# Politecnico di Torino

**Master's Degree in Data Science and Engineering**


**Master's Degree Thesis**

# Convolutional neural networks for statistical post-processing of wind gusts speed

**Supervisors:**
Prof. Roberto Fontana
Prof. Elisa Perrone
Dr. Kirien Whan

**Candidate:**
Francesco Guardamagna

**April 2022**

# Abstract

The wind gusts, are defined by KNMI, the royal Netherlands meteorological institute, as the largest 3-second wind speed value in the last 10 minutes. These sudden bursts in the wind speed are critical, because they can cause different types of severe damages, to people and properties. For KNMI's weather warning system is of great interest to understand, if the speed of wind gusts will or not be above a certain threshold. This can be of course really useful to emit alarms and activate protocols, according to the severity level. Nowadays KNMI, in the Netherlands uses numerical weather prediction models, to forecast future weather, in a certain geographical area. Due to the complex and chaotic nature of the atmosphere system, these models present systematic errors and biases. Moreover NWP forecasts are deterministic, so it's impossible to estimate the amount of predictions' uncertainty. It's a common practice to perform a Statistical Post-Processing procedure over NWP forecasts, to convert deterministic predictions into probabilistic ones and to correct systematic biases and errors. Our project focuses on the use of deep-learning convolutional architectures, for Statistical Post-Processing of deterministic wind gusts predictions, obtained with the HARMONIE-AROME NWP model. CNNs allow us to take advantage of the spatial relationships naturally present in our input data. We propose a convolutional model able to estimate the probability for a specific input example, to be related to a wind gust speed above a certain threshold. In other words we convert our forecasting problem into a binary classification one. We also propose a pre-processing technique, to reduce the number of features, considered during our features selection procedure. Moreover we propose 2 different techniques to overcome the problem of the strongly imbalanced class distribution, that characterizes our training data-set, due to the low number of entries related to high wind gusts speed, that are available.

## Acknowledgement

I want to say thank you to professor Roberto Fontana, who has given me the opportunity to go abroad for my final project and to live a really interesting and stimulating experience. Of course I want to say thank you to Professor Elisa Perrone and Doctor Kirien Whan, who have supervised my thesis work, in the last six months, kindly and patiently supporting me in every phase of the project. On a more personal and intimate level I want to thank my family to have emotionally and economically supported me in the last years, giving me the opportunity to finish my university studies and to follow my dreams. I also want to thank all the friends I've met during the university's years, for accompanying me in this really difficult, but at the same time rewarding adventure.

# List of Figures

# List of Tables



# Contents

# 1 Introduction

This thesis project has been conducted at the Eindhoven University of Technology, in collaboration with the Royal Netherlands Meteorological Institute (KNMI).
Wind gusts, are defined by The Royal Netherlands Meteorological Institute (KNMI), as the largest 3-second wind speed value in the last 10 minutes. These sudden bursts in the wind speed are critical, because they can cause different types of severe damages, to people and properties. For this reason is of great interest for meteorological institutes like KNMI to generate reliable and precise forecasts, to emit alarms and activate protocols, according to the level of severity. Forecasts are usually produced by Numerical Weather Predictions (NWP) models like the HARMONIE-AROME model of KNMI. To be computationally feasible, these models need simplify assumptions, and moreover a perfect definition of the initial conditions is not possible, due to the chaotic nature of the atmosphere system. For these reasons NWP forecasts present systematic errors and biases. Furthermore a single, deterministic NWP forecast, doesn't provide an estimation of its uncertainty. A Statistical Post-Processing procedure is performed, over NWP predictions to correct uncertainty and bias and to transform a deterministic forecast into a probabilistic one. For our project, we perform a Statistical Post-Processing operation, over deterministic NWP forecasts (about weather variables, related to wind gusts), taking advantage of the spatial relationships naturally present in our input data. To do so we used a model based on a Convolutional Neural Network, which is a deep learning architecture able to recognize the spatial patterns present in the input data. To be more precise we transform a forecasting problem into a binary classification one, estimating the conditional probability for the wind gusts speed, to be higher than a certain threshold, given a specific input example. Our input data are in spatial format and are characterized by a large number of predictors. For this reason an exhaustive search, that analyzes all the possible combinations of input variables is unfeasible. To reduce the number of features taken into consideration, during our feature selection procedure, we propose a pre-processing dimensionality reduction technique, based on the level of correlation between the input predictors. Another problem we focus on is related, to the strongly imbalanced class distribution, that characterizes the training data-set we have worked with. In our data-set a few entries related to really high wind gusts speed are available. For this reason if we define our Binary Classification problem, using a high threshold, we obtain a training data-set, characterized by a strongly imbalanced class distribution. We propose two different approaches to solve the class distribution problem.

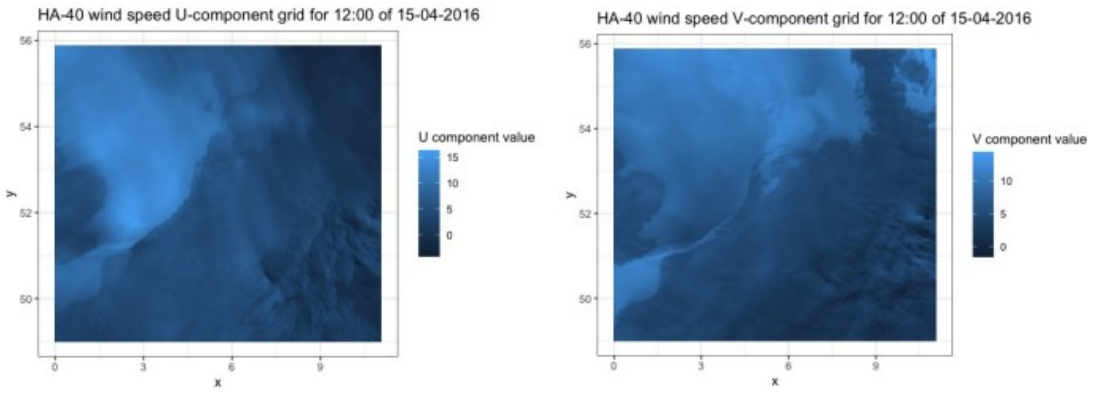
# 2 State of the art

The most common traditional methods, used for statistical post-processing of NWP forecasts are distributional regression approaches, where probability forecasts are estimated, fitting parametric distributions. The parameters of these distributions depend, through a link function, on statistics of the raw predictions, about the weather variable of interest. The two most common examples are Model Output Statistics (MOS) and its extension for ensemble forecasts, Ensemble Model Output Statistics (EMOS), both this methods are described in 4.1.2. The use of Machine Learning and Deep Learning techniques for Statistical Post-Processing, has been the focus of more recent research projects. Machine learning and Deep Learning post-processing techniques enable the incorporation of other predictors, besides the target variable of interest, and allow to estimate more complex relations, between NWP forecasts and the real observations. Despite the importance of wind gusts, for weather warnings, most of the work, concerning Statistical Post-Processing techniques, has focused in the recent years, on other weather variables, like temperature, precipitation and mean wind speed. In [37] Schulz et al. have performed a comparison between traditional statistical post-processing techniques, and more modern Machine Learning and Deep Learning Techniques. They have focused on Statistical Post-Processing of wind gusts ensemble forecasts, experimenting with Machine Learning methods like Quantile Regression Forests, and with Deep Learning Techniques, based on a Multi-Layer Perceptron architecture (the standard Feed-Forward architecture). Our work focuses on Statistical Post-Processing of deterministic and not ensemble forecasts, using a different deep learning architecture, the Convolutional one, able to recognize spatial patterns in the input data. In [35] Veldkamp at al. apply the Convolutional architecture for Statistical Post-processing of 48h lead time deterministic forecasts about the average wind speed. They focused on a different weather variable, compared to the one taken into consideration during our project. Moreover they developed methods able to estimate a continuous conditional

distribution, without transforming their problem into a binary classification task. We also focus on one of the main problem we have to face when we apply deep learning to Weather Forecasting: the scarcity of examples related to extreme events, that are available to train our model. Zoe van den Heuvel et al. in [45], investigated the use of Convolutional networks, for statistical post-processing, of wind gusts deterministic forecasts, comparing the performances with traditional post-processing frameworks, like MOS and Logistic Regression. They converted, like us, a forecasting problem, into a binary classification one, to improve the accuracy of weather warnings. Zoe van den Heuvel et al. in [45], didn't propose solutions to the problem of the imbalanced class distribution. In our work, we also propose a pre-processing technique, able to reduce the number of features considered, during the feature selection procedure.

# 3 Data

The Data we have worked with, are grid deterministic predictions, obtained using the Harmonie-Arome NWP model, from KNMI. The model is run over a grid of 2.5km*2.5km grid boxes. This means that the Harmonie-Arome NWP model produces predictions values for every 2.5 kilometers in the Netherlands. The data refer to three different years: 2015, 2016, 2017. The months considered are the ones from April to October, corresponding to the summer period. The lead times we decided to take in consideration are 3: 11 hours, 12 hours, 13 hours. The data we made use of, contains 27 variables with a physical relationship with the observed wind gusts. The complete list of variables considered, can be found in B. The target variable is the wind gusts speed, recorded by 46 different stations in the Netherlands. The data-set we have worked with contains samples related to all the different stations. We have decided to considered all the station at the same time, developing a general model, that doesn't refer to a single station.
The entries in Input to our model, have been obtained, constructing a smaller square matrix of dimension 100*100 grid boxes, around the geographical position of a specific station. The labels associated to each example are, the real observations, measured by the corresponding stations. Experiments have been performed, also taking into consideration more than one predictor. The different variables have been superimposed, like different channels of an image.




(a) (b)

Figure 1: Examples of grid predictions, about the the U and V orthogonal components of the variable "10m wind speed", forecast by a run of the Harmonie-Arome Model, initialized at 0000-UTC (the magnitude of the wind speed can be obtained using the Pythagorean Theorem, on the U and V components). These images have been taken from [45]

# 4 Background Theory

## 4.1 Numerical Weather Prediction models

Numerical Weather Prediction (NWP) models, are able to predict, using mathematical models, the future weather conditions, starting from the atmosphere current state. The history of NWP models began in the 1920, when Lewis Fry Richardson, was able to produce a six-hour forecast for the state of the atmosphere over two different points in central Europe. He took six months to do so. It was with the advent of computers and computers' simulation, that the time required for the computation, reduce to less than the time of the forecast period itself. As it is reported in [38], NWP models employ a set of equations that describe, the flow of fluids. These equations are translated into computer's code and using governing equations, numerical methods and the parametrization of certain physical processes, the model is run over a specific domain (geographical area). Also the initial and boundary conditions have to be specified, to generate the forecast.

- Numerical Methods: Numerical Methods are used in order to transform spatial and temporal derivatives, into a form, that can be solved by computers.
- Parametrization: Physical Processes, that cannot be directly predicted in full details, by the model's forecast equations, need to be parametrized. These physical processes require a reasonable statistical representation. In this way, we can take into account their effects in the simulation.
- Initial Conditions: The initial conditions define the Atmosphere's current state.
- The Boundary Conditions: The Boundary conditions define the different domains' edges.

The Chaotic Nature of the Atmosphere system, combined with a certain level of approximation needed to represent certain phenomena, introduce uncertainty in Numerical Weather Predictions. For this reason Forecaster have always understood the value of examining multiple NWP forecasts, in order to produce a more reliable prediction. For example the spaghetti diagrams, use statistical and graphical tools, to allow the Forecaster to compare multiple runs of the same model, obtained using different initial conditions or different configurations and parametrizations.

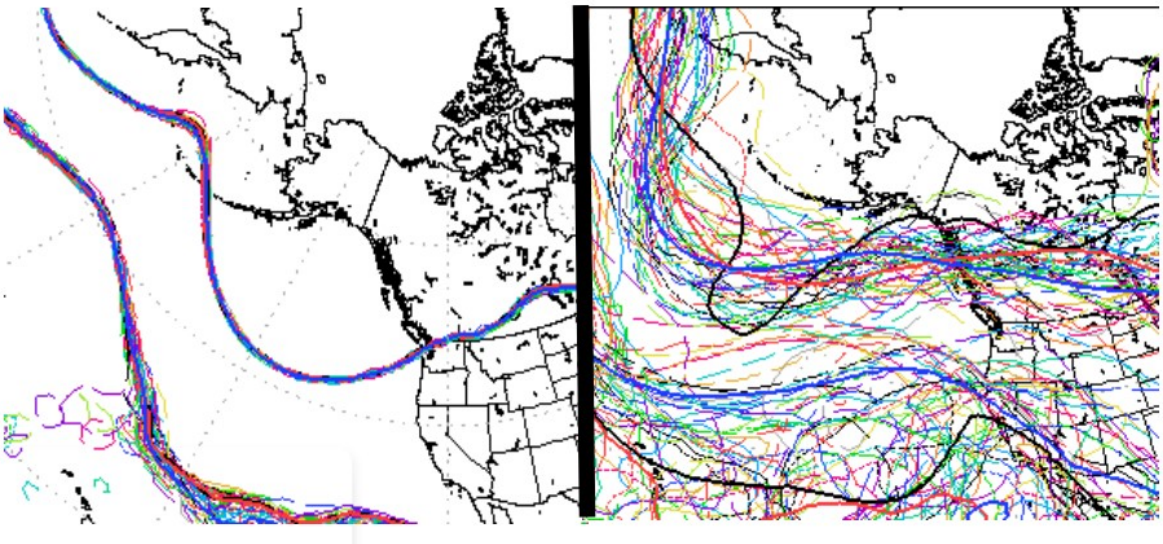


Figure 2: Example of spaghetti plot. This image has been taken from [38]

Figure 2 shows an example of spaghetti plot. Each line is a different model's output, obtained with different initial conditions or parameters. Looking at the figure on the left, we can notice a lower spread in the solutions, and consequently a higher confidence in the prediction. Looking at the figure on the right we can notice a larger spread and consequently a lower confidence.

### 4.1.1 Statistical Post-Processing

As it is reported by Schulz et al. in [37], Numerical Weather Prediction models usually generate probabilistic forecasts in the form of ensembles of deterministic predictions. The different forecasts in the ensemble, are obtained changing the initial and boundary conditions as well as the model specifications. Despite the substantial improvements in the last years, ensemble forecasts continue to exhibit systematic errors and biases. For this reason to obtain accurate and reliable probabilistic forecasts a Statistical Post-Processing operation, based on past, real observations is required. Statistical Post-Processing has in fact become, an essential part of weather forecasting, to correct systematic errors and biases in NWP forecasts. Statistical Post-Processing is also required to obtain a probabilistic forecast, if we are working with deterministic and not ensemble forecasts. A popular technique for statistical post processing is Model Output Statistics (MOS), which is able to estimate a statistical relationship between the predictions, provided by the NWP models and the observed measurements. MOS works with deterministic forecasts, while its extention, the EMOS (Ensemble Model Output Statistics) framework, is able to work with ensemble forecasts. Both MOS and EMOS fit a parametric distribution. MOS and EMOS are the standard and traditional Statistical Post-Processing techniques for weather predictions. EMOS is an extention of the MOS framework, the only difference between these 2 techniques, is in the input data.

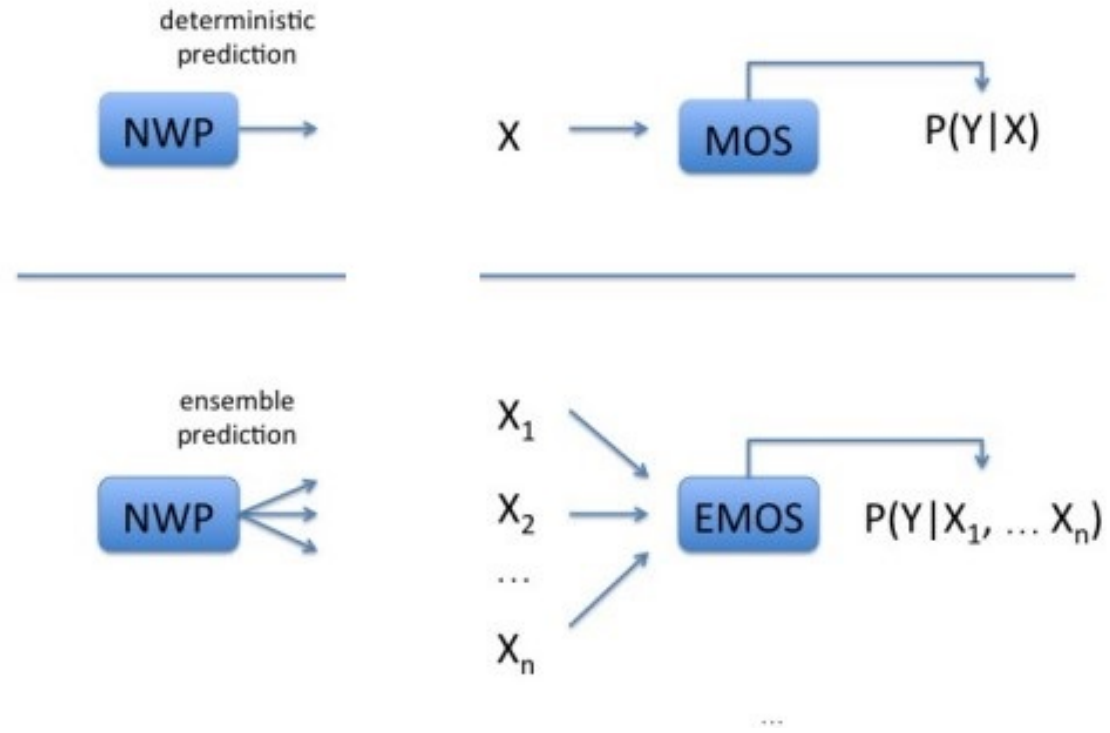



Figure 3: Difference between EMOS and MOS frameworks. This image has been taken from [45]

### 4.1.2 EMOS and MOS frameworks

The following description of the EMOS and MOS frameworks, is based on [37] and [45], by Schulz et al. and Van Den Heuvel et al. respectively. EMOS and MOS are distributional regression approaches. They assume that given a deterministic prediction $x$ (or an ensemble forecast in the case of EMOS), the target variable $Y$, follows a parametric distribution $\Gamma(\theta)$. $\theta \in \Theta$, where $\Theta$ is the parameters' space of $\Gamma$. The parameters' vector $\theta$ is related to the deterministic or ensemble forecast $x$, through a function $g$.

$$P(Y|x) \sim \Gamma(\theta) \quad \theta = g(x) \in \Theta$$

The choice of the parametric family depends on the weather variable of interest. For example Gneiting

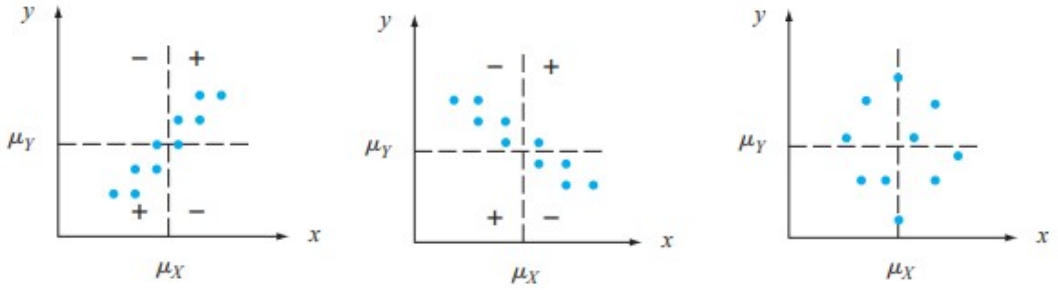
at al. in [15], decide to use a Gaussian Distribution, for temperature and Sea Level Pressure, while Schulz et al. in [37], decide to use a Truncated Logistic Distribution for wind gusts. The parameters of the function $g$ are estimated minimizing a proper loss function like the Continuos Ranked Probability score (CRPS) (The CRPS loss is described in appendix J).

## 4.2 Covariance and Correlation

Let's take into consideration two random variables $X$ and $Y$. If they are not independent, we can be interested in understanding, how strongly they are related to one another. The Covariance, between $X$ and $Y$ can tell us how strongly, the two random variables are related:

$$Cov(X, Y) = E[(X - u_x)(Y - u_y)]$$

$(X - u_x)$ and $(Y - u_y)$ are the deviations of $X$ and $Y$, from their respective mean values, so the Covariance between two random variables is the product of their deviations. Let's suppose that $X$ and $Y$ have a strong positive relationship. Large values of $X$ will occur will large values of $Y$ and small values of $X$ will occur with small values of $Y$. This means that $(x - u_x)$ and $(y - u_y)$ will tend to be both positive or both negative, consequently $(x - u_x) * (y - u_y)$ will tend to be positive ($x$ and $y$ are realizations of the random variables $X$ and $Y$). For a strong positive relationship, $Cov(X, Y)$ should be quite positive. For a negative relationship the signs of $(x - u_x)$ and $(y - u_y)$ will tend to be opposite, yielding to a negative product. For a strong negative relationship $Cov(X, Y)$ should be quite negative. If two variables are not strongly related positive and negative products will tend to cancel one another, yielding to a covariance near 0.




(a) Positive covariance (b) Negative covariance (c) Covariance near zero

Figure 4: These images have been taken from [14]

The main limitation of the Covariance is that, it is sensible to the choice, of the units of measurement. We can solve this problem scaling the Covariance.

The correlation coefficient between two random variables $X$ and $Y$ can be defined as:

$$Corr(X, Y) = \frac{Cov(X,Y)}{\sigma_x \sigma_y}$$

where:

- $\sigma_x$ and $\sigma_y$ are the standard deviations of $X$ and $Y$.

The correlation coefficient is not affected by a linear change in the units of measurement. The strongest possible positive relationship between two random variables $X$ and $Y$, corresponds to $Corr(X, Y) = +1$, while the strongest possible negative relationship corresponds to $Corr(X, Y) = -1$. It's important to specify that $Corr(X, Y)$, is a measure of the degree of linear relationship between $X$ and $Y$. A value of the correlation index smaller than 1 in absolute value, only imply that the relation is not perfectly linear, but there may still be a strongly non linear relationship. Also a value of the correlation index equals to 0 doesn't imply that two random variables are independent, but only that there is a complete absence of linear relationship. If two variables are highly correlated, if we know one of the two features we can derive the other. This means that if we already know one of the two predictors, knowing

the other it's useless, we don't obtain more information. All the information reported in this section, describing the concepts of Covariance and Correlation, are based on [14], by Devore at al.

## 4.3 Machine Learning

Humans differently from machines can learn from past experiences. Instead to let computers do something, we need to provide them precise step by step instructions. With Machine Learning, we let computers learn from past data, improving their performances in fulfilling different tasks. As we can see from figure 5 Machine learning can be defined as an application of AI, that provides to machines, the ability of automatically learn from their experience, without being explicitly programmed to do so. The learning process begins with the observation of the input data and the identification of relevant patterns, which will allow the machine to make better decisions in the future, learning automatically, without the human intervention.

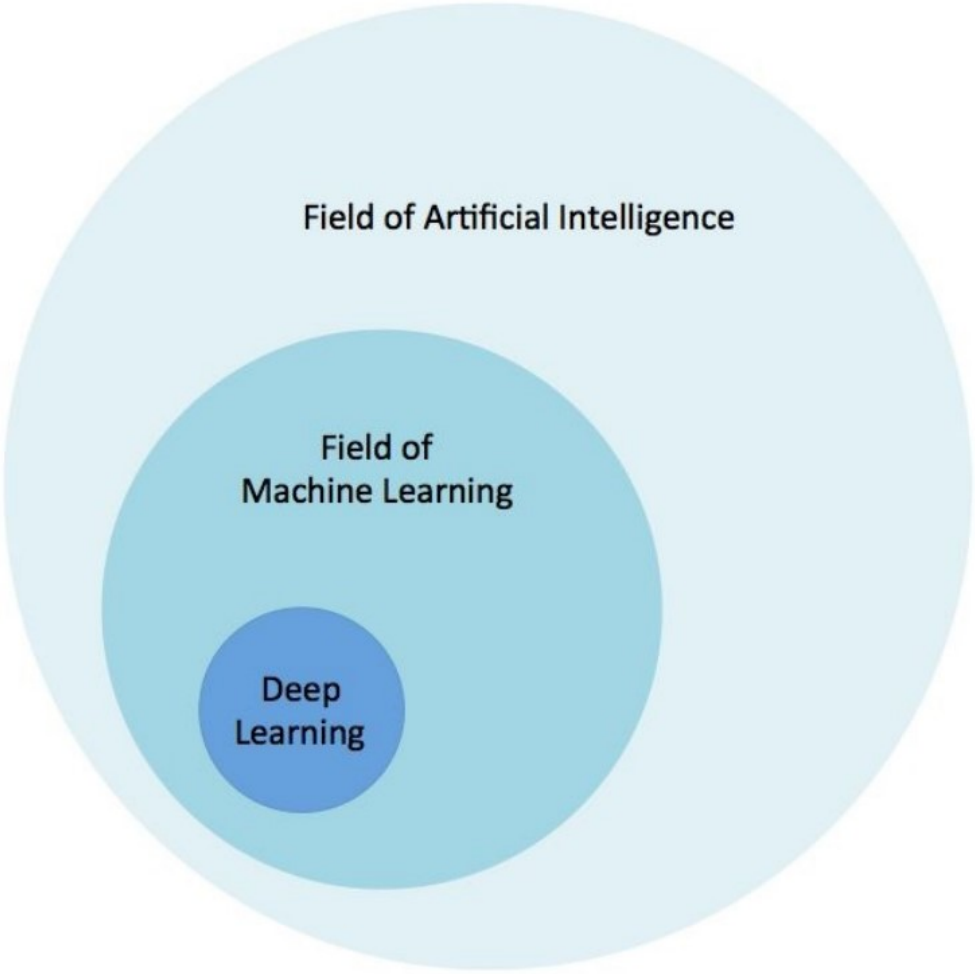



Figure 5: Scheme showing the relationship between AI, Machine Learning and Deep Learning. This image has been taken from [30]

Sections 4.3.1, 4.3.2 and 4.3.3, are based on the information reported in this book [28], by Mohri et al. Section 4.3.5, describing different feature selection approaches, is based on the information reported in [44], by Zengh et al. A more detailed description of the cross-validation techniques, described in section 4.3.4, is reported in [13], by Arlot et al.

### 4.3.1 Typical learning tasks

- Classification: The Classification task consist of assigning a category to each input entry. For example: document classification per topic or image classification.
- Regression: The Regression task consist of assigning a real value to each input entry. This is what happen for example with the prediction of stock values. With regression we can compute the penalty for a wrong prediction, as the distance to the real value.
- Ranking: The Ranking task consists of ordering a set of items, according to a specific criterion. An example is Web search.
- Clustering: The Clustering task consist of partitioning a set of items, into different sub-sets. The objects in each sub-set, will be characterized by common aspects. For example: Clustering customers, based on their purchasing behaviours.
- Dimensionality Reduction: The Dimensionality Reduction task consists of transforming the initial representation of data, into a lower dimensional one. In other words we want to reduce the number of features that characterized our entries, maintaining at the same time the most important characteristics of the original representation.

### 4.3.2 Different stages of the learning process

The first action we need to perform is partitioning the data available to us into: training, validation and test sets. Next, the most relevant features, that will characterized our data, must be selected, based on a prior knowledge about the problem in exam or a statistical analysis (for example we can analyze the level of correlation between the different features). The learning algorithm will then be trained using the training data, tuning at the same time the hyper-parameters of the model, using the validation set. The performances of the model, will finally be evaluated on the test set, which must be kept separate from the training data-set.

### 4.3.3 Different Learning Scenarios

- Supervised learning: In this scenario the training data are labeled examples and the model has no access to the test data, during training. This is the scenario that usually characterized classification and regression problems.
- Unsupervised learning: In this scenario the training data are unlabeled and the model has no access to the test data during training. Clustering is an example of unsupervised learning problem.
- Semi-supervised learning: Training data consist of both labeled and unlabeled examples and the model has no access to the test data, during training. This scenario is common, when labels are expensive to be obtained.
- On-line learning: The on-line learning scenario consists of multiple rounds. At each round the model receive a labeled training example, makes a prediction and the loss with respect to the true label is computed. The objective is to minimize the cumulative loss over all the rounds.
- Reinforcement learning: In the reinforcement learning scenario the agent actively interact with the environment and receives an immediate reward for each action. The objective is to maximize the reward over a series of actions and interactions with the environment.

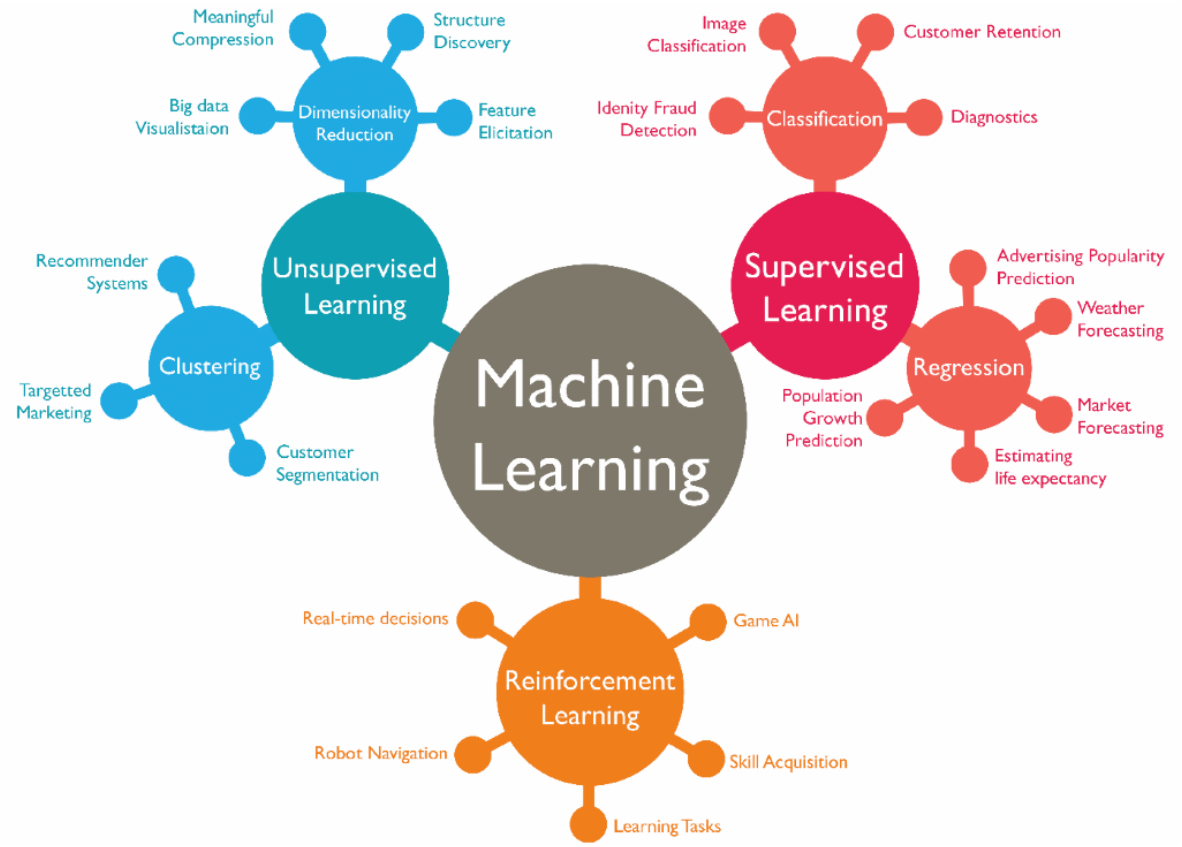



Figure 6: Scheme of the typical learning tasks, with some examples. This image has been taken from [1]

#### 4.3.4 Cross-validation

Cross-validation is a statistical method for evaluating and comparing different learning algorithms or identify the best set of hyperparameters for a specific learning technique. It consists of splitting the training data into two sets: one used for training and the other one used to evaluate the model. Usually different rounds are performed, splitting each time the training data in a different way, such that every training point has the chance to be evaluated against. The mean value of the results obtained, on the evaluation sets, over the different rounds is usually taken as indicator of the model performances. Cross-validation techniques give us a better idea of the model performances, because they take into account the variance in the results. Using Cross-validation methods we can also have a better idea about the model generalization capabilities on unseen data. It's important to specify that, the Cross-validation procedure must be performed only using the training data, keeping separate the data we want to use for testing. After having identified the best learning algorithm and the best set of hyperparamters, for that algorithm, the model is re-trained on the entire training data-set, to be finally evaluated on the test data-set. There are different Cross-validation techniques which can be divided into two groups:

1. Exhaustive Cross-validation: The train and evaluation rounds, are performed on all the possible ways, in which we can divide our training data-set, into training and evaluation sets.
2. Non-exhaustive Cross-validation: The train and evaluation rounds are not performed on all the possible ways, in which we can split our training data. Non-exhaustive methods are an approximation of exhaustive techniques.

Examples of Exhaustive Cross-Validation techniques:

- Leave p-out Cross-validation: With this techniques $p$ observations are used as validation set, while the remaining ones are used for training. This is repeated for all the possible combinations

of $p$ observations as evaluation set. This technique is computationally infeasible, also for training data-set of relatively small dimensions. The number of iterations required by the Leave p-out Cross-Validation Technique is equal to $C_p^n$

where:

- $C$ is the binomial coefficient
- $n$ is the number of samples

- Leave one-out Cross-validation: This technique is a special case of the Leave p-out Cross-validation method, with $p = 1$. This method requires a number of iteration equals to the training data-set dimension, so it's more computationally feasible, compared to the Leave p-out method, but for large training data-set, the time required can still be too much.

Examples of Non-exhaustive Cross-validation methods:

- K-fold Cross-validation technique: With this Cross-validation technique, the training data are divided into $k$ subsets. One subset is used for evaluation, while the remaining $k - 1$ folds are used for training. This procedure is repeated $k$ times, using each one of the $k$ folds, exactly once as validation set. The $k$ results are averaged, to obtain a single estimation of the model performances. The advantage of the K-fold Cross-validation technique is that, all the examples are used for both training and validation. All the examples are used for validation exactly once.

- Monte Carlo Cross-validation technique: With this technique, multiple random splits of the training data, into training and evaluation sets are generated. For each split the model is trained on the training set and evaluated on the validation set. The results obtained over the different splits are averaged, to obtain a single estimation of the model performances. The advantage of this method, with respect to the K-fold Cross-validation technique, is that the proportion between training and validation examples, is not dependent on the number of iterations. The disadvantage of this method is that, some of the examples may never be selected to be part of the validation set, while others may be selected more than once. In other words the validation subsets may overlap.

A graphical comparison of the K-fold and Monte Carlo techniques, is shown in figure 7

#### 4.3.5 Feature Selection

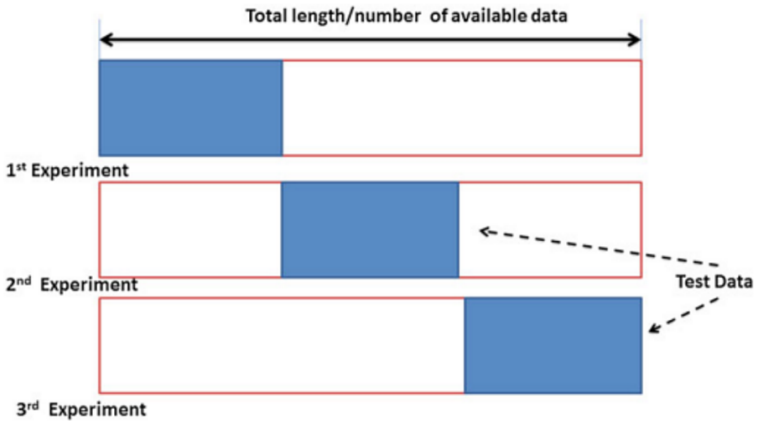
Feature Selection is the process of reducing the number of predictors, that characterized the data, in input to our model. This process is required, to reduce the computational cost of the model and in some cases to improve the performances. Through Feature Selection, we can identify the features, that are more useful, to predict the target variable. At the same time, through Feature Selection, we can eliminate the predictors, that are not relevant for our learning task. These types of predictors can in some cases degrade the model performances. Feature selection methods can be divided into two classes:

1. Unsupervised methods: This type of methods select features, ignoring the target variable.

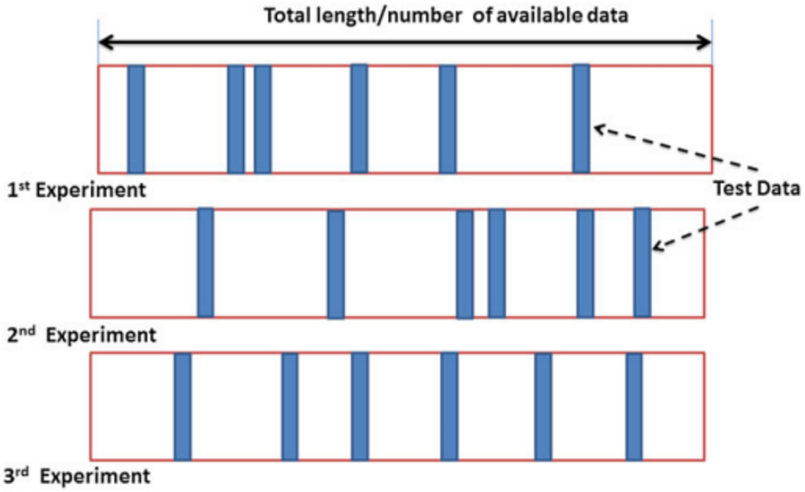
2. Supervised methods: This type of methods select features, based on the target variable.

An example of Unsupervised technique is the one, based on the level of correlation between input features. We can compute the pairwise correlation index between each couple of input features, to then drop one of the two features, involved in the highly correlated couples. If two features are highly correlated it means that, they are bringing redundant and not useful information for our problem in exam. A description of the Pearson correlation coefficient, one of the most used correlation index, is reported in A. The Supervised Features Selection techniques can be divided in:

1. Filter methods: This type of methods are generally used as pre-processing steps. The feature selection in this case, is independent from the Learning Algorithm we are using. Features are selected, based on their scores in specific statistical tests, about the input variables' relationship with the target variable. An example of this tests are the Anova or the Chi-Squared tests.




(a) Graphical representation of K-fold technique




(b) Graphical representation of Monte Carlo Cross-validation technique

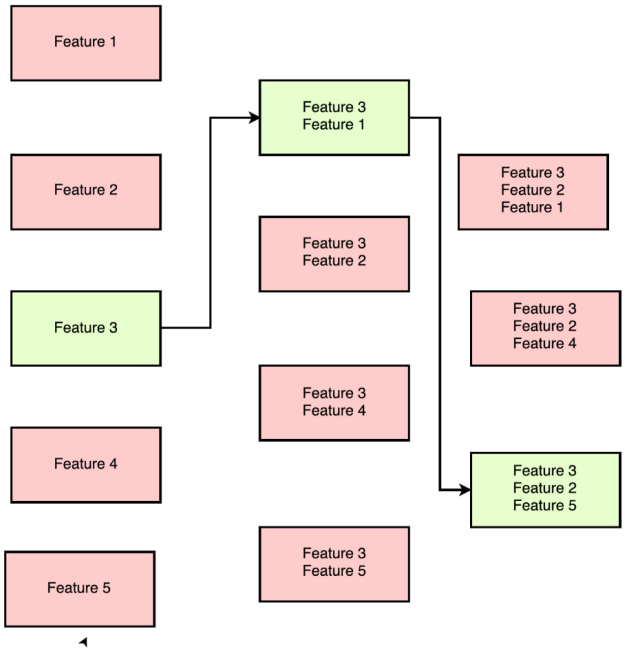
Figure 7: Comparison between Monte Carlo and K-fold Cross-validation techniques. These images have been taken from [33], by Renji Remesan et al.

2. Wrapper methods: With this type of methods our feature selection problem is essentially reduced to a search problem. The model is run, using different subsets of the Input features, and the best set of input variables is identified looking at the model's performances. This type of techniques are computationally expensive. The most common Wrapper Methods are:

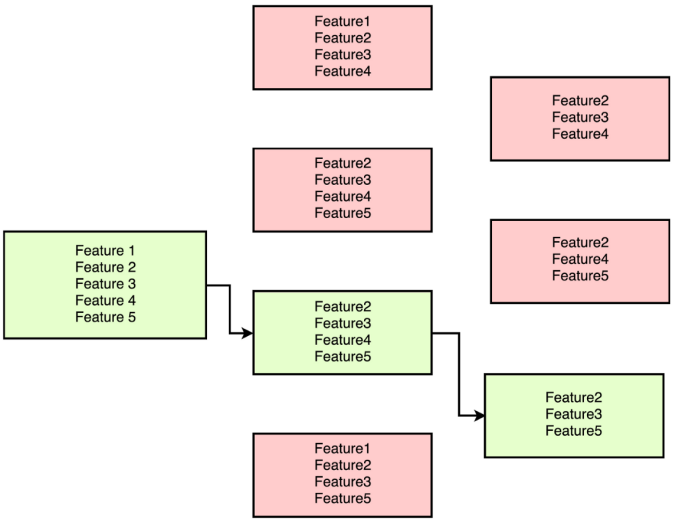
    - Forward Selection: With the Forward Selection Approach, we start with an empty set of features. At each forward step the variable, that has given the best improvement in the model's performances is add to the set of the best features. This procedure goes on until we reach the desired number of features or until adding a new variable, we don't notice an improvement in the model performances.
    - Backward Selection: With the Backward Selection technique, we start from a set containing all the input features. At each step, we try to remove, one at a time, all the variables. The variable, which removal gives us the best improvement in the model's performances, is removed from the set of best predictors. This procedure goes on until the desired number of predictors is reached or we notice no improvements in the model performances.

3. Intrinsic Feature Selection: Some Learning Models perform feature selection automatically as part of the learning procedure. Examples of this models are Lasso and Decision Trees.

Feature Selection is obviously related to Dimensionality Reduction. The objective of both Feature Selection and Dimensionality Reduction techniques is to reduce the number of input variables. The difference is that: Feature Selection Techniques select which features, we have to keep or remove, in order to improve the model's performances, while Dimensionality Reduction techniques like PCA or LDA, define a new data projection, resulting in a completely different set of features.




(a) Graphical representation of the Forward Selection technique




(b) Graphical representation of the Backward selection technique

Figure 8: Graphical comparison between the Forward and Backward Selection, showing the model's runs performed at each step. These images have been taken from [2]

## 4.4 Feed Forward Neural Networks

Neural Networks are a part of the Machine Learning discipline, inspired by how the human brain works and especially by how neurons share information, in the form of electrical signals. Feed Forward Neural Networks are the simplest form of Neural Networks. The goal of a Feed Forward Neural network is to approximate an objective function $y = f^*(x)$, mapping the inputs $x$ to the output $y$. To be more precise a Feed Forward Neural Network defines a mapping $y = f(x, \theta)$ and learns the values of the parameters $\theta$, that best approximate the objective function. A Feed Forward Network can be composed by many layers and each layer can present many nodes, so many inputs and outputs. Even though each node, implement a relative simple transformation, the function represented by the entire network can become arbitrary complex. The name Feed Forward refers to the fact that, in Feed Forward Neural Networks, the information moves in only one direction from the input nodes, through the hidden nodes (if they are present) and finally to the output nodes. The connections between nodes don't create cycles.

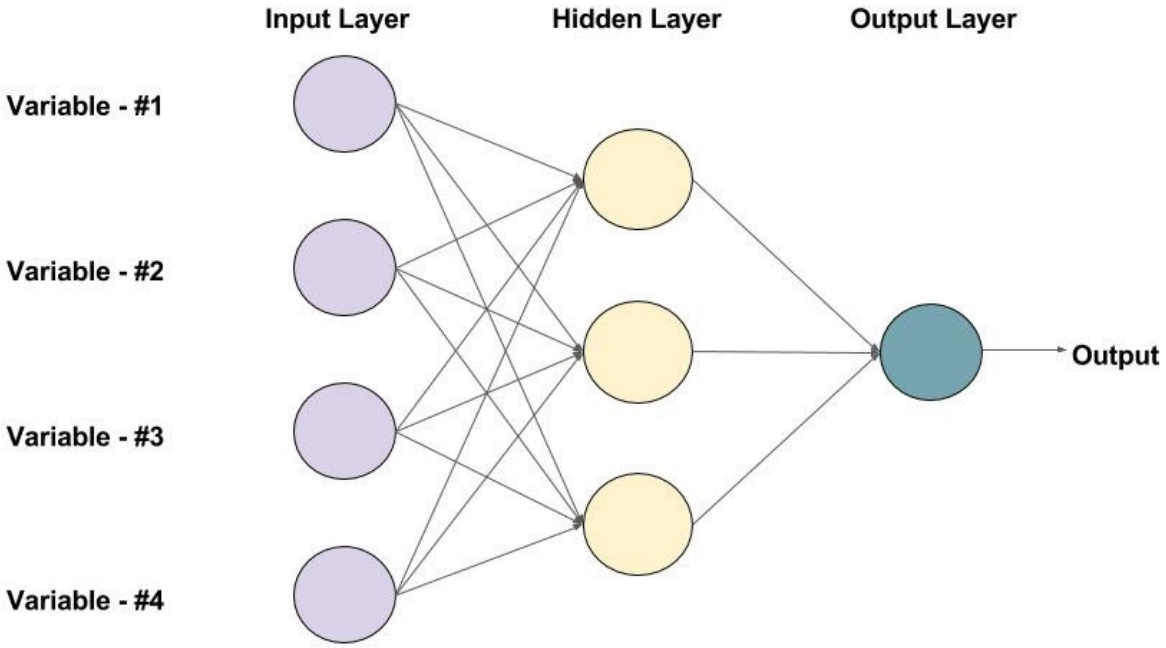



Figure 9: Example of a Feed Forward Neural Network architecture, this image has been taken from [3]

All the information reported in sections 4.4.1, 4.4.2, 4.4.3 and 4.4.4, referring to Feed Forward Neural Networks and their learning procedure, are based on the books: [30] and [16], by Patterson et al. and Goodfellow et al. respectively. Detailed information about the Stochastic Gradient Descent technique are reported in [21], by Ketkar et al. A comparison of the most important activation functions for deep learning can be found in [29], by Nwankpa et al.

### 4.4.1 Perceptron

The Perceptron is a linear binary classifier. The Single Layer Perceptron can be considered as the simplest form of Feed Forward Neural Networks. As we can see from figure 10, in the Perceptron, the input to the activation function is obtained, applying the following formula:

$\sum_{i=1}^{N} x_i w_i$, where $x_i$ is the $i^{th}$ position of the input vector and $w_i$ is the corresponding weight.

Typically with the Perceptron the activation function is a Heaviside step function (a definition of the Heaviside step function is reported in D) with a threshold of 0.5, so in output we can have two possible values 0 and 1, depending on the input. The value in output represents the class assigned to the input entries. We also have a bias term, which doesn't depend on the inputs and in figure 10 is associated with the weight $w_0$. The bias term can move the decision boundaries.

The Perceptron training procedure follows the following steps:

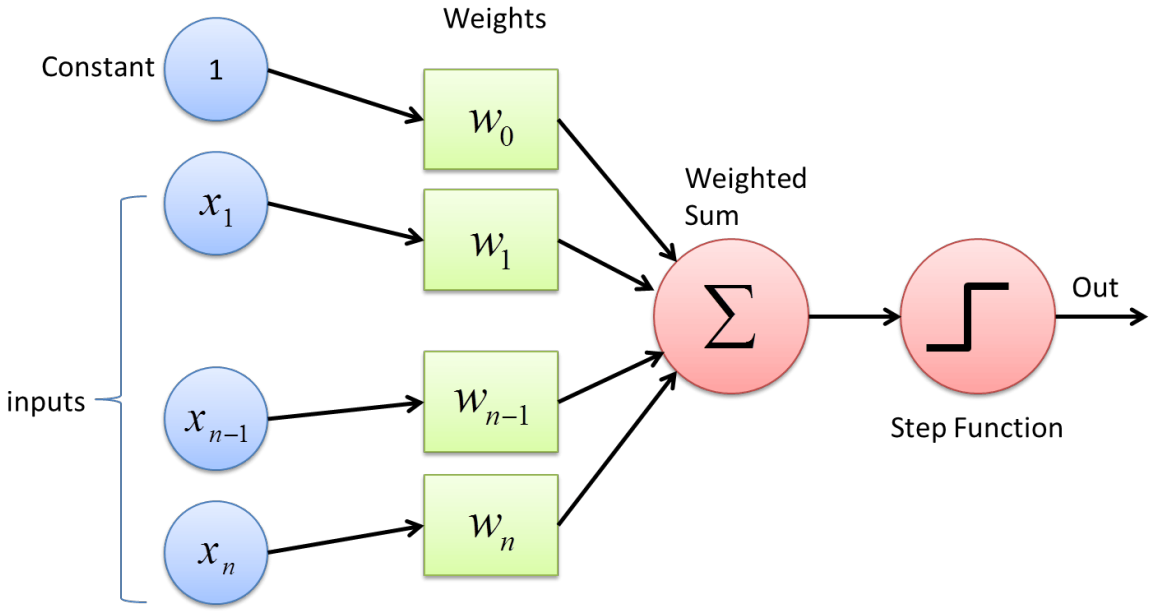



Figure 10: Single layer Perceptron. This image has been taken from [4]

1. The weights are initialized to a small random number or to 0.
2. For each example $j$ in our training data-set $D = \{(x_1, d_1), ..., (x_s, d_s)\}$, where $x_j$ is the $j^{th}$ n-dimensional input vector and $d_j$ is the desired output:
   (a) The Perceptron output is computed as:

   $y_j(t) = f(w(t) \cdot x_j)$, where $f$ is the activation function and the symbol $\cdot$ represents the dot product between $w(t)$ and $x_j$.

   (b) The connections' weights are updated as:

   $w_i(t+1) = w_i(t) + r(d_j - y_j(t))x_{j,i}$ for $0 <= i <= n$ where:
   $r$ is the learning rate, a constant hyper-parameter, while $x_{j,i}$ the $i^{th}$ feature of the input vector $x_j$

The Perceptron's training procedure goes on until all the training examples are correctly classified. It's important to specify that if the training data are not linearly separable the training algorithm will not converge. The Perceptron is able to classify only linearly separable data.

### 4.4.2 Multi-layer Feed Forward Neural Network

The structure of a multi-layer Feed Forward Neural Network consists in one Input Layer, one or more Hidden Layers and one Output Layer. Each layer can present many neurons, which are similar to the single layer Perceptron described in section 4.4.1. The only difference is the activation function, which can change, depending on the purpose of the layer.

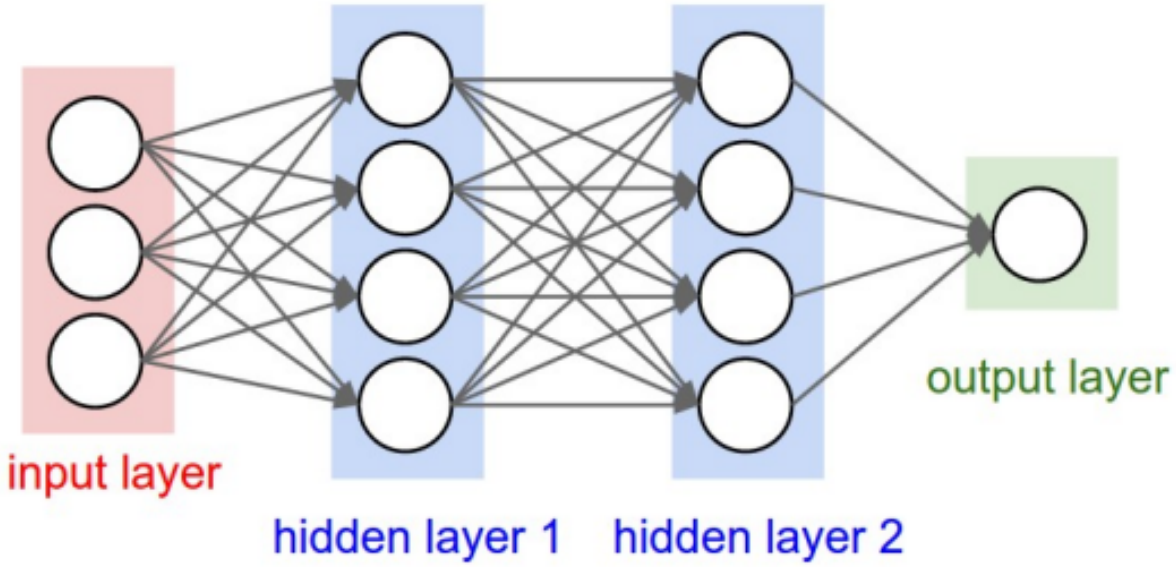



Figure 11: Example of a Multi-layer Feed Forward Neural Network architecture, this image has been taken from [5]

As we can see from figure 11, the neurons in each Layer are connected to all the neurons in the the adjacent layers, this type of architecture is called Fully Connected. In input to the Input Layer we have the raw vectors, while in input to all the other layers, we have the output of the other layers' neurons. It's important to remember that, the output of each layer, pass through an activation function, which can be different, depending on the purpose of the layer. The number of neurons in the input layer is the same as the number of features, that characterized the input data. Like the Perceptron, to each connection is assigned a weight and each layer also present a bias term. Weights and bias terms are progressively changed during the training procedure, in order to approximate the best solution. The Hidden Layers are the key factor, that allows Multi-layer Feed Forward Networks to model non linear function, overcoming the main limitation of the single layer Perceptron, which is able to classify, only linearly separable data.

### 4.4.3 Back-Propagation

The learning procedure for Feed Forward Neural networks consists in progressively adjusting weights and biases, allocating more importance to certain information, minimizing at the same time, the others. In this way the model is able to learn which predictors are related to which outcomes, to then adjusts weights and biases, accordingly. One important aspect of the Feed Forward Networks' learning procedure is the Back-Propagation algorithm. The term Back-Propagation refers to the method used for computing gradients in Feed Forward Neural Networks. The basic idea of the algorithm is that, the partial derivatives of the cost function $J$, with respect to the network's parameters $\theta$, can be decomposed recursively, considering the composition of functions that relate $\theta$ to $J$. The cost function $J$ represents the amount of error made by the network, with respect to the ground truth. Different types of loss function can be used, based on the learning task we are working on. The partial derivatives express how, changing a parameter, the cost function $J$ is affected. In this way we can understand how to update the network's parameters, in order to minimize the cost. The basic mathematical tool, to consider derivatives through a composition of functions is the Chain Rule:

Let's take into consideration the variable $x$, which influences $y$, which influences $z$. We are interested in how a small change in $x$ propagates to a small change in $z$, through a tiny change in $y$. In our case $z$ is the objective function $J(g(\theta))$, $x$ are some parameters $\theta$ and we have intermediate quantities $y = g(\theta)$, which are the network's intermediate activations (the outputs of the intermediate layers). The gradient of interest can be decomposed as:

$$\nabla_\theta J(g(\theta)) = \nabla_{g(\theta)} J(g(\theta)) \frac{\partial g(\theta)}{\partial \theta}$$

Where $\nabla_\theta J(g(\theta))$ is the gradient, with respect to some parameters $\theta$.
The decomposition works also if J, g or $\theta$ are vectors, rather than scalars, in this case the partial derivatives are Jacobian matrices of the appropriate dimensions.

A small change in $\theta$, must be multiplied by $\frac{\partial g(\theta)}{\partial \theta}$, to get the corresponding small change in $g(\theta)$ and a small change in $g(\theta)$, must be multiplied by $\nabla_{g(\theta)} J(g(\theta))$ to get the corresponding small change in $J(g(\theta))$. The ratio of the change in $J(g(\theta))$, corresponding to the change in $\theta$ is the product of the partial derivatives. This interpretation is true in the purely scalar case. If g is a vector we can rewrite the above equation as:

$\nabla_\theta J(g(\theta)) = \sum_i \frac{\partial J(g(\theta))}{\partial g_i(\theta)} \frac{\partial g_i(\theta)}{\partial \theta}$

Recursively applying the chain rule, we are able to compute the gradient of the cost $J$, with respect to each parameter of the network. First we compute the gradient of the cost $J$, with respect to the output neurons. These gradients are then used to compute the gradient of $J$, with respect to the last hidden layer neurons, which directly contribute to the output. We can continue this procedure, computing the gradient of $J$, with respect to the weights of each layer of the network.

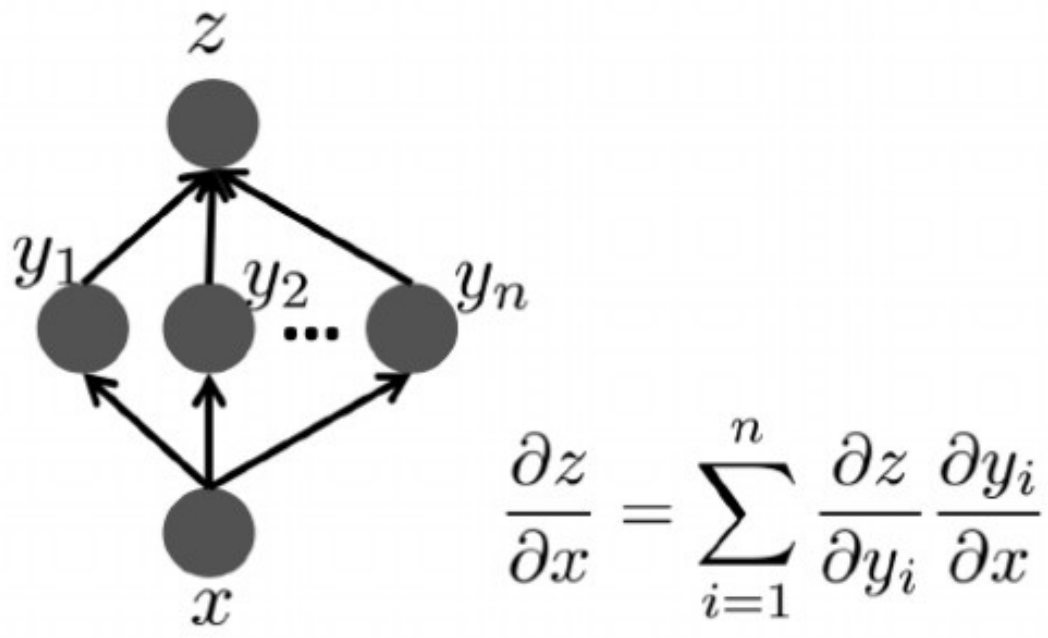



Figure 12: Example of the chain rule application. This image has been taken from [16]

#### 4.4.4 Stochastic Gradient Descent

After having described how the gradient of the cost function, with respect to the network parameters is computed, we can now focus on optimization algorithms, describing in particular the SGD algorithm. The optimizer we decide to use defines: how to progressively change our Network's weights, to reduce the loss. Optimization algorithm are responsible for guiding the Network to the optimum solution. The standard Gradient Descent technique evaluates the cost function and the corresponding gradient, for all the training example at the same time. For really large data-set, this approach is not feasible in terms of memory. With the Stochastic Gradient Descent technique the gradient of the cost function, with respect to the network parameters, is computed over one or a few training examples and the parameters' updates are computed in the following way:

$$\theta = \theta - \alpha \nabla_\theta J(\theta, x^{(i)}, y^{(i)})$$

Where $(x^i, y^i)$ are one or a batch of training examples, with the corresponding labels, which represent the expected outputs. Generally is preferable to use a batch of training example for the update, rather than only one example. Using a batch of training example, we can reduce the variance in the parameters updates, reaching a more stable convergence. The parameter $\alpha$ is called learning rate and represents the magnitude of the updates. The initial learning rate's value is choose a priori, and is then modified following a certain strategy during the training procedure. The optimization procedure goes on for a certain number of epochs (one epoch corresponds to the analysis of the whole training data-set) and the learning rate is reduced approaching the optimum. One of the limitations of the Stochastic Gradient Descent approach is that it can stuck in local minima. We can use other optimizers like AdaGrad, which use adaptive learning rate for each weight, or Adam.

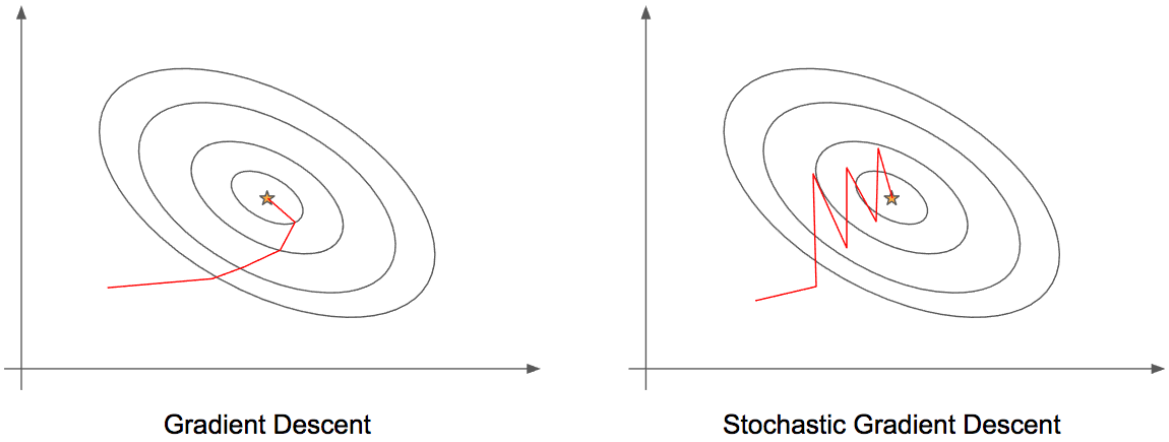



Figure 13: Differences in how standard and stochastic Gradient Descent approach to the minimum of the loss function. This image has been taken from [6]

### 4.4.5 Activations Functions

An activation function defines how the weighted sum of the inputs in a neuron is transformed, into the node's output. The choice of the activation functions in a Neural Network has a large impact on the network's performances and different activations functions may be used in different parts of the network. Usually all the Hidden Layers use the same activation function, while the Output Layer uses a different activation function, which depends on the predictive task the network has to perform. Another thing we need to specify is that, the activation functions are usually differentiable (the first order derivative can be computed given the inputs). This is required by the Back-Propagation learning procedure, described in sections 4.4.3 and 4.4.4. Usually a differentiable non-linear function is used for the Hidden Layers. The Network is, in this way, allowed to learn more complex functions, than the ones a Neural Network can learn using only linear activation functions.
The most common activation functions for the Hidden Layers are:

1. ReLu: The Rectified Linear unit (ReLu) activation function is a piecewise linear function. If the input is positive, the ReLu's output is the input itself, while if the input is negative the ReLu's output is zero. The ReLu activation is now, the most used activation function for Neural Networks' hidden layers. It is easy to implement and is less susceptible to the Vanishing Gradient problem (see section 4.6.1), compared to other activation functions like the Sigmoid or the Tanh.

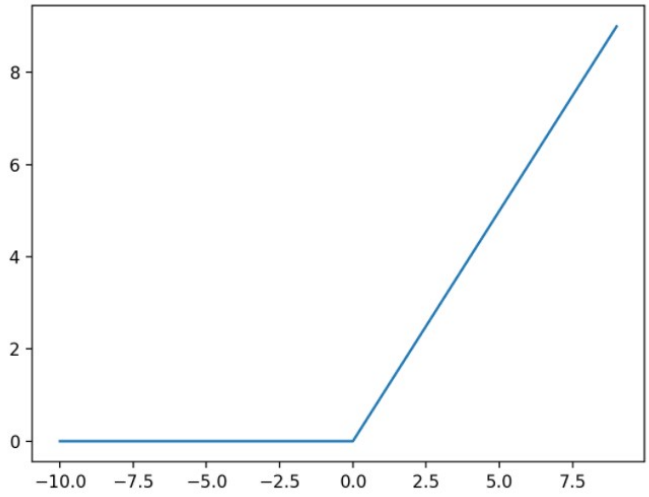



Figure 14: Plot of the ReLu's input/output.This image has been taken from [7]

The ReLu activation function has some limitations, like for example the ones concerning the saturated or "dead" units. If the weighted sum of the inputs to a node is always negative, the ReLu activation function will always output zero. This means that the node's weights will not update, slowing down the training process. To overcome this problem some variants of the ReLu have been proposed.

2. Leaky ReLu: Leaky ReLu or LReLu is a variant of the ReLu activation function, which allows small negative values in output, when the input is negative, mitigating, in this way, the "dead" units problem.

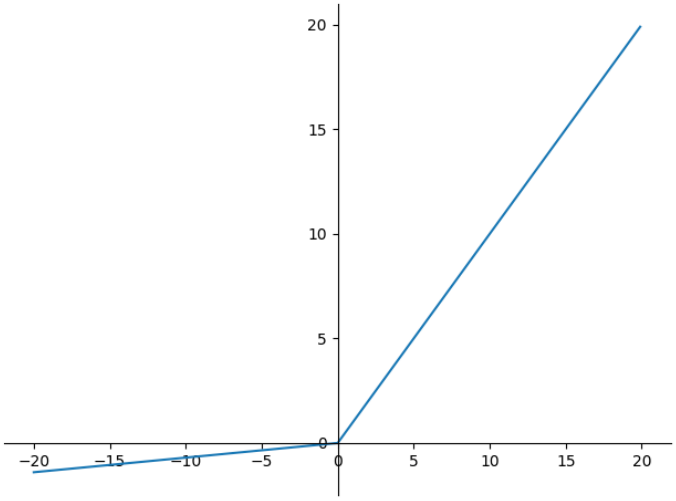



Figure 15: Plot of the LReLu's input/output. This image has been taken from [8]

3. Sigmoid: The Sigmoid activation function can take any real value in input, producing in output a number between 0 and 1. Larger is the value in Input, nearer will be the output to 1, smaller is the value in input nearer will be the output to 0. The sigmoid activation function is defined by the following formula:

$sigmoid(x) = \frac{1}{1+e^{-x}}$

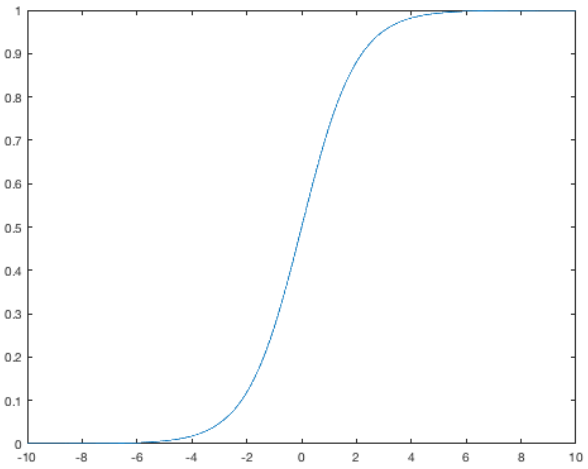



Figure 16: Plot of the Sigmoid input/output. This image has been taken from [9]

4. Tanh: The Iperbolic Tangent activation function is similar to the sigmoid. They have the same s-shape form. The Tanh activation function can take in input any real value, producing in output a value between -1 and 1. The Tanh activation function is described by the following equation:

   $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

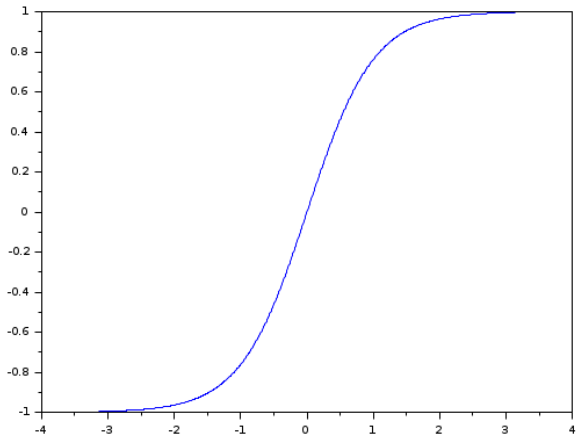



Figure 17: Plot of the Tanh input/output. This image has been taken from [10]

The most common activation functions for the Output Layers are:

1. Linear Activation Function: The Linear Activation Function doesn't change at all the weighted sum of the inputs to a node. The Linear Activation Function produces in output the same value received in input.
2. Sigmoid Activation Function: The Sigmoid Activation Function, can be used, also for the Output Layer. The Sigmoid is used in output from our Network, when we are dealing with a binary

classification problem. The input examples will be assigned to class "0" or "1", based on the Network's single prediction, which will pass through the Sigmoid function.

3. Softmax Activation Function: The Softmax Activation Function is used in output from our Network, when we are dealing with multi-class classification problems. The Softmax function takes in input a vector of real numbers and produces in output a vector of the same length, containing positive values, that sum to 1. The Softmax's output can be interpreted as the probabilities for an input example, to belong to each one of the classes, that define our classification problem. The Softmax Activation Function is defined by the following formula:

   $Softmax(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$

   where:

   - $x$ is a vector of real values of dimension $K$.
   - $x_j$ is the $j^{th}$ position of $x$, $Softmax(x)_j$ is the $j^{th}$ position of the output vector. Also the output vector has dimension $K$.

## 4.5 Convolutional Networks

Convolutional Neural Networks (CNNs) are Deep Learning architectures, well suited for image analysis and recognition. They can be used also in other fields, like sentiment or sound analysis. CNNs are able to learn higher-level features from the input data, through a series of convolutional operations. The biological inspiration for Convolutional Neural Networks is the visual cortex organization: individual neurons respond to stimuli only in a restricted region of the visual field, called Receptive Fields. Such fields overlap, in order to cover the entire visual field. Before describing CNNs' architecture more in details, we have to ask ourselves "Why we can't use traditional Multi-Layer Feed Forward Neural Networks, to analyze and classify images?". All the informations reported in sections 4.5.1, 4.5.2, 4.5.3, 4.5.4, 4.5.5 and 4.5.6, concerning different aspects of CNNs and CNNs' architecture, are based on the books [30], [16], by Patterson et al. and Goodfellow et al. respectively. A description of the Dropout technique explained in section 4.5.7, is reported in [40], by Hinton et al.

### 4.5.1 Limitations of standard Multi-Layer Networks for image classification

Standard Feed Forward Multi-Layer Networks take as input a one-dimensional vector. Images are three dimensional structures, characterized by specific width, height and depth (usually the depth of an image is 3, corresponding to the RGB channels). If for example we convert an image from the CIFAR-10 dataset, which has a dimension of 32*32*3 pixels into a one dimensional structure, we obtain a vector characterized by a length of 3072. Feeding this vector into a standard Feed Forward Multi-Layer Network, every neuron of the first hidden layer, will be associated to 3072 different weights. As we can see standard Multi-Layer Feed Forward Networks don't scale well with images. Moreover converting an image to a one dimensional structure we lose the spatial relationships, present among its pixels. Standard Feed Forward Multi-Layers Networks are not able to capture the spatial relationships in the input images. For this reasons their performances in classifying complex, high resolution images, are really poor.

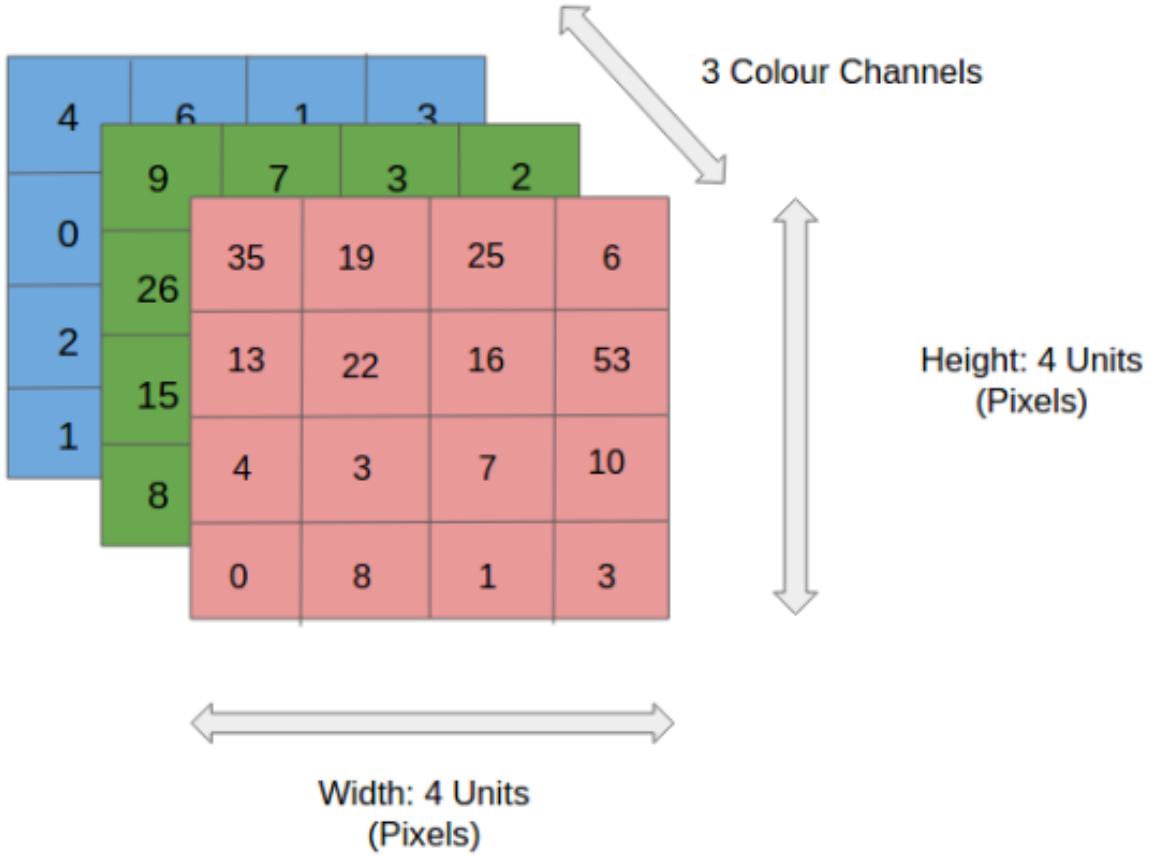



Figure 18: 4*4 RGB image with three channels. This image has been taken from [11]

### 4.5.2 Genearal structure of a CNN

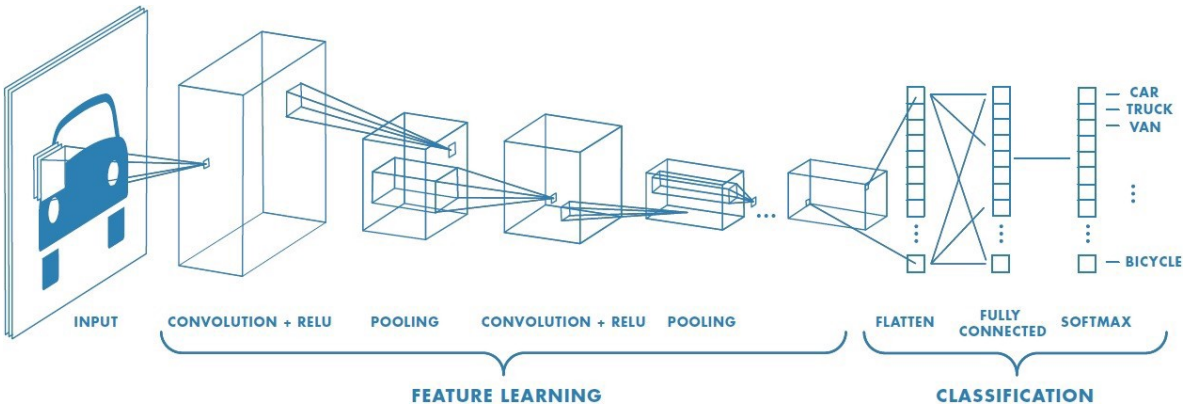



Figure 19: General Structure of a CNN. This image has been taken from [11]

As we can see from figure 19, the structure of a CNN usually consists in three main parts:

1. Input Layer
2. Feature Learning part
3. Classification part

The Input Layer is able to accept three-dimensional inputs, characterized by specific width, height and depth (the depth corresponds, to the colour channels). The inputs can be four-dimensional,

when images are collected in mini-batches for training (the fourth dimension is the batch size). The layers in the Feature Learning part follow a specific pattern: a convolutional layer, followed by a ReLu activation function, followed by a pooling layer. The feature learning part is responsible for the progressive extraction of the higher-level features, that will characterized the input image. These features will be used by the classification part of the CNN, to generate in output the classes' scores, that can be converted to probabilities, using a Softmax Activation Function (see section 4.4.5). Based on these scores or probabilities the input image will be classified in one of the output classes. The classification part consists in one or more Fully Connected Layers. In this section I want also to specify that, differently from standard Feed Forward Networks, which are Fully Connected in CNNs neurons in a layer are connected to only a small region of neurons in the layer before (except for the layers, that constitute the classification part of the network, which are Fully Connected). After having described the general structure of a CNN, we can now talk about all the different layers individually.

### 4.5.3 Convolutional Layer

A Convolutional Layer receives in input a raw image or the output, produced by the previous Convolutional Layer. The most important parameter of a Convolutional Layer, is the Kernel/Filter, which can be seen as a matrix of lower dimensions, compared to the input image. The Filter slides over the input image, performing every-time the operation showed in figure 20, between the kernel itself, and the portion of image covered by the filter. The filter moves to the right, by a number of positions equal to the Stride parameter, untill it covers the entire width of the image. Then it hops down, by a number of positions again equals to the Stride parameter, returning at the same time to the left of the image and the process is repeated. This procedure goes on until the entire image is covered.

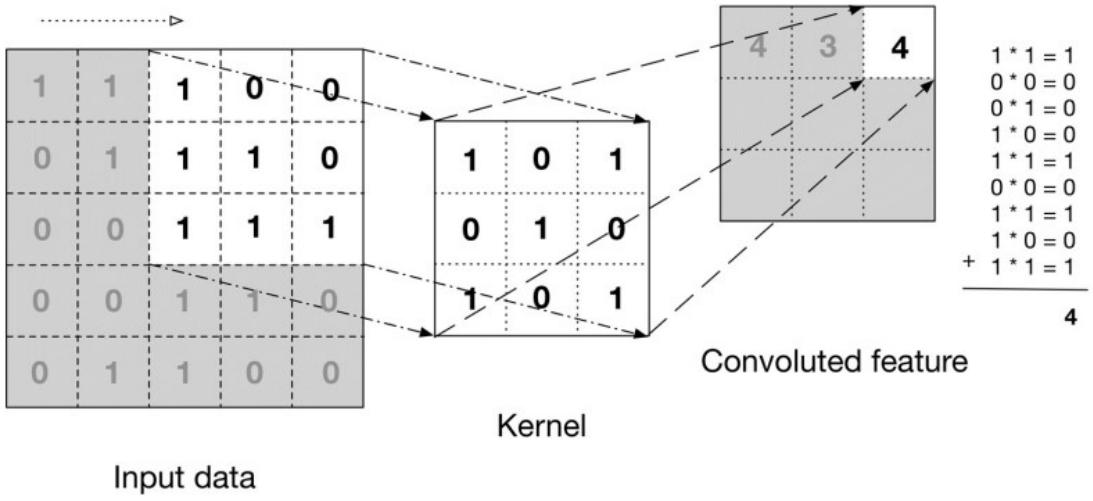



Figure 20: Example of the convolutional operation. This image has been taken from [30]

If the image in input to the convolutional layer has more than one channel, the kernel will have the same depth. The convoluted features obtained, for the different channels will be summed, to generate a one-depth output. The result of the convolutional operation, just described is called activation map. We will have one activation map for each Filter of the Convolutional Layer. This means that the depth of the output of a Convolutional Layer, is the same as the number of Filters of that layer. CNNs usually have more than one Convolutional Layer. The first Conv. Layers usually extract low-level features like edges or colors. The other Conv. Layers extract higher-level features, which will give us a better understanding of the images in our data-set. We need to specify that the output of a Convolutional layer, before reaching the Pooling Layer pass through a ReLu activation function.

### 4.5.4 Pooling Layer

The operation performed by the Pooling Layer, is similar to one performed by the Convolutional Layer, but the scope is different. The objective of the Pooling Layer is reducing the dimensionality of the

output of the preceding Convolutional Layer. There are two different type of Pooling:

1. Max Pooling
2. Average Pooling

The kernel of a Pooling Layer, performing a Max Pooling operation, slides over the input image, selecting every time the maximum value of the portion of image covered by the filter. Instead the kernel of a Pooling Layer, performing an Average Pooling operation, slides over the input image returning every time the average, of the values of the portion of image covered by the filter. This dimensionality reduction operation is performed to decrease the computational cost, necessary to analyze the data, extracting at the same time the most important features.

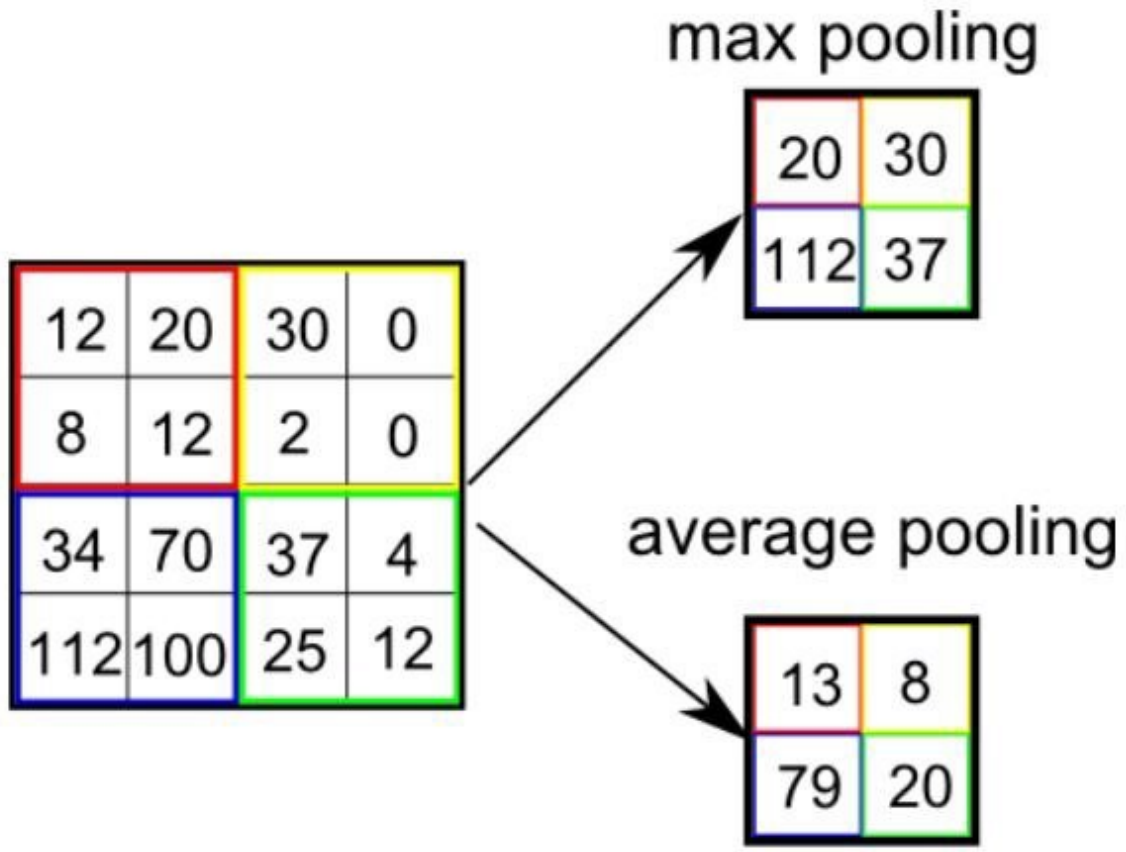



Figure 21: 2 different Pooling techniques. This image is from [11]

#### 4.5.5 Batch Normalization Layer

The Batch Normalization Layers, normalizes the activation (the output, which pass through the ReLu activation function) of the previous layer at each batch, keeping the mean close to 0 and the standard deviation close to 1. Normalization, directly inserted in the CNN architecture, through Batch Normalization Layers, allow us to speed up training. Batch Normalization Layers also reduce the sensitivity of the CNN, about weights' initialization and mitigate the vanishing gradient problem (see section 4.6.1).

#### 4.5.6 Fully Connected Layer

After the feature extraction procedure the High-level features, extracted from the feature learning part of the CNN, are flattened into a column vector. This column vector is fed into a standard, fully connected Multi-Layer Feed Forward Network, which will learn a non linear combination of that features, in order to classify in the proper way the input images. In output from the network we have the classes' scores, and using a Softmax Activation function for a multi-class problem or a Sigmoid for

a binary classification problem, we can obtain from these score the probabilities for the input images to belong to each class.

### 4.5.7 Overfitting problem and Dropout method

In general in Machine Learning overfitting occurs, when the model we are working with presents a low error on the training data-set and a high error on the test data-set. If we incur in the Overfitting problem, it means that, our model is too complex for the problem we are considering. The model fits too much the characteristics of the training data and is not able to generalize. Talking about Neural Networks this can happen for example, when large Neural Networks are trained using a small training data-set. One of the technique, used to prevent overfitting with Neural Networks is called drop-out and consists in ignoring, some randomly selected nodes of a certain layer, at each training step. In this way each update of that layer during training, is performed using a different and random view of the layer. This makes the training procedure more noisy and at the same time the model more robust. Usually with CNNs, the Dropout method is applied only on the Fully-Connected Layer.

## 4.6 Residual Neural Network

All the information reported in sections 4.6.1, 4.6.2, 4.6.3, and in this section, are based on [19], [20], by He et al. The AlexNet convolutional Architecture was the first one, to be more successful, than traditional hand-crafted feature learning, in the ImageNet competion (see [34]), in 2012. AlexNet consists of eight Layers, five Convolutional Layers and three Fully Connected Layers. This architecture is the foundation of the standard CNN architectures, described in 4.5.2. What makes the CNN architecture so powerful, is the idea that, adding more Convolutional Layers, the network will be able to progressively learn more complex features, improving the performances in classifying the input images. However He et al. in [19] has shown that, there is a maximum threshold for the depth of the traditional CNN architecture.

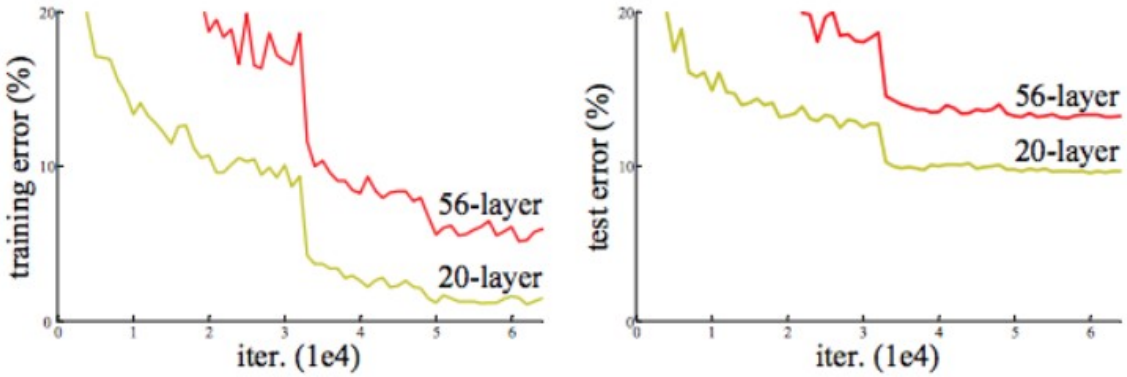



Figure 22: Comparison between the performances of a shallow and a deeper, standard CNN architectures. This image has been taken from [19]

Looking at figure 22 from [19], we can see that the test error of a CNN architecture with 56 layers is higher, compared to the test error of a 20 layers architecture. A deeper architecture is able to estimate a more complex function, so we can think that, we are incurring in an overfitting problem, which can be solved using for example the Dropout technique (see section 4.5.7). The problem is that, not only the test error is higher, using a 56 layers architecture, but also the training error. This means that the reason for the decreasing of the performances, is not the overfitting. One of the reason for the decreasing of the performances, using deeper architectures can be the so called Vanishing Gradient Problem. Another more general problem of deeper architectures, is the so called Degradation Problem: As the architectures get deeper and deeper, it becomes more difficult for the information, to be propagated from the shallower layer, to the deeper ones.

### 4.6.1 Vanishing Gradient Problem

Neural Networks' architectures, characterized by a large number of layers, presenting certain activation functions, can incur in the Vanishing Gradient Problem. Let's take for example into consideration, the Sigmoid Function (see section 4.4.5). This activation function squishes a large input space, into a small output space. This means that, a large change in the input, will correspond to a small change in the output, so the derivative will be small. Gradients in a Neural Network are found using the Back-propagation technique (see section 4.4.3), starting from the final layers to arrive to the initial ones. According to the chain rule the derivatives of each layer are multiplied through the network, from the final layers to the initial ones, to obtain the gradient of the loss function with respect to all the parameters of the network. If we have a large number of layers characterized by an activation function like the Sigmoid, a large number of small derivatives will be multiplied together. This means that, the gradient will exponentially decrease, approaching to the initial layers of the network. A small gradient means that, the weights and biases of the initial layers will not be updated efficiently during the training procedure, or will not be updated at all. One of the possible solution of the Vanishing Gradient Problem is to use an activation function like the ReLu, which doesn't generate a small derivative. Batch Normalization Layers can also mitigate the problem, normalizing the input. Another solution is the one introduced by Residual Networks architectures.

### 4.6.2 Residual Learning

An intuitive idea to solve the Degradation Problem of deeper architectures, could be to connect directly the initial layers to the deeper ones, through an identity mapping. In this way the information is passed directly, through an identity function, bypassing the intermediate layers. Let's define $g(x)$ as the original function a layer has to learn and let's define $h(x) = g(x) + x$, as the new function the layer has to learn, in the Residual Learning Scenario. The $+x$ term represents the so called skip connection, which brings the original value. The layer, using the skip connection, in the Residual Learning Scenario, has only to learn the residue of the input value $x$ or in other words, how to change the input value $x$. Using the Residual learning Approach, is really easy for the layer to learn the identity function, it only needs to bring all the weights to zero ($h(x) = 0 + x = x$). Without skip connections, all the weights and biases have to be modified to learn the identity function from scratch, and this is really difficult for the network. Using Residual Learning we can overcome the Degradation Problem of deeper architectures.

### 4.6.3 The residual block

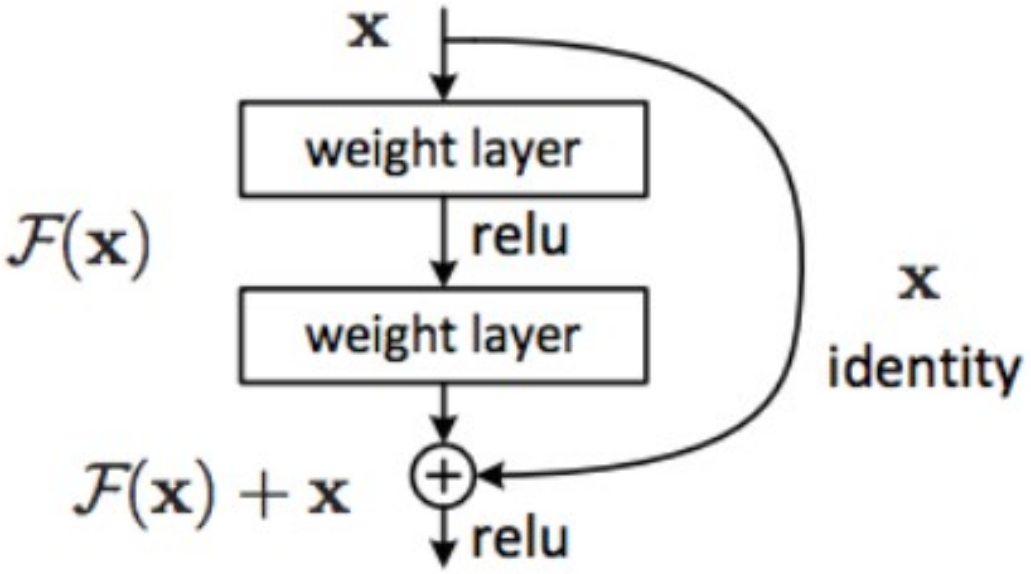



Figure 23: Residual block. This image has been taken from [19]

Figure 23 shows the new layer introduces by the Residual Network architecture. The skip connection of the Residual Block has no parameters, and is just there to add the output from the previous layer to the layer ahead. The skip connection helps to bring the identity function to deeper layers, allowing us to train much deeper architectures. Looking at the structure of the residual block we can notice two things:

1. The addition is performed before the ReLu activation function. This is done because, if we perform the addition after the ReLu, the Residual Layer will learn only positive increments, decreasing a lot the learning capabilities.

2. The Skip Connection is add after two weights layers. This is done because, if we add the skip connection after one weight layer before the ReLu, we obtain a simple linear function, equivalent to just one weight layer, so there is no point in adding a skip connection. We need at least one non-linearity before adding the skip connection, for this reason we add it after two weight layers.

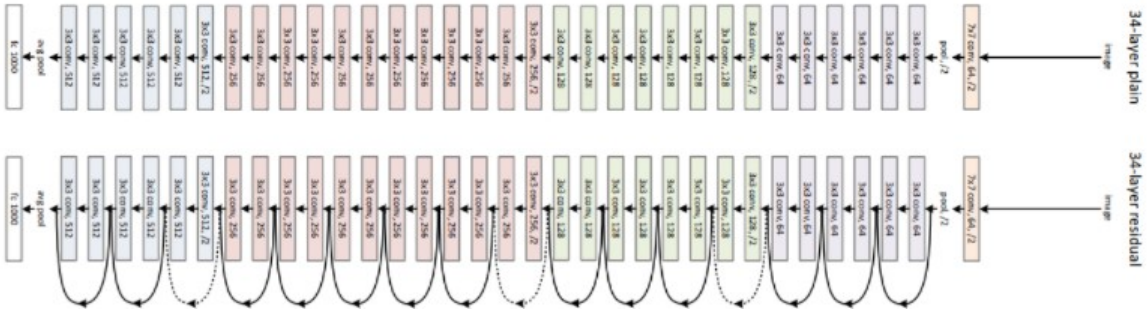



Figure 24: Comparison between a 34-layers standard CNN and a 34-layers Residual Network. This image has been taken from [19]

# 5 Objectives of our project and proposed solutions

## 5.1 Binary classification approach

For KNMI's weather warning system is of great interest to understand, if the speed of wind gusts will or not be above a certain threshold. This can be of course really useful to emit alarms and activate protocols, according to the severity level. For this reason we decided to transform our forecasting problem into a Binary classification one, assigning entries related to labels with a value under or above a certain threshold to two different classes.

### 5.1.1 The architecture of the model

The architecture of the model we designed, to implement the Binary Classification Approach, is based on two components:

1. A Feature Extraction part
2. A Fully Connected part

The architecture of the Feature Extraction part is based on the ResNet50 convolutional architecture, while the Fully Connected part it's a a Dense Layer (Fully Connected) with one output neuron (we are considering a binary classification problem). The activation function that has been used in output is the Sigmoid Activation Function, while the loss function used to train the model is the Binary Cross Entropy Loss. A description of the Binary Cross Entropy Loss can be found in the appendix E The optimizer we decided to use is Adam. The learning rate is dynamically reduced during the training procedure, following a Reduce On Plateau policy (if after five epochs the mean loss doesn't reduce the learning rate is divided by 10). An Early Stopping policy has been implemented as well, in order to avoid the overfitting problem. The Early Stopping policy stops the training procedure if the loss doesn't decrease for a number of epochs equals to ten. To be precise, what we have in output from the network is the probability for a specific example to belong to the positive class (the class of the samples above the threshold). In other words, we are able to generate a probabilistic forecast, estimating the probability for the wind gusts' speed to be higher than a certain threshold, given in input the NWP deterministic predictions, described in section 3.

## 5.2 Problems identified working with the Binary Classification Approach

The large number of features that characterized our input data (see section 3), makes the feature selection and cross validation procedure, really complex. Given the nature of our model and our data, only Wrapper Feature Selection Techniques, can be applied, to identify the best set of predictors, for our Binary Classification Problem. Analyze the performances, of all the possible combinations of input variables and hyper-parameters, it's practically infeasible. For this reason the first problem we have faced, working with the Binary Classification Approach is:

- How we can design a pre-processing technique, to reduce the number of features considered, during our feature selection procedure?

The number of samples in our data-set, related to a high wind gust speed, it's exiguous. If we define our Binary Classification problem, using a high threshold, we end up working, with a Training Data-set, characterized by a strongly imbalanced class distribution. For this reason the second problem we have faced working with the Binary Classification Approach is:

- How we can deal with a Training Data-set, characterized by a strongly imbalanced class distribution?

In the next sections we are going to describe, which are the solutions we adopted, in order to solve these two problems.

### 5.2.1 Dimensionality reduction

The HARMONIE-AROME deterministic forecasts, that constitute our data-set, are characterized by a high number of predictors with a physical relationship to wind gusts (see section 3). To identify the unessential variables the level of correlation between the different features that characterize our data, has been analyzed. Two highly correlated variables bring only redundant information, which slow down the training procedure, without increasing the model's performances. The pairwise Pearson correlation coefficient has been computed for each couple of predictors. The data, considered in our project, are in spatial format, so for each couple of variables, the Pearson correlation coefficient has been computed for each pixel (see figure 27). The mean of the correlation values per pixel, has then been taken as indicator of the level of correlation between the different couples of predictors. Applying this preliminary analysis we have been able to avoid considering, during our feature selection procedure, two highly correlated features, at the same time. In this way we have been able to reduce the number of features taken into consideration. In order to compute the Pearson correlation coefficient, the function correlation, from the python library tfp.stats has been used. A description of the Pearson Correlation Coefficient is reported in the appendix A.

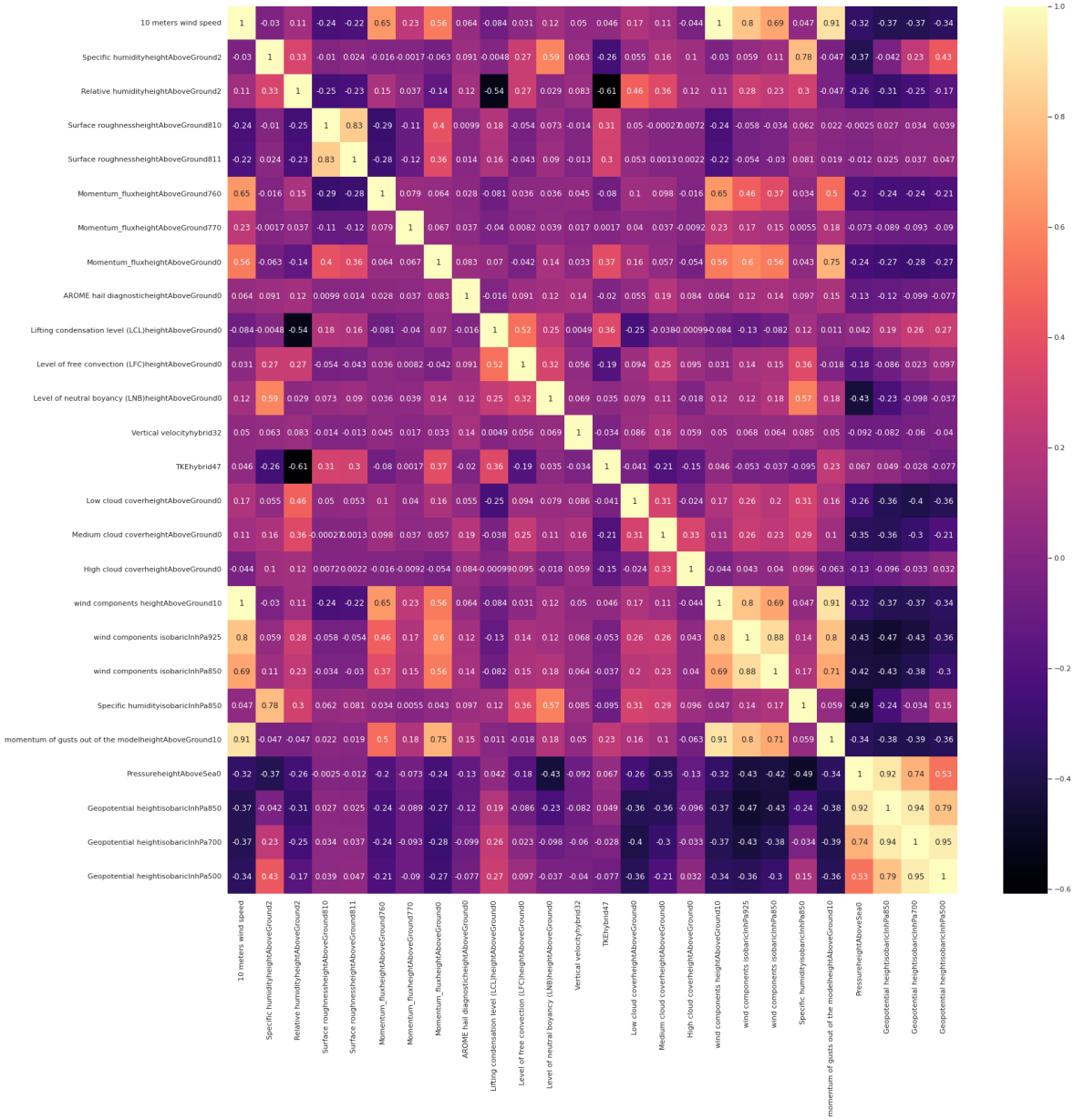



Figure 25: correlation indexes between all the input features for 2015 data

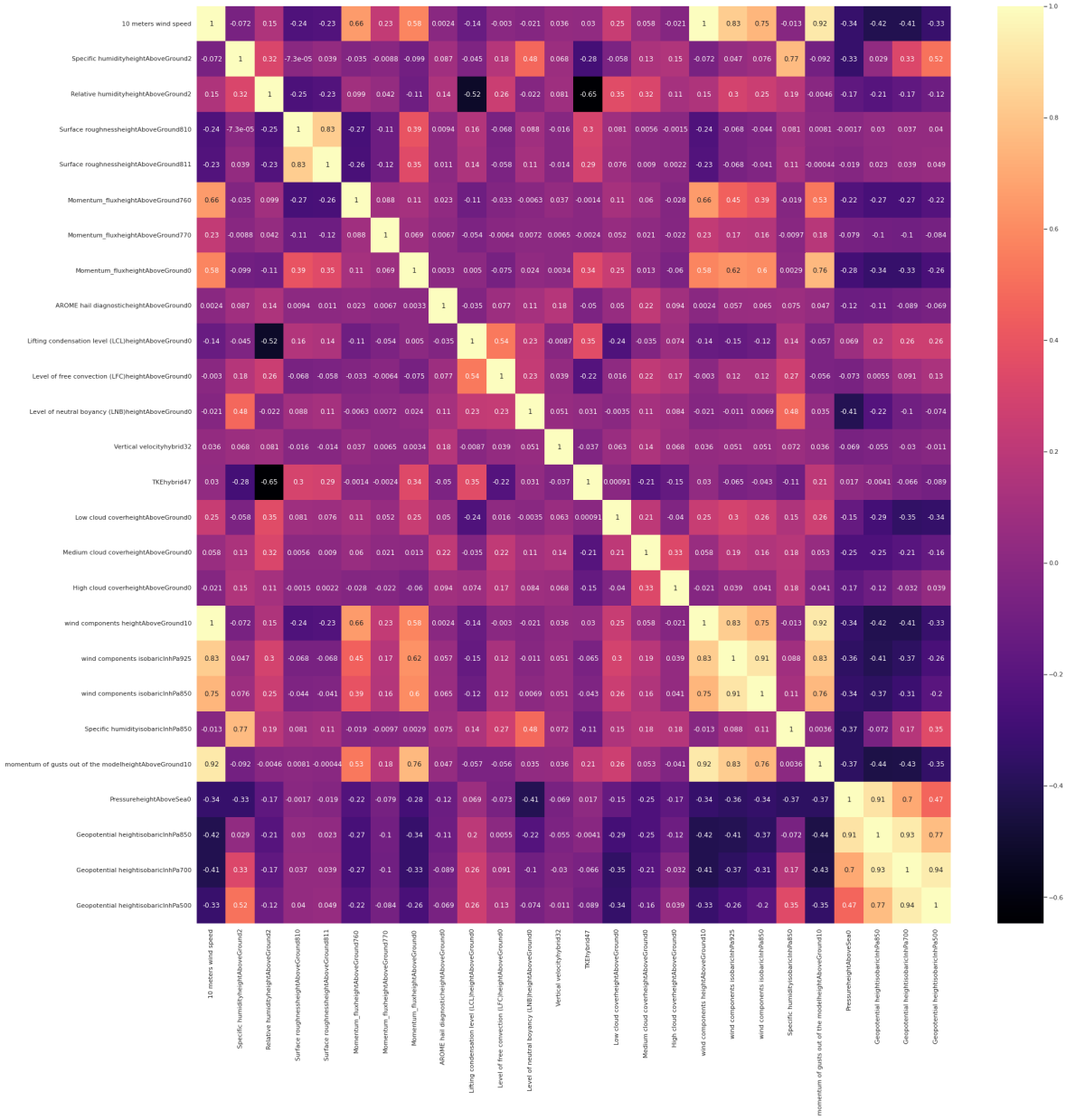



Figure 26: correlation indexes between all the input features for 2017 data

Looking at figure 25 and 26, reporting the correlation matrices obtained for the data, related to the years 2015 and 2017, we can notice that between some variables we have a level of correlation higher than 0.5 in absolute value. This means that we can expect a certain amount of redundant information, in our data-set. Another thing we can notice is that, some of the features we are working with are the same physical variables, they only refer to different altitudes. Obviously between these types of features we can expect a high degree of correlation. This is the case for example of the variable "Geo potential" or the variable "wind components". Also variables that are not the same physical variables can be characterized by a certain amount of correlation. This is the case for example of the variables "10m wind speed" and "Momentum flux". We have to specify that for this pre-processing analysis, the Pearson correlation coefficient has been used, so we only investigated the level of linear correlation between features. Our predictors can still be highly non linear correlated.

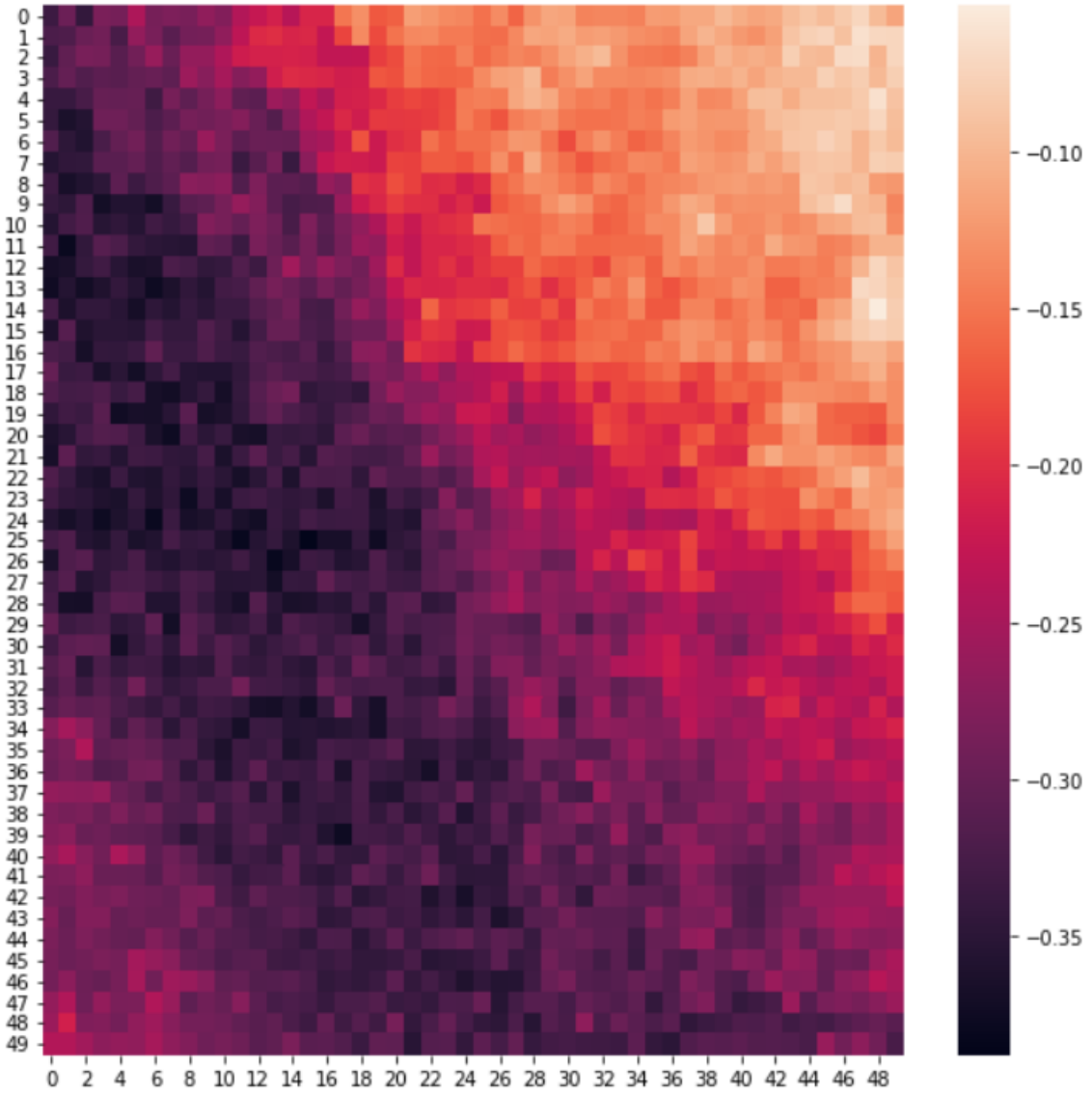



Figure 27: Heat map, showing the level of correlation per pixel between the variables 'wind speed 10m above the ground' and 'Surface Roughness height above the ground 811', over a 50*50 matrix

### 5.2.2 The problem of the training data-set's imbalanced class distribution

As it is reported in [12], the smallest threshold applied by KNMI weather institute, to identify an event as extreme, emitting a code yellow warning, is a wind gust speed of 16m/s or 60km/h.

| Thunderstorms | | | |
|---|---|---|---|
| | Local thunderstorms with one or more of the following phenomena:<br>• gusts (›60 km/h),<br>• And/or locally ›30 mm/h precipitation,<br>• And/or hail size up to 2 cm. | Organized thunderstorms with one or more of the following phenomena:<br>• severe gusts (›75 km/h),<br>• And/or locally ›50 mm/h precipitation,<br>• And/or hail size 2-4 cm. | *Well organized thunderstorms with one or more of the following phenomena:<br>• severe gusts (›100 km/h),<br>• And/or locally ›75 mm/h precipitation,<br>• And/or hail size ›4 cm. |

Figure 28: Tables showing the weather conditions, necessary to emit a code yellow, orange or red. This image has been taken from [12]

The data we decide to use as training data, are the ones related to the years 2015 and 2017. These two years present a larger number of extreme events. As we can see from figure 29, applying a threshold of 16m/s, to define the two different classes, that characterize our Binary Classification problem, we obtain a training data-set characterized by a strongly imbalanced class distribution. A really imbalanced training data-set, affects the performances of a classification model, strongly decreasing its ability in classifying the examples, belonging to the minority class. To mitigate this problem, we decide to follow two different approaches:

1. Downsample the majority class, following a specific logic.
2. Weight in a different way the contributions to the loss function, of samples belonging to the two different classes.

As I said before 2015 and 2017 data have been used as training data-set, while the data related to year 2016, have been used as test data-set. This choice has been done, not only because year 2015 and year 2017 present a larger number of extreme events, but also because in this way we ensure that, no temporal relationships are present between test and training data-sets.

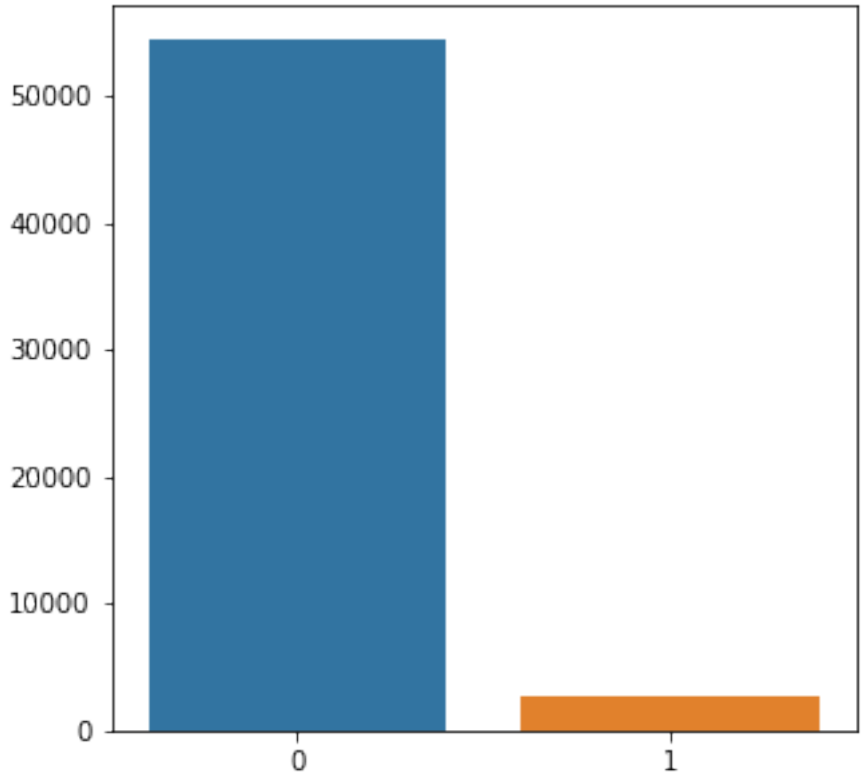



Figure 29: class distribution of the training data-set (year 2015 and 2017)

### 5.2.3 Downsampling techniques

The question behind the two Downsampling Techniques we have designed is: how we can find a way to eliminate entries, that are useless for our training procedure? With the expression useless entries, we mean examples that don't bring useful information, but instead only slow down the training procedure, decreasing at the same time the performances. The two Downsampling Techniques we have developed, keeping in mind the preceding question, respectively consist of:

1. Deleting all the entries, which labels' values are under a certain threshold.
2. Randomly deleting a certain amount of entries related to labels, which values are under a certain threshold.

The intuition behind the first technique is that, entries related to a low wind gust speed are useless to predict extreme events. Eliminating them we obtain a training data-set characterized by a more balanced class distribution, improving the model performances. The intuition behind the second technique is to try to keep, a little amount of information, about entries associated with a lower wind gusts speed. This kind of information can be useful for the model.

#### 5.2.4 Different weights technique

Another approach we have adopted, to mitigate the training data-set's class imbalance problem, consists of assigning different weights to the contributions to the loss function, of entries belonging to the two different classes. A larger weight has been assigned to the contributions of the entries, belonging to the minority class, while a smaller weight has been assigned to the contributions of the entries, belonging to the majority class. To be more precise the weights associated to the samples, belonging to class $i$, have been computed in the following way:

$w_i = \frac{N}{c*N_i}$

where:

- $N$ is the total number of samples.
- $N_i$ is the number of samples belonging to class $i$.
- $c$ is the number of classes.

### 5.3 Feature selection and cross-validation procedure

For our feature selection procedure we have considered a group of twelve different features. These twelve features have been selected combining:

1. The results obtained during the preliminary analysis, about the level of correlation (see section 5.2.1).
2. The list of variables identified by Veldkamp et al. in [35] as more useful for their problem in exam.

The results obtained for the pairwise Pearson correlation coefficients have been taken into consideration, in order to avoid considering, during our feature selection procedure, two highly correlated features at the same time. Two highly correlated features only bring redundant information, which slow down the training procedure, without adding useful insights, about the problem in exam. The complete list of variables considered, during our feature selection procedure is reported in appendix C. To identify the best set of predictors, for our Binary Classification problem, we decided to adopt a Forward Selection Approach (see section 4.3.5). For each combination of features analyzed, a sub-search has been performed, to identify the best set of hyper-parameters, for that specific combination of variables. We decided to focus on two different hyper-parameters:

- The initial learning rate
- The batch size

The optimal number of epoch for each combination of features analyzed, has been automatically identified, exploiting an Early Stopping Policy (the Early Stopping Policy, we decided to adopt, has been described in section 5.1.1). To evaluate the performances of the model, for each combination of features and hyper-parameters, that has been analyzed, three different runs have been executed. For each run, the training data (year 2015 and 2017) have been randomly splitted into training and validation sets. The mean value of the results obtained during the three runs, has been taken as indicator of the performances. During the Feature Selection and Cross-Validation procedure, on the entries used for training, the first Downsampling Technique, described in section 5.2.3, has always been applied, deleting the 50% of the data. Different weights have also been assigned to the contributions to the loss function of the training examples, belonging to the two different classes, following the technique

described in section 5.2.4. Due to the low number of entries related to a wind gust speed higher than the threshold we have worked with (16 m/s), the training data have been randomly splitted, using every time, the sklearn function StratifiedShuffleSplit. This function ensures that in both the training and validation sets, the same proportion of entries belonging to the two classes, will be present. The metric used to evaluate the performances of the model during the cross-validation procedure is the f1-score. This metric has been selected, because it gives a general idea about the model performances on both classes. A description of the f1-score is reported in appendix F.

## 5.4 Confusion Matrix

A Confusion Matrix can be described as a way to summarize and visualize the results obtained, using a Classification Model. The accuracy score alone can be misleading, if we have to analyze the performances of a Classification Model. This is particularly true if we have an unequal number of observations in each class. The classification accuracy is simply the ratio between the number of correctly classified test entries, and the total number test examples. The problem with this metric is that it hides details, concerning the type of errors made by the model.

| | Predicted Positive | Predicted Negative |
|---|---|---|
| Actual Positive | True Positive (TP) | False Negative (FN) |
| Actual Negative | False Positive (FP) | True Negative (TN) |

Figure 30: Structure of the Confusion Matrix of a Binary Classification Problem. This image has been taken from [36]

Looking at figure 30, showing the structure of a Confusion Matrix for a Binary Classification problem, we can notice that, this schema gives us useful insights, about the type of errors performed by the model. Looking at the Confusion Matrix, we can understand better the model performances on both classes, visualizing the number of correctly classified and misclassified examples. A Confusion Matrix, with the structure of the one reported in figure 30 has been produced, for each experiment we have performed on the test set (year 2016). The results reported on those Confusion Matrices, have been used to compute specific categorical scores. All the information reported in this section, concerning confusion matrices as well as all the information reported in section 5.5, describing different categorical scores and the structure of a performance diagram, are based on [36], by Schreurs et al.

## 5.5 Categorical Scores and Performance Diagram

Observing the results reported on the Confusion Matrices, generated for each experiment we have performed on the test data-set, we have been able to compute various verification scores commonly used in meteorology:

1. POD: The probability of detection (POD) is defined as:

   $$POD = \frac{true\ positives}{true\ positives + false\ negatives}$$

2. FAR: The False Alarm Ratio (FAR) is defined as:

   $$FAR = \frac{false\ positives}{true\ positives + false\ positives}$$

3. CSI: The critical success index (CSI) is defined as:

   $$CSI = \frac{true\ positives}{true\ positives + false\ negatives + false\ positives}$$

4. bias: The bias score is defined as:

$$bias = \frac{true\ positives + false\ positives}{true\ positives + false\ negatives}$$

Where *true positives* is the number of samples, correctly predicted as belonging to the positive class, *false positives* is the number of samples, wrongly predicted as belonging to the positive class and *false negatives* is the number of samples, wrongly predicted as belonging to the negative class. The CSI and bias can also be expressed as a function of the POD and of the Success Ratio (SR=1-FAR):

$$CSI = \frac{1}{\frac{1}{SR} + \frac{1}{POD} - 1}$$

$$bias = \frac{POD}{SR}$$

All the terms defined before can be visualized, at the same time, using a performance diagram.

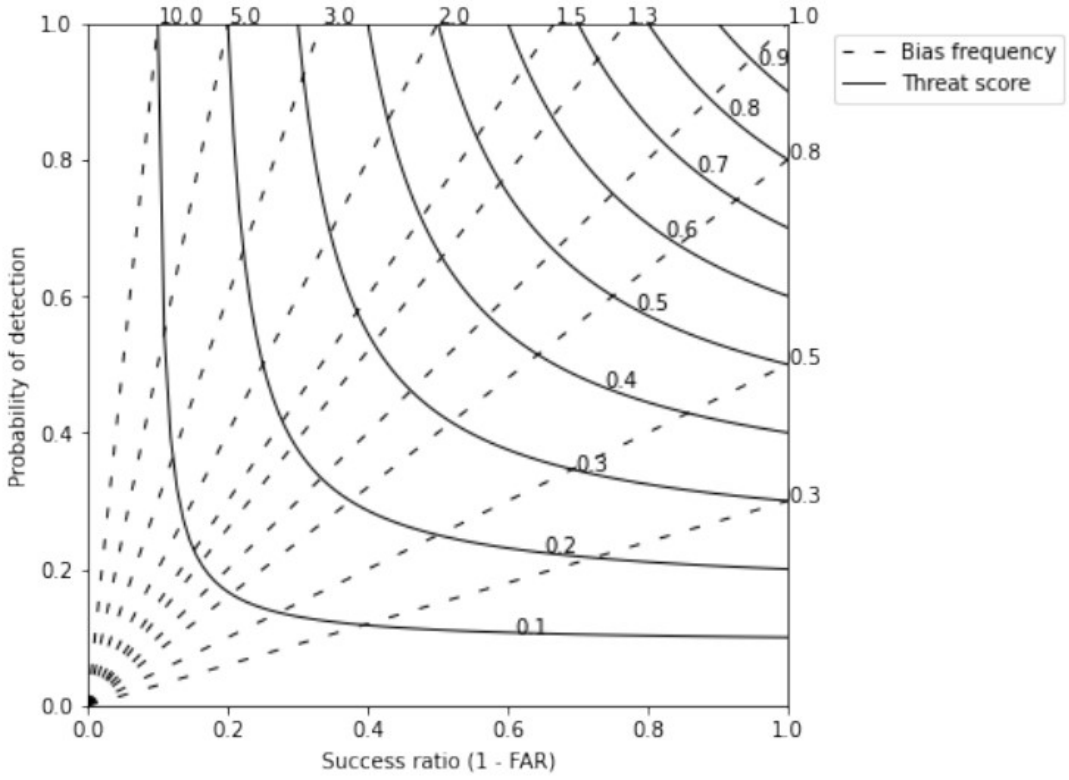



Figure 31: General structure of a performance diagram, this image has been taken from [36]

As we can see from figure 31, on the x-axis of a Performance Diagram is reported the Success Ratio (1-FAR), while on the y-axis is reported the Probability of detection (POD). The dashed lines indicate different biases, while the curved lines indicate different CSI values.

# 6 Results

## 6.1 Results obtained during the feature selection and cross-validation procedure

Like we specified in 5.3, for our Feature Selection procedure we decide to use a Forward Selection approach. After the second step of the forward selection procedure we didn't notice a large improvement in the performances, using two variables, instead of one. For this reason we decide to stop our Forward search at the second step, without investigating the use of three different variables. During our Feature

Selection and Cross-validation procedure the best results have been obtained combining the variables "momentum of gusts out of the model 10m above the ground" with the variable "Relative humidity 2m above the ground", using as hyper-parameters a batch-size of 128 and an initial learning rate of $1^{-4}$. As we already specified in 5.3, the best number of epochs has been automatically identified using an Early Stopping policy, which terminates the training procedure if the loss doesn't decrease for more than 10 epochs. Good results have also been obtained, using the variable "wind speed 10m above the ground", the variable "momentum of gusts out of the model 10m above the ground" and the variable "Momentum-flux height above the ground 777" individually, using as hyperparameters a batch-size of 256 and an initial learning rate of $1^{-4}$. Other variables combined with the variable "momentum of gusts out of the model 10m above the ground" show performances, really similar to the best ones, we have obtained. This is the case for example of the variables "Surface Roughness height above the ground 810" and the variable "Geopotential height isobaric in hPa 500". The complete list of all the experiments we have performed during our Feature Selection and Cross-Validation procedure, with the related results is reported in H.

## 6.2 Results on the test data-set

After the Cross-Validation and Feature Selection procedure, different experiments have been performed. For each experiment the model has been retrained, using the best set of variables and hyper-parameters, identified during the Cross-Validation and Feature Selection procedure. The performances have been, every-time evaluated, on the entire test data-set (year 2016). The two Downsampling Techniques described in 5.2.3, have been combined with the Weights Method, described in 5.2.4, comparing then the results with the ones obtained:

1. Applying only the 2 Downsampling Techniques alone.
2. Giving only different weights to the training entries' contributions to the loss function, without downsampling the training data.
3. Without applying any technique.
4. Applying the simplest possible downsampling technique, so randomly deleting a certain amount of training samples, without following any logic.

All the obtained results can be compared, observing the performance diagram, reported in figure 32. Remember that: on the x-axis is reported the success rate of the model, while on the y-axis is reported the probability of detection of the model.

- The success rate of a binary classification model is computed as $1 - false\ alarm\ ratio$ of the model, where the false alarm ratio is computed as:

  $$\frac{false\ positives}{false\ positives + true\ positives}$$

  The success rate of the model reflects the tendency of the model, to generate false alarms.

- The probability of detection of a binary classification model is computed as:

  $$\frac{true\ positives}{true\ positives + false\ negatives}$$

  The probability of detection of the model, reflects the tendency of the model to generate misses (extreme events not identified as extreme)

The models, characterized by the best performances, are on the top right of a Performance Diagram.

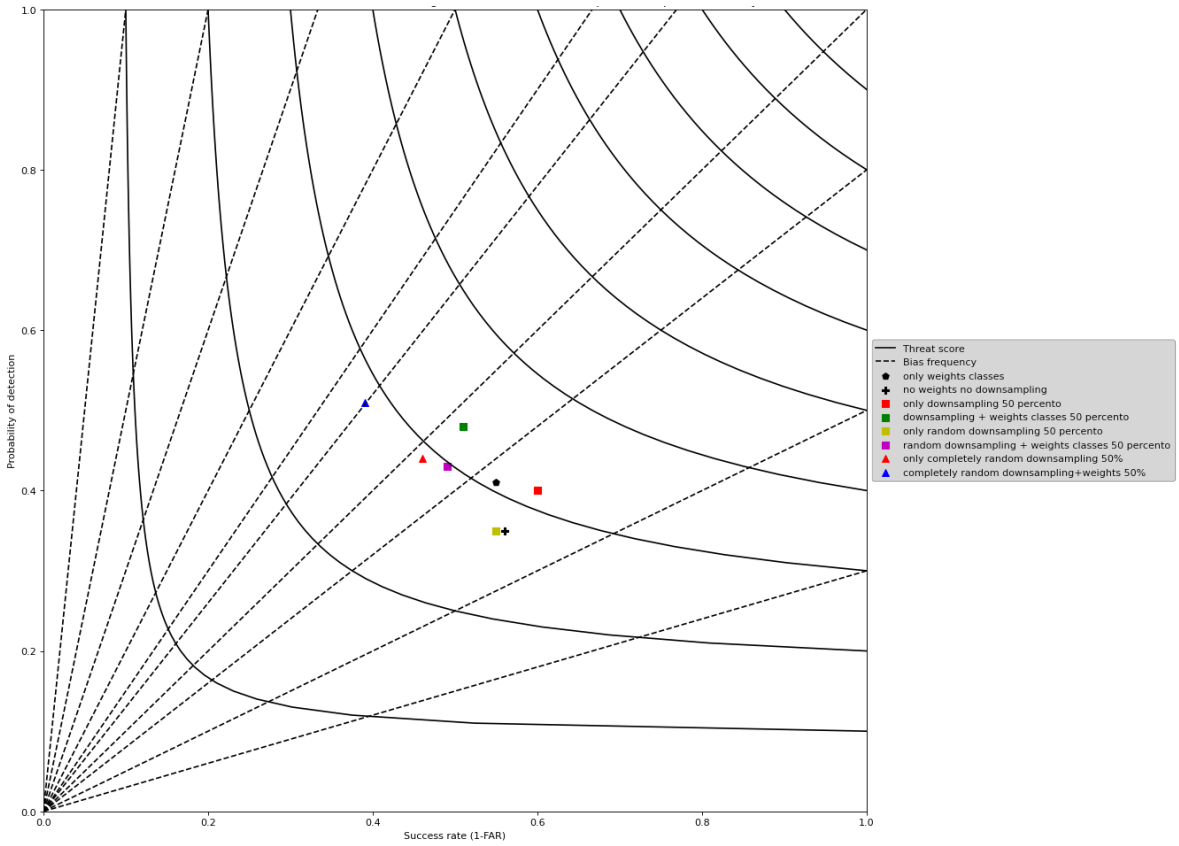



Figure 32: Performance Diagram, showing the results obtained for different experiments, using the variables "Momentum of gusts" and "Relative Humidity", with a threshold of 16m/s

Looking at the performance diagram reported in figure 32, we can make some considerations: Both the Downsampling Techniques described in section 5.2.3, combined with the Weights Method described in section 5.2.4, give us better performances in terms of POD, compared to the ones obtained, using the two Downsampling Techniques alone, and without applying any technique. At the same time we can notice a decrease of the performances in terms of SR. Using the first Downsampling Technique, described in section 5.2.3, alone, we have obtained better performances in terms of both POD and SR, compared to the ones obtained, without applying any technique. The performances, in terms of POD, obtained applying the first Downsampling technique alone, are comparable to the ones obtained, applying the Weights Method alone, but at the same time the performances in terms of SR, are better. The second Downsampling Technique, described in section 5.2.3, applied alone, gives us performances, similar to the ones obtained without applying any technique. The two triangles reported on the Performance Diagram, refer to the results obtained, randomly deleting training examples, without following any logic. This really simple downsampling technique, combined with the Weights Method, gives us better performances in terms of POD, but at the same time the performances in terms of SR largely decrease. In general all the results are between 0.35 and 0.51, in terms of probability of detection and between 0.39 and 0.6, in terms of success ratio.

### 6.2.1 Results on the test data-set in terms of F1 Score and Brier Score

The experiments, performed on the test data-set, using the best set of predictors and hyper-parameters, identified during the Feature Selection and Cross-Validation procedure, have been evaluated not only in terms of the Categorical Scores, described in 5.5. For all the experiments performed, we have also computed the corresponding F1 and Brier Scores. Looking at table 1, reporting the results we obtained, in terms of F1 and Brier Score, for the most significant experiments we have performed on the test data-set, we can notice that the Brier Score is not able to give a general idea about the model's performances on both classes (remember that the best performances correspond to a F1 Score equals to 1 and a Brier Score equals to 0). Analyzing only the Brier Score, we can think that, our

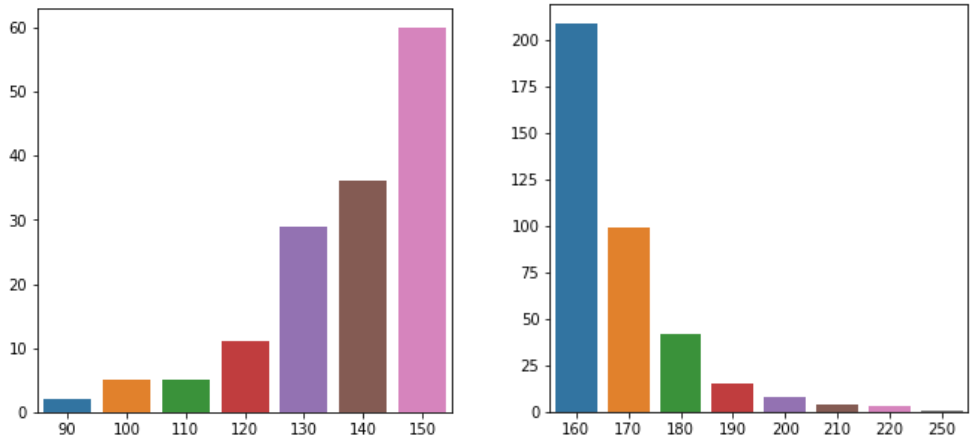
experiments are characterized by good results, but it's only because this metric considers the overall performances of the model. If, like in our case, a classification problem is characterized by a strongly imbalance class distribution, the results obtained, using metrics like the Brier or the Accuracy Scores, can be misleading. A description of the F1 Score and the Brier Score, is reported in appendixes F and G. For each experiments performed, the respective Categorical Scores have been computed, in order to visually compare the results, reporting them on a performance diagram. In this way we can obtain more insights about the model performances, than just looking at the F1 Score, which can be difficult to interpret.

Table 1: Results obtained in terms of F1 Score and Brier Score, for some of the experiments, performed on the test data-set, using the variabels "Momentum of gust" and "Relative Humidity", with hyper-parameters: batch-size=128 and initial-learning-rate=0.0001. The threshold adopted is 16m/s

| **Experiment configuration** | **F1 Score** | **Brier Score** |
|---|---|---|
| No weights, no Downsampling | 0.47679708826205647 | 0.018176101261951422 |
| Only weights | 0.4634146341463414 | 0.01855195093251284 |
| Only first Downsampling Technique 50% | 0.47922705314009656 | 0.017337507037177437 |
| First Downsampling Technique+weights 50% | 0.48648648648648646 | 0.0189613698785704 |

### 6.2.2 Misclassified examples

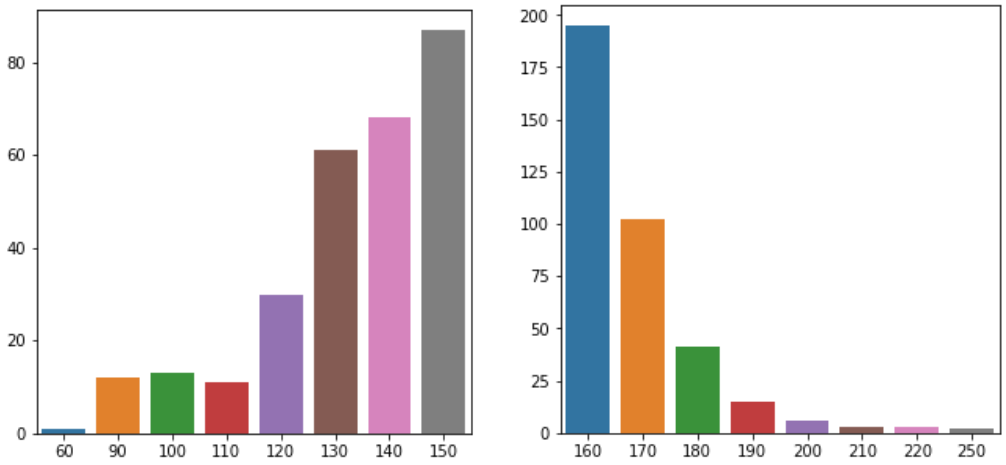
To have a better idea about, the type of errors made by the Binary Classification Model, the distribution of the values of the labels, associated to the misclassified examples, has been analyzed. As we can see from the bar plots reported in figures 33 and 34, showing the misclassified examples for two different experiments, our Binary Classification Model has some problems in identifying the events that are on the border, between the two different classes, defined by the 16m/s threshold. This is a common behaviour for classification models, which usually have problems in correctly classifying samples, that are on the border between the different classes. The labels' values reported in figures 33 and 34, are on a 0.1m/s scale, so a value of 160 corresponds to 16m/s.




(a) False alarms only Second Downsampling Technique, deleting 50% of training data

(b) Misses only Second Downsampling Technique, deleting 50% of the training data

Figure 33: Misclassified samples applying only the Second Downsampling Technique described in section 5.2.3, deleting 50% of the training data.




(a) False alarms Second Downsampling Technique+Weights, deleting 50% of training data

(b) Misses Second Downsampling Technique+Weights, deleting 50% of training data

Figure 34: Misclassified samples applying the Second Downsampling Technique, described in section 5.2.3, plus the Weights Method described in section 5.2.4, deleting 50% of training data.

## 6.3 Results obtained with a more balanced training data-set

To have an idea of how much the model's performances are affected, by the imbalanced training data-set's class distribution, we have performed different experiments using lower thresholds. Looking at the bar plots, reported in figures 35, 36, 37 and 38, we can notice that, using a lower threshold to define the two classes of our binary classification problem, we can work with a training data-set, characterized by a much more balanced class distribution.

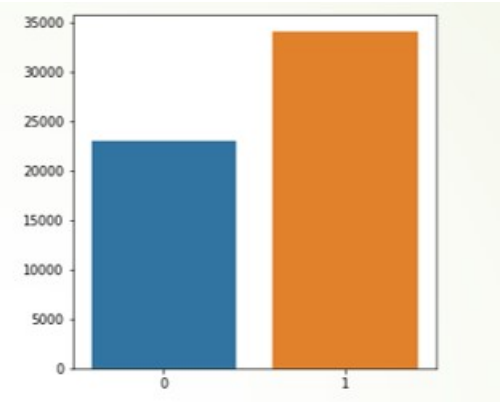



Figure 35: class distribution of the training data-set, 8m/s threshold

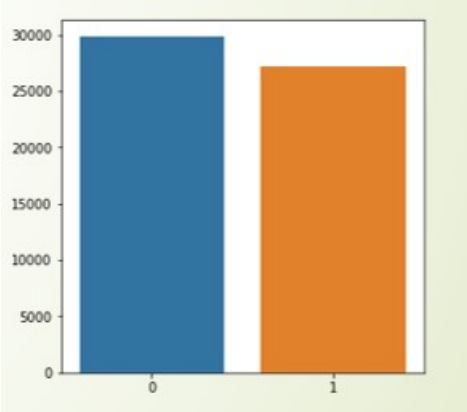



Figure 36: class distribution of the training data-set, 9m/s threshold

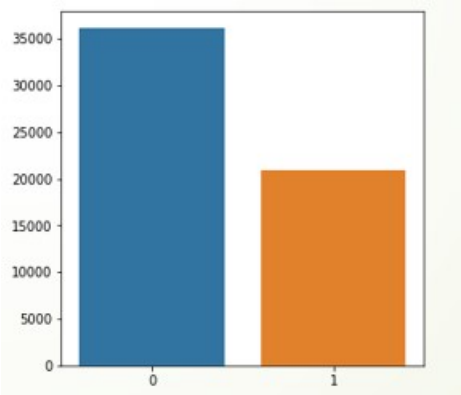



Figure 37: class distribution of the training data-set, 10m/s threshold

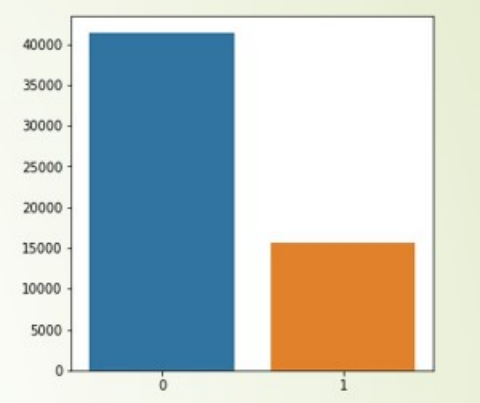



Figure 38: class distribution of the training data-set, 11m/s threshold

All the results obtained for the experiments performed, using lower thresholds, have been reported on a performance diagram to be compared.

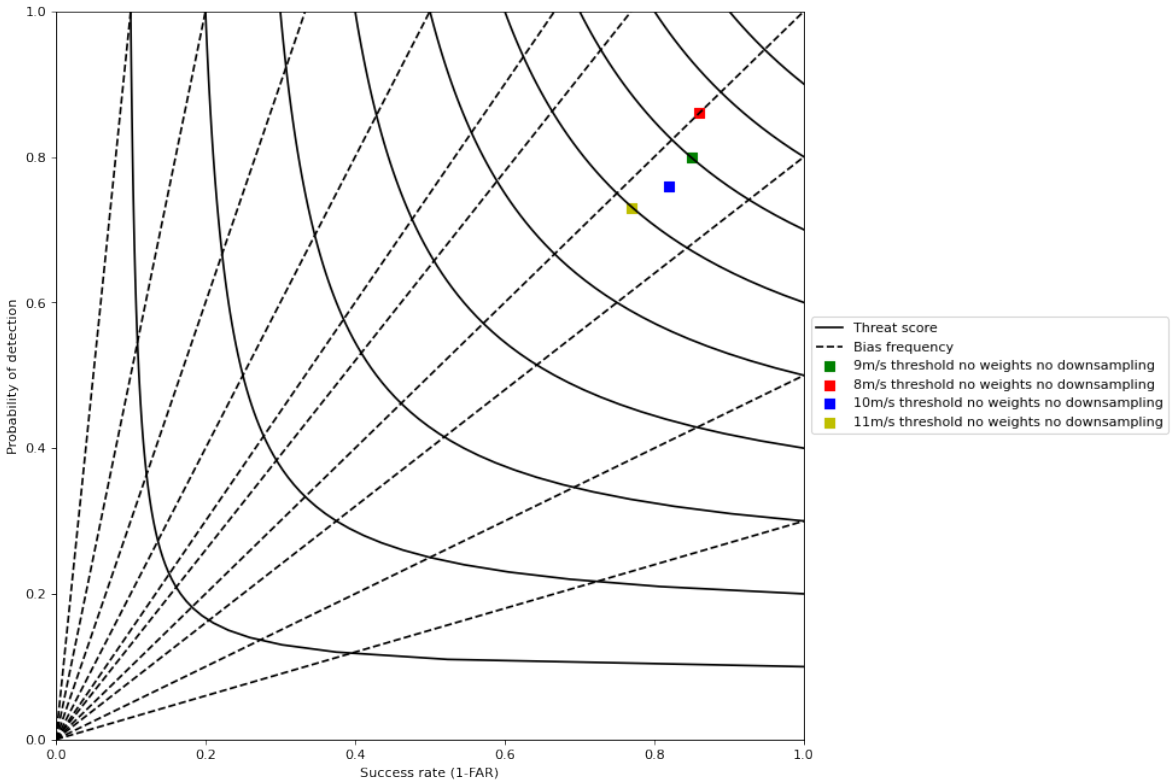



Figure 39: performance diagram, comparing the results obtained for lower thresholds

Looking at figure 39 we can notice that, training the model with more balanced training data-sets, we have been able to obtain a large increase in the performances, both in terms of probability of detection and success rate. This means that for sure, a training data-set characterized by a strongly imbalanced class distribution, largely affects the performances of our Binary Classification model. For all the experiments performed using lower thresholds, the model has been retrained using the best set of predictors and hyper-parameters, identified during the Feature Selection and Cross-Validation procedure.

## 6.4 Other results on the test data-set

The variables "wind speed 10m Above the Ground" and "Specific Humidity 2m Above the Ground", present a physical relationship with the variables "momentum of gusts out of the model 10m Above the Ground" and "Relative Humidity 2m Above the Ground". For this reason we decided to perform some experiments on the test data-set using also these two variables, to have a more general idea about our Binary Classification Model's performances. We also increase the batch-size to 256, maintaining an initial learning rate of $1^{-4}$. The obtained results, have been again reported on a performance diagram to be compared.

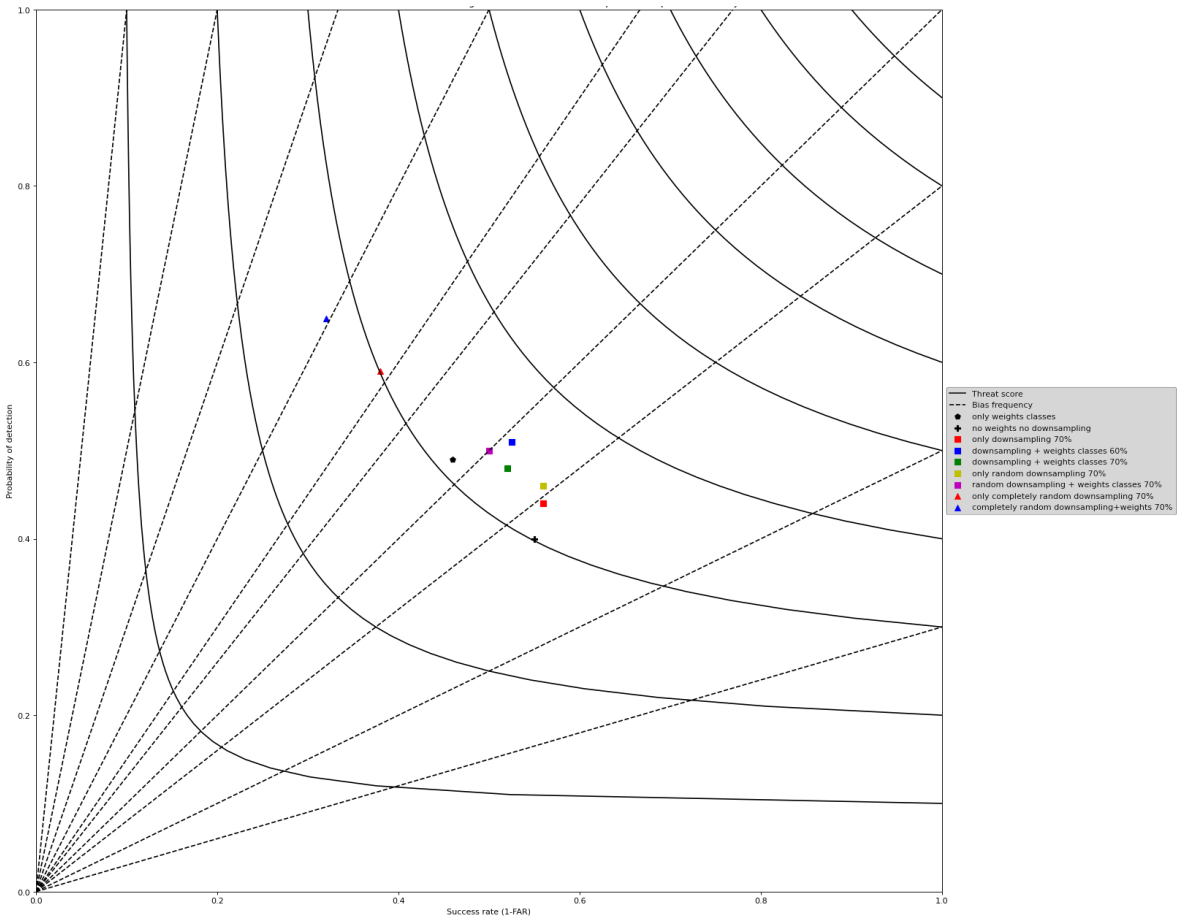



Figure 40: Performance Diagram, showing the results obtained for different experiments, using the variables "Wind speed" and "Specific Humidity", with a threshold of 16m/s

Looking at figure 40, we can make some considerations: Both the Downsampling Techniques, described in section 5.2.3, combined with the Weights Method, described in section 5.2.4, give us better performances in terms of POD, compared to the ones obtained applying the two Downsampling Techniques alone, and the ones obtained without applying any technique. On the other hand the performances in terms of SR are worse. The two Downsampling Techniques applied alone, give us similar results in terms of SR, compared to the ones obtained without applying any technique, but the performances in terms of POD are better. Combining the Weights Method with both the Downsampling Techniques, we have been able to obtain better performances, in terms of SR, compared to the ones obtained applying the Weights Method alone, maintaining similar performances, in terms of POD. The two triangles on the Performance Diagram, refer again to the results obtained, randomly deleting training examples, without following any logic. Applying this really simple downsampling technique, we can notice a large increment of the performances, in terms of POD and a large decrease, in terms of SR. All the results reported, for the variables "Wind speed" and "Specific Humidity" have been obtained deleting the 70% or 60% of the training data, while the results reported for the variables "Momentum of gusts out of the model" and "Relative Humidity" have been obtained deleting 50% of the training data. This choice has been done because, using the variables "Wind speed" and "Specific Humidity", we noticed better performances in terms of POD, deleting a larger amount of training examples.

## 6.5 Normalization

In this section we want only to specify that all the results reported have been obtained, normalizing the data in Input to the model, subtracting the mean and dividing by the standard deviation. This procedure has been performed separately for each input predictor. The mean and standard deviation

values, have been computed, using the training data, to then apply the same transformation to the data used to evaluate the model. Normalize the Input data is a common deep learning pre-processing operation, that allows the model to converge to a better result, with a lower computational effort.

# 7 Conclusions

For KNMI's weather warning system is of great interest to understand, if the speed of wind gusts will or not be above a certain threshold. For this reason can be really useful to transform our forecasting problem into a binary classification one, assigning the samples we are working with to two different classes, based on the values of their labels. Applying this classification approach we have faced the problem of working with a training data-set, characterized by a strongly imbalanced class distribution, due to the low number of samples related to extreme events, that were available in our data-set. This problem largely affects the performances of our model. The two different approaches, described in sections 5.2.3 and 5.2.4, mitigate the class imbalance problem. In particular combining the two Downsampling Techniques we designed, with the Weights Method, we have been able to obtain better performances in terms of probability of detection, without a large decrease of the performances in terms of success rate. The other problem we have faced during our project, is related to the large number of predictors, that characterized the deterministic NWP forecasts, we have worked with. An exhaustive Feature Selection and Cross-Validation procedure, to understand which was the best set of variables and hyper-parameters for our problem in exam, analyzing all the possible combinations, was practically infeasible. The pre-processing technique we designed, based on the level of correlation between the different input predictors, allows us to reduce the number of variables considered during our Feature Selection and Cross-Validation procedure. Analyzing the level of correlation between each couple of input variables, we avoided to consider, during the Feature Selection procedure, two highly correlated predictors, at the same time. Highly correlated features only bring redundant information.

# 8 Future Work

At the first stages of our project we not only design the Binary Classification model, but also other two models, able to estimate a continuous conditional distribution $P(Y|X)$. $Y$ is the target variable of interest (the wind gusts speed), while $X$ are the deterministic NWP forecasts, in input. The two models are:

1. The Quantized Softmax Technique
2. The Bernstein Polynomials Technique

These two models share the same Convolutional architecture of the Binary Classification approach, the difference is in the last output layer, which has been modified, to adapt to the different techniques. The idea behind these two techniques is to use the features extracted by the convolutional architecture, to estimate the continuous conditional distribution $P(Y|X)$. During our project we decide to focus more on the Binary Classification approach, proposing different techniques to mitigate the class imbalance problem, that characterizes our training data-set. For sure will be of great interest to perform in the future, some experiments using also the Quantized Softmax and the Bernstein Polynomials Techniques, to then compare the results, with the ones obtained using the Binary Classification approach.

## 8.1 Quantized Softmax

The Quantized Softmax Technique, as it is described in [35], is a non parametric method (no initial assumptions are required), which approximates the conditional distribution $P(Y|X)$, as a hystogram. $Y$ is the target variable of interest (wind gusts speed), while $X$ are the same deterministic NWP forecasts, used with the Binary Classification Approach. To produce the output hystogram, our problem has been converted into a classification problem, characterized by 300 different classes, one for each bin of the hystogram we want to estimate. The architecture of the model is composed of two parts:

1. The feture extraction part, which is based on the ResNet50 architecture.
2. The fully connected part, which is a Dense layer with 300 output neurons.

The activation function, that has been used in output is the Softmax activation function, which is the standard activation function for multi-class classification problems. We have designed two different functions to train the model: one uses the Categorical Cross Entropy Loss, while the other one the Crps Loss. The Categorical Cross Entropy Loss is described in appendix I, while the CRPS loss is described in appendix J. To evaluate the model performances, we have implemented a function that, given the predictions generated by the network, estimates a certain number of quantiles. The function crps-ensemble of the python library properscoring is then applied, over the estimated quantiles. The best set of quantiles to use, in order to evaluate the model performances, has been determined following [43], by Zamo et al.

## 8.2 Bernstein Polynomials

The Bernstein Quantile Network is a non parametric method (no initial assumptions required), which estimates the quantile function of interest, as a linear combination of the Bernstein Polynomials, following the formula:

$Q(\tau|x) = \sum_{l=0}^{d} \alpha_l(x) B_{l,d}(\tau)$

Where $\tau$ are the quantiles' orders and $B_{l,d}(\tau) = \binom{d}{l}\tau^l(1-\tau)^{d-l} \quad for \ l = 0, ..., d$

The network we designed, uses again the ResNet50 architecture for the feature extraction part, while the fully connected part consists in a Dense layer with eight output neurons (one for each coefficients we need to estimate). The estimated coefficients $\alpha_l$ have to be positive and non-decreasing. For this reason the network targets the increments $\tilde{\alpha_l}$, from which the coefficients are derived using a recursive formula:

$\alpha_0 = \tilde{\alpha_0}, \ \alpha_l = \alpha_{l-1} + \tilde{\alpha_l} \quad for \ l = 1, ..., d$

The loss function we designed, for the training procedure, first computes 99 different quantiles, for each input data example, using the quantile function described before. The mean quantile loss over the 99 estiamated quantiles, is then computed. A description of the Quantile Loss is reported in appendix K. The best set of 99 quantiles to estimate has been determined following again [43]. For testing the same function, as the one described for the Quantized Softmax Technique, has been used, but this time the quantiles have been computed using the quantile function reported before, in this section. We can notice that the Bernstein Polynomials technique is a non parametric method as the Quantized Softmax technique. But The Bernstein Polynomials method requires a lower number of parameters, because the number of outputs is equal to 8, while for the Quantized Softmax technique we have 300 output neurons. The loss function of the Bernstein polynomials technique is for sure more complex, compared to the Quantized Softmax technique. The activation function used for the Bernstein polynomials techniques is the RelU activation function. Using the RelU we ensure that all the outputs will be positive and the coefficients will be non decreasing.

## 8.3 Up-sampling

For our project we decided to balance the training data-set's class distribution, focusing on a down-sampling approach. For sure will be of great interest in the future, to investigate also the use of up-sampling techniques. The Generative Adversarial Networks (GANs) have demonstrated good skills in generating synthetic images really similar to the real ones. This architecture can be applied also to our weather forecasting problem, to generate synthetic examples, related to extreme events. Our input data have similarities, with images' data format. GANs could be able to simulate the spatial patterns, that characterize examples, related to extreme events.

# A Pearson Correlation Coefficient

The Pearson correlation coefficient, is the ratio between the covariance of two random variables and the product of their standard deviations. It can be considered as the normalized version of the covariance and it takes values between -1 and 1. As the covariance this coefficient, can only identify the level of linear correlation between two random variables, which can still be highly non-linear correlated. Using a sample from two random variables $X$ and $Y$, the Pearson coefficient can be expressed in the following way:

$r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{n\sum_{i=1}^{n}(x_i-\bar{x})}\sqrt{n\sum_{i=1}^{n}(y_i-\bar{y})}}$
where:

- $n$ is the number of entries in the sample
- $x_i$ and $y_i$ are the $i^{th}$ entries
- $\bar{x}$ and $\bar{y}$ are the samples' means

# B Predictors that characterize the NWP deterministic forecasts, we have worked with

1. 10 meters wind speed
2. Specific humidity height above the ground 2
3. Relative humidity height above the ground 2
4. Surface roughness height above the ground 810
5. Surface roughness height above the ground 811
6. Momentum flux height above the ground 760
7. Momentum flux height above the ground 770
8. Momentum flux height above the ground 0
9. AROME hail diagnostic height above the ground 0
10. Lifting condensation level (LCL) height above the ground 0
11. Level of free convection (LFC) height above ground 0
12. Level of neutral boyancy (LNB) height above the ground 0
13. Vertical velocity hybrid 32
14. TKE hybrid 47
15. Convective cloud cover height above the ground
16. Low cloud cover height above the ground 0
17. Medium cloud cover height above the ground 0
18. 'High cloud cover height above the ground0
19. Wind components height above ground 10
20. wind components isobaric hPa 925
21. wind components isobaric hPa 850
22. Specific humidity isobaric hPa 850

23. Pressure height above sea 0
24. Geopotential height isobaric hPa 850
25. Geopotential height isobaric hPa 700
26. Geopotential height isobaric hPa 500
27. momentum of gusts out of the model height above the ground 10

# C Variables considered during our Feature Selection and Cross-Validation procedure

1. 10 meter wind speed
2. Specific humidity height above the ground 2
3. Relative humidity height above the ground 2
4. Surface roughness height above the ground 810
5. Momentum flux height above the ground 770
6. AROME hail diagnostic height above the ground 0
7. Lifting condensation level (LCL) height above the ground 0
8. Vertical velocity hybrid 32
9. momentum of gusts out of the model height above the ground 10
10. High cloud cover height above the ground 0
11. Geopotential height isobaric In hPa 500
12. TKE hybrid 47

# D Heaviside Function

The Heaviside step function or unit step function, is a discontinuous function. The output of the Heaviside step function is 1, when the input is positive, is 0 when the input is negative.

$H(x) = 1 \ \ if \ \ x \geq 0$

$H(x) = 0 \ \ if \ \ x \leq 0$

The Heaviside step function can be defined using also a different threshold, instead of 0.

# E Binary Cross Entropy Loss

The Binary Cross Entropy Loss can be defined as:

$B = \frac{1}{N} \sum_{i=1}^{N} -((y_i \log(p_i)) + (1 - y_i) \log(1 - p_i))$

where:

- B is the Binary Cross Entropy Loss
- N is the batch-size parameter.

- $p_i$ is the probability predicted by the Network for the $i^{th}$ sample. Remember that in output from our Network, using the Sigmoid Activation Function, we have the probability for a sample to belong to the positive class or class 1 (We are considering a binary classification problem, classes can be 0 or 1).
- $y_i$ is the class to which the $i^{th}$ sample actually belongs to. The classes can be 0 or 1.

The logarithm is used because it offers less penalty for small differences, and a high penalty for large differences. The probabilities are between 0 and 1 so the log values are negative. To compensate we take into consideration the negative average.

# F F1-score

The F1-score is obtained combining two other metrics: precision and recall. The F1-score can be defined as the harmonic mean of precision and recall:

$F1 = 2 \ \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$

where:

- Precision is the fraction of true positives examples, among the ones that the model classifies as positives
- Recall is the fraction of total examples, classified as positives, among the total number of positive examples

Differently from other metrics like the Brier Score or the Accuracy Score, the F1-Score, in a Binary Classification Problem Scenario, gives us a general idea, about the model's performances on both classes

# G Brier Score

The Brier Score is used to measure the accuracy of probabilistic predictions. The Brier Score can be applied to evaluate the performances of a model, which outputs are probabilities, assigned to a set of mutually exclusive discrete categories or classes. The Brier score is a cost function. Given a set of examples $i \in 1....N$ the Brier Score measures the mean squared difference between, the predicted probabilities, for the set of possible outcomes of each entry $i$ and the actual outcomes. Therefore lower is the Brier Score for a set of predictions, better the predictions are calibrated. Given a Binary Classification problem, the Brier Score can be defined in the following way:

$BS = \frac{1}{N} \sum_{i=1}^{N} (p_i - y_i)^2$

where:

- $p_i$ is the predicted probability for the $i^{th}$ example.
- $y_i$ is the actual outcome, which can have value 0 or 1 (We are considering a binary classification problem).
- $N$ is the number of examples we are considering.

This definition can be used only if we are considering binary events (for example wind gusts speed above the threshold or not).

# H Results of the Cross-validation and Feature Selection procedure

Table 2: results using 1 and 2 variables

| **Features** | **Best hyper-parameters** | **Best mean f1-score** |
|---|---|---|
| 10m wind speed | batch-size=256 initial learning-rate=$1^{-4}$ | 0.7540336762603642 |
| Specific humidity height above the ground 2 | batch-size=128 initial learning-rate=$1^{-4}$ | 0.651250810629865 |
| Relative humidity height above the ground 2 | batch-size=128 initial learning-rate=$1^{-4}$ | 0.616317447733764 |
| Surface roughness height above the ground 810 | batch-size=128 initial learning-rate=$1^{-4}$ | 0.6725195086718044 |
| Momentum-flux height above the ground 770 | batch-size=256 initial learning-rate=$1^{-4}$ | 0.7435976906486009 |
| AROME hail diagnostic above the ground 0 | batch-size=128 initial learning-rate=$1^{-4}$ | 0.30845142739782294 |
| Lifting condensation level above the ground 0 | batch-size=128 initial learning-rate=$1^{-4}$ | 0.5603352928567604 |
| Vertical velocity hybrid 32 | batch-size=128 initial learning-rate=$1^{-4}$ | 0.5596010925674355 |
| TKE hybrid 47 | batch-size=128 initial learning-rate=$1^{-4}$ | 0.6407367440801214 |
| High cloud cover height above the ground 0 | batch-size=256 initial learning-rate=$1^{-4}$ | 0.18960008101843293 |
| momentum of gusts out of the model 10m | batch-size=256 initial learning-rate=$1^{-4}$ | 0.7575750413510286 |
| Geopotential height isobaric In hPa 500 | batch-size=128 initial learning-rate=$1^{-4}$ | 0.6107486121171731 |
| momentum of gusts out of the model 10m<br>10m wind speed | batch-size=256 initial learning-rate=$1^{-4}$ | 0.7623877996958339 |
| momentum of gusts out of the model 10m<br>Specific humidity height above the ground 2 | batch-size=128 initial learning-rate=$1^{-5}$ | 0.7490948508936706 |
| momentum of gusts out of the model 10m<br>Relative humidity height above the ground 2 | batch-size=128 initial learning-rate=$1^{-4}$ | 0.7679143075093817 |
| momentum of gusts out of the model 10m<br>Surface roughness height above the ground 810 | batch-size=256 initial learning-rate=$1^{-4}$ | 0.7649268149280378 |
| momentum of gusts out of the model 10m<br>Momentum-flux height above the ground 770 | batch-size=256 initial learning-rate=$1^{-4}$ | 0.759729343574921 |
| momentum of gusts out of the model 10m<br>AROME hail diagnostic above the ground 0 | batch-size=128 initial learning-rate=$1^{-4}$ | 0.7395058162522723 |
| momentum of gusts out of the model 10m<br>Lifting condensation level above the ground 0 | batch-size=256 initial learning-rate=$1^{-4}$ | 0.7514691283026709 |
| momentum of gusts out of the model 10m<br>Vertical velocity hybrid 32 | batch-size=256 initial learning-rate=$1^{-4}$ | 0.7219321396504131 |
| momentum of gusts out of the model 10m<br>TKEhybrid47 | batch-size=128 initial learning-rate=$1^{-4}$ | 0.7449201414722832 |
| momentum of gusts out of the model<br>High cloud cover above the ground 0 | batch-size=256 initial learning-rate=$1^{-4}$ | 0.7449969263495483 |
| momentum of gusts out of the model 10m<br>Geopotential height isobaric In hPa 500 | batch-size=128 initial learning-rate=$1^{-4}$ | 0.762371842161128 |

# I Categorical Cross Entropy Loss

The Categorical Cross Entropy loss is used for multi-class classification problems and is defined for each example in the following way:

$\sum_{i=1}^{n} y_i * \log \tilde{y_i}$

where:

- $n$ is the number of classes
- $\tilde{y_i}$ is the model output for class $i$, or using a Softmax activation function, the probability for the input example to belong to class $i$.
- $y_i$ is the target value for class $i$

If we use a Softmax activation function the target value for the class to which the example belongs to is 1, while the target value for the other classes is 0. Talking about the Quantized Softmax Technique the target value for the bin to wich the input example falls into is 1, while the target value for the other bins is 0.

# J CRPS loss

The CRPS loss is defined in the following way:

$\int_{-\infty}^{\infty} (F(y) - \mathcal{H}(y - x))^2 dy$

where:

- $y$ is the target value
- $F(y)$ is the estimated cumulative distribution function
- $\mathcal{H}$ is the Heaviside step function (see appendix D).

The crps loss doesn't focus on a single point of the distribution, but allows to consider the all distribution at the same time.

# K Quantile Loss

The quantile loss can be defined as:

$L_\tau = max[\tau(y - z),\ (\tau - 1)(y - z)]$

Where:

- $y$ is the target value
- $z$ is the estimated quantile of order $\tau$

## References of the Images


[1] URL: https://medium.com/analytics-vidhya/machine-learning-intuition-9e20bf6cadf8.

[2] URL: https://medium.com/@sagar.rawale3/feature-selection-methods-in-machine-learning-eaeef12019cc.

[3] URL: https://medium.com/mlearning-ai/training-feed-forward-neural-network-ffnn-on-gpu-beginners-guide-2d04254deca9.

[4] URL: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53.

[5] URL: https://brilliant.org/wiki/feedforward-neural-networks/.

[6] URL: https://pythonmachinelearning.pro/complete-guide-to-deep-neural-networks-part-2/.

[7] URL: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[8] URL: https://clay-atlas.com/us/blog/2020/02/03/machine-learning-english-note-relu-function/.

[9] URL: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/24e59eb7-2e7f-4f5d-8624-e8796b617ac6/c1512698-fee2-4e87-b57b-2b8f86930b45/previews/sigmoid/html/sigmoid_documentation.html.

[10] URL: https://help.scilab.org/docs/6.0.0/en_US/tanh.html.

[11] URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.


## References


[12] Jos Diepeveen et al. 2015. URL: http://euroforecaster.org/newsletter21/07-21-23.pdf.

[13] Sylvain Arlot and Alain Celisse. "A survey of cross-validation procedures for model selection". In: *Statistics surveys* 4 (2010), pp. 40–79.

[14] Jay L Devore. *Probability and Statistics for Engineering and the Sciences*. Cengage learning, 2011.

[15] Tilmann Gneiting et al. "Calibrated probabilistic forecasting using ensemble model output statistics and minimum CRPS estimation". In: *Monthly Weather Review* 133.5 (2005), pp. 1098–1118.

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

[17] Keunhee Han, JunTae Choi, and Chansoo Kim. "Comparison of statistical post-processing methods for probabilistic wind speed forecasting". In: *Asia-Pacific Journal of Atmospheric Sciences* 54.1 (2018), pp. 91–101.

[18] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

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

[20] Kaiming He et al. "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[21] Nikhil Ketkar. "Stochastic gradient descent". In: *Deep learning with Python*. Springer, 2017, pp. 113–132.

[22] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[23] William H Klein and Harry R Glahn. "Forecasting local weather by means of model output statistics". In: *Bulletin of the American Meteorological Society* 55.10 (1974), pp. 1217–1227.

[24] Max Kuhn, Kjell Johnson, et al. *Applied predictive modeling*. Vol. 26. Springer, 2013.

[25] Yunjie Liu et al. *Application of Deep Convolutional Neural Networks for Detecting Extreme Weather in Climate Datasets*. 2016. arXiv: 1605.01156 [cs.CV].

[26] Agnes Lydia and Sagayaraj Francis. "Adagrad - An Optimizer for Stochastic Gradient Descent". In: Volume 6 (May 2019), pp. 566–568.

[27] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[28] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[29] Chigozie Nwankpa et al. "Activation functions: Comparison of trends in practice and research for deep learning". In: *arXiv preprint arXiv:1811.03378* (2018).

[30] Josh Patterson and Adam Gibson. *Deep learning: A practitioner's approach*. " O'Reilly Media, Inc.", 2017.

[31] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[32] Payam Refaeilzadeh, Lei Tang, and Huan Liu. "Cross-validation." In: *Encyclopedia of database systems* 5 (2009), pp. 532–538.

[33] Jimson Mathew Renji Remesan. *Hydrological Data Driven Modelling*. Springer, pp. 64, 65.

[34] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[35] Maurice Schmeits, Simon Veldkamp, and Kirien Whan. "Statistical post-processing of wind speed forecasts using convolutional neural networks". In: *EGU General Assembly Conference Abstracts*. EGU General Assembly Conference Abstracts. May 2020, 16849, p. 16849. DOI: 10.5194/egusphere-egu2020-16849.

[36] Koert Schreurs et al. "Precipitation Nowcasting using Generative Adversarial Networks". In: (2021), pp. 24–25.

[37] Benedikt Schulz and Sebastian Lerch. *Machine learning methods for postprocessing ensemble forecasts of wind gusts: A systematic comparison*. 2021. arXiv: 2106.09512 [stat.ML].

[38] US National Weather Service. *Numerical Weather Prediction (Weather Models)*. URL: https://www.weather.gov/media/ajk/brochures/NumericalWeatherPrediction.pdf.

[39] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[40] Nitish Srivastava et al. "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.

[41] David J Stensrud. *Parameterization schemes: keys to understanding numerical weather prediction models*. Cambridge University Press, 2009.

[42] Stéphane Vannitsem, Daniel S Wilks, and Jakob Messner. *Statistical postprocessing of ensemble forecasts*. Elsevier, 2018.

[43] Michaël Zamo and Philippe Naveau. "Estimation of the continuous ranked probability score with limited information and applications to ensemble weather forecasts". In: *Mathematical Geosciences* 50.2 (2018), pp. 209–234.

[44] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.", 2018.

[45] Kirien Whan Zoë Van Den Heuvel Elisa Perrone. "Post-processing wind gust forecasts using CNNs to improve weather warnings". In: Eindhoven University of Technology, 2021.