Transfer Learning for Rain Detection in Images

by

Jessica Alecci

in partial fulfillment of the requirements for the degree of

Master of Science
in Embedded Systems

at the Delft University of Technology,
to be defended publicly on Monday July 3, 2017 at 3:00 PM.

Student number: 4417623
Thesis committee:

Chair:	Prof. Dr. Alan Hanjalic	Faculty EEMCS, TUDelft
University supervisor:	Dr. Judith Redi	Faculty EEMCS, TUDelft
IBM supervisor:	Dr. Zoltan Szlavik	Center for Advanced Studies, IBM Netherlands
Committee member:	Dr. Christoph Lofi	Faculty EEMCS, TUDelft

TU Delft Delft University of Technology

Abstract

Extreme weather conditions seem to occur stronger and more frequently due to climate change. Very expensive technology is used to predict them, which results problematic when they have to be used in less developed countries. An alternative could be to employ digital sensors, such as phone cameras or existing webcam infrastructures, widely distributed in many countries in the world, and to analyse the captured images. Many methods have been proposed for weather detection including also Convolutional Neural Networks (CNNs). The latter have recently become very popular in the field of computer vision due to their excellent performance in image recognition tasks. However, CNNs are characterized by a high number of learnable parameters that need an equally high number of data points to achieve good performance. Since very big dataset are hardly available, techniques that help to overcome this problem, such as transfer learning, can be used. Different are the transfer learning approaches: the fine-tuning approach, i.e. re-training all the network layers, and the freezing layers approach, i.e. re-training just a subset of the network layers.

In collaboration with IBM Netherlands Center for Advance Studies, we optimized a ResNet-18 architecture, modifying the architecture depth and applying regularization methods, to perform weather detection of images showing rain or no-rain conditions. The architecture was previously trained on the ImageNet dataset and then, through the fine-tuning approach, we re-trained the network layers using as training data points weather images captured by webcams distributed in The Netherland, Belgium and London. In particular, we collected a dataset composed by 397041 images showing scenarios such as city roads, urban and rural areas.

We also adopted the freezing layer approach on an optimized ResNet-18 architecture and made comparison between the two approaches in relation to weather detection task.

Preface

The end of this thesis project marks the end of an amazing journey and at the same time the beginning of an even more exciting one. The period spent at TU Delft for my studies gave me the opportunity to know people from all over the world, who shared with me their own cultures, making me lived their countries with their stories, and with whom I shared my Italianness and the love for my land.

In IBM, where I carried out this thesis project, I had a wonderful and educational experience. I could experience the life in a big company environment and it helped me to understand the future directions I would like to follow.

I would like to thank the people that shared, directly or indirectly this journey with me. Not all of them speak English or, in some case, I just feel more comfortable expressing my gratitude in my mother tongue, hence I apologise in advance with the reader if some part of this preface will be written in Italian.

Ai miei genitori, per i sacrifici che hanno dovuto affrontare per via delle mie scelte e per il grande sostegno ricevuto durante questi anni. A voi che mi avete dato dei valori molto più preziosi di qualsiasi cosa materiale possa mai rappresentare e che mi hanno permesso di essere la persona che sono oggi. Grazie per avermi aiutato a raggiungere questo importante traguardo, nonostante le difficoltà esso abbia comportato.

A Roberto, il mio compagno di vita e il mio più grande sostenitore. Colui che supporta ogni mia scelta nonostante queste possano tenerci separati e che mi sopporta ogni giorno anche da lontano. Grazie per spingermi a fare sempre del mio meglio e a inseguire qualsiasi mia ambizione, ovunque mi porti.

Alla famiglia di Roberto, Francesco, Maria e Giovanni, per essere diventati come una seconda famiglia per me e per aver condiviso quest'esperienza con me. Grazie per il continuo sostegno che mi date e per i vostri instancabili incoraggiamenti.

A Judith, a te va il mio più sentito grazie per la tua preziosissima guida durante gli anni passati qui e per tutti gli insegnamenti trasmessimi non solo in ambito universitario. Sono stata molto fortunata ad avere un supervisore come te, che ha sempre saputo pazientemente indicarmi la via da seguire nonostante gli ostacoli incontrati e gli errori commessi.

To Zoltan, for all his valuable suggestions during my period in IBM and his guidance through this project. I am very grateful for the experiences you shared with me, which allowed me to live a little closer the work of an Engineer.

To Robert-Jan, for all the received "Good Job" while he was in IBM, but most of all for the great opportunity he gave me. I am very excited to start my next new adventure with you.

Last, but not least, to all the flatmates I lived with, to all the friends I found here and to all the people I met in IBM, thanks for having shared with me all or just part of this fascinating journey.

Jessica Alecci
June 2017

Contents

List of Figures

List of Tables

1

Introduction

For millennia human beings have been interested in the weather. Having a knowledge of the weather is important to agriculture, air and marine traffic, to control wildfires and so on. Weather forecasting started in the nineteenth century, based on the collection of quantitative data, and has gotten to impressive accuracy these days. Over the centuries, thanks to technology development, many different instruments and tools have been created to collect observations: Radar, Weather balloons, Satellites, Weather stations and others.

Nowadays, climate change is posing a new challenge to weather forecasting. Extreme weather conditions seem to occur stronger and more frequently [1]. These events could potentially generate floods or landslides, which could threaten human life, disrupt transport and communication, damage buildings and infrastructure or cause loss of crops and livestock. Employing current forecasting technology to predict extreme weather can be effective, but is certainly very expensive due to installation and maintaining costs [2], [3], which results problematic especially when they have to be used in less developed countries. Hence, there is a need to find alternative methods to those already existing, capable of predicting extreme weather conditions at reduced costs and possibly using already existing tools.

In IBM Netherlands Benelux Center for Advance Studies, research projects, part of a bigger project called RainSense [4], on innovative ways of weather sensing are ongoing. There have been experiments related to get data from people using an app, who were asked to take pictures of the sky and report weather conditions, and also experiments on gathering data from audio sensors implemented in umbrellas [5]. All this gathered data is then aimed to be combined with weather station data where available, creating labelled data to be used in the learning phase of machine learning models. Once such models are built, existing infrastructures could be used in places with no weather stations, thus improving the detection of the weather. Furthermore, a better understanding of the current weather could lead to better prediction models, such as those being used at The Weather Company [6].

Indeed, the multitude of digital sensors, and especially cameras, active in our world nowadays, could provide a valuable asset for the monitoring of extreme weather conditions. Millions of cameras (from traffic webcams to cell-phone cameras) take snapshots of outdoor weather conditions at every moment, finely covering vast areas. These images could be automatically analysed to detect bad weather conditions and, for example, generate alerts where needed.

Several are the approaches proposed in the field of computer vision to accomplish such a task. The most common approach involves the extraction of features from images to be used in image processing algorithms, [3], [7], [8] or machine learning techniques [9], [10], [11]. A feature is an individual measurable property able to describe the raw image; it has to be informative, discriminative and independent from the other features used to describe the same input [12]. Examples of features used in literature for weather recognition in images are based on brightness and contrast statistics, since these values tend to be higher in sunny images and lower in rainy and cloudy ones, or sharpness statistics, since in sunny images are sharper than in rainy ones, which result more blurred. However, weather detection from images has still large margins for improvement in terms of accuracy: identifying crucial features for weather detection is generally difficult, time-consuming and requires expert knowledge.

The need for manual feature engineering can be obviated by automated feature learning. Indeed, in the last years, the field of computer vision has radically changed thanks to the introduction of

Deep Neural Networks (DNNs) [13], in particular Convolutional Neural Networks (CNNs), which allow automatic learning of features from raw images. DNNs are multi-layered feedforward neural networks. They consist of simple processing units called artificial neurons, which are organized in layers. Neurons in each layer are connected, through weighted connections, to neurons in the next layers and this cascade of multiple layers is used to automatically extract features that become more complex as the number of layer increases, learning multiple levels of representations that correspond to multiple levels of abstractions. The weights represent the parameters used by the model to transform the input into output. The output of each intermediate layer can be considered as a representation of the original input data, being it a recombination of the inputs, produced by the previous layers, by means of these parameters, which are learnt in order to make this recombination useful to finally predict the right output from the input. Hence, to extract always more complex features a large amount of layers is required, which involves a corresponding increase in the number of parameters that need to be learnt.

In application that involve image analysis, the most used type of DNNs are CNNs, which are inspired by the organization of the animal visual system. CNNs are characterized by the presence in their architecture of at least one Convolutional layer, where each neuron is connected only to a small region, called receptive field or filter, of the layer before. This filter is slided across the entire input of the layer and a convolution operation between the input image and the filter is performed; to each local filter correspond a different hidden neuron in the corresponding hidden layer. What makes CNN different from DNN is that it takes into account the spatial structure of an image; indeed, looking at an image, neighbouring pixels in a region of the image are more likely to be related than pixels far away [14]. CNNs presents also drawbacks, one of this being the high number of meta-parameters that has to be optimally adjusted during the training phase in order to obtain better performance, as it will be explained in Chapter 2.

For the capabilities just mentioned, CNNs seems to be a very promising tool to employ for image analysis. Indeed, they showed to be able to achieve very high performance in image classification and object recognition [15], [16], [17], [18]. In collaboration with IBM Netherlands Benelux Center for Advance Studies, in this thesis project we propose to employ Convolutional Neural Networks for performing weather recognition from images.

1.1. Convolutional Neural Networks for weather detection in images

So far, CNNs have not been widely used for weather recognition. The challenge is twofold: on one hand, the recognition task is subtle, as weather conditions can be sometimes hardly visible in images. On the other hand, to the best of our knowledge, there is a limited amount of data available to train CNNs for this specific task. Indeed, one of the main drawbacks of CNNs is the large amount of data they need for training in order to avoid overfitting. Overfitting is a situation that occurs when the used model is too closely fit to a set of data, that results to be too limited compared to the number of parameters the model has to learn, and begins to describe random error or noise instead of its underlying relationship, loosing its generalization capabilities when tested on previously unseen data [19]. In CNNs this problem is mainly caused by the large amount and size of layers, which results in an equally large amount of parameters that need to be trained, as per the DNNs. In order to learn these parameters in non-trivial way is necessary to use a corresponding high number of training data.

Different are the techniques to contrast overfitting, but in case the available data are numerically scarce a convenient method to employ that is becoming always more popular is Transfer Learning [20]. Transfer Learning is a method to exploit previously acquired knowledge to learn a new task that is not necessarily related to the previous one, inspired by the capacity human beings have in re-using the knowledge they gained from past experiences to learn new tasks. In particular, in machine learning, when the amount of data available to learn a task T, mapping an input x into an output y through a model $f(x, \Theta)$, where Θ is the set of parameters to be learnt, is limited, overfitting could occur. However, if a task T', possibly similar to task T we aim at learning, was learned using a higher amount of data, we might expect that the parameters Θ', used for learning task T' could be re-used and adapted to learn task T. In image analysis that employ CNN, this approach is used always more often [21], [22], [23]. A reason is provided by the latest researches showing that CNNs, applied to various image analysis task, tend to extract similar basic features in the lower layers such as edges, color and contrast [20] which are then differently recombined in more complex features, specialized on the task at hand,

in the higher layers closer to the output of the network. Hence, in this scenario, it can be hypothesized that features learnt to perform specific tasks such as image classification or object detection could be re-used to learn other visual task such as weather detection from images.

In particular, different are the approaches to Transfer Learning: given a network, previously trained on a source task T_s, we could either retrain the whole network using the target training data or we could retrain just a subset of the network layers [20]. Which of these techniques should be applied depends on several factors: the most important are the dataset size and the similarity of the target task to the source one[24]. In presence of a small dataset and a similar target task is suggested to train just a small subset of layers, whereas if the target task is not similar, such as classifying text in one language as target task which was written in another language on the source task, it is suggested to train a bigger subset of layers. Finally, when the dataset is big, either the task is similar or not, it is possible to re-train the entire network.

Based on all these observations, in this research project, we explore the capabilities of state-of-art CNN. Our aim is to investigate methods, based on an existing CNN architecture, able to perform weather recognition, using as data images captured by existing webcam infrastructures. The choice of using existing webcam infrastructures is motivated both by their large distribution in mostly all the countries in the world due to the development of intelligent transportation system [25] and by the automatic process that can be implemented to retrieve them, hence avoiding relying on human intervention to capture images showing weather conditions, which could require a much large amount of time compared to an automatic approach. In particular, we are interested in detecting the presence of rainy weather in images. The rationale between this choice is twofold: First, detecting rain is, more than detecting e.g. cloudy conditions, of key importance for extreme weather prediction, especially when the target is to raise alerts against e.g. flooding. Second, given the scarcity of data and previous work on the matter, we chose to start from a relatively simple problem definition (classification of images into rainy and not rainy), than to trying to discriminate between different intensities of rain [9], which has been proven to bear a high degree of uncertainty in the task definition.

In order to train and evaluate a CNN to predict the presence of rain in images, we need a large dataset consisting of outdoor images and corresponding weather data. However, no public dataset that was both of big dimension and contained images showing various weather condition, captured from traffic webcams, was found, which lead us to collect these data by ourselves. Specifically, we will use a state-of-art architecture, ResNet [18], previously trained on the ImageNet dataset [26], a dataset composed by over 15 million labelled high-resolution images belonging to roughly 22,000 categories, with the purpose of object recognition. We will apply Transfer Learning techniques so to be able to still benefit from the power of large and complex CNNs, such as ResNet, despite the limited amount of data available to train them. In particular, we will apply both the Transfer Learning techniques mentioned above, i.e. re-training the entire network and re-training just a subset of the network layers. The latter choice is motivated by the fact that the collected dataset is of medium size, which would make it not big enough to re-train the entire network and not small enough to not attempt this approach, especially considering that the target task is completely different from the source one, as stated in [27], making us wonder if re-training the whole network would be more beneficial in terms of performance.

Our main goals, then, are to:

1. Collect a dataset containing images showing various weather conditions from existing webcam infrastructures, along with the corresponding labels indicating the specific weather condition and whether the image depicts rain or not.

2. Propose a state-of-the-art CNN architecture optimization for solving our target task.

3. Compare different Transfer Learning approaches, along with the corresponding labels indicating the specific weather condition and whether the image depicts rain or not.

1.2. Research objectives

As discussed before, there is very little ongoing research addressing weather recognition employing CNNs. With an aim to explore more this possibility and at the same time have an insight into different Transfer Learning techniques, this research project addresses the following research question and sub-question.

Is it possible to recognize rain in images employing a state-of-the-art Convolutional Neural Network?

- Given the difficulty of the task and the lack of a huge dataset, can Transfer Learning help in achieving better performance?

To answer this research question we define three objectives:

Objective 1 **Exploring the existing work on weather detection as well as the Deep and Transfer Learning literature towards framing a solution for CNN-based weather detection in images.**
Our aim is to understand the nature of the problem and examine the work that has already been done. A literature study on Deep Learning and Transfer Learning guides us through the set of available tools to achieve our research goals and help us to understand the advantages and the disadvantages these tools involve. A literature study of previous work on weather classification help us to understand the challenges this task presents and the various solutions that were adopted by other researches. This objective is linked to the literature study presented in chapter 2.

Objective 2 **Collecting a large dataset of labelled images representing diverse weather conditions.**
As currently no sufficiently large dataset exists including images of diverse weather conditions, to enable the training of a large convolutional neural network we need to create a new one of our own. Our aim is to create an appropriate dataset containing weather images collected from existing webcams infrastructure so to be able to train and evaluate the chosen architecture. This objective is linked to the dataset described in chapter 4.

Objective 3 **Proposing a CNN architecture optimization.**
Our aim is to propose a CNN architecture optimization for the weather classification task through design space (i.e. parameters of the training process and CNN architecture) exploration of CNN. To fulfil this objective we will apply different Transfer Learning techniques: retraining completely the entire architecture or retraining just a subset of the network layers. This objective is related to chapter 5

1.3. Contributions

This thesis project was carried out at IBM Netherlands Center for Advance Studies, in the period between February 2016 and May 2017, and provides the following contributions:

1. **Weather dataset**
   We created a new dataset containing 397041 labelled images showing different weather conditions with different backgrounds at different day time, which can be required at the following email address: casbnl@nl.ibm.com. However, to the best of our knowledge, this is the first dataset composed by such a big amount of labelled images captured from existing webcam infrastructures that could be used by other researchers for future works.

2. **An optimized CNN architecture using Transfer Learning to detect rain images**
   We propose an optimized state-of-the-art CNN architecture, trained resorting to Transfer Learning techniques, able to discriminate between images presenting rain or not, with an average accuracy of 63% percent.

1.4. Thesis outline

The remainder of this thesis is organized as follows. In chapter 2, we present the literature review in which the field of deep learning is explored along with previous work related to weather conditions classification. In chapter 3 we describe our research methodology. The details on the creation of the dataset are presented in chapter 4. We then move on to the main chapter of this report, chapter 5, where we present our design space exploration and the outcomes regarding the comparison of various Transfer Learning techniques. Finally, in chapter 6 we draw conclusions and provide recommendations for future work.

2

Background and Related Work

This chapter presents the literature review on which this thesis project is based on. In Section 2.1 we present an overview of past works on weather detection. In Section 2.2 we discuss in depth Convolutional Neural Networks and describe, in Section 2.4, the specific CNN that will be used in this thesis project and finally, we conclude this chapter with a review of transfer learning techniques in Section 2.5.

2.1. Automatic weather detection from images

There have been various attempts to detect particular weather conditions from videos or images using both image processing and machine learning. In reviewing them, we start by investigating the main properties characterizing rain, which describe it and make it recognizable in pictures and videos. The visual effects that rain produces in images are complex, its appearance depends on several factors such as its physical properties, the environment and the camera settings.

Rain can be described as a "dynamic weather condition characterized by a collection of randomly distributed water droplets of different shapes and sizes that move at high velocities" [28]. Its visibility in an image depends on the drop distribution and velocities, the environment illumination and background scene and the intrinsic parameters of the camera, such as exposure time and depth of field. It increases when the drop size is bigger and decreases with the brightness of the background scene [29].

Raindrops behave like a transparent sphere, they reflect and refract light from the environment towards the camera, resulting in time varying intensity fluctuations in images and videos when they fall at high velocities. These intensities are motion blurred and dependent on the background due to the limited camera exposure. Hence, it is possible to visualize rain thanks to a combined effect of the dynamics of rain and the photometry of the environment [28].

In the field of image processing there are various works addressing rain detection. In [3] the authors detect the presence of rain or snow in images collected by video cameras. They separate the foreground from the background, using a mixture model, to be able to detect a dynamic weather condition such as rain from the foreground. The latter is indeed used to select potential blobs of rain through the employment of a Histogram of Orientation of Streaks.

The authors of [7] employed image processing techniques for measuring rainfall along with the raindrop size. They used a slow motion camera to capture very high quality images on which various data transformation were applied. They converted RGB images to gray ones on which threshold method was later applied to separate, as in [3], the foreground from the background obtaining binary images. These transformations aimed at visualizing raindrops on the foreground in order to detect them and measure their size.

Image processing techniques were also used in [8] to measure drop sizes from an irrigation spray system. They used a high resolution and high speed camera to capture images, which were then digitized through a scanner. Their research highlight the difficulty of detect water drop with a really small diameter due to the quality of the image.

In the field of machine learning, some authors employed Support Vector Machines (SVM) to classify various weather condition. In [9], the authors used a SVM to process image features so to classify images

captured by a Driver Assistance System in clear weather, light rain and heavy rain. They collected a dataset of 150 video sequences, from which they derived 500000 images, showing expressway, rural and urban environment. The feature of interest were: local contrast, minimum brightness, sharpness, hue and saturation. Their work shows that light rain condition is more difficult to recognize compared to the other two and that the most uncertain decision is between light and heavy rain.

Also in [10] the authors used a SVM to classify image features in sunny, cloudy and overcast. Interesting in this work is the preprocessing phase that precedes the actual classification. In this case, the sky region is extracted from the input images to avoid disturbance of the non-sky region. From the obtained image, the authors select automatically a subset of features from a feature pool through a Multiple Kernel Learning. The dataset used, composed by 1000 images, is collected from a fixed location.

Another classifier used to recognize images captured by vision system in vehicles was presented in [11]. The algorithm is based on Real AdaBoost, a powerful algorithm often used in pattern recognition, and discriminates between sunny, cloudy and rainy weather condition. They used a dataset composed by about 2500 images extracted from the videos captured by the video system in vehicles representing city streets, highways or overpass. As features to analyse, they selected hue, saturation, brightness, gradient amplitude and the mean of gray value calculated in several points within a region of interest in the image.

Surprisingly, despite its success with image analysis tasks, deep learning has not been frequently used for weather detection in images. Indeed, we found just few works.
In [27] the authors perform a weather classification using a slightly modified version of AlexNet [15], i.e. using a binary classifier to recognize sunny and cloudy images, previously trained on the ImageNet dataset. They re-trained the classifier using a dataset composed by 14000 images showing sunny or weather conditions. They analysed the network layer by layer, applying a SVM on each of them to study the differences between the object recognition task and the weather classification task. Their findings show that the classification accuracy improves in the high-level layers. This can be attributed to the fact that low level layers are good in recognizing edges, corner points and primitive shapes, while high level layers mostly capture abstract and task-specific information. Indeed, weather images are affected by many factors such as illumination, reflection, scene and shadow, rather than shape or texture which are object-related information.

Another study aims at recognizing extreme weather conditions [25]. The authors employed a GoogleNet architecture [17] to recognize four categories: sunny, rainstorm, blizzard and fog. They firstly pre-trained the network on the ImageNet dataset and then they fine-tuned it on a dataset they collected. The dataset is composed by about 17000 images showing complex scenes such as city roads, highway or aerial photography collected from the web.

2.2. Convolutional Neural Networks

Image classification is an important task in the field of computer vision and refers to the assignment of one label, from a number of predefined categories, to an input image [30]. This task has proved to be a complex and challenging problem for computers since interpreting images properties from pixel values is non-trivial due to the many variations an image can be subjected to. For example, in the case of object detection, the same object can be captured in different positions or its appearance may vary when exposed to different illumination conditions. However, for machine vision systems is very difficult understanding that the object is the same despite the different way it can be represented. Sophisticated understanding of the data, nearly human-level, developed over human knowledge and past experience, would be necessary to solve this kind of problem. Since explicitly programming a computer to do this is a non-trivial task Machine vision researchers have heavily drawn from Machine Learning, which allows computers to learn from experience, by example and by analogy automatically [31].
In supervised machine learning problems, given an input x, such as an image, we want to predict a label y that describes this input (e.g., the object depicted in the image) using a model $y = f(x, \Theta)$. This model is not known a priori, hence our aim is to use a generic model, described through a set of parameters Θ , that will be then specialized on the target task. In order to do this, we can apply a supervised learning approach which consists in presenting the model with a set of examples of input-label pairing $x - y$ and (iteratively) update its parameters so that the obtained output is as close as possible to the original associated label. To evaluate the difference between the label $\hat{y}$ predicted from

the model and the desired one y, a loss function $L(y - \hat{y})$ is used. The goal of the learning process is to select the parameter Θ values that minimize such function. The method used to adjust the parameters is known as optimization method of which an example is the family of Gradient Descent algorithms which will be introduced in the next section.
One of the most popular approaches to machine learning are Artificial Neural Networks (ANNs), which have been primarily inspired by biological neural systems [32]. An ANN consists of a number of simple and interconnected processors, called neurons, that are connected by links associated with a numerical weight, with each other. The weighted link between two neurons A and B of two adjacent layers expresses the strength of the influence of A on B. An artificial neuron is a mathematical function that operates a weighted sum on the inputs it receives to produce an output. This sum is then passed to a function called activation function, whose role is to introduce non-linearity in the model. The need for non-linearity is due to the necessity to deal with real world problems that are usually non-linear, hence linear functions are often insufficient to model them.

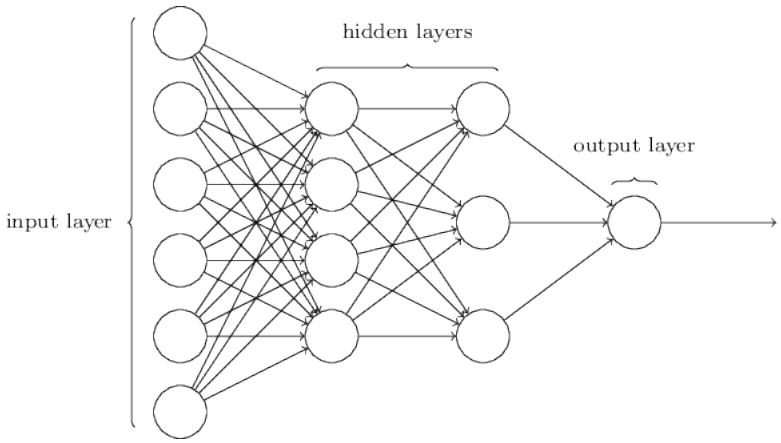

Figure 2.1: General ANN architecture

In Figure 2.1 a general ANN architecture is shown. It consists of one input layer, through which the input data x (e.g. the pixel values of an image) is entered into the model, one or more hidden layers and one output layer which outputs an estimate $\hat{y}$ of the desired output y. The input is propagated forward, layer by layer, until the output layer. A common method for training a neural network is the backpropagation algorithm, used along with an optimization method. The backpropagation algorithm is used to compute the gradient of the loss function, with respect to the network parameters Θ, starting from the output layer and going backwards to the first hidden layer, computing the gradients of each layer's output. These gradients are used by the optimization method to update the parameters in order to minimize the loss function.

When there is more than one hidden layer within the network, the architecture takes the name of Deep Neural Network (DNN). Among several Deep Learning architectures, Convolutional Neural Networks are one of the most popular and most promising tools with respect to image classification and analysis.[13]
CNNs are a type of Deep Feedforward Neural Networks specialized in visual recognition tasks and also widely used for natural language processing, video analysis, speech recognition and so on [33]. Their architecture is inspired by the structure of the animal visual cortex. In the visual cortex small regions of neurons are sensitive to specific regions of the visual field and respond to specific features such as edges [34].

As the name suggests, the network employs a convolution operation in a convolutional layer, which acts as a detection filter for the presence of specific features or patterns present in the original data. Rather than being assigned a priori as in traditional image processing, the parameters of such filters are learnt based on the training data and specialized to solve the task at hand. It has been shown that the lower layers in a CNN are able to detect features that are usually general for each image recognition task such as edges and curves [20]. As the number of layers increases the network gets specialized in detecting features that are more abstract and related to the specific target task [20]. Indeed, a key characteristic of CNN is sparse interaction, a property that concerns the indirect interaction that units

in deeper layers could have with a large portion of the input, describing complex relationships between many variables even when they are not directly connected.

CNN are also very appealing for their ability of reducing the number of parameters to be learnt, compared to the case of fully-connected networks, using a technique called parameter sharing. It is a technique based on the assumption that if a feature is useful to compute at some spatial position, then it should also be useful to compute at a different spatial position. Hence, only one set of parameters is learned for every location instead of separate ones.

2.2.1. CNN Architecture

A CNN architecture consists of layers whose neurons are arranged in 3 dimensions along width, height and depth. In image classification tasks, the model takes as input an image represented, in the case of a RGB image, by a 3D matrix with width and height of the image, and depth of 3 to account for the R, G and B channels. By forwarding the input through the network layers, it is eventually transformed into a 1-D output vector with dimensionality corresponding to the number of classes among which the network is able to discriminate. Usually, there are three main type of layers that are arranged in sequence to compose the CNN, which are the Convolutional layer, the Pooling layer and the Fully Connected layer.

In Figure 2.2 an example of CNN architecture is shown and in the next sections we will explain more in depth the functions of each building block.

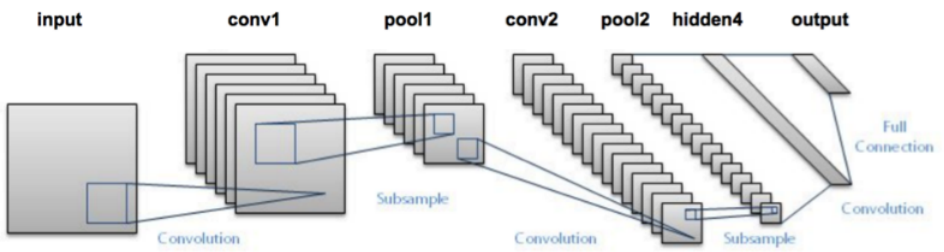

Figure 2.2: Example of CNN architecture

Convolutional Layer

The Convolutional Layer is the main building block in a CNN.

The parameters of this layer are learnable filters, 3D matrices of numerical values, which, in terms of dimensions, are spatially smaller than the input ones. Width and height of the filter depend on a design choice, while their depth is fixed by the number of input channels, which are the number of 2D inputs in the layer. During the forward pass, these filters are slided across width and height of the input at any position. The value with which the filter is slided across the input is called stride. The operation of slicing the filters is mathematically translated into a dot product between the filter and the input at any position. The result will be a 2D output that takes the name of activation map, which will be stacked along the depth dimension, together with the other activation maps, to form the output volume. The spatial size of the output can also be controlled by using zero-padding, a technique of adding zeros around the border of the inputs.

In equation 2.1 the output of the i^{th} filter, denoted by y_i^l, for a convolutional layer l with a total number of C filters is reported.

$$y_i^l = s(\sum_{j=1}^{C^{l-1}} f_{i,j}^l * y_i^{l-1} + b^l) \tag{2.1}$$

b^l is the bias vector of layer l, $f_{i,j}^l$ is the i^{th} filter of the convolutional layer l that is connected to the j^{th} feature map of layer $l-1$, and s is the employed activation function.

During the backward pass a convolution operation is also performed, but filters are spatially flipped along both axes (width and height). The parameters $f_{i,j}^l$ learnt by the network are updated through backpropagation; by doing so the network is learning various types of filter that are specialized in solving the specific task at hand.

Pooling Layer

A Convolutional layer might be followed by a Pooling layer. The main function of the Pooling Layer is to reduce the spatial size of the input, operating independently on every depth slice. It is a non-parametric layer consisting of filters that slide, with a pre-set value of stride, across the input layer to produce the output [13]. Different can be the functions that the filters are able to perform; the most used are:

- Max Pooling: it takes the maximum value of the input within the filter size.
- Average Pooling: it takes the average value of the input within the filter size.

Fully Connected Layer

Usually, at least one Fully Connected (FC) layer is present in CNN, placed before the network output in order to convert the computed features in class scores.
In this layer each neuron is connected to all the neurons in the layer before it. Its function is to learn parameters (weights and biases) to map the layer input into the correspondent output. The output y^l for a FC layer l is reported in equation 2.2

$$y^l = s(y^{l-1} * W^l + b^l) \tag{2.2}$$

W^l and b^l are the weight and the bias vectors of layer l, and s is the employed activation function. Contrary to convolutional layers, FC layers do not support parameter sharing. This results in a substantial increase of the learnable parameters within a CNN.

Activation Function

In order to introduce non-linearity in the network, so to help in learning more complex functions, we use activation functions. Different activation functions exist, however the most popular is the Rectifier Linear Unit (ReLU). ReLU, shown in Figure 2.3, is a function that thresholds the activation at zero. It is described by Equation 2.3, where z is the input to a neuron.

$$s(z) = max(0, z) \tag{2.3}$$

Being nearly linear allows ReLU to preserve the properties of linear models that makes them easy to optimize through gradient-based algorithms. Furthermore, it is easy to implement and greatly accelerate the convergence of the optimization methods [13].

Another popular activation function, shown in Figure 2.4, is the softmax function, which is often used in the output layer of a neural network to model the probability distribution of the output over multiple classes. It is described as in Equation 2.4, where z is a vector of the input to the output layer and $j = 1, ..., K$ indexes the output neurons, with K number of classes. The value associated to each output neuron, corresponding to the probability that the input signal belongs to the class represented by it, is bounded between 0 and 1. Hence, the sum of all the output values is constrained to 1. Bounding the total sum of the probability values to 1 means that increasing the output value of one class makes the others decreasing, so to be able to identify an unique output, preventing two output neurons from being associated with the same value.

$$s(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.4}$$

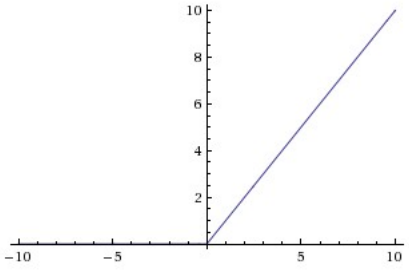

Figure 2.3: ReLU Activation Function

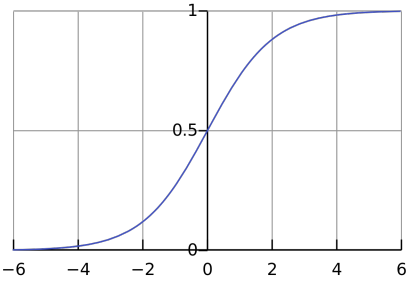

Figure 2.4: Softmax Activation Function

2.2.2. CNN Training

The training phase in CNNs is non-trivial since we are dealing with big architectures. Hence, we introduce various elements that are relevant for the training of CNNs and that will help to understand better the experiments performed in this thesis.

Gradient Descent Optimization

The Gradient descent method is one of the most commonly applied algorithm to optimize neural networks. It aims at minimizing the loss function by finding the direction in which it decreases the fastest through directional derivatives. It is generally used with backpropagation algorithm to determine the extent of the update of network weights at each iteration [35].

Based on the amount of data used to compute the gradient of the loss function, different variants of the gradient descent algorithm exists:

- **Batch gradient descent:** Computes the gradient of the loss function with respect to the network parameters for the entire training dataset. Due to the fact that the parameters are updated just once after calculating the gradients for the whole dataset, this method can be very slow and may require many iterations before converging. Equation 2.5 gives the mathematical expression according which parameters Θ are update at each iteration.

$$\Theta_t = \Theta_{t-1} - \eta * \nabla_\Theta L(\Theta_{t-1}) \tag{2.5}$$

  $\nabla_\Theta L(\Theta)$ is the gradient of the loss function $L(\Theta)$ with respect to the parameters, η is the learning rate, which determines the size of the steps we take to reach a (local) minimum.

- **Stochastic gradient descent (SGD):** Computes the gradient of the loss function with respect to the network parameters for each input-label pair, $x^{(i)} - y^{(i)}$, from the training dataset, increasing learning speed. However, it performs frequent updates with a high variance that cause the objective function to fluctuate heavily, which on one hand could help in escaping local minima but on the other hand could prevent the network from converging to the exact minimum since it will keep overshooting.

$$\Theta_t = \Theta_{t-1} - \eta * \nabla_\Theta L(\Theta_{t-1}; x^{(i)}; y^{(i)}) \tag{2.6}$$

  Θ are the network parameters, $\nabla_\Theta L(\Theta)$ is the gradient of the loss function $L(\Theta)$ with respect to the parameters, η is the learning rate

- **Mini-batch gradient descent:** Computes the gradient of the loss function with respect to the network parameters for a subset of the training dataset including N patterns, $x^{(i,i+N)}; y^{(i,i+N)}$, called mini-batch. In this way It helps to reduce the variance of the parameter updates, leading to a more stable convergence.

$$\Theta_t = \Theta_{t-1} - \eta * \nabla_\Theta L(\Theta_{t-1}; x^{(i,i+n)}; y^{(i,i+n)}) \tag{2.7}$$

  Θ are the network parameters, $\nabla_\Theta L(\Theta)$ is the gradient of the loss function $L(\Theta)$ with respect to the parameters, η is the learning rate

There are several variations of the gradient descent algorithm, which differ in the way network parameters are updated, to make it converge faster or in a more steady way.

- **Momentum:** Is a method that helps to accelerate gradient descent algorithms, gaining faster convergence and soften oscillations. This can be necessary due to the problems that these algorithms encounter when the surface of the loss function curves more steeply in one dimension than in another. The momentum term increases for dimensions whose gradients point in the same direction and reduces updates for dimensions whose gradients change directions. Equations 2.8 and 2.9 give the mathematical expressions for Momentum update rule.

$$v_t = m * v_{t-1} - \eta * \nabla_\Theta L(\Theta_{t-1}) \tag{2.8}$$

$$\Theta_t = \Theta_{t-1} + v_t \tag{2.9}$$

v is the momentum variable at time step t, initialized to zero at the beginning, and $m \in [0, 1]$ is the momentum coefficient, which is useful to dampen the velocity and oscillations in the system.

- **Nesterov Momentum:** Is an improved version of the Momentum method. It takes into account the next position of the parameters to calculate the gradient with respect to the approximate future position of the parameters. Equations 2.10 and 2.11 gives the mathematical expressions for Nesterov momentum update method.

$$v_t = m * v_{t-1} - \eta * \nabla_\Theta L(\Theta_{t-1} + m * v_{t-1}) \tag{2.10}$$

$$\Theta_t = \Theta_{t-1} + v_t \tag{2.11}$$

v_t is the momentum variable at time t, which is initialized to zero at the beginning and $m \in [0, 1]$ is the momentum coefficient.

Reducing Overfitting

Overfitting is a serious problem in networks containing multiple hidden layers such as DNNs. It is defined as the use of models or procedures that violate Occam's razor, which implies that any given complex function is a priori less probable than any given simple function, by using more adjustable parameters than are necessary or by using more complicated approaches than are necessary [19]. Indeed, this is mainly caused by the complex relationship between the input and the output that the model would be able to learn by adjusting parameters but that, with limited training data, compared to the number of network parameters, results to be overspecialised on the training data. Hence, this relationship exists in the training data but not in the test ones, and the model loses the ability to generalize on previously unseen data [35].

Different methods have been proposed to deal with this problem such as regularization, a process that introduces a penalty to the loss function. This penalty term is usually a function of the weights, indeed regularization helps in limiting their growth to achieve a more stable model. There are various regularization techniques of which the most popular are:

- **L1 Regularization:** is a regularization techniques that leads the weight vectors to become more sparse during optimization. In this way the network will prefer a sparse subset of its most important inputs becoming nearly invariant to noisy input. It is implemented as per equation 2.12

$$L = L + \lambda * |\Theta| \tag{2.12}$$

$|\Theta|$ is the sum of the absolute values of all the network weights present in a network and λ is the regularization strength that decides the contribution of term $|W|$ in the cost function.

- **L2 Regularization:** is the most commonly applied regularization method. It penalizes peaky weight vectors preferring diffuse ones so to avoid the network to prefer some of its inputs over the others. Furthermore, employing L2 regularization during gradient descent parameter update leads every weight to linearly decay towards zero. It is implemented as per equation 2.13

$$L = L + \lambda * \Theta^2 \tag{2.13}$$

Θ^2 is the L2 norm of the parameter present in a network and λ is the regularization strength that decides the contribution of term W^2 in the cost function.

- **Dropout:** Although this technique is not a regularization method per se, it acts as one. It randomly drops units and their connections, with a probability p, from the neural network during training to prevent them from co-adapting too much, encouraging each hidden unit to create good features without relying on the other ones to adjust its mistakes [36]. Applying dropout means training a different architecture at every training iteration updating the network parameters based on the average computation of all these architectures. This results in learning a more robust model.

  At test time, all the network connections are restored and their weights are multiplied by the probability value with which they were switched off.

- **Batch Normalization:** Although, as Dropout, Batch Normalization is not a regularization technique per se, it helps in regularizing the model and reduce the need for Dropout [36]. To work, it requires a mini-batch of size greater than one, hence it is not suitable for Stochastic Gradient Descent algorithm. The network parameters typically make the layer activations too big or too small, indicated as "internal covariate shift" in [37], which is tried to be avoided through Batch normalization method by forcing the activations passing through the network to take a unit Gaussian distribution, with zero mean and unit variance, across the mini-batch. Typically, it can be applied over the existing layers of the network, but can be implemented over few selected layers as well. Batch normalization has been shown to help in improving performance even though it is responsible for possible increases in the computational demand due to the mathematical operation it performs. Equations 2.14, 2.15, 2.16 and 2.17 gives the mathematical expression for Batch Normalization method.

$$\mu_B = \frac{1}{N} * \sum_{i=1}^{N} x \tag{2.14}$$

$$\sigma_B^2 = \frac{1}{N} * \sum_{i=1}^{N} (x - mu_B)^2 \tag{2.15}$$

$$\hat{x} = \frac{(x - mu_B)}{\sqrt{\sigma_B^2 + c}} \tag{2.16}$$

$$y = \gamma * \hat{x} + \alpha \tag{2.17}$$

  x and y are the respective inputs and outputs of Batch Normalization layer, N is the size of a mini-batch, and μ_B and σ_B^2 are the computed mean and variance across the mini-batch respectively. In equation 2.16, the input is normalized to have zero mean and unit variance across the mini-batch, and c is a constant used in order to prevent division by zero. In equation 2.17, the output of the Batch Normalization layer is computed by scaling the normalized input by γ and shifting it by α, which are both learnable parameters.

  Usually the running mean and variance is calculated during the training process since the test dataset could not be expected to have mini-batches. Their obtained values are instead used in equation 2.16. In alternative, it is possible to compute the average mean and variance values across the mini-batches in a post-training run.

2.3. CNN overview

In the last few years CNNs have become always more popular for solving computer vision problems, as it can be seen from the winner architectures of the last years ImageNet Large Scale Visual Recognition Challenges (ILSVRCs) [26]. This challenge evaluates algorithms for object detection and image classification involving the recognition of 1000 classes.
The first CNN winning the ILSVRCs, which also made CNNs very popular, was the Alex-net architecture [15] in 2012. This architecture is composed of 5 convolutional layers, max-pooling layers, dropout layers and 3 fully-connected layers as shown in Figure 2.5 and employs ReLU as activation function. It obtained a top-5 error rate of 15.6%, which is the error in classifying an image within the closest 5 classes.

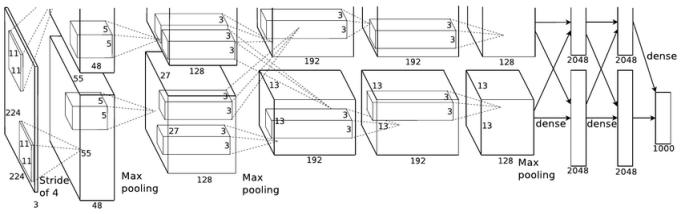

Figure 2.5: AlexNet architecture

AlexNet was then improved in the next year by the authors of [21] tweaking its parameters and achieving a top-5 error rate of 11.2%.

In 2014, VGGNet [16] even though it did not win the competition, showed that it was possible to reduce the number of parameters and at the same time to increase the depth of the network, achieving better performance than the architecture mentioned above with an error-rate of 7.3%. This architecture is composed by more convolutional layers than AlexNet, 13 exactly, which are smaller in terms of filters dimensions leading to a reduction of parameters but being able to learn more high-level features than previous CNN.

Another very important architecture, winner of the ILSVRC 2014 with an error rate of 6.7%, is GoogleNet [17]. This architecture changes the way of structuring CNN architectures, which basically stack single layers one upon each other sequentially, introducing the inception module. The architecture is modularized and the main block is the inception module, which is composed by convolutional layers that are arranged in parallel. Another newelty they introduced is the elimination of the fully connected layer from the architecture, which allowed to save a huge number of parameters, almost 12x fewer than the AlexNet architecture.

Finally, the winner of ILSVRC 2015, was ResNet [18], the deeper architecture, with its 152 layers, winning the competition with the lowest error rate ever achieved of 3.6%, presented in detail in Section 2.4.

2.4. Deep Residual Networks

Among the reviewed CNN architectures in Section 2.3, proved to be the most performing ones in terms of image classification, the best one results to be ResNet, not only because of the great results achieved but also because, as it will be shown,the way it is structured allows to achieve deeper architecture with a reduced number of parameters [18].

Deep Residual Network, better known as ResNet, is a deep architecture that exploits the concept of residual learning[18]. The authors presented a deep residual learning framework whose architecture has been inspired by the philosophy of the VGGNet [16].

The novelty in this network is the addition of a shortcut connection every two layers to a VGG-style network as shown in Figure 2.6. This architecture allows the layers to learn a residual mapping. In other words, formally denoting the desired underlying mapping between few stacked layers (two in the example in Figure 2.6) as $H(x)$, they let the stacked non-linear layers fit another mapping, $F(x) := H(x) - x$, which can be rewritten as $H(x) := F(x) + x$. The latter formula can be realized, indeed, by applying shortcut connections to skip one or more layer.

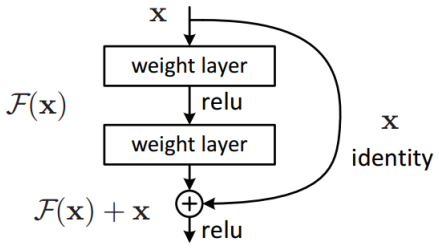

Figure 2.6: Example of Residual Unit

In the following subsections the main building blocks composing ResNet are described.

2.4.1. Architecture

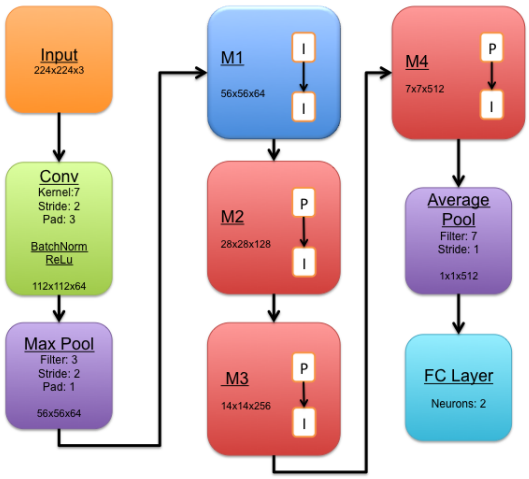

Figure 2.7: ResNet General Representation

The architecture of ResNet can be thought as composed by different units, called Residual units, composed by blocks, some of which are replicated multiple times throughout the network. The number of repetition of the blocks within a ResNet unit depends on the depth of the architecture.

A general architecture can be seen in Figure 2.7. In Figure 2.8 and Figure 2.9 the building blocks composing the ResNet units are shown. Both of them are composed by a sequence of convolutional layers, mostly composed by 3x3 filters and stacked one upon each other. The number of filters is decided according to the block's output feature map size: if the output feature map sizes are the same, the layers have the same number of filters, otherwise, if the feature map size is halved, then the number of filters is doubled. The block shortcuts can either perform an Identity mapping, as in Figure 2.8 or a convolution operation, as in Figure 2.9. In particular, if the input and the output dimensions are equal an identity mapping is performed, otherwise a convolution operation, to perform downsampling using a 1x1 filter and applying a stride of 2, is carried out. This is due to the fact that when the dimensions are equal there is no need to resize the input in order to match with the output, as it happens in the other case. The blocks shown in Figure 2.9 are used to link the ones in Figure 2.8 with each other.

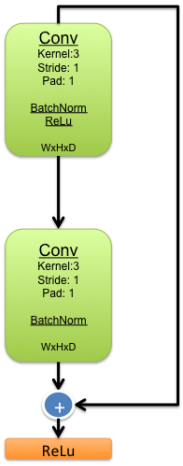

Figure 2.8: ResNet Identity Block with Identity Mapping Shortcut

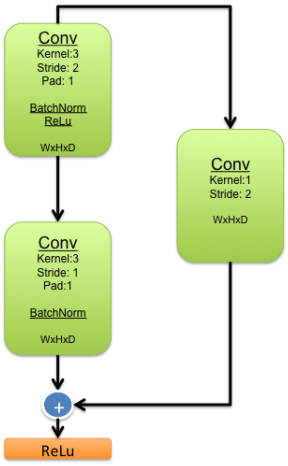

Figure 2.9: ResNet Projection Block with Convolutional Shortcut

There are other type of layers composing ResNet such as the Pooling layers, specifically a max Pooling layer, placed at the beginning of the architecture, and an average one, placed at the end of the last ResNet unit. They are both used to halve the size of their input. Furthermore, a final 1000-way FC layer with softmax function is used for classification.

Several were the ResNet architectures presented by the authors, which were different for the value of their architecture depth, i.e. the number of parametrized layers composing the network. Specifically, for the purpose of this research project, we adopted the smallest architecture, ResNet-18, driven by the small amount of parameters to train compared to the others. In Table 2.1 the structure of the architecture ResNet-18 is described. In this architecture each ResNet unit is composed by two blocks, which can be either different or equal. The blocks performing Identity mapping through their shortcut will be identified as "Identity", while the ones performing the convolution operation as "Projection".

ResNet Module N.	**Module Composition**	**ConvLayer Output Dimensions (WxHxD)**
1	Identity Identity	56x56x64
2	Projection Identity	28x28x128
3	Projection Identity	14x14x256
4	Projection Identity	7x7x512

Table 2.1: ResNet-18 Composition

ResNet architectures were further improved in a follow-up work, [38]. The authors aimed not only at improving their performance, but also at better understanding the behaviour of the network when slight changes were made. Their work was focused on a deep understanding of the two kind of shortcut connections which resulted in determining that the shortcut performing Identity mapping achieves faster error reduction and lowest training loss, while the one performing downsampling through convolution operation leads to both higher training error and loss. In order to improve the performance of the network, the authors carried various experiments on the usage of the activation functions. They adopted Batch Normalization after addition which caused a decreasing in the perfomance due to the alteration Batch Normalization made on the signal passing through the shortcut, impeding information propagation. Another attempt was made by moving ReLU before addition, which led to non-negative output affecting the representational ability of the network since the output of a Residual unit is expected to range in the interval $(-\infty, +\infty)$. Finally, they obtained their best result when they adopted Batch

Normalization and ReLU before addition so to act as pre-activation instead of post-activation functions, as shown in Figure 2.10. This modified architecture was claimed to be able to reduce overfitting, easing the training phase and improving generalization.

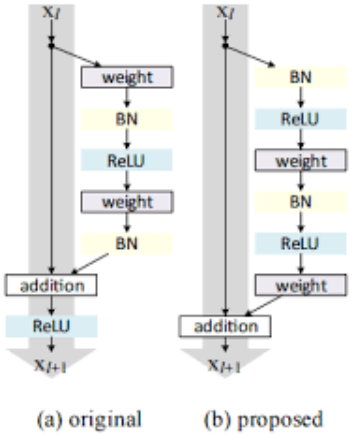

Figure 2.10: Full Pre-Activation on ResNet

The ReLU activation function, located after the element-wise addition, was removed and after the last ResNet unit, before the Average Pooling layer, extra activation was placed.
The new proposed architecture and the corresponding Identity and Projection blocks are respectively shown in Figure 2.11, 2.12 and 2.13.

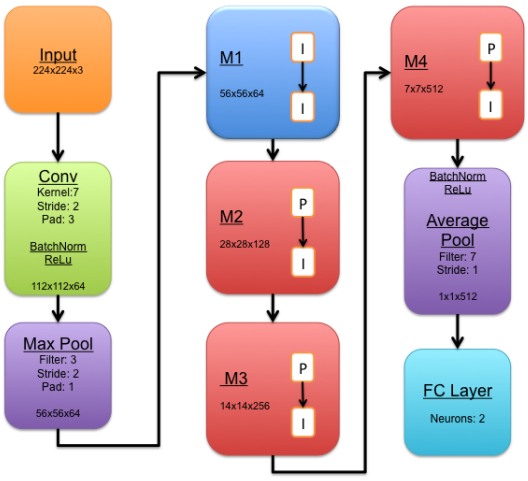

Figure 2.11: ResNet architecture with Full Pre-activation

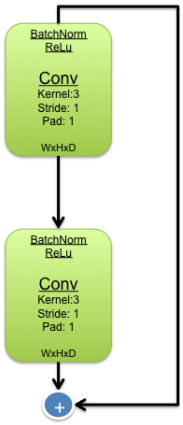

Figure 2.12: ResNet Identity block with Full Pre-activation

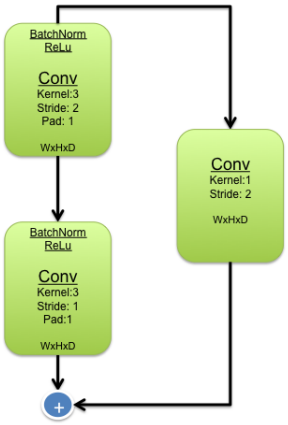

Figure 2.13: ResNet Projection block with Full Pre-activation

The improved performance of the modified architecture were attributed to the Batch Normalization's regularization effect. Contrary to the original version of ResNet, all the inputs to weight layers were normalized. Indeed, in the original version, the signal was normalized by the Batch Normalization but it was after added to the shortcut signal, hence the resulting merged signal was not normalized.

2.5. Transfer Learning

Training an entire CNN from scratch can be cumbersome, as it is relatively rare to have a sufficient big dataset that allows the network to converge, which is also one of the main cause of overfitting. To cope with this problem, is becoming always more common to employ transfer learning techniques. Transfer learning, as already explained in Chapter 1, aims at applying previously learned knowledge to solve new problems faster or with better solutions, extracting the knowledge from a source task to apply it to a target task. Specifically, it involves the concepts of a domain D and a task T. A domain $D = \{\chi, P(X)\}$ consists of a feature space χ and a marginal probability distribution $P(X)$, where $X = \{x_1, ..., x_n\} \in \chi$. Given a domain, $D = \{\chi, P(X)\}$, a task $T = \{\gamma, f(\cdot)\}$ consists of a label space γ and an objective predictive function $f(\cdot)$ learned from the training data, consisting of pairs $\{x_i, y_i\}$, where $x_i \in X$ and $y_i \in \gamma$, which is used to predict the corresponding label $f(x)$ of a new instance x . Given a source domain D_s and a corresponding source task T_s, a target domain D_t and a corresponding target task T_t, transfer learning aims to help improve the learning of the target predictive function $f_t(\cdot)$ in D_t using the knowledge in D_s and T_s, where $D_s \neq D_t$ and $T_s \neq T_t$ [24].

A distinction can be made, based on the different situations concerning source and target domains and the corresponding tasks:

- **inductive transfer learning:** where source and target domain can be either the same or different and source and target task are different but related.
- **transductive transfer learning:** where source and target domain are different but related and source and target task are the same.
- **unsupervised transfer learning:** where source and target domain are different but related but source and target task are the same and no label data are available for the target task.

According to this classification we can associate the inductive transfer learning scenario to our case since image classification has been showed to be different from weather detection [27] and we have data available to use for our target task.

There are various ways in which transfer learning can be performed [20] on CNN. Given a CNN that was previously trained on another task, such as image classification using the ImageNet dataset [26] we can distinguish two approaches:

- **Fine-tuning:** With this approach all the network parameters are retrained, backpropagating the errors into the whole network [25].
- **Freezing layers:** With this approach most of the transferred feature layers are left unchanged during training on the new task. This is motivated by the fact that the first layers contain more generic features, common to many tasks, while the others become progressively more specific to the target dataset [27].

To decide what type of transfer learning should be applied to a specific task there are several factors to take into account. The two most important ones are the size of the dataset [39] and its similarity to the dataset used to originally trained the network [40], e.g. ImageNet. In presence of a dataset that is similar and small compared to the original one, it is suggested to use the freezing layers approach since it is expected the lower-level features to be relevant for the target dataset as well. Furthermore, the small size of the dataset makes the network more prone to overfitting if a fine-tuning approach was employed, suggested when a bigger dataset is available instead. The latter approach can also be used in case we have available a different dataset from the original one. These two approaches, applied to CNN, were investigated by the authors of [20] that showed that in the case of the fine-tuning approach it tends to prevent the performance from dropping since the co-adaption of the features is re-learned, while, in the case of the freezing layer approach, depending on the number of layers that are re-trained, a drop in the performance should be observed. This is particularly the case when the re-training of the network starts from the middle layers of the architecture since a fragile co-adaptation between features is created, especially in the case the target tasks are not very similar. The authors also showed that independently from the scenario is always better to use pre-trained weights than random ones, since it improves generalization performance.

Many have been the successful architectures using transfer learning. The authors in [21] used an architecture similar to AlexNet [15], where filter size was reduced, which was initially pre-trained on the ImegeNet dataset. To verify the generalization ability of their network, they re-trained the last fully connected layer on Caltech-256 [22]. As similar approach was used in [23], which trained an architecture similar to the AlexNet one but with different filter size of which the last layers were re-trained using the VOC dataset [41]. Also, the authors in [42] used a pre-trained AlexNet on the ImageNet dataset that was fine-tuned on a dataset contained toys which differed for shape, color and texture which were captured at different position, illumination conditions and scale. Finally, as already seen in Chapter 2.1, the authors of [27] and [25] applied the discussed transfer learning approaches for weather detection.

2.6. Chapter conclusion

In this chapter, we presented background information in the field of automatic weather detection and Convolutional Neural Network (CNN).

The reviewed work was useful for a better understanding of the challenges our target task involves, i.e. detecting rain in images. Rain is indeed difficult to detect due to its properties, which makes it hardly visible, especially if light rain is intended to be recognized [29]. However, as demonstrated by the authors in [9], discriminating between light and heavy rain seems to lead to a certain grade of uncertainty although heavy rain should be more visible in images. This was one of the reason that influenced our decision on the categories to recognize. Furthermore, the work performed in [27], helped us in gaining a better understanding on the difference between the ImageNet dataset and a dataset containing weather images in relation to the Transfer Learning approaches we decided to apply.

The datasets employed in the reviewed work were taken into consideration as possible dataset to use, however they resulted unusable either because they were not publicly available such as [9], [11] and [25] or because images were captured only at one location such as [10] or because the weather conditions captured were not useful for the purpose of this thesis project such as [27] were only sunny and cloudy images were collected.

Based on the literature study on CNN, it was observed that they have been very successful in the field of object recognition, especially deeper architecture able to reduce the number of parameters avoiding using sequential architectures are gaining always more popularity such as ResNet [18], which we decided to use to perform weather recognition. From the reviewed literature, we also observed the difficulties that could be encountered during the training of CNN, such as overfitting, and the methods that could be employed for solving or mitigating it.

3

Methodology

From the literature study presented in Chapter 2, we observed that CNN based approaches are the state-of-art among other image recognition methods [43]. In particular, ResNet [18] is the most desirable architecture for our research project.

As stated in Chapter 1, our goal is to develop a system that is able to perform a classification between images showing rain and others based on the state-of-art CNN, which as seen in Section 2.3, is the ResNet architecture. In particular, we are interested in analysing images captured by existing traffic webcams infrastructure. However, this task is challenging not only because rain is hardly visible in images but also because of a lack of public datasets with the required characteristics, i.e. containing images captured from traffic webcams. For this reason, this research is articulated in two steps. First, we need to operate the data collection, gathering webcam images with the related weather information to form a large training set for the model.

Second, we need to train the network on the data collected. To do so, we will use Transfer Learning on a predefined architecture. We will start from the original ResNet architecture, trained on the ImageNet dataset, and fine-tune its weights on the newly generated dataset. We will then optimize this model so as to improve its accuracy by limiting overfitting. Finally, we will verify whether fine-tuning only a part of the weights, freezing the weights in the layers closer to the input, thereby reducing the number of parameters to train, can help to improve the model learning.

In the following sections, we provide more detail on the rationale and the methodology followed in the remainder of this thesis to implement the two above-mentioned steps.

The completion of all the phases answer our main research question (*Is it possible to recognize rain in images employing a state-of-the-art Convolutional Neural Network?*), while the completion of the second phase answer the sub-research question (*Which of the proposed Transfer Learning techniques can help in achieving better performance?*).

For each phase we applied a different methodology presented, respectively in section 3.1 and 3.2.

3.1. Dataset creation phase

This section describes the methodology adopted for designing and creating the dataset presented in Chapter 4.

Step 1 As first step, we identified important requirements for the images that would compose the dataset about the weather conditions represented, the variety of the locations where to capture the images and the time frequency at which images should be captured. We also made considerations about the size of the dataset as well as the resolution and the quality of the images.

Step 2 After this, we selected the platform that could satisfy our requirements from which we then crawled and labelled the images.

Step 3 Finally, we organize the collected dataset into the necessary categories to be classified.

3.2. Architecture optimization and training phase

In this section we present the methodology, based on the literature study presented in 2, adopted to explore the influence of transferred features on the network performance using two different approaches, which are presented in Chapter 5 .

As a first step, we had to choose an architecture to apply our Transfer Learning approach to. Since advantages and disadvantages of both methods are mainly problem-dependent we are going to apply both these techniques on a ResNet architecture, which was previously trained on the ImageNet dataset. Multiple are the reason why ResNet was chosen: with respect to the state-of-art, it is the best network for image classification, being the winner of the ILSVRC 2016.

The major advantages of ResNet over the previous winners of ILSVRC, such as [15] or [17], are [18]:

- Mitigating the effect of the Vanishing Gradient Problem, which causes the saturation and then the degradation of the accuracy when the network depth increases.
- Increasing the depth of the network without adding extra parameters, hence enabling the network to learn even more complex, non-linear functions and more abstract features without having the necessity to accordingly increase the training dataset size.
- Obtaining the highest accuracy ever achieved in ILSVRC, especially in the Image Recognition challenge.
- Accelerating training speed.

Given the complexity of our target task, we expect to need to be able to capture very abstract features to solve it, although the limited size of the available dataset prevents us from being able to train very deep architectures. Hence, a deep architecture that, depth being equal, has fewer parameters than the other reviewed architectures (Section 2.3) and has been proved to optimally perform on image recognition tasks, reducing computational times, makes ResNet the desirable architecture to use. Given the dimensionality of the parameter vector in ResNet and the difficulty of the rain detection task, the risk exists that by re-training from scratch the ResNet Architecture we would get overfitting, or no learning at all. For this reason, we decided to resort to Transfer Learning.

Transfer learning techniques have been largely addressed for their ability to improve the process of learning new tasks using the knowledge acquired solving another task especially when there is a limited supply of target training data. We decided to apply two different Transfer Learning approaches: fine-tuning, i.e. re-training the entire network, and freezing layers, i.e. re-training just a subset of network layers. Both these approaches present advantages and disadvantages, mostly related to the dataset size and the type of target task, or otherwise stated, the similarity of the source and the target dataset.

The fine-tuning approach is usually employed when a large dataset is available whether the task is similar to the original one or not. This is because, even though the task is not similar to the one the network was originally trained on, it was shown that initialize the network with weights from a pre-trained model was still beneficial, in terms of generalization performance, compared to training a new network from scratch [20]. On the other hand, the freezing layers approach is suggested to be used when a small dataset is available for the target task, due to overfitting concerns. In particular, it is possible to distinguish two scenarios: if the target task is similar to the source one, then only the higher layers, usually just the last one, of the network need to be retrained since they are the ones typically specialized on the source task; otherwise, if the target and the source tasks are very different it is better to start re-training the network from the lower layers even though the chances to experience overfitting are higher [39], [40].

We started our exploration with the fine-tuning approach because we believed it could be the most performing approach between the two, due to the difference between the source and the target task [27], which makes us evaluating the possibility that the lower layers might be different from the ones usually obtained when performing an object recognition task, and the collected dataset obtained employing the methodology in 3.1 that resulted being of medium size, hence increasing the chances of success. However, being indeed the dataset of medium size we expected to face overfitting.

Hence, for the reasons just mentioned we used the following methodology:

Step 1 As first step, we adopt the fine-tuning approach on the original version of the ResNet architecture to obtain a baseline and check the efficiency of the proposed approach.

Step 2 As second step, we tackle possible overfitting problems applying some of the regularization techniques presented in Chapter 2.

Step 3 After this, if the overfitting problem persists we act on the architecture reducing the number of parameters to train.

Step 4 Finally, we adopt the freezing layers approach on an improved original version of the ResNet architecture, re-training one after the other, the ResNet units.

4

Dataset

In section 2 we have seen various applications dedicated to the recognition of weather conditions. However, no public dataset of big dimensions that could satisfy the requirements we defined in Section 3.1 was available. This has led us to collect a proper dataset to be used for our goals.

4.1. Requirements

For the purpose of training a network that could distinguish between a rain weather and a no-rain one, it is desirable to provide training images that respect specific requirements:

- Images should be captured outdoor.
- Images should be collected in countries where rain condition is more likely to occur, so to be able to capture a bigger number of rain images.
- Webcams should point to the sky or at least they should be placed in such a way that a portion of sky is visible in the captured image.
- In order to allow a better generalization, the dataset should be composed by images where the same place is exposed to various weather conditions.
- For the same reason above, images should also be captured throughout different seasons to try to have a dataset as balanced as possible in terms of weather conditions.
- Images should be collected from a platform that can provide not only multiple pictures in different location throughout the day, but also information about the weather conditions at the moment they have been captured.

4.2. Data collection

We chose Weather Underground [44] as the most reliable source for our data. This website supplies not only webcams images that met the requirements but also real-time weather conditions to attribute to the captured images.

Weather Underground provides access to local weather data thanks to an user community that shares real-time weather data, captured through personal weather stations, on the platform. They claim to possess the largest network of weather stations whose collected data are cross-verified with the ones collected from radars, satellites and weather balloons. These stations are typically owned by government agencies and international airports, but also by private users and data are updated from 1 to 6 hours intervals.

To have more possibilities of capturing rain weather condition, we chose to collect data from webcams placed in countries where rain was a frequent condition. Hence, we selected The Netherlands, Belgium and Southern England, London specifically.

For each country a script was created so to ease the collection process as explained in the next section. Detailed information about the crawling algorithms are given in Appendix A.

Images have been retrieved every day for a period of 10 months, which ranged from April 2016 to February 2017, from 7 am to 10 pm, during summer season, and from 7am to 6pm, during spring and winter season. Every hour one of the three scripts (one for the Netherlands, one for Belgium and one for London) was executed. Their execution was organized in such a way that during a week the scripts were executed at least once in all the scheduled hours.

Weather Underground associates to each webcam a weather station that is as close as possible and that provides the current weather condition. This data was used in order to label the collected images. It worth being noticed that not all webcams are associated to the closest weather station, in which case they are associated to a default weather station of the area where they are placed. No information were available on the distance between a webcam and a default weather station.

Various were the weather conditions associated to images: for the purpose of this thesis, we sorted them in Rain and No-Rain. In Table 4.1 an overview of how labels are grouped in this two categories is shown. To group images in the rain category we made considerations on the similarity, in terms of properties, of the weather conditions that were present in the dataset. In particular, we decided to group dynamic conditions characterized by randomly distributed particles, more or less visible, that move at various velocities. In Figure 4.1 the distribution of images per each label is shown.

RAIN	**NO-RAIN**
[Light/Heavy] Rain	[Light/Heavy] Mist
[Light/Heavy] Snow	[Light/Heavy] Fog
[Light/Heavy] Ice Pellets	[Light/Heavy] Haze
[Light/Heavy] Rain Showers	Patches of Fog
[Light/Heavy] Snow Showers	Shallow Fog
[Light/Heavy] Thunderstorm	Partly Cloudy
[Light/Heavy] Thunderstorms and Rain	Mostly Cloudy
[Light/Heavy] Thunderstorms with Small Hail	Scattered Clouds
[Light/Heavy] Freezing Drizzle	Unknown
[Light/Heavy] Freezing Rain	

Table 4.1: Label subdivision

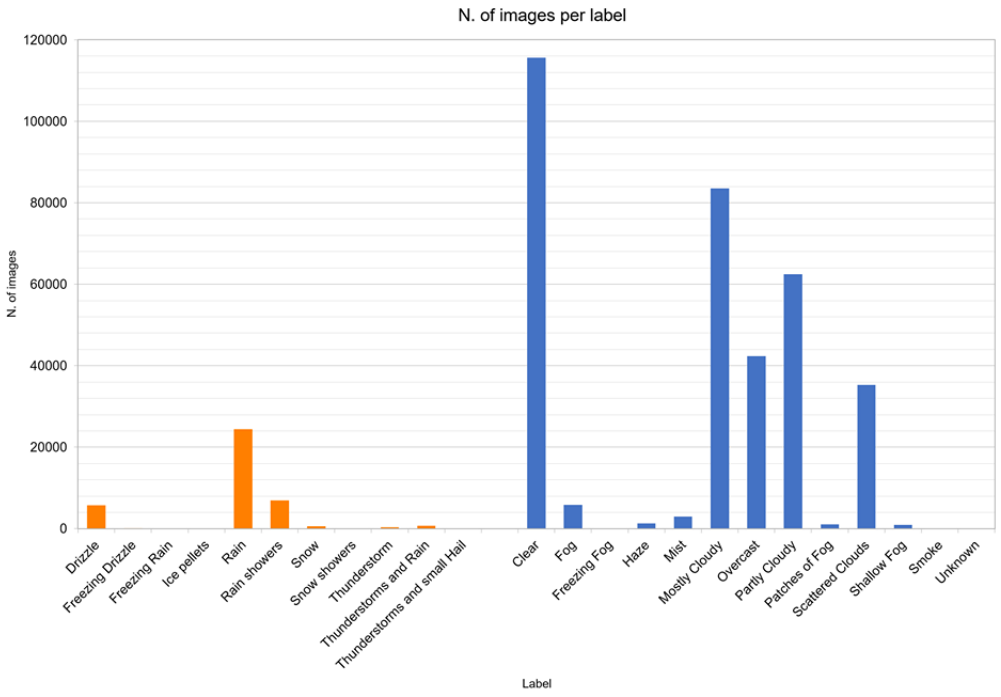

Figure 4.1: Number of images per label

4.3. Collected Data

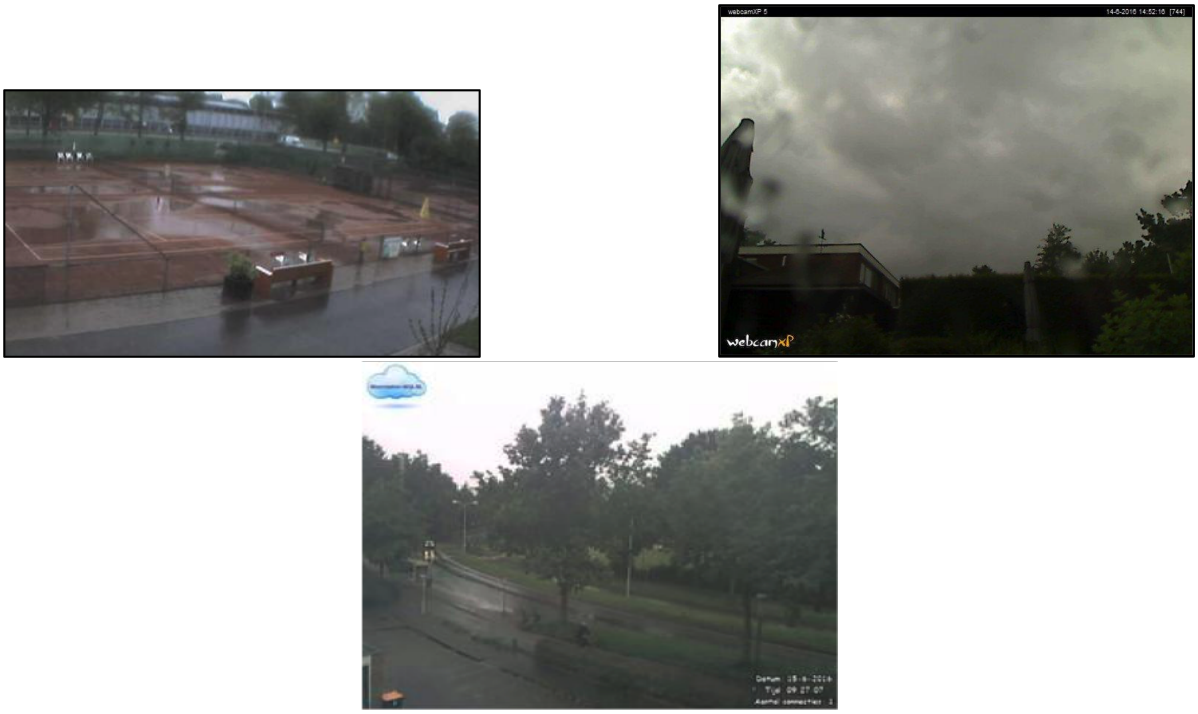

Figure 4.2: Examples of rain images

The obtained dataset consists of 397041 images coming from 774 webcams distributed in the countries mentioned previously.

The collected images were stored in the JPEG format, with an image quality factor set to 75.

Coming from so many different webcams, images have various resolutions. Specifically:400x224, 640x480, 800x500, 1024x576, 1280x1024. The number of images per each resolution is shown in Figure 4.3.

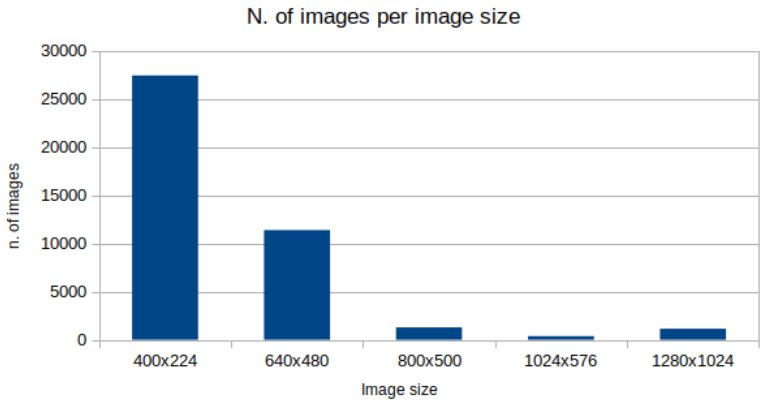

Figure 4.3: Number of images per image size

In the resulting dataset the quantity of no-rain images outnumbers the rain ones by a significant amount. Indeed, the bulk of rain images constitutes just the 10% of the entire dataset.
In Figure 4.4 some example of no-rain images is shown, whereas in Figure 4.2 examples of rain images are reported.

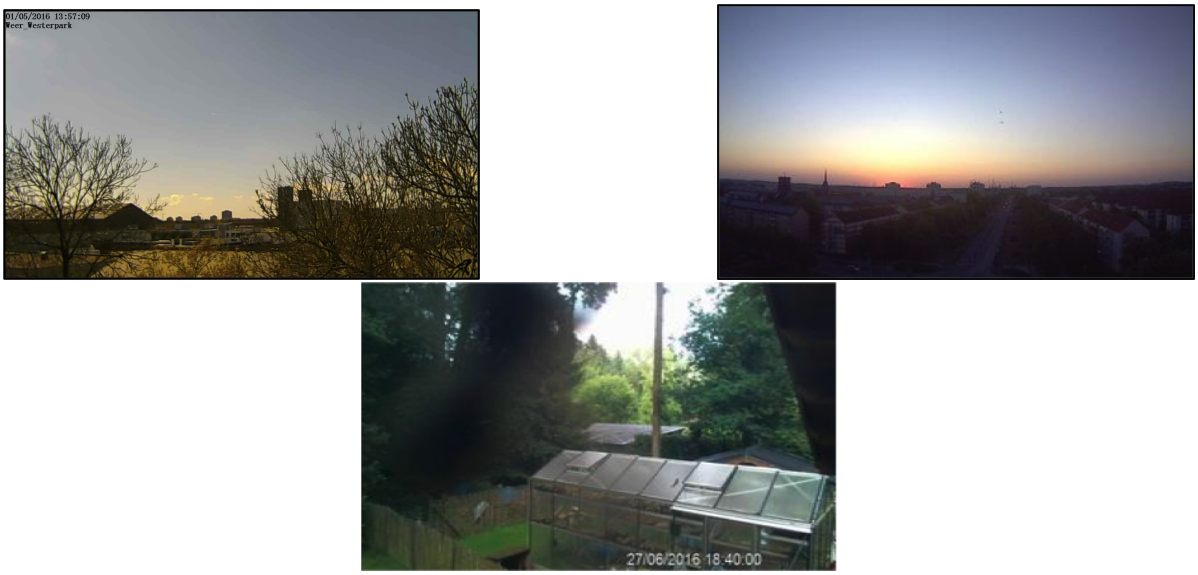

Figure 4.4: Examples of no-rain images

4.4. Dataset Peculiarities

There are some considerations that can be made about the collected images in the dataset:

1. Images are mostly low quality as it is shown in Figure 4.3.
2. There are noisy images: as shown in Figure 4.5, there can be images that are noisy, being completely dark or full of text or even showing indoor location.
3. Weather conditions may be ambiguous, meaning that sometimes is possible to have labels referring to a rain condition even though the picture seems to illustrate a cloudy, almost clear, weather. Although it might appear as a mislabelled image, it is not. This is infact a condition that can occur in countries such as The Netherlands. Examples are shown in Figure 4.6

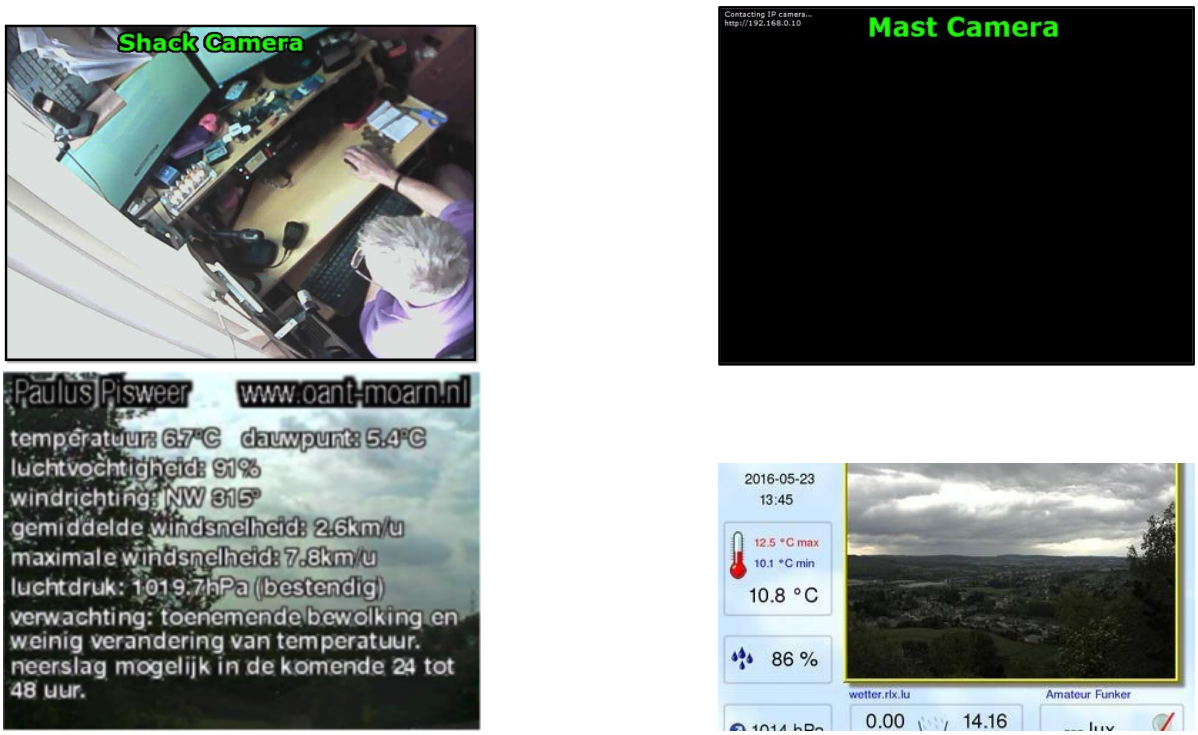

Figure 4.5: Examples of noisy images

Figure 4.6: Examples of ambiguous images

5

Transfer learning for rain detection

In this chapter the experiments made applying two different Transfer Learning approaches on a state-of-art CNN architecture, ResNet, are presented. Our goal is to find an optimized version of ResNet for detecting rain in images captured by existing webcam infrastructures collected in the dataset introduced in Chapter 4 applying Transfer Learning techniques.

As already seen in Section 2.1, the task of automatically classifying rain images is non-trivial. They are characterized by features such as illumination, reflection and shadow that cannot be associated with the ones belonging to other type of images used for object recognition tasks, as those in the ImageNet dataset [27] on which ResNet was previously trained. Transfer Learning, a technique that exploit the knowledge previously acquired in learning a new task, has been proved to improve the generalization performance of neural networks and to be beneficial for visual tasks accomplished through CNNs [20].

As seen in Section 2.5, based on the source and the target task and on the similarity between the source and the target dataset, different transfer learning approaches can be adopted. A fine-tuning approach, re-training all the network layers previously trained on a source dataset, ImageNet in our specific case, could be attempted being it suggested to use when the source and target tasks are either similar or different and the target dataset is big. In this specific case we have available a dataset of medium size and the source and the target task could be considered different since we are attempting to perform classification of images whose features to detect are different from the ones detected in the images used for the source task. Hence, re-training also the lower layers, usually known for recognizing general features such as shape or texture when object recognition or similar tasks are performed, might be useful.

A freezing layer approach,re-training just a subset of all the network layers, could be also attempted being it suggested when the dataset available is not extremely big and the target task is either similar or different. In particular, when the target task is not similar it is suggested to re-train more layers than when the target task is similar. Since, as we said, the dataset collected is of medium size and the target task could be considered different, we decided to freeze most part of the ResNet units. However, to check the effects on the performance that re-training more or fewer layers would have had, we re-trained these units one by one.

As already explained in Section 3.2, the optimization of the architecture was performed using the fine-tuning approach and, subsequently, we used one of the found optimized version of the network to apply the freezing layer approach. To optimize ResNet we acted both on the architecture, applying the improvements suggested in [38], adding dropout layers and removing some of the ResNet units, and on the hyperparameters such as the value of the regularization strength that will be identified as "weight decay" and the batch size.

In the following section, Section 5.1, more details about the experimental setup are given and in Sections 5.2 and 5.3 the results obtained applying both the transfer learning approaches and the abovementioned techniques to optimize the ResNet architecture are reported.

5.1. Experimental Setup

For this thesis project we used a ResNet architecture implemented by Sam Gross, from Facebook AI Research, and Michael Wilber, from CornellTech, on Torch and released on GitHub, along with pre-trained models [45].

The pre-trained models have been trained on the ImageNet dataset so to make easier a future fine-tuning of the chosen architecture. In particular, we use ResNet-18 architecture.

The parameters used for the architecture training are reported below:

- Loss function: Cross Entropy function
- Optimization method: Nesterov momentum based Stochastic Gradient Descendent algorithm.
- Regularization method: L2 regularization.

Learning rate is initially set to 0.001 and then updated during every epoch t in the following way $\eta = \eta * 0.1^{\lfloor \frac{t-1}{30} \rfloor}$. The weight decay is set to 0.0001 and the momentum is equal to 0.9. The mini-batch size is set to 32 and the model is trained for 90 epochs.

The network receive in input an image samples with dimensions 3x224x224. These samples have been previously shuffled and pre-processed as explained in the next subsection.

For the purpose of training the selected CNN architecture, ResNet-18, we created 5 non-overlapping folds of 10000 images each. Each fold was composed by 5000 images belonging to rain category and 5000 belonging to no-rain category so to have balanced folds in terms of number of images representing each category. Following a 5-folds cross validation approach, we used 4 folds (80%) for the training dataset and 1 fold (20%) for the validation set. Through this subdivision during the validation phase, the network will be tested on previously unseen data. During the creation of the folds we made sure that a situation in which images captured at the same place in the validation and in the training set was not possible to occur. Furthermore, we selected images belonging to rain and no-rain categories in order to avoid place matching with the aim of improving the generalization capacities of the network.

We ideally divide the set of experiments in two groups: the first group is composed by the experiments where we performed the architecture optimization employing the fine-tuning approach, i.e. re-training the entire network, and tackling possible overfitting problems through regularization techniques and through the reduction of the parameters. The second group of experiments is related to the application of the freezing layer approach, that is re-training just a subset of all the network layers, on an improved version of the original ResNet-18 architecture. Due to time constraints, we performed a full 5-folds cross validation technique only on the best found architecture of the two groups of experiments. For all the other experiments we used, as training and validation set, the first fold.

As measures to evaluate the performance of the network we used the accuracy, the number of correctly classified images over the total number of images. In particular, two kinds of accuracies were calculated: the best model accuracy, indicating which set of parameters, over all the iterations made per each experiment, allowed the network to perform the best, and the average accuracy, the accuracy calculated as the average of all the accuracies obtained by the network over all the iterations per each experiment. Furthermore, to check the occurrence of possible overfitting problems, we plotted the learning curve, composed by the network accuracies over all the iterations of the training and the validation phase, and the validation loss value over all the iterations to check its trend. Finally, to have an idea on which image category the network was performing better we calculated the confusion matrix.

5.1.1. Image preprocessing

In this subsection the preprocessing techniques the authors of [45] applied on the input images are described. Compared to the original version in [18], some of the pre-processing techniques were substituted or added.

In the images used for the training phase the authors did not follow the same steps for scaling images as in [18], but they preferred to use scale and augmentation ratio applied in [17]. Initially the aspect ratio of the image is distorted and a random crop, covering anywhere between 8%-100% of the distorted image, is taken. The smaller edge of the crop is then scaled to size 224.

Then Color jitter is introduced, randomly vary brightness, contrast and saturation of the image.

Next,a PCA-based lighting noise, originally performed on Alex-Net [15], is applied. This noise is similar to the Color jitter but it alters RGB values directly. In particular, "PCA is performed on

the set of RGB pixel values throughout the dataset. To each image are added multiples of the found principal components with magnitude proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1" [15]. The authors tuned this PCA-based noise on the ImageNet dataset and provide Eigenvalues and Eigenvectors to be used for calculating it on different datasets.

Later, Color Normaliziation is performed, subtracting from each image dimension the mean of every of them calculated across the whole dataset, and dividing each dimension by its standard deviation. Again, both the mean and the standard deviation are already provided. This technique aims at setting each dimension of the data to have zero-mean and unit-variance to remove all intensity values from the image while preserving color values.

Finally, horizontal flip is performed on 50% of the images randomly chosen. All these operations are performed one on top of the other in the order they have been described.

In Figure 5.7 an example of the pre-processing techniques applied to the training set is shown.

On the subset of images used for the validation set the authors applied a different series of preprocessing techniques. The smaller edge is scaled to size 256. Then color normalization is performed and finally either a crop technique like the one performed on the training set or the cropping of the four corners and the center of the image and of its horizontal reflection, can be done, depending on the user preference. The resulting image after the cropping operation is of size 224x224. All these operations are performed one on top of the other in the order they have been described.

In Figure 5.12 an example of the pre-processing techniques applied to the validation set is shown.

Figure 5.1: Original Input Image

Figure 5.2: Random Crop Scale

Figure 5.3: Color Jitter

Figure 5.4: PCA-based noise

Figure 5.5: Color Normalization

Figure 5.6: Horizontal Flip

Figure 5.7: Example of pre-processing in the training set

Figure 5.8: Original Input Image

Figure 5.9: Scaling

Figure 5.10: Color Normalization

Figure 5.11: Cropping

Figure 5.12: Example of pre-processing in the validation set

5.2. Fine-Tuning approach

In this section the experiments performed following the fine-tuning approach are presented.

5.2.1. ResNet-18

We trained ResNet-18 using learning rate, weight decay and momentum equal to, respectively, 0.001, 0.0001 and 0.9.

In Table 5.1 and in Table 5.2 respectively, results and confusion matrix are reported. In Figure 5.13 and Figure 5.14 the learning curve and the validation loss are shown.

Accuracy Best Model	**Average Accuracy**
64.945%	59.633%

Table 5.1: Accuracies ResNet-18

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	1095	3904	21.9%
	Rain	129	4863	97.4%
	Total	89.5%	55.5%	

Table 5.2: Confusion Matrix ResNet-18

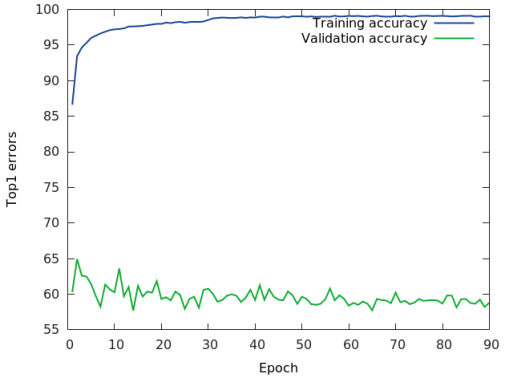

Figure 5.13: Learning Curve ResNet-18

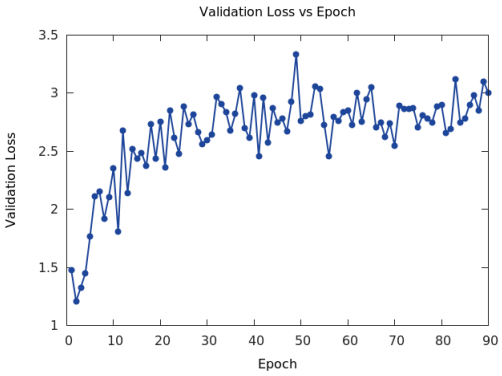

Figure 5.14: Validation Loss ResNet-18

As it can be seen from the graphs the differences between the validation and the training accuracy curves and the increasing trend of the validation loss curve are clear signs of the heavy overfitting the model is subject to. Furthermore, looking at the confusion matrix results, it can be noticed a very low percentage for the no-rain recognition images, i.e. 21.9%, indicating that the model is not able to properly recognize this category.

5.2.2. ResNet-18 with Full Pre-Activation

In order to reduce overfitting the first step was to employ the modified architecture proposed in [38] in a follow-up work on ResNet, described in Section 2.4. Specifically, we adopted the proposed full pre-activation framework consisting in placing Batch Normalization and ReLU before addition. This was proved to be beneficial because of regularization effect of the Batch Normalization.
Learning rate, weight decay and momentum were respectively equal to 0.001, 0.0001 and 0.9.
Results and confusion matrix are listed, respectively, in Table 5.3 and in Table 5.4. In Figure 5.15 and Figure 5.16 the learning curve and the validation loss are shown.

Accuracy Best Model	**Average Accuracy**
71.259%	62.032%

Table 5.3: Accuracies ResNet-18 with Full Pre-activation

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	1377	3622	27.5%
	Rain	172	4820	96.5%
	Total	88.9%	57.1%	

Table 5.4: Confusion Matrix ResNet-18 with Full Pre-activation

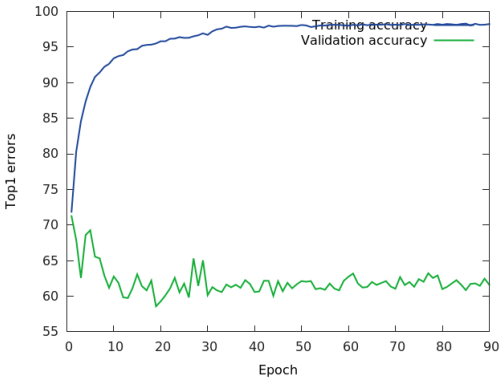

Figure 5.15: Learning Curve ResNet-18 with Full Pre-activation

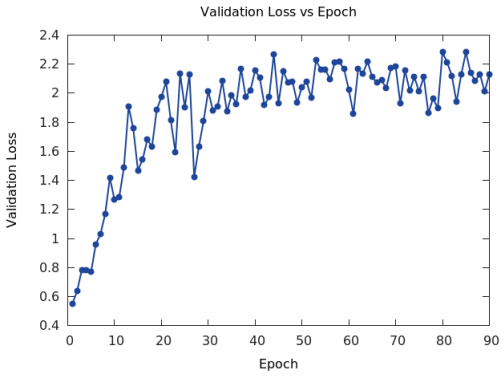

Figure 5.16: Validation Loss ResNet-18 with Full Pre-activation

Compared to the previous experiment in 5.2.1 were we trained the original version of the architecture, an healthy improvement of about 7% in the performance can be noticed when the best model accuracies are compared. However, the improvement results to be minimal, about 3%, when we consider the average accuracies. Both the accuracies increased as the percentage of no-rain images recognized. Nonetheless, the model is still heavily overfitting the training data as the graphs indicate.

5.2.3. ResNet-18 with Dropout layers and various weight decay

Having noticed that just using full pre-activation strategy was not largely helping in contrasting the overfitting phenomenon we started investigating regularization methods.

In [46] the authors faced a similar overfitting problem using ResNet-18. The strategy they adopted to overcome the problem was to employ Dropout layers. In particular, these layers were placed in between two convolutional layers in a block, as shown in Figure 5.17 and Figure 5.18

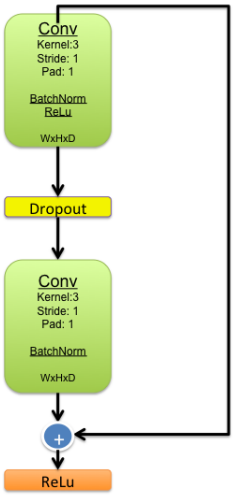

Figure 5.17: ResNet Identity Block with Dropout layer

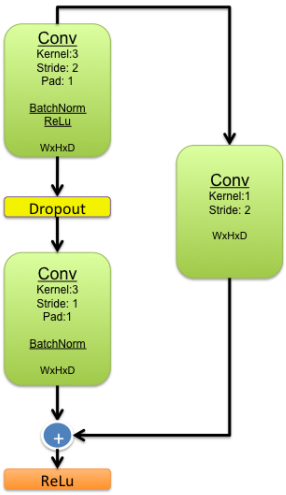

Figure 5.18: ResNet Projection Block with Dropout layer

Furthermore, we also changed the weight decay value to see if and how it affected the model's behaviour.

To verify the influence that the introduction of Dropout layers would have had on the model, we decided to firstly apply them on the original ResNet-18, 5.2.1, without employing full pre-activation used previously in 5.2.2.

Learning rate and momentum were respectively equal to 0.001 and 0.9.

Results are reported in Table 5.5. Confusion matrices, for each experiment, are listed in Tables

5.6, 5.7 and 5.8. In Figures 5.19, 5.21, 5.23 and Figures 5.20, 5.22, 5.24 the learning curves and the validation losses are shown.

Weight Decay	Accuracy Best Model	Average Accuracy
0.001	76.502	67.568%
0.0001	78.179%	67.367%
0.00001	76.037%	67.700%

Table 5.5: Accuracies ResNet-18 with Dropout Layers with different weight decay values

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	2467	2532	49.4%
	Rain	708	4284	85.8%
	Total	77.7%	63.5%	

Table 5.6: Confusion Matrix ResNet-18 with Dropout with weight decay 0.001

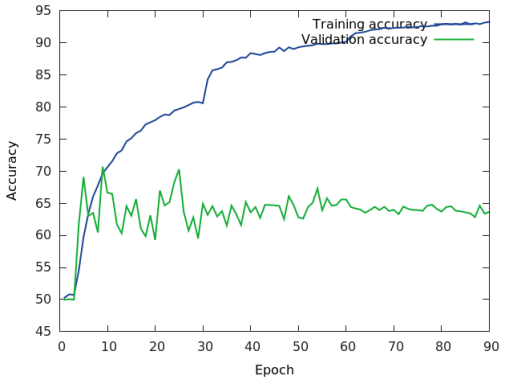

Figure 5.19: Learning Curve ResNet-18 with Dropout with weight decay 0.001

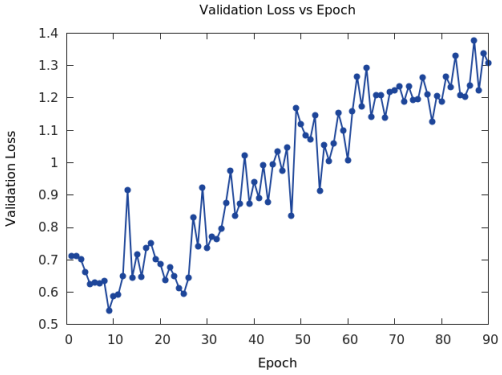

Figure 5.20: Validation Loss ResNet-18 with Dropout with weight decay 0.001

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	2417	2582	48.3%
	Rain	679	4313	86.4%
	Total	78.1%	62.6%	

Table 5.7: Confusion Matrix ResNet-18 with Dropout with weight decay 0.0001

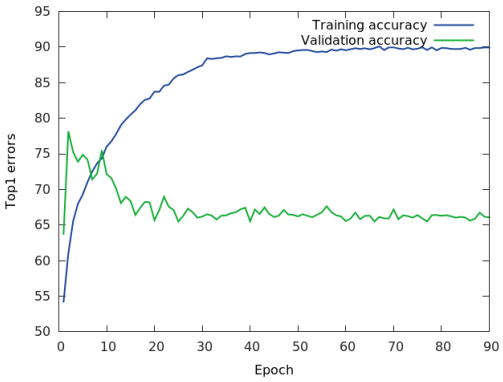

Figure 5.21: Learning Curve ResNet-18 with Dropout with weight decay 0.0001

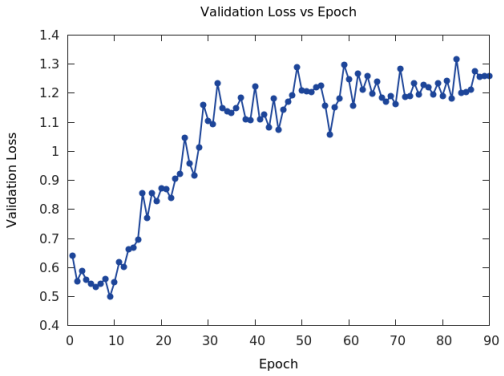

Figure 5.22: Validation Loss ResNet-18 with Dropout with weight decay 0.0001

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	2477	2522	49.5%
	Rain	705	4287	85.9%
	Total	77.8%	63%	

Table 5.8: Confusion Matrix ResNet-18 with Dropout with weight decay 0.00001

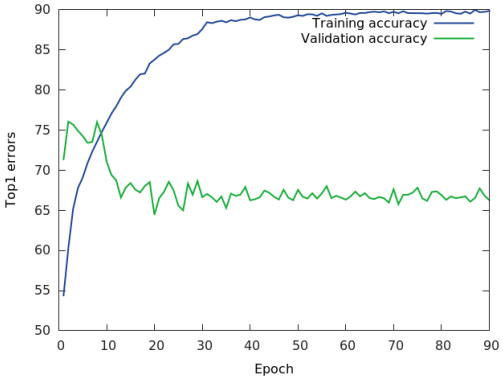

Figure 5.23: Learning Curve ResNet-18 with Dropout with weight decay 0.00001

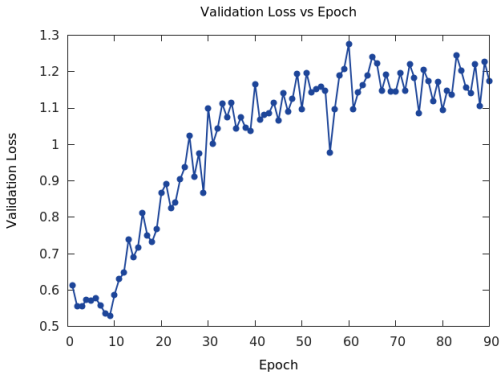

Figure 5.24: Validation Loss ResNet-18 with Dropout with weight decay 0.00001

As can be noticed from the graphs, adding Dropout layers did, indeed, mitigate the overfitting problem. We consider the best performing experiment the one where the weight decay was set to 0.0001 since, compared to the other experiment, the best accuracy resulted to be higher than the other of ~2% , while the average one were all very similar. For the best performing experiment, an increase of the best model accuracy of about 14% is noticed as well as an increase of the average accuracy of about 8%. The percentage of recognized no-rain images improved as well of 27% circa, even though a decrease of the 11% in the recognition percentage of rain images can be noticed.

Besides, from the results it can be inferred that the weight decay does not have a strong influence on the performance. Hence, in the following experiments a weight decay of 0.0001 will be used

5.2.4. ResNet-18 with Full Pre-Activation and Dropout layers

Given the benefits of adding Dropout layers, 5.2.3, in the ResNet units we combined this approach with the full pre-activation one, 5.2.2.

Learning rate, weight decay and momentum were respectively equal to 0.001, 0.0001 and 0.9. Results are reported in Table 5.9. In Table 5.10 confusion matrix is shown. In Figure 5.25 and Figure

5.26 the learning curve and the validation loss are illustrated.

Accuracy Best Model	Average Accuracy
77.455%	67.086%

Table 5.9: Accuracies ResNet-18 with Dropout and Full Pre-activation

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	2318	2681	46.4%
	Rain	607	4385	87.8%
	Total	79.2%	62.1%	

Table 5.10: Confusion Matrix ResNet-18 with Dropout and Full Pre-activation

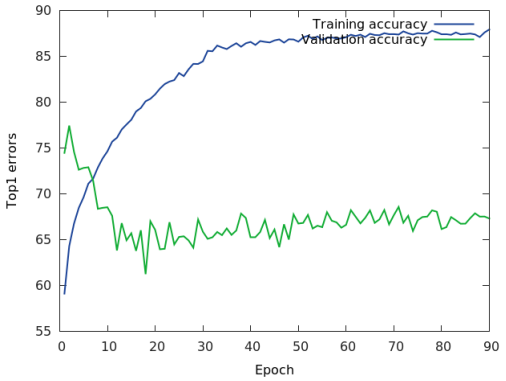

Figure 5.25: Learning Curve ResNet-18 with Dropout and Full Pre-activation

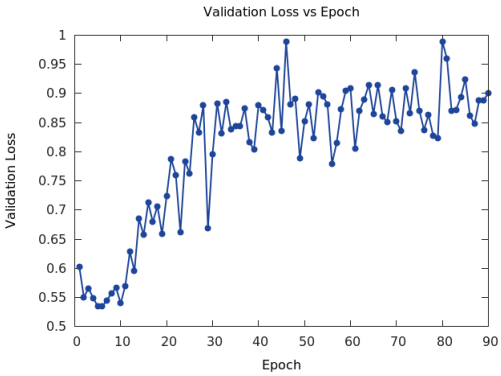

Figure 5.26: Validation Loss ResNet-18 with Dropout and Full Pre-activation

As already seen in 5.2.2, using full pre-activation does not lead to important improvements in terms of average accuracy, compared to 5.2.3. Indeed, in terms of average accuracy, using full pre-activation only we had ~62%, while using dropout only we obtained ~67%. When both approaches were combined they provided an improvement of less than 1% compared to the only addition of dropout, indicating that the addition of full pre-activation is not necessary when dropout is used for regularization. Furthermore, from the confusion matrices, It can be noticed a slight worsening in the recognition of no-rain images as well as a small improvement in the recognition of the rain ones.

5.2.5. ResNet-18 without M1 unit and with Dropout Layers

From the previous experiments we saw that dropout is effective in mitigating the overfitting problem, nonetheless overfitting is still clearly happening, as it can be inferred from the validation loss curves, which keep growing over epochs. Hence, we decide to act on the architecture depth in order to reduce the number of parameters.

To minimize the number of architecture's parameters we removed the first ResNet unit M1 as shown in Figure 5.27. The motivation for this choice is twofold: on one hand, removing the last block of the M4 unit would have led to the modification of the filter dimension of the last average pooling layer and since pooling layers perform subsampling operation, which are lossy, we thought that doubling the size would have led to an higher loss of the information contained in the signal and consequently to worse performance. On the other hand, an alternative would have been removing all the identity mapping blocks to still allow the flowing of the signal without changing any architecture parameter such as filter size of the convolutional and pooling layers, which we wanted to avoid due to the claim the authors made in [38], regarding the blocks whose shortcut perform identity mapping that achieve faster error reduction.

The obtained architecture is composed by 14 weighted layers and a total number of parameters equal to 3965442.

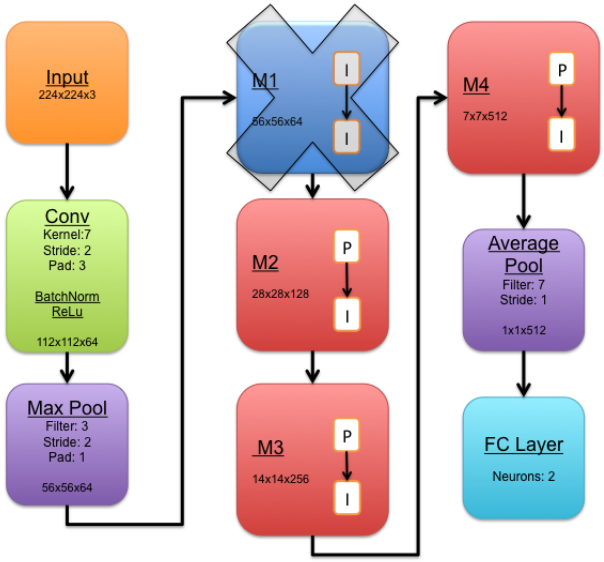

Figure 5.27: ResNet-18, minimized architecture

Learning rate, weight decay and momentum were kept respectively equal to 0.001, 0.0001 and 0.9. Results are listed in Table 5.11 and Learning curve and Validation Loss plots are shown, respectively, in Figure 5.28 and Figures 5.29. In Table 5.12 confusion matrix is listed

Accuracy Best Model	**Average Accuracy**
72.329%	67.266%

Table 5.11: Accuracies ResNet-18 with Dropout without M1 unit

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	2491	2508	49.8%
	Rain	762	4230	84.7%
	Total	76.6%	62.8%	

Table 5.12: Confusion Matrix ResNet-18 with Dropout without M1 unit

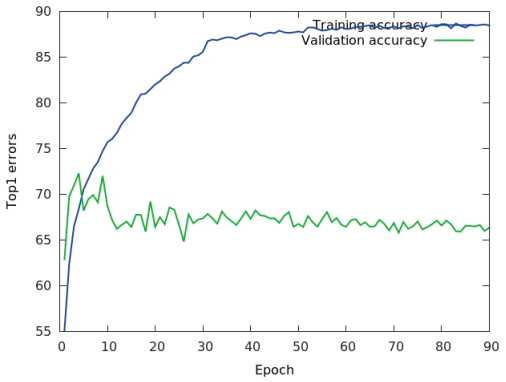

Figure 5.28: Learning Curve ResNet-18 with Dropout without M1 unit

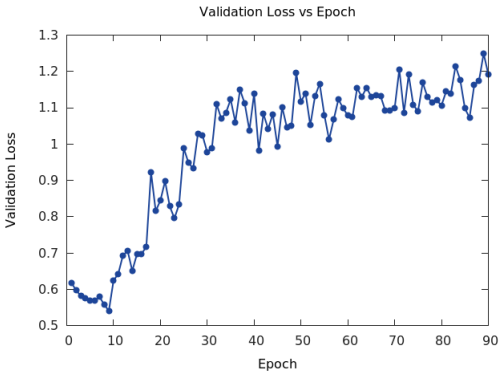

Figure 5.29: Validation Loss ResNet-18 with Dropout without M1 unit

Compared to 5.2.3, which obtained a best model accuracy of about 78% and an average accuracy of about 67%, and 5.2.4, with a best model accuracy of ~77% and an average accuracy of ~67%, a reduction of 6% circa in the best model accuracy can be noticed as well as a very small reduction in terms of average accuracy of less than 0.5%. Furthermore, also the recognition percentage of rain images marginally worsened of ~3% compared to the recognition percentage obtained in 5.2.4, the highest one between the mentioned experiments. However, the recognition percentage of no-rain images is slightly improved of ~1% compared to the percentage obtained in 5.2.3, the highest one between the mentioned experiments.

5.2.6. Chopped ResNet-18

As next step we tried to reduce the depth of the architecture as much as possible so to reduce the total number of parameters at their minimum. The only possible way to do it without changing any hyperparameters, specifically the depth size of the filters in the convolutional layers within the blocks, in the architecture was to remove all the blocks performing Identity Mapping within the ResNet units. Doing so, no changes on the Projection blocks, the blocks responsible for the downsampling of the corresponding input to match the input dimensions of the next unit, had to be made, allowing us to still be able to use the weights of the pre-trained architecture.
The resulting architecture, shown in Figure 5.30, is composed by 8 weighted layers, instead of 18, and a number of parameters of 2414466.

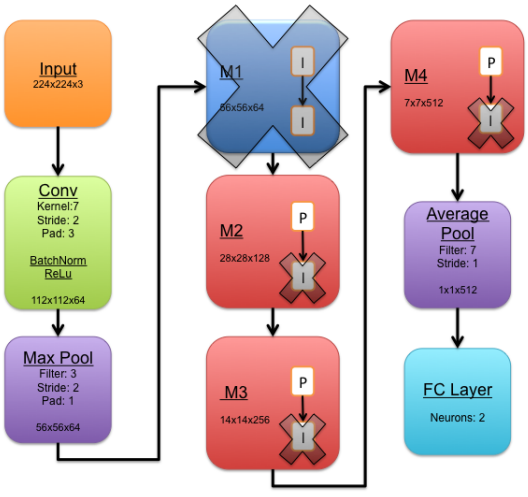

Figure 5.30: ResNet-18, chopped architecture

Learning rate, weight decay and momentum were respectively equal to 0.001, 0.0001 and 0.9. Results are reported in Table 5.13 and confusion matrix in Table 5.14. In Figure 5.31 and Figure 5.32

the learning curve and the validation loss are shown.

Accuracy Best Model	Average Accuracy
73.785%	63.298%

Table 5.13: Accuracies Chopped ResNet-18

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	1524	3475	30.5%
	Rain	192	4800	96.2%
	Total	88.8%	58%	

Table 5.14: Confusion Matrix Chopped ResNet-18

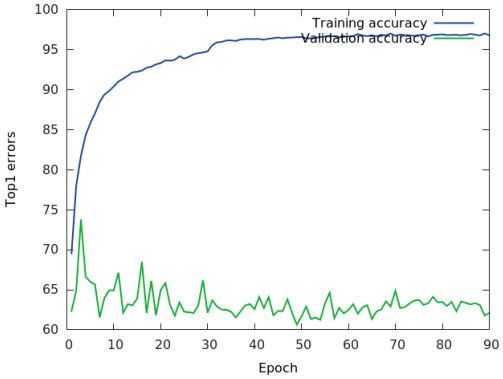

Figure 5.31: Learning Curve Chopped ResNet-18

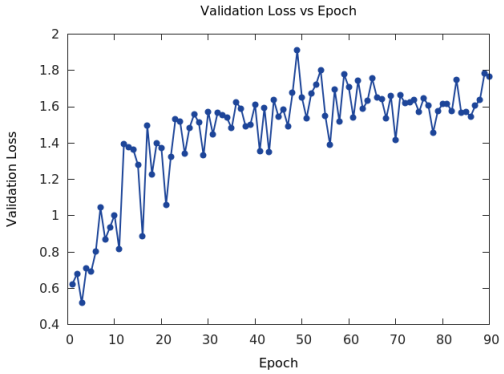

Figure 5.32: Validation Loss Chopped ResNet-18

From the results we can notice that performance are improved compared to 5.2.1. An improvement of about 10% is obtained in terms of best model accuracy, while the improvement in terms of average accuracy is of about 4%. Furthermore, also the recognition rate of the no-rain images increased of ~8%.

5.2.7. Chopped ResNet-18 with Dropout layers

The next step is to add to the previous architecture in 5.2.6 Dropout layers.

Learning rate, weight decay and momentum were respectively equal to 0.001, 0.0001 and 0.9. Results are shown in Table 5.15 and confusion matrix in Table 5.16. In Figure 5.33 and in Figure 5.34 learning curve and validation loss are illustrated.

Accuracy Best Model	Average Accuracy
75.013%	69.005%

Table 5.15: Accuracies Chopped ResNet-18 with Dropout

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	2861	2188	57.2%
	Rain	959	4033	74.9%
	Total	80.8%	65.4%	

Table 5.16: Confusion Matrix Chopped ResNet-18 with Dropout

Figure 5.33: Learning Curve Chopped ResNet-18 with Dropout

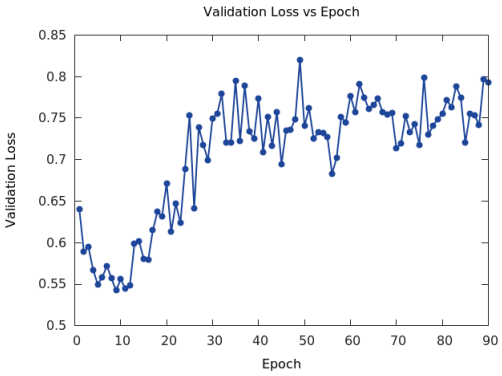

Figure 5.34: Validation Loss Chopped ResNet-18 with Dropout

From the obtained results it can be noticed that, even though the best model performance decreased of about 3% compared to 5.2.3, the average one increased of ~2%. However, the rain recognition rate decreased more than 11%, but on the contrary the no-rain recognition rate increased of ~9%.

5.2.8. Chopped ResNet-18 with Dropout layers and various Batch Sizes

We decided to explore the network's behaviour when the batch size was varied. Specifically, we trained the network using two batch sizes: 64 and 128

Learning rate, weight decay and momentum were respectively equal to 0.001, 0.0001 and 0.9. Results are shown in Table 5.17 and confusion matrices in Table 5.18 and 5.19.
In Figure 5.35, 5.37 and in Figure 5.36, 5.38 learning curves and validation losses are illustrated.

Batch Size	**Accuracy Best Model**	**Average Accuracy**
64	76.213%	70.575%
128	78.026%	73.062%

Table 5.17: Accuracies chopped ResNet-18 with Dropout using various Batch sizes

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	3291	1708	65.8%
	Rain	1232	3760	75.3%
	Total	72.8%	68.8%	

Table 5.18: Confusion Matrix Chopped ResNet-18 with Dropout using Batch size 64

Figure 5.35: Learning Curve Chopped ResNet-18 with Dropout using Batch size 64

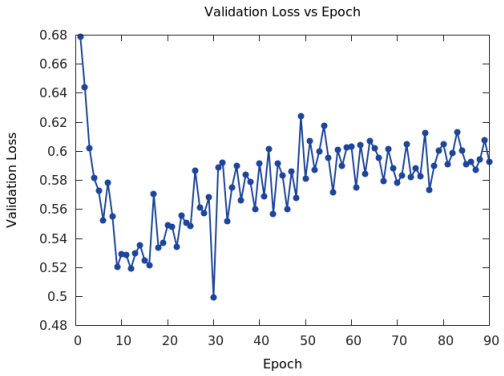

Figure 5.36: Validation Loss Chopped ResNet-18 with Dropout using Batch size 64

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	4007	992	80.1%
	Rain	1699	3293	66%
	Total	70%	76.8%	

Table 5.19: Confusion Matrix Chopped ResNet-18 with Dropout using Batch size 128

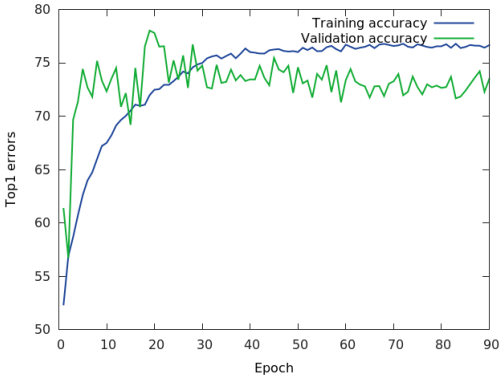

Figure 5.37: Learning Curve Chopped ResNet-18 with Dropout using Batch size 128

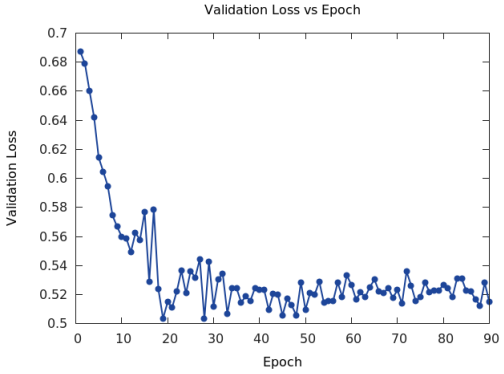

Figure 5.38: Validation Loss Chopped ResNet-18 with Dropout using Batch size 128

Varying the batch size clearly influence the performance of the model. The bigger the size of the batch size the more the overfitting seems to be reduced along with the validation loss that, especially with size 128, is correctly minimized over epochs.
Comparing the performance obtained when the batch size was set to 128 with the one obtained in **??** we can see that in terms of the best model accuracy they are almost the same. However, they improve of about 6% when the average accuracy is compared.

Moreover, looking at the confusion matrices it can be observed how the recognition percentage of both categories differ. Indeed, the recognition percentage of the no-rain category increase with the increase of the batch size, achieving an improvement of more than 30% when the batch size is 128; on the contrary, the recognition percentage of the rain category decreases along with the batch size, up to 20% with a batch size of 128.

5.2.9. ResNet-18 with Dropout layers and Batch size 128

Given the benefit brought by the increment of the batch size, we decided to train the whole ResNet-18 architecture using a batch size of 128, the one with which we obtained the best accuracy using a chopped architecture in 5.2.8.

Results are listed in 5.20 and Confusion matrix in 5.21. In Figure 5.39 and Figure 5.40 learning curve and validation loss are shown.

Batch Size	Accuracy Best Model	Average Accuracy
128	78.966%	73.488%

Table 5.20: Accuracies ResNet-18 with Dropout using Batch size 128

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	3545	1456	70.9%
	Rain	1193	3799	76.1%
	Total	74.8%	72.3%	

Table 5.21: Confusion Matrix ResNet-18 with Dropout using Batch size 128

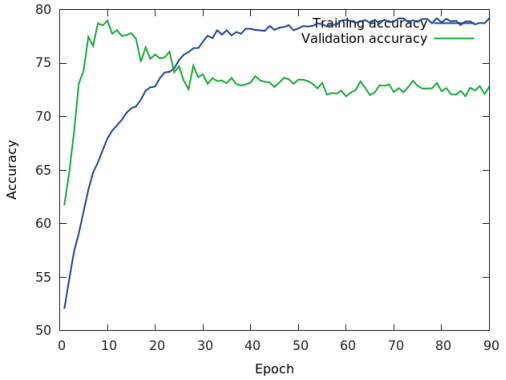

Figure 5.39: Learning Curve ResNet-18 with Dropout using Batch size 128

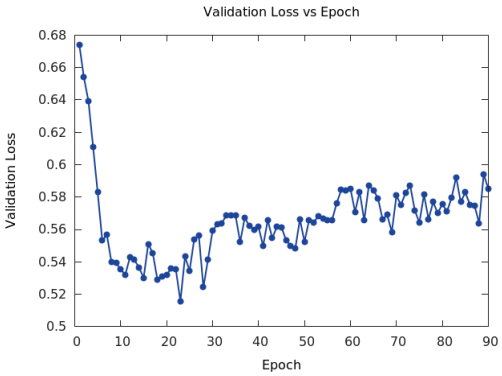

Figure 5.40: Validation Loss ResNet-18 with Dropout using Batch size 128

Comparing the results with the equivalent experiment in 5.2.8, we can notice a very small improvement of less than 1% in terms of both accuracies and an increasing of ~10% in the recognition percentage of the rain category. However, the recognition percentage of the no-rain category decreased of about 10% and the validation loss curve assumes a slight increasing trend after the first 20 epochs.

5.3. Freezing layer approach

In this section the experiments performed following the freezing layer approach are presented. From the previous section we have learnt that to contrast overfitting the best approach consist of applying dropout as a regularization technique and using a bigger batch size than the default one. Furthermore, also reducing parameters through the reduction of the number of blocks within the ResNet units helps in the reduction of the overfitting problem. However, for the purpose of this approach we decided to use the whole architecture of ResNet-18, instead of the chopped version, on which dropout was applied and which was trained with a batch size of 128. This decision was driven by the fact that we re-train ResNet units one by one, keeping the others frozen, hence the number of parameters to re-train will be less than the one in the whole architecture.
For all the following experiments learning rate, weight decay and momentum were respectively equal to 0.001, 0.0001 and 0.9. Furthermore, having noticed that in the previous group of experiments both the learning curve and the validation loss curve were not affected by substantial changes after the first 30 epochs we decided to train the network just for 50 epochs instead of 90. Results are listed in Table 5.22 and Confusion matrices in Table 5.23,5.24, 5.25 and 5.26. In Figure 5.41, 5.43, 5.45, 5.47 and Figure 5.42, 5.44, 5.46, 5.48 learning curve and validation loss are shown.

Frozen layers	Accuracy Best Model	Average Accuracy
FC	73.652%	69.912%
FC, M4	75.017%	71.174%
FC, M4, M3	74.453%	71.466%
FC, M4, M3, M2	74.428%	72.058%

Table 5.22: Accuracies ResNet-18 using the Freezing layer approach

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	3592	1407	71.9%
	Rain	1599	3393	68%
	Total	69.2%	70.7%	

Table 5.23: Confusion Matrix ResNet-18 re-training FC Layer

Figure 5.41: Learning Curve ResNet-18 re-training FC Layer

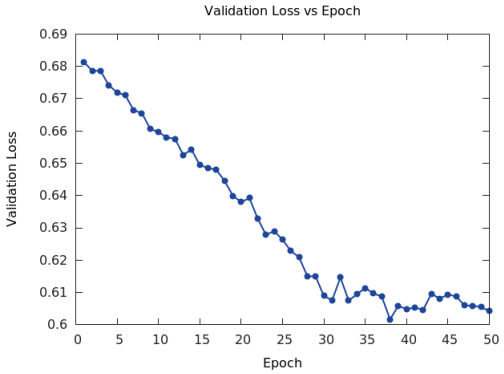

Figure 5.42: Validation Loss ResNet-18 re-training FC Layer

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	3761	1237	75.3%
	Rain	1643	3350	67.1%
	Total	69.6%	73%	

Table 5.24: Confusion Matrix ResNet-18 re-training 512 Unit

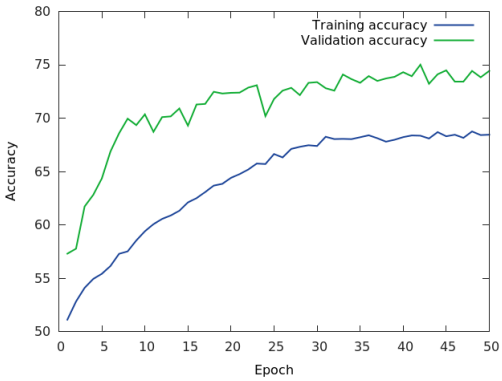

Figure 5.43: Learning Curve ResNet-18 re-training 512 Unit

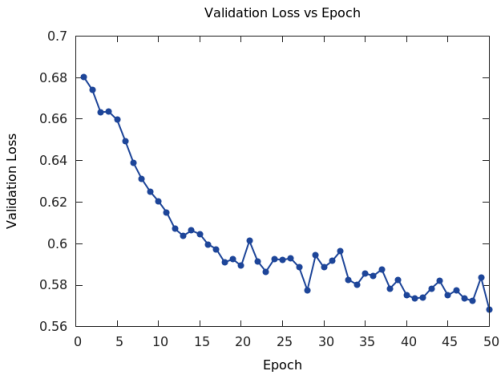

Figure 5.44: Validation Loss ResNet-18 re-training 512 Unit

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	3665	1334	73.3%
	Rain	1516	3476	69.6%
	Total	70.7%	72.3%	

Table 5.25: Confusion Matrix ResNet-18 re-training 256 Unit

Figure 5.45: Learning Curve ResNet-18 re-training 256 Unit

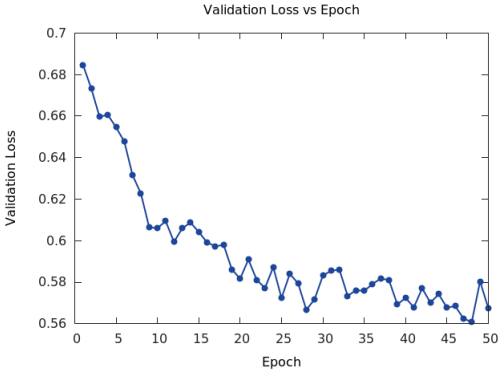

Figure 5.46: Validation Loss ResNet-18 re-training 256 Unit

		Prediction outcome		
		No-Rain	**Rain**	*Total*
Actual value	**No-Rain**	3548	1451	71%
	Rain	1341	3651	73.1%
	Total	72.6%	71.6%	

Table 5.26: Confusion Matrix ResNet-18 re-training 128 Unit

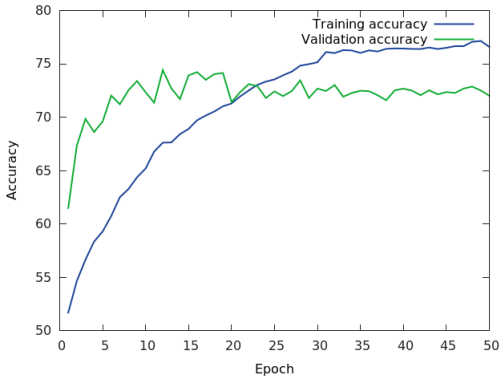

Figure 5.47: Learning Curve ResNet-18 re-training 128 Unit

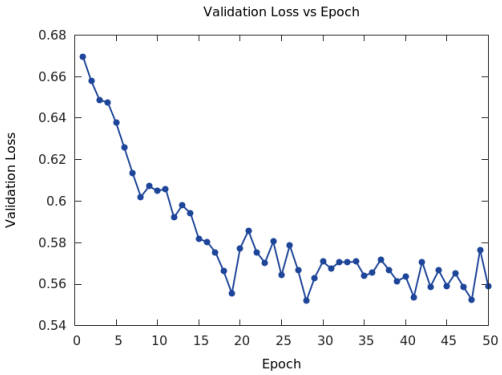

Figure 5.48: Validation Loss ResNet-18 re-training 128 Unit

As it can be seen from the obtained results, as we gradually increase the number of layers to re-train, the average accuracy slightly increase. The same trend is not followed by the best model accuracy, which reaches its highest value when only FC layer and M4 are re-trained. However, being the difference between all the best model accuracy very small, less than 2% between the lowest and the highest one, it can be considered irrelevant. When comparing the best obtained result in term of average accuracy, i.e. when the FC layer and the M4, M3, M2 are re-trained, with the one in 5.2.9, we can notice a diminishment of ~1%. Furthermore, looking at the learning curves, we notice that the gap between

the training and the validation accuracy curves decrease as we gradually increase the number of units to re-train, passing from a situation of underfitting, when the validation accuracy is higher than the training one, to a situation of light overfitting when the validation accuracy is lower than the training one.

5.4. Cross-validation on the best performing architecture

In this section we present the results obtained applying the 5-fold cross-validation approach on the best performing architectures obtained in Section 5.2 and 5.3.Specifically, they are, respectively, the Chopped ResNet-18 with Dropout architecture trained with a batch size of 128 5.2.8 and the ResNet-18 with Dropout architecture trained with batch size 128 and freezing unit M1. Results are shown in Table 5.27 and 5.28

Approach	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Average**
Fine-tuning	78.026%	77.902%	65.277%	60.295%	60.451%	68.390%
Freezing layer	74.428%	79.110%	64.197%	56.566%	60.683%	66.996%

Table 5.27: 5-fold cross validation Best Accuracies ResNet-18 using Fine-tuning and Freezing layer approaches

Approach	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Average**
Fine-tuning	73.062%	74.054%	58.979%	57.251%	52.795%	63.228%
Freezing layer	72.058%	72.897%	61.492%	55.536%	57.559%	63.908%

Table 5.28: 5-fold cross validation Average Accuracies ResNet-18 using Fine-tuning and Freezing layer approaches

From the results we can notice that after the first two folders there is a drop in both the best and the average accuracies of both models. Looking at the corresponding training and validation set we indeed notice that it could be attributed to the data fed to the network. Indeed, in the last three validation sets we notice the presence of a high number of images showing highways which were completely absent in the first two validation sets. Unfortunately, a quantification of these kind of images was not possible to the difficulties in automatically recognizing them.

Comparing the average accuracies obtained by the two approaches both for the best model accuracy and the average accuracy we can see that the difference is very small, almost insignificant in the latter case with ~0.6% of difference, and with less than 2% difference in the former one.

5.5. Discussion

Various approaches and methods to optimize the chosen architecture, ResNet-18, were compared in Section 5.2 and Section 5.3. We started by evaluating the performance on ResNet-18 applying the fine-tuning approach, using as initial weights the one of the network trained on the ImageNet dataset, and we backpropagated the calculated error through the entire network. Due to the strong overfitting, which can be inferred both from the distance between the training and the validation accuracy curves and the increasing trend of the validation loss curve, we decided to act on the regularization, studying the effects that the addition of the full pre-activation framework and dropout would have had on the network.

The addition of full pre-activation did slightly improve the performance. As claimed in [38], this can be presumably attributed to the Batch Normalization's regularization effect. Contrary to the original version of ResNet [18], all the inputs to weight layers have been normalized. Indeed, in the original version, the signal was normalized by the Batch Normalization but next it was added to the shortcut signal, hence the resulting merged signal was not normalized.

On the other hand, the addition of Dropout contributed more in contrasting overfitting with an improvement of ~14% in terms of best model accuracy and of ~8% in terms of average accuracy. This form of regularization, which forces the network to learn several independent representation of the training data, improves generalization by preventing too complex co-adaption on the data [36]. If the relationship between the input and the correct output is complex and the network has enough hidden units to model it accurately, there will typically be many different settings of the weights that can model

the training set almost perfectly, especially if there is only a limited amount of labelled training data [47]; averaging between all the different setting of weights makes the network more robust to unseen data during the test phase.

As other form of regularization, we also varied the weight decay values. This did not produce almost any change in the performance, which could be explained by the fact that we are performing a classification task and not a regression one, meaning that predictions for the observations are changing but not the relative ranking of the scores.

A further step to contrast overfitting was to remove selected ResNet units from the architecture, consequently reducing the number of parameters. This resulted in an improvement in terms of performance, which can be justified by the relationship that connects the number of network parameters to the number of data point in the training set [48].

What mostly helped in contrasting overfitting was varying the batch size. Indeed, using a larger value allowed reducing the variance of the stochastic gradient update by taking the average of the gradients in the mini-batch. This seems to be advantageous since the gradient of a single data point could be noisier than the average one calculated over the mini-batch, making sure that the network will eventually converge to some minimum instead of get stuck in some local minima [49].

Once we found an optimal combination of parameters that minimized the overfitting problem we apply them to the whole ResNet-18. Due to the increased number of parameters, even though in terms of performance it was very similar to its chopped version, in the validation loss curve it can be noticed a slightly increasing trend.

In Table 5.29 an overview of the results obtained from the first group of experiments adopting the fine-tuning approach is shown.

Architecture	**Accuracy Best Model**	**Average Accuracy**
ResNet-18	64.945%	59.633%
ResNet-18 with full pre-activation	71.259%	62.032%
ResNet-18 with dropout	78.179%	67.367%
ResNet-18 with full pre-activation and dropout	77.455%	67.086%
ResNet-18 without M1 unit and with dropout	72.329%	67.266%
Chopped ResNet-18	73.785%	63.298%
Chopped ResNet-18 with dropout	75.013%	69.005%
Chopped ResNet-18 with dropout and batch size 128	**78.026%**	**73.062%**
ResNet-18 with dropout and batch size 128	**78.966%**	**73.488%**

Table 5.29: Accuracies Fine-tuning approach

In Section 5.3, we wanted to study the effect of a different transfer learning approach, gradually freezing ResNet units in the network. We started by freezing all the units and training just the fully connected layers to conclude with freezing just the first unit M1. As we kept decreasing the number of frozen units we notice an increase in the performance. This is explained by the kind of features each layer is specialized to. Indeed, as proved in [20], higher layers are specialized in the features related to the task at hand, while the lower level features are more general features that are similar for different kind of tasks. Furthermore, we saw a similar behaviour of the network with the one presented in [27] when the freezing layer approach was applied. Indeed training a high number of units led to better performance, due to the differences between the weather-related and the object-related features.

Architecture	**Accuracy Best Model**	**Average Accuracy**
ResNet-18 with dropout and batch size 128	78.966%	73.488%
ResNet-18 with frozen M1 unit	74.428%	72.058%

Table 5.30: Accuracies best found architectures

Comparing the results obtained in Section 5.2 for the architecture ResNet-18 with Dropout layers and trained with a batch size of 128, 5.2.9 and the results obtained in Section 5.3 for the same architecture where only the unit M1 was frozen, as shown in Table 5.30, we see a small difference in the performance with ~4% more in the best accuracy and ~1% more in the average accuracy for the former architecture.

Contrary to what is stated in [20], according to which a freezing layer approach should be more beneficial than the fine-tuning one presence of small dataset, fine-tuning seems to perform better when just trained on the first fold. This could be explained by the fact that we froze five weighted layers instead of the three the authors suggested, which are considered generic and reusable.

However, after performing cross-validation on the best architectures obtained in the two groups of experiments, we noticed that performance worsened in the last three folds. We attribute this drop to the images composing the corresponding training and validation sets. Indeed, it was noticed that in the last three validation sets a big amount of pictures captured in highways were present, which were completely absent in the first two validation folds. It could be hypothesized then that using these kind of images in the training set helps to improve the generalization performance, however if a good amount of them is not present in the training set but it is in the validation set, the network find challenging recognizing them leading to worse performance.

In terms of average accuracies they performed very similar obtaining ~63%, with only ~0.6% of difference between the accuracies obtained by the two approaches, showing the effectiveness of both of them.

An overview of the results obtained applying the cross validation approach is shown in Table 5.31.

Architecture	**Accuracy Best Model**	**Average Accuracy**
Chopped ResNet-18 with dropout and batch size 128	68.390%	63.228%
ResNet-18 with frozen M1 unit	66.996%	63.908%

Table 5.31: 5-fold cross validation average accuracies best found architectures

6

Conclusion and Future Work

In this chapter we will summarize the work done, present some of its limitations and provide suggestions for future work.

6.1. Contributions

In this thesis project our main goal was to explore the space design of CNN, ResNet specifically, employing two different transfer learning techniques in order to be able to recognize the presence of rain in images. In order to accomplish our goal we created a dataset suitable for the task at hand. Thus, this work provide the following contributions.

1. **Dataset:** Given the lack of dataset that could meet our necessities, i.e. being formed by images captured from existing camera infrastructures, we created a dataset which could overcome our needs. The dataset consist of 397041 labelled images which were captured from existing webcam infrastructures in The Netherlands, Belgium and South England. 10% of the whole dataset is labelled as rain.

2. **ResNet architecture optimization and comparison of Transfer Learning techniques:** After performing an intensive architecture optimization, we proposed a modified architecture of ResNet characterized by the reduction of the number of weighted layers and the addition of dropout layers. Specifically, this architecture is composed by eight weighted layers and 2414466 learnable parameters and achieve an average accuracy of 63.228% obtained employing a 5-fold cross validation technique. To the best of our knowledge this is one of the first attempt in classifying rain and no-rain images using CNN.

   Furthermore, we compared two different methods for performing transfer learning: the fine-tuning approach, consisting in re-training the whole network and the freezing layer approach, consisting in re-training just a subset of all the network layers. Both of them obtained similar performance. As expected, given the reviewed literature [20], [39], and the size of the dataset, the approach obtaining slightly better results in terms of average accuracy was the freezing layer one with 63.908% accuracy.

6.2. Limitations

There are still many limitations to this work. Indeed, some techniques that could have helped in obtaining better performance have not been tried.

For example, we did not apply any data augmentation techniques to our input. At the beginning of our exploration phase, we tried to apply data augmentation extracting crops from the images; in particular we extracted 5 crops: the central part of the image and the 4 corners. However, during the training we realized it was not useful and, furthermore, it was even worsening the overfitting problem, so we stopped the training due to the large computation time it would have required. We justified the worse performance with the presence of too many crops showing sky pieces whose features could be

hardly useful when entire images had to be recognized in the validation phase. However, we did not make any more try once we find a working architecture.

Another limitation of this work can be identified in the dataset. Indeed, the dataset presents numerous misleading images that contributes in lowering the performances such as ambiguous labelled images, images containing too much text, dark images or indoor ones.

Finally, we focused just on ResNet as architecture to use for our task. However, it could be possible that most recent architecture could be able to better recognize and generalize key features such as [50], [51] and [52].

6.3. Future Work

Based on the outcome of this project, we provide recommendations for future work.

We suggest improving the content of the dataset, discarding those images not bringing any useful information to the task but just representing a form of noise. As noticed during the cross-validation process they can have a strong influence on the final performance, especially when training and validation set are not well balanced in terms of scenarios distribution. In order to do this either manual or automatic detection of these images can be implemented. For instance, regarding the manual detection a crowdsourcing approach could be employ: users across the world could be asked to identify weather conditions among a group of choices or, given a weather condition and a group of image, they could be asked to identify from the most representative to less representative the one that, in their opinion, better represent the given weather condition. Regarding automatic detection approaches instead, they could be used to remove dark and indoor images or, through the employment of OCR algorithms, to remove images with text.

It is also worthwhile investigating the effects on the performance if the number of classes to classify were extended. For instance, based on the number of images per classes in the database, the most represented ones could be detected. We believe this could improve performances since in this way the class extension, in terms of weather conditions represented by each, could be narrowed, making easier the recognition of such classes and avoid the presence of too different characteristic representing a particular weather condition in a single class, as it happens with the no-rain class.

It would be also interesting applying a different kind of input such as video. In this way the dynamicity of rain could be better captured and probably more easily recognizable.

Finally, it would be useful, for gaining an additional insight, exploring feature visualization to see if it is possible identifying which are the feature that allows discriminating between the different classes.

Appendices

A

Data collection algorithm

In this appendix, we describe in details the algorithms used for collecting the data.

We created scripts to download automatically images from the Weather Underground website [44]. An API is provided to easily obtain all the information needed. The API supply various JSON files in which all the necessary data are stored. In particular, the desired informations were the webcam images URLs and the weather conditions measured by the stations associated to each webcam.

Three algorithms were designed, one for each considered area (The Netherlands, Belgium and London). Algorithms were divided by country in order to make easier the next step, i.e. scheduling them to find a trade-off between the cons-trains previously mentioned and the frequency with which the images were updated. The script algorithms were structured in the following way. Initially, the JSON file containing the list of webcams in current area was open and read, after which the JSON elements were converted in Python lists. These lists were scanned employing an iteration method to find the image URLs and the corresponding associated weather stations from which the current weather condition was obtained. In case of missing associated weather station, the API provided a default one for the city taken into consideration, which was used to label the image.

In order to add the label with the relative weather condition EXIF tags were added to the images. To work with EXIF data, the Piexif library, a Python library that supports the manipulation of these data, was employed. For the whole library documentation, please, refer to the following website: "http://piexif.readthedocs.org/en/latest/". Piexif provides various tags, from which some of them were chosen casually. In particular, the weather condition information was stored in the tag called "Make" and an information regarding the last time the image was updated in the website was stored under the tag "Software". The latter data was used to avoid multiple savings of the same image, since it univocally identified each of them. Finally, images were stored in the corresponding folder. Specifically, the folders were named after the webcam ID, whereas the images were named using the name of the city in which they have been captured and data and time at which they have been captured.

A.0.1. Piexif Library

The Piexif library is a Python library developed by the MIT Institute for manipulating EXIF tags and provides just few functions to work with. It is possible to write, read and delete tags, which are several and are represented by different type of variables: int, string and byte. To modify tags in the image it is necessary to make use of the function load, which take as input the name of the file and returns the exif data stored along with it as dictionary. In particular, keys composing the dictionary are: "0th", "Exif", "GPS", "Interop", "1st" and "thumbnail", which are, in turn, dictionaries themselves. We stored the information needed to be annotated, the weather condition and the update time, in the dictionary "0th", respectively, in the keys "Make" and "Software". For the whole library documentation, please, refer to the following website: "http://piexif.readthedocs.org/en/latest/"-.

Bibliography

[1] E. M. Fischer and R. Knutti, *Anthropogenic contribution to global occurrence of heavy-precipitation and high-temperature extremes,* Nature Climate Change **5**, 560 (2015).

[2] C. Lu, D. Lin, J. Jia, and C.-K. Tang, *Two-class weather classification,* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014) pp. 3718–3725.

[3] J. Bossu, N. Hautière, and J.-P. Tarel, *Rain or snow detection in image sequences through use of a histogram of orientation of streaks,* International journal of computer vision **93**, 348 (2011).

[4] I. C. BNL, *RainSense project,* https://www.research.ibm.com/university/cas/benelux/research.html.

[5] R. Hut, *This umbrella 'listens' to rain—for science a new sensor that counts raindrops may someday help scientists study weather and water,* .

[6] IBM, *The Weather Company,* http://www.theweathercompany.com/.

[7] S. Sawant and P. Ghonge, *Estimation of rain drop analysis using image processing,* International Journal of Science and Research (IJSR) (2013).

[8] K. Sudheer and R. Panda, *Digital image processing for determining drop sizes from irrigation spray nozzles,* Agricultural Water Management **45**, 159 (2000).

[9] M. Roser and F. Moosmann, *Classification of weather situations on single color images,* in *Intelligent Vehicles Symposium, 2008 IEEE* (IEEE, 2008) pp. 798–803.

[10] Z. Chen, F. Yang, A. Lindner, G. Barrenetxea, and M. Vetterli, *Howis the weather: Automatic inference from images,* in *Image Processing (ICIP), 2012 19th IEEE International Conference on* (IEEE, 2012) pp. 1853–1856.

[11] X. Yan, Y. Luo, and X. Zheng, *Weather recognition based on images captured by vision system in vehicle,* in *International Symposium on Neural Networks* (Springer, 2009) pp. 390–398.

[12] C. Bishop, *Pattern recognition and machine learning (information science and statistics), 1st edn. 2006. corr. 2nd printing edn,* Springer, New York (2007).

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning* (MIT Press, 2016).

[14] Y. LeCun, K. Kavukcuoglu, and C. Farabet, *Convolutional networks and applications in vision,* in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on* (IEEE, 2010) pp. 253–256.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks,* in *Advances in neural information processing systems* (2012) pp. 1097–1105.

[16] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition,* arXiv preprint arXiv:1409.1556 (2014).

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions,* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015) pp. 1–9.

[18] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition,* arXiv preprint arXiv:1512.03385 (2015).

[19] D. M. Hawkins, *The problem of overfitting,* Journal of chemical information and computer sciences **44**, 1 (2004).

[20] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, *How transferable are features in deep neural networks?* in *Advances in neural information processing systems* (2014) pp. 3320–3328.

[21] M. D. Zeiler and R. Fergus, *Visualizing and understanding convolutional networks,* in *European conference on computer vision* (Springer, 2014) pp. 818–833.

[22] G. Griffin, A. Holub, and P. Perona, *Caltech-256 object category dataset,* (2007).

[23] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, *Return of the devil in the details: Delving deep into convolutional nets,* arXiv preprint arXiv:1405.3531 (2014).

[24] S. J. Pan and Q. Yang, *A survey on transfer learning,* IEEE Transactions on knowledge and data engineering **22**, 1345 (2010).

[25] Z. Zhu, L. Zhuo, P. Qu, K. Zhou, and J. Zhang, *Extreme weather recognition using convolutional neural networks,* in *Multimedia (ISM), 2016 IEEE International Symposium on* (IEEE, 2016) pp. 621–625.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *Imagenet: A large-scale hierarchical image database,* in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (IEEE, 2009) pp. 248–255.

[27] M. Elhoseiny, S. Huang, and A. Elgammal, *Weather classification with deep convolutional neural networks,* in *Image Processing (ICIP), 2015 IEEE International Conference on* (IEEE, 2015) pp. 3349–3353.

[28] K. Garg and S. K. Nayar, *Detection and removal of rain from videos,* in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, Vol. 1 (IEEE, 2004) pp. I–528.

[29] K. Garg and S. K. Nayar, *When does a camera see rain?* in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, Vol. 2 (IEEE, 2005) pp. 1067–1074.

[30] P. Kamavisdar, S. Saluja, and S. Agrawal, *A survey on image classification approaches and techniques,* International Journal of Advanced Research in Computer and Communication Engineering **2**, 1005 (2013).

[31] N. Michael, *Artificial intelligence a guide to intelligent systems,* ISBN **321204662**, 1 (2005).

[32] N. Karayiannis and A. N. Venetsanopoulos, *Artificial neural networks: learning algorithms, performance evaluation, and applications*, Vol. 209 (Springer Science & Business Media, 2013).

[33] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, and G. Wang, *Recent advances in convolutional neural networks,* arXiv preprint arXiv:1512.07108 (2015).

[34] Y. Bengio *et al.*, *Learning deep architectures for ai,* Foundations and trends® in Machine Learning **2**, 1 (2009).

[35] S. Ruder, *An overview of gradient descent optimization algorithms,* arXiv preprint arXiv:1609.04747 (2016).

[36] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting.* Journal of Machine Learning Research **15**, 1929 (2014).

[37] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift,* arXiv preprint arXiv:1502.03167 (2015).

[38] K. He, X. Zhang, S. Ren, and J. Sun, *Identity mappings in deep residual networks,* in *European Conference on Computer Vision* (Springer, 2016) pp. 630–645.

[39] D. Soekhoe, P. van der Putten, and A. Plaat, *On the impact of data set size in transfer learning using deep neural networks,* in *International Symposium on Intelligent Data Analysis* (Springer, 2016) pp. 50–60.

[40] B. Chu, V. Madhavan, O. Beijbom, J. Hoffman, and T. Darrell, *Best practices for fine-tuning visual classifiers to new domains,* in *Computer Vision–ECCV 2016 Workshops* (Springer, 2016) pp. 435–442.

[41] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, *The pascal visual object classes (voc) challenge,* International journal of computer vision **88**, 303 (2010).

[42] A. Borji, S. Izadi, and L. Itti, *What can we learn about cnns from a large scale controlled object dataset?* arXiv preprint arXiv:1512.01320 (2015).

[43] S. Srinivas, R. K. Sarvadevabhatla, K. R. Mopuri, N. Prabhu, S. S. Kruthiventi, and R. V. Babu, *A taxonomy of deep convolutional neural nets for computer vision,* arXiv preprint arXiv:1601.06615 (2016).

[44] *Weather Underground,* https://www.wunderground.com.

[45] S. Gross and M. Wilber, *ResNet training in Torch,* https://github.com/facebook/fb.resnet.torch (2016).

[46] M. S. Ebrahimi and H. K. Abadi, *Study of residual networks for image recognition,* .

[47] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors,* arXiv preprint arXiv:1207.0580 (2012).

[48] L. G. Valiant, *A theory of the learnable,* Communications of the ACM **27**, 1134 (1984).

[49] L. Bottou, F. E. Curtis, and J. Nocedal, *Optimization methods for large-scale machine learning,* arXiv preprint arXiv:1606.04838 (2016).

[50] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, *Inception-v4, inception-resnet and the impact of residual connections on learning,* arXiv preprint arXiv:1602.07261 (2016).

[51] S. Targ, D. Almeida, and K. Lyman, *Resnet in resnet: generalizing residual architectures,* arXiv preprint arXiv:1603.08029 (2016).

[52] S. Zagoruyko and N. Komodakis, *Wide residual networks,* arXiv preprint arXiv:1605.07146 (2016).