitively segment it, the functioning of these intuitions is based on what is called "Gestalt psychology" [37] (from the German *Gestaltpsychologie,* 'psychology of form' or 'representation'). Gestalt psychology explains how a human can perceive an image and organize it in a complex system. In particular, he explains how, looking at the world around him, he is able to perceive complex scenes composed of many groups of objects on a background, which in turn can be made up of other parts and so on. The way in which humans manage to achieve such a remarkable perceptual result, given the fact that visual input is, in a sense, just a spatial distribution of variously colored individual points, is based on principles that define the intuitions behind grouping of parts of the image. There is no precise list of Gestalt principles, but there are some that are more discussed and more commonly used [38]:

- **principle of good form:** the structure of objects that we tend to perceive it is always the simplest.
- **principle of proximity:** formalizes the intuition behind the fact that we tend to group elements that are close to each other.
- **principle of common destiny:** tendency to group together elements that have a coherent movement between them.
- **principle of similarity:** tendency to group similar elements together (in color, shape, size, ...)
- **principle of good continuity:** tendency to group elements by for sea objects and continuous and coherent forms in space.
- **principle of past experience:** the observer tends to group visual elements in a coherent way to the way he saw them in his past.

In general, these principles formalize insights common to the psychology of all humans. The problem is that these intuitions are often very difficult to transform into a mathematical language or an algorithm, and hence the difficulty of segmentation and other tasks in the field of Computer Vision, which like the latter are often intuitive, but very difficult to transform into an algorithm. Moreover, even for humans, at times, the segmentation task is complex and the results, when the subject changes, are often different from each other. This gives us an insight into how, unlike other tasks such as classification, the segmentation problem is particularly difficult even to define [39].

2.2 Classical approaches

As previously stated, the image segmentation task can be faced with several classical approaches. Some of these transform the task into an optimization problem, then trying to solve it with iterative algorithms, while others transform it into a *clustering problem,* transporting the image in a multi-dimensional space and then using Machine Learning algorithms.

2.2.1 Threshold methods

One of the simplest and most intuitive methods is the thresholding method . In practice, it consists in defining a numerical threshold of an image feature, and then classifying the pixels according to this threshold. In its simplest form, by defining a single threshold, the image is segmented into two classes; in its multiclass variant, on the other hand, more thresholds are defined, thus creating intervals that represent the different classes. The feature used most often is that of pixel intensity (in grayscale images), but many others can be used, this depends a lot on the field of application and the nature of the image. For example, a feature that is often very useful for segmenting an image is depth. The main problem is that this type of feature, apart from the

in case it is provided directly from the image (RGB-D), it is often very difficult to extract. Clearly, this method often returns approximate results, and furthermore, finding the optimal threshold value is not at all easy and often requires in-depth knowledge of the domain.

2.2.2 Methods of divisions and funds

As we saw in the previous paragraph, one of the simplest techniques is to define a threshold with which to classify the pixels. Unfortunately, in most cases, finding this threshold is very difficult and very often it is not even unique, i.e. the images sometimes have strong differences from one region to another, consequently a good threshold for one part may not be good for a 'other.
Given this difficulty, one technique to solve it is to divide the image into different parts or, on the contrary, try to join similar parts of the image together. One of the first algorithms that falls into this category is the watershed algorithm *[40],* which treats grayscale images as a topographic map, transforming the intensity of pixels into height. In particular, using the metaphor of a flood, the algorithm divides the image into different hydrographic basins, i.e. the segmented regions of the images are composed of all those points from which the water would end up in the same point (the minimum) (Figure 2.2). Unfortunately, even this algorithm often returns approximate results, in particular, one of the biggest problems is that, by associating a region with each local minimum, it often over-segments the image. In fact, this algorithm is mainly used in interactive systems where the segmentation is assisted by a user who defines the center of the objects of interest.

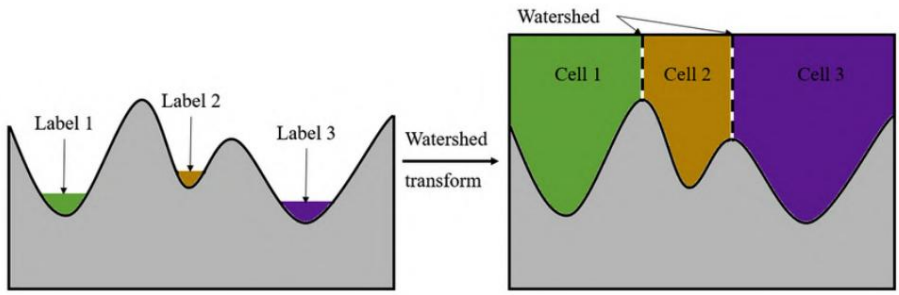


Figure 2.2. Illustration of how the watershed algorithm finds the thresholds (watershed in the image) with which to segment the images.

Finally, there are other methods that, instead of trying to divide the image into regions, try to join together regions that have a certain similarity with a *bottom-up approach.* One category of these are graph-based methods.

2.2.3 Methods based on graphs

This category of methods is based on the representation of the image as a graph *G = (V,* E) where the vertices *V* are the pixels and the arcs *E* are between the adjacent pixels.

One of the algorithms that belongs to this category is the one proposed in [41], in which the weights represent a measure of dissimilarity between the connecting pixels. This measure of dissimilarity *w (e)* can take several forms and the simplest is probably the difference in intensity between the two pixels. Any region *R* has an internal difference measure called Int (R), which is defined as the maximum weight of an arc within the minimum spanning tree of R, i.e. MST (R):

$$\text{Int (R)} = \max w(e).\ e \ddot{y} MST(R) \tag{2.10}$$

Furthermore, for two adjacent regions, i.e. those which have at least one arc connecting them, the difference between them is defined as the minimum weight of an arc connecting them:

$$\text{Dif (R1, R2)} = w(e).\ \min e = (v1, v2) \mid v1 \ddot{y} R1, \tag{2.11}$$

The algorithm iteratively joins together any two regions R1 and R2 if their difference Dif (R1, R2) is less than the minimum internal difference of the two regions
MInt (R1, R2), where

$$\text{MInt (R1, R2)} = \min(\text{Int (R1)} + \ddot{y}(\text{R1}), \text{Int (R2)} + \ddot{y}(\text{R1})). \tag{2.12}$$

Where $\ddot{y}$ (R1) represents how much the difference between the two regions must be greater than the internal differences in order not to be merged, but segmented into two different regions, and is defined as:

$$\ddot{y}(R) = \left| \frac{k}{R} \right| . \tag{2.13}$$

Where *k* is a constant parameter and | *R* | is the size of the region. In reality, this component $\ddot{y}$ can take different forms depending on the specific task and on how we want to define the goodness of a region. Finally, the rule by which the algorithm iteratively decides whether or not to join two regions is the following: join R1 and R2 if Dif (R1, R2) < MInt (R1, R2).

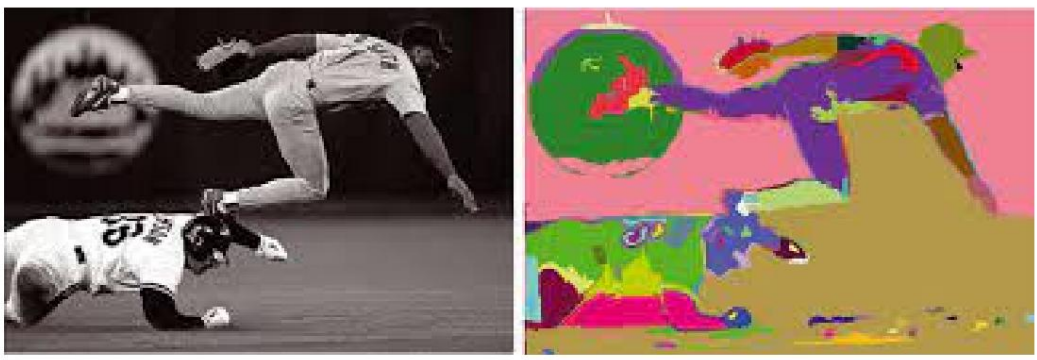


Figure 2.3. Example of the result of the graph-based algorithm proposed in [41] applied on a grayscale image.

Normalized cuts

Another very popular algorithm belonging to this category is the one proposed in [42]. In this algorithm, the weight of an arc *wi, j* between the two pixels *i* and *j* represents their affinity. The idea behind the algorithm is to partition the graph into regions that have weak affinities, ie regions connected by edges with low weights.
To define the cost of a cut between two regions *A* and B, the following measure is defined:

$$\text{cut (A, B)} = \sum_{j \,.\, iÿA,\, jÿB} wi, \tag{2.14}$$

Having defined this cost, it is not enough to find the cuts that minimize it. The problem is that by using only this cost, the resulting cuts are normally those that isolate single pixels, consequently another component that penalizes small subgraphs must be added to the cost. A better measure of the cost of a cut is the following measure, which is the cost of the normalized cut with respect to the association of the two subgraphs:

$$NCut\ (A, \text{B}) = \frac{\text{cut (A, B)}}{assoc\ (A,\ V\ )} + \frac{\text{cut (A, B)}}{assoc\ (B,\ V\ )}. \tag{2.15}$$

Where *assoc (A, V)* = P *uÿA, tÿV wut* that is the sum of the weights of the arcs between the nodes *A* and all the nodes of the total graph *V* (including those *A* itself). In this way, by normalizing the cost with respect to the association, we penalize very small subgraphs. Unfortunately, the difficulty with this algorithm is that minimizing *NCut* is an NP-complete problem.

2.2.4 Method of active contours

Sometimes, the segmentation task is transformed into the task of detecting the boundaries of objects, that is, their edges. In this context, active *contours are used,* also called *snakes* (due to their shape), which are nothing more than curves that, iteratively, tighten around an object, gradually taking its shape. until, hopefully, they match it perfectly. This type of method is often very useful in contexts where, for example, it is necessary to trace the shape of an object in a video or of an object that changes perspective; the reason is that the performance of this method depends very much on the initialization of the curve. In particular, when the curve is initialized with a certain shape and close enough to the object, you are more likely to get more precise segmentation. For example, if in a video the algorithm has managed to segment the object of the previous frame, or the latter has been segmented by hand by the user, that outline, barring major changes, is an excellent initialization for the next frame. (Figure 2.4).

This type of algorithm relies on local information from the pixel intensity gradient and uses it to attract the curve towards the edges of the object. In particular, the edges, being characterized by a sudden change in intensity, have a high gradient, consequently the algorithm iteratively tries to bring the points that form the curve into pixels with high gradients. This mechanism

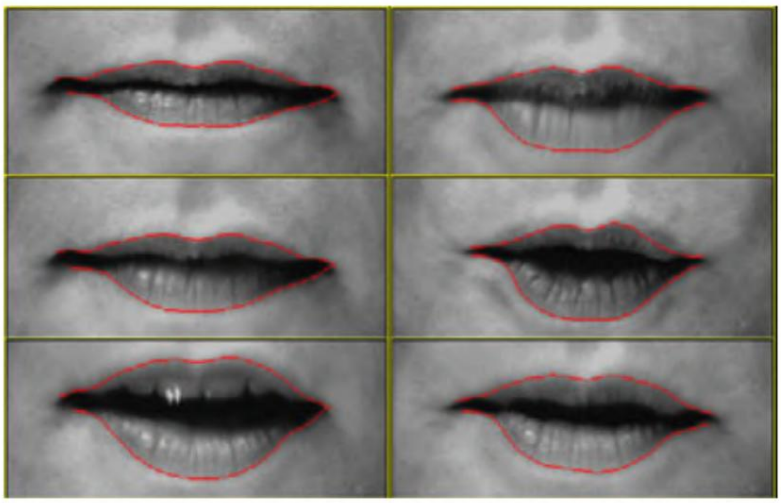


Figure 2.4. Example of application of active contours: lip tracking.

it is formalized with an optimization problem, ie we want to maximize the sum of the intensity gradients of the pixels where the points of the curve are. As for the definition of a contour, it is defined as a set of points in two-dimensional space connected by straight lines:

$$V = \{vi = (xi\ ,\ yi) \mid i = 0, 1, 2, ..., n - 1\}. \quad (2.16)$$

Where *n* is the number of points that form the curve. As mentioned above, we want to maximize the sum of the intensity of the gradients and to do this we use the magnitude of the gradient of the intensity of the pixels squared, that is || *ÿI* || *2* . Also, since the curve must be attracted by higher gradients, the "force field" of the gradient peaks must be expanded, otherwise a point not very close to an edge would not be attracted. To do this we use the Gaussian blur applied to the gradient magnitude and consequently the value to be maximized becomes || ÿnÿ ÿ I || 2 . Finally, the problem is transformed into finding the *Eimage minimum,* where:

$$Eimage = - nX\ \ddot{y}\ \underset{i = 0}{1}\ \| \ddot{y}n\ddot{y}\ \ddot{y}\ I\ (vi)\ \|\ 2\ . \quad (2.17)$$

At this point, the algorithm does nothing but iteratively try to optimize *Eimage* by bringing each point to a new position, until it reaches a certain threshold established in advance. The problem with this version is that it is very sensitive to noise and very often the outline tends to take on strange shapes, due to the presence of the gradient noise around the object. To overcome this difficulty, and because in general we want a curve that respects some natural characteristics, as if simulating a physical object, such as elasticity and softness, a second *Econtour* component is added to the optimization problem, which represents these shape constraints. . In particular, *Econtour* is in turn composed of two other components *Eelastic* and *Esmooth,* which represent the two

features just mentioned. In reality, other components can be added to *Econtour* depending on the specific task and how you want to constrain the shape of the curve. For example, very often when the shape of the object of interest is known a priori, a component can be added to the problem that penalizes shapes other than this. Consequently, the *Econtour* formula is as follows:

$$Econtour = \ddot{y}Eelastic + \ddot{y}Esmooth. \tag{2.18}$$

Where $\ddot{y}$ and $\ddot{y}$ are two coefficients that represent the weight we want to give to the two elasticity and softness constraints. As for the elasticity constraint, it represents the desire that the curve simulate the behavior of an elastic, i.e. when two points of the curve are more distant there is a sort of stronger force of attraction and consequently the curve tends not to space the points too far from each other. Below is the formula of the elasticity component of a point of the contour *v (s).*

$$Eelastic\ (v\ (s)) = \left\| \frac{\ddot{y}}{\ddot{y}sv\ (s)} \right\|^2. \tag{2.19}$$

That is, the square of the derivative before the point of the contour, which represents the intuition of the fact that one does not want great distances from one point to another. As for the softness component, however, it is defined as:

$$Esmooth\ (v\ (s)) = \left\| \frac{\ddot{y}^2}{\ddot{y}\ 2s} v\ (s) \right\|^2. \tag{2.20}$$

That is the square of the second derivative of the boundary point, which instead represents the intuition of the fact that we do not want sudden changes. Since the contour curve is actually composed of discrete points, the formulas of the two components become:

$$Eelastic\ (v\ (i)) = (xi + 1 - xi)^2 + (yi + 1 - yi)^2. \tag{2.21}$$

$$Esmooth\ (v\ (i)) = (xi + 1 - 2xi + \text{xi ÿ } 1)^2 + (yi + 1 - 2yi + \text{yi ÿ } 1)^2. \tag{2.22}$$

and therefore

$$Eelastic = \text{nX ÿ } 1 \underset{i = 0}{} (xi + 1 - xi)^2 + (yi + 1 - yi)^2. \tag{2.23}$$

$$Esmooth = \text{nX ÿ } 1 \underset{i = 0}{} (xi + 1 - 2xi + \text{xi ÿ } 1)^2 + (yi + 1 - 2yi + \text{yi ÿ } 1)^2. \tag{2.24}$$

Finally, the total problem becomes minimizing the following value:

$$Etotal = Eimage + Econtour. \tag{2.25}$$

Figure 2.5 illustrates an example of how the result improves by adding the *Econtour* component .

Figure 2.5. Application example of the active contour method. In both images we can see a blue outline (initialization outline) and a red one (final outline). In particular, on the left the version of the algorithm that minimizes *Eimage has been used,* while on the right Etotal is *minified* and we can see how the one on the right obtains a better result.

As already mentioned, the performance of this algorithm depends a lot on the goodness of the initialization, moreover it performs well especially with images that contain only one object, even if it can be used on images that contain more than one. In particular, depending on the nature of the objects, the parameters *ÿ* and ÿ can be varied, to allow the contour to adapt better to the objects (Figure 2.6). In other cases, as already mentioned, it may be useful to add additional components to *Econtour* in addition to *Eelastic* and *Esmooth.* Finally, there is a variant of the algorithm which, instead of starting the outline from a wider shape and then making it contract on the object, starts the outline from the inside and makes it swell up to shape its edges.

2.2.5 Clustering methods

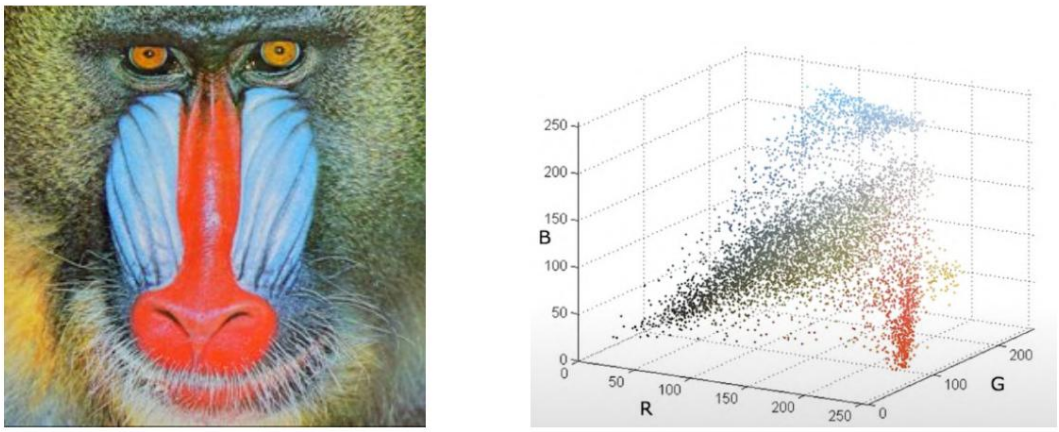
Just as graph-based methods transformed the image into a graph by matching each pixel to a node, so too did clustering methods transform the image into a distribution in a multidimensional Euclidean space. In particular, the first step in a method of this category is to understand which features to rely on to map each pixel to a point in space. One of the simplest choices, in the case of RGB images, is to use the three image channels, namely the amount of red, green and blue (Figure 2.7). Other widely used features are brightness, pixel position, depth (RGB-D), but also more complex features such as texture. Once again, the choice of these features is very important, but above all it strongly depends on the field of application and the nature of the images, therefore a thorough knowledge of both is required.

Once the mapping of the image in the chosen space is completed, each pixel] where *k* is *i* is represented with a feature vector *fi = [f* $i^1, f\,i^2, f\,i^3, \ldots, f\,i^k$ the number of features chosen. As in other methods, a measure of their similarity is used to guide the algorithm in grouping pixels. In particular,

(a) small $\ddot{y}$

(b) large $\ddot{y}$

Figure 2.6. Example of segmentation of two objects with an active contour, but varying the coefficient ÿ. As you can see, lowering $\ddot{y}$ allows the contour to adapt better to the two objects. Clearly, how to modify the two coefficients $\ddot{y}$ and $\ddot{y}$ depends on the nature of the image.



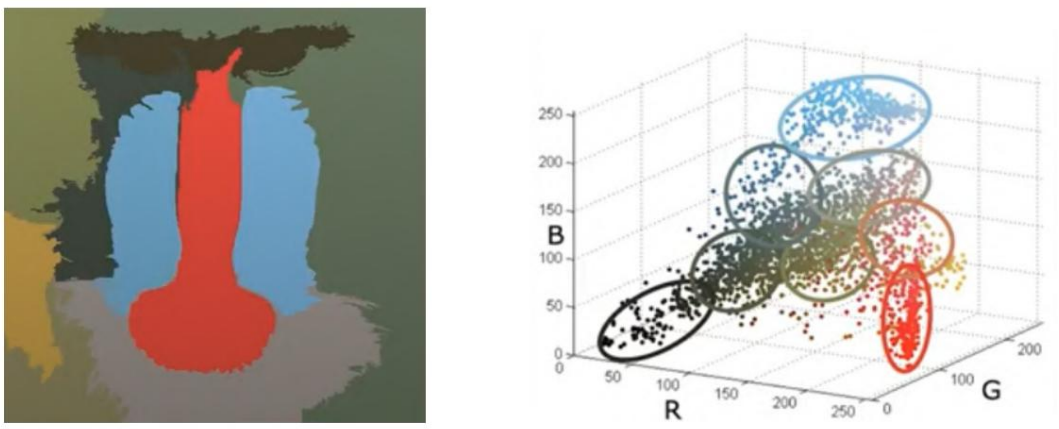
(to) **(b)**

Figure 2.7. The two figures are an example of how an image (a) can be mapped in a three-dimensional space (b) using the three image channels (RGB) as a feature.

the similarity is represented by the distance in the space of the pixel feature vectors and to calculate it is used what is called distance *L* or Euclidean distance, defined in dimensional space as:

$$S(fi, fj) = \sqrt{\sum_{k} (f_{\sim}^{k} - f j^{k})^{2}} \quad (2.26)$$

At this point, the segmentation task has been totally transformed into a clustering problem (Figure 2.8) and any algorithm that solves this type of problem can be used. There are several algorithms that solve this type of problem, among the best known are K-Means and Mean Shift.



(to) **(b)**

Figure 2.8. The two images show the result of a clustering algorithm applied to the image mapping of Figure 2.7a. In (a) the final result, that is the segmented image; (b) instead shows the clusters in three-dimensional space.

K-means

Probably the best known clustering algorithm is *k-means* [43, 44]. Its operation is quite simple and is based on the objective of finding the centroid or midpoint of each cluster, that is to find the point that minimizes the total intra-group variance *V ar,* which is nothing more than a measure of the variability at all. inside the cluster. Formally, the problem is defined as finding:

$$\arg\min_{S.} \sum_{i=1}^{k} | Si | V ar (Si). \tag{2.27}$$

Where *S* = S1, S2, ..., *Sk* is the set of clusters. Once the centroids of each cluster have been found, whose number *k* must be defined in advance, the algorithm classifies each point in space by determining which of the *k* centroids is the closest. As for the first phase of the centroid search, the algorithm starts by initializing *k* centroids and calculating the clusters according to the closest centroid rule.
At this point, the algorithm iteratively repeats this mechanism, recalculating the centroids of the clusters each time until they converge, that is, until the centroids no longer change position or in any case their change is below a certain threshold. Once again, unfortunately, the performance of this algorithm strongly depends on the goodness of the initialization. In this case, the initialization can be done in different ways: the simplest thing is to initialize them micamente, but clearly it is also the least efficient choice; another very simple method consists in initializing them randomly, checking however that none of the centroids are very close and in that case reinitializing them; a third method is to choose centroids that are uniformly distributed in space; Finally, a last method can be to use k-means first on a sub-portion of the distribution, and then use the resulting centroids as initialization to apply the algorithm to the whole distribution.

Mean Shift

As mentioned in the previous paragraph, the k-mean algorithm has the disadvantage of being very sensitive with respect to the initialization of the centroids. Also, another disadvantage of k-mean is that the user has to decide the number of clusters in advance. The *mean shift* algorithm [45, 46] tries to mitigate these two problems. Mean shift is also an iterative algorithm and in particular, the general mechanism is based on finding the modes of the distribution of points in space, which at a high level can be thought of as the points where the distribution is densest. The operation of the algorithm is as follows: for each point it selects a window of dimension W around it, which is the only parameter of the algorithm, and subsequently calculates the mode of the distribution of the points within that window, which will be the center of the window on the next iteration. By repeating this mechanism until the mode shift vector, i.e. the distance of the mode of one iteration to that of the next iteration, is zero or falls below a certain threshold, the algorithm finds the mode towards which the point of departure was "attracted". Finally, the mean shift creates the clusters by finding the "basins of attraction", that is

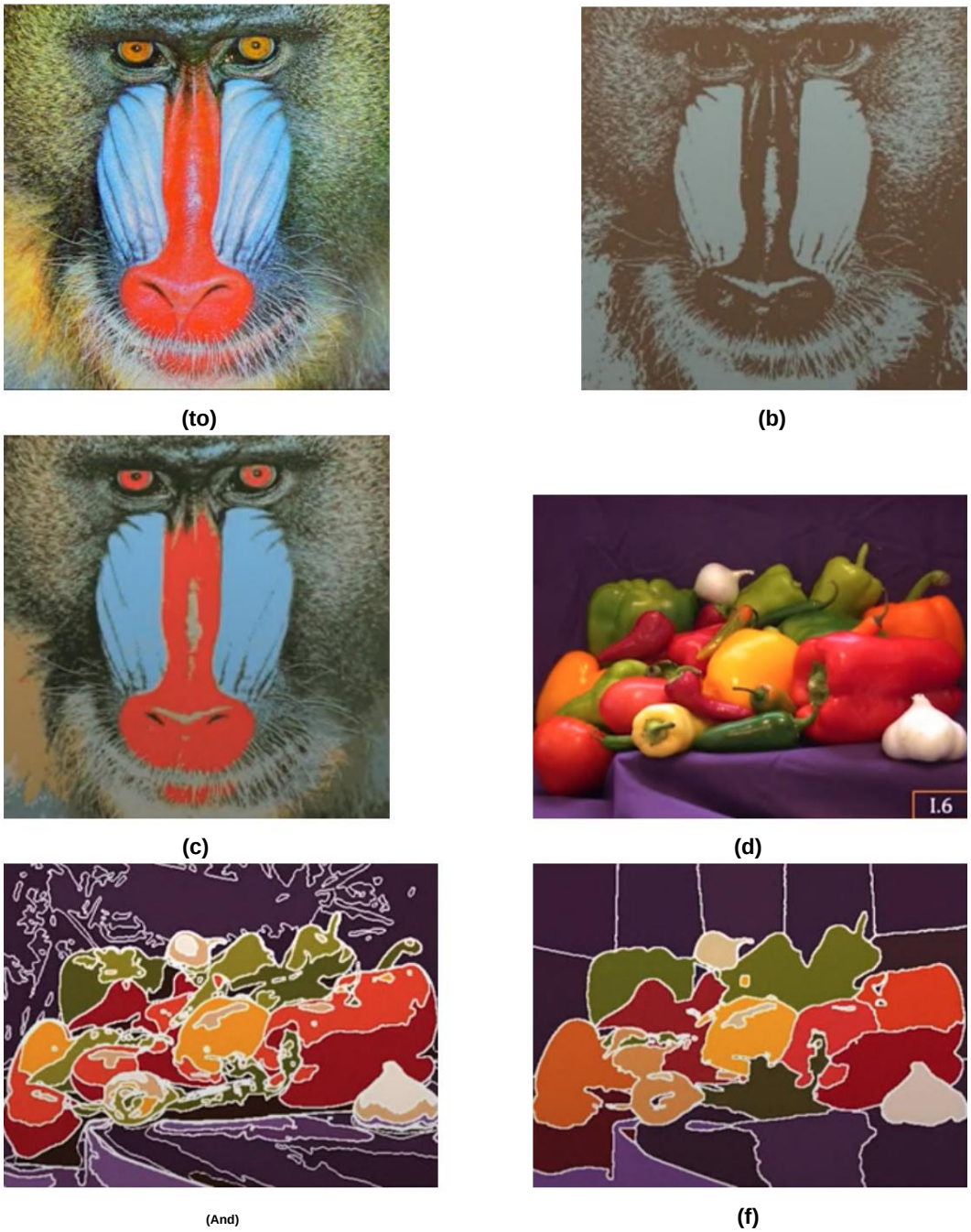


Figure 2.9. The four images show the result of the applied k-means algorithm to RGB images: (a) and (d) are the two original images; (b) show the result with $k = 2$; (c) with $k = 8$; (e) is the result with $k = 16$ and finally (f) is the result with $k = 16$ but in a five-dimensional space that is *(R, G, B, x,* y) where *x* and *y* represent the coordinates of the pixels in the image, while one was used for all the others space in three dimensions or *(R, G,* B).

sets of points that are attracted towards the same fashion. Figure 2.9 shows some examples of the mean shift application.

2.2.6 Supervised Algorithms

In general, as already mentioned, once the image has been mapped in a multidimensional feature space, the segmentation task can be traced back, in the event that the number of classes is known a priori, in a classic Machine Learning classification task. Consequently, all machine learning algorithms for classification become valid for solving the segmentation problem. There are several classification algorithms and the most popular are Support Vector Machines (SVM), Logistic Regressor, Decision Trees, K-nearest in ghbor and many others. The main difference with the algorithms mentioned in the previous paragraphs (2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5) is that the latter fall into the category of unsupervised algorithms, while all Machine Lear classifiers nings just mentioned fall within the supervised one, that is, they need a dataset in order to learn their parameters and be able to classify.

2.3 Deep Learning Approaches

In addition to the classical approaches, there are Deep Learning techniques that use complex architectures, to be able to learn the pattern of interest and in particular, to learn the features that determine the semantics of the pixels, which instead must be done manually when they use Machine Lear ning algorithms (phase of *feature extraction).*

2.3.1 Context-based architectures

This category of architectures uses a type of information that is very important and that the methods we talked about in the previous chapters do not use, that is the context of a pixel. In particular, the context is very important for the semantic classification of a pixel, as often the local features of a pixel such as color, brightness, but also other more complex ones, are not sufficient to distinguish two pixels of different classes. As already seen, in general in the field of Computer Vision the most used type of architecture is CNN, which is based on the presence of convolution layers, and it is thanks to these that architecture is able to extrapolate the context of a pixel. In particular, the category of context-based architectures makes a strong use of convolutions in different forms. One of the variants most used to capture a wider context of the pixel, without however going to increase the computational cost too much, is the dilated convolution. In particular, some, in order to build receptive fields with different amplitudes, make use of convolutions dilated at different scales. For example, the DilatedNet [47] makes use of five different dilation parameters, namely 1,2,4,8 and 16, obtaining a mIoU of 67.6% on the PASCAL VOC 2012 dataset. In [48] the local features of a pixel are merged with the global features of the image, thus adding to the local context of the pixel the global one, obtaining a mIoU of 69.8% on the PASCAL VOC 2012. In [15] instead, a module called *Pyramid Pooling Module* (PPM) is used, which uses a pooling layer at different scales and then merges the results of these pooling together, in order to have context information at different

verse scale (Figure 2.10). In particular, the idea is that with the sole use of the global context, there is a loss of information in the sub-regions, while using the PPM information is captured at different scales, including the global one. The PPM then inspired many other works that, by making changes, have improved performance even more, such as the DeepLabV2 [49], which starting from the idea of the PPM built the Atrous Spatial Pyramid Pooling (ASPP) module, essentially replacing the normal convolution and pooling with convolution dilated at different scales, or with different dilations, obtaining a mIoU of 79.7% on PASCAL VOC.

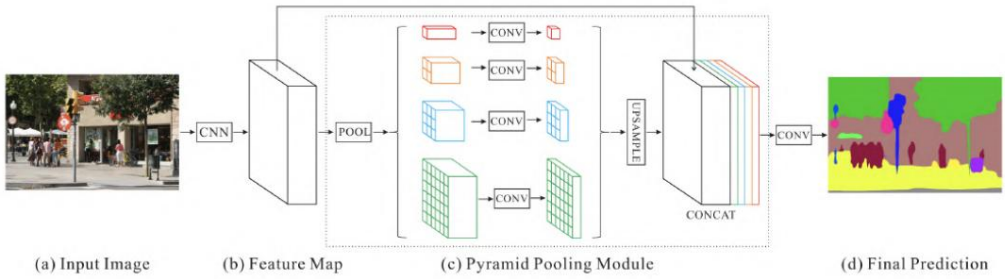


Figure 2.10. Illustration of the Pyramid Pooling Module (PPM) [15].

2.3.2 Feature-enhacement-based architectures

The general concept behind this category of architecture is that, in CNNs, the layer-by-layer extraction of increasingly high-level features causes a loss of spatial information. In particular, mainly due to pooling and convolutions with stride, the features progressively advance in the network, decrease in resolution and consequently lose spatial information, which in tasks such as classification are not important, but which are instead important in the task. segmentation. The solution proposed by this category of architectures to solve this problem is to combine the two aspects, that is to unite the features of the first levels, which have above all spatial information, with those of the deeper levels, which instead do not have much spatial information but contain information on the high-level features. One of the first architectures of this type was FCN [26], which puts this idea into action by adding *skip connections,* thus connecting the features of the intermediate layers with those of the last layers. UNet [8] is another architecture that refers to this concept and is one of the best known architectures in the field of semantic segmentation. In particular, the UNet, so called for its peculiar U shape, unlike the FCN uses skip connections between all the layers, that is, it connects all the features of the first part of the network, responsible for the feature extraction and called encoder, to those of the second part of the network, called the decoder (Figure 2.11).

2.3.3 Architectures based on deconvolution

The first architecture of this type was DeconvNet [50], which has an encoder-decoder structure. In particular, this network uses, apart from the classic poo-

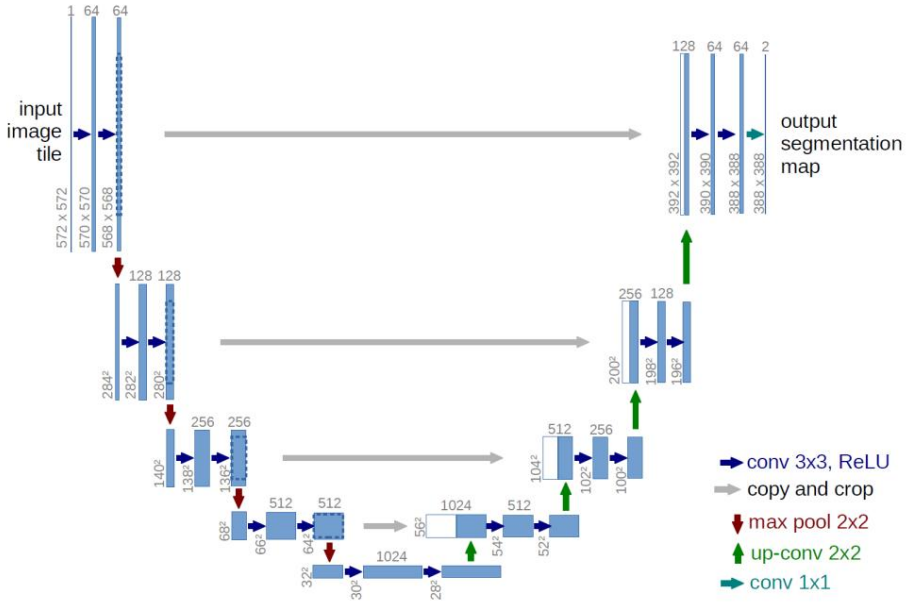


Figure 2.11. UNet architecture [8].

ling, convolution, and so on, two other types of modules, namely unpooling and deconvolution. The idea behind their use is to cope with the problem of the loss of precision in spatial information, in the part of the decoder where the feature maps are upsample to bring them back to their original resolution. In particular, in the encoder pooling layers, the index of the largest cell is stored, so that, in the upsample phase, the unpooling module can recreate the same window that was passed into the pooling layer. The output of unpooling, however, has cells cleared, this because in the corresponding pooling in the encoder only the largest cell is stored, but not the others. To solve this problem, the output of the unpooling module is passed to deconvolution which, thanks to the parameters learned, is able to reconstruct the feature map (Figure 2.12).

2.3.4 GAN-based architectures

This type of architecture is composed of two networks called "segmenter" and " *adversarial network".* The first has the responsibility, starting from the input image, to create a partition in non-intersecting sub-regions, that is the mask. The second, on the other hand, is responsible for distinguishing between the output of the first network and the correct mask. In particular, the two networks are pitted against each other: the segmentator has the goal of not making the second network understand the difference between its output and the real mask, while the second network (Figure 2.13) has the goal to understand the difference. The ANet [51] is one of the first networks of this type used for semantic segmentation and in particular, the main concept behind it is that the opposing network, unlike a classic loss function used for training a network , manages to capture higher level differences between the output of the segmentator and the mask, such as differences in the shape of objects or in their proportions. The first network, that

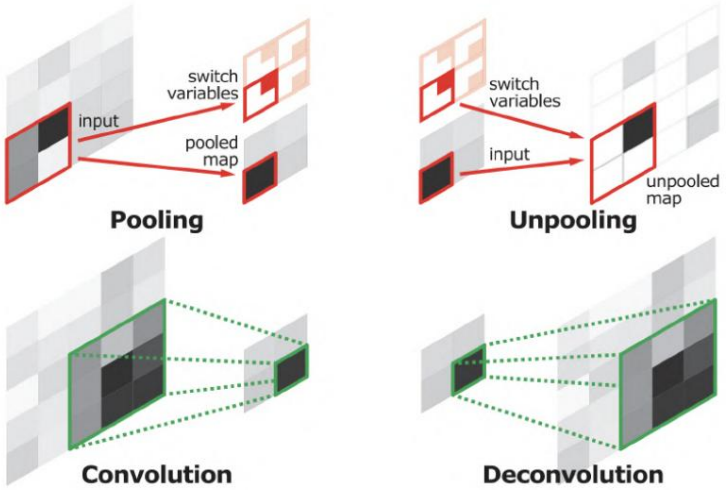


Figure 2.12. Illustration of the two modules of unpooling and deconvolution [50] and their difference with pooling and convolution, respectively.

responsible for the segmentation, uses a compound loss function for training, i.e. the final loss function is the result of the weighted sum of two further loss functions, a classic cross entropy and an adversarial loss function. In particular, the second loss function represents how much the opposing network is able to distinguish between its output and the correct mask.

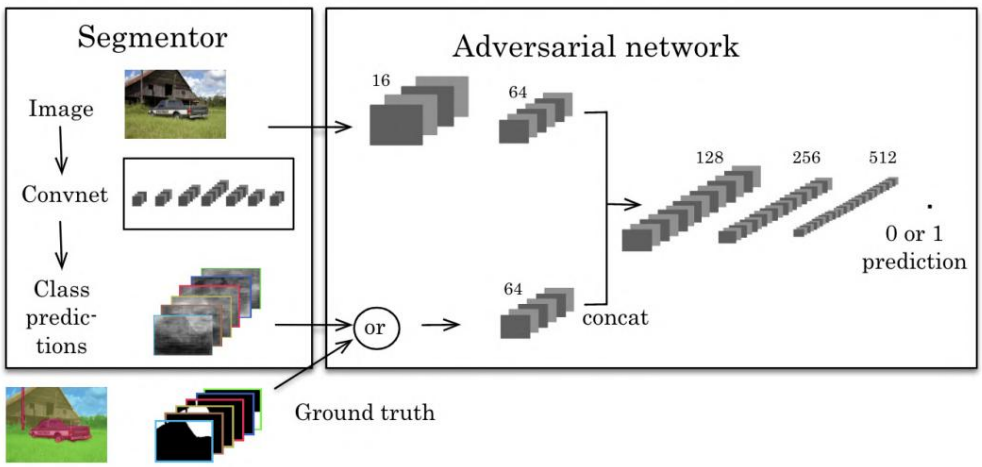


Figure 2.13. Overview of the GAN-based architecture proposed in [51]. On the left, the segmenter takes the RGB image as input, and produces the mask. On the right, the opposing network takes the mask produced by the segmenter and produces a label (1 = correct mask, or 0 = synthetic mask of the segmenter). The opposing network can optionally also take the RGB image as input.

Chapter 3

Deep Learning

Machine Learning (ML) is a branch of Artificial Intelligence (AI) that is concerned with building models that learn from the data it receives. Deep Learning, in turn, is a branch of ML and it is also concerned with building models that learn patterns from data. The peculiarity of the DL lies in the type of models used, or neural networks, also called Artifical Neural Networks (ANN) or Deep Neural Network (DNN). An ANN is a computation model inspired by the structure of the neural networks of the brain, and consists of a large number of basic computation components (neurons), which are linked together in a complex communication network through which it is able to approximate very complex functions. In fact, the use of neural networks becomes fundamental when the pattern we are trying to learn is very complex and above all non-linear.
Precisely this ability to learn complex patterns has made the DL a widely used tool which has brought clear improvements in many fields, including Natural Language Processing, speech recognition, the medical field, intelligent transport systems [52] and many others [53]. Figure 3.1 illustrates different fields of application of the DL. Another field in which the DL has been very successful has been that of image analysis. In particular, for some tasks such as classification and segmentation, neural networks have made clear improvements. In fact, starting from 2012, when AlexNet [54] (Figure 3.2) won the ImageNet challenge by beating the other competitors with an error rate of about 16%, against 26% of the previous year's winner, the neural networks have always dominated the challenges regarding Computer Vision tasks. Another peculiarity of the DL, which distinguishes it from the ML and which makes it so efficient in fields such as Computer Vision, is that the phase of *fea ture extraction* of the ML, in which the features are manually extracted from the data, is implicit in the neural network learning (Figure 3.3). This feature is fundamental as this phase is often very complex, both due to the difficulty of the extraction itself, especially for image data, and because often to know which features are to be extracted, a specific knowledge of the field of application is required. . In neural networks, on the other hand, it is precisely through the learning mechanism that the network understands which are the important features and which determine the nature of the data.

In addition to the advantages, neural networks also have several disadvantages. One of these is that in some respects neural networks are black-box models. In particular,

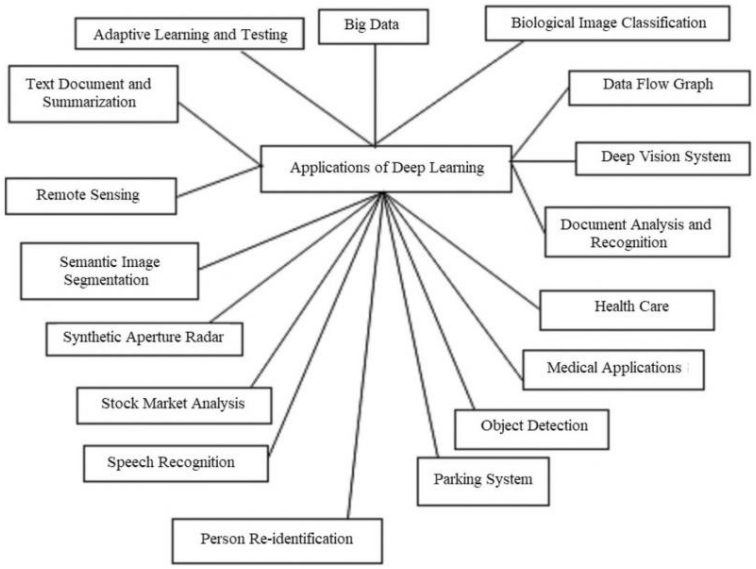


Figure 3.1. Some of the fields of application of Deep Learning [53].

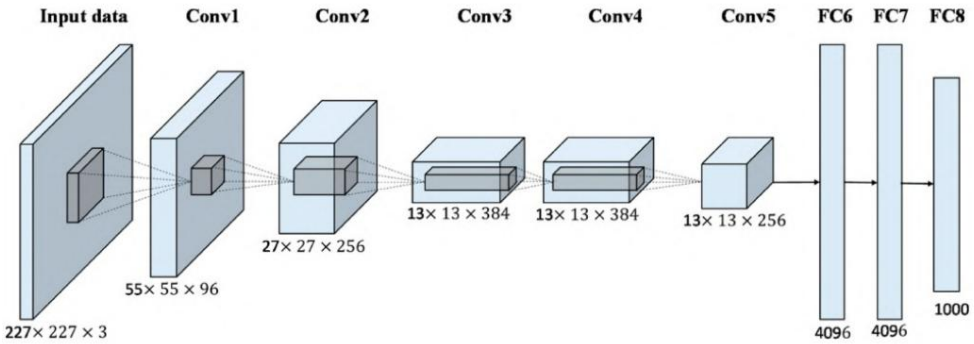


Figure 3.2. AlexNet network architecture.

their functioning is very complex and often, unlike ML algorithms, their behavior and their outputs become difficult to understand and consequently to explain. In fact, in recent years, a rapidly growing field of AI is Explainable AI [55], which deals with algorithms that provide transparency and interpretability to methodologies such as neural networks.

3.1 Perceptron

The perceptron is a mathematical model inspired by the biological neuron model [56] and constitutes the basic computational component of neural networks. While the biological neuron is made up of four main components (soma, dendrites, axon and synapse), the perceptron is made up of two parts (Figure 3.4): in the first

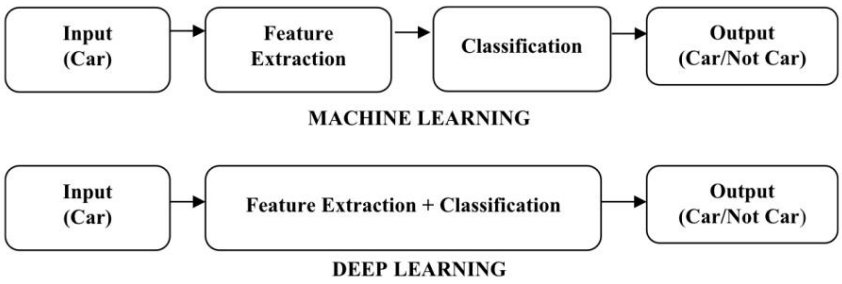


Figure 3.3. Difference between Machine Learning and Deep Learning [53].

part, the scalar product between the input vector *X* = (x1, x2, ..., *xn)* and the perceptor parameters *W* = (w1, w2, ..., *wn)* is calculated , while in the second part the result of the scalar product is given as input to the activation function g, which gives us a scalar. Below is the formula of the perceptor output:

$$y = g\left(\sum_{j=1}^{n} x_j w_j\right). \tag{3.1}$$

The activation function is a critical part of the perceptron, as it provides the network with non-linearity, a fundamental property for approximating complex non-linear functions. In particular, without activation functions, however complex and profound the neural networks may be, they would represent nothing more than a linear function of the input and would become a simple linear regressor. Later we will see in detail which are the most used activation functions in modern neural networks.

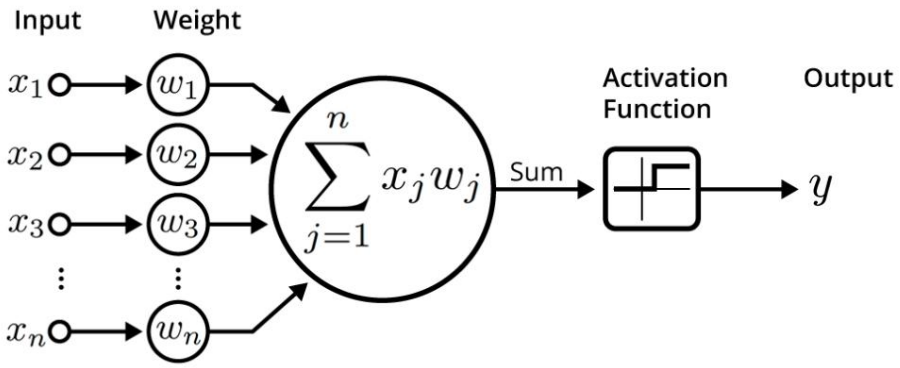


Figure 3.4. Illustration of the structure of a perceptor.

3.2 Multilayer Perceptron

One of the simplest types of neural networks is the Multilayer Perceptron (MLP). MLP is nothing more than a set of perceptrons

connected to each other and organized in layers (Figure 3.5), where the outputs of the perceptors of one layer are the inputs of the perceptors of the next layer. The layers, at least three, are divided into input layer, output layer and hidden layers. The input layer is the one that first receives the data to be processed, the output layer is the one that returns the final output of the network and the hidden layers are all the others that are in the middle.

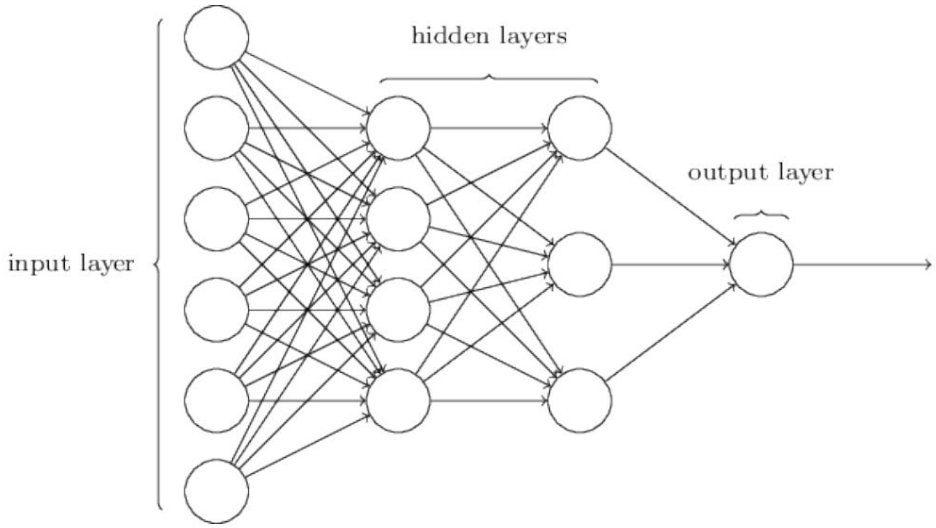


Figure 3.5. Generic architecture of a Multilayer Perceptron.

As previously mentioned, thanks to the use of the activation functions, the final output of the network is a complex and highly non-linear function of the input, complexity and non-linearity that increase with the increase in the size of the network. This property is fundamental and peculiar to neural networks, which are distinguished from other ML models precisely by the ability to represent very complex functions. In fact, neural networks have taken over especially in those tasks where the functions or patterns to be learned are very complex, such as in the field of image analysis.

3.3 Activation functions

As mentioned previously, activation functions represent a cornerstone of neural networks. In particular, the choice of activation functions is a fundamental choice and can have a great impact on the performance of the model.
Within a network, there are often different types of activation functions and in most cases, different types are used for different parts of the network. Typically, the hidden layers use the same typology, while the choice of the one of the output layer depends above all on the task and the type of output to be provided. Below, the most used activation functions present in the literature.

3.3.1 Sigmoid function

One of the best known is the sigmoid function (Figure 3.6), also called sigmoid, a mathematical function with the typical S-curve or sigmoid curve. In particular, the sigmoid does nothing but map the input (any real) on a range that goes from 0 to 1: the larger the input, the closer it will be to 1, while the smaller it is, the closer it will be to 0. Below is the formula:

$$\ddot{y}\,(x) = \frac{1}{1 + e\,\ddot{y}\,x}. \tag{3.2}$$

The sigmoid is often used in the last layer of the neural network, especially in binary classification tasks, where the final output of the network must be mapped in the interval [0,1], as it represents the probability that the input belongs to one of the two predetermined classes.

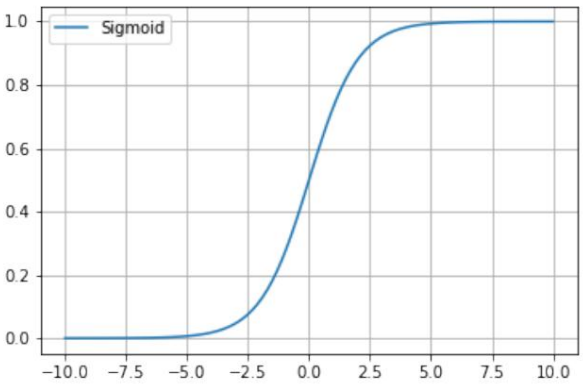


Figure 3.6. Graph of the sigmoid function.

3.3.2 Hyperbolic tangent

The hyperbolic tangent (Figure 3.7), often called “tanh”, is an activation function very similar to the sigmoid. The main difference is that it maps the input to the range [ÿ1, +1] instead of [0, +1].

3.3.3 Rectified Linear Activation Function

The Rectified Linear Unit (ReLU) (Figure 3.8) is perhaps the most commonly used activation function. The reason lies in the fact that it solves one of the main problems of the activation functions, namely the disappearance of the gradient. The latter relates to the fact that in the back-propagation mechanism (which we will address later), above all

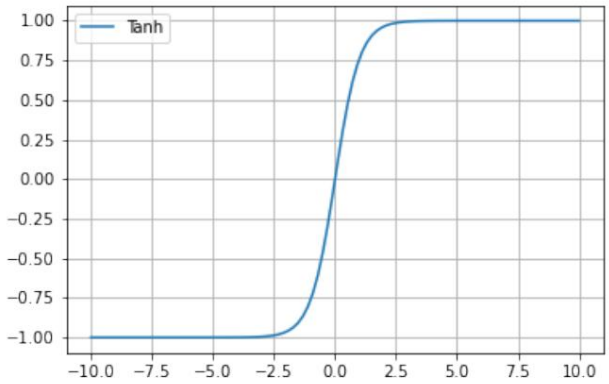


Figure 3.7. Hyperbolic Tangent Graph.

when you have many layers, the gradients that are multiplied, if they have values small, they tend to go towards 0 very quickly and this leads to having changes, especially in the first layers of the network (the last ones in the back propagation), very little if any. Here is the formula:

$$f(x) = x^{+} = \max(0, x). \tag{3.3}$$

Furthermore, the ReLU, given its formula, is also much easier to calculate compared to the others and this property becomes critical especially in neural networks, where the computational cost often becomes an obstacle. On the other hand, a disadvantage of the ReLU is that for negative values, the function has one null derivative. While this can also be a strong point (for the sparsity of the network), is a problem when many of the values input to the functions of activation are negative because many of the neurons are cleared. When this happens the network tends to "die" or tends to no longer update its parameters (hence the name of the Dying ReLU problem). To solve this problem, it was introduced a variant called Leaky ReLU or Parametric ReLU (Figure 3.9). In particular, the variant, unlike the original version, presents in the part negative a small slope, which is not a parameter learned during training, but is chosen previously.

3.3.4 Softmax

In multiclass classification problems, where the number of classes is greater than two, in the output layer of the network the sigmoid function cannot be used, that instead it is used in the context of binary classification. Similarly to the sigmoid, the softmax output can be interpreted as the probabilities associated with each class, in fact all the summed values return 1. When the number of classes is equal to two, the softmax is equivalent to the sigmoid and if you want you can consider the

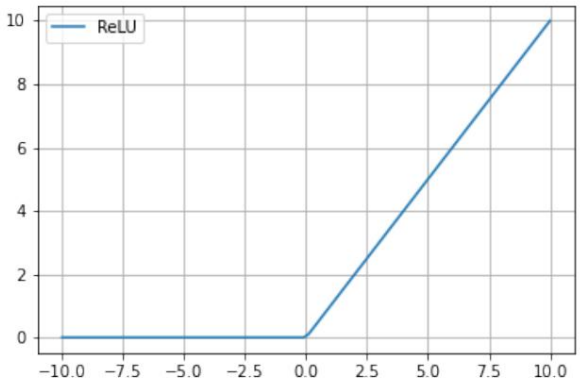


Figure 3.8. ReLU chart.

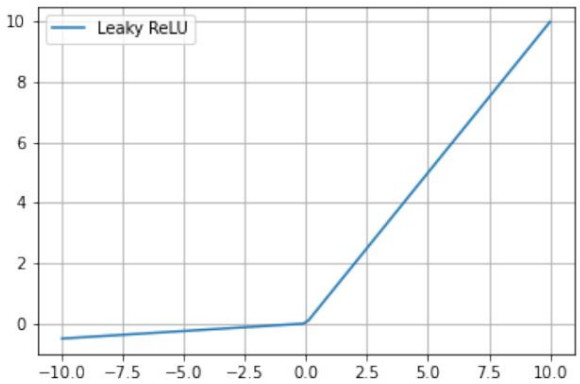


Figure 3.9. Leaky ReLU chart.

sigmoid a particular case of softmax or, equivalently, softmax a generalization of sigmoid. Here is the formula of softmax.

$$sof\ \text{tmax}\ (xi) = \frac{and^{xi}}{P._{xj\,j}\,and}. \tag{3.4}$$

3.4 Learning

The learning phase is a phase common to all ML algorithms. In general, in this phase the model being trained updates its parameters trying to improve its performance, that is, trying to approximate the pattern of interest in the best possible way.

3.4.1 Learning paradigms

In the field of ML, there are four main learning paradigms:

- *Supervised:* as previously mentioned, the ML approach differs from the traditional approach in the fact that whoever develops the model does not define the rules of operation of the world of interest, but rather takes care of building an architecture capable of learn them from the data. That said, the role of data is critical. In fact, especially in the field of DL where the patterns of the world of interest are very complex, it is essential to provide the model with sufficient, as well as representative, data. In this paradigm, the model is supplied with a set of data, called a *dataset,* consisting only of pairs of type X and Y, where the X represents the input and the Y represents the correct output. The pairs that make up the dataset are used by the model as an example of how the pattern it is trying to approximate works.

- *Unsupervised:* in this paradigm instead, in the supplied dataset there is only the input without the correct output. Clearly, this paradigm is only used when obtaining the corresponding labels is not possible or very difficult. An example of this is the field of medical images, where only specialists can produce labels. Another example is when their production is a very long and tedious process, which often happens in segmentation. When this happens and the model has no input-output examples to rely on, the most typical approach is to look in the elements of the similarity dataset, so that they can be grouped into what are called *clusters.*

- *Semi Supervised:* This paradigm is a meeting point between the supervised and unsupervised paradigm. In particular, in this case only a part of the dataset is associated with the corresponding label.

- *By reinforcement:* here the model, also called *agent,* learns by interacting with the external environment. In particular, the agent carries out actions and subsequently evaluates the results, using a numerical value of "reward", which has the objective of encouraging correct actions and discouraging incorrect ones.

3.4.2 Loss functions

In the learning phase, a critical aspect is how the model evaluates its performance. In particular, at each iteration of learning the model must have the ability to evaluate how far its output is from the correct one.
Consequently, the learning model needs a measure of this distance. This role is filled by the loss function, which is decided by the developers before the training phase. The choice of the loss function is fundamental and can have a great impact on the performance of the model. A key feature of the loss function is differentiability. In particular, it is fundamental as its derivative is used precisely to train the

network (we will go into detail on the learning mechanism later). There are many types of loss functions that are chosen based, above all, on the task:

- in regression problems, that is where the output is a value within a continuous interval, one of the most common loss functions is the *MSE (Mean Squared Error),* which calculates the average of the squared differences between the elements of the prediction vector *pred* and the elements of the corrected output vector y.

$$MSE = \frac{1}{n} \sum_{y=1}^{n} (pre - yi)^2. \tag{3.5}$$

- in binary classification problems, one of the most used is *Binary Cross Entropy.*

$$ECB = -\frac{1}{n} \sum_{i=1}^{n} (yi \log (pre) + (1 - yi) \log (1 - pre)). \tag{3.6}$$

- in multiclass classification problems, on the other hand, the Cross Entropy

$$CE = -\frac{1}{n} \sum_{i=1}^{n} yi \log (pre). \tag{3.7}$$

3.4.3 Back propagation of the error

In neural networks, as in almost all ML algorithms, the learning phase is an iterative phase. At each iteration, the network, with its current parameters, processes one or more elements of the dataset, then comparing the output with the correct label through the loss function. At this point, the network updates its parameters trying to improve. To understand what the output layer should look like, we have some reference values, while for hidden layers we have no direct indication. Parameter updating, generalized for both the output and hidden layers, is described by *the delta rule:*

$$ÿwjk = ÿÿÿjok. \tag{3.8}$$

Where *ÿwjk* is the variation of the parameters between neuron *k* and j, *ÿ* is the learning rate, *ok* is the output dl neuron k, i.e. the input of neuron *j* and *ÿj* is the error of the output of neuron j. The difference between the output layer and the hidden layer lies in the value of *ÿj ,* i.e. for the output layer it is the following

$$ÿj = yj (1 - yj) (oj - yj). \tag{3.9}$$

Where *oj* is the output of neuron j, while *yj* is the correct output. On the other side, for hidden layers it is equivalent to

$$ÿj = yj (1 - yj) \text{ X } \sum_{k} ÿkwkj . \tag{3.10}$$

Where instead P *k ÿkwjk* is the weighted sum of the errors of all neurons connected to j. Finally, all neurons update their parameters using the following formula:

$$wjk = wjk + ÿwkj \ . \tag{3.11}$$

3.5 Regularization

In general, in the field of Machine Learning, as well as in the field of optimization of functions, what we try to do is to minimize a function, or in other words, to minimize an error. The main difference between the two fields is that, in ML, in addition to the error on the training set *(training error),* the *test error,* also called "generalization error" , is also taken into consideration . In particular, at the end of the training phase, it is important that the built model has good performance not only on training data (training set), but also and above all on data never seen before (test set) and this skill is called generalization. . The ability of a model to generalize is very important and can depend on many factors. In particular, in addition to the right method and amount of training, the right hyperparameters and many other factors, generalization can above all depend on the choice of the complexity of the model. In general, as far as complexity is concerned, the model can be found in three situations: in the first the model did not approximate well the pattern concerned and its complexity is too low *(underfitting),* in the second the model was able to grasp the true pattern and finally, in the third situation, the constructed model has raised its complexity too much, approximating a more complex pattern but which includes the one concerned (overfitting) (Figure 3.10).

In the case of the first situation there are several solutions, but in general the design choices of the model must be reviewed and in any case its complexity must be increased. In the case of overfitting, on the other hand, a widely used method is that of regulation, whose objective is precisely to bring the model from the third to the second situation. In general, we can define regularization as any modification we make to our model or to the learning algorithm, with the aim of reducing its generalization error but not the training one. Often in practice, especially in the fields of application of Deep Learning, even with a very complex model, it is not necessarily possible to include the pattern concerned. Particularly in the world of DL, models are applied to extremely complex domains and, consequently, the true pattern is almost certainly outside that of the model. This implies that in most cases, the best choice is that of large models with high complexity, accompanied by the right regularization techniques [57].

There are several regularization techniques, but one of the most used general methods is to limit the complexity of the model, also called capacity, by adding a penalty component *ÿ (W) to the loss function,* transforming the loss function, or also called objective function, *J (W; X,* y) in J˜ (W; *X,* y)

where is it:

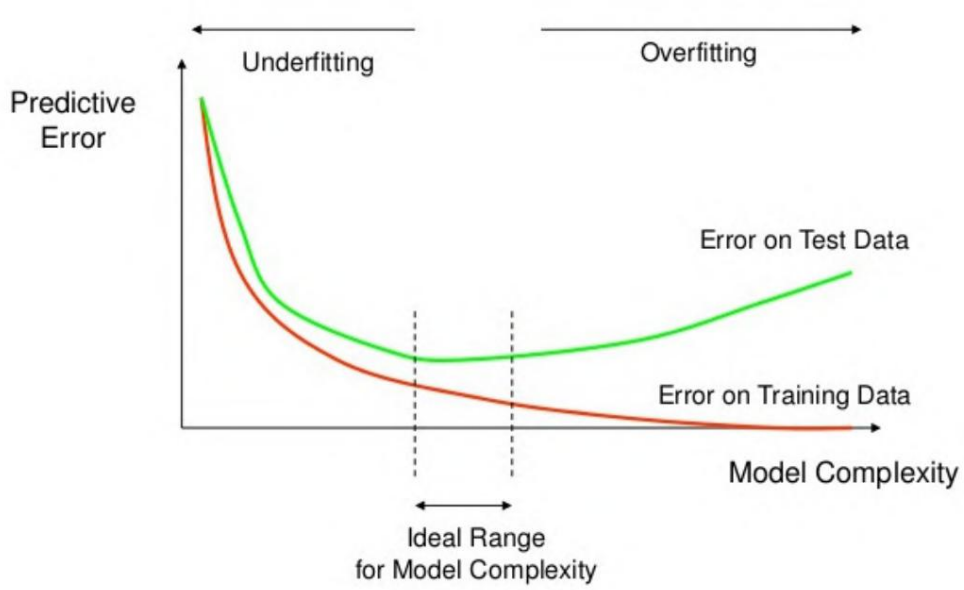


Figure 3.10. Graph of the errors on the training set and on the test set as a function of complexity of the model and illustration of the two states of underfitting and overfitting.

$$J\tilde{}(W; X, y) = J(W; X, y) + \ddot{y}\ddot{y}(W). \quad (3.12)$$

Where $\ddot{y}$ ÿ [0, ÿ] is a hyperparameter representing the weight of the contribution of penalty, *W* are the model parameters, *X* is the training data and *y* is the correct out put. Specifically, this component represents the intention of wanting penalize the complexity of the model, consequently, with its addition during the learning phase, the algorithm does not only try to minimize the training error, but also the complexity of the model. In detail, there are several methods of this kind and in each $\ddot{y}(W)$ takes a different form. Two of the more methods known and used in this category are the L1 regularization and the regularization L2. Another regularization method widely used in neural networks is *dropout.* The operation of this technique is very simple and intuitive: during training, with a certain probability p, some nodes of the network are deactivated, thus effectively transforming the network into a less complex one and reducing its own complexity (Figure 3.11). In detail, as well as the other regulation techniques mentioned above, the dropout tends to keep the parameters of the model more as low as possible and it does so by avoiding making one knot prevail over the others. In particular, by randomly deactivating the various nodes in turn, it forces the network not to unbalance their parameters towards a node, obtaining more balanced and consequently smaller parameters, while without the dropout, the risk is that during training the network can greatly increase the parameters of one node and reset those of others.

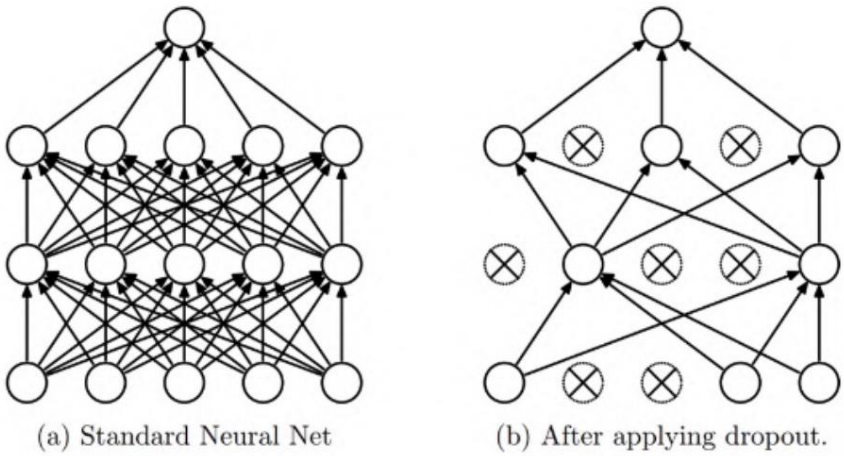


Figure 3.11. Illustration of the dropout technique.

3.5.1 Data augmentation

Apart from regularization techniques, certainly the best method to increase the generalization capacity of a model is to provide it with more input data. Unfortunately, however, as already mentioned, in practice the data are often very difficult to find, especially in some fields of application of the DL. Often, to overcome this difficulty, a family of regularization techniques called *data augmentation is used.* In general, for data augmentation we mean any technique with which fake data are produced, starting from the real ones. In particular, starting from one or more elements of the dataset and applying a transformation of some kind, it is possible to produce an artificial, but nevertheless realistic, datum.

This technique is often used in the field of Computer Vision, where the transformations applied to the data are for example rotations, translations, modification of contrast or brightness and many others (Figure 3.12). In this field, for some tasks such as classification, the data augmentation technique is often very simple, since, given that the classifiers must be invariant with respect to many data transformations [57], the transformations made during the data augmentation must be applied only to images and not to labels, which instead must remain the same, which is not true, however, in the field of semantic segmentation, where they should also be applied equally to masks.

Another widely used type of data augmentation is *noise injection,* that is the technique in which a certain noise is randomly applied to the data of the dataset. This technique is often very important, since the invariance of the models with respect to the noise in the input data is a very important and desired characteristic, and its lack can lead to negative results [58], also from the point of view of Explainable AI . In particular, often the images with the addition of a certain type of noise appear to human eyes identical to the original ones, consequently the variation of the output of the model from one image to another can be a problem for the interpretability of the template.

As previously mentioned, data augmentation is easier for some tasks, while less so for others. Furthermore, recent research [59] has shown

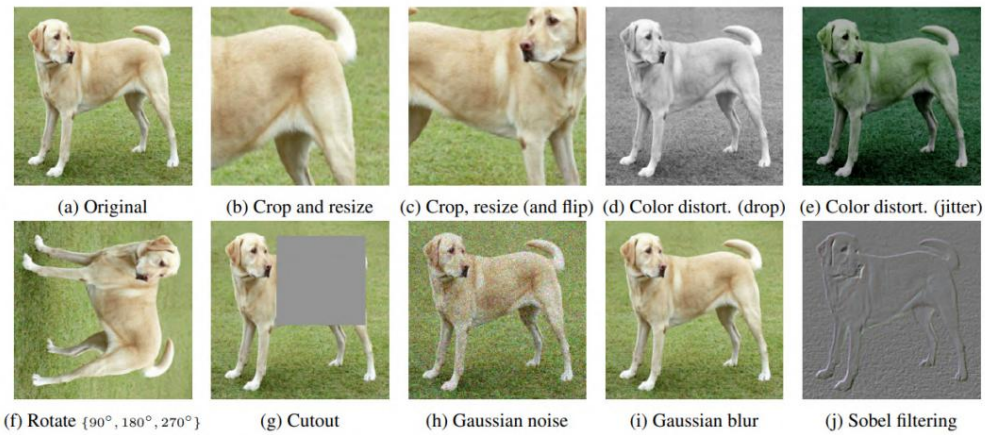


Figure 3.12. Some examples of possible transformations to apply data augmentation to an image dataset.

that this technique can, on the one hand, lead to great improvements in the model's ability to generalize, and on the other hand penalize some classes. In particular, its effects are *class-dependent,* i.e. they depend on the nature of the dataset classes, consequently the use of these techniques can lead to improvements in some classes, but also to a decrease in performance in others, especially when they are applied. the same transformations to the entire dataset.

3.6 Convolutional Neural Networks

Convolutional networks [60], such as the multilayer perceptron, have an input layer, an output layer and several hidden layers. The difference between classical and convolutional networks lies precisely in the presence, among the hidden ones, of convolutional layers (Figure 3.13). In particular, convolutions are matrices whose size is decided in advance, which represent filters that are applied to the image by scrolling on it to extract certain patterns. One of the main advantages is that by scrolling on the image, the convolution is able to recognize the pattern it has learned during training regardless of the position in the image, unlike normal neural networks where neurons, especially the former, are associated with specific portions of the image.

The convolutions applied to images, also called *kernels,* come from the world of Computer Vision, where they are used for image processing. The difference between traditional convolutions and those used in neural networks, lies in the fact that while the values of traditional ones are chosen in advance and designed to apply a certain filter, those in networks start with randomly initialized values, and it is precisely with the training that the network learns which are the patterns to highlight. Within a convolutional network, usually, there are more convolutional layers and going forward in the network, the input image decreases in the first two dimensions (height and width) and increases in the third dimension, that is the number of features, often called channels. In particular, each convolution applied to an image produces one feature and more

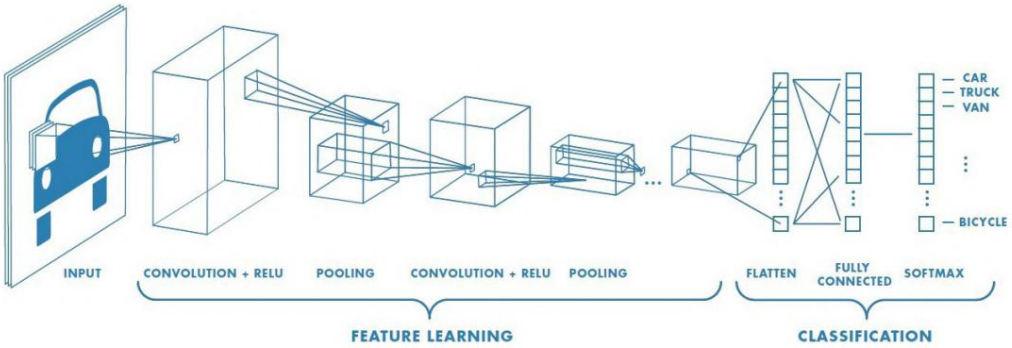


Figure 3.13. Architecture of a generic convolutional network.

you go towards the output layer, the more the features represent higher level patterns. In fact, in the first layers the features represent more elementary patterns, eg example geometric shapes such as points or lines. In the deeper layers, on the other hand, they represent more complex geometric structures, or structures with a meaning semantic. Figure 3.14 intuitively illustrates the concept just explained.

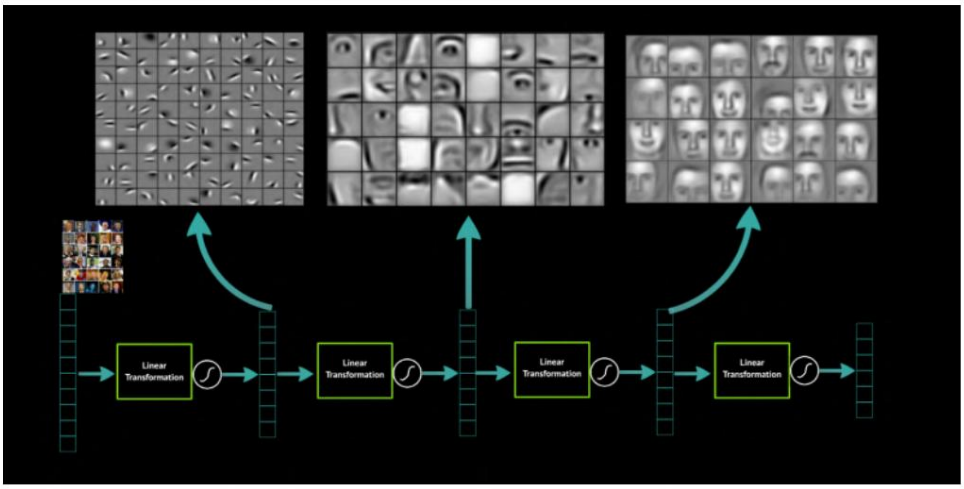


Figure 3.14. Illustration of the concept of low and high level features.

In the network construction phase or later in the hyperparameter optimization phase, the values of three
hyperparameters:

- *kernel size:* the size of the convolution matrix.
- *stride:* indicates the number of pixels with which the window moves at each operation.
- *padding:* denotes the process of adding zeros to each side of the input e
  this hyperparameter indicates how many to add. In particular, the padding
  has the purpose of being able to pass the kernel even on the pixels closest to the edges, and the
  zeros added are just to fill the part of the kernel that comes out of the input
  (Figure 3.15).

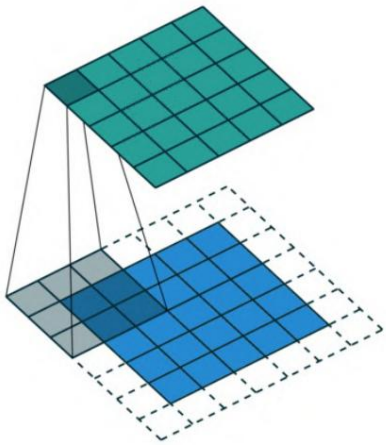


Figure 3.15. Padding illustration.

In addition to convolutional layers, there are usually four other types of layers in a convolutional network:

- *pooling layer:* this layer has the role of reducing the first two dimensions of the data that crosses the network and it too is a sort of window that slides over the image, giving a scalar output. There are different types, but in general, for an input window it returns a statistical value of the latter. One of the most used types is the *max pool,* which returns the maximum value within the window; another widely used is *the average pool,* which instead returns the average of all the values in the window. Unlike convolution, the pool layer has no learnable parameters and its only parameter is the size of the window, which increases the more it reduces the height and width of the data. The importance of the pooling layer lies in the fact that it gives the next layer invariance with respect to small translations of the input, i.e. by slightly modifying the input of the pooling layer, most of its output remains unchanged. This invariance is fundamental, especially in some tasks such as classification, where it is not so important where the feature is, but rather its presence.

- *activation layer:* this is a type of layer in common with all types of neural network. In fact, as already said, this layer has the role of providing non-linearity to the network, which is fundamental for approximating complex patterns.
  In the case of convolutional networks, the most common activation function is the ReLU and often follows the convolution layers.

- *completely connected layer:* this type of layer, also called dense layer, is in effect structured as a multilayer perceptron and is always located at the end of the network. This part of the network is the responsible part of the classification, that is, the high-level features produced by the first layers are passed in input to the neurons of the completely connected layer, so as to produce the final output.

- *final activation layer:* this layer represents the last layer of the network and has the role of translating the output into the desired range. The two most used types are *sigmoid,* in the case of binary classification, and softmax, in the case of multiclass classification.

3.6.1 Advanced convolutional neural networks

VGG

The work [61], which proposed in 2014 the architecture called VGG (Visual Geome try Group, or the name of the research group of the work) investigated the role of the depth of architectures in the performance of convolutional networks. This study resulted in one of the best known architectures in the field of convolutional networks and Computer Vision. As regards the general structure of the architecture (Figure 3.16), which however has different versions depending on the depth, it is made up of a first layer, that is the one that takes the image as input, which has a fixed size of 224x224, consequently with this version of the network, all the images given in input must have that size and this aspect is due to the fact that at the end of the network there are dense layers, also called *fully connected* (FC), which do not allow the network to have different sized inputs.

After that, this first layer is followed by a stack of 3x3 convolution layers, together, clearly, with the activation function (ReLU) layers, with stride 1 and padding 1 (in order to preserve the spatial resolution after convolution).
In some layers of the network there are pooling layers (max pooling), in particular, there are 5 in total and they have a 2x2 window with stride 2. Finally, the last part of the network is composed of three dense layers, of which the first and second have a size of 4096 channels, while the last one has 1000, as it was built to be tested on ImageNet which has 1000 classes and a last layer of Softmax.

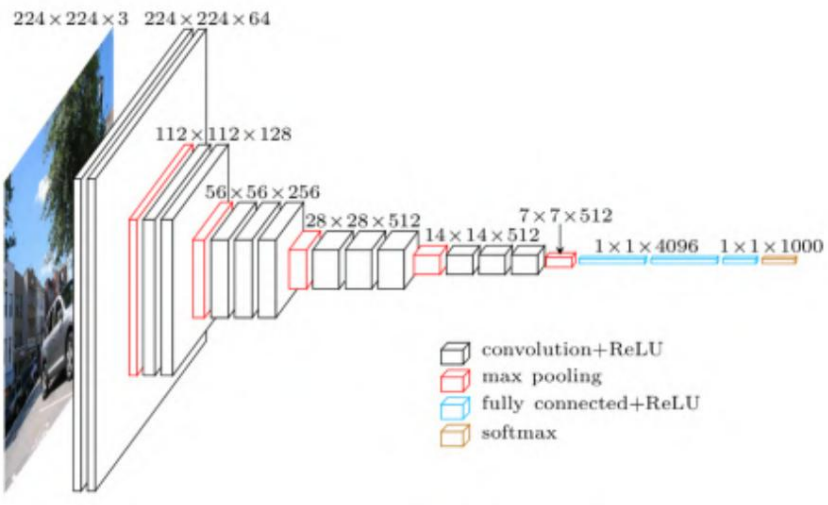


Figure 3.16. Generic architecture of the VGG.

As mentioned previously, the architecture just described takes different forms depending on the depth of the convolution stack. In particular, the

different versions range from a minimum of 11 layers (8 layers of convolutions and 3 dense layers) up to a maximum of 19 (16 layers of convolutions and 3 dense layers), while the width of the network (number of channels) remains consistent in all versions: starting from 64 up to the last convolution layers at 512. The main difference between VGG and previous architectures is above all the use of small convolutions. In particular, previously the tendency was to increase the receptive field with ever larger convolutions, such as in the AlexNet [54], which uses 11x11 convolutions with stride 4, or in the architecture proposed in [62] and in the OverFeat of [63], which use 7x7 convolutions with stride 2. The advantages that the authors of [61] highlight regarding the use of 3x3 convolutions, are mainly due to the fact that the same receptive field produced by a 7x7 convolution is reproducible with a stack of three 3x3 convolutions, with the advantage of incorporating three activation layers, which makes the total function more discriminative, and the fact that in this way the number of parameters of the network are reduced. In particular, a stack of three convolutions has 3 (32C 2) = 27C while a 7x7 convolution has 7 $2C$ 2 parameters.

^{2} parameters, where C is the number of channels of the input,

DenseNet

Another of the best known architectures is DenseNet [64], a convolutional network model inspired by the *feed-forward model.* In particular, the peculiarity of DenseNet is that, unlike the classic convolutional networks within which with L layers there are L connections, i.e. one between each layer and its next, this ar L (L + 1) connections, i.e. one between each layer and all layers architecture presents $\frac{}{2}$
later (Figure 3.17).

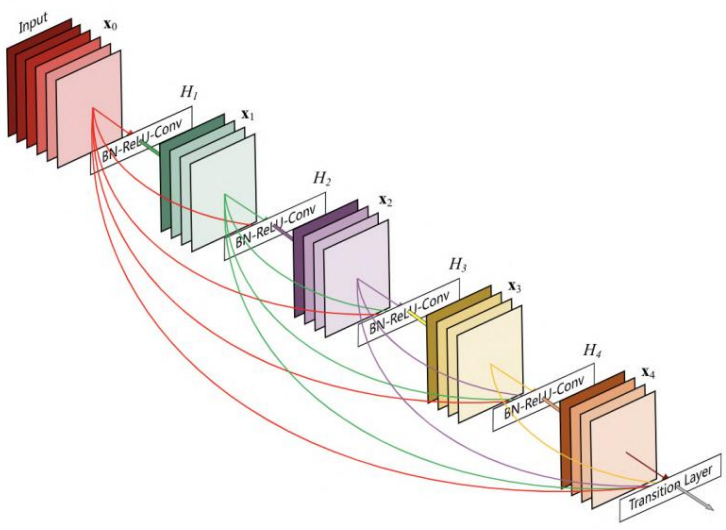


Figure 3.17. Structure of a DenseNet block: each layer takes its outputs as input of all previous strata [64].

The advantages highlighted by the authors of [64] are above all regarding one of the major problems in the field of neural networks, namely the *vanishing gradients .* As already mentioned, this problem concerns the fact that in the error back-propagation mechanism, when there is a very deep network, there is a risk that the information on the gradient, which travels from the last layers of the network to the first ones, vanishes passing through the layers and don't get to the first ones. In reality, the same problem can be seen from another perspective, that is, in very deep networks it is not only the information on the gradient that is in danger of disappearing, but also the information on the input that travels from the first to the last layers. Over the years, up to the publication of DenseNet, in other works [65, 66, 67, 68] this problem was faced with different techniques, which however all had one thing in common, namely the use of short connections to connect neighboring layers. The work done by the authors, in fact, has tried to distil and generalize these approaches proposed previously, creating a simple connection scheme with the aim of maximizing the flow of information between the layers of the network. Moreover, unlike [65], DenseNet, to minimize the loss of information coming from previous layers, does not use the sum as an operation of merging different inputs, but rather the concatenation.

Inception and GoogLeNet module

The work [69], following the trend of those years of increasing the complexity of convolutional networks more and more to increase performance, had the goal of finding a method to increase the complexity of the networks, without however increasing its cost. computational. In particular, the idea behind their work was to create a module, called the "Inception" module, within which all types of components were used simultaneously (convolutions with different dimensions, pooling, ...) , which were usually alternated layer by layer. The intuition of this mechanism is that the choice of the operation to be used for each layer is no longer of who builds the model, but of the model itself, which at each layer is no longer limited to the information of the single operation performed in that layer, but it has available information resulting from different operations performed on the same input. In particular, the authors' choice on which operations to do in parallel in the Inception module fell on: 1x1 convolution, 3x3 convolution, 5x5 convolution and finally max pooling. This choice is the result of experiments made on different combinations, which as a result have judged this as the best. In reality, however, several variants and new versions of the Inception module have been proposed in the following, which have used different combinations [70, 71, 72, 73]. Going into the details of its operation, the Inception module is based on giving the same input parallel to the four different operations, the results of which are then concatenated and passed to the next layer (Figure 3.18). To concate the four different outputs, clearly their first two dimensions (height and width) must coincide, to do this the three convolution operations have specific stride and padding parameters. As for maxpooling, on the other hand, which unlike convolution necessarily reduces the height and width of the input, a certain amount of padding is required to make its output consistent with the others.

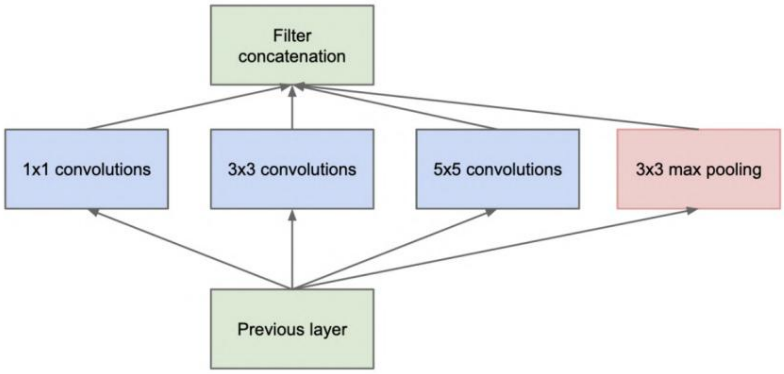


Figure 3.18. Illustration of the Inception module [69], the version without reduction of dimensions with very high computational cost.

The problem with this version of the module is that, by greatly increasing the width of the layers of the network without any kind of particular device, even the its computational cost has increased. To cope with this increase, the authors have proposed an approach that relies on using 1x1 convolutions to reduce the size of the input. In particular, for computationally multiple operations expensive, namely the 3x3 convolution and the 5x5 convolution, add one first 1x1 convolution, called in this case *bottleneck,* which aims to bring the input to a dimension for which applying the convolutions has a computational cost less high. Also, as far as maxpooling is concerned, the 1x1 convolution comes added later, since in this case it is not so much the computational cost of the operation that creates problems, but rather the size of its output (which has same number of channels as the input), the max pooling is followed by a convolution 1x1, in order to reduce the number of channels of its output (Figure 3.19).

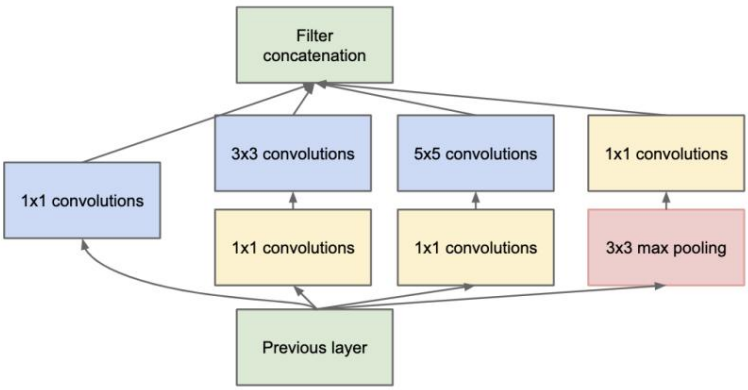


Figure 3.19. Illustration of the Inception module [69], the version with the reduction of dimensions through layers of 1x1 convolutions.

In addition to the conception of the Inception module, the authors of [69] have also proposed an entire architecture based on this type of module, the "GoogLeNet", whose name pays homage to LeNet [74], one of the first neural networks developed convolutional. The GoogleNet (Figure 3.20), in particular, is essentially made up of a stack of Inception modules. One of its peculiarities, beyond the use of the Inception module, is the presence of two further intermediate output layers. In particular, the network presents in two of the Inception modules additional output branches, which end with a layer of Softmax, transforming the output of the two intermediate layers into actual outputs of the network. To make them two effective outputs of the network, it is the fact that the total loss function, which the network uses to train, is calculated with a weighted sum of all three outputs (these two plus the final one), giving more weight to the final one. . By doing so, the loss function forces the network to have the feature maps of those two intermediate layers, already of a good quality to make a prediction and the idea behind this mechanism is that a regularization effect is obtained, pushing the network not to fall into overfitting.

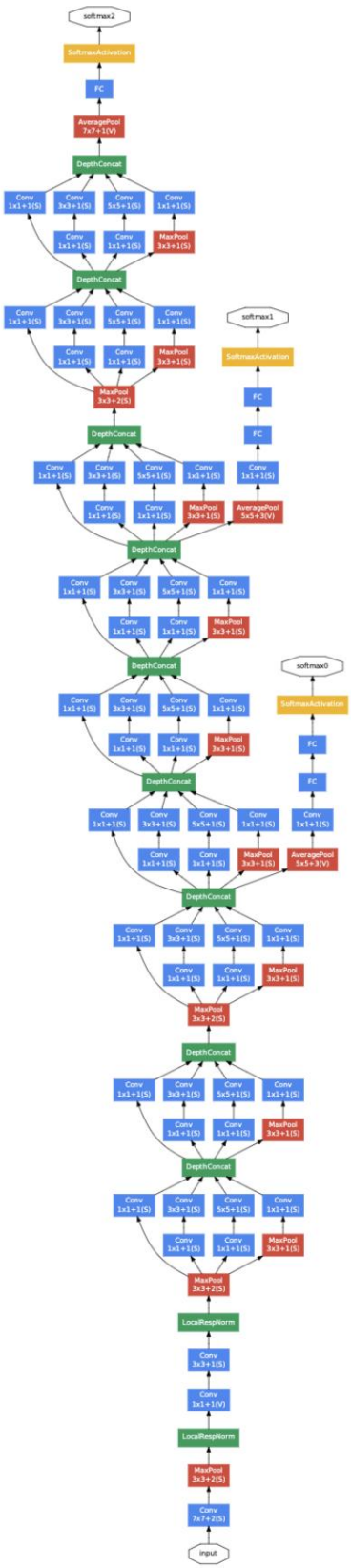


Figure 3.20. GoogLeNet architecture [69].

Chapter 4

Architecture and work methods

This chapter illustrates all the methods used during this work to deal with the semantic segmentation of the FloodNet dataset images. In particular, the whole work was based on trying to solve the main difficulties found in dealing with this dataset, which will be detailed in Paragraph 4.1.
As for the architecture used, the basic idea was to use a Deep Learning architecture, i.e. a neural network. The main reason for this choice can be found in the fact that, as already highlighted above all in paragraph 3, neural networks are able to approximate much more complex patterns than more traditional methods. In fact, the most current works, which deal with tasks similar to that of this work 1.2, mostly use Deep Learning methods. Specifically, the choice of the DeepLabV3 architecture [36] was based instead on overcoming two of the main problems addressed in the dataset (Paragraph 4.1).

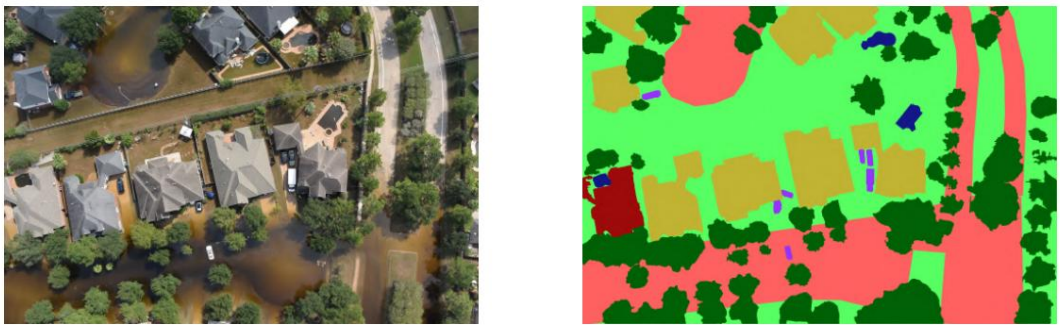
4.1 Difficulties addressed in the dataset

As often happens, the main difficulties faced during this work were inherent to the dataset used. In particular, this aspect is quite common, ie a large portion of the time spent in this type of work often consists in analyzing the characteristics of the dataset to understand its peculiarities, but above all its difficulties. Regarding the specific difficulties faced in the FloodNet dataset, the following are the four main ones:

- the nine classes present within the dataset are strongly unbalanced. In particular, some classes, such as "meadow" and "tree", are much more present in the dataset than other classes, ie the number of images that contain them is much higher than that of the others.
- in addition to the imbalance from the point of view of occurrences, the classes are also strongly unbalanced from the point of view of the scale. In particular, some classes, which are the same as the problem mentioned above, are represented by much larger regions than others. For example, the "vehicle" class, as well as the "swimming pool" class, is represented by much smaller objects than those of the "lawn" or "tree" class and consequently, adding the

problem mentioned in the previous point, the number of pixels of these classes within the dataset is considerably lower than that of others.

- some classes, more than others, present greater intrinsic difficulty. In particular, the two classes that were more difficult to learn were "flooded street" and "flooded building". The motivation, besides being present to a lesser extent within the dataset, is above all their semantic nature. As for the "flooded building" class, its semantics derive entirely from its context and not at all from its local characteristics.
  In particular, if we take only the pixels of a building, a flooded one is indistinguishable from a non-flooded one, as being flooded or not derives from the presence of flood water around the building. As for the "flooded road" class, its semantics can derive both from its context and from its local characteristics, but in some cases large portions of a flooded road can be indistinguishable from a non-flooded one.
  Specifically, a road to be considered flooded does not necessarily have to have water in all its parts, but only one flooded portion is sufficient (Figure 4.1).
  Furthermore, a further difficulty present in both the two classes mentioned above derives from the fact that their definition is more vague than other classes, lending itself more to interpretations, characteristics that, in the dataset masks, are presented in the form of ambiguity.



(to) **(b)**

Figure 4.1. The two figures show an example of how a flooded road can have large portions without water (in the upper right part), making it indistinguishable from a flooded road if we do not consider its context.

- the dataset inside has a large amount of noise, or in some masks there are errors. Furthermore, most of these errors are just in images that present an abundance of those classes mentioned in the previous point ("flooded building" and "flooded street"). Consequently, given the problems mentioned in the previous points, the difficulty in learning them increases even more. Specifically, most of the errors found in the dataset masks can be divided into three categories:

- the first type of error is represented by masks within which some pixels are incorrectly classified. In particular, the two most frequent cases are those in which the "flooded road" class or the "meadow" class are classified as "water" (Figure 4.3).

(to) **(b)**

Figure 4.2. The figure shows an example of an error in the dataset. In particular, as you can see, the pixels that should be classified as "lawn" are instead classified as "water".

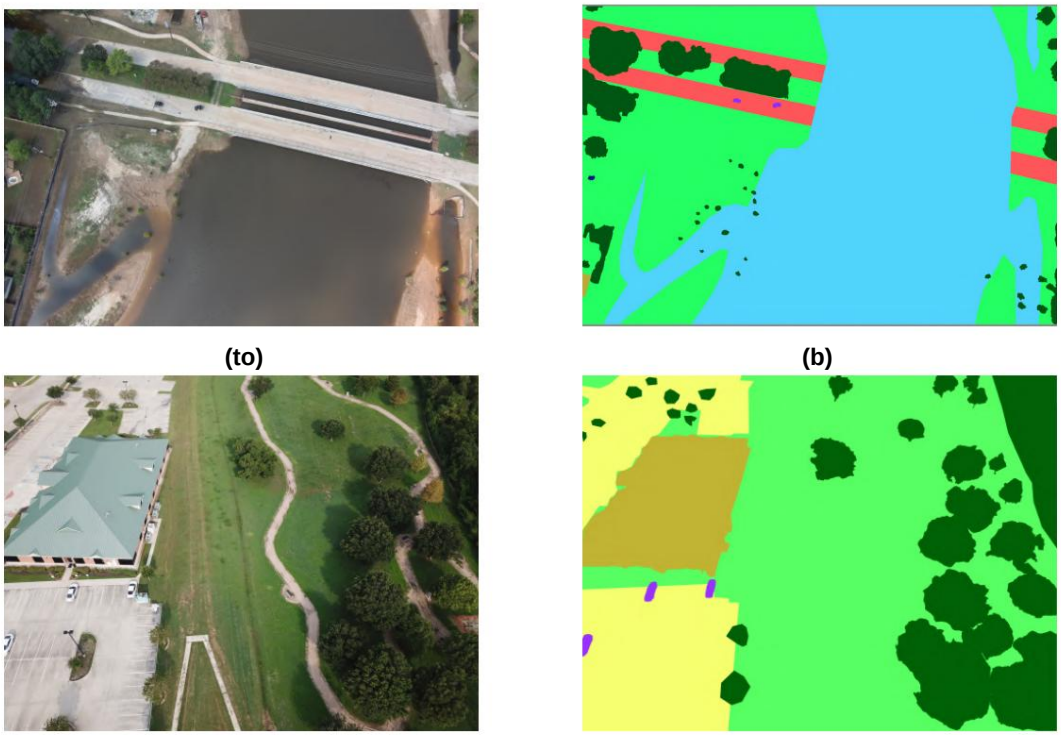
- the second category, on the other hand, concerns those cases in which some occurrences of classes or parts of them are not represented in the mask, but the corresponding pixels are instead classified as belonging to other classes (Figure 4.3). The most frequent cases concern the "road" and "flooded road" classes.

- the third case concerns the presence of portions of the image strongly inconsistent with each other, as well as with a certain degree of "confusion" between the classes. Figure 4.4 illustrates an example.

4.2 Cleaning the dataset and Data Augmentation

To cope with two of the four main difficulties mentioned in the previous paragraph, two main phases of cleaning the dataset *(data cleaning)* and offline data augmentation were implemented. In particular, these two phases were found to be fundamental for providing a dataset composed of elevated images both from a qualitative and quantitative point of view, a factor that is generally indispensable for training a neural network. Starting from the cleaning of the dataset, which served above all to obviate the problem of the presence of errors, this phase was long and complex, as a manual scan was carried out, comparing each image of the dataset with its corresponding mask, in order to identify any errors.

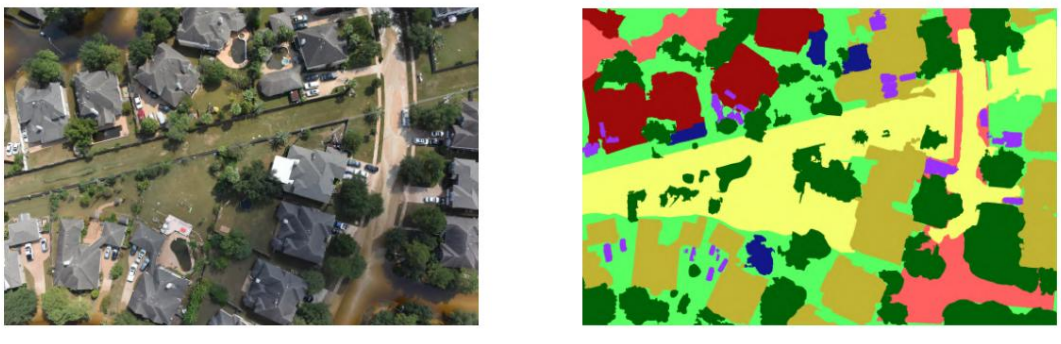
From this scan, a total of 182 images were found to have major errors, such as those described in the previous paragraph. Of these 182, 160 were images containing both "flooded" classes (road and building). Comparing this last datum with the total number of images containing these two classes (approximately 200), it can be noted that, in addition to the intrinsic difficulties of the two classes, this considerable presence of errors has greatly disadvantaged them. In



(to) (b)

(c) (d)

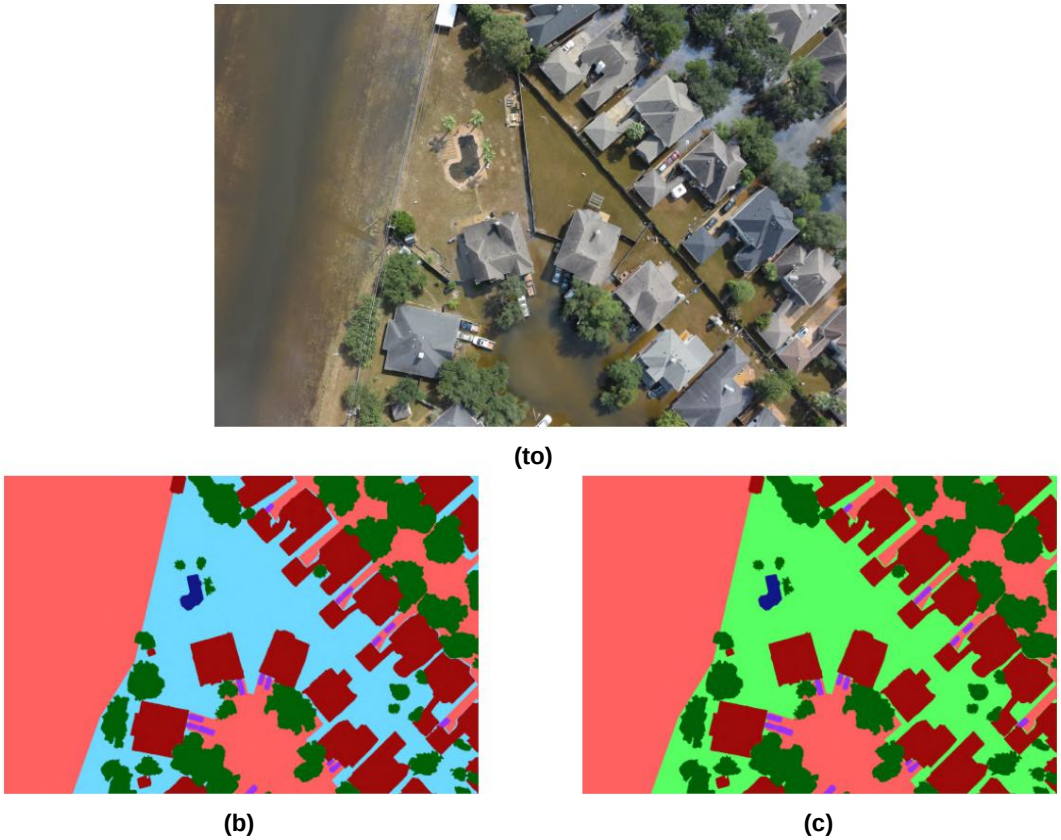
Figure 4.3. The figure shows an example of the second type of error present in the dataset. In particular, as can be seen, in figure (b) a large portion of an occurrence of the "flooded road" class is missing and the corresponding pixels are classified as "water". In figure (d), on the other hand, an occurrence of the "road" class is not reported and the corresponding pixels are classified as "lawn".



(to) (b)

Figure 4.4. The figure shows an example of the third type of error present in the dataset. In particular, as you can see, on the right side of the mask there is an inconsistency represented by the fact that different portions of the same road are classified as "non-flooded road" and "flooded road", even if there is water. In addition, some buildings surrounded by flooded streets are classified as "non-flooded building".
Finally, in the central part of the image an occurrence of the class "non-flooded road" is reported, when in reality it does not appear in the image.

following this phase of ascertaining the consistency of the problem, the correction phase continued. During this phase, some of the errors were correctable through different methods, depending on the nature of the error, while others, due to their nature, did not allow the correction in a not too long time and consequently, in order not to slow down too much the work, have been discarded. In particular, of the 182 total, 45 were not corrected but discarded. Figure 4.5 shows an example of how a mask containing an error was corrected.



(to)

(b) **(c)**

Figure 4.5. The three figures are an example of correction made on the FloodNet dataset masks. Figure (a) is the image to which the two masks (b) and (c) refer. As you can see, in (b) a portion of the lawn is classified as water (light blue), while (c) is the correct version.

While the cleaning phase has tried to overcome above all the fourth problem, the following phase of offline data augmentation has instead tried to solve above all the problem of imbalance towards some classes.

As pointed out by the authors of [59], data augmentation can create undesirable effects, disadvantaging one class over others. For this reason, this offline data augmentation phase was not performed on the entire dataset, but on a group of selected images. In particular, as already mentioned, the purpose of the selection was to increase the presence of those classes that concern the imbalance problem, especially the "flooded road" and "flooded building" classes, without however unbalancing the dataset even more towards very early classes

listen, like the "meadow" class. For the implementation of this augmentation date
the well-known Albumentations library [75] was used and the general mechanism is
was the following: each of the 140 selected images was applied for
three times a random combination of four operations (not always all and
four), producing three more for each image (Figure 4.6). In particular, the
transformations used are: *Rotate,* which consists in the rotation of a quantity
random image degrees; *VerticalFlip,* i.e. the image is rotated to
itself on the axis that crosses it horizontally; *HorizontalFlip,* that is the same
transformation but on the vertical axis; and finally *RandomBrightnessContrast,* which
consists of randomly altering the brightness and contrast of the image.

In addition to using offline data augmentation, to increase the number of
images of the dataset and to overcome the imbalance problem, a second phase of
online data augmentation was used, ie during training.
The aim, in this case, was not to increase the number of images but to obtain
a regularizing effect by adding a different one to each training period
change to each image. The insight behind this method is that in this way, at each
epoch the model sees a slightly different version of the dataset and
this prevents the model from falling into the mechanism as you progress through the training
of overfitting. In particular, the
Albumentations library and the general mechanism is as follows: during an era,
each time an image is loaded for the construction of a batch it is applied
a *VerticalFlip operation,* a *HorizontalFlip* operation and finally a *RandomBright
nessContrast operation.* The key to this method is that, as in the date augmentation done
offline, a probability is associated to each transformation, consequently to
each epoch not all three transformations are applied, but only a random subset of the
three, creating in practice a different version of the dataset to
every era.

4.3 DeepLabV3

The main difficulties that the authors of [76, 49, 36] have highlighted regarding the
application of DCNNs to the general task of semantic segmentation are two:
the first refers to the low resolution of the feature maps produced by the DCNNs,
while the second concerns the presence within the image of different objects
ladder. In particular, especially the second difficulty presents a strong parallel with
one of the main issues addressed in this work. And this is it
the reason for choosing to use this model. Returning to the discussion of the difficulties
of the DCNN in semantic segmentation, the first difficulty is
mainly caused by the combination and alternation of layers of pooling and convulsions
with stride. In particular, this is a feature that represents one of the
strengths of DCNNs in fields such as classification, where reducing the resolution of
feature maps serves precisely to learn very abstract representations
of the data and to acquire the invariance with respect to its local transformations.
Clearly, this strength turns into a weakness in the segmentation task,
where localization is key. The second difficulty, on the other hand, is caused by the
nature of the data and essentially concerns the presence in the image of objects of

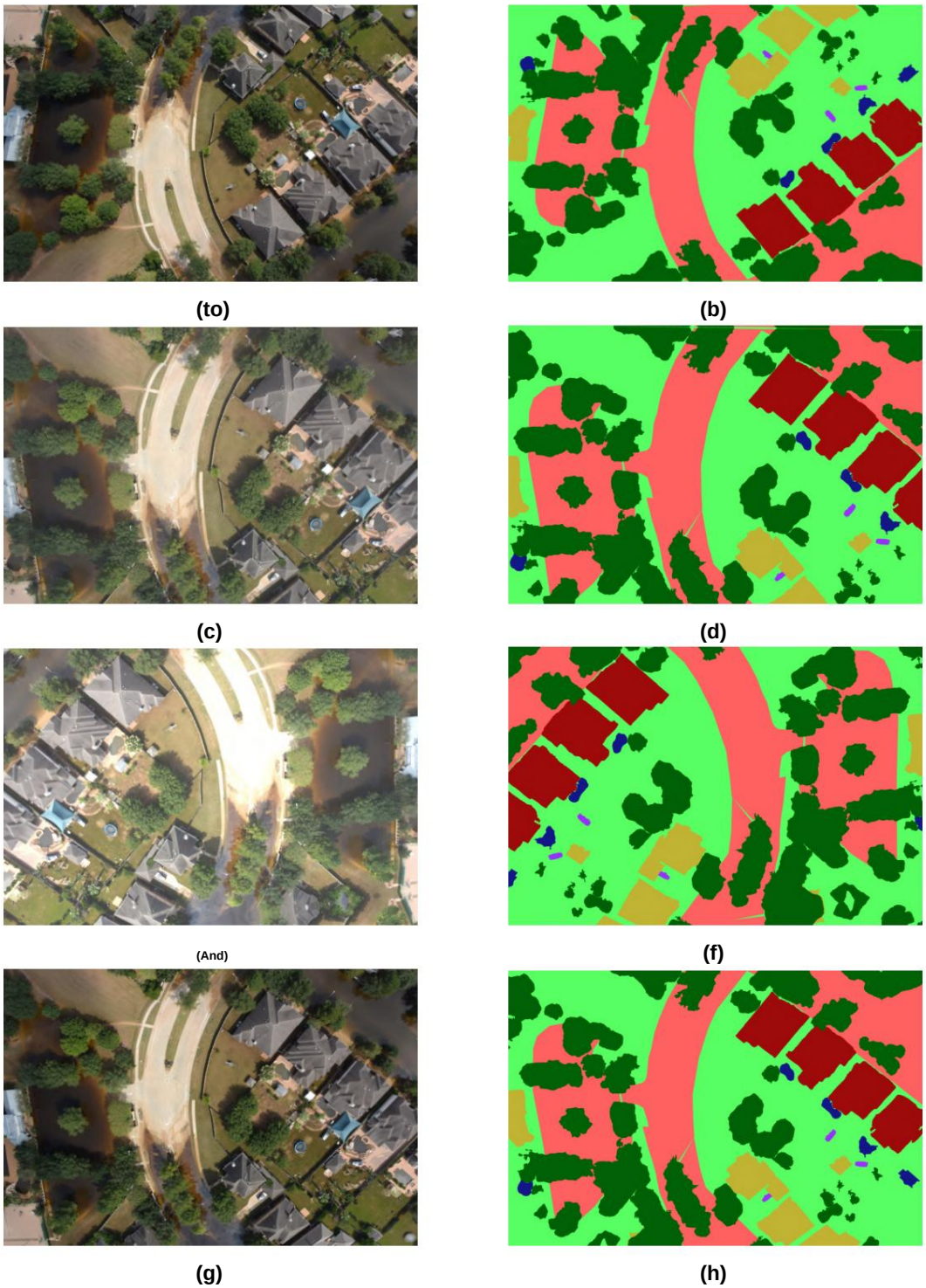


Figure 4.6. The eight figures show an example of the data augmentation used offline before training. Figures (a) and (b) are the original image and mask, while all the others are the result of applying a subset of the operations Rotate, HorizontalFip, VerticalFlip and RandomBrightnessContrast.

different sizes. To solve the first problem, the authors propose a
approach based on the use of a particular type of convolution, namely the
dilated convolution *(dilated convolution* or also *atrous convolution).* In particular, the authors
propose to remove the downsampling step and use a
convolution whose kernel has undergone an upsample, putting gaps in it
(Figure 4.7).

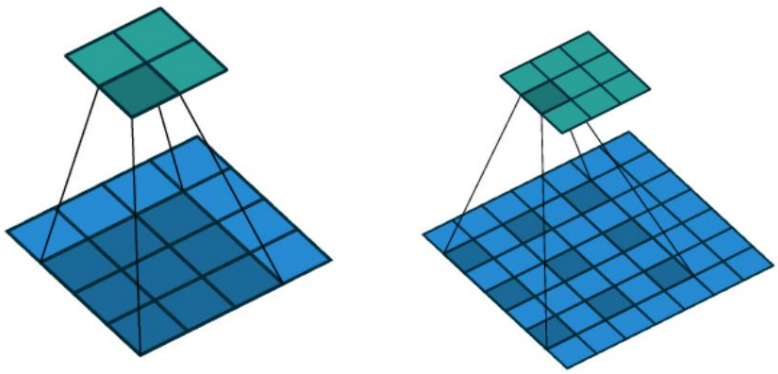


Figure 4.7. Illustration of the difference between a dilated convolution (right) and a normal convolution (left).

As previously said, the main reason for the choice of this specific model consists in the
fact that the authors' work aims precisely to solve the
problem of objects at different scales. In particular, inspired by the idea of the SPP
of [15], propose a similar structure called ASPP (Atrous Spatial Pyramid
Pooling), which is based on the concept of using dilated convolutions with different latation
in parallel. In addition to solving this problem, the DeepLabV3, being
one of the best known models of the context-based category 2.3.1, lends itself to solving
also the problem of the intrinsic difficulty of the two "flooded building" classes
and "flooded road". As regards the specific version of the architecture used in this work, for
the first part of the network, called the *backbone,* which has
the role of producing the feature maps that will then be passed to the ASPP to produce the
final mask, the ResNet101 is used, i.e. a specific version
of the architecture based on residual layers proposed in [65].

4.3.1 Total architecture

Starting from the general structure of the architecture, it is composed of:

- **backbone:** first part of the network responsible for feature extraction. It produces a feature map of 2048 channels from the input image. In in particular, a modified version with dilated convolutions is used of ResNet101.

- **DeepLabHead:** second part of the network, responsible for the production of the final mask. Composed of the ASPP and a convolutional block which produces a volume 9xHxW, where 9 is the number of classes.

- **Bilinear interpolation:** responsible for the last upsampling phase, thanks to which the output of the DeepLabHead is brought to the original size of the image.

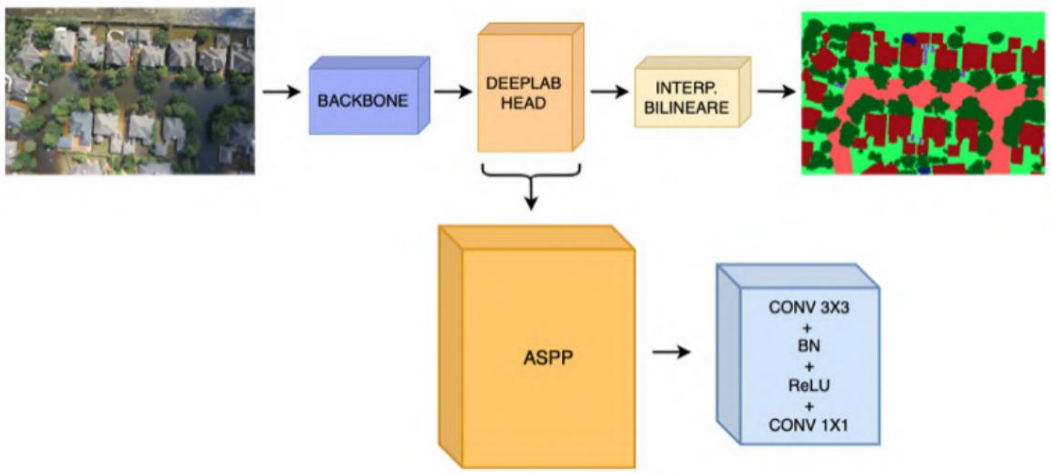


Figure 4.8. Illustration of the total architecture used.

4.3.2 Dilated Convolution

As already mentioned, within DeepLabV3, dilated convolution is used to cope with the problem of reduced resolution of feature maps. In in particular, the problem arises when, in the second part of the network, the features they undergo an upsample to return to the original resolution and produce the output the final. Specifically, the quality of the resulting mask is poor and does not contain very detailed spatial information. With the use of dilated convolution instead, coupled with bilinear interpolation for the upsample phase, the mask produced contains more detailed spatial information (Figure 4.9).

In addition to the speech inherent in the resolution of feature maps, another advantage of dilated convolutions, very useful in the context of this work, is the ability to increase the size of a convolution without increasing the number of parameters and consequently without increasing the computational cost of the architecture, which it often becomes a problem. In particular, one of the main functions of convolutions is to capture the context of a pixel, that is the region around the pixel, from which it strongly depends on its semantics. Often, however, the context of a pixel is wider of what the convolution window, called *the receptive field,* can capture and here the advantage of dilated convolutions comes into play. In particular, upon entering in the detail of their operation, they present one more parameter than to traditional convolution, called dilation, which represents the amount of empty that are inserted into the kernel. The more this parameter increases, the more the window gets bigger and the more the amplitude of the captured context increases, not changing but the actual number of parameters. Furthermore, we can consider convolutions

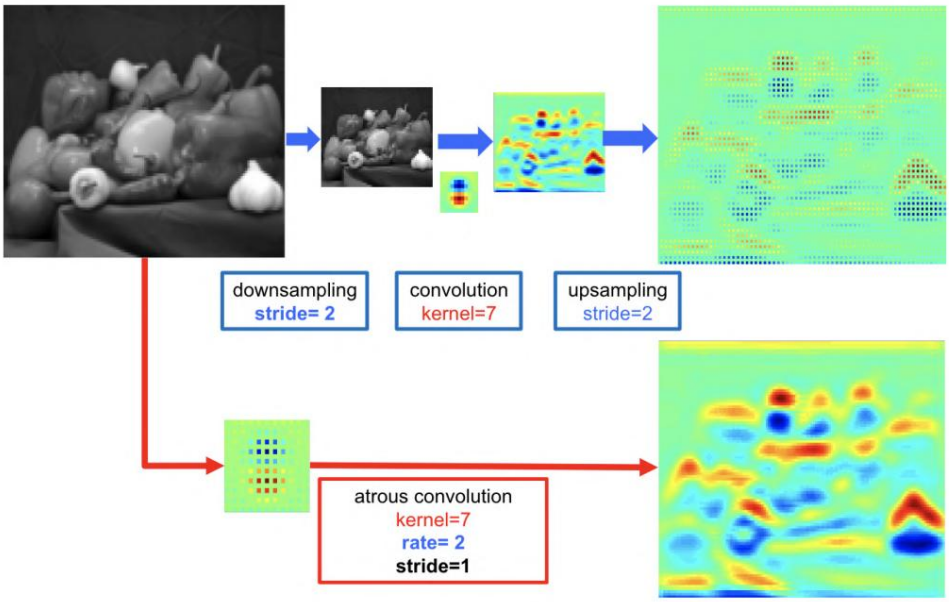


Figure 4.9. Illustration of the dilated 2-D convolution. As can be seen, the mask produced in the upper row (classical convolution) presents more sparse and less detailed information than the mask produced in the lower row (dilated convolution and bilinear interpolation).

dilated as a generalization of normal convolutions, since the latter are a particular case of the former, or with the dilation set to 1 a dilated convolution becomes a normal convolution. Generalizing to the case of 1-D data, the output *y [i]* of a dilated convolution with parameters *w* of length k, which takes as input *x [i]* is defined as:

$$y[i] = \sum_{k=1}^{K} x[i + rk]\, w[k]. \tag{4.1}$$

4.3.3 Residual Neural Networks

Since 2012, deeper and deeper models have been built to increase the performance of neural networks, and the belief has arisen that by increasing the layers of the networks more and more, greater accuracy can be achieved. This belief, however, has been debunked. In particular, it has been shown that, while in theory deeper networks can approximate more complex patterns and consequently obtain better performances, in practice there is a certain threshold above which the accuracy of the models becomes saturated. For example, it has been shown that the "optimal depth" of the models tested on the well-known ImageNet dataset is between 16 and 30 layers, and that a model with 18 layers performs better than one with 34 (Figure 4.10). We can also see a similar result with the CIFAR-10 dataset, with which testing a model with 20 layers and one with 56, the phenomenon is the same (Figure 4.11).

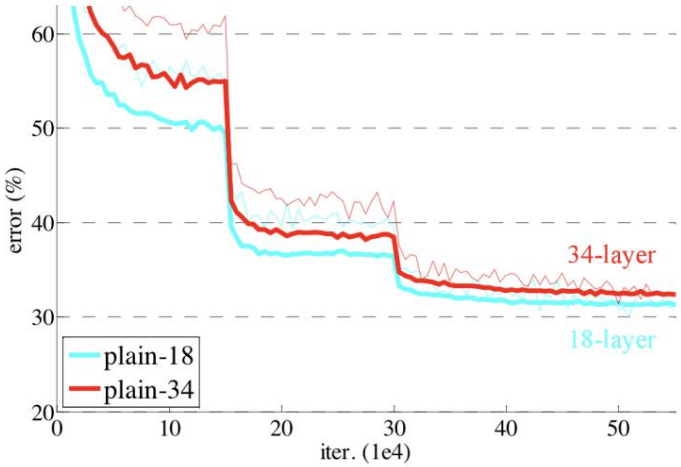


Figure 4.10. ImageNet training of architectures without residual layers *(plain).* The thin curves denote the training error, while the bold curves denote the validation error. As can be seen, the 34-layer network has the highest training error than the 18-layer network throughout the training [65].

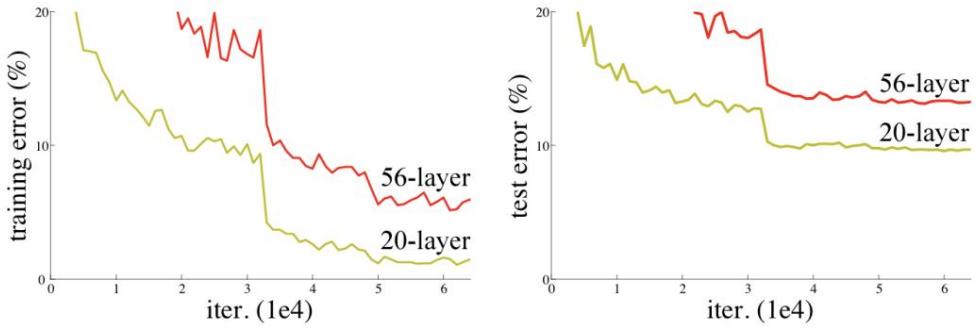


Figure 4.11. Training on CIFAR-10 of architectures without residual layers. Again, throughout the training, the network with more layers (56) has both the training error and the test error higher than the one with fewer layers (20) [65].

The main cause of accuracy saturation lies in the difficulty of optimizing the network due to problems such as the *disappearance of the gradient* and *the explosion of the gradient.* To alleviate this difficulty, a type of layer called residual layer [65] has been introduced, ie a layer within which the output is not *F (x)* (where *x* is the input of the layer) as in normal layers of a network, but *F (x) + x* (Figure 4.12). In particular, the input of the layer is added to the output and this mechanism is implemented with what are called *skip connections.*

Adding *x* to the output of the layer solves the problem of the disappearance of the gradient, since in the case of zeroing the parameters of the layer, its output would not be zeroed anyway, but would be equal to the input. This aspect, in addition to solving the disappearance of the gradient, represents a second advantage of the layers

residuals, i.e. the addition of a residual layer cannot cause particular worsening of the network, both in terms of optimization and performance. In particular, the addition of a residual layer guarantees, in addition to the possible improvements discussed above, not to add particular difficulties to the optimization and not to worsen the performance of the network. This guarantee is given by the fact that, in the event that the addition did not improve the performance, it would not complicate the optimization too much, since with the addition of the x, the network only needs to reset the parameters of the layer to approximate the identity function. In particular, the identity function takes x as input *and* returns x, thus not changing the output of the previous layer.

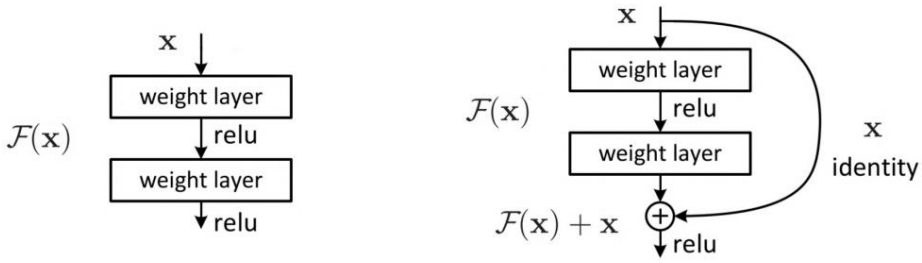


Figure 4.12. Illustration of the difference between a classic layer and a residual layer.

Figures 4.13 and 4.14 show the effect of the residual layers. In particular, it can be seen that in Figure 4.13, compared to Figure 4.10, the situation is reversed, i.e. the model with residual layers and with a total of 34 layers (ResNets-34) has, during all training, both a training error than a lower validation error than the equivalent with fewer layers (ResNet-18). Similarly, in the graph of architectures with residual layers trained with CIFAR-10 (Figure 4.14 on the right) the situation is reversed compared to the graph of *plain* architectures (Figure 4.14 on the left). In addition to this, which according to the authors of [65] is the demonstration that residual layers solve the optimization problems of deeper architectures, these graphs demonstrate that the use of residual layers can improve the performance of individual models, beyond the presence of optimization problems. In fact, when comparing Figures 4.10 and 4.13 it can be seen that even the performance of the smallest model has improved.

ResNet101 as a backbone

As previously mentioned, the version used as feature extractor (backbone) in the architecture of this work is ResNet101, a particular version of the architecture proposed in [65] composed of 101 total layers. In particular, taking up what was mentioned in paragraph 4.3, the original version of the network with residual layers is modified to transform it from a classifier to a feature extractor, while maintaining the number of parameters unchanged thanks to the dilated convections. Going into the details of the architecture used, it is composed of a first block within which we find a 7x7 convolution, which takes the 3 RGB channels as input and returns 64 (number of kernels); a layer of

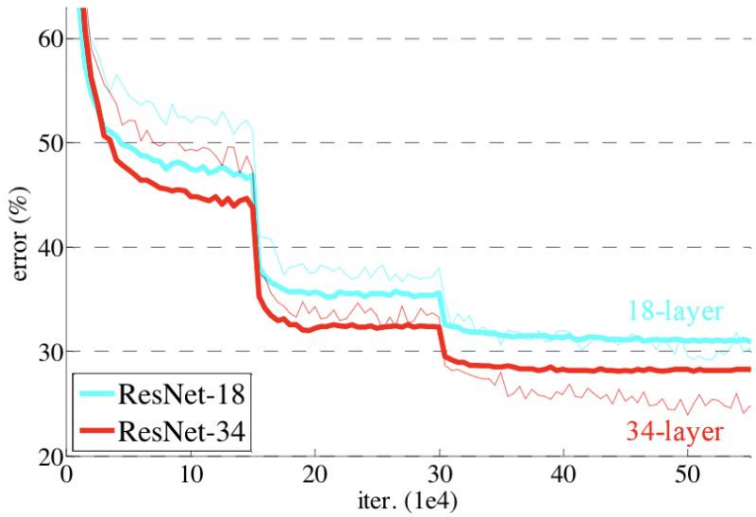


Figure 4.13. ImageNet training of architectures with residual layers. The subtle curves denote the training error, while those in bold denote the validation error [65].

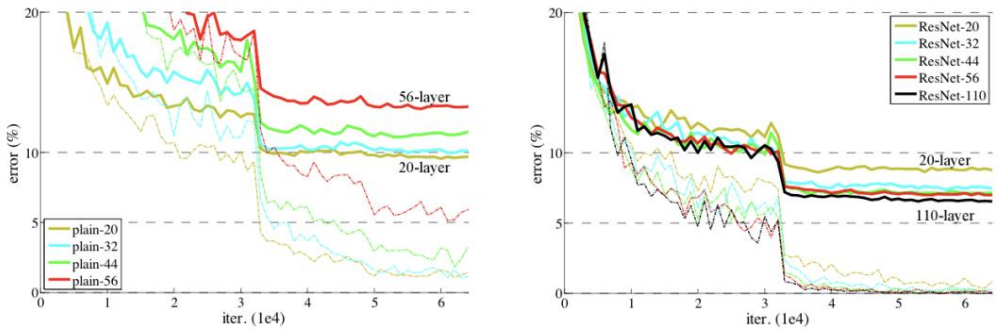


Figure 4.14. Training on CIFAR-10. On the left, the graph of architecture training without residual layers *(plain)* which shows how the deeper architectures are worst. On the right, however, the same graph is shown but concerning architectures with residual layers [65].

batch normalization; one of activation (ReLU) and finally one of max pool. After that, the architecture follows a pattern in common among all the other versions (ResNet18, ResNet34, ResNet50, ...), consisting of four blocks. The latter, consisting of a type of block called *bottleneck* (Figure 4.15b), they differ from each other only in number of bottlenecks and whether or not to use dilated convolution.

In ResNet101 in particular, the four blocks are composed respectively of 3,4,23 and 3 bottleneck blocks and in some blocks the dilated convolution is replaced the classic one (Figure 4.16).

Going into the detail of the bottlenecks, the general structure is composed of one sequence of three convolutions: a 1x1 convolution, a 3x3 (which can be classical

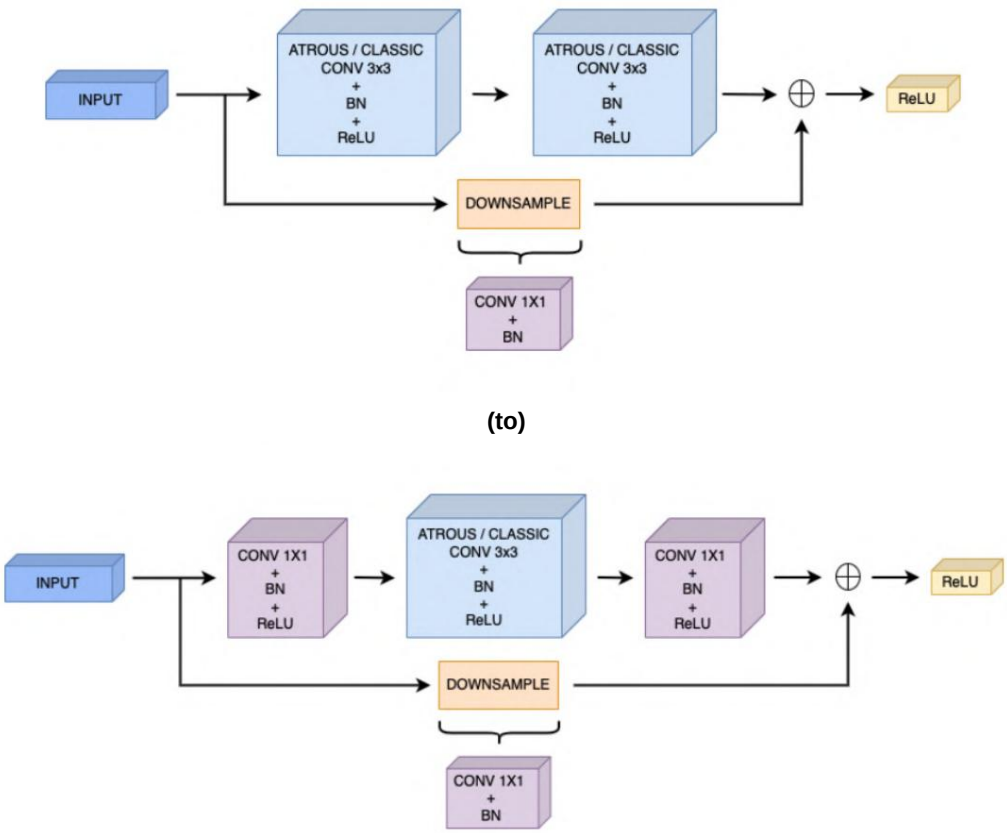


Figure 4.15. The line above shows the original version of the residual layer, while the one below the version used in deeper architectures to decrease the com putational cost of the single block (bottleneck).

or dilated depending on the layer) and finally another 1x1. Furthermore, each convolution is followed by a batch normalization layer and clearly by an activation layer (except the last convolution which is followed only by batch normalization). Finally, as described in the previous paragraph, through a skip connection the bottleneck input is summed with the output of the last convolution and the result is then passed into an activation layer. Since, passing through the convolutions, the input reduces its size, to add input and output the former is previously passed in a downsample layer (Figure 4.15). The rationale for using these convolutions and in particular 1x1 convolutions is to reduce the number of channels before moving on to 3x3 convolution, for computational cost reasons. In particular, the first 1x1 convolution has the role of reducing the channels, while the second has the role of bringing them back to the original number.

The original version of the architecture, intended for classification, after these four blocks had an average pool layer and finally a block of dense layers.
For our purposes, however, the output used was that of the fourth and last block

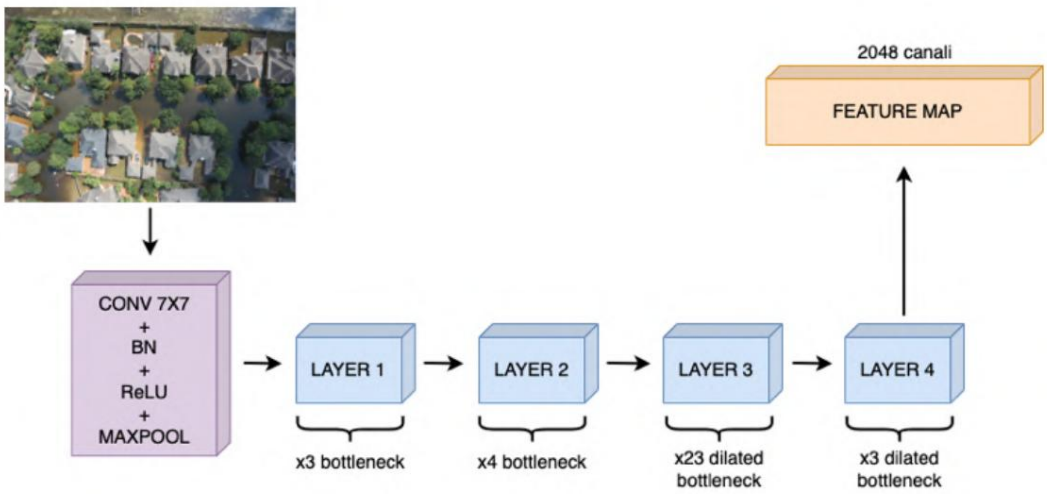


Figure 4.16. Total ResNet101 architecture with dilated convolutions, used as backbone in DeepLabV3.

convolutional, which returns a feature map of 2048 channels. The reason why this version of ResNet was chosen is that it was decided to use one of the deeper versions, as the authors highlight how, by increasing the number of residual layers, performance improves. At the same time, however, given the major limitations of computational resources, deeper versions have not been chosen, such as ResNet152, since even if in some cases the performances are better, the difference is often minimal, while not it is the difference from the point of view of complexity (7.6x109 FLOPs for Resnet101 and 11.3x109 FLOPs for ResNet152 [65]).

4.3.4 Atrous Spatial Pyramid Pooling

Once the feature map is produced, it is then passed to the second part of the network, often called *DeepLabHead.* In particular, this part is composed of two blocks: the ASPP and a block that has a 3x3 convolution in sequence, batch normalization ReLU and finally a last 1x1 convolution that produces a volume of 8 channels, equivalent to the number of classes of the task general idea is to use several convolutions dilated in parallel, varying the dilation, and then concatenate the resulting volumes.

The intuition is to capture spatial contexts and information at different scales. In particular, in the architecture used, the ASPP is composed of:

- a block with a classic 1x1 convolution, batch normalization and ReLU.
- a block for each expansion parameter used (12, 24 and 36) composed from a 3x3 dilated convolution, batch normalization and ReLU.

- a block within which a global average pooling is performed, that is a pooling that reduces the input volume to Cx1x1 where C is the number of channels, a 1x1 convolution, batch normalization, ReLU and finally a bilinear interpolation operation , which has the purpose of bringing the result of the block back to the right size to be chained with the others.

As a result, in total the ASPP has 5 blocks which are executed in parallel. Finally, the outputs of all blocks are concatenated and the result is passed into a last block consisting of a 1x1 convolution, batch normalization, and ReLU (Figure 4.17).

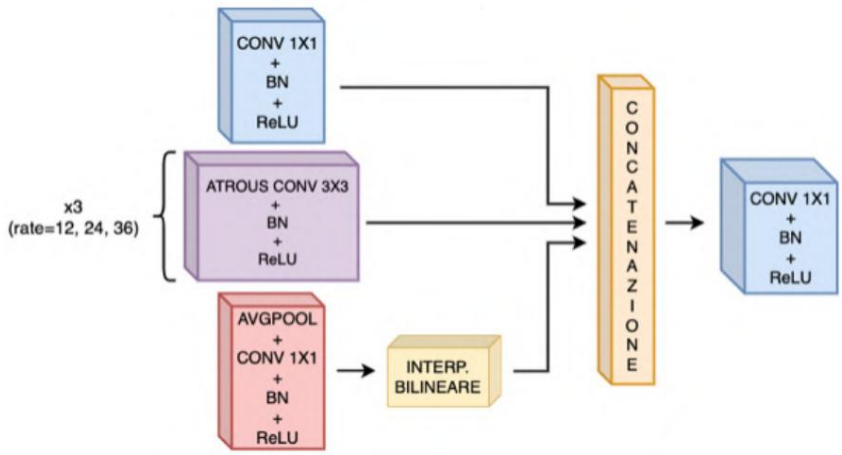


Figure 4.17. Illustration of the architecture of the ASPP module.

Chapter 5

Experiments and Results

In this chapter we will describe the experiments and the results of the work. In particular, some implementation details will be described, such as the libraries and the frameworks used, but also the computational resources exploited. Furthermore, the strategies adopted during the training, the hyperparameters and the various methods tested in the experiments will be described in detail.

5.1 Computational resources

Given that the training of a neural network involves a very high computational cost, not having the availability of a local machine with enough computational power to be able to train the model in an acceptable time, a well-known platform, called Google Colab, was used. In particular, Google Colab allows you to remotely execute code on virtual machines equipped with high computing power, giving above all access to virtual machines with GPUs, which are essential for training a neural network.

Unfortunately, however, Google Colab has severe limitations, not so much from the point of view of the computational power of the resources, but from the point of view of their availability. Specifically, in order to be able to offer computational resources for free and accommodate the high number of requests, the availability of hardware resources undergoes strong fluctuations. In particular, theoretically the maximum life of a virtual machine used is 12 hours. In practice, however, this number is always lower, both due to the fluctuations in requests mentioned above, and due to other mechanisms:

- Notebooks used to execute the code have an idle timeout. Consequently, especially for the training of a neural network, which involves long passive times, this aspect was highly limiting.

- The use of GPUs are prioritized for users who use Colab in a more interactive way. Here too, we find a strong contrast with the type of use that concerns the training of a neural network.

Consequently, given all these aspects, Google Colab's resources were neither guaranteed nor always available. This means that, while working

the availability of the resources necessary to carry out the experiments, had significant limitations, resulting in large slowdowns. In practice, Google Colab's hardware resources have been available for an average of approx 5/6 hours a day. The technical specifications of the supplied hardware are as follows:

- **CPU:** Intel (R) Xeon (R)
- **RAM:** 12GB
- **GPU:** NVIDIA K80 12GB

5.2 Technologies used

As for the implementation of all methodologies, they have been used various libraries and frameworks. In particular, the main ones used are:

- **PyTorch:** an open source Machine Learning framework based on the Python programming language and the Torch library. It is most used in the field of Deep Learning, together with Tensorflow.
one of the frameworks
- **CUDA (Compute Unified Device Architecture):** a hardware architecture for parallel computing developed by NVIDIA. In particular, CUDA allows you to make the most of the computing power of a GPU, parallelizing computations efficiently and the result is that performance, especially as regards the training and inference phase, they are higher than those of a CPU.
- **Albumentations** [75]: a Computer Vision tool based on the language of Python programming, which is meant to make data augmentation fast and flexible. It efficiently implements a rich variety of image transformations, optimized for performance, providing a concise yet powerful data augmentation interface for various Computer tasks. Vision, including classification, segmentation, object detection and others.
- **OpenCV (Open Source Computer Vision Library):** an open library source of Computer Vision and Machine Learning built to provide a common infrastructure for Computer Vision applications and to accelerate use of *machine perception* in commercial products.

5.3 FloodNet dataset

For training and evaluation of the methods and architectures used in this work, the *FloodNet* dataset was used [30]. Specifically, this dataset is was published in 2021 following the FloodNet Challenge of the EARTH VISION 2021 workshop held by CVPR 2021 (Computer Vision and Pattern Recognition Conference), one of Computer Vision's leading annual events, which includes numerous conferences and workshops. Floodnet was one of the first public datasets of its kind: in particular, it was the first public dataset with images and videos

of high resolution and low altitude UAVs, concerning the phase immediately following a natural disaster. Furthermore, FloodNet is one of the few datasets in this area that can be used for three different tasks, namely classification, semantic segmentation and VQA (Visual Question Answering). The dataset images were captured between August 30 and September 4, 2017 in Te xas (USA), immediately after the disaster caused by Hurricane Harvey, with a DJI Mavic Pro quadcopter at 200 feet of altitude. In total, 2343 images were collected with a resolution of 1.5cm per pixel. Regarding the segmentation task, each pixel of each image has been annotated with one of the 9 classes ("flooded building", "non-flooded building", "flooded street", "non alla gata street", "water", " tree "," vehicle "," swimming pool "and" lawn "). As mentioned earlier, FloodNet, at the time of its publication, was a unique dataset of its kind. In particular, prior to its publication, most public datasets with images of natural disasters were of satellite origin. The problem with this type of images is that often, to obtain this type of data, the waits are several days and therefore there are no faithful data to the phase immediately following the disaster. Furthermore, their resolution is clearly much lower than low-altitude images, as a result they often lack detailed information on the damage caused by the event. The dataset is supplied already divided into the three parts of training, validation and testing, with the following proportions: ÿ 60% for the training set, ÿ 20% for the validation set and ÿ 20% for the test set. As already mentioned in paragraph 4.2, during the work the number of total images that make up the dataset has undergone some variations. In particular, the dataset had three versions: a version in which all 182 images found with important errors were discarded, for a total of 2161 images; a version where 182, 137 of these have been corrected and reinserted, for a total of 2298; and finally a last version where in addition to the correct ones, 420 images were added, the result of offline data augmentation, for a total of 2718 images.

Figure 5.1 shows some examples of the images and masks present in the FloodNet dataset.

5.4 Experiments

The main experiments carried out are four. In the first, the DeepLabV3 model was tested on the first version of the dataset, that is the one with 2161 total images.
Specifically, beyond the regularization made with the dropout, no particular strategy or technique was used. In fact, the goal was to set a baseline from which to try to improve. The images were resized to 600 * 800, a *batch size* of 2, a learning rate of 0.01, the Adam optimizer and the Focal Loss loss function were used. Table 5.1 shows the results and as you can see, on most of the classes the model performs poorly. The reason probably lies in the fact that using this version of the dataset, many classes are too little present in the images and moreover, without any particular type of regularization (except dropout), the model struggles to generalize.

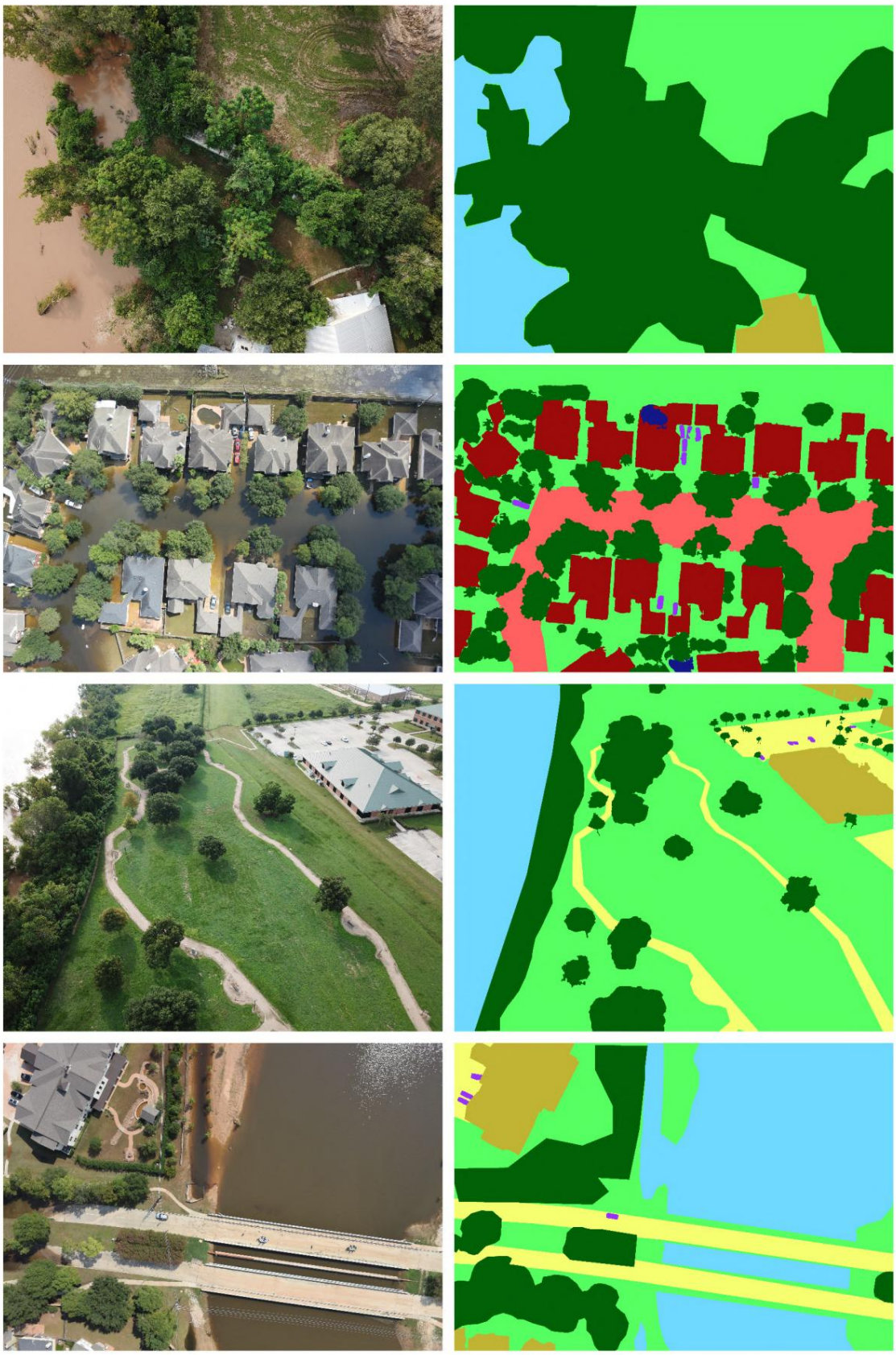


Figure 5.1. Some examples of the images (left) of the FloodNet dataset with matched the corresponding masks (right).

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	mIoU
0.0018	0.087	0.0002	0.28	0.14	0.339	0.006	0.0	0.309	0.129

Table 5.1. Results of the first experiment. Table 5.2 shows the class legend.

Class 1	Flooded building
Class 2	Building not flooded
Class 3	Flooded road
Class 4	Road not flooded
Class 5	Waterfall
Class 6	Tree
Class 7	Vehicle
Class 8	Pool
Class 9	Lawn

Table 5.2. Class legend.

In the second experiment, the online data augmentation phase was introduced discussed in paragraph 4.2. In addition, a method of input normalization, that is, all three channels of the RGB have been divided by 255. The purpose of this normalization is to bring the input from the range [0, 255] to [0, 1], and doing so speeds up training, resulting in a model best with the same number of epochs. The functioning of this mechanism is based on the fact that by normalizing the input, the cost function we try to minimizing during training takes a simpler form to be optimized. Other than that, the hyperparameters from the first experiment remain unchanged. Table 5.3 shows the results of this configuration. How can you note, the normalization and especially the addition of online data augmentation have brought a slight improvement, but the performance is still poor, especially on the more difficult classes, ie "flooded building", "flooded road", "vehicle" and "swimming pool".

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	mIoU
0.0003	0.12	0.001	0.27	0.23	0.38	0.0	0.03	0.52	0.175

Table 5.3. Results of the second experiment.

In the third experiment instead, the second version of the dataset is used, that is the one with 2298 images, inside which the correct masks have been reinserted. In addition to this, we tried to raise the resolution as much as possible of the images, as, some classes such as "vehicle", being represented by very small objects, are strongly disadvantaged by the resizing of images. In particular, being limited by computational resources and trying

to maintain a minimum batch size of 2, the resolution was raised to 750 * 1000.
Table 5.4 shows the results of this third experiment. As you can see, the application of these two strategies has brought about a marked improvement, both as regards the simpler classes, but above all as regards the more difficult ones. The reason, in addition to the fact that by raising the resolution the model receives more information, is above all that, thanks to the cleaning and correction of the dataset, an important number of images have been reintroduced. Specifically, it's not so much the total number of images added that made the difference, but rather the fact that most of these images mostly contained the poorest performing classes. In fact, as shown in Table 5.5, the classes that have had the largest improvement in proportion are "flooded building", "flooded road", "vehicle" and "swimming pool". Here too, the hyperparameters such as learning rate, batch size, optimizer and loss function remained unchanged.

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	mIoU
0.14	0.47	0.06	0.48	0.46	0.55	0.34	0.26	0.81	0.402

Table 5.4. Results of the third experiment experiment.

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8 +	Class 9	mIoU
46500% + 291% + 5900% + 77% + 100% + 44% (inf) + 766% + 55% + 129%									

Table 5.5. Improvements made in the third experiment compared to the second.

Finally, in the fourth and final experiment, the latest version of the dataset is used, that is the one with 2718 total images, the result of offline data augmentation. Table 5.6 shows the results. Here too we find a clear improvement over the previous configuration on all classes, but above all, as shown in Table 5.7, on the two classes "flooded building" and "flooded road".

Here too the main reason is that, thanks to the addition of images containing mainly these two classes, the model has more material to learn them.

Class 1 0.32	Class 2 0.51	Class 3 0.24	Class 4 0.55	Class 5 0.55	Class 6 0.6	Class 7 0.4	Class 8 0.44	Class 9 0.84	mIoU
									0.5

Table 5.6. Results of the fourth experiment experiment.

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	mIoU
+ 128% + 8% + 300% + 14% + 19% + 9% +17 + 69% + 3% + 24%									

Table 5.7. Improvements made in the fourth experiment compared to the third.

Given the improvements made with this configuration, it was decided to push the training phase as much as possible. In particular, inspired by the work of [76, 49, 36], the following strategy was adopted: starting from the eighteenth century, the learning rate was decreased every 100 epochs, multiplying it by 0.1. Pushing the training up to 1000 epochs, the results shown in Table 5.8 were obtained. Specifically, this last training required a total time of about 350 hours, which due to limitations from the point of view of computational resources (Paragraph 5.1), were divided into about 5/6 hours a day, leading to a time total training of about 60 days.

Class 1 0.41	Class 2 0.60	Class 3 0.32	Class 4 0.6	Class 5 0.57	Class 6 0.65	Class 7 0.49	Class 8 0.52	Class 9 0.86	mIoU
									0.564

Table 5.8. Results of the fifth experiment.

Finally, Figure 5.2 shows an example of comparison between the correct masks and the model output.

5.5 Comparison with other works

Table 5.9 shows the comparison between the proposed method and another work of the state of the art [30] concerning the same dataset. A very important factor to consider when comparing the results of this work with those of [30], is that their version of the dataset contains 3200 total images, compared to 2343 in the version of this work. Therefore, their version of the dataset has + 36% of total images, which inevitably entails an important advantage in the training phase. This is also proven by the fact that, in the proposed method, both with the cleaning phase of the dataset and with the offline data augmentation phase, by adding respectively + 6% and + 18% of images, important improvements have been obtained, thus demonstrating the advantage. to have 36% of additional images available. Their further advantage was from the point of view of computational resources. In particular, unlike this work which, as already mentioned, was strongly slowed down for the reasons mentioned above (Paragraph 5.1), their work was based on the continuous availability of a local machine with the Nvidia GeForce RTX 2080 Ti GPU. The continuous availability of computational power that a local machine provides represents an important advantage, especially when compared to an availability of 5/6 hours a day.

Furthermore, again from Table 5.9 it can be noted that, despite the disadvantage deriving from a lower availability of images and a reduced accessibility to computational resources, the proposed method improves the IoUs of the four classes "flooded building", "vehicle", " swimming pool "and" lawn ". Also, an important factor to consider is that the classes "flooded building", "vehicle" and "swimming pool" are three of the four, along with "flooded road", which proved to be the most difficult to learn during work. Finally, all the aspects mentioned above show that the proposed method is promising and that with more adequate resources and with a larger dataset, it could achieve higher performances.

Template	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	mIoU
ENet	0.069	0.473	0.124	0.484	0.489	0.683	0.322	0.424	0.762	0.426
DeepLabV3 +	0.327	0.728	0.52		0.7	0.75	0.77	0.42	0.47	0.84	0.61
DeepLabV3	**0.41**	0.60		0.32	0.6	0.57	0.65	**0.49**	**0.52**	**0.86**	0.564

Table 5.9. Comparison between the developed approach and that of [30].

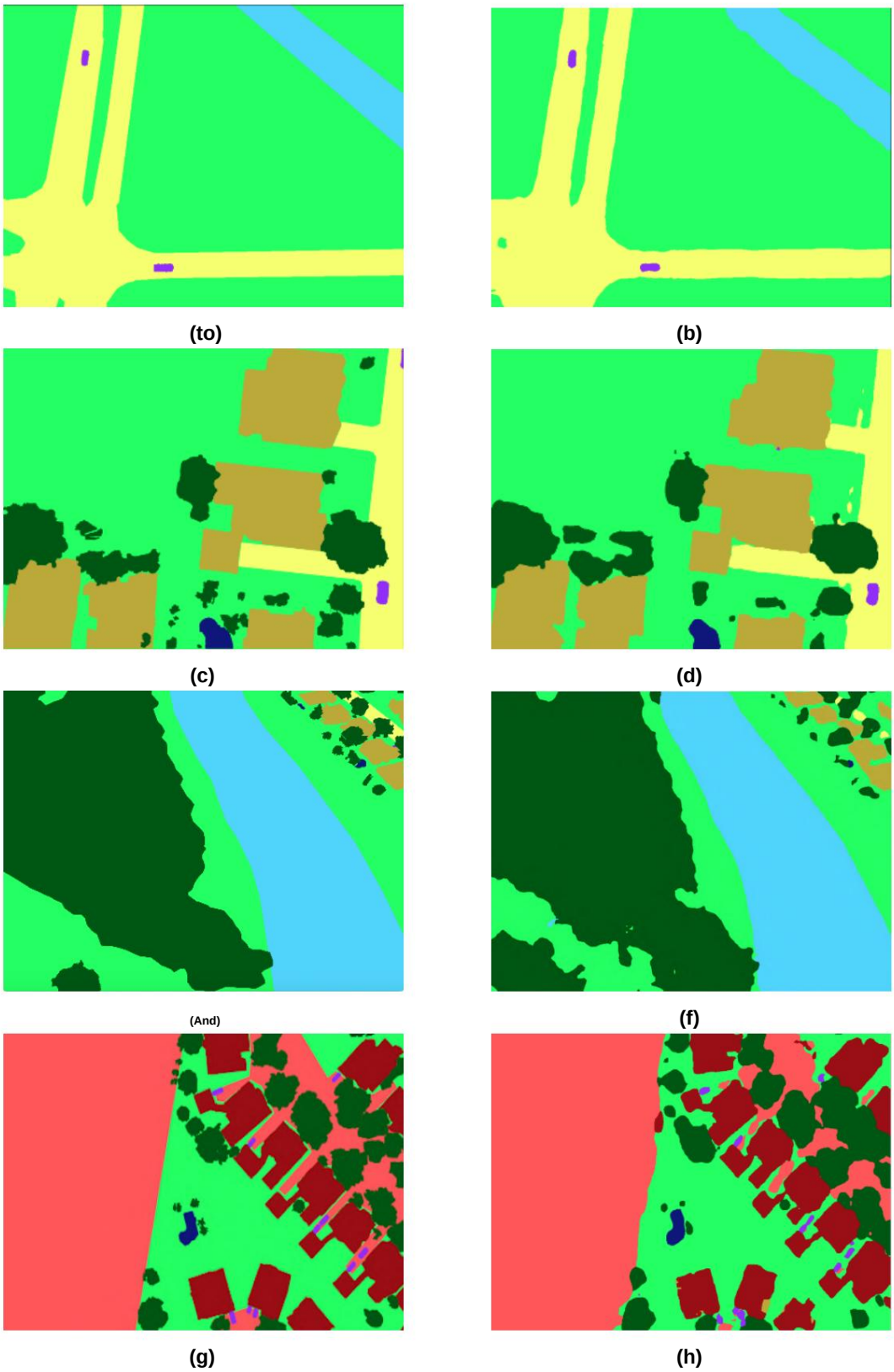


Figure 5.2. The eight images show an example of comparison between the correct masks (a), (c), (e), (g) and the masks produced by the model (b), (d), (f), (h).

Chapter 6

Conclusions

In this paper, an approach based on Deep Learning strategies for the semantic segmentation of the images of the FloodNet dataset was presented. In particular, the developed approach was mainly based on: a Deep Learning context-based architecture never used on this dataset, that is DeepLabV3; a data cleaning phase with which the presence of errors in the dataset masks was addressed; and finally on an offline data augmentation phase, which aimed to increase the number of images in a targeted manner, but also to face other difficulties encountered in facing this task (Paragraph 4.1). The results obtained demonstrate the effectiveness of the proposed method, with results that are close to those of the state of the art due to the following reasons:

- As mentioned in Section 5.1, there have been significant limitations from the point of view of computational resources. In particular, these limitations led to long overhead times, which greatly extended the total time of the experiments, inevitably influencing the results obtained. For example, the last experiment carried out, which was also the most substantial, with a continuous availability of the same hardware resources used, would have ended in about 14/15 days, unlike the approximately 60 days it took.

- As regards the dataset, the version used here presents 2343 images, compared to 3200 (+ 36%) of the state of the art work with which the results were compared. Therefore, even this factor is to be considered limiting for the purposes of evaluating the results.

- The comparison with the DeepLabV3 + architecture (Paragraph 5.5), very similar to the DeepLabV3 used in this work, also considering all the limitations mentioned in the previous points, further highlights the goodness of the approach developed. In particular, even if the mIoU is slightly lower (0.564 vs 0.61), the IoU of three classes, namely "flooded building", "vehicle" and "swimming pool" are higher (respectively 0.41 vs 0.32; 0.49 vs 0.42 ; 0.52 versus 0.47). Furthermore, it must be considered that the three classes mentioned above, in which these improvements have been obtained, represent three of the four most difficult classes.

In conclusion, taking into account all these factors, the developed approach and its results are to be considered more than valid. In particular, the methodologies used, especially the offline data augmentation and the cleaning phase of the dataset, have made a substantial contribution and applying them to other models, with more adequate resources and more extended timescales, could improve the results of the state of the art.

6.1 Future developments

Future developments that could be made to our approach are:

- Reinsert the 45 images discarded during the data cleaning phase, managing to correct them using tools such as the V7 Darwin platform.
- Increased amount of offline data augmentation applied to the dataset. In particular, given the improvements brought about by the addition of images, through both the data cleaning phase and the data augmentation phase, and given that after these two phases the dataset still presented unbalances, even if more slight, a date phase even more substantial offline augmentation could bring further improvements.
- In addition to the increase in terms of quantity, other data augmentation techniques could be added, such as the techniques described in [77], especially useful for improving the performance on classes represented by small objects, such as "vehicle" and "pool".

Bibliography

[1] Margareta Wahlstrom, Debarati Guha-Sapir et al. “The human cost of weather related disasters 1995–2015”. In: *Geneva, Switzerland: UNISDR* (2015).

[2] Sharifah Mastura Syed Mohd Daud et al. “Applications of drone in disaster management: A scoping review”. In: *Science & Justice* 62.1 (2022), pp. 30–42.

[3] Faine Greenwood, Erica L Nelson and P Gregg Greenough. "Flying into the hurricane: A case study of UAV use in damage assessment during the 2017 hurricanes in Texas and Florida". In: *PLoS one* 15.2 (2020), e0227808.

[4] Shijie Hao, Yuan Zhou and Yanrong Guo. “A brief survey on semantic segmentation with deep learning”. In: *Neurocomputing* 406 (2020), pp. 302–321.

[5] Joe Kinahan and Alan F Smeaton. “Image Segmentation to Identify Safe Landing Zones for Unmanned Aerial Vehicles”. In: *arXiv preprint arXiv: 2111.14557* (2021).

[6] Ye Lyu et al. “UAVid: A semantic segmentation dataset for UAV imagery”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 165 (2020), pp. 108–119.

[7] Teja Kattenborn, Jana Eichel and Fabian Ewald Fassnacht. "Convolutional Neu ral Networks enable efficient, accurate and fine-grained segmentation of plant species and communities from high-resolution UAV imagery". In: *Scientific reports* 9.1 (2019), pp. 1–9.

[8] Olaf Ronneberger, Philipp Fischer and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention.* Springer. 2015, pp. 234–241.

[9] Tashnim Chowdhury, Robin Murphy and Maryam Rahnemoonfar. "RescueNet: A High Resolution UAV Semantic Segmentation Benchmark Dataset for Natural Disaster Damage Assessment". In: *arXiv preprint arXiv: 2202.12361* (2022).

[10] Ozan Oktay et al. “Attention u-net: Learning where to look for the pancreas”. In: *arXiv preprint arXiv: 1804.03999* (2018).

[11] Abolfazl Abdollahi, Biswajeet Pradhan and Abdullah M Alamri. "An ensem ble architecture of deep convolutional Segnet and Unet networks for building semantic segmentation from high-resolution aerial images". In: *Geocarto International* (2020), pp. 1–16.

[12] Vijay Badrinarayanan, Alex Kendall and Roberto Cipolla. “Segnet: A deep with volutional encoder-decoder architecture for image segmentation”. In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481– 2495.

[13] Imran Ahmed, Misbah Ahmad and Gwanggil Jeon. "A real-time efficient object segmentation system based on U-Net using aerial drone images". In: *Journal of Real-Time Image Processing* 18.5 (2021), pp. 1745–1758.

[14] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv: 1704.04861* (2017).

[15] Hengshuang Zhao et al. “Pyramid scene parsing network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, pp. 2881-2890.

[16] Liang-Chieh Chen et al. "Encoder-decoder with atrous separable convolution for semantic image segmentation". In: *Proceedings of the European conference on computer vision (ECCV).* 2018, pp. 801-818.

[17] Adam Paszke et al. “Enet: A deep neural network architecture for real-time semantic segmentation”. In: *arXiv preprint arXiv: 1606.02147* (2016).

[18] Dimitrios Marmanis et al. “Semantic segmentation of aerial images with an en semble of CNSS”. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2016* 3 (2016), pp. 473–480.

[19] Kaiqiang Chen et al. "Semantic segmentation of aerial images with shuffling convolutional neural networks". In: *IEEE Geoscience and Remote Sensing Letters* 15.2 (2018), pp. 173–177.

[20] Ruigang Niu et al. “Hybrid multiple attention network for semantic segmentation in aerial images”. In: *IEEE Transactions on Geoscience and Remote Sensing* 60 (2021), pp. 1–18.

[21] Haifeng Luo et al. "High-resolution aerial images semantic segmentation using deep fully convolutional network with channel attention mechanism". In: *IEEE journal of selected topics in applied earth observations and remote sensing* 12.9 (2019), pp. 3492-3507.

[22] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[23] Dengfeng Chai, Shawn Newsam and Jingfeng Huang. “Aerial image semantic segmentation using DCNN predicted distance maps”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 161 (2020), pp. 309–322.

[24] Caio CV da Silva et al. “Towards open-set semantic segmentation of aerial images”. In: *2020 IEEE Latin American GRSS & ISPRS Remote Sensing Conference (LAGIRS).* IEEE. 2020, pp. 16–21.

[25] Asmamaw A Gebrehiwot and Leila Hashemi-Beni. "Three-Dimensional Inundation Mapping Using UAV Image Segmentation and Digital Surface Model". In: *ISPRS International Journal of Geo-Information* 10.3 (2021), p. 144.

[26] Jonathan Long, Evan Shelhamer and Trevor Darrell. "Fully convolutional net works for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 3431–3440.

[27] Ananya Gupta, Simon Watson and Hujun Yin. “Deep learning-based aerial image segmentation with open data for disaster impact assessment ". In: *Neurocom puting* 439 (2021), pp. 22–33.

[28] Abhishek Chaurasia and Eugenio Culurciello. "Linknet: Exploiting encoder re presentations for efficient semantic segmentation". In: *2017 IEEE Visual Com munications and Image Processing (VCIP).* IEEE. 2017, pp. 1–4.

[29] Saheba Bhatnagar, Laurence Gill and Bidisha Ghosh. “Drone image segmentation using machine and deep learning for mapping raised bog vegetation communities ". In: *Remote Sensing* 12.16 (2020), p. 2602.

[30] Maryam Rahnemoonfar et al. “Floodnet: A high resolution aerial imagery from taset for post flood scene understanding”. In: *IEEE Access* 9 (2021), pp. 89644–89654.

[31] Xiaojuan Qi et al. "3d graph neural networks for rgbd semantic segmentation". In: *Proceedings of the IEEE International Conference on Computer Vision.* 2017, pp. 5199-5208.

[32] Shiying Hu, Eric A Hoffman and Joseph M Reinhardt. “Automatic lung segmentation for accurate quantitation of volumetric X-ray CT images”. In: *IEEE transactions on medical imaging* 20.6 (2001), pp. 490–498.

[33] Mark Everingham et al. “The PASCAL visual object classes challenge 2007 (VOC2007) results ". In: (2008).

[34] Marius Cordts et al. “The cityscapes dataset for semantic urban scene un derstanding”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 3213–3223.

[35] Bolei Zhou et al. “Semantic understanding of scenes through the ade20k da taset”. In: *International Journal of Computer Vision* 127.3 (2019), pp. 302–321.

[36] Liang-Chieh Chen et al. “Rethinking atrous convolution for semantic image segmentation ". In: *arXiv preprint arXiv: 1706.05587* (2017).

[37] Wolfgang Köhler. "Gestalt psychology". In: *Psychologische Forschung* 31.1 (1967), pp. XVIII – XXX.

[38] Dejan Todorovic. "Gestalt principles". In: *Scholarpedia* 3.12 (2008), p. 5345.

[39] David Martin et al. "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecolo gical statistics ". In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001.* Vol. 2. IEEE. 2001, pp. 416-423.

[40] Luc Vincent and Pierre Soille. "Watersheds in digital spaces: an efficient algo rithm based on immersion simulations". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 13.06 (1991), pp. 583–598.

[41] Pedro F Felzenszwalb and Daniel P Huttenlocher. “Efficient graph-based ima ge segmentation”. In: *International journal of computer vision* 59.2 (2004), pp. 167–181.

[42] Jianbo Shi and Jitendra Malik. "Normalized cuts and image segmentation". In: *IEEE Transactions on pattern analysis and machine intelligence* 22.8 (2000), pp. 888–905.

[43] Stuart Lloyd. “Least squares quantization in PCM”. In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

[44] James MacQueen et al. "Some methods for classification and analysis of mul tivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability.* Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.

[45] Keinosuke Fukunaga and Larry Hostetler. "The estimation of the gradient of a density function, with applications in pattern recognition". In: *IEEE Transac tions on information theory* 21.1 (1975), pp. 32–40.

[46] Fr Comaniciu and Fr Meer. "Mean shift: a robust approach toward feature space analysis". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.5 (2002), pp. 603-619. doi: 10.1109 / 34.1000236.

[47] Fisher Yu and Vladlen Koltun. “Multi-scale context aggregation by dilated with volutions”. In: *arXiv preprint arXiv: 1511.07122* (2015).

[48] Wei Liu, Andrew Rabinovich and Alexander C. Berg. “ParseNet: Looking Wider to See Better”. In: *CoRR* abs / 1506.04579 (2015). arXiv: 1506.04579. url: http://arxiv.org/abs/1506.04579.

[49] Liang-Chieh Chen et al. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[50] Hyeonwoo Noh, Seunghoon Hong and Bohyung Han. "Learning deconvolution network for semantic segmentation". In: *Proceedings of the IEEE international conference on computer vision.* 2015, pp. 1520–1528.

[51] Pauline Luc et al. "Semantic segmentation using adversarial networks". In: *arXiv preprint arXiv: 1611.08408* (2016).

[52] Shi Dong, Ping Wang and Khushnood Abbas. "A survey on deep learning and its applications". In: *Computer Science Review* 40 (2021), p. 100379.

[53] Shaveta Dargan et al. "A survey of deep learning and its applications: a new paradigm to machine learning". In: *Archives of Computational Methods in Engineering* 27.4 (2020), pp. 1071-1092.

[54] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[55] Filip Karlo Došiloviÿ, Mario Brÿiÿ and Nikica Hlupiÿ. “Explainable artificial in telligence: A survey”. In: *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO).* IEEE. 2018, pp. 0210–0215.

[56] Frank Rosenblatt. "The perceptron: a probabilistic model for information sto rage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[57] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep learning.* MIT press, 2016.

[58] Douglas Heaven. "Deep trouble for deep learning". In: *Nature* 574.7777 (2019), pp. 163–166.

[59] Randall Balestriero, Leon Bottou and Yann LeCun. “The Effects of Regula rization and Data Augmentation are Class Dependent”. In: *arXiv preprint arXiv: 2204.03632* (2022).

[60] Jiuxiang Gu et al. "Recent advances in convolutional neural networks". In: *Pattern Recognition* 77 (2018), pp. 354–377.

[61] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv: 1409.1556* (2014).

[62] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolution in networks". In: *European conference on computer vision.* Springer. 2014, pp. 818–833.

[63] Pierre Sermanet et al. "Overfeat: Integrated recognition, localization and detection using convolutional networks". In: *arXiv preprint arXiv: 1312.6229* (2013).

[64] Gao Huang et al. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, pp. 4700–4708.

[65] Kaiming He et al. "Deep residual learning for image recognition". In: *Pro ceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 770–778.

[66] Gao Huang et al. "Deep networks with stochastic depth". In: *European with ference on computer vision.* Springer. 2016, pp. 646–661.

[67] Rupesh K Srivastava, Klaus Greff and Jürgen Schmidhuber. "Training very deep networks". In: *Advances in neural information processing systems* 28 (2015).

[68] Gustav Larsson, Michael Maire and Gregory Shakhnarovich. "Fractalnet: Ultra deep neural networks without residuals". In: *arXiv preprint arXiv: 1605.07648* (2016).

[69] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 1– 9.

[70] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International with ference on machine learning.* PMLR. 2015, pp. 448–456.

[71] Christian Szegedy et al. “Rethinking the inception architecture for computer vision ". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 2818-2826.

[72] Christian Szegedy et al. "Inception-v4, inception-resnet and the impact of re sidual connections on learning". In: *Thirty-first AAAI conference on artificial intelligence.* 2017.

[73] François Chollet. “Xception: Deep learning with depthwise separable convolution”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, pp. 1251–1258.

[74] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[75] Alexander Buslaev et al. “Albumentations: Fast and Flexible Image Aug mentations”. In: *Information* 11.2 (2020). issn: 2078-2489. doi: 10. 3390 / info11020125. url: https://www.mdpi.com/2078-2489/11/2/125.

[76] Liang-Chieh Chen et al. "Semantic image segmentation with deep convo lutional nets and fully connected crfs". In: *arXiv preprint arXiv: 1412.7062* (2014).

[77] Mate Kisantal et al. "Augmentation for small object detection". In: *arXiv preprint arXiv: 1902.07296* (2019).