Bachelor's thesis

Bachelor's degree in Mathematics

Bachelor's degree in Informatics Engineering

Extracting weather features from outdoor scene images using Convolutional Neural Networks

---

May 2022

Author: Francesc Martí Escofet

Advisor: Shuran Song

Tutor: Jordi Torres Vinyals

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de Formació Interdisciplinària Superior

COLUMBIA UNIVERSITY

Acknowledgments

First of all, I would like to thank Prof. Shuran Song for giving me the opportunity to do my bachelor's thesis under her supervision and for guiding me in my research. I would also like to thank Columbia University for making this stay possible using their resources and being able to be part of their Columbia Artificial Intelligence and Robotics Lab (CAIR). I would also like to acknowledge Professor Jordi Torres for his supervision on my project from Barcelona.

In addition, I would like to thank Centre de Formació Interdisciplinària Superior (CFIS), Prof. Miguel Ángel Barja, Prof. Toni Pascual and Prof. Eduardo Alarcón for giving me the opportunity to do my final thesis project in one of the top universities in the world. Also, I would like to thank fundació CELLEX and CFIS' private sponsors for their economical support.

Finally, I would like to thank my parents Eduard and Dolors, and my family M.Àngels E., Romà, Jenny, M.Àngels M., Cata, Júlia, Mihir and Laura for encouraging me to take this opportunity and their support throughout the whole experience. Moltes gràcies a tots.

Abstract

In this thesis we construct a new dataset with images extracted from webcams around the world using EarthCam, a public website where they have multiple live cameras showing outdoor scenes and any user can take a screenshot of the camera at any time and it becomes publicly available, they call this the *Hall of Fame*. We associate the images with their weather conditions, obtained via the OpenWeather API, such as the main weather description and the temperature.

We also develop convolutional neural networks to predict a weather condition for an image and we propose some new model architectures for being able to predict the temperature from images from places the model has not been trained on.

Keywords: Artificial intelligence, computer vision, deep learning, convolutional neural networks

MSC2020: 68T07

Resum

En aquesta tesi construïm un nou conjunt de dades amb imatges extretes de webcams situades arreu del món utilitzant EarthCam, una web pública on tenen diverses càmeres en directe que mostren escenes exteriors i on qualsevol usuari pot realitzar una captura de pantalla en qualsevol moment i es publica a la web, ho anomenen *Saló de la Fama*. Aleshores associem les imatges amb les seves condicions meteorològiques, obtingudes a través de l'API d'OpenWeather, com la descripció de la condició meteorològica principal i la temperatura.

També desenvolupem vàries xarxes neuronals convolucionals per predir la condició meteorològica per una imatge i proposem noves arquitectures per predir la temperatura d'imatges d'escenes en les quals el model no ha estat entrenat.

Paraules clau: Intel·ligència artificial, visió artificial, aprenentatge profund, xarxes neuronals convolucionals

MSC2020: 68T07

Resumen

En esta tesis construimos un nuevo conjunto de datos con imágenes extraídas de webcams situadas alrededor del mundo usando EarthCam, una web pública donde tienen múltiples cámaras en directo que muestran escenas exteriores y donde cualquier usuario puede realizar una captura de pantalla en cualquier momento y se publica en la web, lo llaman *Salón de la Fama*. Entonces asociamos las imágenes con sus condiciones meteorológicas, obtenidas mediante la API de OpenWeather, como la descripción de la condición meteorológica principal y su temperatura.

También desarrollamos diferentes redes neuronales convolucionales para predecir la condición meteorológica principal de una imagen y proponemos nuevas arquitecturas para predecir la temperatura de imágenes de escenas donde el modelo no ha sido entrenado.

Palabras clave: Inteligencia artificial, visión artificial, aprendizaje profundo, redes neuronales convolucionales

MSC2020: 68T07

Contents

List of Figures

1 Introduction

For many years the task of identifying visual attributes from images has been heavily studied, mainly, previous works have been focused on recognizing 'explicit visual attributes' such as textures, color distribution [7] or semantics categories [16]. During the years, as the computer vision techniques improved, many works have been studying what is called 'subtle attributes.' These subtle attributes are usually not clearly defined in an explicit form but the human mind can usually recognize them. One example of this is a method proposed by Hays and Efros where they estimated geographic information from images [10].

As human beings, we usually associate colors with some adjectives, for example, red is usually associated with 'warm' or 'hot' and blue with 'cold'. In the case of temperate climate, white is associated with 'cold' and 'winter' due to winter snow, green with 'fresh' and 'summer' as the flora becomes green in 'spring' and 'summer' and yellow light is associated to 'warm' due to the color of sunlight. Given an outdoor image, a person can make an approximate guess on the ambient temperature by looking at the amount of light, the colors on the image and the present man-made objects, and this is one of the tasks we will be focusing on in this thesis.

Another problem in the field of data science is the creation of datasets, currently there exists a lot of datasets but most of them have been labeled by humans, and that increases a lot the cost of creating a new dataset or expanding an existing one to have more images, as, with bigger models, bigger datasets are needed. For that reason, in this thesis we first create a new dataset and label it automatically using data from weather stations around the world so that people are not needed to label each image.

After that, we explore different methods to extract weather features from images such as the main weather condition and the temperature. In the case of predicting the temperature we propose different convolutional network architectures for the model being able to predict temperature from cameras it has not been trained on.

2 Related work

In this section of the thesis, we will show some related work in the field of extracting weather data from images.

One of the largest public database with outdoor webcams images is the Archive of Many Outdoor Scenes (AMOS) [14], a project started in 2007 by Jacobs et al. It consists of a dataset of hundred of millions of images gathered from publicly accessible outdoor webcams from around the world. It has been used to predict wind speed and vapor pressure [15] and other conditions from the images.

2.1 Label prediction

In the case of labeling an image with a weather condition, one of the first works was done by Lu et al. [23] in 2014, they introduced a new dataset consisting of 10k images labeled as either *Cloudy* or *Sunny*. They compute for each image what they call *weather features*, a multi-dimensional feature vector formed by concatenating five hand-crafted feature components: Sky, Shadow, Reflection, Contrast, and Haze.

A few years later, the same authors of the previous paper combined hand-crafted features and features extracted by a CNN architecture to increase the performance of their method [24].

In 2015, Elhoseiny et al. [6] used the dataset introduced in [23] and used the AlexNet architecture [19] to predict the labels and achieved a 54.8% relative improvement in comparison to the previous work using hand-crafted features.

The problem with many of these architectures were that they considered that each image could only have one unique label, but in real-world conditions weather can have multiple labels, i.g. in one image it can be rainy and sunny at the same time.

In 2017, Li et al. [22] proposed using auxiliary semantic segmentation of weather cues to comprehensively describe the weather conditions.

In 2019, Ibrahim et al. [13] used the ResNet architecture [11] to extract various weather and visual conditions such as Dawn/dusk, day and night for time detection, and glare for lighting conditions, and clear, rainy, snowy, and foggy for weather conditions. In Figure 2.1 is shown the architecture used for them to predict multiple labels.

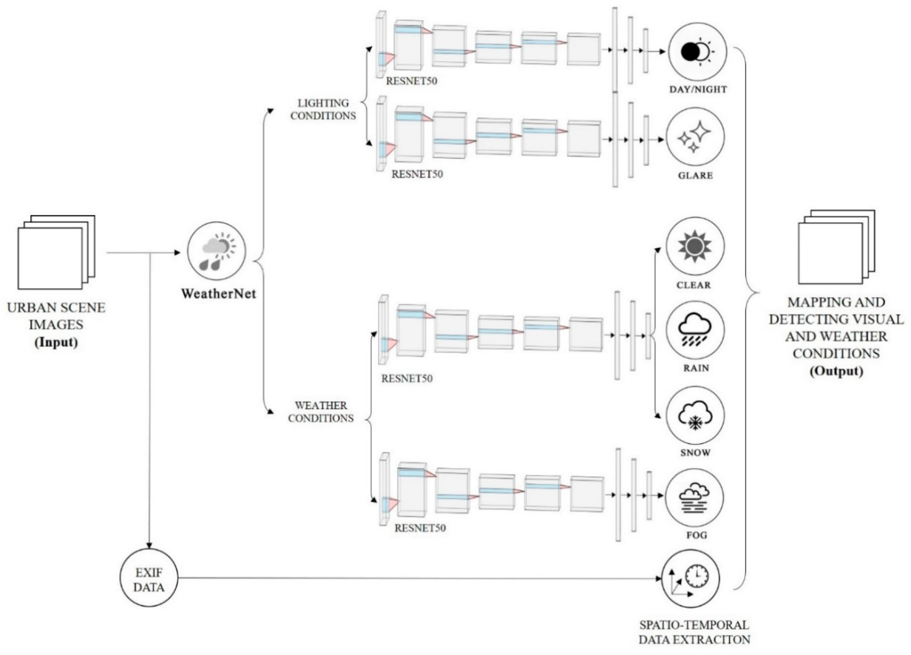


Figure 2.1: WeatherNet archicture [13]

2.2 Temperature prediction

In the more concrete case of temperature prediction using images, one of the first works on it is a work by Glasnet et al. in 2015 [9]. They study interactions between the appearance of an outdoor scene and its temperature from images from a specific camera with past recordings. They managed to achieve impressive results using handcrafted features and simple regression models without using deep convolutional networks.

After this work, Volotkin et al. [34] used deep neural networks to predict temperature as well as the time of the year of an image achieving some better results than the works using hand-crafted features. As in the work of Glasner et al. [9], they use the VGG-16 [29] architecture for their CNNs pretrained on ImageNet. Some of their more important findings are the following: i) The pooling layers provide better features than the fully connected layers. ii) The quality of the features improves a little with fine-tuning of the CNN on training data.

In 2018, Chu et al. [2] studied ambient temperature prediction based on deep neural networks in two different scenarios: predicting the ambient temperature of a single outdoor image and predicting the temperature of the last image in a sequence of images. In the first scenario, they used visual features extracted by a CNN. In the second scenario they take into account the temporal evolution of visual appearance and they construct a recurrent neural network to predict the temperature of a given image sequence.

3 Artificial Intelligence, Deep Learning and Computer Vision

This chapter contains an introduction to Artificial Intelligence, Deep Learning and Computer Vision field. More specifically, it contains an introduction to Artificial Neural Networks and how they work.

3.1 Artificial Intelligence

Artificial Intelligence (AI) is an immense research field for trying to simulate human intelligence processes by machines. The most popular textbooks define the field as the study of 'intelligent agents' that receive percepts from the environment and take actions that affect the environment to achieve its goals.

AI includes many different subfields such as speech processing, computer vision, natural language processing, knowledge representation...

3.2 Deep Learning

In the recent years, Deep Learning (DL) has become one of the fastest growing subfields of AI. DL is part of a broader family of machine learning methods based on Artificial Neural Networks (ANN). The advantage of deep learning over some classical machine learning methods like linear regression or a support vector machine is that an ANN can process unstructured data like images or text and can process them without the need for human experts.

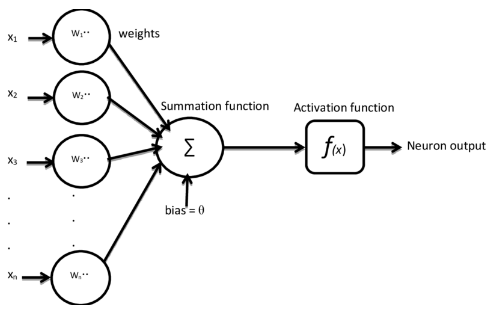
3.2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computing systems inspired by the biological neural networks that make up the brains of animals. Biologically, a brain neuron receive a series of stimuli (also named inputs) and produces a specific output signal depending on them. In the AI world, a neuron is a node that performs a mathematical function depending on its inputs $\boldsymbol{x}$ and produces an output y:

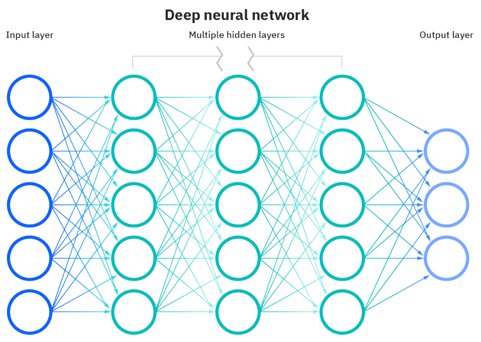
$$y = f(\boldsymbol{W}^T \boldsymbol{x} + \theta) \tag{1}$$

Where $\boldsymbol{W}$, θ and f are parameters of the neuron called weights, bias, and activation function, respectively. In 3.1a we can see a diagram of how an artificial neuron works.

When we start grouping neurons and using their outputs as the inputs of other neurons we call it a Neural Network (NN). A NN has at least one input layer and one output layer, but more layers can be added between them, these are called hidden layers. A NN with one or more hidden layers is called a Deep Neural Network (DNN).



(a) An artificial neuron [35]



(b) A neural network with 1 input layer, 1 output layer and 3 hidden layers [5]

Figure 3.1: Structure of simple neural networks

3.2.1.1 Activation functions

To introduce nonlinearity in neural networks, we need to use activation functions, these are nonlinear mathematical functions that are applied to the output of a neuron before passing the value to the next layer. The most used are the following ones:

sigmoid

The *sigmoid* takes any real value as input and outputs a value in the range $[0, 1]$, the larger the input, the closer the output will be to 1, whereas the smaller the input, the output will be closer to 0.

tanh

The $tanh$ takes any real value as input and outputs a value in the range $[-1, 1]$. It has a similar shape as the $sigmoid$ function. The larger the input, the closer the output will be to 1, whereas the smaller the input, the ouput will be closer to -1.

ReLU

The *rectified linear unit* $(ReLU)$ is the most widely used actually as the hidden layers activation function. It is common because it is simple and effective in solving some of the problems of the activation functions previously used, such as $sigmoid$ and $tanh$. In particular, it is less sensitive to the vanishing gradient problem, although it can suffer from other problems, such as saturated units.

The plots and formulas of these activation functions are shown in Figure 3.2.

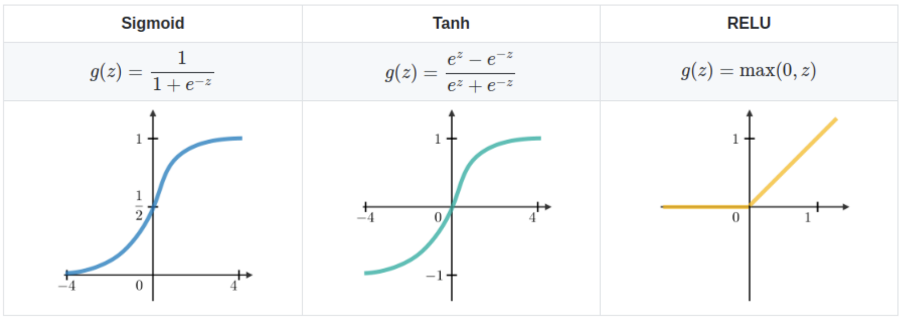


Figure 3.2: Graphs of the most common activation functions: $sigmoid$, $tanh$ and $ReLU$ [17]

3.2.2 Convolutional Neural Networks (CNNs)

One of the main architectures of neural networks for working with image data are Convolutional Neural Networks (CNNs). They were introduced by Fukushima in 1980 [8], the Neocognitron introduced the two basic types of layers in CNNs: convolutional layers and downsampling layers. In 1989 LeCun et al.[20] used the backpropagation algorithm (a gradient based method) to learn the convolution kernel coefficients directly from images of handwritten numbers. This learning was fully automatic and performed better than manual coefficient design. This approach became one of the foundations of modern computer vision.

CNNs are designed to process images because they use their spatial structure to extract additional information, treating differently pixels which are close together from pixels which are far apart in the image. CNNs architectures are mainly composed of three types of layers:

- **Convolutional layers:** They are the most important layers in the architecture and the ones after which the architecture is named, they perform an operation called *convolution*. In the context of neural networks, a convolution is a linear operation that involves the multiplication of a set of weights with the input, similarly to a traditional neural network. As the input to the network is two-dimensional, these multiplication is performed between an input data and a two-dimensional array of weights, which is called a filter or a kernel. These layers consists of a number of predefined filters, each of them smaller than the input as it allows the same filter to be multiplied by the input multiple times at different points on the input, usually these filters are applied systematically to each filter-sized patch of the input data. In the case of images with more than one channel (e.g. 3 channels for RGB images), the filter has a third dimension which extends to all the channels.

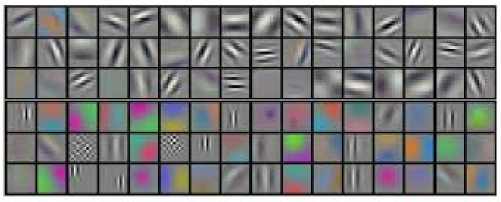
  Each of these filters produces a two-dimensional result which is called *feature map*. After a feature map is created, each value on it is passed through a non-linear function, usually a $ReLU$.

  Figure 3.3a shows an example of the feature maps extracted by the VGG-16 architecture [29] and Figure 3.3b shows an example of the filters learned by AlexNet [19].

- **Pooling layers:** These layers are responsible for reducing the dimensionality of the feature maps by combining the outputs of neuron clusters in one layer into a single output. These are layers without learnable parameters. The most common pooling layers are *Max Pooling* and *Average Pooling*. Their operation is really instinctive, first they divide the feature map in regions of the size of the kernel of the layer, usually $2x2$, and later they compute their function on each of these patches, taking the maximum of their values or averaging them respectively.

  Figure 3.4 shows an example of a feature map that goes through these layers.

(a) Visualization of the feature maps extracted from the first Convolutional Layer in the VGG-16 Model [1]



(b) Visualization of 96 convolutional filters of size $11x11x3$ learned by the first convolutional layer of AlexNet on $224x224x3$ input images [19]

Figure 3.3: Visualization of the feature maps and the filters on some CNNs

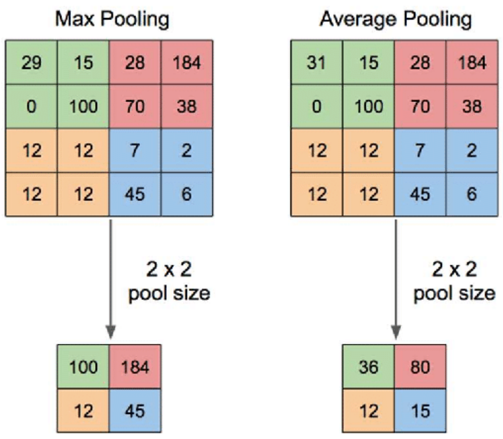


Figure 3.4: Example of feature map of size $4x4$ going through a *Max Pooling* and a *Average Pooling* layers with a kernel size of $2x2$ [36]

- **Fully connected layers:** Finally, the last layers are *fully connected layers*. These layers connect every neuron in one layer to each neuron in the following. Before the feature maps are passed through these layers, they are flattened, so the spatial dimension is lost before going through them. Each of these layers has a lot of trainable parameters and they tend to overfit the data, so, usually, CNNs use a small number of fully connected layers. The number of outputs depends on the task is the architecture designed to solve, for example, if we are trying to classify images in 100 classes the output will be of size 100, each one containing the probability of the image belonging to that class.

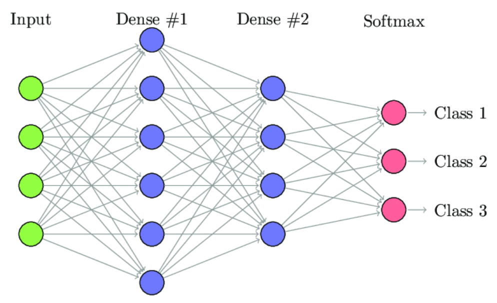


Figure 3.5: An example of fully connected layers with three output cells: each one containing the probability of the image being of a specific class respectively [25]

3.2.2.1 Most popular CNNs architectures

Although the convolutional and pooling layers can be combined in very different ways, there are some architectures that work better than others due to their structure.

The first famous CNN that inspired most of the following works in computer vision is *LeNet-5* developed by LeCun et al. in 1998 [21]. They worked on their architecture for a decade to achieve an efficient architecture. In Figure 3.6 we can see the architecture of the network. The architecture of *LeNet-5* was quite simplistic if we compare it with the actual CNNs architecture, but it provided a basis from which to develop new architectures. Their simple architecture was the following:

$$Input \Rightarrow Conv \Rightarrow AvgPool \Rightarrow Conv \Rightarrow AvgPool \Rightarrow FC \Rightarrow FC \Rightarrow Output$$

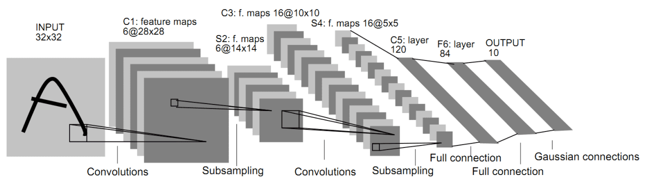


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Figure 3.6: LeNet-5 Architecture [21]

From 2010 to 2017 the *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) [28] took place and some of the winners of the competition have become the most widely used architectures for CNNs.

In Figure 3.7 the evolution of the top 5 error in the classification task can be seen how it evolved from 28.2% to 3.57% in only 5 years. It also shows the depth of the models and it can be seen that over the years the evolution has been towards deeper architectures.

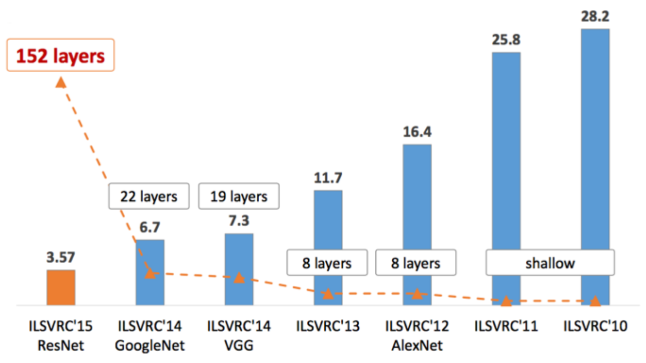
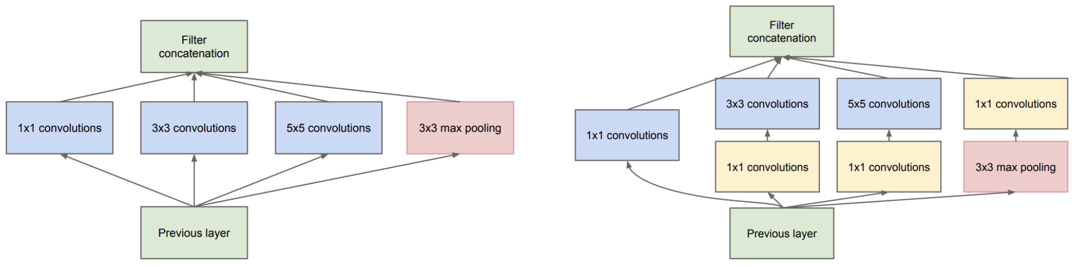


Figure 3.7: Evolution of the top 5 error in the classification task and architecture depth [33]

In 2012, *AlexNet* [19] won the ILSVRC classification challenge. They created a deeper and more complex CNN that had kernels of different sizes and more channels than *LeNet-5*. They also started using $ReLU$ in place of using $tanh$ or $sigmoid$ as an activation function.

The next important breakthrough arrived in 2014, the winner of the competition was *GoogLeNet* [30]. They designed a deeper network, it had 22 layers, nearly tripling the 8 layers used in *AlexNet*, and more than any of their predecessors. Although it is a much deeper network they managed to reduce the number of parameters by $12x$ in comparison with *AlexNet*, they obtained that by using Global Average Pooling instead of fully connected layers. Also, to deal with the issue of training deeper models they used multiple auxiliary classifiers between the model to prevent de gradient from dying. However, one of their major new ideas was using kernels of various sizes in parallel as shown in Figure 3.8.

The 2014 runner-up was *VGG* [29], which was also a big breakthrough as it was the beginning of very deep CNNs. While all previous CNN architectures used larger receptive fields (e.g., *AlexNet* used $11x11$ convolution kernels), *VGG* just used $3x3$ convolutions. According to the *VGG* authors, multiple $3x3$ convolutions stacked together are capable of replicating larger convolution filters and can outperform them. The architecture of this model is show in Figure 3.9.



(a) Inception module, naive version (b) Inception module with dimension reductions

Figure 3.8: Inception modules of the *GoogLeNet* architecture [30]

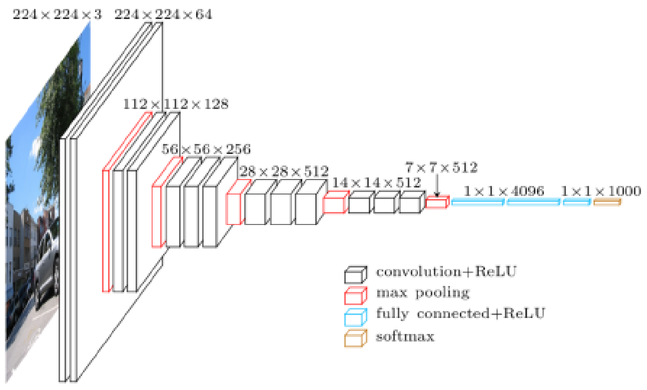


Figure 3.9: Architecture of the *VGG* network [29]

As CNNs got deeper and computer power increased, it was expected that CNNs would reach higher accuracy, but researchers soon noticed the problem of simply stacking multiple convolutional layers to create deeper models: the *vanishing gradient*.

The vanishing gradient problem appears when trying to train deep networks using gradient-based learning methods and backpropagation. In such type of methods the weights of the CNN update proportionally to the partial derivative of the error function with respect to the current weight. The problem is that traditional activation functions such as *sigmoid* or *tanh* have gradients smaller than 1 and as backpropagation computes the gradients using the chain rule, as we go higher in our network, the gradients become smaller reaching a point where they are practically 0 and the weights are not updated.

In 2015, ILSVRC was won by the *ResNet* architecture [11]. This model introduced shortcut or residual connections, which created an alternate path for the gradient to skip the middle layers and directly reach the initial layers, solving the vanishing gradient problem. The winner of that year was *ResNet-152*, which contained 152 layers, back then it was the deepest CNN, with a top 5 error of 3.57%, less than the human error, which is 5%.

The core idea of *ResNet* is that it is built of residual blocks, where these shortcuts connections that skip a few layers are introduced, a diagram of a residual block is shown in Figure 3.10.

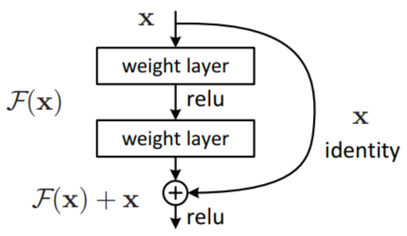

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3.10: Residual block [11]

In a regular CNN architecture, when we consider several stacked layers, we have a mapping from the input x to the output y, which is a function $y = H(x)$. In the residual blocks, the desired mapped function by the layers is $F(x) := H(x) - x$, so the output function is $H(x) = F(x) + x$.

These skip connections help solve the vanishing gradient problem as the gradient can backpropagate through them, but they also help by allowing these blocks of layers to learn the identity function, which ensures that if we add more layers it will perform at least as well as the lowest layer, all is needed is that $F(x) = 0$ which is easier and the output will be $H(x) = F(x) + x = 0 + x = x$.

Another advantage of these shortcuts connections is that they do not add extra parameters or computational needs.

Figure 3.11 shows the architecture of a *Resnet-34*.

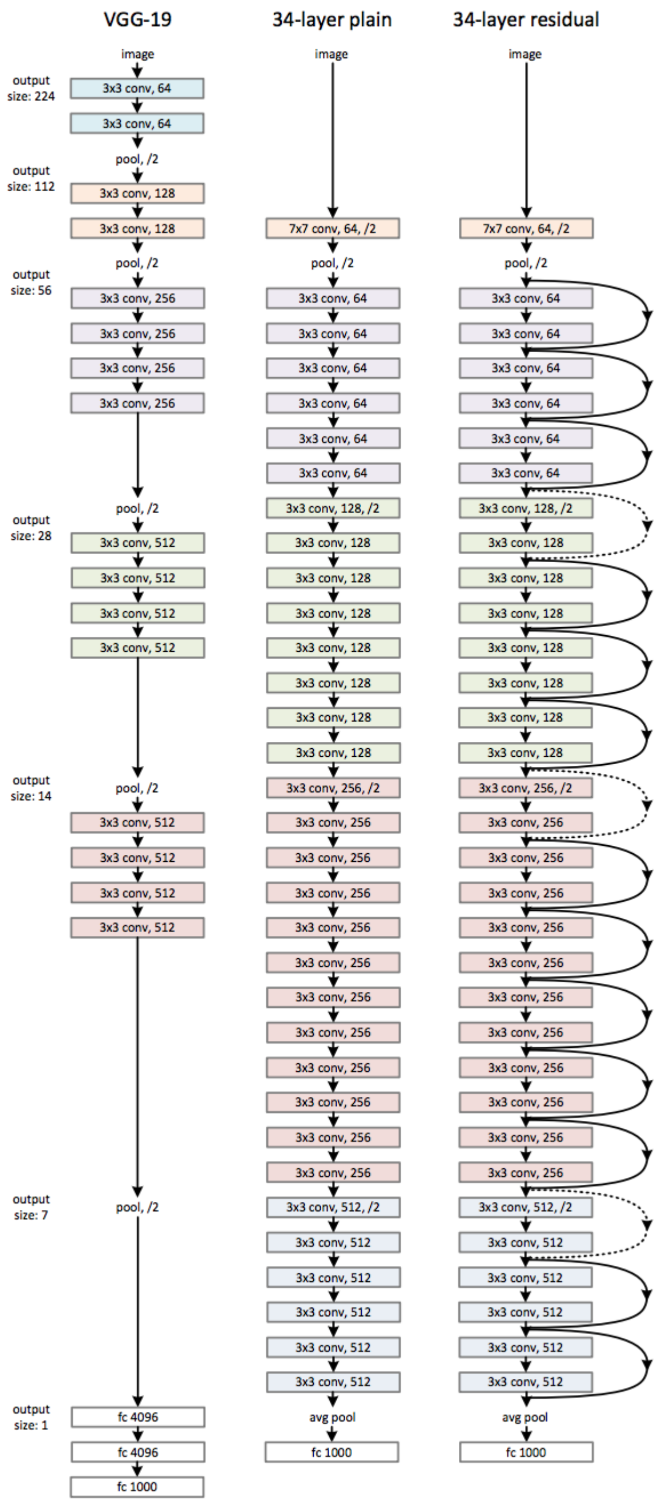


Figure 3.11: *ResNet-34* architecture compared to *VGG* and a plain 34 layer network [11]

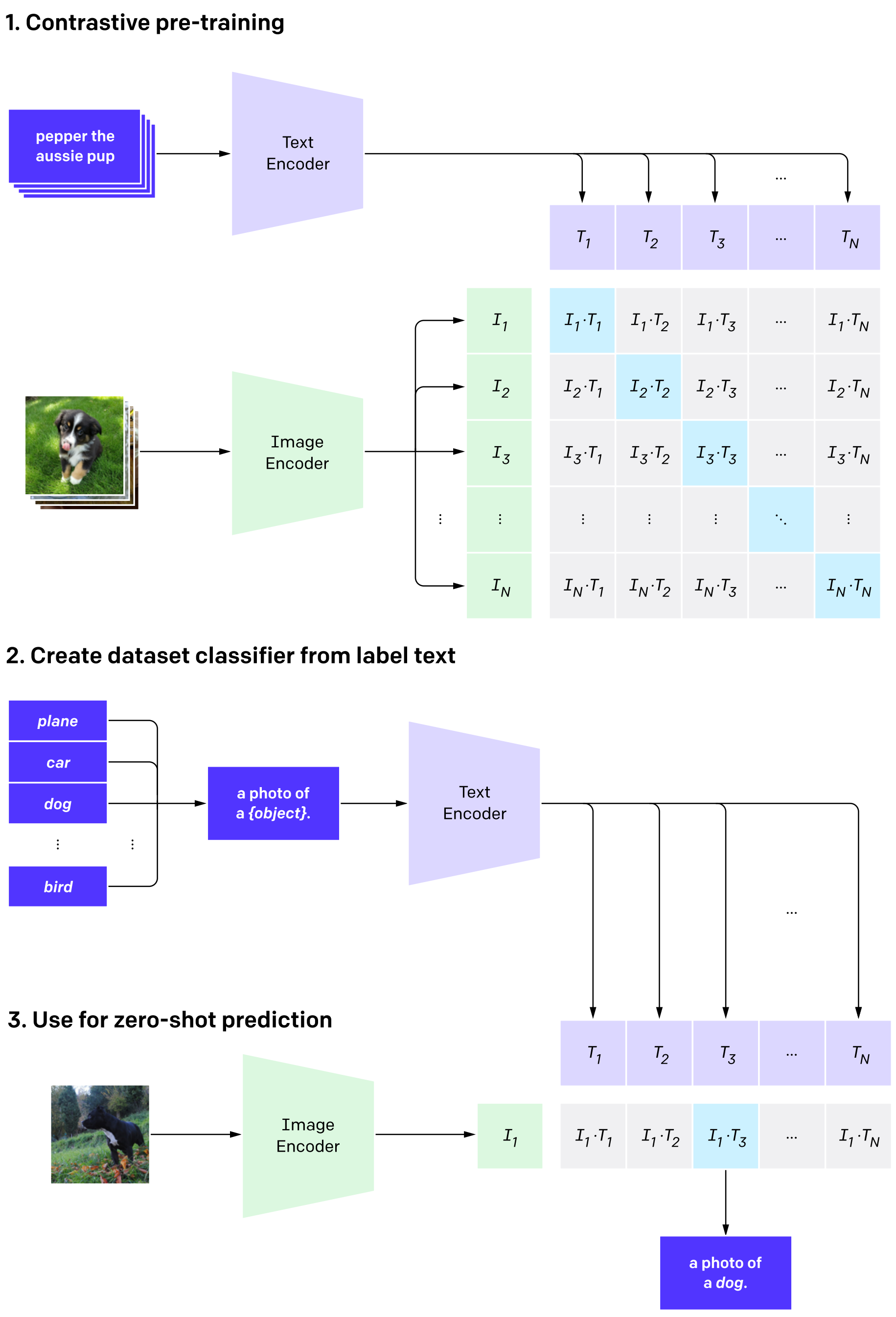
3.2.2.2 CLIP

CLIP [26] was introduced in 2021 by Radford, Wook et al. It is a contrastive approach to learn image representations from text, with a learning objective which maximizes similarity of correct text-image pairs embeddings in a large batch size. Specifically, given a batch of N image-text pairs, CLIP learns a multi-modal embedding space, by jointly training an image-encoder and a text-encoder, such that the cosine similarity of the valid N image-text pairs is maximized, while the remaining $N^2 - N$ pairs is minimized.

After pretraining, natural language is used to reference learned visual concepts or describe new ones enabling zero-shot transfer. Zero-shot learning is a problem where at test time, the model observes samples from classes which were not observed during training and needs to predict the class they belong to using some form of auxiliary information.

Figure 3.12 shows how the learning and testing process of the CLIP architecture works.

They manage to obtain SOTA results, for example, they match the accuracy of the original ResNet-50 [11] on ImageNet without needing to use any of the 1.28 million examples on which it was trained.



1. Contrastive pre-training
pepper the aussie pup
Text Encoder
Image Encoder
T_1 T_2 T_3 ... T_N
I_1 I_2 I_3 ... I_N
$I_1 \cdot T_1$ $I_1 \cdot T_2$ $I_1 \cdot T_3$ ... $I_1 \cdot T_N$
$I_2 \cdot T_1$ $I_2 \cdot T_2$ $I_2 \cdot T_3$ ... $I_2 \cdot T_N$
$I_3 \cdot T_1$ $I_3 \cdot T_2$ $I_3 \cdot T_3$ ... $I_3 \cdot T_N$
$I_N \cdot T_1$ $I_N \cdot T_2$ $I_N \cdot T_3$ ... $I_N \cdot T_N$
2. Create dataset classifier from label text
plane
car
dog
bird
a photo of a {object}.
Text Encoder
3. Use for zero-shot prediction
Image Encoder
I_1
T_1 T_2 T_3 ... T_N
$I_1 \cdot T_1$ $I_1 \cdot T_2$ $I_1 \cdot T_3$ ... $I_1 \cdot T_N$
a photo of a dog.

Figure 3.12: CLIP learning and testing process [26]

3.2.3 Training of neural networks

Neural networks are used to represent one function that maps an input $\boldsymbol{x} \in \mathbb{R}^n$ to an output $\boldsymbol{y} \in \mathbb{R}^m$ as $\boldsymbol{y} = f(\boldsymbol{x})$. This function f is the composition of all the layers in the NN, each one with their set of parameters: weights and biases. The objective of training a NN is to find the values that fit best this set of parameters so that the output of the network is closest to the desired output for each input. In order to tune this parameters we use a set of inputs called *training set* and we fit the values to these samples.

For being able to evaluate how well a set of parameters is working, we need to have a function that returns a value that tells how far the predictions of the NN are from the desired output, which is called *ground truth*, this function is called *loss function*. Let $\hat{\boldsymbol{y}} \in \mathbb{R}^m$ the output of the NN in a sample and $\boldsymbol{y} \in \mathbb{R}^m$ the ground-truth label for that sample, then the loss function returns a real number that measures how far is $\hat{\boldsymbol{y}}$ from $\boldsymbol{y}$.

There are many different loss functions depending on the task of the NN, for example, in the case of a regression task, L^2 or L^1 losses are the most used. In the case of a classification task, the most used loss function is the cross-entropy. Equation 2 shows an example of these loss functions.

$$L^1(\hat{\boldsymbol{y}}, \boldsymbol{y}) = |\hat{\boldsymbol{y}} - \boldsymbol{y}|, \quad L^2(\hat{\boldsymbol{y}}, \boldsymbol{y}) = (\hat{\boldsymbol{y}} - \boldsymbol{y})^2, \quad L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{c=0}^{C-1} \boldsymbol{y}_c log(\hat{\boldsymbol{y}}_c) \qquad (2)$$

After the loss function is calculated, the parameters need to be updated according to this value so they adapt to our specific task and dataset. There are many different algorithms that calculate these updates such as Stochastic Gradient Descent (SGD), Adam [18], AdaGrad [4]... But they all have in common that first-order derivatives of the loss function with respect to each parameter are needed for the update.

Backpropagation [27] is the algorithm that calculates these derivatives and is the core algorithm of NN training. It uses the chain rule recursively to calculate the gradients: Consider two neurons from consecutive layers, let $\boldsymbol{w}$ and b be the weights and bias of the first neuron and suppose that we know the derivative of the loss with respect to the output of the first unit $y = f(\boldsymbol{w}x + b)$, which is the input of the second unit, $\frac{\partial L}{\partial y}$, then applying the chain rule we obtain:

$$\frac{\partial L}{\partial w_i} = \frac{\partial y}{\partial w_i}\frac{\partial L}{\partial y}, \quad \frac{\partial L}{\partial b} = \frac{\partial y}{\partial b}\frac{\partial L}{\partial y}, \quad \frac{\partial L}{\partial x_i} = \frac{\partial y}{\partial x_i}\frac{\partial L}{\partial y} \tag{3}$$

As we know that $y = f(\boldsymbol{w}x + b)$ we can easily compute $\frac{\partial y}{\partial w_i}$, $\frac{\partial y}{\partial b}$ and $\frac{\partial y}{\partial x_i}$ and once we know $\frac{\partial L}{\partial x_i}$ we can repeat this process recursively until we reach the first layer.

4 Dataset

In this section of the thesis, we will explore the dataset used for this work and explain how it was collected.

The used images are collected from the EarthCam website, a network of tourism webcams with a searchable database of iconic places and views from around the world. They provide live streaming video, time-lapse cameras and photo documentation. More specifically, the images are extracted from what it is called *Hall of Fame*, a system that the users can use to take screenshots of the current image of the camera and then this image becomes publicly available.

After preprocessing the different cameras and images, such as discarding cameras with no geographic information or indoor cameras, we decided to use 123 cameras and downloaded the images between December 2020 and December 2021 discarding the images taken during the night as usually they are too dark for them to be useful in our task. The total number of images is 275002.

For the weather data we used the OpenWeather API, an API from which you can get historical weather data for a set of coordinates with hour granularity, this data includes temperature, humidity, atmospheric pressure, main weather condition and other conditions. Using the timestamp of each image, we were able to associate the current weather condition with each image.

Figure 4.1 shows an example of some images extracted from the dataset with the name of the camera and the weather conditions associated with it: the main current weather condition, which can be one of *Clear*, *Cloudy*, *Rain* or *Snow* and the temperature in Kelvin degrees.

4.1 Weather condition labels

In the case of the weather condition labels we had to do a little preprocessing, as some labels were not accurate due to various causes:

- Weather data has hour granularity and in one hour the main weather condition can change a lot.

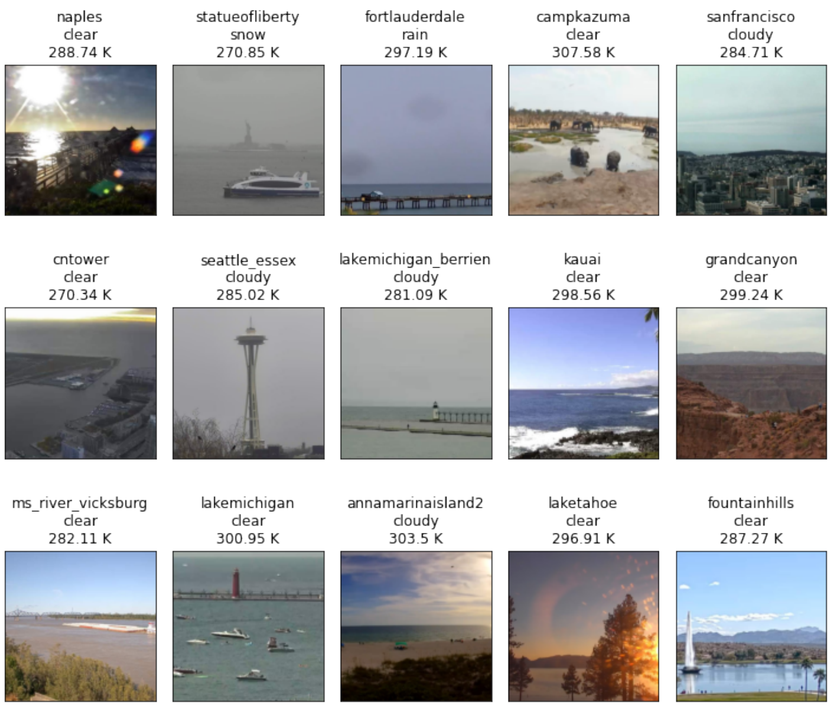


Figure 4.1: Example of images in the dataset with the name of their camera and their weather condition

- The visual weather conditions can change a lot in a small area, and sometimes, depending on where the camera is looking at, it can differ.
- The weather data is sometimes an approximation as they may not have a weather station at the location of the camera.

This website shows the main weather condition codes returned by the OpenWeather API. We will use these codes to assign a class to our images. The API divides the codes in 7 groups:

- **Group 2XX: Thunderstorm:** Used to describe different types of thunderstorm. We will classify it as *rain*.
- **Group 3XX: Drizzle:** Used to describe different types of drizzle. We will classify it as *rain*.
- **Group 5XX: Rain:** Used to describe different types of rain. We will classify it as *rain*.

- **Group 6XX: Snow:** Used to describe different types of snow. We will classify it as *snow*.

- **Group 7XX: Atmosphere:** Used to describe different types of atmospheric phenomenons such as mist, haze, dust, fog, sand... After checking some images with this class we decide to discard them as there are not many examples and most of them the difference is not clear visually. Figure 4.2 shows an example of some images with this code.

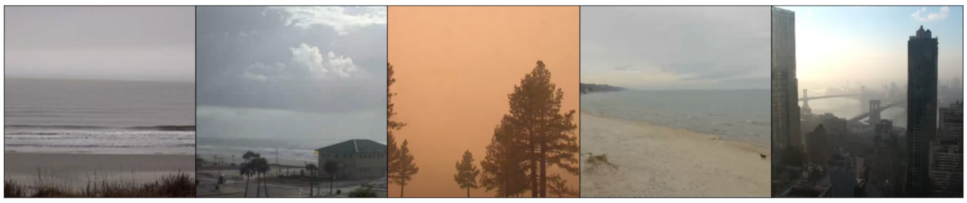


Figure 4.2: Some examples of images where the weather code is from the group 7XX: Atmosphere

- **Group 800: Clear:** Used to describe a clear sky. We will classify it as *clear*.

- **Group 80X: Clouds:** Used to describe different levels of clouds in the sky. There are 4 levels:

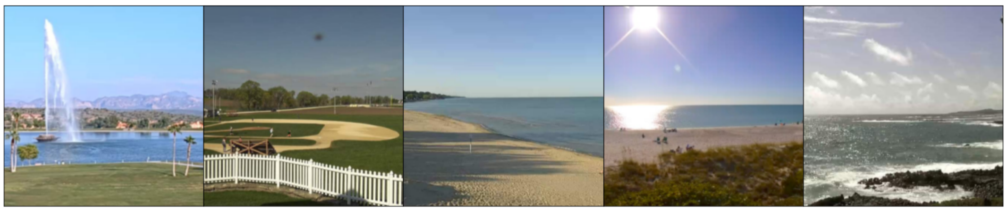
  - **801: Few clouds:** 11-25%

  - **802: Scattered clouds:** 25-50%

  - **803: Broken clouds:** 50-85%

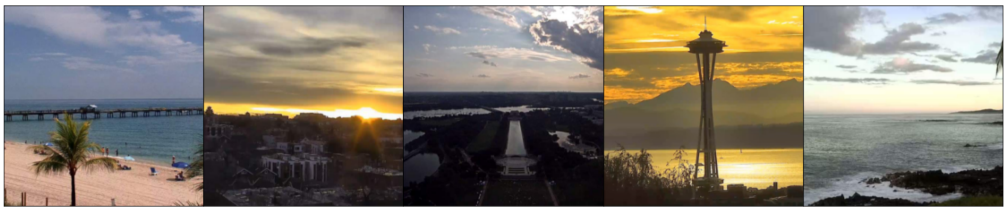
  - **804: Overcast clouds:** 85-100%

  Figure 4.3 shows some examples of images of the different 80X: Clouds conditions codes. It can be seen that the weather data is not always accurate as there are some images where the proportion of covered sky by clouds does not match the associated weather data. Also, Figure 4.3a shows some example images where the weather code is 800: Clear, it can be seen that some images also have clouds. For that reason, since the majority of images have associated some of these codes, we will discard images where the associated code is 801, 802 or 803 and we will classify images with weather code 804 as *cloudy*.


(a) Some examples of images where the weather code is 800: Clear

(b) Some examples of images where the weather code is 801: Few Clouds

(c) Some examples of images where the weather code is 802: Scattered Clouds


(d) Some examples of images where the weather code is 803: Broken Clouds

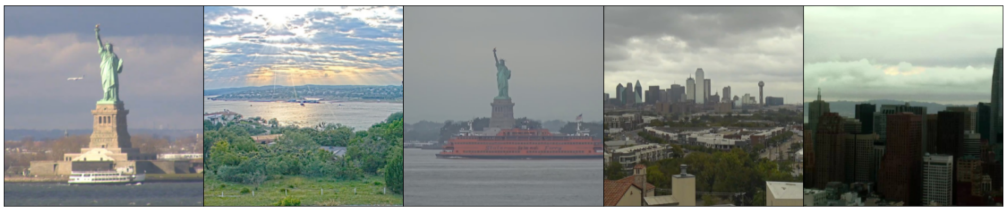

(e) Some examples of images where the weather code is 804: Overcats Clouds

Figure 4.3: Some examples of images where the weather code is from the group 80X: Clouds

One of the big problems found is the hour granularity of the weather data and that in one hour the weather can change a lot, to solve that we will only use images where the condition of the previous hour and the condition of the next hour are the same. Figure 4.4 shows two examples of rapidly changing weather conditions.

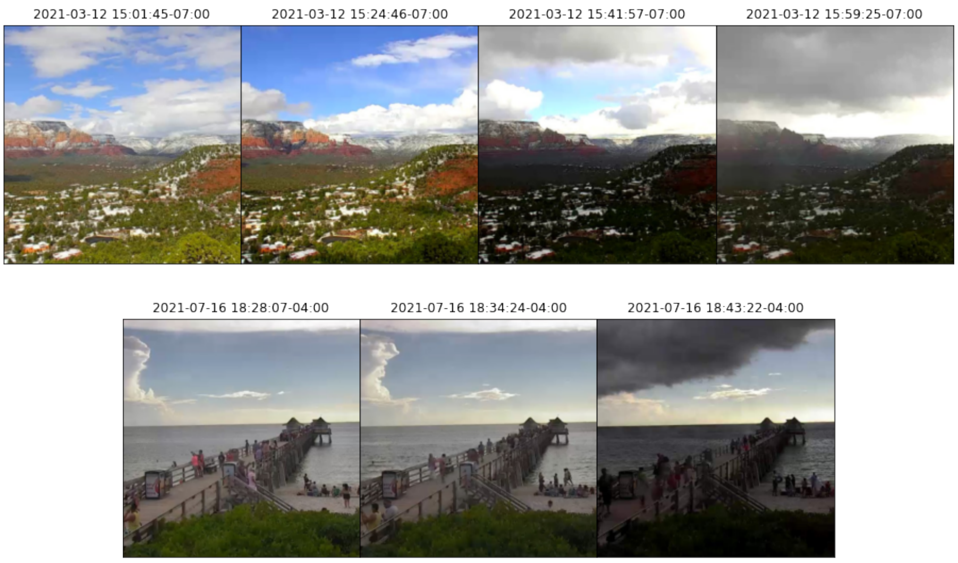


Figure 4.4: Some examples of sequence of images where the weather conditions change rapidly

After all that preprocessing we end with 124610 images with the following distribution:

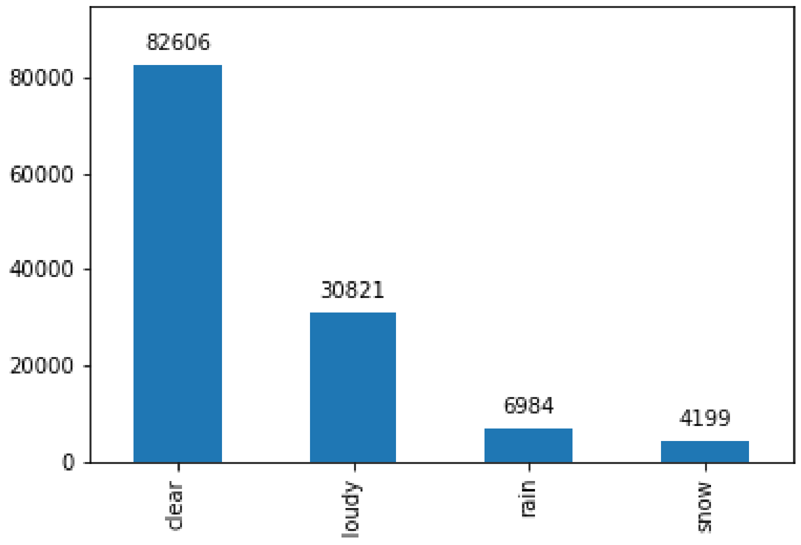


Figure 4.5: Class distribution of the final dataset

We can clearly see that the dataset is quite unbalanced as there are nearly 20x images with the *clear* class than with *snow* class but we will be able to solve this problem using weighted losses.

4.2 Temperature

In the case of temperature, we will use the weather data and we will interpolate linearly using the temperature in the previous and in the next hour using the following formula:

$$temperature = (1 - \frac{t \,\%\, 3600}{3600}) * previous\_temp + \frac{t \,\%\, 3600}{3600} * next\_temp \tag{4}$$

Where t is the timestamp (in seconds) of the image and $\%$ is the modulo operator.

We get the following distribution of temperatures:

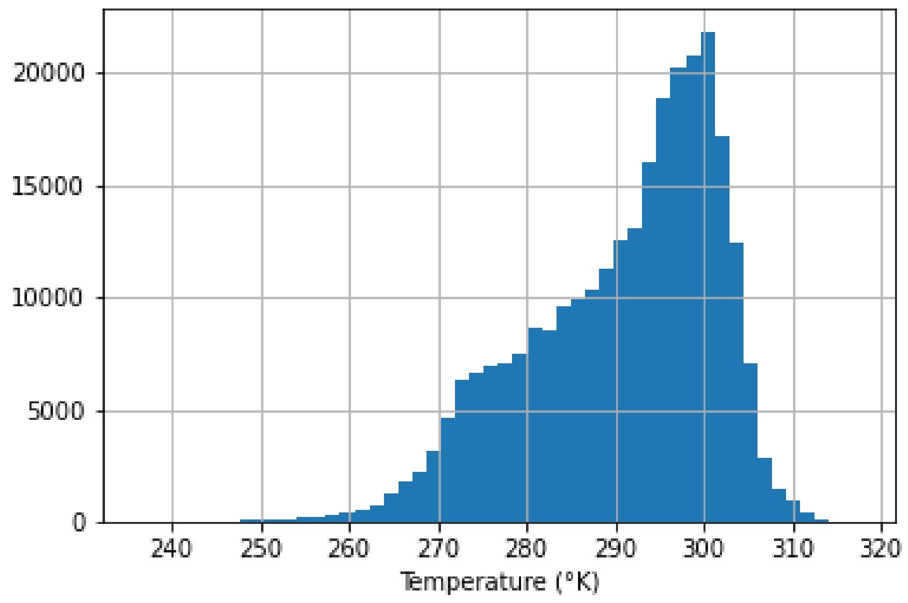


Figure 4.6: Temperature distribution of the dataset

Units are in the International System (SI), which are Kelvin degrees. Remember that Kelvin degrees are just a translation of Celcius degrees using the following formula:

$$T_{(°K)} = T_{(°C)} + 273.15 \tag{5}$$

To speed up the training of our neural network we will standarize the temperature value so it has a 0 mean and a standard deviation of 1 using the following formula:

$$T_{norm} = \frac{T - \mu}{\sigma} \tag{6}$$

Where $\mu = \frac{\sum_{i=1}^{N} T_i}{N}$ and $\sigma = \sqrt{\frac{\sum_{i=1}^{N}(T_i - \mu)^2}{N-1}}$ are the mean and unbiased standard deviation of the samples.

5 Weather classifier

In this section of the thesis we will explore the different architectures and models used to predict the main weather condition from an image.

This type of task is called multi-class classification as we have a set of data points of size N, in our case RGB Images, where each sample $\{x_i\}_{i=1}^{N}$ is labeled with a discrete label y_i that has k different classes, i.e. $y_i \in \{0, \ldots, k-1\} \; \forall i \in \{1, \ldots, N\}$.

The goal in this case is that the output represents the probability distribution of the input image belonging to each class, therefore we need to have k outputs, each one representing the probability of the image to be labeled with that class, consequently, all the outputs need to sum 1.

To solve this issue, the **Softmax function** is used at the end of the neural network. The softmax function is a generalization of the sigmoid function (3.2.1.1) to multiple dimensions. It takes a vector $\boldsymbol{z}$ of size k as input and normalizes it into a probability distribution. It is given by the following formula:

$$\text{Softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=0}^{k-1} e^{z_j}} \; \forall i \in \{0, \ldots, k-1\} \text{ where } \boldsymbol{z} = (z_0, \ldots, z_{k-1}) \in \mathbb{R}^k \quad (7)$$

5.1 Metrics

In this case, as we are training a classifier, there are multiple metrics that are of our interest when evaluating its performance. Some metrics are measured for the entire dataset, such as *accuracy*, but others are measured separately for each class and then averaged. There are multiple ways to average the results:

- **Macro average:** Arithmetic mean of the scores for each classes.
- **Weighted average:** The mean is calculated using the support (number of samples) of each class to give more importance to classes that have more representation.

5.1.1 Accuracy

Accuracy is the simplest metric used when we evaluate a multi-label classifier, it is defined as the proportion of correct predictions:

$$Accuracy = \frac{\sum_{i=1}^{N} I(\hat{y}_i = y_i)}{N} \tag{8}$$

Where $\hat{y}_i$ is the class prediction for sample i, y_i is the ground-truth class for sample i and I is the indicator function, which returns 1 if the classes match and 0 otherwise.

The main issue with this metric is that when we have an unbalanced dataset, as in our case, if the model just predicts the most represented class it will have high accuracy but it would be a useless model.

5.1.2 Precision

Precision is a metric measured for each class, it calculates how many of the predictions that the model made for that class were actually correct, it is defined with the following formula:

$$Precision_c = \frac{TP_c}{TP_c + FP_c} \tag{9}$$

Where TP_c is the number of True Positives of class c, i.e. the number of samples predicted as class c and their label is class c and FP_c is the number of False Positives of class c, i.e. the number of samples that are predicted as class c but their label is not class c.

5.1.3 Recall

Recall is a metric measured for each class too, it indicates how many of the samples of that class has the model predicted correctly into that class, it is defined with the following formula:

$$Recall_c = \frac{TP_c}{TP_c + FN_c} \tag{10}$$

Where TP_c is the number of True Positives of class c, i.e. the number of samples that are predicted as class c and their label is class c and FN_c is the number of

False Negatives of class c, i.e. the number of samples whose label is class c but are not predicted as class c.

5.1.4 F1-score

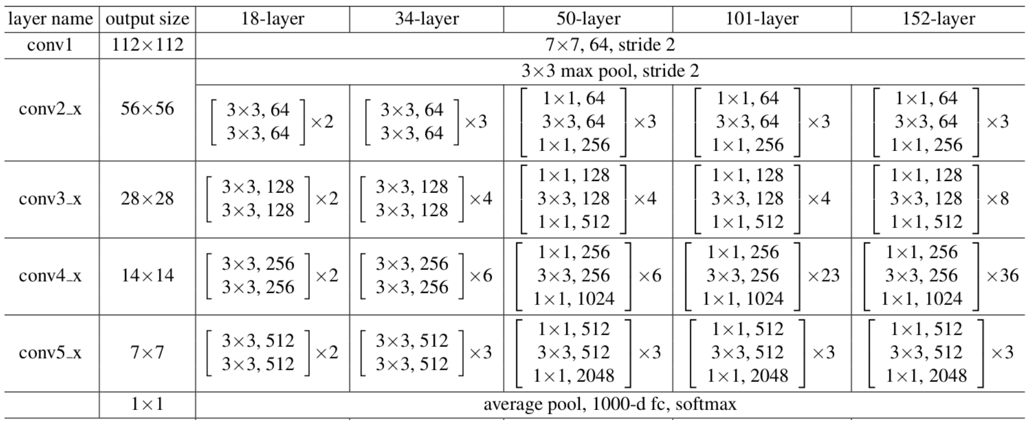
The F1-score of a class is the harmonic mean between its precision p_c and its recall r_c:

$$F1\text{-}score_c = \frac{2Precision_c \cdot Recall_c}{Precision_c + Recall_c} = \frac{TP_c}{TP_c + \frac{1}{2}(FP_c + FN_c)} \tag{11}$$

As in our case, we are interested in having a balance between precision and recall, we will use this metric.

5.2 ResNet

As explained in 3.2.2.1, ResNet [11] architecture has become one of the most used architectures since it won the ILSVRC competition in 2015. We will use two of their variants, ResNet-18 and ResNet-50, as the CNN in our network and we will change the last fully connected layers to adapt it to our task of weather classification.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

Figure 5.1: Different architectures for ResNet [11]

As we can see, the main block structure is different for the deeper versions (ResNet-50, ResNet-101, and ResNet-152), 1x1 convolutions are introduced in order to reduce the training time as they are used to reduce the dimensions and then restore the original dimensions leaving the 3x3 layers with reduced input/output dimensions.

We will use the pretrained weights on ImageNet as we are working on RGB images too but as our task is quite different from the ImageNet we will also retrain some layers of the ResNet besides the last fully connected layers by setting the gradients to *True* or *False* as desired.

5.3 CLIP

To be able to compare the results we will use the pretrained ResNet-50 using CLIP [26]. They use the improvements introduced in ResNet-D by He et al. [12] and the antialiased rect-2 blur pooling from Zhang [37].

They also replace the global average pooling layer with an attention pooling mechanism. The attention pooling is implemented as a single layer of 'transformer-style' multi-head QKV attention where the query is conditioned on the global average-pooled representation of the image [32, 3].

Using CLIP pretrained weights, we will use two different architectures to fine-tune the weights for our task:

1. Use CLIP training architecture with the text encoder and the visual encoder and maximize the cosine similarity between the valid image-text pairs using the following texts:
   - "A picture of a sunny day"
   - "A picture of a cloudy day"
   - "A picture of a rainy day"
   - "A picture of a snowy day"
2. Use the features extracted by the vision model and adding a fully connected layer at the end with 4 outputs so it can be used as a classifier.

We will set the gradients to *True* or *False* as desired to retrain only some of the layers of the model.

5.4 Loss

The Loss function determines how correct or wrong the current network is. As we are working with a multi-class classification problem we will use the **cross-entropy** loss, which is defined by the following formula for each sample:

$$\text{CE}(x_i) = -\sum_{j=0}^{k-1} t_j \cdot \log\left(\text{Softmax}(f(x_i))_j\right) \tag{12}$$

Where x_i is a sample of the dataset, f is the function calculated by the neural network and $\mathbf{t} \in \mathbb{R}^{\mathbf{k}}$ is the target vector, a one-hot encoded vector where the value at the position y_i is 1 and the rest of the values are 0.

As in the target vector only one value is different from 0 the rest of the elements of the sum are canceled, and we can use the following simpler formula:

$$\text{CE}(x_i) = -\log\left(\text{Softmax}(f(x_i))_{y_i}\right) \tag{13}$$

A problem of using the cross-entropy loss is that it gives equal importance to all samples and while this is no problem for a balanced dataset, in our case, as we have a heavily unbalanced dataset we will use different weights for each sample:

$$\text{CE}(x_i) = -w_i \cdot \log\left(\text{Softmax}(f(x_i))_{y_i}\right) \tag{14}$$

We will try two different weighting strategies:

- **Weighted per class:** This is the most commonly used strategy to weight each sample when we have an unbalanced dataset. Each sample is weighted according to its class, the weights of each class are calculated using the following formula proposed by King and Zeng in 2001 [31]:

$$w_i = W_{y_i} = \frac{N}{k * n\_samples\_y_i} \text{ where } y_i \in \{0, \ldots, k-1\} \tag{15}$$

  Where N is the total number of training examples, k is the number of classes and $n\_samples\_y_i$ is the number of samples of the y_i class.

- **Weighted per class and camera:** As each camera has a different distribution of classes we will try using different weights for each camera and each class. We will use the formula proposed by King and Zeng [31] but using the corresponding numbers for each different camera, i.e.:

$$w_i = W_{cam_i,y_i} = \frac{N_{cam_i}}{k_{cam_i} * n\_samples\_cam_i\_y_i} \text{ where } y_i \in \{0,\dots,k-1\} \quad (16)$$

  Where N_{cam_i} is the number of samples from the camera sample x_i, k_{cam_i} is the number of classes present in the samples from the corresponding camera and $n\_samples\_cam_i\_y_i$ is the number of samples of class y_i of the camera.

6 Temperature regressor

In this section of the thesis we will explore the different architectures and models used to predict the temperature from an image.

This type of task is called regression, as we have a set of data points of size N, in our case RGB Images, where each sample $\{x_i\}_{i=1}^{N}$ is labeled with a continuous label y_i, i.e. $y_i \in \mathbb{R}\ \forall i \in \{1, \ldots, N\}$.

One of the characteristics that all previous works on predicting temperature from images [2, 9, 34] share is the fact that they all use images from the same places as training images and for evaluation. In our case, we will use the images on some cameras as training and evaluate on images from different cameras such that the CNN has to learn patterns that can be found on any image and is able to generalize to images taken from other cameras.

6.1 Metrics

In the case of regression there are four main metrics that are commonly used for evaluating the performance of a model:

6.1.1 Mean Squared Error (MSE)

Mean Squared Error (MSE) is one of the most popular used loss function for regression problems, it is calculated as the mean of the squared differences between predicted and expected target value:

$$MSE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{\sum_{i=1}^{N} (\hat{y}_i - y_i)^2}{N} \tag{17}$$

Where $\boldsymbol{y} = \{y_1, \ldots, y_N\}$ are the target value for all the elements in the dataset and $\hat{\boldsymbol{y}} = \{\hat{y}_1, \ldots, \hat{y}_N\}$ are the predicted values for each sample.

The squaring has the effect of magnifying large error, so if MSE is used as the loss function, it penalizes the models more for large error. The units of the MSE are squared units so usually Root Mean Squared Error (6.1.2) is used to report values as it has the same units as the prediction.

6.1.2 Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE) is an extension of MSE, it is the square root of MSE, as it has the same units as the target variable is common to train the model using MSE and use RMSE to evaluate and report its performance. The RMSE can be calculated using the following formula:

$$RMSE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sqrt{MSE(\hat{\boldsymbol{y}}, \boldsymbol{y})} = \sqrt{\frac{\sum_{i=1}^{N} (\hat{y}_i - y_i)^2}{N}} \tag{18}$$

6.1.3 Mean Absolute Error (MAE)

Mean Absolute Error (MAE) is a widely used metric and loss function too as like RMSE it has the same units as the target variable.

Unlike RMSE and MSE, MAE does not penalize more large errors than smaller errors as it increases linearly with the error value.

MAE is calculated using the following formula:

$$MAE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{N} \tag{19}$$

6.1.4 $\boldsymbol{R^2}$

The coefficient of determination (R^2), is the proportion of the variation in the dependent variable, the temperature in our case, explained by the model. It is defined with the following formula:

$$R^2(\hat{\boldsymbol{y}}, \boldsymbol{y}) = 1 - \frac{SS_{Res}}{SS_{Total}} = 1 - \frac{\sum_{i=1}^{N} (\hat{y}_i - y_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2} \tag{20}$$

Where $\bar{y} = \frac{1}{N} \sum_{i=1}^{N} y_i$ is the mean value of $\boldsymbol{y}$.

6.2 Using all cameras

For the first experiments we will use images from all the cameras for training and testing to check that the CNN is able to learn patterns such that temperature is a feature that is possible to be predicted.

We will use two variants of the ResNet [11] architecture, ResNet-18 and ResNet-50, the CNN will be used as a feature extractor, fine-tuning some of the layers, and we will add some fully connected layers at the end of the network with one output neuron that will output the predicted temperature for the image.

Figure 6.1 shows a diagram of the architecture of the network.

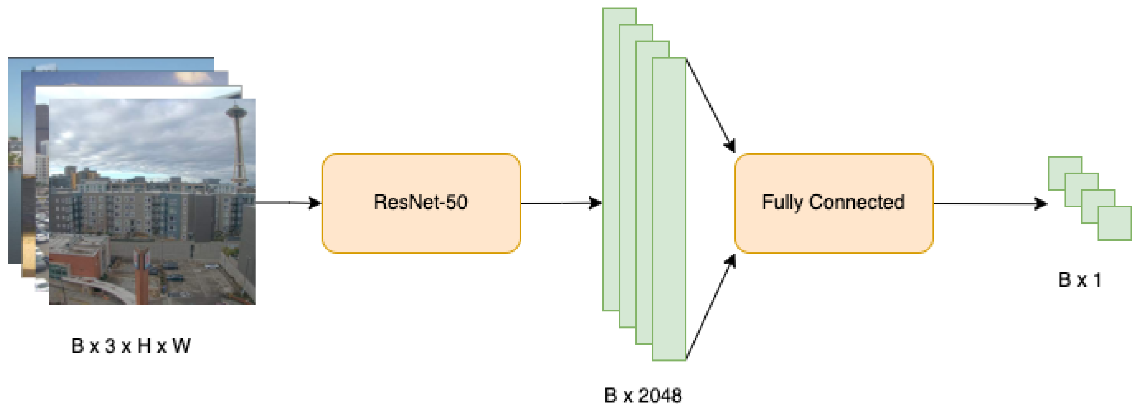


Figure 6.1: Architecture of the network for predicting temperature from images using only the image to predict the temperature from as input. B is the batch size and H and W the image height and width respectively. In this diagram $B = 4$

6.3 Different evaluation cameras

As explained in 6, all of the previous work done about temperature prediction use images from the same scenes/cameras for training and evaluation. This part of the thesis focuses in different architectures tested for trying to learn patterns that can be found on any image.

We choose 7 cameras to use as testing images that have 55292 images in total so that they make up 20.11% of the complete dataset of 275002 images. We will use the remaining 219710 images as training images for the network. Figure 6.2 shows some images extracted from the evaluating cameras and their temperature. We can see that they are quite different between each other so we have a representative result.

Also, figure 6.3 shows the probability density functions for training and evaluation data and for each testing camera, we can see that both training and testing images have similar probability density functions when we aggregate all the cameras but some cameras have quite different distributions, for example, camera *campkazuma* looks like a normal distribution but *statueofliberty* looks more like two different normal distributions, probably one for the summer season and another one for the winter season.

The main problem with using different scenes/cameras for training and evaluating is that, when we evaluate the performance on the testing images, the neural net has not seen similar images during training and it does not know how to predict the temperature for images from that camera, to solve that, we propose different architectures that use what we call **reference images**, the basic idea is that in addition to the image from which we want to predict the temperature, we also input into the neural network some other images with their temperature from the same camera/scene as the main image. The idea behind that is that the network can know what the images are like from that camera and what the temperature is like on that camera.

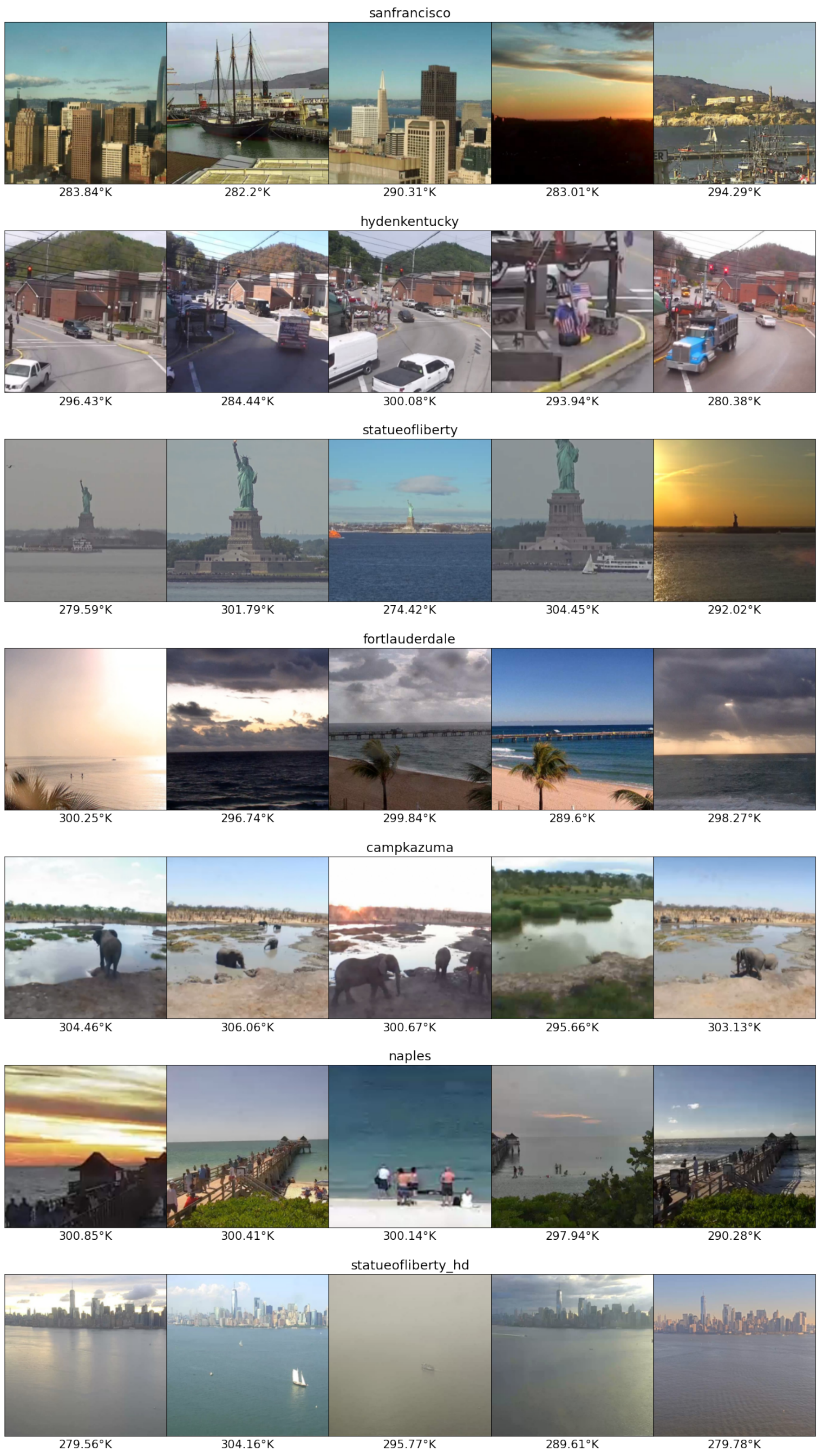


Figure 6.2: Example images extracted from the evaluation cameras and their temperature

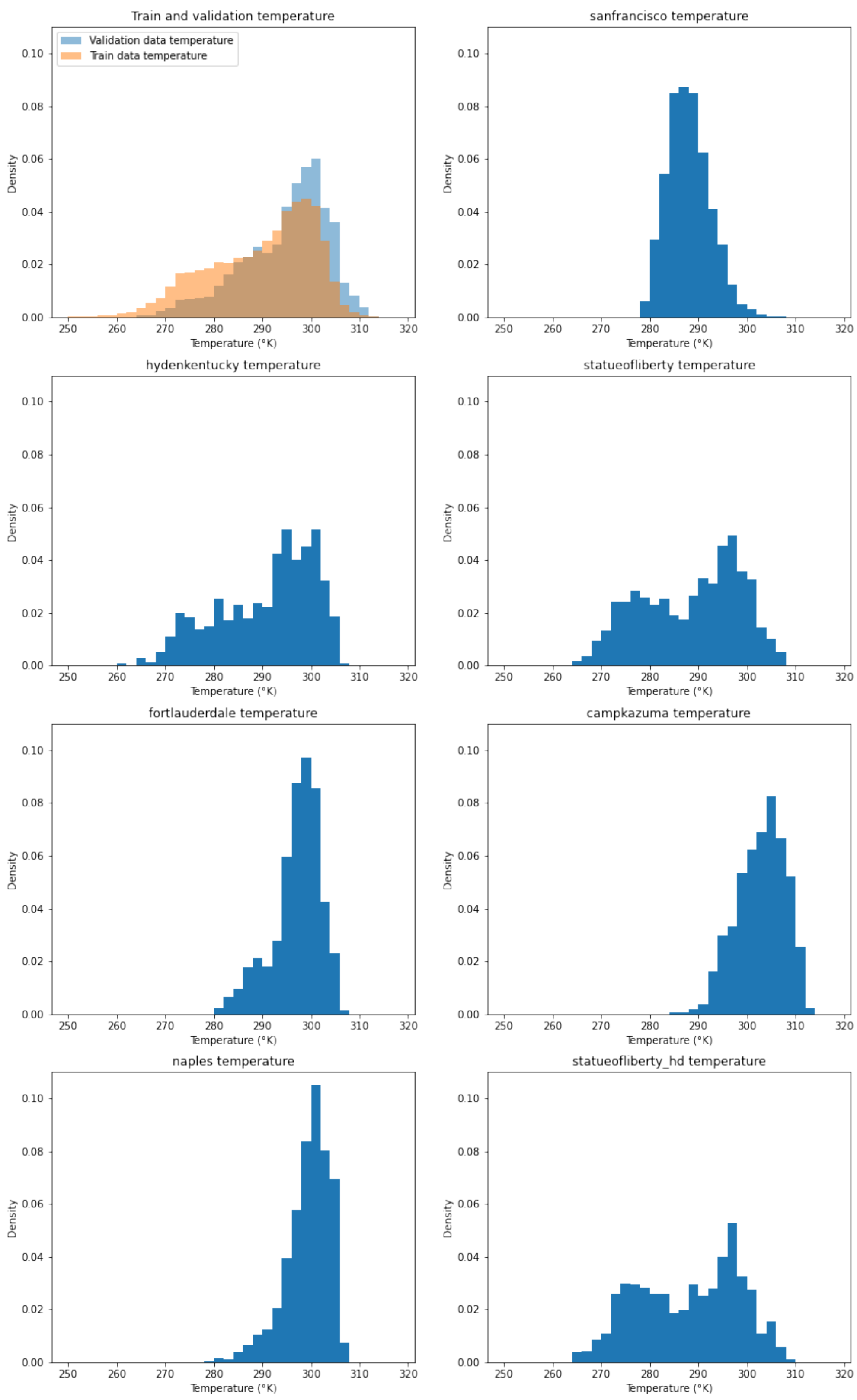


Figure 6.3: Probability density function of the temperature for train and validation data and for each camera on the validation dataset

6.3.1 Without reference images

The first experiments we will try and that we will use as baseline will be using the same architecture as in Section 6.2 and shown in figure 6.1. This will allow us to check if there is a big or a small difference in the performance of a model in the case we use different cameras for evaluating than the ones used in training.

6.3.2 Using reference images

As explained in 6.3 we will input to the network some images from the same camera with their temperature in addition to the image we are trying to predict the temperature from for the network to know how the images in that camera look like. We will choose these images randomly from all the images from the same camera but for them to be representative of the camera temperature we will sort the images of the camera by temperature and choose references within a range from around specific percentiles. For example, in the case we use 3 reference images we will choose images from around the 0.1, 0.5 and 0.9 percentile with a range of 0.05 around each percentile, i.e. we will choose one image randomly that their temperature is between the 0.05 and 0.15 percentile, another between 0.45 and 0.55 and a last one between 0.85 and 0.95.

6.3.2.1 Batch concatenating

Our first architecture will be using what we call **batch concatenating**, what we do is concatenate the main images and the references images in the batch dimension, in such a manner that we effectively have a batch size of $B * (N\_REF + 1)$ where B is the original batch size and N_REF is the number of reference images and the $+1$ comes from the original image. After concatenating all the images in the batch dimension, we pass them through a ResNet-18 network getting a feature 512-dimensional vector for each image, the main images, and the reference images. Next we reshape the $B * (N\_REF + 1)$ (512)-dimensional vectors such that we have B $(512 * (N\_REF + 1))$-dimensional vectors, basically we have one vector for each main image with the features of the main images and their references images. To this vector we concatenate the references temperatures at the end, getting B $(512 * (N\_REF + 1) + N\_REF)$-dimensional vectors that we pass through fully connected layers and finally getting B (1)-dimensional vectors which represent the temperature prediction for each of the main images.

We will set the gradients to *True* or *False* as desired to retrain only some of the layers of the ResNet model.

Figure 6.4 shows a diagram of this architecture.

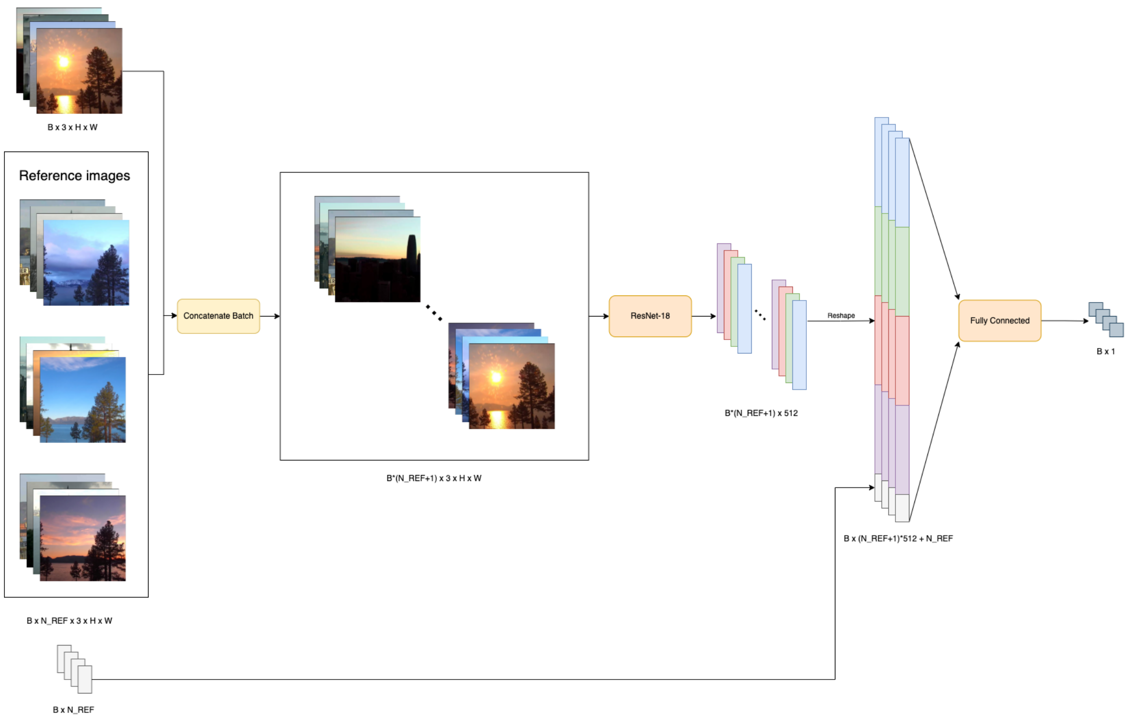


Figure 6.4: Architecture of batch concatenating network. B is the batch size, N_REF is the number of reference images used, and H and W the image height and width respectively. In this diagram $B = 4$ and $N\_REF = 3$

6.3.2.2 Channel concatenating

The second architecture we propose is called **channel concatenating**, instead of concatenating all the images in the batch dimension we concatenate them in the channel dimension (the second dimension as we use channel first representation) getting "$(N\_REF+1)*3$ - channels images". Following that we pass these images through a modified ResNet-18 where we modified the number of input channels of the first convolutional layer to match the number of channels of the new input images.

For the initialization of the new weights of the first layer we will use the weights of the RGB channels of the regular ResNet architecture, i.e. we will initialize the weights of each of the red channels of every image, the main image and the reference images, with the weights of the red channel of the pretrained resnet, we will do the same with the green and blue channels. Figure 6.5 shows a diagram of how these weights are initialized.

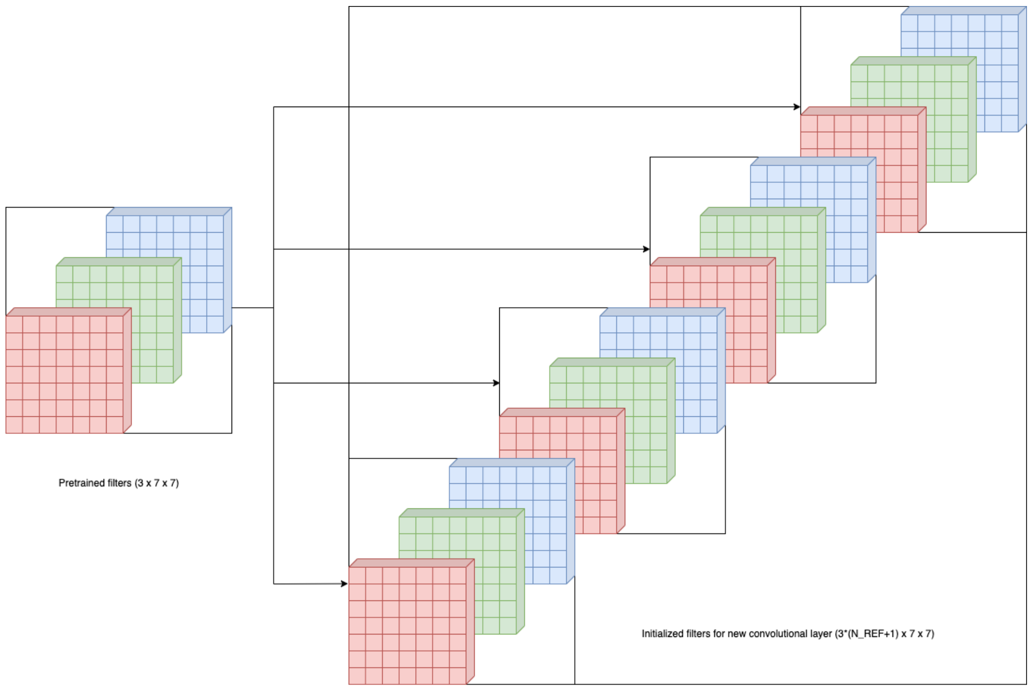

Pretrained filters (3 x 7 x 7)
Initialized filters for new convolutional layer (3*(N_REF+1) x 7 x 7)

Figure 6.5: Diagram showing how weights are initialized in the first convolution layer for the channel concatenating architecture. N_REF is the number of reference images used. In this diagram $N\_REF = 3$.

From the ResNet-18 we output B (512)-dimensional vectors where we concatenate the references temperatures getting B $(512 + N\_REF)$-dimensional vectors that we pass through fully connected layers and we finally get B 1-dimensional vectors which represent the temperature prediction for each of the main images.

In this case, as we modified the first layer of the network and added some new weights, we will have to retrain all the ResNet model.

Figure 6.6 shows a diagram of this architecture.

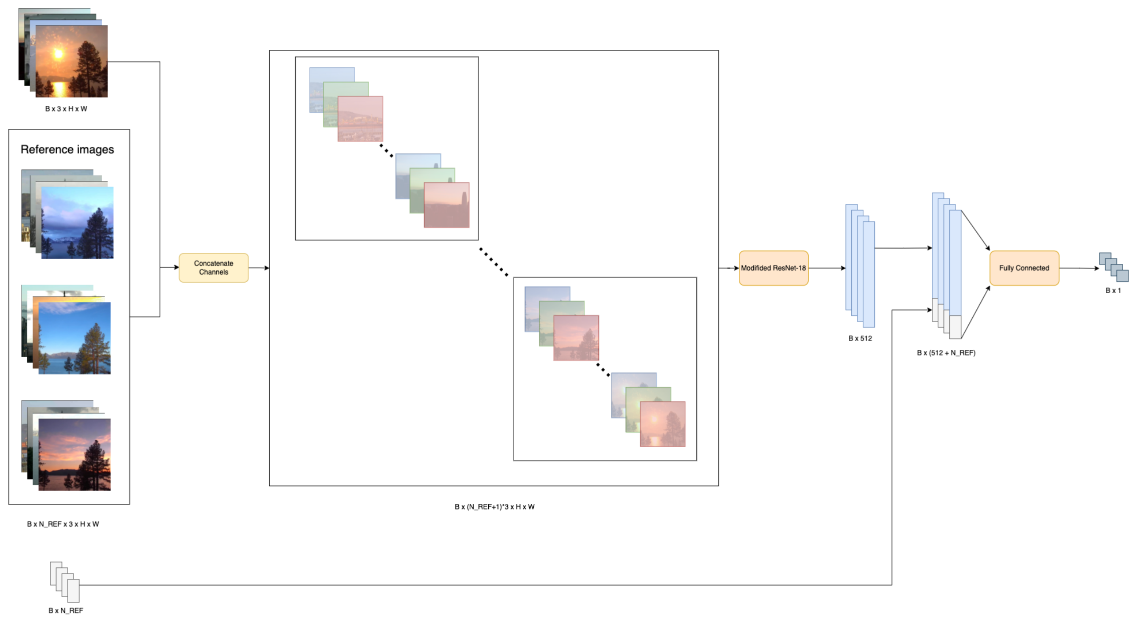


Figure 6.6: Architecture of channel concatenating network. B is the batch size, N_REF is the number of reference images used, and H and W the image height and width respectively. In this diagram $B = 4$ and $N\_REF = 3$

6.3.2.3 Two CNNs

The last architecture we propose we call it **two CNNs**, because it uses two different convolutional networks, one for the main images and another for the reference images.

For the main images we make a forward pass through a ResNet-18 and get one (512)-dimensional vector for each image. In the case of the reference images we use batch concatenating, we concatenate all the reference images in the batch dimension and pass them through another ResNet-18 to get one feature vector for each reference image. After that, we reshape this feature map so that we have one feature vector for each group of reference images and concatenate this with the feature map of the main image and the reference temperatures, obtaining B $(512 * (N\_REF + 1) + N\_REF)$-dimensional vectors that go through a fully connected layer, finally obtaining B (1)-dimensional vectors with the temperature prediction.

Figure 6.7 shows a diagram of this architecture.

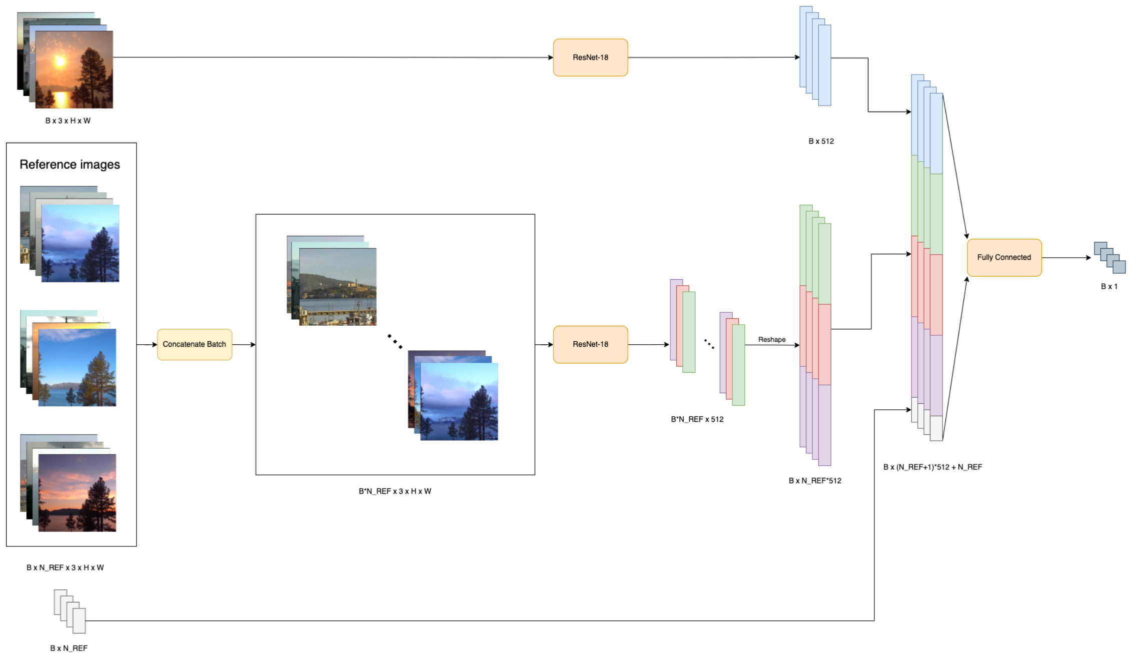


Figure 6.7: Architecture of two CNNs network. B is the batch size, N_REF is the number of reference images used, and H and W the image height and width respectively. In this diagram $B = 4$ and $N\_REF = 3$

6.4 Loss

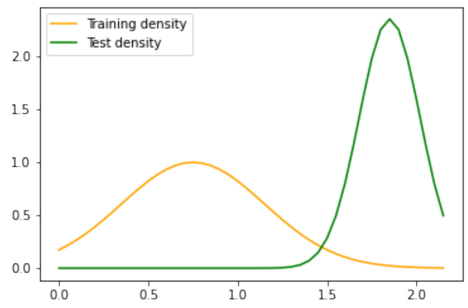
As explained in 6.1 there are multiple metrics that can be used as the loss function in the case of a regression task. In our case, we will use MSE as the loss function, but we will use some modifications using weights to try to obtain better results.

The problem we are trying to solve is the fact that the distribution of the input data to the model is different between the training data and the test data, as they come from different webcams.

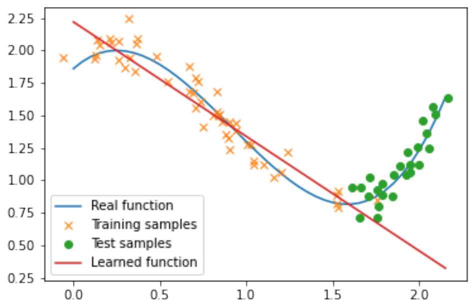
This problem is called **Dataset Shift**, and it occurs when training and test joint distributions are different. That is when $P_{training}(y, x) \neq P_{test}(y, x)$. There are multiples types of dataset shift but we will focus on our case, which is **Covariate shift**.

Covariate shift is the case when the distribution of x changes but the rest stays the same, formally it is defined as the case where $P_{training}(y|x) = P_{test}(y|x)$ and $P_{training}(x) \neq P_{test}(x)$.

Figure 6.8 shows an example of Covariate shift, in Figure 6.8a we can see how $P_{training}(x) \neq P_{test}(x)$, both are normal distributions but their mean and standard deviation are different and in Figure 6.8b we can see how a model learns a function using the training samples that is completely from the real function $P_{training}(y|x) = P_{test}(y|x)$.



(a) Density functions of x for the training and testing samples



(b) Plot with training and testing samples, the real function we want to predict and the learned function using only the training samples

Figure 6.8: Example of covariate shift

In our case, when we use different cameras for training and testing we have a case of Covariate Shift, as the distribution of the input images change between training and testing but the distribution of the temperature does not change as seen in Figure 6.3.

To try to solve this problem, we will weight each sample of the training dataset differently on the loss function according to how "similar" it is to the images on the test dataset. The idea behind this is to give more importance to the images that are in some way close to the testing images. To measure that "similarity" we will train a binary classifier to classify the input images into the training or the testing dataset. After that we will use the outputs of this classifier on each image of the training dataset to weight them on the loss function for the regressor.

We will use a ResNet-18 architecture and modify the fully connected layer to just output one neuron because in binary classification we only need one output neuron to be able to classify, and we will train it using the Binary Cross-Entropy Loss.

As the output of our classifier will be the raw logits and their range is $\mathbb{R}$, we have to normalize them to be able to use them in our loss function, we will use the following loss function:

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{\sum_{i=1}^{N} w_i(\hat{y}_i - y_i)^2}{N} = \frac{\sum_{i=1}^{N} e^{f(classifier(x_i))}(\hat{y}_i - y_i)^2}{N} \tag{21}$$

Where $classifier(x_i)$ is the raw output (logits) of the classifier neural network for image x_i and f is a function that maps $\mathbb{R}$ to $[0, 1]$ or $[-1, 1]$. We will use the following different functions as f:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad sigmoid(x) = \frac{1}{1 + e^{-x}}, \quad scaled\_sigmoid = \frac{2}{1 + e^{-x}} - 1 \tag{22}$$

Figure 6.9 shows the graph of these three functions.

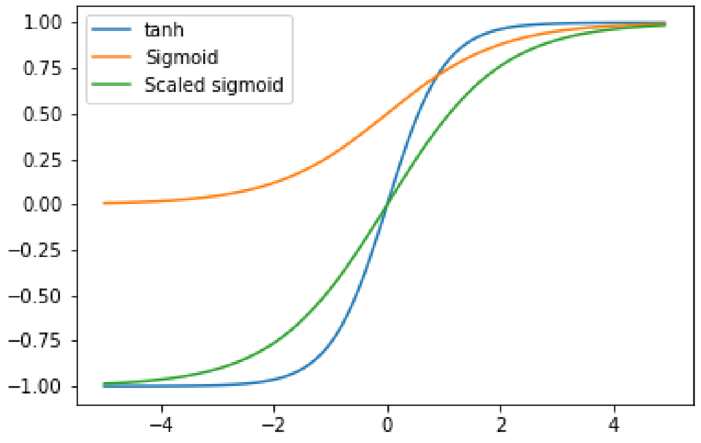


Figure 6.9: Graph of the used scaling function for the weights: $tanh$, $sigmoid$ and $scaled\_sigmoid$

7 Results

In this section of the thesis we will expose the obtained results.

7.1 Weather classifier

For the weather classifier we used 65% of the dataset to train the model, 20% as the validation dataset and the remaining 15% as a test dataset that we will only evaluate with the best model.

In the following Table 1 we can see the best results obtained by each architecture in the weather classifier task. In the case of ResNet-18 and ResNet-50 we trained different convolutional layers of the architecture in addition to the last fully connected layer, and in the case of ResNet-18 the best results were achieved when we fine-tuned the entire network initialized with ImageNet weights, but in the case of ResNet-50, the best results were obtained when we retrained only the last convolutional block named *conv5_x* in Figure 5.1. That is probably because when we trained the complete model in ResNet-50 it had 24M parameters in comparison to the 11M that full ResNet-18 model had, in the case we only retrain *conv5_x* of ResNet-50, it had 15M parameters.

As explained in 5.3, we used two different architectures for using the CLIP pretrained weights, the first one is to use the visual encoder and the text encoder and maximize the cosine similarity between the valid image-text pairs as how CLIP was trained. We also tested retraining only some layers of the model and the best model results were obtained when we retrained the whole visual encoder but the text encoder was frozen. In the second case we used only the visual encoder and added a fully connected layer at the end of the model so it can be used as a classifier, the same way as we did with ResNet-18 and ResNet-50. In this case, the best results were achieved when we retrained the last convolutional block named *conv5_x* and the attention pooling mechanism.

In the case of weighting strategies we can see that weighting the samples just by class instead of class and camera works better in all architectures.

Metric / Model	Weights	Acc.	Macro F1	Weighted F1	Clear F1	Cloudy F1	Rain F1	Snow F1
ResNet-18	Class	0.8183	0.7258	0.8266	0.8971	0.7062	0.598	0.702
ResNet-50	Class	**0.8618**	**0.7877**	**0.8653**	**0.9208**	**0.7676**	**0.7014**	**0.761**
CLIP-RN50 (1)	Class	0.8266	0.7247	0.8343	0.9027	0.7234	0.6292	0.6436
CLIP-RN50 (2)	Class	0.8018	0.6981	0.811	0.8871	0.683	0.5723	0.6502
ResNet-18	Class & Cam	0.8007	0.7053	0.8078	0.8865	0.6646	0.5706	0.6994
ResNet-50	Class & Cam	0.8535	0.7735	0.8566	0.9179	0.7465	0.6803	0.7495
CLIP-RN50 (1)	Class & Cam	0.8221	0.712	0.8236	0.9083	0.6695	0.5773	0.6928
CLIP-RN50 (2)	Class & Cam	0.7724	0.651	0.7802	0.8768	0.6066	0.4907	0.6298

Table 1: Validation dataset metrics for the different net configurations

In Table 1 it can be seen that the best results in the validation dataset are obtained with the ResNet-50 architecture and weighting the samples only by class instead of by class and camera. In the following Table 2 we can see the results of the best model in the test dataset.

Metric / Model	Weights	Acc.	Macro F1	Weighted F1	Clear F1	Cloudy F1	Rain F1	Snow F1
ResNet-50	Class	0.8591	0.7744	0.863	0.921	0.7635	0.6845	0.7285

Table 2: Test dataset metrics for the best model

Also, Figure 7.1 shows the confusion matrix for the test dataset, it is normalized by rows (the ground truth class), so the numbers on the diagonal indicate the recall of each class. It shows that in the case of a *clear* image, it sometimes confuses it with a *cloudy* image, this is probably because the labels are generated automatically and, as explained in Section 4.1, they can be wrong for some different reasons, and one of the labels that can have a little confusion is the level of cloud cover.

Also, in the case of *rain* and *snow* images it sometimes classifies them as *cloudy*, this is probably because in some cameras the quality does not allow the image to see rain or snow falling and, as the sky is probably cloudy, it classifies the image as *cloudy*.

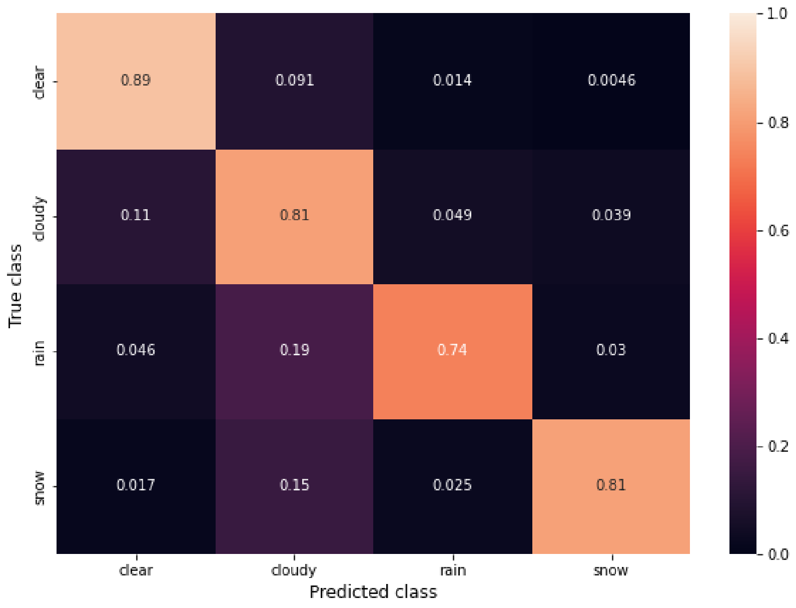


Figure 7.1: Test dataset confusion matrix for the best ResNet-50 model

7.2 Temperature regressor

As explained in 6.2, first we will train some networks to prove that it is actually possible to predict temperature from images using images from all the cameras to train the CNN and test it. Table 3 shows the results on the validation set, we used 75% of the data for training and the rest for validation.

	R^2	RMSE (°K)	MAE (°K)
ResNet-18	0.8488	4.153	2.973
ResNet-50	0.8262	4.452	3.286

Table 3: Temperature prediction results (R^2, RMSE, and MAE) when using all cameras for training and testing

We can see that the RMSE error is a little bit more than 4°K which is an improvement from the the previous works [2, 9, 34], these results prove that predicting the temperature from the images in our dataset is possible.

As explained in 6.4, to try to solve the *Covariate Shift* problem in our dataset, we trained a binary classifier to distinguish images from the training dataset and images from the test dataset to weight differently each image in the loss function. In the following Tables 4 and 5 we can see the confusion matrix and the metrics of the camera classifier that we will use to weight differently each sample for the training when using different cameras for training and evaluating. We can see that it is a really good classifier as it is not a difficult task, each camera is static, and although the images can be zoomed in and cropped at different sizes, which can make the scene change, it is an easy task for a CNN.

		Predicted class	
		Evaluating camera	Training camera
True class	Evaluating camera	13607	223
	Training camera	296	54624

Table 4: Confusion matrix for the validation dataset for camera classifier

		Training cameras (0 class)			Evaluating cameras (1 class)		
Metric / Model	Accuracy	Recall	Precision	F1	Recall	Precision	F1
ResNet-18	0.9925	0.9946	0.9959	0.9953	0.9839	0.9787	0.9813

Table 5: Validation dataset metrics for the camera classifier

In the following Table 6 we can see the results when we use different cameras for training and evaluating with the different proposed architectures, different number of reference images, and all the weight functions explained in 6.4.

Architecture	NUM_REF	Weights function	R^2	RMSE (°K)	MAE (°K)
ResNet-18	0	None	0.0413	8.652	6.963
ResNet-50	0	None	0.0706	8.519	6.76
Batch Concatenating	3	None	0.3717	7.004	5.458
Channel Concatenating	3	None	0.2706	7.546	5.923
Two CNNs	3	None	0.3513	7.117	5.553
Batch Concatenating	5	None	0.3679	7.025	5.478
Channel Concatenating	5	None	0.2179	7.814	6.083
Two CNNs	5	None	0.3903	6.899	5.398
Batch Concatenating	3	Tanh	**0.4788**	**6.379**	**4.956**

Table 6 continued from previous page

Architecture	NUM_REF	Weights function	R^2	RMSE (°K)	MAE (°K)
Batch Concatenating	3	Sigmoid	0.3545	7.099	5.528
Batch Concatenating	3	Scaled Sigmoid	0.4084	6.796	5.329
Channel Concatenating	3	Tanh	0.4094	6.79	5.209
Channel Concatenating	3	Sigmoid	0.2842	7.476	5.842
Channel Concatenating	3	Scaled Sigmoid	0.3469	7.141	5.566
Two CNNs	3	Tanh	0.3849	6.93	5.333
Two CNNs	3	Sigmoid	0.3407	7.175	5.629
Two CNNs	3	Scaled Sigmoid	0.3646	7.043	5.593
Batch Concatenating	5	Tanh	0.4366	6.632	5.209
Batch Concatenating	5	Sigmoid	0.4608	6.488	5.036
Batch Concatenating	5	Scaled Sigmoid	0.4152	6.757	5.187
Channel Concatenating	5	Tanh	0.3691	7.018	5.453
Channel Concatenating	5	Sigmoid	0.3478	7.136	5.509
Channel Concatenating	5	Scaled Sigmoid	0.3572	7.084	5.526
Two CNNs	5	Tanh	0.4522	6.54	5.105
Two CNNs	5	Sigmoid	0.403	6.827	5.381
Two CNNs	5	Scaled Sigmoid	0.4029	6.828	5.311

Table 6: Temperature prediction results (R^2, RMSE and MAE) when using different cameras for training and testing

The first thing that can be clearly seen is that the results are definitely not as good as when we used all cameras for training and evaluating as shown in Table 3 as here the RMSE is around 7°K in comparison to the nearly 4°K achieved when using the same cameras for training and evaluation.

Another thing shown in the table is that the reference images clearly help the CNN to predict the temperature as if we use a regular ResNet with one input image the error is much higher than when we add reference images to the input. From the three proposed architectures, Batch Conatenating, Channel Conatenating and Two CNNs we can see that Channel Concatenating is the one that performs the worst and Batch Concatenating is usually the one that performs the best. Also, it looks that with 3 reference images the results are better than with 5 reference images, except in the case of Two CNNs architecture, one of the reasons of this may be because the dimensionality of the feature vector with 5 references may be too high, but more experiments with different number of references should be done to confirm this hypothesis.

Also in the case of weight functions, we can clearly see that $tanh$ usually outperforms the other two and also the base case where all the images are equally weighted except in the Batch Concatenating architecture with 5 references.

In the following Table 7 we can see the different metrics (R^2, RMSE and MAE) of the best model (Batch Concatenating with 3 Reference images and $tanh$ as the weight function) on the validation dataset for each camera. In the next Table 8 the mean and standard deviation of each validation camera is shown.

	R^2	RMSE (°K)	MAE (°K)
ALL CAMERAS	0.4788	6.379	4.956
campkazuma	-1.1281	7.1287	5.7551
fortlauderdale	-0.1268	5.278	4.2063
hydenkentucky	0.3399	8.1281	6.6968
naples	0.041	4.462	3.4506
sanfrancisco	-0.9715	6.1128	4.9587
statueofliberty	0.4077	7.8338	6.1381
statueofliberty_hd	0.4231	7.9296	6.3153

Table 7: Temperature prediction results (R^2, RMSE, and MAE) on the different testing cameras for the best model

	mean	std
ALL CAMERAS	294.848	8.836
sanfrancisco	288.065	4.354
hydenkentucky	290.886	10.006
statueofliberty	288.513	10.179
fortlauderdale	297.154	4.972
campkazuma	302.890	4.887
naples	299.445	4.557
statueofliberty_hd	288.021	10.442

Table 8: Mean and standard deviation of the validation cameras

We can see that the cameras of which the temperature distribution is more similar to the training temperature distribution (as seen in Figure 6.3), which are *statueofliberty*, *statueofliberty_hd* and *hydenkentucky*, are the ones with higher R^2 score although their RMSE error may be higher. This is because their variance is also higher, as shown in Table 8, and although the RMSE error is higher, if we recall the formula in Equation 20, it divides by the variance. In the case of *campkazuma* and *sanfrancisco*, the R^2 is much lower than in the other cases, around -1, that is because their standard deviation is much lower, around 4.5 in both cases, in comparison to the standard deviation of 10 in other cases.

Figure 7.2 shows 5 examples of each testing camera, its real temperature, and the prediction made by the best model. We can see that the error of each of the samples in each camera is quite similar to the errors reported above so they are good representative of the errors of each camera.

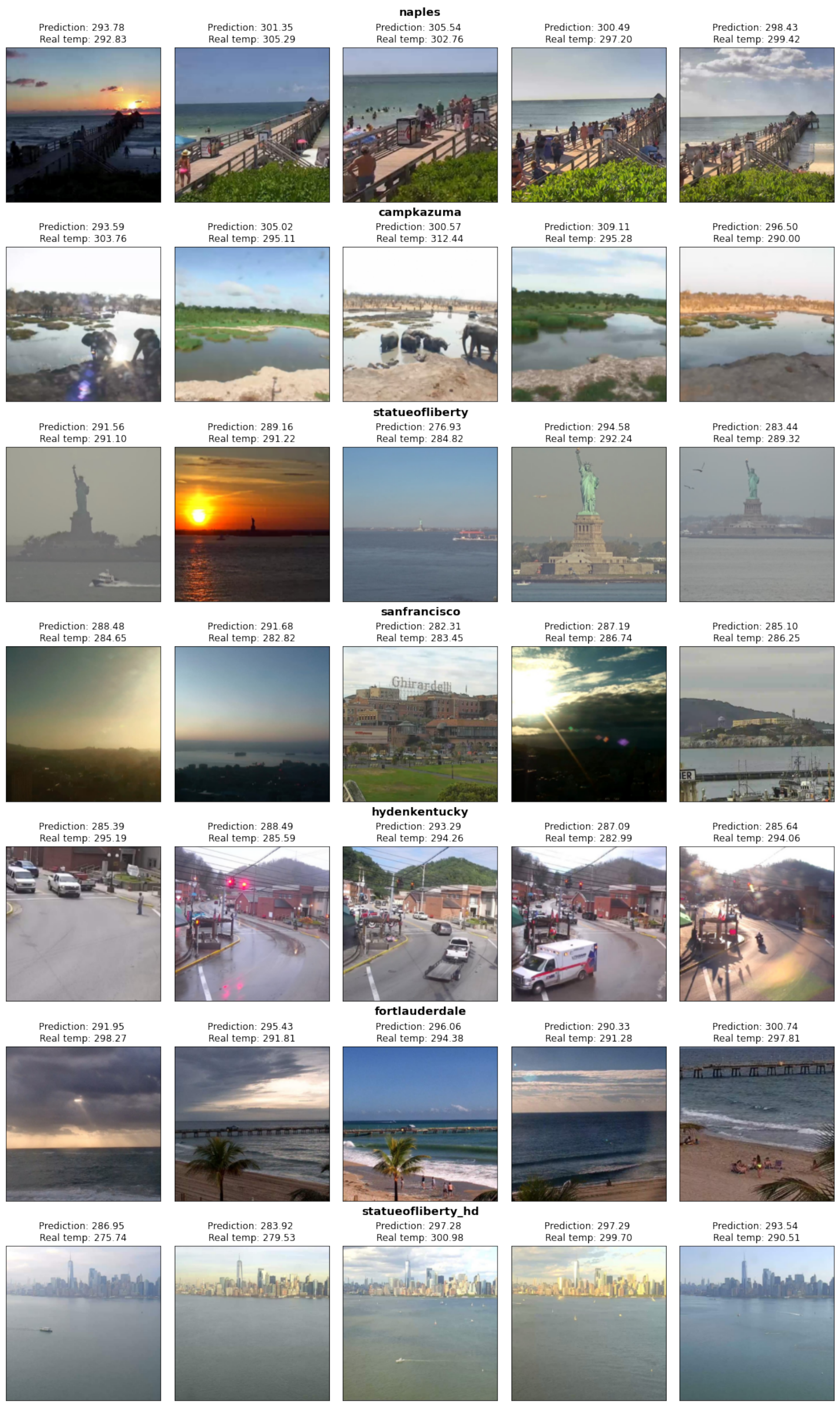


Figure 7.2: Temperature prediction results of the best model for some examples of each testing camera

8 Conclusions

In this work we have created a new dataset in a fully automated way without the need of human help to label the images and have proven that it is an accurate dataset with which deep learning models can work and are tested to be effective. This method of creating a dataset could help to create much larger datasets and extend the existing ones, as with the evolution of deep learning then huge datasets are needed to train the bigger models. In our case we have labeled more than 275000, but more images could easily be downloaded and labeled in case our model needed more data.

After that we have used the images to train a model to be able to classify the main weather condition achieving a weighted F1 score of 0.863.

Following that, we have worked on a more difficult task, predicting the ambient temperature of an image, it is more difficult as the cues that can help a model predict the temperature are usually more subtle than the ones that indicate the main weather condition, which are more explicit in the image. As explained in Section 6, all previous works done in predicting temperature use images from same cameras for training and evaluating, in our case we proposed three different architectures so that our model can predict images from any camera, even if it has not been trained on images from that place.

We proposed three different architectures: Batch Concatenating, Channel Concatenating and Two CNNs using References Images so the model can use this images for being able to know how the different temperatures look like in that specific location. The results show that concatenating the images in the Batch dimension is the architecture that achieves the best results.

We also suggested a method for trying to solve what is called Covariate Shift, training an auxiliary model to detect which training images are more 'similar' to the testing images. This model is used to weight differently each sample in the training and the results show that it helps the model to perform better.

Future Work

Although the main goals of this project have been achieved, there are always new ideas that can improve the results obtained in this thesis.

One of the first tasks to do would probably be to extend the used dataset with more cameras and new images to be able to use more images to train the network and evaluate the temperature regressor in more different cameras. This could be done by using the AMOS dataset or finding more datasets where they have screenshots of outdoor webcams.

Another future work would be to try more different architectures changing the way that reference images are used and try using different numbers of references images and more hyperparameters if the computing resources allow it.

References

[1] Jason Brownlee. *How to visualize filters and feature maps in Convolutional Neural Networks*. July 2019. URL: https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/.

[2] Wei-Ta Chu, Kai-Chia Ho, and Ali Borji. "Visual Weather Temperature Prediction". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2018, pp. 234–241. DOI: 10.1109/WACV.2018.00032.

[3] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=YicbFdNTTy.

[4] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html.

[5] IBM Cloud Education. *What are neural networks?* Aug. 2020. URL: https://www.ibm.com/cloud/learn/neural-networks.

[6] Mohamed Elhoseiny, Sheng Huang, and Ahmed Elgammal. "Weather classification with deep convolutional neural networks". In: *2015 IEEE International Conference on Image Processing (ICIP)*. 2015, pp. 3349–3353. DOI: 10.1109/ICIP.2015.7351424.

[7] Vittorio Ferrari and Andrew Zisserman. "Learning Visual Attributes". In: *Advances in Neural Information Processing Systems*. Ed. by J. Platt et al. Vol. 20. Curran Associates, Inc., 2007. URL: https://proceedings.neurips.cc/paper/2007/file/ed265bc903a5a097f61d3ec064d96d2e-Paper.pdf.

[8] Kunihiko Fukushima. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36 (1980), pp. 193–202.

[9] Daniel Glasner et al. "Hot or Not: Exploring Correlations between Appearance and Temperature". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 3997–4005. DOI: 10.1109/ICCV.2015.455.

[10] James Hays and Alexei A. Efros. "IM2GPS: estimating geographic information from a single image". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587784.

[11] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[12] Tong He et al. "Bag of Tricks for Image Classification with Convolutional Neural Networks". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 558–567. DOI: 10.1109/CVPR.2019.00065.

[13] Mohamed R. Ibrahim, James Haworth, and Tao Cheng. "WeatherNet: Recognising Weather and Visual Conditions from Street-Level Images Using Deep Residual Learning". In: *ISPRS International Journal of Geo-Information* 8.12 (2019). ISSN: 2220-9964. DOI: 10.3390/ijgi8120549. URL: https://www.mdpi.com/2220-9964/8/12/549.

[14] Nathan Jacobs, Nathaniel Roman, and Robert Pless. "Consistent Temporal Variations in Many Outdoor Scenes". In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. 2007, pp. 1–6. DOI: 10.1109/CVPR.2007.383258.

[15] Nathan Jacobs et al. "The Global Network of Outdoor Webcams: Properties and Applications". In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '09. Seattle, Washington: Association for Computing Machinery, 2009, pp. 111–120. ISBN: 9781605586496. DOI: 10.1145/1653771.1653789. URL: https://doi.org/10.1145/1653771.1653789.

[16] Dinesh Jayaraman, Fei Sha, and Kristen Grauman. "Decorrelating Semantic Visual Attributes by Resisting the Urge to Share". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1629–1636. DOI: 10.1109/CVPR.2014.211.

[17] Bhavika Kanani. *Activation functions in neural network*. Oct. 2019. URL: https://studymachinelearning.com/activation-functions-in-neural-network/.

[18] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by

Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[20] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[21] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[22] Xuelong Li, Zhigang Wang, and Xiaoqiang Lu. "A Multi-Task Framework for Weather Recognition". In: *Proceedings of the 25th ACM International Conference on Multimedia*. MM '17. Mountain View, California, USA: Association for Computing Machinery, 2017, pp. 1318–1326. ISBN: 9781450349062. DOI: 10.1145/3123266.3123382. URL: https://doi.org/10.1145/3123266.3123382.

[23] Cewu Lu et al. "Two-Class Weather Classification". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 3718–3725. DOI: 10.1109/CVPR.2014.475.

[24] Cewu Lu et al. "Two-Class Weather Classification". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12 (2017), pp. 2510–2524. DOI: 10.1109/TPAMI.2016.2640295.

[25] Charlotte Pelletier, Geoffrey Webb, and François Petitjean. "Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series". In: *Remote Sensing* 11 (Mar. 2019), p. 523. DOI: 10.3390/rs11050523.

[26] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. DOI: 10.48550/ARXIV.2103.00020. URL: https://arxiv.org/abs/2103.00020.

[27] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

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

[29] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556.

[30] Christian Szegedy et al. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[31] Michael Tomz, Gary King, and Langche Zeng. "ReLogit: Rare Events Logistic Regression". In: *Journal of Statistical Software* 8.2 (2003), pp. 1–27. DOI: 10.18637/jss.v008.i02. URL: https://www.jstatsoft.org/index.php/jss/article/view/v008i02.

[32] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[33] Armando Vieira. *The revolution of depth*. June 2016. URL: https://medium.com/@Lidinwise/the-revolution-of-depth-facf174924f5.

[34] Anna Volokitin, Radu Timofte, and Luc Van Gool. "Deep Features or Not: Temperature and Time Prediction in Outdoor Scenes". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2016, pp. 1136–1144. DOI: 10.1109/CVPRW.2016.145.

[35] Joseph Yacim and Douw Boshoff. "Impact of Artificial Neural Networks Training Algorithms on Accurate Prediction of Property Values". In: *Journal of Real Estate Research* 40 (Nov. 2018), pp. 375–418. DOI: 10.1080/10835547.2018.12091505.

[36] Muhamad Yani, S Irawan, and Casi Setianingsih. "Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail". In: *Journal of Physics: Conference Series* 1201 (May 2019), p. 012052. DOI: 10.1088/1742-6596/1201/1/012052.

[37] Richard Zhang. *Making Convolutional Networks Shift-Invariant Again*. 2019. DOI: 10.48550/ARXIV.1904.11486. URL: https://arxiv.org/abs/1904.11486.