BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE IN ENGLISH

DIPLOMA THESIS

Birds Classification From Images Using Transfer Learning

Supervisor
Assist. PhD. Alina Delia Călin

Author
Galan Maria Lavinia

2022

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ ÎN LIMBA ENGLEZĂ

LUCRARE DE LICENŢĂ

Clasificarea păsărilor în imagini folosind învățarea prin transfer de cunoștințe

Conducător științific
Asist. Dr. Alina Delia Călin

Absolvent
Galan Maria Lavinia

2022

ABSTRACT

The birds play an important role in the global ecosystems, being excellent environmental indicators. Moreover, bird watching is a common hobby, but for this activity the identification of the species is required. The visual data provides an easier recognition of the birds species, compared to the audio recordings. In this context, an intelligent system for classification of birds from images becomes a necessity.

The domain of image classification and computer vision has recently found outstanding accomplishment in Convolutional Neural Networks (CNN). Lately, the use of the pre-trained CNN models offers much better perspectives of the work that can be done in order to obtain efficient models and accurate results.

This thesis suggests a transfer learning-based strategy for the classification of 52 species of birds that are commonly found in Europe. Experiments using various deep learning models were conducted and after analyzing the results, the InceptionV3 architecture was integrated into the structure of the final models. The paper presents an original solution, in the form of a fusion between two machine learning models that were trained on different datasets, one containing high quality images and another having average quality images that involve different sizes, backgrounds and natural elements that may interfere with the view of the actual bird. Another innovation is represented by one of the used datasets which was created and filtered manually. The fusion model outperformed the base models taken individually and also accomplished good results relative to the state-of-the-art approaches. The accuracy scored by the fusion model was 88.88% for validation, with a validation loss of 0.4212, and 87.95% for testing, with a loss of 0.5926.

The machine learning model was further integrated in a web application destined for average, non-specialized users, but it may become helpful also for the specialized users, such as ornithologists or bird watchers. The application allows the user to see the top 3 predictions for an uploaded image and a selected model.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Birds are important actors in the global ecosystems, contributing significantly to the overall health of nature. They play a critical role in controlling agricultural pests; for instance, according to [12], birds consume more than 400 million insects a year. As reported by U.S. Forest Service, there are around 2,000 species of birds, including the ones in the honeyeaters or hummingbirds' families, that also act as pollinators, especially at high altitudes or in warm climates. Further, their contribution in seed dispersal is vital for preventing the extinction of certain species of plants or for enhancing environmental diversity, as both frugivorous and omnivorous migratory birds can transport seeds to new regions across hundreds of kilometers [13].

Moreover, bird watching is a hobby gaining in popularity: approximately 45.1 million American birdwatchers were counted by the U.S. Fish and Wildlife Service. This activity, practiced mostly by wealthy and civilized persons, promises to add major contributions to the ecotourism sector. The statement that the popularity of bird watching may follow an increasing trend is further supported by the recently discovered benefits that natural bird sounds have on humans' health. It is demonstrated that the songs of birds can improve attention and anxiety problems and can diminish stress and annoyance [14].

Despite the valuable contribution of birds to ecological balance and to individuals' peace of mind, the extinction risk among the species of birds is a serious threat. Due to the habitat loss, the human persecution or the introduced predators, a considerable number of avian families are about to disappear [15]. These dangerous situations can reveal more matters of concern, as the birds are veritable indicators of environmental problems. In this context, monitoring the populations of birds, their migration tendencies and changes of habitat becomes a necessity in conserving the natural ecosystems around the world.

1.2 Objectives

This paper aims to study the design and implementation of an intelligent classification system and to provide an efficient method for accurately classifying various species of European birds from high quality images, but also from pictures with average quality, such as the ones taken by a mid-range smartphone. The created system is desired to be integrated in a web application, such that persons interested in birds, for instance, the bird watchers, could easily determine the species captured in photos taken by themselves.

The main challenge in this problem is the large variation of possible images, caused by the diverse positions that the bird can have (flying, sitting with different orientations) and also by the fact that the pictures are taken at different moments of the day, in various locations and they may contain occlusions of the bird, generated by leaves, trees and other natural elements.

1.3 Thesis Structure and Original Contributions

The research in this thesis refers to the theory, design and experiments of several object classification models. The thesis is divided into 5 chapters, as follows:

- The first chapter (the current one) presents the introduction to the problem that is aimed to be solved and the objectives that are stated.
- The second chapter explores the state-of-the-art in the birds classification problem. Related articles were selected and the methodology and results claimed by their authors are presented.
- The third chapter focuses on the theoretical aspects behind the models and the experiments that are going to be done for this problem. The attention is focused mainly on three concepts: Artificial Neural Networks, Convolutional Neural Networks and Transfer Learning, as in image analysis transfer learning has given very accurate results when combined with deep learning architectures. This approach involves using a model that was already trained on a large data set of images, which will be then fine-tuned on specific tasks and features for the domain of interest.
- The fourth chapter will provide details about the personal experiments conducted in order to solve the birds classification problem, the results and about the application that was created afterwards. The original contributions are mostly included in this chapter, through the dataset, the fusion model and the integration of the model in an own web application.

- The last chapter offers personal conclusions and also perspectives about the future work that can be done in order to optimize the provided solution or to integrate it in more advanced systems.

Chapter 2

State of the Art

There have been various approaches using audio data rather than images for birds classification in the recent years. When classifying the species by sounds, the most common methodology is to generate spectrograms from the audio files, and to turn them into the input of a convolutional neural network. Even if this approach comes with several advantages, such as an unnecessary line of sight and the uniqueness of each species' call, there exists several drawbacks in using the sounds for the classification: the birds may not produce any noise over an extended period of time and the birds are harder to be identified by their sound even by the humans. The following section will describe the best solutions existent at this time for classification of birds from visual data.

2.1 Existing Solutions

2.1.1 VGG-16 Network For Classifying Birds Species From Images

Article [1] proposed a transfer learning approach for the bird classification problem. The used dataset consists of 1600 images of 27 species of birds, collected manually from various online sources (Google, Flickr, Pinterest). The images in the dataset have different backgrounds and surroundings and capture the birds in multiple stages of motion. The data was selected in such a manner that the classification will be performed considering mainly the characteristics of the birds, regardless of the elements in the background. In the pre-processing phase, the images were resized from an average dimension of 550 x 600 to a target size of 224 x 244, which is the required size of the input in a VGG-16 network.

The authors removed the last two layers of a VGG-16 network: a fully connected hidden layer with 4096 neurons and a fully connected output layer with 1000 neurons, corresponding to the number of categories in the ImageNet dataset. After dropping these two final layers, the output of the network will consist of 4096-

dimensional vectors with representative features. Further, three classification methods on the extracted features were applied.

Support Vector Machine (SVM) with linear kernel was used to find the optimal separation between the classes from the training sample. They also applied K-Nearest Neighbor (K-NN) algorithm which assigns the object to be classified to the class that is the most common among the k closest neighbors. The used metric was the Euclidian distance function and the chosen values for k were 5, respectively 10. Another classification method presented in this paper was the Random Forest technique in which after the features from the images are sent down to every decision tree, the response generated by the most trees yields the predicted class.

As shown in the Fig. 2.1, the highest accuracy (of 89%) was obtained with the SVM method. The Random Forest algorithm provided an accuracy of 87%, while the K-NN method had an accuracy of 85% and 82% for k = 10 and k = 5, respectively.

Method	Accuracy Rate	Standard deviation
SVM	0.89	0.03
KNN (k = 5)	0.82	0.02
KNN (k = 10)	0.85	0.01
Random Forest	0.87	0.02

Figure 2.1: Results obtained in [1], using different classification algorithms

2.1.2 Transfer Learning and Multi-stage Training For Birds Classification

Another relevant approach is presented in [2]. The authors used for the dataset samples from CVIP 2018 Bird Species challenge, consisting of 150 images split across 16 categories of birds for training, and 158 images for validation. The images' resolutions vary from 800 x 600 to 4000 x 6000. The training dataset was augmented with different techniques, including gaussian blur, gaussian noise, flip, contrast, hue, affine transform, reaching after this step a total of 1330 images.

Because of the diversity of backgrounds and surroundings, whose characteristics might have interfered with the birds' attributes, both training and testing images were cropped automatically such that the pictures would contain only the regions of interest, namely, the birds. This localization was achieved with Mask R-CNN, a model that was trained using the COCO dataset which consists of 80 object categories, for which 1.5 million object instances are available.

The actual classification is performed also based on a transfer learning approach.

Figure 2.2: Cropped images after the birds were identified by Mask R-CNN (Source: [2])

The used architectures are Inception V3 and Inception ResNet V2 which will form together an ensemble model. At the time of testing, a prediction vector is generated from both Inception ResNet V2 and Inception V3 weights. In the case in which Mask R-CNN manages to find one or more birds in the input image, a batch with the cropped bird images is generated and then evaluated using both network weights. The resulted prediction vectors are compared and the class corresponding to the prediction value with the highest weight is chosen as a final result. If the Mask R-CNN does not identify any bird in the image (though, the number of such cases is low) then the entire initial image is evaluated with the same method described above.

Model architecture	Data subset	Train	Validation	Test
Inception V3	Images	91.26	12.76	30.95
	Images + Crops	93.97	15.50	41.66
Inception Resnet V2	Images	**97.29**	29.17	47.96
	Images + Crops	92.29	**33.69**	**49.09**

Figure 2.3: F1-Scores obtained with Multistage Training (Source: [2])

The results of this approach are presented in the table from Fig. 2.3. The Inception ResNet V2 architecture provided the best accuracies. The cropping phase also increased the accuracy by 2-4%.

2.1.3 Classifying Birds Images Using a Transfer Learning Approach

A comparison between the results obtained using different pre-trained models is presented in [3]. The resources for the dataset are taken from Kaggle, from the 2020 version of Birds400 dataset. Over 27,000 images belonging to 200 different classes of birds will be used for training, while each of the testing and validation sets contains 1,000 images. The images are resized to the dimension of 224 x 224 and then

normalized to reduce the disturbances and noise. Data augmentation methods like rotation, flip, zoom or shift operations were also applied to the pictures.

CNN Model	Batch size	Training accuracy(%)	Testing accuracy(%)	Epochs	Learning rate
VGG16	**32**	**99**	**96**	**50**	**0.0001**
ResNet50	32	99	85	50	0.01
MobileNetV2	32	99	95	50	0.0001

Figure 2.4: The best results obtained with each of the three used models (Source: [3])

For the classification, three pre-trained networks are used, in order to compare their performances on this particular problem. The first experiments are conducted on a VGG-16 network whose last layer was modified to a Dense layer with 200 neurons (corresponding to the 200 categories of birds) having Softmax as activation function. The configuration with 50 epochs, a batch size of 32 and a learning rate of 0.001 provided the best results in this case, namely, a training accuracy of 99% and a testing accuracy of 96%. In the second group of experiments, the ResNet50 model was used. The best accuracies that were achieved are 99% for training and 85% for testing; they were obtained using 50 epochs, 32 samples in a batch and the learning rate equal to 0.01. For the last analysis, the authors chose MobileNetV2, which is smaller than the previous used networks with regard to the learnable parameters. The results are similar to the ones achieved by VGG-16. For the same configuration, the obtained accuracy for training is 99% and for testing 95%. These results are also shown in the form of a table in Fig. 2.4 and graphically in Fig 2.5.

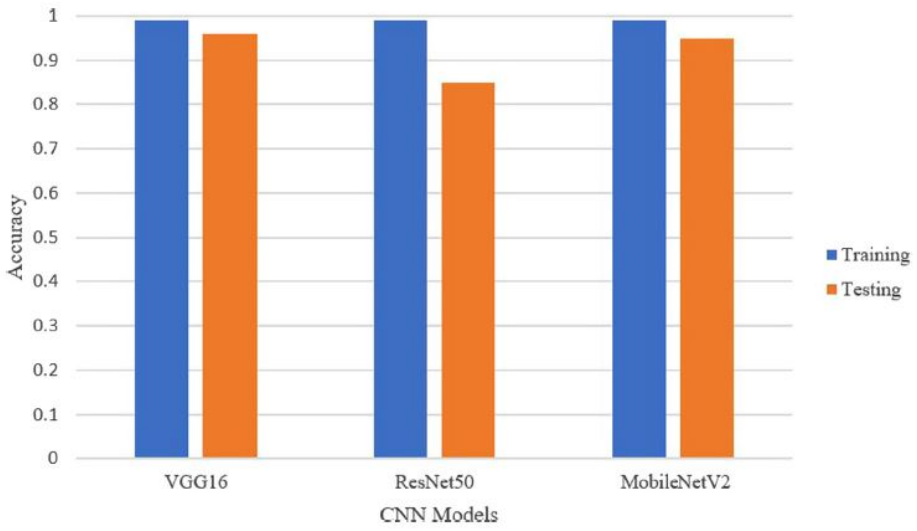

Figure 2.5: A comparative plot regarding the results achieved with the three networks (Source: [3])

Chapter 3

Theoretical background

3.1 Artificial Neural Network (ANN)

Inspired by the biological neural networks that make up the structure of the human brain, the Artificial Neural Networks have major importance in deep learning algorithms, being one of the most popular and efficient instances of learning systems.

An ANN is a collection of units or nodes, that are located in a graph with more layers and connected between them through direct links. Each link is also assigned a numerical weight, which determines the strength of the connection between two neurons. Therefore, an artificial neural network can be defined as a sorted triple (N, V, w), where N is the set of neurons and V is a set of tuples denoting the connections between neuron i and neuron j, $V = \{(i, j)|i, j \in N\}$. The function $w : V \to \mathbb{R}$ expresses the weights, and $w(i, j)$ is the weight corresponding to the link between neuron *i* and *j* [16].

3.1.1 Artificial Neuron

Fig. 3.1 shows the similarities between the biological neurons (a) and the artificial neurons (b). The artificial neurons were designed in a way that matches the structure of the biological counterparts. The dendrites in the biological neuron correspond to the input vectors in the artificial model, as their role is to allow the cell to receive signals from the neighbouring neurons. The cell body (or soma) is responsible for processing the received signals and for determining whether the information should be transmitted further or not. The equivalent of the soma is the node. The axon correlates with the output since its biological role is to carry the information processed by the soma to other neurons. The synapses are transposed into weighted connections.

The data processing at the neural level is performed in three phases: network's input preparation, activation and retrieval of the output. In the first step, the input

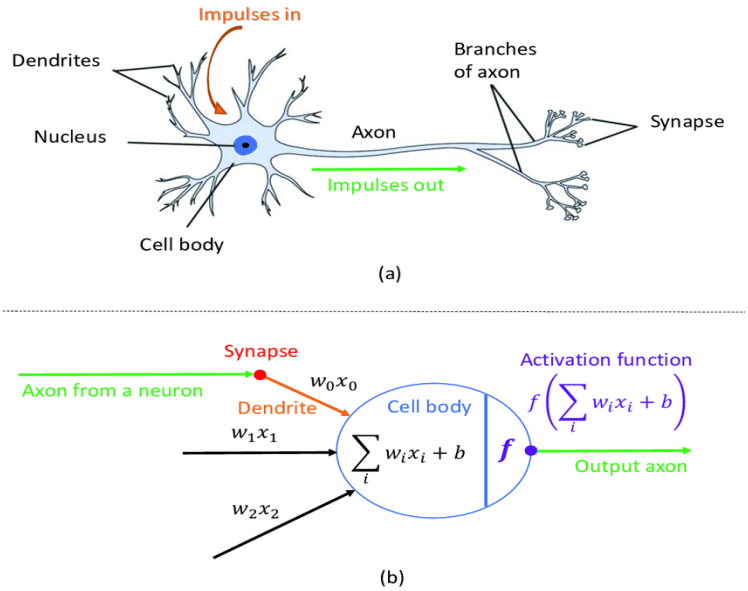

Figure 3.1: The correlation between the Biological Neuron (a) and the Artificial Neuron (b) (Source: [4])

vectors are transformed into scalar inputs using the propagation function. Each neuron i receives as input multiple signals $(x_1, x_2, ..., x_n)$ representing the output of the other n neurons with which it is connected. The propagation function maps these outputs into the network input net_i that will be further processed by the activation function, taking into account the weights of the links between the current neuron and its neighbours. In general, the used expression for the propagation function is the weighted sum:

$$net_i = \sum_{j=1}^{n} (x_j * w(j,i)) \tag{3.1}$$

In the second phase, the activation (or transfer) function is applied to the neuron's input. In nature, each neuron is either excitatory or inhibitory, meaning that its activation state will increase or, respectively, decrease the firing rates of the linked neurons. The activation function simulates this behaviour, indicating the extent of a neuron's activity [16]. The neuron i will be activated when its input surpasses the threshold value which is uniquely assigned to i. The threshold values can be adjusted during the learning process. To simplify these modifications, usually it is used a bias neuron that allows the threshold values to be considered connection weights. The bias neuron is a neuron connected to all the neurons in the network, whose output is always 1. Therefore, 3.1 becomes:

$$net_i = b + \sum_{j=1}^{n} (x_j * w(j,i))$$

where *b* represents the bias value.

Lastly, the output function computes from the activation state the output value that will be transmitted to the neurons connected to the current neuron. It is usually the identity function, thus the output is directly the activation state.

Common activation functions

The activation functions may be linear or non-linear. The non-linear functions are more widely used, as they overcome certain drawbacks of the linear ones such as the impossibility of using backpropagation. Another drawback is the collapsing of all the layers from a network into a single layer, as the last layer becomes a linear combination of only the first layer. Some of the most used activation functions are:

- Sigmoid Activation Function – it is a non-linear function that maps any real number into the interval $(0, 1)$. It is monotonical, continuous and differentiable over the whole domain of real numbers. Moreover, its derivative can be expressed in terms of itself, facilitating the use of the backpropagation algorithm that utilizes gradient descent. Because of its codomain, this function is commonly used when the output a model must predict is a probability.

- Hyperbolic Tangent (Tanh) Activation Function – it resembles the sigmoid function, but its output values are placed between -1 and 1 (it is zero-centered, thus the values can be mapped as strongly negative, neutral and strongly positive [17]).

- Rectifier Linear Unit (ReLU) Function – it has become one of the most popular activation function in the last years, being used often in computer vision and deep learning. It is more computationally efficient than sigmoid or tanh functions. Moreover, compared to sigmoid and tanh functions which saturate in both directions, ReLU suffers significantly less from the vanishing gradient problem [18]. As a disadvantage, it is not differentiable at 0 and moreover, because the gradient is 0 for negative numbers, some weights might not get adjusted during the backpropagation phase, causing the neurons to become stuck in an inactive state (the *Dying ReLu* problem).

- Softmax Function – it maps a vector x of N elements into a normalized probability distribution with N probabilities, its expression being:

$$\varphi(x)_i : \mathbb{R}^N \to (0,1)^N; \varphi(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N}(e^{x_j})}$$

for i = $\overline{0, N}$ and $x = (x_1, x_2, ..., x_N)$. The values in the output belong to $(0, 1)$ and their total sum is 1. The Softmax function is used as the activation function of

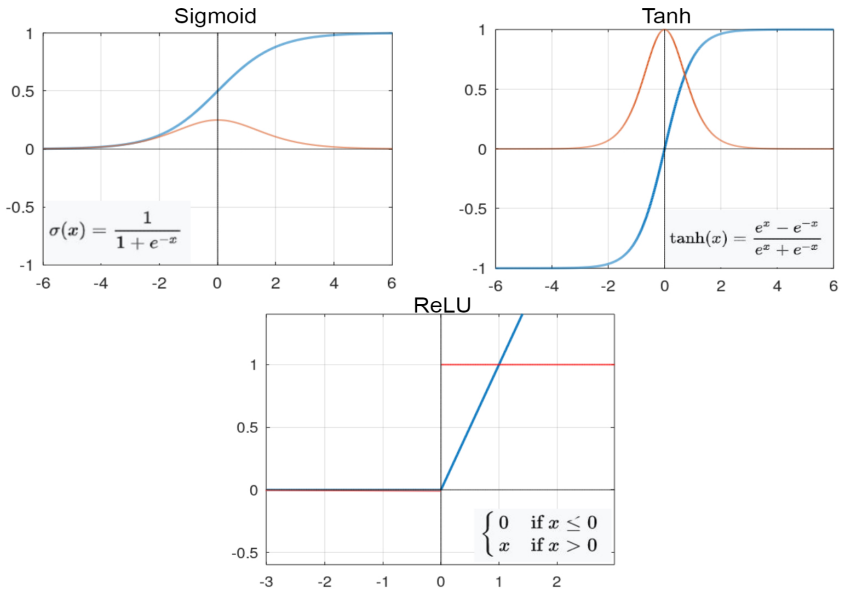

Figure 3.2: The expressions and plots for the Sigmoid, Tanh and ReLU activation functions. The blue lines represent the functions and the red lines their derivatives.

the last layer in the multi-class classification problem, where the result is the class to which corresponds the maximum probability.

3.1.2 Neural Networks Architectures

The flow of impulses in neural networks can be either in only one direction or in recurrence. Depending on this aspect, we define multiple categories of neural networks.

The **Feedforward Neural Network** is one of the first designed networks. In this topology, the layers in the network are clearly delimited: one input layer receives the input signals which will be processed further by one or more hidden processing layers, that are followed by one output layer. Each neuron from a layer is allowed to have direct connections only with the neurons from the next layer, towards the output layer.

In the case of **Recurrent Networks**, a neuron can feed back the signals into another neuron or layer that has already processed that signal, creating a sort of internal recurrence. The addition of this recurrence to a neural network is necessary for producing a dynamic behaviour in case of problems requiring an internal memory to reinforce the learning process, such as pattern recognition or time series [19].

3.1.3 Learning in Multilayer Networks

In the learning phase, the processed data is transmitted from the input layer to the hidden layers and then further to the output layer. At each layer, the propagation function (weighted sum) is applied and the activation function is propagated to the next layers. This step is called forward propagation.

Once the signals reach the output layer and an output is obtained, the error is calculated through the cost function which evaluates the performance of the neural network. The expression of the cost function can be the Mean Squared Error:

$$C_{MSE} = \frac{1}{2m} * \sum_{i=1}^{m} (Y_i - \hat{Y_i})$$

where m is the number of samples used in training, Y_i is the predicted value for the ith sample and $\hat{Y_i}$ is the actual value (the ground truth) that corresponds to the ith sample. Another cost function (used in the classification models) is the Cross-Entropy cost function. Considering p the distribution for the actual value, q the probability distribution predicted by the neural network and n the number of possible classes, the cross-entropy is defined as follows:

$$H(p, q) = -\sum_{i=1}^{n} p_i(x) log(q_i(x))$$

The learning process generally implies the minimization of the cost function. For this purpose, the backpropagation algorithm is used.

Backpropagation is the most used algorithm for supervised learning in multilayer feedforward neural networks. The algorithm may be described as follows: firstly, compute the errors corresponding to the output units; then, starting with the output layer, propagate the error values back to the previous layer and update the weights between the current two layers accordingly, until the first hidden layer is reached [20]. The idea that lies at the heart of the backpropagation is to repeatedly apply the chain rule of differentiation in order to compute the influence of each weight in the neural network with respect to the cost function. Thus, backpropagation provides information about how changing the weights and biases affects the overall behaviour of the model.

The Gradient Descent is an optimization algorithm for minimizing an objective function parameterized by the weights of a network by updating the parameters in the direction of the steepest slope of the function with respect to its parameters [21]. By using the derivatives, it provides two decisions: in which direction to move and how large the step should be in order to reach the minima of the cost function. This method also uses as hyperparameter a learning rate that controls the modification

step. A higher learning rate might provide a faster convergence, but also a risk of overshooting the minima and diverging, as shown in Fig. 3.3.

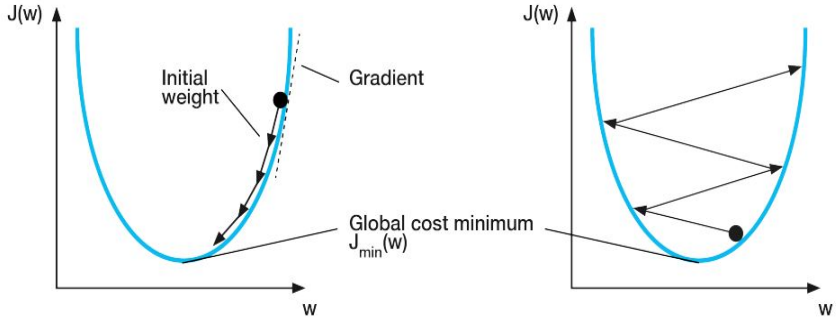

Figure 3.3: Gradient Descent with a) an optimal learning rate and b) a learning rate that is too high (Source: [5])

Depending on how much data is used when computing the gradient of the cost function, three variants of gradient descent can be distinguished. The Batch (or Vanilla) Gradient Descent (BGD) relies on the mean of all the individual losses in the entire dataset when optimizing the model. In contrast, the Stochastic Gradient Descent (SGD) takes only one training sample and performs the gradient calculations. Because of the frequent updates, the convergence towards the minima in the case of SGD is noisy. To obtain a more stable convergence but to preserve the advantages in terms of efficiency of the SGD (computationally faster, easier to fit in memory), the Mini-Batch Gradient Descent can be used. It is a mixture between the BGD and SGD, splitting the training dataset in smaller groups, called batches. The updates are performed for every mini-batch of training samples.

Overfitting Problem

When a model is too complex for its training dataset, the problem of overfitting might occur. An overfit model will generate a poor predictive performance, because it overreacts to minor fluctuations and changes in the input data [4]. To reduce overfitting, one may increase the number of samples in the dataset or decrease the complexity of the model by excluding some layers, action that will result in a smaller number of parameters in the network. Besides these techniques that can be undesired (if we want to keep fixed the training dataset and the network depth), there are some regularization methods, such as considering some percent of the hidden neurons dropped out (Dropout) or adding an extra parameter to the cost function (L1 and L2 regularization), that may be used.

3.2 Convolutional Neural Networks (CNN)

The concept of Convolutional Neural Network was introduced by Yann LeCun in 1998, in paper [22]. The authors propose the convolutional neural network as a solution to the handwritten characters classification problem. Being specifically designed to handle the variability of two dimensional shapes, the CNN outperformed all of the other used techniques.

The CNNs represent a specialized type of deep feedforward neural networks that use the mathematical operation called convolution for extracting high-order features from the input data. They were particularly created for processing visual data and in a short amount of time, they become a disruptive technology, breaking all of the state-of-the-art results in various domains far beyond the initial image processing domain where they were created [23]. Such domains in which the CNNs were proven to perform well are video analysis, natural language processing, speech recognition or time series forecasting. Moreover, they become even more efficient and, therefore, popular because the standard CNNs can be significantly accelerated when run on GPUs (for example, [24] shows that the GPU implementation of standard CNN was 20 times faster than the equivalent CPU implementation).

The inspiration for the CNN lies in the discovery made by David H. Hubel and Torsten Wiesel from 1959 [22]. They showed that there are specific cells in the visual cortex of animals that individually respond to small regions of the visual field. The area of the visual space in which visual stimuli affect the activity of a neuron is known as the receptive field of that neuron. The cells act as local filters over the input visual space, similar to the neurons in CNNs.

The following subsections will provide details about using convolutional neural networks for image analysis, as the main purpose of this paper is finding an accurate and efficient solution for visual recognition.

3.2.1 Architecture of Convolutional Neural Networks

There exists a large variety of CNN architectures, but they all follow the same pattern of layers [7]. In essence, the CNN translates the data from the input layer into a set of class scores computed in the output layer, passing the data through all connected layers. Three major groups of layers can be distinguished: the input layer, the feature-extraction (learning) layers and the classification layers, those being shown also in Fig. 3.4.

The **input layer** receives the representation of an image. The input is three-dimensional, describing generally the spatial form of the image (width x height) and the depth corresponding to the color channels. The number of channels that are

associated with a pixel specifies how many numerical parameters are required for obtaining the color of that pixel (what amount of each color is needed to obtain that color). In the case of RGB, for example, the pixels in the image are a combination of 3 primary colors: red, green and blue. Thus, a RGB picture will have three channels, one for each such color (red, green, blue). Additionally, another dimension might be added to the input, in the case in which the training samples are batched together into mini-batches. The fourth dimension will represent the index of the sample within the mini-batch.

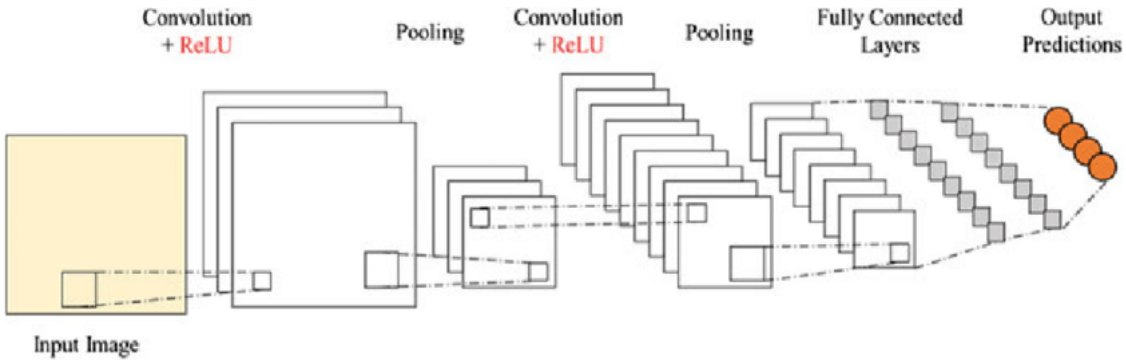

Figure 3.4: The architecture of a typical CNN model (Source: [6])

The **feature-extraction component** is the core of the convolutional neural networks. Essentially, the machine learning takes place in this phase, as the patterns are automatically learned, unlike in the traditional algorithms where they were hand-designed. These layers search for various features in the pictures and progressively construct features of a higher order, performing in this manner the automatic learning.

The layers belonging to this segment receive as input one or more tensors and output, likewise, one or more tensors. A tensor can be defined as a generalization of a matrix, where each element is accessed in terms of an arbitrary number of indices, because of the varying dimensionality of the data. It can be visualised as a series of matrices, one on top of others. The feature-extraction layers follow a general repeating pattern of a sequence formed by a convolutional layer and a pooling layer.

Convolutional Layer

The convolutional layer is the main building block of a CNN, doing most of the computational effort. It contains neurons that perform the mathematical operation called convolution (known as feature detector) in order to process, filter signals and detect matching patterns in the input. The learnable parameters in the convolutional layers are the filters. The filters are spatially small (in terms of width and height), usually square shaped. Their size is given by a special hyperparameter, called filter

size. The filters are used to process a finite region of the input space, by sliding it over the width and the depth of the image. At each step in the sliding process, the dot product between the values from the filter (the set of weights) and the data in the covered input region is computed. The result of this operation will represent a singular entry of the output activation (or feature) map. The convolutional operation is depicted in Fig. 3.5. The activation maps that are obtained for each filter are stacked together along the depth dimension of the output, in order to construct the third dimension of the output shape.

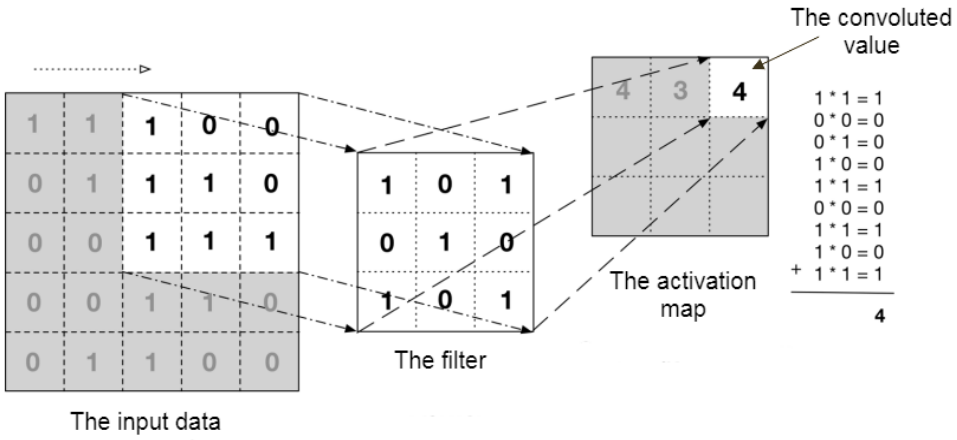

Figure 3.5: The convolution operation (Source: [7])

Some of the most important properties of a convolutional layer are the local connectivity, the spatial arrangement and the shared weights and bias [25].

The local connectivity is controlled with the receptive field. When managing high-dimensional inputs, images being such an example, it may be inefficient and impractical to create connections between every neuron and all the neurons from the previous layer. Rather than that, each hidden neuron is connected to only a submatrix of adjacent input neurons or, in other words, to a local region in the input. This connectivity will generate a spatial extent called receptive field, this being a hyperparameter. The shape of the receptive field will be given by the filter size of the layer. The connections will be local in the two-dimensional space (on width and height), and will be always complete along the depth axis; therefore, the extent of the connectivity will have the same width and height as the filter and the same depth as the input [25].

The spatial arrangement property refers to what is the number of neurons in the output volume and how they are organized. There are three hyperparameters that affect these aspects. The depth of the output corresponds to the number of used filters, each of which learns different features in the input image. The stride controls how many unit steps the filter window will shift each time when the filter function

is applied. For example, considering the stride equal to two, the filter shifts two units at a time. Increasing the stride value will result in smaller spatially output volumes. The padding prevents the filter from moving out of the input data. Applying the padding to an image represented as a matrix of pixels means adding a suitable number of rows and columns around it, such that the convolution process can be properly applied to each pixel individually. The padding helps control the spatial size of the output volumes, especially in the cases in which the intention is to keep the spatial size of the input volume equal to the size of the output volume [7].

Lastly, the parameter sharing scheme is used for controlling the total number of parameters. The parameters count can be significantly reduced, because the utility of computing a feature found at the position (x, y) may be also present in the case of another position, (x', y'). Thus, for detecting the same feature independently of the position where it is located, a simple intuition is to use the same set of weights and bias term for all the neurons in the hidden layer. Denoting an individual two-dimensional slice of depth as a depth slice (for example, a volume of size $(50 \times 50 \times 32)$ has 32 depth slices, each having the size equal to (50×50)), the constraint according to which the neurons in each depth slice share the same set of weights and bias is applied. In practice, in the backpropagation algorithm every neuron from a specific volume computes the gradient corresponding to its weights, but the computed gradients are added up across each depth slice and only a single set of weights per slice will be updated[25].

Given that the input size of a convolutional layer is $W_i \times H_i \times D_i$ and that it uses four hyperparameters – the number of filters K, their spatial extent F, the stride S and the padding P – the dimensions of the output, $W_o \times H_o \times D_o$, are given by the formulas:

$$W_o = \frac{W_i - F + 2P}{S} + 1 \tag{3.2}$$

$$H_o = \frac{H_i - F + 2P}{S} + 1 \tag{3.3}$$

$$D_o = K \tag{3.4}$$

The total number of weights for every filter is equal to $F * F * D_i$, thus in the entire convolutional layer there will be $(F * F * D_i) * K$ weights and K biases.

The ReLU layers are commonly utilized in convolutional neural networks. Over the input data thresholding the ReLU layer applies an element-wise activation function. Running this function over the input volume changes the numerical value of the pixels but the spatial dimensions are maintained the same when the result is returned. There are no parameters or additional hyperparameters in ReLU layers.

Pooling Layer

Inserting a pooling layer after successive convolutional layers is a common approach. The reason for following convolutional layers with pooling layers is to progressively reduce the two-dimensional spatial size of the input volume and to control overfitting, reducing the number of parameters and, thus, the number of computations inside the convolutional network.

The pooling layer operates on every depth slice of the input individually. The most common downsampling techniques are max and average pooling. In the max pooling process, a filter is moved across the output of the convolutional layer with a stride of the same size and the maximum value from the covered region of pixels is selected. The average pooling uses the same principle, but the output corresponding to a specific region will be computed as the average of the values from that region. The most used dimensions for the filter in the pooling layers is the size 2×2, with a stride of 2. This configuration downsamples each depth slice from the input, on the spatial dimensions, by a factor of two, dropping 75% of the activations [7]. The depth dimension is kept unchanged, only the spatial dimensions being affected by the pooling operation.

The pooling layer can solve the issue in which the objects in the input images are not perfectly centered in every single picture. This process can be seen as a translation, as a method used by the network for funneling the relevant information in the right position in every input that it receives.

Flatten

The flattening step is a simple step involved in building a convolutional neural network. It implies transforming the generated pooled feature map into a one-dimensional vector, that will be further fed into an artificial neural network.

Dropout

If overfitting occurs, then the predictions made by the model will be accurate only for the data on which it has been trained, any minor change in the data causing poor predictions. To prevent this problem, in the convolutional neural networks a dropout layer may be used. A dropout layer will set some of the activation values to zero, avoiding in this way the risk of having a network that is too fitted to the training samples. In other words, some neurons will be ignored during the training process. The aim is to obtain a network that is capable of making predictions even when having some activations dropped out. Models containing dropout in the hidden layers will accomplish better generalizations on data they have not seen before.

Batch Normalization

To speed up the training process in the CNNs, at each batch the activations of the previous layer can be normalized. This method will apply a transformation which keeps the mean activation near 0 and the standard deviation just about 1 [7]. Higher learning rates can be used if normalization is applied for each mini-batch of input records in the training. Batch normalization decreases also the sensitivity of the training process regarding weight initialization and plays the role of a regularizer, as it decreases the need for using other types of regularization.

Finally, the **classification component** is placed at the end of the convolutional network, consisting in a fully connected layer, that has a connection between all of its neurons and every neuron in the previous layer. There exist some architectures of CNN that contain multiple fully connected layers in the end of the network. The last layer in the classification part computes the class scores that will represent the output of the neural network. The output volume will have a dimension equal to $(1 \times 1 \times n)$, where n represents the number of possible output classes that are evaluated.

Fully connected layers have both hyperparameters and normal parameters for the layer. They apply the activation function to the input volume and its parameters, namely the weights and the biases of the neurons, to perform transformations on the input data volume.

3.3 Transfer Learning

Researchers have recently paid increasing attention to deep learning, which was proven to be a successful solution to a variety of real-world applications. However, the data dependency is a serious problem in deep learning, since a considerable amount of training data is needed for understanding of high-level features and patterns in the input. Moreover, most machine learning models perform well when the data for training and validation is extracted from the same feature space, following a fixed distribution. If the distribution suffers changes, the need for collecting new data and reassembling the model might occur [26].

Transfer learning is an useful approach in machine learning which aims to solve the problem of insufficient data. In this approach, a model trained on a specific task is repurposed on a second, related task, creating an optimization that allows rapid progress in modeling the second task. In other words, this method tries to transfer the knowledge from a source domain to a target domain, a significantly higher performance on the new task being achievable, even with a small amount of

data [8]. The process is illustrated in Fig. 3.6.

A more formal definition uses two important notions: a) a domain which can be represented as $D = \{\chi, P(X)\}$, containing two parts: a feature space χ and a marginal probability distribution $P(X)$, with $X = \{x_1, ..., x_n\} \in \chi$ and b) a task defined by $T = \{\gamma, f(\cdot)\}$, γ being the label space and f(·) the objective predictive function (from a probabilistic point of view, it can be written as $P(y|x)$) which is to be learnt from the training data that consists of pairs of the form $\{x_i, y_i\}$, with $x_i \in X$ and $y_i \in \gamma$. Thus, given a source domain D_S and a learning task T_S, a target domain D_T and a learning task T_T, the transfer learning process aims to help improve the learning of the objective predictive function $f_T(\cdot)$ in D_T using the knowledge provided by D_S and T_S. We consider that $D_S \neq D_T$ (the feature space between the domains are different or the marginal probability distributions between the two domains are different) or $T_S \neq T_T$ (the label spaces of the tasks are different or the conditional probability distributions are different) [27]. In addition, in most the cases, the size of the source domain D_S is much larger than the size of D_T.

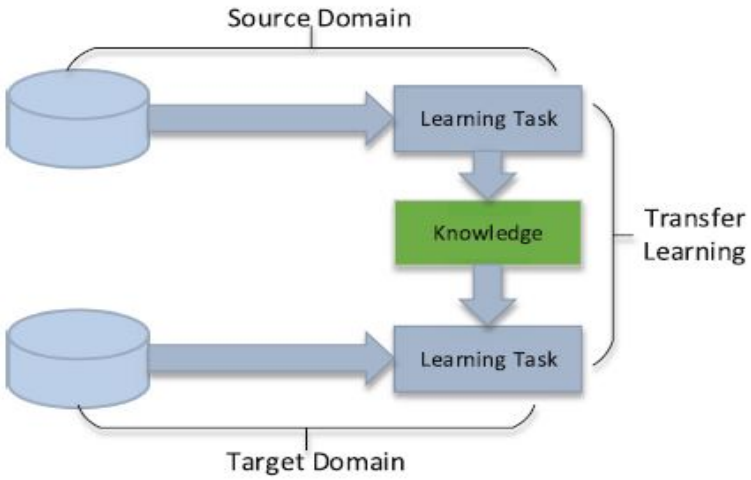

Figure 3.6: The transfer of knowledge in transfer learning (Source: [8])

The transfer learning implies three perspectives: when, what and how to transfer [27]. The first issue means in which situations the transferring of knowledge shall or shall not be done. There exist cases in which the source and target domain are not related to each other, therefore a brute-force transfer not only that would not be effective, but it might even damage the performance of learning in D_T, resulting in a negative transfer. The second research perspective refers to what parts of knowledge are transferable between the domains or the tasks. Some knowledge could be specific for a given domain or a given task, while other may be common across certain domains, thus the performance of learning in the target domain and task can be significantly improved. Further, the ”how to transfer” problem involves

the algorithms that need to be developed for transferring the knowledge.

Depending on the relations established between the source and target domains and tasks, various settings for transfer learning can be identified: inductive, transductive and unsupervised transfer learning.

The inductive transfer learning reflects the case in which the source and target tasks are different ($T_T \neq T_S$) and the source and the target domains might be the same or not. One common example of inductive learning is using a deep learning model that was trained on a source domain and task and fine-tuning different layers in the model for the target task. If labeled data in the source domain is available, then this setting is similar to the multitask learning, with the difference that in the multitask learning both source and target tasks are done at the same time. Otherwise, if not enough labels are available in the source domain, the learning is similar to the self-taught learning. The source task can be trained to recognize patterns in the unlabeled data, and these patterns can be used further within a supervised learning task in the target [28].

In the transductive transfer learning setting, the source and target tasks are similar but the domains differ. In this case, there is no or a few labeled data in the target domain but a large amount of labeled data in the source domain. An example of transductive transfer is the use of synthetic images in a source task to improve the performance in a target task involving real world images.

Lastly, the unsupervised transfer learning setting is similar to the inductive one, in the sense that the source and the target tasks are distinct, but the difference is that unsupervised transfer learning aims to solve unsupervised tasks in the target domain, such as clustering and dimensionality reduction. Both domains do not own labeled data.

3.3.1 Transfer Learning with Convolutional Neural Networks

Transfer learning with CNN implies that the knowledge can be transferred at the level of parameters. The well trained CNN models use the parameters of the convolutional layers to solve a new task in the target domain. There are two kinds of approaches to leverage the CNN models: feature extraction and fine-tuning. In the feature extraction case, the layers of the pre-trained model are frozen, meaning that their parameters are not updated during the backpropagation phase. It is a good strategy for avoiding overfitting. In contrast, the fine-tuning approach updates the parameters provided by the pre-trained model during the model fitting. This method allows also the freezing of certain layers together with the fine-tuning of the rest of the layers, depending on the needs of the problem. The states of the network become the starting points in the retraining step, resulting in a better per-

formance within a shorter training time. Various combinations can be done with these techniques, some of them being depicted in Fig. 3.7. In the approach from Fig. 3.7a the fully connected layers from the pre-trained model that were used for classification are discarded and another machine learning algorithm (SVM, Random Forest) is attached to the feature extractor. In the other approaches, the skeleton of the model remains the same. Fig. 3.7b uses the same feature extraction approach, feeding the extracted results in new fully connected layers that will be trained on the new specific task. Fig. 3.7c and d represent the schemes in the fine-tuning method: in c only some layers from the pre-trained model are fine-tuned, the rest of them being frozen, while in d the whole model is fine-tuned from scratch, a procedure that is the most time-consuming, as it updates the whole set of parameters during training.

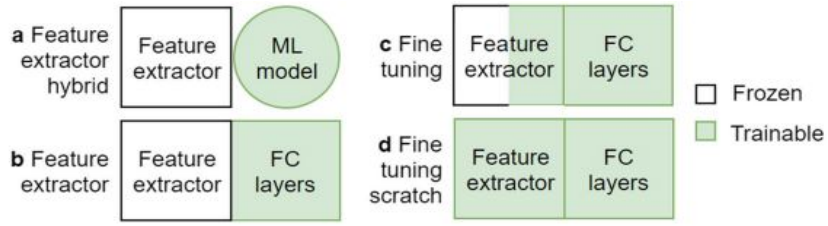

Figure 3.7: Four types of transfer learning approach (ML: Machine Learning, FC: Fully Connected) (Source: [9])

3.3.2 Pre-trained Models

There are various deep learning networks and architectures with state-of-the-art performance that have been developed in the domain of computer vision. In the computer vision field, the most popular pre-trained architectures are VGGNet, Inception, XCeption, EfficientNet, MobileNet. This thesis will focus more on the Inception model, as it was the model chosen for the conducted experiments.

Inception Architecture

The Inception model was proposed in [10] for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). The authors stated that increasing the size of a deep neural network is the most straightforward method to improve its performance. However, this approach comes together with two drawbacks: an increased number of parameters, which makes the network prone to overfitting, especially if the samples in the training data are limited (there are situations when the enlarging of the training set is expensive – for instance, if it involves fine-grained visual categories) and also the increasing of the computational

resources. One solution to this problem would be to change the fully connected to sparsely connected architectures, even inside the convolutions.

The Inception architecture is based on the idea of finding how to approximate and cover an optimal local sparse structure in a convolutional network with readily available dense components. Because of the significant variation in the location of the information of interest in an image, it may become hard to choose the right filter sizes for the convolution operation. If the information is distributed more globally, then a larger filter is preferred, whereas a smaller filter is preferred for information that is distributed more locally. The solution provided by the Inception architecture is to have filters of different sizes that operate on the same level, getting rather "wider" than "deeper" in the data.

The Inception module is represented in Fig. 3.8. It performs the convolution operation on the input using filters of three different sizes: 1×1, 3×3 and 5×5. The 1×1 convolution occurs between the input data and a filter with the spatial dimensions of 1×1, to generate an output of the form $1 \times 1 \times K$, with K being the number of filters. This filter is used for dimensionality reduction, to decrease the computational cost that would be generated by the 3×3 and 5×5 convolutions. Moreover, although it does not learn any spatial patterns that occur in the input data, it learns patterns across the depth of the input, enabling the network to learn more and, thus, becoming dual-purpose. The 3×3 and 5×5 convolutions are used for making the network learn various spatial patterns at different scales, as a result of varying the sizes of the filters. The learnt patterns are distributed across all dimensional components of the input: the width, height and depth. A naive approach, that would perform the convolution operation between the filters and the raw input, would result in a considerable increase of the computational cost. This is why the filters of the sizes 3×3 and 5×5 are applied on the so-called bottleneck layers, which represent the output with a reduced depth dimension resulted from the convolution between the input and the 1×1 filters. Additionally, an alternative parallel pooling path in an Inception module is considered to have a supplementary beneficial effect. Eventually, the Inception module is a combination of the above described layers, with their output feature maps concatenated into a single output vector that will form the input of the next stage. To ensure that all of the output feature maps have the same width and height (to make possible their concatenation), the padding is used.

An Inception network is a network consisting of such modules stacked upon each other, with occasional max-pooling layers with stride of 2, to halve the resolution of the grid [10]. The proposed Inception network was the GoogLeNet, which consists of 9 Inception module stacked linearly, being 22 layers deep.

Further improvements of the Inception architecture have been presented in [11], in terms of both accuracy and computational complexity. The Inception-v2 explores

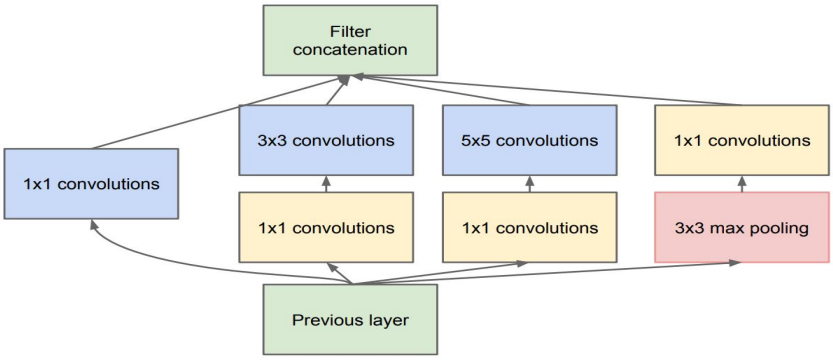

Figure 3.8: The Inception module with dimension reduction (Source: [10])

the upgrades made in order to decrease the computational costs. The modifications are represented in Fig. 3.9. Firstly, as reflected in Fig. 3.9a, the 5×5 convolution was factorized in two 3×3 convolution operations to improve the computational speed. A gain of 28% in efficiency was reported. Moreover, the convolutions having the filters with the size $n \times n$ were factorized into a combination of $1 \times n$ and $n \times 1$ convolutions (Fig. 3.9b), this method being cheaper by 33% than the initial version in the case of $n = 3$. Lastly, the filter banks in the module were expanded such that they would become wider instead of deeper, to avoid an excessive reduction in dimension and, thus, the loss of information.

The Inception-v3 model includes all the upgrades stated for Inception-v2 and additionally uses the RMSProp optimizer, factorized 7×7 convolutions and Batch Normalization in the fully connected layer of the auxiliary classifier, not only in the convolutions. Another regularization applied in Inception-v3 is label smoothing, a technique through which the model is encouraged to be less confident, by adding a regularizing component to the cost function, preventing overfitting.

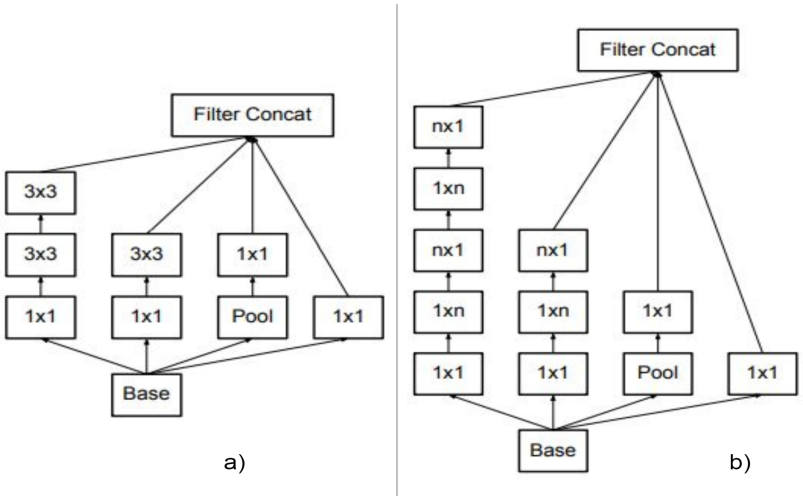

Figure 3.9: The improvements performed on the Inception module (Source: [11])

Chapter 4

Birds Classification Application

4.1 The AI Model

4.1.1 Data

The first dataset used in this thesis was the Birds400 dataset from Kaggle [29]. It contains data about 400 species of birds: 58,388 images for training, 2,000 images for validation and 2,000 images for testing (5 images per species). Each image is square shaped, having the dimension 224×224, and contains only one bird, which occupies at least 50% of the pixels in the picture. Due to this fact, the accuracy obtained by the model is expected to be high.

Using [30] and [31] I manage to select from the dataset only those species that live in Europe. Moreover, for certain species, there were some mentions to be made regarding their habitat or lifestyle:

- *A* - species that accidentally or rarely are found in Europe
- *I* - species introduced in Europe as a consequence of human activity
- *E* - species that are endemic in Europe
- *Ext* - extincted species
- *Ex* - species that were locally extincted in Europe (extirpated)

After applying the filter according to which the birds from the dataset must be found in Europe, in the original images dataset only 116 categories were left, together with 17,168 images for training, 580 images for validation and 580 images for testing. For the further experiments, the birds that appear in Europe only rarely or accidentally (those with mention *A*) were eliminated and thus 57 categories of birds remained in the dataset. For matching the data in the second dataset (described below), the categories were reduced to a number of 52. The data for validation was

merged with the images aimed for training, such that in the experiments the validation data would be chosen randomly. The remaining 52 species are represented by 7,617 images. Samples from the obtained dataset are shown in Fig. 4.1.

Figure 4.1: Training images from the Birds400 dataset

Due to the fact that the Birds400 dataset contains only high quality images, in which the bird occupies the most of the foreground, another set of images with more examples from real life was considered to be necessary for more accurate predictions. The real life cases may include situations in which parts of the birds are hidden or camouflaged behind some natural elements, the birds are captured while flying and not in an ideal static position or in which the pictures might have average and even low quality. Such situations are covered by the newly created dataset.

The second used dataset is a subset of an upgraded version of the iNaturalist dataset published in 2017 [32]. It contains observations regarding approximately 10,000 species of natural beings, taken by multiple users from iNaturalist. The original subset concerning birds was downloaded from [33], incorporating about 100,000 images distributed across 1,486 species of birds. I implemented a Python script to automatically identify the birds of interest (the species that were selected also from the Birds 400 dataset), using their scientific names which could be found in the names of the folders. Then the photos were filtered, because some of them were invalid or irrelevant in the context of the species classification problem. Photos with the birds' eggs, feathers or in which the bird was very hard to be seen even by the human's eye were removed (some examples are shown in Fig. 4.2).

This operation led to a total of 52 species represented by 6,362 images. Due to the limited resources available, the photos are not equally distributed across the categories, the distribution being reflected in Fig. 4.3. To solve the data imbalance problem, techniques for data augmentation were applied on the dataset, in order to

Figure 4.2: Invalid images from the iNaturalist dataset that were eliminated

generate 300 images for each category.

The fusion model combines the Birds400 and iNaturalist datasets, using samples from both sources, in order to provide more accurate predictions. A contrast between the images from the two datasets can be observed in Fig. 4.4, where two species were chosen for exemplification. The photos with Great Gray Owl and Pelican from the left (Birds 400 dataset) are much more qualitative than the ones from the right, taken from the iNaturalist dataset.

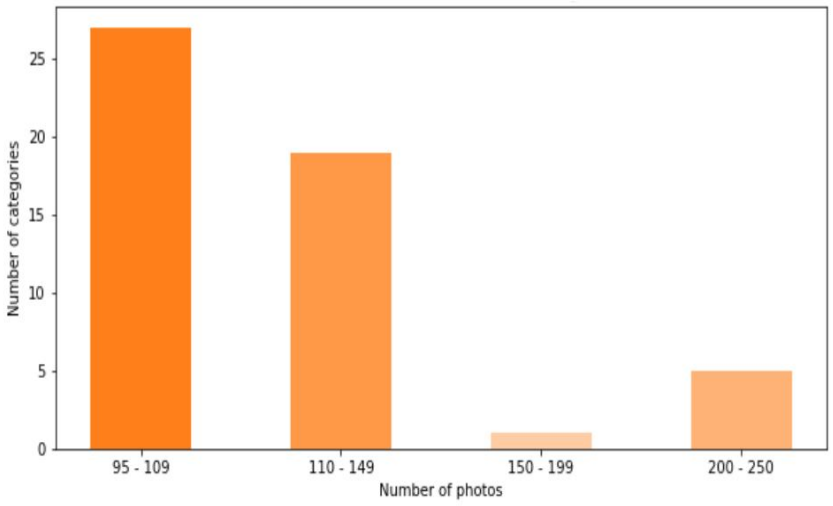

Figure 4.3: Data distribution across birds categories in the iNaturalist dataset

For testing purposes, 15 images for each category were selected from the images dd dedicated for test in the Kaggle Birds400 dataset and from eBird website [34]. They represent both high and average to low quality images.

Figure 4.4: Samples of the same species (Great Gray Owl - top; Pelican - bottom) taken from the Birds 400 dataset (left), respectively from iNaturalist dataset (right)

4.1.2 Technologies

The AI models were implemented using Python programming language. For creating and training the models, TensorFlow was used. Tensorflow is an open source machine learning framework that facilitates the implementation of machine learning models and enables fast execution of numerical operations with large amounts of data. It simplifies and offers flexibility in all the stages of model development, from collecting data to training and refining the created model.

Keras is a deep learning API written in Python, running on top of TensorFlow. It was developed with a focus on enabling fast experimentation, as being able to go from idea to result as fast as possible is essential in doing good research. It provides implementations for most of the transfer learning models.

The experiments were conducted on online notebooks, namely Google Colaboratory and Kaggle Notebook, as the GPU support that they offer significantly reduces the running time, compared to running locally on my machine. The Google Colaboratory uses Nvidia K80 having a memory of 12GB while Kaggle Notebook provides a Nvidia P100 GPU with a memory of 16GB.

4.1.3 Experiments

The first experiments used the Birds400 dataset and the 116 categories of birds that appear in Europe. The validation images were the ones provided by the dataset, as it was already split in train, validation and test data. The experiments used a pre-trained model as feature extractor (thus, all the layers in the pre-trained model were frozen) and a classification subnetwork, formed by fully connected layers and dropout layers, that would be fed with the features provided by the pre-trained model. The pre-trained models used in this preliminary phase are VGG16, MobileNet and InceptionV3. The resulting networks used Adam optimizer and categorical crossentropy loss and were trained during 20 epochs. The best results provided by each model are shown in Fig. 4.5.

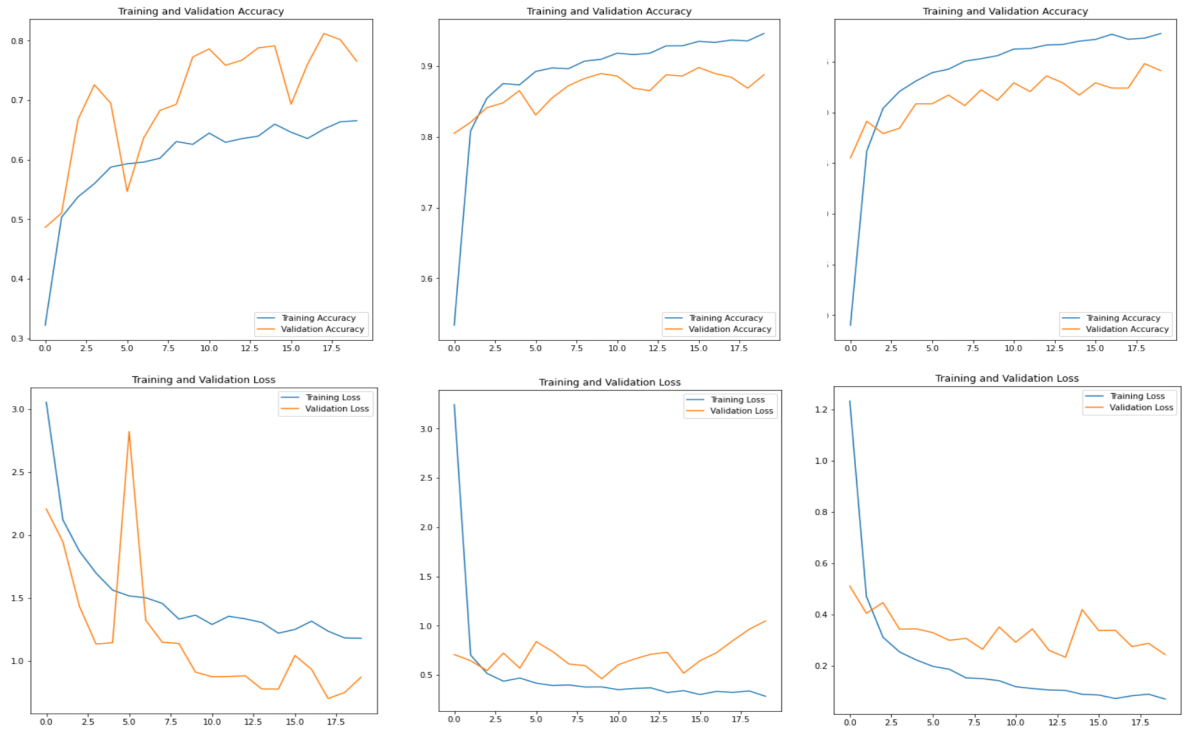

Figure 4.5: The preliminary results (top - training and validation accuracy; bottom - training and validation loss) obtained (from left to right) with VGG16, MobileNet and InceptionV3

Also some experiments using a custom model trained from scratch were conducted, but the results were not as good as the ones generated by the pre-trained models and moreover, this approach was very slow from the running time perspective. Analyzing the obtained results for each pre-trained architecture, I decided to continue the further experiments with InceptionV3, as it generated the best results. For the following experiments, the number of classes was reduced, in order to match the iNaturalist dataset. Thus, the following models used 52 classes from the

Birds400 dataset.

The pre-defined data for training and validation was combined, and then the validation data was randomly selected from the combined set. The splitting was done using StratifiedShuffleSplit class from sklearn library, in order to keep the proportions of splitting intact (such that the percent of splitting will be applied to each class individually). The chosen percent for splitting was 20%. The images were preprocessed being resized to the dimension of 299 $\times$ 299 and augmented on-the-fly using ImageDataGenerator. The best results on the new data configuration were provided by the model whose architecture is shown in Fig. 4.6. For further refer-

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 8, 8, 2048)        21802784
_________________________________________________________________
flatten (Flatten)            (None, 131072)            0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              134218752
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 200)               102600
_________________________________________________________________
dense (Dense)                (None, 52)                10452
=================================================================
Total params: 156,659,388
Trainable params: 134,856,604
Non-trainable params: 21,802,784
_________________________________________________________________
```

Figure 4.6: The architecture of the best model used on Birds400 dataset (Model 1)

ences, it will be called Model 1. It used the InceptionV3 model that was frozen and whose last fully connected layers used in the original classification were removed, followed by a Flatten layer, a Dense layer with 1024 units, a Dropout layer with the rate of 0.35, a Dense layer having 512 neurons, another Dropout layer with the rate of 0.35, a Dense layer with 200 neurons and, in the end, a Dense layer for classification with 52 neurons, corresponding to the 52 categories. All the Dense layers, exept the last one, use ReLU as activation function. Being used for classification, the last Dense layer uses Softmax activation function. The model used the Adam optimizer and the categorical crossentropy loss. It was trained during 25 epochs, with a callback for saving the best model in terms of validation loss and accuracy in a .h5 file. After the 25 epochs were completed, the model was fine-tuned, by unfreezing the last 10 layers from the Inception model, in order to test how the fine-tuning will affect the results. However, two callbacks for preventing overfitting were defined:

a callback for stopping the training if there is no improvement in the model's metrics in 4 consecutive epochs and one for reducing the learning rate on plateau, if the metrics do not improve in 2 consecutive epochs. The graphics describing the results obtained during training and validation are shown in Fig. 4.7, for the case in which the model with the best loss was fine-tuned and Fig. 4.8 for fine-tuning the model with the best accuracy. The results will be discussed in subsection 4.1.4.

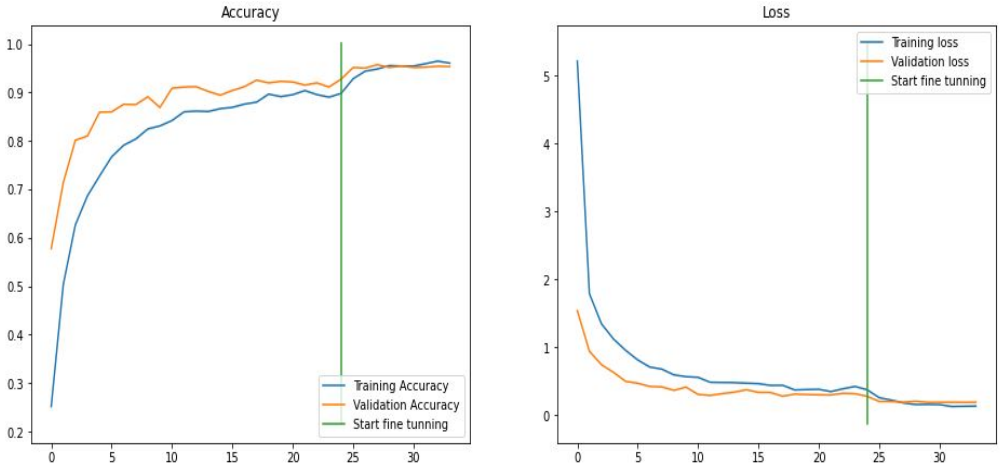

Figure 4.7: The results obtained on Birds400 dataset, using Model 1, by fine-tuning the model with the lowest validation loss

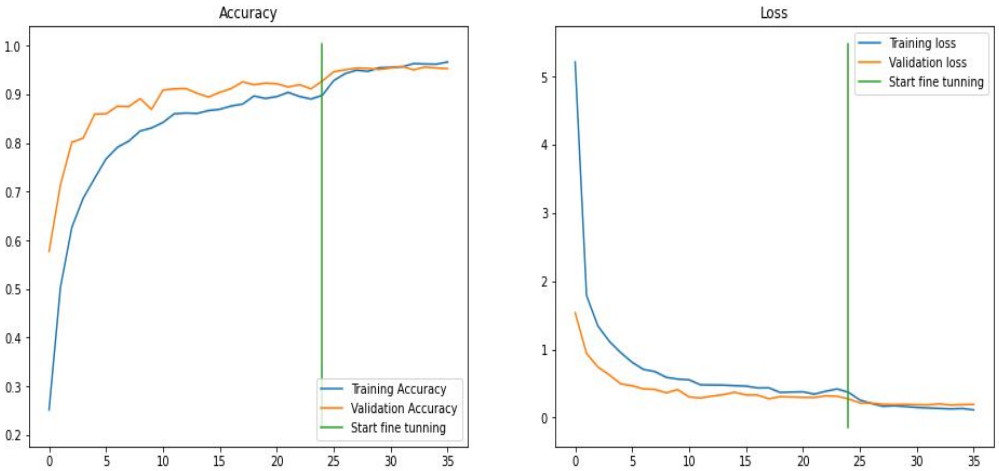

Figure 4.8: The results obtained on Birds400 dataset, using Model 1, by fine-tuning the model with the highest validation accuracy

The further experiments were conducted on the iNaturalist dataset. Initially, the model was trained on the existing data which was augmented, but without expanding the dataset. Thus, the data imbalance problem was still present. The architecture and the methodology were very similar to the ones used for Model 1. In this case, the experiments tried more regularization techniques, using Dropout and Activity

Regularization layers. The graphics with the accuracy and loss curves are shown in Fig. 4.9. It may be observed that the fine-tuning produces a larger gap between the training and validation curves, denoting the beginning of overfitting.

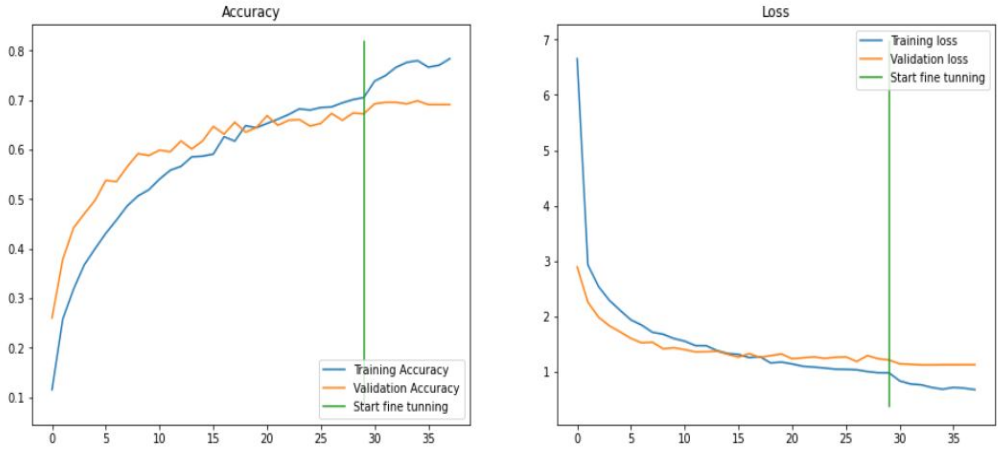

Figure 4.9: The results obtained on iNaturalist dataset, with data imbalance

The next step in improving this model was to create an equal distribution of images across the existing categories. In order to do this, ImageDataGenerator was used, with different augmentation methods: random zoom, rotation, shift or horizontal flip. New images, derived from the original ones, were created and saved in the dataset, such that there would exist 300 images per class. The images were also augmented in place before training and were resized to the dimension 299×299. The architecture of the model used on the augmented dataset is shown in Fig. 4.10. It will be further referenced as Model 2. As in the case of Model 1, the dropout for each layer is at the rate 0.35 and all the layers use ReLU activation function, except the last layer which uses Softmax activation for classification. The model was compiled with Adam optimizer, combined with the categorical crossentropy loss. It was trained with the frozen InceptionV3 model during 30 epochs, then it was fine-tuned on the last 10 layers of the Inception part and run another 8 epochs, until no improvement in the metrics could be observed. Similarly to Model 1, the fine-tuning was performed on the models with the smallest validation loss and the highest validation accuracy. The plots for accuracy and loss curves are shown in Fig. 4.11 for fine-tuning the model with the lowest validation loss and in Fig. 4.12 for the model with the highest validation accuracy, fine-tuned.

In order to combine the advantage of a high accuracy obtained on high quality photos with the advantage of the generalization capability resulted from training the models on images with different backgrounds and sizes, a fusion between Model 1 and Model 2 was created. Various combinations of the fine-tuned versions that generated the best results were used. The images were passed through the layers

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 8, 8, 2048)        21802784
_________________________________________________________________
flatten (Flatten)            (None, 131072)            0
_________________________________________________________________
activity_regularization (Act (None, 131072)            0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              134218752
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 200)               102600
_________________________________________________________________
dense (Dense)                (None, 52)                10452
=================================================================
Total params: 156,659,388
Trainable params: 134,856,604
Non-trainable params: 21,802,784
_________________________________________________________________
```

Figure 4.10: The architecture of the best model used on augmented iNaturalist dataset (Model 2)

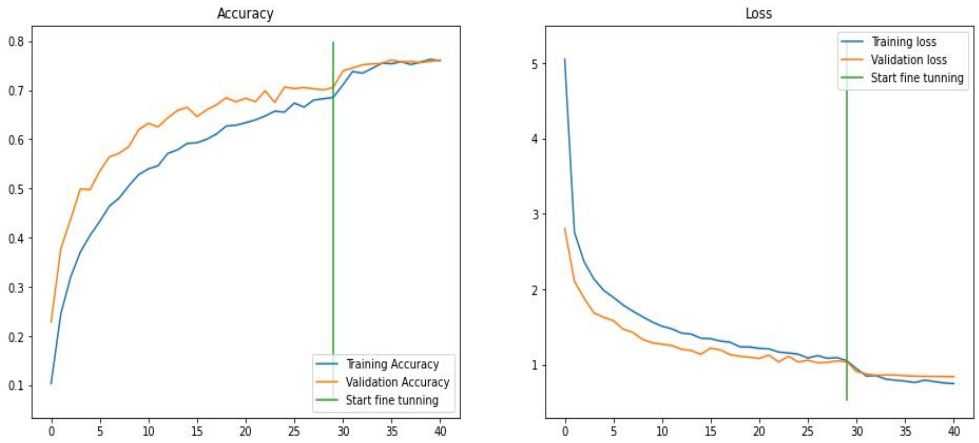

Figure 4.11: The results obtained on iNaturalist dataset, with augmented data (300 images per class), using Model 2, when the model with the smallest validation loss was fine-tuned

of both Model 1 and Model 2, until a given intermediate layer, in order to extract different features from the data. Then, the feature maps provided by both models were concatenated and fed into an ANN. This procedure is illustrated graphically in Fig. 4.13.

In more concrete terms, all the available images were analyzed firstly by Model 1 and then by Model 2. For each image, the output of the first Dense layer after the Inception component (the Dense layer with 1024 neurons) in the Model 1 was con-

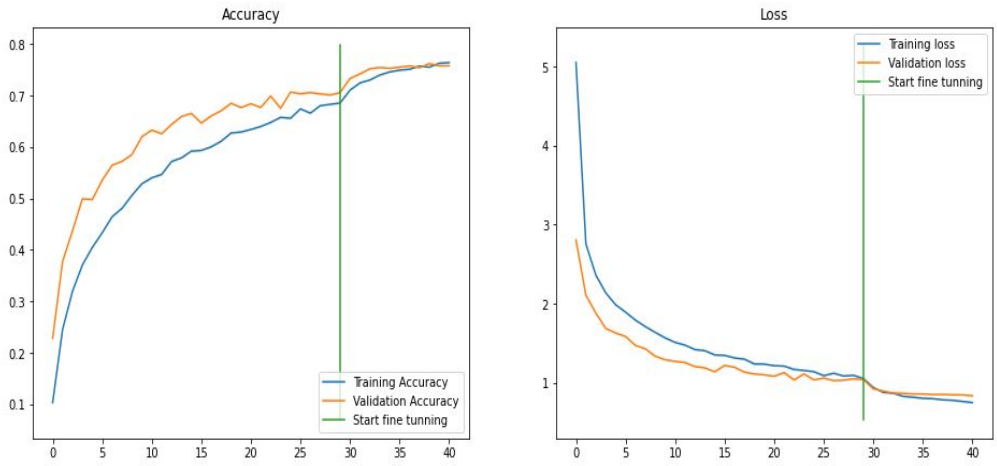

Figure 4.12: The results obtained on iNaturalist dataset, with augmented data (300 images per class), using Model 2, when the model with the highest validation accuracy was fine-tuned

catenated with the output of the corresponding Dense layer in the Model 2. From this concatenation it resulted an array with 2048 values for each image. These arrays were saved in individual files, split in folders according to the species of the bird. In this way, the input "dataset" for the fusion model was created. For further experiments, also the output of the second Dense layer, having 512 units, was considered, case in which each input for the fusion model had 1024 values.

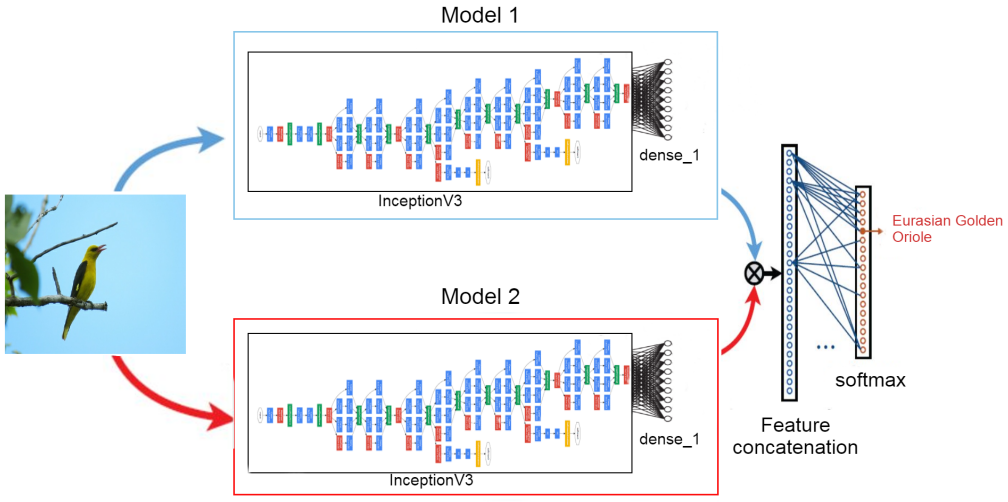

Figure 4.13: The methodology behind the fusion model for birds classification in images

Different architectures for the fusion ANN were experimented. They were mainly combinations of Dense and Dropout layers, having a various number of units and, respectively, various rates. The most used architecture is shown in Fig. 4.14. The

first Dropout layer has the rate equal to 0.1, while the other ones have the rate 0.35.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dropout (Dropout)            (None, None, 2048)        0
_________________________________________________________________
dense_1 (Dense)              (None, None, 400)         819600
_________________________________________________________________
dropout_1 (Dropout)          (None, None, 400)         0
_________________________________________________________________
dense_2 (Dense)              (None, None, 256)         102656
_________________________________________________________________
dropout_2 (Dropout)          (None, None, 256)         0
_________________________________________________________________
dense_3 (Dense)              (None, None, 128)         32896
_________________________________________________________________
dense (Dense)                (None, None, 52)          6708
=================================================================
Total params: 961,860
Trainable params: 961,860
Non-trainable params: 0
```

Figure 4.14: The architecture of the fusion model

The sparse categorical crossentropy loss was used, while for the optimizers, the choices were Adam and SGD. SGD was used with a specific value (different from 1 and from the size of the training sample) for the batch_size parameter in the fit() function, such that the used algorithm will become the Mini Batch Gradient Descent. The number of epochs varies between 30 and 40, but as in the case of the base models, a callback for stopping the training in case of no improvement in the model's metrics and one for reducing the learning rate on plateau were used, making the training process to stop earlier.

The experiments conducted for the fusion model implied different combinations of versions of Model 1 and Model 2, as follows: models with the best loss were combined together, considering, also for the fusion model, the version with the smallest validation loss being the best result, and analogous for the models with the highest accuracy. In both cases, I extracted the feature maps from both dense_1 and dense_2 layers from Model 1 and 2. By dense_1, I am referring to the first fully connected layer in the base model that comes after the Inception component – in this particular case, the dense layer having 1024 units. Similarly, dense_2 denotes the second fully connected layer after the Inception part, having 512 units. In each resulted case, both optimizers were tested, with different values for the batch size and for the number of epochs. The most relevant combinations are detailed below.

Firstly, the fusion was performed between models having the smallest validation loss, extracting the features from dense_1 layer. The learning curves from Fig. 4.15 correspond to the configuration with Adam optimizer, a batch size of 100 and 40

epochs, while Fig. 4.16 shows the curves obtained with SGD optimizer with 64 samples in a batch and 30 epochs (the training stopped at epoch 27).

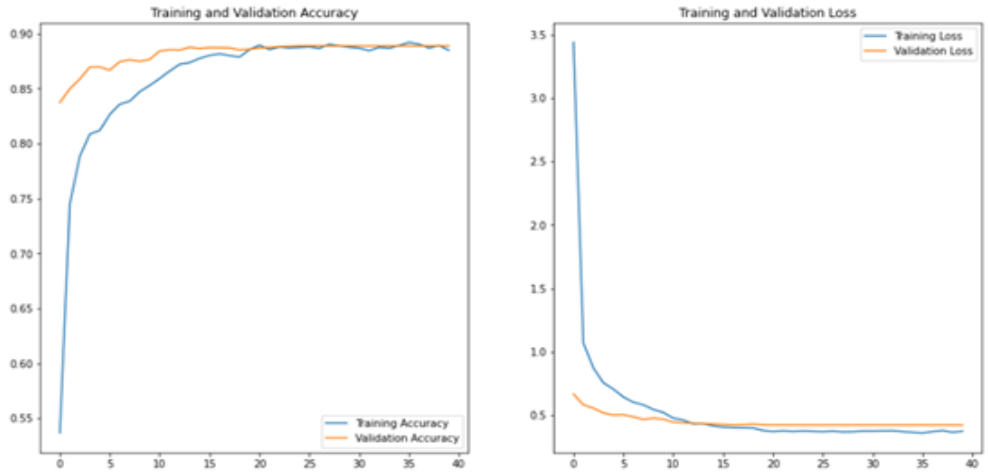

Figure 4.15: The results obtained on the combined datasets, using the Fusion Model between models with the best loss (Adam optimizer, a batch size of 100 and 40 epochs), extracting the features from dense_1 layer

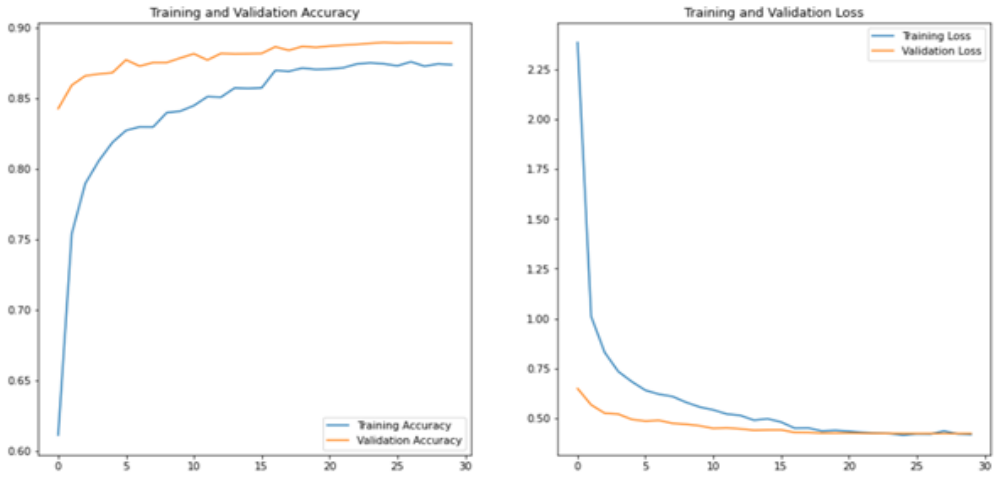

Figure 4.16: The results obtained on the combined datasets, using the Fusion Model between models with the best loss (SGD optimizer, a batch size of 64 and 27 epochs), extracting the features from dense_1 layer

The next experiments extracted the features also from dense_1 layer but used the best versions of Model 1 and 2 in terms of validation accuracy. In Fig. 4.17 it may be observed the plot of the learning curves when using Adam optimizer, a batch size of 100 and 35 epochs (again, due to the used callbacks, the training stopped after 31 epochs). Fig. 4.18 shows the curves obtained with SGD optimizer with 32 samples in a batch and 30 epochs (the training stopped at epoch 27). When using SGD optimizer with a batch size of 32 and 35 epochs, the obtained learning curves are the ones from Fig. 4.18.

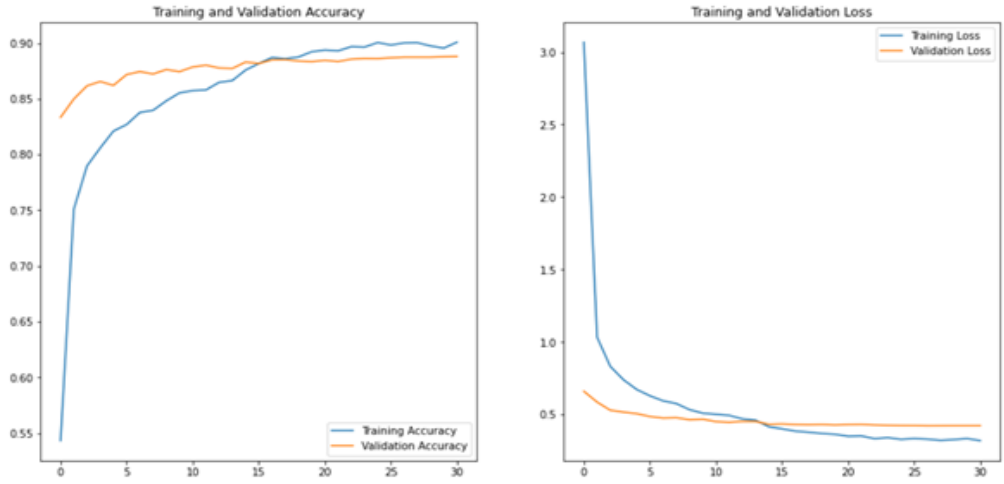

Figure 4.17: The results obtained on the combined datasets, using the Fusion Model between models with the best accuracy (Adam optimizer, a batch size of 100, 31 epochs), extracting the features from dense_1 layer

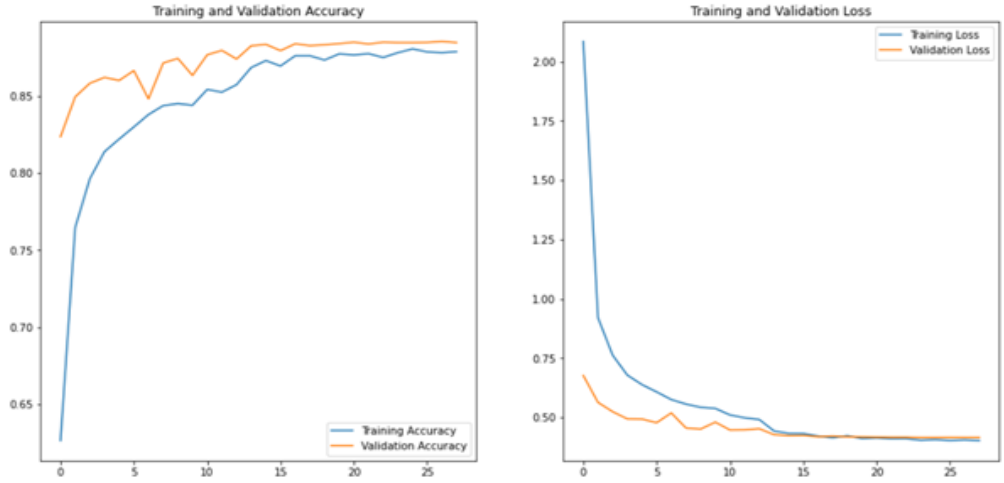

Figure 4.18: The results obtained on the combined datasets, using the Fusion Model between models with the best accuracy (SGD optimizer, a batch size of 32, 28 epochs), extracting the features from dense_1 layer

Further, the features were extracted from the dense_2 layer in the base models. In this case, the best results were given by the SGD optimizer. In the case of the models with the best loss, a batch size of 64 and 35 epochs were used. The training stopped after 22 epochs, as it may be observed in Fig. 4.19. The same configuration was used for the base models with the highest validation accuracy, their fusion being trained 30 epochs. The resulted training curves are shown in Fig. 4.20.

4.1.4 Results

The extended results provided by Model 1 and 2 are shown in Table 4.1. As it was mentioned in subsection 4.1.3. Experiments, the best results were selected in terms

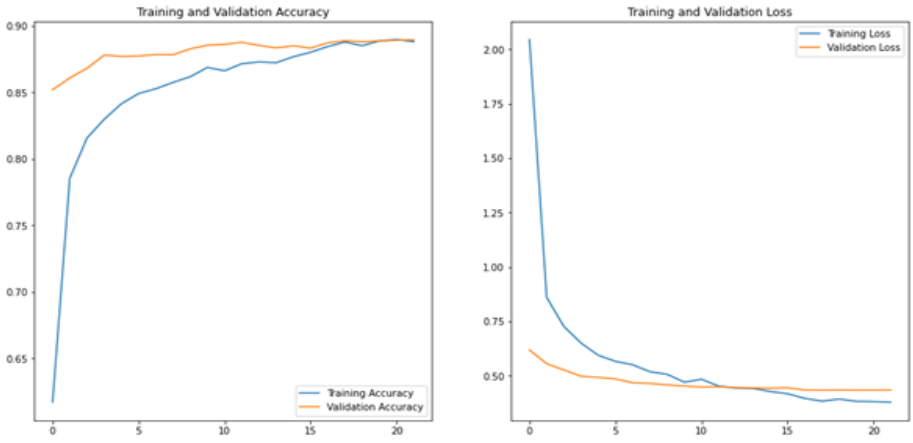

Figure 4.19: The results obtained on the combined datasets, using the Fusion Model between models with the best loss (SGD optimizer, batch size = 64, number of epochs = 22), extracting the features from dense_2 layer

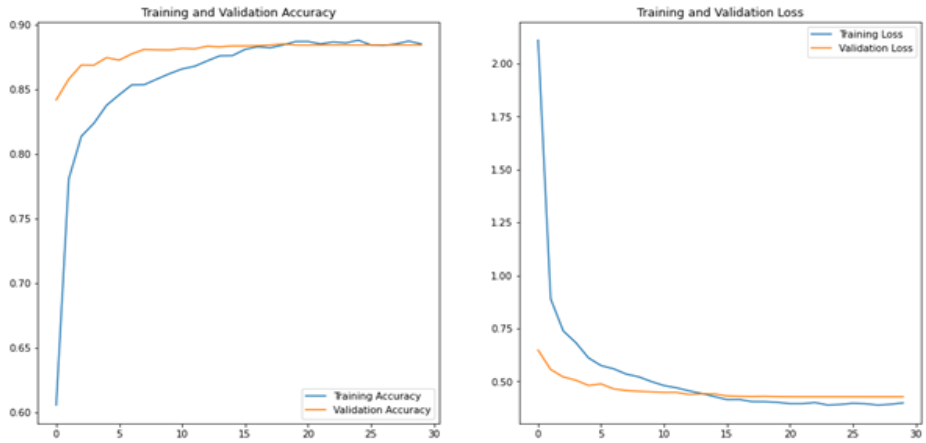

Figure 4.20: The results obtained on the combined datasets, using the Fusion Model between models with the best accuracy (SGD optimizer, batch size = 64, number of epochs = 30), extracting the features from dense_2 layer

of both validation loss and accuracy; this subsection will present also a comparison between the performances of the models, depending on the used selection criteria.

Stage	Model	Selection Criteria	Validation Accuracy	Validation Loss	Training Accuracy	Training Loss
Before Fine-Tuning	1	Smallest Validation Loss	92.73%	0.2737	89.78%	0.3700
After Fine-Tuning	1	Smallest Validation Loss	95.45%	0.1863	95.44%	0.1553

Before Fine-Tuning	1	Highest Validation Accuracy	92.73%	0.2737	89.78%	0.3700
After Fine-Tuning	1	Highest Validation Accuracy	95.71%	0.1865	95.61%	0.1411
Before Fine-Tuning	2	Smallest Validation Loss	70.54%	1.0240	66.55%	1.1181
After Fine-Tuning	2	Smallest Validation Loss	76.12%	0.8386	76.02%	0.7478
Before Fine-Tuning	2	Highest Validation Accuracy	70.64%	1.0339	65.55%	1.1378
After Fine-Tuning	2	Highest Validation Accuracy	76.15%	0.8449	75.49%	0.7753

Table 4.1: The results obtained during the learning process by Model 1 and Model 2 in all the scenarios described in subsection 4.1.3. Experiments

From Table 4.1, it may be observed that in the case of Model 1, before the fine-tuning, the version with the smallest validation loss registered also the highest validation accuracy. Moreover, as it was expected, the results obtained by Model 1 are considerably high, due to the quality of the dataset. The performance of Model 2 is also satisfying, considering that the dataset contained a larger variety of backgrounds, sizes and objects that might obstruct a good view of the birds. Regarding the fine-tuning process, it may be noticed that the results after the fine-tuning are significantly better than the results obtained when the Inception model was used exclusively as a feature extractor.

Table 4.2 shows the results obtained by the best versions of the fusion models, in various scenarios that were experimented. However, the values for the loss and accuracy are similar in the described cases and do not fluctuate considerably. The obtained results are situated lower than the results provided by Model 1, but significantly higher than the results given by Model 2.

In order to compare the performance of the three models on unseen data, the models were evaluated on a dataset containing 15 images for each species. For Model 1 and Model 2, their fine-tuned versions were considered. Table 4.3 shows the comparison between the best results provided by the three models. It is clear that the fusion model managed to boost substantially the testing accuracy and to diminish the loss.

Extraction Layer	Selection Criteria	Optimizer	Batch Size	Validation Accuracy	Validation Loss	Training Accuracy	Training Loss
dense_1	Smallest Valid. Loss	Adam	100	88.88%	0.4212	88.50%	0.3732
dense_1	Smallest Valid. Loss	SGD	64	88.89%	0.4212	87.35%	0.4168
dense_1	Highest Valid. Accuracy	Adam	100	88.80%	0.4231	90.08%	0.3205
dense_1	Highest Valid. Accuracy	SGD	32	88.56%	0.4157	87.82%	0.4049
dense_2	Smallest Valid. Loss	Adam	100	89.14%	0.4481	90.51%	0.3254
dense_2	Smallest Valid. Loss	SGD	64	88.88%	0.4354	88.78%	0.3845
dense_2	Highest Valid. Accuracy	Adam	100	88.91%	0.4459	90.65%	0.3179
dense_2	Highest Valid. Accuracy	SGD	64	88.50%	0.4294	88.41%	0.4040

Table 4.2: The results obtained during the learning process by the fusion model

Model	Selection Criteria	Extraction Layer	Testing Accuracy	Testing Loss
Model 1	Smallest Valid. Loss	-	76.538%	1.2599
Model 1	Highest Valid. Accuracy	-	77.051%	1.3046
Model 2	Smallest Valid. Loss	-	77.435%	0.8865
Model 2	Highest Valid. Accuracy	-	77.051%	0.8878
Fusion	Smallest Valid. Loss	dense_1	87.948%	0.5926
Fusion	Highest Valid. Accuracy	dense_1	87.051%	0.5924
Fusion	Smallest Valid. Loss	dense_2	87.692%	0.6312
Fusion	Highest Valid. Accuracy	dense_2	86.795%	0.6439

Table 4.3: The results obtained when testing the three models

Further, for the fusion model, the confusion matrix was created using the same data that was used for testing. It is shown in Fig. 4.21. The confusion matrix sum-

marizes the performance of the classification model, denoting for each class how many samples from input were correctly classified and what are the categories that are prone to be confused. Each row of the matrix represents the instances in an actual category while each column denotes the instances in the predicted class. For instance, from Fig. 4.21, it might be observed that the species of birds Northern Fulmar is often confused with the Albatross; indeed, they have many physical similarities and they both live in oceanic regions, so they are prone to be mistaken one with each other.

Finally, I aim to compare the created models with the existing approaches in the literature, presented in subsection 2.1) Existing Solutions. To summarize better the results, Table 4.4 was created. It describes the proposed methodologies and the obtained results. Analysing the overall results, for this synthesis the Model 1 with the highest validation accuracy and the Model 2 with the smallest validation loss were chosen. For the Fusion model, I selected the configuration which uses the base models with the smallest validation loss and extracts features from dense_1 layer, using Adam optimizer with a batch size of 100.

Model (From)	Methodology	Dataset	No. of classes	Validation Accuracy	Testing Accuracy
[1]	VGG-16 & SVM	custom	27	89%	–
[1]	VGG-16 & Random Forest	custom	27	87%	–
[2]	Mask R-CNN & InceptionResNetV2	custom	16	33.69%	49.09%
[2]	Mask R-CNN & InceptionV3	custom	16	15.5%	41.66%
[3]	VGG16	Birds400	200	**96**%	-
[3]	MobileNetV2	Birds400	200	95%	-
Model 1 (mine)	InceptionV3	Birds400	52	95.71%	77.05%
Model 2 (mine)	InceptionV3	iNaturalist	52	76.12%	77.43%
Fusion Model (mine)	Fusion between Model 1 & Model 2	Birds400 & iNaturalist	52	88.88%	**87.95** %

Table 4.4: The comparison between the results existing in the literature and the models proposed in this thesis

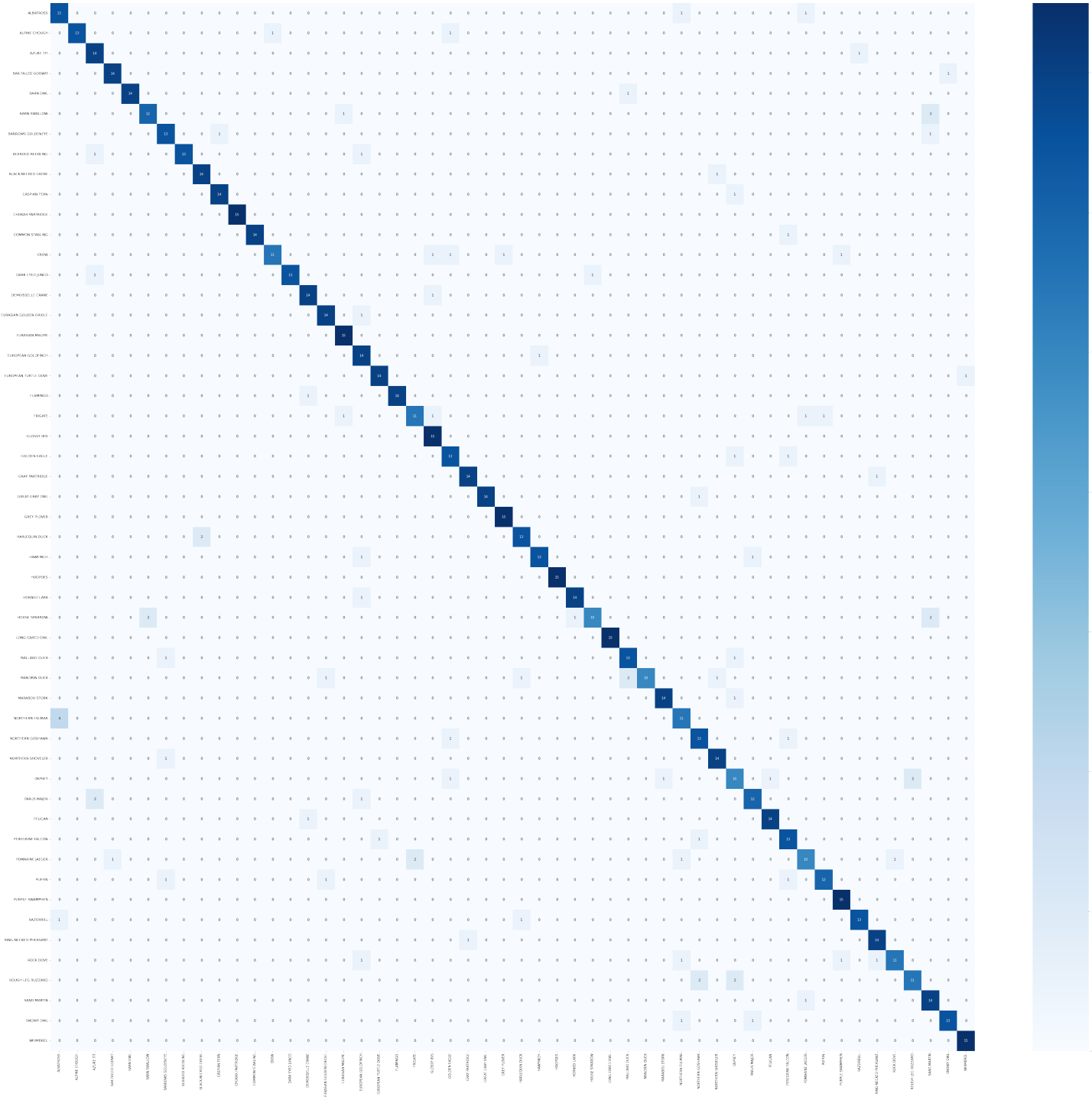

Figure 4.21: The confusion matrix for the fusion model, generated using the testing data

4.2 Software Application

The AI model was integrated in a web application, with back end and front end. The chosen model was the fusion between the fine-tuned versions of Model 1 and Model 2 with the best loss, that uses Adam optimizer.

4.2.1 Back End

The back end part was implemented in Python using Flask, a micro web framework that provides useful features and tools for creating web servers. On the server side, two classes that handle the predictions were created, their diagram being visible in Fig. 4.22. The Prediction class models a certain prediction for an image, storing data about the predicted species (common and scientific name), the accuracy of the prediction (the probability that the bird from the image is the predicted one) and the name of the model that made the prediction (Model 1, Model 2 or Fusion). The method *toDict()* generates a dictionary with the attributes of the prediction in order to facilitate the "jsonifying" of the prediction.

The Classifier class encapsulated the three predictive models that are read from the file and set up in the moment of initialization, using the private method *setup_models*. The Model 1 was trimmed such that it would be formed by two parts: the part needed in fusion model as feature extractor (the attribute *model1*) and the part used for the classification in the base model (the attribute *model1_cont*). The same procedure was applied to Model 2. This trimming decreased the storing space occupied by the models if it were to store the feature extractor and the whole model as two separate entities (because the application aims to use all three models in order to do predictions, according to the user's choice). The *predict* method uses the specified model to generate the top-3 predictions (the predictions with the highest probabilities) for a given image. The image is firstly pre-processed in order to have the required dimensions for the network. If the choice is Model 1, then the image is passed through model1; the extracted features are further fed into the continuation of the trimmed Model 1 (model1_cont), and a prediction is obtained. The Model 2 uses the same principle. For the fusion model, the pre-processed image is passed through model1 and model2, the features are then concatenated. The resulted tensor is fed into the fusion model. Other attributes in the Classifier class (*species* and *scientific_names*) are used for identifying the species common and scientific names by the index in the prediction vector.

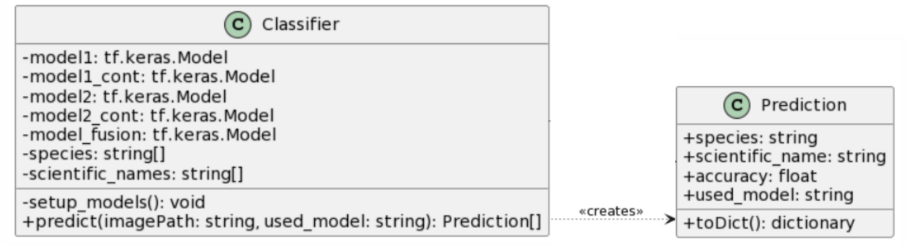

Figure 4.22: Classifier and Prediction classes

The call to the server for realizing the prediction is done at the route /api/classifi-

cation via POST method. The request needs to contain a "file" field to which an image with the extensions .png, .jpg or .jpeg is associated. If this condition is met, the image is saved locally and the classifier will use the model specified in the "model" field (if there is no such field, the default considered model is fusion) to generate the prediction; after that, the response to the request will contain an array with the generated predictions in the JSON format.

4.2.2 Front End

The front end was implemented using Angular 11, a framework for building single-page client applications using HTML and TypeScript. The front end application contains a component for the main page through which the user can receive a prediction for an uploaded photo. Screenshots from the main page are included in Fig. 4.25 and 4.26. This page contains an input field for images, a selection form for choosing the model and a Submit button which will trigger a call to the server in the moment it is clicked. The sent request is of type FormData, containing the file and the specified model. When the response from the server is received, a bar chart illustrative for the results is shown. The bar chart was implemented as a separate (child) component that receives via @Input() decorator an array of predictions. It uses the module ng2-charts that creates responsive and user-friendly charts with some given options in order to generate a more pleasant looking horizontal bar chart. Also, for the selection form, the Mat Form Field, Mat Select and Mat Option modules were used.

Additionally, on the *shared* directory there is a model for the prediction, containing the corresponding attributes for the Prediction class in the back end. There is also a classification service that communicates with the Flask server, sending to it the uploaded image and the selected model and receiving the result.

The main page can be accessed at the route "/main-page". The empty route "/" will redirect the user also to the main page.

4.2.3 Design

To summarize the details about the interaction of the user with the system, an UML use case diagram may be used. In the case of my application, all the operations that are available to the user are reflected in the use case diagram from Fig. 4.23.

As mentioned before, the user may upload an image and choose a classification model and after one submits the required information, the predictions will be displayed.

The sequential order in which the interactions between the components of the system occur is described in the sequence diagram, in Fig. 4.24. After the user

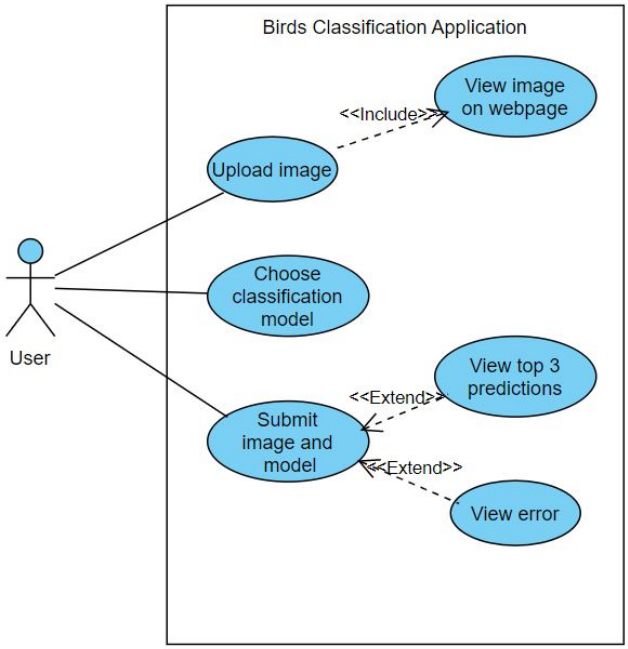

Figure 4.23: The use case diagram for the developed application

uploads the image and specifies the desired model, the data is sent to the server through the classification service component. On the server side, the received information is processed by the classifier, which returns the generated predictions. The server sends back the response, and the main page will provide the predictions to the child bar chart component, such that the results will be displayed to the user in a graphical representation.

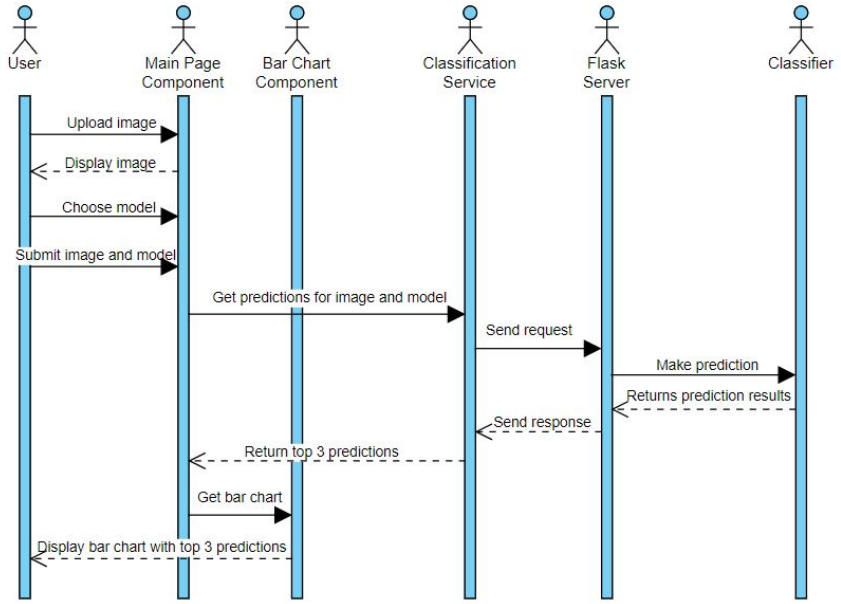

Figure 4.24: The sequence diagram for the developed application

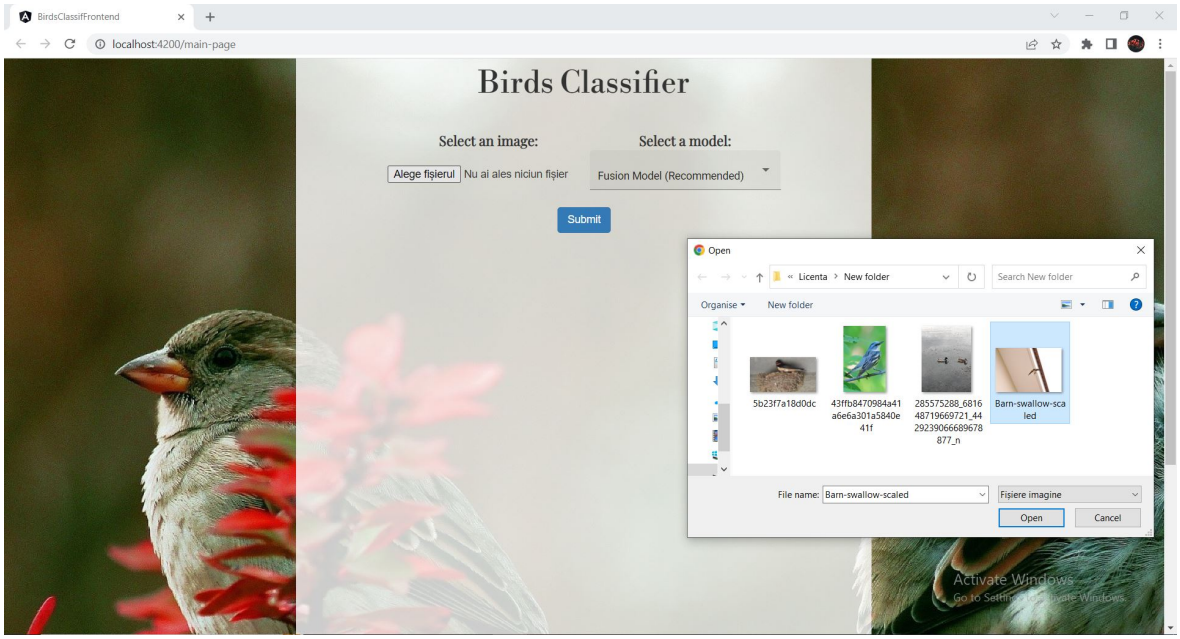

Figure 4.25: The main page of the web application allowing the user to upload an image and to choose a model

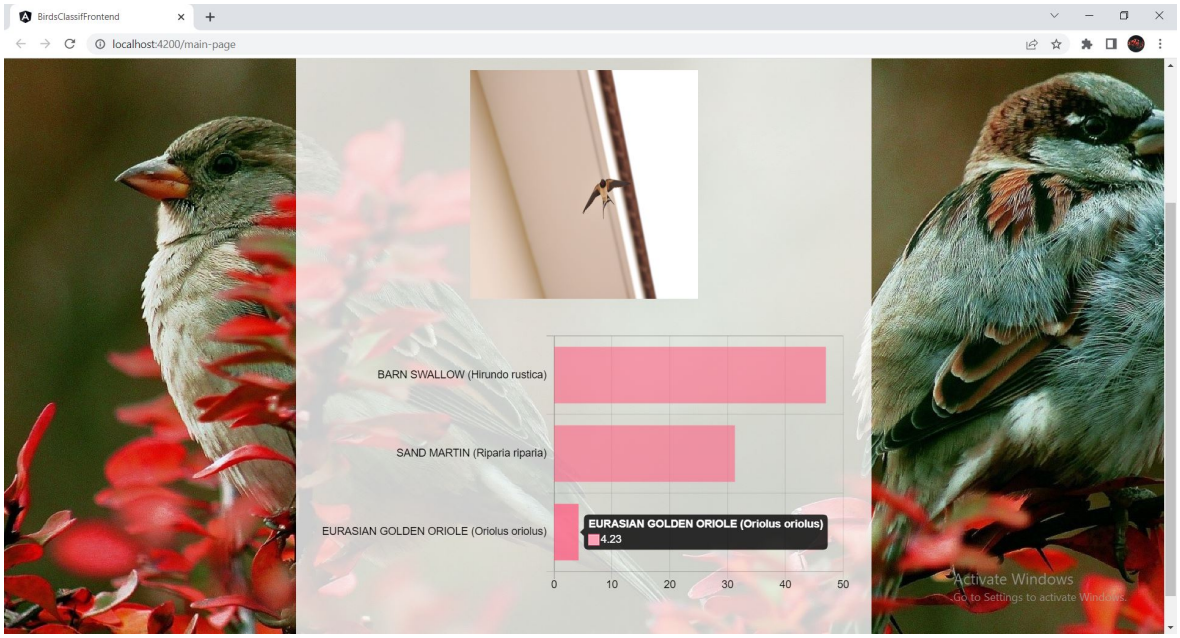

Figure 4.26: The main page of the web application after the results were provided from the server

Chapter 5

Conclusions and Future Work

Computer vision is a popular research area nowadays, as it trains the computer to derive meaningful information from visual data, in order to interpret and understand it. This thesis provides a relevant solution for classifying various species of birds from images: a machine learning model built as a fusion between other two models that have been previously trained on different datasets.

A pre-trained model was used in this fine-grained visualisation problem, in order to obtain better results. After several experiments, the InceptionV3 architecture was chosen, as it generated the best results in comparison with the other attempts. Firstly, the results obtained by the model which uses the Birds400 dataset during training and validation seemed indeed satisfactory. But during testing phase, the problem of a poor performance came out, when trying to classify birds from images similar to the ones that most of the average users would upload. Thus, the misleading solution provided by this model was not enough. After that, I decreased the number of possible classes to 52, I collected the data from the iNaturalist dataset and trained another model. It had better generalization capabilities than the first model, but overall the accuracy was noticeably smaller, especially when it came to classifying birds from HQ images. Going further with my research, I discovered the idea of using a fusion model that will incorporate two networks trained on different datasets. Thus, a fusion model that uses the first two attempts was created; it outperformed the base networks taken individually, having a validation accuracy of 88.88% and a testing accuracy greater by about 10% than the other two models, on the same testing samples.

The restricted hardware resources problem lies among the limitations encountered in developing this project. The alternative exemplified in the thesis was using online notebooks and environments with GPU support. However, a larger computational power provided by the machine on which the models run could facilitate trying more different combinations for parameters involved in the training process (the learning rate, the batch size, the number of epochs etc). The more experiments

are conducted, the better the results might become.

Another limitation was represented by the relatively small size of the available data. A large number of classes necessitates more samples from each category, therefore, enhancing the dataset with more various images (and not only with augmented data) is a good solution for improving the classification performance of the model. In the future, more species of birds can be added if there exists sufficient visual data available, in order to increase the diversity determined by the models.

The presented approach can be integrated in various contexts: one of them is the proposed web application which can help the bird watchers identify more easily the species. Another situation for integrating the model would be for monitoring the behaviour of birds and their migrating tendencies, using surveillance cameras or information provided by humans. This activity is necessary especially in the context of global warming and massive pollution, as birds are excellent indicators for environmental issues. Thus, the proposed model could be combined in the future with other methods for video classification, or the location of the identified bird could be used in order to improve the predictions. Certainly, the artificial intelligence field, together with all its subareas, is in a continuous progress, so new developments and superior algorithms are expected to appear, facilitating further explorations in this domain.

Bibliography

[1] Shazzadul Islam, Sabit Ibn Ali Khan, Md. Minhazul Abedin, Khan Mohammad Habibullah, and Amit Kumar Das. Bird species classification from an image using vgg-16 network. In *Proceedings of the 2019 7th International Conference on Computer and Communications Management*, ICCCM 2019, page 38–42, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450371957. doi: 10.1145/3348445.3348480. URL https://doi.org/10.1145/3348445.3348480.

[2] Akash Kumar and Sourya Dipta Das. Bird species classification using transfer learning with multistage training. In Chetan Arora and Kaushik Mitra, editors, *Computer Vision Applications*, pages 28–38, Singapore, 2019. Springer Singapore. ISBN 978-981-15-1387-9.

[3] Tejalal Choudhary, Shubham Gujar, Kruti Panchal, Sarvjeet, Vipul Mishra, and Anurag Goswami. A deep learning-based transfer learning approach for the bird species classification. In Deepak Garg, Kit Wong, Jagannathan Sarangapani, and Suneet Kumar Gupta, editors, *Advanced Computing*, pages 43–52, Singapore, 2021. Springer Singapore. ISBN 978-981-16-0404-1.

[4] Giorgio Roffo. Ranking to learn and learning to rank: On the role of ranking in pattern recognition applications. 06 2017.

[5] Nicolo Ceneda. *Quantile Regression of High-Frequency Data Tail Dynamics via a Recurrent Neural Network*. PhD thesis, 05 2020.

[6] Maher Al-Zuhairi, Biswajeet Pradhan, Helmi Shafri, and Hussain Hamid. *Applications of Deep Learning in Severity Prediction of Traffic Accidents*, pages 793–808. 01 2019. ISBN 978-981-10-8015-9. doi: 10.1007/978-981-10-8016-6_58.

[7] Josh Patterson and Adam Gibson. *Deep Learning: A Practitioner's Approach*. O'Reilly Media, Inc., 1st edition, 2017. ISBN 1491914254.

[8] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning, 2018. URL https://arxiv.org/abs/1808.01974.

[9] Hee E. Kim, Alejandro Cosa-Linan, Nandhini Santhanam, Mahboubeh Jannesari, Mate E. Maros, and Thomas Ganslandt. Transfer learning for medical image classification: a literature review. *BMC Medical Imaging*, 22(1):69, Apr 2022. ISSN 1471-2342. doi: 10.1186/s12880-022-00793-7. URL https://doi.org/10.1186/s12880-022-00793-7.

[10] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014. URL https://arxiv.org/abs/1409.4842.

[11] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015. URL https://arxiv.org/abs/1512.00567.

[12] Martin Nyffeler, Cagan Sekercioglu, and Christopher Whelan. Insectivorous birds consume an estimated 400–500 million tons of prey annually. *The Science of Nature*, 105, 07 2018. doi: 10.1007/s00114-018-1571-z.

[13] Duarte S. Viana, Laura Gangoso, Willem Bouten, and Jordi Figuerola. Overseas seed dispersal by migratory birds. *Proceedings of the Royal Society B: Biological Sciences*, 283(1822):20152406, 2016. doi: 10.1098/rspb.2015.2406. URL https://royalsocietypublishing.org/doi/abs/10.1098/rspb.2015.2406.

[14] Rachel T. Buxton, Amber L. Pearson, Claudia Allou, Kurt Fristrup, and George Wittemyer. A synthesis of health benefits of natural sounds and their distribution in national parks. *Proceedings of the National Academy of Sciences*, 118(14):e2013097118, 2021. doi: 10.1073/pnas.2013097118. URL https://www.pnas.org/doi/abs/10.1073/pnas.2013097118.

[15] Ian P. F. Owens and Peter M. Bennett. Ecological basis of extinction risk in birds: Habitat loss versus human persecution and introduced predators. *Proceedings of the National Academy of Sciences*, 97(22):12144–12148, 2000. doi: 10.1073/pnas.200223397. URL https://www.pnas.org/doi/abs/10.1073/pnas.200223397.

[16] David Kriesel. *A Brief Introduction to Neural Networks*. 2007. URL http://www.dkriesel.com.

[17] Jianli Feng and Shengnan Lu. Performance analysis of various activation functions in artificial neural networks. *Journal of Physics: Conference Series*, 1237

(2):022030, jun 2019. doi: 10.1088/1742-6596/1237/2/022030. URL https://doi.org/10.1088/1742-6596/1237/2/022030.

[18] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL https://proceedings.mlr.press/v15/glorot11a.html.

[19] G. Ciaburro and B. Venkateswaran. *Neural Networks with R: Smart models using CNN, RNN, deep learning, and artificial intelligence principles*. Packt Publishing, 2017. ISBN 9781788399418. URL https://books.google.ro/books?id=IppGDwAAQBAJ.

[20] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. Pearson, 3 edition, 2009.

[21] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016. URL https://arxiv.org/abs/1609.04747.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[23] Antonio Gulli and Sujit Pal. *Deep Learning with Keras*. Packt Publishing, 2017. ISBN 1787128423.

[24] Kyoung-Su Oh and Keechul Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2004.01.013. URL https://www.sciencedirect.com/science/article/pii/S0031320304000524.

[25] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. URL https://cs231n.github.io/convolutional-networks/#conv.

[26] Ahmed Azab, Mahnaz Arvaneh, Jake Toth, and Lyudmila Mihaylova. *A review on transfer learning approaches in brain–computer interface*. 09 2018. ISBN 9781785613999. doi: 10.1049/PBCE114E.

[27] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. doi: 10.1109/TKDE.2009.191.

[28] Ricardo Ribani and Maurício Marengoni. A survey of transfer learning for convolutional neural networks. pages 47–57, 10 2019. doi: 10.1109/SIBGRAPI-T.2019.00010.

[29] Birds 400 – species image classification, . URL https://www.kaggle.com/datasets/gpiosenka/100-bird-species. Online; accessed 13 January 2022.

[30] Bird checklist of the world - europe. https://avibase.bsc-eoc.org/checklist.jsp?lang=EN&p2=1&list=ioc&synlang=&region=EUR&version=text&lifelist=&highlight=0, . Online; accessed 15 January 2022.

[31] L. Svensson, D. Zetterström, K. Mullarney, and P.J. Grant. *Birds of Europe*. Princeton field guides. Princeton University Press, 1999. ISBN 9780691050546. URL https://books.google.ro/books?id=MfmzQgAACAAJ.

[32] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The inaturalist species classification and detection dataset, 2017. URL https://arxiv.org/abs/1707.06642.

[33] 200k images of birds (inaturalist), . URL https://www.kaggle.com/datasets/sharansmenon/inatbirds100k. Online; accessed 6 April 2022.

[34] ebird – the cornelllab of ornithology. URL https://ebird.org/home. Online; accessed 10 May 2022.