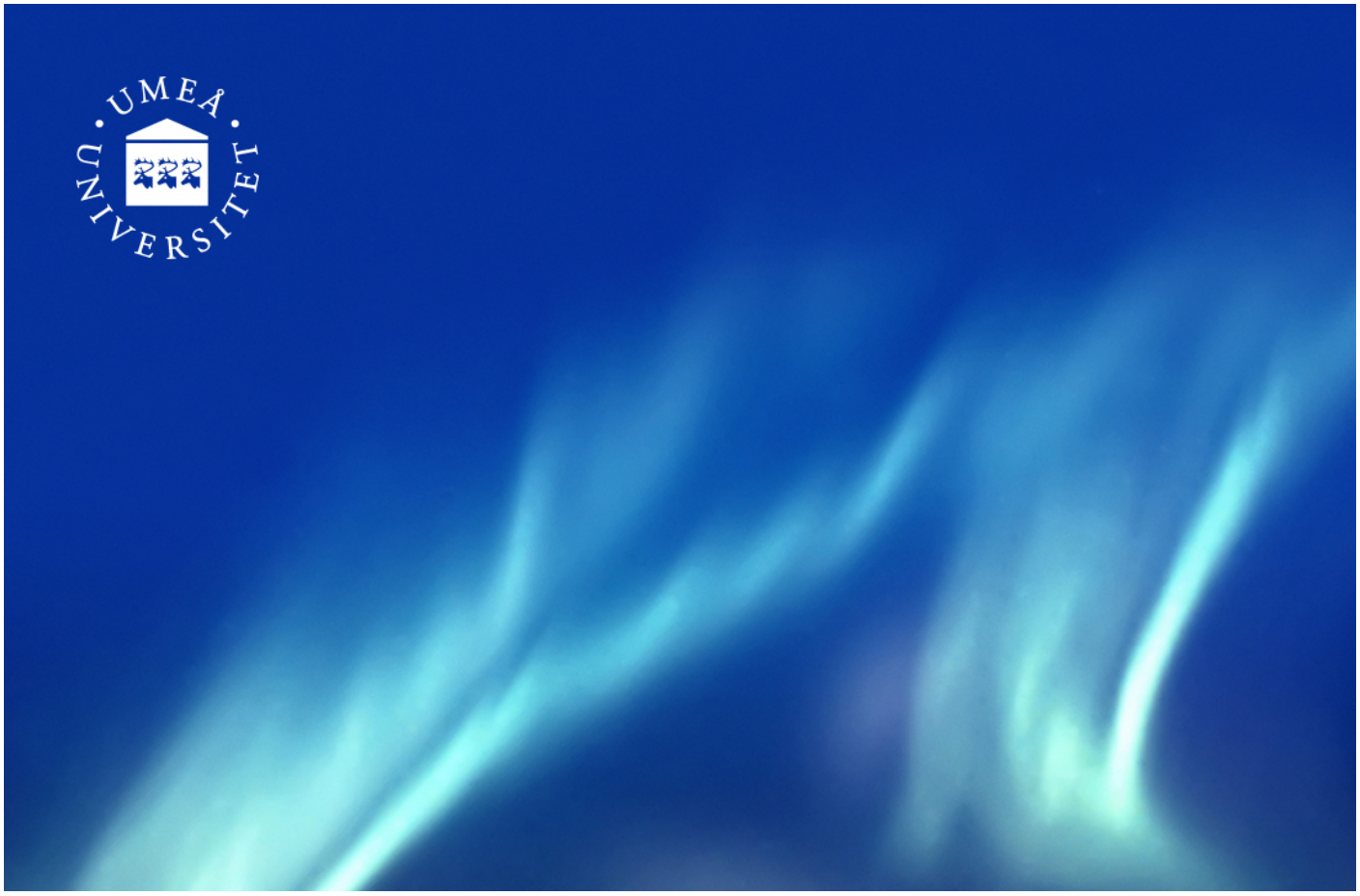

Final Project

Neural Network Models for Birdsong Classification

Classifying birds based on limited data

Anna Malmgren

June 23, 2022

Student
Spring 2022
Final Project
Deep Learning - metoder och tillämpningar

Google Colab Files

Three different Google colab files were used when performing this project. In the first file the raw data was explored. The second colab file performed the feature extraction and generated the needed datasets. Finally, the third colab file defined the models, their training and evaluation process, and the hyperparameter tuning.

Links to the colab files:

1. Data Exploration
2. Feature Extraction
3. Training and Evaluation

Contents

1 Introduction

Sustainable development is an important goal for increasing the quality of life for humans as well as animals. One important part of this is to halt biodiversity loss, as described in the United Nations 2030 Agenda for sustainable development [1]. According to the International Union for Conservation of Natures Red List of Threatened Species, more than 40,000 species are threatened with distinction, and about 13 percent of the world's bird species risk distinction [2]. Hence, wildlife monitoring has become a valuable resource of information needed in ecology research [3], [4]. The use of acoustic monitoring has been widely adopted, among other things due to its ability to cover larger areas and challenging environments [4], [5]. However, traditional methods for bird, and wildlife monitoring are both time consuming and expensive as field experts manually have to process huge amounts of data [6], [7]. Lately, advances in the deep learning field, yielding models with higher performance, have resulted in an increased interest in automated acoustic detection and classification [7], [8], [9]. Even though deep learning has made much progress, there are still challenges. When it comes to birdsong classification one of the challenges is the complex and diverse nature of birdsong, more over the audio data often contains background noise, birdsong from multiple birds, and species, in the same audio clip, making the classification process difficult [4].

This paper is a report for the final project for the course *"Deep Learning - metoder och tillämpningar"* held at Umeå University. The aim of this paper was to study the problem of acoustic birdsong classification, with the objective to propose a deep neural network model for the aforementioned task.

1.1 Related Work

Audio classification is subject of intense research, not least in the field of deep learning. The related work section in this paper presents a short summary of some recent attributions in the field, focusing on birdsong classification.

An interest in Long-Short-Term Memory (LSTM) has been seen in some of the studies. Yan et al. proposed a deep learning model composed of a 3d convolutional neural network (CNN) where the LSTM cells were used to replace the traditional fully connected layers [4]. The study also suggested using aggregate feature images as input to the model, and explored combinations of Chroma, Log-Mel Spectrogram, and Mel Frequency Cepstrum Coefficient (MFCC) [4]. The main idea of the feature fusion being that multiple audio features should make up for information loss in a single feature. Yan et al. reported that the proposed model and aggregate feature images were effective for birdsong classification [4]. Liu et al. instead proposed using a Bi-directional LSTM network combined with a dense convolutional network (DenseNet), using the audio feature MFCC as input data [10]. They compared the model with other deep learning models, namely VGG-16, VGG-19, ResNet, and DenseNet, and concluded that the proposed model achieved the highest accuracy in their experiments [10].

Recent studies have also shown an interest in transfer learning. Disabato et al. proposed using transfer learning in order to achieve an algorithm able to handle constraints, such as low memory and power capacities, and limited computational speed, creating a

model feasible for Internet of Things (IoT) devices [11]. They proposed using spectrogram images as input for the model, and base the model itself on ResNet-18 combined with a fully-connected layer to output the predictions [11]. In another study Kumar et al. compared different deep transfer models to explore if deep learning was an appropriate method for birdsong classification. They performed experiments using the models ResNet50, InceptionV3, EfficientNet, and Xception, and achieved an overall high accuracy, hence confirming deep learning as a fitting method for birdsong classification [12].

The problem of losing important features in CNNs was addressed in a study by Liu et al [13]. They proposed a multi-scale method which decomposed convolutional kernels, constructing kernels of multiple scales. To decrease the risk of feature loss they also proposed an ensemble multi-scale CNN (EMSCNN), which concatenated the output from multiple multi-scale CNNs [13]. In their study they report that the EMSCNN performed better than state of the art models such as LeNet, VGG16, ResNet101, MobileNetV2, EfficientNetB7, and Darknet53 [13].

In addition, to encourage the research in bio-acoustics and artificial intelligence, annual audio classification competitions have been held. One well known competition being the BirdCLEF challenge, which has presented bird sound recognition challenges annually since 2014 [14]. Another competition is the Challenge on Detection and Classification of Acoustic Scenes and Events (DCASE), which usually includes a task of localization or classification of wildlife such as birds [15].

1.2 Problem Formulation

Birdsong classification is considered a complex problem. The audio data often contains noise, multiple birds in the same clip, and the birdsong itself can be diverse, making the classification a challenging task [4], [16]. Recent research has focused much on deep learning approaches, proposing CNN, LSTM networks, reinforcement learning, and combinations of the aforementioned networks [4], [8], [10], [11], [13]. With those challenges in mind, this project aims to investigate which deep learning approach gives the highest performance for the birdsong classification problem, yielding the following questions:

Q1: Given a limited dataset, which neural network can provide the best performance; CNN, LSTM-CNN, or Transfer learning?

Q2: Based on the result for Q1, what are good hyperparameters for a model based on the best performing neural network?

Q3: Does clipping the birdsong audio into different lengths affect the performance of the model?

1.3 Scope/Limitation

The deep learning networks to explore have been limited to CNN and combining CNN and LSTM. Transfer learning will focus on high performing models, hence not exploring simpler transfer models such as MobileNetV2. Furthermore this project limits itself to focus on single feature input for the models, thus not investigating the potentials of feature fusion.

1.4 Outline

The rest of the paper is organized as follows. Chapter 2 describes what data has been used, and how it has been preprocessed. The deep learning models, the training process, and evaluation methods are also described. Chapter 3 presents the project's results by providing charts, tables, and descriptive texts. Finally, in chapter 4, the results are discussed.

2 Method

This chapter begins with presenting the raw data collected for this project, followed by the feature extraction and preprocessing of the data. Next the different models are presented, and finally the training process and evaluation method are defined.

2.1 Data

The data used for this project was the *Bird songs from Europe dataset* from Kaggle [17]. The dataset is built on data from the xeno-canto website and contains 50 different species, which in turn contains 43 audio file records of male birdsong each [17], [18]. Below follows some visualizations made when investigating the dataset, for the complete code and all visualization, go to the colab data exploration notebook. Figure 2.1 shows where the birdsong audio recordings have been made. As can be seen most of the recordings were from the United Kingdom, Poland, and Spain.

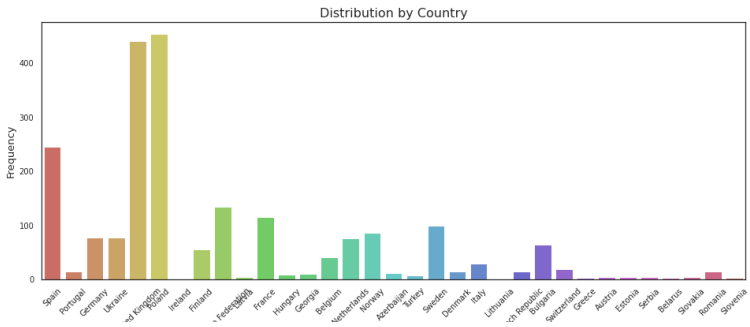

Figure 2.1: The chart visualizes in what countries the audio files have been recorded.

As can been seen in figure 2.2 most of the recordings were relatively recent, the majority ranging from year 2011 to 2019.

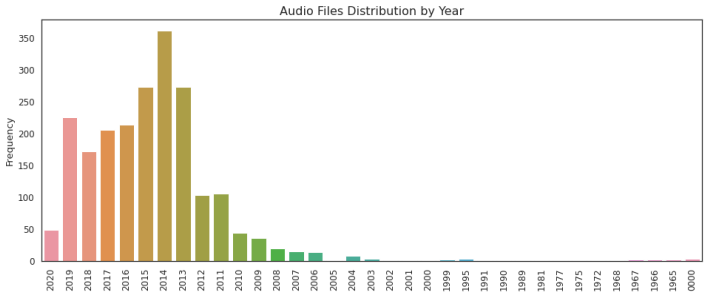

Figure 2.2: The chart visualizes in what year the audio files have been recorded.

The length of the audio recordings varied as well. The minimum length was 30 seconds, the most frequent duration, however, was between 50 - 100 seconds, as can be seen in figure 2.3.

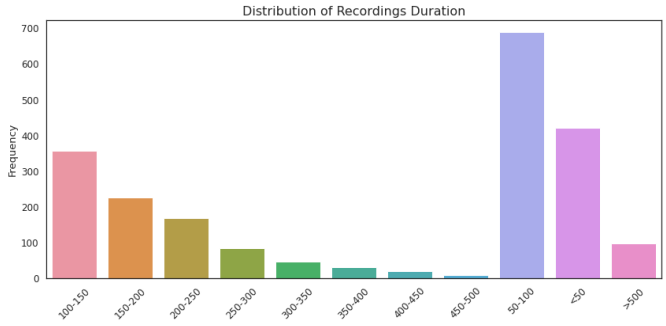

Figure 2.3: The chart visualizes the duration of the audio files in seconds, divided into intervals for easier overview.

Audio signals can be described by different features, either in the time domain, frequency domain, or the time-frequency domain. To classify audio using deep learning a good audio feature representation is essential. However, what features to use depends heavily on the system's task [19]. Figure 2.4 plots the waveform of the audio signal from three random audio files in the dataset.

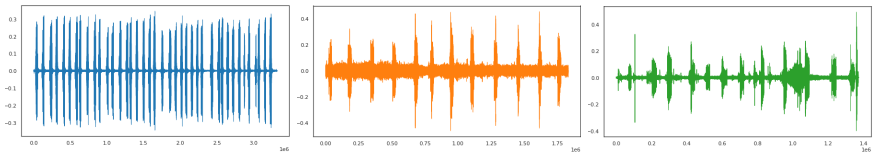

Figure 2.4: The figure visualizes the waveforms of three different audio files. The leftmost waveform is from a Sonus naturalis, the middle is from a Strix Aluco, and the rightmost waveform is of a Sylvia Curruca.

The waveform of an audio signal is a representation in the time domain, however, it is very common to visualize audio in the time-frequency domain where the signal's frequency spectrum is displayed over time, i.e., a spectrogram. Figure 2.5 shows the same three audio signals as in figure 2.4, however, visualizing their respective spectrograms instead.

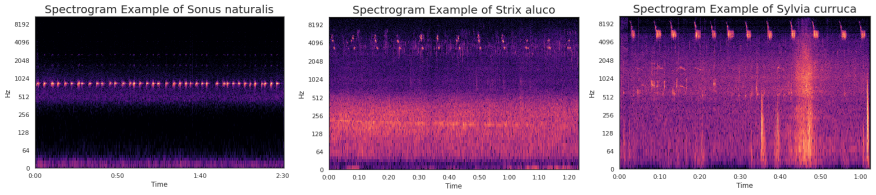

Figure 2.5: The figure visualized the spectrogram representations of the sound signals from three different audio files.

Mel Frequency Cepstral Coefficients (MFCCs) have been a frequently used feature in audio processing [19], [20]. MFCCs are a short-term power spectrum of an audio signal, based on a discrete cosine transform (DCT). As DCT has been shown to contribute to some degree of information loss, it's sometimes preferred to not include the DCT, yielding a log-mel spectrum instead [19]. Figure 2.6 shows the log-mel spectrograms generated from the same three audio files as in figure 2.4.

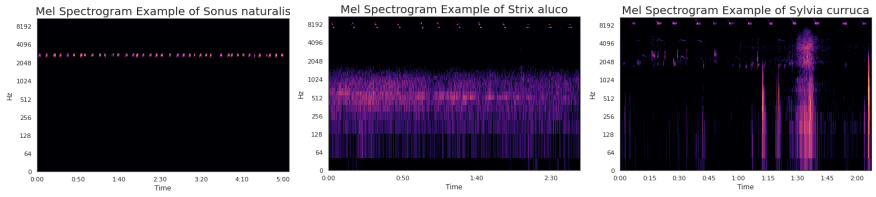

Figure 2.6: The figure visualizes the log-mel spectrograms of three different audio files, the same audio files shown as traditional spectrograms in figure 2.5 above.

2.2 Feature Extraction and Datasets

Deep learning approaches make it possible to use the sound signals, i.e. the waveforms, as input to the model, removing the process of manually extracting features, however, this approach usually requires huge amounts of data and computational power [19]. Another common audio signal representation for deep learning networks is the log-mel spectrogram, which often requires less data as well as training [19]. Therefore, based on the limited amount of data, the audio feature selected for this project was the log-mel spectrogram. As with the spectrogram, the mel spectrogram is based on computing the short-time Fourier transform, however, as the name suggests, the mel spectrogram visualizes the mel scale over time instead of the frequency over time [21] . In this project the python library Librosa was used to transform the audio signals to mel spectrograms, displayed in decibels as it's log-scaled, hence creating log-mel spectrograms [22]. Figure 2.7 displays some examples of the final log-mel spectrogram images, created for the dataset used in this project.

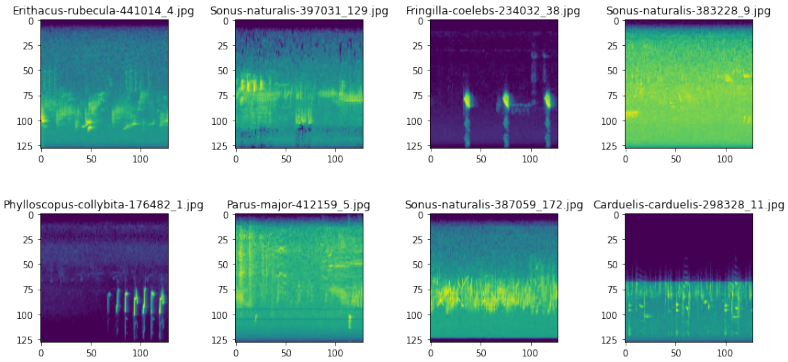

Figure 2.7: The figure shows eight normalized log-mel spectrograms. The log-mel spectrograms are generated from 5 second audio clips of birdsong, using the Librosa library.

The Bird songs from Europe dataset described in chapter 2.1 was used to generate this project's datasets. The audio files were originally in mp3 format, to get uncompressed audio files, they were transformed into waveform audio file format (wav). The waveform of each audio file was passed through a band-pass filter, filtering out too low, or high, frequencies. To further reduce noise, the audio signals were passed through librosa's pre-emphasis method. As the audio files were of different lengths, it was decided to clip each file into 5 second clips. Thus all mel spectrograms were based on the same amount of samples, and more images could be generated. To be able to explore if the duration of the audio clips affected the model's performance, a separate dataset based on

3 second audio clips, were generated as well. Each audio clip was transformed to a log-mel spectrogram, using Librosa's default settings of 128 mel bands, a Fourier transform window of 2048 samples, the overlap, i.e. the samples between successive frames, set to $(duration \times sample\_rate)/(melbands - 1)$. The sample rate was set to 32000 Hz instead of Librosa's default value of 22050, as the lower sample rate created some empty mel bins. For the complete code, and all configurations, see the colab file feature extraction. The created log-mel spectrograms were normalized using z-score normalization and scaled with min-max scaling, as shown in [23]. Finally, the generated log-mel spectrograms were saved as single channel images, and the dataset was created by selecting the log-mel spectrogram images from 15 bird species. The dataset was split into 90% training data and 10% test data, and the training data was further split into 20% validation data. Figure 2.8 shows the data distribution between the selected classes in the dataset, revealing an unbalanced dataset.

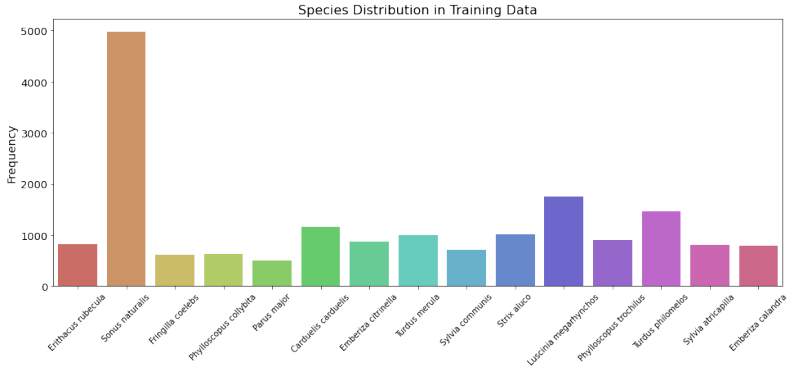

Figure 2.8: The figure shows the number of data records for each class in the dataset. As can be seen the dataset is unbalanced, with the class Sonus naturalis containing, by far, the most records.

2.3 Models

As the birdsong audio data were to be fed to the neural network as log-mel spectrogram images, the use of a CNN model was considered appropriate. Hence, two baseline models, a simple CNN described in table 2.1, and a slightly larger CNN shown in table 2.2 were defined. As some recent work suggested the use of both CNN and LSTM, a CNN-LSTM baseline model, presented in table 2.3, was explored as well [4], [10].

Table 2.1: Summary of the simple CNN model, *f* is short for filters, *ks* for kernel size, *ps* for pooling size, and *dr* for dropout rate. For brevity 2 × indicates the use of two identical layers.

Simple CNN Model		
Layer (Type)	Configurations	Output shape
2 × Conv2d	f=32, ks=(3 * 3)	(None, 128, 128, 32)
Batch Normalization		(None, 128, 128, 32)
MaxPooling2D	ps=(2 * 2)	(None, 64, 64, 32)
2 × Conv2d	f=64, ks=(3 * 3)	(None, 64, 64, 64)
Batch Normalization	-	(None, 64, 64, 64)
MaxPooling2D	ps=(2 * 2)	(None, 32, 32, 64)
Flatten	-	(None, 65536)
Dense	units=128	(None, 128)
Dropout	dr=0.3	(None, 128)
Dense	units=15, softmax	(None, 15)
Total params: 8,456,047		
Trainable params: 8,455,855		

Table 2.2: Summary of the larger CNN, *f* is short for filters, *ks* for kernel size, *ps* for pooling size, and *dr* for dropout rate. For brevity 2 × indicates the use of two identical layers.

Larger CNN Model		
Layer (Type)	Configurations	Output shape
2 × Conv2d	f=32, ks=(3 * 3)	(None, 128, 128, 32)
Batch Normalization		(None, 128, 128, 32)
MaxPooling2D	ps=(2 * 2)	(None, 64, 64, 32)
2 × Conv2d	f=64, ks=(3 * 3)	(None, 64, 64, 64)
Batch Normalization	-	(None, 64, 64, 64)
MaxPooling2D	ps=(2 * 2)	(None, 32, 32, 64)
2 × Conv2d	f=128, ks=(3 * 3)	(None, 32, 32, 128)
Batch Normalization		(None, 32, 32, 128)
MaxPooling2D	ps=(2 * 2)	(None, 16, 16, 128)
Flatten	-	(None, 32768)
Dense	units=128	(None, 128)
Dropout	dr=0.3	(None, 128)
Dense	units=15, softmax	(None, 15)
Total params: 4,483,695		
Trainable params: 4,483,247		

Table 2.3: Summary of a combined CNN-LSTM model, *f* is short for filters, *ks* for kernel size, *ps* for pooling size, and *dr* for dropout rate. For brevity 2 × indicates the use of two identical layers.

CNN-LSTM Model		
Layer (Type)	Configurations	Output shape
2 × Conv2d	f=32, ks=(3 * 3)	(None, 128, 128, 32)
Batch Normalization		(None, 128, 128, 32)
MaxPooling2D	ps=(2 * 2)	(None, 64, 64, 32)
2 × Conv2d	f=64, ks=(3 * 3)	(None, 64, 64, 64)
Batch Normalization		(None, 64, 64, 64)
MaxPooling2D	ps=(2 * 2)	(None, 32, 32, 64)
2 × Conv2d	f=128, ks=(3 * 3)	(None, 32, 32, 128)
Batch Normalization		(None, 32, 32, 128)
MaxPooling2D	ps=(2 * 2)	(None, 16, 16, 128)
Permute	(2, 1, 3)	(None, 16, 16, 128)
Time Distributed	Flatten	(None, 16, 2048)
LSTM	units=128, dr=0.25	(None, 16, 128)
LSTM	units=128, dr=0.25	(None, 128)
Dense	units=15, softmax	(None, 15)
Total params: 1,535,471 Trainable params: 1,535,023		

As can be seen in table 2.3, the CNN in the CNN-LSTM model has the same configuration as the larger CNN presented in table 2.2. To connect the convolutional layers with the LSTM layers the TensorFlow Keras method *Permute* was implemented to get the correct input structure for the LSTM.

In addition to the three baseline models described above, two transfer learning baseline models were implemented. The Xception model, which is a deep CNN that makes use of depthwise separable convolutions, was chosen as the model for transfer learning [24]. Depthwise separable convolutions are often used to reduce the size and the computational cost compared to a standard CNN [24]. The model was retrieved using Keras built in function *Xception* [25]. The number of layers to use from the Xception model was explored and presented in chapter 3.1. The first transfer learning model was based on Xception and two dense layers, and the second model was based on Xception and two LSTM layers, as can be seen in tables 2.4 and 2.5 respectively.

Table 2.4: Summary of the transfer learning model with Xception and dense layers, *dr* is short for dropout rate, and *layer* is the selected output layer from Xception

Xception Model (X-Base)		
Layer (Type)	Configurations	Output shape
Xception output	layer='block3_pool'	(None, 16, 16, 256)
Flatten	-	(None, 65536)
Batch Normalization	-	(None, 65536)
Dense	units=128	(None, 128)
Batch Normalization	-	(None, 128)
Dropout	dr=0.5	(None, 128)
Dense	units=15, softmax	(None, 15)
Total params: 8,812,847 Trainable params: 8,521,999		

Table 2.5: Summary of the transfer learning model with Xception and LSTM layers, *ps* is short for pooling size, *dr* for dropout rate, and *layer* is the selected output layer from Xception

Xception-LSTM Model (X-LSTM)		
Layer (Type)	Configurations	Output shape
Xception output	layer='block3_pool'	(None, 16, 16, 256)
MaxPooling2D	ps=(2*2)	(None, 8, 8, 256)
Permute	(2, 1, 3)	(None, 8, 8, 256)
Time Distributed	Flatten	(None, 8, 2048)
LSTM	units=128, dr=0.25	(None, 8, 128)
LSTM	units=128, dr=0.25	(None, 8, 128)
Dense	units=15, softmax	(None, 15)
Total params: 1,408,687 Trainable params: 1,248,655		

All the models were implemented using TensorFlow and Keras. All the baseline models described above were trained and evaluated using the same dataset, and training hyperparameters, described in chapter 2.4. The baseline model with the best performance were selected for further hyperparameter tuning. The implementation of the models, their training, tuning, and evaluation, can be found in the training and evaluation colab file.

2.4 Training and Evaluation

As the models were implemented using TensorFlow and Keras, so was the training. The models where trained using callback methods for early stopping, and reduced learning rate on plateau. The early stopping method monitored the validation loss, with a patience of 4 epochs, and was set to restore the best weights. The reduce learning rate on plateau method was also configured to monitor the validation loss, with a factor of 0.2 and a

patience of 2 epochs. The batch size was set to 32, maximum epochs to 100, and the optimizer used was Adam. Figure 2.9 shows a high level overview of the process of selecting and tuning the final model.

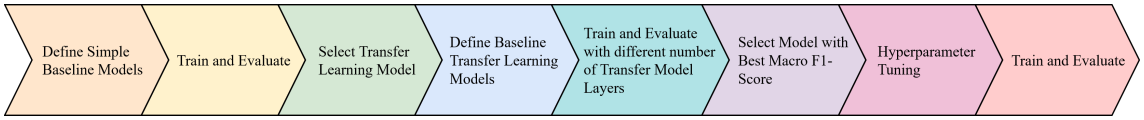

Figure 2.9: High level overview of the process of retrieving the final model for this project.

As this project aimed to solve a classification problem, and as cross entropy is the common choice as loss function for such problems, it was selected as the loss function for the models in this study [26]. Keras offer various implementations of cross entropy loss functions, in this case the categorical cross entropy function was implemented. The reasoning behind the choice being that, naturally the binary functions were ruled out due to the multiclass classification task in this project, and the sparse categorical cross entropy function was ruled out since the F1-score metric required the labels as binary matrices.

As can be seen in chapter 2.2 the generated datasets for this project were unbalanced, the class *Sonus naturalis* contained far more records than the other classes, hence accuracy was considered insufficient as a performance metric. In a study Saito and Rehmsmeier compared Receiver Operating Characteristics (ROC) plots and Precision/Recall plots [27]. Based on the results they argued that Precision/Recall often was the better performance measure for binary classification with unbalanced datasets [27]. It should be noted that Saito and Rehmsmeier's findings may not be applicable in this context, as this study concerns multiclass classification. However, based on the aforementioned findings, and as precision and recall was considered easier to understand than ROC, the macro F1-score was selected as the main performance metric for evaluating the models' performance in this project. The F1-score can be defined as the harmonic mean of precision and recall, the macro F1-score is obtained by calculating the F1-score for each class and then return the mean of those F1-scores [28]. Furthermore, as extra information, the accuracy and weighted F1-score, which is defined by TensorFlow's documentation as "the mean weighted by the number of true instances in each class", were added to the metrics [29].

After training and evaluating all the baseline models, the model with the best performance, i.e. the model with the highest macro F1-score, was selected for hyperparameter tuning. The explored hyperparameters, their values, and the model's performance are presented in chapter 3.2, and the full implementation can be found in the training and evaluation colab file.

3 Result

This chapter presents the project's results. First the results of training and evaluating the baseline models are described, followed by the results of tuning the baseline model with best performance. Lastly the final model and its performance is presented.

3.1 Training and Evaluation

The training and validation loss and macro F1-score were plotted for each model. As can be seen in figure 3.1, the simple CNN doesn't overfit but is too simple to achieve high performance. Similarly the larger CNN, while starting to slightly overfit, still doesn't provide desired performance. Figure 3.1 also shows that the CNN-LSTM model had the best performance of the three models, however, at the cost of overfitting more.

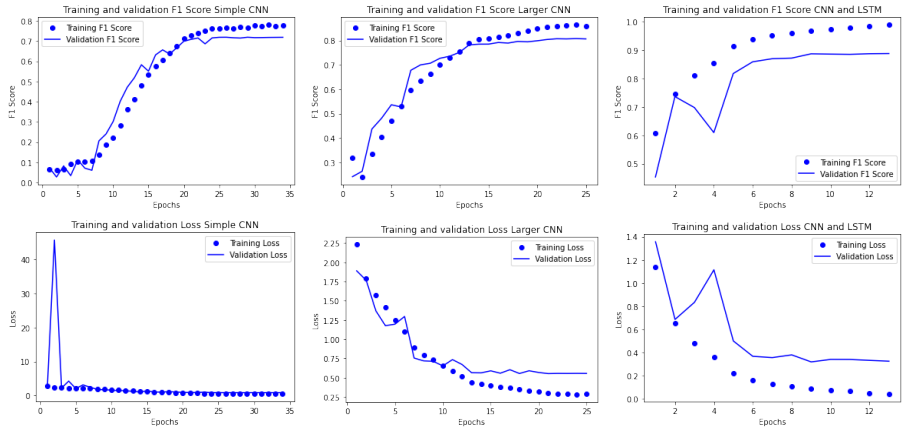

Figure 3.1: The training and validation macro F1-score (top row) and loss (bottom row) for the two baseline CNN models and the CNN-LSTM model.

As can bee seen in figure 3.2 the transfer learning models had the ability to achieve higher performance, especially the transfer learning model with LSTM layers (X-LSTM).

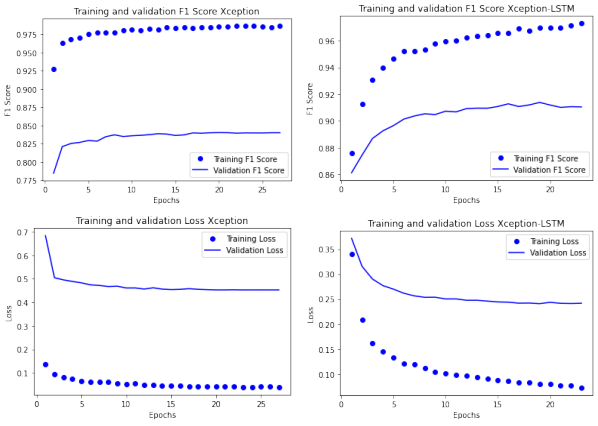

Figure 3.2: The training and validation macro F1-score (top row) and loss (bottom row). The leftmost column shows the dense (X-Base) model implementing Xception layer *block3_pool* and tuning from model layer 4. The rightmost column shows the X-LSTM model implementing Xception layer *block3_pool* and tuning from model layer 14

However, it should be mentioned that the plots in figure 3.2 also show that the transfer learning models were overfitting more than the simpler models.

After training, each of the models were evaluated. Table 3.1 shows the results of the two baseline CNN models and the CNN-LSTM model. As with the training plots in figure 3.1, the evaluation confirms that the CNN-LSTM model had the best performance, with both highest F1-scores as well as lowest loss of the three.

Table 3.1: Summary of the evaluation of the simple CNN, larger CNN, and the LSTM-CNN model. The model with best performance, the CNN-LSTM, is marked with bold text

Baseline Models				
Model	Macro F1	Weighted F1	Accuracy	Loss
Simple CNN	0.7165	0.7868	0.7902	0.8170
Larger CNN	0.8039	0.8583	0.8597	0.5527
CNN-LSTM	**0.8869**	**0.9191**	**0.9192**	**0.3173**

The results of the transfer learning baseline models were dependent on which layers were included from the Xception model. Both including too many or too few layers affected the performance negatively, as can be seen in table 3.2. Generally the X-LSTM transfer learning model performed better than the X-Base model. As the table 3.2 shows, the best performance was achieved by the LSTM transfer learning model that included Xception's layers til the layer named *block3_pool*, and fine tuned the final model from layer 14.

Table 3.2: Summary of evaluating the two transfer learning models using different layers from the Xception model. The configurations achieving the best performance are marked in bold, i.e., using LSTM layers, Xception layer *block3_pool*, and tuning from layer 14.

Transfer Learning Layers					
Model	Xception Output Layer	Tuning Layer	Macro F1	Accuracy	Loss
X-Base	block7_sepconv3_bn	4	0.8518	0.8937	0.4295
X-LSTM	block7_sepconv3_bn	4	0.8427	0.8863	0.4155
X-Base	block5_sepconv3_bn	4	0.8618	0.9014	0.3667
X-LSTM	block5_sepconv3_bn	4	0.8753	0.9107	0.3525
X-Base	block3_pool	4	0.8397	0.8821	0.4524
X-LSTM	block3_pool	4	0.9076	0.9363	0.2405
X-LSTM	block2_pool	4	0.8876	0.9203	0.3084
X-LSTM	**block3_pool**	**14**	**0.9138**	**0.9378**	**0.2409**

3.2 Hyperparameter Tuning

As shown in chapter 3.1, the baseline model which provided the best performance was the X-LSTM transfer learning model, hence, it was selected as the best model. The next step was to tune some of its hyperparameters, starting with the batch size, shown in table 3.3.

Table 3.3: Summary of the tuning of the model's batch size. The choice of batch size was made based on the best macro F1-score, hence batch size 64 was selected for the final model.

Batch Size Tuning Results			
Batch size	Macro F1-score	Loss	Epochs
32	0.9091	0.2485	25
64	**0.9182**	**0.2450**	**26**
128	0.9123	0.2447	28

After deciding on the batch size, the number of LSTM layers to implement and their number of units were explored. Table 3.4 presents a summary of the explored combinations and their results. As can be seen, including too many or too few layers or units decreased the models performance. The results also indicated that if multiple LSTM layers were used, it was preferable to implement a higher number of units in the first LSTM layer. When it comes to the number of layers, the results suggest that two LSTM layers were the optimal amount, with 256 units followed by 128 units.

Table 3.4: Summary of the explored number of LSTM layers as well as the number of units for each layer. The number of layers and units were selected based on the highest macro F1-score, in this case 2 LSTM layers with 256 and 128 units, and is marked with bold in the table.

Layers and units					
Layer 1	Layer 2	Layer 3	F1	Loss	Epochs
64	-	-	0.8703	0.3301	31
128	-	-	0.8828	0.2896	22
256	-	-	0.8924	0.2725	25
64	64	-	0.9045	0.2517	25
128	128	-	0.9182	0.2450	26
128	256	-	0.9136	0.2365	24
256	**128**	-	**0.9235**	**0.2211**	**26**
256	256	-	0.9112	0.2409	19
256	128	64	0.9107	0.2373	21
256	128	128	0.9120	0.2412	21
256	256	128	0.9102	0.2476	18

To reduce the overfitting, regularization techniques were explored. Table 3.5 summarizes the explored dropout rates for the two LSTM layers. As both the macro F1-score and the

Loss were very close in value for the dropout rate 0.25 and the dropout rate 0.3, the best value for dropout rate couldn't be confirmed.

Table 3.5: Summary of the evaluated dropout rates for the two LSTM layers

Dropout Rate			
Layer 1	Layer 2	Macro F1-score	Loss
0.25	0.25	0.9235	0.2211
0.3	0.3	0.9219	0.2210
0.35	0.35	0.9152	0.2218
0.4	0.4	0.9139	0.2287

To further regularize the LSTM layers, TensorFlow Keras *kernel regularizer*, which is applied to the kernel weights, and the TensorFlow Keras *activity regularizer*, which is applied to the output of the layer, was explored. Table 3.6 shows the tested configurations and their results. The results suggest that both kernel- and activity regularization should be used to achieve the best performance.

Table 3.6: Summary of the explored kernel and activity regularization configurations for the LSTM layers

Regularization LSTM Layers				
Kernel (L2)	Activity (L2)	F1-score	Loss	Epochs
0.01	-	0.8880	0.4014	57
0.001	-	0.9187	0.3390	44
0.0001	-	0.9224	0.3595	36
0.0001	**0.0001**	**0.9283**	**0.3279**	**60**
0.00001	0.00001	0.9124	0.3039	21

Finally regularization for the dense layer was tuned. As can be seen in table 3.7, implementing a L2 kernel regularizer with the value 0.0001 gave higher performance than not implementing any regularizer.

Table 3.7: Summary of tested regularization configurations for the model's dense layer.

Regularization Dense Layer			
Kernel (L2)	F1-score	Loss	Epochs
None	0.9242	0.3482	32
0.001	0.9226	0.3364	44
0.0001	**0.9283**	**0.3279**	**60**

3.3 Final Model

Based on the exploration of the baseline models, presented in chapter 3.1, and the tuning, presented in chapter 3.2, the final model was defined. Table 3.8 shows a summary of the final model.

Table 3.8: Summary of the final model, *ps* is short for pooling size, and *dr* is short for dropout rate.

Final Model		
Layer (Type)	Configurations	Output shape
Xception output 'block3_pool'	tune from layer 14	(None, 16, 16, 256)
Batch Normalization	-	(None, 16, 16, 256)
MaxPooling2D	ps=(2*2)	(None, 8, 8, 256)
Permute	(2, 1, 3)	(None, 8, 8, 256)
Time Distributed	Flatten	(None, 8, 2048)
LSTM	units=256, dr=0.3, l2=1e-4	(None, 8, 256)
LSTM	units=128, dr=0.3, l2=1e-4	(None, 128)
Dense	units=15, softmax	(None, 15)
Total params: 2,719,919 Trainable params: 2,559,887 Trainable params tuning: 2,662,927		

The final model was trained and evaluated with the dataset based on 5 second audio clips, as well as with the dataset based on 3 second audio clips, described in chapter 2.2. As can be seen in figures 3.3 and 3.4, the model overfits for both the datasets, even though the 3 second clips generate more data.

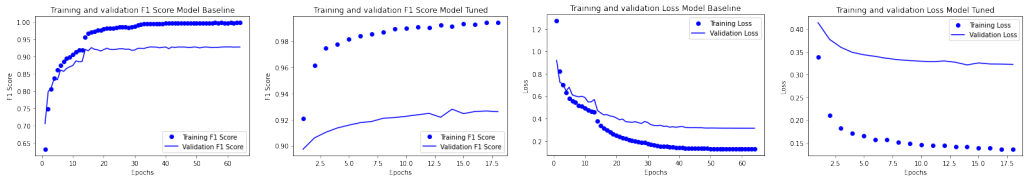

Figure 3.3: Training and validation macro F1-score and loss, with the dataset based on 5 second audio clips. The plots display the training and validation results for both pre-, and post-fine-tuning of the Xception layers

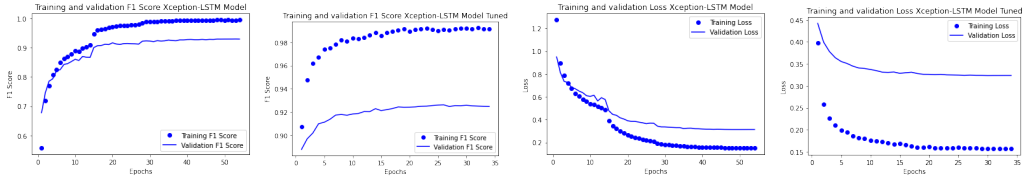

Figure 3.4: Training and validation macro F1-score and loss, with the dataset based on 3 second audio clips. The plots display the training and validation results for both pre-, and post-fine-tuning of the Xception layers.

Table 3.9 summarizes and compares the performance of the final model trained and evaluated with the 5 second, and 3 second audio clips based datasets respectively.

Table 3.9: Summary of the performance results from training the final model with the 3 seconds, and the 5 seconds, log-mel spectrogram images.

Final Model			
Dataset	Macro F1-score	Weighted F1-score	Loss
3 sec	0.9258	0.9487	0.3235
5 sec	0.9280	0.9490	0.3212

As can be seen in table 3.9 the performance was very similar for the two datasets. The dataset with log-mel spectrogram images based on 5 second audio clips performed slightly better, however, the difference was too small to decide if it was by chance or not.

Finally a confusion matrix was drawn, based on the final model's predictions using the 3 seconds mel-spectrogram images dataset, shown in figure 3.5. As shown by the matrix, the model's accuracy was highest for the bird species with the most records in the dataset, and lowest for the bird species with the least amount of records in the dataset.

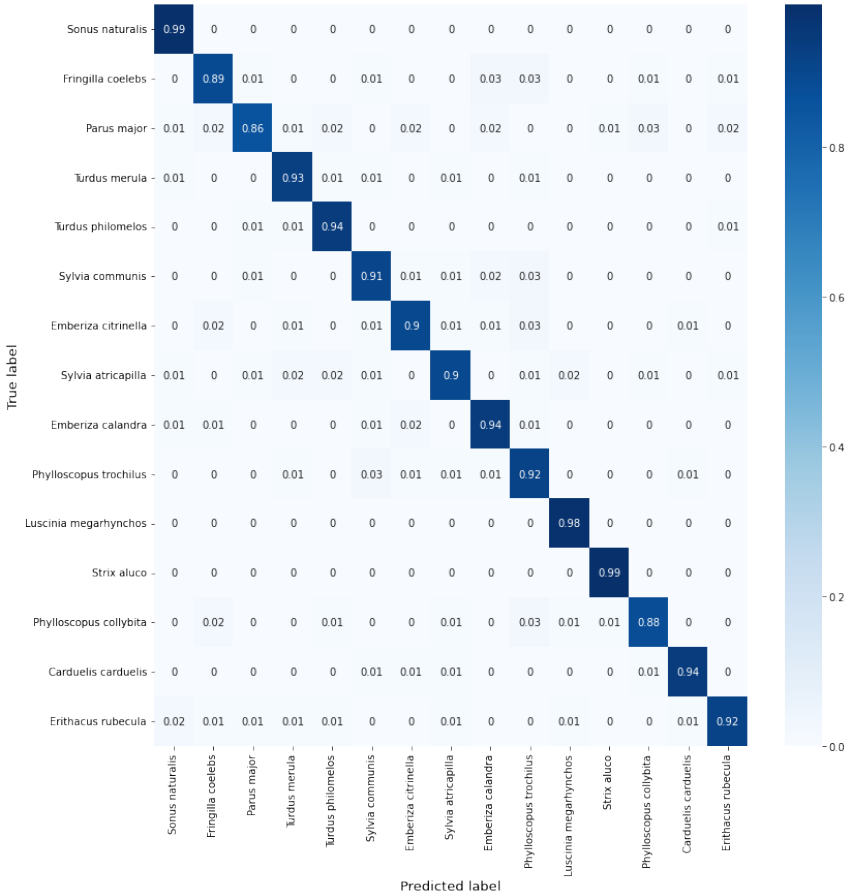

Figure 3.5: Confusion matrix showing the predicted label and the true label, for the final model, using the 3 second log-mel spectrogram dataset.

4 Discussion

One objective of this project was to explore what neural network seemed to be best suited for the task of birdsong classification. Similarly to the findings in the studies by Yan et al., and Liu et al., the results of the experiments in this project suggested that the use of LSTM layers can increase the model's performance significantly [4], [10]. Both Disabato et al. and Kumar et al. proposed transfer learning as an appropriate approach for birdsong classification, which was supported by the findings in this study as well [11], [12]. However, neither Disabato et al. nor Kumar et al. combined the transfer learning model with LSTM layers, which was the approach that gave the best performance in this project [11], [12].

As for the hyperparameters, it seemed to be a fine balance between trying to achieve a model complex enough to correctly predict the bird species, and at the same time trying to reduce the model's overfitting. Hence, in this study, using relatively few layers of the Xception model, which was used for transfer learning, connected to only two LSTM layers and one dense output layer provided the best results. Even though regularization in the form of dropout, and L2 regularization were implemented, overfitting couldn't be avoided.

The project's final model was trained and tested on two versions of the dataset. One version with log-mel spectrograms based on 5 second audio clips, and another with log-mel spectrograms based on 3 second audio clips. However, no significant difference in the model's performance was found, which could indicate that the duration of the audio clips doesn't affect the models performance by much. However, as only 3, and 5 second audio clips were explored, further explorations would have to be made to draw any conclusions.

To summarize, the findings of this study indicates that combining CNN and LSTM networks achieves higher performance than the use of only CNNs. Based on the results, connecting a transfer learning model with LSTM layers yields the best performing model.

As the dataset was quite limited, the model was found to easily overfit and the challenge was to make the model complex enough to be able to accurately predict the labels, and at the same time simple enough to decrease the overfitting as much as possible. Thus the most interesting future work might be to explore how to effectively increase the quantity or the quality of the dataset. For example generate audio clips with overlap, explore different ways to augment the data, or fuse different features to reduce the information loss, thus increasing the quality of the data. It could also be interesting to explore the use of waveforms as input data to a LSTM network.

References

[1] Martin. The sustainable development agenda. [Online]. Available: https://www.un.org/sustainabledevelopment/development-agenda/

[2] The IUCN red list of threatened species. [Online]. Available: https://www.iucnredlist.org/en

[3] A. Digby, M. Towsey, B. D. Bell, and P. D. Teal, "A practical comparison of manual and autonomous methods for acoustic monitoring," vol. 4, no. 7, pp. 675–683. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/2041-210X.12060

[4] N. Yan, A. Chen, G. Zhou, Z. Zhang, X. Liu, J. Wang, Z. Liu, and W. Chen, "Birdsong classification based on multi-feature fusion," vol. 80, no. 30, pp. 36 529–36 547. [Online]. Available: https://link.springer.com/10.1007/s11042-021-11396-9

[5] P. H. Wrege, E. D. Rowland, S. Keen, and Y. Shiu, "Acoustic monitoring for conservation in tropical forests: examples from forest elephants," vol. 8, no. 10, pp. 1292–1301. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/2041-210X.12730

[6] Y. Harjoseputro, I. Yuda, K. P. Danukusumo *et al.*, "Mobilenets: Efficient convolutional neural network for identification of protected birds," *IJASEIT (International Journal on Advanced Science, Engineering and Information Technology)*, vol. 10, no. 6, pp. 2290–2296, 2020.

[7] J. Xie, K. Hu, M. Zhu, J. Yu, and Q. Zhu, "Investigation of different cnn-based models for improved bird sound classification," *IEEE Access*, vol. 7, pp. 175 353–175 361, 2019.

[8] A. Kumar and S. D. Das, "Bird species classification using transfer learning with multistage training," in *Workshop on Computer Vision Applications*. Springer, 2018, pp. 28–38.

[9] L. Nanni, G. Maguolo, S. Brahnam, and M. Paci, "An ensemble of convolutional neural networks for audio classification," *Applied Sciences*, vol. 11, no. 13, p. 5796, 2021.

[10] H. Liu, C. Liu, T. Zhao, and Y. Liu, "Bird song classification based on improved bi-lstm-densenet network," in *2021 4th International Conference on Robotics, Control and Automation Engineering (RCAE)*. IEEE, 2021, pp. 152–155.

[11] S. Disabato, G. Canonaco, P. G. Flikkema, M. Roveri, and C. Alippi, "Birdsong detection at the edge with deep learning," in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2021, pp. 9–16.

[12] Y. Kumar, S. Gupta, and W. Singh, "A novel deep transfer learning models for recognition of birds sounds in different environment," *Soft Computing*, pp. 1–21, 2022.

[13] J. Liu, Y. Zhang, D. Lv, J. Lu, S. Xie, J. Zi, Y. Yin, and H. Xu, "Birdsong classification based on ensemble multi-scale convolutional neural network," *Scientific Reports*, vol. 12, no. 1, pp. 1–14, 2022.

[14] ImageCLEF - the CLEF cross language image retrieval track | ImageCLEF / LifeCLEF - multimedia retrieval in CLEF. [Online]. Available: https://www.imageclef.org/

[15] DCASE. [Online]. Available: https://dcase.community/

[16] D. Stowell and M. D. Plumbley, "Automatic large-scale classification of bird sounds is strongly improved by unsupervised feature learning," *PeerJ*, vol. 2, p. e488, 2014.

[17] Bird songs from europe (xeno-canto). [Online]. Available: https://www.kaggle.com/monogenea/birdsongs-from-europe

[18] W.-P. Vellinga, "Xeno-canto - bird sounds from around the world," type: dataset. [Online]. Available: https://www.gbif.org/dataset/b1047888-ae52-4179-9dd5-5448ea342a24

[19] H. Purwins, B. Li, T. Virtanen, J. Schlüter, S.-Y. Chang, and T. Sainath, "Deep learning for audio signal processing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 206–219, 2019.

[20] L. Nanni, G. Maguolo, and M. Paci, "Data augmentation approaches for improving animal audio classification," *Ecological Informatics*, vol. 57, p. 101084, 2020.

[21] D. de Benito-Gorron, A. Lozano-Diez, D. T. Toledano, and J. Gonzalez-Rodriguez, "Exploring convolutional, recurrent, and hybrid deep neural networks for speech and music detection in a large audio dataset," vol. 2019, no. 1, p. 9. [Online]. Available: https://asmp-eurasipjournals.springeropen.com/articles/10.1186/s13636-019-0152-1

[22] librosa librosa 0.9.1 documentation. [Online]. Available: https://librosa.org/doc/main/index.html

[23] H. Sura. Audio classification. [Online]. Available: https://medium.com/@hasithsura/audio-classification-d37a82d6715

[24] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *CoRR*, vol. abs/1610.02357, 2016. [Online]. Available: http://arxiv.org/abs/1610.02357

[25] K. Team. Keras documentation: Xception. [Online]. Available: https://keras.io/api/applications/xception/

[26] A. Géron, "Softmax regression," in *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, second edition ed., ser. Covid-19 collection. O'Reilly, pp. 148–151.

[27] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets," vol. 10, no. 3, p. e0118432. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0118432

[28] A. Géron, "Precision and recall," in *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, second edition ed. O'Reilly Media, Inc, pp. 92–96.

[29] tfa.metrics.f1score | TensorFlow addons. [Online]. Available: https://www.tensorflow.org/addons/api_docs/python/tfa/metrics/F1Score