POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Hierarchical classification model for content-based geolocation of outdoor images with visual explanations

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Luca Loria

Advisor: Prof. Mark James Carman

Academic year: 2020-2021

1. Introduction

Human intervention on the planet has led to an evident increase of natural disasters such as hurricanes, floods, and landslides. With the advent of social media technologies, many people are posting photos and pictures related to natural disasters, often taken from unknown and untrusted sources.
By geolocating images, emergency services may coordinate vehicles and people in order to tackle natural disasters more promptly and effectively, saving people lives and belongings. This work aims to help them by exploiting geolocation features from images.
Modern deep learning techniques and theories have made this work possible thanks to the ability of performing content-based geolocation.
As a further refinement, as this work has to be intended as a support to the geolocating operations, it provides the prediction of a set of possible world areas in which the image could have been taken and a set of visual explanations, such as saliency and interactive world maps.

2. Related Works

The image geolocation problem has started becoming interesting since the late 2000, with the IM2GPS [2] work that started from a pure computer vision approach, and then it evolved into deep learning based techniques. In particular, two main directions were followed for solving this problem: the geolocation by classification (GBC) approach and the image based localization (IBL) approach.
The **geolocation by classification (GBC)** is intended to treat the geolocation problem as a classification task: they are based on the division of the world map into a set of cells and predicting the correct one. The pioneer works exploiting this technique were PlaNet [8] and the work of Müller-Budack et al. [4]. The latter also introduced the concept of hierarchical classification that was the idea of classifying an image on differently-grained world division and combine the predicted probabilities in order to extract the final confidence levels.
The **image-based localization (IBL)** is a retrieval-based image localization technique that is performed for estimating the geographical position of a query image by applying a similarity measure that provide the nearest reference images from a precomputed database. The most popular work exploiting the IBL is the Revisited IM2GPS [7], that combines the IM2GPS approach with modern deep learning techniques.
As far as this work is concerned, we decided to

focus on the GBC approach.

2.1. Cell Generation

Building an adaptive set of classes depending on the dataset is crucial in the task definition and in the model accuracy. With the term adaptive, we intend to divide the world up in finer-grain regions where the largest number of images are taken, while it makes sense to keep vast areas for those locations which do not have a sufficient number of images coming from them. In the literature [4, 8], we found the Google S2 Geometry library [1] which enables us to deal with grid building in an adaptive way. We took inspiration from the [4] GitHub repository and modified it in order to shape the results to the problem structure. In order to keep the cell as balanced as possible, we define two thresholds (t_{min} and t_{max}) which are used in the cell generation algorithm for limiting the number of images per cell.

2.2. Visual Explanations

The visual explanations are tools used for refining the model predictions, in order to provide a further insight to the users. In particular, we took inspiration from the Grad-CAM [5] model which uses the gradients of any target concept flowing into the final convolutional layer to produce a saliency map, highlighting the most important pixels in the image for predicting the output class through a saliency map.

3. Research Questions

In order to set a path for the current research work, we define a list of questions that we address in this thesis. In particular:

1. Is it possible to provide a deep learning model which is able to geolocate any kind of outdoor image from a worldwide perspective?
2. Is it possible to improve the performance of an image geolocating model by adding the hierarchical information of how the world was divided up directly inside the model?
3. Can a deep learning model provide a significant visual explanation that could be used to understand why the image has been predicted in such?
4. What is the best way to predict the location information? How can we evaluate that the model is predicting reasonable locations?

4. Approach and Model architectures

When dealing with the outdoor geolocation problem, the first operation we perform is a set of data augmentation techniques in order to exploit more information from the same training data. Immediately after, we place as input of the feature extractor a batch of 32 RGB resized images (224,224,3). The output of this block of the general scheme produces a volume of extracted image features shaped as (1280,1280,8). As far as the Cell Classifier is concerned, we decide to implement a baseline Fully-Connected multi-class model and a very powerful set of custom neural networks. The latter embeds the hierarchical information concerning the grid generation in order to refine the predictions of leaf cells.

4.1. Hierarchical Models

The hierarchical model constitutes the core of the novelties developed in this thesis work. After creating a baseline fully-connected model for classifying the encoded features coming from the Feature Extractor, we want to find out whether the model could perform even better by exploiting the information coming from the hierarchical cell generation. Our work aims to make use of the grid generation knowledge, especially how cells are generated from bigger cells and how they are linked together: the hierarchical information is exploited directly inside the model structure.

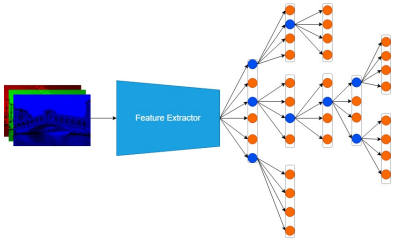

Figure 1: Visual representation of the model hierarchy. From the left to right, the input image divided in the RGB channel, the Feature extractor and the hierarchical cell representation in terms of neurons. Blue neurons represent intermediate cells, while orange neurons represent leaf cells.

As it is possible to deduce from Fig. 1, the aim of our work is to create a hierarchical relationship between cells of different size: this leads the procedure to build a tree-like structure in which deep cells are fractions of cells from upper levels. In order to keep the hierarchical dependency concept that we explained previously during the models' implementation, we transformed the tree structure into a destination-oriented directed acyclic graph (DODAG), expressed through a set of connection matrices that are used to express connections between neurons in a fully-connected set of layers, which represent the levels of the hierarchy.
The output encoding of the Feature Extraction is firstly linearized into a vector-shaped tensor and then fed into a Dense layer activated by a ReLU function, with the purpose of properly combining the features. Immediately after it, the hierarchical classification module takes place. As a first operation, the Hierarchical Prediction layers are defined: they consist of a set of Dense layers in a number that is equivalent to the number of hierarchy levels (depth). Their main purpose is to predict the target cell of the current (batch of) image at their corresponding level of the hierarchy.
As the output of the Hierarchical Prediction layers are identified by tensors of values, it is possible to combine them using matrix operations. According to this, we decided to multiply each output of the previous layer by the corresponding level's **binary_matrix**.
After the matrix multiplication, the output tensor will be shaped as the output of the model, that is a set of K_{max_depth} tensors with equal size (number of leaf cells).
The final operation that needs to be performed before the model prediction is the combination of the latter tensors in order to result in a single final output layer. In these terms, we define several variants that produce a set of different hierarchical models:
Product of the branch nodes' probabilities (**HMC**): computes the element-wise product between all the hierarchical prediction tensors.

$$Tensor_{out} = \prod_{i=lvl_0}^{max\_depth-1} Tensor_{lvl_i} \quad (1)$$

All the elements of the same hierarchical tree branch are multiplied together in order to discriminate regions depending on their hierarchy.
Exponential sum of logarithms of the nodes' probabilities (**HMC-sol**): each of the Hierarchical Prediction Tensors are passed through a log function and then element-wise summed up together. Finally, the output tensor is the exponential of that sum.

$$Tensor_{out} = e^{\sum_{i=0}^{max\_depth-1} log(Tensor_{lvl_i})} \quad (2)$$

This implementation is intended to solve a precision issue caused by long multiplication chains of decimal numbers (floating-point numbers which are ≤ 1).
Smoothed sum of the branch nodes' probabilities (**HMC-smooth**): smooths each of the Hierarchical Prediction Tensor by a power of α which depends on the depth of the tensor and compute the average of each branch, as expressed in the next formula.

$$\forall k \in \{0, \dots, leaf\_size\} : Tensor_{out_k} = \frac{\sum_{i=0}^{max\_depth-1} Tensor_{lvl_i,k} \cdot \alpha^{max\_depth-i-1}}{\sum_{j=0}^{bottom\_lvl-1} \alpha_k^{bottom\_lvl-j-1}} \quad (3)$$

This implementation gives more importance to the probabilities at deep levels of the hierarchy.

5. Dataset

In the literature [3], it is possible to find only a few datasets of geotagged pictures, mainly circumscribed to a specific area or a coerced task. As the task requires us to cover a very broad and heterogeneous set of images, we decided to crawl ourselves a dataset on the Flickr platform: we collected a vast set of outdoor heterogeneous places by exploiting the search function offered by the Flickr API. We used terms related to urban ('city', 'building', 'street', etc.) and natural ('landscape', 'mountain', 'woods', etc.) scenes as well as words for specific areas of the world ('pyramid', 'savanna', 'tundra', etc.). In order to obtain the largest possible set of outdoor images, we also exploited the PyDictionary and GoSlate library in order to generate respectively any possible English synonym and translations in a set of given languages (Italian,

French, Spanish, Portuguese, German, Chinese and Swedish). In the end we were able to collect around one million geotagged outdoor images containing outdoor scenes of any possible kind.

6. Experimental Results and Evaluations

In this section, we discuss the experiments and evaluations that were performed on the deep learning models that we explained in the previous sections. In particular, we discuss how we analyze the outdoor geolocation problem and compare the model performance in different conditions.

6.1. Experimental Setup

All the experiments we run share common settings, and we are going to illustrate them in this section. As far as the dataset is concerned, all the outdoor geolocation models that we will be dealing with are trained over the Flickr images that we personally crawled as thoroughly explained in Chapter 5. In particular, we automatically split the one million image dataset into two separated set: training (90%) and validation (10%). The test set is extracted offline. During the model training, we feed the neural network with batches of 32 random images. The models optimize a SparseCategoricalCrossEntropy loss function, exploiting the Adam optimizer endowed with several regularization functions. In order to compare different models, we use different evaluation metrics, that are the validation loss and accuracy and the Improvement over Random (IoR) which is able to measure how many times the model is performing better than the random prediction, as shown in the next formula:

$$IoR = \frac{Accuracy_{valid}}{Prob_{Random}} \tag{4}$$

The way the random classifier probability is computed is a uniform probability distribution over the target cells of the outdoor geolocation problem ($Prob_{Random} = \frac{1}{N_{classes}}$). This metric is more significant than accuracy when the number of classes increases, as accuracy values are shrunk to zero while the IoR is expressed in reasonable values.

6.2. Experiments

In order to evaluate the best models and compare different implementations, we run several experiments that allow us to understand the best settings for solving the outdoor geolocation problem.

Experiment A - Model backbone: which state-of-the-art CNN model could be the best feature extractor for the Outdoor Geolocation problem. The experiment runs on modern CNN models (VGG19, ResNET152, EfficientNetB0 and EfficientNetB1) over the same classification task. The results show that the EfficientNet backbone outperforms all its competitors. In particular, the Keras EfficientNetB1 [6] implementation showed by far the best performance among the tested networks.

Experiment B - Number of Images: trend of the Outdoor Geolocation Model performance when increasing the number of images in the training dataset. In order to have a sufficient number of samples for analyzing the performance trend, we choose to extract different numbers of training image (10,000 - 40,000 - 160,000 - 640,000) for the current experiment. By plotting validation accuracy (and also IoR) versus dataset size with logarithmic access shows a steep slope except from a plateau in the middle segment. This behavior demonstrates that the accuracy will not converge even after 640,000 images. For this reason, we can assume that increasing the number of images will result in an exponential increase up to a point until we see decay after the base error rate in the validation accuracy.

Experiment C - Number of Target Cells: the effect that the number of target classes might have on the Outdoor Geolocation problem. The results show that when the number of cells is low, the model seem to have a low IoR probably due to underfitting: the model is underestimating the problem. On the other hand, when the number of cells is high enough (over 50) the model's IoR starts dropping and the reason for this can be found in the overfitting problem, as there are not enough images for enabling the model to be trained over that number of classes.

Experiment D - Hierarchical Model: compare the hierarchical models' performance against a baseline non-hierarchical multi-class model in terms of validation performance and

training time.

Model name	Loss	Accuracy	IoR
MC	4.799	0.023	1.66
HMC	4.301	0.033	2.38
HMC-sol	4.104	0.032	2.23
HMC-smooth	3.980	0.031	2.23

Table 1: Models validation performance (loss and accuracy) and IoR (Improvement over Random).

Model name	Time per epoch (s)
MC	9,256
HMC	10,253
HMC-sol	15,960
HMC-smooth	19,253

Table 2: Models' time per Epoch expressed as the average time that we measured throughout the training.

As far as the validation performance is concerned (Tab. 1), it is easy to denote how hierarchical models outclass the non-hierarchical one. However, this results must be correlated to the time it takes to train each of the previously enlisted models: the non-hierarchical model is faster than its hierarchical counterparts, but there is an exception: the HMC is only 1.09x slower than the MC. These results demonstrate that the HMC is better than any other analyzed model, due to its training speed (very close to the multi-class model) and validation accuracy.
Analysis A - Model Calibration: correctness and calibration of the HMC model prediction through traditional machine learning techniques.

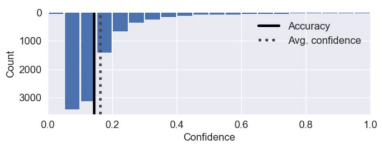

Figure 2: The calibration histogram plot for the predicted confidence. The distance between the two vertical lines represent how well calibrated the model is.

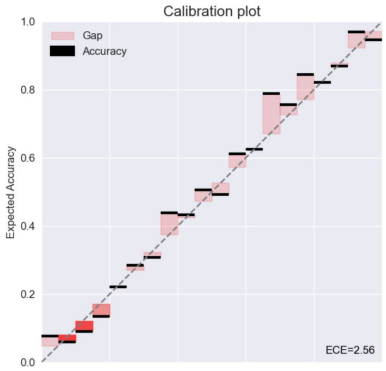

Figure 3: The calibration plot for the outdoor geolocation model. Black lines on top of bins represent the accuracy for that specific bin. The more they are aligned with the bisector, the better the calibration is.

From the first two plots, it is possible to deduce that the model is well calibrated. In fact bins of Fig. 3 are not distant from the bisector of the Cartesian plane, while the Fig. 2 vertical lines are almost aligned. This leads us to think that the model uncertainty is reasonable, as the predicted probabilities are very close to the expected probabilities of the same outcomes.
Analysis B - Visual Explanations: the HMC model explainability features in generating visual explanations, in terms of saliency and visual world maps. With this analysis we want to provide a better understanding of the model predictions: in this way, users are not only able to understand exactly where the model predicted the location of the image using a visual tool (**Visual World Map**), but also which parts of the picture were relevant for the prediction (**Grad-CAM**).

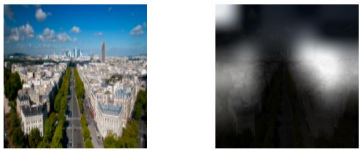

Figure 4: Example of a Grad-CAM superimposition. On the left side, the original image. On the right side, the Saliency map of the left image.

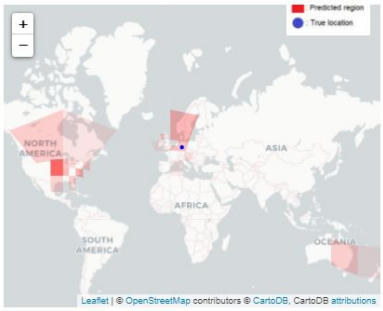

Figure 5: Example of a visual interactive world map.

7. Demonstration Website with Visual Explanations

The main purpose of our demonstration website is to test models on user-uploaded images or randomly chosen images from our dataset. The whole application has been developed using the Flask framework and embeds the HMC and MC models described in the previous chapters. The two models are responsible for producing the confidence levels for the cells of the world grid: they do not only tell the best performance, but produce a specific confidence value for each of the cells. In this way, it is possible to produce interactive world maps and compare their performance. In the meantime, **Grad-CAM** is used for computing the visual explanation (heatmap) of the geolocation models. The combination of visual features from heatmaps and prediction maps that show which grid cells were closer to the target, enables users to have a better understanding of how the task works and how close the model came to predicting the correct location.

8. Conclusions and Future Research Directions

Through this thesis work, we provided answers to a list of research questions that we posed in the Section 3. In particular, we tried to deal with the outdoor geolocation problem exploiting a set of deep learning techniques that could solve the problem without any additional information beyond the image's pixels. We are able to say that we created a hierarchical deep learning model that is able to correctly geolocating outdoor images with higher performance than its multi-class counterpart (HMC has demonstrated to be the best one). Through calibration, we were able to demonstrate the correctness of the predictions. Finally, we provided a set of visual explanation tools by exploiting Grad-CAM and an interactive world map that refine predictions by providing further insights on the model behavior.

References

[1] Google. Google s2 geometry: spherical geometry library for manipulating geographic data. `https://github.com/google/s2geometry`, 2021.

[2] James Hays and Alexei A. Efros. im2gps: estimating geographic information from a single image. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[3] Carlo Masone and Barbara Caputo. A survey on deep visual place recognition. *IEEE Access*, 9:19516–19547, 2021.

[4] Eric Müller-Budack, Kader Pustu-Iren, and Ralph Ewerth. Geolocation estimation of photos using a hierarchical model and scene classification. In *ECCV (12)*, volume 11216 of *Lecture Notes in Computer Science*, pages 575–592. Springer, September 2018.

[5] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, pages 618–626. IEEE Computer Society, 2017.

[6] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[7] Nam N. Vo, Nathan Jacobs, and James Hays. Revisiting IM2GPS in the deep learning era. *CoRR*, abs/1705.04838, 2017.

[8] Tobias Weyand, Ilya Kostrikov, and James Philbin. Planet - photo geolocation with convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, 2016.

POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Hierarchical classification model for content-based geolocation of outdoor images with visual explanations

Tesi di Laurea Magistrale in
Computer Science and Engineering -
Ingegneria Informatica

Author: **Luca Loria**

Student ID: 944679
Advisor: Prof. Mark James Carman
Academic Year: 2020-21

Abstract

A common need shared across several fields of application is the demand for geolocating outdoor images in order to understand where they were taken. This could be the case of the interventions of emergency services: with the advent of social media technologies, many people are posting photos and pictures related to natural disasters, often taken from unknown and untrusted sources, that could lead rescuers to wrong locations. For this reason, we create a collection of artificial intelligence models which are able to find the correct location of an image. This is performed by splitting the world into a set of regions, and finding the one in which such a picture comes from.

In order to boost the accuracy of these self-learning models, we design new models for the hierarchical region classification task which directly encode the hierarchy of the regions into the neural architecture, specifically the feedforward layers of the CNN-based classification network. Cells are generated following a hierarchical process, and the models reflect the hierarchy by replicating the structure within the neural network. The use of such knowledge in the network structure results in better predictions both in terms of accuracy and calibration of the predicted confidence values. In order to prove the effectiveness of the hierarchical structure, we evaluate the models that we create using machine learning assessments that measure the quality and the calibration of the predictions. Finally, in order to provide users with a comprehensible prediction, we implement a set of visual explanations that are correlated to the model output. In particular, we provide a saliency map which evidence the pixels that were useful for the prediction and a world map which shows an overlay of the cells, together with their confidence levels. From a qualitative analysis, we are able to show how the saliency map is useful for highlighting areas of the image that are helpful for the prediction (buildings, trees, etc.) and how the world map can give hints on the true location of the image: if the model cannot predict the exact location, this visual world map can provide an overview of where the most likely regions are placed over the earth so that users can be helped in finding the correct location.

Keywords: outdoor geolocation, hierarchical classification, explainability, artificial intelligence, deep learning, calibration

Abstract in lingua italiana

Un'esigenza comune a più campi di applicazione è il bisogno di geolocalizzare le immagini outdoor per capire dove sono state scattate. Potrebbe essere il caso degli interventi dei servizi di emergenza: con l'avvento dei social media, molte persone pubblicano fotografie e immagini relative a disastri naturali, spesso prese da fonti sconosciute e non attendibili, che potrebbero portare i soccorritori a intervenire in luoghi sbagliati. Per questo motivo creiamo un set di modelli di intelligenza artificiale in grado di trovare la corretta posizione di un'immagine. Ciò viene eseguito suddividendo il mondo in un insieme di regioni e trovando quella da cui proviene tale immagine. Per aumentare l'accuratezza di questi modelli di autoapprendimento, creiamo una serie di implementazioni che tengano conto del suddetta distribuzione delle regioni, a partire dal loro processo di generazione: le aree sono generate gerarchicamente e i modelli si avvalgono di queste informazioni. Infatti, riflettono le informazioni gerarchiche replicandone la struttura all'interno della rete neurale. L'uso di informazioni gerarchiche nella struttura della rete porta a migliori previsioni sia in termini di accuratezza che di calibrazione dei valori di confidenza predetti. Al fine di dimostrare l'efficacia della struttura gerarchica, valutiamo i modelli che creiamo utilizzando alcuni metodi di apprendimento automatico che misurano la qualità e la calibrazione delle previsioni. Infine, per fornire agli utenti una previsione più comprensibile, implementiamo una serie di spiegazioni visive che sono correlate all'output del modello. In particolare, forniamo una mappa di salienza che evidenzia i pixel utili per la previsione e una mappa del mondo che mostra una sovrapposizione delle celle, insieme ai loro livelli di confidenza. Da un'analisi qualitativa, siamo in grado di mostrare come la mappa di salienza sia utile per evidenziare le aree dell'immagine utili per la previsione (edifici, alberi, ecc.) e come la mappa del mondo possa dare indicazioni sulla vera localizzazione dell'immagine: se il modello non può prevedere la posizione esatta, questa mappa può fornire una panoramica di come sono distribuite le regioni più probabili, in modo che gli utenti possano essere aiutati a trovare la coppia di coordinate corretta.

Parole chiave: geolocalizzazione all'aperto, classificazione gerarchica, spiegabilità (AI), intelligenza artificiale, deep learning, calibrazione (AI)

Contents

1 | Introduction

The introduction aims at illustrating the goal of the thesis and how a hierarchical approach can be useful to enhance the accuracy of an image's location prediction.

1.1. Goal

Human intervention on the planet has led to an evident increase of natural disasters such as hurricanes, floods, and landslides. Some causes of these unforeseen events have to be found in man-made buildings, infrastructures, and pollution processes. With the advent of social media technologies, many people are posting photos and pictures related to natural disasters, often taken from unknown and untrusted sources. Moreover, modern social media are used to remove the GPS-related information on images due to privacy issues. Here comes the need of emergency services to understand whether such images belong to the natural disaster that is occurring on the cited locations or it concerns other past events on different sites. By geolocating images, emergency services may coordinate vehicles and people in order to tackle natural disasters more promptly and effectively, saving people lives and belongings.

This thesis aims to help emergency services in tackling natural disasters by exploiting geolocation features from images. An application based on the modern AI theory can make use of features from images without any cues from the developers. In such a way, an ML model can predict the exact location and a set of similar locations from which the image could have been taken. Modern deep learning techniques and theories have made this work possible thanks to the ability of performing content-based geolocation. In particular, the DNN provides a layered structure that enables the geolocating task to be tackled at several hierarchical levels (Continent, Wide-regions, Countries, Regions ...). A relatively small number of works have been developed for such tasks: the majority tries to predict the exact location of an image without exploiting any hierarchical features or restricts the space search either geographically or to a constrained set of images. Some other works prefer to focus on a landmark-based approach, that are used for suggesting facilities and services to users whenever the system recognizes a certain marker [30]. This

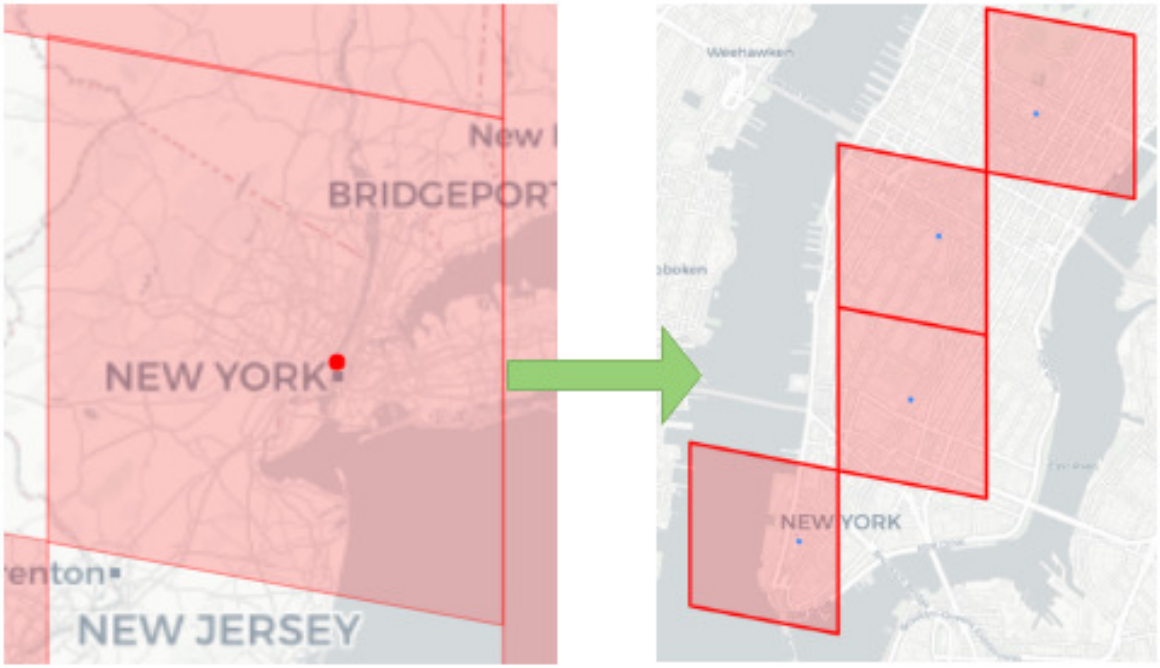

Figure 1.1: On the left side, a cell covering the whole city of New York because of a low number of images. On the right side, the only city of Manhattan divided up in many subregions thanks to a high number of images in that area.

approach is not suitable for the worldwide geolocating task for natural disasters, as they often happen to be in places which are not recognizable through well-known landmarks.

According to the Deep Learning theory, neural network-based classification approaches tend to outperform regression-based approaches, so trying to guess directly a specific GPS set of coordinates (latitude and longitude) is an incredibly difficult task. For this reason, this work aims to instead divide the world up recursively in a set of cells that cover the most densely populated areas of the planet and predict in which of these cells the image under analysis was taken. Rather than a specific set of coordinates, providing a region in which the picture was shot produces intrinsically an error, but as long as regions are small, the prediction is still accurate.

1.2. Hierarchical approach

In order to comply with the task of geolocating an image on a specific part of the planet, this work is able to divide up a specific section of the world in small areas, provided that a high number of images comes from that region. The application we wish to create will divide the world map hierarchically, in order to reach a certain balance of number

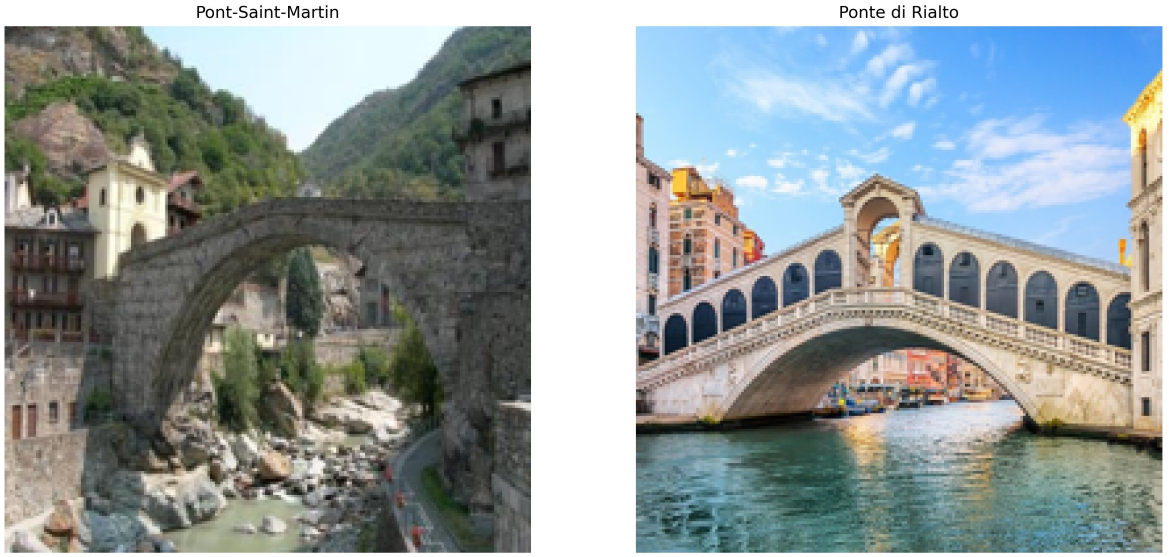

Figure 1.2: Pont-Saint-Martin in the region of Val d'Aosta, Italy and Ponte di Rialto in Venice. They both have an arch-shape due to the Roman heritage in architecture.

of images per region. For example, if the dataset includes a high number of images from New York, this work is able to divide the city in very small regions, such as different parts of Manhattan (Fig. 1.1).

Moreover, in order to perform an accurate prediction, the AI geolocating technique implemented in this work is able to exploit relevant features from images that can be used to discriminate certain parts of the world thanks to the shape of buildings, the type of trees, the animals present, etc. Such features are often common even at country or continent level, thus this information can be exploited to discriminate all the cells from those areas from other cells in different parts of the planet. For example, arch bridges are more frequent in the areas colonized by the Roman Empire, therefore the Deep Learning model can exploit such information in order to give more importance to cells belonging to such areas e.g., Italy or France. This work has to be intended as a support to the geolocating operations, as it provides the prediction of a set of possible cells and a set of visual explanations that identifies which areas of the image were relevant for the prediction. In this thesis, we will combine the state-of-the-art algorithms for cell generation together with deep learning techniques for image analysis and classification. The explanations will exploit modern feature extractors with the visual explanatory models that are most used in current deep learning applications.

2 | Related Works

In this chapter, we review the most relevant works that helped us during the development of the thesis. We analyze projects that aimed to solve the image geolocation problem and papers that helped to understand the different tasks. Moreover, we discuss visual explanation techniques and how they can be useful for improving the quality of a model prediction.

2.1. Background on Deep Learning

This section aims to provide a solid background in the Deep Learning field. We describe the basic and most general concepts related to image classification networks, such as the major entities (layers, neurons) in neural networks and the most common structure for solving image classification tasks.

2.1.1. Image Classification General Model Structure

Modern deep learning approaches are known for their excellent performance in the image data-driven classification task. In particular, modern feature extractors based on convolutional neural networks have proven to be very effective in well-known image classification tasks, such as ImageNet [4]. The idea behind these models follows a general structure [8] and we are illustrating it in this section.
If we are to represent a general structure for the most common image classification models, we can refer to Fig. 2.1. This general scheme depicts two main building blocks:

- A **Feature extractor**, that is a convolutional neural network aimed to extract relevant information from the image and encode them in a numerical representation for the next step;
- A **Fully-Connected classifier**, that mixes up encoded image information (features) in order to finalize the prediction.

Let us examine in depth this blocks and the theoretical models behind them.

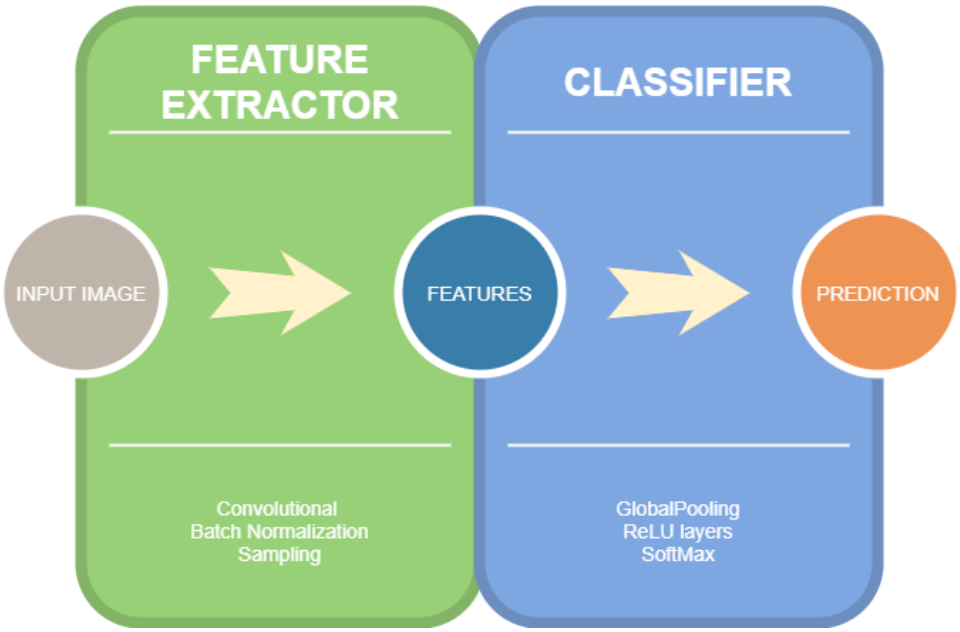

Figure 2.1: General structure subdivision in its two main parts. On the left, the Feature extractor that includes Convolutions, Sampling, and Batch normalization. On the right, a Classifier for predicting the output cells of the model. It includes a Global Pooling, a set of Dense layers and some advanced mathematical functions.

Feature extractor (CNN)

The feature extractor model is the part of a deep learning model that is devoted to manipulate and extract relevant information from the input data in order to help the model in predicting the target label. In particular, in visual deep learning models this module consists in image-related manipulations in order to extract pixels, lines, patterns and finally objects that could be relevant for the correct prediction of the output class. In the deep learning field, the state-of-the-art feature extractors are refinements of convolutional neural networks. In order to understand how this kind of neural networks work, it is necessary to explain what convolution is and how it is applied, in this case, to images: the convolution is a mathematical operation between two functions f and g.

$$h(t) = (g * f)(t) \tag{2.1}$$

As an output, the convolution produces a function h that indicates how the function g influences f. In deep learning nomenclature, f is called **Input** while g is called **Kernel**. Finally, h can be referred to as **Feature Map**. Clear examples of this application can be found in the signal analysis field, in which periodic functions have to be quantized with a certain rate. In this case, this operation can be identified by indicating the signal as f

and the quantizing (also called weighting) function as g:

$$h(t) = \int f(a) \cdot g(t-a)da \tag{2.2}$$

The output function h represents the convolution result of g over f. In this case, the convolution has been expressed as an integral due to the continuous domain of f and g. When the involved functions are defined over a discrete interval, it is necessary to replace the integral with a summation over all the possible values of the domain. This kind of convolution takes the name of **discrete convolution**.

$$h(t) = \sum_{a} f(a) \cdot g(t-a) \tag{2.3}$$

In the computer vision field, convolution can be used to manipulate images and any kind of multidimensional arrays (also known as tensors) in order to obtain a different representation.
As we often deal with 3-dimensional images (RGB), it is necessary to define a multi-axis convolution that takes into account all the dimensions of the input data. In means of mathematical formulas, we can express the input as $I(i,j,k)$ and the kernel as $K(i,j,k)$ and express the convolution of K over I as:

$$S(i,j,k) = (IK)(i,j,k) = \sum_{m}\sum_{n}\sum_{p} I(m,n,p)K(i-m, j-n, k-p) \tag{2.4}$$

The output function $S(i,j,k)$ will still be an RGB image, but its pixel will be manipulated as intended by the kernel K. The main goal of the convolution operation in CNNs is to extract known patterns (thanks to the kernels) that are gradually built at each convolution. In principle, the first operation may identify relevant points in the image. The second one could mix these points so to obtain lines, the next one could pick out shapes, then objects and so on as shown in 2.3.

In deep learning, the category of neural networks that makes use of convolution operations (Fig. 2.2) are often referred to as Convolutional Neural Networks (CNN). In particular, the CNNs are constituted by a set of Convolutional Layer in which the convolution operation is applied over the input tensor (array) a number of times which equivalent to the number of Kernels stored inside the layer.
A peculiarity of neural networks is that all the values of the kernels are trainable, that is to say they change through time when training the model so to obtain better representation of the tensors for the final prediction.

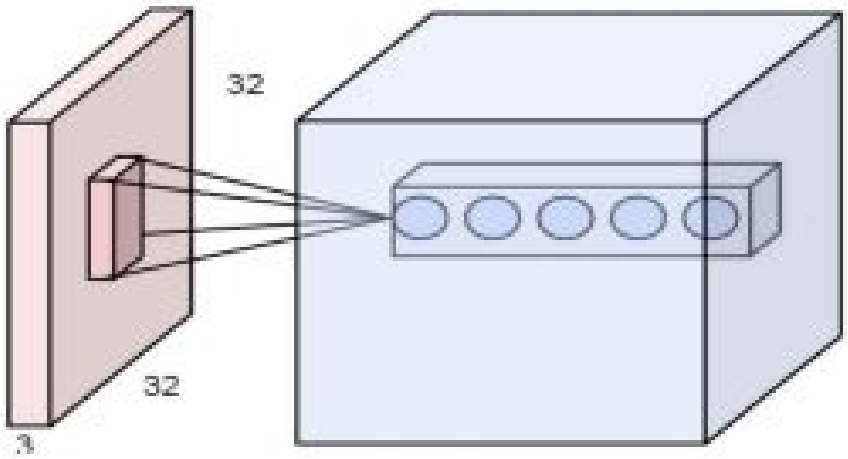

Figure 2.2: Convolution operation picture taken from [1]. On the right-hand side of the image, it is possible to see how each kernel applied on the input image (left) generates a new representation of the output independently of the other kernels.

When dealing with image data, CNNs apply a set of operation that are not only devoted to convolution, but also pooling (localized reduction of the input values through functions as local averages, ...) and non-linear activation functions.
In the end, the output of a CNN is a tensor which inherits all the transformations applied throughout the model. The aim of this multidimensional array is to provide a set of values, called features, that can help the model to predict the correct output labels in a more precise way. For example, it could tell whether there are certain oddly shaped windows or a specific kind of tree. This enables the next step to work on more refined information that could be relevant for the final prediction. Every operation applied by the CNNs over tensors results in a change of its shape. In principle, we can see a tensor flowing through the model as a **volume** that changes in depth (channels) when a convolution is applied ($depth_{new} = Num_{kernels}$, as explained in Fig. 2.2) and changes in size when pooling takes place ($width_{new} = width_{old} \div pool_{size}$, the same holds for the height). Nowadays, many feature extractors have been implemented for solving the most common image classification problems. Some examples of the most advanced and performing state-of-the-art feature extractors are VGG [23], ResNet [10], Inception [25] and EfficientNET [26]. Empirical experiments have evinced that modern CNNs perform brilliantly at classifying images, thanks to a set of manipulations on the input image to obtain its most relevant features for the output prediction (Fig. 2.3).

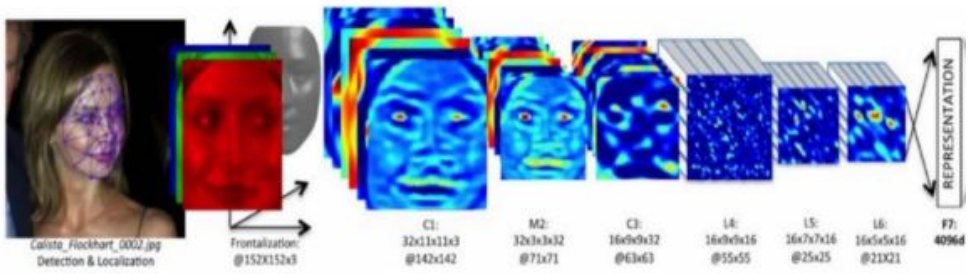

Figure 2.3: CNN manipulation picture taken from [1]. It shows how the network gradually extracts features from the image in order to provide relevant information for the next steps.

Global Pooling

As it is possible to deduce from Fig. 2.1, the output of the Feature Extractor is directly piped to the Cell Classifier. In particular, the input to this module is represented by a CNN volume of features represented by floating-point values. As deep learning classifiers work with bidimensional data (batch, linearized feature array), the feature map (volume) generated by the Feature Extractor needs to be flattened in a vector-styled representation. This operation is performed as explained in Fig. 2.4 by the GlobalPooling function: depending on the specific operator, this layer is responsible for reducing the dimension of the input volume to its slices in order to obtain a single output slice in a vector-styled fashion. Whenever features are ready for the Cell Classifier, they are fed into a Fully-Connected set of layers.

Fully-Connected Classifier (FCNNs)

In this section we discuss the second part of the model structure which is responsible for mixing up the feature flowing from the previous part (Feature Extractor).
Fully-Connected Neural Networks are the natural evolution of the old concept of Perceptron. The latter computational block was constituted by a set of parameters (a bias and a number of weights) and a step/sign function. In particular, the perceptron unit is designed to mimic the behavior of the human brain's neurons: they accumulate the input values in a weighted sum (same as the electrical sources of the brain). Whenever the summation goes beyond a given threshold (h_j function, which could be either a sign or a step function), the perceptron is activated and sends the output signal (weighted sum) to the next units to which it is connected. Both weights and bias are trainable. In particular, every perceptron performs a weighted sum of its input and submits it to a linear activation function:

$$f(x) = W^t * x + b \tag{2.5}$$

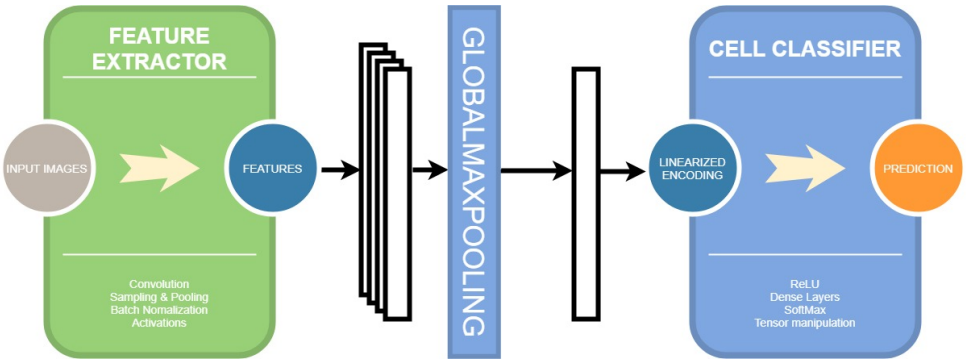

Figure 2.4: In the picture, it is possible to understand the global pooling operations of linearizing the feature dimension.

and the result of f(x) is then submitted to the activation function h_j:

$$output = h_j(x|w_j, b) \tag{2.6}$$

The problem of such a unit was that it could learn only a limited set of functions (only linear). With the advent of neural networks, perceptrons have been replaced by neurons that are endowed with non-linear activation functions, such as the ReLU (continuous approximation of the perceptron discrete function).

Neurons are grouped together inside Dense layers, which are identified by their parameters (weights and biases). As far as the Dense layer is concerned, we can summarize its functioning in the following expression:

$$h(x) = g(W^t * x + b) \tag{2.7}$$

g can represent the application of the non-linear function (most of the time a ReLU) to an affine transformation of the input values (encoded linearized features).

In order to propagate information flowing through models, Dense layers are fully-connected, therefore each neuron of a given layer X_1 is connected to all the neurons of the next layer X_2. This allows to mix up the information thanks to the weighted sum of all inputs coming from the previous set of neurons, in each neuron of the current layer.
The mesh-like connections are the reason behind the name Fully-Connected Neural Networks. Depending on the different non-linear activation function of the Dense layers, they can assume different behaviors. For example, some could be designed to shape an unknown function that the neural network wants to compute, while others could be used

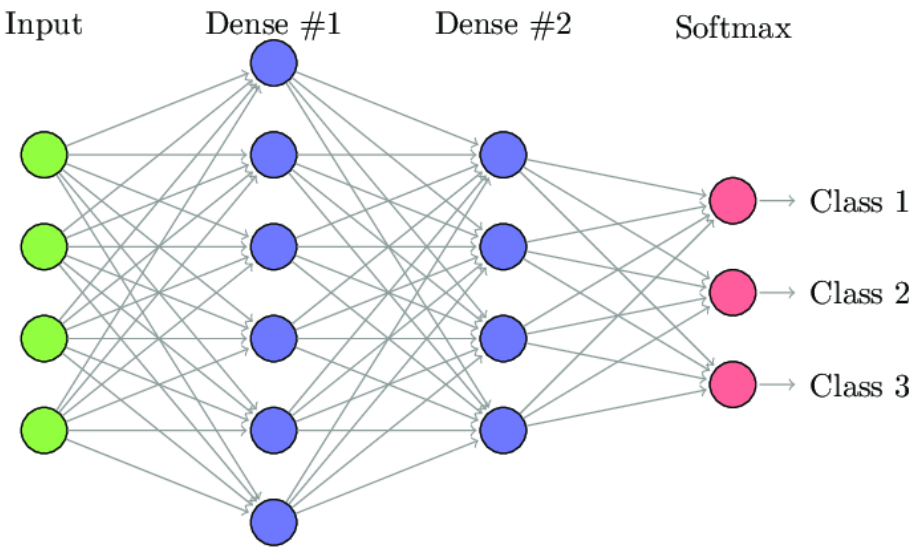

Figure 2.5: Fully-connected network structure from [19].

to compute the final probability values, also known as confidence levels.
In the image classification task, FCNNs are usually placed after the CNN in order to mix up the extracted features and compute the final prediction.

2.1.2. Activation functions

The activation functions are the elements that trigger neurons whenever the information feeds forward in neural networks: depending on the layers' goal, engineers must know which is the most suitable function for a certain type of operations. In this section, we illustrate the state-of-the-art activation functions of neural network layers and show their behaviors.

ReLU

The ReLU (Rectified Linear Unit) is an activation functioned defined as the positive part of its argument. It can be expressed as:

$$g(z) = max\{0, z\} \tag{2.8}$$

In order to introduce the necessary non-linear behavior of fully-connected neural networks layers, hidden layers use to be characterized by ReLU functions (Fig.2.6).
The main advantage of exploiting the ReLU function is that it is easy to optimize thanks

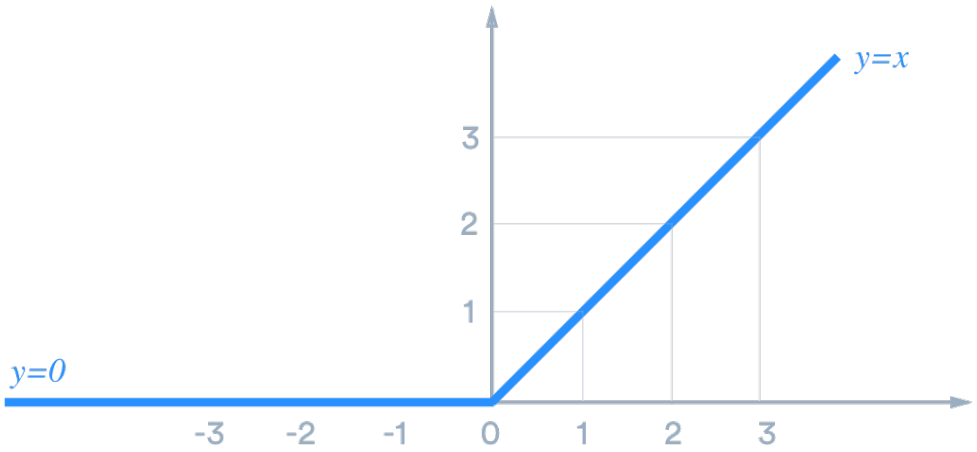

Figure 2.6: Cartesian representation of the ReLU function[1]

to its similarity to linear units. Moreover, its derivatives remain large whenever the unit is active (greater than zero) while it is zero in any other case, speeding up the gradient computation and, subsequently, the neural network backpropagation.

Through the recent years, researchers have designed new variations of the ReLU functions, such as the Leaky ReLU that avoids zero values that could affect the gradient computation [6].

SoftMax

The SoftMax function is a differentiable function for representing probability over a discrete variable with n possible values. If we consider the output of a linear layer z, the softmax function can be expressed as:

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{2.9}$$

The main feature of this function is that it normalizes the output variable inside the interval [0,1] and also biases the result towards the maximum value, in the sense that it stirs the result as close to one as possible.

2.2. Geolocation problem

The image geolocation problem has started becoming interesting since the late 2000, with the IM2GPS [9] work, and then it evolved into deep learning based techniques. In particular, we will be showing that, after the research direction was started from a pure

[1]Source: `https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7`

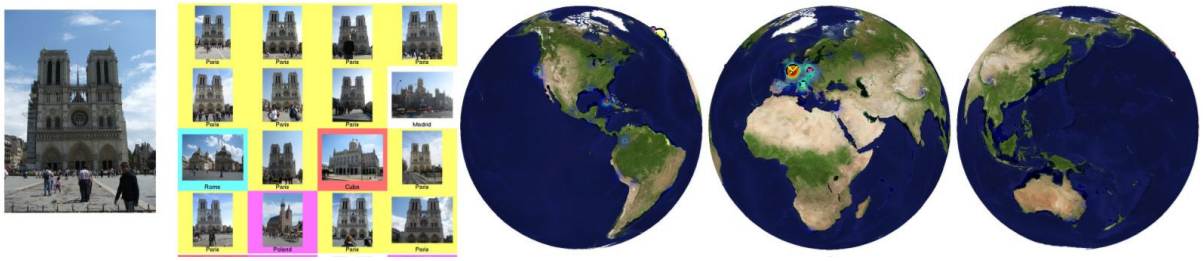

Figure 2.7: From left to right: target, k-nn, three representations of the probability distribution of the image over the world.

computer vision approach, two main directions were followed for solving this problem: the geolocation by classification (GBC) approach and the image-based localization (IBL) approach.

In this work, both approaches have been analyzed, but in the end, we decided to follow the approach of geolocation as a classification task, especially taking cues from PlaNet [29] and Müller-Budack et al. [17] hierarchical classification models.

2.2.1. IM2GPS

IM2GPS [9] was the first work that aimed to geolocate any type of image in any part of the world. The authors started their work from the extraction of geolocation features directly from the image and inspired all the future works that will be analyzed along this section. The authors created a database of images taken from very significant parts of the world and compared the incoming query images with such a repository. In particular, they developed an algorithm for estimating a probability distribution over geographic locations from an image using a purely data-driven scene matching approach. The scene matcher algorithm is based on a set of computer vision techniques, such as color and texture histograms (bins), Gist Descriptor and geometric context. At inference time, the model is built to find the most similar images in the database compared to the query with respect to the previously enlisted features in a k-Nearest Neighbour (k-NN) fashion. This work shows that even a system only based on image retrieval (similarity) performs far better than humans and any previously existing computer vision systems did in the task of geolocating images without any cues but the pixels.

2.2.2. Geolocation by Classification (GBC)

Solving geolocation as a classification task has become more popular when deep learning started to become powerful enough to outperform any machine learning model in the

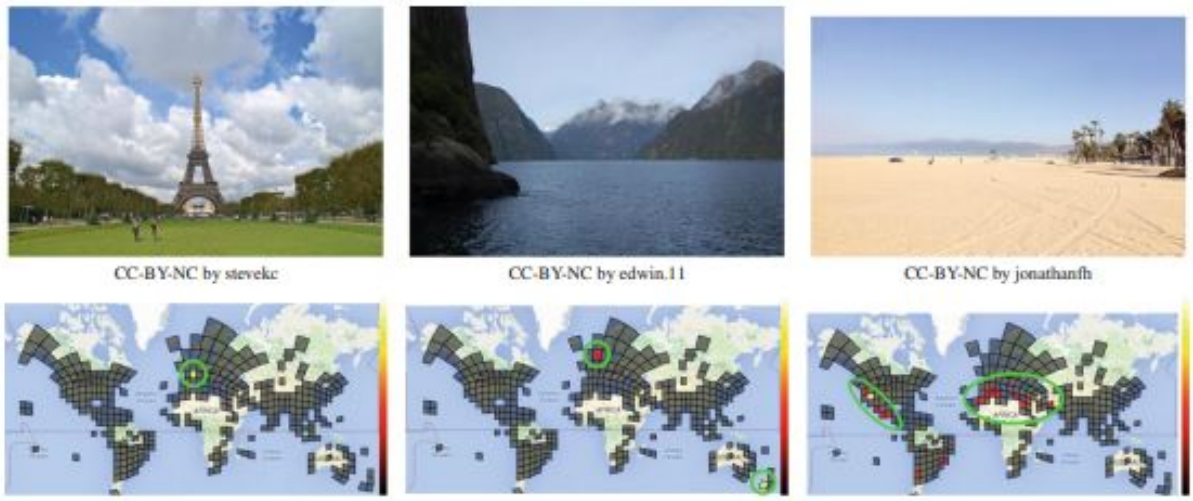

Figure 2.8: PlaNet examples of predictions on different pictures and locations.

classification task. Furthermore, the geolocation by classification techniques are based on the division of the world map into a set of cells and predicting the correct one. Different ways of dividing up the world map has been taken into account and the main ones are discussed in this section.

PlaNet

PlaNet [29] is one of the first works that aims to perform image geolocation as a classification problem from a worldwide perspective. The majority of works that are to be found in literature are restricted either by a specific set of images or a specific area, while this work aims to be working without any specific constraint. In particular, the paper aims to localize any type of photo taken at any location using just its pixels (Fig. 2.8). The model was very innovative due to its data-driven nature, which allowed developers to train an effective model without having any clue on the kind of images and features the model was trained with. The authors took the world map and divided it into cells using the S2 Google's algorithm (Appendix A). In order to have balanced classes, a couple of thresholds (minimum and maximum) for the number of images per cell are set. The images are passed through a convolutional neural network based on Inception with batch normalization. Such a model is trained using a custom training loss that is based on the euclidean distance (in meters) between the centroid of the predicted cell and the real coordinates of the image under analysis. As a further effort, the authors extended the image geolocation task to photo albums, in order to show the location correlation between images coming from the same set. LSTMs were used for this specific task with several settings and configurations.

The model for this advanced task is composed by the plain PlaNet model and an LSTM unit that provides the recursive part. In order to obtain better performance, the authors mainly worked on the LSTM sequence, making experiments on label offsets, repeated sequences (first pass creates the representation, second pass makes the prediction) and a bidirectional LSTM (capped at 25 images). The LSTM bidirectional setting showed improvements with respect to the plain PlaNet model, but it is meant for solving a different problem (album correlation).

The results that the authors obtained were impressive for that time. The main reason for such a performance can be found in the impressive amount of training data they crawled (189 million images), even though the training phase took months.

Hierarchical Model and Scene Classification

The work of Müller-Budack et al. [17] aims to improve existing geolocation models by exploiting a geographical subdivision of the world into cells and solving the problem as a classification task and exploit hierarchical properties of the cell subdivision process. The authors aim to generate a set of cell hierarchies exploiting the S2 algorithm: in particular, they generated three different-grained subdivisions (coarse, middle, fine) by varying the two thresholds set by the algorithm. After that, an algorithm is run over any image in order to find the correspondence between the image coordinates and the corresponding cell in each of the previously mentioned subdivisions. In the end, the algorithm discards all the images for which the algorithm has not found a corresponding cell for each subdivision. In order to get a higher prediction accuracy, they added a scene classification network at the beginning of the model that is able to tell whether an image is indoor, urban, or natural. The result is fed as an input to the geolocation estimator that fuses this information together with the feature extractor. Whenever the preprocessing operations have been performed, the model is responsible for predicting the cells of the respective groups and multiplying them in order to get the final prediction (YOLO9000 [20] approach). By using this multitasks approach they were able to outperform any current network, including the older PlaNet and IM2GPS, even using a more restricted set of images. The performance improvement can be found in the use of a more modern set of feature extractors (ResNet-50) rather than older ones, an accurate and filtered dataset and the hierarchical model which is able to exploit features at different granularity levels. For example, it is common to find Pagodas in Asia and only the ones with wooden flat roofs are common in Japan: in this way, the model can learn that Pagodas are Asian (coarse-grained subdivision) and, more particularly, Japanese (mid-grained subdivision) and can restrict the search space accordingly.

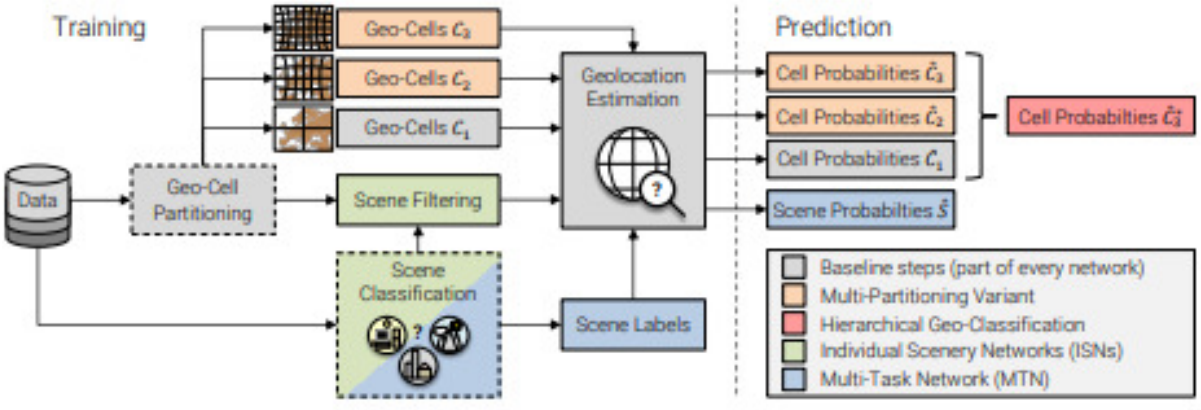

Figure 2.9: Global pipeline of the Hierarchical model proposed by Müller-Budack et al [17]. From left to right: pre-processing, multi-grained predictions and scene classification, feature fusion and cell prediction.

2.2.3. Image-Based Localization (IBL)

The image-based localization (IBL) is a retrieval-based image localization technique that is performed for estimating the geographical position of a query image by applying a similarity measure that provide the nearest reference images from a precomputed database. This approach forces models to have a very large number of images from different locations around the world and keep them in a database for the inference.

IM2GPS Revisited

With this paper [28], the authors want to improve the old IM2GPS model [9] by exploiting modern deep learning techniques. In particular, the goal is to estimate the geographic location of a query image by applying kernel density estimation to the locations of its nearest neighbors in the reference database. Differently from IM2GPS, the author's effort focussed on using a deep learning model for feature extraction (classification loss at training time) and use its result in order to define a DML for inference (image retrieval at test time).

As far as the classification training is concerned, the researchers avoided using an image density based method for splitting up the world into regions. Instead, they used to quantize the world map by dividing either vertically or horizontally each of the cells, depending on the size of the resulting cells (whichever side is bigger) until the number of images in each cell is below a certain threshold t_{img} or the physical area is smaller than another threshold t_{area}. In order to look at the models' behavior, they tried several partitioning

Figure 2.10: Revisited IM2GPS procedure, from left to right.

settings by varying such verges. As far as inference time is concerned, predictions are calculated by a k-nn density estimation based on a Gaussian kernel that takes into account the label similarity (GPS distance) and also the query image and reference image features (Fig. 2.10). By comparing the results that authors achieved by several models, the work showed that classification-based networks work better than image retrieval ones. In fact, this work outperforms the old IM2GPS but is not as good as PlaNet. If the performance metric focussed on a fine-grain localization (street-level), the IBL approach seems to work better than classification, but in any other case, the latter is better. Even considering the spatial and time complexity, when performing classification, the model needs to keep only the neural network (feature extraction and estimator) whereas the IBL needs to keep the whole database of images in order to compute the k-nn similarity, together with a complex k-d-tree data structure that is necessary to fasten the search. Also, the model needs a massive number of examples, since it can't generalize multiple objects in the image: for example, an image contains a house roof seen often in Italy and a boat style from the Southern regions, etc.

Self-supervising Fine-Grained Region Similarities for Large-Scale Image Localization

This work [5] develops a technique that improves the similarity metric used in IBL in order to find the most similar set of images from the model dataset in order to perform geolocation. In particular, the authors tackle the problem of finding true positives over a noisy GPS dataset: as a matter of fact, the fundamental challenge of image retrieval-based methods is to discard all the similar-looking locations in a GPS-tagged dataset. The goal of the authors is to find what they call Fine-Grained Region similarities, that are correspondences between similar true positive pairs of images at the level of their regions (top, left, bottom, right). In fact, image-level labels might not coincide with the region-level labels: pairs of images that are found to be true positive pairs might have different region-level labels. In order to find the fine-grained region similarities, the work defines a self-supervised network that extracts possible true positive pairs and uses them in order

Figure 2.11: (a): image classification with a single label all over the image; (b): image classification with 4 split region labels and similarities.

to enhance the performance of the model in the next training iteration. In particular, the model is composed of a backbone encoder (dense feature map) based on VGG-16 with ReLU activation functions and a VLAD layer (aggregates into a compact feature vector). In the meantime, the loss function is the combination of a hard-softmax loss L_{HARD} and a soft-softmax loss L_{SOFT} . The first is evaluated on-the-fly by the network in order to compute the image-to-region similarities for selecting the most difficult negative regions, while the second is able to compute fine-grained similarities to encode more discriminative local features for difficult positive regions from the network of the previous generation. The performance of this work is remarkable, even though it is limited to the set of images from the Pitts30k dataset, thus the problem is restricted geographically. In our opinion, if the problem would require to be scaled on a worldwide perspective, it would get a significant drop of performance, as the number and the heterogeneity of different locations would grow too much to be handled by a similarity-based technique.

2.3. Geolocation Task Configuration

In order to work with deep learning models, it is necessary to select a machine learning task and configuration of the geolocation problem. In particular, it is necessary to decide whether it is reasonable to investigate in order to directly predict an image pair of coordinates or to treat the problem as a classification task over a finite set of regions in a worldwide setting. The first approach implies to treat the problem as a multiple linear regression task over latitude and longitude, as they range respectively as floating-point numbers between [-180,+180] and [-90,+90]. The main difference between the two approaches is that the first is more precise than the second, as the latter introduces an intrinsic error due to the cell size: the model predicts the correct cell with a certain spatial dimension and the image could be located anywhere inside this area. On the other hand, predicting the exact coordinate through a regression task is far too difficult, especially using deep learning approaches that deals with images. The second approach is the one suggested by [9, 17, 29] and consists in dividing up the world in a finite set of regions.

Figure 2.12: Pair of images from the city of Milan. They were taken from a distance of less than 1 kilometer.

In these settings, deep learning models have to be trained in order to predict the correct region in which the image falls into. This rationale leads the problem towards a multi-label multi-class classification task. In particular, neural networks have proven to work accurately with classification tasks, therefore the second approach has been preferred over the first one.

2.3.1. Cell Generation

Geolocating outdoor images is a very difficult problem due to the heterogeneity and diversity of photos within the same areas. In some cases, pictures may change a lot even inside the same city: if we consider the city of Milan, it is common to find skyscrapers and modern buildings in the Porta Garibaldi area, while the biggest Chinatown (via Paolo Sarpi) is a few kilometers distant and accordingly every house and shop is different (Fig. 2.12). If we had enough images from each of the Fig. 2.12 locations, a good idea would be to define two separated cells for the different areas. Therefore, we think that building an adaptive set of classes depending on the dataset is crucial in the task definition and in the model accuracy. With the term adaptive, we intend to divide the world up

in finer-grain regions where the largest number of images are taken, while it makes sense to keep vast areas for those locations which do not have a sufficient number of images coming from them.

Algorithm 1
Generate a set of cells starting from a dataset that only contains the association Image name — (Latitude, Longitude)

Input: Dataset of images and coordinates, lvl_{start}, lvl_{end}, t_{max}, t_{min}
Output: Hierarchical tree of cells, Cells information

```
Let lvl_curr = lvl_start.
Let Cells = [ ]
while lvl_curr ≤ lvl_end do
    Let curr_lvl_cells = [ ]
    for cell in Cells do
        if cell.images > t_max then
            new_cells = GenerateCells(cell, lvl_curr)
            curr_lvl_cells.add(new_cells)
        end if
    end for
    Cells.add(new_cells)
    lvl_curr+ = 1
end while
for cell in Cells do
    if cell.images < t_min then
        Cells.remove(cell)
    end if
end for
return Cells
```

In order to shape the hierarchical world division for our task, we decided to change the algorithm conditions to stop whenever the number of images of a specific cell would be below a certain threshold t_{max}. In the literature [17, 29], we found the Google S2 Geometry library [7] which enables us to deal with grid building in an adaptive way. More information can be found in Appendix A. Moreover, we decided to discard all those cells that contained less than a t_{min} number of images in order to keep the cell as balanced as possible (bounded between t_{min} and t_{max}). The procedure of generating the hierarchical division of the world in a grid of cells can be described as in Alg. 1. We took inspiration from the [17] GitHub repository and modified it in order to shape the results to the problem structure. Once the cells have been defined, it is sufficient to assign them to each of the dataset images in order to have the dataset ready for training and validation. Whenever a new test set is introduced, it is sufficient to assign its images to the cells generated for training.

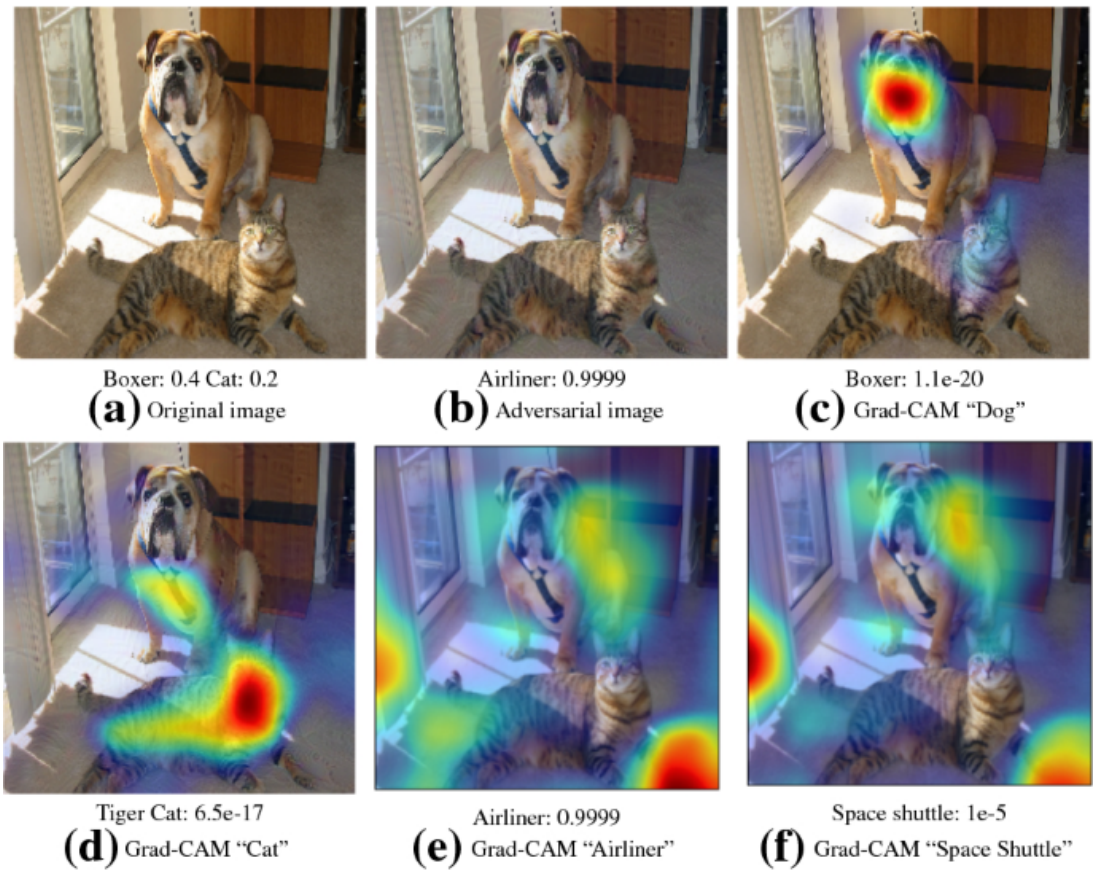

Figure 2.13: An example of the Grad-CAM [22] output on a classification task over the ImageNet dataset.

2.4. Visual Explanation

In this section, we will discuss the work related to the visual explanation that helped throughout the process of identifying the relevant areas of the images for the prediction.

2.4.1. Grad-CAM

The Gradient-weighted Class Activation Mapping (Grad-CAM) [22] uses the gradients of any target concept flowing into the final convolutional layer to produce a saliency map, highlighting the important pixels in the image for predicting the output class.
More in particular, this approach aims to produce a heatmap that identifies which are the most important pixels that contributed to the final prediction of the deep CNN.
Thanks to their versatility, Grad-CAM can be used over CNN that covers many tasks, such as CNN with fully-connected layers, with structured output and so on.

The core idea behind Grad-CAM is that it uses the gradient information flowing into the last convolutional layer of the CNN to assign importance values to each neuron for a particular decision of interest. In the same work, the authors show a further variant of the Grad-CAM that exploits the Guided backpropagation in order to visualize gradients with respect to the image, where negative gradients are suppressed when backpropagating through ReLU layers. In practice, it aims to capture pixels detected by neurons, not the ones that suppress neurons (negative gradient).

The main feature that the authors wanted to bring is interpretability: when dealing with deep neural networks, developers work in a black-box fashion, without really understanding what is happening inside the network when a prediction is made. Grad-CAM provides a visual explanation of what is really happening at pixel-level. When AI proves to be significantly better than humans, it can even be called machine teaching.

3 | Research Questions

In this section, we pinpoint which are the research questions that we address in this thesis work. In particular:

1. Is it possible to provide a deep learning model which is able to geolocate any kind of outdoor image from a worldwide perspective?
2. Is it possible to improve the performance of an image geolocating model by adding the hierarchical information of how the world was divided up directly inside the model?
3. Can a deep learning model provide a significant visual explanation that could be used to understand why the image has been predicted in such a location?
4. What is the best way to predict the location information? How can we evaluate that the model is predicting reasonable locations?

We argue that exploiting modern feature extractors it is possible to obtain features that are unique to certain locations, such as palms for coastal areas and certain jagged peaks for young mountains, such as the Alps.

A certain number of works related to these questions have been explained in the previous chapter, such as PlaNet [29], IM2GPS [9] and its new interpretation [28] and the Hierarchical Approach proposed by Müller-Budack et al. [17]. Although this works offer good performance even at a fine-grained level, the image geolocation problem has not been taken into account for the last three years. In particular, the research directions lead toward a landmark-based kind of geolocation [30], without the possibility of geolocating images taken in remote or unpopular places of the earth, such as small villages or forests, where natural disasters and catastrophes are getting more frequent nowadays. The advantage of creating a deep learning model is that all of this information has not to be explicitly clarified, as the model is able to get them from the feature extractor and mix them up in the predictor layers.

As far as the second question is concerned, previous works [9, 17] tried to create a set

of possible grids of cells as an overlay to the world map and tried to predict in any of these settings. At the end, the final prediction was the product of the cells' confidence that were overlapped in the same world region. Instead of following these approaches, we argue that it would make a better fit to use a single hierarchical world grid of cells and exploit the information of the cells hierarchical generation directly inside the deep learning model: we think that predicting at each level of the hierarchy and mixing up all the branches' confidence would make a more accurate solution rather than having several grids.

As a further refinement, this work aims to provide users with a demo that produces predictions followed by a set of visual explanations coming directly from the model. In the case of emergency services, providing evidence of the reasons behind a prediction may clarify even further the prediction: by the means of a map with the most probable regions it is possible to get the correct location even if the model failed to give the exact one. Moreover, we assert that having a visual indication concerning which parts of the image were relevant for the prediction may lead to a more accurate understanding of the real geographical position of an image.

Finally, we want to demonstrate that the model we build is predicting reasonable results. With this, we want to test the model calibration [2] and balancing with respect to the number of samples in the target classes (grid cells).

4 | Approach and Model architectures

In this section, we analyze how we approached the outdoor image geolocation problem detailed in the research questions. In particular, we describe the novelties that we introduced in terms of the hierarchical model structure, that is to say including the cell dependencies directly inside the deep learning model.

As an implementation of the Image Classification General Model Structure that we described in the section 2.1.1, we discuss the models that we created for solving the outdoor image geolocation problem. The first operation we perform is a set of data augmentation techniques in order to exploit more information from the same training data. Immediately after, we place as input of the feature extractor a batch of 32 RGB resized images (224,224,3). The output of this block of the general scheme produces a volume of extracted image features shaped as (1280,1280,8).
In order to select the best feature extractor for the outdoor image classification task, we ran several experiments over different state-of-the-art convolutional neural networks and analyzed their performance. Therefore, we empirically found that EfficientNET in the B1 implementation with pre-trained weight on the ImageNet task is the one that outperforms the others in terms of training time and accuracy (Experiment A).

In order to link the Feature Extractor with the classifier, we needed to implement a GlobalPooling function that could flatten the output volume produced by the first building block. After some empirical experiments, we discovered that the GlobalMaxPooling operation performed better than other functions, such as GlobalAveragePooling and Flattening.

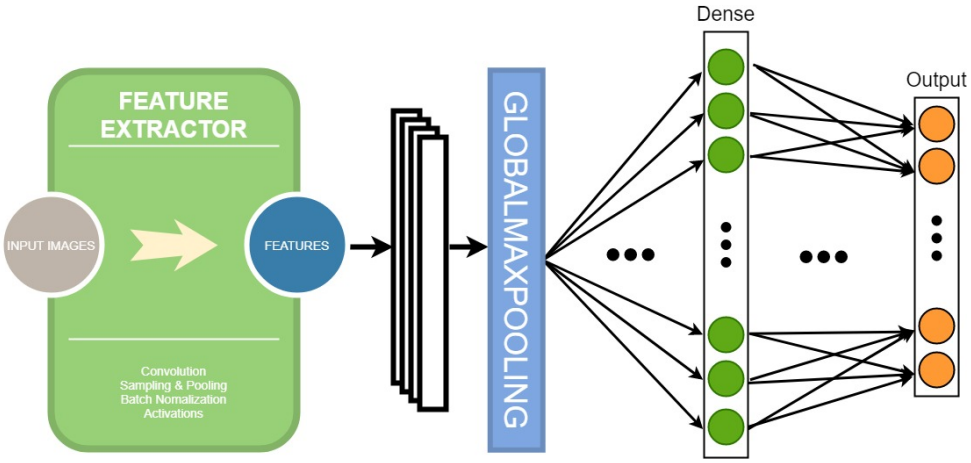

Figure 4.1: In the picture, it is possible to observe the structure behind the basic multi-class model in its layers structure. Green dots represent ReLU activated neurons, while Orange dots represent SoftMax activated neurons.

As far as the last building block is concerned, we train a set of Cell Classifiers implementing:

- a standard Fully-Connected multi-class model;
- a very powerful set of custom neural networks.

The latter embeds the hierarchical information concerning the grid generation in order to refine the predictions of leaf cells. It is important to mention that all models have been trained using the CategoricalCrossEntropy loss function (described in chapter 6). In the next sections, we discuss all of our models in details.

4.1. Basic Multi-Class Model

The basic multi-class model is represented by a cell classifier based on the fully-connected architecture explained in the section 2.1.2. In particular, this model aims to fully combine features from the GlobalMaxPooling layer in order to obtain a correct and final representation for the output layer of the neural network. This operation is necessary for combining information flowing from the feature extractor in order to obtain a new representation that is as close as possible to the output cell descriptions. As depicted in Fig. 4.1, the features are combined inside a Dense Layer constituted of 1280 neurons. This layer embeds a set of 1280 x 1280 weights that helps neurons to compute a weighted sum of the inputs and apply a ReLU activation function. In particular, the model learns a specific mixed feature description for each of the cells of the world grid and the output

of the above expression is the closest representation to the one used for predicting the final grid cells. Whenever the outputs of the Dense Layer are ready, they are piped into the last network layer, which is called Prediction Layer. In particular, it is characterized by a number of neurons which is equivalent to the output classes of the task. In order to provide a confidence as the output of the model, the Prediction layer embeds a SoftMax activation function, that outputs a set of probability values associated to each of the leaf cells of the problem.

In the end, the output values of the Prediction layer constitute the confidence level for each of the output targets of the classification task, that are the cells of the grid-based world division.

4.2. Hierarchical Models

The hierarchical model constitutes the core of the novelties developed in this thesis work. After creating a naive fully-connected model for classifying the encoded features coming from the Feature Extractor, we want to find out whether the model could perform even better by exploiting the information coming from the hierarchical cell generation. Some related works in the field of image geolocation tried to improve the model performance by creating different kinds of features encoding [31, 32] and some others exploited some computer vision and geometrical properties of the image [5, 21]. The main works using these techniques happened to operate well only when restricting the field of application only to certain categories of images (e.g. landmarks) or to specific areas, such as North America or even specific cities. Whenever the scale of the geolocating task aims to get a worldwide perspective, many of these models would perform worse than their constrained settings, as encoding for small areas is easier to manage rather than a global representation. In order to enhance the model performance, the authors of [17] tried to generate a certain number of grid-based cell divisions at different grain levels and combine them together in order to obtain a hierarchical model. Although this latter model can be called hierarchical, it does not really exploit all the embedded knowledge inside the cell generation procedure. Our work aims to make use of the grid generation knowledge, especially how cells are generated from bigger cells and how they are linked together. In our case, the hierarchical information is not only exploited in the inference phase, when predictions from different levels of granularity of the world grid are combined, but directly inside the model structure.

As it is possible to deduce from Fig. 4.2, the aim of our work is to create a hierarchical relationship between cells of different size: this leads the procedure to build a tree-like

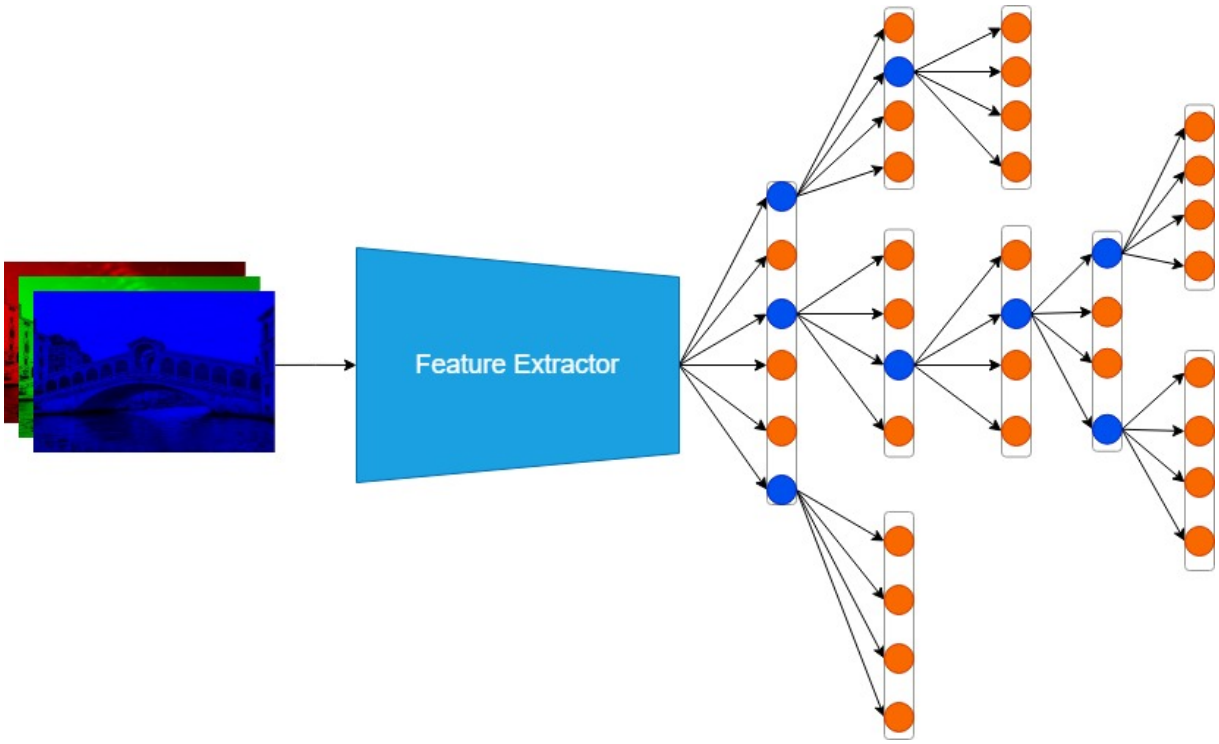

Figure 4.2: Visual representation of the model hierarchy. From the left to right, the input image divided in the RGB channel, the Feature extractor and the hierarchical cell representation in terms of neurons. Blue neurons represent intermediate cells, while orange neurons represent leaf cells. The arrows stand for the dependencies between cells.

structure in which deep cells are fractions of cells from upper levels. Therefore, it is possible to identify groups of cells that are generated from the same parent cell, and this inevitably drive us towards a tree structure of cell dependencies.

In our case, we can represent the cell hierarchy using a tree-like structure, with a small expedient: there is no single root of the tree structure. This is uniquely due to the Google S2 Geometry library [7], that is responsible for generating cells: the procedure (Algorithm 1) starts from six different cells (projections of six cube faces on a unitary sphere, as explained in Appendix A), therefore it produces six different root nodes entailing six different trees. In order to keep the hierarchical dependency concept that we explained previously during the models' implementation, we transformed the tree structure into a destination-oriented directed acyclic graph (DODAG), as explained in Fig. 4.3. This structure enables us to express the dependencies between hierarchical cells through neural network connections. In our implementation, we decided to adapt the fully-connected neural network model to the hierarchical DODAG by switching off the connections that were not present inside the graph. All the implementation details will be given in the

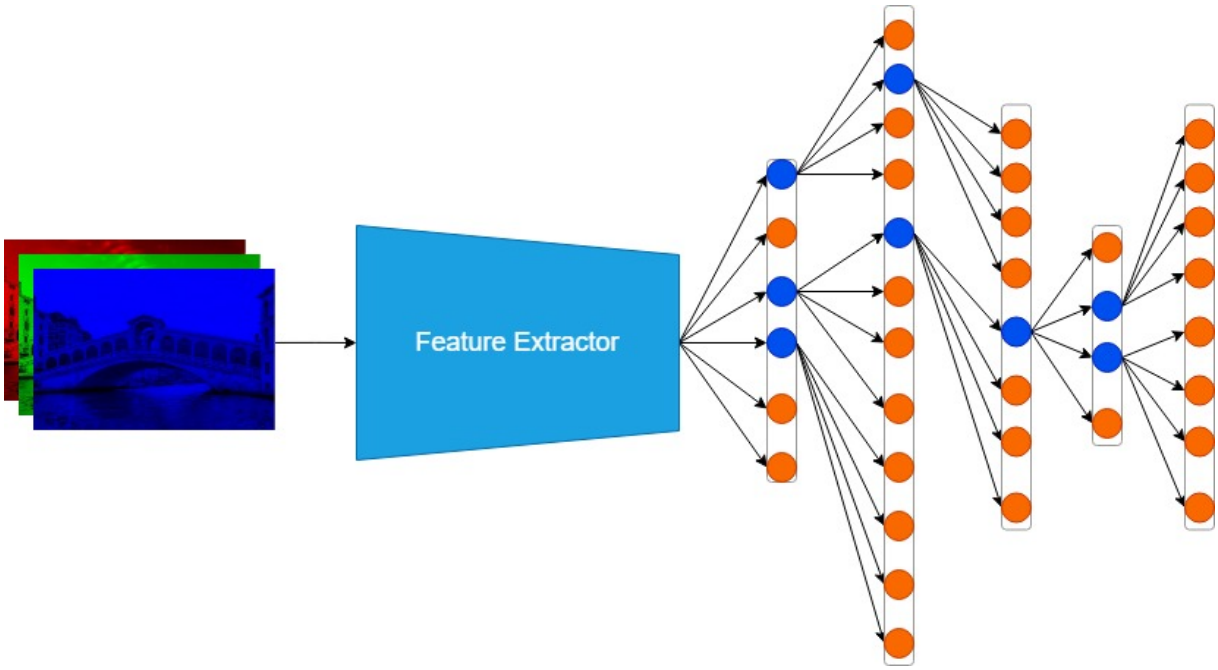

Figure 4.3: Hierarchy-level based visual representation of the model (DODAG). Blue neurons represent intermediate cells, while orange neurons represent leaf cells. Every box identifies a level of the hierarchy.

next section. It is for this reason that we decided to name these image geolocation models as "sparsely-connected hierarchical models".

4.2.1. Sparsely-connected Model Construction

In order to explain how the hierarchical model construction works, it is necessary to recap how the process of generating cells works. As described in the Algorithm 19 and in Appendix A, the procedure starts creating the first level cells (lvl_{start}) as projections of the six faces of a cube on a unitary sphere. After this operation, it computes the number of images per each cell and decides whether to split the cells even further, depending on the threshold t_{max}. This step is repeated until convergence (the procedure reached the last level of hierarchy lvl_{end}). In the end, all those cells that contain a number of images that are below a given threshold t_{min} are discarded for preserving the class balance. The output of the Cell Generation procedure includes both information about the generated cells and also any details related to the recursive formation of the world grid: whenever a cell gets divided, this information is recorded as a link between it and its generated children. Finally, a destination-oriented directed acyclic graph (DODAG) of cell dependencies is produced by the algorithm in a linked list implementation. An important feature of this graph is that for each cell (node) there is exactly one incoming arc (except for lvl_{start}

cells). This data structure is then piped in a .csv file directly into the hierarchical model definition.

The previously described operations have to be intended as pre-processing operations that are computed directly on the dataset in order to build up the problem structure. Once the dependency graph of hierarchy has been created, in short DGH, it is possible to begin with the hierarchical model construction. The procedure is described in the Algorithm 2.

Algorithm 2
Generates a set of binary matrices for building up the model structure

Input: hierarchy_graph
Output: binary_matrices

Let leaf_cells = hierarchy_graph.leaves
Let hierarchy_matrix = []
Let curr_branch = []
for cell in leaf_cells **do**
    Let curr_cell = cell
    Let lvl = current_cell.lvl
    while $lvl \geq 0$ **do**
        Let curr_lvl_cells = hierarchy_graph[lvl]
        Let parent_cell = curr_lvl_cells.children.contains(curr_cell.id) ▷ Traversing the graph backwards. The exactly-one arc feature of the graph prevents having more than one parent cell.
        curr_branch.insert(index=0,parent_cell)
        curr_cell = parent_cell
        lvl = lvl - 1
    end while
    hierarchy_matrix = $hierarchy\_matrix \cup curr\_branch$
    curr_branch = []
end for

Let lvls_size = []
for lvl in $\{hierarchy\_graph.bottom\_lvl, \ldots, -1\}$ **do**
    lvls_size.insert(index=0, hierarchy_tree[lvl].size)
end for

Let $\epsilon = 10^{-2}$
Let binary_matrices = []
for i in {0,...,bottom_lvl + 1} **do**
    binary_matrices.append([])
end for

28: **for** branch in hierarchy_matrix **do**
29: **for** i in $\{0, \ldots, lvls\_size.length\}$ **do**:
30: Let binary_array = $\bigcup_{j=1}^{lvls\_size[i]} [\epsilon]$
31: **if** i < branch.length **then**
32: binary_array[branch[i]] = 1
33: **end if**
34: binary_matrices[i].append(binary_array)
35: **end for**
36: **end for**
37: binary_matrices = $binary\_matrices^T$
38: **return** binary_matrices

The main rationale behind the Algorithm 2 is to create a set of matrices that could be used to build the connections between neurons in a fully-connected set of layers, mimicking a sparsely-connected network. Starting from the hierarchy graph provided by the Algorithm 19, the latter procedure extracts the leaf cells, that are all elements of the graph that do not have any outgoing arc (rows 1:3).

Successively, the algorithm extracts all the hierarchy graph branches starting from the leaves and going backwards until the lvl_{start} cells. In particular, the graph is analyzed in a bottom-up approach, analyzing the arcs in the opposite direction to discover the parent of the current cell and rebuild the leaf cell hierarchies. It is important to underline the fact that leaf cells do not have to be placed at the same hierarchical depth: the generating process of the world grid may stop for some cells while continuing for others that happen to have more than t_{max} images in them. In the end, the curr_branch list is produced for each leaf cell, containing its hierarchical path up to one of the root cells, and it is merged together with the other leaf branches inside the hierarchy_matrix (rows 4:16). At this point the algorithm computes the number of cells for each level of the hierarchy and stores the results in the lvls_size list (rows 18:21).

As for the last operation, the procedure prepares a set of empty matrices called **binary_matrices** that will be used as placeholders for the output matrices (rows 23:27). When the algorithm builds the matrices (rows 28:36), it places ϵ in correspondence to the links that have to be neglected inside the model structure, while putting 1 where the link must remain active. Instead of using zero values as placeholders, we decided to use a small number ($\epsilon = 10^{-2}$) due to a technical issue: after several attempts we discovered that the deep neural networks would stop learning due to zero values (vanishing gradient descent), thus we opted to replace them with a small number that could make the links negligible with respect to the links identified by ones. Before returning the binary_matrices, they

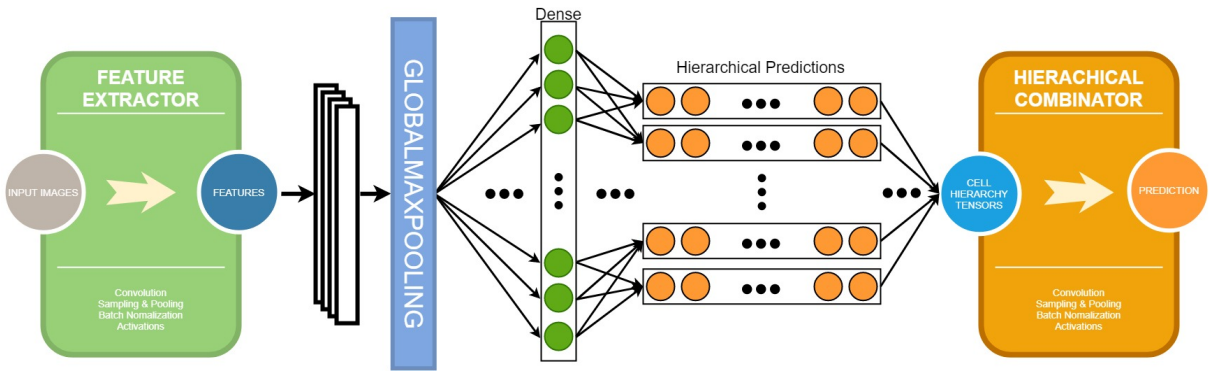

Figure 4.4: Sparsely-Connected model structure. On the right of the dense layer, it is possible to find a set of SoftMax layer that represent the Hierarchical Predictions, each at a different level of the hierarchy

are transposed in order to match the dimensions of the neural network layers.

Whenever the binary_matrices are ready, we can start creating the Hierarchical Model. As depicted in Fig. 4.4, the output encoding of the Feature Extraction is firstly linearized into a vector-shaped tensor and then fed into a Dense layer activated by a ReLU function, with the purpose of properly combining the features. Immediately after it, the hierarchical classification module takes place. As it is built in the same way for each of the hierarchical settings that we implemented, we will discuss it here. As a first operation, the Hierarchical Prediction layers are defined: they consist of a set of Dense layers in a number that is equivalent to the number of hierarchy levels (depth). Moreover, each of these layers contain a number of neurons which is equivalent to the number of world grid cells in that specific hierarchical level. These numbers can vary depending on the cell generation procedure parameters (lvl_{start}, lvl_{end}, t_{min}, t_{max}).

The main purpose of the Hierarchical Prediction layers is to predict the target cell of the current (batch of) image at their corresponding level of the hierarchy (an example can be found at Fig. 4.4). Because of their predicting nature, such Dense layers are activated by SoftMax functions.

In order to combine the predictions at different hierarchical levels, the model exploits the **binary_matrices** that we described before. The idea behind using this particular form of representing the hierarchy through matrices is due to the tensor objects of neural networks. Tensors are mathematical primitives that describe the relations between objects inside a vector space. Some subsets of tensors are vectors and scalars. When applied into the deep learning field, these objects can be easily used as synonyms of layers. Due to their numerical nature, tensors can be represented in a matrix form, therefore they can

be used as operands in any matrix operation.

As the output of the Hierarchical Prediction layers are identified by tensors of values, it is possible to combine them using matrix operations. According to this, we decided to multiply each output of the previous layer by the corresponding level's **binary_matrix**. In particular, each of the binary_matrices presents a shape of $[N_{lvl\_size}, M_{leaf\_size}]$ while hierarchical prediction tensors are shaped as $[1, N_{lvl\_size}]$. These sizes make it possible to multiply the Tensors by the corresponding level's binary_matrix in order to obtain a set of Tensors shaped as $[1, M_{leaf\_size}]$. In means of a mathematical formulation:

$$\begin{aligned} \forall i \in \{0, \ldots, K_{max\_depth}\} : \\ Tensor\_out_i = Tensor\_in_i \times binary\_matrices_i \end{aligned} \tag{4.1}$$

which is a tensor whose shape is derived as follows:

$$[1, M_{leaf\_size}] = [1, N_{lvl\_size}] \times [N_{lvl\_size}, M_{leaf\_size}]$$

After the matrix multiplication, the output tensor will be shaped as the output of the model, that is a set of K_{max_depth} tensors with equal size (number of leaf cells).

The final operation that needs to be performed before the final prediction is the combination of the cell_hierarchy tensors in order to result in a single final output layer. In order to experiment some possible variants of the hierarchical model, we decided to implement three different hierarchical model structure that differ on the operation applied over the set of cell_hierarchy tensors in order to combine them. In particular, we named them:

- **Product** of the branch nodes' probabilities (HMC);
- **Exponential sum of logarithms** of the nodes' probabilities (HMC-sol);
- **Smoothed sum** of the branch nodes' probabilities (HMC-smooth).

In the next sections, we analyze their operations in details.

4.2.2. Product of the branch nodes' probabilities (HMC)

In this section we discuss how the hierarchical prediction tensors are combined in order to produce a final tensor to be used for the final prediction. In the latter layer descriptions, we explained how we build the model's intrinsic hierarchy of cells, and we derived a set of K_{max_depth} tensors shaped as $[1, M_{leaf\_size}]$. Each of the neurons of this layer represents one of the leaf cells (from 1 to M_{leaf_size}) that are to be intended as the classification

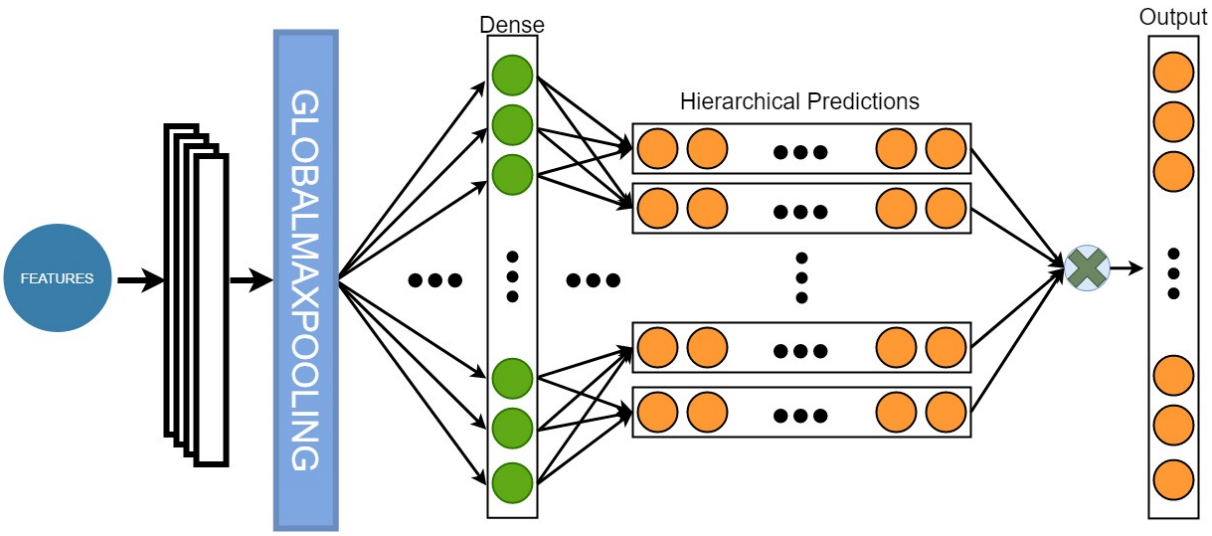

Figure 4.5: Product model structure. Each of the hierarchical prediction tensors are multiplied together in order to produce the output layer prediction.

target. In order to combine these result, the "Product of the branch nodes' probabilities", as the name states, computes the following operation

$$Tensor\_out = Tensor_{lvl_0} \cdot \ldots \cdot Tensor_{lvl_{max\_depth-1}} \tag{4.2}$$

which represents the element-wise product between all the hierarchical prediction tensors (Fig. 4.5).

This operation is not only useful to correlate probabilities of the nodes belonging to the same path from the root to the cell nodes, but also discriminating specific areas of the world with respect to others: as making a prediction on a high level of the hierarchy is by definition simpler than a deep level due to the number of cells of such layer, the multiplication of the children of a high-level node can benefit of the high-level prediction. For example, if we consider a high-level cell covering the Europe area, if it gets a high probability out of its softmaxed neuron, the multiplication with its children will get a boost benefit against cells whose parent cell is North America, that received a lower score.

In order to find out the weakness of this version of the hierarchical model, we analyzed how probability is propagated along the branches, and we found out that when the hierarchy gets too deep, the leaf probabilities can reach very low numbers because of the chained multiplication of nodes branch probabilities (floating-point numbers which are ≤ 1). This problem may raise precision issues whenever numbers get too small, producing underflow

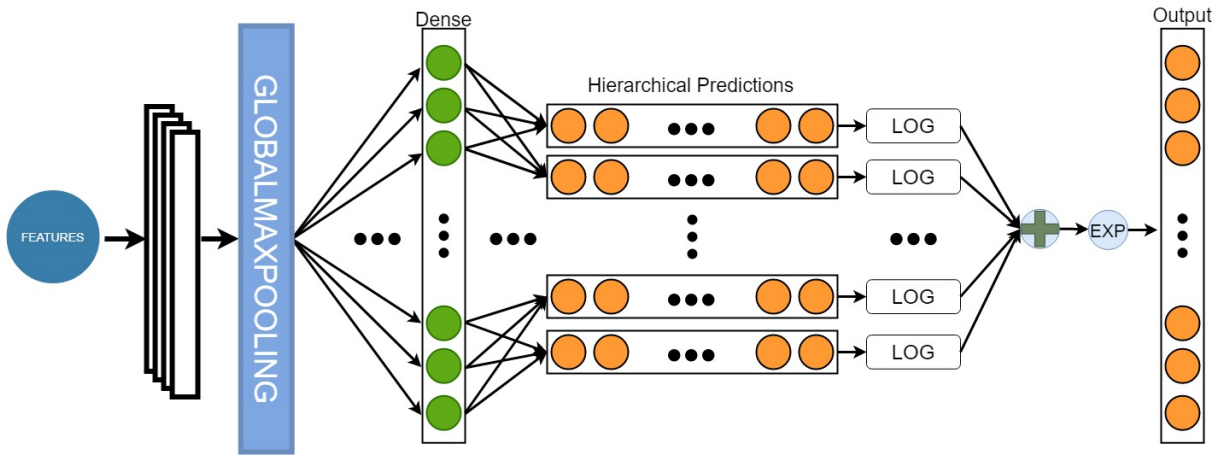

Figure 4.6: Exponential sum of logarithms model structure. Each of the hierarchical prediction tensors are passed through a log function and then element-wise summed up together. Finally, the output tensor is the exponential of that sum

of memory.

4.2.3. Exponential sum of logarithms of the branch nodes' probabilities (HMC-sol)

Underflow and precision issues can result in low performance or even in unfortunate vanishing gradient descent: when computing the gradient of the output in order to perform the neural network backpropagation, the model may obtain very small numbers that result in shrinking the gradient to zero, thus preventing the model to update its weights (which is equivalent to stop learning).

This version of the hierarchical model was designed specifically to tackle precision issues that might result from having long chains of small number multiplications. In order to work with bigger number, we thought to exploit one of the major logarithm property:

$$log(P_0 \cdot \ldots P_N) = log(P_0) + \ldots + log(P_N) \tag{4.3}$$

Therefore, we thought of replacing the product between the hierarchical prediction tensors with the sum of their natural logarithm, as in the following formula:

$$\begin{aligned} log(Tensor\_out) = log(Tensor_{lvl_0}) + \ldots + \\ log(Tensor_{lvl_{max\_depth-1}}) \end{aligned} \tag{4.4}$$

Finally, in order to obtain the same values as the product's, we raised the sum of the logarithm as the exponent of the Nepero number:

$$Tensor\_out = e^{\sum_{i=0}^{max\_depth-1} log(Tensor_{lvl_i})} \tag{4.5}$$

Although being more precise than the previous hierarchical model version, the current model's complex set of operations over the model tensors implies a longer training and inference time than the HMC. This aspect will be thoroughly analyzed in the Experiment D of section 6.

4.2.4. Smoothed sum of the branch nodes (HMC-smooth)

In order to create different versions of the hierarchical model, we analyzed other existing hierarchical settings even different fields of application, and we discovered that it is frequently used in the word generation domain. In particular, we designed a system that we named HMC-smooth that we discuss in this section. The rationale behind this model is to exploit the hierarchical properties of the world grid generation, giving more importance to cells at deep levels of the hierarchy. Instead of discriminating cells in a certain large area (e.g. Europe) against others through probability multiplications, we thought of averaging the probabilities of branch nodes, giving more importance to nodes which are close to the leaf. In order to obtain such a result, we decided to introduce a new hyperparameter α that corresponds to the smoothing factor. This value is multiplied to the entries of each of the leaf cells branches: starting from the leaf, alpha is multiplied to the corresponding layer after being raised to the power of $max\_depth - i - 1$. In this way, the powers of α are used as weights in the weighted average of the branch nodes' probabilities.

This model forced us to slightly modify the procedure of the binary_matrices generation, due to the addition of the α parameter. As it is possible to deduce from Fig. 4.7, we want to multiply each of the Hierarchical Prediction Tensor by a power of α as in the next formula:

$$\begin{gathered} \forall i \in \{0, \dots, max\_depth - 1\} : \\ Tensor\_pred_i = Tensor_{lvl_i} \cdot \alpha^{max\_depth-i-1} \end{gathered} \tag{4.6}$$

Now that the tensors have been smoothed, it is necessary to normalize them. This operation is necessary as different branches may have different length, thus it is also necessary to compute the normalizing factor of each of the leaf cell branches. In order to obtain the latter information, we decided to modify the Algorithm 2 so to compute the normalizing factor (denominator) for each branch. Therefore, we added the following lines in the for

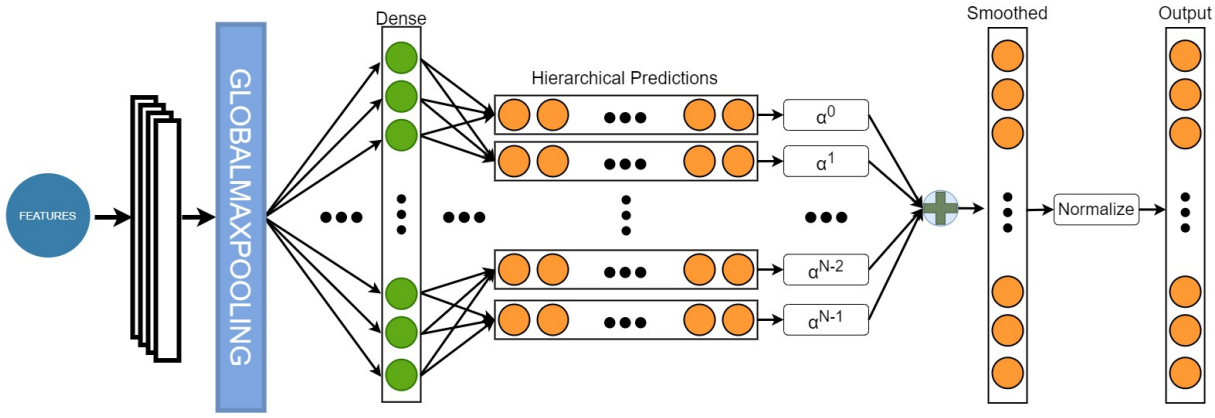

Figure 4.7: Smoothed hierarchical model structure. Each of the hierarchical prediction tensors are multiplied by the corresponding value of alpha and then summed together. In the end, a normalization operation is applied.

loop at line 4.

1: Let norm_factor = 0
2: **for** i in {0, . . . , lvl + 1} **do**
3: norm_factor = norm_factor + $\alpha^{bottom\_lvl-i-1}$ ▷ bottom_lvl is the max hierarchy depth level
4: **end for**
5: normalizing_factors.add(1 / norm_factor)

Once all the normalizing factors have been computed, it is possible to complete the Smoothed hierarchical model by summing the Hierarchical Prediction tensors together and apply an element-wise multiplication with the normalizing factors. In the end, if we are to explain the whole model functioning in a single mathematical formula, we could describe it as:

$$\forall k \in \{0, \ldots, leaf\_size\} : \\ Tensor\_out_k = \frac{\sum_{i=0}^{max\_depth-1} Tensor_{lvl_i,k} \cdot \alpha^{max\_depth-i-1}}{\sum_{j=0}^{bottom\_lvl-1} \alpha_k^{bottom\_lvl-j-1}} \tag{4.7}$$

In this way, every leaf cell can be uniquely identified by the branch it belongs to in terms of probabilities, smoothing and normalization factors.

5 | Dataset

In this section we discuss which data has been used to train and evaluate the models and the hierarchical dependency that is possible to deduce directly from data.

The goal of this work is to geolocate outdoor images of any kind (urban, natural and so on). Moreover, the outdoor geolocation involves an intrinsic heterogeneity problem: as places looks very diverse between different areas of the world (class-level heterogeneity), so do pictures coming from the same specific region: for example, if we take into consideration the Ligurian region in Italy, it is possible to find photos of the port area of Genova but also pictures that might depict the Appenini landscape of the backcountry. Moreover, pictures of houses in coastal cities, such as Rapallo, look very different from countryside barns (Fig. 5.1). In the literature [14], it is possible to find only a few datasets of geotagged pictures, mainly circumscribed to a specific area or a coerced task. As the task requires us to cover a very broad and heterogeneous set of images, we decided to crawl ourselves a dataset on the Flickr platform.

5.1. Flickr dataset

In order to tackle a worldwide geolocation problem, it is necessary to collect a very large set of images. In order to get a perfect prediction of any outdoor image, we would need to collect a photo for every possible location of the world. As this is not possible, we collected a vast set of outdoor heterogeneous places by exploiting the search function offered by the Flickr API. In fact, we decided to query for a collection of terms that can include several and remote parts of the world as well as cities and countrysides in an outdoor setting. For example, we used urban words as 'city', 'building', 'street', 'monument', 'parkway', 'fountain', 'architecture' and 'skyscraper' as well as natural ones such as 'countryside', 'landscape', 'mountain', 'woods', 'forest', 'river', 'lake', 'camping', 'beach' and so on. We also included words for specific areas of the world, such as 'pyramid', 'savanna', 'tundra', 'volcano', 'fjord', 'pagoda' and specific natural events such as 'typhoon', 'flood' and 'earthquake'. In order to obtain the largest possible set of outdoor images, we also exploited the PyDictionary and GoSlate library in order to generate respectively any

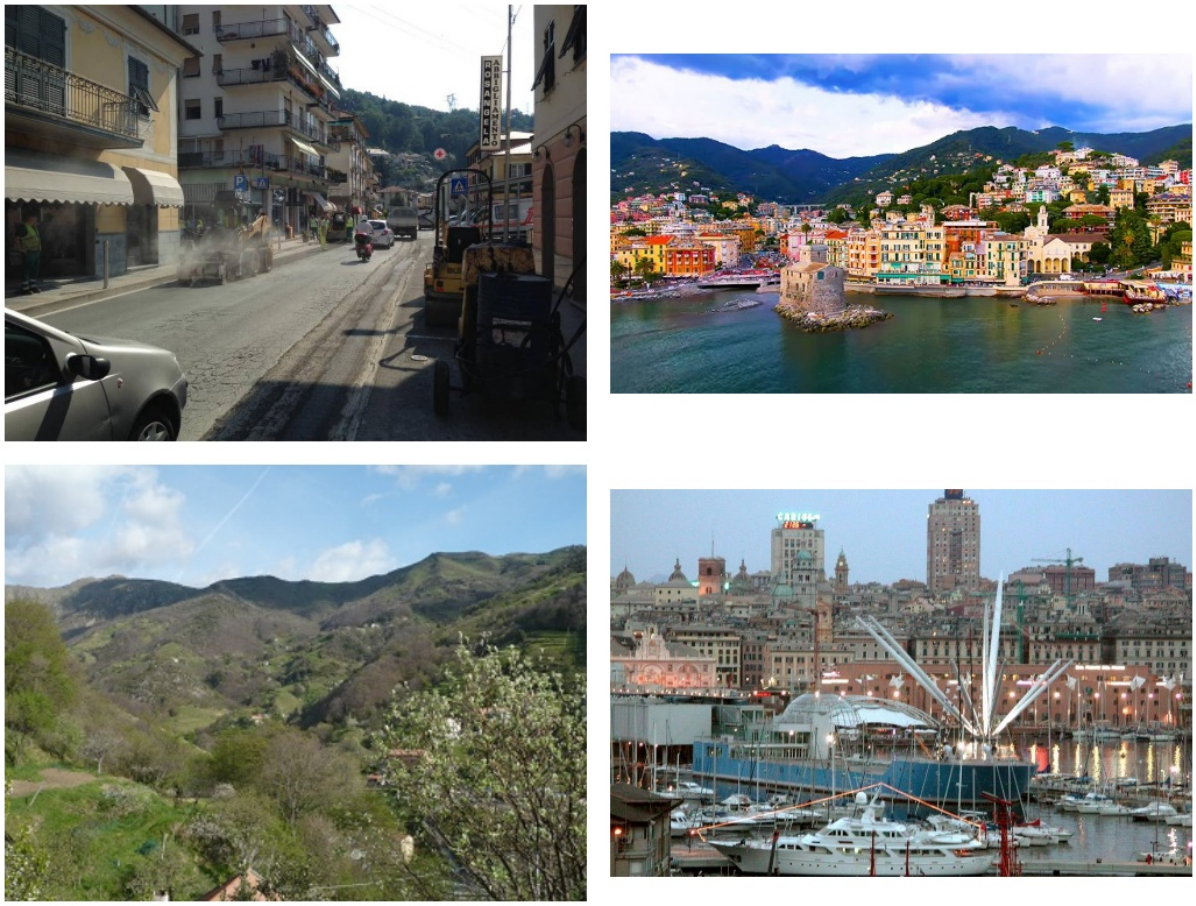

Figure 5.1: Heterogeneity of images coming from the Ligurian region (Italy).

possible English synonym and translations in a set of given languages (Italian, French, Spanish, Portuguese, German, Chinese and Swedish).

After a preprocessing cleaning of the corrupted, duplicated and wrongly sized images, we have been able to collect about one million geotagged outdoor images containing outdoor scenes of any possible kind. In particular, we were able to collect information about latitude, longitude and tags for each of the images that passed through the preprocessing operations.

5.1.1. Statistics

In order to understand how the data we collected is distributed, we performed some computations in order to retrieve statistics.
As a first operation, we decided to work on the image tags. In particular, we grouped them under two main categories: urban and natural. Results can be found in the Table 5.1.

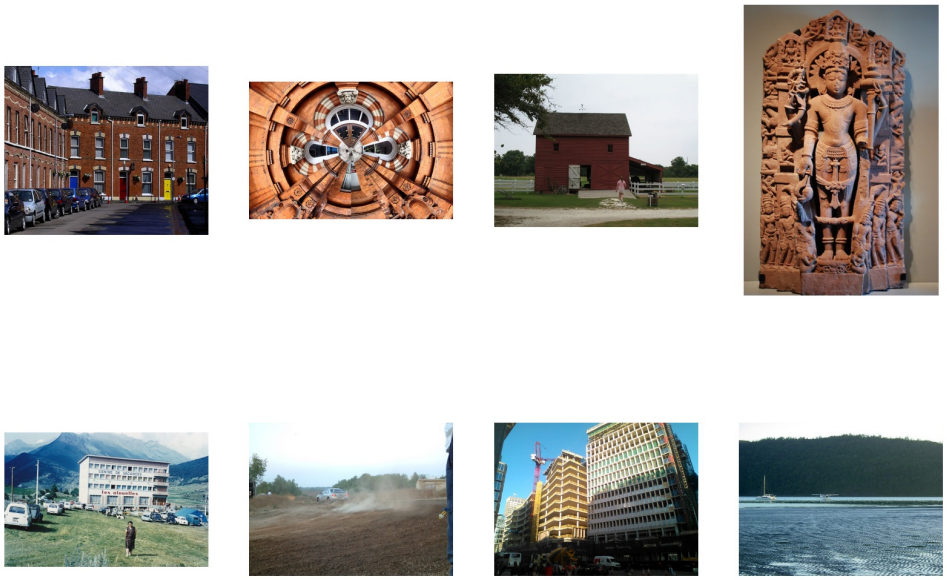

Figure 5.2: An example of eight outdoor images coming from the Flickr dataset we crawled for training and testing. In particular, these images are used for the testing phase.

Type	Number of images
Natural	476,960
Urban	493,605
Total	970,565

Table 5.1: Image tags type and distribution.

In the end, we put an example of how data is distributed in a possible setting of the world grid, more specifically with 72 cells containing between 3000 and 30000 images each (Tab. 5.2). In particular, we counted the number of grid's cells and images in each continent.

Continent	Number of cells	Number of images
Africa	2	22,163
Americas	35	440,200
Antarctica	1	6,663
Asia	5	52,050
Australia	2	21,575
Europe	27	378,424
Total	72	926,053

Table 5.2: Image location distribution.

The number of total images differs from the previous table because of the preprocessing operations that we discussed in the Section 2.3.

This statistical analysis allows us to understand where data is located in order to know where to crawl new images. In the case in which we would like to cover more uncovered sections of the Earth, we could look for tags or queries that are associated to those specific areas of the planet.

5.1.2. Hierarchical features

In order to understand how the problem could have been faced, we decided to analyze some sample images in the Flickr dataset that we described previously.

From a qualitative analysis, we found that pictures of the same areas share common features. For example, it is possible to find similar road signs in almost any part of North America (USA). The same holds for road signs pictures coming from Europe (more particularly in Italy). In fact, from the Fig. 5.3, it is possible to deduce that American road signs use to have a specific font, shape and size, while Italian signs are different.

Moreover, all the indicator signs are green, while speed limit signs are white, while European signs tend to have a different font from American's and different colors, which relate to the indicated content inside the sign itself.

This was just an example to show how discriminative features could be: if one of our model would be able to extract the road sign features, it could immediately detect that the picture was taken in America, thus it could give more relevance to all the American cells, discriminating other world location areas. Any kind of object, shape or line could be unique to specific areas of the world, both at continent, country and even city or sub-city

Figure 5.3: On the left side, there are four Italian road sign pictures, whereas on the right side there are four American road sign images.

level. Therefore, it is possible to find a discriminative feature at any depth of the hierarchy, and this can be an outstanding improvement in terms of classification performance.

For this reason, we decide to make use of the hierarchical information about features, and we introduce this knowledge directly inside our hierarchical models. A huge impact that deep learning models make in this field of application is that they can automatically determine and find features without any cues from developers: therefore, the discriminative information from the road sign example that we previously described (font, color, shape and so on) are automatically learned by the model that autonomously creates an internal representation and is able to detect them inside any picture.

6 | Experimental Results and Evaluations

In this section, we discuss the experiments and evaluations that were performed on the deep learning models that we explained in the previous sections. In particular, we discuss how we analyze the outdoor geolocation problem and compare the model performance in different conditions. In order to provide a structured approach, we decide to follow a common pattern for all experiments and analyses:

- **Problem description**: which problem is going to be analyzed, which models will be involved and what we want to prove with the experiment;
- **Method**: how we study the problem and how we face it;
- **Results and Discussions**: what comes out from the experiment and what we can deduce and considerate from the results.

6.1. Experimental Setup

As a further notice, it is important to underline some general information that is common in every all the experiments we illustrate. In particular, we exploit a single server machine endowed with two Graphic Processing Units with the following characteristics: an Nvidia RTX 2080 Ti, that was used mainly to train the outdoor geolocation models and a secondary Nvidia RTX 2070 that was mainly used for inference and running demonstrations.

6.1.1. Data

As far as the dataset is concerned, all the outdoor geolocation models that we will be dealing with are trained over the Flickr images that we personally crawled as thoroughly explained in Chapter 5. In particular, we automatically split the one million image dataset into two separated set: training (90%) and validation (10%). The test set is extracted

offline.

During the model training, we feed the neural network with batches of 32 random images. In order to generate the batch, we define a **Data Generator** which is responsible for randomly picking the desired number of images and apply a set of random transformation (data augmentation).

Data Augmentation

In order to provide more training data to the model, we implement a set of operations that manipulate the images of our dataset. With a probability draw of 50%, the DataGenerator decides whether to apply one of the following transformation to a certain image:

- Rotate: rotates the image of a multiple of 90 degrees;
- Flip: randomly flips the image over the horizontal and vertical axes;
- ColorAugmentation: changes image brightness, contrast and saturation.

All these operations have been performed over vectorized images using the TensorFlow built-in functions.

6.1.2. Loss function and optimizer

When dealing with the training of the models we illustrate, we opted to use the SparseCategoricalCrossEntropy, a loss function that represents the state-of-the-art loss in the classification task. In particular, in its Sparse configuration, this loss function enables designers to specify each target class as an integer value (from 0 to $N_{classes} - 1$). As far as the outdoor geolocation task is concerned, the integer values represent a numbering of the leaf cells we predict as class targets. In mathematical terms, the CategoricalCrossEntropy can be expressed as:

$$CCE(x, y) = -\sum_{i=0}^{N_{classes}} y_i * log(x_i) \tag{6.1}$$

where x_i represents the model prediction and y_i stands for the ground truth label for a specific input image.

In order to compile the models for training, we have to define which optimizer could be the best fit for the training loss we previously described. The state-of-the-art in the deep learning field offers the possibility to choose within a broad number of pre-implemented optimizers, such as AdaGrad, AdaDelta, RMSprop, SGD and so on. After

several attempts, we found the Adam function as the best optimizer for the outdoor geolocation task. The Adam optimizer is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. Moreover, it is computationally efficient and requires little memory, which makes it very good for large and complex models [12].

6.1.3. Regularization functions

Deep learning models are subjected to the overfitting phenomenon: this circumstance happens when the neural network learns all the embedded information in the training set up to a certain level that does not allow it to generalize new images. In order to recognize and tackle this problem, we applied cross-validation techniques, and we found out that this phenomenon started to happen after a few epochs. For this reason, we decide to introduce some regularization functions that enables us to control the pace of learning of the model so to avoid the model overfitting. In order to cope with this model training issue, we decide to employ a set of functions that could help the neural networks to speed up and enhance the training procedure.

Early Stopping

The goal of the Early Stopping [3] is to find the optimal number of epochs for training before overfitting. The deep learning training is performed by submitting the whole dataset to the model as many times as it needs in order to learn the correct classification of all the images. Every round in which this operation is called epoch. When the training phase takes too few epochs, the model is not able to learn enough information on the dataset, therefore it requires a larger number of dataset submissions. On the other hand, if the dataset is set forth too many times, the model will be overfitting, which means that it will learn the training set so accurately that it will not be able to generalize to different images. The solution to this tuning is the Early Stopping, an overfitting prevention technique that stops the training after a number of consecutive epochs (patience) in which a specified learning parameter is not improving. In our implementation, we set the "Validation Loss" as the learning parameter, with patience of 3 epochs.

Reduce LR on plateau

The Adaptive Learning Rate is a special callback function which aims to adapt the learning rate of the model to its learning curve. In the ReduceLROnPlateau implementation, this function aims to gradually reduce the learning rate whenever a monitored learning

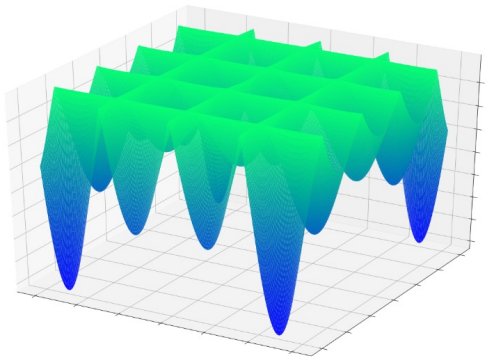

Figure 6.1: An oddly shaped function with a various number of minima.[1]

parameter will sit inside a restricted set of values (thresholds). Therefore, after expressing the monitored parameter, the patience and the reducing factor the algorithm will be able to adapt the learning rate in order to follow the optimal learning curve. The rationale behind this function can be identified in the following description: machine learning problems are optimization problems in which models aim to find the absolute minimum of a loss functions. As deep learning problems are identified by very complex and oddly shaped loss functions, they could be endowed with many local minima that could mislead the optimization algorithm when finding the real absolute minimum. In particular, optimizers follow the loss gradient descent at a given pace which is dictated by the learning rate. If this rate happens to trap the optimizer inside a local minimum, it will not be able to escape the concave pits of the function as the gradient descent will bounce between the two sides of the minimum. Therefore, the model will be unable to find the absolute minimum. By modifying the value of the learning rate, the deep learning model is able to avoid this issue.

As far as our geolocation models are concerned, we set the Validation Loss as the monitored parameter, patience = 3 and factor = 0.2.

Dropout

The Dropout [24] is a deep learning technique that enables engineers to switch off certain neurons during the training phase with a given probability, thus preventing the model overfitting. This procedure is able to slow down the pace of learning of a layer's neurons by creating binary masks that are applied to the immediately previous layer which switch off the neurons corresponding to zero values. In our implementation, we placed a Dropout

[1]Source: `https://www.allaboutcircuits.com/technical-articles/understanding-local-minima-in-neural-network-training/`

layer immediately after the first Dense layer of the Cell classifier, with a probability of switching off neurons of 0.5. Moreover, the EfficientNet implementation that we are exploiting throughout this section's experiments has a tunable Dropout setting (some Dropout layers are placed inside the feature extractor architecture) and we set it at 0.4 probability.

Weight Decay

The Weight Decay (also known as Ridge Regression) is an overfitting prevention technique that aims to penalize complexity: the aim of this technique is to include the number of parameters of a certain model (or layer) directly inside the loss function, in order to penalize models with many trainable weights. This procedure goes along with the learning rate and has proven to prevent overfitting and improve generalization [13]. Moreover, this function is tunable through a λ parameter, that represents the discriminating factor of the model complexity.

$$w_{new} = argmin_w \sum_{n=1}^{N} (t_n - g(x_n|w)) + \lambda * \sum_{q=1}^{Q} (w_q)^2 \tag{6.2}$$

The first summation represent the model fitting, while the second is the regularization part that is responsible for balancing the fitting of the training set (overfitting prevention).

The optimal λ value can be found through a standard grid search technique. For this particular set of experiments, we set it to 10^{-5}.

Checkpoint callback

This callback is way simpler than the previous ones: its only purpose is to save the model checkpoints after each iteration, so to either saving the history of the model weights evolution during the training phase or the best model weights so far. In our setting, we decided to keep only the best weights in terms of validation loss performance.

6.1.4. Evaluation Metrics

In order to compare the models' performance within the experiments, we need to define a set of quantitative evaluation metrics: they are meant to analyze how the model would behave when generalizing to new images with respect to a random classifier (Improvement over Random) and to compare each other's performance on the same task (Validation performance). We do not use other traditional machine learning evaluation metrics, such

as Recall or Macro Average, as regions are relatively balanced in terms of number of samples thanks to the cell generation algorithm.

Validation Loss and Accuracy

A good practice that engineers do when training a machine learning model is to evaluate the generalization performance after every epoch of training. This operation allows us to measure the learning trend of the model on a different set of data that was not submitted to the neural network training. For this reason, we compute the following pair of metrics:

- The **Validation Loss**;
- The **Validation Accuracy**.

As far as the Validation Loss is concerned, by exploiting the same loss function we use for training, we compute the SparseCategoricalCrossEntropy over a separated set of images (Validation set). This procedure mimics the behavior of the true model performance on unknown data. Moreover, the validation loss is able to tell us additional information on the model learning procedure. In particular, if the training loss gets lower than the validation loss, it means that the model is overfitting: the model learns the training dataset features too much, and it is not able to generalize to new images (in this case represented by the validation set). As far as the Validation accuracy is concerned, we compute the ratio of correctly classified images of the validation set.

Improvement over Random (IoR)

By far, the outdoor image classification task that we create involves numerous target cells. This leads the model accuracy implicitly to drop to small floating-point values as soon as the number of target cells start to grow. After a certain target cells, it gets really difficult to understand what the accuracy stands for and even more to compare different model values. In order to make results more interpretable, we design a new evaluation metric that is able to compare the model accuracy with a random classifier over the same classification task. Hence, we introduced the IoR. The IoR acronym stands for Improvement over Random and represents how many times the model is performing better than the random prediction, as shown in the next formula:

$$IoR = \frac{Accuracy_{valid}}{Prob_{Random}} \tag{6.3}$$

The way the random classifier probability is computed is a uniform probability distribution over the target cells of the outdoor geolocation problem ($Prob_{Random} = \frac{1}{N_{classes}}$).

6.2. Experiment A: Model Backbone

As the first experiment, we want to investigate which state-of-the-art CNN model could be the best feature extractor for the Outdoor Geolocation problem. For this reason, we compare a set of well-known encoders over the same task, and we analyze their performance.

6.2.1. Problem Description

This experiment aims to analyze the training and validation performance of different settings of the feature extractor part of the outdoor geolocation model. In particular, we compare the same cell classifier (Multi-Class classifier) with different feature encoders. In particular:

- EfficientNetB0;
- EfficientNetB1;
- VGG19;
- ResNet152;

In order to show the magnitude of the backbone models, we provide a table (Tab. 6.1) that illustrates the number of parameter and layers of each of the previously mentioned models.

Backbone Name	# Layers	# Parameters
VGG19	26	143,667,240
ResNet152	152	60,419,944
EfficientNetB0	237	5,330,571
EfficientNetB1	253	7,856,239

Table 6.1: Feature extractor layers and number of parameters.

The goal of this experiment is to find the best feature encoder/extractor for the outdoor geolocation model. This definition is not only applied to the classification performance, such as loss and accuracy, but also related to the training time and the number of epochs to converge.

6.2.2. Method

In order to compare the performance of the feature extractor we enlisted previously, we decided to keep the same cell classifier, which was a non-hierarchical multi-class classifier. Moreover, all the feature extractors that we test have been kept completely frozen for all the experiments. Therefore, no encoding layers have been trained. The classification task solved by all models was performed over 99 classes, with a single softmax layer after the GlobalMaxPooling of the feature extractors.

6.2.3. Results and Discussions

In this section, we show a comparison between the model performance over the same classification task. In particular, we analyze the validation loss, accuracy and IoR. It is important to notice that all the feature extractors have been kept untrained throughout the training phase: all the layers were initialized with their pre-trained version of ImageNet [4], and kept frozen for the whole duration of the training.

Backbone Name	Validation Loss	Validation Accuracy	IoR
VGG19	6.985	0.015	1.485
ResNet152	6.324	0.020	1.980
EfficientNetB0	5.158	0.023	2.277
EfficientNetB1	4.607	0.026	2.574

Table 6.2: Feature extractor validation performance.

This experiment demonstrates that the EfficientNet feature extractor was the most adapt for the image outdoor geolocation problem among the state-of-the-art CNN encoders. From the Tab. 6.2, it is possible to deduce how the EfficientNet backbone outperforms all its competitors. In particular, the Keras EfficientNetB1 implementation showed by far the best performance among the tested networks. Although more complex implementations such as EfficentNetB4 and EfficentNetB7 have been tested, the EfficientNetB1 demonstrates to outperform its counterparts. In some cases, the network was too complex that it was not even able to reduce the training loss (not learning), thus we decided to avoid showing such a performance.

6.3. Experiment B: Number of Training Images

Now that the best model encoder has been identified, we move on to study the Outdoor Geolocation problem more in details. The current experiment is intended to study the trend of the Outdoor Geolocation Model performance when increasing the number of images in the training dataset. Such an experiment helps us in understanding whether the number of images that we are submitting to the model training are sufficient and if there is still a performance growth margin.

6.3.1. Problem Description

The need for improving the model performance is crucial in the outdoor geolocation problem. A classifier which is able to detect the correct location of an image, it's the key in order to produce a good model for this task. From the theory of deep learning, we know that increasing the number of samples in the training dataset leads to better results up to a certain asymptote, that is set intrinsically in the problem description. For this reason, we wanted to create an experiment that is able to show how the performance trend of an outdoor geolocation model can be related to the number of training images that are used for training it. The model used for this experiment is a Multi-class classifier, with learning rate set at 10^{-3} and a number of epochs set to 10.

The goal of this experiment is to prove that the performance trend of an outdoor geolocation model is positive and increases its slope when the number of images grows. In particular, we want to see if the trend tends to keep the same slant, or it will converge at a certain point.

6.3.2. Method

Concerning the current experiment methodology and approach, we formulated the problem by defining the constant parameters for the Cell generation algorithm (Alg.1) as $t_{min} = 2000$ $t_{max} = 20000$. This setting produced 89 classes, and the outdoor geolocation model has been shaped accordingly (89 softmax neurons for the final layer). In order to study the previously explained trend, we extract a fixed number of images from the Flickr Dataset. In particular, in an effort to keep a balanced set of classes, we drew the same amount of images from each of the 89 target cells, and we filled the remaining slots with randomly drawn images.

$$N_{images\_per\_class} = round(\frac{N_{images\_needed}}{N_{classes}}) \tag{6.4}$$

6.3.3. Results and Discussions

In order to have a sufficient number of samples for analyzing the performance trend, we choose to extract different numbers of training image (10,000 - 40,000 - 160,000 - 640,000) for the current experiment. As a further analysis, we also want to include the IoR - Improvement over Random - of the current tests (random guess = 0.0112). The results can be found in the following table.

# Images	Validation Loss	Validation Accuracy	IoR
10,000	12.429	0.006	0.534
40,000	6.170	0.009	0.828
160,000	5.762	0.011	0.935
640,000	5.283	0.019	1.655

Table 6.3: Validation performance and IoR (Improvement over Random) of the non-hierarchical model to vary the number of target classes.

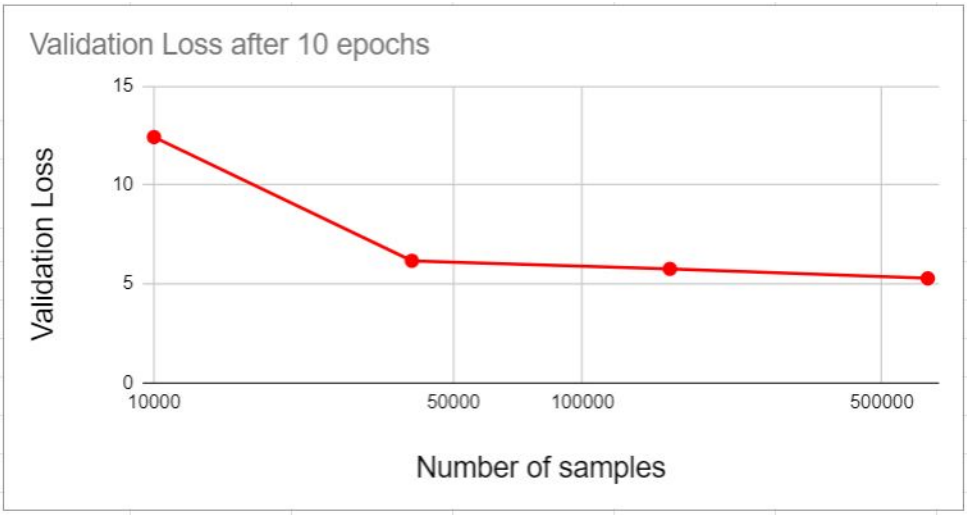

Figure 6.2: Validation Loss trend of the model related to the number of images, in a logarithmic scale.

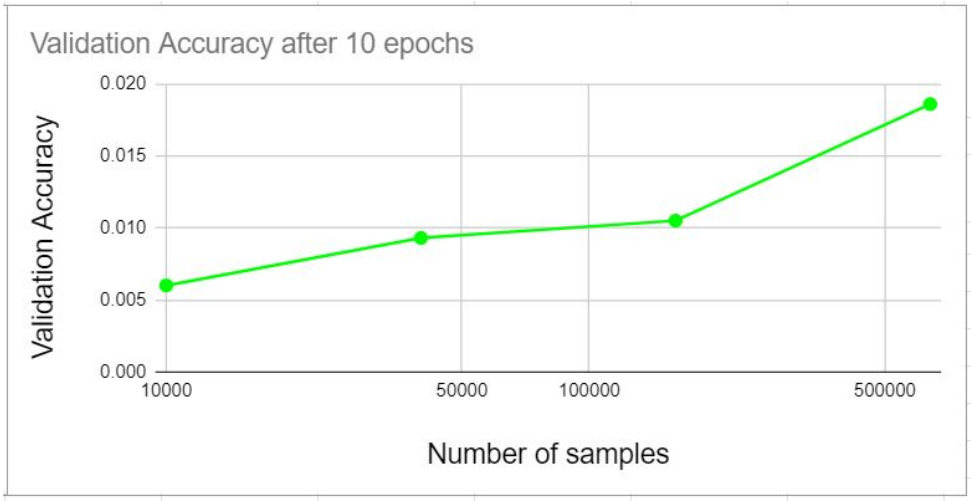

Figure 6.3: Validation Accuracy trend of the model related to the number of images, in a logarithmic scale.

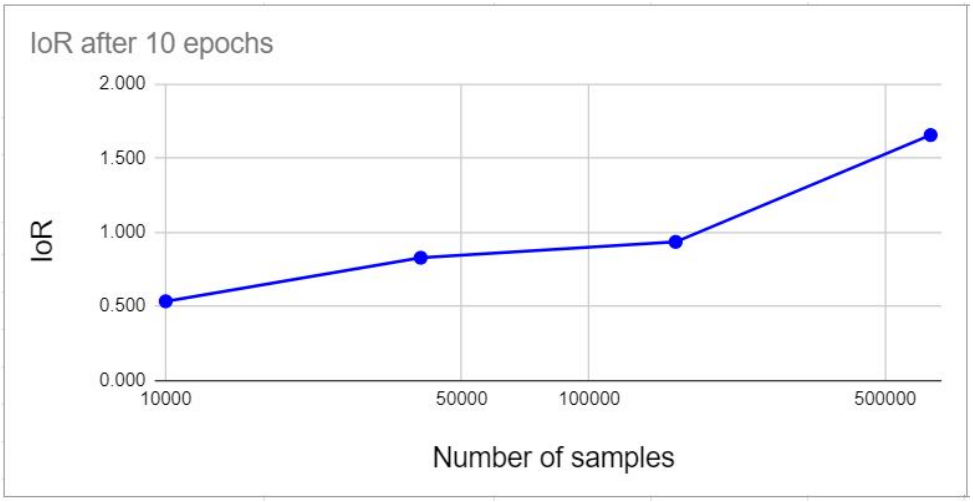

Figure 6.4: IoR (Improvement over Random) trend of the model related to the number of images, in a logarithmic scale.

From the previous results, we are able to answer the questions that we posed in the problem description and goal section of this experiment: as Fig. 6.3 shows, plotting validation accuracy versus dataset size with logarithmic access shows a steep slope except from a plateau in the middle segment. This behavior demonstrates that the accuracy will not converge even after 640,000 images. For this reason, we can assume that increasing the number of images will result an exponential increase up to a point until we see decay after the base error rate in the validation accuracy (the curve show in Fig. 6.3 is logarithmic, thus linear trends are actually exponential in the linear plane).

In the meantime, Fig. 6.2 shows how the validation loss starts steeply decreasing until converging in a plateau after around 50,000 training images. This result seems to prove that the validation loss will not improve much further with an increase in the number of images, even though small decrease in validation loss is associated to large improvements in the validation accuracy.

Finally, Tab. 6.3 shows that the IoR metric improves a lot when the number of image is increasing (Fig. 6.4): especially from 160,000 to 640,000 images, the IoR is almost doubled, and this trend seems to behave accordingly with more pictures.

6.4. Experiment C: Number of Target Cells

From the previous experiment, we were able to see that increasing the number of images is a good way of improving the model performance. Now, we want to investigate the effect that the number of target classes might have on the Outdoor Geolocation problem. In particular, experiment C is intended to study the trends of the Outdoor Geolocation Model performance increasing the number of leaf cells (target labels).

6.4.1. Problem Description

This experiment aims to analyze the models' behavior when varying the number of classes in the classification task: all the tests will be performed over the same Multi-class classifier of the previous experiment, which will only be adapted for classifying over the specific number of target classes (leaf cells). Finally, we used the same settings as Experiment B's, except from the number of epochs, which was set to 5.

The goal of this experiment is to prove that the model will produce similar relative performance with respect to the random prediction. In details, we want to show the performance trend rather than single value, as we believe that classification tasks with a lot of classes will not worsen the performance when compared to the random prediction.

6.4.2. Method

In order to run the following experiment, we needed to create several world-grid configurations. This was achieved by varying the t_{min} and t_{max} threshold parameters in the cell generation algorithm (Algorithm 1). In particular, we defined the following settings:

# Classes	t_{min}	t_{max}	# Images (%)
4	80,000	350,000	72.6
14	25,000	100,000	87.4
50	5,000	40,000	93.9
110	3,000	15,000	87.7

Table 6.4: Cell generation algorithm parameters and the percentage of images involved.

We would like to remember that the total number of images in our Flickr Dataset is 970565 and the percentages inside the table are related to this number.

6.4.3. Results and Discussions

The results we achieved with the previously enlisted settings can be found in the next table.

# Classes	Validation Loss	Validation Accuracy
4	1.554	0.278
14	2.936	0.091
50	4.926	0.031
110	5.537	0.013

Table 6.5: Validation performance of the non-hierarchical model, increasing the number of target classes.

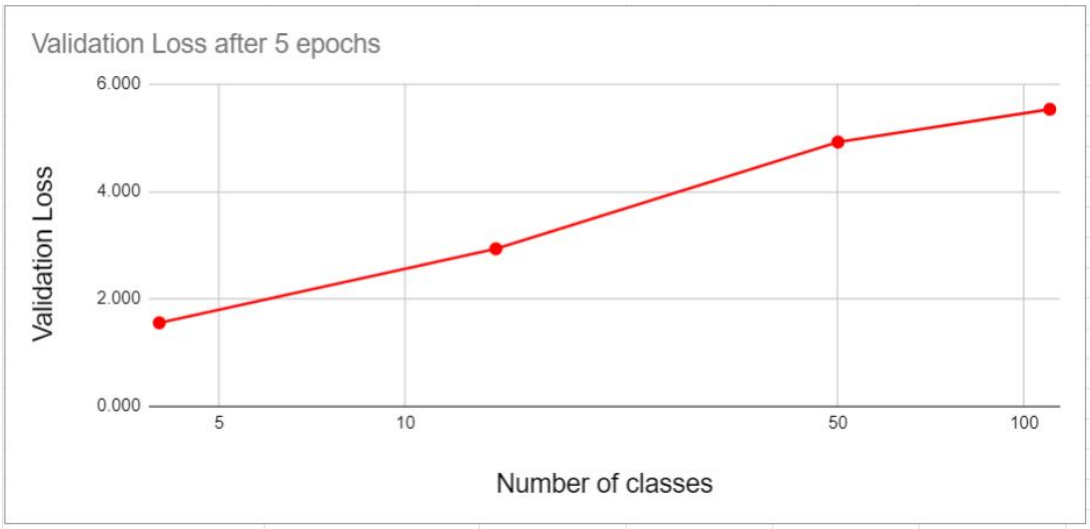

Figure 6.5: Validation Loss trend of the model related to the number of classes, in a logarithmic scale.

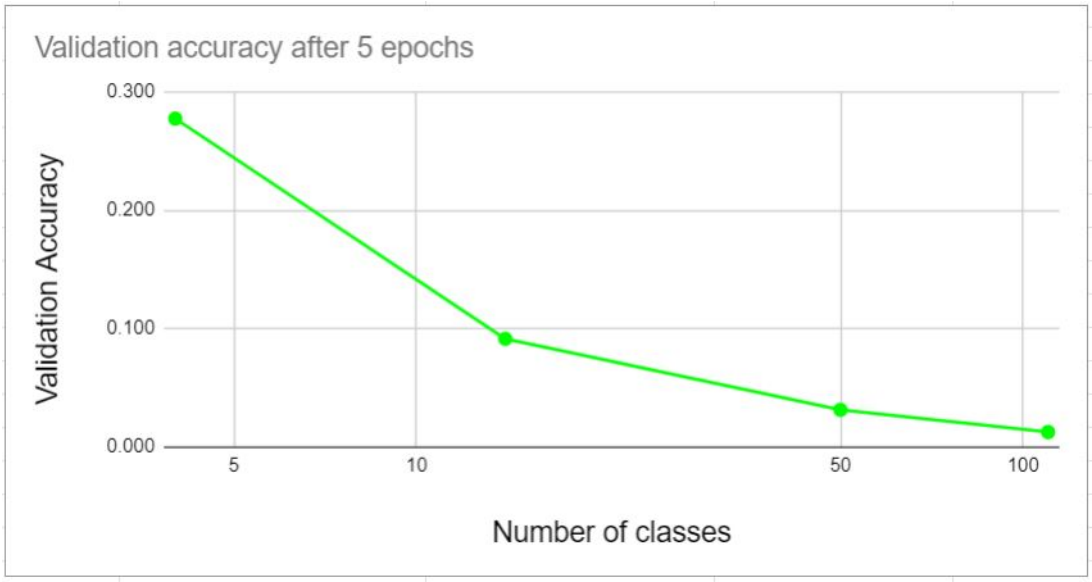

Figure 6.6: Validation Accuracy trend of the model related to the number of classes, in a logarithmic scale.

# Classes	Random probability	IoR
4	0.250	1.11
14	0.071	1.28
50	0.020	1.57
110	0.009	1.38

Table 6.6: Random probability and IoR of the non-hierarchical model to vary the number of target classes.

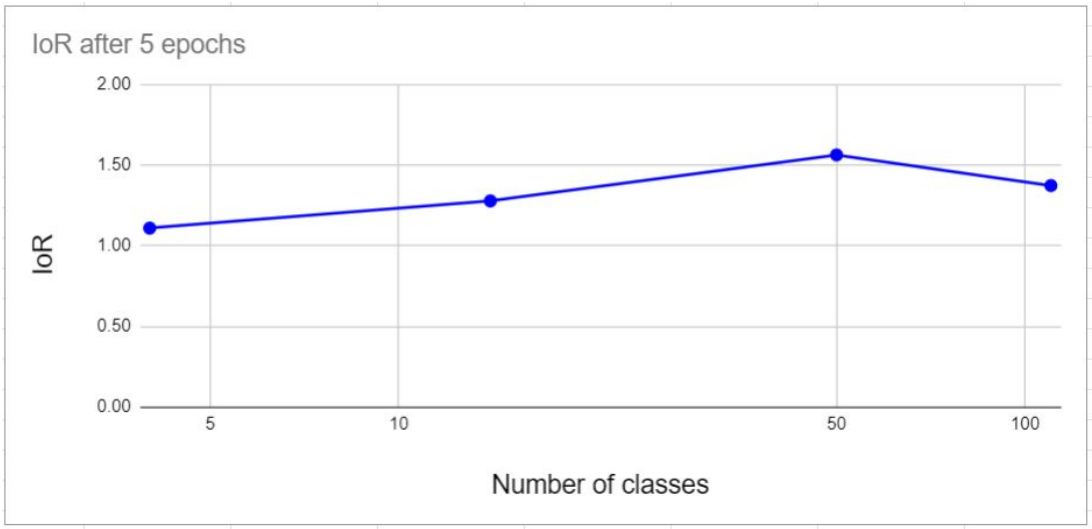

Figure 6.7: IoR (Improvement over Random) trend of the model related to the number of classes, in a logarithmic scale.

From the previous results, we are able to analyze the behavior of the model in the image outdoor geolocation problem by varying the number of target classes (leaf cells). In particular, Tab. 6.5 and Fig. 6.6 and Fig. 6.5 show the model validation performance: the trends that are visible in the figures in a logarithmic scale (base 4) show a reasonable drop of the validation accuracy and a significant increase of the validation loss. The trends are linear for the central sections and get on a plateau when reaching the highest number of classes: this demonstrates that after a certain number of targets, the model performance will stop dropping steeply, but they will remain constant around the same values.

As far as the IoR metric is concerned, the Tab. 6.6 shows how the performance is kept almost stable for all the given configurations. When the number of cells is low, the model seem to have a low IoR probably due to underfitting: the model is underestimating the problem. On the other hand, when the number of cells is high enough (over 50) the model's IoR starts dropping and the reason for this can be found in the overfitting

problem, as there are not enough images for enabling the model to be trained over that number of classes. Hence, the model cannot learn better than it is doing, unless the number of training images would be increased. An important aspect that needs to be highlighted is due to the IoR (Fig. 6.7) related to the percentage of images involved: when the training image number increases, so the IoR does. One conjecture that we make is that by increasing the number of images allows the outdoor geolocation model to learn better features, as we deduced from the previous experiment.

6.5. Experiment D: Hierarchical Model

Now that the Outdoor Geolocation problem has been analyzed, we want to compare the hierarchical models' performance against a baseline non-hierarchical multi-class model: we will compare them in terms of validation performance and training time. From an engineering point of view, the latter is important to determine whether the model performance variation is worth the time it takes to train the models.

6.5.1. Problem Description

In the Experiment D, we want to compare the performance between a standard non-hierarchical multi-class model against the performance we obtain following our novel approach. With performance, we do not only include accuracy and loss but also the training time. The list of the models that will be compared is:

- Multi-Class (**MC**) classifier;
- Hierarchical Multi-Class (**HMC**) classifier;
- Hierarchical Multi-class with sum-of-logarithms (**HMC-sol**) classifier;
- Hierarchical Multi-class with smoothing of probabilities (**HMC-smooth**) classifier.

The structure each of these models can be found in section 4.2.

The goal of the current experiment is to prove that hierarchical geolocation models can be more powerful than standard fully-connected models. The aim is to compare the results of each of the models over the test set, and also to see how much impact the hierarchical geolocation approach shows in terms of training and inference time.

6.5.2. Method

In order to compare the previously explained performance indicators, we decided to run each model singularly in their best possible settings. The best hyperparameter configuration can be found in Tab. 6.7.

Model name	Learning Rate	Epochs	Smoothing (α)	Time per Epoch (s)
MC	10^{-4}	4	-	9,256
HMC	10^{-2}	5	-	10,253
HMC-sol	10^{-4}	10	-	15,960
HMC-smooth	10^{-6}	4	0.9	19,253

Table 6.7: Models' best settings in terms of hyperparameters. The number of epochs is determined by the Early Stopping. The Time per Epoch is the average time that we measured throughout the training.

For comparing the models, we run an experiment for each of them by training over the whole dataset with the same TensorFlow seed(42) so to obtain reproducible results.

6.5.3. Results and Discussions

In this section, we provide the comparison of the model results over the same dataset.

Model name	Validation Loss	Validation Accuracy	IoR
MC	4.799	0.023	1.66
HMC	4.301	0.033	2.38
HMC-sol	4.104	0.032	2.23
HMC-smooth	3.980	0.031	2.23

Table 6.8: Models validation performance and IoR (Improvement over Random).

From the previous results, it is possible to deduce several considerations. As far as the validation performance is concerned (Tab. 6.8), it is easy to denote how hierarchical models outclass the non-hierarchical one: both in terms of validation loss and accuracy, the performance of the hierarchical model is a noticeable improvement from the MC. In particular, the HMC-smooth model shows the best performance in terms of validation loss, while the HMC stands out as far as the validation accuracy is concerned. However,

this results must be correlated to the time it takes to train each of the previously enlisted models. In particular, it is easy to see how the non-hierarchical model is faster than its hierarchical counterparts, especially concerning models with complex operations (HMC-sol, HMC-smooth). In fact, the HMC-smooth takes on average more than twice the training time than the MC, but there is an exception: the HMC is only 1.09x slower than the MC.

These results demonstrate that the HMC is better than any other analyzed model, due to its training speed (very close to the multi-class model) and validation accuracy.

6.6. Analysis A: Model Calibration

From the previous experiment, we were able to find which model is the best among the one we proposed (HMC). With this analysis, we want to investigate the correctness and calibration of its prediction through traditional machine learning techniques. Moreover, we are also going to look at the Confusion Matrix for further evaluations.

6.6.1. Problem Description

In order to measure the quality of the predictions provided by a certain machine learning model, it is necessary to compare the confidence scores it produced with the actual outcome probabilities of the target classes. Moreover, we would like to check whether the model could be biased towards the prediction of a particular set of target cells. This can be checked through a confusion matrix.

Calibration Metric

The calibration metric is a model evaluation measure that enables engineers to understand how well uncertainty is being measured by the model. In other terms, this evaluation metric is able to determine whether a model's predicted probabilities of outcomes reflect true probabilities of those outcomes [2, 18].

Confusion Matrix

The confusion matrix is a visual representation of the comparison between the predicted classes of a test dataset and the corresponding ground truth labels. In order to produce such a result, we set the true class on the ordinate axis and the predicted class on the abscissa axis: if the confusion matrix shows high numbers on the diagonal and low numbers elsewhere, it means that the model is predicting correctly almost all images. In other cases,

when a specific column or row shows high values, it means that the model is trying to predict that class more frequently than necessary.

With this analysis, we want to test our deep learning models in order to check if they are either under confident or overconfident in their prediction, entailing an unbalance between the predicted probability and the accuracy of the classifier on such predictions. Moreover, we want to check for unbalanced predictions that can be biased towards a subset of output classes.

6.6.2. Method

In order to measure uncertainty, we employ some specific diagrams that are able to plot the comparison between the confidence values of the model predictions against the expected accuracy of the same model on the same test images. Moreover, we plot the histogram bars of the predicted confidence with respect to the number of images for the same levels. In the end, we show the confusion matrix of the outdoor geolocation model, in order to check whether some target cells are predicted more predominantly than others, maybe due to class unbalance.

6.6.3. Results and Discussions

In this experiment, we produced three different plots:

- In the first plot (Fig. 6.8), the calibration matrix is shown. It is provided through a reliability diagram, in which we set the number of bins at 20. The plot is provided by the work of [11]. The ECE stands for Expected Calibration Error and indicates a summary of the plot discrepancy between the two axis. In other words, it's a measurement of the gaps across all bins, weighed by the number of examples in each bin;
- The second plot (Fig. 6.9) represents the histogram bars for the predicted confidence with respect to the number of examples for each level. The dashed line is the average confidence, while the full-line stands for the accuracy. The closer they are, the better the model is calibrated;
- The last plot (Fig. 6.10) represents the confusion matrix for the outdoor geolocation model. The colors represent the values of each of the matrix cells: bright colors stand for high values, while dark colors indicate low values.

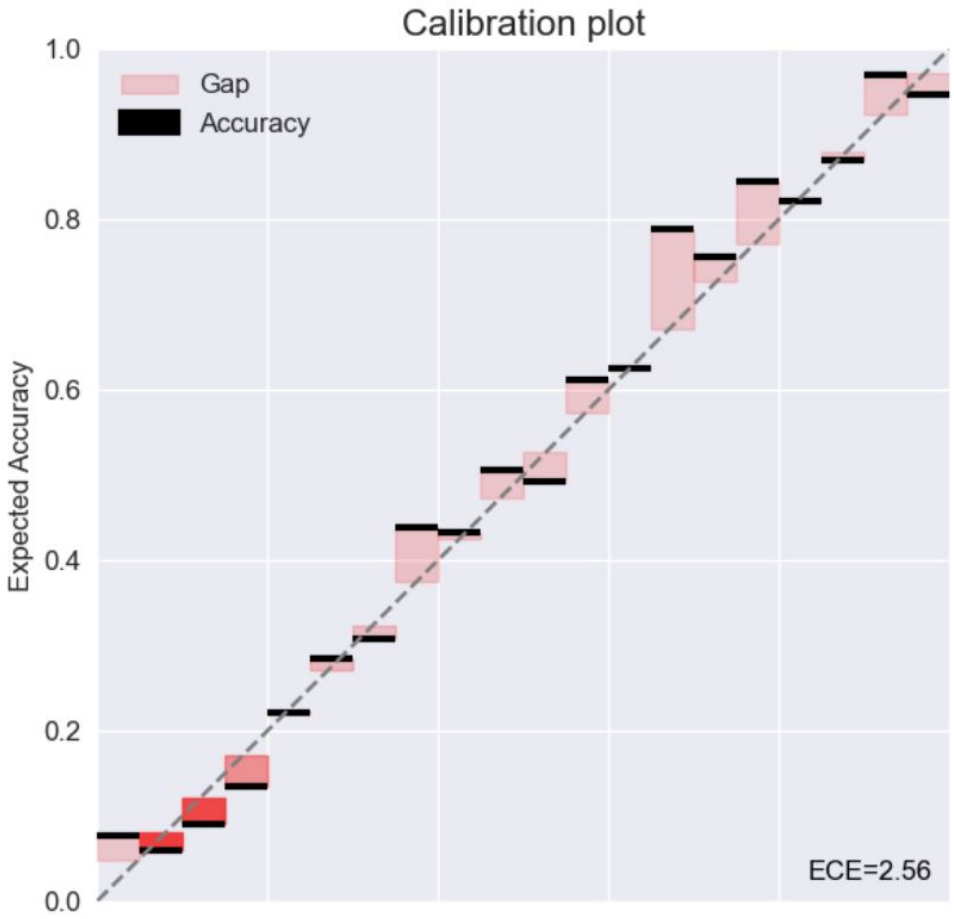

Figure 6.8: The calibration plot for the outdoor geolocation model. Black lines on top of bins represent the accuracy for that specific bin. The more they are aligned with the bisector, the better the calibration is.

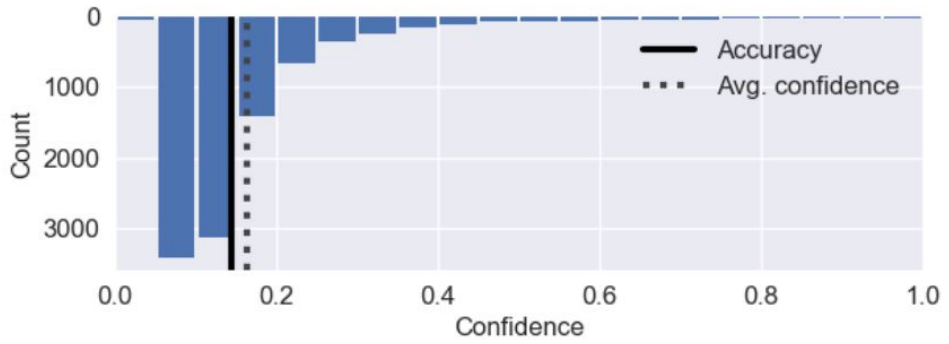

Figure 6.9: The calibration histogram plot for the predicted confidence. The distance between the two vertical lines represent how well calibrated the model is.

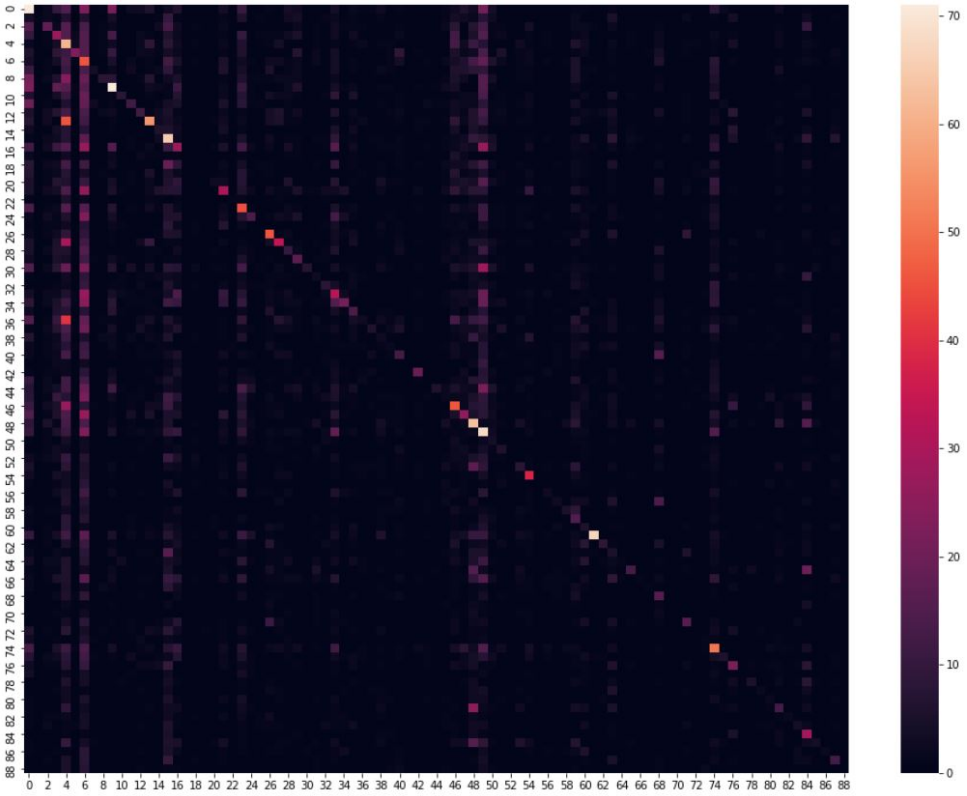

Figure 6.10: Confusion matrix for the outdoor geolocation model. Light color on the diagonal indicates that the model is predicting classes in a correct way.

From the first two plots, it is possible to deduce that the model is well calibrated. In fact bins of Fig. 6.8 are not distant from the bisector of the Cartesian plane, while the Fig. 6.9 vertical lines are almost aligned. This leads us to think that the model uncertainty is reasonable, as the predicted probabilities are very close to the expected probabilities of the same outcomes.

Moreover, the third plot is able to tell us that in general the model is not biased towards a specific subset of target cells. Only in the case of classes 4, 6, 48 and 49, the model seems to show a prevalence of prediction errors: this could easily be due to the fact that the outdoor geolocation model tries to locate images that it is not able to place in those areas. A possible solution could be easily identified, as in Experiment B: submit more images into the training phase.

6.7. Analysis B: Visual Explanations

After evaluating the HMC model calibration, we want to test its explainability features in generating visual explanations, in terms of saliency and visual world maps.

6.7.1. Problem Description

Although the image outdoor geolocation models are able to provide a prediction of the correct location of a given picture with a certain accuracy, users would not be able to understand much from the target label and the confidence level. For this reason, we want to introduce a more detailed approach that is able to explain what and how the model predicts.

With this analysis we want to provide a better understanding of the model predictions: in this way, users are not only able to understand exactly where the model predicted the location of the image using a visual tool, but also which parts of the picture were relevant for the prediction. In these terms, we provide a qualitative analysis of the visual explanations, correlated with some image and map examples.

6.7.2. Method

In order to provide a visual explanation of the outdoor geolocation prediction, we implemented two different procedures:

1. **Grad-CAM** model applied to the feature extractor;
2. **Visual World Map** with cell representations.

Grad-CAM

Firstly, we decided to employ the Grad-CAM model from [22]. In order to visualize the channels influence, we activated the Grad-CAM on the last convolutional layer (top_conv) of the EfficientNet feature extractor. Once the results are generated, they are encoded in an activation map. In order to show the Saliency Map of the prediction, we applied a grayscale color map to the activation's. Subsequently, we superimposed the colored map over the original image with a transparency multiplier of 0.4, therefore we obtain the Saliency Map of the prediction.

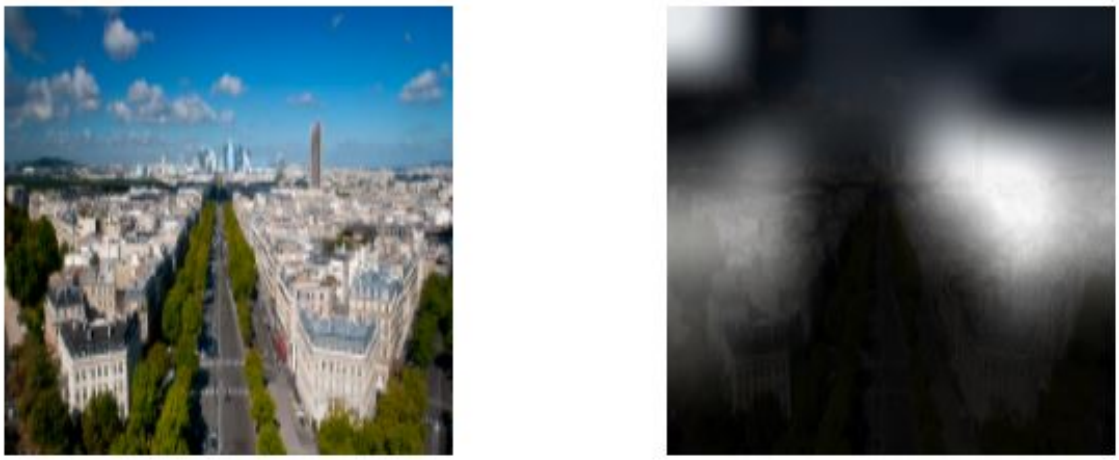

Figure 6.11: Example 1 of a Grad-CAM superimposition. On the left side, the original image. On the right side, the Saliency map of the left image.

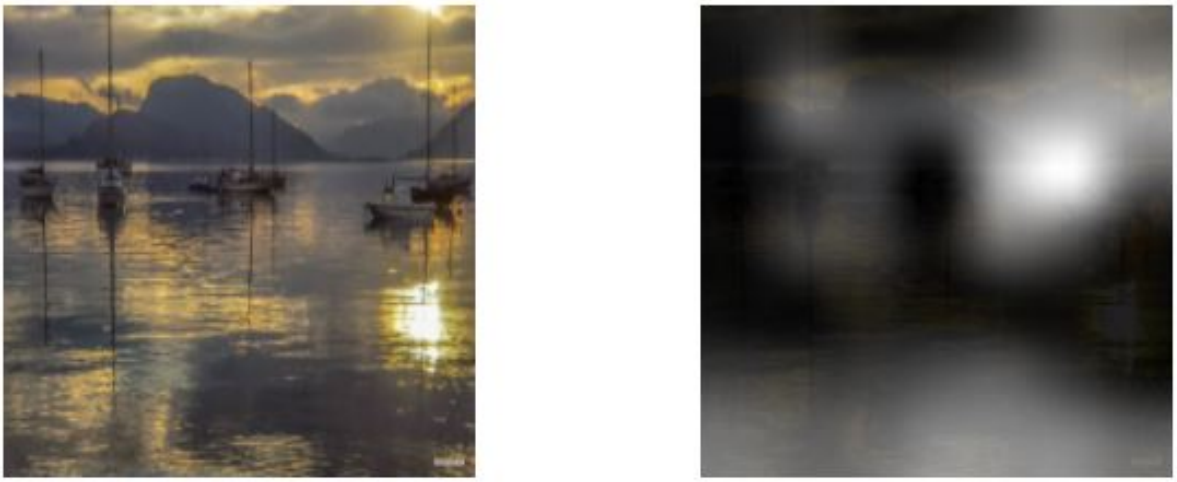

Figure 6.12: Example 2 of a Grad-CAM superimposition. On the left side, the original image. On the right side, the Saliency map of the left image.

Visual World Map

As a further explanation, we want to show a visual representation of the cell predictions that could be more intuitive for the end user. For this reason, we decide to implement a visual world interactive map, over which all the model's target cells are depicted: once the image outdoor geolocation model produces its prediction, it provides a set of confidence values. Each of the entries of such a set can be identified as the probability of an image of falling into a location of the corresponding cell. As this could not be very intuitive at a first glance, we wanted to make a visual representation of the confidence levels. In order to accomplish such task, we superimposed the cells over the world map with a transparency level which is proportional to the probability predicted for the same cells. In particular, if a cell confidence is below 0.05 it will not be drawn. On the other hand, if it reaches above 0.8 probability, it will be capped at such level. After this preprocessing operations, we multiply each confidence by 5 in order to make cells more visible.

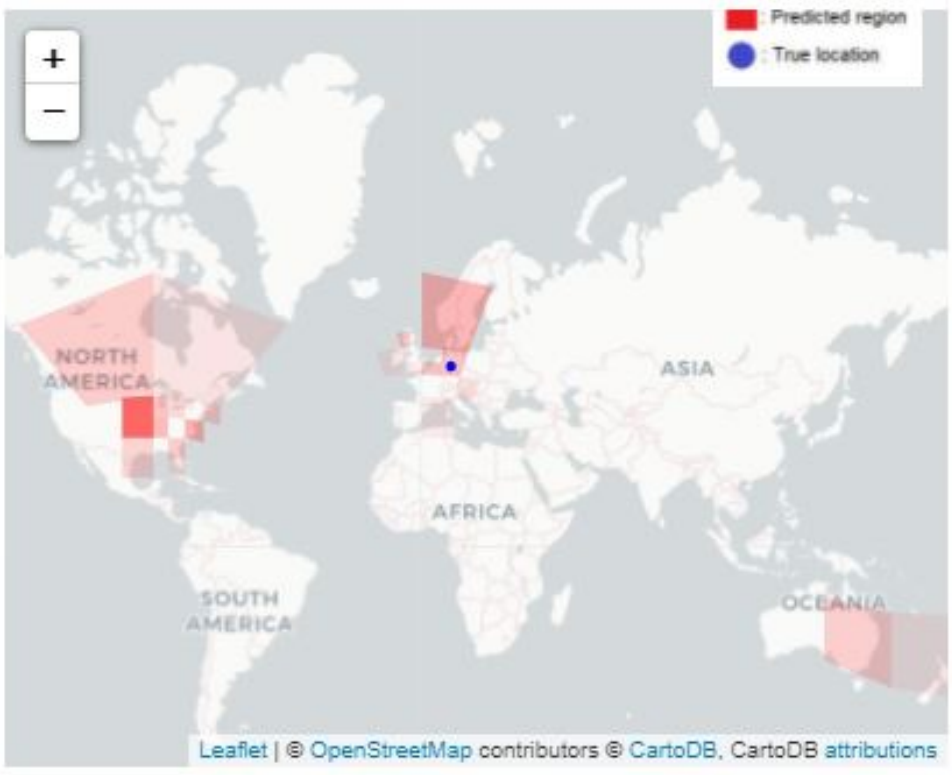

Figure 6.13: Example 1 of a visual interactive world map.

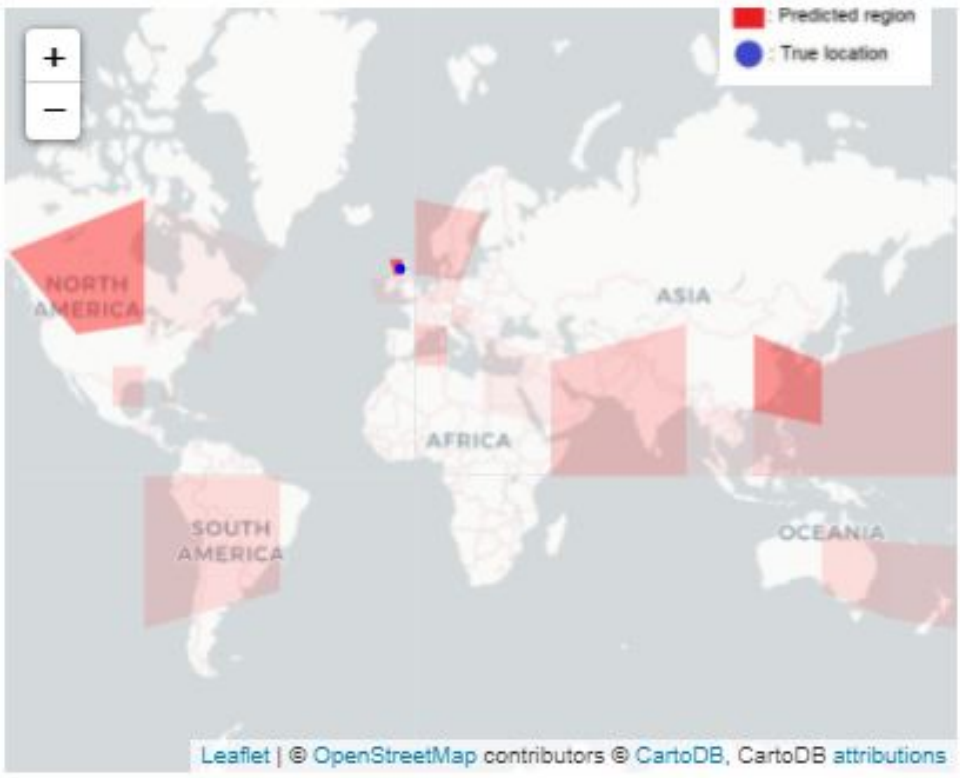

Figure 6.14: Example 2 of a visual interactive world map.

6.7.3. Results and Discussions

The qualitative analysis of the visual explanation has been performed over the Hierarchical model (HMC) that we illustrated in the Experiment D. By providing visual explanation to the outdoor geolocation models, users are enabled to understand better the outdoor geolocation problem. In fact, they are able to detect:

- Where pictures were taken with a visual tool;
- The confidence level that the model is giving to such a prediction;
- The cells with similar level to the top prediction: in this case, if the model failed to get the exact model location, users could identify the correct one among the highest probability cells;
- What drove the models' prediction from the image: which areas were considered as the most relevant, and so on.

As far as the last point is concerned, from qualitative experiments, we are able to understand that the model can focus on specific objects, such as lighthouses, and associate this information to all the cells that could have such objects, as coastal regions.

In the end, visual explanations have proven to give a more complete understanding of the model predictions: even if the model fails to correctly geolocate outdoor images, it provides enough information in order to obtain the correct location: by looking at all the cells with a high alpha values (opaque), users can see which areas are the most likely to be the target of the classification, even though the model has not been able to find the correct one.

7 | Demonstration Website with Visual Explanations

In this section we discuss the demonstration website that we developed in order to show the concrete functioning and explanations of our models. In particular, we will show how the website is structured and how the model computes all the information concerning the image predictions.

The main purpose of the demonstration website (Fig. 7.1) is to illustrate the outdoor image geolocation problem and a working model that is able to detect image location in real time. The whole application has been developed using the Flask framework and embeds the HMC and MC models described in the previous chapters. The website structure is constituted by two main sections that we called:

- **Metrics** section;
- **Geolocation** section.

7.1. Metrics Section

The metrics section is meant to show the best model performance and the image outdoor geolocation task. As it is possible to deduce from the Fig. 7.2, the introduction explains how the model works in terms of target cells and their visual representation. On the left-hand side, we created an interactive world map exploiting the Folium library [15]. On the interactive map, it is possible to move around locations and see where the target cells are placed on the Earth surface. In particular, the 4 vertices of the 2-D projected cells uniquely identify their trapezoidal shape, even though a correct representation would depict borders as curves (more precisely geodesics) on a spherical surface (details can be found in Appendix A). Moreover, it is possible to identify the centroids of each cell, which are an arithmetic average of the latitude and longitude information of the cell's training images. When the mouse points over a specific region, the world map shows a tooltip

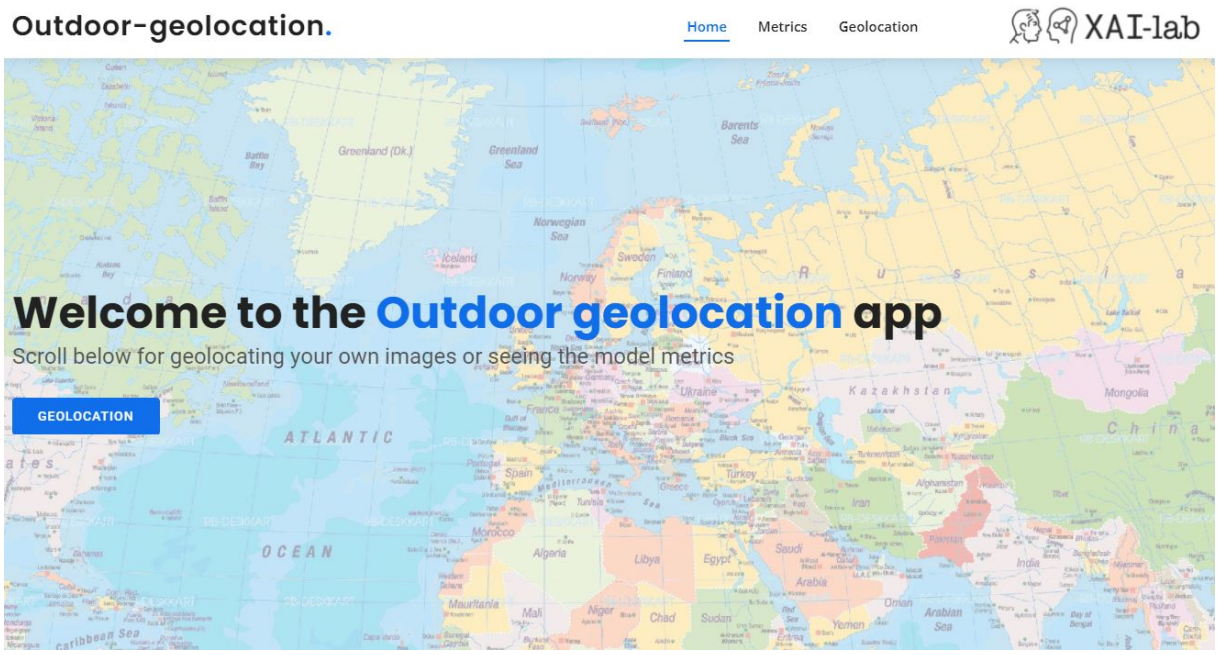

Figure 7.1: Introductory section to the demonstration website.

containing some additional information related to the hovered area:

- the prediction ID;
- the number of images contained inside the cell's boundaries;
- the hierarchy level of the cell: this might change between cells, because in some areas there could be less training images than others. More information about the cell generation algorithm can be found at the Algorithm 19.

On the other hand, the right-hand side of this section contains the information on the current best model performance. In particular, we decided to indicate

- the geolocation problem parameters (72 cells, between 3000 and 30000 training images per cell);
- the test performance on our test set, including information regarding the top-1, top-5 and top-10 correct predictions in percentage of the total number of test images;
- The numerical validation performance, in terms of validation error and accuracy (parameters that are used for compiling the model for training).

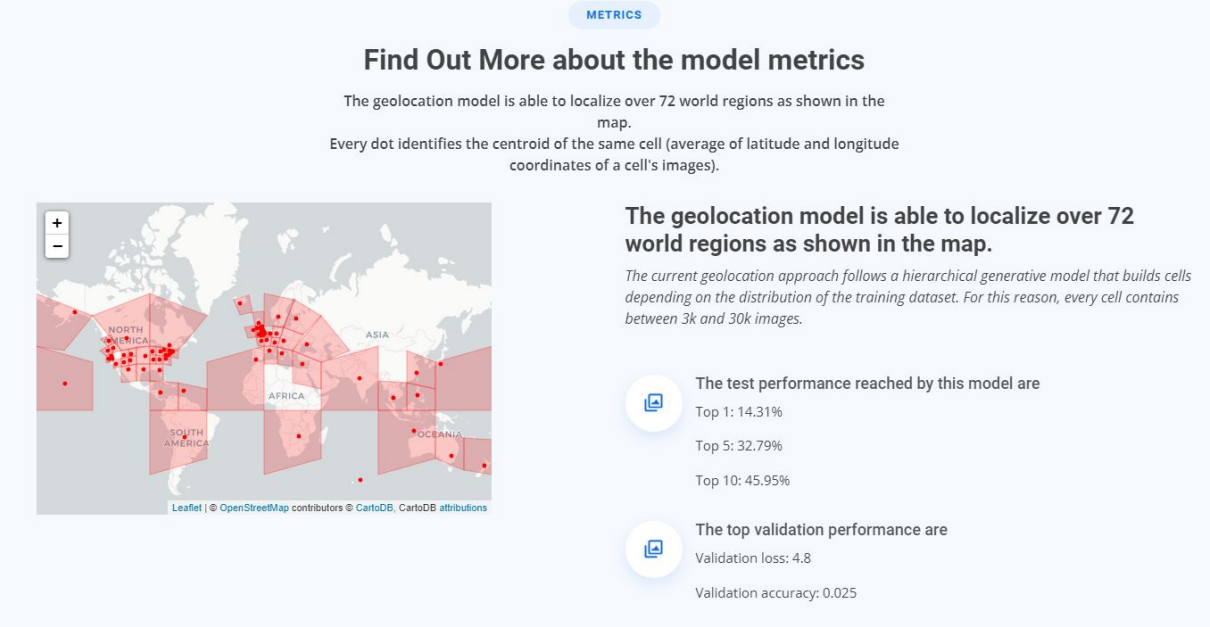

Figure 7.2: Metrics section of the demonstration website.

7.2. Geolocation Section

The geolocation section is the core of our demonstration website. In fact, it is possible to directly test models on user-uploaded images or randomly chosen images from our dataset. As shown in Fig. 7.3, at the top of the screen the website provides two tabs for starting the model prediction: the left button allows users to test the image prediction on a random image taken from our test set, while the right form allows users to manually upload an image that they would like to be predicted.

Once one of the above-mentioned buttons have been pressed, the model runs in the background, computing several pieces of information that will be shown at the end of the operations. In particular, the web server that hosts the demonstration website is running in background two different deep learning models:

- The hierarchical geolocation model **HMC**, used for computing the geolocation prediction. It is the best model we obtained in order to perform geolocation;
- The **MC** model, used as a baseline comparison for the HMC.

The two models are responsible for producing the confidence levels for the cells of the world grid. They do not only tell the best performance, but produce a specific probability value for each of the cells. In this way, it is possible to compare their performance. In the meantime, **Grad-CAM** is used for computing the visual explanation of the geolocation

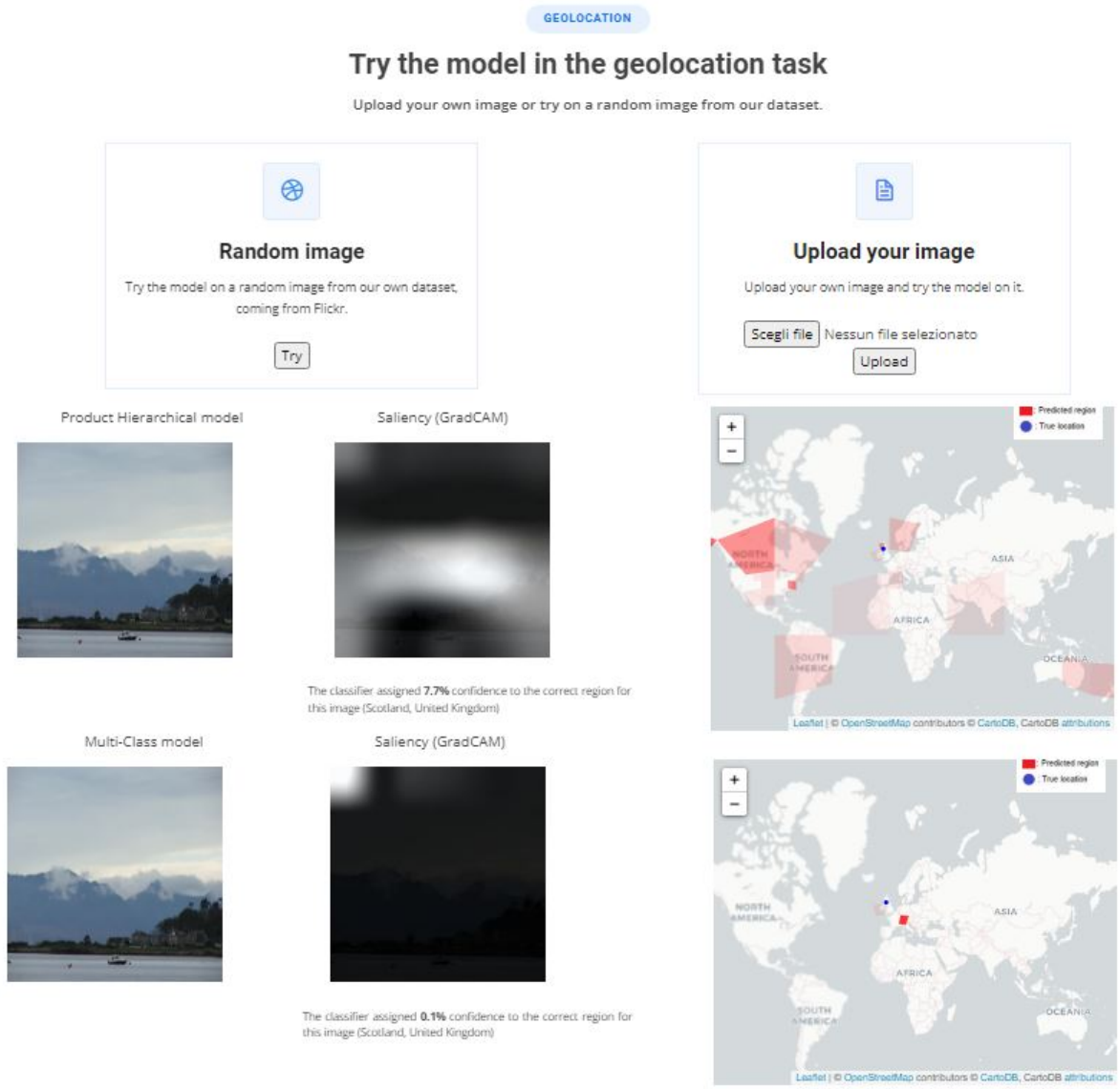

Figure 7.3: Geolocation section of the demonstration website.

models: it is able to produce a heatmap of the last convolutional layer of the Hierarchical geolocation model and apply it on a lower alpha-level over the input image. In the end, as the output of the prediction computation process, the web server produces for the two geolocation models:

- an **array of confidence levels**, that will be used for computing the prediction map;
- a **Grad-CAM image**, that contains the overlay heatmap of the last convolutional layer of the outdoor geolocation model;
- the **ground-truth information**, included the cell-names and the confidence level

for such cell;

- the **visual world map**, which is an interactive map similar to the one used in the Metric section. This map shows all the predicted cells above a 0.05 confidence level. For each cell, they are drawn on the map on an alpha-level which is proportional to the confidence level of the same cell.

The main purpose of the demonstration of this work is to show the model performance but also a set of visual explanations of the outdoor image geolocation prediction. In particular, we were able to create a Grad-CAM operation on the geolocation model that highlights which parts of the image were useful for the current prediction. With heatmaps, we are able to understand how the model extracts features and which parts of the image are more relevant for the final prediction, even if neural networks work in a black-box setting. Moreover, interactive maps allow users to get a better feel for how the model predicted the final location and which areas happen to be most similar to the target in terms of features. We also applied a reverse-geocoding of the centroids of each cell in order to obtain names for cells (e.g. "London", "New Zealand") to provide a more user-friendly visualization of the cells. In particular, we exploited the information related to the "Region-name","Sub-country-name" and "Country-name" [16]. Finally, the combination of visual features from heatmaps and prediction maps that show which grid cells were closer to the target, enables users to have a better understanding of how the task works and how close the model came to predicting the correct location.

8 | Conclusions and Future Research Directions

In this final section we provide an answer to the research questions we posed in Chapter 3, based on the approach we followed and the experimental results we achieved. Moreover, we discuss the possible future direction in this research field, especially focussing on the possible improvements of our models and different task approaches.

Through this thesis work, we provided answers to a list of research questions that we posed in the Chapter 3. In particular, we tried to deal with the outdoor geolocation problem exploiting a set of deep learning techniques that could solve the problem without any additional information beyond the image's pixels. Firstly, we investigated the state-of-the-art in the deep learning field, including different model structures, modern feature extractors and related works that tried to solve similar problems to the one we posed and analyzed. In this direction, we found that several research papers illustrate some solutions to the image geolocation problem, working on a limited set of categories of images [30] or constrained regions of the world [21]. Some other works, such as PlaNet [29] and IM2GPS [9] were amongst the first in tackling the problem on a worldwide scale and opened the research direction in the image geolocation field. After that, the work of [17] introduced the concept of hierarchy related to the world division on a grid of cells and transforming the geolocation problem into a classification task. We took inspiration from all the work we cited in the Chapter 2 in order to create an innovative approach for the outdoor image classification over the grid of cells, that we described thoroughly in Chapter 4. The main novelty that we introduced is related to the exploitation of the hierarchical knowledge of the cell generation directly inside the deep learning model structure. Successively, inside the Chapter 5 we describe the dataset we used for training our models and the way we collected it. We provide some insights on the dataset statistics and image relationships (hierarchical feature similarities). Once everything was set up for the training of the models, we ran a set of experiments on the models in order to compare them and get insights on their behaviors and performance. The analysis of Chapter 6

shows relevant information on the outdoor geolocation models and allows us to draw some conclusions: the Hierarchical Multi-class(HMC) classifier based on the product operation is able to outperform baseline multi-class image classifiers and other hierarchical variants (HMC-sol, HMC-smooth). As far as the model explainability is concerned, we were able to make use of state-of-the-art technologies, such as the Grad-CAM [22] and other visual tool [15] in order to produce relevant visual explanations that we were able to show through the demonstration website described in chapter 7.

8.1. Answering the Research Questions

In this section we provide an answer to the research questions that we posed in Chapter 3. Firstly, we want to express them again here and provide an answer for each of them.

1. Is it possible to provide a deep learning model which is able to geolocate any kind of outdoor image from a worldwide perspective?
2. Is it possible to improve the performance of an image geolocating model by adding the hierarchical information of how the world was divided up directly inside the model?
3. Can a deep learning model provide a significant visual explanation that could be used to understand why the image has been predicted in such a location?
4. What is the best way to predict the location information? How can we evaluate that the model is predicting reasonable locations?

As far as the **first** question is concerned, we can answer positively: in order to analyze the problem from a worldwide perspective, we decided to crawl a relevant dataset that could cover any region of the world, even the wildest and least explored, by exploiting the Flickr platform. In Chapter 5, we explain thoroughly how we collected such data and how we preprocessed it. Furthermore, we exploited the Google S2 Geometry library[7] in order to create a set of cells that could cover almost the whole world surface. For this reason, we were able to train a deep learning model on a set of class that could cover the whole world.

In order to answer the **second** question, we took inspiration from the work of [17]: the authors were able to create a hierarchical model exploiting a set of different world-grids on different granularity levels and fusing the predictions of the cells in each of the different settings. Instead of exploiting this kind of hierarchy, we decided to exploit a single world-grid division and to shape the deep learning model around it (as described in Chapter

4). Once the model structure has been defined, we ran a set of experiments in order to compare different hierarchical models and baselines. In particular, the Experiment D is able to tell us that each of the different hierarchical models that we developed outperforms the baseline (non-hierarchical). Moreover, we can say that the HMC model is the best performing one, both in terms of validation performance and training time: it takes only 1.09x the training time of the baseline multi-class (MC) classifier in order to be trained.

Once the model performance questions have been answered, it is time to reply to the **third** question: state-of-the-art visual explanation techniques involve mainly the use of activation map-based techniques. In this direction, we decided to exploit Grad-CAM and visual tools in order to provide visual explanations. Through the qualitative results of the Analysis B, we are able to provide more information on the model prediction: the saliency map is able to tell which pixels were relevant for the final prediction and the visual world map is able to show all the different region confidence levels in order to provide a set of possible locations in which the image could have been taken. From a qualitative point of view, we are able to say that the visual explanation tools that we developed produce a clearer and more understandable explanation of the model prediction.

Finally, we provide an answer to the **fourth** question. In order to obtain a reasonable prediction, we think that the model should be well calibrated and produce balanced results. Therefore, we wanted to analyze the best model we obtained (HMC) in order to measure its calibration and the goodness of its predictions. In the Analysis A, we analyzed the HMC model in terms of calibration and confusion matrices. The results show that such a model is well calibrated, and the confusion matrix is reasonable. In these terms, we can say that the approach we followed is good at predicting reasonable location information, and we were able to evaluate it through confusion matrices and calibration plots.

8.2. Future Research Directions

The results that we achieved are to be intended as a starting point for the outdoor image geolocation problem. In fact, we were able to improve a baseline model performance, but there is still margin to improve the model performance and to introduce new visual tools for improving the model explainability. In this section, we want to illustrate a list of possible future research directions that could be followed after this thesis work.

- **Object Detection** and **Textual Explanations**: the current thesis work mainly focuses on the geolocation problem, with a further development on visual explanations. In order to provide a better understanding of the model predictions, we

think that we could provide additional information rather than the saliency and visual world map. In fact, the next works could focus on the object detection inside the scene: for example, a neural network could be trained in order to acknowledge the presence of certain types of trees or buildings, recognize where they are located and show users their position on a superimposed image. This could help both the understanding of the model prediction, but also it could be integrated inside the geolocation classifier: the information related to the detected object could be useful also for the final prediction, either with self-learned (deep learning techniques) or predefined rules. Moreover, we think that the object detection could be integrated with a system based on text generation in order to provide Textual Explanations together with Visual Explanations. A system based on text generation could help users to improve their experience, as words could provide insights on the prediction that are not recognizable through visualization;

- **World Cell Preservation**: the cell generation algorithm is based on two different thresholds, which we named as t_{max} and t_{min}, that are respectively used for deciding whether to split a certain cell any further and to drop a cell in the case it has a low number of samples. This definitely provides a good setting for the model training, as classes are balanced and the training is performed accordingly. Inevitably with this implementation, a set of world regions remain uncovered by the problem solution. In order to prevent the region drop, the cell generation algorithm must be modified: some possible implementation that we thought are for example to avoid splitting into four sub-cells every time the split operation is required, but instead decide to divide either vertically or horizontally depending on the sample distribution. This operation could be performed by a stochastic version of the cell generation algorithm that works similarly to tree learning algorithms such as regression/decision trees;

- **Investigating IoR**: in this thesis work, we define a new evaluation metric for self-learning systems which we named as Improvement over Random (IoR). As we thoroughly explained in section 6.1.4, the IoR is computed as the ratio between the model accuracy and the random prediction. As for the current implementation, we decided to use a uniform probability distribution as the random probability, therefore it is computed as $\frac{1}{N_{classes}}$. This is for sure a valid implementation, but we think that other possible random probability configurations could be tried to improve the soundness of the evaluation metric. Some different random models could be:

  - *Largest target class prediction*: compute the random probability as if we had a model that always predicts the target class that has the largest amount of

training data;

- *Gini Index*: use a more accurate version of the random prediction, which depends also on the number of samples for each target class.

- **Visual World Map improvements**: as for the current implementation, the demonstration website provides a visual world map for each prediction. On the map, users can see where the regions are located (in a trapezoidal shape) over the world map layout and the confidence level that the model assigned to each of the cells. In this direction, some possible improvements can be made.
  - Draw cells as if they were on a sphere: in fact, the cell generation process performed through the Google S2 Geometry library, generates cells on a sphere and only afterwards it projects them on a surface. This procedure could definitely give more detailed information on the grid cells, avoiding for example overlapping regions.
  - Investigation on different shading scale for the world cells: the current color shading for the cell drawing depends only on their confidence level; the confidence level is wrapped inside a pair of thresholds of color transparency and the shading is only proportional to the confidence. In order to make the shading scale more accurate, we think that including the image density of the cells inside the transparency formula would provide a better visualization.

- **Hierarchy-based confusion matrix**: the current ordering of classes is based on the cell generation algorithm. In particular, when the procedure decides to split a cell, it generates four new sub-cells and puts them in the first positions of the target cell list without taking into account the neighboring property of the world regions. In this way, when we generate the confusion matrix for the model we train, we are not provided with any kind of correlation within cells: we argue that with a reasonable cell numeration, we could group neighbor regions and see their correlation on matrices which are able to express the dependencies among the target classes. We could see groups of cells that are highly dependent one to another and see their hierarchical correlation (which provides further insights on the geolocation problem).

Bibliography

[1] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017. doi: 10.1109/ICEngTechnol.2017.8308186.

[2] A. Bella, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. *Calibration of Machine Learning Models*, pages 128–146. Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques. IGI Global, Hershey, PA, USA, 2010. ISBN 9781605667669. doi: 10.4018/978-1-60566-766-9.ch006. URL `https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-60566-766-9.ch006`.

[3] R. Caruana, S. Lawrence, and L. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, page 381–387, Cambridge, MA, USA, 2000. MIT Press.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[5] Y. Ge, H. Wang, F. Zhu, R. Zhao, and H. Li. Self-supervising fine-grained region similarities for large-scale image localization. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *Computer Vision – ECCV 2020*, pages 369–386, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58548-8.

[6] I. J. Goodfellow, Y. Bengio, and A. Courville. Deep learning. `http://www.deeplearningbook.org`, 2016.

[7] Google. Google s2 geometry: spherical geometry library for manipulating geographic data. `https://github.com/google/s2geometry`, 2021.

[8] T. Guo, J. Dong, H. Li, and Y. Gao. Simple convolutional neural network on image classification. In *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, pages 721–724, 2017. doi: 10.1109/ICBDA.2017.8078730.

[9] J. Hays and A. A. Efros. im2gps: estimating geographic information from a single image. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[11] M. Hollemans. Reliability diagrams. `https://github.com/hollance/reliability-diagrams`, 2021.

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[13] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, page 950–957, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. ISBN 1558602224.

[14] C. Masone and B. Caputo. A survey on deep visual place recognition. *IEEE Access*, 9:19516–19547, 2021. doi: 10.1109/ACCESS.2021.3054937.

[15] MIT License. Folium: python library for plotting python data on leaflet.js maps. `https://github.com/python-visualization/folium`, 2021.

[16] MIT License. Geopy: Python library for several popular geocoding web services. `https://github.com/google/s2geometry`, 2021.

[17] E. Müller-Budack, K. Pustu-Iren, and R. Ewerth. Geolocation estimation of photos using a hierarchical model and scene classification. In *ECCV (12)*, volume 11216 of *Lecture Notes in Computer Science*, pages 575–592. Springer, Sept. 2018. doi: https://doi.org/10.1007/978-3-030-01258-8_35. URL `http://openaccess.thecvf.com/content_ECCV_2018/html/Eric_Muller-Budack_Geolocation_Estimation_of_ECCV_2018_paper.html`.

[18] J. Nixon, M. Dusenberry, G. Jerfel, T. Nguyen, J. Liu, L. Zhang, and D. Tran. Measuring calibration in deep learning, 2020.

[19] C. Pelletier, G. Webb, and F. Petitjean. Temporal convolutional neural network for the classification of satellite image time series. *Remote Sensing*, 11:523, 03 2019. doi: 10.3390/rs11050523.

[20] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger, 2016.

[21] T. Salem, S. Workman, and N. Jacobs. Learning a dynamic map of visual appearance. *CoRR*, abs/2012.14885, 2020. URL `https://arxiv.org/abs/2012.14885`.

[22] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, pages 618–626. IEEE Computer Society, 2017. ISBN 978-1-5386-1032-9. URL `http://dblp.uni-trier.de/db/conf/iccv/iccv2017.html#SelvarajuCDVPB17`.

[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014. ISSN 1532-4435.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions, 2014.

[26] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[27] M. Tan and Q. V. Le. Efficientnetv2: Smaller models and faster training, 2021.

[28] N. N. Vo, N. Jacobs, and J. Hays. Revisiting IM2GPS in the deep learning era. *CoRR*, abs/1705.04838, 2017. URL `http://arxiv.org/abs/1705.04838`.

[29] T. Weyand, I. Kostrikov, and J. Philbin. Planet - photo geolocation with convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, 2016.

[30] T. Weyand, A. Araujo, B. Cao, and J. Sim. Google landmarks dataset v2 - a large-scale benchmark for instance-level recognition and retrieval, 2020.

[31] L. Yan, Y. Cui, Y. Chen, and D. Liu. Hierarchical attention fusion for geo-localization, 02 2021.

[32] Y. Yin, Y. Zhang, Z. Liu, S. Wang, R. R. Shah, and R. Zimmermann. Gps2vec: Pre-trained semantic embeddings for worldwide gps coordinates. *IEEE Transactions on Multimedia*, pages 1–1, 2021. doi: 10.1109/TMM.2021.3060951.

A | Google S2 Geometry Library

This appendix is devoted to the description of the Google S2 Geometry library[7] that we used for generating the grid-based hierarchical world division.

The Google S2 Geometry is a library for spherical geometry that aims to have the same robustness, flexibility, and performance. Instead of its main competitors, S2 enables developers to work in a three-dimensional setting (**sphere**) which fits almost perfectly with the planet Earth geometry. Thanks to its spatial indexing technique, this library enables a very high-performance set of operations over a single space of coordinates (latitude and longitude), especially efficient query operations for finding neighboring regions, measuring distances, computing centroids and so on. In particular, this work exploits the hierarchical world division framework offered by the S2 library, that is intended to work as follows:

1. Consider a cube as for the top level of the hierarchy and take its six faces separately;
2. Project each of its six faces on a unitary sphere, creating six level-0 cells;
3. each of the previous cells is divided further in four equal cells;
4. recursively apply point 3. until a certain stop condition is reached.

The hierarchical procedure can range from level 0 up to level 30, creating at maximum $6 * 4^{30}$ cells (if plotted on the Earth surface, they correspond to 1 cm^2).

Every cell is uniquely identified by a **hexadecimal ID** together with a set of **4 geodesics**, which are lines on the spherical three-dimensional space. For this reason, when we try to depict a cell on a flat screen map, it happens to seem spherical too. By exploiting the **LatLong** class of the S2 Geometry library, we are able to extract additional information from the cell definition. In our case, as we want to depict the cells on a flat map, we can extrapolate the four points that represent the vertices of the cell. Unfortunately, having only this latter information does not provide us enough information to show the curved borders of cells (sphere cells), therefore when we try to draw them on a flat map, we approximate such borders as straight segments between the vertices.

The cell indexing follows the S2 space-filling curve in a fractal shape. In particular, six

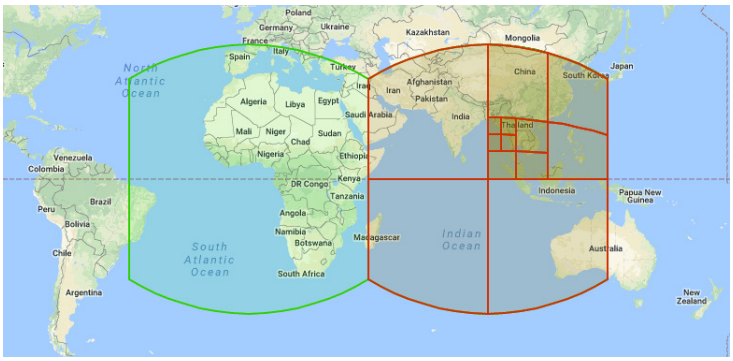

Figure A.1: An example of a recursive hierarchy of cell generation from the cube projection until depth 5.[1]

Hilbert curves linked together to form a single continuous loop over the entire sphere. The **Hilbert curve** is a surjective mathematical function that maps

$$f : [0, 1] \to [0, 1] \times [0, 1]$$

to visit every point of the codomain. By composing the S2 space-filling curve, the authors obtain a function

$$g : [0, 1]^6 \to S^2$$

which is constructed by mapping 6 copies of the Hilbert curve to the 6 faces of a unit cube, reflecting and rotating the curves as necessary so that they link together seamlessly into a continuous loop. Finally, the cube is mapped to the unit sphere using a transformation that minimizes distortion.

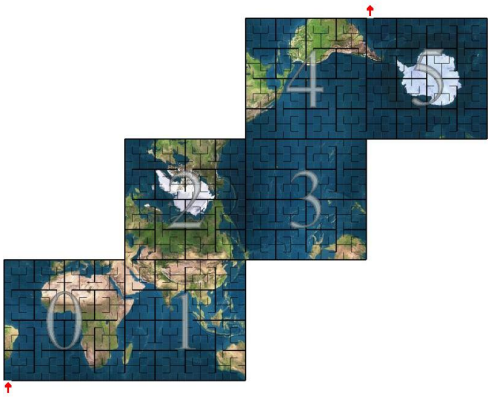

Figure A.2: Flattened representation of the fractal of six Hilbertian curves projected on the six faces of a cube [2]

[1]Source: `https://s2geometry.io/devguide/s2cell_hierarchy`
[2]Source: `https://s2geometry.io/devguide/img/s2cell_global.jpg`

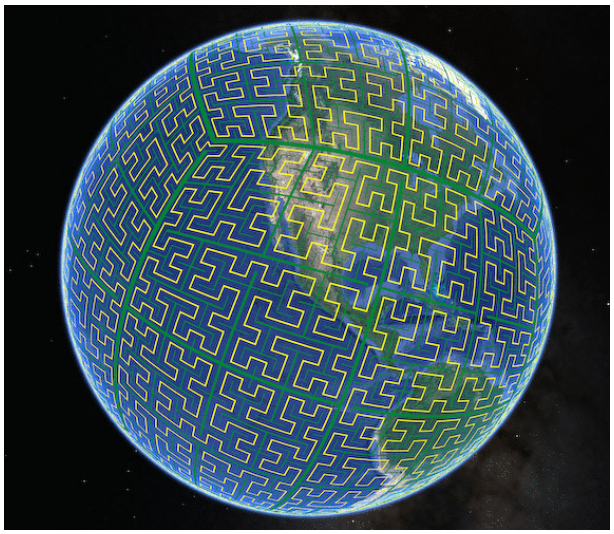

Figure A.3: Fractal of six Hilbertian curves that wraps around the world surface in order to fully cover it.[3]

The numbering of the cells follow this set of curves so to have cells with similar hexadecimal IDs one close to the other, enabling the library to speed up the process of finding neighbor cells. Exploiting this feature, the S2 Geometry library enables to group close cells in order to create S2CellUnions, that are collection of cells that can be used to approximate oddly shaped regions on the unitary sphere. In the case in which cells that share the same parent are grouped under the same CellUnion, the library will tend to normalize their coordinate representation by replacing them directly with the common parent cell.

[3]Source: `https://s2geometry.io/devguide/img/s2curve-small.gif`

B | EfficientNet

In this section we are going to describe one of the most performing architecture for feature extraction in the state-of-the-art. EfficientNet[26] is not only good at doing its job, but it also makes it extremely faster than any other existing model.

For modern feature extracting models, Convolutional Neural Networks are a de-facto standard in the field of deep learning. Many authors proposed different models[23][10][25] throughout the past years, with the main purpose of improving the model performance in terms of loss and accuracy. A careful analysis conducted by the authors of [26] showed that all the state-of-the-art feature extractor puts as an objective function the purpose of improving one of the next three model parameters:

1. **Depth**: most common way of improving deep neural networks. The rationale behind deepening deep learning models can be found in thinking that models can capture richer and more complex features while generalizing well on new tasks. This is partially true, as deep models also bring the vanishing gradient problem, preventing too complex network to learn as the gradient propagation gets gradually shrunk towards zero;

2. **Width**: often used for small-sized models, wider neural networks tend to capture more fine-grained features and be easier to train. Having very large but shallow models bring them to lose high level features;

3. **Resolution**: some neural networks try to increase the input image resolution in order to retrieve potentially more fine-grained patterns, but this has shown little to no improvements in terms of model accuracy.

The idea behind the EfficientNet architecture is to improve the previous three parameters all together, as it has been proven that they are not independent. Therefore, the authors thought to create a **compound scaling method** that exploits a parameter ϕ to uniformly scaling depth, width, and resolution.

After a set of empirical experiments in a grid search, the authors found the best settings for the model training. The more important feature of this procedure is that it seems

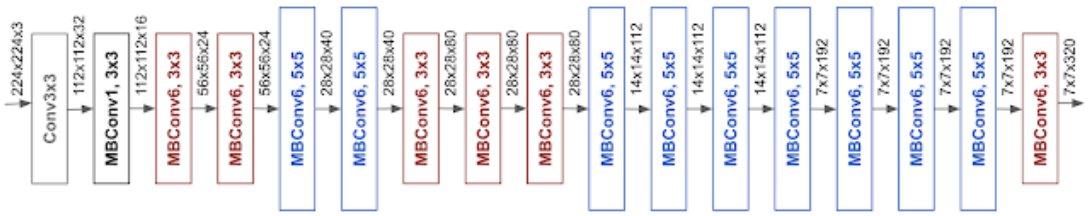

Figure B.1: Global architecture of EfficientNet in its B0 implementation.[1]

that FLOPS are strictly correlated with the three parameters of the grid search. In particular, FLOPS are linear with the depth and increase in a quadratic way with width and resolution. With this intuition, the authors aimed to create a model that not only focuses on improving accuracy, but also the FLOPS required to compute such a task. Therefore, they set a search space in the domain

$$ACC(m) \times [\frac{FLOPS(m)}{T}]^{\omega} \tag{B.1}$$

Note that T and m stand for target FLOPS and model, while w is an empirical parameter set to $\omega = -0.07$

B.1. EfficientNet architecture

The EfficientNet architecture has been created gradually to improve the model performance through an increase of the model's parameters.

The model is represented by a set of **MBConv** building blocks with squeeze-and-excitation optimization. The MBConv is also known as mobile inverted bottleneck convolution[mobilenetv2-paper]. MBConv are Inverted Residual convolutional blocks, which adopt the narrow $\longrightarrow$ wide $\longrightarrow$ narrow approach, hence the inversion. Firstly, it widens the input with a 1x1 convolution, then it uses a 3x3 depthwise convolution (which greatly reduces the number of parameters), then the block uses a 1x1 convolution to reduce the number of channels so input and output can be added. In terms of neural layers, MBConv is contituted of two rounds of:

- Convolution;
- BatchNormalization;
- ReLU6 activation;
- DepthwiseConvolution.

[1] `https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html`

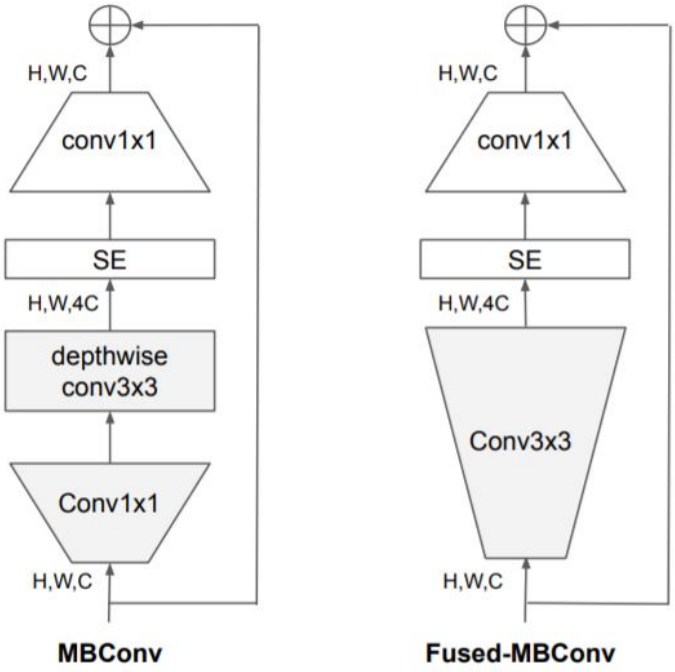

Figure B.2: On the left-hand side, the EfficientNet MBConv, while on the right-hand side the EfficientNetV2 FusedMBConv improvement

Those layers are followed by an Add Layer which sums the input of the block together with the output of the previous steps.

Several building blocks are combined in order to form the EfficientNet implementation (in Fig. B.1, the EfficientNetB0 structure is shown).

The authors created eight different settings (EfficientNetB0 - B7) and the most complex one showed the best performance in the ImageNet task over all the state-of-the-art deep learning models.

B.2. EfficientNetV2

Together with the advent of tensor unit accelerators, researchers have found the possibility of implementing even further the EfficientNet performance [27]. In particular, they designed a new residual convolutional block, called FusedMBconv, which is an updated version of MBconv described in the previous section.

Differently from the previous implementation, **FusedMBConv** (Fig. B.2) replaces the initial widening of the 1x1 convolution of MBconv and directly pipes the input channels into the 3x3 convolutional layer. This change has shown greater performance on server

accelerators. Moreover, when performing progressive learning, the network grows together with the image size: at every learning step, the initial stages are enlarged by adding new neural layers able to cope with the larger images. As a further detail, EfficientNetV2 is provided with regularization adaptation. In fact, at early stages, the model is subjected to weaker regularization which gets stronger when the image size increases.

Finally, EfficientNetV2 is a major update of the older version, specifically in terms of training FLOPS (server accelerator boosting) and model parameters (FusedMBConv is shallower than MBConv).

List of Figures

List of Tables

List of Symbols

Acronym	Description
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
DNN	Deep Neural Network
CNN	Convolutional Neural Network
$LSTM$	Long-Short Term Memory
$FCNN$	Fully-Connected Neural Network
$ReLU$	Rectified Linear Unit
RGB	Red-Green-Blue
GBC	Geolocation by Classification
IBL	Image-Based Localization
DML	Data Manipulation Language
GPS	Global Positioning System
$VLAD$	Vector of Locally Aggregated Descriptor
$Grad-CAM$	Gradient-weighted Class Activation Mapping
$DODAG$	Destination Oriented Directed Acyclic Graph
DGH	Dependency Graph of Hierarchy
API	Application Programming Interface
CCE	Categorical Cross Entropy
SGD	Stocastic Gradient Descent
LR	Learning Rate
ES	Early Stopping

Acronym	Description
IoR	Improvement over Random
MC	Multi-Class
HMC	Hierarchical Multi-Class
$HMC - sol$	Hierarchical Multi-class with sum-of-logarithms
$HMC - smooth$	Hierarchical Multi-class with smoothing of probabilities
ECE	Expected Calibration Error
$FLOPS$	FLoating-point Operations per Second
$MBConv$	MobileNet convolution

Acknowledgements

Now that my academic path comes to an end, it is time to thank all the people that helped and supported me along the way. In the first place, I would like to thank my thesis advisor, Professor Mark James Carman, one of the most passionate person that I was lucky enough to work with: among all professors along my academic career, he has been the most wise and helpful. I could see his passion and pride in every meeting we had: the way he works drives you to do the same, giving you inspiration and transmitting the passion for the research. He also made me consider starting a PhD and I will probably think about it for the next call.

Also, I want to thank my parents Luigi and Laura and all of my family, which never stop believing in me and supporting me in all of my decisions and I want to apologize with them for all the times that they had to bear my university anxieties.

I cannot forget to mention my group of historical and true friends, Albe, Manuel, Andre, Paga, Pansus, Davide and Yuri which helped me to detach from the difficult and stressing time I had at university, letting myself go and enjoying very good times. Also, I want to thank my friends at Polimi, Ema, Andrea, Ivan, Umbe, PC, Cristian and Aba, which company made lessons and study hours always enjoyable and funny.

I want to dedicate a few lines also to thank a very special person: my friend and mentor David Tagart Del Re. He is a very special man, one of a rare kind that is always ready to help you when you are in need. He aims at improving you, as a professional but especially as a person. With his lifelong mission of helping people to reach their dreams, he contributed in shaping the person I am today and I will never be grateful enough for what he did for me.

Finally, I want to thank all the people that I did not mention but have been part of this magnificent journey. Now I am ready to continue pursuing my dreams and once they will be reached, I will set a higher score as I am always aiming for the best.