Πανεπιστήμιο Δυτικής Αττικής

Σχολή Μηχανικών

Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Διπλωματική εργασία

Μελέτη του αντίκτυπου της πανδημίας COVID-19 στην ποιότητα ατμοσφαιρικού αέρα με χρήση βαθέων νευρωνικών δικτύων σε πολυτροπικά δορυφορικά δεδομένα

Αγαπίου Αντώνης
711141081

Επιβλέπων:

Δρ. Αθανάσιος Βουλόδημος
Επίκουρος Καθηγητής

Αιγάλεω - Αθήνα, Οκτώβριος, 2021

University of Wests Attica

School of Engineering

Department of Informatics and Computer Engineering

Diploma Thesis

Impact assessment of COVID-19 lockdowns on air quality using deep neural networks on multimodal satellite data

Agapiou Antonis
711141081

Supervisor:

Dr. Athanasios Voulodimos
Assistant Professor

Aigaleo - Athens, October, 2021

Η παρούσα διπλωματική εργασία παρουσιάστηκε από τον Αγαπίου Αντώνη (711141081) στις 11/10/2021. Εγκρίθηκε από την εξεταστική επιτροπή

Athanasios Voulodimos Digitally signed by Athanasios Voulodimos Date: 2021.11.05 22:53:19 +02'00'	Georgios Bardis Digitally signed by Georgios Bardis Date: 2021.11.11 19:29:24 +02'00'	Eftychios Protopapadakis Digitally signed by Eftychios Protopapadakis Date: 2021.11.06 15:30:16 +02'00'
Αθανάσιος Βουλόδημος Επίκουρος Καθηγητής	Γεώργιος Μπαρδής Επίκουρος Καθηγητής	Ευτύχιος Πρωτοπαπαδάκης Υπότροφος ΕΣΠΑ

Αγαπίου Αντώνης
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών
Πανεπιστήμιο Δυτικής Αττικής

Δήλωση συγγραφέα διπλωματικής εργασίας

Ο κάτωθι υπογεγραμμένος Αγαπίου Αντώνιος του Νικολάου, με αριθμό μητρώου 711141081 του Τμήματος Μηχανικών της Σχολής Πληροφορικής και Υπολογιστών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι: «Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος. Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου». Επιθυμώ την απαγόρευση πρόσβασης στο πλήρες κείμενο της εργασίας μου μέχρι 31/12/2021 και έπειτα από αίτηση μου στη Βιβλιοθήκη και έγκριση του επιβλέποντα καθηγητή.

Ο Δηλών

Acknowledgements

I would like to express my sincere thanks to those who offered feedback on the content and made this thesis possible . Firstly, I would like to thank my supervisor Dr. Athanasios Voulodimos for all the great advice and guidance he has provided and for offering this interesting topic. Next, I would like to thank Dr. Maria Kaselimi for her continuous support and fruitful discussions throughout the making of this thesis. Additionally, I want to extend my gratitude to my family for their care and support during my studies.

10/19/2021
Agapiou Antonis

Περίληψη

Σκοπός αυτής της εργασίας είναι να εκτιμηθεί το αντίκτυπο των lockdown της COVID-19 πανδημίας στην ατμοσφαιρική ποιότητα με την χρήση μοντέλων βαθείας μάθησης. Το κύριο κίνητρο πίσω από αυτό το ερευνητικό έργο είναι η ανάγκη κατανόησης της αντίδρασης της ατμοσφαιρικής ποιότητας σε κοινωνικοοικονομικές δραστηριότητες όπως τα lockdowns και η εφαρμογή μοντέλων ικανά να μάθουν αυτήν την σχέση. Χρησιμοποιήσαμε δεδομένα πολλαπλών περιοχών τα οποία περιέχουν δορυφορικές εικόνες CO και NO2 συγκεντρώσεων σε συνδυασμό με πολιτικές που εφαρμόσθηκαν για την μείωση της εξάπλωσης του κορονοϊού. Για την ολοκλήρωση αυτής της εργασίας υλοποιήθηκαν μοντέλα βαθείας μάθησης, ειδικότερα CNN-LSTM, ConvLSTM και 3D-UNet και έπειτα αξιολογήθηκαν μεταξύ τους για να βρεθεί το καλύτερο μοντέλο. Τα αποτελέσματά μας έδειξαν καλή απόδοση στο 3D-UNet νευρωνικό δίκτυο, ξεπερνώντας τα CNN-LSTM και ConvLSTM μοντέλα. Έπειτα, μελετήσαμε την επίδραση των lockdown χαρακτηριστικών στην προγνωστική ικανότητα του προτεινόμενου μοντέλου. Συνοψίζοντας, τα ευρήματα μας έδειξαν πως δεν είχαν σημαντική επίδραση στην ποιότητα των προγνώσεων του μοντέλου.

Λέξεις Κλειδιά: Βαθεία μάθηση, Όραση Υπολογιστών, Ατμοσφαιρική ποιότητα, Πρόγνωση χρονοσειρών

Abstract

The purpose of this diploma thesis is to assess the impact of the lockdowns caused by the COVID-19 pandemic on air quality by using deep learning models. The primary motivation behind this research project is the need to understand how air quality responds to social-economic activities such as city-level lockdowns and implement models able to learn that relationship. We use datasets from multiple areas that contain satellite imagery of CO and NO2 air pollutants' concentrations in combination with policy responses for the COVID-19 pandemic to predict their respective future image occurrences. Deep learning models, namely., CNN-LSTM, ConvLSTM, and 3D-UNet models are implemented to complete this task and then evaluated to discover the best performing network. Our results showed good performance on the 3D-UNet based network, surpassing both the CNN-LSTM and ConvLSTM networks. We then assess the effect of the lockdown features on the prediction performance of the 3D-UNet model. Overall, our findings show that the lockdown features don't have a notable impact on the model's predictions' quality.

Keywords: Deep learning, Computer Vision, Air quality, Time-series prediction

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This first chapter provides an overview of the challenges around air pollution and motivation for this research, followed by details of the research goal. Lastly, a summary of the contributions is presented, along with the outlines of the thesis structure.

1.1 Motivation

Air is vital for all living beings on earth and the necessity of healthy air has always been of great importance. Rapid urbanization and industrialization had a profound influence on air pollution making it a major modern-day issue. Air pollution is defined as all destructive effects of any sources which contribute to the pollution of the atmosphere and/or deterioration of the ecosystem. It is made up of many kinds of pollutants including materials in solid, liquid, and gas phases [GRB16]. Air pollution is usually caused by energy production from power plants, industries, residential heating, fuel burning vehicles, natural disasters etc.

Human health concern is one of the most important consequences of air pollution, especially in urban areas. Long-term exposure to pollutants can cause neurological, reproductive and respiratory health problems that can even lead to death [Man+20] and according to the World Health Organization air pollution kills an estimated seven million people worldwide every year, 4.2 million of which occur as a result of exposure to ambient (outdoor) air pollution [WHO]. Environmental degradation is another detrimental consequence of air pollution. The most important environmental effects of air pollution are as follows: global warming, photo-chemical smog, acid rain, aerosol formation and depletion of ozone .

All the above mentioned raise an urgent need to anticipate and plan for pollution fluctuations to help communities and individuals better mitigate the negative impact of air pollution. To achieve that it is important to understand how pollution responds to changes in social and economic activity.

With the outbreak of the Coronavirus disease 2019 (COVID-2019), human activities were limited or even prohibited as a series of measures in order to slow down social interactions. These regulatory measures have been implemented across the countries, positively affecting air quality, especially due to the reduction in pollutants emissions by transportation and industries [Wan+20]. Therefore it is important to understand how pollution responds to restriction policies, such as a city-level lockdown, as it inform us how different pollutants may respond to milder forms of restrictions on human activities such as pedestrianised zones,congestion charging and urban planning more generally.

Nowadays the use of deep learning is spreading in the field of remote sensing, with applications ranging from detection and classification of land use and monitoring to the prediction of many natural or anthropogenic phenomena of interest. With the constant increase of remote sensing data and the continuous research on end-to-end models, deep learning can be used to solve many remote

sensing and geoscience problems.Accurate time series forecasting of remote sensing data such as air quality is one such problem, and much effort has been made by researchers to create models capable of fitting the underlying time series.

1.2 Research Goal

The goal of this thesis is to design and examine deep learning models to directly predict air quality from satellite imagery and policy responses to the Coronavirus pandemic in order to assess the impact of these policy responses to air quality. The image data required are publicly available from the Copernicus Sentinel-5P mission and the policy responses data is also publicly available from the Coronavirus Government Response Tracker. For the process of modeling and predicting air quality data we use many state-of-the-art deep learning models-i.e.,fully convolutional networks,auto encoders and recurrent neural networks. Moreover, we use the best performing model to examine and analyse the effects of the lockdowns features on air quality.

1.3 Contributions

This thesis consists of several contributions. Firstly, it provides a dense overview of existing deep learning approaches that deal with the problem of remote sensing data prediction. Secondly it performs an exploratory data analysis of air pollutants, and policy responses to the Coronavirus pandemic. Thirdly it provides a comparison of the performance of multiple deep learning models used to predict air quality data. Furthermore, the proposed model from the comparison is able to predict the effect of the policy responses in the atmosphere, which can be used to quantify this effect on urban environments such as cities. Not only that, but the proposed model is able to learn spatio-temporal data, which means it can be used in different areas and in different time periods. Last but not least, all Tensor Flow implementations are freely available to the research community as is the data extraction and generation source code used to feed the deep learning models[1].

1.4 Overview of the Study

The subsequent chapters of this thesis are structured as followed:

Chapter 2: Background Theory covers the theoretical concepts that are required to understand our deep learning implementations and their consecutive evaluations. It provides a deeper look into neural networks and explains how they are trained. Further, more advanced neural network architectures are explored, namely convolutional neural networks (CNN) for spatial learning, recurrent neural network (RNN) models for sequential learning, and auto-encoders.

Chapter 3: Related Work presents related work to our task. These works are about air quality prediction projects, remote sensing data prediction and sequence-to-one prediction in general.

Chapter 4: Datasets describes in detail the process of data extraction, as well as data preprocessing that was used on each dataset. Furthermore, Chapter 4 mentions also the creation of the input dataset that was used to feed the deep learning models.

Chapter 5: Methodology, the full methodology for the prediction is presented. This chapter describes the architectures and implementations details of the deep learning models, as well as the reasons they were chosen for this particular task.

[1] https://github.com/AntonisAgap/thesis

Chapter 6: Experiments,includes a presentation of all experiments and their results with graphic presentations. Furthermore, Chapter 6 presents the comparison of the results of each model that was used in the experiments.

Chapter 7: Conclusions,concludes the work of this paper with an evaluation, discussion of the results, as well as highlight the identified possible improvements for future work.

Chapter 2

Background Theory

This chapter introduces relevant background theory for the reader to pick up on key terminology used throughout this thesis. Section 2.1 explains in short the COVID-19 pandemic and it's responses by the governments. In the Section 2.2 air quality is defined along with the causes and effects of its decrease . Section 2.3 describes the structure of simple feed-forward networks, their training process and some regularization methods to improve their performance. In Sections 2.4 2.5 and 2.6 more advanced model architectures are presented that take advantage of the data's spatial and temporal properties. Namely, Section 2.4 presents the Convolutional Neural Networks, Recurrent Neural Networks are explained in Section 2.5 and lastly Encoder-Decoder Networks are described in Section 2.6. Lastly, in Section 2.7 the Batch Normalization technique is explored.

2.1 COVID-19 pandemic

As of September 2021 the COVID-19 pandemic is an ongoing global pandemic of coronavirus disease 2019 (COVID-19), caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2), which was the cause for over 4.14 million deaths worldwide. The disease transmits when people breathe in air contaminated by droplets and small airborne particles, making the virus' spread rapid. In order to stop the spread of the virus, restriction policies have been applied by the governments. Social distancing, face masks in public, city-level lock downs are some of those policies.

2.2 Air Quality

Air quality is the degree to which air is suitable or clean enough for humans, animals, or plants to remain healthy. Air quality decreases through the release of pollutants into the air, which are harmful to the health of humans and other living beings, or cause damage to the climate or to the materials. The presence of air pollutants in the atmosphere defines air pollution. Table 2.1 lists all the 'criteria air pollutants', along with a short description of each. 'Criteria air pollutants' is an internationally used term to describe air pollutants that have been regulated and are used as indicators of air quality.

2.2.1 Causes of air pollution

Sources of air pollution are either anthropogenic (human-made) or natural. Anthropogenic sources are mostly related to the burning of fuel. Fossil fuel power plants, vehicles, and burn practices in agriculture are such examples. Other than combustion, fumes from solvents, waste deposition, and fertilized farmland also contribute to air contamination. Natural sources include among other

volcanic eruptions, windstorms, biological decay, and forest fires. The severity of air pollution in an area depends on three factors: the number of pollutants, the rate at which they are released into the atmosphere, and how long they are trapped in an area. If air pollutants are in an area with good airflow, they will mix with the air and quickly disperse. Air pollutants tend to remain in the air where there are certain conditions like light winds or obstacles that restrict the transport of these contaminants away from an area. Lastly, the increase of industrial and socioeconomic activities in cities heavily impacts the rapid reduction of air quality over those areas.

2.2.2 Effects of air pollution

Effects of air pollutants have a negative impact not only on human health but also on the whole environment [GRB16]. Ecologically, air pollution can cause major environmental damages to the groundwater, soil, air. It is also a serious threat to wildlife. Researchers on [Lov+09],[Mel+16] show the severe impact of air pollutants on diminishing species diversity. Haze, temperature inversion, acid rain, and global climate change are other major environmental impacts of air pollution [Man+20]. In terms of health hazards, long-term effects of air pollution on the onset of diseases such as respiratory infections and inflammations, cardiovascular dysfunctions, and cancer is widely accepted; hence, air pollution is linked with millions of death globally each year.

Name	**Information**
Carbon monoxide (CO)	Main sources of CO are combustion of fossil fuels, biomass burning, and atmospheric oxidation of methane and other hydrocarbons.
Lead (PB)	Lead is a naturally occurring heavy metal that can be released into soil,air and water through soil erosion, volcanic eruptions, sea spray and bushfires.
Nitrogen Dioxide (NO_2)	Nitrogen dioxide enters the atmosphere as a result of anthropogenic activities such as fossil fuel combustion and biomass burning, as well as natural processes including microbiological processes in soils, wildfires and lightning.
Ozone (O_3)	At high concentrations ozone becomes harmful to the health of humans,animals and vegetation. Ozone is also an important greenhouse-gas contributing to ongoing climate change.
Particulate matter (PM_x)	Breathing in particle pollution can be harmful to human health. $PM_{2.5}$ denotes the diameter of the particulate matter is less than 2.5 microns, and for PM_{10} it is 10 microns. Most particles form in the atmosphere as a result of complex reactions of chemicals such as sulfur dioxide and nitrogen oxides, which are pollutants emitted from power plants, industries and automobiles.
Sulphur Dioxide (SO_2)	(SO_2) enters Earth's atmosphere through both natural and anthropogenic processes, though the majority is of anthropogenic origin.

Table 2.1: Criteria air pollutants. NO_2 and CO are the pollutants of interest in this thesis.

2.3 Neural Networks

The term 'neural network' has its origins in attempts to find mathematical representations of information processing in biological systems . They were first introduced back in 1943 by the neurologist Warren McCulloch and the mathematician Walter Pitts. Their work inspired Frank Rosenblatt to develop the so-called perceptron algorithm in 1962 [Bis06]. But by the 1990s, powerful alternative machine learning techniques such as Support Vector Machines were favored by most researchers, as they seemed to offer better results. However nowadays, with the availability of large amounts of training data and computational power, neural networks frequently outperform other machine learning techniques on very large and complex problems.

2.3.1 Basics

Rosenblatt's *perceptron* is the simplest form of an ANN architecture. It corresponds to a two-class model in which the weighted sum $\sum_{i=1}^{n} w_i x_i = w \cdot x$, where w and x are vectors whose components are n weights and n inputs respectively, is used to construct a generalized linear model of the form:

$$y(x) = f(\sum_{i=1}^{n} w_i x_i + w_0) \tag{2.1}$$

where the nonlinear activation function $f(\cdot)$ is given by a step function of the form

$$f(x) = \begin{cases} +1 & a \geq 0 \\ -1 & a < 0 \end{cases} \tag{2.2}$$

A bias weight w_0 is typically included, which provides the ability to shift the results for a better fit. A single perceptron can learn linear functions correctly. However, by building a network of these models arranged into more layers consisting of multiple non-linear activation functions in each, the network can now solve more complex problems. Such network architecture is known as a *multilayer perceptron* (MLP).

An MLP is composed of one (pass through) input layer, one or more "hidden" layers and a final layer called the output layer. It is common to describe the structure of a neural network as a graph whose input,hidden and output variables are represented by nodes, and the weight parameters are represented by links between the nodes. See Figure 2.1 for an example of a feed-forward MLP with two hidden layers.

We can write down the neural network's function corresponding to Figure 2.1 as follows. The output of the m_{th} hidden unit of the first hidden layer is obtained by calculating the weighted sum of the D input values, adding the bias w_{m0} and then transforming it using a differentiable, nonlinear activation function $g_1(\cdot)$.

$$u_m = g_1 (\sum_{d=1}^{D} w_{md}^{(1)} x_d + w_{m0}^{(1)}) \tag{2.3}$$

where $w_{md}^{(1)}$ denotes a weight in the first layer, going from input d to output m. Similarly the output of the l_{th} hidden unit of the second hidden layer and k_{th} output unit of the output layer are obtained.

$$z_l = g_2 (\sum_{m=1}^{M} w_{lm}^{(2)} u_m + w_{l0}^{(2)}) \tag{2.4}$$

$$y_k = \tilde{g} (\sum_{l=1}^{L} w_{kl}^{(3)} z_l + w_{k0}^{(3)}) \tag{2.5}$$

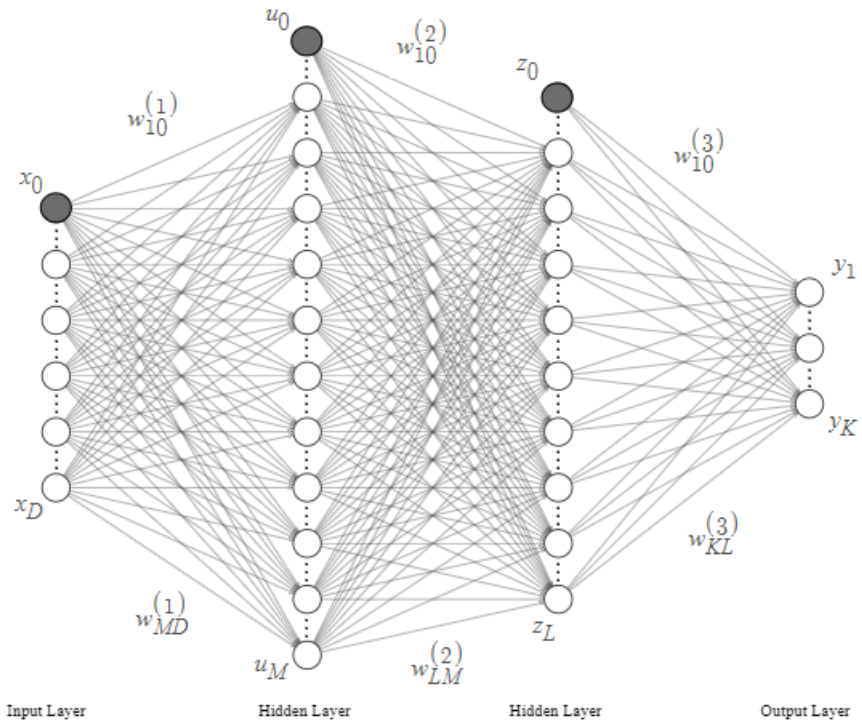


Figure 2.1: An example graph of an MLP with an input layer x of D nodes, two hidden layers u of M nodes and z of L nodes and lastly an output layer y of K nodes. The grayed out nodes represent the additional input and hidden variables x_0, u_0 and z_0 respectively, whose links denote the bias weight parameters.(Based on [Bis06])

Using (2.3),(2.4) and (2.5) the overall network's function of a four layers feed-forward MLP can be defined as:

$$y_k(x,w) = \tilde{g}\left(\sum_{l=0}^{L} w_{kl}^{(3)}\, g_2\left(\sum_{m0}^{M} w_{lm}^{(2)}\, g_1\left(\sum_{d=0}^{D} w_{md}^{(1)}\, x_d\right)\right)\right) \tag{2.6}$$

The notation $\tilde{g}$ is used for the output units' activation function, and its choice is determined by the nature of the data and assumed distribution of target variables [Bis06]. The activation functions used in this thesis are the *sigmoid*, *hyperbolic tangent* and *ReLU* functions. Their graphical representations can been seen in 2.2

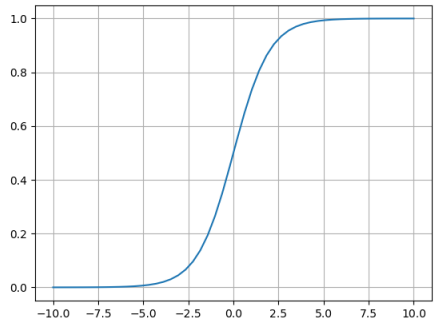
The *sigmoid* function g(x) can be regarded as a smoothed version of a step function, which squashes real numbers to range between $[0, 1]$. It is a bounded differentiable real function, which is given by the relationship

$$g(x) = \frac{1}{1+e^{-x}}, \tag{2.7}$$

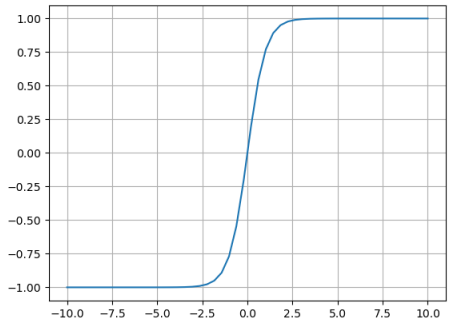
The function appears in the output layers of the deep learning architectures, and is usually used for predicting probability based output and has been applied successfully in binary classification problems, modeling logistic regression tasks as well as other neural network domains [Nwa+18]. Its main advantages is that its derivative takes the simple form:

$$g'(x) = g(x)(1-g(x)), \tag{2.8}$$

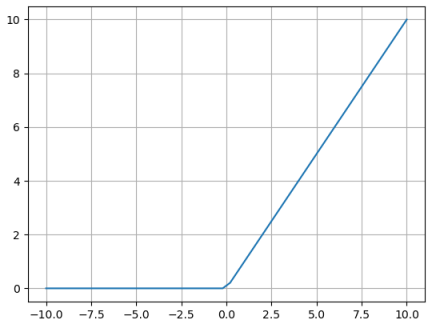
which is straightforward to verify and understand. However, the *sigmoid* function suffers major drawbacks, which include gradient saturation, slow convergence and non-zero centred output



(a) $g(x) = \dfrac{1}{1+e^{-x}}$



(b) $g(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$



(c) $g(x) = max\,(0,x)$

Figure 2.2: Graphical representations of the (a) sigmoid, (b) hyperbolic tangent and (c) ReLU activation functions

thereby causing the gradient updates to propagate in different directions.

The *hyperbolic tangent* or *tanh* function was proposed to remedy some of these drawbacks. The *tanh* function is a zero-centered function, which squashes real numbers to the range $[-1,1]$, thus the output of the *tanh* function is given by:

$$g(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2.9}$$

By providing a zero centered output the function aids the back-propagation training progress which will be discussed in Section 2.3.2. Nonetheless, the *tanh* function also suffers by gradient saturation.

Lastly we have the *rectified linear unit (ReLU)* function, which has been the most widely used activation function for deep learning applications with state-of-the-art results to date [Nwa+18]. The *ReLU* function performs a threshold operation to each input element where the values less than zero are set to zero, therefore the output of the function is given by

$$g(x) = max\,(0,x) = \begin{cases} x_i, & x_i \geq 0 \\ 0, & x_i < 0 \end{cases} \tag{2.10}$$

This function rectifies the values of the inputs less than zero thereby forcing them to zero and eliminating the vanishing gradient problem observed in the earlier types of activation functions. Moreover, unlike the aforementioned activation functions, *ReLU* guarantees faster computation,

since it doesn't involve expensive operations such as exponentials and divisions. Unfortunately, *ReLU* can be fragile during training, causing some of the units to never activate during the whole training, but with a proper setting of the learning rate this is less frequently an issue.

2.3.2 Network Training

The final goal of a neural network's training algorithm is to find a set of weights w, that minimize a loss function[1] $E(w)$ [TSK06] :

$$argmin_w E(w) \tag{2.11}$$

The loss function is the function that computes the distance between the model's predicted output y and the expected output $\hat{y}$. Similarly to the activation functions, the form of the cost function is highly determined by the nature of the input and target variables. In this thesis the cost functions which are used are: the Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE).

The *Mean Squared Error* is usually the default loss function used in regression problems. Its output is calculated as the average of the squared differences between the predicted and actual values.

$$E(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.12}$$

The result is always a positive number and an MSE of 0.0 is considered a perfect value. Similarly the *Root Mean Squared Error* is calculated by the rooted *MSE*.

$$E(w) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.13}$$

Lastly the *Mean Absolute Error* is calculated as the average of the absolute difference between the actual and predicted values.

$$E(w) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.14}$$

As mentioned earlier, minimizing the loss function is the goal of the training process. Many optimizations techniques have been developed to solve such minimization problems and most of those techniques are based on the *gradient descent* method. *Gradient descent* makes use of the derivative of the loss function $E(w)$ with respect to the weights w and iteratively tries to find the global minima of the function by doing small steps towards the negative gradient. By using *gradient descent* any trainable weight parameter of the neural network can be updated by calculating:

$$w_i^{(\tau+1)} = w_i^{(\tau)} - \lambda \nabla E(w_i^{(\tau)}) \tag{2.15}$$

$$\nabla E(w^t) = \frac{\partial E}{\partial w_i^{(\tau)}} \tag{2.16}$$

where τ labels the iteration step and $\lambda > 0$ denotes the *learning rate* or *step size* that is used to scale the gradient and control how much to change each input variable with respect to the gradient [Bis06]. Choosing the correct learning rate is crucial since a learning rate that is too large can cause the model to converge too quickly to a suboptimal solution, whereas a learning rate that is too small can cause the process to stuck on a local minima.

[1] Sometimes also called *cost function*, *error function* or *objective function*.

Calculating the gradients for nodes in the hidden layers of a feed-forward network can be a challenge without knowing their outputs' value. An iterative differentiation algorithm called *error back-propagation* is applied to solve this problem. Each iteration of the algorithm, also called *epoch*, is composed of two passes: a forward pass and a backward pass. In the forward pass: the input data is propagated to the input layer and goes through the hidden layers. Afterwards, the network's predictions are measured from the output layer, and lastly the network's errors are calculated using those predictions and a loss function $E(w)$. Once the network's errors are calculated, the forward pass ends and the backward pass starts. In the backward pass, the network's errors are propagated from the output layer to the input layer passing through the hidden layers calculating the gradients of the nodes. After the backward pass ends, the weights are updated using a *gradient descent*-based algorithm.

In practice the *vanilla gradient descent* is a poor approach to update the weights of a neural network since it uses the whole training dataset to perform just *one* update and in result can be very slow and is intractable for datasets that don't fit in memory. *Stochastic gradient descent* (SGD) solves this problem by updating the parameters based on one data point at a time so that [Bis06],

$$w_i^{(\tau+1)} = w_i^{(\tau)} - \lambda \nabla E_n(w_i^{(\tau)}) \tag{2.17}$$

The update can be also based on a sample of data points, where the size of the sample is known as *batch size*. The optimizer that is used in this thesis is called *adaptive moments estimation* (Adam) algorithm [KB15]. This algorithm is usually preferred as the default optimization method for most deep learning applications since it's been empirically shown that it compares favourably to other *gradient descent*-based methods [Rud16] [KB15]. Adam is an extension to SGD, but unlike SGD, which maintains a single learning rate for all weight updates, the Adam algorithm computes individual adaptive learning rates for different parameters. This is achieved by utilizing the estimates of first (the mean) $m^{(t)}$ and second (the uncentered variance) $u^{(\tau)}$ moments of the gradients,

$$m^{(\tau)} = \beta_1 m^{(\tau-1)} + (1-\beta_1)\nabla E_n(w^{(\tau-1)}) \tag{2.18}$$

$$u^{(\tau)} = \beta_2 u^{(\tau-1)} + (1-\beta_2)\nabla E_n(w^{(\tau-1)})^2 \tag{2.19}$$

to calculate the parameters' exponential moving average of the gradients and the squared gradients. The hyperparameteres $\beta_1, \beta_2 \in [0,1]$ control the decay rates of the previous mentioned moving averages. Using these, the *update rule* of the *Adam* algorithm can be written as:

$$w_i^{(\tau+1)} = w_i^{(\tau)} - \frac{\lambda}{\sqrt{\hat{u}^{(\tau)}} + \varepsilon}\hat{m}^{(\tau)} \tag{2.20}$$

, where $\hat{m}^{(\tau)}$ and $\hat{u}^{(\tau)}$ are the bias-corrected first and second moment estimates respectively,

$$\hat{m}^{(\tau)} = \frac{m^{\tau}}{1-b_1^{\tau}} \tag{2.21}$$

$$\hat{u}^{(\tau)} = \frac{u^{\tau}}{1-b_2^{\tau}} \tag{2.22}$$

They hyperparameters typically require little tuning and the authors propose default values of 0.001 for λ, 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ε [KB15].

Before training a machine learning model, the dataset is usually split into a *training*, a *validation* and a *testing* dataset. The *training dataset* is used for the calculation of the model's parameters as described and is usually consisted of the more substantial part of the dataset. The *validation* dataset is a smaller subset of the *training* dataset and is used to evaluate the network's performance

during the training process. After the training process, the *testing dataset* is used to evaluate the performance of the network's prediction capability on unseen data of the same type. The network's ability to correctly predict outputs of a dataset other than the training dataset is called "generalization", and therefore the objective of such a network is to be able to generalize on any kind of data from the same type [Bis06].

2.3.3 Regularization

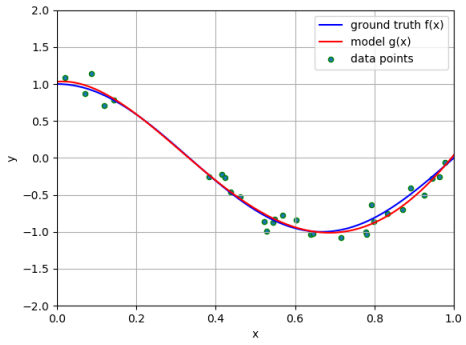
As already stated, our goal is to find a representation that generalizes well. Common problems that have to be prevented when neural networks are trained are the effects of *under-fitting* and *over- fitting*. An under-fitted model is incapable of capturing the variability of data and instead it makes a strong assumption about the data paying little attention to the data points. An example of such model can been seen in (b) of Figure 2.3, where the model makes the assumption that the data is linear, and fails to learn the relationships between the inputs (features) x and outputs (labels) y. Reducing under-fitting can be usually achieved by increasing the model's complexity and training time [Koe19]. Over-fitting is the opposite of under-fitting. An over-fitted model has a high variance, because it will change significantly depending on the training data. It learnt some random regularity contained in the set of training patterns and essentially learns to memorize the training data points and noise instead of learning the relationships between them. A model like that has very bad generalization, making over-fitting one of the biggest problems in training neural networks. In (c) of Figure 2.3 an example of an over-fitted model can been seen. Ideally we want to train a model like (a) in Figure 2.3, where the model's function approximates the true function almost perfectly. We can evaluate a model's generalization by comparing the loss functions of the training, validation and testing datasets. Usually a model that is under-fitted will have high training, validation and testing error, whereas an over-fitted model will have extremely low training error but a high validation and testing error.

Avoiding the effects of over-fitting and under-fitting can be achieved by using *regularization* to our deep learning problem. *Regularization* is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. Furthermore, *regularization* from the start must always be considered, unless the *training* dataset contains a huge amount of samples[GBC16]. In this thesis regularization techniques such as *dropout* and *early stopping* are employed to address the over-fitting and under-fitting problems.

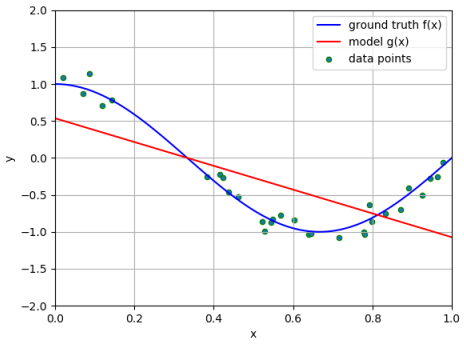
Dropout

Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. The term dropout refers to the temporal removal of units from a layer in a network, along with all its incoming and outgoing connections, as can be seen in Figure 2.4. This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer, which in turn prevents units from co-adapting too much. [WWL13]. Essentially by applying dropout to a network's training, we sample a new *thinned* network from it, for each training case.

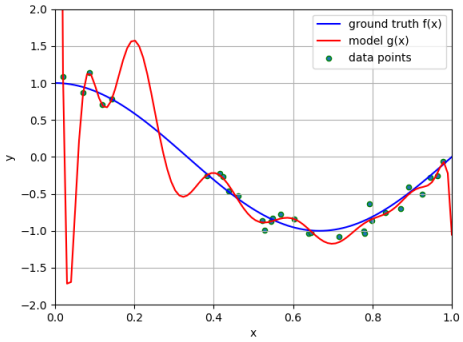
The choice of which units to drop is random and is defined by a hyperparameter called *dropout rate*. The dropout rate specifies the probability at which outputs of the layer are dropped out, or inversely, the probability at which outputs of the layer are retained. A common value for the hidden layers is a probability of 0.5, which seems to be optimal for a wide range of networks and tasks. However, for the input units, the optimal probability of retention is usually closer to 1, such as 0.8 [Sri+14]. Dropout is used only in the training process of the neural network and may be implemented on any or all hidden layers in the network as well as the visible or input layer. It is not used on the output layer. Furthermore, it can be used with most types of layers, such as dense fully connected layers, convolutional layers, and recurrent layers. This technique was



(a) Good fitted function



(b) Under fitted function



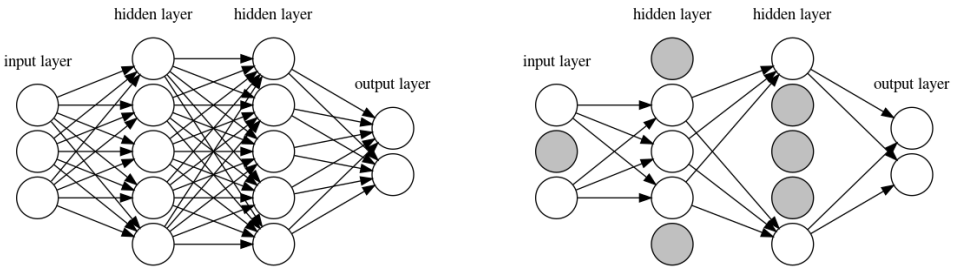
(c) Over fitted function

Figure 2.3: Graphical representations of different fitted functions

found to improve the performance of neural networks in a wide variety of application domains [DSH13][Sri+14][GBC16]. However, one of its major drawbacks is that it increases training time. A dropout network typically takes 2-3 times longer to train than a standard neural network of the same architecture, since the parameter updates on the dropout network are very noisy[Sri+14].

Early stopping

When training a large network, there will be a point during the training when the model will stop generalizing and start learning the statistical noise in the training dataset. As discussed above, this is the effect of over-fitting, making the model less useful at making predictions on new data. Another approach used in this thesis to solve over-fitting is by using a *trigger*, that will stop the training process when the generalization performance starts to decrease. This approach is called *early stopping* and is probably the most commonly used form of regularization in deep learning. In this thesis, a model's weights will be saved as long as its performance is better than the model's at the prior epoch. In that way , every time the error on the validation set improves, we store the best model's copy. An increase in validation loss over a given number of epochs will trigger the end of the training process, leaving us with the model's weights, whose generalization error (validation loss) was the lowest. Early stopping is recommended to be used almost universally [GBC16] since it's simple and offers almost always better performance.



(a) Feed-forward Neural Net (b) After applying dropout

Figure 2.4: Graphical representation of the dropout regularization of a neural network. **Left**: A standard feed-forward neural network with 2 hidden layers. **Right**: An example of a "thinned" neural network after applying dropout to the network on the left. Grayed out units have been dropped (Based on [Sri+14])

2.4 Convolutional Neural Networks

In the previous section, neural networks have been described that exhibit a full connection of neurons from one layer to the next. These types of layers are called *fully-connected layers*. Although *fully-connected layers* allows us to train powerful networks, there are some downsides to them. One of them is the inability to effectively learn on more complicated datasets such as images. Given a dataset of gray scale images [2] with the standardized size of 32x32 pixels each, a traditional feed-forward neural network would require 1024 input weights and that's just the input layer. Considering the input layer, the network's layers would become very large and their connections would increase exponentially. This would end up in a network that is either too time-consuming to train, or too big to even be able to be stored in memory. Furthermore, by training such a network, we would discard each image's spatial structure by flattening them into one-dimensional vectors. Therefore we wouldn't be able to use the knowledge of the pixels spatial relations to build efficient models to learn from image data.

This section introduces *convolutional neural networks* (CNNs) that are designed for the purpose of effectively learning from tabular image data, even though they are now used for a wide range of tasks other than computer vision . In addition to their efficiency in achieving accurate models, CNNs tend to be computationally efficient, since they require fewer parameters than fully-connected architectures and because convolutions are easily parallelized across GPU cores. CNNs emerged from the study of the brain's visual cortex, and they have been used in image recognition since the 1980s [Gro17]. An important milestone was a *1998* paper [Lec+98], which introduced the famous *LeNet-5* architecture,widely used to recognize handwritten check numbers, see Figure 2.5. Nowadays, the use of CNN-based architectures is ubiquitous in the field of computer vision, and they became so dominant that practitioners often apply CNNs whenever possible, even on tasks with one-dimensional sequence structure, such as voice recognition or natural language processing. The basic elements of CNNs —i.e., convolutional layers, pooling layers, padding and stride as well as its detailed advantages are described in the following sections.

[2] A gray scale image is one in which the value of each pixel consists of a single value, thereby the image has only one channel.

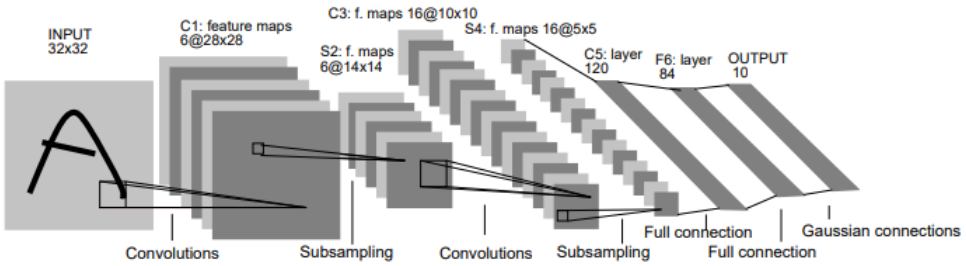


Figure 2.5: Graphical representation of LeNet-5 architecture. The convolution and sub sampling layers extract high level features from the images, while the full connection layers are used to accurately predict the label using the high level features.(From [Lec+98])

2.4.1 Structure

Convolutional neural networks are networks that consist of at least one *convolutional* layer. In other words, *"Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers"* [GBC16]. Therefore in order to accurately describe what a CNN is, it's essential we define the *convolution operation*.

Generally speaking, a *convolution* is a mathematical operation of two functions $f(x)$ and $g(x)$ which is typically denoted with an asterisk. *Convolution* is defined as the integral of the product of the two functions after one is reversed and shifted:

$$(f * g)(t) = \int_{\infty}^{-\infty} f(\tau) \cdot g(t-\tau) d\tau \tag{2.23}$$

In *convolutional neural networks* terminology, the first argument to the convolution (the function f) is termed as the *input* and the second argument (the function g) as the *kernel*. Moreover, the output of $(f * g)(x)$ is called *feature map*. Technically, the *convolution* as described in the used of the *convolution neural networks* is actually *cross-correlation*, since the *kernel* isn't flipped prior to being applied to the input, see Figure 2.6. *Cross correlation* is defined as:

$$(f * g)(t) = \int_{\infty}^{-\infty} \overline{f(t-\tau)} \cdot g(\tau) d\tau \tag{2.24}$$

, where $\overline{f(t-\tau)}$ denotes the complex conjugate of $f(t-\tau)$. Nevertheless, in deep learning, it is still referred to as *convolution operation*. Furthermore, since we are dealing with discrete images of multiple channels in this thesis, we can reformulate the equation 2.24 as:

$$(I * K)(x,y) = \sum_{i=1}^{n_H} \sum_{j=1}^{n_W} \sum_{k=1}^{n_C} K_{i,j,k} \cdot I_{x+i-1,y+j-1,k} \tag{2.25}$$

, where n_H is the height of the image, n_W the width and n_C the number of channels. The input I is an image of size $n_W \times n_H \times n_C$ and K is a three-dimensional kernel with a window's (receptive field) size of $k \times k \times n_C$. Being more specific, 2.25 is a *discrete convolution*. *Discrete convolutions* are the bread and butter of CNNs and their advantages over affine transformations for learning will be further discussed in Section 2.4.2.

Convolution layer

The core computation of a N-dimensional convolutional layer is a N-dimensional cross correlation operation. A convolutional layer cross correlates the input and kernel and adds a scalar bias to

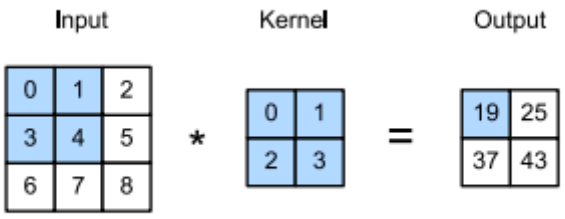


Figure 2.6: Two-dimensional cross correlation operation. The shaded portions are the first output element as well as the input and kernel tensor elements used for the output computation. The kernel window will slide, both from left to right and top to bottom to calculate the other three output elements. (From [Zha+19a])

produce the feature map. The feature map can be regarded as the learned learned representations (features) in the spatial dimensions (e.g., width and height) to the subsequent layer. It is common for a convolutional layer to use multiple kernels in parallel for a given input to produce multiple feature maps. This gives the model multiple different ways of extracting features from an input. The kernels are typically initialized randomly, just as a a fully-connected layer would [Zha+19a], and their number is specified by the *kernel depth*. In summary, in a *convolution* layer, the *kernel* iteratively slide across the whole input space. In every iteration step, it attempts to extract features that are only dependent on a small neighboring region with the size of the kernel using the *cross correlation operation* and outputting a feature map. The process is repeated several times using different *kernels*, resulting in multiple feature maps. Furthermore, for computational efficiency or simply downsampling, it's standard to use a hyperparameter called *stride* to skip intermediate locations by moving the kernel window more than one element at a time. Stride refers to the number of rows and columns traversed per slide. When the stride is one then we move the filters one pixel at a time, when its two then the filters jump two pixels at a time as we slide them around and so forth. Lastly, sometimes its convenient to pad the input data around the border with zeros. The size of this padding is specified by the hyperparameter *zero-padding*. *Zero-padding* allows us to control the spatial size of the output, which is essential in this thesis, because it enables us to preserve the spatial size of the input so the input and output dimensions are the same. For a graphical example of a convolution that makes use of *zero-padding* and *stride* see Figure 2.7.

By using the equation 2.25, the *bias* and the *stride* terms, we can formulate the output of a neuron in a convolutional layer as:

$$z_{i,j,k} = b_k + \sum_{u=1}^{f_h} \sum_{v=1}^{f_w} \sum_{k'=1}^{f_{n'}} x_{i',j',k'} \cdot w_{u,v,k',k} \quad with \begin{cases} i', & = i \times s_h + u \\ j', & = j \times s_w + v \end{cases} \tag{2.26}$$

, where $z_{i,j,k}$ is the output of the neuron located in row i, column j in feature map k of the convolutional layer l. Notations s_h and s_w are the dimensions of the *stride*, while f_h, f_w are the dimensions of the receptive field and f'_n the number of feature maps in the previous layer or channels if its the input image. The output is denoted as $x_{i',j',k'}$. The *bias* term for feature map k is denoted as b_k, whereas $w_{u,v,k',;}$ is the connection weight between any neuron in feature map k of the layer l and its input is located at row u and column v and feature map k'.

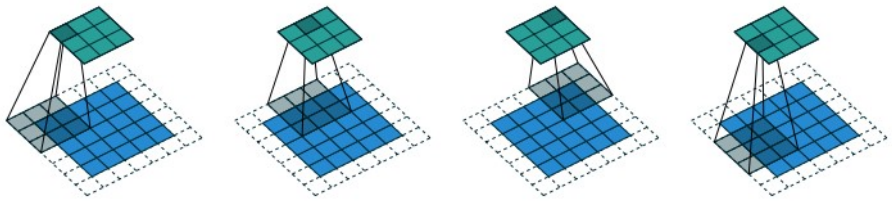


Figure 2.7: Graphical example of a convolution with a 3×3 kernel over a 5×5 input using a stride of 2×2. The input has been padded with a 1×1 border of zeros (*zero-padding*). (From [DV16])

Each convolutional layer is usually followed by a non-linear activation function, preferably a rectifier linear unit (ReLU). As can been seen in Figure 2.5 the convolutions layers are stacked together. This architecture allows the network to concentrate on low-level features on the first convolution layer,then assemble them into higher-level features in the convolution layer, and so on. This hierarchical structure is one of the main reasons why CNNs work so well in computer vision tasks [Gro17].

Pooling layer

It is a common practice to periodically insert an additional layer that performs *sub sampling* onto the feature maps. This layer is referred to as the *pooling* or *sub sampling* layer. Its function is to progressively reduce the spatial size of the representation in order to reduce the computational load, the memory usage, and the number of parameters, thereby also controlling the effect of over fitting. The *pooling layer* operates upon each feature map separately to create a new set of the same number of pooled feature maps. Much like the *convolution layer*, a *kernel slides* across the input feature maps applying a pooling operation over the portion of the image covered by its kernel's window's size. The size of the pooling operation is smaller than the size of the feature map; generally, it is almost always with a window size of 2×2 pixels and with a *stride* of 2 pixel to avoid any overlap. Two common pooling operations used are: *max pooling* and *average pooling*. In *max pooling* the function returns the maximum value of the receptive field's values, whereas in *average pooling* the function returns the average value. Usually, the *max* function is preferred in *pooling layers* since it performs a lot better than the *average* function. In all cases, the result of a pooling operation is a summarized version of the features detected in the input, thereby pooling helps to make the representation become approximately *invariant* to local translations of the image. An example of *pooling layers* can be seen in Figure 2.5, where the *pooling layers* are added after the *convolution layers*.

Transposed convolution layer

As explained in Section 2.4.1, the *convolution* operation usually transforms the input into lower dimensional *feature maps* . *Convolutional autoencoders* usually need the transformation going in the opposite direction, i.e., from something that has the shape of the output of some convolution to something that has the the shape of its input while maintaining the same connectivity pattern. This operation is referred to as a *transposed convolution*[3] and it is an essential part of the *decoding* layer of autoencoders. Just like the standard convolutional layer, the *transposed* convolutional layer is also defined by the *padding* and *stride* hyperparameters. However, in *transposed* convolutions the

[3]The *transposed convolution* is often called *deconvolution*. But because it is not actually performing the reverse effect of a convolution, which is meant by the mathematical term of a deconvolution, it is strongly discouraged to name it so. An alternative name that is also often used is *upconvolution*.

forward and backward passes are swapped. This *swap* can be achieved by transposing the convolution matrix on the output feature map convolution, which will generate the spatial dimensions same as that of the input feature map.

2.4.2 Advantages

As mentioned above, *discrete convolutions* are the reason why CNNs are so popular for learning data such as images, sound clips etc, as they are able to take advantage of the implicit structure of such data and exploit their topological information. Other neural networks apply *affine transformations*[4] over the data, whereas a *discrete convolution* is a *linear* transformation, which leverages three important ideas that can help improve a machine learning system. These are: sparse interactions, parameter sharing and equivariant representations [GBC16].

Sparse interactions

Convolutional networks, typically have sparse interactions. This is accomplished by making the kernel smaller than the input. When only a few input unit contribute to a given output unit, we can detect small, meaningful features with kernels that occupy far less elements than the original input. Therefore, we need to store fewer parameters, which in turn reduces the memory requirements of the model and improves its statistical efficiency (see Figure 2.8). Furthermore, units in the deeper layers may indirectly interact with a larger portion of the input. This allows the network to efficiently describe complicated interactions between many variables by constructing such interactions from simple building blocks that each describe only sparse interactions.

Parameter sharing

In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer. In a convolutional neural network the same parameter can be used for more than one function, see Figure 2.8. This is referred to as *parameter sharing*[5]. The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for each location we learn only one set, reducing the storage requirements for the model's parameters [GBC16]. Therefore, convolution is dramatically more efficient than *affine* transformations in terms of the memory requirements and statistical efficiency.

Equivariant representations

In the case of convolution, the particular form of parameter sharing causes the layer to have a property called *equivariance* to translation. To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$. Because the kernel and its parameters are reused at every position, the model learns the same representations at every position [GBC16]. This is useful when processing time-series data, as the convolution produces a sort of timeline that shows when different features appear in the input. Similarly with images, the feature map contains where certain features appear in the input, and therefore by moving these features in the input, its representations will move the same amount in the output. This property is very useful since its practical to share parameters of

[4] An affine transformation receives a vector as input that is multiplied with a matrix to produce an output, to which a bias vector is usually added.

[5] The value of the weight applied to one input is tied to the value of a weight applied elsewhere, thereby we can also use the term *tied weights* instead of *parameter sharing*

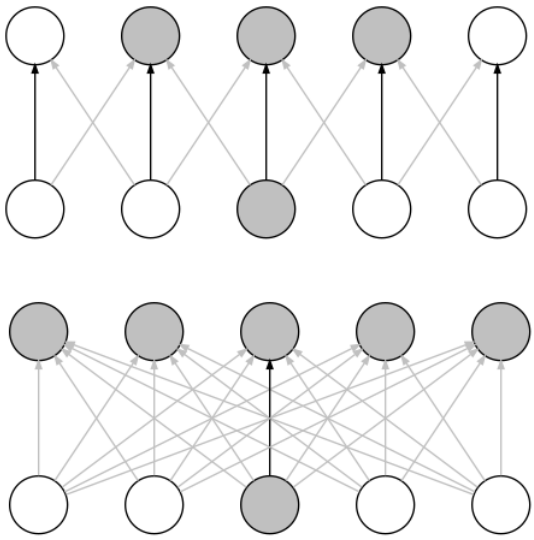


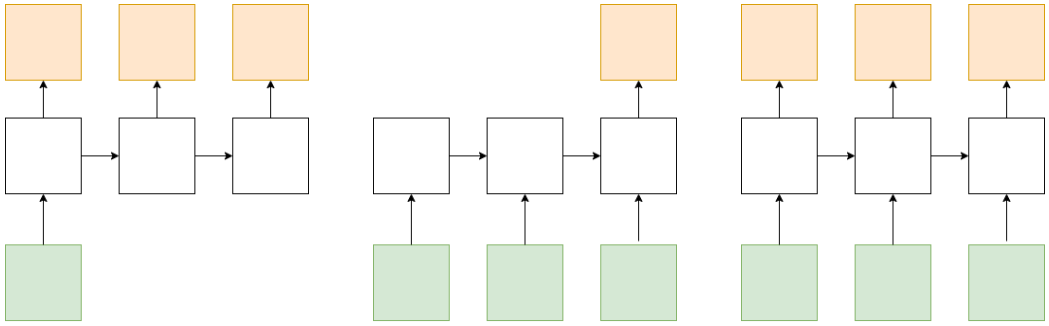
Figure 2.8: Graphical representations of parameter sharing and sparse connections. Black arrows indicate the connections that use a particular parameter in two different models. *(Top)* The black arrows indicate that the central element of a kernel in a convolutional model is used at all input locations (parameter sharing). *(Bottom)re* The fully connected model has no parameter sharing and the central element of the weight matrix is only used once. Furthermore, the highlighted input units denote that they affect all the highlighted output units. *(Top)* When the output is formed by convolution with a kernel of width 3, only three outputs are affected by the input unit (sparse connections). *(Bottom)* When the output is formed by matrix multiplication, all the outputs are affected by the input, therefore connectivity is no longer sparse. (Based on [Zha+19a])

features that appear more or less everywhere in the image (such as edges). Unfortunately, convolutions are translation equivariant and not invariant, but as discussed before translation invariance can be achieved by combining convolution layers with pooling layers.

2.5 Recurrent Neural Networks

A glaring limitation of the neural networks described until now is that their memory is kind of static and their predictions are mostly based on the current inputs only. In this thesis it's essential to take advantage of the data's spatial properties as well as its temporal properties. On the previous section CNNs were explored as a way to efficiently process spatial information. On this section the *Recurrent Neural Networks*, or RNNs will be introduced, which were specialized to better handle sequential data. Sequential data can be sequences in the input, the output, or in the most cases both, see Figure 2.9. RNNs make use of a looping mechanism, allowing them to store information by transferring it from a one step of the back propagation to the next. Furthermore, unlike feed-forward and convolution networks, RNNs can analyze on sequences of arbitrary lengths, rather than on fixed-sized inputs. This property allows them to be extremely useful on tasks such as *natural language processing*, *timeseries prediction*, *image captioning* and more. In the last few years, there have been incredible success applying RNNs to a variety of problems [Gro17].

In this section, an overview about their structure is given, as well as the main problems they are facing, and the solutions used to solve them.



(a) One to Many (b) Many to One (c) Many to Many

Figure 2.9: Graphical examples of different recurrent networks input-output modes. The green squares represent the inputs, the white squares the recurrent cells and the red squares represent the outputs. In (a) we have a fixed-size input to a sequence output, an example of such mode would be an image captioning application, where it takes an image and outputs a sequence of words. Further, in (b) we can see that the sequence is now in the input, an example of that could be a sentiment analysis application where a given sentence is classified as an expression of positive or negative sentiment. And lastly in (c) the network takes a sequence as an input and produces a sequence as an output. An example of that could be a video classification application where we would label each frame of the video (Based on [Kar15])

2.5.1 Structure

As mentioned above, RNNs extend the functionality of feed-forward networks to also take into account previous inputs $X_{0:t-1}$ and not only the current input X_t. An example of a layer of recurrent neurons unrolled through time[6] can been seen in Figure 2.10, where at each time step t, every neuron receives both the input vector X_t and the output vector from the previous time step H_{t-1}. This process of passing information from the previous iteration to the hidden layer can be mathematically formulated with the notation proposed in [Zha+19a] . For that, we denote the hidden layer block H as the aggregation of the hidden layers of the network. A hidden variable can be calculated as:

$$H_t = \varphi_h(X_t W_{xh} + H_{t-1} W_{hh} + b_h) \tag{2.27}$$

, where we denote the hidden state and the input at time step t respectively as $H_t \in \mathbb{R}^{n \times h}$ and $X_t \in \mathbb{R}^{n \times d}$, whereas n is the number of samples, d is the number of inputs of each sample and h is the number of hidden units. Furthermore, $W_{xh} \in \mathbb{R}^{d \times h}$ denotes the weight matrix, while $W_{hh} \in \mathbb{R}^{h \times h}$ denotes the hidden-state-to-hidden-state matrix. Lastly, $b_h \in \mathbb{R}^{1 \times h}$ is the bias parameter and φ_h notates the activation function, which is usually a *ReLU* or a *tanh* function[7] For a times step t, the output variable can be computed similarly to the one of a vanilla MLP:

$$O_t = \varphi_0(H_t W_{ho} + b_o) \tag{2.28}$$

Since the output O_t includes H_t, which is calculated by recursively using H_{t-1}, the RNN includes information of all hidden states. Therefore, an RNN has a form of *memory*, and the part that preserves the information from the previous time steps is called a *memory cell*. A layer of recurrent neurons, is a very basic memory cell, but in the Section 2.5.2 a more powerful type of cell will be explored.

[6] A network unrolled through time is the representation of the network against the time axis.
[7] Although many researchers prefer to use the *tanh* function rather than the *ReLU* one [Gro17].

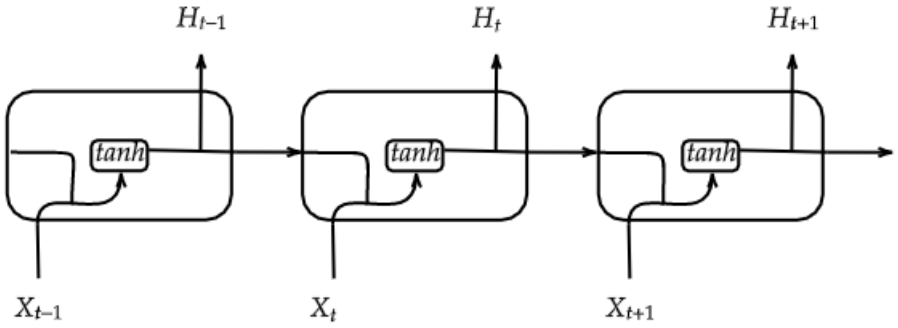


Figure 2.10: Structure of a RNN cell unrolled through time. (Based on [Ola15])

Backpropagation Through Time (BPTT)

Training an RNN, would require to properly backpropagate the errors through the network. To accomplish that, a technique called Backpropagation Through Time (BPTT) is used, which is a specific application of the back propagation algorithm in RNNs [Wer90]. Conceptually, BPTT works by unrolling the network one time step at a time in order to obtain the dependencies among model variables and parameters. After, backpropagation is applied to compute and store gradients. Therefore to update the weights, the gradient of a loss function $E(w)$ at a particular time step t depends not only on the input but also on the gradients of previous states at all the previous time steps. The total loss for a given sequence of input values paired with a sequence of output values would be the sum of the losses over all the time steps. Hence we can formulate the gradient calculation as:

$$\frac{\partial E}{\partial W_{hh}} = \sum_{t=1}^{T} \frac{\partial e_t}{\partial O_t} \cdot \frac{\partial O_t}{\partial \varphi_o} \cdot W_{ho} \sum_{k=1}^{t} \frac{\partial H_t}{\partial H_k} \cdot \frac{\partial H_k}{\partial W_{hh}} \tag{2.29}$$

$$\frac{\partial E}{\partial W_{xh}} = \sum_{t=1}^{T} \frac{\partial e_t}{\partial O_t} \cdot \frac{\partial O_t}{\partial \varphi_o} \cdot W_{ho} \sum_{k=1}^{t} \frac{\partial H_t}{\partial H_k} \cdot \frac{\partial H_k}{\partial W_{xh}} \tag{2.30}$$

, where e_t is a loss term at timestep t of the overall loss function E:

$$E(O, Y) = \sum_{t=1}^{T} e_t(O_t, Y_t) \tag{2.31}$$

Since the number of derivatives required for a single weight update is depending on the number of timesteps of the input sequences, BPTT can be very computationally expensive as that number increases. One way to solve this, is by using a modified version of the BPTT algorithm, called *Truncated Backpropagation Through time*, or TBPTT. In this modified BPTT, the algorithm establishes an upper bound for the number of times the gradient can flow back to [Sch19]. This way, each sequence is processed one timestep at a time and *periodically* (according to the upper bound) the BPTT update is performed backwards, reducing the cost of parameters' updates.

Drawbacks

As in most neural networks, vanishing or exploding gradients is a core issue in RNNs. As we can see from Equation 2.29 and Equation 2.30, each time step's gradient is calculated with the respect to the effects of the gradients in the time step before it. If there are small adjustments to the time steps before, then adjustments to the current time step will be even smaller, causing the gradients

to exponentially shrink as it back propagates down and finally vanish. This basically stops the contribution of states that happened far earlier than the current time step towards the current time step [Sch19]. After a while, the RNN's state contains virtually no trace of the first inputs, making it unable to learn long-term dependencies. Similarly, big adjustments to the time steps before, can cause the gradient to explode, which results in values that weight too much making the model's learning too noisy. In theory, RNNs are capable of learning data with long-term dependencies, but in practice they fail to do so [Ola15]. This problem has been extensively explored in [BSF94].

2.5.2 Long Short-Term Memory

Long Short Term Memory networks, or LSTMS are a special variant of RNN, which were explicitly designed to solve the long-term dependency problem. They were introduced by Hochreiter and Schmidhuber (1997) [HS97], and were refined and popularized by many people in following work [Gra+06] [Arr+19]. Over the years, LSTM networks have worked tremendously well on a large variaty of problems, and are now widely used.

The core idea is that the network is able to decide what information to store in the long-term state, what to throw away, and what to read from it. This is accomplished by using a *memory cell state* $C^{(t)}$. which is updated using at attentive gating mechanism. At each time step, the cell state is controlled by three trainable *gates*. The *forget gate* F_t is used to decide what information to throw away from the cell state, the *input gate* I_t is used to decide what information to store in the cell state and update it, and lastly the *output gate* O_t is used to decide what parts of the cell state to output. It's important to note, that these gates interact *linearly* with the cell state, simplifying the gradient flow backwards through time, which in result prevents the vanishing gradient problem vanilla RNNs faced. The gates F_t, I_t and O_t can be computed using the following equations:

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f) \tag{2.32}$$

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i) \tag{2.33}$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o) \tag{2.34}$$

, where $W_{xi}, W_{xf}, W_{xo} \in \mathbb{R}^{d \times x}$ and $W_{hi}, W_{hf}, W_{ho} \in \mathbb{R}^{h \times h}$ are the weight matrices, while $b_i, b_f, b_o \in \mathbb{R}^{1 \times h}$ are their respective biases. The sigmoid function σ is used to transform the output $\in (0, 1)$ to a vector with entries $\in (0, 1)$.

Furthermore, the input gate I_t computes a vector of new candidate values $\tilde{C}_t$ using a *tanh* activation function, which results to having an output $\in (-1, 1)$. The new candidate cell has its own weight matrices $W_{xc} \in \mathbb{R}^{d \times h}$, $W_{hc} \in \mathbb{R}^{h \times h}$ and biases $b_c \in \mathbb{R}^{1 \times h}$. This can be mathematically formulated as:

$$\tilde{C}_t = tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c) \tag{2.35}$$

Putting things together, to update the old cell state $C_{t-1} \in \mathbb{R}^{nxh}$ into the new cell state C_t we calculate:

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t \tag{2.36}$$

, where $\odot$ denotes the Hadamard product[8]. Lastly to compute the hidden states $H_t \in R^{n \times h}$ the cell state C_t is put through a *tanh* function and multiplied by O_t:

$$H_t = O_t \odot tanh(C_t) \tag{2.37}$$

[8]The Hadamard product computes the element-wise product of two matrices with the same dimension. Also known as Schur product.

A graphical representation the LSTM's mechanism and further explanation can been seen in Figure 2.11.

In short, an LSTM cell can learn to recognize an important input, store it in the long-term as long as it is needed, and learn to extract it whenever is needed. This is the reason why LSTMs networks are capable of modeling long-term sequential dependencies in data such as time series, texts, audio, and more. However its important to consider that their multiple memory cells increase the training's computational complexity and memory requirements.

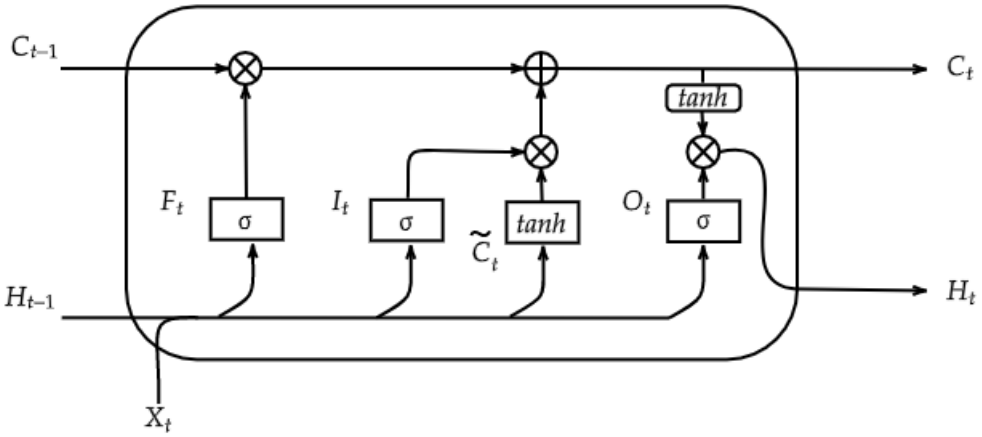


Figure 2.11: Structure of a LSTM cell. The symbol $\bigotimes$ notates the element-wise multiplication, while $\bigoplus$ notates the element-wise addition. Firstly, the forget gate F_t looks at the previous hidden state H_{t-1} and input X_t and outputs a number between 0 and 1, indicating how much of the information it should keep. Next, the input gate I_t decides which values to update, while the *tanh* function creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state C_t. The new state C_t is calculated using the Equation 2.36. which works by forgetting the information F_t decided and adding the new candidate values, scaled by how much we decided to update each state value. Following the values we want to output are calculated through the output gate O_t, and lastly the new hidden state H_t is calculated from the values of the cell state C_t with pushed values between -1 and 1 multiplied by the output of O_t. (Based on [Zha+19a])

2.6 Auto-encoders

An auto-encoder is a neural network that is trained to reconstruct its input in a *self-supervised* learning technique [GBC16]. Usually an auto-encoder network is composed of two parts. An *encoder* and a *decoder* part[9]. In the encoder part, the auto-encoder takes input $x \in \mathbb{R}^d$ and maps it to a representation $h \in \mathbb{R}^{d'}$ using a deterministic function of the type $h = f(x)$. Following, the decoder part produces a reconstruction $r = g(h)$. Due to the analogy to encoding and decoding, the learned representation is also referred to as *code*. An auto-encoder typically has a similar architecture of that of a MLP network, except that the number of neurons in the output layer must be equal to the number of inputs. Lastly, it's important to note that autoencoders are specifically designed to be unable to learn to copy the input perfectly, since learning to set $g(f(x)) = x$ isn't very useful. Therefore, they are restricted in ways that forces them to learn useful properties of the data in order to *approximately* reconstruct the desired output [GBC16].

[9]The encoder and decoder part are also called *recognition network* and *generative network* respectively.

Auto-encoders have been extensively researched, and have been successfully implemented over the years in applications such as dimensionality reduction, feature learning and more. However, in the recent years auto-encoders have received increasing attention because of their relations with *generative models* [GBC16]. This means that auto-encoders are able to generate new data that is similar to the input data.

2.6.1 Convolutional auto-encoders

Fully connected auto-encoders ignore the spatial properties of data such as images. Spatial locality can be preserved by using convolution layers (parameter sharing) as explored in Section 2.4.2. Convolution auto-encoders (CAEs) make use of this property by using convolution layers throughout the network [Mas+11]. Just like the fully connected auto-encoders they are compromised of a an encoder and a decoder part, however in CAEs the encoder part consists of convolution and max pooling layers to down sample the input data and extract valuable features, while the decoder part consists of transposed convolution layers that are used to reconstruct an image of the same spatial dimensions as the input using the extracted feature maps. Transposed convolution layers were explained in Section 2.4.1. A visual representation of the architecture of a simplified CAE can been seen in Figure 2.12. Usually, CAEs are trained end-to-end to learn filters able to extract features that minimize reconstruction errors in image reconstruction tasks. They have been successfully implemented on many image-to-image translations tasks such as image segmentation, image denoising and more.

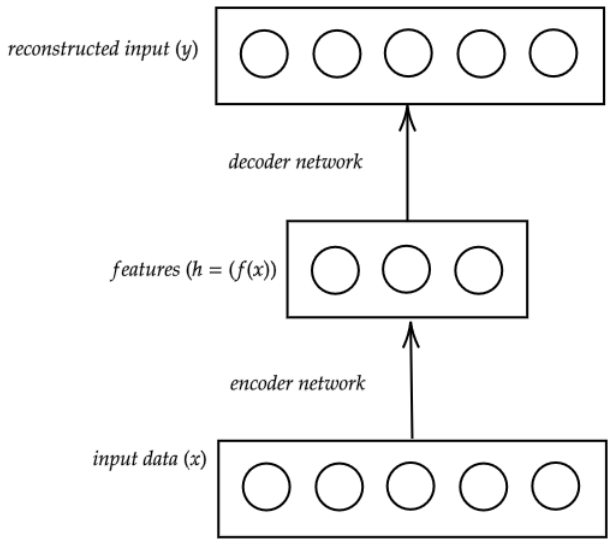


Figure 2.12: An example of a convolution auto-encoder. In this network the original input x is down sampled into a lower dimensions representation h with the *encoder network*, and after the original input is reconstructed using the *decoder network*. An application of such network could be inputting images with noise and outputting the images without noise (denoising application).(Based on [Kar18])

2.7 Batch Normalization

Training neural networks with many hidden layers can be very challenging. One reason for this difficulty is that the distribution of the network's activations changes due to the changes in network parameters during training. This can cause the learning algorithm to forever chase a moving target.

This phenomenon has been defined as *Internal Covariate Shift* [IS15]. Internal covariate shift requires careful parameter initialization and lowering learning rates which slows down the training process or even halts it in models with saturating nonlinearities.

One modern technique that tries to solve this problem is called *batch normalization* [IS15]. It helps coordinate the update of multiple layers in the model by scaling the output of the layer. Specifically it standardizes the activations of each input variable per mini-batch to have a mean of zero and a standard deviation of one. In effect, it compensates the covariate shift between two layers of the network. Batch normalization can be mathematically formulated as:

$$\tilde{Z}^{(i)} = \beta + \gamma * \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 - \varepsilon}} \tag{2.38}$$

, where $\tilde{Z}^{(i)} \in \mathbb{R}^d$ is layer's output, while $Z^{(i)} \in \mathbb{R}^d$ is the output of the previous layer to be normalized. The notations $\mu = \frac{1}{n}\sum_i Z^{(i)}$ and $\sigma = \frac{1}{n}\sum_i (Z^{(i)} - \mu)$ are the mean and standard deviation respectively. The two trainable parameters $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ allow the model to choose the optimum distribution for each hidden layer, whereas $\varepsilon \in \mathbb{R}$ serves as a regularization parameter used for numerical stability [IS15].

Batch normalization can be implemented during training in two different modes. One mode is using a running average of mean and standard deviation and maintaining them across the mini batches. However, this has been observer to lead in unstable training [Iof17]. Alternately, the mean and standard deviation of each input variable per mini-batch can be calculated and use these statistics to perform the standardization. This is the mode used throughout this work.

In conclusion, *batch normalization* is an important method to employ during training since it offers us many advantages. The researchers in [IS15] showed that *batch normalization* not only speeds up training, since it allows use of higher learning rates, but even achieves a higher accuracy compared to the same versions of the networks with no batch normalization.

Chapter 3

Related work

As mentioned above, understanding the COVID-19 responses impact on air quality is of great importance and much research has been dedicated to examine that exact impact since the start of the pandemic. This chapter presents the related work of researchers that associate COVID-19 policy responses and air quality using machine learning based methods. In addition it also discusses existing deep learning approaches that have addressed the issue of air quality prediction and remote sensing imagery frame prediction in general.

3.1 Examining COVID-19 and air pollution link

The COVID-19 pandemic has caused disastrous health and socioeconomic crises across the globe and many questions have been raised about the interconnection between the pandemic and air quality.

3.1.1 COVID-19 mortality and air pollution link

Recent studies of COVID-19 in several countries identified links between air pollution and death rates. [Tra+21] explored potential links between major fossil fuel-related air pollutants and SARS-CoV-2 mortality in England. The authors collected air quality data and COVID-19 cases and related deaths data to analyze their association using generalised statistical models. They concluded that a small increase in air pollution leads to a large increase in the COVID-19 infectivity and mortality rate in England. In another study, [MMS21] employed deep learning models to explore the relationship between pollution and the number of deaths from COVID-19 in New York state. Data on confirmed deaths (total and daily) due to COVID-19 were collected by NYC health and air pollutant data ($PM_{2.5}$ and NO_2) was compiled from the AQIGN database. Their results confirmed that pollutants such as $PM_{2.5}$ and NO_2 accelerated COVID-19 deaths.

3.1.2 COVID-19 response policies and air pollution link

On the other side, city restrictions that were implemented to fight the pandemic have caused significant changes in air quality. Recent data released by NASA (National Aeronautics and Space Administration) and ESA (European Space Agency) indicates that nitrogen dioxide (NO_2) in some of the epicenters of COVID-19 such as Wuhan, Italy, Spain and USA etc. has reduced up to 30% [MLS20]. This sudden change could have had a positive impact on human health where air pollution is a major concern. In [CEL20] they calculated that the reduction of NO_2 concentrations could have prevented as many as 496 deaths in Wuhan city, 3,368 deaths in Hubei province and 10,822 deaths in China as a whole. Additionally, they quantified the impact of Wuhan COVID-19 lockdown on concentrations of four air pollutants: sulphur dioxide (SO_2); nitrogen dioxide (NO_2);

carbon monoxide (CO_2); and particulate matter (PM_{10}). In order to isolate the effect of the policy interventions on pollutant concentrations the authors cleverly implemented machine learning to remove the confounding effects of weather conditions on them. Further, they use a (ridge) augmented synthetic control method on daily weather normalised pollution numbers of thirty cities to estimate how concentrations levels in Wuhan have changed relative to the synthetic control. The city-level hourly concentrations of the four pollutants were collected from 'Qingyue Open Environmental Data Center' between January 2013 to February 2020. Summarizing their results, the authors found that Wuhan experienced a significant reduction in concentrations of NO_2 and PM_{10} as a results of the COVID-19 lockdown. Concentrations of NO_2 fell by as much as 24 $\mu g/m^3$ during their analysis period in January/February 2020 (a reduction of 63% from the pre-lockdown level of 38 $\mu g/m^3$), while PM_{10} fell by approximately 24 $\mu g/m^3$ (a reduction of 35% from the pre-lockdown level of 62 $\mu g/m^3$). The authors found no significant reductions in concentrations of SO_2 or CO.

A different strategy is applied by [Wan+21] that includes using the Community Multi-scale Air Quality (CMAQ) model to simulate air pollutants before, during and after the the COVID-19 outbreak in the PRD. Therefore, January 5 to 22 was Period I before the pandemic, Period II during the pandemic was from January 23 to February 19, and February 20 to March 9 was Period III after the pandemic. The authors used numerical hourly data of six key pollutants: O_3,CO,NO_2,SO_2 and $PM_{2.5}$, which were obtained from the China National Environmental Monitoring Center. Their results showed that during Period II air quality improved significantly with $PM_{2.5}$, NO_2, and SO_2 decreased by 52%, 67%, and 25%, respectively, while O_3 had no obvious changes in most cities, which mainly was due to the synergetic effects of emissions and meteorology. Lastly they point out that in Period III, the increase of secondary components was faster than that of primary $PM_{2.5}$(PPM), which indicated that changes in PPM concentration were more sensitive to emissions reduction.

In [RZ21] the authors proposed an alternative approach to analyze the effect of the COVID-19 lockdown on air pollutants concentrations, which is based around machine learning modeling. A Gradient Boosting Machine (GBM) algorithm was used to build 24 models (one for each city area) to predict NO_2,CO,$PM_{2.5}$ and SO_2 numerical concentrations in Quito,Ecuador as if COVID-19 quarantine never existed. The authors processed four years and six months of atmospheric pollution and meteorological data on the period prior and during COVID-19 quarantine. The evaluation of their models showed a high predictive performance for NO_2 and CO, a good one for SO_2 and fair for $PM_{2.5}$, varying for different districts of their city. The quantification of the concentration changes due to the reduced human activity, was obtained by comparing the values provided by the models to the real measurements. They observed a clear reduction for NO_2 ($-53\% \pm 2\%$), SO_2 ($-45\% \pm 11\%$), and CO ($-30\% \pm 13\%$) concentrations during the full lockdown. The reduction in $PM_{2.5}$ concentrations displayed some variability depending on the type of monitoring site (traffic site: $-21\% \pm 3\%$, industrial site: $-12\% \pm 11\%$). The authors conclude by highlighting the reliability of their predictions,which supports the idea than an ML-based modeling is a robust method to quantify the impact of any event on air pollution.

Similarly, [Lov+21] utilized machine learning to analyze local air quality improvements during the COVID-19 lockdown in Graz, Austria. The authors collected six years of data (2014-2019) from five measurements sites in Graz. Numerical concentrations of NO_2, PM_{10},and O_3 were set as target variables for the models, while relative humidity, air pressure, air temparature, precipitation, wind direction, and wind speed data were set as predictive variables. Unlike [RZ21] they did not include the lag-values of the respective (predicted) pollutant as well as no other pollutant concentrations as predictors in the respective models. The machine algorithm used in this study was Random Forest Regression (RF). Further, they compared the predicted pollutant concentrations to the measured (true) values to explore the impact of the restrictions on air quality. Their findings indicated that the city's average concentration reductions for the lockdown period were:

-36.9% to -41.6% for NO_2 and -6.6% to -14.2% for PM_{10}. However they also estimated an increase of 11.6% to 33.8% for O_3, which they attribute to the inverse relationship of O_3 with NO_x concentrations. The authors state that the reduction in pollutant concentrations, especially NO_2 can be explained by the significant drops in traffic-flows during the lockdown period (-51.6% to -43.9%) in the area of Graz. Since their prediction models showed good generalization, the authors, much like [RZ21], argued that machine learning is a suitable tool to analyze pollution changes during events where atmospheric emissions change rapidly and conclude by stating that although machine learning showed similar results to the observational methods, it enabled for far more robust comparisons with the observed time series.

3.2 Spatiotemporal sequence forecasting

A main goal of this study is to use a data-driven machine learning model to predict air pollutants over an area. Machine learning modeling has been used extensively in the field of remote sensing data prediction and timeseries forecasting in general.

3.2.1 Air quality prediction

Recent advancements in sensor technology and big data provided the scientific community with continuously expanding resources of data collection. Numerical readings of air quality monitoring stations have been widely used to accurately predict air pollutants concentrations.

[Zhe+15] forecast the reading of an air quality monitoring station using a data-driven method that considered meteorological data, weather forecasts, and air quality data of the station and other stations within a few hundred kilometers. Their predictive model compromised of four major components-i.e., a linear regression-based temporal predictor to model the local factors of air quality, a neural network-based spatial predictor to model global factors, a dynamic aggregator combining the predictions of the spatial and temporal predictors according to meteorological data, and an inflection predictor to capture sudden changes in air quality. Their method achieved an accuracy of 0.75 for the first 6 hours and 0.6 for the next 7-12 hours in Beijing.

However [Zhe+15] could not capture the long-term features and short-term features at the same time, and could not learn both temporal and spatial properties in one model. [Zha+19b] proposed a data driven model, called as long short-term memory - fully connected (LSTM-FC) neural network, to predict $PM_{2.5}$ contamination of a specific air quality monitoring station over 48h using historical air quality data, meteorological data, weather forecast data, and the day of the week. The LSTM-FC model could memorize long-range temporal dependence and integrate spatial information The authors evaluated their novel model on a dataset containing records of 36 air quality monitoring stations in Beijing from 2014/05/01 to 2015/04/30 and compared it with artificial networks (ANN) and long-short term memory (LSTM) models on the same dataset. Their findings showed that compared with ANN and LSTM, their proposed model gave better predictive results based on RMSE and MAE metrics.

In another study, [Lia+20] conducted a series of experiments, using datasets for three different regions to obtain the best prediction performance of five different machine learning methods—i.e.Adaptive boosting (AdaBoost), artificial neural networks (ANN), random forest, stacking ensemble, and support vector machines (SVM). Types of data they used as predictors involve numerical observations of six pollutants (O_3, SO_2, $PM_{2,5}$, PM_{10}, CO, and NO_2), meteorological data and time data (the day of the month, day of the week, and the hour of the day) from January 2008 to December 2018. The target variables of the models were the Air Quality Index (AQI) of the regions in three different time intervals (1-h,8-h,24-h). The authors used root mean squared error (RMSE), mean squared error (MAE) and R-squared (R^2) as performance metrics for the aforementioned methods. Their results showed that all five machine learning methods produced promising

results with AdaBoost providing the best MAE results, stacking ensemble the best RMSE results and SVM the worst results among all methods explored.

Lastly as previously discussed, [RZ21] used a Gradient Boosting Machine algorithm to predict air pollutants using the lag-values of the respective pollutants as an input to their models and [Lov+21] used a Random Forest Regression algorithm using weather, environmental and temporal variables as predictive values.

3.2.2 Remote sensing imagery forecasting

In this study we are interested in air quality prediction based on satellite imagery. Therefore it is important to review related works that make use of remote sensing imagery in their prediction models. Remote sensing imagery forecasting falls broadly into two categories: those that explicitly model time, e.g., with a recurrent neural network (RNN), and those that use a convolutional neural network (CNN) to transform input images into a desired output image, therefore the related works of this subsection will be grouped into those two categories.

Recurrent Neural Network Approach

[Shi+15] formulated precipitation nowcasting as a spatiotemporal sequence forecasting problem in which both the input and the prediction target are spatiotemporal sequences. By extending the previous discussed FC-LSTM to have convolution structures in both the input-to-state and state-to-state transitions, the author proposed the convolution LSTM (ConvLSTM) and used it to build an end-to-end trainable model for precipitation nowcasting. When evaluated on a synthetic Moving-MNIST dataset and a radar echo dataset, the proposed model consistently outperformed both the FC-LSTM and the state-of-the-art operational ROVER algorithm.

The proposed ConvLSTM model of [Shi+15] has been widely used in the field of remote sensing. Researchers in [Kim+17] used a two-stacked ConvLSTM for their proposed data-driven precipitation prediction model called DeepRain. The model was used to predict the numerical amount of rainfall from weather radar data, which was a three-dimensional and four-channel data. According to their findings DeepRain reduced RMSE by 23.0% compared to linear regression.

Researchers in [Xia+19] also used ConvLSTM as a building block for their proposed model. One of the main goals of the study was to develop a spatiotemporal deep learning model that can model and capture both the spatial and temporal dependencies of sea surface temperature (SST), and predict the next's day SST field frame accurately and holistically in a data-driven end-to-end manner. The authors conducted their experiments using a 36-year satellite-derived SST data in a subarea of the East China Sea. Their model was compared to three different models, which included LSTM, linear support vector regression (SVR) and a persistence model. For the LSTM model, two different kinds of settings were compared. One treated each pixel in an SST field as an individual sample (LSTM-1 feature) model and the other treated all the pixels in a field as a sample (LSTM-n features) model. The four models were evaluated using different statistics metrics, including RMSE, MAPE, Pearson corellation coefficient and KDE. Their results showed that the ConvLSTM-based model consistently outperformed the other three models. Moreover, the ConvLSTM-based model could directly predict the SST fields while the linear SVR model and the LSTM (feature-1) model had to make predictions pixel by pixel.

Both [Kim+17] and [Xia+19] showed promising results for the ConvLSTM model. However [Mos+20] argued that the ConvLSTM architecture underperforms when the size of images and length of sequences become higher in comparison with other models. [Mos+20] conducted experiments and tested several parameters in order to compare three different LSTM architectures for the problem of prediction in remote sensing images time series. The three models that were compared were: the Stack-LSTM, the CNN-LSTM and the ConvLSTM. The authors used a set of 158 images from the Sentinel-1 mission to train and test the models. Different lengths of train

sequences and image resolutions have been applied when evaluating the models and the authors used MAE, RMSE as well as the Structural Similarity Index SSIM as the performance metrics. Their results showed that the ConvLSTM model performed poorly when the size of images and length of sequences increased. In addition to that, because of the convolutions operations, time processing of the ConvLSTM model was significantly higher than with the other two. Furthermore, the CNN-LSTM model seemed to produce the better results even though the stack-LSTM had the lowest training loss and processing time. The authors attribute the good generalization of the CNN-LSTM to the CNN part of the model, which was able to extract important features and then the LSTM network could memorize how they were changing over time. They conclude by recommending the CNN-LSTM architecture for forecasting tasks using earth observations images time series.

Convolutional Neural Network Approach

Given the usefulness of convolutional neural networks, many researchers treated the remote sensing imagery forecasting problem as an image-to-image translation problem. A widely used model to solve such problems is the well-known encoder-decoder architecture called UNet [RFB15]. UNet was originally designed for biomedical images segmentation tasks. Unlike LSTMs, UNet has no explicit modeling of memory. It takes an input image (or multiple concatenated images) and outputs a single classification map. The architecture of the UNet consists of a contracting path to capture context and a symmetric expanding path that enables precise localization. The authors in [RFB15] included data augmentation in the architecture to teach the network the desired invariance and robustness properties, when only few training samples are available. Their results showed that UNet outperformed many state-of-the-art models on many biomedical segmentation tasks.

In [Agr+19], the authors proposed a network structure for weather forecasting that is based on UNet. The implementation of [Agr+19] aimed at classifying four different rain intensities ($<$ 0.1mm/h, $<$ 1.0mm/h, $<$ 2.5mm/h, $>$ 2.5mm/h) one hour into the future. To this end, multiple precipitation maps (of the past hour) are concatenated and used as input to the UNet architecture. Their results showed that the proposed model outperformed traditional numerical methods for short-term nowcasting predictions.

In a similar study, [Søn+20] the authors classified 512 classes instead of just four, as opposed to the model described in [Agr+19]. The authors in [Søn+20] proposed MetNet, a neural network that forecasts precipitation up to 8 hours into the future. MetNet used as input radar and satellite data and produced as an output a probabilistic precipitation map of 512 classes. The model's architecture consisted of five parts: the input layer, the spatial downsampler, the temporal encoder, the spatial aggregator and the output layer. Their results showed improvements over the traditional numerical method for short-term-nowcasting HRRR for up to 8 hours of lead time.

Similar to the studies of [Agr+19] and [Søn+20], [TSM21] proposed a UNet-based model called SmaAt-Unet, which rather than predicting just classes, it predicts exact rain intensities. SmaAt-Unet is a smaller and attentive version of a UNet architecture. Their novel model made two modifications in the original UNet architecture. Firstly they added the convolutional block attention modules (CBAM) mechanism to the encoder part in order to direct the network to pay more attention to features important for the task at hand. Secondly they used depthwise-separable convolutions (DSCs) in order to reduce the number of parameters without sacrificing performance. For comparison, the authors also trained other UNet architectures that either have none or one of the modifications previously mentioned. The models were trained using the MSE loss function between the output images and the ground truth images using a dataset of 400,000 normalized 256x256 images. In addition to MSE, the authors calculated different scores for the performance evaluation, such as Precision, Recall, Accuracy and F-1 score, critical success index (CSI), false alarm rate (FAR) and Heidke Skill Score (HSS). Their results showed that implementing each mod-

ification alone slightly decreased the performance in comparison with the original Unet. However their proposed model SmaAt-Unet which incoporated both modifications into plain UNet, resulted in a better performance than UNet combined with each of the modification alone. In conclusion, their proposed model performed on par to the UNet architecture which is way bigger than itself on a precipitation nowcasting task.

In another study, the authors in [FAM20] propose four UNet-based models for remote sensing imagery forecasting. Modern enhancement techniques such as residual connections, inception modules and asymmetric convolutions were used to extend the UNet architecture and produce four different models, each one being an extended version of the previous one. These are 3DDR-UNet, Res-3DDR-UNet, InceptionRes-3DDR-UNet and AsymmInceptionRes-3DDR-UNet. The data which was used in their paper consists of satellite images provided by the Copernicus observation program. The selected data start on 01/03/2017 and end on 13/02/2019, with an hourly temporal resolution. The dataset consists of four weather variables-i.e., Eastward current velocity, Northward current velocity, Seawater salinity and Sea surface height.Therefor each time-step of each variable was represented by a 135x135 image. The proposed models were examined under different setups, i.e. different seasons and numbers of hours ahead using MSE as the loss function and performance metric. Among those models, AsymmInceptionRes-3DDR-UNet and InceptionRes-3DDR-Unet have shown superior performance thanks to the use of parallel convolutions. However, the incorporation of asymmetric convolutions and additional parallel branches made the AsymmInceptionRes-3DDR-UNet perform slightly better than the latter, yielding the most promising results.

Chapter 4

Datasets

This chapter presents an overview of all the different datasets that were used throughout this thesis. Two different datasets are used since we are focusing on the COVID-19 lockdowns' impact on air quality. Specifically, the *Sentinel-5P* dataset is used to model air quality data, while the *Oxford Coronavirus Government Response Tracker* dataset is used to learn the impact of the lockdowns. Both of the datasets' pre-processing procedures and characteristics are explained in Sections 4.1 and 4.2, respectively. Moreover, samples of the datasets are presented, alongside graphical representations and tables of their analysis. Lastly, the generation method of the input dataset used to train and test the deep learning models is described in Section 4.3.

4.1 Sentinel-5P datasets

The Sentinels are a constellation of satellites developed by the European Space Agency (ESA) to operationalize the Copernicus[1] program, which aims at providing accurate, timely, and easily accessible information about the Earth's environment. The Sentinel missions include radar and super-spectral imaging for the land, ocean, and atmospheric monitoring. Each Sentinel mission is compromised of two satellites, providing robust datasets for the Copernicus services. The objectives of each Sentinel mission can be seen in Table 4.1.

Mission	Objective
Sentinel-1	The Sentinel-1 mission provides all-weather, day and night radar imaging for land and ocean services.
Sentinel-2	Sentinel-2's main objective is land monitoring, by providing with high-resolution optical imagery.
Sentinel-3	It's primary objective is marine observation. That include studying sea-surface topography, sea and land surface temperature, and ocean and land colour.
Sentinel-4	This mission aims to provide with continuous monitoring of the composition of the Earth's atmosphere.
Sentinel-5/5-P	The Sentinel-5/5P's objective is dedicated to air quality monitoring, by providing timely data on a multitude of trace gases and aerosols affecting air quality and climate

Table 4.1: Objectives of the Sentinels missions under the Copernicus programme.

[1] https://www.copernicus.eu/

In this thesis, we are interested in gathering air quality data from the Sentinel-5P mission. As shown in Table 4.1, this mission's main objective is to perform atmospheric measurements relating to air quality, climate forcing, ozone, and UV radiation. Its data includes concentration measurements of ozone, methane, formaldehyde, aerosol, carbon monoxide, nitrogen oxide, and sulfur dioxide. In order to measure these concentrations, the Sentinel-5 satellite uses Tropomi (TROPOspheric Monitoring Instrument) [2], which is a spectrometer able to sense ultraviolet (UV), visible (VIS), near (NIR), and short-wavelength infrared (SWIR) in the atmosphere. Tropomi takes measurements that cover an area of approximately 7 km long and 2600 km wide in a spatial resolution of 0.01 arc degrees every second.

Associated with different levels of TROPOMI processing, three different data products are produced. These are Level-0 products, Level-1B products, and Level-2 products. Each level of a data product is generated by its previous layer. Level-0 products are time-ordered, raw satellite telemetry without temporal overall, including all sensor data for atmospheric and calibration measurements. Level 1-B products are geolocated and radiometrically corrected top of the atmosphere Earth radiances in all spectral bands, while the Level-2 products are geolocated total columns of ozone, sulfur dioxide, nitrogen dioxide, carbon monoxide, formaldehyde and methane, tropospheric columns of ozone, cloud, and aerosol information. There are also two main types of processing: NRT (Near Real Time) and OFFL (Offline). For NRT processing, the products are available within 3 hours after sensing, whereas for OFFL processing, the data is available from 12 hours to 5 days after sensing, depending on the product. However, data mining of the Level-2 data products can be challenging since the data is binned by time and not by latitude and longitude. Not only that, but the time that takes for the satellite to pass over the same geographical point on the ground is 16 days. To make the data mining process more manageable, we used the Sentinel-5P's Level-3 data products that are publicly available on the Google Earth Engine's[3] Data Catalog, which are generated and maintained by the Google Earth Engine's team. Specifically the two datasets that are used in this thesis are:

- the *Sentinel-5P OFFL CO: Offline Carbon Monoxide* dataset,
- and the *Sentinel-5P OFFL NO2: Offline Nitrogen Dioxide* dataset.

Each of these datasets contain high-resolution rasters[4] of CO and NO2 concentrations respectively. More information about each dataset can be found in the Earth Engine's data catalog[5] website.

4.1.1 Data mining

In order to mine the data, the Earth Engine's Python API is used. For each dataset, the band that showed the measurements of each air pollutant's concentrations was downloaded. These bands are: the vertically integrated CO column density and the total vertical column of NO2. All of these bands have a spatial resolution of 1113.2 meters. Table 4.2 shows more information about these bands.

Furthermore, since we are interested for air quality during the lockdown periods, the dataset was filtered to be between 1/1/2020 (start of the pandemic) and 22/9/2021 (present), with a time interval of 10 days to minimize any loss of data caused by the orbit of the satellite. Moreover, the datasets' images cover the areas of: California, Delhi, Democratic Republic of Congo (DRC),

[2] http://www.tropomi.eu/

[3] Google Earth Engine is a geospatial processing service, powered by the Google Cloud Platform. https://earthengine.google.com/

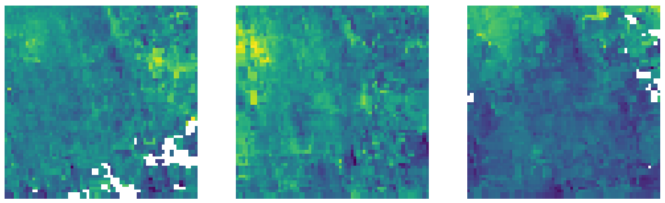
[4] Rasters are spatial data models that define space as an array of equally sized cells, arranged in rows and columns, and composed of single or multiple bands, where each cell contains a value representing information.

[5] https://developers.google.com/earth-engine/datasets/catalog/sentinel-5p

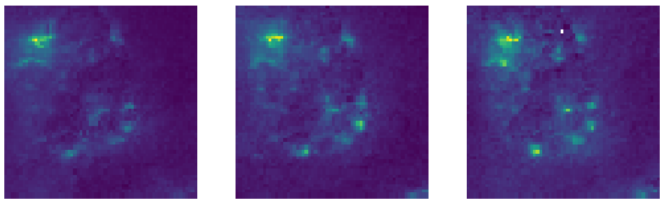
Air pollutant	Description	Units	min	max
CO	Vertically integrated CO column density.	mol/m^2	-34.43	5.71
NO2	Ratio of the slant column density of NO2 and the total air mass factor.	mol/m^2	-0.00051	0.0192

Table 4.2: Bands descriptions. The min and max values are estimated.

Greece, South Korea and United Kingdom. The areas were selected according to their pollution level and strictness of lockdown policies. In order to lower the computational cost of the models' training, all images were reshaped to dimensions of $64 \times 64 \times 1$. Samples of the three datasets can been seen in Figure 4.1. In total, 378 images were downloaded for the CO dataset and 372 for the NO2 dataset.



(a) Sample images of CO concentrations.



(b) Sample images of NO2 concentrations.

Figure 4.1: Sample images of the two datasets. The measurements are from the South Korea area. Furthermore, from left to right the creation dates of each image are: (2021/07/14 - 2021/07/24), (2021/07/24 - 2021-08-03), (2021-08-03, 2021-08-13).

As can been seen in Figure 4.1a, even with a time interval, there existed some kind of data loss. In the CO dataset 39% of its images had missing pixels, while in the NO2 dataset the number was closer to 52%, see Tables 4.3a, 4.3b. This probably was caused as a side-effect of the satellite's orbit around Earth, as aforementioned. However, even if the time interval to compose the final image is increased, the problem remains.

4.1.2 Data pre-processing and characteristics

Missing values in datasets can cause problems for machine learning algorithms; therefore, it was essential to pre-process the data and impute the missing values. The approach for data imputation for the three datasets is composed of two parts. First, *linear interpolation* is applied to estimate as many as possible unknown values. Further, any unknown value that remained to be estimated was filled using the *mean* of the known data points.

Area	Total number of images	Images with data loss
California	63	21
Delhi	63	3
Democratic Republic of Congo	63	4
Greece	63	44
South Korea	63	49
United Kingdom	63	26
Total	378	147

(a) Data loss of the CO dataset.

Area	Total number of images	Images with data loss
California	62	55
Delhi	62	28
Democratic Republic of Congo	62	30
Greece	62	15
South Korea	62	18
United Kingdom	62	47
Total	372	193

(b) Data loss of the NO2 dataset.

Table 4.3: Data loss of the two datasets. Every image that contained 1 or more *NaN* values was considered as an image with data loss.

Data imputation process

In general, interpolation is a technique used to estimate unknown data points using the knowledge of its neighboring known data points. One of the most common methods of interpolation is *linear interpolation*. Linear interpolation approximates values of some function f using two known values of that function at other points. However, since images are not one-dimensional data, a generalized form of linear interpolation called *linear barycentric interpolation* is used. Barycentric interpolation on 2D data uses only three near-neighbors data points to interpolate.

As a first step, the convex hull of the input data is triangulated. In this case, the triangulation that is constructed is the *Delaunay* triangulation. Following, each unknown data point inside the convex hull was interpolated using barycentric interpolation. Specifically, for an unknown data point x in the interior of a triangle tile formed by x_1, x_2, x_3 the following interpolation was used to approximate its value:

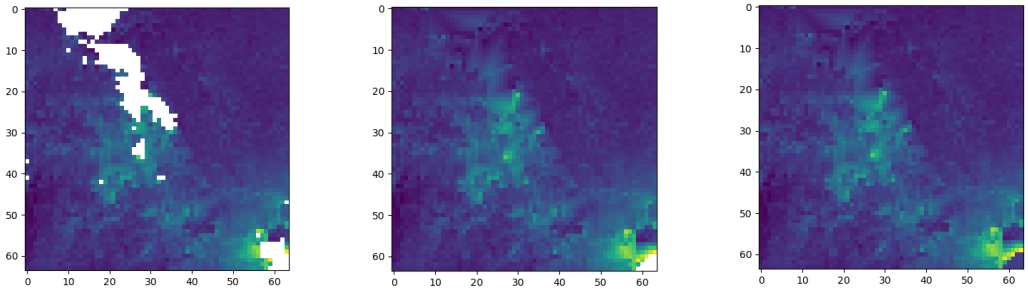
$$f(x) \approx \sum_{i=1}^{3} a_i f(x_i) \tag{4.1}$$

, where $a_i > 0$ are the barycentric coordinates of x, which are used as weights in the interpolation.

After, all unknown data points x that were outside of the convex hull of the input data, were filled using the *mean* of all known data points x_i of the image:

$$x = \frac{1}{n} \sum_{i=i}^{n} x_i \tag{4.2}$$

An example of the data imputation process can be seen in Figure 4.2. At first, the image was interpolated using linear barycentric interpolation. However, pixels at the bottom right corner were outside the convex hull of the image, and in effect, the algorithm did not estimate their values. These unknown data points were given the value of the mean of all known pixels, as calculated in Equation 4.2. Furthermore, the statistical descriptions of the Sentinel-5P CO and NO2 datasets after the data imputation are described in the Table 4.4. By looking at the table, the area of Democratic Republic of Congo has the highest *CO* concentrations, whereas Greece has the lowest. Moreover, South Korea has the highest *NO2* concentrations, that are almost double compared to the other areas.



(a) Original image

(b) Image after barycentric interpolation

(c) Image after mean interpolation

Figure 4.2: Example of the data imputation process. The image is taken from the NO2 dataset and covers the area of the United Kingdom from 2021/02/04 to 2021/02/14.

	California	Delhi	DRC	Greece	South Korea	United Kingdom	Total
mean	30.76	37.73	48.10	33.05	41.16	33.60	37.40
std	6.81	7.03	14.54	3.53	5.71	3.89	9.79
min	16.51	12.27	21.15	2.85	11.79	20.44	2.85
25%	26.71	33.99	37.42	30.46	37.79	30.95	31.46
50%	29.87	38.10	45.47	32.96	41.14	34.08	35.78
75%	33.76	41.82	55.74	35.48	44.56	36.49	41.02
max	313.3	101.2	148.4	109.24	90.48	52.65	313.3

(a) *CO* dataset. The values in the table have been multiplied with 10^3, for better presentation.

	California	Delhi	DRC	Greece	South Korea	United Kingdom	Total
mean	17.39	29.12	17.40	17.62	43.45	29.60	25.74
std	12.60	18,74	16.08	6.66	36.37	21.84	22.93
min	-46.38	-10.64	-23.48	-14.75	-30.16	-80.61	-80.61
25%	11.25	20.33	9.50	13.69	23.96	16.12	13.54
50%	14.69	26.01	12.65	16.65	33.02	24.27	19.91
75%	19.89	33.46 8	19.15	20.19	49.67	37.89	30.39
max	501.7	585.8	614.2	211.9	699.2	406.3	699.2

(b) *NO2* dataset. The values in the table have been multiplied with 10^6, for better presentation.

Table 4.4: Statistical descriptions of the two Sentinel-SP5 datasets, after data imputation. Column *Total* shows statistics for the whole dataset.

4.2 Oxford Coronavirus Government Response Tracker datasets

COVID-19 has prompted a wide range of responses from governments worldwide, creating the pressing need to track these responses to more efficiently evaluate how to address COVID-19. Oxford COVID-19 Government Response Tracker (OxCGRT) [Hal+21] provides a systematic way to track government responses to COVID-19. OxCGRT collects publicly available information on 23 indicators by a team of over one hundred Oxford University students and staff. These indicators range from spanning containment and closure policies to health system policies, such as testing regimes. In this thesis we are interested in policies that can have an effect on air quality. They can be policies that restrict movement of vehicles such as cars, buses, or even airplanes. Therefore the indicators that were selected are:

- *Close public transport indicator.*
- *Stay at home requirements indicator.*
- *Restrictions on internal movement indicator.*
- *International travel controls indicator.*

These indicators are reported for each day a policy is in place and are measured on a simple scale of severity. For example, the *close public transport* indicator has a severity scale from 0 to 2, where code 0 means no measures, whereas code 2 means that citizens are prohibited from using public transports. Furthermore, the *stay at home requirements* indicator has a severity scale from 0 to 3, where code 3 indicates that citizens are required not to leave their houses with minimal exceptions. Moreover, the restrictions on *internal movement* indicator scale from 0 to 2, with code 2 meaning that internal movement restrictions are in place. Lastly, the *international travel controls* indicator has a severity scale from 0 to 4, where code 4 means ban on all regions or total border closure. For more information about these four indicators, see Table 4.5.

The four datasets were downloaded from the OWID's (Our World In Data) platform[6], which takes data directly from the OxCGRT project. The datasets consist of areas names, areas codes, dates and the aforementioned indicators. Every dataset was filtered in two ways. Firstly it was filtered between 1/1/2020 and 22/9/2021, and after, it was filtered to contain data only for the six areas mentioned in Section 4.1.1. In addition, since the dataset contained policy indicators for the United States and not its states, the United States indicators were used for the California area. Similarly, the policy indicators for India were used for the Delhi area. Thus, each area had data of 631 values of four different indicators, which represented every day that each policy was applied. Table 4.6 gives more information about the datasets, whereas Figure 4.3 provides a cross-area comparison analysis of the different policy responses.

[6]https://ourworldindata.org/policy-responses-covid

Name	Description	Severity scale
Close public transport	Record closing of public transport	0: No measures, 1: Recommend closing (or significantly volume of transportation), 2: Require closing (or prohibit most citizens from using it.
Stay at home requirements	Record orders to "shelter-in- place" and otherwise confine to home.	0: No measures, 1: Recommend not leaving the house, 2: Require not leaving the house with exceptions for "essential" trips, 3: Require not leaving house with minimal exceptions.
Restrictions on internal movement	Record restrictions on internal movement.	0: No measures, 1: Recommend not to travel between regions/cities, 2: Internal movement restrictions in place.
International travel controls	Record restrictions on international travel.	0: No measures, 1: Screening, 2: Quarantine arrivals from high-risk regions, 3: Ban on arrivals from some regions, 4: Ban on all regions or total border closure.

Table 4.5: Information about OxCGRT's policy responses indicators. The coding instructions column shows the levels of the severity scale of each indicator and their respective descriptions. Based on [Hal+21]

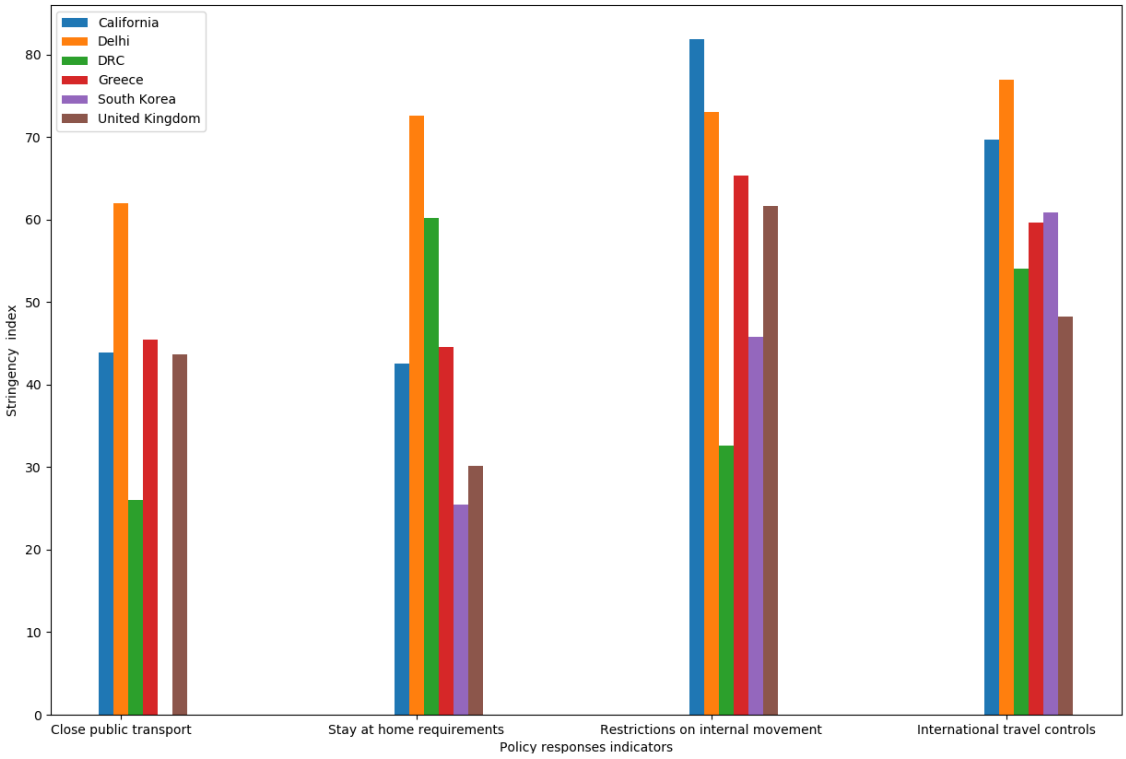


Figure 4.3: Cross-area comparative analysis of the policy responses indicators using their respective Stringency Index, see Table 4.6.

	California	Delhi	DRC	Greece	South Korea	United Kingdom
Code 0	76	181	365	73	631	79
Code 1	555	117	203	541	0	552
Code 2	0	333	63	17	0	0
Stringency index	43.9	62.0	26.0	45.5	0.0	43.7

(a) *Close public transport indicator* dataset.

	California	Delhi	DRC	Greece	South Korea	United Kingdom
Code 0	74	25	87	82	176	241
Code 1	309	56	0	276	427	210
Code 2	248	331	491	249	28	180
Code 3	0	219	53	24	0	0
Stringency index	42.5	72.6	60.2	44.6	25.5	30.1

(b) *Stay at home requirements indicator* dataset.

	California	Delhi	DRC	Greece	South Korea	United Kingdom
Code 0	73	128	334	108	176	191
Code 1	82	84	182	221	332	102
Code 2	476	419	115	302	123	338
Stringency index	81.9	73.0	32.6	65.3	45.8	61.6

(c) *Restrictions on internal movement indicator* dataset.

	California	Delhi	DRC	Greece	South Korea	United Kingdom
Code 0	32	25	62	73	34	159
Code 1	0	47	83	85	0	0
Code 2	36	2	330	0	252	199
Code 3	563	338	0	474	345	273
Code 4	0	219	156	0	0	0
Stringency index	69.7	76.9	54.1	59.6	60.9	48.2

(d) *International travel controls indicator* dataset.

Table 4.6: Descriptions of the four OxCGRT datasets. Each cell contains the number of days where a level of individual policy response indicator was applied in that area. The Stringency Index is a composite of each indicator that was calculated similarly to [Hal+21]. Specifically, each value was rescaled between 0 and 100 by the respective indicator's maximum value. The rescaled values are then averaged to get the composite indices. It's important to note that the value and purpose of this composite index are to allow for simple cross-area comparisons of the policy responses and is not used for the analysis of a specific area.

By examining the datasets' analysis, the area of Delhi has the highest Stringency Index across all the indicators, except the restrictions on internal movement. Therefore, we can assume that the highest traffic reduction was in the area of Delhi. California, Greece, and the United Kingdom had similar Stringency Indices, with the United Kingdom having the lowest one. The Democratic Republic of Congo was the second most strict area on the stay-at-home policy, although it had

lower indices on the other indicators than the countries mentioned earlier. Similarly, South Korea also had lower indices, with the close public transport indicator equal to zero.

All things considered, it should be noted that the above indicators can not truly represent the policies that were applied in each area since there is no information on how well the policies were enforced, nor they take into account significant nuance and heterogeneity these policies exhibit [Hal+21].

4.3 Input dataset generation

In this thesis, the machine learning models must detect relationships between the lockdown policies and the air pollutants. Therefore, it is essential to use both of the datasets mentioned above in the training process of the models. In order to do that, we generated an input dataset that contains the aggregated data of both the Sentinel-5P's and OxCGRT's datasets. Consequently, we generated a multi-band raster for every Sentinel-5P satellite image. Each raster has five bands; one is the air pollutant's concentrations image, and the other five are the respective policy responses indicators.

Firstly, the pixel values of the Sentinel-5P's datasets and the indicators' values of the OxCGRT's datasets were normalized between 0 and 1. Normalization of the input variables ensures that all of the datasets' values are on the same scale, as well as making training faster and reducing the chances of getting stuck on local optima [Hua+20]. Furthermore, since the policy responses indicators were provided on a daily basis, we needed to aggregate their values to a single 10-day interval value. However, since the indicators are of ordinal data type, computing the mean of its sample is considered inappropriate [Jam05], for the reason that although they have a ranked order, the intervals between their ranks cannot be presumed equal. Therefore, to aggregate the values of a policy response indicator sample $x \in \mathbb{R}^{10}$ of ten days, we use the *median*[7] value of that sample, which is calculated as:

$$median(x) = x_{(n+1)/2} \tag{4.3}$$

Further, to create the raster band for each indicator sample's median we generate a matrix $X = [median(x)_{i,j}] \in \mathbb{R}^{h \times w}$, where h is the height of the Sentinel-5P's image, and w is it's width. Lastly we generate the input dataset $S = \{Z_i\}_{i=1}^{n}$ for each air pollutant, where $Z_i \in \mathbb{R}^{w \times h \times b}$ is a raster of height h and width w with b bands/channels. Specifically, $Z_{w,h;1}$ is the air pollutant band, $Z_{w,h;2}$ is the restrictions on internal movement indicator band, $Z_{w,h;3}$is the international travel controls indicator band, $Z_{w,h;4}$ is the close public transport indicator band, and $Z_{w,h;5}$ is the stay at home requirements indicator band.

In total 378 rasters of shape $64 \times 64 \times 5$ have been generated for the *CO* dataset, and 372 rasters of the same shape have been generated for the *NO2* dataset. An example of the input's dataset is shown in Figure 4.4.

[7] The median is the middle value that separates the greater and lesser halves of a data sample.

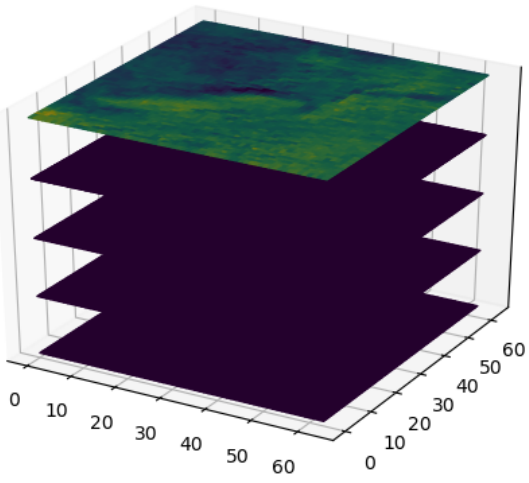


Figure 4.4: Example raster of the input's dataset. The raster covers the area of the United Kingdom between the dates of 2020/01/01 and 2020/01/11. From top to bottom, the first band is the CO's concentrations band from the Sentinel-5P's dataset, while the other four bands are the medians of the OxCGRT's policy responses indicators of that time period.

Chapter 5

Methodology

This chapter provides an overview of the problem this thesis is researching and the deep learning models that were implemented to solve it.

5.1 Problem Formulation

This research aims to predict the next image occurrence of air quality data over an area using its past five occurrences of air quality and lockdown measures data. Therefore, given a dataset, $S = \{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \in X := [0,1]^d$ and $y_i \in Y := [0,1]^{do}$, the goal is to approximate the non-linear function $f(x;\theta)$ with $f_\theta : X \to Y$ as accurately as we can. The notations d and do denote the input and output dimensions respectively, while θ denotes the set of parameters of f_θ. Specifically, let $x_i = \{x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4}, ..., x_{i_{t-1}}, x_{i_t}, x_{i_{t+1}}, x_{i_{t+2}}, ..., x_{i_{t+n}}\}$ be a time series of input rasters used for training the models, with dimensions (W, H, C). For a given sequence $z_i = \{zi_1, zi_1, ..., zi_{t-1}, z_i\}$ as input for prediction, the objective is to predict $y_i = f(z_i)$, where $z_{i_{t+1}} = y_i$. W, H and C denote the width, height and number of channels of each raster and t denotes the timestep. To achieve this, three models were implemented. These models are the CNN-LSTM, ConvLSTM and 3D-UNet model, which are all presented in the following sections.

5.2 CNN-LSTM model

We need to implement a sequence-to-one prediction model to forecast the next raster occurrence of a given time series. Therefore, it is crucial to take advantage of both the given rasters' spatial properties and the sequences' temporal properties. As aforementioned in Chapter 2, CNNs can learn the spatial properties of the data while LSTM networks can learn the temporal ones. Therefore, one approach for working with spatiotemporal data is to combine both networks by placing LSTM layers after the CNN layers. Such architecture is called Convolutional-LSTM (CNN-LSTM) and was first introduced as the Long-Term Recurrent Convolutional Network (LRCN)[Don+16]. This architecture is composed of two sub-models: the CNN model and the LSTM model. The CNN model extracts important features of the data, which are then flattened in a 1-D tensor and are then given to the LSTM model to learn the sequential properties of the data. The results of the LSTM layers are then reshaped with dimensions equal to d_o.

In our implementation, one *2DConvolutional* layer has been inserted, along with one *MaxPooling2D* layer and a *Flatten* layer transforming the extracted features to a appropriate shape for the LSTM layers. Afterwards, two *LSTM* layers are stacked and a *Dense* layer. Lastly, a *Reshape* layer is used as mentioned. The graphical representation of the implemented model can been seen in Figure 5.1.

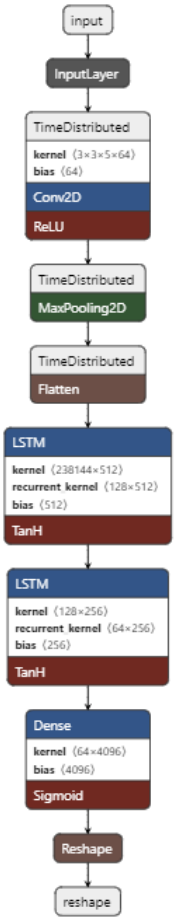


Figure 5.1: Implemented CNN-LSTM model. Since there is a single CNN model and a sequence of LSTM models, we wrap the entire CNN input model in a *TimeDistributed* layer, to apply the same CNN layer to each timestep independently.

5.3 Convolutional LSTM

Another way of working with spatiotemporal data is to use the model proposed in [Shi+15]. The researchers merged the functionalities of the CNN and LSTM architecures, to create a LSTM-variant cell called the Convolutional LSTM (ConvLSTM) cell. The code idea of the ConvLSTM cell is to handle all inputs, hidden states, cell or gate outputs as 3D tensors, in order to preserver their spatial properties. This is achieved by replacing the matrix multiplication operation used in the vanilla LSTM (see Figure 2.11) with the convolution operation in the input-to-state and state-to-state transitions, as shown in Figure 5.2.

By using the latter modification and the Equations 2.32 to 2.37, we can formulate the input,states, gates and output of the ConvLSTM as:

$$F_t = \sigma(X_t * W_{xf} + H_{t-1} * W_{hf} + b_f) \tag{5.1}$$

$$I_t = \sigma(X_t * W_{xi} + H_{t-1} * W_{hi} + b_i) \tag{5.2}$$

$$O_t = \sigma(X_t * W_{xo} + H_{t-1} * W_{ho} + b_o) \tag{5.3}$$

$$\tilde{C}_t = tanh(X_t * W_{xc} + H_{t-1} * W_{hc} + b_c) \tag{5.4}$$

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t \tag{5.5}$$

$$H_t = O_t \odot tanh(C_t) \tag{5.6}$$

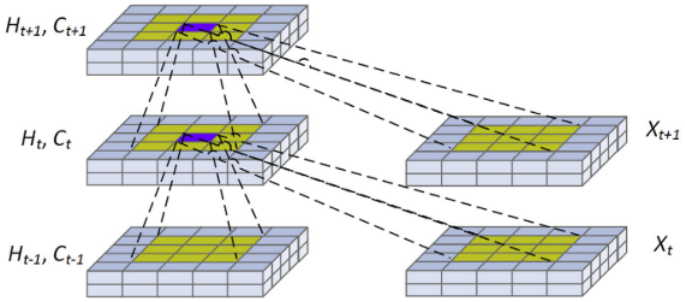


Figure 5.2: Inner structure of ConvLSTM. (From [Xia+19].)

Regarding the implemented ConvLSTM model, only three *ConvLSTM* layers have been stacked, associated with *BatchNormalization* for the reasons mentioned in Chapter 2.7. Furthermore, we add *Dropout* in each*ConvLSTM* layer to avoid the phenomenon of overfitting. Lastly, we put a *2DConvolutional* layer to reshape the output data to the appropriate dimensions. The graphical representation of the ConvLSTM model is shown in Figure 5.3.

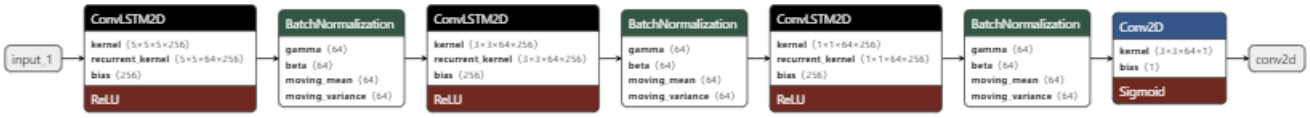


Figure 5.3: Implemented ConvLSTM model.

5.4 3D-UNet

The problem of this thesis can be also treated as a future frame prediction problem, where each frame is a raster. In this context, another approach to consider is the use of: *Auto-encoders* (See Section 2.6). As discussed in Chapter 2, auto-encoders have been successfully implemented for such problems [Liu+18] [FAM20], thereby its important to implement them to our problem.

As mentioned in Chapter 3, one of the most popular auto-encoders is the UNet architecture, which is also used as our core model. Contrary to the vanilla UNet, we are using *3DConvolutional* layers to be able to feed the network with sequences of multi band rasters. Specifically, our 3D-UNet model is based on the 3DDR-UNet model from [FAM20]. The 3DDR-UNet architecture is able to capture both spatial and temporal dependencies by manipulating 3-dimensional data in the encoder part and 2-dimensional data in the decoder, see Figure 5.4. Afterwards, the time dimension is reduced from t (number of timesteps) to 1 before the expansion part of the decoder. This configuration allows the network to extract features on the encoder part and averaging them in a weighted fashion over the time dimension. Similarly to the vanilla UNet, the number of convolutional filters

grow exponentially in the encoder part, and shrink again in the decoder part. By giving both of the pooling operations and *deconvolutions* kernel sizes of $1 \times 2 \times 2$ the temporal dimension of the data remains unchanged. In summary, the network is trained to perform a regression for every pixel to accurately map the set of input rasters to the output.

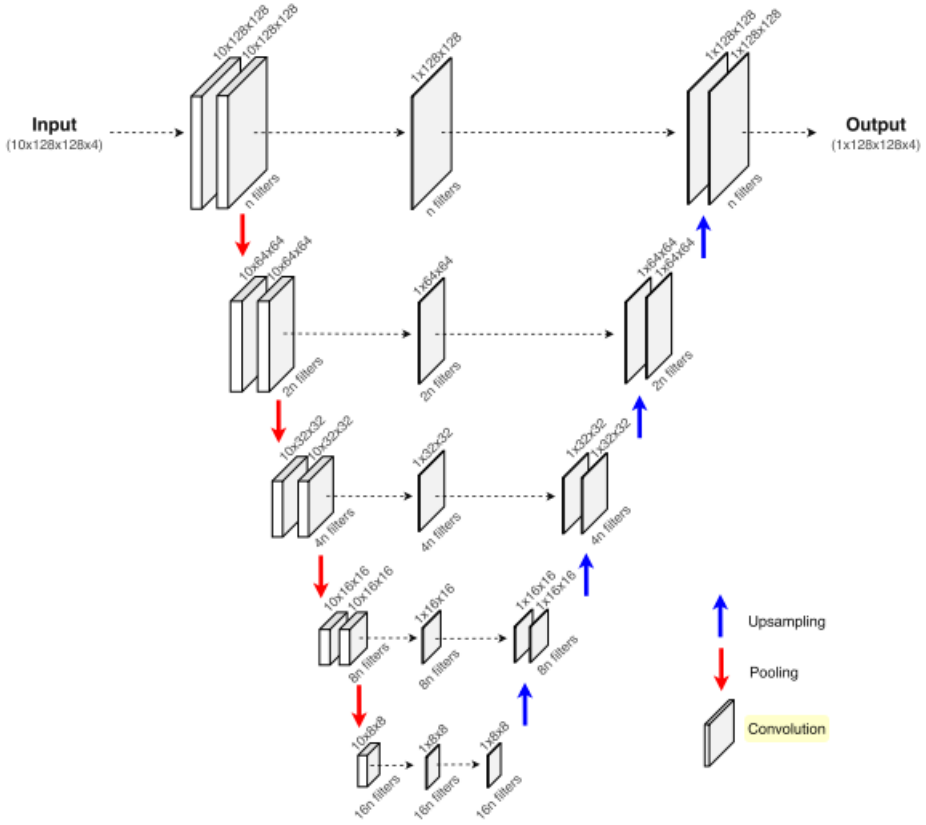


Figure 5.4: Graphical representation of the 3DDR-UNet model' architecture. The annotations above the convolutions/deconvolutions represent the output shape of those layers, whereas the number of filters is specified below them. (From [FAM20].)

Chapter 6

Experiments

This chapter presents the experiments conducted in this Thesis. First, Section 6.1 describes the experimental setup, including the dataset generation for the training and testing of the models, as well as a brief introduction of the evaluation metrics that are used.

6.1 Experimental Setup

All of the experiments were done using the *Python 3*[1] programming language. Furthermore, for the training, testing, and implementation of the deep learning models, the *Keras API*[2] of the *TensorFlow 2*[3] framework was used. For more information about the hardware and software specifications, see Table 6.1a and Table 6.1b .

CPU Model Name	Intel Xeon Broadwell
CPU Freq	2.6 GHz
CPU Type	Dedicated
No. vCPUs	4
RAM	8GB

(a) Hardware specifications.

	Version
Python	3.7.4
TensorFlow	2.4.0
Keras	2.3.1

(b) Software specifications.

Table 6.1: System specifications

6.1.1 Training and testing dataset

In order, to use the input dataset for the training and testing of the model, it was split in the correct form by generating the X and Y arrays keeping their chronological order. Each sample of X is a sequence of five rasters, where each raster has the shape $64 \times 64 \times 5$, whereas each Y sample is a raster of shape $64 \times 64 \times 1$, since we only want to predict the next air pollutant band occurrence. An overview of the arrays is shown in Table 6.2. Lastly the arrays mentioned above were split to create the ($X\,train$,$Y\,train$) and ($X\,test$,$Y\,test$) datasets. Specifically, 80% was selected as the training dataset and 20% was selected as the testing dataset. In addition, 20% of the training dataset was used as a validation dataset during training.

This process has been repeated for both of the different air pollutants datasets (CO, NO2).

[1] https://www.python.org/
[2] https://keras.io/
[3] https://www.tensorflow.org/

X	Y
$[X_1, X_2, X_3, X_4, X_5]$	$[X_6]$
$[X_2, X_3, X_4, X_5, X_6]$	$[X_7]$
$[X_3, X_4, X_5, X_6, X_7]$	$[X_8]$
...	
$[X_{N-5}, ..., X_{N-1}]$	$[X_N]$

Table 6.2: Overview of the training/testing dataset.

6.1.2 Evaluation Metrics

Generally low training loss values indicate that the model is learning well the underlying representations of the data. However, it does not mean that it is efficient. Therefore, it's important to consider other parameters as well to evaluate our models. Since, this is regression problem the Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error(see Section 2.3.2) are used to evaluate the predictions' quality. Moreover since the output of the models are 2D raster satellite images, the Structural Similarity Index Metric (SSIM) is used [Wan+04] for a better evaluation. SSIM's values vary between 0 and 1, where values closer to 1 indicate good similarities. The SSIM is calculated as:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{6.1}$$

, where x and y are the images to be compared.

Furthermore, the training and testing time are used as metrics to evaluate the computational cost of each model.

6.1.3 Hyperparameters

Before setting parameters for the models, combinations of losses, optimizers and learning rates have been tested. Ultimately, all models were trained using MAE as the loss function and the Adaptive Movement Optimization *Adam* optimizer (see Section 2.3.2). For the Adam optimizer the values 0.9 and 0.999 have been chosen as the $\beta 1$ and $\beta 2$ hyperparameters. Furthermore, the learning rate of 0.001 has been used with a decaying rate of 0.01 in order for the model to learn more complicated representations when reaching a plateau whilst training. Lastly, early stopping has been applied, by saving the model with the lowest validation loss as the training continues. Each model has been trained for 150 epochs with a batch size of 16.

As for the ConvLSTM model, each of its *ConvLSTM* layers has *64 filters*. The first ConvLSTM layer has a *kernel size* of 5×5, the second's size is 3×3, and the third's size is 1×1. The *3DConvolutional* layer has 1 filter in order to output only the air pollutant's band and a kernel size of $3 \times 3 \times 3$. Lastly, each layer's output was passed through a *ReLU* activation function. Table 6.3 shows more information about the ConvLSTM's parameters.

The CNN-LSTM's model *2DConvolutional* layer has a *kernel size* of 3×3 and the number of its *filters* is 64. The following *MaxPooling2D* layer has a *kernel size* of 2×2 and *stride size* of 1×1. Moreover, it's LSTM layers have a unit size of 16 and 64, while the last *Dense* layer has 4096 units for each pixel of the output's image. As with the ConvLSTM model, all layers pass their outputs through a *ReLU* activation function. More information about the CNN-LSTM model's parameters are shown in Table 6.4.

As been aforementioned, the 3D-UNet model is composed of three parts; the encoder, the bottleneck and the decoder part. For the encoder part, four convolutional blocks have been stacked. Each convolutional block has two *3DConvolutional* layers and a *3DMaxPooling* layer. The number

of convolutional filters grows exponentially after each convolutional block from 8 filters to 64. Every convolutional layer has a kernel size of 3×3 and every max pooling layer has a kernel size of $1 \times 2 \times 2$, in order to keep the temporal dimension of the data unchanged. The bottleneck part of the network is composed of a *3DMaxPolling* layer and three *3DConvolutional* layers of 64 filters. Further, the decoder part is composed of four up-sampling blocks, where each block shrinks the number of filters from 64 to 8. In the end a *2DConvolutional* layer is added with 1 filter and a kernel size of 1×1 to output the predicted image. Similarly to the other two models, each convolutional layers' output is pushed through a *ReLU* activation function. More information about the model's parameters can been seen in Table 6.5.

Layer	**Number of parameters**
ConvLSTM2D	1702912
BatchNormalization	512
Dropout	0
ConvLSTM2D	1180160
BatchNormalization	512
Dropout	0
ConvLSTM2D	131584
BatchNormalization	512
Dropout	0
Conv3D	3457

Table 6.3: ConvLSTM's parameters overview. The model has a total number of $3,018,881$ trainable parameters and 768 non-trainable parameters.

Layer	**Number of parameters**
ConvLSTM2D	2944
MaxPooling2D	512
Flatten	0
LSTM	16258112
LSTM	20736
Dense	266240

Table 6.4: CNN-LSTM's parameters overview. The model has a total number of $16,548,032$ trainable parameters.

6.2 Experimental Results

Two experiments have been conducted to assess the COVID 19 lockdowns impact on air quality. In the first experiment the three mentioned deep learning models are compared in order to find the best model for the air quality prediction task. Secondly, the proposed model is compared to a version of itself, which is trained using only the air pollutants' concentrations satellite images. This is to explore how much the lockdowns features actually affect our model's air quality predictions performance.

Layer	Parameters
Conv3D	1088
Conv3D	1736
MaxPooling3D	0
Conv3D	3472
Conv3D	6928
MaxPooling3D	0
Conv3D	13856
Conv3D	27680
MaxPooling3D	0
Conv3D	55360
Conv3D	110656
Dropout	0

(a) Encoder

Layer	Parameters
MaxPooling3D	0
Conv3D	221312
Conv3D	82048
Conv3D	442496
Dropout	0

(b) Bottleneck

Layer	Parameters
UpSampling3D	0
Conv3D	20544
Conv3D	65600
Conv3D	221248
Conv3D	110656
UpSampling3D	0
Conv3D	5152
Conv3D	16416
Conv3D	55328
Conv3D	27680
UpSampling3D	0
Conv3D	1296
Conv3D	4112
Conv3D	13840
Conv3D	6928
UpSampling3D	0
Conv3D	328
Conv3D	1032
Conv3D	3464
Conv3D	1736
Conv3D	434
Conv2D	3

(c) Decoder

Table 6.5: 3D-UNet's parameters overview. The model has a total number of $1,522,429$ trainable parameters.

6.2.1 Experiment 1: Comparison of machine learning models for air quality prediction using the COVID-19 dataset.

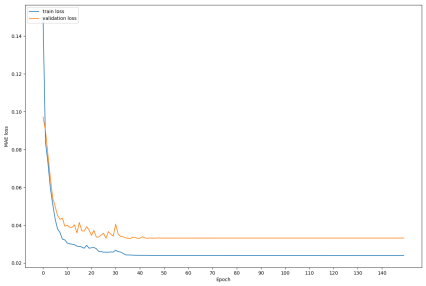
This subsection presents the results of the three models trained with the full datasets that were described in Section 4.3. The Figures 6.1 and 6.2 show the training and validation loss across the training of the models for the *CO* and *NO2* datasets respectively. Furthermore, Tables 6.6 and 6.7 show the evaluation metrics of each model for the two datasets. Lastly, examples of the predicted images of the proposed model for each area are displayed in Figures 6.3 and 6.4.

	MSE	RMSE	MAE	SSIM	Train time (seconds)
CNN-LSTM	0.0030	0.055	0.033	0.42	13,800
ConvLSTM	0.0019	0.043	0.026	0.54	30,300
3D-UNet	0.00069	0.026	0.016	0.74	8,850

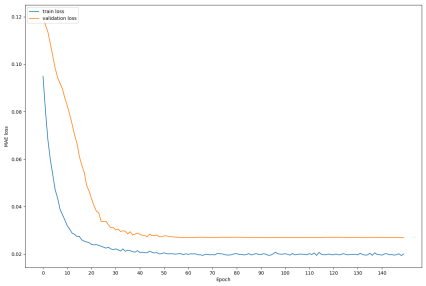
Table 6.6: Evaluation metrics of the three models for the CO dataset.

	MSE	RMSE	MAE	SSIM	Train time (seconds)
CNN-LSTM	0.0029	0.054	0.039	0.43	9,750
ConvLSTM	0.0025	0.050	0.029	0.51	29,550
3D-UNet	0.0002	0.014	0.0091	0.87	6,900

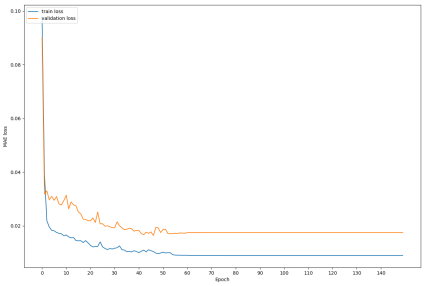
Table 6.7: Evaluation metrics of the three models for the NO2 dataset.



(a) CNN-LSTM model.

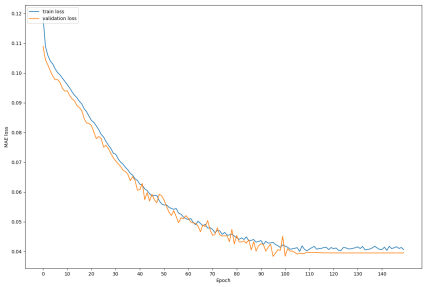


(b) ConvLSTM model

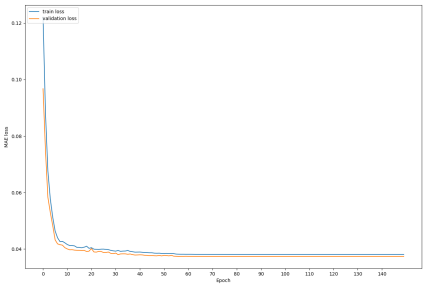


(c) 3D-UNet model

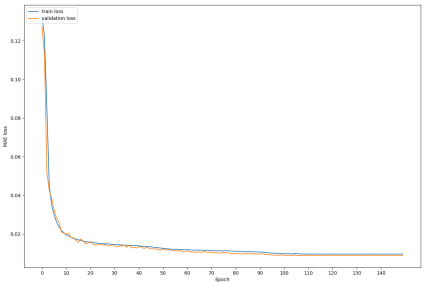
Figure 6.1: Training and validation losses of the three models during training for the CO dataset.



(a) CNN-LSTM model.

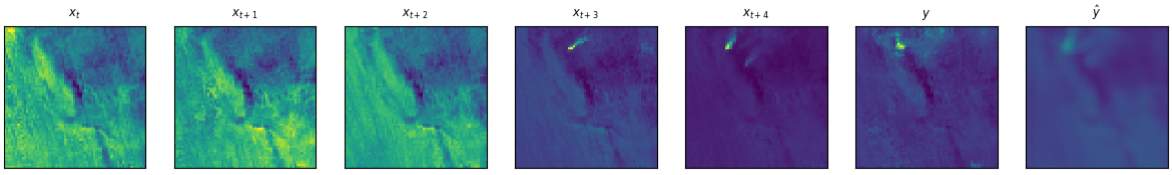


(b) ConvLSTM model

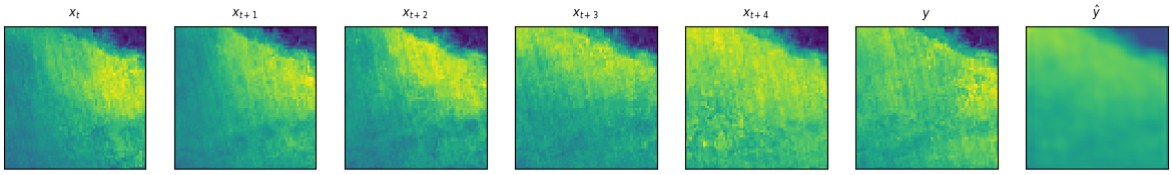


(c) 3D-UNet model

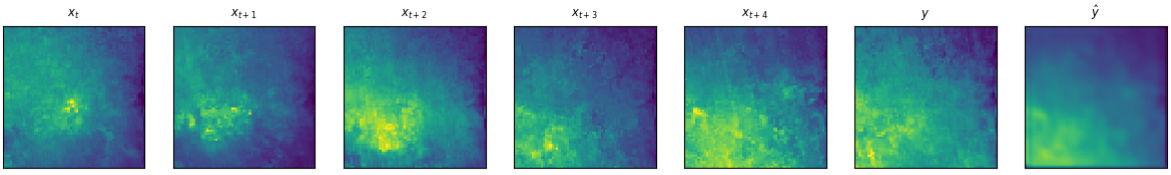
Figure 6.2: Training and validation losses of the three models during training for the NO2 dataset.



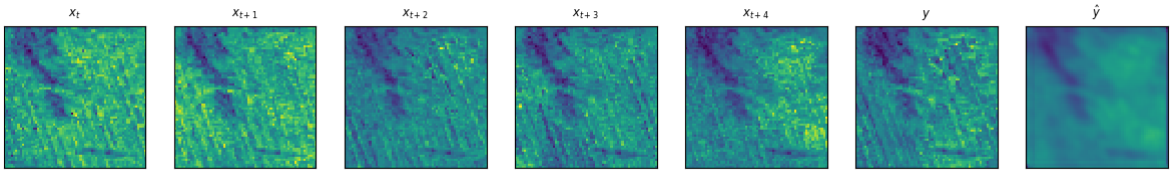
(a) California.



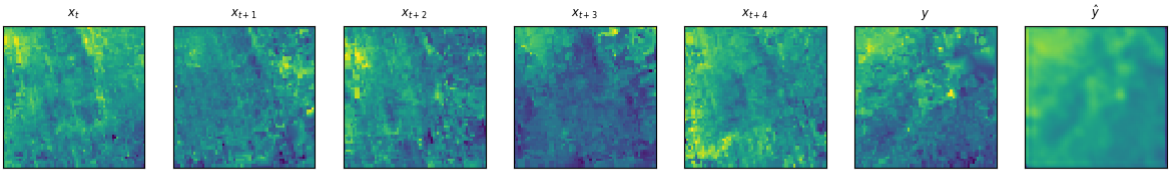
(b) Delhi.



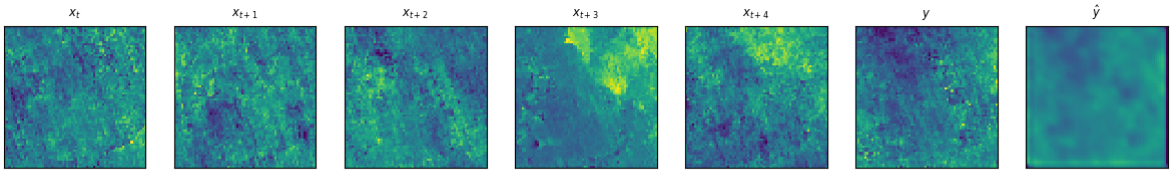
(c) Democratic Republic of Congo.



(d) Greece.

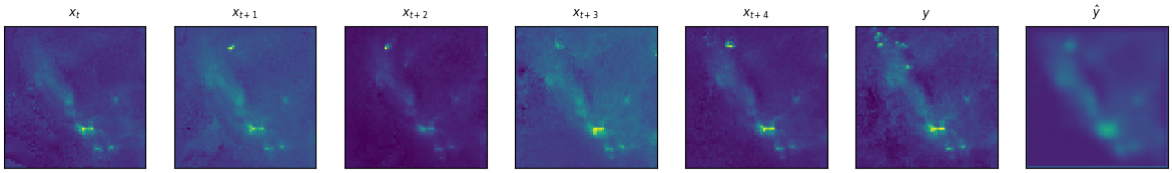


(e) South Korea.

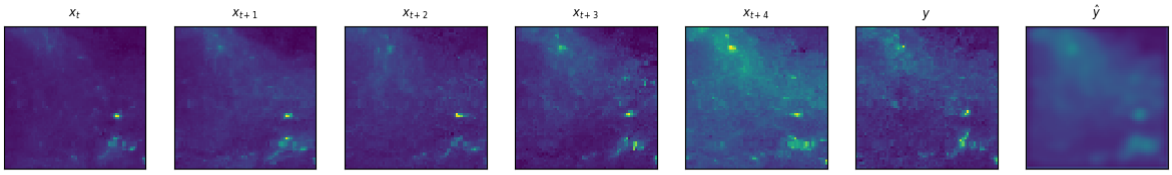


(f) United Kingdom.

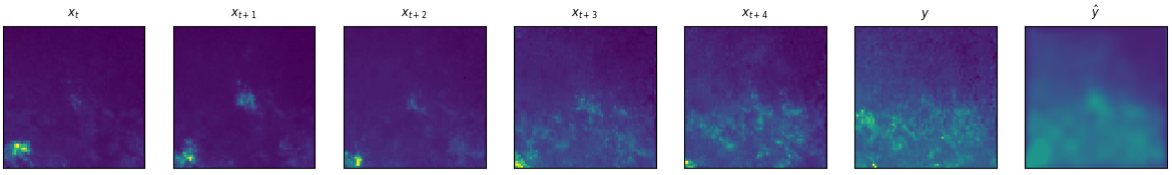
Figure 6.3: Prediction examples from the CO dataset using the 3D-UNet. From left to right the first five images of each area are the input data for the model, whereas the sixth and seventh image is the ground truth and the prediction respectively. Note that the policies bands of the input images are not shown in this figure for better presentation.



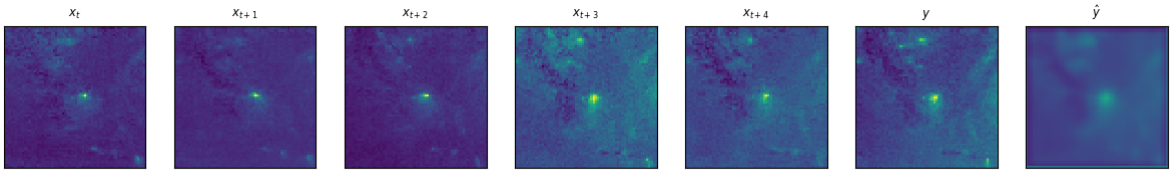
(a) California.



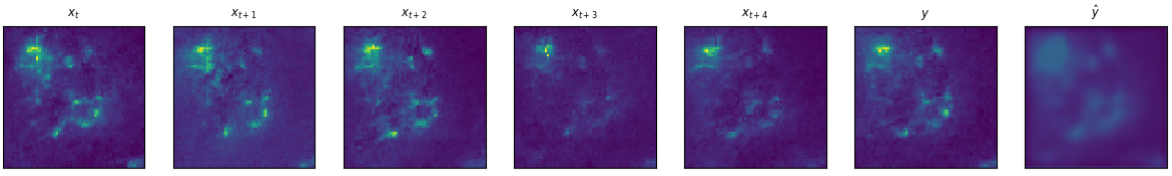
(b) Delhi.



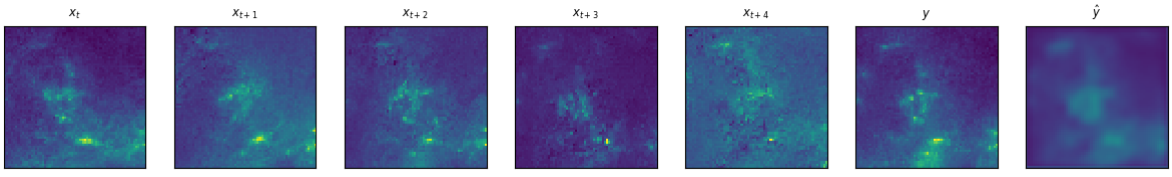
(c) Democratic Republic of Congo.



(d) Greece.



(e) South Korea.



(f) United Kingdom.

Figure 6.4: Prediction examples from the NO2 dataset using the 3D-UNet. From left to right the first five images of each area are the input data for the model, whereas the sixth and seventh image is the ground truth and the prediction respectively. Note that the policies bands of the input images are not shown in this figure for better presentation.

6.2.2 Experiment 2: Comparison of the proposed machine learning model using the OxCGRT dataset and without using it.

This subsection presents the results of the comparison between the proposed 3D-UNet model when trained using the OxCGRT dataset and when trained without it. Tables 6.8 and 6.9 show the metrics on the predictions of each model on the test dataset, whereas Tables 6.10 to 6.11 show the metrics differences of the two models based on each area's whole dataset. The name 3D-UNet (C19) corresponds to the 3D-UNet that was trained using the OxCGRT dataset.

	MSE	RMSE	MAE	SSIM
3D-UNet (C19)	0.00069	0.026	0.016	0.74
3D-UNet	0.00078	0.027	0.017	0.74

Table 6.8: Evaluation metrics of the three models for the CO dataset.

	MSE	RMSE	MAE	SSIM
3D-UNet (C19)	0.00021	0.014	0.0091	0.87
3D-UNet	0.00039	0.017	0.011	0.86

Table 6.9: Evaluation metrics of the three models for the NO2 dataset.

	MSE diff.	RMSE diff.	MAE diff.
California	-0.00001	0	0
Delhi	-0.00002	-0.001	-0.001
DRC	-0.00012	-0.002	-0.003
Greece	-0.00001	0	0
South Korea	-0.00003	-0.002	-0.0001
United Kingdom	-0.00001	-0.001	0

Table 6.10: Metric differences of the 3D-UNet (C19) and 3D-UNet predictions of the whole CO dataset grouped by area. The differences were calculated by subtracting the 3D-UNet metrics from the 3D-UNet (C19) metrics.

	MSE diff.	RMSE diff.	MAE diff.
California	-0.0001	-0.003	-0.004
Delhi	-0.00014	-0.002	-0.003
DRC	-0.00019	-0.005	-0.007
Greece	-0.0001	-0.005	-0.004
South Korea	-0.0008	-0.012	-0.009
United Kingdom	-0.0002	-0.003	-0.003

Table 6.11: Metric differences of the 3D-UNet (C19) and 3D-UNet predictions of the whole NO2 dataset grouped by area. The differences were calculated by subtracting the 3D-UNet metrics from the 3D-UNet (C19) metrics.

Chapter 7

Conclusion

This study aims to evaluate the impact of Covid-19 lockdowns on air quality using deep learning models. We started by providing the reader with the theoretical background on the different deep learning techniques used. Further, we explained the generation process of the input dataset, which contained both air quality and lockdown data. Moreover, we implemented and compared three different deep learning models that can use spatiotemporal data to make future air quality frame predictions. The best-performing model is then compared to a version of itself trained without lockdowns data to assess the impact of those features on the model's prediction performance. Our findings showed a slight performance increase when the lockdown data was used. More specifically, 6.11 and 6.10 show that the lockdown features affect more the areas with the highest air pollutants concentrations; Democratic Republic of Congo for the CO dataset, and South Korea for the NO2 dataset.

7.1 Discussion

After comparing the three different models, we were surprised by the better performance of the 3D-UNet architecture over the other two models. The results showed the potential of such an architecture, which could learn spatiotemporal data using only convolutional layers. In addition, because the 3D-UNet only used convolutional layers, the processing time and the number of parameters of the model were much lower than the other two. Nevertheless, we believe that there is still space to further improve the performance of all three models by choosing better parameters and generally better optimization. It is also important to note that the choice of learning rate and batch size played an influential role in the performance of this task.

Furthermore, the results of our second experiment showed that the lockdown features had a minimal positive impact on our model's performance. We consider that this happens for several reasons. First and foremost, the impact of the COVID-19 lockdowns on air quality over these areas might not be significant enough to increase our proposed model's performance majorly. Nonetheless, Tables 6.11 and 6.10 indicate that the lockdown features slightly increase performance more in areas where the air pollution was the highest, which might show that these areas' air quality was impacted the most by the lockdown policies. Additionally, it should be pointed out that the lockdown features were very noisy since it was impossible to know how well each of these policies was enforced, which could significantly affect the outcome of their impact on air quality. All things considered, a spatiotemporal predictive model has been implemented, which is capable of using lockdown and air quality features for its predictions.

7.2 Future Work

There are at least the following three proposals for future work:

Firstly, all of the deep learning models that were explored should be more thoroughly examined and fine-tuned. We believe that not only the proposed model but the other two models can obtain even better results. One way to achieve this is by performing a more extensive hyperparameter search. Moreover, more advanced techniques such as inception modules, residual blocks, asymmetric convolutions, and bidirectional LSTM layers should be explored to further increase our models' performances [FAM20]. Unfortunately, this is beyond the timeframe of this theses; hence finding acceptable results in the first experiment was the prime motivation.

Secondly, since the 3D-UNet approach yielded such promising results, exploring other powerful convolutional autoencoders such as the Generative Adversarial Networks (GANs) is also essential. GANs have shown great promise in the field of future frame prediction [Lee+18]; therefore, implementing them for this thesis' task should be further examined.

Lastly, and probably most importantly, we believe that a better feature selection could achieve more insightful results about the impact of lockdowns on air quality. In this thesis, we had access to 10-day aggregated air quality data (see Section 4.1). We believe that more accurate features such as daily air quality data could better reflect air quality change over an area. Furthermore, as mentioned, the lockdown features were very noisy since every lockdown was differently enforced and applied, which means that every lockdown had a different effect on vehicle traffic, which had a different effect on air quality. Therefore, as used in [TGC21], features such as daily traffic data could prove very useful for the model to better understand the relationships between lockdowns and air quality. Lastly, as mentioned in Section 2.2.1, other factors such as weather and temperature significantly affect air quality and should be taken in consideration in the prediction part of the experiment in future works.

Bibliography

[Wer90] Paul Werbos. "Backpropagation through time: what it does and how to do it." In: *Proceedings of the IEEE* 78 (Nov. 1990), pp. 1550–1560. DOI: 10.1109/5.58337.

[BSF94] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult." In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory." In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[Lec+98] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[Wan+04] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. "Image quality assessment: from error visibility to structural similarity." In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[Jam05] Susan Jamieson. "Likert Scales: How to (ab) Use Them." In: *Medical education* 38 (Jan. 2005), pp. 1217–8. DOI: 10.1111/j.1365-2929.2004.02012.x.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[Gra+06] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks." In: vol. 2006. Jan. 2006, pp. 369–376. DOI: 10.1145/1143844.1143891.

[TSK06] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson Education, 2006.

[Lov+09] Gary M. Lovett, Timothy H. Tear, David C. Evers, Stuart E.G. Findlay, B. Jack Cosby, Judy K. Dunscomb, Charles T. Driscoll, and Kathleen C. Weathers. "Effects of Air Pollution on Ecosystems and Biological Diversity in the Eastern United States." In: *Annals of the New York Academy of Sciences* 1162.1 (Apr. 2009), pp. 99–135. DOI: 10.1111/j.1749-6632.2009.04153.x. URL: https://doi.org/10.1111/j.1749-6632.2009.04153.x.

[Mas+11] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. "Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction." In: *Artificial Neural Networks and Machine Learning – ICANN 2011*. Ed. by Timo Honkela, Włodzisław Duch, Mark Girolami, and Samuel Kaski. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 52–59. ISBN: 978-3-642-21735-7.

[DSH13] George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. "Improving deep neural networks for LVCSR using rectified linear units and dropout." In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 2013, pp. 8609–8613. DOI: 10.1109/ICASSP.2013.6639346.

[WWL13] Stefan Wager, Sida Wang, and Percy S Liang. "Dropout Training as Adaptive Regularization." In: *Advances in Neural Information Processing Systems*. Ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper/2013/file/38db3aed920cf82ab059bfccbd02be6a-Paper.pdf.

[Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[IS15] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[Kar15] Andrej Karpathy. *Understanding LSTM Networks*. May 2015. URL: :%20http://colah.github.%20io/posts/2015-08-Understanding-LSTMs.

[KB15] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A method for stochastic optimization." In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015), pp. 1–15. arXiv: 1412.6980.

[Ola15] Christopher Olah. *The Unreasonable Effectiveness of Recurrent Neural Networks*. Aug. 2015. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." In: (2015). eprint: arXiv:1505.04597.

[Shi+15] Xingjian Shi, Zhourong Chen, Hao Wang, Dit Yan Yeung, Wai Kin Wong, and Wang Chun Woo. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting." In: *Advances in Neural Information Processing Systems* 2015-Janua (2015), pp. 802–810. ISSN: 10495258. arXiv: 1506.04214.

[Zhe+15] Yu Zheng, Xiuwen Yi, Ming Li, Ruiyuan Li, Zhangqing Shan, Eric Chang, and Tianrui Li. "Forecasting fine-grained air quality based on big data." In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 2015-August (2015), pp. 2267–2276. DOI: 10.1145/2783258.2788573.

[Don+16] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. *Long-term Recurrent Convolutional Networks for Visual Recognition and Description*. 2016. arXiv: 1411.4389 [cs.CV].

[DV16] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning." In: (2016), pp. 1–31. arXiv: 1603.07285. URL: http://arxiv.org/abs/1603.07285.

[GRB16] Adel Ghorani-Azam, Bamdad Riahi-Zanjani, and Mahdi Balali-Mood. "Effects of air pollution on human health and practical measures for prevention in Iran." In: *Journal of Research in Medical Sciences* 21.5 (2016). ISSN: 17357136. DOI: 10.4103/1735-1995.189646.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[Mel+16] Abdelwahid Mellouki, Christian George, Fahe Chai, Yujing Mu, Jianmin Chen, and Hong Li. "Sources, chemistry, impacts and regulations of complex air pollution: Preface." In: *Journal of Environmental Sciences* 40 (Feb. 2016), pp. 1–2. DOI: 10.1016/j.jes.2015.11.002. URL: https://doi.org/10.1016/j.jes.2015.11.002.

[Rud16] Sebastian Ruder. “An overview of gradient descent optimization algorithms.” In: (2016), pp. 1–14. arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[Gro17] Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st. O’Reilly Media, Inc., 2017. ISBN: 1491962291.

[Iof17] Sergey Ioffe. *Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models*. 2017. arXiv: 1702.03275 [cs.LG].

[Kim+17] Seongchan Kim, Seungkyun Hong, Minsu Joh, and Sa-kwang Song. “DeepRain: ConvLSTM Network for Precipitation Prediction using Multichannel Radar Data.” In: (2017), pp. 3–6. arXiv: 1711.02316. URL: http://arxiv.org/abs/1711.02316.

[Kar18] Md. Rezaul Karim. *Practical Convolutional Neural Networks : Implement advanced deep learning models using Python*. Birmingham: Packt Publishing, 2018. ISBN: 9781788392303.

[Lee+18] Alex X. Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. “Stochastic Adversarial Video Prediction.” In: *CoRR* abs/1804.01523 (2018). arXiv: 1804.01523. URL: http://arxiv.org/abs/1804.01523.

[Liu+18] Wen Liu, Weixin Luo, Dongze Lian, and Shenghua Gao. *Future Frame Prediction for Anomaly Detection – A New Baseline*. 2018. arXiv: 1712.09867 [cs.CV].

[Nwa+18] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning.” In: (2018), pp. 1–20. arXiv: 1811.03378. URL: http://arxiv.org/abs/1811.03378.

[Agr+19] Shreya Agrawal, Luke Barrington, Carla Bromberg, John Burge, Cenk Gazen, and Jason Hickey. “Machine Learning for Precipitation Nowcasting from Radar Images.” In: NeurIPS (2019), pp. 1–6. arXiv: 1912.12132. URL: http://arxiv.org/abs/1912.12132.

[Arr+19] Leila Arras, Jose Arjona-Medina, Michael Widrich, Grégoire Montavon, Michael Gillhofer, Klaus-Robert Müller, Sepp Hochreiter, and Wojciech Samek. “Explaining and Interpreting LSTMs.” In: Sept. 2019, pp. 211–238. ISBN: 978-3-030-28953-9. DOI: 10.1007/978-3-030-28954-6_11.

[Koe19] Will Koehrsen. “Overftting vs . Underftting : A Complete Example.” In: *Towards Data Science* (2019), pp. 1–12.

[Sch19] Robin M. Schmidt. “Recurrent Neural Networks (RNNs): A gentle Introduction and Overview.” In: 1 (2019), pp. 1–16. arXiv: 1912.05911. URL: http://arxiv.org/abs/1912.05911.

[Xia+19] Changjiang Xiao, Nengcheng Chen, C. Hu, Ke Wang, Z. Xu, Yaping Cai, Lei Xu, Zeqiang Chen, and Jianya Gong. “A spatiotemporal deep learning model for sea surface temperature field prediction using time-series satellite data.” In: *Environmental Modelling and Software* 120.December 2018 (2019), p. 104502. ISSN: 13648152. DOI: 10.1016/j.envsoft.2019.104502. URL: https://doi.org/10.1016/j.envsoft.2019.104502.

[Zha+19a] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. http://www.d2l.ai. 2019.

[Zha+19b] Jiachen Zhao, Fang Deng, Yeyun Cai, and Jie Chen. "Long short-term memory - Fully connected (LSTM-FC) neural network for PM2.5 concentration prediction." In: *Chemosphere* 220 (2019), pp. 486–492. ISSN: 18791298. DOI: 10.1016/j.chemosphere.2018.12.128. URL: https://doi.org/10.1016/j.chemosphere.2018.12.128.

[CEL20] Matthew A. Cole, Robert J.R. Elliott, and Bowen Liu. "The Impact of the Wuhan Covid-19 Lockdown on Air Pollution and Health: A Machine Learning and Augmented Synthetic Control Approach." In: *Environmental and Resource Economics* 76.4 (2020), pp. 553–580. ISSN: 15731502. DOI: 10.1007/s10640-020-00483-4.

[FAM20] Jesús García Fernández, Ismail Alaoui Abdellaoui, and Siamak Mehrkanoon. "Deep coastal sea elements forecasting using U-Net based models." In: 2015 (2020). arXiv: 2011.03303. URL: http://arxiv.org/abs/2011.03303.

[Hua+20] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. *Normalization Techniques in Training DNNs: Methodology, Analysis and Application*. 2020. arXiv: 2009.12836 [cs.LG].

[Lia+20] Yun Chia Liang, Yona Maimury, Angela Hsiang Ling Chen, and Josue Rodolfo Cuevas Juarez. "Machine learning-based prediction of air quality." In: *Applied Sciences (Switzerland)* 10.24 (2020), pp. 1–17. ISSN: 20763417. DOI: 10.3390/app10249151.

[Man+20] Ioannis Manisalidis, Elisavet Stavropoulou, Agathangelos Stavropoulos, and Eugenia Bezirtzoglou. "Environmental and Health Impacts of Air Pollution: A Review." In: *Frontiers in Public Health* 8.February (2020), pp. 1–13. ISSN: 22962565. DOI: 10.3389/fpubh.2020.00014.

[Mos+20] Waytehad Rose Moskolaï, Wahabou Abdou, Albert Dipanda, and Dina Taiwe Kolyang. "Application of LSTM architectures for next frame forecasting in Sentinel-1 images time series." In: *arXiv* (2020), p. 13. ISSN: 23318422.

[MLS20] Sulaman Muhammad, Xingle Long, and Muhammad Salman. "COVID-19 pandemic and environmental pollution: A blessing in disguise?" In: *Science of the Total Environment* 728 (2020), p. 138820. ISSN: 18791026. DOI: 10.1016/j.scitotenv.2020.138820. URL: https://doi.org/10.1016/j.scitotenv.2020.138820.

[Søn+20] Casper Kaae Sønderby, Lasse Espeholt, Jonathan Heek, Mostafa Dehghani, Avital Oliver, Tim Salimans, Shreya Agrawal, Jason Hickey, and Nal Kalchbrenner. "MetNet: A Neural Weather Model for Precipitation Forecasting." In: (2020), pp. 1–17. arXiv: 2003.12140. URL: http://arxiv.org/abs/2003.12140.

[Wan+20] Yichen Wang, Yuan Yuan, Qiyuan Wang, Chen Guang Liu, Qiang Zhi, and Junji Cao. "Changes in air quality related to the control of coronavirus in China: Implications for traffic and industrial emissions." In: *The Science of the total environment* 731.December 2019 (2020), p. 139133. ISSN: 18791026. DOI: 10.1016/j.scitotenv.2020.139133.

[Hal+21] Thomas Hale, Petherick Anna, Philips Toby, and Webster Samuel. "Variation in government responses to COVID-19." In: (2021). URL: https://www.bsg.ox.ac.uk/research/research-projects/covid-19-government-response-tracker.

[Lov+21] Mario Lovrić, Kristina Pavlović, Matej Vuković, Stuart K. Grange, Michael Haberl, and Roman Kern. "Understanding the true effects of the COVID-19 lockdown on air pollution by means of machine learning." In: *Environmental Pollution* 274 (2021). ISSN: 18736424. DOI: 10.1016/j.envpol.2020.115900.

[MMS21] Cosimo Magazzino, Marco Mele, and Samuel Asumadu Sarkodie. “The nexus between COVID-19 deaths, air pollution and economic growth in New York state: Evidence from Deep Machine Learning.” In: *Journal of Environmental Management* 286.March (2021), p. 112241. ISSN: 10958630. DOI: 10.1016/j.jenvman.2021.112241. URL: https://doi.org/10.1016/j.jenvman.2021.112241.

[RZ21] Yves Rybarczyk and Rasa Zalakeviciute. “Assessing the COVID-19 Impact on Air Quality: A Machine Learning Approach.” In: *Geophysical Research Letters* 48.4 (2021), pp. 1–11. ISSN: 19448007. DOI: 10.1029/2020GL091202.

[TGC21] Animesh Tiwari, Rishabh Gupta, and Rohitash Chandra. “Delhi air quality prediction using LSTM deep learning models with a focus on COVID-19 lockdown.” In: (2021). arXiv: arXiv:2102.10551v1.

[Tra+21] Marco Travaglio, Yizhou Yu, Rebeka Popovic, Liza Selley, Nuno Santos Leal, and Luis Miguel Martins. “Links between air pollution and COVID-19 in England.” In: *Environmental Pollution* 268 (2021), p. 115859. ISSN: 18736424. DOI: 10.1016/j.envpol.2020.115859. URL: https://doi.org/10.1016/j.envpol.2020.115859.

[TSM21] Kevin Trebing, Tomasz Stańczyk, and Siamak Mehrkanoon. “SmaAt-UNet: Precipitation nowcasting using a small attention-UNet architecture.” In: *Pattern Recognition Letters* 145 (2021), pp. 178–186. ISSN: 01678655. DOI: 10.1016/j.patrec.2021.01.036. arXiv: 2007.04417.

[Wan+21] Siyu Wang, Yanli Zhang, Jinlong Ma, Shengqiang Zhu, Juanyong Shen, Peng Wang, and Hongliang Zhang. “Responses of decline in air pollution and recovery associated with COVID-19 lockdown in the Pearl River Delta.” In: *Science of the Total Environment* 756 (2021), p. 143868. ISSN: 18791026. DOI: 10.1016/j.scitotenv.2020.143868. URL: https://doi.org/10.1016/j.scitotenv.2020.143868.

[WHO] WHO. *Air Pollution*. URL: https://www.who.int/health-topics/air-%20pollution#tab=tab_1/. (accessed: 21.05.2021).