Dipartimento di Scienze e Innovazione Tecnologica

Corso di Laurea Magistrale in Informatica

Tesi di Laurea Magistrale

CleanAir Project: a Neural Network approach to cities pollution monitoring

Relatore:

Prof. Luigi Portinale

Correlatore:

Prof. Attilio Giordana

Candidato:

Alessandro Abluton

Anno Accademico 2019/2020

Abstract

CleanAir Project is a proposal of an innovative way of monitoring ultrafine particles (UFPs) in urban environments. Such particles may have severe impacts on human health and the lack of knowledge about their spatio-temporal distribution inhibits a deep understanding of these effects. In this thesis I focus on developing the deep learning module of CleanAir Project's application, in charge of predicting UFPs concentrations in an urban environment and producing high-resolution pollution maps for different time aggregation levels.
Due to the infeasibility of collecting enough data by myself and also in order to have a valid scientific comparison, the data set produced by OpenSense reasearch team has been used to train and test the regressive model. Such data set is composed by almost 20 million measurements made by mobile stations across the city of Zurich for a year, and they also developed a regressive model to produce high-resolution UFPs maps. I developed a Convolutional Neural Network to predict UFPs concentrations in a portion of a city given the corresponding map image and some information about weather conditions.
The model's capabilities have been evaluated using the same metrics used by OpenSense and the results are really close to each other. This fact and also some fundamental differences in how the data are aggregated make CleanAir Project a promising work, worth of further investigations and studies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis project has been initially inspired by my participation to the 2017 Climathon in Alessandria, a hackaton whose focus is on proposing new ideas to help fighting climate change; in that event I had the first intuition that eventually brought this project to life: "Having a single value of pollution for a whole city is not descriptive enough but to have multiple values for different areas of the urban perimeter would require too much monitoring stations!"
So I came up with the idea of using a little number of cheap monitoring stations that could move around the city everyday, and the municipal vehicles are perfect carriers for these stations, because they travel across the urban perimeter multiple times per day by doing their normal jobs. After gathering all data, I'm proposing a new way of handling them: use a Deep Convolutional Neural Network and a Multilayer Perceptron to predict pollution at different time agglomerations at any given zone in a city.

1.1 Brief introduction to Artificial Intelligence and Deep Learning

Deep Learning is actually a subset of Artificial Intelligence techniques that focus on Deep Neural Networks, which are particular structures capable of approximating mathematical functions by imitating the really complex neural connections that govern our brain. That being said, Deep Learning is a much specific bit of the whole world of Artificial Intelligence, which is made up by a lot different techniques each with it's own advantages and disadvantages, it is therefore mandatory to first introduce some of the more traditional methods available.

1.1.1 The concept of Intelligence

Before actually starting to talk about Artificial Intelligence we must define the concept of "intelligence" for humans, because our goal is to give computers some form of intelligence. Let us consider a newborn baby: everything he/she sees it's new, never seen before, how will the baby be able to learn?

The answer is simply through experience, the old statement *"we learn through our mistakes"* has never been more true because what the newborn does is actually failing a lot and then learning to associate the action **A** that brought to a bad outcome **O** to

something bad (e.g. a penalty **P**) therefore he/she will be less prone in the future to repeat the action that brought failure.

Let us look at a real-world example: said newborn sees the kitchen's fires lit for the first time, his/her mother isn't there at the moment so the newborn decides to touch the fire. He/She gets hurt, mommy comes immediately and after healing the wound tells to his baby not to do it again in the future. When does the **learning** actually happens? every time the newborn does an action that has a bad outcome, the neural connections that brought his/her brain to perform A become less likely to be activated, as if a penalty P is applied in some form on those connections; in this particular example the action is to touch the fire, the sad outcome are wounds, and the penalty is made up by both the pain and the mother's speech about how the fire is dangerous.

The whole challenge of Artificial Intelligence is to enable this process of learning, meaning that we want our software to be less likely to perform an action with some bad outcome; to achieve this goal a software has to know both how much it failed and how to correct itself. A lot of different mathematical techniques can be used to achieve this result and most of them have nothing in common, but the goal of each one is exactly the same and is well summarized in a definition of Intelligent Software given by Tom Mitchell [17]:

*"A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P** improves with experience **E**."*

Let us analyze each component of this definition:

- **T**: a class of tasks (or problems) such as Classification, Regression, Clustering
- **P**: some measure of performance, each task has it's most suitable ones
- **E**: the experience, meaning some time that needs to pass in which the learning software actually learns how to handle it's assigned task, hopefully reaching the optimal solution

The experience E is the actual process of trying an action, evaluating it's outcome and adapting the system accordingly reiterated over and over in time until some form of goal is reached, for example in Regression tasks frequently the Mean Squared Error is used as measure of performance and therefore the goal to reach could be that value, calculated between the system's predicted values and the original ones, falling beneath a threshold ϵ.

Having this kind of goal means that we're making a strong assumption: *"Repeating the process of trying some action over time, while slightly changing said action in each iteration, will eventually lead to lower the MSE value"*, but what guarantees do we have that the system is going to tweak the action in a good way, one that will lead to low MSE values, or more generally good values on any kind of performance metric?

To tweak an action in a proper way the system needs to actually **learn** over time how to handle it's task, exactly like the baby newborn does, and that goal is achieved through the help of some sort of **loss function**, which is a function that is able to tell the system how much it is mistaking. Those components are the "building blocks"

through which we can analytically describe every problem, and describing a problem is always the starting point to actually solve it.

Once understood this concept we can think of Artificial Intelligence as a set of techniques that, through mathematical tricks, try to give computers the ability to "learn from experience" exactly as humans do; so that if an action A is executed in the context of a task T, the measures of performance P let the system correct itself based on A's outcome evaluation.

The experience E is the actual process of trying an action, evaluating it's outcome and adapting the system accordingly, reiterated over and over in time until some form of goal is reached (e.g. the MSE value falls beneath a threshold ϵ).

1.1.2 Different learning methods

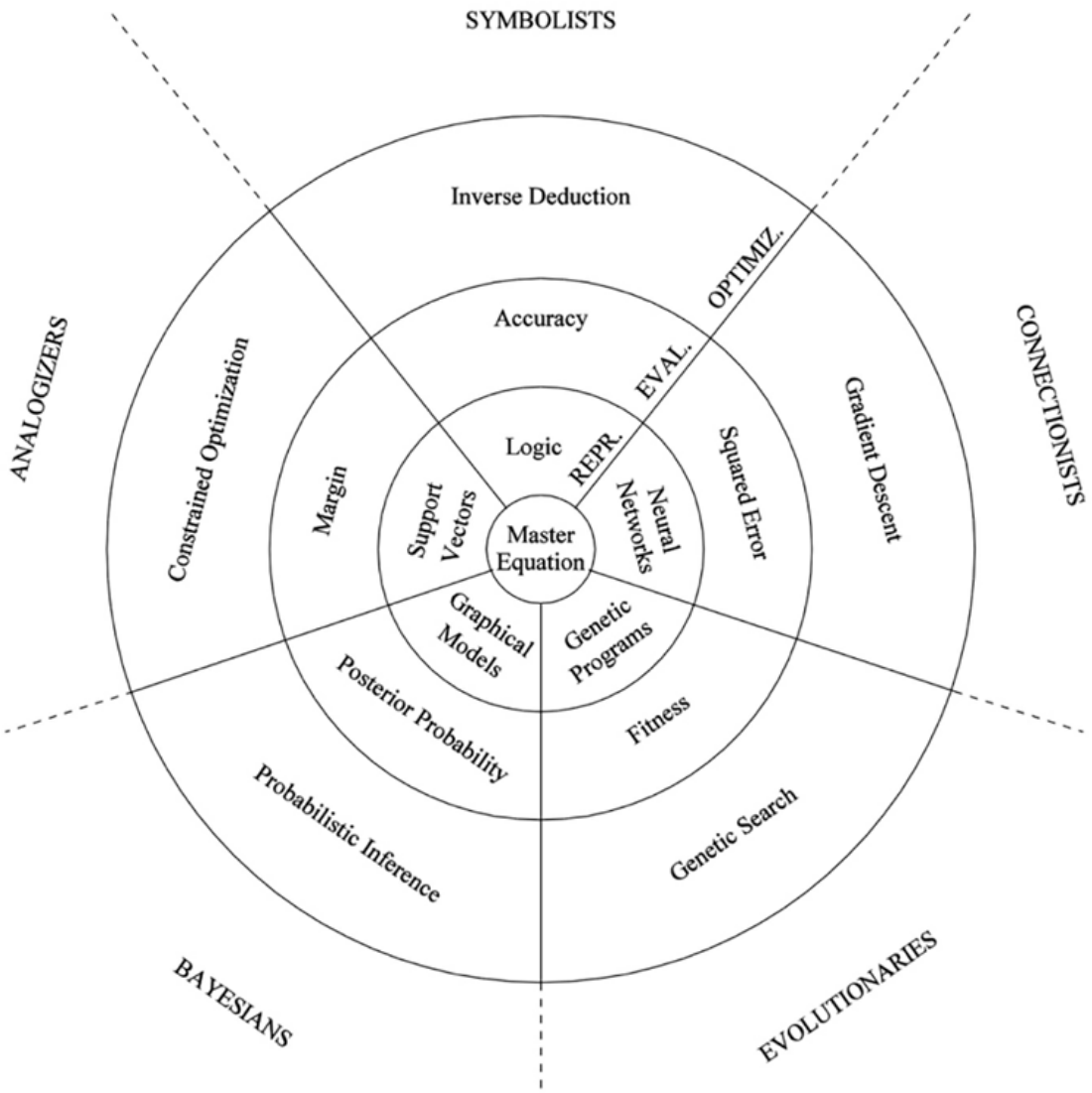


Figure 1.1: Domingues's ontolgy for AI methods

There are a lot of different methods to achieve the goal of having a system capable of learning over time, each with it's own peculiarities. I like to use an ontology written

by Pedro Domingues [7], who splits all these different methods in five tribes, depicted in Figure 1.1:

1. The Symbolists. They focus on the premise of inverse deduction: they don't start with a premise to work towards conclusions, but rather use a set of premises and conclusions and work backward to fill in the gaps.

2. The Connectionists. They mostly try to digitally re-engineer the brain and all of its connections in a neural network. The most famous example of the connectionist approach is what is commonly known as 'Deep Learning'. Their techniques have proved very efficient in e.g. image recognition and machine translation.

3. The Evolutionaries. Their focus lies on applying the idea of genomes and DNA in the evolutionary process to data processing: their algorithms will constantly evolve and adapt to unknown conditions and processes.

4. The Bayesians: Bayesian models will take a hypothesis and apply a type of "a priori" thinking, believing that there will be some outcomes that are more likely. They then update their hypothesis as they see more data.

5. The Analogizers: This machine learning discipline focuses on techniques to match bits of data to each other. Probably the most famous example of this type of machine learning, is the Amazon or Netflix recommendations: "If you have watched/bought this, you will probably like...".

Domingues claims that the ultimate goal of Aritificial Intelligence is to find a Master Algorithm, one that could be able to optimally solve any given task, that's why in Figure 1.1 we can see in the center the "Master Equation".

Each one of these methods has seen bot successes and failures over time, and has it's own peculiarities. In this thesis I'm focusing on Neural Networks, the Connectionist's building block to achieve the goal of having a learning system.

1.1.3 Neural Networks

In it's most generic acceptation a Neural Network is mathematical model that tries to reproduce the human brain's cognitive functions by mimicking the actual neural networks inside of it.

The *basic unit* of these networks are the so called *artificial neurons*, which can be seen as structures that receive some input and fire an output if the weighted sum of it's input overcomes a threshold ϵ.

An artificial neuron could be thought of as a "digital representation" of an actual neuron in our brain; as seen in Figure 1.2 a real neuron cell receives input through it's dendrites in the form of electrical signals and processes it, if the combination of all the signals it is receiving through it's dendrites is high enough it will fire a signal through the axon, that signal will become an input signal for all the neurons connected to that axon's terminal.

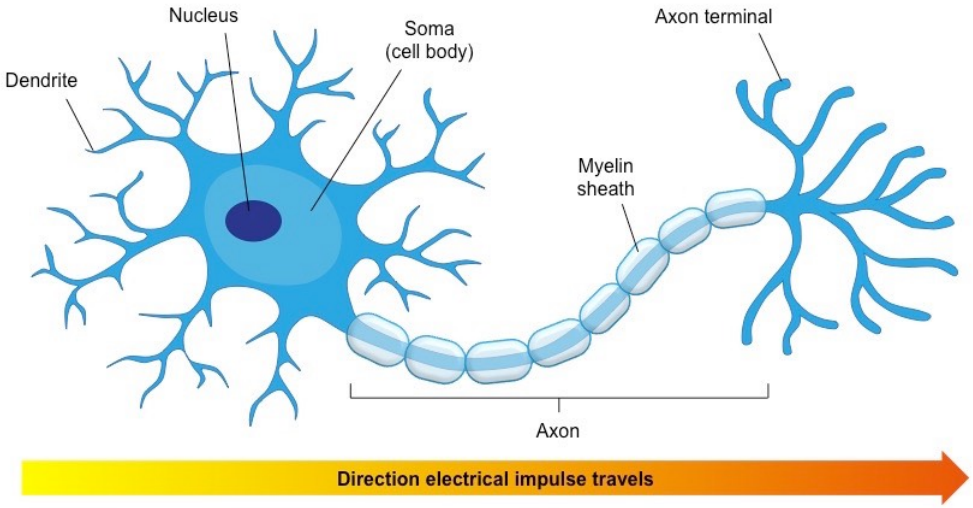


Figure 1.2: Simple representation of a neuron cell

Rosenblatt's Perceptron

From the early 1940s a lot of different implementations of a digital neuron have been proposed, one that has become universally accepted is the Single Layer Perceptron proposed by Frank Rosenblatt in 1958 [27].

In that model the neuron is represented as in Figure 1.3, it receives an array x of inputs, each element of x has it's weight w_i and there's also a bias parameter w_0 that helps in regularizing the model's outputs. Rosenblatt uses also a **linear activation function** $out(t)$, which represent the threshold value above mentioned; the neuron computes a weighted sum of it's inputs, the passes the result to the activation function: if the value is high enough and passes the threshold the whole neuron will fire a 1 ($out(t) = 1$), otherwise it won't fire anything ($out(t) = 0$).

$$\begin{cases} 0 & \text{if out(t)} < \epsilon \\ 1 & \text{if out(t)} \geq \epsilon \end{cases} \tag{1.1}$$

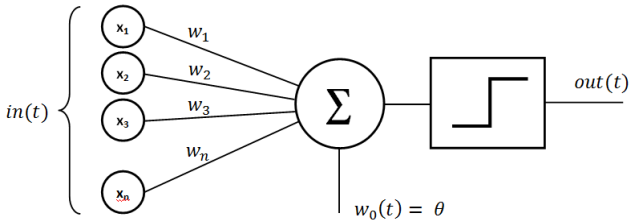


Figure 1.3: Rosenblatt's representation of an artificial neuron

The first proposed network by Rosenblatt consisted only in one single neuron as depicted in Figure 1.3, but even if it's underlying concept was mathematically strong and trusted it wasn't able to approximate functions with sufficient precision, mostly because the fact of having a linear activation function makes it able to approximate well only linear functions, and it hardly happens that a real-world problem can be solved by such functions. This weakness is the reason why it never grew so much interest in the scientific community at it's earliest stages.

Multilayer Perceptron

The next advancement was made by creating an actual network of neurons known as Multilayer Perceptron (Figure 1.4), which greatly improved the approximation capabilities of these models. The network is made up by three layers of neurons, with

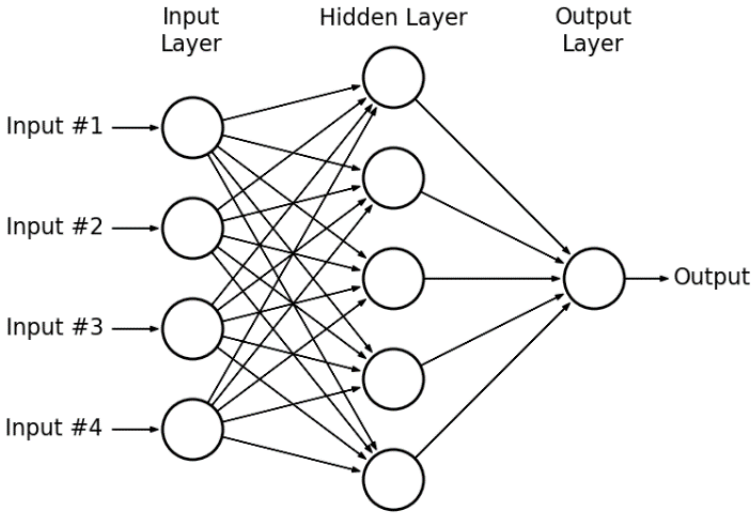


Figure 1.4: An example of Multilayer Perceptron

the hidden layer that usally has more neurons than the input layer and the output one that is either made up by a single neuron for regression task, or by n neurons for n-class classification tasks (meaning that the i-th output neuron holds the probability that the j-th sample belongs to class i).

This added complexity gives the model the ability of approximating *non-linear functions* because every hidden layer is actually defining an hyperplane intersected with input data's vector space. The splitting of that space in more and more complex regions lets the system describe at best the values of input data.

Perceptron's learning algorithm

Mathematically speaking, learning for a single Perceptron (or neuron) means tweaking the weights $\mathbf{w}$ associated to the inputs $\mathbf{x}$ so that the hyperplane generated by the weighted sum of inputs and weights will separate linearly the input data in the best possible way.

To achieve that goal we use the difference between the correct value and the one produced by the network: the grater the error, the more weights tweaking still needs to be done in order to obtain the best possible hyperplane.
Given a set of weights $\mathbf{w}$ and a set of inputs $\mathbf{x}$, the value of i-th weight will be modified with this formulae:

$$w_i = w_i + \eta E x_i \tag{1.2}$$

$$E = (t - y(x)) \tag{1.3}$$

E represents the error between the original value t and the network's produced one $y(x)$ and it is referred to as *loss function*, while η is a constant known as *learning rate* with values in range $[0, 1]$ used to set a value to tweak the weights with, the closer to

1 the higher are the changes be made to said weights.

Setting a proper learning rate is mandatory to get good results, both in terms of training speed and prediction's accuracy; as seen in Figure 1.5, if set too close to 1 the weights will be changed too much in each iteration and there will be a high risk of completely missing an actual minimum, whereas if set to close to 0 the learning process would be really accurate but tremendously slow and therefore infeasible in terms of time and money.

Assuming that the set of input samples is linearly separable and η is small enough, this kind of learning will always eventually converge.

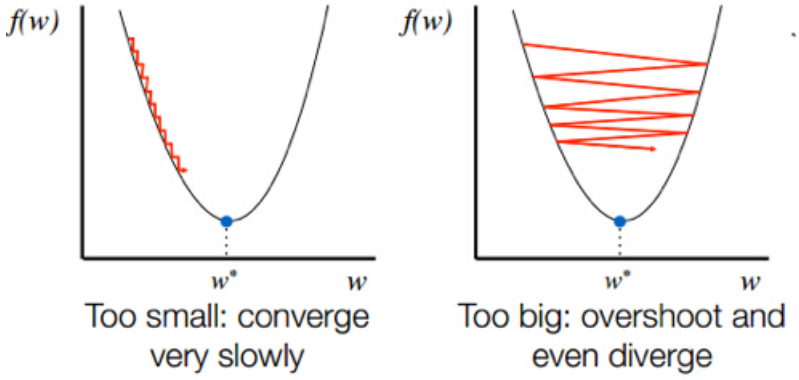


Figure 1.5: Learning rate problem

Multilayer learning, Gradient Descent

The main goal of this method is reaching the lowest possible value of some loss function, by using the best values for the weights vector **w**. The weights are modified in accordance to the loss function's minimum, calculated by deriving said function in respect to each component of **w**:

$$w_i = w_i + \Delta w_i \tag{1.4}$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \tag{1.5}$$

$$E = \frac{1}{k} \sum_{i=1}^{k} (t_i - y(x_i))^2 \tag{1.6}$$

$$y(x) = \sum_{i=1}^{k} x_i w_i \tag{1.7}$$

The learning rate is a positive constant but as we can see in (1.5) it is negated, because the goal is to reach the *lowest* value of E.
The loss function used in this example E is the Mean Squared Error, but actually any function that is able to give a form of error works, the difference is in what kind of error we are considering: different loss functions produce different trained models because each one gives more importance to different aspects of the prediction error.

To further clarify let us consider the Mean Squared Error, given the fact that every error gets squared by using this kind of function we are penalizing large mistakes much

more than the small ones, an error of 200 means an MSE of 40000, where an error of 0.1 produces an MSE of 0.01.

The **Gradient Descent Algorithm** is the process of repeating this operation over and over in time, until some goal is reached; usually it could be a fixed number of iterations to perform, the loss function reaching a fixed value, but any kind of stopping criteria can be experimented.

1.1.4 Convolutional Neural Networks

There are a lot of different models of neural networks, each with it's best use case; given that in this thesis I'm making use of map images to predict pollution levels I focused my work upon Convolutional Neural Networks, the state-of-the-art model for all kinds of computer vision tasks.

Convolutional networks (LeCun, [14]) also known as CNN, are a particular model of neural network capable of handling data known to have a grid-like topology such as time series (can be thought as a 1-D grid of all the samples taken at regular time intervals, or images (2-D grid of pixels).

These networks are one of the main reason why Deep Learning gained so much interest in recent times, thanks to the amazing results achieved in computer vision tasks (e.g. handling of images and video related tasks) a lot of which are actually shaping the new society of this century.

One of the most iconic examples is the chinese government's model for Facial Recognition: in 2018 it reached almost perfection with values of accuracy as high as 95% (human accuracy is estimated to be 93%) in this task thanks to the really complex and powerful surveillance system spread country-wide. A lot could be argued about the method used by that government to get that much accuracy, but ethical implications regarding these kind of networks are out of the scope of this thesis.

An intuitive definition that describes convolutional network is given by Ian Goodfellow [9]:

Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers

The real game-changing peculiarity of this network is that it actually learns how to extract features from images, so that those features are descriptive enough to make the prediction process as optimal as possible, and as the definition says what makes it possible is the **operation of convolution**.
That is because one of the most challenging problems in image recognition tasks is the **feature extraction**: what features of a dog's picture make it an actual dog picture? what should the model be looking for, to determine whether the image it is analyzing contains a dog, a cat or both of them?

A convolutional network is made up by a fixed number of *convolutional layers*, each with the task of extracting different feature sets from an image. The first layers will be the ones that recognize wider features such as contours of things and the last ones will be recognizing patterns and colors that only in appearance have nothing to do with the problem to be solved, but they are actually really useful to get good results.

The Convolution Operation

In its most generic abception, convolution is a linear operation on two functions of a real-valued argument, and it is usually denoted with an asterisk:

$$s(t) = \int_a^b w(y)x(t-y)dy \tag{1.8}$$

$$s(t) = (x * w)(t) \tag{1.9}$$

There are differences between the meaning of each component, depending if convolution is used in a classical math context or in convolutional network; in classical math terminology $w(y)$ is a probability density function normalized in $[a, b]$.
Instead in convolutional networks terminology the first argument x is called the **input** and the second one w is the **kernel**, the output of this operation is called **feature-map** and t is the **time index**. In this form it's impossible to take *continue* measures, therefore we must switch to the *discrete* variant of convolution (example for 1-D signals):

$$s(t) = \sum_{k=-\infty}^{+\infty} w(k)x(t-k) \tag{1.10}$$

Usually the input **x** is a multidimensional array of data and the kernel is a multidimensional array of parameters randomly generated and then subsequently adapted by the learning algorithm, so that the convolution operation highlights the more suitable features of the image.

These kind of multidimensional parameters will be referred to as **tensors**.

When handling images or any kind of 2-D data, the convolution has to be performed on two dimensions:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{1.11}$$

where I is the intensity value of the pixel in position *(m, n)* in the image and K is the kernel.

Convolution is **commutative**, therefore (1.11) could be written like this:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{1.12}$$

usually the latter formula is used in machine learning applications, because the values of n and m will variate less than the former one.

The commutative property of convolution arises because we have flipped the kernel relative to the input, in the sense that as m increases, the index into the input increases, but the index into the kernel decreases.
It's important to notice that many neural network libraries implement the **cross-correlation** but call it convolution, which is actually the same but without flipping the kernel:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{1.13}$$

In any case, the learning algorithm will learn the proper values of the kernel in the right place, so an algorithm based on convolution with kernel flipping will learn a kernel that is flipped relative to the kernel learned by an algorithm without the flipping.

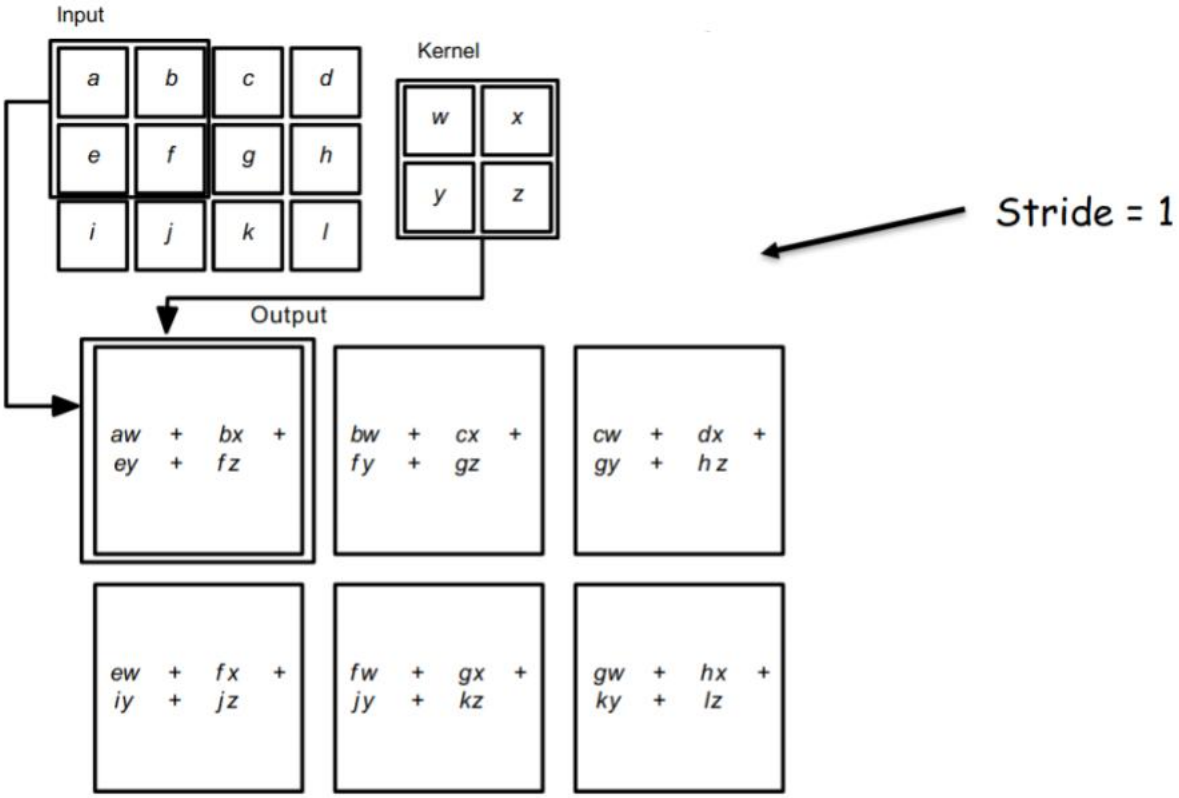


Figure 1.6: An example of Convolution on an image, with stride=1: the first time K is applied to the pixels *"a, b, e, f"*, but in the next iteration it will be applied to the pixels *"b, c, f, g"*

Why use convolution

The use of convolution leverages three important concepts: **sparse interactions**, **parameter sharing**, **equivariant representations**, and it also provides a convenient way of dealing with inputs of variable size.
In traditional neural networks *every output unit interacts with every input unit*, while convolutional networks tipically have sparse interactions (also referred to as **sparse connectivity** or **sparse weights**). This is caused by the kernel being much smaller than the input, and that gives the ability to detect small and meaningful features (that occupy only a few thousands pixels) such as edges, corner or particular patterns. For a graphical representation of sparse interaction see Figures 1.7 and 1.8. This idea means that it's not anymore necessary to hold all the network's parameters at once, which both improves statistical efficiency and lowers the memory requirement.

Parameter sharing refers to using the same parameter for more than one function in a model, while in traditional networks each parameter is used only one time when computing the output of a layer. In convolutional networks each element of the kernel is used at every position of the input (except perhaps the boundaries of

the image); this means that rather than learning a single set of parameters for every location, and that reduces even more the storage requirements to k parameters.

See Figure 1.9 for a representation of parameter sharing in action.

The specific form of parameter sharing used in this networks causes the layers to have a property called **equivariance to translation**. A function is equivariant when if its input changes, the output changes in the same way: $f(g(x)) = g(f(x))$, and in the context of convolution this means that input and output of a layer are changing in the same way.

Practically speaking, when dealing with 2-D data like images convolution produces a **feature map** which is basically a map of where certain features appear in the image; if we move the object in the input its representation will move by the same amount in the output, so it's really useful to share parameters through the deeper layers of the network.

As an example, when the first layer detects edges it is really useful to share those parameters across the network because that same small edge will appear in a lot of different parts of the image.

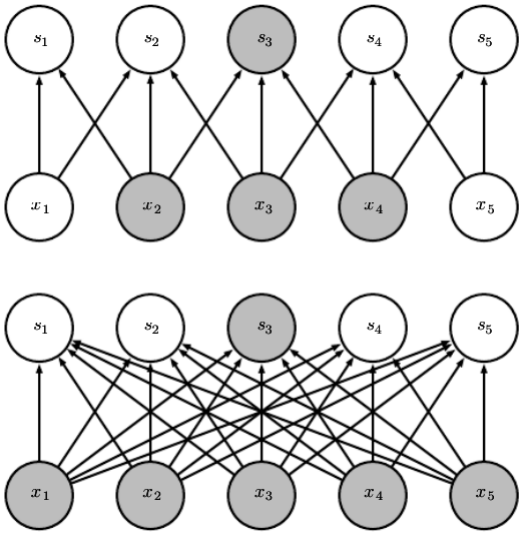


Figure 1.7: Graphical representation of *sparse connectivity from above*: the convolution with a kernel of size 3 gets the highlighted s_3 output unit to be connected only to input units x_2, x_3, x_4, in both networks the highlighted units form the **receptive field** of s_3

1.1.5 Pooling

A typical layer of a convolutional network consists of three stages (see Figure 1.10):

1. The first stage is called **convolutional stage** and performs several convolutions in parallel to produce a set of linear activations.

2. The second one is sometimes called **detector stage** and runs each linear activation through a nonlinear activation function, such as the rectified linear activation function.

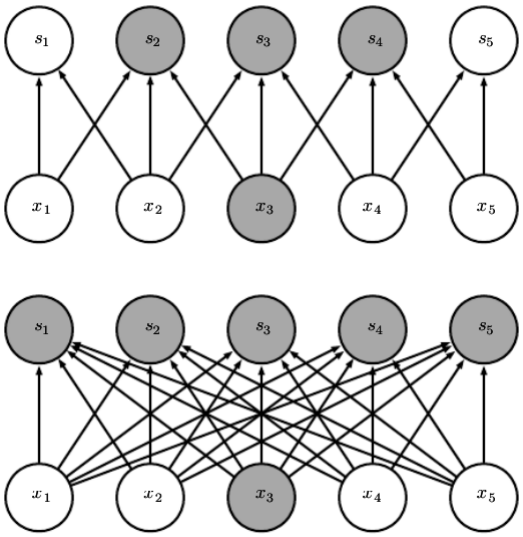


Figure 1.8: Graphical representation of *sparse connectivity from below*: when convolution is performed with a kernel of size 3 the input unit x_3 is connected only to output units s_2, s_3, s_4

3. The last stage is called **pooling stage** and uses a pooling function to further modify the layer.

A **pooling function** replaces the output of the network at a given location with some sort of statistic summary of the nearby outputs; let us consider the **max pooling function** as an example: this operation reports the maximum output value within a rectangular window (Zou and Chellappa, 1988). Independently from the pooling function used, this stage helps to make the representation become approximately **invariant** to small translations of the input. Invariance to local translations can be a very useful property if one cares more about whether some features are present in the image than exactly knowing their position.

Kernel learning, Backpropagation of the error

The feature extraction is made by the convolution between the image and a kernel. What a convolutive network learns is actually how to find the best values for the kernel so that the best representative features are extracted.

If we consider the values of the kernel as weights to learn, it becomes possible to use any kind of learning algorithm already used for traditional networks, for the goal is always the same: *minimize the loss function by modifying the weights of the network.*

The most important learning algorithm for neural networks based on gradient optimization is called **backpropagation**. This technique modifies the weights of the network backwards, from the output to the input, so that after calculating the loss function all the weights of the former layers will be updated in accordance to the error.

Backpropagation is an efficient method based on minimization of the loss function, therefore what it actually does is calculating the partial derivatives of each parameter of the network in order to apply a Gradient Descent Algorithm on the network.
It uses the **chain rule** to calculate the derivative of a composite function, keep in

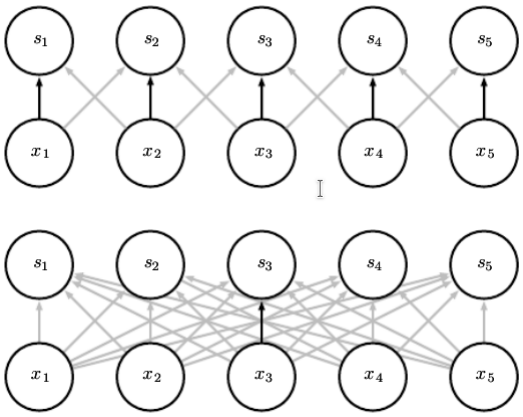


Figure 1.9: Graphical representation of *parameters sharing*: the arrows indicate the connections that use a particular parameter in both models. In convolutional networks thanks to parameter sharing only the kernel's parameters are shared between units

mind that each activation can be seen as a composite function made up by all the linear combination in input to its unit.

Let us consider a network with a single input unit, one hidden unit and one output: the derivative of the loss function against the weights to modify can be written as follows:

$$E(\sigma(z)), z = x_i * w_i \tag{1.14}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \sigma}\frac{\partial \sigma}{\partial z}\frac{\partial z}{\partial w_i} \tag{1.15}$$

This exact method can be used to modify the kernel's weights in accordance to the cost function. Assuming that:

- $x^l_{i,j} \longrightarrow$ it's the result of convolution stage at layer l
- $w^l_{m,n} \longrightarrow$ are the weights of the kernel used at layer l
- $o^l_{i,j} \longrightarrow$ it's the nonlinear activation function of layer l, executed on the output of convolution ($o^l_{i,j} = ReLU(x^l_{i,j})$)
- $b_l \longrightarrow$ is the bias for layer l
- $HxW \longrightarrow$ are the dimensions of the feature map produced by the layer
- $K1xK2 \longrightarrow$ are the dimensions of the kernel

it becomes possible to formally prove how to calculate the variation to apply for every single pixel of a kernel:

$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2}\frac{\partial E}{\partial x^l_{i,j}}\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} \tag{1.16}$$

In (1.16) $x^l_{i,j}$ is equivalent the result of the convolution performed between the kernel K for the layer l and the feature map produced by the previous layer *l-1*, so it could be

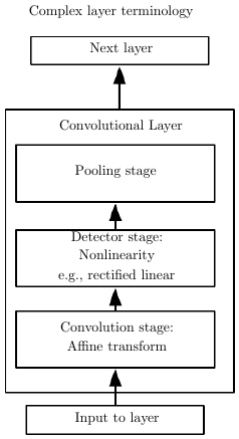


Figure 1.10: A schema of a typical convolutional neural network layer, usually the network is seen as a small number of complex *convolutional layer*, each one made up by a *convolution stage* followed by a stage that introduces nonlinearity called *detector stage* and in the end there's the *pooling stage* that shrinks the output to smaller dimensions

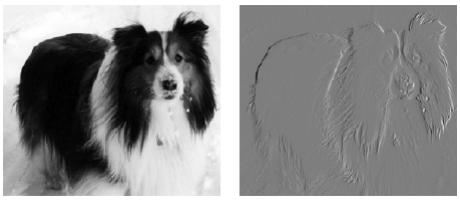


Figure 1.11: An example of *edge detection*: the image on the right was formed by taking each pixel in the original image and subtracting the value of its neighbouring pixels on the left

written as $x^l_{i,j} = \sum_m \sum_n w^l o^{l-1}_{i+m,j+n} + b^l$. By expressing $x^l_{i,j}$ in terms of convolution we get:

$$\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} = \frac{\partial}{\partial w^l_{m',n'}}(\sum_n \sum_n w^l_{m',n'} o^{l-1}_{i+m,j+n} + b^l) \tag{1.17}$$

The partial derivative is zero everywhere except for the one with indexes $m = m'$ and $n = n'$, therefore the output will be the pixels of the features map produced in layer *l-1* associated with the pixels of the kernel at level l:

$$\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} = \frac{\partial}{\partial w^l_{m',n'}}(w^l_{0,0} o^{l-1}_{i+0,j+0} + \cdots + w^l_{m',n'} o^{l-1}_{i+m',j+n'} + \cdots + b') \tag{1.18}$$

After deriving, the weights with indexes m', n' will become 1, therefore considering that every weight except those two becomes zero only the activation will remain: $o^{l-1}_{i+m',j+n'}$.
The last term of (1.16) to be calculated is $\frac{\partial E}{\partial x^l_{i,j}}$, that represents the actual error to backpropagate through the network and will be referred to as $\delta^l_{i,j}$:

$$\delta^l_{i,j} = \frac{\partial E}{\partial x^l_{i,j}} \tag{1.19}$$

this value states the variation of the pixel i, j in the feature map o^{l-1} in respect to the loss function. By applying the **chain rule** we obtain:

$$\frac{\partial E}{\partial x^l_{i',j'}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \frac{\partial E}{\partial x^{l+1}_{i'-m,j'-n}} \frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'-m,j'-n} \frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} \quad (1.20)$$

By subtracting the kernel's indexes to the ones of the feature map $(i'-m, j'-n)$ we obtain the points influenced by the previous layer within the region of convolution. Intuitively the a pixel that's an inpu for layer l will influence the value of much more pixels in the features map that goes in input to layer *l+1*.
The term $x^{l+1}_{i'-m,j'-n}$ represents the output of layer *l+1*, therefore it can be written as the convolution between the output of layer l and the kernel of layer *l+1*:

$$\begin{aligned}\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} &= \frac{\partial}{\partial x^l_{i',j'}} \left(\sum_{m'}\sum_{n'} w^{l+1}_{m',n'} o_{i'-m+m',j'-n+n'} + b^{l+1} \right) \\ &= \frac{\partial}{\partial x^l_{i',j'}} \left(\sum_{m'}\sum_{n'} w^{l+1}_{m',n'} f(x^l_{i'-m+m',j'-n+n'}) + b^{l+1} \right) \end{aligned} \quad (1.21)$$

In equation (1.21) only the weights with indexes m', n' and the activation functions applied to pixels i', j' will have values, every other term will be equal to zero:

$$\begin{aligned}\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} &= \frac{\partial}{\partial x^l_{i',j'}} \left(w^{l+1}_{m',n'} f(x^l_{0-m+m',0'-n+n'}) + \cdots + w^{l+1}_{m,n} f(x^l_{i',j'}) + \cdots + b^{l+1} \right) \\ &= \frac{\partial}{\partial x^l_{i',j'}} \left(w^{l+1}_{m,n} f(x^l_{i',j'}) \right) = w^{l+1}_{m,n} \frac{\partial}{\partial x^l_{i',j'}} \left(f(x^l_{i',j'}) \right) = w^{l+1}_{m,n} f'(x^l_{i',j'}) \end{aligned} \quad (1.22)$$

The partial derivative of equation (1.20) can be rewritten as:

$$\frac{\partial E}{\partial x^l_{i',j'}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'-m,j'-n} w^{l+1}_{m,n} f'(x^l_{i',j'}) \quad (1.23)$$

It is now possible to **formalize backpropagation**:

$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2} \frac{\partial E}{\partial x^l_{i',j'}} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} = \frac{\partial E}{\partial x^l_{i',j'}} \delta^l_{i,j} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} \quad (1.24)$$

where $\delta^l_{i,j}$ is obtained via equation (1.23) while $\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}}$ corresponds to $o^{l-1}_{i+m',j+n'}$. By using the convolution operator $*$ between those two components a matrix will be generated which will calculate the kernel's values when it gets applied to a region of the output feature map.

1.2 Why studying climate change?

The Earth's climate has continuously changed through history, just in in the last 650.000 years there have been seven cycles of glacial advance and retreat; the last ice age ended about 11.700 years ago and set the start of the modern climate era.

The very most part of these changes are considered to be the cause of small variations in our planet's orbit, that consequently change the amount of solar energy the Earth receives.

According to the Fifth IPCC assessment report on climate change [12], even if there have been cycles in climate change, the current warming trend is of particular significance because it's most likely mankind's fault (with a probability higher than 95%), since it started in the mid twentieth century with the first industrial revolution and has continued to grow at unprecedented rates, never seen in history.

Thanks to all the satellites orbiting around our planet, scientist have been able to demonstrate that *carbon dioxide and other pollution-related gases have a heat trapping nature*, their ability to affect the transfer of infrared energy through the atmosphere is the scientific base of many instruments used by NASA. It is therefore safe to say:

There is no question that increased levels of greenhouse gases must cause the Earth to warm in response

Even if Earth isn't new to climate changes as said before, evidences like ice cores drawn from Greenland or Antarctica, tree rings,inco ocean sediments, reveal that the current warming is occurring almost ten times faster than the ones happened after ice ages, as seen in Figure 1.12.

Such an unprecedented increase in temperatures is having observable effects on al-

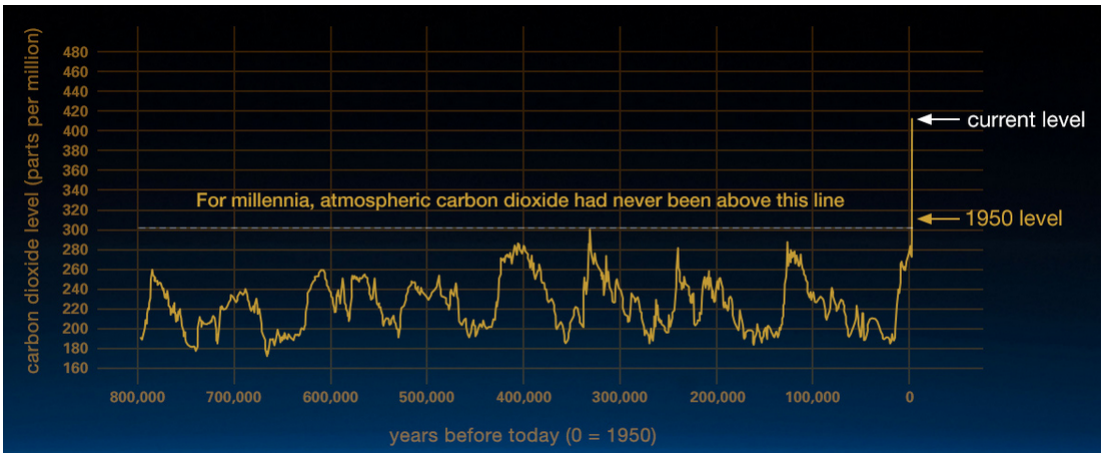


Figure 1.12: History of climate change, comparison of both atmospheric samples contained in ice cores and more recent direct measurements; a cyclic trend emerges, the lowest points are the ice ages

most every kind of environment present on Earth: glaciers are shrinking more and more, ice on rivers and lakes disappears sooner than ever, plant and animal ranges have shifted and even trees are flowering sooner.
Each one of these consequences could be considered not harmful in itself, but the indirect damages caused are almost unimaginable: for example, loss of sea ice is actually making every sea in the world to rise alarmingly, causing both loss of dry land (fundamental for agriculture and therefore our sustenance) and longer, more intense waves that further accelerate this process.

It is now a common belief amongst scientists that global temperatures will continue to rise for decades to come, due to all the greenhouse gases produced by human activities.

The Intercontinental Panel on Climate Change (IPCC), which includes more than a thousand of scientists from almost every part of the world, has forecasted a rise of up to 4 degrees Celcius over the next century.

Climate change affects economy

According to IPCC's scientists, *the range of published evidence indicates that the net damage costs of climate change are likely to be significant and to increase over time.*

Every sector of economy is both the victim and the murderer; in Figure 1.13 there's a pie plot describing estimates values of how much each sector is responsible for the production of greenhouse gases. Intuitively enough the transportation and electricity sectors are the most impacting ones, mostly because for both of them a large role is still played by burning carbon fossil fuels (over 90% of transports worldwide are still using petroleum based fuels).

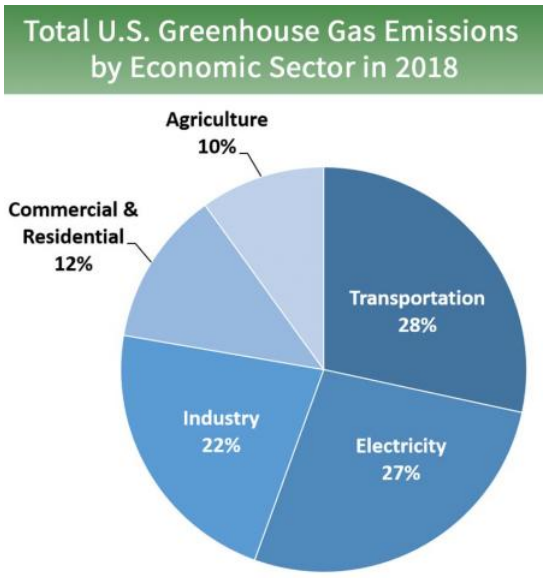


Figure 1.13: Greenhouse gases emissions by economic sector, source USA Environmental Protection Agency. In 2018 a total amount of 6,667 million metric tons of CO_2 equivalent has been produced.

How will the economy be affected by this changes?
As said before maybe the biggest risk is the sea level rising due to the increasing temperature, but also the temperature in itself could easily become a problem, because it initiates a chain of bad outcomes:

1. Coastal areas are currently being adapted to the increasing sea level
2. If the forecasting will become real in future, a whole lot of really difficult challenges will arise, such as:
    - Relocation of whole towns
    - Greatly reduced productivity of harvest due to the lack of dry soil
    - Therefore, prices of basic food will rise exponentially

- More wars, to gain possession of the limited resources such as soil and drinkable water

3. Extreme meteorological phenomena will become more frequent, causing widespread poverty
4. Higher temperatures will also ease the spread of diseases

Another important factor to consider is that workers worldwide will be more likely to suffer from really bad condition of life, therefore diminishing greatly their productivity. All the health systems worldwide will be overwhelmed and therefore health conditions of individuals will be a great problem in the future.

How are we fighting?

The very first cause of climate change is mankind itself: our need for commodity always and everywhere is the main cause. A lot of examples could be made in this regard: we want to have perfect temperature in our houses both in winter (by warming them) and in summer (by cooling them), we use carbon-fueled transports even when a walk could be absolutely feasible, the list could go on and on for a long time, sadly.

Luckily in these last years a lot of initiatives carried on by governments, scientists and citizens are quickly rising awareness about this topic.

As an example consider the beautiful project *Fridays for future* founded by Greta Thunberg: even if it's nothing else than public informative events where the topic of climate change is discussed, it brought hundreds of thousands of people (mostly youngsters) together listening to all these facts, and maybe it will bring good changes in everyone's behaviour.

The most important change must come from each and every one of us humans; it's mandatory to change our behaviours to a more virtuous standard, otherwise all the efforts made by any organization in the world will have close to none effects.
There are some technologies that try to capture pollution from the air via complex air filtering techniques, but they surely are too costly to be an actual solution to the problem. Lots of efforts are also being put in developing efficient and accurate methods to monitor pollution worldwide, ranging from satellites to pollution monitoring stations.

1.3 CleanAir Project

CleanAir Project began in December 2017 in the context of a Climathon: an hackaton focused on ideas about fighting climate change, held in thousands of cities across the world each year.

I participated to the challenge with a few colleagues from university just for the fun of taking part to an hackaton; we had 24 hours to come up with an innovative idea, make a slideshow and try to convince the judges that your team is the best one.

Initially I spent some time in researching news about climate change, with the hope that they would give me good ideas; after a while I came across the problem of monitoring the pollution in cities, a challenge still to overcome due to the lack of both

reliable and inexpensive technologies.
Sadly, climate change is not a problem that affects us at the current time (at least, not with its most severe consequences) and therefore it's highly unlikely that organizations such as governments will ever be positive in spenging lots of money in this regard, and that's why I thought that one of the key concepts in my idea should have been the inexpensiveness.

The biggest constraint to monitoring large territories is that a large number of monitoring stations is needed, due to the fact that air (and the particles it contains) moves around because of the wind and any other weather factor. But handling large numbers of stations is not a feasible challenge to anyone, not only due to the cost of actually building and deploying lots of stations, but also due to the fact that each station will require maintenance as time passes by.

Then I also thought that in Artificial Intelligence *the power lies in the numbers*, meaning that usually the only constraint is that a lot of data is needed to let the system learn; therefore I came up with the idea behind CleanAir Project: *Using a small number of* ***mobile*** *monitoring stations, mounted on top of municipal vehicles.*

By placing the mobile stations on top of the municipal vehicles the problem of having lots of static stations falls, because those vehicles (coaches, police cars, any kind of service vehicle actually) are used by the workers whom by doing their job will eventually cover the city perimeter, therefore greatly reducing the number of needed stations.

Me and my colleagues implemented this idea, thanks also to the help of the University and some professors who provided help and resources as much as they could, in the context of a national challenge organized by CINI (Comitato Interuniversitario Nazionale per l'Informatica).

The challenge was called CINI Smartcities 2018 and as the name suggests, the idea was having teams of undergraduates challenging each other with actual implementations of ideas in the field of Computer Science applied to Cities.
CleanAir was presented at the challenge in combination with FireStop, another environmental project developed by other colleagues; after some rounds of selections, the challenge ended at the Gran Sasso Science Institute (L'Aquila, Abruzzo, Italy) where me and my team got the second place amongst 15 other Italian universities, thanks to the combination of CleanAir Project and Firestop.

Also, this thesis was accepted to participate in CINI 2019 Ital-IA [13] conference, where all Italian researchers and research groups on Artificial Intelligence presented what they were working on; even if this is an undergraduate thesis I have been accepted to the conference and given a 10 minute speech to explain what I was working on, it has been a truly wonderful experience.

1.3.1 App architecture

The application has three major modules:

- Data gathering
- Data analysis and storage
- Data visualization

we'll now analyze and explain each module in depth, to show how all these tasks are accomplished. A high-level view of the app's architecture is shown in Figure 1.14.

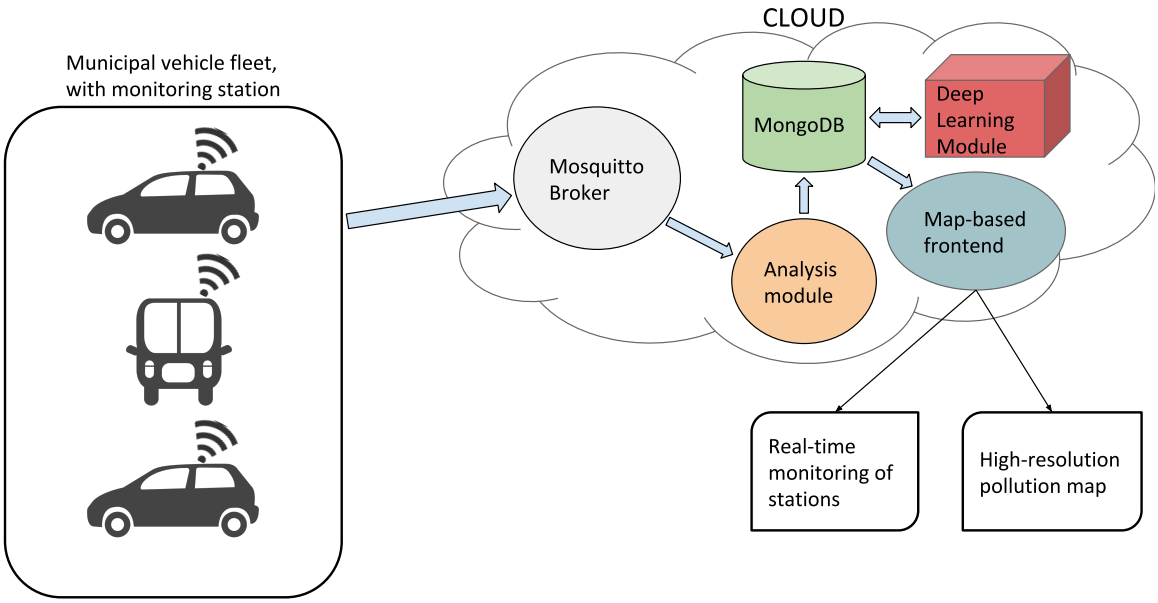


Figure 1.14: High-level schema of CleanAir Project architecture

Data gathering

This module includes both how the stations work and how to gather data from the monitoring stations and send them to the cloud; it is accomplished by means of the MQTT protocol used in Mosquitto's implementation.

Figure 1.15 shows a schema of the station, which is composed by:

- Arduino Nano. Used to read raw misurations from the sensors.
- Raspberry PI 3B. Used as main computing unit, it receives data from the Arduino, bufferizes it and after a fixed amount of time a mean data is sent to the cloud, in order to avoid peeks as much as possible.
- GPS. So that each station can communicate its position
- Sensors. Being still in a proof-of-concept state, the station has only pretty cheap sensors, one for Particulates (PM10/2.5) and one "Air Quality sensor" able to detect different gases depending on how it is (manually) calibrated

Data analysis and storage

This module receives the data sent by the stations (e.g. it's subscribed to the same topic where the stations send the data), aggregates them and saves the result in a MongoDB instance.

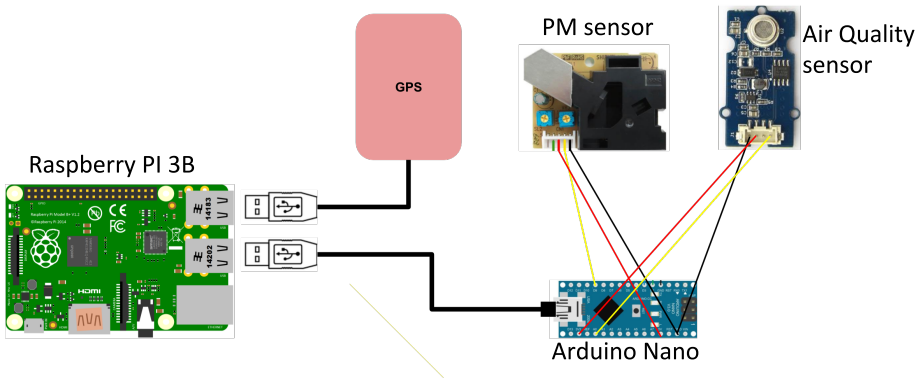


Figure 1.15: The proof-of-concept monitoring station

The whole city upon which CleanAir is applied would be divided in a fixed number of cells in a grid topology, these cells are saved as a separate collection in the database using the following format:

```
{
    cell_id: integer,
    coordinates: latlon format,
    samples: {}
}
```

When a new sample is sent by a station, this module finds the cell whom the sample belongs to via the *haversine formula*, that calculates the great-circle distance (the shortest distance over the earth's surface) between two points:

$$\begin{aligned} a &= \sin^2(\Delta\frac{\phi}{2}) + \cos(\phi_1)\cos(\phi_2)\sin^2(\Delta\frac{\lambda}{2}) \\ c &= 2\arctan^2(\sqrt{a}, \sqrt{1-a}) \\ d &= R \cdot c \end{aligned} \tag{1.25}$$

where ϕ is the latitude, λ is the longitude, R is the Earth's radius (mean radius = 6,371km) and d is the distance in meters between the two points.

Once the correct cell is found, that sample is inserted in the field *samples* of the cell, so that it will be easy to know what data belongs to a specific cell.

The data is then analyzed by the Deep Learning Model, that produces the high-resolution pollution heatmaps; being it the core of this thesis, it wil be discussed in greater detail in the next chapters.

Data visualization

This module is the only one actually interacting with the user, it offers a simple graphical interface that offers two main services:

1. Real-time monitoring of the stations. Figure 1.16, each active station is seen on the map as a marker, the user can click on it and see what data that station is sending and can also decide to follow a single station; in that case all the other markers will disappear and the map will focus its center on the selected station

2. High-resolution pollution map. In this proof-of-concept the system draws a heatmap on the city, with a spatial resolution of $10000m^2$ and lets the user explore said map

The interface has been developed as a Single-Page-Application using modern frontend technologies such as React, Javascript and CSS.

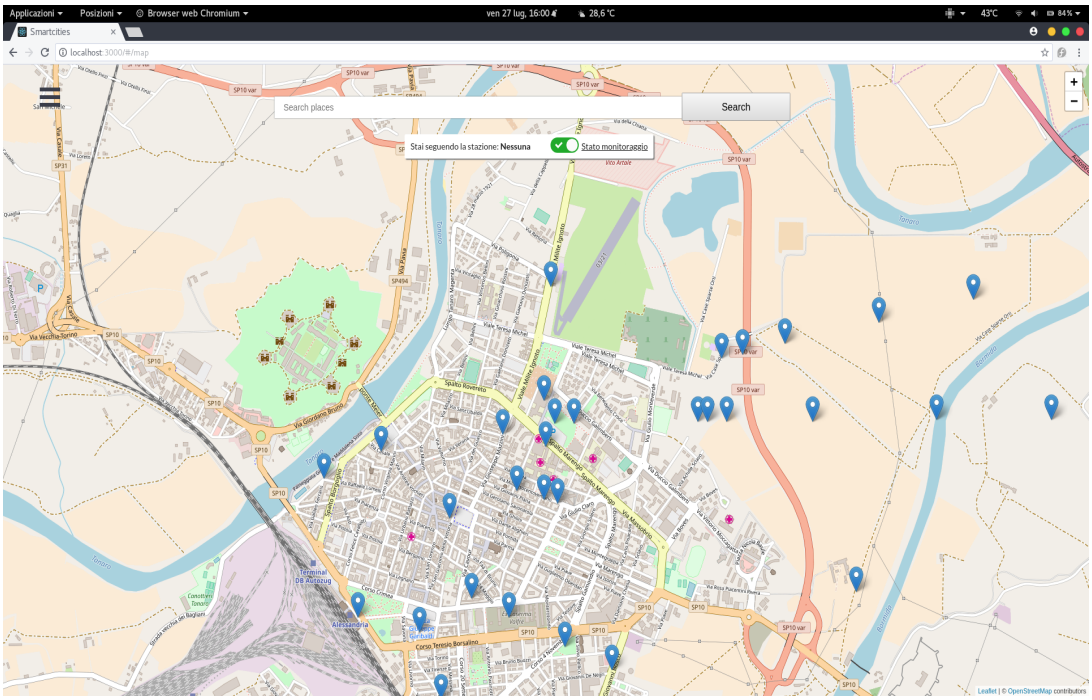


Figure 1.16: An example of the graphical interface: each marker is the position of a station; the markers visible on the map are actually simulated for the sake of having enough data to show

1.4 OpenSense Zurich

One of the main problems about actually implementing this project was the lack of data: it would have been absolutely unfeasible to actually build all the stations needed and furthermore it was impossible to let them take measurements for a sufficiently long time.
That is because the data on which we are trying to apply deep learning techniques are about weather, therefore in order to have statistically significant data we must at miminum have a whole year worth of measurements, so that every season would be monitored equally.

Due to the unavailability of good data, I started to search for similar projects and luckily I found OpenSense project from Zurich [21]. It is a research group that aims at investigating community-based sensing using a network of wireless sensors to monitor

air pollution.
They focus on creating efficient and reliable methods to sense air pollution via a fleet of mobile sensors, mostly on the infrastructure side. In Zurich they experimented a fleet of ten mobile sensors mounted on top of streetcars of the public transport system (Figure 1.17), gathering more than two years of data.

Thankfully OpenSense made the data publicly available to the scientific community worldwide, and considering the strong resemblance with CleanAir Project, I started working on those rather than the ones that me and my colleagues gathered in Alessandria, while experimenting our proof-of-concept monitoring station.

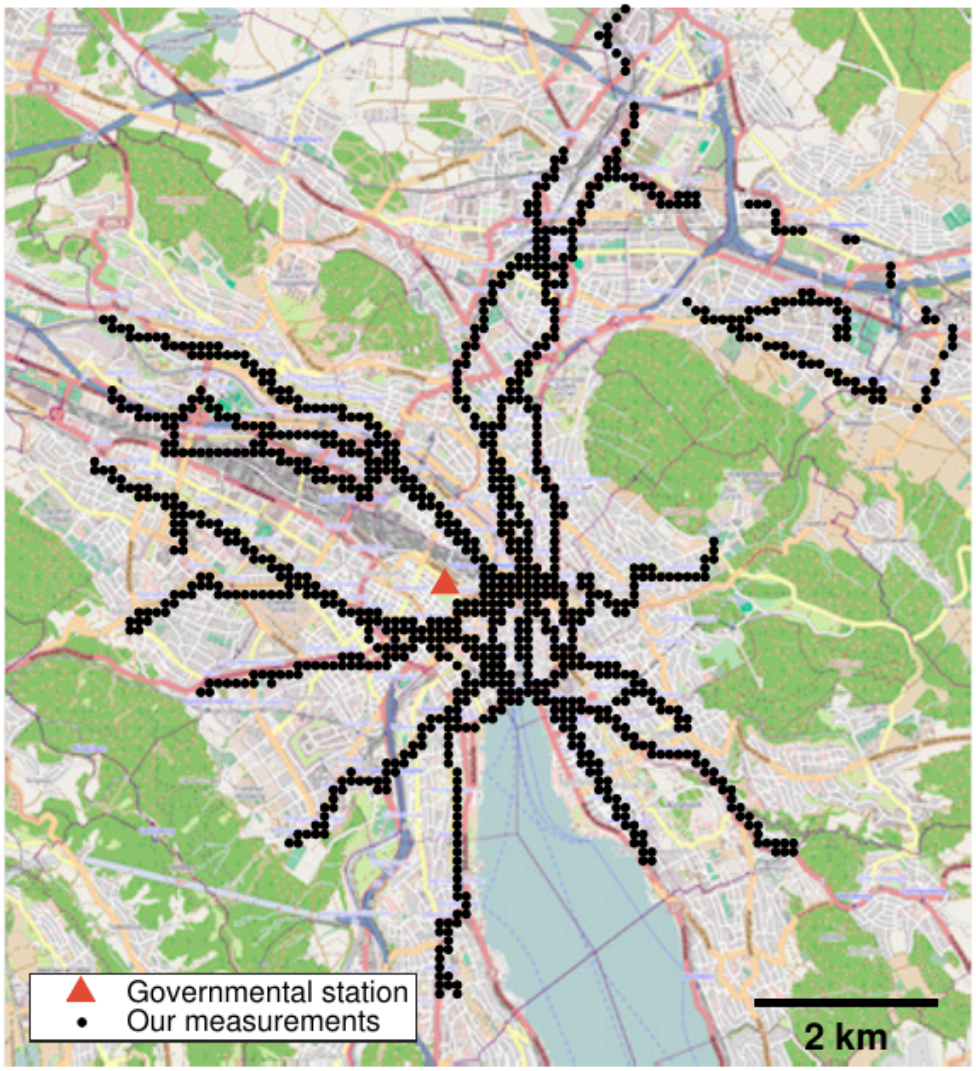


Figure 1.17: OpenSense's stations in Zurich: ten public transport streetcars are equipped with a monitoring station; the dots denote locations with at least 50 measurements over the course of two years

1.4.1 The station

Being OpenSense a professional organization, they have been able to develop a far more efficient and reliable monitoring station than the proof-of-concept me and my

colleagues built for the CINI Challenge 2018. Their stations are equipped with very good sensors and are also really well engineered in order to avoid water and dust entering the insides, see Figure 1.18.

Each station is operating when its streetcar is active, so on average 20h per day; typically from 1:00 AM to 5:00 AM the streetcars are off so no measurements are taken. All data is transmitted via a GSM module to the cloud, it's also important to notice that the stations buffer the data for a fixed amount of time, after that time has passed it sends a mean value of the gathered measurements.

For a complete and in depth explanation of how the stations work (how data is pre-processed before being sent to the cloud, what specific sensors are used) the reader can refer to the paper "Deriving High-Resolution Urban Air Pollution Maps Using Mobile Sensor Nodes" [4], upon which I based my entire work.

(a) Streetcar with sensor node on top (red circle).

(b) Inner view of the air quality sensor node.

Figure 1.18: (a) the stations are mounted on top of streetcars. (b) Inner view of the station, equipped with UFP, CO, O_3, NO_2 sensors, a GPS module and a GSM module

Chapter 2

Work description

The ultimate goal of this project is to have high-resolution pollution maps, in order to give a city's government the ability to know exactly how the air quality is, in the most detailed way possible.

CleanAir Project has a really big limitation: cheap monitoring stations can be a really good ally in monitoring air quality, but the measurements are made with far too imprecise sensors to be usable in a legal way.

This means that if a station always measures too high levels of pollution near a specific building (e.g. a productive site of a company) the gathered data cannot be used in a court, the council cannot press charges. Therefore rather than being an instrument of maximum precision *CleanAir aims at being a support tool in decision making for the planning of city-wide strategies against pollution.*

Being this a rather unexplored field, there weren't too much studies already completed when I started working on this project; only the OpenSense Project in Zurich had a publicly available dataset tailored exactly for my needs, therefore I based my work on them, in order to have a comparison possibility against such a trustworthy study.

In this chapter both the OpenSense approach and the CleanAir one are presented.

2.1 OpenSense approach

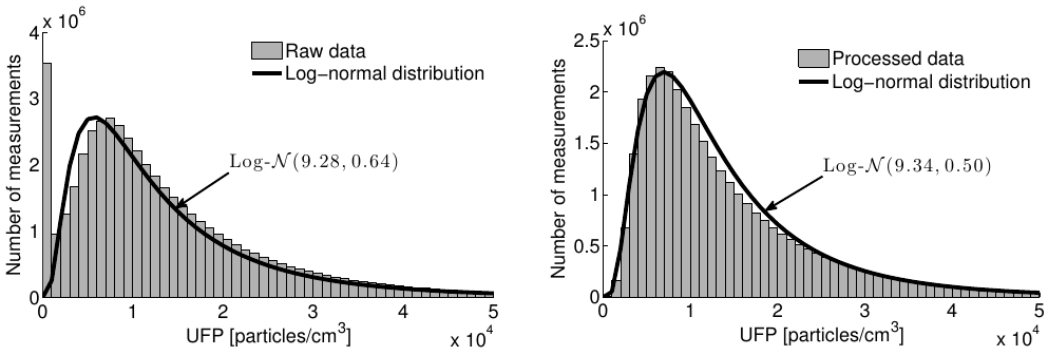
The researchers in OpenSense proposed a complete method to produce high resolution pollution maps over Zurich [21] specifically for Ultrafine particles (UFPs), which are particles with a diameter of less than 100nm; in ambient air the UFPs are mainly man-made as bioproducts of specific high temperature processes such as a car engine running.

They propose an approach based on mobile monitoring stations in order to enhance the spatial resolution by trading off temporal resolution. The basic idea is the same as CleanAir Project: *a fleet of mobile monitoring stations can cover a large area without the need of a lot of elements.*

They deployed ten monitoring stations and put them on top of public streetcars for the city of Zurich, for a more detailed explanation on this regard take a look at 1.4.

2.1.1 Data preprocessing

Good data quality is a must for the development of reliable pollution maps, that's why the research group has put a lot of focus in preprocessing raw data by calibrating the sensors and filtering the results; Figure 2.1 gives a graphical representation of the results of preprocessing and filtering the data.



(a) Raw data. (b) Processed data.

Figure 2.1: (a) Raw data and their log-normal distribution. (b) Preprocessed and filtered data, the log-normal distribution now fits much better

The actual process of validating the data is as follows:

1. Analyze the statistical distribution of the monitored particles concentrations. usually ambient air pollutants closely follow a log-normal distribution (see Figure 2.1) Log-$N(\mu, \sigma^2)$.

2. Evaluate the baseline signal of each device. They examine the correct offset of each MiniDiSCs (the pollution sensor) by looking at their baseline signals, *i.e.* low-pass filtered measurements; a similar baseline signal is expected across all devices, because they take measurements in the same region. As baseline signal *they take every 90 minutes* (the average time spent by a streetcar to cross the city twice) *the 20th percentile of the measured concentrations.*

3. Compare the produced measurements against the samples took by two high-quality stations in the same time period but in different locations in Switzerland. See Figure 2.2.

In conclusion the corrected statistical distribution, the matching devices baseline signals and the good correlation between the processed data and the NABEL produced ones, guarantee a good quality of the processed data, therefore suitable to be used for producing accurate high-resolution pollution maps.

2.1.2 Data aggregation

Temporal data aggregation

The processed and filtered data are **aggregated in different time periods and for each time period a separate model is learned**, yielding to 989 models with yearly, seasonal, monthly, biweekly, weekly, daily and semi-daily (meaning from midnight to midday and vice versa) temporal resolutions.

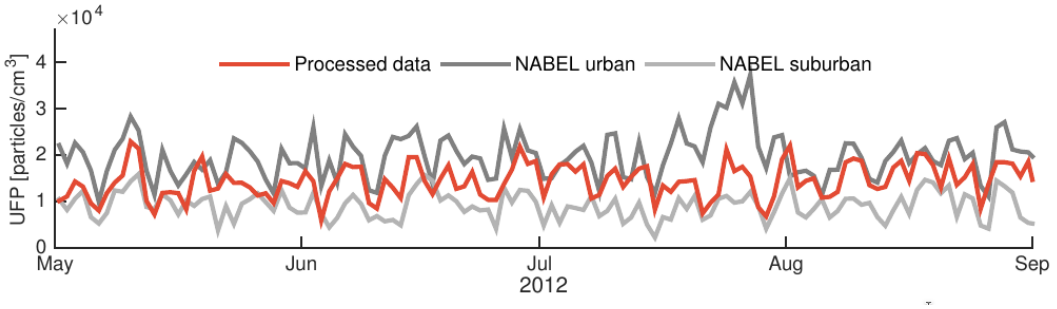


Figure 2.2: Daily average UFP concentrations measured by OpenSense in Zurich (red) against the two high-quality stations called NABEL at both urban (Pearson r=0.49) and suburban (r=0.55) locations

Spatial data aggregation

The measured UFP concentrations are projected on a grid with 100m x 100m cells that completely covers Zurich. In their deployement they had a large number of measurement location but they found out that with many locations it was difficult to get predictions with high R^2 values; they later found empirically that restricting to use the first 200 cells with higher number of measurements brought them the best results.

The measurements are unevenly distributed among 300-1300 different cells, depending on the analyzed temporal subset of the data; as an example in Figure 1.17 only the cells with at least 50 samples are showed, when projecting *the full two-year data set on the grid*.

2.1.3 Regression

In order to create those high-resolution pollution maps the **Land-Use Regression models** (LUR) are a powerful ally, as they are widely used to asses spatial variation of air pollutants, typically at an intraurban scale [1].

LUR models use land-use informations and traffic characteristics (also called *explanatory variables*) to predict pollution levels for location non covered by measurement devices; the general idea is as follows:

1. At each measurement location (locations having an actual measurement device) the dependencies between the explanatory variables and the monitored pollutants are evaluated via linear regression.

2. The relationships found in 1. are used to predict concentration levels at locations without measurement devices, but with available land-use data.

The regression task is performed by a Generalized Additive Model (GAM) for it supports non-linear relationships between the monitored concentration levels and the explanatory variables and such models have been successfully used to analyze and model the spatio-temporal variability of particulate matter [11].

Explanatory variables

The researchers at OpenSense selected 12 explanatory variables, to be used with the LUR model, depicted in Table 2.1. The variables related to the terrain (building hei-

Variable [unit]	Variable [unit]
Population [inhabitants/ha]	Industry [industry buildings/ha]
Building height [floor levels/ha]	Heating [oil and gas heatings/ha]
Terrain elevation [average m/ha]	Road type [busiest road type/ha]
Distance to next road [m]	Distance to next large road [m]
Terrain slope [average degree/ha]	Terrain aspect [average degree/ha]
Traffic volume [vehicles per day/ha]	Distance to next traffic signal [m]

Table 2.1: Detailed list of the 12 explanatory variables used by OpenSense for their LUR model

hgts, altitude, aspect of the terrain itself) are statistical estimates fetched from the Swiss Federal Statistical Office, while the variables related to streets (road type, distances to next roads, traffic signals) are extracted from OpenStreetMap. The average daily traffic volumes are obtained from the Department of Waste, Water, Energy and Air of the Canton Zurich.

Not all the variables have been used, due to their pair-wise correlation: *removing variables that have high correlation with each other helps to better distinguish individual contributions of different variable to the predicted UFP value.* Both population density and number of gas and oil heating households are not used due to their high linear relationship with each other and with the number of floor levels.

They also empirically found that using the distance to the next traffic signal didn't improve any of the models, so it isn't used.

Generative Additive Model

To actually predict the UFP concentration the following GAM is used:

$$ln(c_{num}) = a + s_1(A_1) + s_2(A_2) + \cdots + s_n(A_n) + \epsilon \tag{2.1}$$

where c_num denotes the UFP concentration, a is the intercept, $s_1(A_1) \dots s_n(A_n)$ denote the smooth functions s applied to the explanatory variables A, ϵ is the error term.

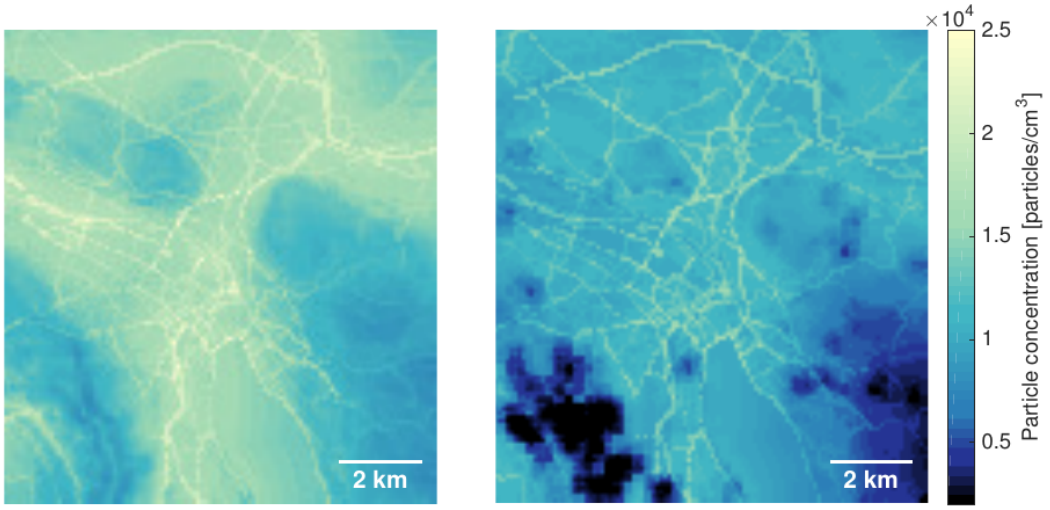
As for the GAM input, depending on the chosen temporal aggregation, *only the n cells with highest number of measurements are used* (for the full two-year data set the first 200 cells with highest number of measurements are used).

Usually those cells are the ones containing a tram stop, this ensures that the calculated means c_i^m are reliable and provide a good trade-off between spatial distribution of the input and model perfomance. They empirically discovered that introducing prior weights to the cells did not bring any improvements to the model.

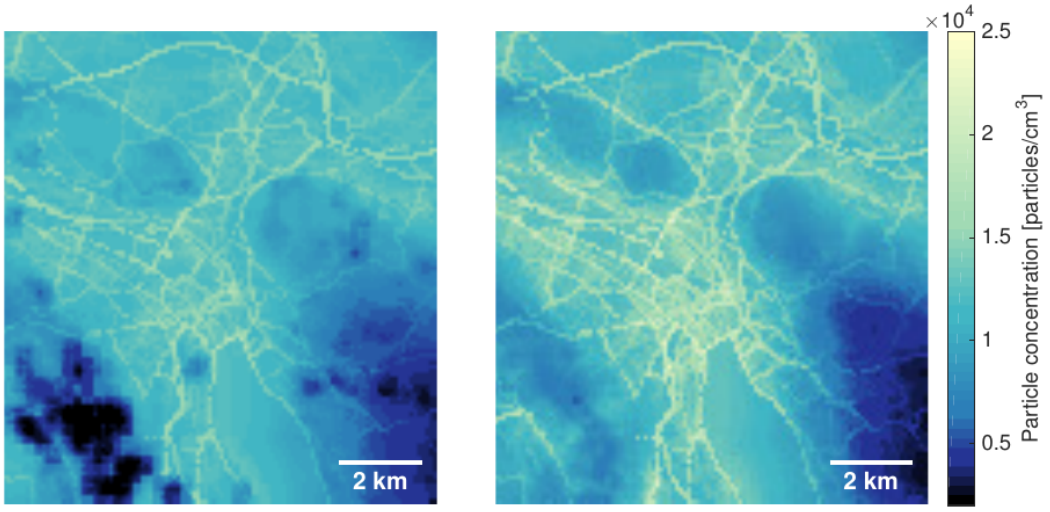
2.1.4 Model evaluation

To evaluate the performance of the produced models the researchers of OpenSense use three standard metrics:

- *Factor of 2 measure (FAC).* Quantitatively analyzes the scatter plots of predicted UFPs against the measured ones by evaluating the fraction of data points that



(a) Winter (January–March). (b) Spring (April–June).



(c) Summer (July–September). (d) Fall (October–December).

Figure 2.3: Output of the seasonal model, seasonal UFP concentration heatmaps with a spatial resolution of 100m x 100m

lie inside the two factor area, which is the fraction of data that satisfies the following:

$$0.5 \leq \frac{c_i^P}{c_i^m} \leq 2.0 \tag{2.2}$$

where c_i^P is the predicted value and c_i^m is the measured one, all for the cell i of the grid. This measure is used based on the assumption that an accurate model for pollution maps should have a relative scatter less than a factor of two. [3]

- *Root-mean-square-error (RMSE).* Quantifies the difference between the predicted and measured particles concentrations, this metric is generally used when large errors are to prevent as much as possible, due to the error being squared:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(c_i^P - c_i^m)^2}{N}} \tag{2.3}$$

- *Adjusted coefficient of determination (R^2).* Indicates from 0 to 1 how well the predictions fit the measurements, meaning that having $R^2 = 1$ denotes a perfect fit. This measure reflects the linear relationship between the predicted values and the measured ones, hence, it's insensitive to additive and multiplicative errors.

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} c_i^m$$
$$R^2 = \frac{\sum_{i=1}^{n}(c_i^m - c_i^P)^2}{\sum_{i=1}^{n}(c_i^m - \bar{y})^2} \tag{2.4}$$

2.1.5 Application model

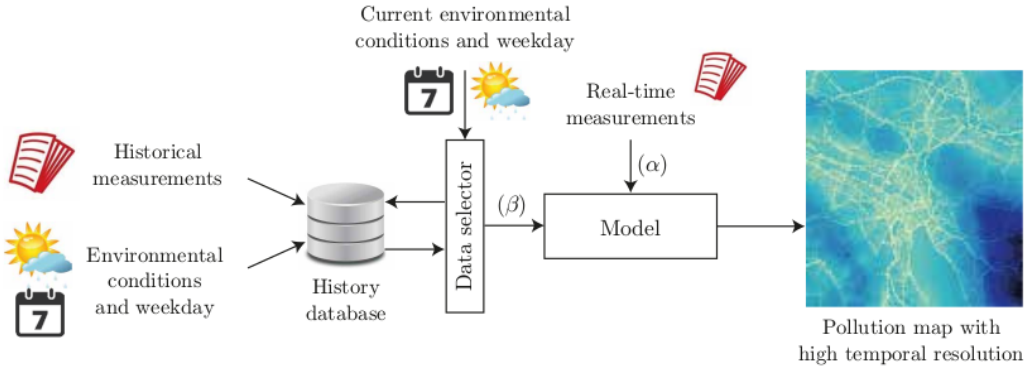


Figure 2.4: The complete conceptual model of OpenSense's application. Additional data from a database with historical measurements (β) is used to enhance the original dataset (α). The data selector picks only historical measurements which were measured under similar (e.g. similar average condition in the modeled time period) environmental conditions are used.

Figure 2.4 shows the complete conceptual model of the application produced by OpenSense; the history database is used to enhance the model capabilities on daily and semi-daily time periods, where the lack of data was inhibiting the model from performing well. Using this database reduced the RMSE by 26% in said time periods.

The data selector is the key component to enhance the data set, because its job is to fetch historic measurements based on the average environmental condition of the modeled time period, which are the most likely to be similar to the real measurements of that time period.

They empirically decided which environmental variable to consider and how much fuzziness to concede to the selector, for example allowing 20% of deviation on 15° temperature means that the selector will return all measurements that have 15° ±20%.

2.2 CleanAir Project approach

Having OpenSense's work as a baseline, I propose a different method to produce high-resolution pollution map given a fleet of mobile monitoring stations.

The proposed method uses a similar approach to the problem of data aggregation, but uses a Convolutional Neural Network to automatically extract the explanatory

variables used by OpenSense and a Multilayer Perceptron to actually perform the predictions (instead of the Generalized Additive Model used by them).

The biggest challenge in data science is always about the data: having large amounts of good quality data is the *conditio sine qua non* to solve a problem. In my case to have enough data was simply impossible, due to the lack of reliable monitoring stations (me and my colleagues built one prototype only) and also at least a year of measurements is mandatory to let the system learn how seasons progress and relate to each other.

Thankfully I found about OpenSense's work and I have been able to download a complete year of measurements made by them, therefore with a wonderful spatial coverage and also a lot of already done pre-processing work (as explained in Section 2.1.1).

2.2.1 Network

The Neural Network used to make the actual predictions is a slightly modified version of the traditional LeNet network (Figure 2.5).

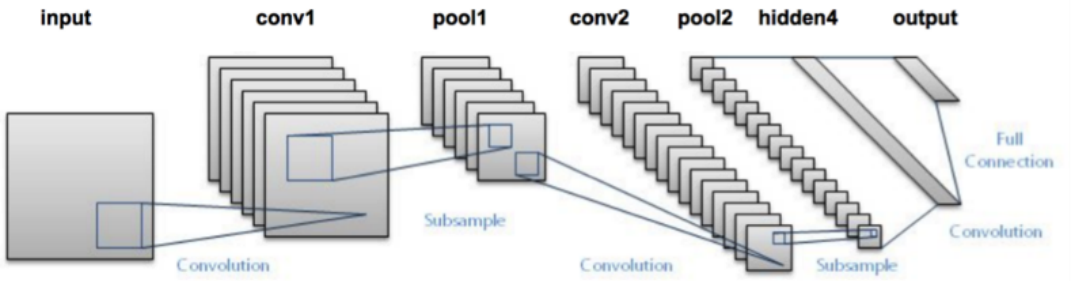


Figure 2.5: The traditional LeNet convolutional network

Figure 2.6 shows a schema of the network used, the main principle is that *given a portion of map and some weather data, we want to predict UFP concentrations*; now each component is presented.

Convolutional Layers

This is the part of the network in charge of automatically extracting the describing features of a given map. The basic idea is extracting unique explanatory variables for each cell, rather than using a fixed and static set of variables as OpenSense does.

We want to automatically generate those variables so that *each portion of map has its own explanatory variables*, represented by the feature map produced by the Convolutional Layers.

Generally the main purpose of the convolutional layers in a CNN is the **feature extraction**, meaning that the kernels applied in those layers are adjusted in order to produce the most useful feature map for the given task (as explained in 1.1.4).

In this context, the kernels applied are adjusted to obtain a feature map that sufficiently describes a geographical area, which is the image of the map provided in input.

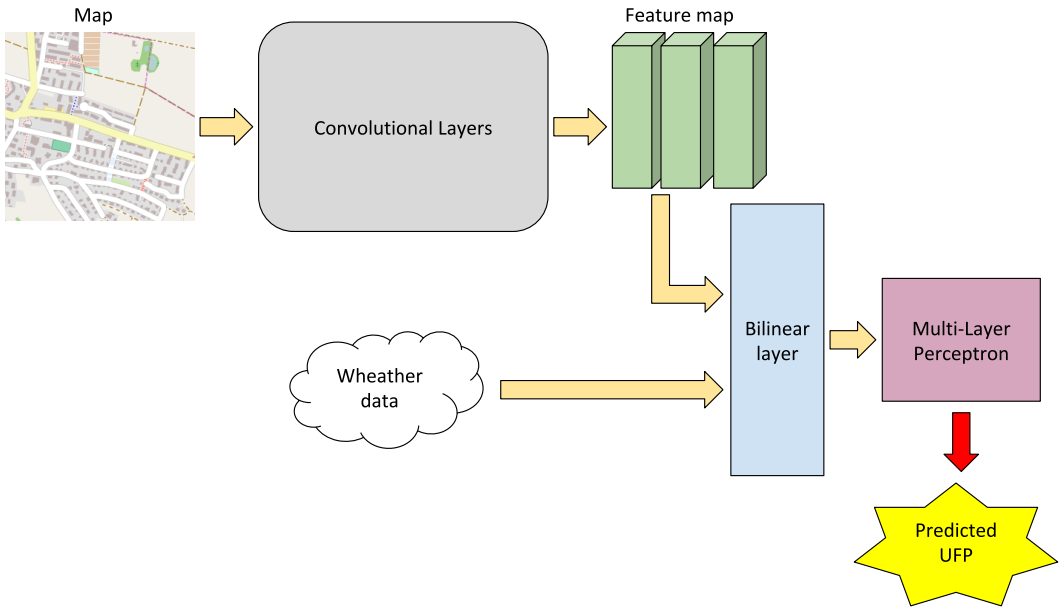


Figure 2.6: High-level schema of the implemented network, a slightly modified LeNet network

Bilinear Layer

The fundamental block of this network is this layer: it applies a bilinear combination to two inputs of any shape. It is often used to combine two different convolution streams in single feature map, for CNN used in classification tasks.

The function applied is the following:

$$y = x_1^t A x_2 + b \tag{2.5}$$

where x_1 is the feature map produced by the convolutional layers and x_2 are the weather data; A and b are learned, giving the model the ability to learn how to combine the map with the weather data in the best way.

The fundamental problem that was keeping the model from learning properly was that each cell is used more than one time (for example, in a daily time frame each cell holds 7 samples, so it will pass through the network 7 times) and therefore the final Multilayer perceptron would see the same image over and over, with only the weather data and the label to predict actually changing. Given that a 128x128 pixels image is actually a matrix of 128x128 values, and the weather data set has 8 features, using a standard Linear layer would basically cause the weather features to have close-to-zero impact on the prediction.

By seeing the same map portion with only varying weather data, the model wasn't able

to learn enough correlations to actually predict the UFP value. With the introduction of this Bilinear layer, each map image is combined with the weather data used.

Let us consider an example, on a daily time frame the model will see 7 sligthly different images for each cell because every cell produces 7 samples, each of which is combined with the map image associated with the cell, eventually causing the final Multi-Layer Perceptron to receive 7 different representations of the same cell. Without this layer, the Perceptron would see 7 equal feature maps and just 8 varying features, but considering that feature maps usually have hundreds of thousands of features those 8 values couldn't make a noticeable impact on predictions.

Final MLP

The final Multilayer perceptron has the task of actually making the prediction about UFP concentrations based on the output of the Bilinear layer. It is a standard MLP with a relatively high number of nodes, it is composed by 4 fully connected layers of neurons, each layer has half the nodes of its preceding one.

Further investigations in the number of nodes and layers could be performed, in order to fine-tune this final component and find the best trade-off between performances and costs.

2.2.2 Data set

The dataset used is made up by the following features:

- **time**. Time index of the sample, depends on the chosen time period
- **num_particles**. Concentration of UFP particles, the label to predict (UFP/cm^3).
- **cell_id**. Unique integer value, identificative for a cell.
- **temperature**. Average temperature (C°).
- **atm_pressure**. Average atmospheric pressure (hPa).
- **humidity**. Average air humidity (%).
- **wind_direction**. Average wind direction, originally it was a nominal feature (a possible value could be "Wind blowing from the east") so it is transformed to numerical by assigning an integer value to every possible nominal value.
- **wind_speed**. Average wind speed (km/h)
- **lat, lon**. Coordinates of this cell's center.
- **time_index**. An integer representing the time index for a sample, i.e. in the daily time frame all mondays have a time_index of 1, tuesdays have 2 and so on, in order to help the model catch correlations between similar time points.

In the data made publicly available from OpenSense there wasn't any weather information, therefore I found a data set containing hourly measurements from a weather monitoring station in the center of Zurich and merged those two toghether.

The map images used have a resolution of 128x128 pixels and come from the OpenStreetMap API; the original images were 256x256 but for the sake of performance have been reduced, no further pre-processing is performed.

Data aggregation

The spatial aggregation of data is performed in the same way as OpenSense, I created a grid with 100m x 100m cells that covers Zurich and whenever a new measurement comes to the system is assigned to its covering cell, as explained in 1.3.1.

The temporal aggregation of the data set is performed on fixed time periods: **daily, weekly, monthly, seasonal**; each cell has the information about average measurements for each time period, as a csv file (or table).

- Daily concentrations: 7 rows, average concentrations of each day of the week
- Weekly concentrations: 54 rows, average concentrations for each week of the year (the week zero is from 01/01 to 07/01)
- Monthly concentrations: 12 rows, average concentrations for each month
- seasonal concentrations: 4 rows, average concentrations for each season, the northern meteorological division [16] is used to assign each measurement to a season (all seasons are exactly 3 months long, for example spring lasts from 01/03 to 31/05)

The entries of these tables are going to be the inputs for the neural network; no further adjustments are made because of the good pre-processing work already done by OpenSense team.

To further clarify how the data are aggregated let us consider an example, suppose that we want to create the daily concentrations table having the whole set of measurements belonging to a cell: all measurements took on a monday will be averaged in order to get the monday average concentrations, and so on for each day of the week.

Chapter 3

Methodologies and tools

In this chapter all the technologies and frameworks used in this research are briefly presented (only the ones relative to the deep learning module of the application).

3.1 Python as programming language

The programming language used in this research is Python3.7, the last stable release of one of the most used programming languages in the world of data science.

The object-oriented nature of Python facilitates data scientists to execute tasks with better stability, modularity, and code readability. While Data Science is only a small portion of the diverse Python ecosystem, this language is rich with specialized deep learning and other machine learning libraries and popular tools like scikit-learn, Keras, TensorFlow and the one that I've used in this thesis, PyTorch. Undoubtedly, Python enables data scientists to develop sophisticated data models that can be plugged directly into a production system.

Thanks to its wonderful support to the most common data set file formats (CSV, JSON,...) it facilitates the work of data scientists, giving them the possibility to focus on developing the best model rather than being stuck on small problems such as having to write a procedure to parse a JSON file.

It also has some disadvantages: due to its extreme object oriented "mindset" it is hardly ever used in production ambient, because of the overhead that all the frameworks bring to the table. Most of the times this isn't a problem though, because usually a scientist would develop and train the model using python and only once the model is considered to be ready it gets translated to a more efficient language, such as plain C, because almost every major machine/deep learning framework is also available in more efficient languages.

3.2 Technologies for data aggregation and analysis

The data gathering process (meaning having a method to access the monitoring stations data) relies on the MQTT protocol and is out of the scope of this thesis. A brief description can be found in 1.3.1.

3.2.1 MongoDB

MongoDB [19] is a document based database, classified as NoSQL. It is designed with ease of development and mostly scaling in mind, therefore it is really suitable to Big Data applications, where the number of data to handle can grow exponentially.

In MongoDB a record is called **document**. A document is essentially a JSON object, therefore made up by "field:value" couples; each field may contain a single value, an array, another document or an array of other documents.
Documents are stored in **collections**, which are analogous to tables in SQL databases.

The fact of using this document based approach brings three main advantages:

- **Documents correspond to native data types in many languages**. The JSON standard is available as a native data type in many of the modern languages such as JavaScript or Python (in the form of Dictionary, the Python's name for this kind of data type).
- **Embedded documents and arrays reduce the need for expensive joins**. Traditional SQL databases are based on relationships between tables, in order to avoid as much as possible redundancy of data; such an approach grants minimal storage requirements and thanks to the atomic transactions it also grants safety of data. In Big Data applications usually the most important requirement is the quickness of reading and writing to the database rather than the safety of atomic transactions.
- **Dynamic schema supports polymorphism**. MongoDB is known as a "schema-less" database, meaning that it does not force any kind of structure for the documents in a collection and therefore it is possible that a collection may contain documents with different structures; this approach mimics the object oriented programming languages.

The key features of MongoDB are the following:

- **High Performance**. The embedded data models greatly reduce the need for I/O operations on the database, and the indexes enable the support for faster queries, also including keys from embedded documents.
- **Rich query language**. Read and write operations (CRUD) are supported by a powerful query language, enabling clients to perform complex queries and automated data aggregations with ease.
- **High availability**. MongoDB's support for **replica sets** provides automatic failover and data redundancy, a replica set is essentially a cluster of servers mantaining the same database in multiple copies.
- **Horizontal scalability**. A core functionality is the horizontal scalability of MongoDB databases, thanks to **sharding** the data of a single database can be distributed in a cluster of servers.

3.2.2 Pandas

As said in the official Pandas GitHub repository [25], Pandas is "a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way towards this goal."

The two main data structures in Pandas are Series and DataFrames:

- **DataFrame**. Array of 2 dimensions, rows and columns.
- **Series**. Array of 1 dimension, constitutes a single column of a DataFrame.

These structures have really complex mechanism that enable the users to interact easily with the data. Making operations such as selections, slicings, aggregations of any kind is almost as easy as handling a normal array in Python.

The only downside of Pandas is that all the flexibility and ease of use granted by such structures means that a lot of overhead data is used, such as really complex indexes. When performing I/O operations on large amounts of data Pandas can become really slow, mostly when handling tasks such as merging thousands of files togheter: each time a file is opened via Pandas a DataFrame is created. To perform those kind of tasks a more direct approach is advisable, such as the library "csv" available as a native package in every Python installation, which is much lighter even though it doesn't grants the same capabilities.

3.2.3 Matplotlib

One of the main jobs of a data scientist is often **data visualization**, which basically is the practice of visualizing data in graphs, icons, presentations and more. It is most commonly used to translate data stored in complex structures into digestible insights, in order to make it understandable also for a non-technical audience.

Matplotlib ([15]) is the most popular data visualization library in Python's ecosystem. It supports basically any kind of graphical representation, from simple line plots to really complex plots, made up by many subplots and/or multiple lines.

In my thesis I've used the **pyplot interface**, an interface to matplotlib with commands that strive to be similar to MATLAB ones.

I've chosen this interface due to the high availability of tutorials on the internet and furthermore to its simplicity, at the cost of being slightly more rigid than the original matplot library.

3.3 Technologies for Deep Learning

3.3.1 PyTorch

PyTorch [26] is a machine learning framework that strives to accelerate the path from research prototyping to production development. It has been originally developed by Facebook for its services and open sourced in early 2017, but in a short time it has become really popular and used by companies such as Twitter or Salesforce.

It is a port to Python of the Torch deep learning framework which can be used for building deep neural networks and executing *tensor computations*, Torch is based on Lua.

Essentially *PyTorch is a Python package which provides tensors computations*, where **tensors** are multidimensional arrays (just like numpy's ndarrays) which can lay on the GPU memory, and offers a *dynamic computation graph*, meaning that instead of using a predefined static graph PyTorch builds the computational graph via the **autograd** package, therefore it is possible to modify the network (e.g. the computational graph) during runtime.

Tensors

Deep Learning is essentially computations on tensors, which can be seen as *generalizations of a matrix, that can be indexed in more than 2 dimensions.* Actually, matrices and vectors are special cases of torch.Tensors, where their dimension is 1 and 2 respectively, see Figure 3.1 for a graphical representation.

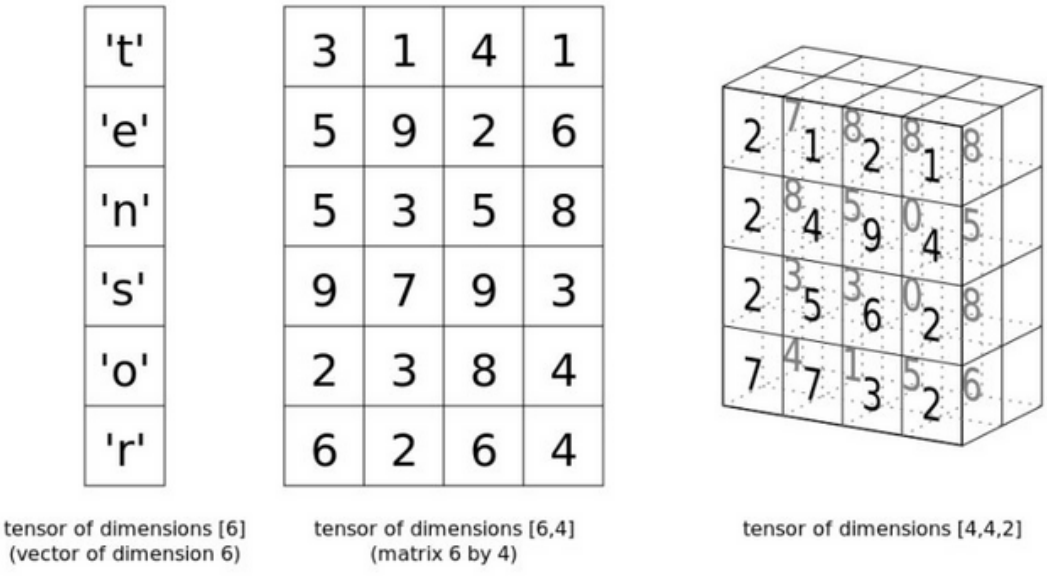


Figure 3.1: Tensors of different dimensions

Tensors are really similar to NumPy's ndarrays, with the big addition of having the ability to reside on the graphics card memory and therefore highly enhancing the performances.

As said before, all computation in deep learning are essentially mathematical operations performed on data that may have more than one dimension. When said data is made up by images or audio the computations can become really heavy (think about multiplying two matrices of 256x256 elements, and such operation may be performed thousands and thousands of times to even finish one single experiment) so it becomes fundamental to accelerate the computation.

To do so tensors have been introduced: data structures able to reside on the GPU memory, so that all the computations can be performed by such piece of hardware, which is actually specialized in graphical computations, but performs very well nevertheless thanks to the fact that *computations on images* (the job for which GPUs were invented) *computations on images are exactly the same as tensors computations*: matrices operations have to be performed.

Autograd package

The concept of a computation graph is essential to efficient deep learning programming, because it allows the user to not have to write the back propagation gradients himself. *A computation graph is simply a specification of how your data is combined to give you the output.*
Since the graph totally specifies what parameters were involved with which operations, it contains enough information to compute derivatives, and therefore lets the system automatically take care of writing the back propagation gradients and also enables the dynamic computation graph feature, one of the most important and distinctive features of PyTorch.

Autograd is the package in charge of keeping this mechanism working, and *it is essentially a way of adding to each tensor of the model the information about how it was created.* To understand this concept let us consider an example: when two tensors are added together an output tensor is generated, but it has only the information about its data and its shape; if autograd is enabled on this object than the tensor will keep track of how it was created, so it will know what tensors created it and using what operation.

Generally speaking, autograd is an engine to automatically compute vector-jacobian product with the goal of automating the differentiation of each tensor: the backpropagation algorithm needs the derivatives of each tensor to work, therefore via autograd those derivatives are calculated each time the backpropagation runs, letting the user change the computation graph. For a detailed explanation see [5].

Essentially it computes the sum of gradients of a given tensor, with respect to the graph leaves. The whole graph is differentiated using the **chain rule**, and the gradients are accumulated in the leaves.

Object-oriented framework

In 2019 PyTorch became the framework of choice at ICML [10] and many others important scientific conferences, stating its domain upon the scientific and academic communities.

PyTorch found its success mostly due to the fact that it tends to incline more towards Python when compared with any other library, therefore it is relatively easy to understand and it feels more natural, native and in line with Python code.

The Python nativeness of this framework means that it is built with object-oriented programming in mind, therefore even complex models are easily manageable and the actual code that implements them is relatively easy to understand, for those who are familiar with the object-oriented mindset.

Networks are implemented by means of the **nn.Module** interface, that requires only two methods:

- `__init__()`: constructor method, where the structure of the network is defined. There are lots of different layers, activation functions, normalization layers, any kind of operation needed to implement deep neural networks.
- `forward(sample)`: defines how a sample passes through the network, each layer is essentially a method that requires a sample in input and outputs the result of its operation, such as `output = network.forward(sample)`.

torch.utils.data.Dataset is the data set interface, it defines how the data are read from memory and how a single sample is extracted from the data. It is up to the user of the Dataset class to decide how many samples extract at a time, what samples to extract, etc...
Three methods are required:

- `__init__()`: constructor method, where the data are read and kept in memory (a file *.csv* could be read with Pandas library, the resulting DataFrame would be an attribute of the class Dataset).
- `__len__()`: returns the number of samples in the data set.
- `__getitem__(idx)`: given an index value *idx*, this method returns the sample associated with that index.

Once the Dataset and Network are defined the training script can be written quickly thanks to **torch.utils.data.DataLoader**, an utility that automates the process of querying the Dataset object for a sample. It provides lots of useful features such as multithreaded data loading and batch loading (extracting more than one sample at a time) that make the process of data loading as efficient and easy as possible.

3.3.2 OpenCV

OpenCV [20] is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. It has a modular structure, meaning that it includes several shared or static libraries; the following modules are available:

- **Core functionality**. It defines basic data structures used by the other modules.
- **Image Processing**. A module that includes algorithms to perform linear and non-linear image filtering, geometrical transformations, color space conversions, histograms, any kind of processing work that could be done on an image.
- **Video Analysis**. Automatic video analysis, including motion estimation, background subtraction and object tracking algorithms.

- **Camera calibration and 3D reconstruction, Object detection, ...**

Essentially OpenCV is a massive library used in almost any kind of Computer Vision task, thanks to the availability of lots of algorithms.

In the context of CleanAir Project, OpenCV has been used to preprocess the map images, and more specifically only to resize them from 256x256 to 128x128 pixels; no further manipulations have been needed, mostly because there wasn't any need for geometrical transformations or color space changings.

Chapter 4

Work description

In this chapter I'll explain how I actually implemented all the functionalities discussed in 2.2.

4.1 Statistical analysis

Some simple statistical analysis have been performed on the data set before starting to aggregate the data, in order to deeply understand the actual feasibility of this research.

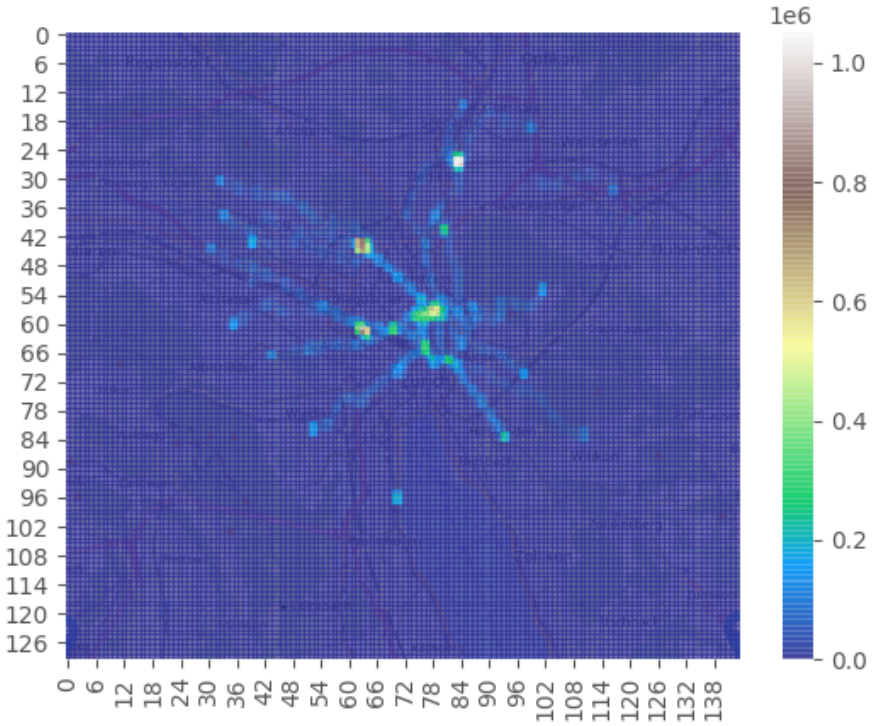


Figure 4.1: Heatmap describing the number of measurements per cell, before any kind of temporal aggregation

Figure 4.1 shows the number of measurements per cell before any kind of temporal aggregation. As expected the central cells have the highest number of measurements and therefore the statistical validity of the resulting aggregated data is noticeable, due to the fact that averages are performed on a high number of samples. This figure also helps to show the geographical distribution of the data set, which is pretty good considering the goal of the project.

Being this model based on map images, having the largest possible number of "labeled" cells is fundamental, in order to let the model see as much types of terrain morphologies as possible during training.

4.1.1 Correlation

For each time frame the pair-wise correlation between the features of the data set has been analyzed, in order to check if there were features with too much correlation that would be useless for predictions.

	UFP	temp.	atm_press.	humidity	wind_dir.	wind_speed	lat	lon
UFP	1.0	0.2873	0.0743	0.0743	0.0061	-0.3747	0.0989	-0.1074
temp.	0.2873	1.0	0.3324	-0.7469	0.4611	0.0768	-0.0422	0.2990
atm_press.	0.0743	0.3324	1.0	-0.1791	0.1732	-0.1793	-0.1474	0.1656
humidity	0.0743	-0.7469	-0.1791	1.0	-0.6254	-0.3955	0.1394	-0.0839
wind_dir.	0.0061	0.4611	0.1732	-0.6254	1.0	0.0538	-0.1481	-0.1199
wind_speed	-0.3747	0.0768	-0.1793	-0.3955	0.0538	1.0	-0.0584	-0.0202
lat	0.0989	-0.0422	-0.1474	0.1394	-0.1481	-0.0584	1.0	0.1979
lon	-0.1074	0.2990	0.1656	-0.0839	-0.1199	-0.0202	0.1979	1.0

Table 4.1: Correlation between features of the average Monday, from the daily time frame

	UFP	temp.	atm_press.	humidity	wind_dir.	wind_speed	lat	lon
UFP	1.0	-0.2576	0.2016	0.1844	-0.2861	-0.2087	0.0111	0.0524
temp.	-0.2576	1.0	-0.6514	-0.6457	0.4791	0.5379	0.3329	-0.0164
atm_press.	0.2016	-0.6514	1.0	0.1350	-0.5018	-0.5163	-0.2556	0.0746
humidity	0.1844	-0.6457	0.1350	1.0	-0.5712	-0.4980	-0.3153	-0.0132
wind_dir.	-0.2861	0.4791	-0.5018	-0.5712	1.0	0.7860	0.3905	-0.0998
wind_speed	-0.2087	0.5379	-0.5163	-0.4980	0.7860	1.0	0.3883	0.1017
lat	0.0111	0.3329	-0.2556	-0.3153	0.3905	0.3883	1.0	-0.0531
lon	0.0524	-0.0164	0.0746	-0.0132	-0.0998	0.1017	-0.0531	1.0

Table 4.2: Correlation between features of the average Week 1 of the year, from the weekly time frame

The tables 4.1, 4.2, 4.3, 4.4 are representing the pair-wise correlation between all the features, including the label to predict, the UFP concentration. By looking at those tables some considerations can be made:

- *No feature should be eliminated*, in all time frames there aren't pair of features with too much correlation
- *The correlation values arise as much as the temporal granularity arises.* The highest values of correlation can be found in the seasonal aggregation (4.4), this

	UFP	temp.	atm_press.	humidity	wind_dir.	wind_speed	lat	lon
UFP	1.0	0.1283	0.1371	0.0821	0.0233	0.0244	0.0541	-0.0102
temp.	0.1283	1.0	0.3771	-0.7357	0.1667	-0.1032	-0.0070	0.0189
atm_press.	0.1371	0.3771	1.0	-0.1989	0.7360	0.4406	-0.4735	-0.2923
humidity	0.0821	-0.7357	-0.1989	1.0	-0.3371	-0.1522	-0.0158	0.2183
wind_dir.	0.0233	0.1667	0.7360	-0.3371	1.0	0.6383	-0.2956	-0.4918
wind_speed	0.0244	-0.1032	0.4406	-0.1522	0.6383	1.0	-0.2296	-0.3078
lat	0.0541	-0.0070	-0.4735	-0.0158	-0.2956	-0.2296	1.0	0.1993
lon	-0.0102	0.0189	-0.2923	0.2183	-0.4918	-0.3078	0.1993	1.0

Table 4.3: Correlation between features of the average January, from the monthly time frame

	UFP	temp.	atm_press.	humidity	wind_dir.	wind_speed	lat	lon
UFP	1.0	0.3124	-0.0716	-0.1640	0.2223	0.2480	-0.0227	-0.1460
temp.	0.3124	1.0	-0.2921	-0.6826	0.4755	0.3550	-0.1515	-0.2490
atm_press.	-0.0716	-0.2921	1.0	0.2683	-0.2173	-0.1937	-0.1199	0.0998
humidity	-0.1640	-0.6826	0.2683	1.0	-0.7245	-0.7507	0.1797	0.4412
wind_dir.	0.2223	0.4755	-0.2173	-0.7245	1.0	0.5872	-0.2843	-0.4842
wind_speed	0.2480	0.3550	-0.1937	-0.7507	0.5872	1.0	-0.0641	-0.2435
lat	-0.0227	-0.1515	-0.1199	0.1797	-0.2843	-0.0641	1.0	0.1991
lon	-0.1460	-0.2490	0.4412	0.4412	-0.4842	-0.2435	0.1991	1.0

Table 4.4: Correlation between features of the average Winter season, from the seasonal time frame

is most likely due to the fact that by increasing the temporal granularity each period has much more measurements (the number of measurements took in a season is several orders of magnitude higher than the ones took in a single day).

4.1.2 UFP concentration analysis

Figure 4.2 shows the average UFP concentrations for each time frame. Such concentrations are obtained by setting a time index and then averaging all cell's concentration values, to further clarify let us consider an example: to have the average Monday from the daily time frame means setting considering only the average Monday for each cell and averaging those in the average Monday for the whole city of Zurich. Figure 4.3 shows the standard deviation of UFP concentrations in the whole city for each time frame, in light of these plots some considerations can be made over each time frame:

- **Seasonal**: As expected summer is the season with less pollution thanks to the lack of house heating, this hypothesis is confirmed by the increasing trend as the seasons get colder.

- **Monthly**: The final month of each season sees a variance increase due to the changing temperatures and the consequent change in people's behaviour, which happens asynchronously around the city and therefore pollution is distributed in a much sparser way than the other months.

- **Weekly**: This time frame is the more noisy one, as for its variance. The same "spiking" trend as for the monthly time frame can be noticed in a greater time

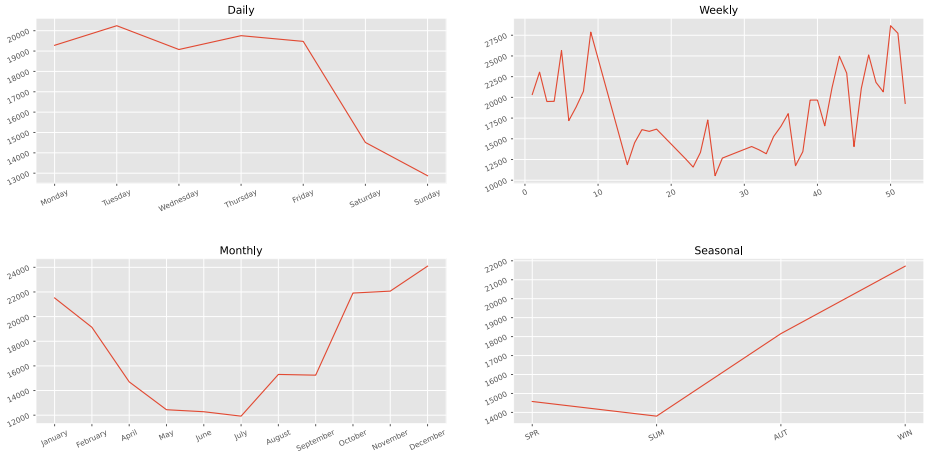


Figure 4.2: The average UFP concentration for each time frame, average value for the whole city.

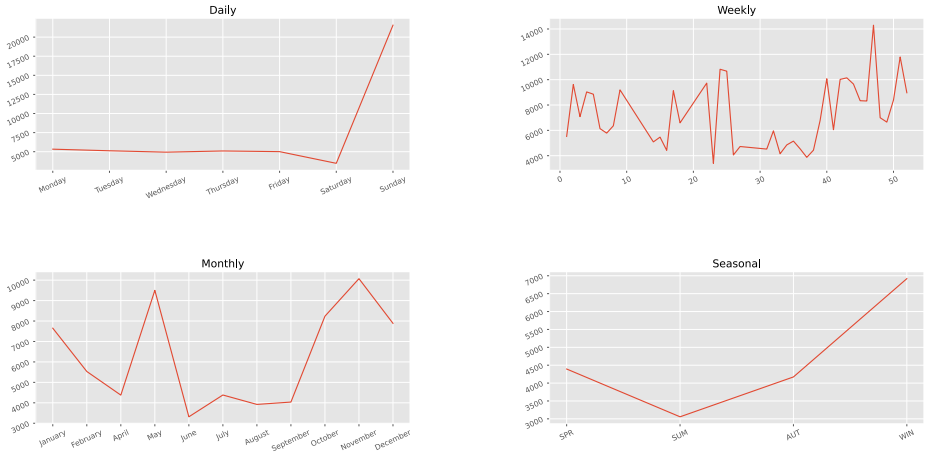


Figure 4.3: The standard deviation of UFP concentrations for each time frame.

resolution, last weeks of the final month of each season present a higher variance; the mean value also confirms the seasonal trend.

- **Daily**: During the average work days pollution levels have a relatively stable trend while weekends see a good decrease of concentrations, most likely due to people not having to use carbon-fueled transports to reach their workplace.

Figure 4.4 shows the geographical distribution of yearly UFP concentrations across the whole city of Zurich, to further asses the distribution of such particles.

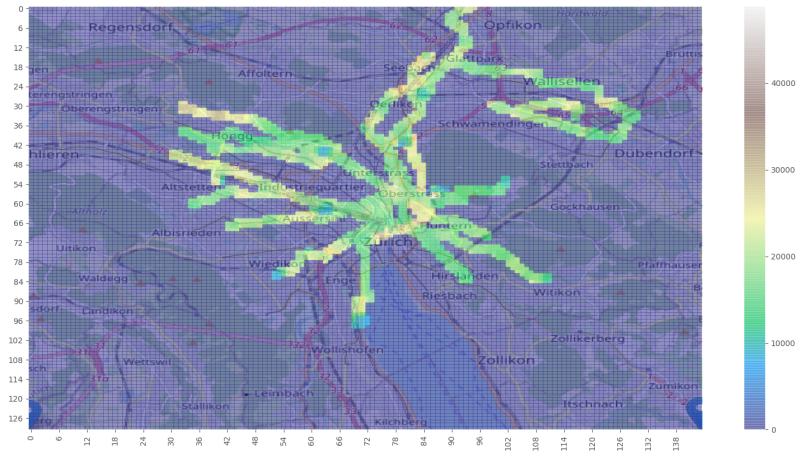


Figure 4.4: The yearly average UFP concentration for each cell.

4.2 Data aggregation

The very first problem to face was data aggregation, because it had to be performed on two levels: both a temporal and spatial, due to the fact that the data set has both a time index (the timestamp of each measurement) and a geographical index (the location in which the measurement was taken).

4.2.1 Spatial aggregation

To aggregate the measurements spatially I followed the method used by OpenSense: a grid with cells of 100m x 100m that covers the whole city of Zurich.

OpenSense did not said how they built the grid, so I had to find my own method:

1. Define a square over Zurich by means of its south-west and north-east corners (in latlon format), said points are defined by me via OpenStreetMap's markers.

2. From the south-west point add 100 meters to the east to find the next cell's center, repeat this process until the south-east corner is met.

3. After a row is completed, from the south-east add 100 meters to the north to find the northern neighbour, then repeat step 2.

With this algorithm I have been able to create the covering grid over Zurich, such grid is then saved as a collection in MongoDB, where the cells are the document of the collection. I have also created grids with cells of 250m x 250m and 500m x 500m, using the same method, to experiment different spatial granularities.

After the grid was ready, the actual aggregation of data could be performed. The mechanism I choose to aggregate the data at the spatial level is simple: *iterate trough the dataset, each measurement gets assigned to its belonging cell*, where a measurements belongs to a cell if the cell's center is the closest one to where the measurement was made.

The step of finding the nearest cell for a measurement is performed via the *haversine formula* (1.25); the only problem left was that such a naive approach would require a really long time to conclude and therefore was not feasible for this project. That's why I decided to make use of MongoDB's powerful query mechanisms.

I saved both the grid and the dataset on MongoDB in separate collections, then I used the **$geowithin** operator [18]. It is a query operator focused on geographical queries, it selects all documents in a collection with geospatial data that exists entirely within a specified shape.

In this context the specified shape is the square represented by a cell, and the documents are all the measurements taken inside the "bounding shape".

There is only one constraint to use this operator: all the geospatial data has to be compliant to the GeoJson standard, which means respecting this format:

```
{
    type: <GeoJSON type> ,
    coordinates: <coordinates>
}
```

A GeoJson type could be a Point, a LineString, or a Polygon. From the center of a cell its corners are calculated by subtracting and adding 50 meters from each coordinate respectively, in order to form a Polygon.

By using this operator I was able to make use of the fantastic optimizations performed by MongoDB via its complex system of indexes and geo-indexes.

4.2.2 Temporal aggregation

As explained in 2.2.2 I choose a different approach than OpenSense, the dataset is first spatially aggregated and then averaged on these fixed time frames:

- **Daily**: average day of the week, 7 measurements per cell.
- **Weekly**: average week, 54 measurements per cell.
- **Monthly**: average month, 12 measurements per cell.
- **Seasonal**: average season, 4 measurements per cell.

To perform this complex aggregations I made use of the powerful mechanisms offered by pandas ([24]), that are also strongly connected to the **datetime** library.

The connection with datetime has been fundamental, because pandas let me read the dataset and simultaneously parse the dates in it, so that manipulating said dates or accessing some parameters became really easy.

Each time frame required the use of different mechanisms:

- Daily: the `datetime.day_name()` procedure returns the weekday of a given date, then aggregation is performed by weekdays.
- Weekly: the `datetime.weekofyear()` procedure returns the week number of a given date, which is the order of the week, meaning that the week from 01/01 to 07/01 will be the week number 1.

- Monthly: every datetime object has attributes regarding the components of a date, so to access the month of a date is as easy as date.month.
- Seasonally: I choose to aggregate by metereological seasons, so I created a dictionary that associated every month with its season:

```
    seasons = {
    1: 'WIN', 2: 'WIN', 3: 'SPR',
    4: 'SPR', 5: 'SPR', 6: 'SUM',
    7: 'SUM', 8: 'SUM', 9: 'AUT',
    10: 'AUT', 11: 'AUT', 12: 'WIN'
}
```

  The aggregation is then performed by extracting the month of a date and associating that month with its corresponding season.

4.3 Regression with Convolutional Neural Networks

The framework used to build, train and evaluate the Convolutional Neural Network described in 2.2.1 is PyTorch. The motivations behind the choice of this framework instead of something more popular such as Google's Tensorflow is dictated by the customization needs of this project.
I had to create a sort of "custom" version of a CNN, where the last MLP receives additional data, and the object-oriented structure of PyTorch let me produce clean and well organized code using a class structure, feature that I found extremely useful and also makes this framework relatively easy to learn for someone who has a programming background like me.

4.3.1 Dataset

After the aggregation process applied on the data set, each cell has 4 files associated containing the aggregated measurements belonging to the cell for each time frame. All those files are merged in 4 comprehensive data sets, containing all the aggregated measurements of each cell for each time frame.
The Dataset class then reads one of those files (depending on a parameter required by the constructor) via the Pandas library and saves the resulting DataFrame as an attribute.

Map Images

Each cell has an attribute that references to its corresponding map image, saved in the file system of the server running MongoDB and also present in the 4 aggregated files; those images have been gathered by means of one of the many OpenStreetMap Map Tile Servers [22].

OpenStreetMap provides map tiles to its users by means of the Tile Map Server protocol [23] also known as TMP, which is a protocol for serving maps as tiles i.e. splitting the map up into a pyramid of images at multiple zoom levels.

The map tiles are usually images with a resolution of 256x256 pixels and are fetched

via *tile numbers*, which is a referencing system for map tiles at different levels of zooms. The Tile Server used to fetch the map images for this project has been the following:

```
https://tiles.wmflabs.org/osm-no-labels/${z}/${x}/${y}.png
```

where z is the zoom level and x, y are the coordinates in tile numbers. The conversion from latlon format to tile numbers is performed in according to the SlippyMap standard as explained in [6]:

$$x = \left\lfloor \frac{lon + 180}{360} \cdot 2^{zoom} \right\rfloor \tag{4.1}$$

$$y = \left\lfloor \left(1 - \frac{\ln(\tan(lat \cdot \frac{\pi}{180}))}{\pi} + \frac{1}{\cos(lat \cdot \frac{\pi}{180})}\right) \cdot 2^{z-1} \right\rfloor \tag{4.2}$$

To help the model better understand the environment surrounding a cell, the images are fetched using a zoom level of 15, which means they cover more or less an area of 300m x 300m; this increase in size helped the model to get closer to the basic idea of this project: *using small portions of map, learn how the pollutants move in topologically similar zones.*

Item fetching

When the method `__getitem(idx)__` is invoked the Dataset class has to return the sample corresponding to the specified index *idx*; in the context of CleanAir the fetching of a sample consisted in retrieving the idx-th line from the DataFrame, reading the corresponding map image (via SciKitLearn's imread [28] function) and returning a dictionary such as the following:

```
sample = {
    image: torch.Tensor,
    weather_data: torch.Tensor,
    ufp_label: torch.Tensor
}
```

Where `image` is a Tensor representing each channel as a matrix of pixel values, `weather_data` is a flattened Tensor containing the weather measurements (explained in 2.2.2) for that cell at a given time point, and `ufp_label` is the average UFP concentration to predict.

4.3.2 Training

The training process has been feasible thanks to the help of the GARR Cloud [8], who provides a Kubernetes environment to access cloud computing instances and furthermore powerful GPUs, fundamental to train deep neural networks much faster.

I had access to a powerful NVIDIA QUADRO V-100, a PCI-E card with 16GB of video memory, more than sufficient to hold batches of samples as big as 64 units or more.

Loss function

To train a model means using some sort of error measure and tweak the model's parameters accordingly. In the context of Regression the functions available are all about

measuring the difference between the predicted and original values, the difference lies in the implications of using one function or another.

The two most used loss functions for regression tasks are the Mean Square Error (also known as MSE or L2 Loss) and the Mean Absolute Error (also known as MSE, L1 Loss or Root Mean Square Error).

L1 minimizes the absolute differences between the estimated values and the existing target values, while L2 minimizes the squared differences between the estimated and existing target values.

Even if apparently the two functions are really similar, there is a big difference between minimizing the absolute error or the squared one. Using a squared error means that in case of an outlier the loss function will be really high and therefore the model's weights will be heavily tweaked to minimize that single error.

As a result, using L1 loss function is more robust and is generally not affected by outliers. On the contrary L2 loss function will try to adjust the model according to these outlier values, even on the expense of other samples. Hence, L2 loss function is highly sensitive to outliers in the dataset.

In 4.1 the mean value and variance of UFP values to predict has been analyzed, and due to the relatively high variance of this data set (which was expected, because the objects of the study are particles moving around the air) the L1 loss function has been chosen to train the network:

$$\text{MAE} = \sum_{i=o}^{n} |y_i - h(x_i)| \tag{4.3}$$

Adam Optimizer

The optimizer used in this training process has been one of the most used and known ones, Adam [2]. It is an algorithm built for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters.

Adam is different to classical stochastic gradient descent because it has an **adaptive learning rate**, hence it changes the learning rate value during training based on both the average first moment and the second moment of the gradients.

The authors describe it as combining the features of two other extensions of the stochastic gradient descent:

- **Adaptive Gradient Algorithm** (AdaGrad): maintains a per-parameter learning rate that improves performance on problems with sparse gradients.
- **Root Mean Square Propagation** (RMSProp): per-parameter learning rate adapted based on the average of recent magnitudes of the gradients for the weight (essentially how quickly it is changing). This approach does well on online and non-stationary problems, for example on noisy data sets.

The algorithm consists in calculating an exponential moving average of the gradient and the squared gradient with the parameters β_1 and β_2 in charge of handling the decay

rates of these moving averages. Those parameters are recommended to start close to 1.0 in order to get a bias of the moment estimates towards zero, the bias is overcome by first calculating the biased estimates before recalculating the bias-corrected estimates.

4.3.3 Evaluation

The model's performances have been evaluated with the same metrics used by OpenSense, in order to being able to confront the results: *Root Mean Square Error (RMSE)*, *Coefficient of determination* R^2 and the *Factor of 2 measure (FAC2)*.

Root Mean Squared Error

It quantifies the difference between the predicted and real UFP value:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(c_i^P - c_i^m)^2}{N}} \tag{4.4}$$

where N denotes the number of cells used in the evaluation, c_i^P is the model's predicted UFP value and c_i^m is the true one, for the cell i.

Adjusted coefficient of determination

Indicates in a range from 0 to 1 how well the predictions fit the measurements, it basically reflects the linear relationship between predicted and measured values and therefore it is insensitive to additive and multiplicative errors.

$$\begin{aligned} \bar{y} &= \frac{1}{n}\sum_{i=1}^{n} c_i^m \\ R^2 &= \frac{\sum_{i=1}^{n}(c_i^m - c_i^P)^2}{\sum_{i=1}^{n}(c_i^m - \bar{y})^2} \end{aligned} \tag{4.5}$$

Factor of 2 measure

Quantitatively analyzes scatter plots of predictions by evaluating the fraction of data points that lie inside the *factor two* area, i.e. the fraction of data that satisfies the following:

$$0.5 \leq \frac{c_i^P}{c_i^m} \leq 2.0 \tag{4.6}$$

Chapter 5

Conclusions

In order to have the most scientific confidence possible about the capabilities of CleanAir Project's deep learning module, the results are compared to those obtained by OpenSense.

Even though the two projects are similar, some fundamental **differences** lie in how the data are aggregated both spatially and temporally. The **temporal aggregation** performed by OpenSense produced one model per time unit for each time agglomeration considered (meaning that they produced one model per day for the daily time agglomeration, and so on) whereas CleanAir produces one model for each time agglomeration, capable of handling the average time units inside of it (the daily time frame produces one model capable of handling each average day of the week).

The differences about **spatial aggregation** are even greater: OpenSense states [4] that they found that using only the first 200 cells with the highest number of measurements to train and test their model gives the best results, CleanAir on the other hand uses almost 20000 cells in the 100m x 100m resolution obtaining comparable results even if using a smaller number of measurements per cell than OpenSense.

To test the model three different grid resolutions have been experimented: 100x100m, 250x250m, 500x500m.

5.1 Results discussion

Tables 5.1, 5.2, 5.3 show the results of each model at the three different spatial resolutions experimented: 100m x 100m, 250m x 250m, 500m x 500m. A lot more resolution and even cell shapes could be experimented, amongst the ones considered in this research the 250m x 250m cells have proven to be the most accurate, therefore this could be a good starting point for further fine-tuning processes. Figure 5.1 shows the met-

	Daily	Weekly	Monthly	Seasonal
FAC2	0.987	0.934	0.973	0.992
RMSE	2932.414	4815.764	3354.047	2316.726
R2	0.372	0.287	0.532	0.526

Table 5.1: Metrics values for each model, 100m x 100m resolution

	Daily	Weekly	Monthly	Seasonal
FAC2	0.981	0.977	0.989	1.000
RMSE	2833.370	3740.108	2622.925	1874.893
R2	0.586	0.516	0.597	0.815

Table 5.2: Metrics values for each model, 250m x 250m resolution

	Daily	Weekly	Monthly	Seasonal
FAC2	0.969	0.951	0.984	0.987
RMSE	4584.494	4528.340	3683.122	3674.997
R2	0.292	0.317	0.327	0.542

Table 5.3: Metrics values for each model, 500m x 500m resolution

rics values obtained by OpenSense, for each time aggregation level they considered; the sub-weekly models show weaknesses mostly in terms of R^2.

According to [3] one of the most important factor when evaluating an air quality model is the FAC2 score; considering that my system has FAC2 scores slightly higher (in average) than the ones obtained by OpenSense, and given that OpenSense claims to have reached *close to state-of-the-art results*, it is safe to state that CleanAir Project's approach is a valid alternative to more traditional models and could be worth further studies.

5.2 Advantages of using a Neural Networks approach

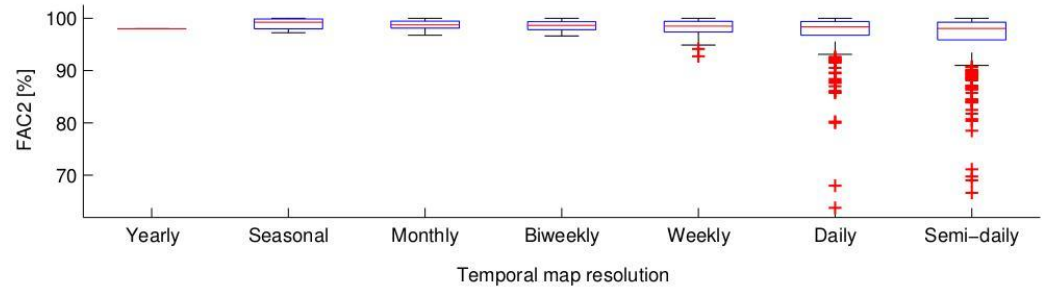
Another fundamental difference between OpenSense and CleanAir are the **explanatory variables** 2.1.

OpenSense uses a fixed set of what they define explanatory variables in order to give their regressive model some awareness of the context, and those variables are statistical estimates coming from Federal Institutions such as the Swiss Federal Statistical Office.

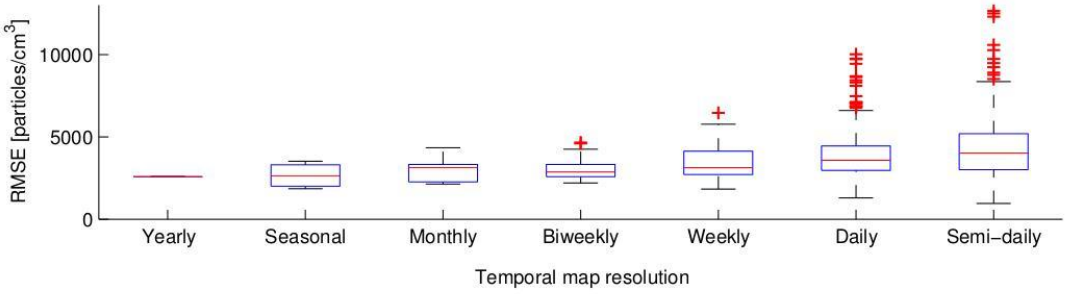
CleanAir adopts a completely automatized approach thanks to the use of a Convolutional Neural Network: *the convolutional layers automatically extract a set of explanatory variable for each unique map portion.*
This approach has proven to be equally capable than the other one, but has the game changing advantage of **automatic explanatory variables extraction**, a quality that enables **CleanAir to be applied virtually anywhere in the world** where a fleet of mobile monitoring stations is available, **without the need to carefully craft and select explanatory variables manually**.

5.3 Future developments

The biggest limit of this module is the lack of data: only a year of measurements have been used to train the models and in a real-world application this wouldn't be enough in this times of high weather variance.



(a) Factor of 2 statistic (FAC2).



(b) Root-mean-square error (RMSE).

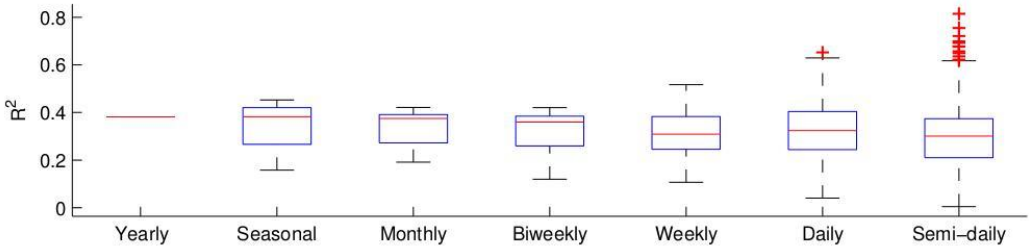


Figure 5.1: Results obtained by OpenSense reasearch team.

In spite of the unavailability of long-time historic data, the models capabilities have proven to be comparable to the ones produced by OpenSense, therefore it is safe to state that this research shown promising results and could be worth of a more accurate and sophisticated study.

Improvements could be made basically anywhere, in the context of the deep learning module:

- **Deploying physical monitoring stations**. Having an own fleet of monitoring station would mean being able to control the flow of data directly from the physical station, enabling experiments with different kinds of raw data pre-processing.
- **Experimenting with different CNN architectures**. For the purpose of this research a simple and well known architecture has been used for the Convolutional Neural Network such as the LeNet has been used, but in these last years tens of new and promising architectures have been proposed and could be worth of investigation.
- **Real-time map images**. A game changing data would be the ability of using real-time satellite images of the cells, in order to be able to associate to each

measurement the image shoot by a satellite at the same hour. This would bring a lot of improvements to the model, because the feature extraction would produce a lot more informations (for example, the yellowing of the leaves on trees would induce the model to understand the passing of seasons, and therefore to catch possible relationships hidden before).

Ringraziamenti

Il primo ringraziamento va ai professori Luigi Portinale e Attilio Giordana che mi hanno supportato in ogni modo possibile, di fatto rendendo realizzabile quella che inizialmente era solo un'idea originale presentata ad un hackaton.

Tali docenti mi hanno dato modo di presentare alla CINI SmartCities Challenge 2018 questo progetto e conseguentemente di poterlo realizzare, fino a farlo diventare la mia tesi di laurea. Mi hanno supportato in ogni fase dello sviluppo del progetto, sia dal punto di vista di reperimento delle risorse computazionali che dal punto di vista teorico, dimostrandosi sempre disponibili e prodighi di consigli.

Voglio dedicare un ringraziamento speciale al prof. Giordana, il quale è stato fondamentale nello sviluppo dell'intero progetto dal punto di vista teorico ed ha creduto per primo nell'idea.

Grazie a CleanAir Project e a chi l'ha reso possibile, ho avuto modo di sviluppare moltissime altre abilità al di fuori dell'ambito scentifico, e di questo sarò eternamente grato all'intero corpo docente e all'Università intera, grazie ai quali ho partecipato a moltissime attività legate più o meno direttamente a questo progetto. Ho avuto modo di presentare il mio progetto davanti a pubblici anche numerosi, organizzare eventi divulgativi verso il mondo manifatturiero alessandrino, o anche semplicemente costruire fisicamente una stazione di monitoraggio e scontrarsi con i problemi più banali.

Altre figure fondamentali nello sviluppo di questo progetto sono i miei colleghi e amici Christopher, Matteo, Marco, Manuel, Emanuele, Alfred e tanti altri. Ciascuno di voi mi ha aiutato nel corso di questi anni e per questo vi ringrazio, avete saputo sopportarmi e comprendere anche in momenti di estrema difficoltà.

Uscendo dalla sfera scolastica non posso che non ringraziare gli amici di una vita, che essendo troppi per essere nominati singolarmente racchiuderò in "Capedritte". Alcune delle esperienze più belle e intense della mia vita le ho vissute insieme e grazie a voi, che riuscite spesso a farmi uscire dalla cosiddetta "comfort zone", la quale troppo spesso sono restìo a lasciare, e per questa e mille altre motivazioni vi ringrazio.

Infine il ringraziamento più importante e sentito va alle persone a cui devo tutto, la mia famiglia. Non penso che le parole possano bastare ad esprimere un sentimento così vasto, per cui ripropongo ciò che scrissi nella relazione finale della Laurea Triennale:

> *Grazie a mio padre Aldo che mi insegna ad essere Uomo, grazie a mia madre Silvana che mi insegna ad Amare e grazie a mia sorella Elisa, che mi insegna a Ridere.*
>
> *La Via prosegue senza fine*

Lungi dall'uscio dal quale parte.
Ora la Via è fuggita avanti,
Devo inseguirla ad ogni costo
Rincorrendola con piedi alati
Sin all'incrocio con una più larga
Dove si uniscono piste e sentieri.
E poi dove andrò? Nessuno lo sa.
(J.R.R. Tolkien, Il Signore degli Anelli)

Bibliography

[1] "A review and evaluation of intraurban air pollution exposure models". In: *Exposure Science and Environmental Epidemiology 15* (2005), pp. 185–204.

[2] *Adam: A Method for Stochastic Optimization*. URL: https://arxiv.org/abs/1412.6980.

[3] "Air quality model performance evaluation". In: *Meteorology and Atmospheric Physics 87* (2004), pp. 167–196.

[4] David Hasenfratz et al. "Deriving High-Resolution Urban Air Pollution Maps Using Mobile Sensor Nodes". In: *Pervasive and Mobile Computing* (2015), pp. 3–7, 14.

[5] *Autograd: automatic differentiation*. URL: https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html.

[6] *Conversion of latlon coordinates to tile numbers, TMS protocol*. URL: https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames#Lon..2Flat._to_tile_numbers.

[7] Pedro Domingues. *The Master Algorithm*. 2015.

[8] *GARR Cloud*. URL: https://cloud.garr.it/.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 9.

[10] *ICML: International Conference on Machine Learning*. URL: https://icml.cc/.

[11] "Influence of meteorology on PM10 trends and variability in Switzerland from 1991 to 2008". In: *Atmospheric Chemistry and Physics 11* (2011), pp. 1813–1835.

[12] *IPCC Fifth Assessment Report*. URL: https://www.ipcc.ch/assessment-report/ar5/.

[13] *Ital-IA, Convegno Nazionale CINI sull'Intelligenza Artificiale*. URL: http://www.ital-ia.it/.

[14] Y. LeCun and Y. Bengio. "Convolutional Networks for Images, Speech, and Time-Series". In: *The Handbook of Brain Theory and Neural Networks*. Ed. by M. A. Arbib. MIT Press, 1995.

[15] *Matplotlib: Python Plotting*. URL: https://matplotlib.org/.

[16] *Meteorological vs Astronomical seasons*. URL: https://www.ncei.noaa.gov/news/meteorological-versus-astronomical-seasons.

[17] Tom Mitchell. *Machine learning*. 1997. Chap. 1.1.

[18] *MongoDB, $geoWithin operator*. URL: https://docs.mongodb.com/manual/reference/operator/query/geoWithin/.

[19] *MongoDB, The database for modern applications*. URL: https://www.mongodb.com/.

[20] *OpenCV documentation*. URL: https://docs.opencv.org/master/d1/dfb/intro.html.

[21] *OpenSense – TEC - Computer Engineering Group — ETH Zurich*. URL: https://tec.ee.ethz.ch/research/networked-embedded-systems/opensense.html.

[22] *OpenStreetMap Tile Servers*. URL: https://wiki.openstreetmap.org/wiki/Tile_servers.

[23] *OpenStreetMap TMP protocol*. URL: https://wiki.openstreetmap.org/wiki/TMS.

[24] *Pandas aggregation mechanisms*. URL: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.aggregate.html.

[25] *Python Data Analysis Library GitHub repository*. URL: https://github.com/pandas-dev/pandas.

[26] *PyTorch, open source machine learning framework*. URL: https://pytorch.org.

[27] Frank F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65 6 (1958), pp. 386–408.

[28] *scikit-image framework*. URL: https://scikit-image.org/docs/dev/api/skimage.io.html#skimage.io.imread.