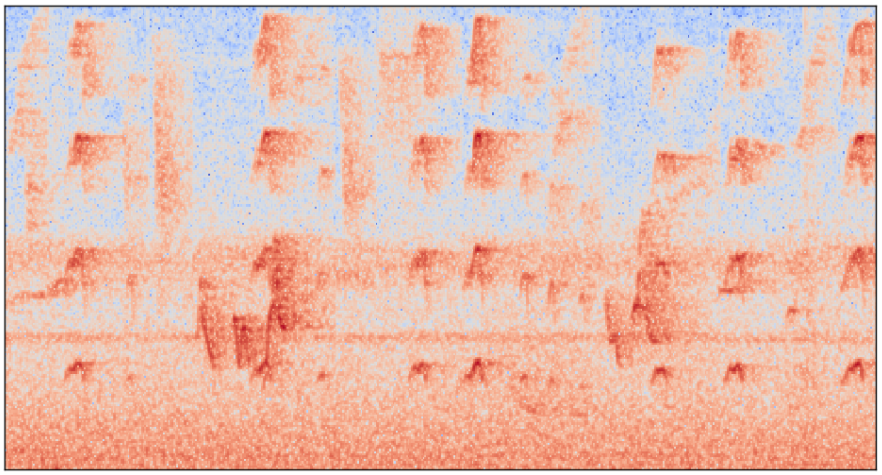


Bird Species Identification using Convolutional Neural Networks

Master's thesis in Computer Science - Algorithms, Languages and Logic

JOHN MARTINSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

Bird Species Identification using Convolutional Neural Networks

JOHN MARTINSSON

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Bird Species Identification using Convolutional Neural Networks
JOHN MARTINSSON

Supervisor: Alexander Schliep, Department of Computer Science and Engineering
Examiner: Richard Johansson, Department of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of a time-spectral representation of a bird song recording

Typeset in LaTeX
Gothenburg, Sweden 2017

Bird Species Identification using Convolutional Neural Networks
JOHN MARTINSSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

An area of interest in ecology is monitoring animal populations to better understand their behavior, biodiversity, and population dynamics. Acoustically active animals can be automatically classified by their sounds, and a particularly useful ecological indicator is the bird, as it responds quickly to changes in its environment.
The aim of this study is to improve upon the state-of-the-art bird species classifier [1], which is implemented and used as a baseline. The questions asked are: Can deep residual neural networks learn to classify bird species based on bird song and how well to they perform? Do multiple-width frequency-delta data augmentation or meta-data fusion further increase the accuracy of the model?
The questions are answered by training a deep residual neural network on one of the largest bird song data sets in the world, with and without the use of multiple-width frequency-delta data augmentation and meta-data fusion, and by comparing the results with the baseline.
The study shows that deep residual neural networks can learn to classify bird species based on bird song and that the mean average precision of the classifier nearly matches the state-of-the-art. We further develop a proof of concept for meta-data fusion which indicates that fusion of elevation data can be used to increase the accuracy of the model, and in particular decrease its coverage error.
Possible ways forward are to tune the hyper parameters of the deep residual neural network, fuse time of recording and geological location data into the model, or to move towards the more realistic, but less studied, open set problem of continuous classification rather than the N-class problem which is studied in this thesis.

Keywords: Machine Learning, Convolutional Neural Networks, Bioacoustical Monitoring, Bird Species Classification

Acknowledgements

After six months of thesis work, I'm finally able to say that I'm finished. The experience has been testing at times, but also interesting and fun. Taking a project from idea to proposal, to plan, and, finally, to reality has been a rewarding process. I have received great help throughout the project, and would like to thank the people involved. To my supervisor, Alexander Schliep, thank you for taking on my thesis, and for your insights, guidance and the positive atmosphere you brought to the project. Elias Sprengel, thank you for guiding me during the development of the baseline, and for answering my questions. Alexis Joy and Hervé Goeau, thank you for kindly agreeing to evaluate my submission runs for the hidden test set of the BirdCLEF 2016 competition. Richard Johansson, thank you for taking on the examination of the project, and for a fruitful half-time discussion.

John Martinsson, Gothenburg, April 2017

Contents

List of Figures **xi**

List of Tables **xiii**

List of Abbreviations **xv**

1 Introduction **1**
1.1 Problem Formulation . . . 1
1.1.1 Signal Classification . . . 2
1.1.2 Feature Extraction . . . 3
1.2 Goals and Questions . . . 3
1.3 Outline . . . 4

2 Previous Work **5**
2.1 Bird Classification Challenges . . . 5
2.1.1 MLSP 2013 . . . 5
2.1.2 NIPS4B 2013 . . . 6
2.1.3 BirdCLEF 2016 . . . 6
2.1.4 Summary . . . 7
2.2 Signal Processing . . . 7
2.2.1 Spectrogram . . . 7
2.2.2 Mel Frequency Cepstral Coefficients . . . 8
2.3 Bird Species Classification . . . 8
2.3.1 Preprocessing . . . 8
2.3.2 Data Augmentation . . . 9
2.3.3 Convolutional Neural Network . . . 11

3 Methods **17**
3.1 Multiple-Width Frequency-Delta Data Augmentation . . . 17
3.2 Deep Residual Neural Network . . . 18
3.2.1 Residual Unit . . . 18
3.2.2 Architecture . . . 20
3.3 Meta-Data Fusion . . . 20
3.3.1 Elevation . . . 22
3.4 Data set . . . 23
3.5 Evaluation . . . 24
3.5.1 Mean Average Precision . . . 25

3.5.2 Area Under the ROC Curve . . . 26
3.5.3 Top-n Accuracy . . . 26
3.5.4 Coverage Error . . . 26

4 Results **29**
4.1 Multiple-Width Frequency-Delta Data Augmentation . . . 29
4.2 Deep Residual Neural Networks . . . 30
4.3 Meta-Data Fusion . . . 32
4.4 Data Analysis . . . 33
4.5 Optimization Methods . . . 34

5 Discussion **39**

6 Conclusions **43**

A Implementation Details and Usage **I**
A.1 Data set . . . I
A.2 Preprocessing . . . II
A.3 Training . . . II
A.3.1 Configuration File . . . III
A.4 Run Predictions . . . IV
A.5 Evaluation . . . IV
A.6 Hardware . . . IV

List of Figures

2.1 *The figure shows how the binary image changes after each step (from top to bottom): (i) the spectrogram of the signal used as reference, (ii) the binary image after median clipping of the amplitude spectrogram, (iii) the binary image after performing a binary erosion, and (iv) the binary image after performing a binary dilation. The mask is then derived from the final binary image by checking which columns contain at least one non-zero value.* 10

3.1 *A simplified residual unit which consists of two stacked convolutional layers activated using ReLUs. The input of the first convolutional layer is additively merged with the output of the last convolutional layer, which is what we refer to as a shortcut.* 19

3.2 *This figure shows the architecture of the original residual [2] (left), and the improved residual, unit using "full pre-activation" [3] (right). Note that the activation is no longer done after the residual unit, but rather inside it, hence "pre-activation". This means that the function f in Equation 3.4, which is applied to the data flowing through the shortcut, becomes an identity function.* 20

4.1 *The training history for the baseline method. The figure shows the change in training and validation loss (top image), and the change in training and validation accuracy (bottom image) with respect to the training epoch.* 30

4.2 *The training history for the 18-layer residual neural network. The figure shows the change in training and validation loss (top image), and the change in training and validation accuracy (bottom image) with respect to the training epoch.* 31

4.3 *The accuracy of the model for the sound classes ranked by accuracy. The plot shows that the accuracy of the model varies a lot between different sound classes. It is 100% for the highest ranked and 0% for the lowest ranked classes.* 33

4.4 *Confusion matrices for the residual neural network (a) and (b), and the baseline (c) and (d) when each has been trained and evaluated twice on the BCResnetBot100, and BCCubeRunBot100 respectively. A perfect accuracy would result in a clean diagonal line. The rows are the ground truth labels, and the columns are the classes predicted by the model. Some sound classes seem to pick up more predictions than others, meaning that the network seems to favor prediction of some sound classes over others.* 35
4.5 *The figure shows the number of training segments (blue) plotted on the right y-axis with respect to 5% chunks of the sound classes ranked by the number of training segments in each 5% chunk. It also shows the average number of predictions that the chunks of sound classes receive (red) and the expected average number of predictions for each chunk (green) plotted on the left y-axis.* 36
4.6 *Validation loss (top) and accuracy (bot) for seven different optimizers when training the baseline model on the BCSubset data set.* 37

List of Tables

2.1 *The network architecture used in the baseline. The first column contains the type of the layer, the second column contains the configuration of the layer, and the third column contains the output shape of the layer (rows, columns, channels).* 14

3.1 *The architecture of the 18-layer deep residual neural. The configuration of a basic block, e.g., "64 3x3 kernels, 2x2 stride" should be read as: the number of filters of the convolutional layers in the basic block is 64, their kernel sizes are 3x3, the stride size of the first convolutional layer is 2x2, whereas the stride size of the second convolutional layer is 1x1.* 21

3.2 *The architecture of a basic block. The input layer is simply the output of the layer to which the basic block has been connected. Meaning that if the previous layer has an output of shape (n, m, d) then the input layer gets that output shape. Both convolutional layers use the number of filters which is specified when constructed (see Table 3.1), but only the first convolutional layer uses the specified stride size, the second always has a stride size of 1x1.* 21

3.3 *The names of the data sets used in this thesis, how many sound classes each data set contains, and a short description of each data set. BCWhole is the entire BirdCLEF 2016 data set, BCSubset consists of 20 randomly chosen sound classes from BCWhole, BCCubeRunBot100 consists of the 100 sound classes for which the baseline had the worst accuracy, and BCResnetBot100 consists of the 100 sound classes for which the residual neural network had the worst accuracy; when trained on BCWhole.* 24

4.1 *The MAP, AUROC, top-1 accuracy, top-5 accuracy, and the CE of the baseline when trained with the raw spectrogram, MFCCs, and MWFD features. Each of these models was trained on BCSubset.* . . . 29

4.2 *The MAP, AUROC, top-1 accuracy, top-5 accuracy, and the CE of the baseline, and the residual neural network. Each method has been trained on the BCWhole data set three times, and then the average of the evaluation scores and their standard deviation has been computed.* 31

4.3 *The MAP score for the baseline, the residual neural network, and the previous state-of-the-art when evaluated on the hidden BirdCLEF test set with and without consideration of the background species.* 32

4.4 *The MAP, AUROC, top-1 accuracy, top-5 accuracy, and CE of the baseline, and the residual neural network when meta-data fusion of the elevation is used. Each method has been trained on the BCWhole data set three times, and then the average evaluation and the standard deviation of these have been taken.* 32
4.5 *The MAP, AUROC, top-1 accuracy, top-5 accuracy, and the CE of the baseline, and the residual neural network when trained on dataset BCCubeRunBot100 and BCResnetBot100 respectively.* 34

A.1 *The main software libraries used during development, their respective version numbers, and their most useful methods for this project.* . . . I

List of Abbreviations

AUROC Area Under the Receiver Operating Characteristic Curve. 23, 24

BN Batch Normalization. 17

CE Coverage Error. 23
CNN Convolutional Neural Network. 1, 3

FFT Fast Fourier Transform. 8

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 3, 37

MAP Mean Average Precision. 7, 23, 29
MFCC Mep Frequency Cepstral Coefficients. xiii, 3, 8, 15, 16, 27, 37
MLSP Machine Learning for Signal Processing. 1, 5–7
MWFD Multiple-Width Frequency-Delta. xiii, 3, 4, 27, 37, 38

NIPS4B Neural Information Processing Scaled for Bioacoustics. 1, 6, 7

ReLU Rectified Linear Unit. 12, 16

SIBL Single-Instance Binary-Label. 2
SIML Single-Instance Multi-Label. 2
SISL Single-Instance Single-Label. 2
STFT Short-Time Fourier Transform. 7, 8

1
Introduction

An important problem in ecology, which is the study of interactions between organisms and their environment, is to monitor animal populations, especially with the continuing threat of climate change [4].

The use of acoustics to monitor and classify animals in their natural environments has received a lot of interest lately [5–13]. Classification of animal species based on recorded sound data is, for example, useful when monitoring breeding behavior, biodiversity, and population dynamics [7, 14]. Birds are a particularly useful ecological indicator, as they respond quickly to changes in their environment.

Bird classification can be done manually by domain experts; however, with growing amounts of data, this rapidly becomes a tedious and time-consuming process. Therefore automatic tools which can aid in this process are needed. Several bird species identification challenges such as the BirdCLEF [15], the Neural Information Processing Scaled for Bioacoustics (NIPS4B) 2013, and the Machine Learning for Signal Processing (MLSP) 2013 Bird Classification Challenge [8] have been held recently, all with the goal of creating and evaluating such automatic classifiers on bird song recordings taken from the field. A promising classification technique has proven to be convolutional neural networks [1, 8]. In the 9th annual MLSP competition the authors conclude with the words "convolutional neural nets achieved excellent results without extensive feature engineering, hence further investigation of such methods is warranted" [8], and the winners of BirdCLEF 2016 used a convolutional neural network (CNN) which was trained on augmented spectrogram data computed from bird song audio files [1]. In this thesis, a new convolutional neural network architecture called deep residual neural networks [2] is evaluated in this problem domain to see if this model, originally designed for image recognition, is useful in the audio domain. A new data augmentation technique called multiple-width frequency-delta data augmentation is investigated [16], and a proof of concept for a meta-data fusion method is developed. (The full source code for the project can be found at: `https://github.com/johnmartinsson/bird-species-classification`.)

1.1 Problem Formulation

The challenges addressed in this thesis could be divided into two main parts: (i) signal classification, and (ii) feature extraction. In this section, these challenges are described in more detail and each challenge is formulated as a problem which the thesis aims to address.

1.1.1 Signal Classification

There are several ways of defining a bird species classification problem. First of all, we need to establish what we want to classify: Are we interested in the absence or presence of a bird in the recording? Are we interested in how many individuals there are? Do we need to classify the actual species of the birds present? Or is it the start and end timings of each individual song that we seek?

Considering only the presence or absence of a bird in a recording makes it a *single-instance binary-label* (SIBL) problem. That is, for each audio recording there are two possible classes, either a bird is present (singing) in the recording, or it is not. This is useful in, e.g, a system which collects data out in the field. If the system is able to recognize when a bird is present in a data sample, then it can make an informed decision about whether or not to store that sample to disk.

If we are interested in the actual species of the singing bird the number of possible classes increases to the number of observable bird species making it a *single-instance single-label* (SISL) problem. There is still only one instance to classify, namely the recording, but there are a lot of different bird species around the world, which means that the sound classes are no longer binary. This is useful if we want to reason about the data collected out in the field, and answer questions such as: What bird species are present? How diverse is the bird population? Et cetera.

If we are interested in all the singing species in a recording we need the ability to classify multiple bird species per recording, making it a *single-instance multi-label* (SIML) problem. Which is an accurate formulation of the problem at hand, but it is also a hard problem to solve.

In this thesis we consider only the most prominently singing bird species in each recording, i.e., we treat it as a single-instance single-label problem. However, the data used will contain recordings where multiple birds are singing at the same time which means that in these instances the background species will be treated as noise.

Formally, let $X = \{\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_n\}$ be the set of bird song recordings, or training samples where each $\bar{\mathbf{x}}_i \in X$ is associated with a main species $y_j \in Y$. Let $Y = \{y_1, \ldots, y_n\}$ be the set of all species labels, and let $f : X \to Y$ be a function which maps a training sample to the ground truth main species. Then the SISL classification problem is to find a parameter set w such that the parameterized function $f_w : X \to Y$ is a good estimate of f. How good the estimate f_w is can be determined by introducing a loss function where $loss(f(\bar{\mathbf{x}}), f_w(\bar{\mathbf{x}}))$ is defined to be small if the estimate $f_w(\bar{\mathbf{x}})$ is close to the ground truth $f(\bar{\mathbf{x}})$, and larger otherwise. That is, the loss tends towards zero if the prediction is close to the ground truth. The optimization problem can then be defined as:

$$w = \arg\min_{w} \sum_{\bar{\mathbf{x}} \in X} loss(f(\bar{\mathbf{x}}), f_w(\bar{\mathbf{x}})). \tag{1.1}$$

That is, we want to find the parameter set w such that the total loss for the model f_w when evaluated on the training data X is as small as possible. However, keep in mind that the optimization is based on the loss of the *training data*. This does not guarantee that the model performs well on data which it has not seen before. If it does we say that the model *generalizes* well. In order to estimate how well the model generalizes the data is usually split into training data and test data,

and then optimized, or trained, on the former, but only evaluated on the latter.

In this thesis, f_w will be modeled using a convolutional neural network (CNN) as in the winning solution of the BirdCLEF 2016 bird classification challenge [1]. However, in addition to a conventional CNN, we propose the use of *residual learning* [2] with *identity mappings* [3] as part of the CNN architecture. These modifications to the CNN architecture have shown promising results and was part of the winning solution of the ILSVRC 2015 classification task. The modifications should enable the training of much deeper networks, and deeper networks are commonly associated with better classification results.

1.1.2 Feature Extraction

Another problem when training neural networks is deciding what features to use as input to the network. In particular, how to extract abstract features, which contain as much information about the original input as possible, but with a lower dimensionality to enable efficient training.

The design of a feature extraction method can be seen as a trade-off between *separability* and *contraction* [17]. We want a *contractor*, or feature extractor, $\Phi(x)$ which reduces the dimensions of data point x without sacrificing separability, that is, we want $\Phi(x) \neq \Phi(x')$ if $f(x) \neq f(x')$, where f is a true classifier. If this property holds we say that Φ separates f.

Sprengel et. al [1] use a spectral computation as the contractor, and input noise-filtered chunks of spectrograms to the CNN. These chunks are augmented in four ways: time shifting, pitch shifting, noise adding, and combining same class audio files. This thesis proposes an alternative feature representation where Mel-frequency cepstral coefficients (MFCCs) are augmented with *multiple-width frequency-delta* data augmentation (MWFD) [16] to see if it can improve classification accuracy in the bird song domain.

1.2 Goals and Questions

The goal of this thesis is to improve upon the state-of-the-art bird species classifier [1] which was used to win the BirdCLEF 2016 bird recognition challenge. We propose the use of a new convolutional neural network architecture [2] and a recently published technique for augmenting features extracted from audio signals [16]. We also investigate the use of meta-data, in addition to the audio data, as a way of increasing classification accuracy. The classifier should be able to identify the most prominently singing bird in an acoustical soundscape. An acoustical soundscape is the composition of geophony (e.g., wind, trees, and rain), biophony (e.g., birds, frogs, and insects) and anthropophony (e.g., airplanes, cars, and trains).

With this goal in mind there are three main research questions which this thesis aims to answer: (i) Can deep residual neural networks be used to classify bird species based on acoustical data recordings, and how well do they perform? (ii) Can multiple-width frequency-delta data augmentation be used to improve classification accuracy in this problem domain? And (iii) can additional meta-data of the recordings be used to improve classification accuracy?

1.3 Outline

In this chapter, we have explained to the reader why the problem is important, what it can be used for, and have given the reader an introduction and formulation of the problem at hand. The outline of the rest of the thesis is:

- In Chapter 2 we review the results produced during several of the most recent bird species identification challenges, and present the theory of a state-of-the-art bird species classifier which will be used as the baseline in this thesis.
- In Chapter 3 we present the theory for the multiple-width frequency-delta data augmentation technique, the deep residual neural network, the meta-data fusion and the evaluation methods used to produce the results in the thesis.
- In Chapter 4 the results of the baseline, the MWFD data augmentation, the residual neural network, and the data-fusion method are presented and compared. We also show results from an analysis of the used data set.
- In Chapter 5 we summarize and discuss what we set out to do, what has been achieved, what problems arose, and propose possible routes for future work.
- Lastly, in Chapter 6 we draw conclusions from the results and the discussion.

2
Previous Work

In this chapter previous work on automated species classification using bioacoustics is reviewed with a strong focus on the literature where acoustical classification methods have been used to identify singing birds in audio recordings.

2.1 Bird Classification Challenges

Several bird species classification challenges with closely related, but different, task descriptions have been held during the last few years. The interest and participation in these challenges have been high which indicates that these are relevant problems and that there is a need to solve them.

The challenges are usually to predict which species are present in a set of recordings with hidden labels, called the test set, and to submit the predicted species for each test data point for evaluation against the ground truth labels. The task description can vary from predicting only the presence or absence of birds in a recording to predicting all actively singing bird species [8]. That is, the challenges have a varying degree of difficulty. In the rest of this section, we present the results of some of the most recent such challenges in chronological order.

2.1.1 MLSP 2013

The IEEE International Workshop on Machine Learning for Signal Processing (MLSP) announced a bird species classification challenge in the year of 2013 [8]. The challenge was to determine all of the acoustically active bird species in each audio recording of a test set with a total of 19 different bird species. That is, the task was treated as a single-instance multi-label problem.

The data set consisted of 645 ten-second audio recordings which were split into a training set (50%) and a test set (50%). The bird species labels for each recording in the training set was made public, but the labels for each recording in the test set was kept secret.

79 teams participated in the challenge, and 8 out of 10 of the top-ranking teams supplied a two-page summary of their methods [8], showing a strong interest in the problem. The winning team used a random forest (RF) classifier, where features were extracted from the input using template matching.

Templates were computed using a custom time-frequency segmentation technique where each boxed segment was stored as a template [18] and computed only from the 81 audio recordings which were labeled with a single sound class. The spectrogram

of each recording was then computed. Features were extracted for each spectrogram by computing the normalized cross-correlation map between the spectrogram and each template, the similarity between the template and the spectrogram were then evaluated at the maximum value of the normalized cross-correlation map using a template matching method, meaning that each spectrogram yields a feature vector with the same length as the number of templates that was extracted from the 81 signal label recordings. The method also used features available as a baseline in the challenge such as histogram of segments, which were appended to the feature vector and used as input to the classifier.

Many of the teams designed task specific features; however, one team which came in fourth place used raw spectrogram data to train a convolutional neural network. It is stated by Briggs et. al [8] that further investigation of the use of convolutional neural networks in this problem domain is warranted.

2.1.2 NIPS4B 2013

In the Neural Information Processing Scaled for Bioacoustics (NIPS4B) Bird Species Classification Challenge the task description was similar to that of MLSP 2013. Participants were asked to identify all actively singing birds in each of the test files. However, the number of possible species was 87 instead of 19, and the recordings could vary in length (from around 0.5s to 5.5s). This is also formulated as a single-instance multi-label classification problem.

The winning solution [19] of this challenge used a similar approach to [18]. The main difference is that additional features are extracted for each audio file. In addition to the features extracted by evaluating the template matching at the maximum value of the normalized cross-correlation map, the feature vector is further augmented with file and segment statistics (e.g. mean, standard deviation etc).

2.1.3 BirdCLEF 2016

The BirdCLEF challenge uses a very large data set with bird recordings. The data is taken from the bird song database xeno-canto [1], which is a web-based service where bird enthusiasts can upload and share recordings of bird songs. At the time of writing (2016-12-21, 19:12:33) there are 335,268 recordings of 9,684 different foreground bird species, and 10,256 background species, recorded by 3,390 recordists, which totals to over 4,850 hours of recorded sound.

The BirdCLEF data set is a subset of the xeno-canto database and consists of 999 different species recorded in South America (Brazil, French Guiana, Surinam, Guyana, Venezuela and Colombia) [20]. The data set totals to about 33,200 recordings, which have been normalized to 44.1 kHz 16-bit mono format (right channel) audio files.

The task is to determine the bird species present in each recording. Participants were asked to provide a ranked list with the most probable bird species for the recordings of a hidden test set. The data set is divided into 1/3 test data and 2/3 training data, and the metric used to evaluate the classification performance of the

[1] http://www.xeno-canto.org

runs supplied by the participating teams is the mean average precision (MAP) over all recordings in the test set.

The best performing method [1] used a convolutional neural network, where the input to the network was segments of the spectrogram computed from the sound files. The sound files are preprocessed by extracting two sound classes from each audio file: noise and signal (bird vocals), which are divided into equally long sound segments of around 3 seconds. The signal segments represent the actual bird vocals which each have a bird species associated with it. The samples shown to the neural network are then loaded and augmented at random. Each signal segment is additively combined with another same class signal segment chosen at random [21], as well as three randomly chosen noise segments. The samples are then further augmented by a random shift in the time domain, and a small random shift (5%) in the frequency domain.

2.1.4 Summary

In MLSP 2013 the winning solution were random forests trained on probabilities derived from template matching of species-specific spectrograms [8]. The winning solution of NIPS4B 2013 by Lasseck et al. [19] used these results as a starting point but introduced an additional set of features statistically derived from the audio files. Lasseck also used a similar method to win the BirdCLEF 2015 challenge [22]. However, during the BirdCLEF 2016 challenge, it was shown that convolutional neural networks trained on spectral data computed from the sound recordings could outperform other state-of-the-art systems [1]. This thesis uses the work of Sprengel et al. [1] as a starting point and a baseline, and explore the use of a new convolutional neural network method called deep residual neural networks [2] as well as a new data augmentation technique called multiple-width frequency delta data augmentation [16].

2.2 Signal Processing

In this section two commonly used signal processing techniques will be explained. Raw audio data is not suited as input for a neural network, and therefore the audio signal is usually transformed into a time-spectral representation.

2.2.1 Spectrogram

The spectrogram of a discrete audio signal $\bar{\mathbf{x}} = x_1, \ldots, x_n$ is computed in two or three steps. First, a Short-Time Fourier Transform (STFT) is applied to the audio signal. The STFT is computed in a standard way by splitting the signal into different overlapping frames, and then compute the Discrete Time Fourier Transform (DTFT) for each frame, which results in a matrix with complex values (see Equation 2.1):

$$STFT\{\bar{\mathbf{x}}\}(m,\omega) \equiv X_m(\omega) = \sum_{n=-\infty}^{\infty} x_n w(n - mR) e^{-j\omega n} \tag{2.1}$$

where x_n is the input signal at time n, $w(n)$ is a length $M = 512$ Hann window centered around n, and $R = 128$ is the hop size between successive frames. That is, we use a Hann window of size 512 with a 75% overlap. We use the *librosa.stft* method of the library *librosa* [23] to compute the STFT.

Secondly, the squared amplitude of the magnitude of the STFT is computed, which we call an *ampspectrogram* (see Equation 2.2), and thirdly the natural logarithm of the amplitude spectrogram is computed (see Equation 2.3), which we refer to as a *logspectrogram*.

$$ampspectrogram\{\bar{\mathbf{x}}\}(\omega) \equiv |\bar{\mathbf{X}}(\omega)|^2 \tag{2.2}$$

$$logspectrogram\{\bar{\mathbf{x}}\}(\omega) \equiv log_e(|\bar{\mathbf{X}}(\omega)|^2) \tag{2.3}$$

2.2.2 Mel Frequency Cepstral Coefficients

Mel Frequency Cepstral Coefficients (MFCCs) have been a standard choice as the audio features used in speech recognition, and their success owes much to the amount of information they contain about the audio signal in such a compressed form [24]. The Mel Frequency Cepstrum approximates the way the human auditory system deals with sound. The following steps are used to compute the MFCCs:

(i) compute the power magnitude spectrogram of the signal,
(ii) combine the FFT bins into Mel-frequency bins,
(iii) take the logs of these powers, and
(iv) perform a Discrete Cosine Transform.

The MFCCs are now the amplitudes of the resulting spectrum. In this thesis, the MFCC are computed using the *librosa.feature.mfcc* method of the library *librosa* [23].

2.3 Bird Species Classification

In this section, the state-of-the-art method used to win the BirdCLEF 2016 challenge [1] is explained in more detail. This method will be used as a baseline.

2.3.1 Preprocessing

The audio files are first preprocessed into a format that can be used to train the neural network. The audio files are normalized to mono-channel 16-bit wave data re-sampled from 44,100 Hz to 22,050 Hz.

Bird Song Detection

The bird song recordings are separated into two different sound classes: signal (bird vocals) and noise. The separation lets us train the neural network on the most relevant data, and it gives us access to a noise class which can be used to augment training samples. After the recording has been separated into a signal wave and

a noise wave these are each split into 3-second segments which are stored to disk. The noise segments can later be used to augment the training samples shown to the network which should improve generalization (see Section 2.3.2).

The signal part is extracted by first computing a signal mask $\bar{\mathbf{v}}$ for the given sound wave $\bar{\mathbf{x}}$, and then use the mask to extract the relevant part of the sound wave, where $v_i = 0$ indicates that x_i is not part of the signal, and $v_i = 1$ indicates that it is part of the signal.

The mask is derived from a binary image which is computed by analyzing the normalized amplitude spectrogram of $\bar{\mathbf{x}}$. Let $\widetilde{\mathbf{b}}$ be the binary image, and let $\widetilde{\mathbf{s}}$ be the normalized amplitude spectrogram of $\bar{\mathbf{x}}$, where $\widetilde{\mathbf{b}}$ and $\widetilde{\mathbf{s}}$ have the same dimensions. The pixel at index (i, j) in the binary image is then set to one if $s_j^{(i)}$ is t times larger than the row and column median of $\widetilde{\mathbf{s}}$ at row i and column j as seen in Equation 2.4.

$$b_j^{(i)} = \begin{cases} 1, & \text{if } s_j^{(i)} > t \times median(\bar{\mathbf{s}}^{(i)}) \wedge s_j^{(i)} > t \times median(\bar{\mathbf{s}}_j) \\ 0, & otherwise \end{cases} \tag{2.4}$$

The binary image is further processed by applying a binary erosion followed by a binary dilation on the image, both using a kernel size of 4 by 4, which smooths out the regions marked as bird vocals (see Figure 2.1), and the signal mask $\bar{\mathbf{v}}$ is derived from the binary image by setting v_j to one if the column $\bar{\mathbf{b}}_j$ contains one one. The mask is also smoothed by performing two more binary dilutions (kernel size 4), and is then re-scaled such that $|\bar{\mathbf{v}}| = |\bar{\mathbf{x}}|$. The images in Figure 2.1 have been manually compared, for each step, with the corresponding image presented by Sprengel et. al [1] for the same sound file, and the method produces similar results.

Algorithm 2.1 Noise/Signal Mask Computation

```
1: procedure ComputeMask(r̄, t)
2:     Pxx ← spectrogram(r̄)
3:     Pxx ← normalize(Pxx)
4:     BinaryImage ← medianClipping(Pxx, t)
5:     BinaryImage ← erosion(BinaryImage, (4, 4))
6:     BinaryImage ← dilation(BinaryImage, (4, 4))
7:     mask ← computeMask(BinaryImage)
       return mask
```

The signal mask is computed by setting the threshold $t = 3$, and the noise mask is computed by setting $t = 2.5$ and then inverting the mask at the end (flipping 0s to 1s, and 1s to 0s). This may leave parts of the wave which are marked as neither signal nor noise (2.5-3). These parts are considered to not contribute with any relevant information for the network, and they are simply ignored.

2.3.2 Data Augmentation

Data augmentation is a way of increasing the number of training samples in a data set by augmenting the training data. Even if the BirdCLEF dataset is one of the largest bird song data sets available, the number of training samples per bird species is rather small; on average around 30 samples per sound class. Data augmentation

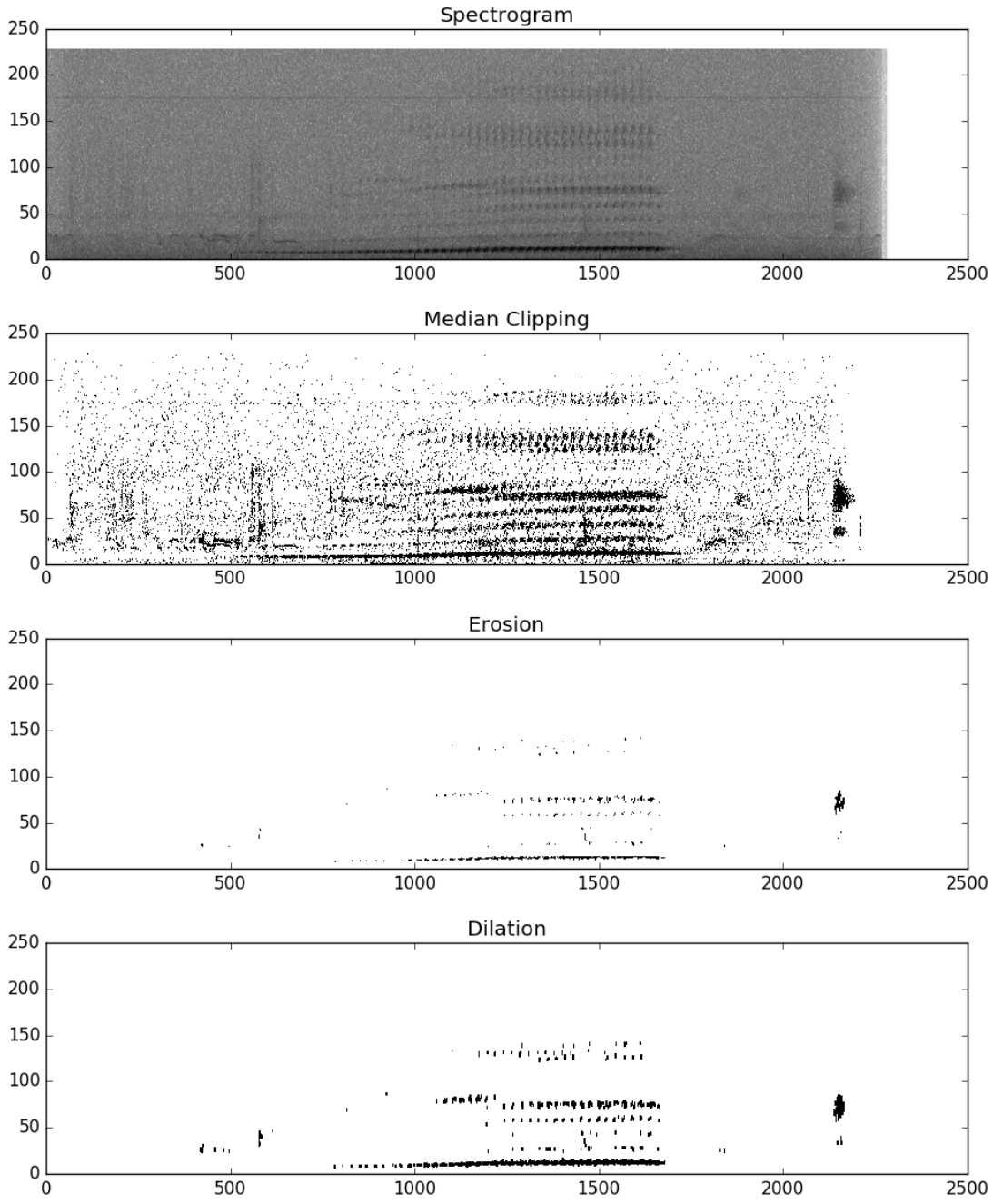


Figure 2.1: *The figure shows how the binary image changes after each step (from top to bottom): (i) the spectrogram of the signal used as reference, (ii) the binary image after median clipping of the amplitude spectrogram, (iii) the binary image after performing a binary erosion, and (iv) the binary image after performing a binary dilation. The mask is then derived from the final binary image by checking which columns contain at least one non-zero value.*

can also be used to make the training samples harder by introducing noise to the signal, which is intended to prevent overfitting and make the model generalize better due to a higher noise invariance. In this section, the data augmentation techniques used in the baseline [1] will be explained.

Same Class and Noise Addition

In order to improve the convergence rate of the neural network, the training samples are augmented by additively combining each sample with another same class sample, which lets the network see more relevant data at once. The samples are also additively combined with three random noise segments, to make the network more noise invariant, and therefore generalize better. Each sample shown to the neural network is thus a combination of two randomly chosen same class signal segments $\bar{\mathbf{x}}_1$ and $\bar{\mathbf{x}}_2$, and three randomly chosen noise segments $\bar{\mathbf{n}}_1$, $\bar{\mathbf{n}}_2$, and $\bar{\mathbf{n}}_3$ of the same length:

$$\bar{\mathbf{s}}_{aug} = \alpha\bar{\mathbf{x}}_1 + (1-\alpha)\bar{\mathbf{x}}_2 + \beta(\bar{\mathbf{n}}_1 + \bar{\mathbf{n}}_2 + \bar{\mathbf{n}}_3), \tag{2.5}$$

where $\alpha \in [0, 1)$ is chosen uniformly at random, $\beta = 0.4$ is used as a dampening factor of the noise segments. All segments are three seconds long and sampled at 22,050 Hz (see Section 2.3.1).

Time and Pitch Shift

The time shift augmentation is done by splitting a spectrogram sample into two parts, along the time axis, and then place the second part before the first. That is, a wrap-around shift in the time domain. The pitch shift augmentation is done in a similar way, but in the frequency domain (vertically), and the shift is only around 5%. Larger shifts are said to not be beneficial [1]. The shifts are performed on the logarithmic spectrogram of the same class and noise augmented training samples, just before they are shown to the neural network.

2.3.3 Convolutional Neural Network

The classifier presented by Sprengel et. al [1] is a convolutional neural network (CNN). A convolutional neural network is a neural network architecture pioneered by LeCun et. al [25] and later popularized by Krizhevsky et. al [26] with the introduction of the AlexNet. A typical CNN consists of convolutional layers and pooling layers followed by a fully connected neural network. The convolutional layers and the pooling layers are supposed to learn how to extract relevant, locally distortion invariant, features from the input [25], and the fully connected neural network is supposed to learn how to classify these features. The features learned by the convolutional layers could be, e.g., edges of objects in an image.

In the rest of this section, convolutional layers and pooling layers will be defined, the input of the network will be summarized, the architecture used in the baseline will be defined, and the initialization, optimization and loss function used will be explained.

Convolutional Layer

A convolutional layer takes an image as input, and produces a set of feature maps as output. The input image can contain multiple channels (e.g. RGB), which means that the convolutional layer learns a mapping from a 3D volume to another 3D volume. The layer consists of a number of convolution kernels each of which is made up of adjustable weights. The weights are adjusted during optimization using stochastic gradient decent. The feature maps are produced by performing a discrete convolution between each kernel in the convolutional layer and the input volume yielding one feature map for each kernel.

Performing a convolution can be thought of as sliding a window along the input image and computing the dot product between the kernel and the values of the neighborhood patch, or the so called receptive field, in the image as seen through the window. The window has the same size as the kernel, and it is important to understand that the kernel has a depth extending through all the channels of the input image.

Formally, let $\mathbf{w}_k^l$ and b_k^l denote the weights and the bias term for the l-th layer of the k-th convolution kernel, and let $\mathbf{x}^l \in \mathbb{R}^{WH}$ be the l-th channel, or layer, of an input image of shape (W, H, D). We can then define the feature value, $c_{i,j}^k$, at position (i, j) of the resulting k-th feature map as:

$$c_{i,j}^k = f(\sum_{l=1}^{D} \mathbf{w}_k^l \cdot \mathbf{x}_{i,j}^l + b_k^l), \tag{2.6}$$

where D is the depth of the input (the number of channels), $\mathbf{x}_{i,j}^l$ is the receptive field centered around (i, j) of the l-th layer, and f is a rectified linear unit (ReLU) defined as:

$$f(x) = max(0, x). \tag{2.7}$$

Note that the resulting number of feature maps is the same as the number of kernels used, and that the weights and bias are reused in the computation of the dot product for each receptive field, for each kernel and layer, called weight sharing. The idea is that if the kernel is useful for finding local features at position (x_1, y_1) in the image, then it should also be useful at a different position (x_2, y_2).

The configuration of the convolutional layer is thus specified by the number of kernels used, the width and height of the kernels, and the stride size which determines the spacing between the receptive fields. For example, a stride size of 1×2 would slide the window along the image with a step size of one pixel horizontally and two pixels vertically making the width of the resulting feature map the same as that of the input image, but the height of the feature map half the height of the input image.

The output size of the convolutional layer is therefore determined by the number of kernels used and the stride size. The number of kernels determines the depth of the output volume and the stride size determines the width and height of the output volume. For example, if the input volume of a convolutional layer is an image of shape $(32, 32, 3)$, the number of kernels used is 16, and the stride size is 1×2, then the output volume would be of shape $(32, 16, 16)$.

Zero padding of the input image has been assumed in this section. Since the

window will extend outside the bounds of the input at the edges some values will not be defined, and zero padding is basically assuming that all these values are zero.

Pooling Layer

The feature maps computed by a convolutional layer are then run through a pooling layer, which is designed to merge semantically similar features and thus reduce the size of the feature maps [27].

The pooling layer used is max pooling which simply computes the maximum value of local receptive fields in each of the feature maps. The spacing of the fields is determined by the stride size and the size of the fields is determined by the kernel size of the pooling layer. Again, let $\mathbf{x}^l_{i,j}$ be the receptive field of the l-th input layer centered around (i, j), then the value of the l-th pooled layer, $p^l_{i,j}$, can be defined as:

$$p^l_{i,j} = max(\mathbf{x}^l_{i,j}), \tag{2.8}$$

where max takes the maximum value of the receptive field.

For example, if the stride size is 2×2 then the width and the height of the pooled feature maps are halved because we step through the 2×2 neighborhoods of each feature map with a step size of 2 pixels horizontally, and 2 pixels vertically. That is, if the input of a max pooling layer with a kernel of size 2×2 and a stride size of 2×2 has shape $(32, 32, 4)$, then the output will have shape $(16, 16, 4)$. The input is assumed to be zero padded to handle edge cases.

Input of the CNN

To summarize: the final input of the convolutional neural network is the logarithmic spectrogram of the augmented data points. The augmented data points are computed from the original sound files by applying the following steps:

(i) normalize sound files,
(ii) separate noise and signal segments,
(iii) combine same class and noise segments to form a unique data point,
(iv) compute the logarithmic spectrogram of the audio segment,
(v) drop the 4 lowest and 24 highest frequency bands, and
(vi) time-shift and pitch-shift the spectrogram.

The resulting shifted logarithmic spectrogram is then re-sized to 256×512 pixels to have even powers of two, yielding an input vector of shape $(256, 512, 1)$, which is used as input to the convolutional neural network.

Architecture

The architecture of the convolutional neural network model described by Sprengel et. al [1] uses a normal input layer, then five similar building blocks with different configurations, where each building block consists of a batch normalization [28]-, convolution-, and max pooling layer. Finally, the network has a dense layer and a softmax layer with a 40% dropout [29] on each. The dropout is used to introduce

Table 2.1: *The network architecture used in the baseline. The first column contains the type of the layer, the second column contains the configuration of the layer, and the third column contains the output shape of the layer (rows, columns, channels).*

Layer (type)	Configuration	Output Shape
InputLayer		(256, 512, 1)
BatchNormalization		(256, 512, 1)
Convolution2D	64 5x5 kernels, 1x2 stride	(256, 256, 64)
MaxPooling2D	2x2 kernel, 2x2 stride	(128, 128, 64)
BatchNormalization		(128, 128, 64)
Convolution2D	64 5x5 kernels, 1x1 stride	(128, 128, 64)
MaxPooling2D	2x2 kernel, 2x2 stride	(64, 64, 64)
BatchNormalization		(64, 64, 64)
Convolution2D	128 5x5 kernels, 1x1 stride	(64, 64, 128)
MaxPooling2D	2x2 kernel, 2x2 stride	(32, 32, 128)
BatchNormalization		(32, 32, 128)
Convolution2D	256 5x5 kernels, 1x1 stride	(32, 32, 256)
MaxPooling2D	2x2 kernel, 2x2 stride	(16, 16, 256)
BatchNormalization		(16, 16, 256)
Convolution2D	256 3x3 kernels, 1x1 stride	(16, 16, 256)
MaxPooling2D	2x2 kernel, 2x2 stride	(8, 8, 256)
BatchNormalization		(8, 8, 256)
Flatten		(16384)
Dropout	dropout 0.4	(16384)
Dense		(1024)
Dropout	dropout 0.4	(1024)
Dense		(999)
Total Params	19,523,883	

noise to the model and prevent overfitting. A dropout of 40% means that for each epoch there is a 40% chance of a neuron being deactivated. The whole network architecture is summarized in Table 2.1. All the convolutional layers use rectified linear units (ReLUs) as activation functions. The two dense layers at the end use ReLU and softmax activation respectively.

Initialization

The weights of deep neural networks can be initialized by pooling from a standard distribution with zero mean and a small fixed standard deviation. However, if the weights in the network start too small, then the signal which is propagated through the layers of the network may shrink too fast to be useful, or, conversely, if the weights start too large, the signal may grow too quickly as it passes through the network making it to large to be useful. Therefore it can be difficult to get very deep networks to converge. Another reason for convergence issues can be that the gradients become unstable with bad initialization, resulting in vanishing or exploding gradients [30]. If the gradients become small the convergence rate towards the

optimum also becomes small, and the training time for the network increases. That is, the initial weights of the network affect how the signal is propagated forward in the network, and how the gradients are propagated backward in the network, and, ideally, both of these should be stable.

Glorot and Bengio [30] suggests what they call a normalized initialization:

$$W_j^{(i)} \sim U\Big[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j} + n_{j+1}}\Big], \tag{2.9}$$

where $U[-a, a]$ is the uniform distribution in the interval $(-a, a)$, and n_i is the size of the layer i.

However, this initialization method assumes that the activation functions are linear, which is not true for ReLUs. In this thesis, the initialization method used is the one proposed by He et. al [31], which takes ReLUs into account.

Optimization

The optimization method used is stochastic gradient descent with a learning rate, η, of 0.001, a Nesterov momentum, μ, of 0.9, and a decay rate of 1e-6. In Section 1.1.1 we explained that the optimization problem can be defined as:

$$w = \arg\min_w \sum_{\bar{\mathbf{x}} \in X} loss(f(\bar{\mathbf{x}}), f_w(\bar{\mathbf{x}})), \tag{2.10}$$

where w is the parameters that minimize the loss over the training set X for the parameterized function f_w. Gradient descent is used to update the parameters in the direction which decrease the loss the most:

$$w_{t+1} = w_t - \eta \Delta Q(w_t) = w_t - \eta \sum_{i=1}^{N} \Delta Q_i(w_t), \tag{2.11}$$

where $Q_i(w) = loss(f(\bar{\mathbf{x}}_i), f_w(\bar{\mathbf{x}}_i))$ is the loss of the model for training sample $\bar{\mathbf{x}}_i \in X$. Computing the full gradient at each update, however, is costly. Therefore stochastic gradient descent is used which is performed "on-line" and approximates the real gradient using only one sample, or a subset of samples. For simplicity, only one sample is used in this explanation. The update rule for stochastic gradient decent can then be defined as:

$$w_{t+1} = w_t - \eta \Delta Q_i(w_t), \tag{2.12}$$

where we compute the gradient of the loss function at training sample i, and update the parameters in the negative direction of this gradient scaled by the learning rate η.

The update rule using Nesterov momentum can be defined as:

$$v_{t+1} = \mu v_t - \eta \Delta Q_i(w_t + \mu v_t) \tag{2.13}$$

$$w_{t+1} = w_t + v_{t+1}, \tag{2.14}$$

where v_t represents the momentum which accumulates if the direction of the gradient stay the same through multiple updates, and $\mu \in [0, 1]$ is the Nesterov momentum coefficient [32]. The update rule is applied once for each training sample during an epoch of training, and the training samples are shuffled.

Loss Function

The *loss function* is a measure of how well the network model performs on a set of labeled data points. The most commonly used loss functions are mean squared error and cross entropy both of which are continuous, differentiable, and can be computed efficiently, which is needed during the optimizaion to compute the gradient.

In this thesis we use the cross entropy loss function as it has three other nice properties: (i) it is always positive, (ii) the loss tends to zero when the estimate of the network tends to the desired output, and (iii) the amount of loss, which is determined by the partial derivatives of the loss function, is in proportion to how wrong the estimate of the network is. Especially (ii) and (iii) are properties which one intuitively would expect from a learning framework. We expect the loss to become smaller the more correct we are, and we expect to learn more from greater mistakes.

The single-label categorical cross entropy function can be defined as:

$$C = -\frac{1}{N}\sum_{n=1}^{N}\Big[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\Big], \tag{2.15}$$

where N is the total number of training samples, $y_n = f(\bar{\mathbf{x}}_n)$ is the desired output for data point $\bar{\mathbf{x}}_n$, and $\hat{y}_n = f_w(\bar{\mathbf{x}}_n)$ is the estimated output.

3
Methods

This chapter explains the methods used to implement and evaluate the bird species classifiers in this thesis. The following steps stay the same as in the baseline presented in Section 2.3: preprocessing (see Section 2.3.1), same class and noise augmentation (see Section 2.3.2), time shift and pitch shift augmentation (see Section 2.3.2), optimization (see Section 2.3.3), and loss function (see Section 2.3.3).

The methods explained in this chapter have to the best of the author's knowledge not been used for acoustical bird species classification before. Data augmentation techniques, as well as convolutional neural network, have previously proven to be useful in this problem domain [1,8], therefore it is of interest to further explore such methods.

3.1 Multiple-Width Frequency-Delta Data Augmentation

Mel-frequency cepstral coefficients (MFCCs) are commonly used features when training audio classifiers (see Section 2.2.2). Han et. al [16] presents a novel data augmentation technique which is used to improve the usefulness of such features. The idea is to compute the *delta features* of the MFCCs, which contain additional information about the trajectories of the MFCCs. This gives the network not only local static information about the signal, but also dynamic information about how the signal will proceed in the (near) future [16]. The multiple-width frequency-delta (MWFD) data augmentation is computed as such:

$$d_t = \frac{\sum_{k=1}^{K} k(x_{t+k} - x_{t-k})}{2\sum_{k=1}^{K} k^2}, \tag{3.1}$$

where d_t is the delta feature computed from frame t in terms of the static Mel-frequency cepstrum coefficients x_{t-K} to x_{t+K}. The delta width refers to $2K+1$, and Han et. al [16] propose widths $\Delta 3, \Delta 11$ and $\Delta 19$ (i.e., $K = 1$, $K = 5$, $K = 9$). The input to the CNN is then the vector, or four channel image, $(static, \Delta 3, \Delta 11, \Delta 19)$, where *static* refers to the MFCCs. Consequently, if $|static| = 12$ then $|\Delta 3| = |\Delta 11| = |\Delta 19| = 12$, meaning that we would augment a 12 feature input vector to a 48 feature input vector. Edge cases are solved by reusing the first or last coefficient in *static* respectively.

We use the first 32 MFCCs following the results of [33], which show that 32 MFCCs are a sweet spot for identifying frog calls, and insects. These are extracted using the library *librosa* [23], which computes a Mel-spectrogram using a 2048 length

FFT window and a hop size of 512. The audio segments processed are three seconds each with a sample rate of 22,050 Hz which yields an MFCC vector of shape $(32, 129, 1)$, which in turn is resized to $(32, 128, 1)$ in order to have even powers of two. The deltas are then computed from the resized MFCCs, using *librosa.feature.delta*, and used to augment the MFCCs as described in this section. The final shape of the feature vector is thus $(32, 128, 4)$ which is used as input to the classifier.

3.2 Deep Residual Neural Network

Deep residual neural networks were poineered by He et. al [2], and use "shortcut connections" in order to improve the convergence rate and classification accuracy of very deep CNNs. Deep networks are usually harder to train [30] because of vanishing or exploding gradients, which has mostly been mitigated by using normalized weight initialization [30], ReLU activation [34] and intermediate normalization layers [28]. The reason behind the shortcuts is that by letting the signal flow more easily through the network the problem of exploding or vanishing gradients can be reduced, which in turn makes the deep network easier to train.

A residual network is made of smaller building blocks called residual units, which are basically stacked convolutional and normalizing layers where the input of the unit is additively merged with the output of the last stacked layer to simulate a shortcut.

3.2.1 Residual Unit

Consider two stacked convolutional layers in a CNN, and assume that they learn the mapping $\mathcal{H}(x)$, where x is the input to the first layer, and $\mathcal{H}(x)$ is the output of the second layer. The hypothesis is that it is easier to learn the residual function $\mathcal{F}(x) = \mathcal{H}(x) - x$ than the mapping $\mathcal{H}(x)$ itself [2]. The mapping $\mathcal{H}(x)$ then becomes $\mathcal{F}(x) + x$.

The reason for this is based on the assumption that the mapping $\mathcal{H}(x)$, to be learned by the stacked layers, is usually closer to an identity mapping than a zero mapping [2], meaning that it is easier to learn the part of the mapping which is the difference between the identity mapping and the optimal mapping than the optimal mapping itself. In the extreme case, if the optimal mapping is the identity mapping, then it is easier to push $\mathcal{F}(x)$ towards zero than to learn the actual mapping.

This is implemented in the network by a *shortcut*, where the input, x, of a stacked layer is connected to the output of the stacked layer, using a simple additive unit. This building block is what we call the *residual unit* (see Figure 3.1 for a simplified overview of a residual unit).

In the simplest case, where the dimensions of the output of the residual function and the input are the same, the residual unit can be defined as:

$$y_l = x_l + \mathcal{F}(x_l, W_l) \tag{3.2}$$

$$x_{l+1} = \sigma(y_l), \tag{3.3}$$

where x_l and x_{l+1} is the input and output of the l-th residual unit, $\mathcal{F}$ is the parameterized, or learned, residual function, W_l are the parameters of the stacked layers,

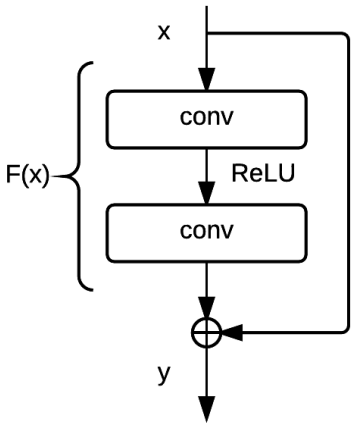


Figure 3.1: *A simplified residual unit which consists of two stacked convolutional layers activated using ReLUs. The input of the first convolutional layer is additively merged with the output of the last convolutional layer, which is what we refer to as a shortcut.*

and σ is the ReLU [34] activation function.

More generally, the residual units can be defined as:

$$y_l = h(x_l) + \mathcal{F}(x_l, W_l), \tag{3.4}$$

$$x_{l+1} = f(y_l), \tag{3.5}$$

where x_l and x_{l+1} are the input and output of the l-th residual unit, f is the activation function, $\mathcal{F}$ is the learned residual function, h is the mapping applied on the data flowing through the shortcut, in Equation 3.2 above it would be an identity mapping, and W_l is the parameters associated with the l-th residual unit. If the dimensions of $\mathcal{F}(x_l, W_l)$ and x_l are not the same then h would need to adjust the dimension of x_l by, e.g., subsampling.

The original residual unit is improved by what is called "full pre-activation" [3], which is a change in the order of the stacked layers of the unit. The realization was that if f is the identity function then Equation 3.5 can be inserted into Equation 3.4 which gives the network three nice properties: (i) the same residual property as seen in each unit also becomes present between any two units in the network, (ii) the computations become more efficient, and (iii) the backward propagating gradients are very unlikely to vanish even with arbitrarily small weights.

The change in the order of the original residual unit [2] compared to the improved [3] can be seen in Figure 3.2, where the left image shows the original residual unit and the right image shows the improved residual unit. Here BN stands for *batch normalization*, ReLU stands for *rectified linear unit* [34], which is a type of activation function, and *weights* are those of the convolutional layers.

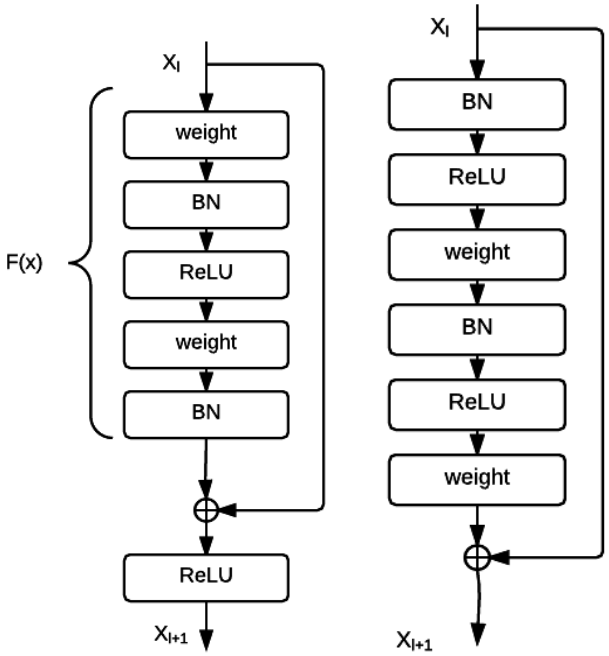


Figure 3.2: *This figure shows the architecture of the original residual [2] (left), and the improved residual, unit using "full pre-activation" [3] (right). Note that the activation is no longer done after the residual unit, but rather inside it, hence "pre-activation". This means that the function f in Equation 3.4, which is applied to the data flowing through the shortcut, becomes an identity function.*

3.2.2 Architecture

The architecture of an 18-layer deep residual neural network consists of an initial convolving and max pooling layer followed by eight basic blocks (with different configurations), and at the end the result is pooled, flattened and used as input to the final dense layer which has one neuron for each class label, activated by a softmax function. The layers are applied to the input in the same order as given above (see Table 3.1).

A basic block consists of two convolving layers, each preceded by batch normalization and ReLU activation, and the input of the block is allowed to flow directly to the output of the stacked layers via a shortcut realized using an additive merge layer (see Table 3.2).

3.3 Meta-Data Fusion

Most of the recordings in the data set have meta-data associated with them (see Section 3.4 for example), some of which could possibly be used to improve the classification accuracy of the learned model f_w.

Not all of the recordings in the data set have complete meta-data, which means that it would be hard to train a classifier using this data. However, it should be possible to take a Bayesian approach and combine, or fuse, this information with a

Table 3.1: *The architecture of the 18-layer deep residual neural. The configuration of a basic block, e.g., "64 3x3 kernels, 2x2 stride" should be read as: the number of filters of the convolutional layers in the basic block is 64, their kernel sizes are 3x3, the stride size of the first convolutional layer is 2x2, whereas the stride size of the second convolutional layer is 1x1.*

Layer (type)	Configuration	Output Shape
InputLayer		(256, 512, 1)
Convolution2D	64 7x7 kernels, 2x2 stride	(128, 256, 64)
MaxPooling2D	3x3 kernel, 2x2 stride	(64, 128, 64)
BasicBlock	64 3x3 kernel, 1x1 stride	(64, 128, 64)
BasicBlock	64 3x3 kernel, 1x1 stride	(64, 128, 64)
BasicBlock	128 3x3 kernel, 2x2 stride	(32, 64, 128)
BasicBlock	128 3x3 kernel, 1x1 stride	(32, 64, 128)
BasicBlock	256 3x3 kernel, 2x2 stride	(16, 32, 256)
BasicBlock	256 3x3 kernel, 1x1 stride	(16, 32, 256)
BasicBlock	512 3x3 kernel, 2x2 stride	(8, 16, 512)
BasicBlock	512 3x3 kernel, 1x1 stride	(8, 16, 512)
AveragePooling2D	8x16 pool size, 1x1 stride	(1, 1, 512)
Flatten		(512)
Dense	He normal, softmax	(999)
Total Params	11,691,751	

Table 3.2: *The architecture of a basic block. The input layer is simply the output of the layer to which the basic block has been connected. Meaning that if the previous layer has an output of shape (n, m, d) then the input layer gets that output shape. Both convolutional layers use the number of filters which is specified when constructed (see Table 3.1), but only the first convolutional layer uses the specified stride size, the second always has a stride size of 1x1.*

Layer (type)	Configuration	Output Shape
InputLayer		(n, m, d)
BatchNormalization		(n, m, d)
Activation	ReLU	(n, m, d)
Convolution2D	c kernels, sxs stride	(n/s, m/s, c)
BatchNormalization		(n/s, m/s, c)
Activation	ReLU	(n/s, m/s, c)
Convolution2D	c kernels, 1x1 stride	(n/s, m/s, c)
Merge	[InputLayer, Convolution2D]	(n/s, m/s, c)

model which has been trained on only the audio data, hence meta-data fusion. As a proof of concept the elevation of a recording and the probability that the recording belongs to a specific bird species is combined using Bayes theorem to compute the posterior probability of a bird species given a recording and an elevation. Around 90% of the recordings have elevation data associated with them.

3.3.1 Elevation

We define the event of observing bird j as B_j, the event of observing a certain elevation e as E, and the event of observing a certain bird song s as S. Using Bayes theorem we can then express the posterior probability of observing bird j given the evidence e and s as:

$$Pr(B_j|E,S) = \frac{Pr(ESB_j)}{Pr(ES)} = \frac{Pr(B_j)Pr(S|B_j)Pr(E|SB_j)}{\sum_{i=1}^{n} Pr(B_i)Pr(S|B_i)Pr(E|SB_i)}, \tag{3.6}$$

and the posterior probability of observing B_j given the evidence S as:

$$Pr(B_j|S) = \frac{Pr(B_j)Pr(S|B_j)}{\sum_{i=1}^{n} Pr(B_i)Pr(S|B_i)}. \tag{3.7}$$

The normalizing factor in Equation 3.7 is constant and by denoting it as C we get:

$$Pr(B_j|S) \times C = Pr(B_j)Pr(S|B_j). \tag{3.8}$$

We now observe that the right-hand side of Equation 3.8 appears both in the numerator and the denominator of Equation 3.6 (although with different indices), and by inserting Equation 3.8 into Equation 3.6 we get:

$$Pr(B_j|E,S) = \frac{Pr(B_j|S)Pr(E|SB_j)}{\sum_{i=1}^{n} Pr(B_i|S)Pr(E|SB_i)}. \tag{3.9}$$

The posterior probability $Pr(B_j|S)$ can be estimated by the classifier, but the likelihood, $Pr(E|SB_j)$, of the evidence E given SB_j must be computed. In order to do this we assume that the events E and S are independent given B_j meaning that $Pr(E|SB_j) = Pr(E|B_j)$. Which allows us to rewrite Equation 3.9 as:

$$Pr(B_j|E,S) = \frac{Pr(B_j|S)Pr(E|B_j)}{\sum_{i=1}^{n} Pr(B_i|S)Pr(E|B_i)}. \tag{3.10}$$

Since the number of observed elevations for each bird species is rather small we need to approximate their distribution. The distribution is modeled by mixture density of a Gaussian and the uniform; tending towards the uniform if the number of observations is small, and towards the Gaussian if the number of observations is large. The reason for this is that we do not want zero probabilities for any observable elevation. The hypothesis is that this allows the model to generalize better.

Formally, let O_j be the set of observed elevations for bird j. The likelihood of observing e given the bird j is defined as:

$$h(e|j) \equiv (1 - \alpha(k_j))f(e|\mu_j, \sigma_j) + \alpha(k_j)(\frac{1}{e_{MAX}}), \tag{3.11}$$

where

$$f(e|\mu,\sigma) = \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.12}$$

is the probability density of a normal distribution, k_j is the cardinality of O_j, μ_j is the mean of O_j, σ_j is the standard deviation of O_j, e_{MAX} is the maximum observable altitude, and $\alpha(k)$ is a weight function defined as:

$$\alpha(k) = \begin{cases} 1, & if\ k < 10 \\ 1/k, & otherwise, \end{cases} \tag{3.13}$$

which means that if the number of elevation observations, k, is less than 10 then the mixture density is the uniform, and if $k > 10$ the mixture density tends towards the Gaussian.

We have assumed that birds can not be observed at elevations below sea level, or above e_{MAX} which have been set to 5000 meters above sea level.

The final meta-classifier using both the bird song and elevation as evidence to estimate the probability of it belonging to a bird j can now be defined as:

$$\mathcal{F}_w^{(j)}(s,e) = \frac{f_w^{(j)}(s)h(e|j)}{\sum_{i=1}^{N} f_w^{(i)}(s)h(e|i)}, \tag{3.14}$$

where $f_w^{(j)}(s)$ is the probability of bird j given the song s as predicted by the learned classifier, and $h(e|j)$ is the likelihood of observing e given the bird j.

3.4 Data set

The data used is the same as in the LifeCLEF 2016 bird identification task (BirdCLEF 2016) [20], which consists of about 33,200 recordings and is split into a publicly available training set of around 24,600 recordings and a private, or hidden, test set of around 8,600 recordings. There are 999 different bird species in this data set. The recordings are taken by bird song enthusiasts out in the field and can, therefore, contain other background bird species, be of varying lengths (seconds to half-hours), and contain varying amounts of noise. In some recordings there are humans speaking.

Each recording in the data set has an associated XML file with meta-data, for example:

```
<Audio>
  <MediaId>10000</MediaId>
  <FileName>LIFECLEF2014_RN10000.wav</FileName>
  <ClassId>kunsua</ClassId>
  <Date>2010-01-19</Date>
  <Time>14:00</Time>
  <Locality>RPPN Prima Luna, Nova Trento</Locality>
  <Latitude>-27.258</Latitude>
  <Longitude>-49.02</Longitude>
  <Elevation>500</Elevation>
```

Table 3.3: *The names of the data sets used in this thesis, how many sound classes each data set contains, and a short description of each data set. BCWhole is the entire BirdCLEF 2016 data set, BCSubset consists of 20 randomly chosen sound classes from BCWhole, BCCubeRunBot100 consists of the 100 sound classes for which the baseline had the worst accuracy, and BCResnetBot100 consists of the 100 sound classes for which the residual neural network had the worst accuracy; when trained on BCWhole.*

Dataset	Number of Classes	Description
BCWhole	999	The whole BirdCLEF data set
BCSubset	20	Subset of 20 classes from BCWhole
BCCubeRunBot100	100	100 hardest classes for the baseline
BCResnetBot100	100	100 hardest classes for the resnet

```
  <Author>Evair Legal</Author>
  <AuthorID>SMJXFKLATM</AuthorID>
  <Content>call, song</Content>
  <Comments />
  <Quality>1</Quality>
  <Year>BirdCLEF2014</Year>
  <BackgroundSpecies />
  <Order>Passeriformes</Order>
  <Family>Furnariidae</Family>
  <Genus>Sclerurus</Genus>
  <Species>scansor</Species>
  <Sub-species />
</Audio>,
```

which can be used, in addition to the sound data, when performing the classification.

The different data sets used during the development of the methods in this thesis are listed in Table 3.3. Each data sets is split into 90% training data and 10% validation data, as done by Sprengel et. al [1], to make the results comparable. The whole data set is called BCWhole (BirdCLEF Whole), but since the whole data set is quite large, and it takes a considerable amount of time to train a model on it, a subset of it was created with only 20 randomly selected bird species to be used during development called BCSubset.

We have also created two other subsets called BCCubeRunBot100 and BCResnetBot100, which consists of the data for the 100 sound classes for which the baseline, and the residual neural network performed worst - after having trained and evaluated them on BCWhole (see Table 3.3). These two data sets are used to get some insight on if some classes are harder than others, and why this may be the case.

3.5 Evaluation

The classifier is evaluated with respect to mean average precision (MAP) as used in the BirdClef2016 Challenge [20], area under the ROC curve (AUROC) as used

in the MLSP 2013 Challenge [8], top-1 accuracy, top-5 accuracy, and coverage error (CE).

We will distinguish between the accuracy scores during training of a classifier, and the final evaluation score for the classifier. During training the score will refer to the top-1 accuracy averaged over each individual training sample, however, the final evaluation scores for a model will be based on the averaged prediction probabilities of the segments which a recording has been split into. That is, we perform a prediction on each individual segment for a recording, and then average these predictions to get the final prediction for said recording. The same is done by Sprengel et. al [1] making results comparable.

3.5.1 Mean Average Precision

The mean average precision (MAP) is a measure of how relevant the predictions of a classifier are with respect to the ground truth labels. There is no penalty for predicting many classes, the important thing is that the ground truth labels are ranked high.

Formally, let $f_w : X \to Y$ be the learned model which predicts the probability of a data point $\bar{\mathbf{x}} \in X$ belonging to each of the possible sound classes, and let $f : X \to Y$ be a true classifier, and the let the set of ground truth labels $G = f(\bar{\mathbf{x}})$. Further, let $K = (k_1, \ldots, k_n)$ be the list of predicted labels ranked by the predicted probabilities $f_w(\bar{\mathbf{x}})$, where k_i is the predicted species label at rank i.

The mean average precision can then be defined as:

$$MAP = \frac{\sum_{\bar{\mathbf{x}} \in X} AP(\bar{\mathbf{x}})}{|X|}, \tag{3.15}$$

where $|X|$ is the number of test audio files, and $AP(\bar{\mathbf{x}})$ is the *average precision* for data point $\bar{\mathbf{x}}$ computed as:

$$AP(\bar{\mathbf{x}}) = \sum_{i=1}^{n} P(i) \times CiR(i), \tag{3.16}$$

where n is the number of predictions, i.e., the total number of sound classes, $P(i)$ is the precision at cut-off i in the ranked list of predictions for $\bar{\mathbf{x}}$, and $CiR(i)$ is the change-in-recall at cut-off i defined as:

$$CiR(i) = \begin{cases} 1/n, & \text{if } k_i \in G \\ 0, & \text{otherwise.} \end{cases} \tag{3.17}$$

This metric reflects that good predictions should come first. As an example, lets say that there are only two different sound classes, and that ground truth labels are $G = f(\bar{\mathbf{x}}) = \{2\}$, the learned classifier predicts the probabilities $f_w(\bar{\mathbf{x}}) = (0.7, 0.3)$, which means that $K = (1, 2)$. Then $P(1) = 0$ since the sound class at cut-off 1 does not exist in G, $P(2) = 1/2$ since we now have one correct prediction out of two in the ranked list at cut-off 2. The change-in-recall is $CiR(1) = 0$, and $CiR(2) = 1/2$. The average precision is thus $P(1)CiR(1) + P(2)CiR(2) = 0.25$. The mean average precision is then the averaged average precisions over all the data points in the test set.

3.5.2 Area Under the ROC Curve

The area under the receiver operating characteristic curve (AUROC) is a metric which measures the expectation that a positive data point drawn uniformly at random is ranked higher than a negative data point drawn uniformly at random. This metric is usually used in binary classification problems, but can be extended to single-label problems using a one versus all approach, where each class is considered separately.

Following the convention in this thesis, we let X denote a set of data points, and let f denote a true classifier. Considering a certain class y we let $\bar{\mathbf{x}}^+$ be a positive drawn uniformly at random from the set X^+, where $X^+ = \{\bar{\mathbf{x}} | \bar{\mathbf{x}} \in X, f(\bar{\mathbf{x}}) = y\}$, and let $\bar{\mathbf{x}}^-$ be a negative drawn uniformly at random from the set X^-, where $X^- = \{\bar{\mathbf{x}} | \bar{\mathbf{x}} \in X, f(\bar{\mathbf{x}}) \neq y\}$. The AUROC metric can then be defined as:

$$AUROC = Pr(score(\bar{\mathbf{x}}^+) > score(\bar{\mathbf{x}}^-)), \tag{3.18}$$

where $score(\bar{\mathbf{x}})$ represents the probability that $\bar{\mathbf{x}}$ belongs to class y. The score function is a result of the learned classifier f_w which predicts the probabilities that a data point $\bar{\mathbf{x}}$ belongs to each respective class.

A random classifier is expected to get an AUROC score of 0.5, and a true classifier (f in the example above) is expected to get an AUROC score of 1.0.

3.5.3 Top-n Accuracy

The top-n accuracy looks at the n highest ranked classes predicted by the model, and if one of these predictions are the ground truth class the prediction is considered accurate, otherwise not.

Let $X = \{\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_k\}$ be the set of training samples, let f be a true classifier, and let f_w be the learned model of that classifier. The top-n accuracy score is then computed by checking if the true class label $f(\bar{\mathbf{x}}_i)$ is in the top-n predictions from $f_w(\bar{\mathbf{x}}_i)$ for each $\bar{\mathbf{x}}_i \in X$, scoring one if it is and zero otherwise, and then averaging these scores.

Formally, let $top@n(\bar{\mathbf{x}}, f_w)$ be a function which creates a set of the n most probable class label predictions for the training segment $\bar{\mathbf{x}}$, given the learned classifier f_w. We then define the top-n accuracy score as:

$$\textit{top-n accuracy}(f_w) = \frac{1}{|X|} \sum_{\bar{\mathbf{x}}_i \in X} score(\bar{\mathbf{x}}_i, f_w), \tag{3.19}$$

where

$$score(\bar{\mathbf{x}}, f_w) = \begin{cases} 1, & \text{if } f(\bar{\mathbf{x}}) \in top@n(\bar{\mathbf{x}}, f_w) \\ 0, & otherwise. \end{cases} \tag{3.20}$$

3.5.4 Coverage Error

The coverage error is the average number of top-scored predictions which are needed to cover all of the ground truth labels. Since we consider a single-label problem in this thesis, we only need to cover the main species.

Formally, let $X = \{\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_n\}$ be the set of training samples, let $f : X \to Y$ be a true classifier, and let $f_w : X \to Y$ be the learned model of that classifier, then the coverage error can be defined as:

$$coverage\ error = \frac{1}{n} \sum_{i=1}^{n} rank_{f(\bar{\mathbf{x}}_i)}(f_w(\bar{\mathbf{x}}_i)) \tag{3.21}$$

where $rank_{f(\bar{\mathbf{x}})}(f_w(\bar{\mathbf{x}}_i))$ is the rank of the ground truth class $f(\bar{\mathbf{x}})$ in the probabilities predicted by $f_w(\bar{\mathbf{x}}_i)$.

4
Results

In this chapter, we present the results produced when training and evaluating the baseline and the methods proposed in this thesis. All results except those in Table 4.3 come from evaluation on the local validation set. The results in Table 4.3 come from evaluation on a hidden test set performed by the organizers of BirdCLEF. The chapter also contains results from a data analysis done on the data set, which is intended to give a better understanding of what is hard about the data set and what problems could arise from the data. The only method which actually improves upon the state-of-the-art baseline is meta-data fusion, and the reader is referred to Section 4.3 if only interested in the positive results.

4.1 Multiple-Width Frequency-Delta Data Augmentation

The MWFD data augmentation technique does improve upon plain MFCCs, meaning that the classification accuracy is better for a model trained on the augmented data than the MFCCs. However, it does not improve upon the raw spectrogram representation (see Table 4.1). The MWFD data representation is more compact than the raw spectral data, and it does perform nearly as well, meaning that it has an advantage when the computational resources are limited. The MWFD data augmentation was only tested while training on the smaller BCSubset data set. The reason being that training on the whole data set (BCWhole) takes a considerable amount of time, and since the method does not seem to improve upon the use raw spectral data for the smaller data set, it was not deemed necessary to test this method further.

Table 4.1: *The MAP, AUROC, top-1 accuracy, top-5 accuracy, and the CE of the baseline when trained with the raw spectrogram, MFCCs, and MWFD features. Each of these models was trained on BCSubset.*

Method	MAP	AUROC	Top-1	Top-5	CE
CubeRun (Spectrogram)	74.3%	92.2%	69.5%	91.5%	2.15
CubeRun (MFCCs)	69.2%	92.2%	64.4%	89.8%	2.7
CubeRun (MWFD)	72.6%	93.5%	67.8%	89.8%	2.2

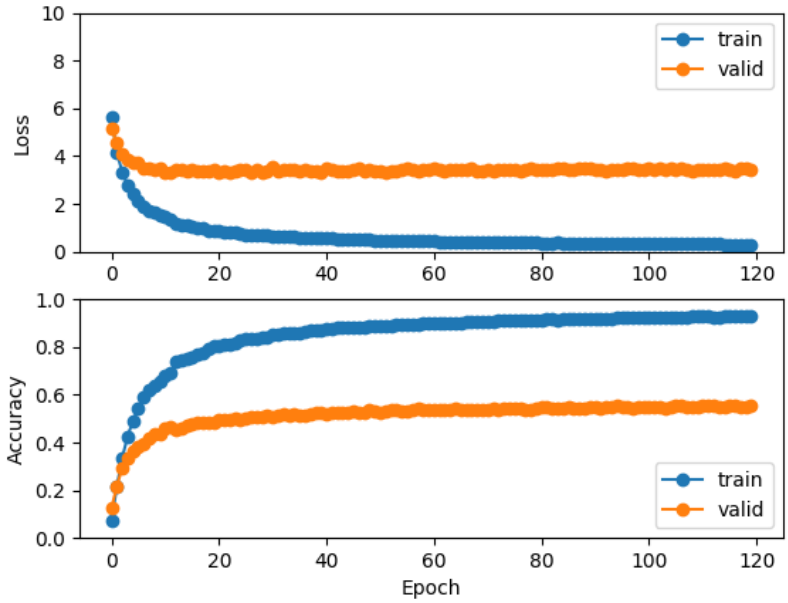


Figure 4.1: *The training history for the baseline method. The figure shows the change in training and validation loss (top image), and the change in training and validation accuracy (bottom image) with respect to the training epoch.*

4.2 Deep Residual Neural Networks

In this section, we present the results for the deep residual neural network. The residual network does converge and learns to classify bird species based on bird song. The classification accuracy is comparable to that of the baseline, but slightly worse, and the generalization of the model varies a lot between epochs.

The training history for the baseline and the deep residual network can be seen in Figure 4.1 and Figure 4.2 respectively. The figures show the loss for the models when evaluated on the training data (blue) and the validation data (orange) with respect to the training epoch. They also show the top-1 accuracy for the models when evaluated on the training data (blue) and the validation data (orange) with respect to the training epoch. The top-1 accuracy shown in the figures is computed with respect to each individual segment in the training and validation data, i.e., it is not the average of the segments in each recording. Each network was trained for 120 epochs, which resulted in 51 hours on the GPU for the baseline, and 70 hours on the GPU for the residual neural network (see Appendix A.6 for hardware specification).

In Table 4.2 we present the MAP, AUROC, top-1, top-5, and CE scores for the baseline model and the residual network model when trained on the BCWhole data set (containing all classes and recordings). The scores of the residual network are comparable with the scores of the baseline, however, it is not an improvement. The scores in Table 4.2 are based on the averaged prediction over all segments in a recording, while the scores in the plots are the accuracy of each individual segment. Therefore the top-1 score in the table may be higher than that shown

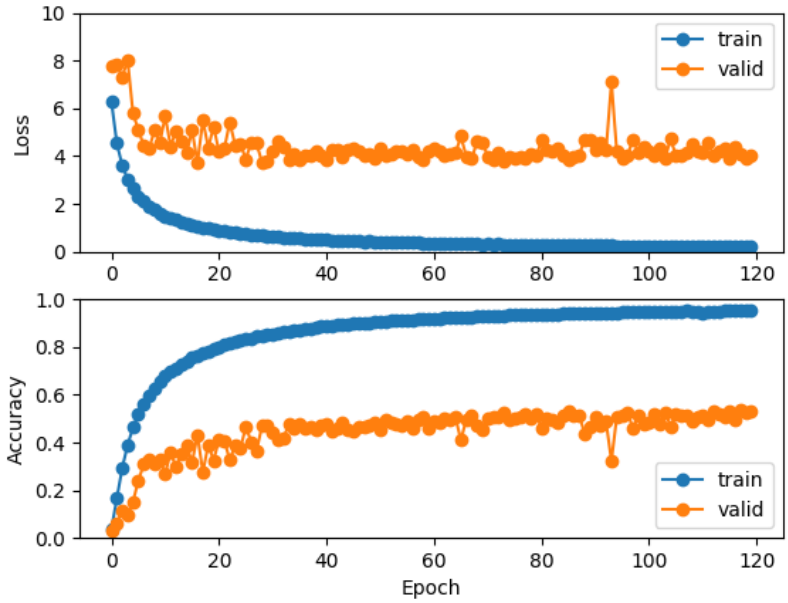


Figure 4.2: *The training history for the 18-layer residual neural network. The figure shows the change in training and validation loss (top image), and the change in training and validation accuracy (bottom image) with respect to the training epoch.*

in the plots. As an example, let us say we have a recording $\bar{\mathbf{r}}_1$ of sound class l_1. The recording has been preprocessed and split into three segments $\{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \bar{\mathbf{x}}_3\}$ and we get $\{f_w(\bar{\mathbf{x}}_i)|i \in \{1,2,3\}\} = \{l_1, l_1, l_2\}$ for the learned classifier f_w. Then the top-1 accuracy over each individual segment would be $66.\bar{6}\%$ while the mean top-1 accuracy would be 100%. Note that this example is an oversimplification since the predictions are not binary, but rather pseudo probabilities.

In order to see how well the methods generalize to a larger test set, consisting of previously unseen data, two submissions were created for the BirdCLEF 2016 challenge and sent to the organizers for evaluation. The competition evaluates each run using the MAP metric. The scores for the baseline and the residual neural network can be seen in Table 4.3 as well as the previous state-of-the-art score for this data set. The table shows the MAP score for each run with and without consideration of the background species, and what is interesting is that the MAP scores for both methods are actually higher than when evaluated on the local validation set.

Table 4.2: *The MAP, AUROC, top-1 accuracy, top-5 accuracy, and the CE of the baseline, and the residual neural network. Each method has been trained on the BCWhole data set three times, and then the average of the evaluation scores and their standard deviation has been computed.*

Method	MAP	AUROC	Top-1	Top-5	CE
CubeRun	$67.3 \pm 0.4\%$	$96.9 \pm 0.0\%$	$63.5 \pm 0.3\%$	$80.0 \pm 0.4\%$	25.3 ± 0.3
Resnet 18	$64.4 \pm 1.1\%$	$96.5 \pm 0.0\%$	$60.3 \pm 1.1\%$	$78.1 \pm 1.1\%$	28.7 ± 1.5

Table 4.3: *The MAP score for the baseline, the residual neural network, and the previous state-of-the-art when evaluated on the hidden BirdCLEF test set with and without consideration of the background species.*

Method	MAP (with)	MAP (without)
Baseline	55.8%	69.7%
Resnet 18	53.8%	67.9%
Previous	55.5%	68.6%

To summarize: these results indicate that, residual networks can learn to classify bird species, residual networks learns training data well, but the variance in generalization is high, and residual networks do not improve upon the baseline, but the results are comparable.

4.3 Meta-Data Fusion

Meta-data fusion of elevation data improves the performance of both the baseline and the residual neural network. In particular, the coverage error is reduced, which means that we can predict a lower number of bird species and still cover the ground truth bird species on average.

Reducing the coverage error could be of interest in, e.g., a smartphone application which aids the user with bird species identification. Let us say that the user walks in the woods and records a bird song, the application then returns the five most probable bird species for that recording. If the coverage error, on a data set which represents the population of birds in those particular woods, is five, then we have a good reason for why the top-5 predictions are enough.

All of the evaluation scores for the baseline and the residual neural network can be seen in Table 4.4. Considering the baseline the AUROC is increased by 0.6 percentage points, the top-5 accuracy by 1.3 percentage points, and the coverage error (CE) is decreased by 4.2 when compared to not using the data fusion (compared to Table 4.2). For the residual neural network, the AUROC is increased by 0.5 percentage points, the top-5 accuracy by 1.5 percentage points, and the coverage error is decreased by 4. The MAP and top-1 scores stay roughly the same.

The improved scores all reflect how high the rank of the ground truth class is in the prediction, meaning that fusion of elevation data makes the model predict the true class higher, however, it does not seem to be enough to push the prediction to

Table 4.4: *The MAP, AUROC, top-1 accuracy, top-5 accuracy, and CE of the baseline, and the residual neural network when meta-data fusion of the elevation is used. Each method has been trained on the BCWhole data set three times, and then the average evaluation and the standard deviation of these have been taken.*

Method	MAP	AUROC	Top-1	Top-5	CE
CubeRun	$67.5 \pm 0.1\%$	$97.5 \pm 0.0\%$	$63.4 \pm 0.7\%$	$81.3 \pm 0.2\%$	21.1 ± 0.3
Resnet 18	$65.1 \pm 1.1\%$	$97.0 \pm 0.0\%$	$60.1 \pm 1.2\%$	$79.6 \pm 1.0\%$	24.7 ± 1.2

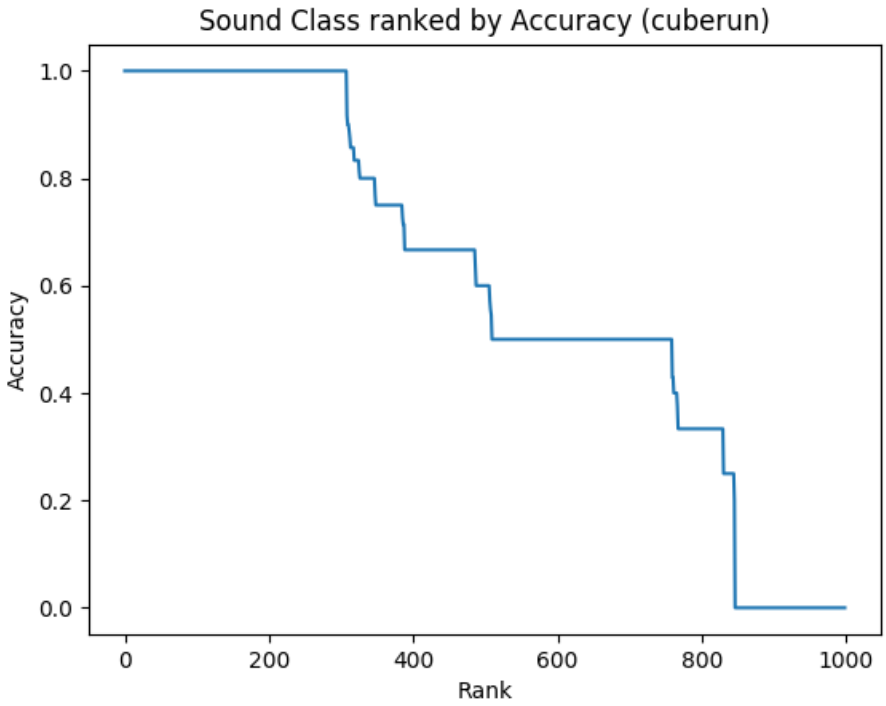


Figure 4.3: *The accuracy of the model for the sound classes ranked by accuracy. The plot shows that the accuracy of the model varies a lot between different sound classes. It is 100% for the highest ranked and 0% for the lowest ranked classes.*

the highest rank, since the MAP and top-1 accuracy seem unaffected.

4.4 Data Analysis

The accuracy of the model varies a lot for the different sound classes as can be seen in Figure 4.3, which shows a plot of the accuracy for each sound class ranked by accuracy. The model has a perfect accuracy for the (around) 350 sound classes on which it performs best, then the accuracy decreases (somewhat linearly) until it hits a zero accuracy for the (around) 180 classes on which it performs worst.

Both the residual neural network and the baseline were trained on the 100 sound classes on which they respectively had the worst performance (BCResnetBot100 and BCResnetBot100). The accuracy when trained and evaluated on BCCubeRunBot100 and BCResnetBot100 is significantly lower than when trained and evaluated on BCWhole (except for the CE score, which is not really comparable since the number of sound classes is decreased tenfold) (see Table 4.5).

Since BCCubeRunBot100 and BCResnetBot100 each consists of only 100 sound classes in contrast to BCWhole which consists of 999 sound classes one would expect the accuracy to increase rather than decrease when trained and evaluated on these data sets. However, this is not the case, which indicates that there is something fundamentally hard about these sound classes, which warrants further investigation.

The confusion matrices, which are shown in Figure 4.4, for the methods when

trained and evaluated on the BCResnetBot100, and BCCubeRunBot100 data sets reveal that there is a small tendency to confuse sound classes for one or two specific classes. That is, the networks favor the prediction of these classes over others. In particular, the residual network favors prediction of sound classes 47, 61, and 74 (each getting more than 10 predictions), and the baseline favors sound class 93 in one of the training sessions. The baseline has a lower tendency to favor sound classes.

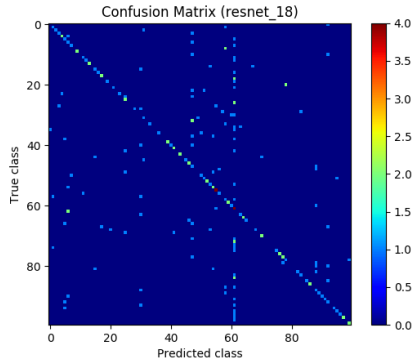
A possible explanation for this favoritism is that the data set is quite uneven. The number of training segments for each sound class vary a lot, and in Figure 4.5 we can see that the number of predictions that some sound classes receive (red line) are higher than the expected number of predictions for these sound classes (green line) when the number of training segments is large, and that the number of predictions that sound classes receive are lower than expected when the number of training segments is small. Meaning that sound classes with many training segments are favored over those with few training segments in the predictions.

4.5 Optimization Methods

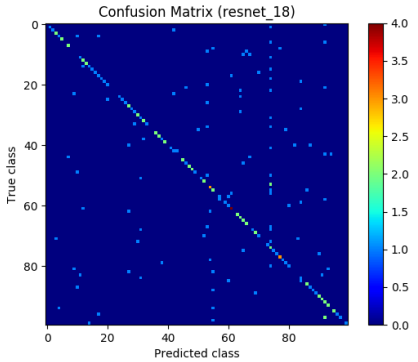
Seven different optimization methods were tested to see how the choice of optimizer affects the convergence rate during training (see Figure 4.6). The stochastic gradient descent and adadelta optimizers both achieved good convergence rates, while the rmsprop, adamax, nadam, adam and adagrad optimizers barely achieved any convergence at all. The adamax optimizer seems to start converging after around 60 epochs, however very slowly. These results should only be considered as indicative, but are included to get a picture of how the choice of optimizer can affect training.

Table 4.5: *The MAP, AUROC, top-1 accuracy, top-5 accuracy, and the CE of the baseline, and the residual neural network when trained on dataset BCCubeRunBot100 and BCResnetBot100 respectively.*

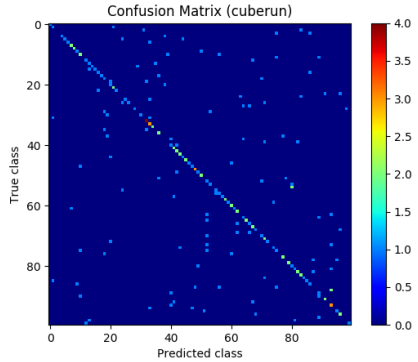
Method	MAP	AUROC	Top-1	Top-5	CE
CubeRun	54.3 %	92.8 %	49.3 %	71.8 %	6.5
Resnet 18	47.7 %	90.5 %	42.3 %	69.2 %	10.6



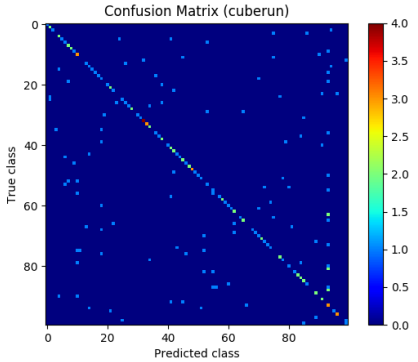
(a) *The confusion matrix for the residual network when trained on BCResnetBot100. The network seems to favor sound classes: 61, and 47.*



(b) *The confusion matrix for the residual network when trained on BCResnetBot100. The network seems to favor sound class 74.*



(c) *The confusion matrix for the baseline when trained on BCCubeRunBot100. The network does not seem to favor any sound classes.*



(d) *The confusion matrix for the baseline when trained on BCCubeRunBot100. The network seems to favor sound class 93.*

Figure 4.4: *Confusion matrices for the residual neural network (a) and (b), and the baseline (c) and (d) when each has been trained and evaluated twice on the BCResnetBot100, and BCCubeRunBot100 respectively. A perfect accuracy would result in a clean diagonal line. The rows are the ground truth labels, and the columns are the classes predicted by the model. Some sound classes seem to pick up more predictions than others, meaning that the network seems to favor prediction of some sound classes over others.*

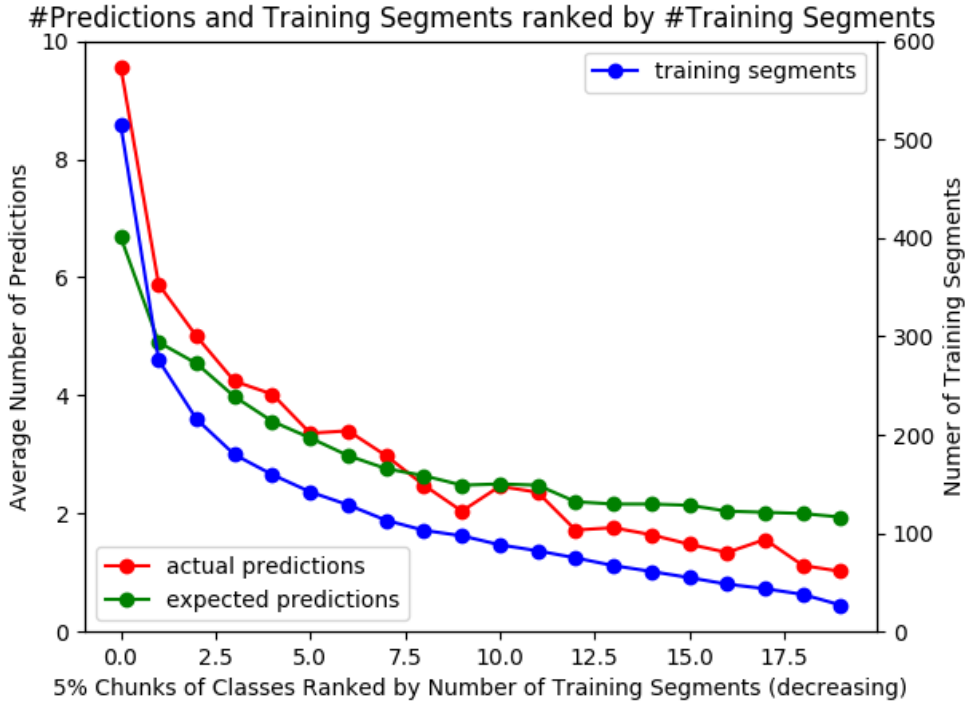


Figure 4.5: *The figure shows the number of training segments (blue) plotted on the right y-axis with respect to 5% chunks of the sound classes ranked by the number of training segments in each 5% chunk. It also shows the average number of predictions that the chunks of sound classes receive (red) and the expected average number of predictions for each chunk (green) plotted on the left y-axis.*

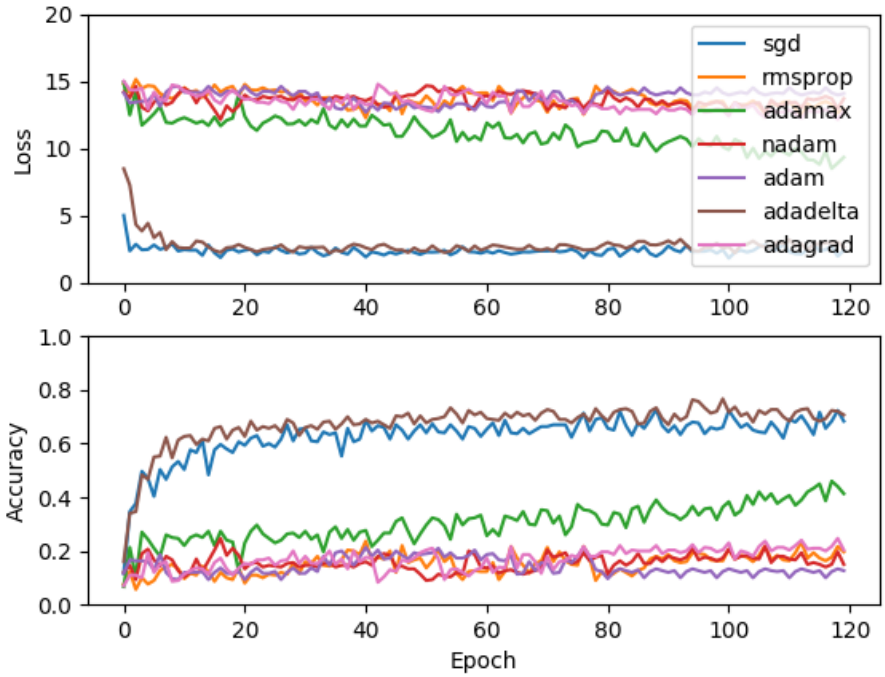


Figure 4.6: *Validation loss (top) and accuracy (bot) for seven different optimizers when training the baseline model on the BCSubset data set.*

5
Discussion

The goal of this project was to improve upon the state-of-the-art bird species classifier presented by Sprengel et. al [1]. A small literature survey of recent results published from different bird species classification challenges was conducted which indicated that convolutional neural networks and data augmentation techniques could be key for pushing this technology even further.

Convolutional neural networks are originally designed for image recognition, but have proven to be useful also in the audio domain [1, 8, 16]. During the last years of ILSVRC, there has been a correlation between increased network depth and increased accuracy, which led us to believe that this could also be true for convolutional neural networks when applied in the audio domain. Therefore the use of deep residual neural networks was proposed.

We implemented and evaluated the preprocessing, data augmentation and the convolutional neural network techniques proposed by Sprengel et. al [1] as a baseline, and used the same preprocessing and data augmentation techniques but for training a deep residual neural network.

The deep residual neural network did achieve a classification accuracy comparable to the state-of-the-art, but not quite matching it. However, the residual neural network has not been tuned or customized for this problem domain, but rather used as presented in the original papers. Tuning the hyper-parameters could possibly improve performance further. It is also a quite new network architecture and an area of active research. For example, Zagoruyko et. al [35] consider increasing the width of the network rather than the depth and argue that width is more important than depth, which could be the case also for this task.

A common problem when training neural networks is often the lack of available training data. This has led to the development of different kinds of data augmentation techniques. In addition to the data augmentation techniques used by Sprengel et. al [1], we therefore implemented and evaluated a technique called multiple-width frequency-delta data augmentation [16], which augments MFCCs instead of raw spectral data.

The MWFD data augmentation technique does seem to improve the accuracy when compared to pure MFCCs, which is in accordance to [16], meaning that the technique works as intended. However, it does not seem to improve upon the raw spectrogram representation, which is in line with some recent findings by Stowell et. al [36] who found that the use of raw Mel spectra often outperform MFCCs. One of the main reasons for using MFCCs is that it is a compact feature representation [36]. That is, the dimensionality is lower if compared to, e.g., raw spectrograms. In this thesis, the MWFD feature vector is eight times smaller than the raw spectral

representation, but the accuracy is comparable to that of the raw spectral feature vector, which means that if computational resources are limited the MWFD data representation has an advantage over raw spectral data.

One aspect which has been largely unaddressed in the bird challenge community is the use of additional meta-data such as elevation, geographical location and the time of recording. This is also pointed out by Sprengel et. al who writes: "the dataset provides us with a lot of meta-data: Date, time and location to name a few. We are currently only relying on the sound files but incorporating these values could greatly increase our score because we could narrow down the total number of species which we need to consider" [1]. Therefore a proof of concept was developed for a simple data fusion method.

The fusion of elevation information into the model does seem to increase the rank the ground truth classes for the model, but not enough to affect the top-1 accuracy. It may be possible to use a similar method for time and location as well which may increase the rank even further.

However, the use of meta-data does come with some assumptions about the available data set. One is that we have enough observations of birds at different elevations to get a good understanding of how each bird species is distributed at different elevations. In the worst case all recordings of a bird species are taken at the same place by one recordist, which would restrict the elevation data that is useful for identifying that bird to a very narrow distribution. This is the main reason for why we have chosen to model the elevation data distributions for each bird species as a mixture density of the normal distribution and the uniform distribution, tending towards the uniform if the number of elevation observations is low.

After evaluation of the baseline and the residual neural network, it became evident that the accuracy for the different sound classes varies a lot, which led to an analysis of the data. A problem with the data set is that the number of training samples for each bird species is quite uneven; it can vary from around 10 recordings to 200 recordings. This means that the classifier trained on the data set can start to favor, or overfit to, the well represented bird species. This tendency is indicated in Figure 4.5 where we see that the network favors the prediction of classes which have lots of training samples. A possible way to mitigate this would be to weigh the sound classes by their relative amount of training samples when shown to the neural network during training.

However, it may not only be the number of training samples which affect this favoritism; it can also be the variability of the recordings themselves. As shown in Figure 4.4 the networks tend to favor one or two classes over the others. These sound classes have been manually inspected and can only be described as a cacophony of different bird songs and noises, while some of the classes for which the network performed well had a much clearer foreground species and less noise. To address this problem it may be necessary to modify the loss function and consider both main species, and the background species, during training. That is, a way forward could be to move from the single-instance single-label problem formulation into the single-instance multi-label problem formulation, which is a harder problem, but possibly with a greater potential if done right.

Future Work

As mentioned in the discussion a way forward could be to tackle the harder single-instance multi-label problem and modify the loss function such that it considers the background species. This may, however, necessitate larger volumes of training data, which may not be available.

Another way forward could be to use the results from the recent bird audio detection challenge [37] hosted by the Queen Mary University of London where the objective is to detect presence or absence of bird song in audio recordings. This could be useful either in the preprocessing step, or to move towards the more realistic open set scenario of continuous bird species monitoring rather than the N-class problem which is studied in this thesis. In the preprocessing step it could be used to actually classify which parts of the recordings are noise and which are bird song. This could result in a more accurate representation of the training data. In a continuous classification system, where birds are detected continuously as they sing, we could use it to classify which data should be stored for further analysis.

Recent work on the continuous problem has shown that a classification system which uses random forests and unsupervised feature learning can be used to detect trends in, e.g., nocturnal bird migration [38]. Therefore an interesting way forward for this technique would be to see if it can be used in a similar setting, where region specific classifiers are trained and used to detect trends in bird species behavior.

6
Conclusions

In this thesis, we set out to improve the classification accuracy of the state-of-the-art bird species classifier presented by Sprengel et. al [1]. The examined methods are deep residual neural networks [2, 3], multiple-width frequency-delta data augmentation [16], and fusion of elevation meta-data into the model. We wanted to answer the following research questions: (i) Can deep residual neural networks be used to classify bird species based on acoustical data recordings, and how well do they perform? (ii) Can multiple-width frequency-delta data augmentation be used to improve classification accuracy of the classifier in this problem domain? and (iii) can the additional meta-data of the recordings be used to improve classification accuracy?

The main findings are: Firstly, deep residual neural networks can be used to classify bird species based on acoustical data recordings, and the performance is close to the state-of-the-art, but not quite matching it. Secondly, multiple-width frequency-delta data augmentation can not be used to increase classification accuracy when compared to raw spectral data, but the accuracy is close to the state-of-the-art and has an advantage over raw spectral data when computational resources are limited. Finally, the use of additional meta-data increases the rank of the ground truth species in the predictions of the models, but it does not seem to be enough to push it to the highest rank, which means that the model has to predict fewer species, on average, to cover the ground truth, but the actual top-1 accuracy does not seem to be affected.

Through an analysis of the data set used we also found that the relative number of training samples for each bird species is quite uneven, which seems to lead to a favoritism, or over prediction, from the model of bird species with the most recordings, and that some bird species are harder to classify than others.

Bibliography

[1] Elias Sprengel, Martin Jaggi, Yannic Kilcher, and Thomas Hofmann. Audio Based Bird Species Identification using Deep Learning Techniques. 2016.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Arxiv.Org*, 7(3):171–180, 2015.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. *arXiv preprint*, pages 1–15, 2016.

[4] Cağan H Sekercioğlu, Richard B. Primack, and Janice Wormworth. The effects of climate change on tropical birds. *Biological Conservation*, 148(1):1–18, 2012.

[5] Forrest Briggs, Balaji Lakshminarayanan, Lawrence Neal, Xiaoli Z Fern, Raviv Raich, Sarah J K Hadley, Adam S Hadley, and Matthew G Betts. Acoustic classification of multiple simultaneous bird species: a multi-instance multi-label approach. *The Journal of the Acoustical Society of America*, 131(6):4640–4650, 2012.

[6] T Mitchell Aide, Carlos Corrada-Bravo, Marconi Campos-Cerqueira, Carlos Milan, Giovany Vega, and Rafael Alvarez. Real-time bioacoustics monitoring and automated species identification. *PeerJ*, 1:e103, 2013.

[7] Jaderick P. Pabico, Anne Muriel V. Gonzales, Mariann Jocel S. Villanueva, and Arlene a. Mendoza. Automatic identification of animal breeds and species using bioacoustics and artificial neural networks. *arXiv preprint*, pages 1–17, 2015.

[8] Forrest Briggs, Yonghong Huang, Raviv Raich, Konstantinos Eftaxias, Zhong Lei, William Cukierski, Sarah Frey Hadley, Adam Hadley, Matthew Betts, Xiaoli Z. Fern, Jed Irvine, Lawrence Neal, Anil Thomas, Gabor Fodor, Grigorios Tsoumakas, Hong Wei Ng, Thi Ngoc Tho Nguyen, Heikki Huttunen, Pekka Ruusuvuori, Tapio Manninen, Aleksandr Diment, Tuomas Virtanen, Julien Marzat, Joseph Defretin, Dave Callender, Chris Hurlburt, Ken Larrey, and Maxim Milakov. The 9th annual MLSP competition: New methods for acoustic classification of multiple simultaneous bird species in a noisy environment. *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*, 2013.

[9] Peter Jancovic and Munevver Kokuer. Acoustic recognition of multiple bird species based on penalised maximum likelihood. *IEEE Signal Processing Letters*, 22(10):1–1, 2015.

[10] Alexander N G Kirschel, Dent A Earl, Yuan Yao, Ivan A Escobar, Erika Vilches, Edgar E Vallejo, and Charles E Taylor. Using Songs To Identify Individual Mexican Antthrush Formicarius Moniliger: Comparison of Four Classification Methods. *Bioacoustics-the International Journal of Animal Sound and Its Recording*, 19(1-2):1–20, 2009.

[11] R. Bardeli, D. Wolff, F. Kurth, M. Koch, K. H. Tauchert, and K. H. Frommolt. Detecting bird sounds in a complex acoustic environment and application to bioacoustic monitoring. *Pattern Recognition Letters*, 31(12):1524–1534, 2010.

[12] Daniel T. Blumstein, Daniel J. Mennill, Patrick Clemins, Lewis Girod, Kung Yao, Gail Patricelli, Jill L. Deppe, Alan H. Krakauer, Christopher Clark, Kathryn A. Cortopassi, Sean F. Hanser, Brenda Mccowan, Andreas M. Ali, and Alexander N G Kirschel. Acoustic monitoring in terrestrial environments using microphone arrays: Applications, technological considerations and prospectus. *Journal of Applied Ecology*, 48(3):758–767, 2011.

[13] Jason Wimmer, Michael Towsey, Paul Roe, and Ian Williamson. Sampling environmental acoustic recordings to determine bird species richness. *Ecological Applications*, 23(6):1419–1428, 9 2013.

[14] Brett J. Furnas and Richard L. Callas. Using automated recorders and occupancy models to monitor common forest birds across a large geographic region. *The Journal of Wildlife Management*, 79(2):325–337, 2 2015.

[15] Alexis Joly, Herve Goeau, Herve Glotin, Concetto Spampinato, Pierre Bonnet, Willem-Pier Vellinga, Robert Planque, Andreas Rauber, Robert Fisher, and Henning Müller. LifeCLEF 2014: Multimedia Life Species Identification Challenges. (ii):229–249, 2014.

[16] Yoonchang Han and Kyogu Lee. Acoustic scene classification using convolutional neural network and multiple-width frequency-delta data augmentation. 14(8):1–11, 2016.

[17] Stéphane Mallat. Understanding Deep Convolutional Networks. *Philosophical Transactions of the Royal Society A*, pages 1–17, 2016.

[18] Yonghong Huang, Forrest Briggs, Raviv Raich, Konstantinos Eftaxias, and Zhong Lei. The Ninth Annual MLSP Data Competition. *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*, 2013.

[19] Mario Lasseck. Bird song classification in field recordings: Winning solution for NIPS4B 2013 competition. *Proc. of int. symp. Neural Information Scaled ...*, pages 1–6, 2013.

[20] Alexis Joly. LifeCLEF Bird Identification Task 2016. 2016.

[21] Naoya Takahashi, Michael Gygli, Beat Pfister, and Luc Van Gool. Deep Convolutional Neural Networks and Data Augmentation for Acoustic Event Detection. *arXiv preprint arXiv: ...*, (August):1–5, 2016.

[22] Herve Goeau, Herve Glotin, Willem Pier Vellinga, Robert Planque, Andreas Rauber, and Alexis Joly. LifeCLEF bird identification task 2015. *CEUR Workshop Proceedings*, 1391, 2015.

[23] librosa 0.5.0.

[24] Beth Logan. Mel Frequency Cepstral Coefficients for Music Modeling. *International Symposium on Music Information Retrieval*, 28:11p., 2000.

[25] Yann LeCun, Leeon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012.

[27] Yann LeCun, Yoshua Bengio, and Hinton Geoffrey. Deep learning. *Nature Methods*, 13(1):35–35, 2015.

[28] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167*, pages 1–11, 2015.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: prevent NN from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[30] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 9:249–256, 2010.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR*, abs/1502.0, 2015.

[32] Ilya Sutskever, James Martens, George E Dahl, and Geoffrey E Hinton. On the importance of initialization and momentum in deep learning. *Jmlr W&Cp*, 28(2010):1139–1147, 2013.

[33] Ruben Gonzalez. Better than MFCC audio classification features. *The Era of Interactive Media*, pages 291–301, 2013.

[34] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, (3):807–814, 2010.

[35] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. *Arxiv*, 2016.

[36] Dan Stowell and Mark D. Plumbley. Automatic large-scale classification of bird sounds is strongly improved by unsupervised feature learning. *PeerJ*, 2:e488, 2014.

[37] Dan Stowell, Mike Wood, Yannis Stylianou, and Hervé Glotin. Bird detection in audio: a survey and a challenge. 2016.

[38] Justin Salamon, Juan Pablo Bello, Andrew Farnsworth, Matt Robbins, Sara Keen, Holger Klinck, and Steve Kelling. Towards the automatic classification of avian flight calls for bioacoustic monitoring. *PLoS ONE*, 11(11):1–26, 2016.

[39] Francois Chollet. Keras, 2015.

[40] Eric Jones, Travis Oliphant, and Pearu Peterson. SciPy: Open source scientific tools for Python, 2001.

[41] Stéfan van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in Python. *PeerJ*, 2:e453, 1 2014.

[42] Stéfan Van Der Walt, S. Chris Colbert, and Gaël Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science and Engineering*, 13(2):22–30, 2011.

A

Implementation Details and Usage

In this appendix implementation details for the project are listed, and we explain how the project files can be used to reproduce the results in this thesis. The full source code for the project can be found at:

`https://github.com/johnmartinsson/bird-species-classification`

and since usage instructions are susceptible to change due to updates, or bugs in the code, the reader is referred to the github page if the instructions in this section is not working.

The main library used to build the network models in this thesis is the Keras deep learning library [39], which is aimed at academic researchers in the field of deep learning. Both of the neural network architectures are implemented in this library. Other libraries used include: librosa [23], scipy [40], scikit-image [41] and numpy [42], please see Table A.1 for a summary of the versions and their most useful methods for this project.

A.1 Data set

The data set can be downloaded from the BirdCLEF home page after registration to their system. The raw data set consists of a *wav* directory which contains all of the recordings and a *xml* directory with one XML file for each recording containing the class id and all other meta-data information. The test set is private and it is necessary to contact the organizers in order to get a method evaluated.

Table A.1: *The main software libraries used during development, their respective version numbers, and their most useful methods for this project.*

Library	Version	Main Methods
Keras	1.2.1	
librosa	0.4.3	stft, feature.{mfcc, delta}
scipy	0.18.1	misc.imresize
skimage	0.12.3	morphology.{binary_erosion, binary_dilation}
numpy	1.12.0	

A.2 Preprocessing

In order to, as close as possible, reproduce the results in this thesis we must first preprocess the raw data set. The recordings are first down sampled to 22,050 Hz using the linux command-line tool *sox*:

```
for i in *; do sox $i -r 22050 tmp.wav; mv tmp.wav $i; done
```

which down samples the audio files in the current working directory. The next step is to split the down sampled audio files into signals and noise. Simply run:

```
python preprocess_birdclef.py --xml_dir=<path/to/xml/dir> \
                              --wav_dir=<path/to/wav/dir> \
                              --output_dir=<path-to-output-dir>
```

and it should load the XML data files in *xml_dir*, and each wave file in *wav_dir* and split these wave files into signal and noise segments which are saved in their respective sound class directory (all noise segments in one directory, and the signal segments in their respective class id directory). The resulting *output_dir* should contain a *signal* directory with all the signal segments for each of the 999 sound classes, and a *noise* directory with all the noise segments.

When this is done the next step is to split the data into a validation set, and a training set:

```
python create_dataset.py --src_dir=<path/to/signal/dir> \
                         --dst_dir=<path/to/destination/dir> \
                         --subset_size=<subset-size> \
                         --valid_percentage=<validation-percentage>
```

where the source directory should be the signal directory produced before, the destination directory is the destination of the final data set, the subset size can be used to only chose a random subset of the whole data set, and the validation percentage is how many percent of the split that should be validation data (in this thesis 10%).

A.3 Training

The models are trained using a modified version of Keras ImageDataGenerator which is a part of the image preprocessing module. The data generator has been modified to load sound files instead of image files, and is therefore called a SoundDataGenerator. The generator lets the input data flow from a specified data directory where the signal files for each sound class must be contained in a different folder. The sound data generator can be configured to load the sound files as spectrograms, MFCCs, and MWFD data augmented MFCCs. It can also be configured to apply time-shift, pitch-shift, same class and noise augmentation using the respective boolean flags.

The main training script is run by calling:

```
python3 train.py --config_file=conf.ini
```

where "conf.ini" is an ini file containing the configuration information for the training instance (see Appendix A.3.1 for more derails).

The resulting weights, and data from the training will be put in a directory such as: *2017_02_09_000112_resnet_18*, which is named after the date, time and name of the model used during this training session. The directory will contain *weights.h5*

which are the final weights of the model from the training session, *stdout.log* which is a log file of training session, *history.pkl* which is a pickle file with a dump of the training history from the training session, containing training- and validation- loss and accuracy.

A.3.1 Configuration File

The configuration file is divided into three sections: model, paths and training. The model sections contains the following variables:

```
[MODEL]
BatchSize = 16
NumberOfClasses = 999
NumberOfEpochs = 6
NumberOfIterations = 20
NumberOfValidationSamplesPerEpoch = 17196
NumberOfTrainingSamplesPerEpoch = 124994
InputShape = (256, 512, 1)
InputDataMode = spectrogram
ModelName = resnet_18
```

The code is written in such a way that it divides a training session into chunks which are supplied to a job queue in Sun Grid Engine. In the above example the training session would be split into 20 iterations, or jobs (*NumberOfIterations*), each job would run for 6 epochs, and the number of training samples shown during each epoch would be 124994 using a batch size of 16. The input shape of the model would be $(256, 512, 1)$, which corresponds to rows, columns and channels respectively. $InputDataMode \in \{spectrogram, mfcc, mfcc\_delta\}$ is the representation of the input data so the model would take the logarithmic spectrogram of the augmented training samples as input, and $ModelName \in \{cuberun, resnet\_\{18, 34, 50, 101, 152\}\}$ is the name of the neural network model, so the model used would be *resnet_18* which is a residual neural network with 18 layers.

The paths section is used to specify where the noise-, training-, and validation data is:

```
[PATHS]
BaseName = None
NoiseDataDir = /path/to/noise
TrainingDataDir = /path/to/train
ValidationDataDir = /path/to/valid
```

where *NoiseDataDir* is the path to the directory containing all noise segments, *TrainingDataDir* is the path to the directory containing the training data, *ValidationDataDir* is the path to the directory containing the validation data. The parameter *BaseName* can be supplied if training should be continued for a previously (un)finished training job.

The optimizer, loss function, and data augmentation used during training can be configured in the training section:

```
[TRAINING]
Optimizer = adadelta
```

```
LearningRate = 0.001
Decay = 1e-6
Momentum = 0.9
Nesterov = True
LossFunction = categorical_crossentropy
TimeShiftAugmentation = True
PitchShiftAugmentation = True
SameClassAugmentation = True
NoiseAugmentation = True
```

where $Optimizer \in \{sgd, msprop, adamax, nadam, adam, adadelta, adagrad\}$ is the optimizer that should be used, if *sgd* (stochastic gradient decent) is used, then the learning rate, decay, momentum and nesterov parameters can be set, otherwise these can be ignored. It is also possible to set the loss function, and which of the different data augmentation techniques should be used.

A.4 Run Predictions

Once the data has been preprocessed, and the model has been trained, it is necessary to test the model on some previously unseen data. This can be done by calling the script:

```
python run_predictions.py --experiment_path=<path/to/experiment>
```

where the *experiment_path* is the directory created during training, which contains the weights and the configuration file of the model. The predictions will automatically be run for the validation data pointed to in the configuration file. The predictions will be saved to a pickle file in the directory pointed to.

A.5 Evaluation

A model can be evaluated after the predictions have been made by running:

```
python evaluate.py --experiment_path=<path/to/results>
```

where the path to the results of the training session is specified, for example: *2017_02_09_000112_resnet_18*. This will print the top-{1-5} accuracy, MAP, AUROC, and CE scores in the console, and return a dict with the results. This can be used in the *ipython* interpreter in order to collect results from multiple evaluations, and then compute mean and standard deviations, or in a separate script.

A.6 Hardware

The hardware used when training the neural networks in this thesis are an Intel(R) Core(TM) i7-5930K CPU running at 3.50GHz and a GeForce GTX Titan X GPU with 12 GB RAM. All training sessions have been run on the GPU in this system.