Baden-Wuerttemberg Cooperative State University Mannheim

Project Report Advanced Machine Learning

Bird Species Classification

Course of Study Business Informatics

Field of Study Data Science

Author:	Annika Dackermann, Max Bernauer, Valentin Moritz Müller
Matriculation Number:	5562028, 5763624, 4616344
Company:	SAP SE
Course:	WWI20DSB
Director of Studies	Prof. Dr. Bernhard Drabant
Academic Supervisor:	Prof. Dr. Maximilian Scherer
Completion Period:	22.05.2022 – 23.07.2023

Statutory Declaration

We herewith declare that we have composed the present thesis with the title "*Bird Species Classification*" ourselves and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned.

The thesis in the same or similar form has not been submitted to any examination body and has not been published.

Mannheim, 23.07.2023 A.Dackermann [illegible] [illegible]

Place, Date Annika Dackermann, Max Bernauer, Valentin Moritz Müller

Contents

List of Figures

List of Tables

List of Acronyms

AI	artificial intelligence
API	application programming interface
CNN	convolutional neural network
CSS	Cascading Stlye Sheets
CV	computer vision
IC	image classification
mAP	mean Average Precision
ML	machine learning
NN	(artificial) neural network
OD	object detection
ResNet	residual neural network
REST	representational state transfer
UI	user interface
VGG	Visual Geometry Group

1 Introduction

1.1 Background and Motivation

Nowadays, apps and other technologies have made it easier to get access to information about bird species. (Shankland, 9/6/2017; Birding Outdoors, 2021) For instance, apps can identify birds based on their appearance, or other apps alert users when a rare bird species has been spotted in the area. (Shankland, 9/6/2017) Between 2010 and 2020, many apps have been launched that deal with birds. (Birding Outdoors, 2021) These apps include field guides, bird identification, and bird identification based on appearance. (Birding Outdoors, 2021)

This work involves the development of a web application for the classification of bird images. This application can be utilized for educational purposes, enabling the classification of birds and the closer integration of students with science. (Jeno et al., 2017; Cornell Chronicle, 7/23/2023) Furthermore, the application can help to monitor and protect endangered bird species by recognizing rare birds in images. (Green, 2/21/2022) This can reduce the manual work required to collect conservation data. (Green, 2/21/2022) In addition, the application can be used to address the difficulty of identifying birds, as some birds look similar. (Birding Outdoors, 2021)

1.2 Objective

The goal of this work is to develop a web application that can be used to identify different bird species. The user should be able to upload a picture of a bird on the frontend so that it can be classified by a model. Then the name of the bird species should be presented. In addition, information about the identified bird is provided, either in the form of a summary or the complete Wikipedia entry.

1.3 Structure of the Paper

This work is divided into two parts, theoretical, which describes the basics of CV, and practical, which describes the architecture and functions of the application.

Chapter 2 presents the theoretical foundations of this paper. First, CV is explained, which is important to understand to perform image classification and identify birds in the given image. Additionally, classification architectures and Web App components are described.

Chapter 3 deals with the technical implementation and architecture of the project. Subsequently, the structure and content of the data set are described. Afterward, the model and its evaluation are presented, which is used to measure the performance of the trained model. Next, backend and frontend of the web app are introduced by describing the functionalities and the integration of the model into the app. Moreover, this chapter also outlines the process for retrieving bird species information and displaying it on the frontend.

Finally, the work is summarized in the most important points, and a critical reflection is conducted.

2 Theoretical Foundation

2.1 Computer Vision

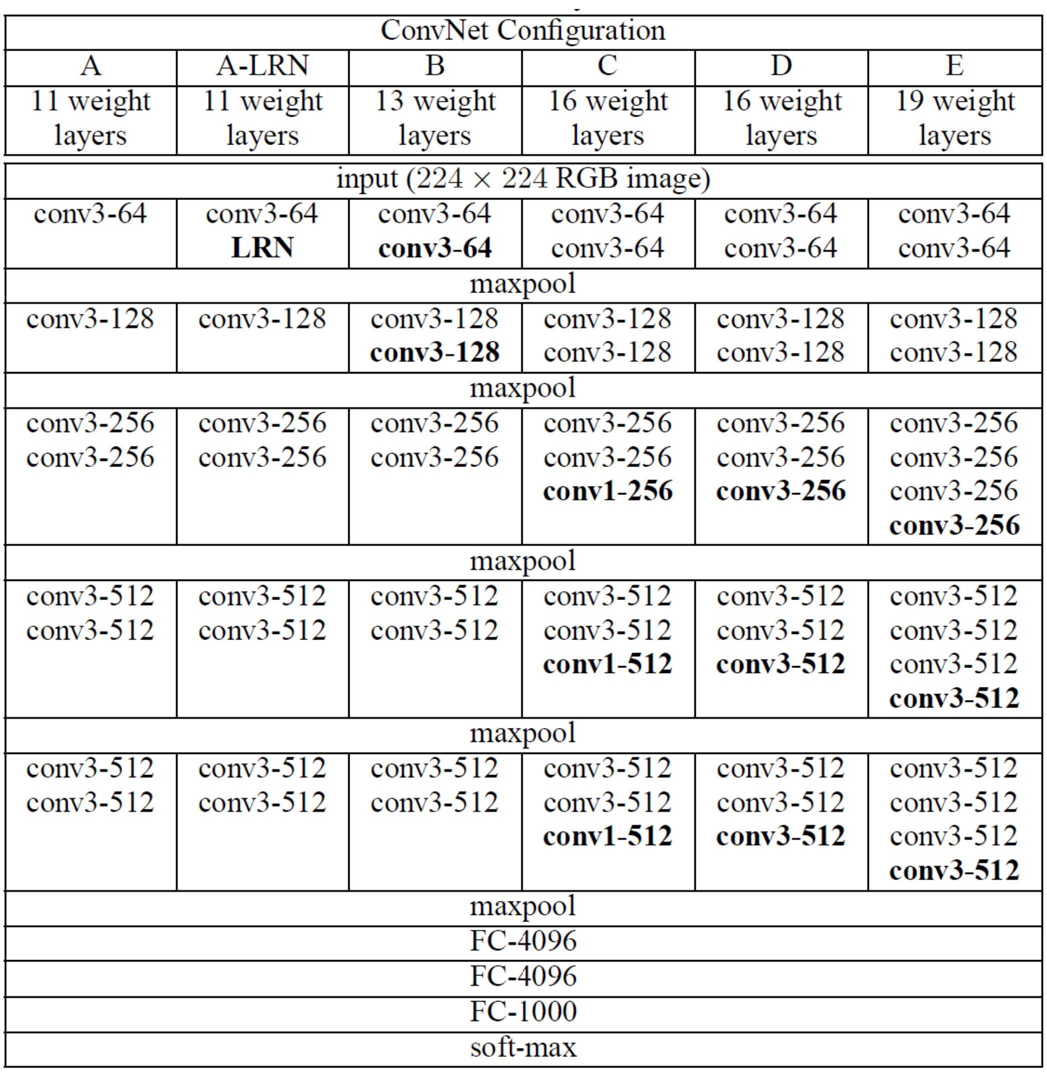
The discipline of *computer vision* (CV) can be placed in the intersection of artificial intelligence (AI) and artificial vision, a field that explores ways to extract information from images. Apart from CV, artificial vision also includes the related, but in no way identical discipline of machine vision, which focuses on the implementation of real-world systems who are able to perform artificial vision tasks. That means it requires knowledge stemming from physics and engineering, instead of mostly mathematics and computer science like CV, which focuses on the processing and analysis of already captured images and usually requires them to be of a certain quality. For the task of implementing a system for bird species classification, mainly CV is required, since the artificial vision task of capturing the images is left to the users of the application (Batchelor, 2012, pp. 10–18; Gad et al., 2018, pp. xvii–xix).

Normally, the problem of bird species classification would consist of two sub-problems. Before the *image classification* (IC) task can be performed to determine which of several categories, like in this case bird species, an image belongs to, it would normally be required to run an *object detection* (OD) task to determine whether there are any of the objects of interest, in this case birds, in an image. Even though many OD algorithms are capable of performing classification tasks, if can be beneficial to use the OD purely for detection purposes, and a sophisticated IC model for the classification, especially in cases where the quality of the predictions is of big importance like traffic sign detection and recognition. In some cases, OD can be seen as an extension of IC, as the former often builds on the techniques used in the latter. Due to a mix of time constraints and only a suitable data set being available for IC, as well as the application being intended to receive pictures from the user with a bird in good view, this project and thus this report will focus in IC (Wang et al., 2022, pp. 1–2; Gad et al., 2018, p. 188; Sermanet & LeCun, 2011, p. 2809; Wei et al., 2018, pp. 5884–5885).

In CV, the most important type of utilized machine learning (ML) model for all tasks is a *convolutional neural network* (CNN). Many different ways to implement CNNs exist, but all are at their core a type of *(artificial) neural network* (NN). Like the name suggests, those models are inspired by the human brain and its structure of connected *neurons* which "fire" when activated, and also implementable in a variety of ways that can make them suitable to many different tasks. Each NN consists of multiple *layers*, which are *fully connected* for normal NNs meaning every neuron in every layer is connected to each one in the next layer. The first layer is called *input layer*, the last one the *output layer*, while all the ones in between are termed *hidden layers*. The large number of connections in fully connected models leads to a very high number of parameters, which impacts performance both at inference and training. CNNs are not fully connected, instead they feature the eponymous convolutions, as well as filters and pooling. Convolutions are able to transform an input-tensor to a feature map. Each neuron in a convolutional layer only processes inputs in its receptive field, the size of which is determined by the filter, leading to fewer connections and thus faster training and inference at runtime. Pooling combines the output of a receptive field to a specific value, for example its average value in *average pooling* or maximum value in *max pooling*, leading to dimensional reduction. CNNs can nevertheless make use of fully connected layers, this is for example required for classification of images that have been processed by the convolutional layers. One other reason this makes them so well-suited to CV it that tensors in general are a natural fits for a mathematical representation of image data, since images consist (in a common representation which is also utilized in this project) of a two-dimensional array of pixels, each of which has an additional value for red, blue and green assigned (Gad et al., 2018, pp. 45, 183; Sermanet & LeCun, 2011, pp. 2809–2810; Wei et al., 2018, p. 5885; Simonyan & Zisserman, 2014, pp. 2–3). In the following section 2.2, multiple architectures for CNNs who have been considered for this project will be introduced.

2.2 Classification Architectures

For the IC problem, three types of models, all suited to the problem, will be introduced. The first one was introduced by Oxford University's *Visual Geometry Group* (VGG) in 2014 and thus is usually called the VGG architecture. This configuration of CNN introduced an at the time of its creation increased depth, even though much deeper models are common nowadays. A VGG model is made up of multiple "convolutional blocks". Each consists one to four convolutional layers with a relatively small receptive field size of 3 x 3, followed by a max pooling layer. Commonly, five such blocks are followed by three large fully connected layers to classify the processed images into the respective categories. Figure 2.1 illustrates how the architecture of VGG models looks like (Simonyan & Zisserman, 2014, pp. 1–3).



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.1: Overview of the different configurations of VGG models proposed in the original paper introducing the architecture
(Simonyan & Zisserman, 2014, p. 3)

Simonyan and Zisserman developed different sub-types of the VGG architecture, some of which are depicted in Figure 2.1. The exact name is determined by the number of weight layers, for example the model with 16 weight layers is commonly referred to as VGG-16. In the original paper, they range from 11 to 19 weight layers, but more types have been implemented since then. VGG can reach good performance on image classification tasks and can be applied to a wide array of problems, but lacks many of the refinements to CNNs developed since then, which will be shown in newer models in the next few sections

(Simonyan & Zisserman, 2014, pp. 3–14; He et al., 2016, pp. 772–773).

The next approach described in this section is a type of CNN called *residual neural network* (ResNet). Generally, very deep NNs are susceptible to a problem called *vanishing or exploding gradients*. This phenomenon causes gradients to become very small or very large, leading to the weights of the NN barely or rapidly changing. This of course interferes with training, as it leads to extremely small or extremely big changes in the loss function, making optimization difficult in both cases. Another problem occurring on large NNs is called *degradation*. Degradation causes the accuracy of deep classification models to first saturate and then to decline rapidly. ResNets are CNNs that solve both aforementioned problems by the introduction of *residual learning*, which enables information flow across more than one layer, meaning a neuron of one layer can be connected to one that is not necessarily in the next layer. This is accomplished by utilizing so-called *shortcut or skip connections*. Other than the connection to a later layer, these connections are also performing identity mapping instead of a nonlinear activation (He et al., 2016, pp. 770–773).

The mitigation of the problems mentioned in the last paragraph enables the construction of NNs that are much deeper than the ones implemented without residual learning. Like with VGG different variations of ResNet exist and are named after the number of layers they consist of, like ResNet50 or ResNet101. Such networks are able to deliver good performance, especially on complex problems and at the time of their introduction were able to outperform state-of-the-art methods on *ImageNet*, a database of images in 1000 different classes (He et al., 2016, pp. 774–777).

The final model introduced it *EfficientNet*, which was introduced by Tan and Le in 2019. True to its name, it is able to achieve to outperform all other CNN architectures that were common at the time of its creation, while possessing around an order of magnitude fewer parameters. The exact comparison, again on the classical benchmark of the ImageNet data set, can be seen in Figure 2.2 (Tan & Le, 2019, p. 6105).

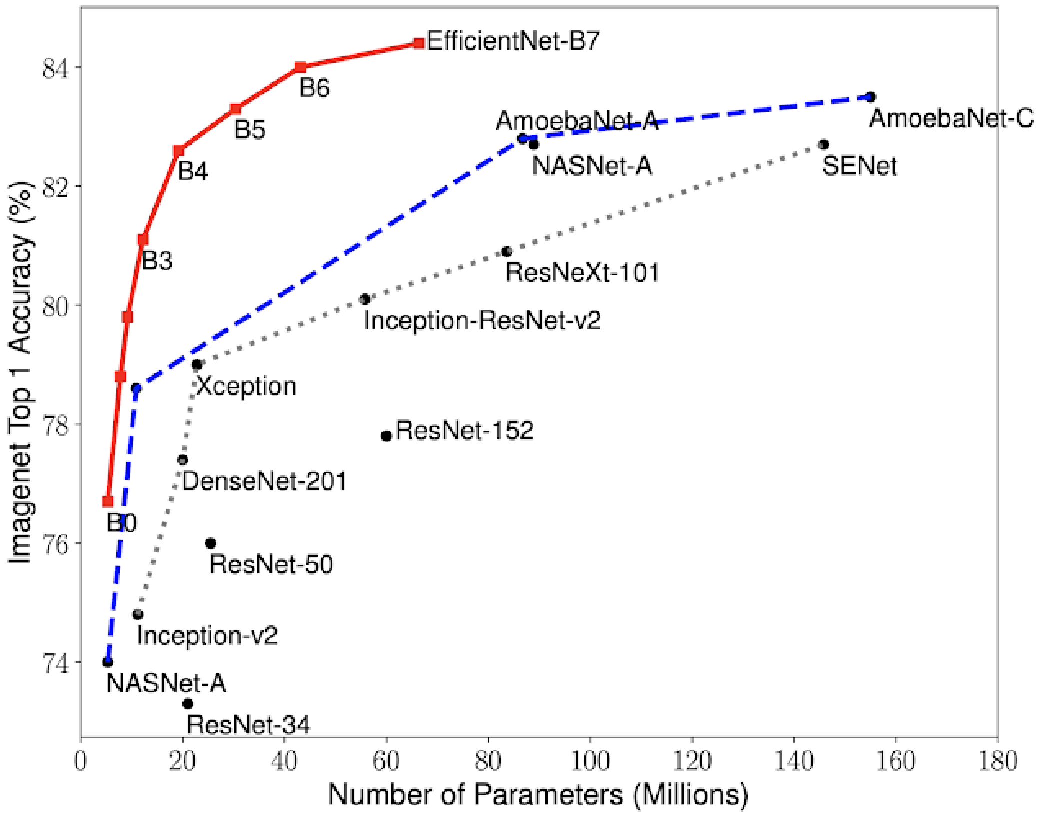


Figure 2.2: Performance and parameters of different IC models in the ImageNet data set (Tan & Le, 2019, p. 6105)

EfficientNet is able to achieve this impressive feat by multiple properties. First, the scalability is ensured by both a good baseline network, and the newly developed method of *compound scaling*, which scales not only a single parameter out of width, depth or resolution as other architectures do, but all three parameters together. The baseline network is based on the *mobile inverted bottleneck MBConv* as its main building block which was created by the same authors in earlier papers. Both allows for scaling the EfficientNet architecture from the baseline B0 up to B7, based on a factor that dictates the compound scaling. All architectures' performances and their comparison are shown in Figure 2.2 (Tan & Le, 2019, pp. 6108–6110).

2.3 Web App Components

Web applications are commonly implemented as distributed systems, consisting of *frontend*, *backend*, and database. A database is responsible for persistence, and not required in this case, since only the model will be saved and can be integrated into the backend. Commonly, the frontend handles user input and the presentation of results to the user, and is accessed by and rendered in the browser, and is often called the *client-side* of the application. The backend, also called *server-side* handles more complex tasks like calculations, and is also responsible for coordinating the access to other parts of the application, like in this case the model. Both services are required in a modern web application, and communicate via an so-called *application programming interface* (API), which is a term that encompasses all points in a system that are designed to interact with another system. Commonly, APIs use the standard of *representational state transfer* (REST), which imposes certain requirements to unify communication, and is characterized by the communication of not the entire, but only the change of state (Sohan et al., 2017, pp. 53–55; Abdullah & Zeki, 2014, pp. 85–87)

3 Technical Implementation

This chapter aims to show the process of implementing a web application to perform bird classification. It starts with an overview of the web application architecture and how the components work together before each component is discussed in detail. When explaining the model, the used data set and the performed preprocessing steps of us and the data set creators are also discussed.

3.1 Project Architecture

Basically, the developed web app can be divided into three parts: Frontend, backend and the CV model. The basic interaction of the three components can be seen in Figure 3.1.

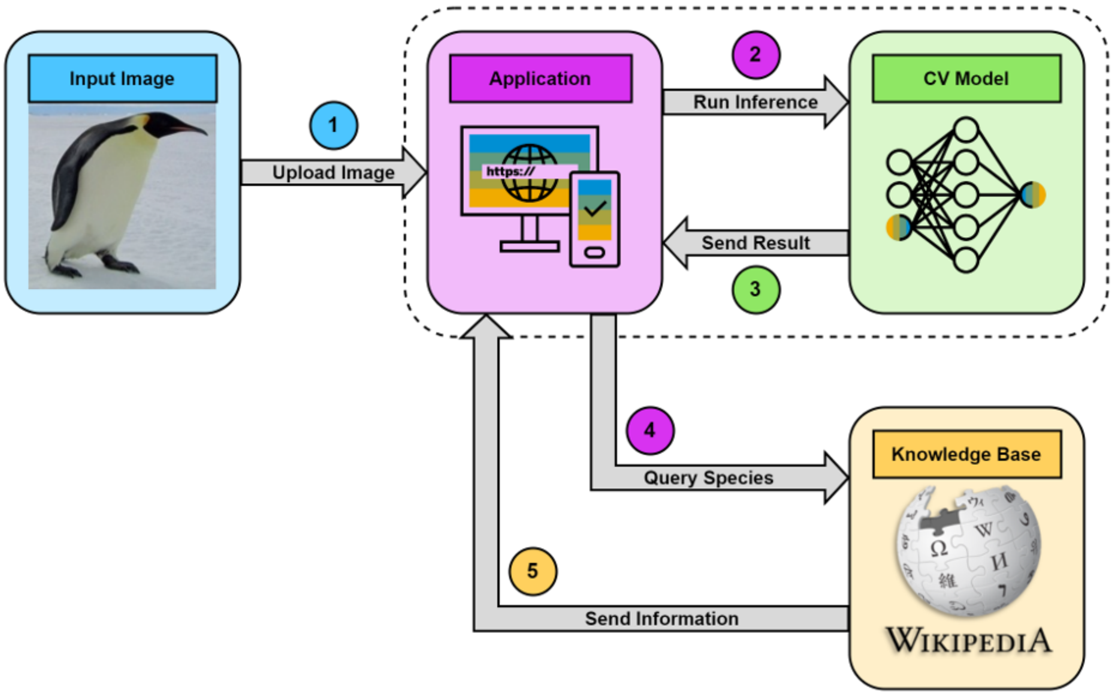


Figure 3.1: Architecture of the application
(Own Illustration)

The first point of contact a user has with the developed web app, and thus the starting point of the process behind it, is the frontend. The frontend enables the user to interact with the backend and the classification model by allowing them to upload an image of a bird to trigger the classification process. This uploaded image is then passed to the backend for further processing, i.e., classification and research of additional information. After the processing steps in the backend, the predicted bird species and the retrieved additional information are sent back to the frontend and displayed to the user. With this step the process behind the web app is completed.

Looking at the backend, it is noticeable that three process steps converge there, namely the interaction with the frontend, the interaction with the backend and the start of the Wikipedia search for additional information about the bird. Like mentioned in the explanation of the purpose of the frontend, the interaction between front- and backend includes receival of the uploaded image and the return of the classification and the retrieved information. Interaction with the classification model in the backend means passing the uploaded image to the model and receiving the classification to pass it to the search for additional information and the the frontend to complete the process. The biggest processing step of the backend is retrieving additional information about the bird from Wikipedia. For this, a Wikipedia/Google search query is created. The query is based on the classification of the model. To end the process the result of the query is then returned to the frontend together with the classification, as described above.

The most comprehensive part of the web app and the most important step in the process of the application is the model for classifying uploaded images. It receives the uploaded images from the backend, classifies them and returns the classification to the backend. This may sound relatively simple and like few steps at first glance, but in the model itself it is a complex process. The classification process was already touched on in 2 and is explained in more detail in 3.3.

The three individual components of the web app, namely frontend, backend and the classification model, are brought together in multiple Docker containers, which enable launching the web app independently of operating system and manage the installation of required packages.

3.2 Data Set

To train the classification model the BIRDS 525 SPECIES – IMAGE CLASSIFICATION data set from Kaggle was used. The data set contains of a total of 89.885 224x224x3 jpg images of 525 different bird species, which are distributed into training, test and validation sets as follows: 84.635 training images, 2.625 validation images, and 2.625 test images. To ensure balance while evaluating the model performance, the test and validation sets consist of 5 images for each species. In the training set, the number of images per class varies, so it is not perfectly balanced as the test and validation sets, yet it was ensured that each class contains at least 140 images. The highest number of images for one class is 273, which is nearly double the lowest number of images for one class but can be neglected as these high number of samples are outliers ($\tilde{5}$% of all classes) and most of the 525 classes have between 150 and 200 images.

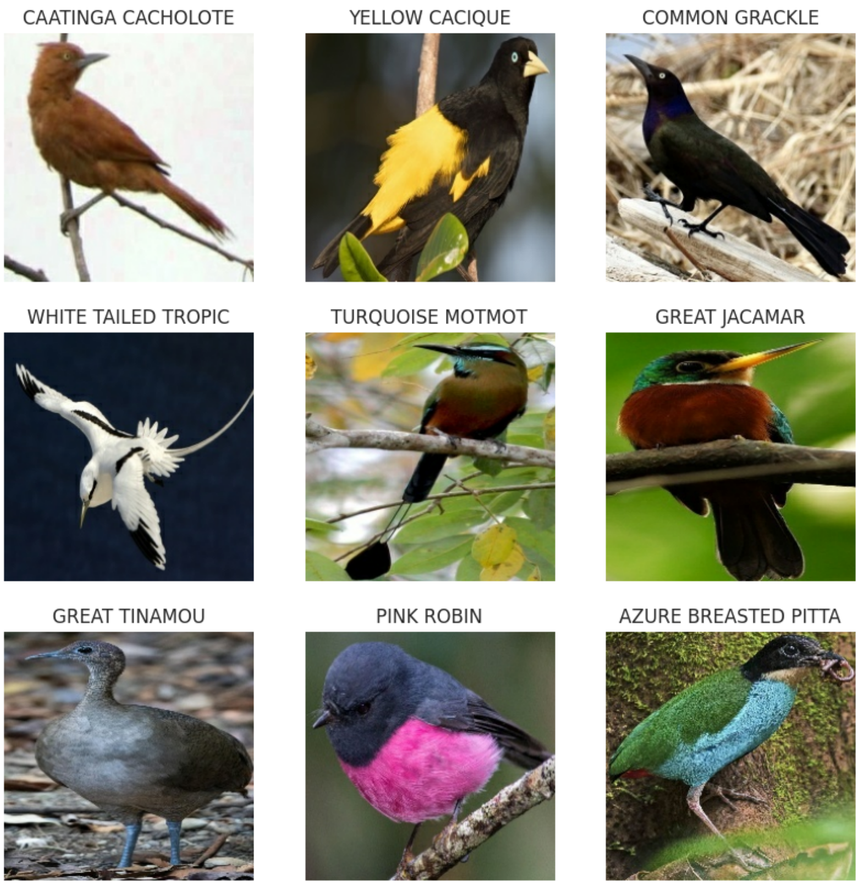
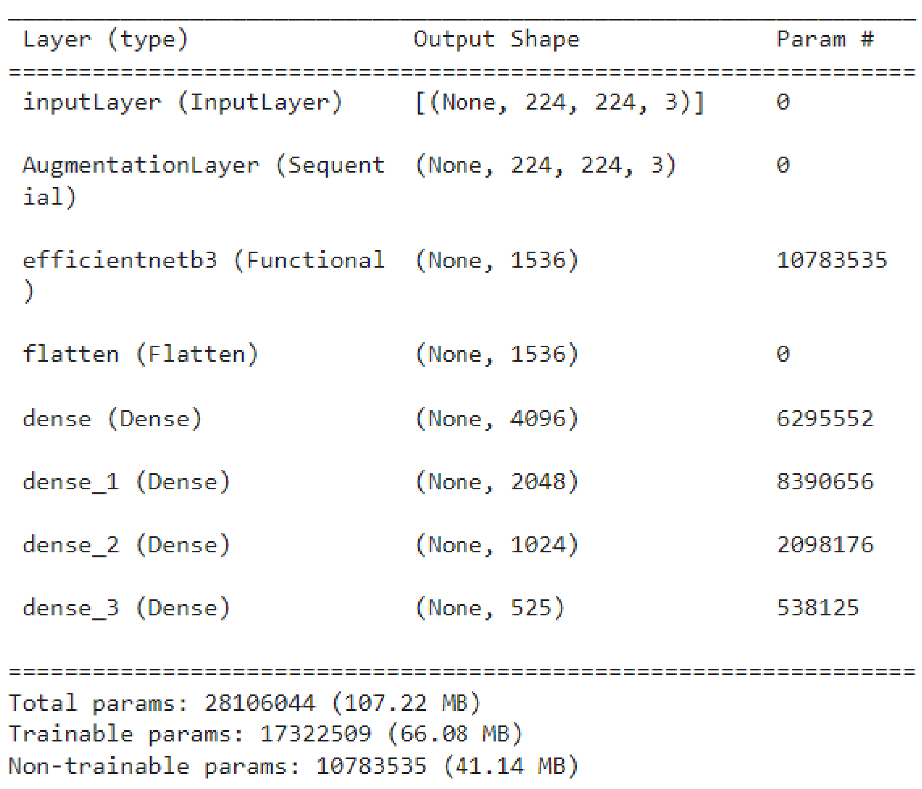


Figure 3.2: Images of the data set
(Own Illustration)

Additionally, there are several special features of the data set that must be noted here. The first thing that was done by the creators of the data set was to pre-clean it by filtering out duplicates, near duplicates and images with low information. This prevents leakage between train, test and validation set and increases the variability of the images. Looking at the images in Figure 3.2, it can be seen that the images are of very high quality and care has been taken to ensure that there is only one bird per image and that the bird accounts for at least 50% of the image. To optimize the evaluation of the models, the images for the test/validation sets were hand-selected by the data set creators to ensure the best images are used. It should be noted, however, that these sets consist mainly of male birds. This is due to the fact that the male/female ratio of the entire data set is approximately 80 to 20. Male and female birds differ significantly in appearance. Male birds are far more diversely colored, whereas female birds are balder. This dis-balance between male and female birds can cause the models to perform much worse when classifying female birds. However, this was not tested in detail as the test set only contains male birds because these have the greatest differences among themselves. ("BIRDS 525 SPECIES- IMAGE CLASSIFICATION", 2023)

3.3 Model

For the bird classification task of the web app an pretrained EfficientNetB3 from *TensorFlow* was utilized as base model. The base model was then enhanced with augmentation layers, a preprocessing layer and additional classification layers.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inputLayer (InputLayer)     [(None, 224, 224, 3)]     0

 AugmentationLayer (Sequent  (None, 224, 224, 3)       0
 ial)

 efficientnetb3 (Functional  (None, 1536)              10783535
 )

 flatten (Flatten)           (None, 1536)              0

 dense (Dense)               (None, 4096)              6295552

 dense_1 (Dense)             (None, 2048)              8390656

 dense_2 (Dense)             (None, 1024)              2098176

 dense_3 (Dense)             (None, 525)               538125

=================================================================
Total params: 28106044 (107.22 MB)
Trainable params: 17322509 (66.08 MB)
Non-trainable params: 10783535 (41.14 MB)
```

Figure 3.3: Layers of the model
(Own Illustration)

The data augmentation layers are used to boost the number of available training samples. During augmentation, the images are rotated, flipped, and translated as well as the contrast is changed, or the image is zoomed in or out in order to cover a broader range of possible input images of users. In previous versions of the EfficientNet from *TensorFlow*, the pre-processing layer was used to normalize the input data, whereas now it is only used to align the API entry points of the model. ("tf.keras.applications.efficientnet.preprocess_input", 2023) The four added classification layers are used to interpret the information gained from the CNN. The layers have a relatively high number of neurons as the images are relatively big (224x224x3) and as there are 525 different classes which need to be recognized.

The EfficientNetB3 is trained using an Adam optimizer and sparse categorical cross-entropy loss, which is suitable for multi-class classification. During training, the performance on the validation data set is monitored and triggers early stopping if no improvement to the loss occurs after 5 epochs. This prevents overfitting. The training progress is displayed in Figure 3.4 and shows a very consistent performance across training and validation data.

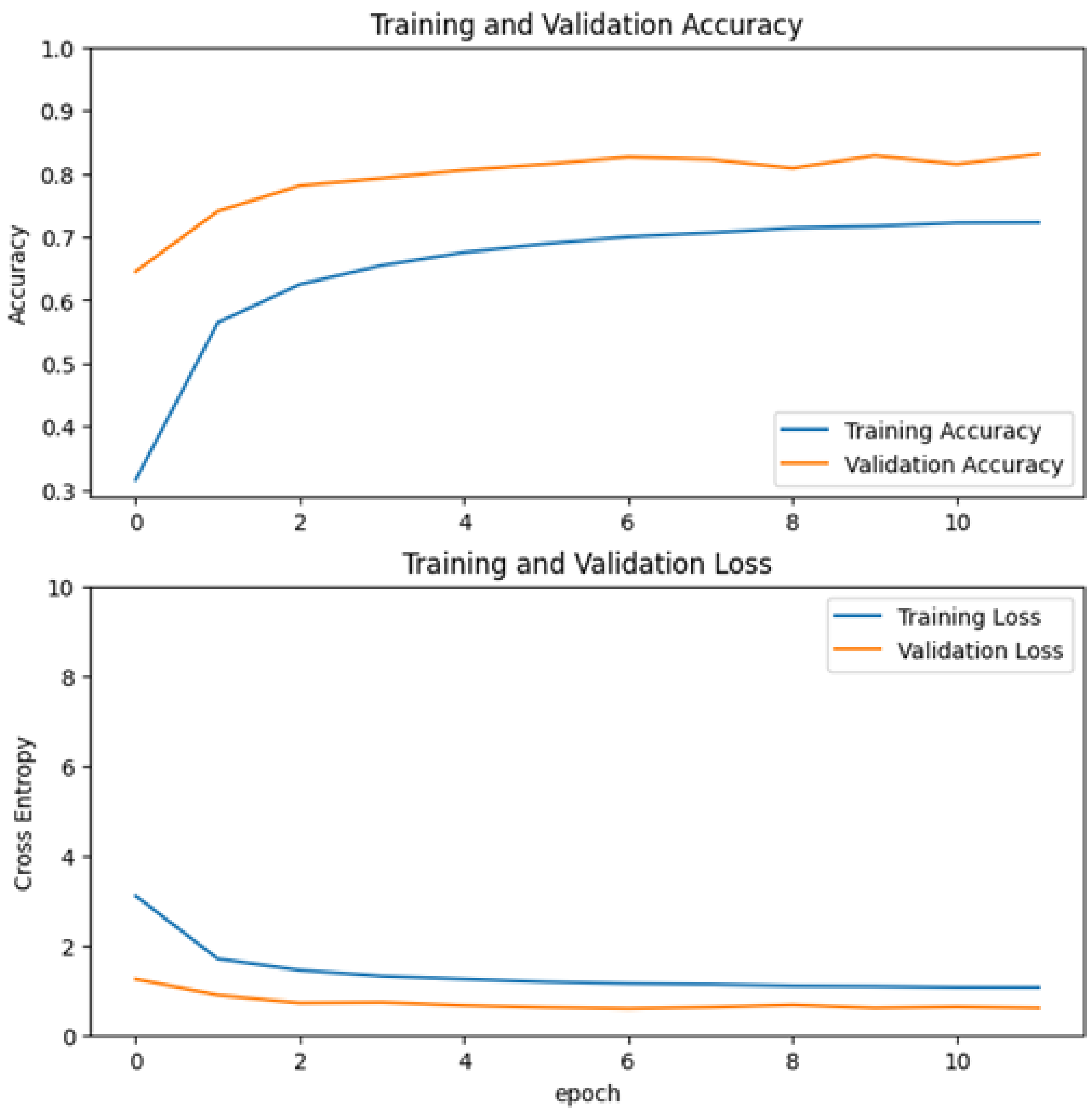


Figure 3.4: History of accuracy and loss of the EfficientNetB3
(Own Illustration)

Later, an evaluation of the model is conducted. The model accuracy is determined on all sets, as shown in Table 3.1, predictions are made for all images in the test set, and additionally, precision, recall and F1-score are determined and illustrated in **??** From the predictions, confusion matrices are calculated and, due to the relatively high number of classes, filtered for classes which show the most issues. One such matrix is depicted in Figure 3.5, showing three classes where the precision of the predictions was equal to or smaller than 0.25. Looking at the wrong predicted classes, it has to be noted that the predicted birds and the actual birds look very similar. There are only small differences in size of the bird or in the coloring, e.g., the predictions for the black necked stilt have the exact same colors (black/white) as the actual bird and only differ in size.

Data Set	**EfficientNetB3**
Training	82.99%
Validation	84.53%
Testing	85.37%

Table 3.1: Accuracy of the implemented model on all three sets

Metric	**Average**	**EfficientNetB3**
Accuracy	-	0.85
Precision	Macro	0.88
Precision	Weighted	0.88
Recall	Macro	0.85
Recall	Weighted	0.85
F1-Score	Macro	0.85
F1-Score	Weighted	0.85

Table 3.2: Different error metrics of the implemented model on the testing set

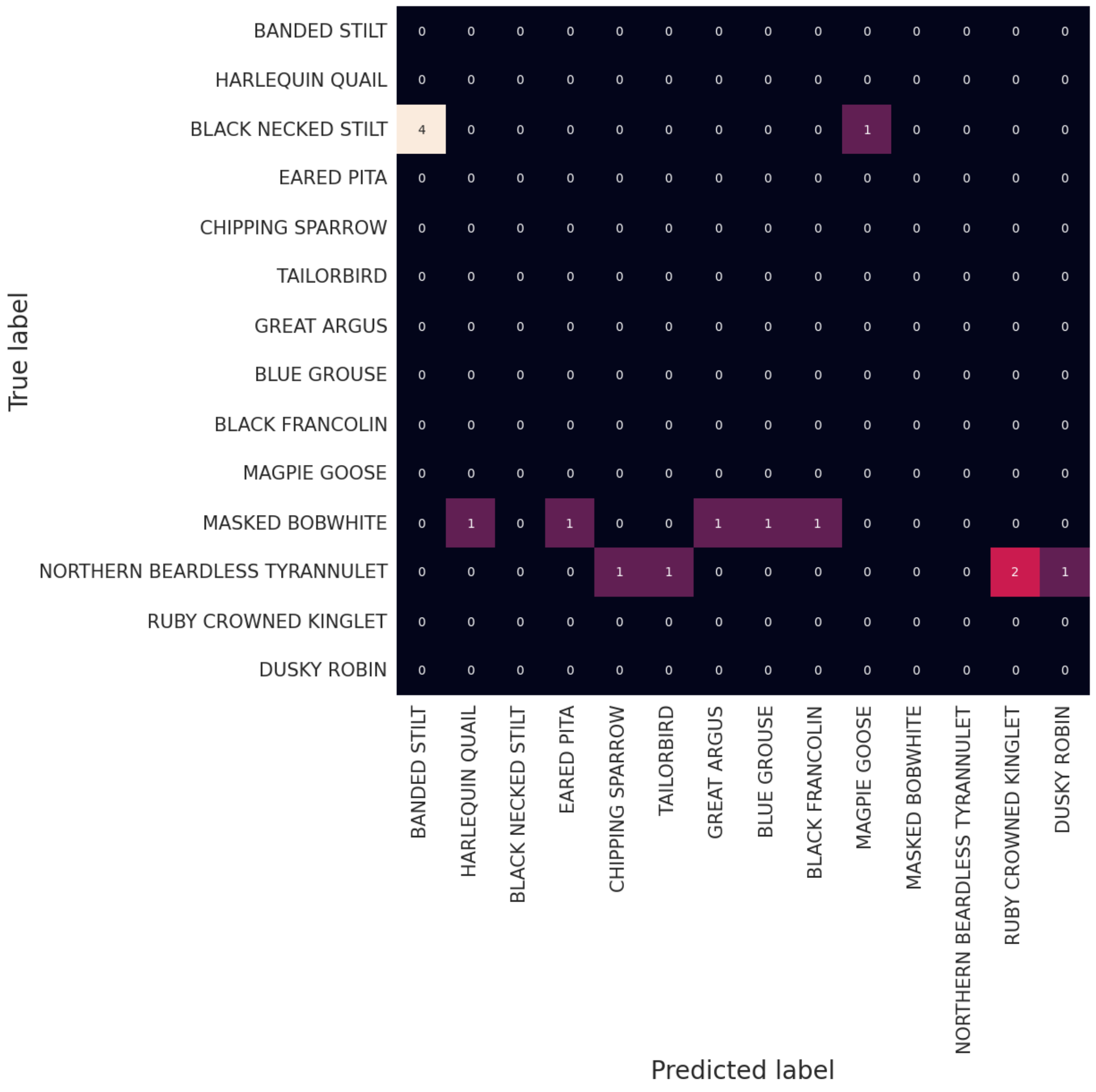


Figure 3.5: Confusion Matrix for the worst predicted classes
(Own Illustration)

As both tables show quite clearly, the EfficientNetB3 shows a good performance across all data sets and metrics. However, it is noticeable that the accuracy on the test and validation sets is higher, and the best accuracy was achieved on the test set. This can be attributed to the fact that, as described in section 3.2, the images for these two sets were selected manually and the highest quality images, i.e., those that are easiest to interpret, form the test set. It is therefore easier for the model to classify the images from the test set than those from the training set.

3.4 Backend

The backend is responsible for storing and organizing data and it belongs to the server side of the website. FastAPI is a web framework that is utilized to easily construct a REST-conform API using Python (Python, 7/14/2021). In addition, three functions with different paths are used to create APIs. These are used to submit input data to the backend and back to the frontend.

The first API-endpoint, `classify-birds`, is used for the integration of the trained EfficientNetB3 model. Additionally, there is a `test` endpoint to check if the model and backend are functional. The last API-endpoint, `bird-information`, allows searching for information about the identified bird in the input image. To accomplish that, a google search with the Python library Googlesearch is conducted. In this context, a search is made for the corresponding Wikipedia page of the respective bird species. Then this information is passed to the Wikipedia library, which stores either the summary or the entire page in a variable and sends it back to the frontend. The decision to determine whether the returned information contains the summary or the complete page is determined in the frontend through the selection of additional information.

Furthermore, it is also possible to access the Swagger user interface (UI) on the page `docs`. On this page, an interactive API documentation is automatically generated, and describes the structure of the API and can be used to test them in the browser without calling the frontend (Swagger, 7/23/2023).

3.5 Frontend

The React framework Next.js is utilized to build the web application by building blocks (Nextjs, 7/23/2023). The application is a single-page application that uses the REST-API the backend provides to retrieve predictions and information. Cascading Stlye Sheets (CSS) is used to style the application. On the website, images can be selected, uploaded and displayed through a button. Then, the trained model classifies the uploaded image in the backend application. After the model has identified the bird species, the name of the species and the confidence score are displayed to the user. Then, with another button, the summary of the Wikipedia page can be retrieved and displayed on the UI. In addition, by checking the box for detailed information, it is also possible to retrieve the full Wikipedia pages text and show it to the user.

3.6 Docker

Docker is used for deployment of the application, because it allows to deploy without additional software and in a operation-system-independent way. The components of the application are deployed on form of what is called *Docker-Containers*. For this, each container needs a *Dockerfile*, which allows for the the configuration of parameters like the port they are accessible at, and to set up commands, which in turn allow to install required dependencies for the applications, namely the Python modules required by the backend using pip and the `requirements.txt`, and the modules for the Next.js frontend with npm and the `package.json`. Because multiple Dockerfiles are required, a `docker-compose.yml` file serves as a central starting point, which allows for some additional configuration (Docker, 2023).

4 Conclusion

4.1 Summary

The goal of this project, to develop a web application that classifies birds, was achieved through creating a frontend that enables users to upload images, a CV model that classifies the images and a backend that connects the frontend and the model and retrieves additional information in form of summaries of complete Wikipedia entries about the bird.

For creating the frontend of the web app Next.js was utilized to build a single-page application of predefined building blocks. To improve the user experience CSS was used to style the application. The frontend itself then offers the possibility to select, upload and display images through clicking a button. After classifying the image and retrieving additional information from Wikipedia the classification together with a confidence value and the summary or the whole Wikipedia page are presented.

To classify the uploaded image a pretrained EfficientNetB3 was combined with additional self-defined augmentation and classification layers. The augmentation layers were used to rotate, flip, translate and change the contrast and zoom level of the images to cover a broader range of possible input images and to boost the number of samples to train on. To interpret the information extracted from the EfficientNetB3 four classification layers with a relatively high number of neurons were used as the images are quite big (224x224x3) and as there are 525 classes to predict. Overall the model performed quite good with an accuracy of 85.37% on the test set.

To combine the frontend and the classification model on the server side of the app the web framework FastAPI was utilized to create several API. In total three APIs were created, one to interact with the model for sending the image and retrieving the classification and the confidence, a second one the check the functionality of the model and the third one to retrieve additional information about the bird in form of summaries or whole pages from Wikipedia. The three API are accessible via Swagger UI to be able to test them independently from the frontend.

Finally, the three components were then combined in a Docker container to make the web app usable for everyone, regardless of operating system or installed packages.

4.2 Critical Reflection and Outlook

All in all, the results of the project were satisfying as the goal of developing a web app that classifies birds was achieved. Nevertheless, there are some challenges that have to be mentioned here. Firstly, there was a bug in *TensorFlow* versions below 2.13 which prevented the model from being saved. If this bug had not been fixed with the update to *TensorFlow* 2.13, it would not have been possible to adequately integrate the trained model into the web app. It would have had to be re-trained each time before using it with the help of training checkpoints, which would have increased the time to prediction considerably.

The second challenge which had to be solved was the search for additional information. At the start, the search for the entry of a bird was carried out directly via the Wikipedia API using the result of the classification. This caused problems because if the name in the classification did not match 100% with the name of Wikipedia entry; no entry could be found. This problem was solved by a workaround using the google search. The google search was used to search for the Wikipedia entry of the bird, which lead much more often to the correct result.

Third, there were two more model related challenges to solve. The first challenge was that at the beginning of the project the classification layer did not have enough neurons to learn and classify the large number of classes and the relatively large images. This was solved by simply increasing the number of neurons in the four classification layers. The second model related challenge was to find a model that made the right trade-off between the number of parameters/performance and the needed training time, since only limited training resources were available for the project. This problem was solved by using an EfficientNetB3, which has already good performance by using a comparatively low number of parameters. The performance of the EfficientNetB3 is based on the used compound scaling.

The goal of the project was fully achieved, but of course, as with any project, there are points that can be further developed in the future.

The first thing which could be done to enhance the develop web app would be to implement bird detection. This would require the development of a second model that recognizes birds in images and cuts them out before the classification. Such an detection model would give the advantage that even images where the bird is not the center and main part of the image could be classified.

The second thing that could be done to extend the developed web app would be to connect the web app directly to cameras or deploy it on smartphones to enable live classification. For this, however, two basic components of the web app would have to be adapted. First, in addition to the classification, an additional detection would have to be implemented, as mentioned above. Furthermore, both models (classification and detection) would have to be modified so that they can work with video inputs. Both of these changes are topics whose implementation and required computing power should not be underestimated.

Third, and most obviously, it could be tried to increase the performance of the classification by testing and training larger models, such as a VGG model or an AmoebaNet. These larger models could learn the small and subtle differences between individual birds even better and thus classify them better. Additionally, the focus of a new model could be on the classification of female birds, since they were not part of the test set as described in section 3.2 and show considerable differences to their male counterparts.

Bibliography

Abdullah, H. M., & Zeki, A. M. (2014). Frontend and backend web technologies in social networking sites: Facebook as an example. *2014 3rd international conference on advanced computer science applications and technologies*, 85–89.

Badgerland Birding. (2023). What's the difference between birding and bird watching? https://badgerlandbirding.com/2023/04/23/whats-the-difference-between-birding-and-bird-watching/?utm_content=cmp-true

Batchelor, B. G. (2012). *Machine vision handbook*. Springer.

Birding Outdoors. (2021). 7 ways technology is used in birding (with examples!) - birding outdoors. https://birdingoutdoors.com/technology-in-birding-a-birders-guide/

BIRDS 525 SPECIES- IMAGE CLASSIFICATION. (2023). Retrieved July 20, 2023, from https://www.kaggle.com/datasets/gpiosenka/100-bird-species

Cornell Chronicle. (7/23/2023). Ai-powered birdnet app makes citizen science easier | cornell chronicle. https://news.cornell.edu/stories/2022/06/ai-powered-birdnet-app-makes-citizen-science-easier

Docker. (2023). Docker docs: How to build, share, and run applications. https://docs.docker.com/

Gad, A. F., Gad, A. F., & John, S. (2018). *Practical computer vision applications using deep learning with cnns*. Springer.

GeeksforGeeks. (7/11/2019). Frontend vs backend. *GeeksforGeeks*. https://www.geeksforgeeks.org/frontend-vs-backend/

Green, G. (2/21/2022). Five ways ai is saving wildlife – from counting chimps to locating whales. *The Guardian*. https://www.theguardian.com/environment/2022/feb/21/five-ways-ai-is-saving-wildlife-from-counting-chimps-to-locating-whales-aoe

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Jeno, L. M., Grytnes, J.-A., & Vandvik, V. (2017). The effect of a mobile-application tool on biology students' motivation and achievement in species identification: A self-determination theory perspective. *Computers & Education*, *107*, 1–12. https://doi.org/10.1016/j.compedu.2016.12.011

Li, W., Li, D., & Zeng, S. (2019). Traffic sign recognition with a small convolutional neural network. *IOP conference series: Materials science and engineering*, *688*(4).

Liu, T., Ma, L., Cheng, L., Hou, Y., & Wen, Y. (2021). Is ecological birdwatching tourism a more effective way to transform the value of ecosystem services?-a case study of birdwatching destinations in mingxi county, china. *International Journal of Environmental Research and Public Health*, *18*(23). https://doi.org/10.3390/ijerph182312424

Nextjs. (7/23/2023). What is next.js? | learn next.js. https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs

On The Feeder. (2022). Differences between birdwatching & birding - which one's right for you? https://www.onthefeeder.com/birdwatching-vs-birding/

Python, R. (7/14/2021). Using fastapi to build python web apis. *Real Python*. https://realpython.com/fastapi-python-web-apis/

Saraswat, A. (1/11/2020). Bird watching and ecotourism. *Himalayan Gypsy*. https://www.himalayan-gypsy.in/bird-watching-and-ecotourism/

Sermanet, P., & LeCun, Y. (2011). Traffic sign recognition with multi-scale convolutional networks. *The 2011 international joint conference on neural networks*, 2809–2813.

Shankland, S. (9/6/2017). The flight stuff: Bird watching soars on digital wings. *CNET*. https://www.cnet.com/culture/birdwatching-easier-in-digital-age-with-apps-and-cameras/

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

Sohan, S., Maurer, F., Anslow, C., & Robillard, M. P. (2017). A study of the effectiveness of usage examples in rest api documentation. *2017 IEEE symposium on visual languages and human-centric computing (VL/HCC)*, 53–61.

Swagger. (7/23/2023). What is swagger. https://swagger.io/docs/specification/2-0/what-is-swagger/

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *International conference on machine learning*, 6105–6114.

tf.keras.applications.efficientnet.preprocess_input. (2023). Retrieved July 20, 2023, from https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet/preprocess_input

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2022). Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*.

Wei, L., Runge, L., & Xiaolei, L. (2018). Traffic sign detection and recognition via transfer learning. *2018 Chinese Control And Decision Conference (CCDC)*, 5884–5887.