Politecnico di Torino

Master of Science Degree in Mechatronics Engineering

Master of Science Degree Thesis

An Artificial Intelligent methodology to estimate the population density of urban areas to compute risk maps for Unmanned Aircraft Systems

Supervisors

Prof. Alessandro RIZZO

Dr. Stefano PRIMATESTA

Candidate

Luca SALTALAMACCHIA

December 2020

Summary

As the adoption and usage of Unmanned Aircraft Systems (UAS) is growing over time, the safety related to UAS flight is questionable, especially when UASs operate in urban areas. Urban areas are characterized by high population density, therefore a potential UAS crash may involve people injuries or even people death, as a consequence, it is essential to preserve human safety in mission planning.

This thesis is based on a previous work done by the research group, in which a path planning algorithm for UASs has been developed to determine a safe flight mission. The proposed methodology uses a risk-based map that quantifies the risk of fly over specific areas. In order to compute a safe path for the UAS several input parameters are required, but, unfortunately, it's not always easy to obtain those specific data, such as the population density and distribution that is one of the most important parameter for the risk assessment. The thesis is carried out within the activities of the Amazon Research Award "From Shortest to Safest Path Navigation: An AI-Powered Framework for Risk-Aware Autonomous Navigation of UASs" granted to Prof. Rizzo.

The main contribution of this thesis is the use of an Artificial Intelligent methodology to estimate the population density of urban areas using aerial images. First of all, the Bing Maps REST Services has been used to easy download aerial images. Then, a Convolutional Neural Network has been developed in order to estimate the population density and distribution and to define the risk-based map for the UAS flights.

Due to the big amount of computational effort needed to train an enough depth CNN, the pre-trained CNN VGG16 has been chosen as starting point thanks to its specificity for large-scale image recognition and its capability to classify common objects over 1000 categories. As a starting point, the pre-trained CNN has been readapted to recognize and classify 224 x 224 RGB images taken from Bing Maps. Then, more and more complexity has been added with the goal of classifying images into multiple ranges of population density values in order to distinguish which zones are with an higher population density, or rather, an higher risk.

As last step, to get even more accurate results, it has been decided to develop another training approach which would avoid errors due to misleading images. Therefore, the Neural Network has been trained through the numerical regression to predict the exact value of the population density of aerial images. This value could be used to compute risk-maps and to plan safe paths for UAS.

Acknowledgements

Table of Contents

List of Tables VIII

List of Figures IX

Acronyms XII

1 Introduction 1
- 1.1 Previous work . . . 2
  - 1.1.1 Population density estimation state of art . . . 3
- 1.2 Current work . . . 4
- 1.3 Outline . . . 4

2 Introduction to Neural Network 5
- 2.1 Basic Neural Network structure . . . 6
  - 2.1.1 Neural Networks Artificial Neurons . . . 7
  - 2.1.2 Descendent gradient . . . 10
  - 2.1.3 Backpropagation . . . 12
- 2.2 Introduction to Convolutional Neural Networks . . . 13
  - 2.2.1 Input image . . . 13
  - 2.2.2 Convolutional 2D layer . . . 14
  - 2.2.3 Pooling 2D layer . . . 15
  - 2.2.4 Flatten layer . . . 16
  - 2.2.5 Dense layer . . . 16
  - 2.2.6 Dropout layer . . . 16
- 2.3 Transfer learning . . . 17

3 Software 19
- 3.1 Bing Maps REST Services Application Programming Interface . . . 19
- 3.2 TensorFlow . . . 22
  - 3.2.1 Keras . . . 23
  - 3.2.2 VGG 16 . . . 23

3.3 Scikit-learn 25
3.3.1 Test 25
3.4 Python packages 29
3.4.1 Requests 29
3.4.2 Pyproj 30
3.4.3 Numpy 30
3.4.4 Matplotlib 31

4 Project development 32
4.1 Bi-class classification problem 32
4.2 Multi-class classification problem 43
4.3 Numerical regression to estimate population density 54

5 Conclusion and future work 60

Bibliography 62

List of Tables

2.1 NAND logic 9

3.1 Confusion matrix 27
3.2 Classification report 28

4.1 Test 1 - Classification results 40
4.2 Test 2 - Classification results 41
4.3 Test 3 - Classification results 42
4.4 Test 1 - Confusion matrix results 44
4.5 Test 1 - Classification report results 44
4.6 Test 2 - Confusion matrix results 45
4.7 Test 2 - Classification report results 45
4.8 CNN configurations for tests 46
4.9 Test 7 - Confusion matrix results 47
4.10 Test 7 - Classification report results 47
4.11 Test 8 - Classification report results 48
4.12 Test 9 - Confusion matrix results 50
4.13 Test 9 - Classification report results 50
4.14 Test 10 - Confusion matrix results 51
4.15 Test 10 - Classification report results 52
4.16 Test 11 - Confusion matrix results 53
4.17 Test 11 - Classification report results 53

List of Figures

1.1 Previous workflow structure 2
1.2 The main architecture of the proposed methodology 4

2.1 An example of fully connected Neural Network structure 6
2.2 Fully connected Neural Network structure 7
2.3 Perceptron (Artificial Neuron) 7
2.4 NAND gate 8
2.5 Sigmoid function 9
2.6 Hyperbolic tangent function 10
2.7 ReLU function 10
2.8 RGB input image matrices example 13
2.9 Convolutional operation example 14
2.10 Max pooling operation example 15
2.11 Transfer Learning example 18

3.1 Aerial image over Politecnico of Turin 21
3.2 TensorFlow environment structure 22
3.3 VGG 16 architecture 24
3.4 Underfitting example 26
3.5 Overfitting example 26
3.6 Request query example 29

4.1 Example of an aerial image classified in the City class 34
4.2 Example of an aerial image classified in the Country class 34
4.3 Graphical represetation of the order used to download the aerial images of a specific large area 34
4.4 Test 1 - Convolutional Neural Network configuration 36
4.5 Test 3 - Loss and accuracy trend 39
4.6 Example of an image classified in Class 1 49
4.7 Example of an image classified in Class 1 49
4.8 Test 1 - Convolutional Neural Network configuration 55

4.9 Test 1 - Scatter plot results 56
4.10 Test 2 - Scatter plot results 57
4.11 Test 3 - Convolutional Neural Network configuration 58
4.12 Test 3 - Scatter plot results 59

Acronyms

UAS
Unmanned Aircraft System

ENAC
Ente Nazionale per l'Aviazione Civile

EASA
European Aviation Safety Agency

FAA
Federal Aviation Administration

VLOS
Visual Line-Of-Sight

BVLOS
Beyond Visual Line-Of-Sight

NN
Neural Network

MSE
Mean Squared Error

MAE
Mean Absolute Error

CNN
Convolutional Neural Network

RGB

Red-Green-Blue

API

Application Programming Interfaces

REST

Representational State Transfer

CRUD

Create-Read-Update-Delete

HTTP

HyperText Transfer Protocol

URL

Uniform Resource Locator

TPU

Tensor Processing Unit

VGG

Visual Geometry Group

WGS84

World Geodetic System 1984

PIL

Python Imagery Library

Chapter 1

Introduction

The use of Unmanned Aircraft Systems (UASs) is growing over time to provide several applications, such as data collection, surveillance, mapping, search and rescue, to name a few. However, even if their technology is growing very fast, it is not the same for their safety. In fact, human safety when UASs are used to provide autonomous flying over urban areas has been questioned. At the moment, the national aviation authorities, such as ENAC (*Ente Nazionale per l'Aviazione Civile*) in Italy, EASA (*European Aviation Safety Agency*), as the European agency, and FAA (*Federal Aviation Administration*) in the US, provide rules for the lightweight unmanned aircraft operations. The majority part of this operation must be performed in Visual Line-Of-Sight (VLOS) and, to provide operations over populated areas, they require the use of harmless drones. However lastly, these agencies are opening to perform also Beyond Visual Line-Of-Sight (BVLOS) even over cities. Urban area are characterized by high population density therefore a potential UAS crash may involve people injuries or even people death, as a consequence, it is essential to preserve human safety in mission planning. So, the importance of a risk-based approach of UAS flight is also highlighted by ENAC and FAA.

This thesis is based on a solid preliminary work done by the research group, in which a path planning algorithm for UASs has been developed to determine a safe flight mission.The aim of this thesis is to extend the current risk assessment tool by developing a Convolutional Neural Network to estimate the population density over urban areas. Then, the estimated population density will be used as input layer for the risk assessment. The thesis is carried out within the activities of the Amazon Research Award "From Shortest to Safest Path Navigation: An AI-Powered Framework for Risk-Aware Autonomous Navigation of UASs" granted to Prof. Rizzo.

1.1 Previous work

The previous work [1] [2] done by the research team is a tool for the risk assessment, to enable risk-informed decision making. It can be schematized as in Figure 1.1.

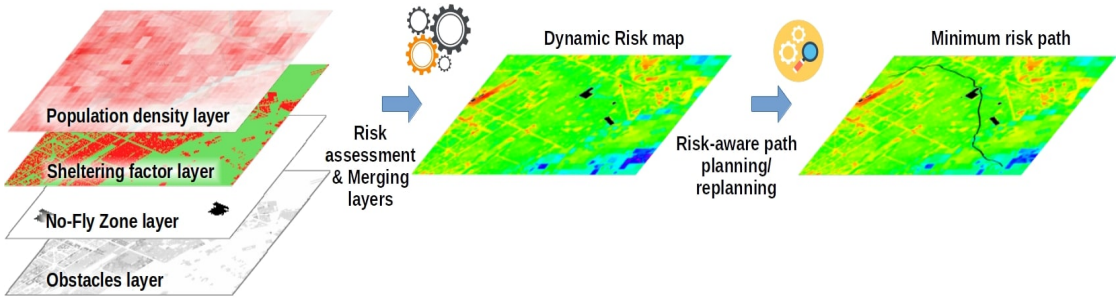

Figure 1.1: Previous workflow structure

This tool is based on the definition of a risk map that quantifies the risk for the population on ground when UASs flies over a urban area. The risk map is a two-dimensional cell-based map in which each cell is associated to a specific risk value. Cells are squared-shaped and they represent a bounded area with labeled geographical references. The risk map is generated as the sum of multiple contribution of several layers, that in turn are a location-based map, containing useful data. The layers to compose the risk map are the following:

- population density layer;
- sheltering layer;
- obstacle layer;
- no-fly zone layer.

Therefore, the risk value is computed as a probability in time to have a fatalities as a consequence of three different events:

- the loss of control of the vehicle with uncontrolled crash on the ground, it is considered four descent event types;
- the impact with at least one person after a crash;
- the impact falls into a fatality.

Thus, this probability can be expressed as the product of the probabilities that the three above events could happen:

$$P_{casualty}(x, y) = P_{event} \cdot P_{impact}(x, y) \cdot P_{fatality}(x, y) \tag{1.1}$$

In order to generate the risk map from which compute the lowest risk path, the $P_{casualty}$ is computed for each descent event type and for each cell of the map.

The population density layer is one of the most important elements in the risk assessment, since it describes how many people can be involved in a casualties due to a drone crash on ground. The attention is focused on the P_{impact}, that is the probability to impact at least one person when a UAS crashes down on ground. It is a function of the impact area and the population density and is defined as:

$$P_{impact}(x, y) = \rho(x, y) \cdot A_{exp} \tag{1.2}$$

where ρ is the population density and A_{exp} is the exposed area to the crash.

Thus, it is compulsory to have a fine resolution grid with the population density. However, this grid is not always available or it has too wide resolution to be used for UASs path planning over cities. So, it is requested to develop a methodology in order to create the population density layer autonomously without any need of third part data.

1.1.1 Population density estimation state of art

In the literature, there are several works whose aim is to quantify people distribution due to the fact that is a fundamental component of many decision making process, such as gauge future demand of food and water, evacuation planning, risk management and more.

As the processing image tools are improving rapidly with the usage of the convolutional neural network [3] [4], in the [5] authors proposed a deep learning approach to estimate population density from aerial images. The goal reached by this research group is to create a grid of predicted population density based on classification, with cell dimension of 0.01°x0.01° ($\approx$ 1105 m^2 at the equator). Since an important component of neural networks requires a "ground truth" to be used for input labeling purpose to train the network in order to reach a generalization of the problem to be able to predict also never seen data. Thus, the most fundamental part is that "ground truth" on which rely on, but unfortunately doesn't exist any official database. In this case the US Census was used that in turn provides an estimation of the population density.

Also the work in [6] proposed a deep learning approach that in this case is based on the numerical regression prediction, using census data of Tanzania and Kenya.

Both approaches observe the same issue such as the prediction on abandoned buildings, stadium and others misleading features within a aerial image.

1.2 Current work

The methodology proposed in this thesis is an Artificial Intelligent approach to estimate the population density over urban areas to compute the risk map for UASs risk-informed path planning. It is an upgrade of research group and it can be sketched as in Figure 1.2.

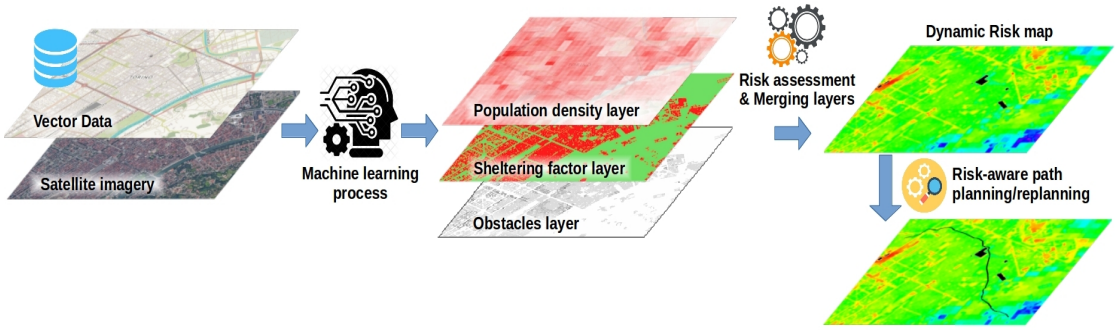

Figure 1.2: The main architecture of the proposed methodology

The aim is to estimate population density over specific areas to be used as input layer for the risk assessment. A deep learning approach was used due to the absence of an official population density distribution database for each country, so it was decide to develop a non-linear methodology to estimate it. As "ground truth" is required a tiny grid resolution, that's why a database containing a realistic estimation of the population density is used.

As output of the machine learning approach was followed a classification approach based on population density division for the images, as in the literature. Then in order to create the same layer that was used as input for the risk assessment input in the previous work, was followed also the numerical regression approach in which the exact value of population density was tried to predict.

1.3 Outline

The thesis is organized as follow. In chapter 2 the main theoretical aspects of neural network environment will be analyzed, whereas the chapter 3 deals with the software utilized for the development of the project. Chapter 4 will deploy the whole project development in details. Finally the chapter 5 discuss the obtained results and the further improvements.

Chapter 2

Introduction to Neural Network

In this chapter some basic concepts about Neural Networks (NNs) are explained (Figure 2.1). The basic idea behind a Neural Network is to simulate lots of densely interconnected brain cells inside a computer to provide is image processing, pattern recognizing or making decisions. The concept behind this simple but fundamentals idea is that if it works in nature, it should work artificially on a computer.

The first attempt towards neural network was born in 1943 by the neurophysiologist Warren McCulloch and mathematician Walter Pitts who wrote a paper on how brain neurons works. They modeled a simple neural networks through electrical circuits. In 1949, *The organization of behaviour* by Donald Hebb, reinforced the concept of the experience of neurons that creates strengthened pathways between linked neurons. Only in 1959 the researchers Bernard Widrow and Marcian Hoff of Stanford University create the first attempt of a neural network, called *Madaline*, applied on a real-world problem. The aim of this network was the reduction of phone line echoes through an adaptive filter. Moving forwards with the decades the theoretical comprehension behind neural networks is growing but the future of them lies only on hardware development. As with the human brain, the machine learning needs attempts, training or large example cases before it is able to make properly decisions, this could be done within reasonable amount of time only if the hardware is powerful enough. In the last years, thanks to the growth of computational power,researchers develop even more accurate Neural Networks. This encourage the use of machine learning on even more fields such as image recognition or system description for those that are not explainable by physics law [7] [8].

To conclude, the goal is to develop a neural network which is trained with a whole bunch of data which comes in the form of different inputs along with labels of what they are supposed to become. Once it has been trained, the NN has to be

able to perform good prediction also for data never seen before, generalizing what is beyond the input argument.

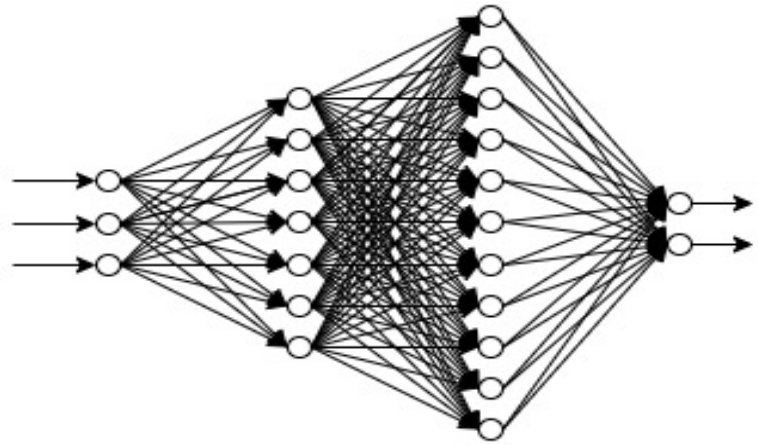

Figure 2.1: An example of fully connected Neural Network structure

2.1 Basic Neural Network structure

The most basic neural network, also known as *Multilayer perceptron* or fully connected layers, is composed by neurons which are split into different layers, those are of three types (Figure 2.2):

- *Input layer*;
- *Hidden layer*;
- *Output layer*.

There's no intrinsically difference between them excepting that the *Input layer* is directly linked to the *input* and the *Output layer* that is linked to the *output*, whereas the *Hidden layers* have connections only between other layers. From a programmer point a view, while the design of input and output layers is easily understandable, for the hidden layer is quite an *art of design* because there's no specific rules that define how many neurons per layer or how many hidden layers should be used to have the best performances. To solve this issue, researcher develop some heuristic way to design those layers but in most of the cases it's just a trade off between the number of layers against the time required to train the neural network.

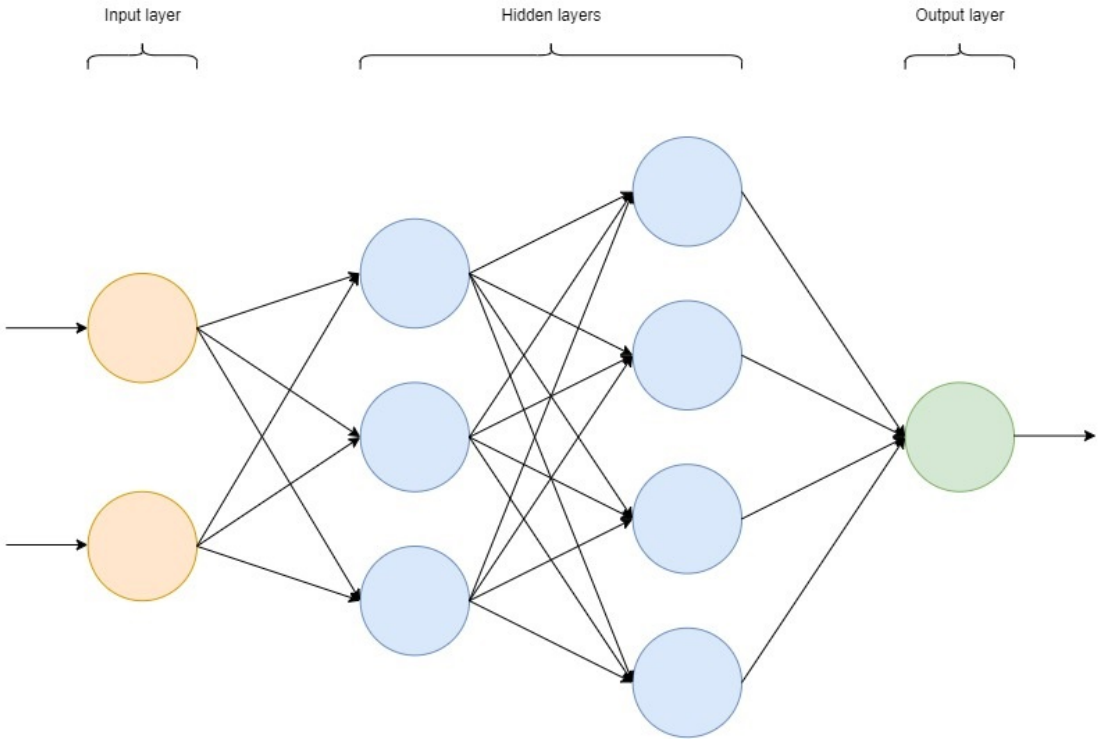

Figure 2.2: Fully connected Neural Network structure

2.1.1 Neural Networks Artificial Neurons

The core of each Neural Network are the artificial neurons. The first attempt to develop those neurons has been done in the fifties in the USA with the so called *perceptron*.

A *perceptron* [9] (Figure 2.3) takes one or more binary value as input to produce a single binary output value, so it is basically a container that holds a number, also known as *activation*.

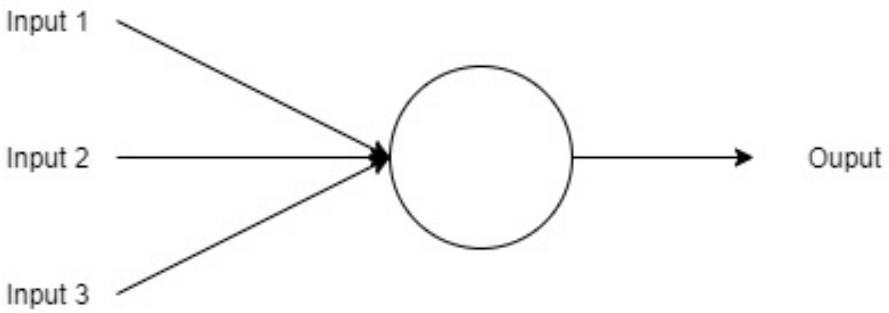

Figure 2.3: Perceptron (Artificial Neuron)

The way in which the output is produced is realized thanks to introduction of *weights* and *biases*. While the former expresses the relative importance of the *i-th* input to the output, the latter through the weighted sum of each input component decides if the output should be either 0 or 1, or in a more biological way, if neurons light up. That could be expressed in such algebraic way:

$$output = \begin{cases} 0 & \text{if } \sum_i w_i Input_i - b \leq 0 \\ 1 & \text{if } \sum_i w_i Input_i - b > 0 \end{cases} \tag{2.1}$$

Through this mathematical decision process output's main contribution are highlighted and, as a consequence, a decision is taken. It is important to highlight that each neuron is linked to all previous neurons layer, so the weights are kind of representation of how strength are that links. Therefore as the number of layer increases, every neuron takes a decision based on the output previously computed, so every step leads to a more abstract level therefore, more complex choices could be taken.

The perceptron could be seen in another fashion, *i.e.* as a NAND gate, if the right value of weights and biases are chosen (Figure 2.4).

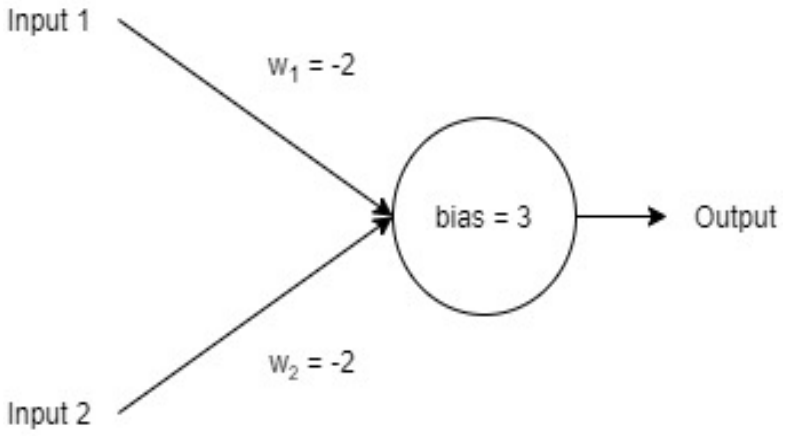

Figure 2.4: NAND gate

As could be easily seen from Table 2.1, only if both binary inputs are equal to 1 the output would be a 0, so exactly as a NAND gate.

As the computational power is growing fast over time, researcher understand that a neuron which has a binary output is strongly boundary due to the fact that even a small change for biases and weights could lead only to a flipped or fixed output, in other words only a deeply change is allowed. So using perceptron it's very difficult to perfectly adapt the neural network to different input datasets.

Input 1	Input 2	Output
0	0	$0 \cdot(-2) + 0 \cdot (-2) + 3 = 3 \Longrightarrow 1$
0	1	$0 \cdot(-2) + 1 \cdot (-2) + 3 = 1 \Longrightarrow 1$
1	0	$1 \cdot(-2) + 0 \cdot (-2) + 3 = 1 \Longrightarrow 1$
1	1	$1 \cdot(-2) + 1 \cdot (-2) + 3 = -1 \Longrightarrow 0$

Table 2.1: NAND logic

That's the reason why the *Sigmoid neurons* has been developed whose output is no more binary but corresponds to the *Sigmoid function* shown in Figure 2.5. From now, the neuron doesn't hold a binary number but a float one, included between 0 and 1.

$$output = \begin{cases} 0 & \text{if } \sigma(\sum_i w_i Input_i) - b \leq 0 \\ 1 & \text{if } \sigma(\sum_i w_i Input_i) - b > 0 \end{cases} \quad \text{with} \quad \sigma = \frac{1}{1 + e^{-x}} \tag{2.2}$$

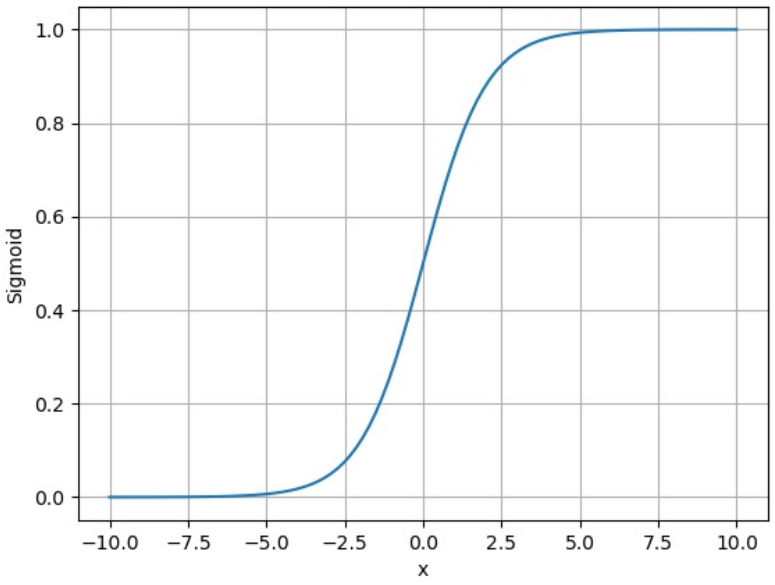

Figure 2.5: Sigmoid function

The *Sigmoid function* represents a different manner in which the neurons are *fired*, generally is called *activation function*. So the use of *Sigmoid neurons* provide a smoothed output that, given even a small variation for weights and biases, is able to produce a $\Delta output$ linear dependent with these variations.

Once the concept of different *activation function* was developed, in addition to *Sigmoid* others function was used in order to allow the model to create even more complex maps between inputs and outputs, that's essential if dealing with complex data such as images and videos. Figure 2.6 and Figure 2.7 show smoothed outputs of different of *activation function*.

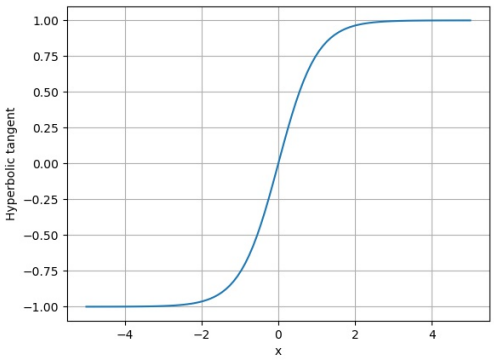

Figure 2.6: Hyperbolic tangent function

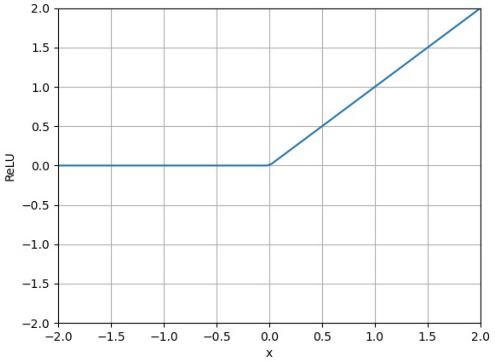

Figure 2.7: ReLU function

2.1.2 Descendent gradient

Until now, we have explained that, given an input, neurons light up through mathematical process that produces an output based on bias and weight values. Therefore, the relevant point is to get the best possible improvements with training data through an algorithm that autonomously adjusts all the weights and biases, considering that they are random initialized. So a *Cost function* is defined in order to quantify how well the training is going on, initially as parameter to be monitored as been chosen the MSE (*Mean Squared Error*).

$$C(w,b) = MSE(w,b) \equiv \frac{1}{2n}\sum \|y(x) - a\|^2 \tag{2.3}$$

where

- w represents all the weights in the network;

- b represents all the biases in the network;
- x is the input;
- y is the labeled output;
- a is the predicted output.

It can be easily understand that *C(w,b)* is a non-negative function, sum of non-negative terms, furthermore, more it becomes smaller, more accurate is the result.

So our goal is the achieve of $C(w,b) \approx 0$, then in other words minimizing that function through the *gradient descent*. It is an easy task if we would find the global minimum of a function *f(x,y)*, since it could be done analytical. Unfortunately, when talking about NN it's dealing with a function dependent on thousand and thousand of parameters, so even for the most powerful it could be freaky. Therefore, the alternative solution is the use of the *gradient* of the cost function:

$$\nabla C \equiv \Big(\frac{\delta C}{\delta v_1}, ..., \frac{\delta C}{\delta v_n}\Big) \tag{2.4}$$

In the Equation (2.4), v is a generic vector, representing weights and biases, from which the cost function depends. Moreover, it can be approximated that a variation on v causes a variation on C:

$$\Delta C \approx \nabla C \cdot \Delta v \tag{2.5}$$

From this point it's possible to choose Δv in order to have $\Delta C < 0$ in such this way:

$$\Delta v = -\eta \nabla C \tag{2.6}$$

where η is a small positive parameter, also known as *learning rate*. Replacing δv of Equation (2.7) with Equation (2.6):

$$\Delta C = -\eta \|\nabla C\|^2 \tag{2.7}$$

This guarantee that ΔC is non-positive so C will always decrease. Summing up, what happens is that every step the gradient according to (2.7) finds an even more minimum until it reaches a local minima, if lucky it could corresponds also to the global one. Another adjustment is to use a fixed variation step $\|\Delta v\| = \epsilon$, with $\epsilon > 0$, so what is done is trying to finding on which direction the cost function decreases more rapidly. This is the so called *stochastic descent gradient*. To be clear, it's not mandatory to use the a MSE as cost function but only a choice. For example also the MAE (*Mean Absolute Error*) could be used, or a cost function build *ad hoc* for a specific network.

2.1.3 Backpropagation

Backpropagation is an algorithm that messes up the computation of the *stochastic descent gradient* speeding up considerably the training time. The reason why this is done is for ensuring that changes that minimize the cost function involve all weights and biases of all layers all together. In other words, it's important to be focused on all the complete structure not to a specific layer or, worst, to a specific neuron.

Once an input is given to a NN, the output will be either the "answer" and "error", how far the predicted value is from the real one. This error is the basis to compute the gradient to adjust weights and biases. Instead of moving forward through the network as when it is given an input, the idea is to *back propagate* the error, or rather, to compute the error from the last layer moving backward until the first one is reached. Since the *backpropagation* is a very complex mathematical problem, below is presented a simplified way just to have a better understanding. Let's consider:

$$output = \sigma\Big(\sum w_i^l a_i^l - b\Big) \tag{2.8}$$

where:

- σ is the *activation function*;
- w_i^l is the weight of the *i-th* neuron of *l-th* layer;
- a_i^l is the activation value of the *i-th* neuron of *(l-1)-th* layer;
- b is the bias;

If the NN has 4 layers the output of the last will be equal to $\sigma\Big(\sum w_i^4 a_i^4 - b\Big)$ for the *i-th* neuron. The problem is that the predicted output is not exactly the real one so it must be change a bit. In order to modify it, b, w_i^4 or a_i^4 has to be changed. Focusing on a_i^4, this parameter is the input of *i-th* neuron of *l-th* layer but it can be seen as the output of *i-th* neuron of previous layer.

$$a_i^4 = output = \sigma\Big(\sum w_i^3 a_i^3 - b\Big) \tag{2.9}$$

Therefore if this activation value has to be modified, weights and biases of the last-second layer must be changed but also this output has an activation value that depends on the previous layer parameters and, so on, until the very first layer is reached. That's the concept behind the idea of the *backpropagation* of the error.

2.2 Introduction to Convolutional Neural Networks

A CNN (Convolutional Neural Network) is a specific deep learning algorithm which takes as input an image and it's able to underline the main objects, colors or patterns, so to differentiate one from another. It has the same structure of a common neural network but it is composed by specific layers that let in an more easily way to classify images. The whole structure was thought to better fit an image database through the usage of relevant filters, the core central part an image processing. The filters basically simplify images in such a way that the CNN could understand also the more sophisticated ones. It is composed by first layers whose aim is to extract the features, through layer such as *Convolutional* and *Pooling* which have specific connected neurons. Lastly there are the fully connected layers that provide the output.

2.2.1 Input image

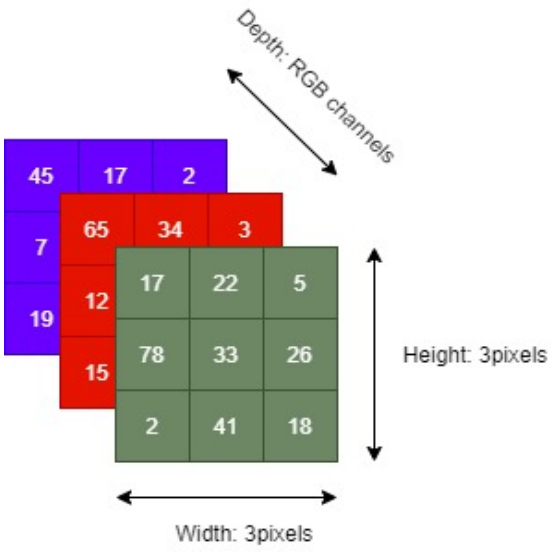

Figure 2.8: RGB input image matrices example

The input image (Figure 2.8) is a bi-dimensional matrix, for grey-scale images, or a three-dimensional matrix, for RGB (Red-Green-Blue) images. In Figure 2.8 is shown a 9 pixels image that represents a 3×3×3 matrix as input for the CNN, that looks like quite simple to process. However, considering a 4K RGB image there are 4096×2160 pixels that corresponds to 26542080 parameters. So the aim of the

CNN is to reduce those parameters in a such a way they are more easily processed but without loosing feature that may are critical to have a good prediction.

2.2.2 Convolutional 2D layer

As said in the above introduction, the aim of this layer is to extract, as mush as possible, high-level features through the *convolutional* operation which is based on *kernels*, or rather, *filters*.

Filters are simple matrices that are multiplied by submatrices of the image, they slide right till it parses all the image width and then hop down to the left beginning until the whole image is not covered. In Figure 2.9 a simple example is briefly presented.

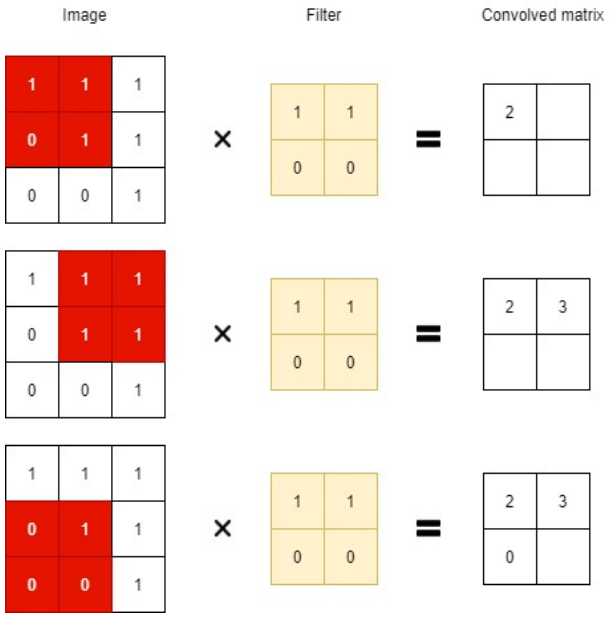

Figure 2.9: Convolutional operation example

As could be seen, the amount of cells that the filters slide to the right, or down, after each computation is equal to 1, so called *Non-Strided* kernel, but it could be increase as a project preference. Kernel has a depth equal to the image one, 1 for the greyscale images or 3 for the RGB ones. As the computational operation are finished, two ways for the output are presented, on one hand the dimension of the output is reduced with respect to the input one and on the other hand the filter operation doesn't affect the output dimension. In case of the former is the *Valid Padding* or *Same Padding* for the latter, this is done by the augmentation of the image size of the matrix by 1 with values equal to 0 (*i.e.* a 3x3 matrix turn into

a 4x4 matrix with the top and the left line composed by only 0 to not affect the image features.

A CNN doesn't have a single *Convolutional layer* but can have multiple ones, where, conventionally, the first one is used to extract basic features as edges, colors while the following ones try to extract at each step more high-level features in order to be able to have a wholesome understanding of the data.

2.2.3 Pooling 2D layer

After an image is convolved, the *Pooling layer* has the goal to reduce the spatial size of the matrix to decrease the computational effort to process the data. In addition, the *Pooling layer* is a features highlighter for the dominant ones and, as a consequence, a noise suppressant. For the *pooling* operation must be set the *filter* and the *Stride* value, that is how many cells has the filter to slide right or down. In this case the filter dimension, width and height, represent the size of the submatrix on which two possible operations can be done: the *Max Pooling*, the most used, or alternatively the *Average Pooling*. As the name suggests, for the first case the computed value is the maximum within the submatrix covered by the filter, while the second one computes not the maximum but the average value. Similar to the convolutional layer, the *pooling* operation affects the output dimension. However, the same concept of *Same Padding* and *Valid Padding* can be applied also for this layer. Figure 2.10 shows an example of pooling operation.

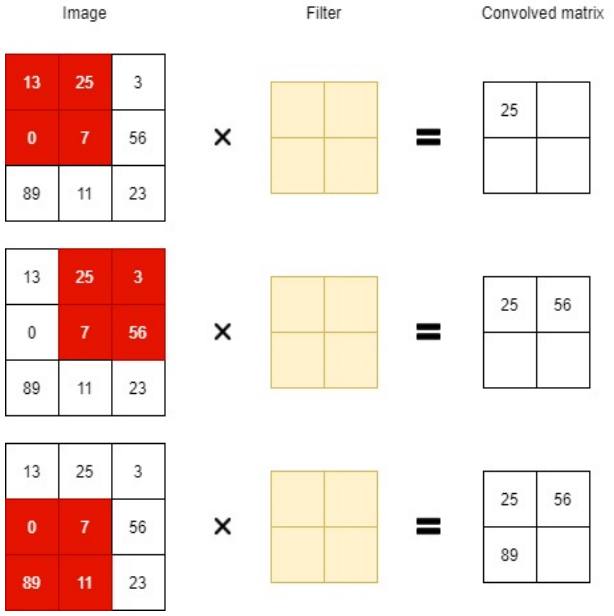

Figure 2.10: Max pooling operation example

2.2.4 Flatten layer

This layer takes as input the convolved and pooled maps in order to transform them into a single column vector that is attached to the fully connected layer which will provide the output. Flatting a tensor means to remove all the dimensions except for one. Thus, it takes the output and reshapes it in order to have a single column vector whose length is equal to number of elements contained by the output tensor.

2.2.5 Dense layer

The *Dense* layer is one of the most simple and used, it is the regular fully connected layer. It takes as input the previous results, from the *Flatten layer* if it is the first one, or, otherwise from another *Dense* layer, and after the below computation it produces an output, whose number is equal to number of neurons that it is composed by.

$$output = activation(dot(input, kernel) + bias) \tag{2.10}$$

where

- **activation** is the activation function of the single neurons, it could be *Sigmoid*, *ReLU*, *linear* or others;
- **kernel** is the weights vector;
- **dot()** is the *dot product* between the weights vector and input

The aim of this layer is to learn non-linear combinations of the high-level features extracted from the *convolutional* and *pooling* layers that finally leads to a classification operation of the input image. The higher the number of neurons, the higher the numbers of features that the layer is able to consider as essential to do a good prediction. This doesn't means that the neurons must be set as the higher value possible, considering the available computation power, because this could fall into *overfitting* or *underfitting*. Each Dense layer add to a CNN a number of parameters equal to the product of the number of input neurons and output neurons, exemplifying the number weights, plus the number of output neurons, exemplifying the number of biases.

2.2.6 Dropout layer

This layer doesn't affect the processed input but only the neurons. It is specifically used to simply the training while preventing overfitting. It set to 0 the input learning rate of a pre-set rate of neurons, that change every epoch. So it basically freeze weights and biases of randomly chosen neurons, with a specific rate that is a parameter of the layer to be set.

2.3 Transfer learning

Even though machine learning achieved even more better results, it still has some limitations in some specific scenario. The computational power, time spent to train, collection of sufficient amount of data could be a huge problems. That's why is used the transfer learning approach, following the generalization theory of transfer learning that says that transfer is the result of the generalization experience. Transfer learning is based on a Neural Network already trained by someone else for its specific task, and, then re-used as a starting point to develop another Network to provide another task. It's a technique that's made machine learning more easier to be performed. Researcher demonstrate that Neural Network trained to extract feature on a big amount of data, such as *ImageNet* with over 1.4 million images, are highly transferable to a high variety of recognition tasks. Obviously, the more two problems are related each other, the easier is that the Neural Network positively reacts to the new task. Once the Neural Network is compiled we have a network whose first part is completely trained for a similar task whereas the second part is built with new fully connected layers with random initialized values for weights and biases. As suggestion, the transferred Neural Network model has to be taken as it is. In other words, when starting train a new complete model, the aim is to fit weights and biases for the new layers while those of transferred layers has to be frozen, otherwise the whole concept of transfer learning is meaningful. As optional step, once the network reaches a good value of accuracy, the *fine-tuning* could be done which consists in unfreezing the complete model and re-train everything with a low level of learning rate. In this way, there's the possibility the re-adapt the pre-trained model to new data in order to reach a better accuracy in prediction.

The Figure 2.11 reports an example showing a graphically idea of how the *Transfer learning* works. Transfer learning is very useful also because there's no the exact point where the new NN must be linked to the the just trained one but it is a project decision. In order to take the best decision many considerations must be done, starting from the amount of data available till the computational power, or rather, the similarity between old and new tasks. It's important to keep in mind that generally first convolutional layers detect just simple features as edges, patterns, texture. Only the last layers are able to recognize high-level features. To sum up, transfer learning is perfect to optimize training time and also to reach better prediction results even if a small input data is used or low computational power is available.

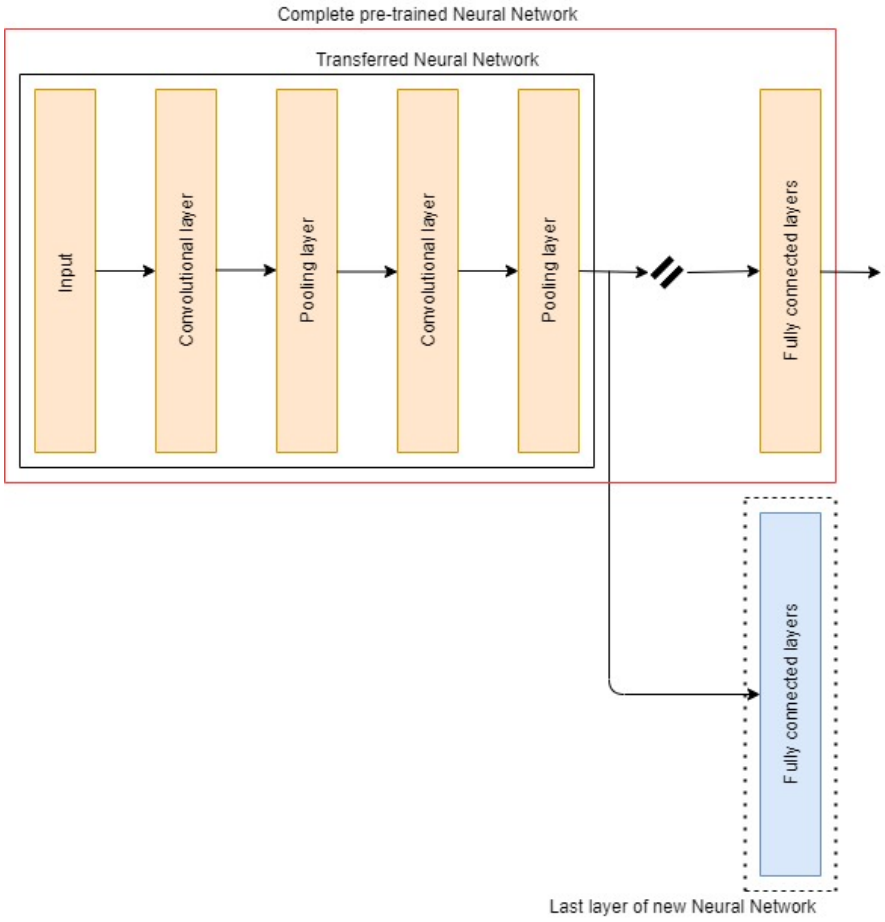

Figure 2.11: Transfer Learning example

Chapter 3

Software

In this chapter we will be focused on those essential software to develop a complete structure for a NNs. So not only the network structure is considered but also how it is tested after training or how the database is created. It is crucial to understand that every single part of the whole structure is fundamental. Someone can built a NN structure that perfectly adapt to do his task, minimizing the total number of neurons or choosing an *ad hoc* activation function and so on. However, if the input dataset is not well-finished, due to a not enough variety or quantity of the image, the train will fail, running up against *overfitting* or *underfitting*.

The whole project is developed using *Python*. Python it is an object-oriented, high-level programming language with a Rapid Application Development. Moreover, it supports lots of packages and modules, essential to connect different components building a complete structure.

This chapter will analyze those software, Python libraries, API or whatever has been used for programming, underlining the more useful aspects that involve the creation of a NN.

3.1 Bing Maps REST Services Application Programming Interface

It is a REST (*Representational State Transfer*) Services Application Programming Interfaces (API) developed by Microsoft[10]. A REST service is a software architectural style that provides interoperability between local computer script and Internet following a client-server approach. It allows to access and manipulate the resources provided by the API through a set of predefined *stateless* operations. Moreover, if the *HTTP* (*HyperText Transfer Protocol*) is used the *CRUD* (*Create Read Update Delete*) methods are exposed and, as a consequence, the action to be taken is defined by the *Uniform Resource Locator* (*URL*).

Using HTTP or HTTPS protocols, a REST Services API is accessible using the Python library *requests*. For more details go to section 3.4.1.

This API provides an easily way to create static maps that can be downloaded as images. A token is required for accessing to the API, but after a quickly registration it's possible to download a limited amount of image per (50.000 images per day). Thus, the services perfectly adapt to the need of create a consistent input database. The image is given as *response* of the HTTP **GET** request. So, in order to save it, a *for-cycle* has implemented to ensure that image data are decompressed, those data are saved in 128 byte chunks in binary mode file and then saved into a *jpg* format.

Bing Maps offers a very useful services because there are several parameters that could be set in order to choice the retrieved image exactly as wanted. Regarding those parameters, not all are compulsory, most of all are optional but this let the possibility to really customize images. As it is evident, one required parameter is to specify where is needed the static map, it can be specified in two different ways:

- *centerPoint* : the center point of the area that the image has to capture, it has to be provided with two numbers representing latitude and longitude. In addition also the *zoomLevel* has to be set. As the name suggest, it expresses how much the image is zoomed choosing from a value between 0 and 20(*i.e.* *zoomLevel* = 19 is equal to a 100x100 meters image). Unluckily, the documentation doesn't explain the correspondence between *zoomLevel* value and image width and height (previous example is got empirically);

- *BoundingBox* : this second way is very suitable if the wanted images must cover a specific area that it is specified through four parameter: south latitude, west longitude, north latitude, east longitude. These four parameters are the boundaries of the image. Therefore, it is the best way to obtain an image with a fixed a priori known dimensions.

Another important aspect is the choice of image characteristics that could be so heterogeneous. The variety of the choice includes the possibility to get image of:

- Street map;
- Aerial, an overhead view;
- Aerial with Labels, an overhead view with road and buildings in addition;
- Roads
- others;

Moreover, the resolution of the image can be set. Width can vary between 80 and 2000 while the height between 80 and 1500, that's quite a good range. Not

analyzing other features such as the pushpin on the map, possibility to set a route and several others.

Figure 3.1: Aerial image over Politecnico of Turin

The Figure 3.1 illustrate an image example that was used as database to train the Network. It is an image with 250×250 meters of dimension fixed by the *BoundingBox*, *Aerial with labels* type, 800×800 pixels as resolution. All of this parameters can be seen from the URL used for the request:

```
https://dev.virtualearth.net/REST/V1/Imagery/Map/Aerial?
mapArea=45.060983,7.657832,45.06324886018748,7.658736209994505&
mapLayer=Basemap,Buildings&imagerySet=AerialWithLabels&mapSize:
800,800&key=token
```

3.2 TensorFlow

TensorFlow is an open source end-to-end platform for machine learning initially developed by Google. Embedded and mobile systems support TensorFlow, it runs on every GPUs, CPUs and also on TPUs (Tensor Processing Units), a specialized hardware to perform math tensor.

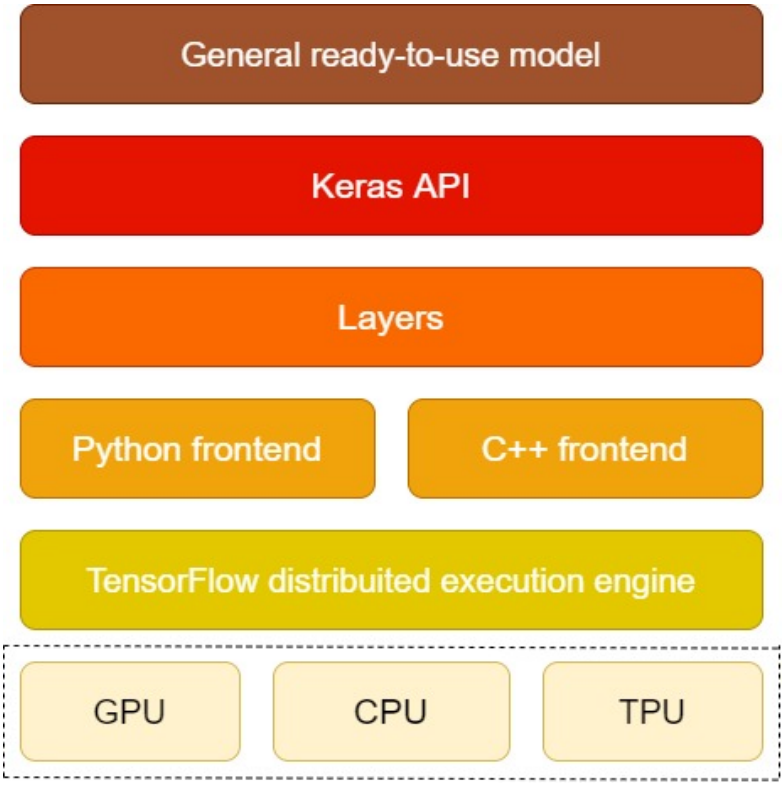

Figure 3.2: TensorFlow environment structure

The Figure 3.2 simply explain where TensorFlow acts in a machine learning environment. So starting from the low level hardware reaching more complex software, it is composed by:

- *hardware*;
- *distributed execution engine*, it abstracts away the hardware with distributed execution, based on runtime information that basically manages the data flow with the best efficiency;
- *User frontend*, programming language used by the programmer to interface with TensorFlow;

- provides common *layers* used for built a NN;
- it offers also high-level API, such as *Keras* (see 3.2.1)
- provides also trained model to be used for transfer learning.

3.2.1 Keras

Keras [11] is the high-level API of TensorFlow, an interface for developing deep learning problems using the advantages of TensorFlow platform. It provides essential abstractions and building blocks for elaborating machine learning solutions. In other words, Tensorflow provides the environment on which compute the gradient descent while Keras API provides tools to really built on your own the Neural Network. In addition to the model creation, this API offers all the needed tools to the machine learning workflow starting from the data processing and training functions until output deployments. Thanks to the easy-to-use API, NNs all around the world get an enormous growth in the machine learning field due to the not difficult way into the model can be built. Instead of coding with hundreds and hundreds lines to implement just a single layer, Keras offers the way to create a layer just re-calling it and setting within the function some parameters, such as padding type, stride, filter size, output shape and so on. In addition, it holds out the available open source pre-trained NNs. To sum up, it really extends the possibility to develop a machine learning approach to everyone.

3.2.2 VGG 16

VGG 16[3] is a CNN developed by Karen Simonyan and Andrew Zisserman during the yearly ImageNet Large Scale Visual Recognition Challenge (ILSVRC)[12] organized by ImageNet on 2014. This CNN is the one used as basis of the model developed for this thesis goal. So, this section briefly explains how it was thought and developed.

Imagenet is an online database of image whose purpose is only to create this dataset to be used by researcher to develop even more complex algorithm to recognize, manipulate, or whatever, multimedia data. This online database it's not the owner of the image but it provides URL to be utilized for educational and research purposes, the philosophy is *"good research needs good resource"*[13]. So, during this competition the best researcher works on challenge to better classify over 1000 classes the image dataset.

The VGG research group ends the competition with the first and the second place in the localisation and classification tracks respectively and made the the two best-performing models publicly available to encourage further research on image processing.

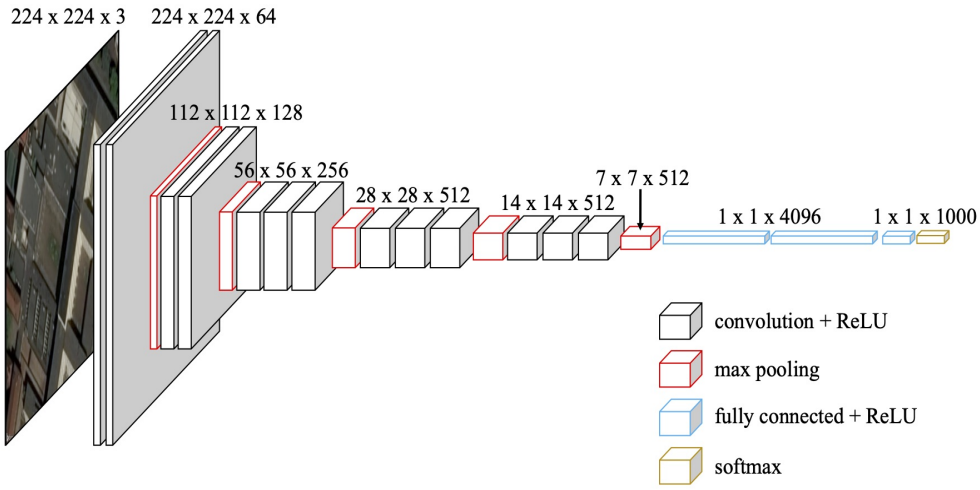

Figure 3.3: VGG 16 architecture

Figure 3.3 shows the architecture of the Convolutional Neural Network model. The compliant input of the model is an RGB image of 224×224 pixels. The whole architecture consists on 16 weight layers implemented in the following way:

- 2 Convolutional2D layer with 64 output filters;
- MaxPooling layer;
- 2 Convolutional2D layer with 128 output filters;
- MaxPooling layer;
- 3 Convolutional2D layer with 256 output filters;
- MaxPooling layer;
- 3 Convolutional2D layer with 512 output filters;
- MaxPooling layer;
- 3 Convolutional2D layer with 512 output filters;
- MaxPooling layer;

- Flatten layer;
- Dense layer with 4096 neurons;
- Dense layer with 4096 neurons;
- Dense layer with 1000 neurons;
- Softmax layer;

Differently from previous network models, the peculiarity of this one is its depth (16 layer) more than the other netowrks, feasible due to the usage of 3×3 filters into the convolutional layer. Such filters are with the minimum size to recognize and understand the notion of left/right or up/down. The Convolutional layers have stride value equal to 1 with the *same padding* option that doesn't affect the spatial domain of the matrix.The MaxPooling layer is performed with a 2×2 filter and a stride equal to 2, its task is to reduce the spatial domain of the image matrix. Finally the last fully connected layers, that ends with a Softmax activation function that helps to better classify image.

Just to highlight how much ImageNet and researchers work are important, this VGG 16 Neural Network model was trained for three weeks with four NVIDIA Titan Black GPUs, everyone costs about 2.500 € each. Therefore, it was a huge job because not everyone can have this hardware.

3.3 Scikit-learn

Scikit-learn is an open source library for machine learning[14] that provides tools for data preprocessing or model evaluation.

3.3.1 Test

This section is focused on the output of the NN. As just said, the whole NN aspects are fundamentals even the output and also how to analyze it to understand if there's something that is went wrong. The output let the developer understand if the model is[15]:

- *underfitting* (Figure 3.4), the model is not able to recognize the main pattern maybe due to a too small database or training time;

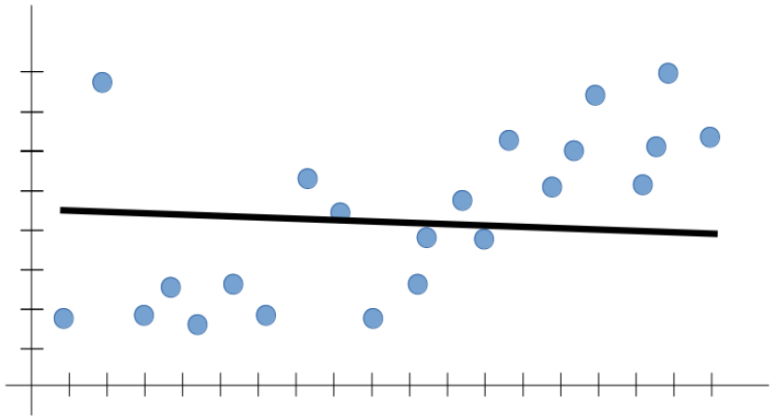

Figure 3.4: Underfitting example

- *overfitting* (Figure 3.5), happens when a model is too trained on a database. So, on one hand it reach an overall cost function value very tiny, on the other hand the model is not able to generalize the main pattern but only minor ones that almost all are present only on traning database. To sum up, the model become almost perfect to predict the training data but it's not able to predict those one that it never seen before.

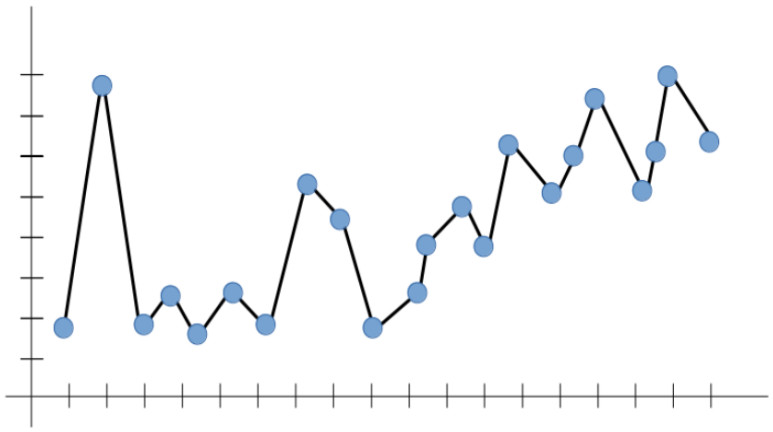

Figure 3.5: Overfitting example

To be specified that exist two ways to choose the output of a NN:

- *Classification*, in which a certain input is labeled into a specific class represented by a neuron on the last layer (at least 2 classes are needed but for VGG 16 the classification is over 1000);

- *Numerical regression*, in which a certain input as represented by a single number (integer, float) on the neuron output. So the NN is forced to manipulate data in order to find that function that generalize the dataset.

Below sketched, some methods provided by *Scikit-learn* to judge how far the model is from a good generalization of the problem in order to do those adjustments that improve model performance. If the model is well-trained those functions are just a demonstration.

Confusion matrix

A *confusion matrix* is a table used for classification problem on machine learning, it helps to visualize the performance of a NN[16].

		predicted value			
		Class 1	Class 2	Class 3	Class 4
true value	Class 1	27	3	55	31
	Class 2	23	2	64	33
	Class 3	22	3	50	41
	Class 4	7	0	37	82

Table 3.1: Confusion matrix

Table 3.1 is representing a real example done on a trained NN with never seen images. As labels indicate, rows represents the ground truth, while columns the predicted value from the NN. In an ideal condition the matrix would be diagonal that means that the NN is able to correct predict all the images given as input, but in real life it's quite impossible to achieve. This table is useful not only for hoping that is diagonal but also too see how the error is done. To explain it with an example, if studying the first column of value it can be seen that there are 79 $(27 + 23 + 22 + 7)$ predicted to belong to first class but only 27 really belong to to that class. Let's focusing on the prediction error, it can be noticed that only 7 values are wrongly predicted to fourth class that's a good value with respect to the 23 and 22 errors respectively from second and third class. Therefore, can be figured out that the maybe either the model is not sufficiently trained, or more probably, that the database has quite similar images that belong to first, second and third class. To sum up, this table let understand not only how good the model is, but also gives an idea of which direction the work has to be changed in order to improve performance.

Classification report

This report is a numerical way to check how good the model is. It is based on the confusion matrix, that it computes itself, to calculate those scores that gives a better idea on how is reacting the model to the training[17].

This report provides:

- *precision*, is the ability of the model to classify an image belonging to a class that actually belongs to a different one, $\frac{\text{right predicted values belonging to class } n}{\text{all predicted values belonging to class } n}$;
- *recall*, is the ability of the model to classify an image belonging to a class that actually belongs to that one, $\frac{\text{right predicted values belonging to class } n}{\text{true values belonging to class } n}$;
- *f1-score*, is a weighted harmonic mean of *precision* and *recall*, equal to $2\frac{\text{recall}\cdot\text{precision}}{\text{recall}+\text{precision}}$, where the 1 is the best score, while 0 is the worst. It is not a score that gives an evaluation of a CNN training but rather is used to compare two different neural networks results;
- *support*, is the number of total instances of a specific class that are collected within the test dataset.

	precision	recall	f1-score	support
Class 1	0.34	0.23	0.28	116
Class 2	0.25	0.02	0.03	122
Class 3	0.24	0.43	0.31	116
Class 4	0.44	0.65	0.52	126
accuracy			0.34	480
macro average	0.32	0.33	0.29	480
weighted average	0.32	0.34	0.29	480

Table 3.2: Classification report

Table 3.2 shows an example of classification report, directly linked to the confusion matrix previously explained.

Coefficient of determination

Widely used in statistics, the *coefficient of determination*, or R^2, is the proportion between the variance in the dependent variables, the predicted ones, with respect to independent ones. On machine learning is used when a NN is develop on numerical

regression, or rather, to predict an exact value not to classify. Thus, it measures how far are the prediction from the ground truth in such this way:

$$R^2 = 1 - \frac{\sum_i (y_i^{predicted} - \bar{y})^2}{\sum_i (y_i^{predicted} - y_i^{true})^2} \tag{3.1}$$

where $\bar{y}$ is the variance.

3.4 Python packages

Python is one of the most popular programming language nowadays, since it has a simpler syntax, is faster and there are a huge variety of libraries. Below the main used in this thesis are explained.

3.4.1 Requests

It a simple and elegant Python library. It allows to send HTTP 1.1 requests in a very easily way. So, one hand there is the Bing Maps Services API that offers an interface as a server, on the other hand there is the Python library request that let the implementation as a client. Practically, it allows to send those specific HTTP requests to a server that if the request has been done successfully it gives back the image as response. For thesis purpose, only the *get* method was implemented, it can be used only to retrieve data that what is needed. Moreover, a very useful stuff is that with the *requests* library there's no need to manually add the query to the URL, simplifying the creation of the complete URL.

```
{
    "query": {
        "mapLayer": "Basemap,Buildings",
        "imagerySet": "AerialWithLabels",
        "mapSize": "224,224",
        "key": "token"
    }
}
```

Figure 3.6: Request query example

Figure 3.6 shows a query example used in the code. The script reads this query from a *.txt* file and just adds it as a parameter of the function *get*. So from the user point of view, it's very practical that just changing few parameters in a file there's the possibility to retrieve a different image. Finally, to complete the URL, the running script computes every step the necessary GEO reference. Just in case, was developed both method to catch images either the one with only the center point or the one with the boundary area.

3.4.2 Pyproj

Pyproj is a Python interface to *PROJ* that is a cartographic projections and coordinate transformations library. This library implements the forward and inverse geodetic computation. Let's focus on the latter. This one involves the possibility to determine the forward and back azimuths and in addition also the distance given the latitudes and longitudes of initial and end point. This computation are based on the *World Geodetic System 1984* (WGS84), it a geodetic coordinates system based on a reference ellipsoid realized in 1984. This method is to one used by the GPS. Therefore, this library is used both for:

- in case the GEO references of all the images are given, a method was implemented to computes the distance between two adjacent images to avoid overlapping;

- in case is requested a database over a specific city given a starting point, a method was implemented to autonomously compute all the GEO references for images based on retention of distances between images, as before to avoid overlapping.

3.4.3 Numpy

Numpy is fundamental Python library for scientific computing, it provides a wide variety of operation that could be done on arrays such as algebraic, shape manipulation and so on. Python doesn't have arrays but only lists, the latter has a fixed dimension while the former grows dynamically. Vectorization is the absence of any explicit loop, that is what *Numpy* basically do. This concept is useful either to have less code, that means less bug, and a faster computation. Considering that a NN is trained with a huge image database, where each image is a matrix collected within an input vector, it's clear why this library is fundamental for thesis purpose.

3.4.4 Matplotlib

As the name suggest, *Matplotlib* is a library whose aim is to emulate the Matlab graphical commands using Python, but it is completely separated from Matlab. Used to plot training value over time or also to have a graphical representation for both predicted output over real one or how the input data are distributed.

Chapter 4

Project development

This chapter analyzes how the project was developed step by step, covering all the relevant aspects that lead to final results. It starts with the first approach to machine learning with TensorFlow and Keras and then continue going deeper into the neural networks environment. The goal to achieve is to built and train a Convolutional Neural Network that is able to estimate the population density over urban areas. The thesis work is based on the image processing concept in fact the estimation has to be reached given as input an aerial image that is then processed by the neural network to achieve the goal.

The development can be divided into three different phases:

- a first approach to the neural network environment with a bi-class classification problem;
- a second step that consists in a multi-class classification problem based on population density;
- a last step in which was built a CNN to directly estimate the population density value through numerical regression.

4.1 Bi-class classification problem

In this first phase of the project the NN environment was approached just to develop a network that is able to classify aerial images into two different ranges, that are:

- city;
- country.

Also for this first attempt is used the *VGG 16* as base model for the complete Neural Network structure. Considering that this is the first trial, the *VGG 16* was used as just the researchers train it without taking out any layer, moreover all the weights and biases are fixed, in other words they are specifically set as not trainable parameters. This choice has a limitations that is that the input images must respect a resolution size, the one that was used to firstly train this network, a 224×224 pixels RGB images. To this base model were added others *Dense* layer in order to reduce the number of output neurons. It has to be moved from 1000 into 2, so only two neurons that predict the output. This means that there is a neuron that lights up when the input image belongs to *country* class and another neuron which instead lights up when the input can be associated to *city* class.

The first attempt is introduced below highlighting all the steps that lead to the final trained NN. Moreover, all the set parameters are briefly explained.

As a starting point, the input database of aerial images was created and, as said before, it is fundamental if wanted a NN strong enough to generalize this problem. Therefore, images are not captured casually but they are chosen and everyone was checked in order to avoid those ones that could be misleading. Errors due to deceived images are frequently, it's reasonable because for example an image taken over a city could catch a park or a stadium that could lead to misunderstandings. Photos cover an area of 250×250 meters. So to create this database, images belonging to *country* class are quite simple to find out, on the contrary images belonging to *city* are a little bit more difficult to identify. That's why to collect images from the latter class are picked from several, looking for those ones that clearly represents city avoiding those which contain big tree-lined avenue, car parking, stadium and so on. In Figure 4.1 and Figure 4.2 are shown an image example belonging to each class.

A script autonomously collects images within an infinite *while-cycle* given as input to cover an area over three big city: Rome, Milan and Turin. The request to the Bing Maps API needs the 4 parameters (south latitude, west longitude, north latitude, east longitude) in order to use the *boundingBox*, since a fixed area for each cell is compulsory. So, the script starts from the left-down corner (south latitude and west longitude of the whole covered area are the GEO references of this starting point), it computes the coordinates for an area of 250×250 meters and then requests the image from Bing Maps API. Once it is downloaded and saved, longitude is increased so it slides right to the next cell and the process is repeated until the last point is reached, with coordinates south latitude and east longitude. Once the first line is done, the scripts compute the latitude in order to hop up to the next line of cells to be saved. This is done autonomously until the cell with GEO reference north latitude and west longitude, or even a little bit greater, is caught. This downloading flow for the images is presented in Figure 4.3.

Figure 4.1: Example of an aerial image classified in the City class

Figure 4.2: Example of an aerial image classified in the Country class

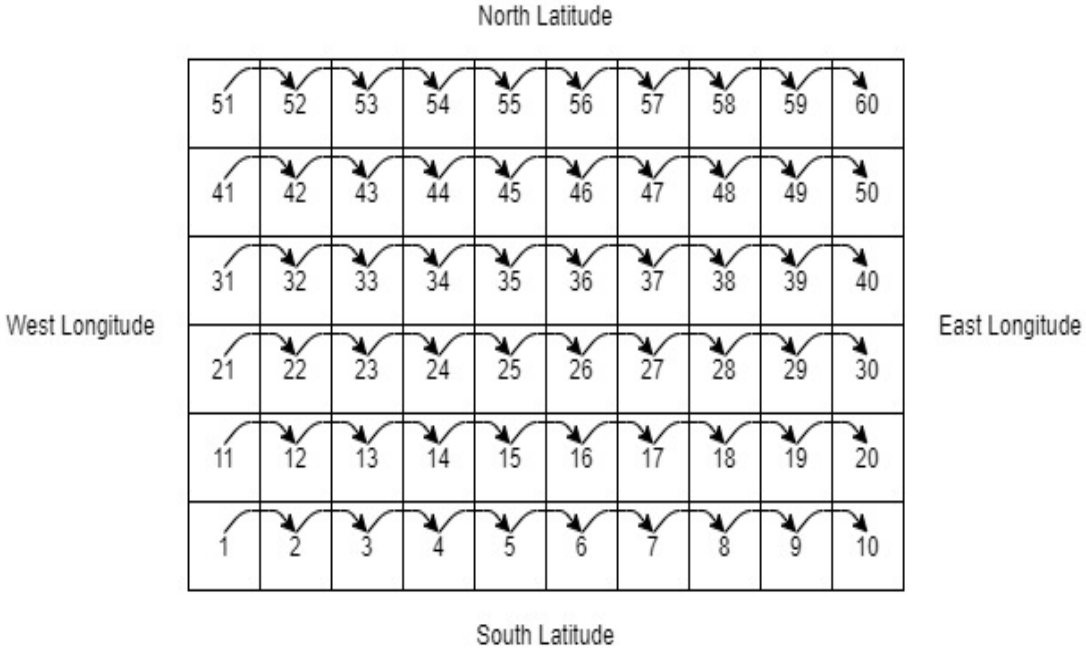

Figure 4.3: Graphical represetation of the order used to download the aerial images of a specific large area

After that an enough big database is created, images are looked over to find 500 belonging to both classes, city and country. Below are two example of images chosen to belong to that classification.

Once the images has been picked, the core script starts. It concerns the database loading, the neural network creation, training and evaluation.

Firstly there is the database generation method, where downloaded images are transformed in a usable way for the neural network, it is done through specific functions provided by *Keras*. Every single photo of the database follows this flow:

- *load_img()*, the image is loaded as a PIL (*Python Imaging Library*) instance;
- *img_to_array()*, the image is transformed into a (224,224,3) matrix, a three-dimensional matrix where each pixel has its specific between 0 and 255 for all three Red-Green-Blue channels;
- the matrix is labeled with the right belonging class, that is 0 and 1 respectively for country and city class;
- matrix and label are appended to the complete vectors respectively for the input and output of the NN;

At the end of this process, two different vectors representing the input image and the directly linked labeled output. So, the input is a three-dimensional matrix where the *i-th* image in *i-th* position is identified by the label in *i-th* position of the output vector. To fit the model in the next step, this two vectors are split in half, 250 and 250, in order to create:

- *training set*, those inputs used to train the network, in other words, used to compute adjustments to do on weights and biases;
- *validation set*, those inputs used while training to check if the network is not overfitting on *training set*. Therefore, they are used to confirm that the adjustments proposed after a training epoch are for a generalization of the problem.

The further step is the creation and training of the Neural Network. Firstly the *VGG 16* model is loaded and the new fully connected layers are attached to it. As just said, the aim is to reduce the number of the outputs until 2. This opportunity is offered by the *Dense* layer in which could be set the parameter for the output shape. Hardly, the NN get a good prediction value if the output is moved from 1000 to 2 in one single step, that's way some intermediate step are not compulsory but recommended.

It's chosen to put three fully connected layers in such this way:

- *Dense*, with 256 neurons output;
- *Dense*, with 64 neurons output;
- *Dense*, with 2 neurons output.

The final configuration, VGG 16 plus the new layers, was plotted through the specific function *plot_model*, provided by Keras. It is shown in the Figure 4.4. The "?" concerns the amount of items that the neural network has to manage, therefore, it is unknown a priori.

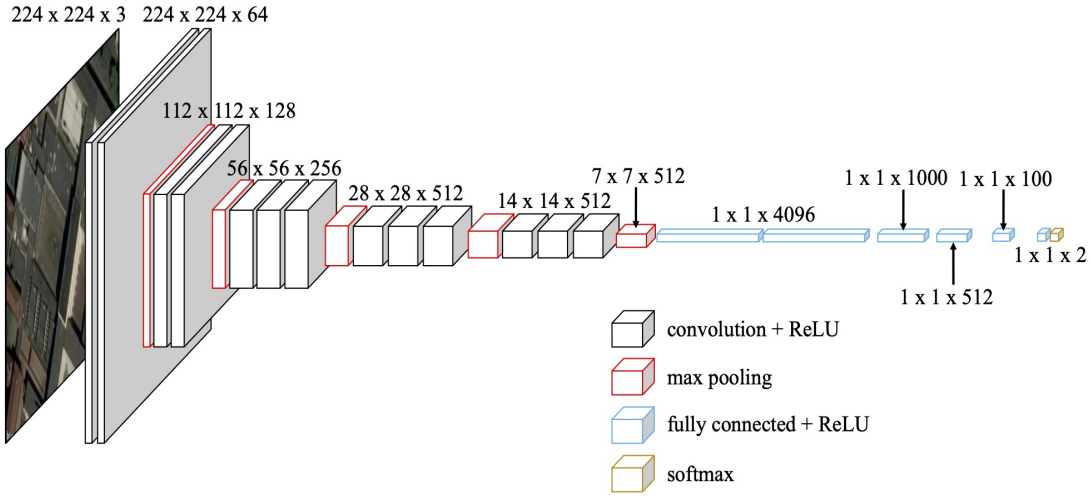

Figure 4.4: Test 1 - Convolutional Neural Network configuration

Weights and biases of the new layer are randomly initialized with a uniform distribution.

Successively the real training part starts, it is implemented thanks to the *compile*() and $fit()$ functions provided by Keras API. The former configures the NN for the training, specifying:

- *loss*, the loss function to be minimized, also known as cost function. The *categoricalcrossentropy* was chosen;
- *optimizer*, algorithm that implements the gradient descendent. The SGD (*Stochastic Gradient Descent*) was chosen;

- *metrics*, one or more parameters used to judge how good the training is. The *accuracy* was chosen.

Instead, the $fit()$ is the function that really starts the train, also this one need some input arguments to be specified:

- training and validation sets, both image input and labeled output;
- *epochs*, how many time the entire dataset is used to train the model, basically how long is the training section. 100 epochs was used;
- *batch_size*, usually the whole database is not given to the model as a unique bunch of data but it is subdivided into smaller batches, the size is specified by this parameter. There's no mathematical rules to fix it but as rule of thumb it is chosen to be a 2^x. In addition, it depends on the database size because a too small size involves overfitting while a too wide size involves underfitting. A value equal to 32 was chosen;
- *callbacks*, the possibility of monitoring one or more parameters to determine if a train has to be stopped due to a no more increasing performances of the model. As an example a callback can monitor the loss function and epoch after epoch check if there's an improvement on it. Moreover, this callback check if is not exceeded a pre-fixed number of epochs, known as (*patience*), without any increase on performances. If the threshold of epochs is overpass, the callback stops the training and save the model with the best result, not the one of the last epoch. Used because when setting the training span through epochs, it's unknown if the model needs all of them, less or more. So in case it needs less epochs, it is very useful to avoid both overfitting and waste of time.In both training and validation sets we have decided to monitor accuracy and loss function.

The final step is the model evaluation through tests to check out if the model is able to correctly classify never seen images among *country* or *city* classes.

Table 4.1 (at the end of the section) shows the results of the test done on the first trained neural network. It shows where the NN classify the input image and which is the probability related to that classification. It is done on 20 images, 10 for each class, that the trained neural netowrk has never seen before. Immediately, it can be noticed that the NN is not well-trained due to the fact that all the probabilities are around the 50% with a peak on 53.44%. Therefore, is not important that maybe some of them are correctly predicted because the classification probabilities highlight the uncertainty of the neural network. As an example, if looking at *Image 9* for the NN it belongs to country class with a probability of 51.98% that can be seen also that for the NN there's a probability of 48.02% to belong to city class.

This firstly result are a classical problem of underfitting. This problem can be provoked by:

- too low training span, NN doesn't have the physical time to reach the minimum of the cost function;
- too small input dataset, NN doesn't have enough data to find the main pattern that better generalize the problem;
- too much neurons added with the Dense layers, the training has to modify too much weights and biases and it cannot with this database size or training span;
- too few neurons added with the Dense layers, the number of weights and biases are not sufficient to have a better generalization of the problem.

So after that results, was decided to increase the database dimension was doubled, from 500 images to 1000 images while all the other parameters are left as for the first test.

As can be seen from Table 4.2 there is an improvement on prediction accuracy. Values are still around the 50% but there is a generally grows up, in fact the probability peak is 59.59%, higher with respect to the 53.44% of the previous test. Since was achieved better performance just expanding the input dimension, it was chosen to double again the size of the database, from 1000 to 2000 images. As before, the other parameters are fixed.

Table 4.3 shows the test results done where the neural network classification has quite perfect results. The worst case of prediction is 86.87%. To be noticed that there are 14 over 21 images with an error on prediction less than 1% and the average accuracy value of prediction is 97.79%. Moreover, there's only one error on prediction.

Figure 4.5 shows the evolution over epochs of the accuracy and the loss function of this last test. As can be seen, the accuracy reaches quite perfect values, around 97%. Moreover, the trend of the curve follows the correct behaviour: a rapid increase during the firstly epochs, due to random initialization of weights and biases, followed by a little growth of both parameters, the search of the global minimum and the accuracy. Both the curves follow this shape, train line and test line (validation set).

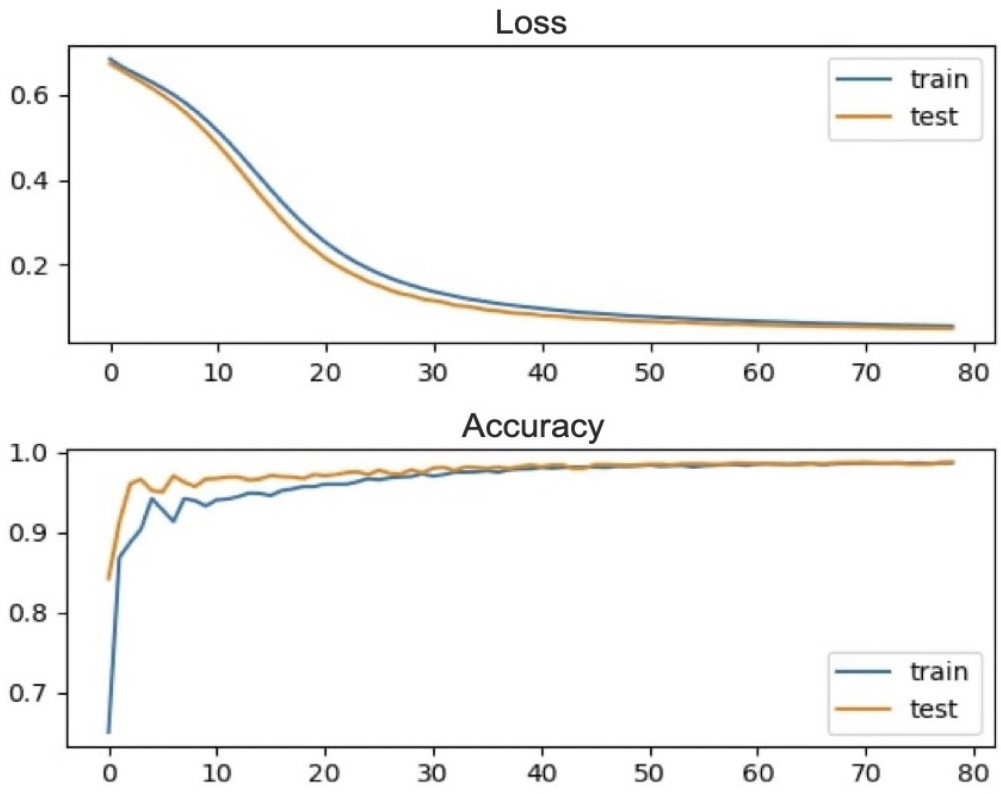

Figure 4.5: Test 3 - Loss and accuracy trend

To sum up, the last one trained CNN has an input database enough big to get quite perfect values on prediction among which class belong images, that are selected one by one. So trying to achieve better results it's possible but instead of perfectly fine-tune a CNN on this kind of problem, has been chosen to go ahead into a more complex classification problem.

Image	True class	Predicted class	Prediction probability
1	Country	Country	51.69%
2	Country	Country	50.26%
3	Country	Country	52.79%
4	Country	Country	52.19%
5	Country	City	50.66%
6	Country	Country	52.16%
7	Country	Country	52.24%
8	Country	Country	51.80%
9	Country	Country	51.98%
10	Country	Country	52.22%
11	City	City	52.89%
12	City	City	53.23%
13	City	Country	51.99%
14	City	City	52.83%
15	City	City	53.44%
16	City	City	52.37%
17	City	Country	50.70%
18	City	Country	50.10%
19	City	Country	50.62%
20	City	Country	50.28%

Table 4.1: Test 1 - Classification results

Image	True class	Predicted class	Prediction probability
1	Country	Country	52.49%
2	Country	Country	50.43%
3	Country	Country	52.93%
4	Country	Country	53.98%
5	Country	City	50.22%
6	Country	Country	52.21%
7	Country	Country	53.74%
8	Country	Country	53.69%
9	Country	Country	53.16%
10	Country	Country	52.37%
11	City	City	56.81%
12	City	City	58.08%
13	City	Country	53.29%
14	City	City	55.94%
15	City	City	59.59%
16	City	City	54.79%
17	City	Country	50.58%
18	City	Country	51.63%
19	City	Country	50.76%
20	City	Country	51.43%

Table 4.2: Test 2 - Classification results

Image	True class	Predicted class	Prediction probability
1	Country	Country	99.70%
2	Country	Country	93.45%
3	Country	Country	99.94%
4	Country	Country	99.86%
5	Country	Country	95.95%
6	Country	Country	98.94%
7	Country	Country	99.90%
8	Country	Country	99.62%
9	Country	Country	93.86%
10	Country	Country	99.68%
11	City	City	99.83%
12	City	City	99.89%
13	City	Country	99.25%
14	City	City	99.59%
15	City	City	99.93%
16	City	City	99.87%
17	City	City	90.47%
18	City	City	86.87%
19	City	City	99.69%
20	City	City	99.52%

Table 4.3: Test 3 - Classification results

4.2 Multi-class classification problem

A multi-class problem deals with classification where the images are to be collocated into more than two ranges. In this case we decided to classify into 4 ranges. The structure of the network could be quite similar but the learning problem grows up exponentially. Therefore, this kind of problem cannot be approached as the one in the previous section. In other words, all the input images for training cannot be selected in a human way but is needed a numerical *ground truth* on which rely on, to be used for labeling purpose. To solve this issue, was used a database which contain an estimation of the population density for 2018 100×100 meters cells over urban area. Moreover, connected to this value of population density is linked the GEO coordinates of the center point of each cell. Firstly, this fundamental database was used to download the images for the training input vector.

Once the database is downloaded, it has to be linked with an output. As first attempt, we decided to equally distribute range limits among the minimum and maximum value of the population density within the dataset. Thus, each image as its own population density value that directly link this image to a class label, where the first class presents the lowest population density values whereas the fourth the highest values. The database was then split into three categories:

- *training set*, same purpose as the previous section;
- *validation set*, same purpose as the previous section;
- *test set*, images used after a CNN is completely trained to test it, computing the confusion matrix and classification report.

Divided in such this way: 1020 images was for *training set* whereas 548 was for the *validation test*, proportioned respectively as 65% and 35%. The remaining 450 were used for the *test set*.

Dataset was used to train a neural network with the same structure as the last test developed in the previous section. So, the complete transferred model plus three fully connected layers with respectively 256, 64 and 4 neurons as output. Only the last layer was changed, where each output neuron is designate to represent one class among the four possible choices, due to the new classification problem.

The test results are shown in Table 4.4 and Table 4.5. Clearly both tables show up that the training was not successfully done. Especially the support column of the classification report highlights that this range division is not appropriate because the input images are not equally distributed among the four ranges. The majority part was condensed into a single class so reasonably also the training images are not well-divided. Thus, this kind of division leads to an underfitting problem for those data belonging to Class 1 and 4 while images belonging to Class

		predicted value			
		Class 1	Class 2	Class 3	Class 4
true value	Class 1	0	0	12	5
	Class 2	0	0	173	168
	Class 3	0	0	35	47
	Class 4	0	0	3	7

Table 4.4: Test 1 - Confusion matrix results

	precision	recall	f1-score	support
Class 1	0.00	0.00	0.00	17
Class 2	0.00	0.00	0.00	341
Class 3	0.16	0.43	0.23	82
Class 4	0.03	0.70	0.06	10
accuracy			0.09	450
macro average	0.05	0.28	0.07	450
weighted average	0.03	0.09	0.04	450

Table 4.5: Test 1 - Classification report results

2 are overfit. Therefore, a database structured in such this way will lead to a failed training for sure, it must be modified.

To solve this issue, a new arrangement for the range limit to it are attached to better divide input data into the the four classes. In this case, the attention was focused not on the value of population density itself but to divide images in such a way that to the same quantity belongs to each class, net of small delta. Moreover, a random shuffling was applied to the database before dividing it into the three sets, training, validation and test. It is shuffled either to avoid possibility to give as input sequential images and to assign to each set an equally distribution of data belonging to each class, net of small delta. Practically, the input vector is shuffled once if the equally distribution is still holding after the shuffling, otherwise the vector is shuffled and checked again until data are well-posed to start training the CNN.

Then the CNN was trained again with the same structure but with the restructured database. Results are shown in Table 4.6 and Table 4.7.

		predicted value			
		Class 1	Class 2	Class 3	Class 4
true value	Class 1	114	2	2	4
	Class 2	111	1	1	8
	Class 3	112	1	0	12
	Class 4	83	1	1	27

Table 4.6: Test 2 - Confusion matrix results

	precision	recall	f1-score	support
Class 1	0.27	0.93	0.42	122
Class 2	0.20	0.01	0.02	121
Class 3	0.00	0.00	0.00	125
Class 4	0.53	0.24	0.33	112
accuracy			0.30	480
macro average	0.05	0.28	0.07	480
weighted average	0.03	0.09	0.04	480

Table 4.7: Test 2 - Classification report results

The *support* column, Table 4.7, shows that to each class belongs the same amount of images. Nevertheless, the CNN is not enough depth to able to classify images among all the four classes. That's why a series of tests were done in order to gradually increase the depth, adding new *Dense* layer and also adding neurons to each layer of the fully connected ones. There is a gradually increase because more it's not necessarily better, it's important to avoid a too much growth of parameters otherwise the CNN could reach even worst results.

Table 4.8 introduces the CNN architecture for a series of tests whose aim is to increase the neural network depth. Each one represents a specific test configuration for the neural network structure, the first two column are those tests whose results have just been explained.

Convolutional Neural Network architecture							
Test	Layers						
1	VGG 16	Dense (256)	Dense (64)	Dense (4)			
2	VGG 16	Dense (256)	Dense (64)	Dense (4)			
3	VGG 16	Dense (512)	Dense (64)	Dense (4)			
4	VGG 16	Dense (512)	Dense (256)	Dense (4)			
5	VGG 16	Dense (512)	Dense (256)	Dense (64)	Dense (4)		
6	VGG 16	Dense (512)	Dropout (0.3)	Dense (256)	Dense (64)	Dense (4)	
7	VGG 16	Dense (512)	Dropout (0.3)	Dense (256)	Dropout (0.3)	Dense (64)	Dense (4)
8	VGG 16	Dense (738)	Dense (512)	Dense (256)	Dense (64)	Dense (4)	

Table 4.8: CNN configurations for tests

For those tests are increased both depth, adding new hidden layer, and size of each layer, adding neurons per layers, as just said, there's no theoretical rule of thumb to determine this changes. That's why neurons and layers were added slowly. Moreover as it increases the quantity of weights and biases, was added to the model structure also a *Dropout* layer from *Test 6* with a rate of frozen neurons of 0.3, to avoid possible overfitting caused by too much parameters that has to be train.

As expected, adding more complexity to the CNN let it grows its ability to classify images among the four classes, thanks to more parameters needed to better generalize such a huge difficult classification problem. Below are analyzed *Test 7* and *Test 8* results, judging how the new structures react to training.

		predicted value			
		Class 1	Class 2	Class 3	Class 4
true value	Class 1	13	88	4	14
	Class 2	11	81	16	21
	Class 3	2	68	13	31
	Class 4	10	29	8	71

Table 4.9: Test 7 - Confusion matrix results

	precision	recall	f1-score	support
Class 1	0.36	0.11	0.17	119
Class 2	0.30	0.63	0.41	129
Class 3	0.32	0.11	0.17	125
Class 4	0.52	0.60	0.56	118
accuracy			0.37	480
macro average	0.38	0.36	0.33	480
weighted average	0.37	0.37	0.33	480

Table 4.10: Test 7 - Classification report results

Results of *Test 7*, on Table 4.9 and Table 4.10, demonstrate that for this kind of classification problem the CNN required a more complex structure. Nevertheless, even if the accuracy average value of classification increases until 0.37, clearly it is not sufficient. Moreover, from the confusion matrix on Table 4.9 could be figured out that there are too much prediction errors between non consecutive classes. This way of thinking could be done due to this specific class division. In other words, each class has a range of minimum and maximum population density in which an image has to be classified, from the lowest values of Class 1 ranges to the highest of Class 4 ranges. Therefore, there's a big difference between predicting an image to belong the lowest range of population density or to the highest. That means that the CNN is not able to recognize those features that let it able to estimate how many people could stay in that specific area. As an example, it can be figured out just focusing on first column of Table 4.9.. It can be seen that there are 10 images predicted to belong to Class 1 that actually are part of Class 4. Thus, 10 images, probably photos over the city center, are predicted as a park, for example. This problem is reasonably linked to misleading images, for example abandoned

factories have almost zero persons on that area but from an overhead point of view it appears as populated. The same reasoning can be done on column four of the same table: there are 14 images predicted on Class 4 but actually they are part of Class 1. The situation is the opposite but both are caused probably by misleading images within the database.

	precision	recall	f1-score	support
Class 1	0.34	0.23	0.28	116
Class 2	0.25	0.02	0.03	122
Class 3	0.24	0.43	0.31	116
Class 4	0.44	0.65	0.52	126
accuracy			0.34	480
macro average	0.32	0.33	0.29	480
weighted average	0.32	0.34	0.29	480

Table 4.11: Test 8 - Classification report results

For the *Test 8* only the classification report on Table 4.11 was reported because it is an example of overfitting. It can be understood from the comparison between the classification report of that test and the previous discussed above. Even if the complexity of the neural network is increased that didn't lead to a better prediction. That's happen due to the too much weights and biases to be set with respect to the input database size. So again, more is not better.

Summing up, the *Test 7* highlights the requirement to focus on those images that could be misleading. Therefore, the work started with those images that belong to Class 1 that were selected and checked one by one. Every single image was examined in order to check if it was congruous with respect to the others that full fill the same belonging class.

Figure 4.6 and Figure 4.7 are both images belonging to the Class 1 and, as it is quite evident, photos are very different one from the other.

This was happened because the database was divided in order to have each class with the same quantity of images but this kind of operation doesn't care about how images are composed by. As Table 4.5 figures out, in fact the majority of images have similar values for population density so a division in this way can lead to have two opposite images belonging to the same Class 1. Images which reveal a very low population density are few. To fix this issue, was decided to transform that

Figure 4.6: Example of an image classified in Class 1

Figure 4.7: Example of an image classified in Class 1

Class 1. Fixing the thresholds values of the population density of that class, were chosen 95 images among all to be the ones that represent a very low population density value. In order to avoid possible underfitting due to the quantity difference between Class 1 and the others, these 95 images were overlapped that is a common practice when there's no other way to add images to database. Practically, they were just copied until the right quantity was reached. After that each 95 copied images were rotated with different angles. That's why, from a neural network point of view a rotate image is analyzed as a different one with different features. The leftover part of the database was divided as in the previous tests among the other three classes. Due to overlapping the database contained 2523 images so it is bigger than before. That's why also the quantity of images per set was changed into 1312 for training, 672 for validation, 539 for test. This new database was then tested on a CNN whose structure is the one with the best results, *Test 7* of Table 4.8. So it is composed by the VGG 16 complete model plus 4 fully connected layers with respectively 512, 256, 64 and 4 neurons as output, without any dropout layer. Results are listed on Table 4.12 and on Table 4.13.

If focusing on the first row of the confusion matrix (Table 4.12) could be seen that almost all images are predicted correctly, confirmed by the 0.83 recall value of Class 1 into Table 4.13. Therefore, the work done on database for what concern Class 1 images had lead to a better classification. In addition, also the precision value of Class 1 raise up due to less error of predicting an image belonging to that

		predicted value			
		Class 1	Class 2	Class 3	Class 4
true value	Class 1	99	10	1	9
	Class 2	12	71	27	34
	Class 3	18	59	21	38
	Class 4	28	32	16	64

Table 4.12: Test 9 - Confusion matrix results

	precision	recall	f1-score	support
Class 1	0.63	0.83	0.72	119
Class 2	0.41	0.49	0.45	144
Class 3	0.32	0.15	0.21	136
Class 4	0.44	0.46	0.45	140
accuracy			0.47	539
macro average	0.45	0.48	0.46	539
weighted average	0.45	0.47	0.45	539

Table 4.13: Test 9 - Classification report results

class while it actually belongs to another one. First column of Table 4.12 highlights that there are still data predicted wrongly predicted on Class 1. That's means that the images belonging to other classes are also misleading. Therefore it was decided to check one by one images of others classes looking for those that cause misunderstanding during the CNN training.

Starting from Class 3, each image belonging to that class was selected and analyzed, checking its robustness. In other words, each photo was examined with respect to the others and it was judged as consistent for that range of population density. This was done because the used database deploys an estimation of population density over urban area that is a collection of multiple factors that not all of them can be caught through an aerial image. After all the images ranked as misleading were selected, they were substituted with a rotate photo taken from those left in the same class. The substituted image was not chosen casually but was used the ones that are in the population density neighbourhood of the image that has to be removed. After all, 59 images are removed and replaced. Once the

images belonging to Class 3 are substituted, the same way of thinking was applied to those that are part of Class 4. In this case 50 photos are removed and replaced.

Finally, a new CNN was trained in order to check if the new modified database lead to better prediction result. As before, the CNN was trained with the same configuration of layers but for this attempt was modified the division quantity for the training and evaluation categories (training, validation and test) that respectively had 1477, 795, 251. It was reduced the test set because this set is used to compute the confusion matrix and classification report, so to evaluate the CNN instead of the training and validation sets whose purpose is modify the network parameters to reach better prediction results. Hence, it was decided not so much to reduce that test set but to enlarge the others because a bigger database can improve the training due to a more variety of photos on which the CNN is trained. Even if those categories were modified, the database was still randomly shuffled as before in order to still have an equilibrium of quantity for each class and also an equilibrium of quantity for each phase set.

The CNN was trained with the modified database, results are listed on Table 4.14 and Table 4.15.

		predicted value			
		Class 1	Class 2	Class 3	Class 4
true value	Class 1	58	8	0	2
	Class 2	8	45	3	1
	Class 3	13	28	8	9
	Class 4	15	22	7	24

Table 4.14: Test 10 - Confusion matrix results

The new results show an increasing accuracy that affects all the classes, except for the Class 1 that still maintains the optimal prediction values already reached. There is an improvement even for the Class 2 predictions, whose belonging database was not modified, however it reflects the adjustments done for the others. The worst improvement is on Class 3 prediction that is the only one that basically preserve the same accuracy on prediction, very low. In addition, it can be noticed that the higher is the population density, the worst the CNN is able to correctly predict images. In fact Class 3 and Class 4 lead to worst results compared to the other two.

	precision	recall	f1-score	support
Class 1	0.62	0.85	0.72	68
Class 2	0.44	0.79	0.56	57
Class 3	0.44	0.14	0.21	58
Class 4	0.67	0.35	0.46	68
accuracy			0.54	251
macro average	0.54	0.53	0.49	251
weighted average	0.55	0.54	0.50	251

Table 4.15: Test 10 - Classification report results

Finally, was arisen the possibility to enlarge the input database thanks to new population density data estimations over a predetermined area. It was done a trade-off between the covered area of each image and the number of image because the larger the area is, the lower is the number of input data. As an example, if it had been chosen a photo which covers an area of 250×250 meters, in this case the database would have had 7315 numerical values of population density estimation. On the contrary, if wanted an higher number of estimated values, this would run up against a too reduced dimension for each photo in which it's not possible to recognize any pattern. Finally, it was decided to select 150×150 meters as covered area per cell. So, the downloaded images are a little bit bigger than before, this let each one of them to include within itself a better pan, for example more buildings avoiding those images which present a single huge crossroad. This dimension for the covered area corresponds to a database composed by 20382 elements. That's almost ten times bigger than the one used so far.

Once the database has been enlarged with all the new downloaded images, it had been used to train a new neural network. In addition, also the neural network complexity was raised up, considering that finding the main features could be more difficult if there are too input data with respect to the complexity of the CNN, this could lead to underfitting. Thus as first attempt, the decision was fallen back into a CNN whose structure is the one shown on Table 4.8 for the *Test 8* configuration. The reason why is that this more complex structure for a neural network was fallen back into overfitting when tested on the previous database due to too much weights and biases to be set with respect to quantity of input population density estimations. Nevertheless, for this attempt was available a new database bigger than before. In addition, an adjustment was done on an argument for the *compile*() function within which was modified the *batch_size*, from 32 to 256. This is done

in order to avoid possible underfitting or overfitting due to a wrongly proportion between new database size and batch dimension. Moreover, this adjustment helps to speed up the training time. A new database implies also new value for the training, validation and test sets. However, excluded those images for the test set, it holds the proportion of 65% and 35% between respectively training and validation sets. So, images were subdivided into chunks of 11294, 6082, 3006 respectively for the training, validation and test sets. The remaining code wasn't modified.

		predicted value			
		Class 1	Class 2	Class 3	Class 4
true value	Class 1	511	173	37	3
	Class 2	289	300	151	20
	Class 3	60	199	402	98
	Class 4	8	44	311	400

Table 4.16: Test 11 - Confusion matrix results

	precision	recall	f1-score	support
Class 1	0.59	0.71	0.64	724
Class 2	0.42	0.39	0.41	760
Class 3	0.45	0.53	0.48	759
Class 4	0.77	0.52	0.62	763
accuracy			0.54	3006
macro average	0.56	0.54	0.54	3006
weighted average	0.56	0.54	0.54	3006

Table 4.17: Test 11 - Classification report results

As can been from Table 4.16 and Table 4.17, even if the previous trained neural networks had have several modifications such as complexity of the networks or images substitution, the enriched database on which the new CNN was trained lead to better results on prediction. That means that the preceding database wasn't big enough to be able to find those generalize features to classify images. The confusion matrix columns show that the classification errors of the trained CNN are higher on the nearest class with respect to the correct one. As an example, for the first column there are 289 images wrongly predicted on Class 2, 60 on Class 3

and only 8 on Class 4. Thus, if the CNN wrongly predicts an image classification, the probability that the error is the nearest class is very high. In other words, this confusion matrix shows that the neural networks has understood which are the main difference between classes and that the images are differently divided in classes based on an increasing value among them. Same reasoning can be done for all classes. It's reasonable to think that if applied the same method used before to check database composition it's possible to get even better scores, because there still are misleading images.

Summing up, these final scores highlight that the CNN understand the increasing order of the classes but instead of repeating the same checking operations on the new database it's possible to go to the further step that is the *numerical regression* to estimate population density, the final goal of this thesis.

4.3 Numerical regression to estimate population density

A numerical regression approach is used to find that function that most nearly pass through the input data. In a neural network environment the first part composed by convolutional and pooling layer has still the aim to extract features from images while the last fully connected layer has now the goal to find that function that best fit the data. This target is specified to the neural network just arranging the last layer, also known as output layer, of the fully connected ones with a single neuron. In this way the CNN is forced not to classify the input data, as a matter of fact at the end of the CNN there's no more 2 or more neurons. On the contrary there a single neuron whose objective is to predict a single specific numerical value. So the aim of the CNN is to set weights and biases in order to be able to correctly fit all the input data.

Practically, the CNN was built as so far, it was changed only the last layer output from n neurons to a single one. In addition, also its activation function was changed from *softmax*, specific for classifying, into *linear* because for the numerical regression it's not wanted to modify the numerical output, what the CNN is trying to predict. The other layers which compose the fully connected layers maintain the same $ReLU$ activation function.

Obviously, also the tests to evaluate the neural network performances were changed, the confusion matrix and the classification report are no more useful. So the new tests were inclusive of a graphical approach and a numerical one. For a graphical approach was intended that the predicted output was used to create a scatter plot on which the x axis defines the real population density value, whereas the y axis defines the predicted one. If a model perfectly predict all the value,

all the points of the scatter plot have to lie on the bisector. Therefore, with the graphical approach is analyzed how much the plotted point are far away from the bisector and if from their shape can be figured out a line. In addition to the plot, also the *coefficient of determination*, see section 3.3.1 for more details, was used as complementary to the graphical approach. So, the plot gives the visual representation of the predicted output, whereas the coefficient of determination gives the numerical value of how good the prediction is.

As always, the first attempt was done with the same architecture of the last test done, it is shown in Figure 4.8.

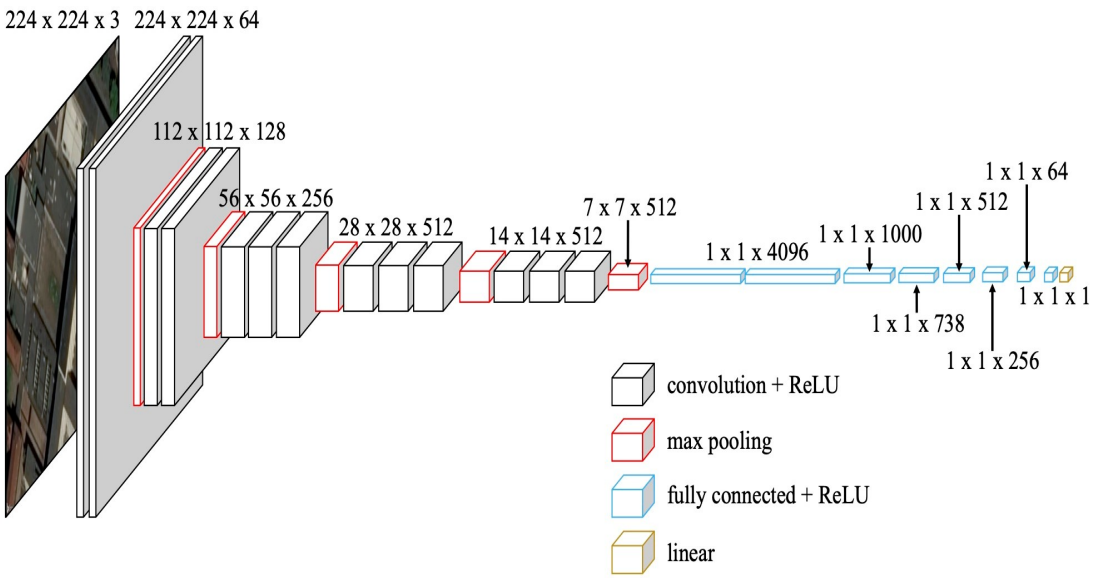

Figure 4.8: Test 1 - Convolutional Neural Network configuration

The database used as input is the one downloaded for the last test of the *Multiclass problem* section, with 20382 elements whose division sets are now adjusted. Training set was modified to have 11264 elements, whereas training set had 6144 too. The remaining 2974 images were used for the test phase. This calibration was done in order to have the first two sets to be multiple of the *batch_size* value, 256. That's because was wanted that the last batch of the training epoch is still equal to 256, so this could be done if and only if the training and validation sets quantities are multiple of that number.

As before, before the training the database is still shuffled with the same criteria but in this case was done over 10 ranges, that was just a project decision and didn't have any symbolism. The aim was still to avoid the possibility to give as input

sequential images and to allot to each set an equally distribution of the images. Thus, using 10 ranges means to increase the randomness and variety for each set.

Last but not least, it was revised also the *compile*() function that configures the CNN for the training. For the *loss* function to be minimized was chosen the *mean squared error* and for the *metrics* was chosen the *mean absolute error*. In this way was possible to directly check during training the error on prediction. Moreover was changed also the *optimizer*, it was used the *adam*, it implements the *Adam algorithm* that is a stochastic gradient descent method based on adaptive estimation of first-order and second-order moments.

Once everything was modified, the first attempt of training the CNN with this new goal was done, the resulting scatter plot is shown in Figure 4.9.

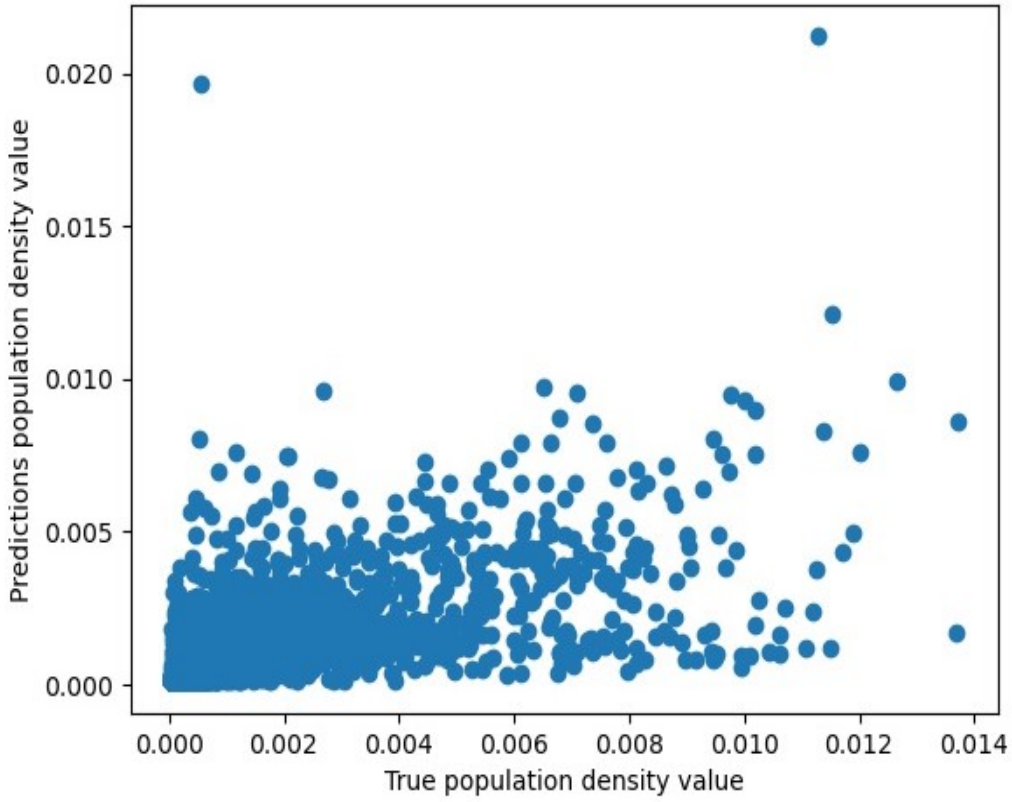

Figure 4.9: Test 1 - Scatter plot results

As can been seen from the figure, the CNN was not able to find that function that fits the input data. The shape of the scatter plot doesn't lie on the bisector, on the contrary its shape is not a line but only a points cloud. Moreover, the plot highlights that the population density value to be estimated have a very tiny

value and its distribution it's not uniform. So, it's difficult to discern points whose distribution is gathered close to 0 instead of being uniformly distributed.

So, the further step was to redistribute all the labeled output data. It was done firstly changing from a linear distribution into a logarithm one. After that, the new data were scaled uniformly in a range between 0 and 1. Finally, the new scaled data are employed to train a new CNN whose architecture was the same as the first attempt (Figure 4.8).

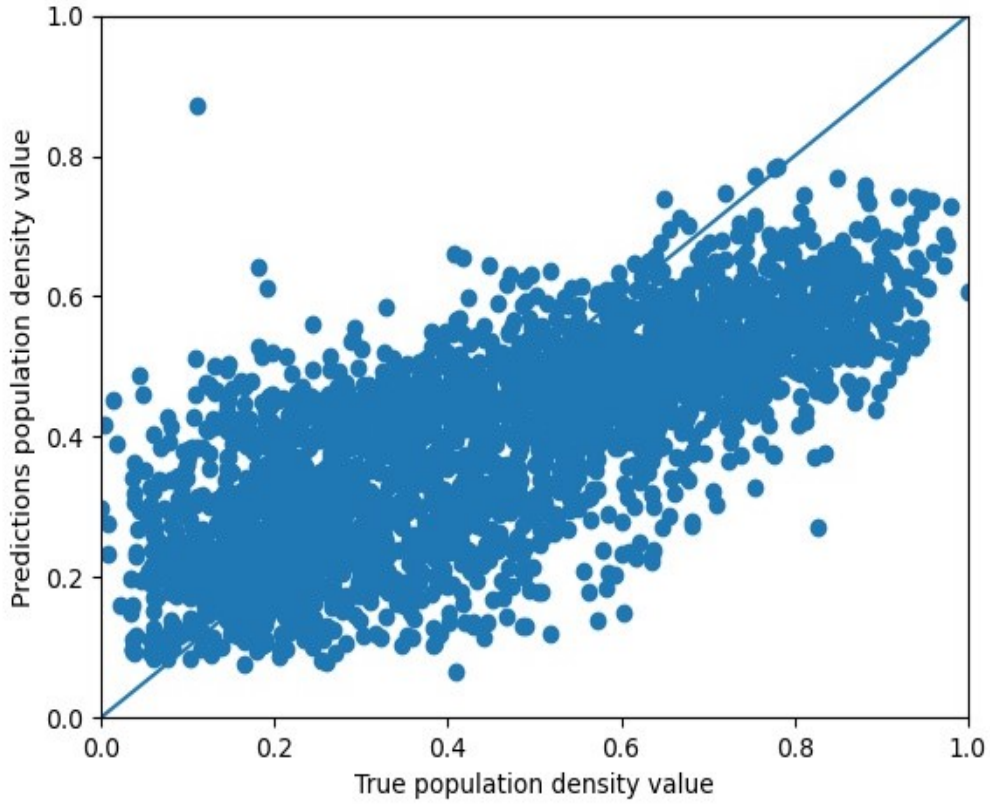

Figure 4.10: Test 2 - Scatter plot results

Figure 4.10 clearly exposes a scatter plot whose shape is similar to a thick line and its orientation is quite similar to the bisector of that plane, the continuous line plotted. From the plot can be highlighted that the central values are well predicted, whereas the errors become bigger when getting closest to boundary limits. The graphical result is also supported by the coefficient of determination value that is 0.54. It is a good value considering that 0.75 [5] value is an optimal prediction value.

This last test has produced an optimal result on estimating the population density value given as input an aerial image. So the aim now is to reduced the thick

of that plot and was decided to initialize a further step to create and train a more adapted CNN to that specific scope. As said on the introduction, the architecture of the CNNs trained so far were based on the VGG 16 model that was not modified at all but on its output layer were attached fully connected ones to classify images before and then estimate a numerical value. However, the complete VGG 16 model was trained to classify common objects over 1000 classes, so weights and biases were set to pursue that purpose. These VGG 16 parameters were set always as not trainable ones. As just said, the firstly convolutional layers of the model extract low-level features from the images instead of the lastly ones that are responsible of extracting the high-level and more complex characteristics of the images. So, was decided to attach two more convolutional layers to the transferred model in order to train specific layers for high-level features. Those new layers have to train to directly recognize specific characteristics for population density estimation instead of being adapted from another image processing goal. So, the VGG 16 model was "cut" from the *Flatten* layer due to the fact that the convolutional layer works with bi or three-dimensional matrix not with a vector, therefore the new two convolutional layers were attached directly to the last pooling layer of the transferred model. To be more specific, all the VGG 16 fully connected layers were "cut" and they were replaced with identical ones. Nevertheless, even if the structures are identical, there's a fundamental difference between them that is that the new layers has randomly initialized weights and biases instead of just set ones.

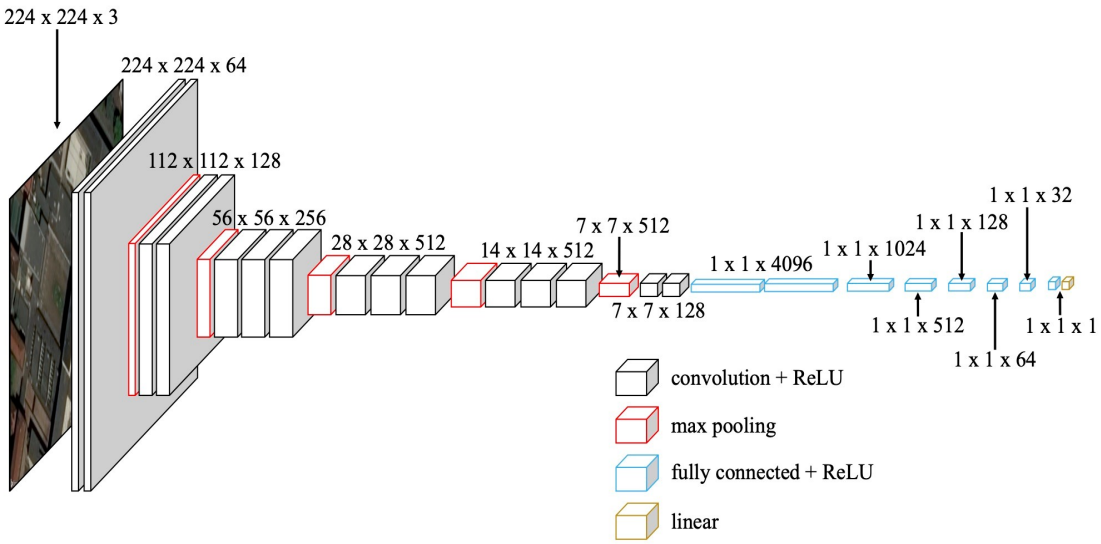

Figure 4.11: Test 3 - Convolutional Neural Network configuration

Figure 4.11 shows the new convolutional neural network architecture. It was tested on the same database division for each phase set, results are shown in scatter plot of Figure 4.12.

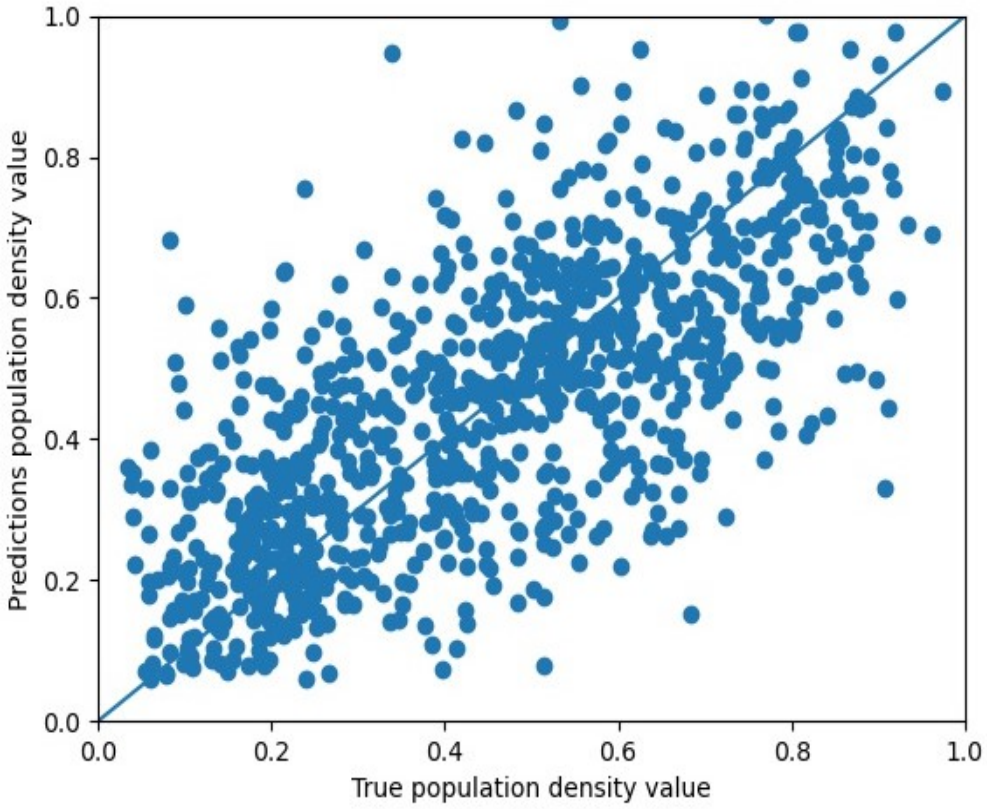

Figure 4.12: Test 3 - Scatter plot results

The scatter plot shows that the points follow the line but there are too much wide prediction error for all the values, from the lowest to the highest. The coefficient of determination is 0.43. Therefore, it seems that CNN starts to understand the main features to generalize the problem but it could need more training epochs, in fact the train stops only after the whole 100 epochs that means that the CNN need more time and computational effort to have better prediction. In addition, also an even more depth structure for the neural network could help to improve estimation performance.

Chapter 5

Conclusion and future work

In this thesis a Convolutional Neural Network was developed both to classify aerial images, based on population density values, and to directly estimate the population density over urban area. The aim is to extend the previous work done by the research team that concerns a risk-informed path planning for UASs in order to perform autonomous operations in BVLOS over urban areas. An artificial intelligent methodology was proposed in order to compensate for the lack of data regarding the population density distribution, in fact they are not always easily obtainable or even accessible.

Starting from a pre-trained Convolutional Neural Network model, the VGG 16, initially we developed a Convolutional Neural Network that was trained in order to classify aerial images into two different categories: city or country. For this first classification problem was achieved quite perfect results with 95% of accuracy.

The successive step was to develop a Convolutional Neural Network whose aim is still the images classification applied on a more complex problem, *i.e.* a prediction among four different classes with increasing population density values. As *ground truth* to support the training, it was used a database which contains a realistic estimation of the population density, covering Milan, Turin and Rome. For this classification problem, all the tests done lead to the optimal results achieving a 55% of accuracy for prediction.

The last method implemented was based on numerical regression approach to directly estimate the population density value to be used to create layers for the risk-aware UAS path planning. Also this method achieves optimal results with a coefficient of determination of 0.54, a good results compared with the literature [5].

Moreover, we developed one last preliminary step with a new customized Convolutional Neural Network. The plotted results highlight that the new customized CNN has promising features, suitable to be extended and improved to obtain even better results on estimating population density.

Future works will include the use of a more compliant database to avoid all the possible misunderstandings due to misleading images. Even the Convolutional Neural Network architecture has to be more customized, in order to find the optimal trade-off between complexity of the network and its performances. In addition, further training have to be supported by a specific hardware for neural networks.

Finally, once the CNN was correctly able to estimate population density distribution, the next step will be the creation of a network that is able to predict values from dynamic maps.

Bibliography

[1] Stefano Primatesta, Luca Spanò Cuomo, Giorgio Guglieri, and Alessandro Rizzo. «An innovative algorithm to estimate risk optimum path for unmanned aerial vehicles in urban environments». In: *Transportation research procedia* 35 (2018), pp. 44–53 (cit. on p. 2).

[2] Stefano Primatesta, Alessandro Rizzo, and Anders la Cour-Harbo. «Ground risk map for unmanned aircraft in urban environments». In: *Journal of Intelligent & Robotic Systems* 97.3 (2020), pp. 489–509 (cit. on p. 2).

[3] Karen Simonyan and Andrew Zisserman. «Very deep convolutional networks for large-scale image recognition». In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on pp. 3, 23).

[4] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. «Accelerating very deep convolutional networks for classification and detection». In: *IEEE transactions on pattern analysis and machine intelligence* 38.10 (2015), pp. 1943–1955 (cit. on p. 3).

[5] Caleb Robinson, Fred Hohman, and Bistra Dilkina. «A deep learning approach for population estimation from satellite imagery». In: *Proceedings of the 1st ACM SIGSPATIAL Workshop on Geospatial Humanities*. 2017, pp. 47–54 (cit. on pp. 3, 57, 60).

[6] Patrick Doupe, Emilie Bruzelius, James Faghmous, and Samuel G Ruchman. «Equitable development through deep learning: The case of sub-national population density estimation». In: *Proceedings of the 7th Annual Symposium on Computing for Development*. 2016, pp. 1–10 (cit. on p. 3).

[7] Caroline Clabaugh, Dave Myszewski, Jimmy Pang. *Neural Netowrks History: The 1940's to the 1970's*. `https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html` (cit. on p. 5).

[8] Caroline Clabaugh, Dave Myszewski, Jimmy Pang. *Neural Networks History: The 1980's to the present*. https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history2.html (cit. on p. 5).

[9] Michael A Nielsen. *Neural networks and deep learning*. Vol. 2018. Determination press San Francisco, CA, 2015 (cit. on p. 7).

[10] Microsoft. *Bing Maps Rest Services*. https://docs.microsoft.com/en-us/bingmaps/rest-services/ (cit. on p. 19).

[11] Keras Special Interest Group. *Keras*. https://keras.io/ (cit. on p. 23).

[12] ImageNet. *ImageNet Large Scale Visual Recognition Challenge*. http://www.image-net.org/challenges/LSVRC/ (cit. on p. 23).

[13] ImageNet. *ImageNet*. http://www.image-net.org/about-overview (cit. on p. 23).

[14] Lars Buitinck et al. «API design for machine learning software: experiences from the scikit-learn project». In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122 (cit. on p. 25).

[15] Towards data Science. *What Are Overfitting and Underfitting in Machine Learning?* https://towardsdatascience.com/what-are-overfitting-and-underfitting-in-machine-learning-a96b30864690#_=_ (cit. on p. 25).

[16] Scikit-learn. *Confusion Matrix*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html#sklearn.metrics.confusion_matrix (cit. on p. 27).

[17] Scikit-learn. *Classification report*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html (cit. on p. 28).