University of Manchester
School of Computer Science

B.Sc. Computer Science
Third Year Project Report
May 2017

Automatic Birdsong Recognition

Author: Victor F. Lampreia Rodrigues

Supervisor: Dr. Andrea Schalk

Abstract

Automatic birdsong recognition is a sparsely studied subject. This project takes an image recognition based machine learning approach to explore the viability of accurate automatic identification of a high number of species present in actual field recordings around the world. This task involves the application of image processing and cross-correlation mechanisms to extract characteristic song segments from spectrograms, used to help identify similarities between bird songs and classify any species present in recordings using a random forest classifier. We conclude that an acceptable accuracy is achievable with the approached methods, with possible limitations due to computational expenses.

Contents

INTRODUCTION

The ability to recognise bird species through the sounds they produce has benefited ornithologists and amateurs in the observation of such creatures in the wild. Most birds vocalise, for communicative purposes involving mating, territorial claims, and coordination efforts.

Vocalisations are categorised into songs and calls, which differ by the duration and complexity of the sounds. Calls are relatively simple in structure and are often brief. Conversely, songs are usually composed of long sequences of sounds, often featuring structurally complex and melodic tones, with some repetition.

The trained ear can distinguish birds through their songs and calls, but with over 10,000 uniquely catalogued species around the world, this becomes a far cry from practicality. Some methods have been developed to assist in the identification of species. The introduction of the use of sonograms for instance has made identification both quicker and more precise for experienced ornithologists, undoubtedly influencing the breadth and depth of scientific inquiry into the behaviour of birds.

Current manual and semi-automatic methods for bird sound recognition are limited by the cost and challenge of analysing enormous amounts of field recordings. A fully automatic mechanism would be highly advantageous and could allow for large scale, close to real-time or passive classification.

Example applications of such a system include population monitoring and migration tracking of species in the field of biogeography and for conservation efforts, as well as the possibility to support public software for the general birdwatcher.

The problem itself touches on many facets of computer science, such as digital signal processing and analysis, image recognition, machine learning, and performance optimisation.

1.1 Assumptions and Scope

The primary aim of this project is to research and develop potential methods for automatic birdsong recognition, such that the system functions with little to no user interaction. The end-goal is to produce a program which is able to identify the species of the most prominent bird present in a recording. It is not the goal of this project to produce commercial software ready for public consumption.

Recordings used for development and evaluation originate from arbitrary locations and sources around the world. They do not undergo any form of manual selection or processing. In similar vein, they are also not subjected to rigorous quality control, aside from basic metadata-based quality filtering. This is to ensure that validation represents real-world performance as much as possible.

For the purposes of this project, some restrictions in scope are imposed to simplify some problem areas:

- A general level of quality is ensured by selecting only higher quality recordings as defined by the data source. This is a reasonable restriction as it increases the signal to noise ratio.
- The number of species the system should know about is limited to 12 randomly picked labels.
- Only bird songs are considered.

1.2 Existing Approaches

Automatic birdsong recognition has become increasingly popular in the field of data science within the past decade. Several competitions have taken place to develop solutions to the problem, each with varying requirements and data-sets.

The MLSP 2013 challenge (Kaggle n.d.) for example has lead to a few successful approaches with a fairly normalised data set consisting of 645 10 second audio files covering 19 individual species, as well as data which may be used directly as features for classification. The most successful entry (Fodor 2013) achieved an AUC of 95.6% using spectrogram cross-correlation and a binary relevance (ensemble) approach with random forests.

The LifeCLEF 2014 challenge (Goëau et al. 2014) expanded on this by offering a much larger data set of 14027 non-controlled audio files sourced from Xeno-Canto (*Xeno Canto* n.d.), covering 501 different species. A winning solution (Lasseck 2014) achieved an AUC of 91.5%, with a precision of 51.1%. The solution made use of a combination of audio scene analysis techniques, spectrogram cross-correlation, and an extremely randomised trees classifier.

The author had also competed in the NIPS 2013 (**nips**) competition using spectrogram segment analysis and spectrogram cross-correlation with an ensemble of extremely randomised trees classifiers. An AUC of 91.6% was achieved.

A few commercial software solutions exist, such as Warblr (*Warblr* n.d.) and Chirpomatic (*Chirpomatic* n.d.), although these appear to be limited in scope and accuracy.

This task has also been tackled in 2016 at the University of Manchester as a third year project, which used a dynamic time warping algorithm to measure similarities between waveforms, acheiving an accuracy of 63% across 4 species (Palmer 2016).

1.3 Our Approach

Our initial approach is inspired by similar sound recognition problems and existing solutions. Such problems include the recognition of voice, music, and animal vocalisations of species other than birds. It is observed that these problems, while similar in nature, differ in practice. This is due not only because of the structure of the sound being analysed, but also due to the intformation contained within those structures.

For example, the structure of human voice is well known, and patterns are automatically identified with good accuracy today. Specialised methods for modelling and analysing voice

recordings exist, such as hidden Markov models, dynamic time warping, and deep neural networks. Human speech recognition is however a different task entirely, as the goal here is to identify word utterances, and not the individual speaking. One could consider the problem of determining identity, which may provide insight into useful methods.

Music recognition is trivial in the case of identifying pure reproductions, the major challenge in this area being noise reduction and distortion compensation. Music can be easily identified using statistical methods to compare pitch variations along the duration of the recording. Bird song however contains many variations and transpositions within the same species which make it difficult to find an archetypal sequence of pitch variations. The issue of noise reduction remains relevant to our problem.

A common approach to solving these problems is to first reduce it to an image recognition problem by using the spectrographic representation of the audio. Elements descriptive of specific labels are searched for within a the spectrogram of a target example. The simplicity and previous successes of this approach has driven the direction of this project.

Our approach uses a combination of computer vision and machine learning techniques to construct a fully automatic recognition system. Standard image processing methods are used to process spectrograms and extract sections of song which may be used to identify a particular species, much like how an ornithologist visually inspects the song spectra. These sections are then cross-correlated against new samples, using the resulting matching probabilities to train a multi-class machine learning algorithm.

1.3.1 Process Overview and Document Outline

The project is divided into four discrete parts, which follow the logical flow of data in the system. Each part is detailed in its relevant section, including discussion on possible alternatives and improvements to the mechanisms that have been developed.

1. **Collection:** Data is sourced from field recordings taken in uncontrolled environments. The variety provides a good estimate of real-world performance and introduces many quality related issues.

2. **Preparation and Selection:** Recordings are filtered and selected to maintain reasonable quality levels. Spectrograms are then generated for use in the next stage.

3. **Preprocessing and Feature Extraction:** Noise is reduced as much as possible to identify key regions of interest within the spectrogram image. These are extracted as templates and cross-correlated against other recording spectrograms to form a feature vector of matching probabilities.

4. **Classification and Evaluation:** The resulting data is then fed to a classifier and evaluated using techniques designed to reduce statistical bias.

1.3.2 Architectural Overview

The program is designed around the flow of data in the system, split into discrete blocks. A monolithic approach was taken, such that the whole system is self-contained. Effort was taken to ensure a flexible framework was built to support fast implementation turnaround. This was

achieved to limited success, as there was no focus on maintaining a fully operational program at all times. For this reason, unit tests were not considered for implementation.

A summary of technologies and libraries used is available in Appendix C. Figure 1.1 describes the layout of the system by visualising the flow of data.

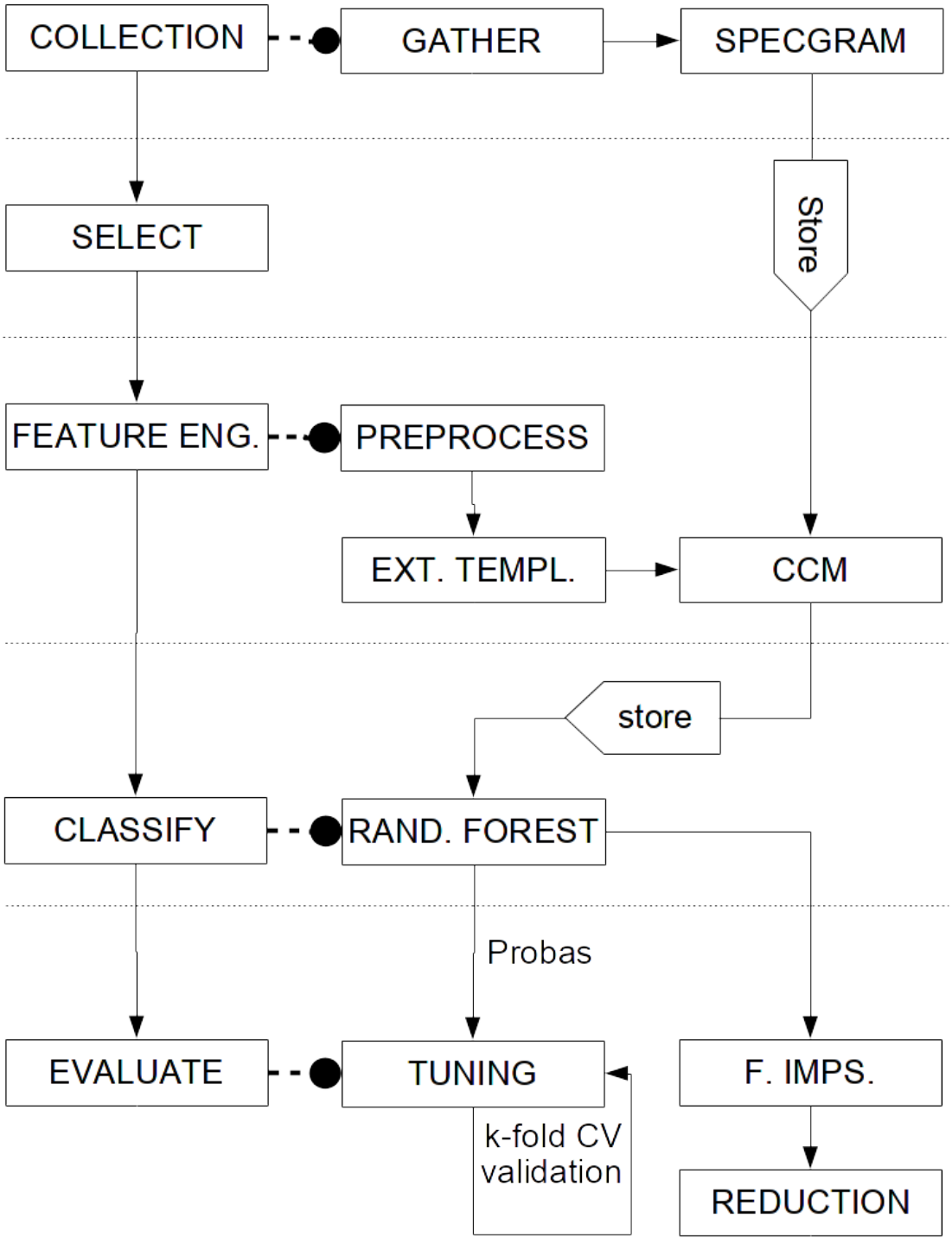


Figure 1.1: System block diagram

DATA COLLECTION AND PREPARATION

The raw data from which classifications are made is composed entirely of field recordings and metadata. This section walks through the particularities of this data, the mechanism for its collection, and preliminary preparation for use in the next stage, feature extraction.

2.1 Data Source: Xeno-Canto

Xeno-Canto (*Xeno Canto* n.d.) provides over 353830 recordings of 9728 individual species. Recordings consist of both calls and song, uploaded both by professional and amateur users. These may originate from any part of the world.

Recordings vary in duration, ranging from a few seconds to over 10 minutes. The content may be densely packed, with multiple birds vocalising simultaneously, or sparse with long durations of silence. Additionally, multiple species may be present in a single recording, however higher quality recordings tend to contain a singly identifiable species, and are otherwise labelled.

Xeno-canto provides all audio as dynamic MP3. These are normalised as described in Section 2.2

2.1.1 Metadata

Xeno-canto provides the following metadata with each recording:

- Date and time
- Recording location
- Species recorded
- Existence of other species

This program only makes use of the prominent species tagged in the recording. Although not used in our implementation, the location of the recording could be used to improve the accuracy of the classifier, as some species are restricted to certain parts of the world. Including this information in the feature set is likely to provide some accuracy by itself, but perhaps without much precision. It could however be used to narrow down the set of possible species, so long as special care is taken to account for migratory patterns. We have not pursued this.

2.1.2 Automatic Sample Retrieval

Manually selecting and downloading recordings is a time-consuming process. A public API does not exist for Xeno-canto, therefore we developed a web scraper specifically for automatic retrieval in `Python 2.7` using the `lxml` package.

The scraper allows the user to filter samples on species and on recording quality before downloading a sample by examining the metadata present in the HMTL. Filtering may be done by exclusion or selection.

When continuous fetching is desired, an interval may be set in order to reduce strain on Xeno-Canto's servers. Once an interval is set, the scraper will continuously download samples at the specified rate until it has been interrupted by the user.

The scraper was ran throughout the course of development, for an approximate accumulated time of 40 hours, with an average interval of 60 seconds between downloads. 2211 audio samples were collected, totalling just over 45 hours.

2.2 Preparation

Before the data can be further processed and used, some basic preparation must be done to normalise the data. This section discusses the transformative steps and the end representation that the program will work with from the initial state until classification.

2.2.1 Resampling

Because the source recordings vary in format, we first normalise these to the following format:

- **Format:** Raw WAV
- **Audio channels:** 1
- **Bit depth:** 16-bits
- **Sample rate:** 22050 Hz

If a recording is found to have more than a single channel, all but the left channel are discarded. It is unclear whether discarding a channel is better than summing them. Although summing retains all information, it may also introduce further noise. Conversely, it is also possible for the left channel to have a worse signal to noise ratio than the right. The number of multi-channel recordings is low enough however for this not to be much of an issue.

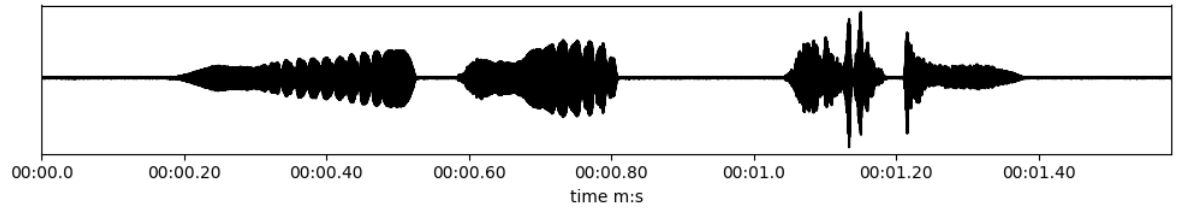
Sample rates are reduced to lower the memory and processing demands of analysing all 45 hours of audio. Our image recognition approach holds greater importance to the general shape of individual vocalisations in spectrograms, rather than high time-frequency details. Because of this, the sample rate reduction is found to have no significant reductions in classification quality.

The recording duration is left unaltered, so durations may differ from sample to sample.

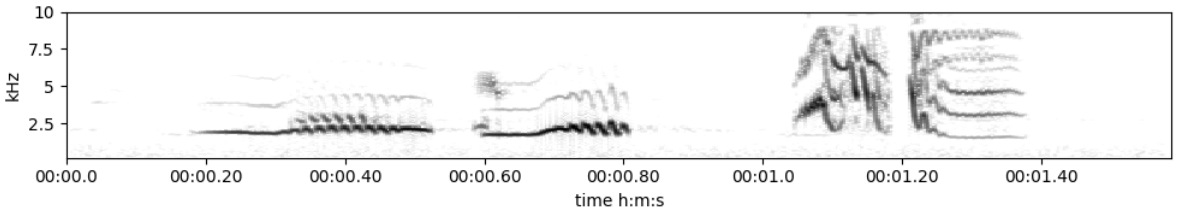
The FFmpeg (n.d.) program is used to convert all source material. We have written a shell script to make this procedure fully automatic, executable when new recordings have been sourced.

2.2.2 Spectrogram Representation

A spectrogram is a visualisation of a sound as energy in the frequency spectrum in function of time. This allows us to intuitively visualise the shape of a sound as a 2d image. Figure 2.1 shows how the spectrogram representation retains amplitude information from a waveform, while exposing the frequency composition of the waveform.



(a)



(b)

Figure 2.1: Waveform (a) and corresponding spectrogram (b) of a bird song recording

Our image recognition approach makes use of this spectrographic representation.

Spectrogram construction is performed for all recordings, sequentially, and immediately stored on disk. The audio is first loaded as raw pulse-code modulation (PCM). The windowed fast Fourier transform (FFT) method is then used to generate a spectrogram from the PCM.

Modern hardware handles FFT computations very well, making this stage very quick. The Matplotlib (Hunter 2007) `specgram` function is used to generate spectrograms.

The following parameters are used during spectrogram construction:

- **NFFT**: 512.
  Defines the number of PCM data points used in each chunk.

- **Window**: Hann window of 512 points.
  Windowing is used to merge overlapping chunks.

- **Overlap**: 75%.
  Overlap defines the number of points overlapping between chunks.

Spectrograms are converted to monochrome to simplify operations in the stages that follow. This does not reduce the information present in the spectrogram.

Frequencies above 10000Hz and below 100Hz are removed from the spectrograms as these are less likely to contain relevant vocalisations (*All About Birds* n.d.). These cuts will result in a reduction of baseline noise, as a lot of this falls under 100 Hz.

2.3 Sample Selection

A local database of birdsong samples is created, using collections extracted from Xeno-Canto through the described automatic procedure. Over the course of several days, 2211 total distinct samples have been downloaded and processed. 189 individual species have been catalogued, with approximately half of these represented by more than 10 individual samples. The top 10

species, and their sample counts are shown in Table 2.1. See Appendix A for a full list. The average length of each recording is approximately 1 minute.

Due to resource limitations imposed by template matching, not all samples are used in the final evaluation of the algorithm. Although all samples are processed as described in Section 2.2 and 3.2.1, only a selection is passed through feature extraction and classification stages.

The following criteria has been found to result in a good balance between data volume and time cost:

- Reject species with less than 20 samples
- Retain first 20 samples of each species
- Reject species with less than 2000 templates
- Keep first 3000 templates of each species

To reduce bias, samples and templates may instead be picked at random, however we did not have time to implement a method for tracking this random sampling.

Table 2.1: Top 10 species by sample count

Samples	Species Label
102	Eurasian Blackcap
101	Common Blackbird
98	Eurasian Wren
69	Common Chiffchaff
66	Grey-breasted Wood Wren
55	Spotted Towhee
49	Garden Warbler
46	Common Whitethroat
45	Eurasian Blue Tit
41	River Warbler

FEATURE SELECTION AND ENGINEERING

3.1 Useful Feature Identification

This section explores the information given by the spectrogram image representation of an audio file, and what features may be extracted to characterise and identify bird species.

3.1.1 Song Segment Formalism

Bird song structure is well defined in terms of segmentation (Catchpole and Slater 1995). The labelling of is defined in a naturally descending order of granularity, from large sequences to basic singular elements, and are identified by their length and duration of silence between sounds:

- **Song sequence:** An entire song is a complete end-to-end sequence of multiple or a single phrase. The same phrase is may be repeated exactly or with variation within a song. The first and last phrases of a song are usually different.
- **Phrase:** A phrase consists of a series of usually similar syllables. Phrases at the end of a song are often not composed of equal syllables.
- **Syllable:** Syllables may be simple or complex vocalisations. Complex syllables can be further partitioned into individual elements.
- **Element:** Elements form the most basic of vocalisations, for instance, sweeps or tones.

3.1.2 Distinctive Features in Spectrograms

It is common knowledge that bird songs are generally similar or consistent within a species, and differences exist between those different species. However, there exist some inconsistencies internal to a species which may become problematic if our system does not have enough samples to account for the variance.

One instance of such variation is regional, where the geographical location of birds has been found to correlate with nuanced differences in detail. Local isolation of populations also plays a part in minute variations, referred to as dialects (Podos and Warren 2007).

It has also been observed that birds learn aspects of their song, which affects the exactness of the reproduction (Kroodsma 1983). Recordings of younger birds is therefore expected to differ, possibly to a significant extent. This is an interesting subject in itself: it is plausible that an automatic system may be constructed which identifies not only the species of bird but also

an estimate of its age by similar cross-correlation methods.

Birds are able to produce a large variety of sounds. These vary from simple pure tones to complex harmonics with amplitude and frequency modulations (**fager2004**).

Our image recognition approach makes direct use of the structures present in bird songs. The intuition is that spectrograms contain all the information necessary to distinguish one bird song from another, and that these may be segmented and catalogued such that they form the basis of truth for a statistical classification system. Such a system is essentially the automatic equivalent of manual spectrogram analysis performed by skilled ornithologists, and somewhat an analogue of natural pattern recognition done when observing sounds in the field.

Section 3.2.1 discusses the issue of selecting an appropriate granularity for use in cross-correlation.

3.1.3 Potential Alternative and Additional Features

There are a number of potentially useful features which have not been tested or measured directly in this project. These are not limited to image recognition approaches, however some are implicitly included in our approach.

Variations in amplitude

Amplitude variations are present in many bird vocalisations. These have been seen to be dynamic and dependant on the environment and social context (Brumm 2004) It is therefore likely that this feature is not significantly important for species recognition.

This feature is however included in the cross-correlation approach since amplitude is represented in spectrograms, although bearing inconsistencies due to both the dynamic nature of the amplitude variations, and the granularity of the extracted segments.

Statistical analysis

It is possible to extract information regarding the energy distribution in the spectrogram directly. Such information may be useful as the difference in song results in observable differences in fundamental frequencies and bandwidth. Additionally, it is possible that certain harmonics are characteristic to a limited number of unique species.

Although not likely to be useful for direct classification of species as many species likely share the same bandwidth and frequency distribution, these features may be helpful in narrowing down the set of possible species, which would save significant time by reducing the number of necessary cross-correlations.

Segment lengths and repetition frequencies

Segment statistics such as minimum, maximum and mean duration may be useful to help identify bird species. The silence between segments, as well as the repetition rate may also be a unique or indicative characteristic.

These features are implicitly included through image recognition, but only at the level of granularity afforded by the template extraction mechanism. The direct inclusion of extensive segment statistics has been shown to boost the classification accuracy to some significance (Lasseck 2013).

3.2 Template Extraction

Before template images can be extracted for cross-correlation, they must first be identified in the source spectrograms. If matching results are to be consistent, spectrograms must undergo noise reduction and segmentation. This section details the mechanisms developed for preprocessing spectrogram images and filtering segments so that only the relevant templates are included for extraction.

Each spectrogram is processed in sequence using functionality provided by the OpenCV (Bradski 2000) library, and immediately stored on disk. Parallelisation is possible here, but the computational cost of this stage is not large enough to have justified sparing development time on this aspect.

3.2.1 Spectrogram Preprocessing

Before any features can be cleanly identified, their pixel boundaries must be found. This is a non-trivial problem due to noise and recording artefacts that may be present in each spectrogram. The complexity of the problem is increased by their inconsistency.

Figure 3.1 shows an example of a noisy spectrogram. Notice how baseline and background noise makes it difficult to determine the exact boundary of some vocalisations.

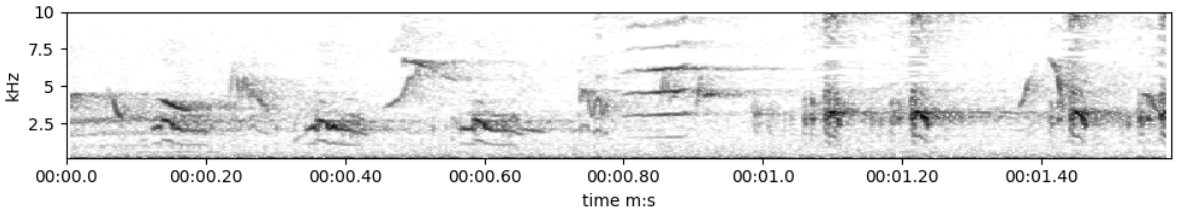


Figure 3.1: Noisy spectrogram of a segment of a Great Reed Warbler's song.

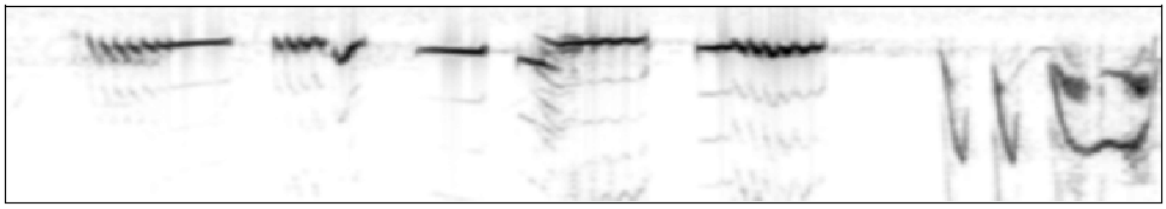
Even with the metadata-based quality prefiltering done when downloading recordings from Xeno-canto, noise levels remain at unsuitable levels for direct pixel isolation.

Removing noise and unwanted artefacts manually is precise, but tedious and impractical at this scale.. Because a high number of samples is required for our approach, we developed an automatic noise reduction stage. Standard computer vision techniques for noise reduction are used, as well as techniques for discrete object identification.

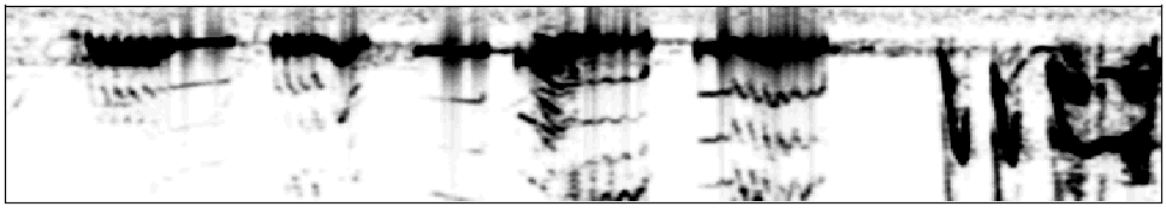
The mechanism

A filter is first applied to the spectrogram, which reduces the noise in the image by smoothing it just enough until granular noise is reduced sufficiently (Figure 3.2a). This is accomplished using a Gaussian filter with a 5x5 kernel and sigma of 0. Several other filtering methods were also tested, including a median filter, but these were not compared directly.

The image is then thresholded. (Figure 3.2b). Otsu's binarisation thresholding technique (Otsu 1979) was used to select an optimal threshold value. Because of pixel intensity inconsistencies throughout individual spectrograms, specifically in areas where noise is prominent, an adaptive thresholding algorithm was also tested. This does not appear to provide generally better results. Thresholding is performed twice, initially as a truncation method, then again



(a) Gaussian filter, $k = 5x5, \sigma = 0$



(b) Otsu's threshold truncation

(c) Otsu's threshold binarization

(d) Dilation followed by erosion, $k = 7x7$

(e) Closing, 3x3 ellipse

Figure 3.2: Preprocessing stages in order.

for binarisation (Figure 3.2c). This appears to perform better in observed cases than binarising immediately.

Further noise reduction is then performed using dilation and erosion (Figure 3.2d), which removes small segments and joins pixel groups which are in close proximity to each other. Notice that the order of operations is flipped here since we are working with an inverted spectrogram. A kernel size of 7x7 is used.

Most remaining holes are then filled using a closing morphology algorithm, using an ellipse of size 3x3 (Figure 3.2e).

These procedures transform the spectrogram into a binary image consisting of discrete pixel segments. This representation simplifies the extraction procedure as described in Section 3.2.2.

Granularity considerations

It is important to consider and evaluate the granularity to aim for when isolating sections of song. It is not clear if phrases perform better than syllables or elements for example. An evaluation should therefore be performed to determine the best granularity.

Comprehensively testing different granularities bears the weight of template matching for all new templates, which is extremely time consuming.

Further, achieving consistent granularity across all spectrograms is not a trivial task, and is certainly not possible if a single parameter set is used for all spectrograms.

A loose aim is therefore taken to extract the smallest possible non-singular vocalisations, as close as possible to a syllable. This of course does not always work, but the developed mechanism provides generally acceptable results in the end.

Quality and consistency

It is difficult to arrive at a single set of optimal parameters which work well across all spectrograms. An adaptive method is therefore suggested (but not implemented): Preprocessing parameters may be specified on a per-recording basis if prior knowledge is gathered for expected vocalisation/section counts, template dimensions, and frequency range. Since there is no known structured source of data for this, it is necessary to manually specify these on a per-species basis. Alternatively a fully automatic method is conceivable by feeding back the classification results with each granularity setting. This however would be incredibly time consuming on standard hardware, and may be sensitive to other factors in the classification pipeline.

Consistent quality becomes less of a concern as the number of samples used for training increases. It can be shown that as the sample count increases, the number of valid templates tends to increase. The number of noise also increases, however these tend not to inter-correlate as valid templates do, and are mostly handled well by less sensitive classifiers such as random forests (Robnik-Šikonja 2004) as long as there are not too many of them.

Quality is therefore moderated after preprocessing in the selection stage as described in Section 3.2.2.

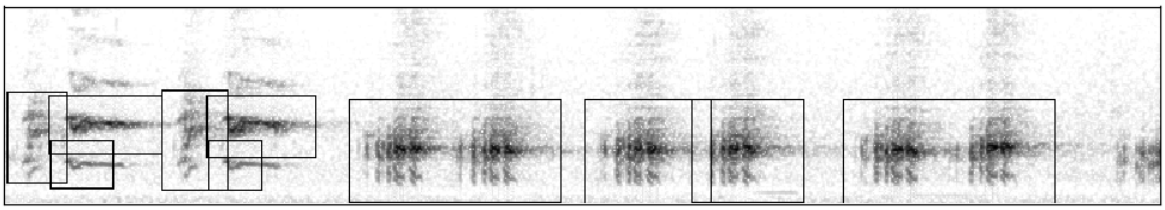


Figure 3.3: Example of selected templates with varying granularity

3.2.2 Basic Extraction and Selection Filtering

Extraction mechanism

A contour detector is ran on the now binary image, and the bounding boxes of each contiguous contours are computed. This boundng box is expanded in all directions by 10 pixels, and used to extract a template from the original spectrogram image. This forms a single template. The template is then blurred using a Gaussian filter with a sigma of 1.5.

Blurring the template allows us to reduce its size, which brings performance boosts to template matching. Blurring the template also improves it's generality, which makes matches more likely. Care must be taken not to over-generalise templates. The same operation is made to the spectrograms subject to cross-correlation mapping for consistency.

Selection Mechanism

Given optimal results the extracted segments represent parts of song at the desired granularity. This is often not the case however. Noise may persist after preprocessing, and new deformations may appear, such as incorrectly joined or incomplete segments.

Some effort is taken to reduce the number of undesirable templates after preprocessing. In addition to noise, undesirable templates include extremely specific or extremely generalised templates.

For example, if a sound is featured in a single instance of bird song, but not in any other recording of songs of that species, then it is an extreme outlier, which may occur due to a bird's individual expression or originate from external sources. It is non trivial to detect outliers during feature extraction.

Generalised, or weak templates are easier to detect. Such templates will have a very faint structure (low contrast), or a complete lack of structure. These may be analysed directly and ignored, as they are likely to correlate, although very weakly, to the majority of spectrograms irrespective of the species.

Although these rejections bring a theoretical boost to classifier accuracy and performance, they mainly benefit performance by reducing the number of cross-correlations computed.

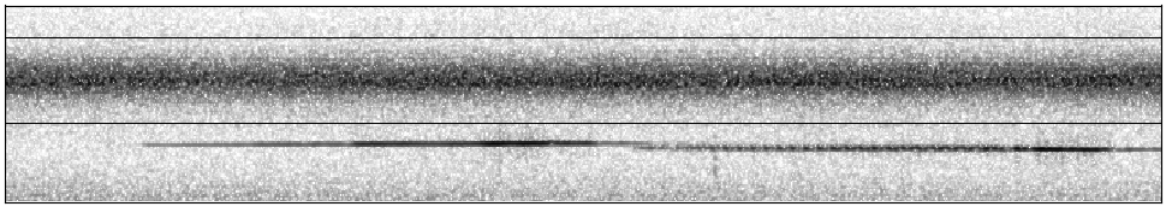


Figure 3.4: Spectrogram featuring rejected extremely large noise segment

Complete elimination of all undesirable features is impossible without also removing some desirable features. A balance must therefore be struck, in both preprocessing and selection stages. Performing selection manually is infeasible given the quantity of pixel segments. Selection is therefore done automatically by considering the dimensionality and pixel values of each template. Results are shown in Figure 3.6

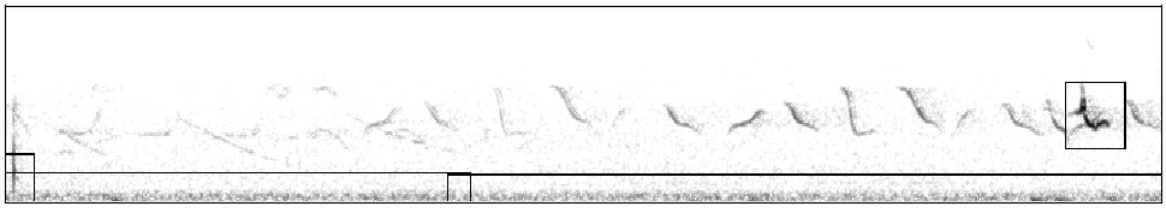
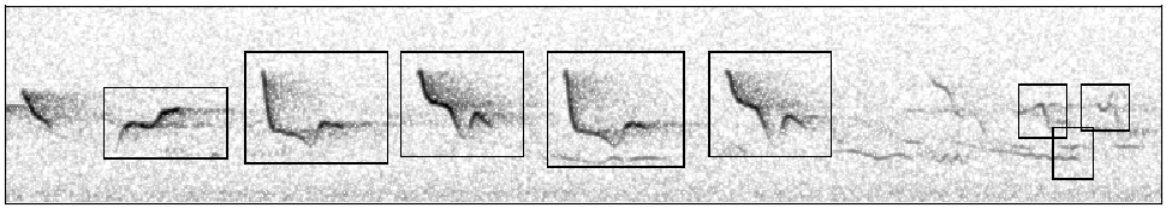


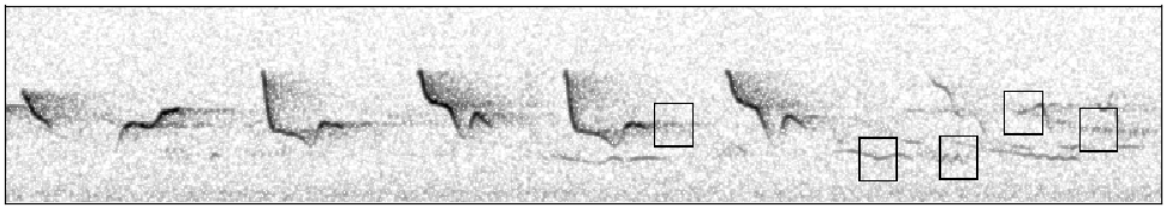
Figure 3.5: Spectrogram featuring erroneously rejected weak templates

Templates with the following properties are automatically rejected:

- **Area smaller than 50 pixels:** Templates with small dimensions are too small to be of any significance and are most likely noise or artefacts from preprocessing;
- **Area larger than 10000 pixels:** Templates with extremely large dimensions are most likely artefacts from preprocessing, high energy noise or continuous external noises. It is possible for sections of song to be merged together into a single pixel region due to preprocessing errors.
- **Maximum intensity lower than average intensity of the spectrogram:** Low maximum intensity is an indicator background or baseline noise. This filter eliminates some of these templates, however not very well. Local contrast may be a better property to test for, but this has not been explored.



(a)



(b)

Figure 3.6: A noisy spectrgoram showing accepted (a) and rejected (b) templates

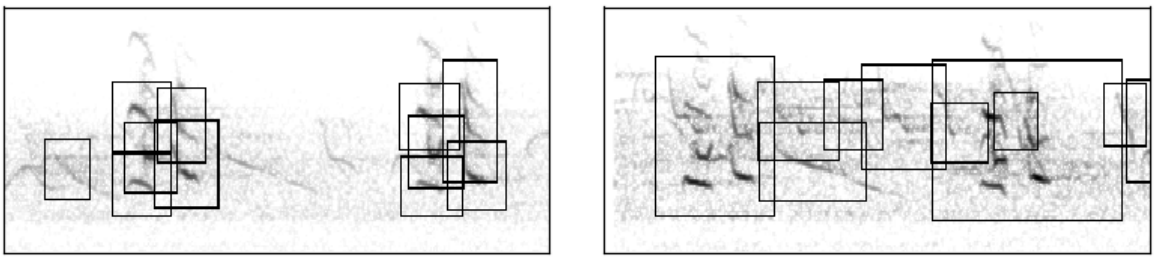
3.2.3 Advanced Template Elimination

It is desirable to reduce the template count further. This can be made possible by recognising correlations between candidate templates. This subsection outlines some speculative methods for selecting better templates.

Spatial inclusion

Imperfections in the preprocessing mechanism lead to erroneous pixel segments. Incorrect gaps, joins and other inconsistencies in structure persist or materialise. If ignored, these are extracted along with valid templates. If infrequent, the additional templates will factor less as an accuracy penalty, and more as a performance issue.

In many of these cases, one template's bounding box intersects or is contained entirely within another template's bounding box as shown in Figure 3.7a. These errors can be corrected by taking the bounding box of their unions. In contrast, Figure 3.7b shows that this does not always work, and may result in large, incorrect groupings.



(a) (b)

Figure 3.7: Example of erroneous segmentations correctable (a) and uncorrectable (b) by merging their bounding boxes

Inter-template correlation

As expected, templates will have some level of inter-correlation. This leads to a database of many hundreds or thousands of very similar templates. Merging these in some form has been considered but not approached. This would benefit performance dramatically, but it is most likely to cause a noticeable dip in accuracy. For this to succeed, the result of the merge must represent all related templates equally, and it is clear that this would not be as accurate as individual templates.

It can also be argued that templates with little to know inter-correlation may be independent anomalies, such as noise or external signals irrelevant to the subject species.

Species-specific template statistics

It may be possible to use information regarding average dimensionality and mean frequency information to determine the relevance likelihood of a particular template. Such metrics require an existing set of validated templates, which may be gathered by filtering a non discriminated set of templates by their measures importances.

3.3 Feature Engineering

With a collection of extracted templates from each spectrogram, feature vectors are constructed through template matching. This is done by taking the matching probabilities of cross-correlating all templates against samples of known species. These feature vectors are then used to train a

machine learning algorithm to classify new recordings from their spectrograms after template matching.

This section describes the mechanisms used to perform cross-correlation mapping for feature vector construction and a short analysis of the computational expense involved.

3.3.1 Batch Processing and Result Persistence

Once initial reductions are been made through the selection process as described in Section 3.2.2, template matching is performed in batched stages.

Batching

Further processing is batched in groups of four species, which are selected in random order without replacement. Each batch undergoes template matching, the results of which are stored on disk. Once a batch is completed it is merged with previous results by cross-template matching the new templates against the old spectrograms and the old templates against the new spectrograms. The new results are appended to the old feature vectors and stored on disk.

This is an effective method for dividing work into manageable chunks while storing intermediate results. The cost analysis in the following section justifies the need for batching.

Persistence

It is desireable to store intermediate and final results to disk for reloading on-demand, as these values are extremely expensive to compute. Storage is written using the Python `Pickle` module. Each result file is self-sufficient, containing all the necessary information to classify and manipulate feature sets:

- Ordered list of **feature vectors** for each sample;
- Ordered list of numeric **labels** for each sample;
- Ordered list of unique **ids** for each sample;
- Mapping from numeric labels to real labels;
- List of **used templates**, in order of appearance in feature vectors.

3.3.2 Template Matching

Template matching is a method for computing the similarity of an image within another. We use a normalised cross-correlation method for template matching. This mode was chosen over squared-difference and cosine-coefficient because it has been used with good results in similar spectrogram template matching tasks (Lasseck 2013). Normalisation is used to reduce the effect of background noise in the spectrogram images.

The result of cross-correlation is a 2-dimensional vector with dimensions equal to those of the target image, with floating point values ranging from 0 to 1. Greater values indicate a higher level of correlation for the template in that area. This can be visualised as a heat-map of the template's correlation.

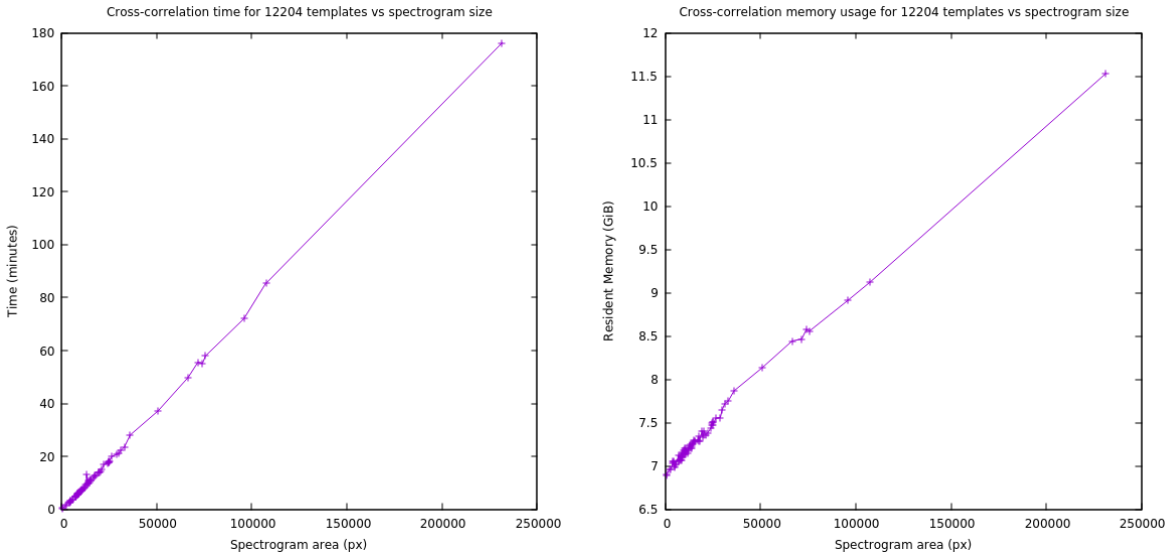
Section 4.4 discusses the mechanism and analysis of individual feature importances.

Implementation and cost

We use Open-CV's `matchTemplate` function with the normalised CCM method. Its implementation uses a Fourier analysis approach for improved performance over the traditional raw statistical methods. This works by first applying a FFT to the template and target images before performing the cross-correlation.

Even with such optimisations, template matching takes a considerable amount of time. Template matching takes on average 16 minutes per spectrogram, or 0.07 seconds per template, given a mean area of 21941 and 34 pixels for spectrograms and templates respectively. Considerably larger spectrograms occur often enough to significantly add to the costs, potentially in the order of hours. Considering the quantity of templates being analysed, the time required quickly amounts to days.

To gain insight into how the resource usage responds to the size of a spectrogram, a feature extraction run was performed for 80 spectrograms of varying sizes, with 12204 total templates. Spectrograms and templates were selected at random following the same selection and filtering procedures described in previous sections. Total time to completion was approximately 22 hours on the machine listed in Appendix B.



(a) (b)

Figure 3.8: Time (a) and resident memory (b) usages as pixel count increases for cross-correlation of a single spectrogram against 12204 templates

Aside from dimensionality reduction as described in Section 3.2.2, template matching is parallelised using Python's `multiprocessing` module. Spectrograms are split into even chunks equalling the number of usable CPUs on the machine, joining results upon completion. This cuts the cost down to approximately $1/4$ of the original time on our machine.

Additional optimisations can, but have not been implemented:

- Template matching can be confined to the general area at which the template was extracted. This is possible if the frequency bands, or image coordinates at which a template was extracted is stored for this purpose. If implemented, the search area for template

matching should be increased by some value to account for natural or artificial deviations.

- Open-CV's template matching function supports on-GPU calculations. Doing so would result in considerable gains in speed, however this requires an NVIDIA CUDA device.
- Open-CV uses FFT. Fast Normalised Cross Correlation (FNCC) is a faster alternative to FFT based methods of template matching (Briechle and Hanebeck 2001). Replacing calls to an implementation which uses FNCC may result in some performance gains, however an existing implementation has not been found, and writing one from scratch is costly without a guaranteed noticeable impact.

3.3.3 Feature Vector Construction

The template matching results are used to construct a feature vector for each recording. The feature vector of each sample therefore encodes the extent at which it's recording contains similar characteristics to each species' song. Feature vectors compose the training and test data for the machine learning stage described in Chapter 4.

The length of each feature vector is constant across all species and is equal to the number of extracted templates. The values of each element in the vector are derived from the template matching results by taking the maximum value from the cross-correlation mapping. The maxima intuitively gives us the highest matching probability for a template, but discards information of other possible matches. This extraction has shown promising results (Fodor 2013).

CLASSIFICATION AND EVALUATION

4.1 Classification

With feature vectors constructed, we select a machine learning algorithm to train and classify birdsongs, forming our model. This section discusses the considerations made when choosing an algorithm, and details our choice of algorithm and variations in its mechanism. Parameter tuning is also discussed is Section 4.3

4.1.1 Approaches to Classification

We are approaching this as a multi-class machine learning task, that is, the classification of a sample as a single label given a set of possible labels. A multi-label approach could also have been taken, identifying multiple labels. Given that most recordings appear to have a single dominant vocalising bird, we opt for the multi-class approach.

There exist many suitable machine learning algorithms. It is desirable to obtain a probabilistic output from which we can gauge the confidence of our classifier.

We may also wish to use an ensemble approach through binary relevance to isolate label features. Doing so may allow us to use a wider variety of verification metrics and to identify label-specific behaviours.

The primary consideration when selecting a classifier is of course the nature of the data being classified. In our case, we are working with a small sample set, but with a very high dimensionality. Feature values are continuous and normalised. Samples are labelled, therefore a supervised learning algorithm is appropriate. Good computational performance is desirable considering the high feature count. In addition, it is useful to have an efficient means of analysing the impact of individual templates, in our case measurable directly through feature importance.

There is no hard and fast rule to selecting an optimal classifier, therefore once an initial algorithm has been chosen, it may be compared to others through multi-run cross-validation, comparing accuracy results. (Bouckaert 2003).

4.1.2 Overview of the Random Forest Classifier

A random forest (Breiman 2001) is an ensemble of decision tree classifiers. Each tree is developed using randomly sampled training data. Because decision trees are capable of multi-class classification, forests are also support this natively.

The random forest algorithm is well suited for the given task: It is relatively performant, not particularly sensitive to bias or noise (Robnik-Šikonja (2004)), and it is possible to extract

individual feature importances directly without having to develop computationally expensive mechanisms. Random forests also exhibit good scalability because the algorithm can be easily parallelised (Breiman 2001).

It does not allow for online learning however since trees have to be recomputed, and so the addition of extra training samples, features and thus labels may become expensive for larger additions. This is somewhat counteracted by the short time required to train an entire forest on our data.

Although we have not done so, it is possible to employ an ensemble of binary random forest classifiers with probabilistic output. In such configuration each forest is responsible for classifying a single species, while considering only the features of that species. This allows for less expensive updates, as we can add and modify species without needing to rebuild all of the forests. Although this method ignores higher-order interactions between class features, it has been shown to exhibit good performance (Lasseck 2013).

Given this suitability, we have chosen to use a single multi-threaded multi-class random forest.

Extremely Randomised Trees

Extremely Randomised Trees (Geurts, Ernst, and Wehenkel 2006) is an algorithm similar to the traditional Random Forest, however the thresholds during splitting are randomised, instead of being computed for optimal performance. This has the benefit of being faster, with the drawback of being more sensitive to noisy features.

Using this classifier shows no major accuracy or performance differences. A speed increase is likely to be observed with many more trees.

4.1.3 Parameter Selection

The most impactful parameters for random forest performance are the number of trees and the maximum number of features usable at each split. Random forests are generally trained so that trees are fully grown to minimise bias, and then increasing the number of trees as much as necessary to minimise variance (Breiman 2001). Altering the remaining parameters mostly results is tree pruning, which may be desirable if over-fitting occurs.

- **Max features:** Defines the maximum number of features that a tree may use at each split. Increasing this value generally increases performance at the cost of diversity. Decreasing this value reduces variance at the increase of bias in individual trees. The ratio of good features to noise affects the choice of this parameter, as reducing the feature sample size decreases the chance of picking good features from the entire set.

- **Number of estimators:** Defines the maximum number of trees to build. Higher values are generally always better, being detrimental only to speed. Increasing the number of trees decreases variance, becoming less correlated as features are randomly selected.

  A random forest will eventually converge with the increase in tree count, from which the gains begin to diminish (Breiman (2001)).

- **Min samples per leaf node:** Defines the size of the leaf nodes. Smaller sizes leads to an increase in sensitivity to noise. Directly affects the depth of the trees.
- **Max depth:** Defines the maximum depth at which a tree may be built. If this value is set to `none` with the minimum samples to split set to 1, then trees will always be fully developed.
- **Min samples split:** Defines the minimum number of sample observations per node required before splitting on a feature.

The following parameters are set by default and used to gauge the effectiveness of tuning:

- Estimators: 10
- Max features: $sqrt(n)$
- Min samples to split: 2
- Min samples leaf: 1
- Max depth: `none`

Note that max features is a function of the number of features.

For comparison, the following parameters have found to be optimal through the tuning mechanism discussed in Section 4.3, and is used throughout evaluation in Sections 4.2 and 4.4:

- Estimators: 300
- Max features: $0.33 * n$
- Min samples to split: 2
- Min samples leaf: 1
- Max depth: `none`

4.2 Evaluation and Validation

To validate our model, several metrics are taken to measure its performance. Although random forests are less likely to over-fit, it can still occur, and is nonetheless important to detect.

It is equally important to correctly sample the data as to avoid bias. This section discusses first the framework around which the performance of our model is measured, the metrics used, and finally an analysis of the results.

Section 4.3 analyses the performance of the classifier with varying parameters, in order to find their optimal values.

4.2.1 Validation Strategy and Sample Space

In effort to reduce sensitivity to variance, all measurements are performed through a multi-run stratified 10-fold cross validation. This gives us a 90/10 train/test data split, and is repeated with a new random sample shuffling 10 times. Stratification ensures that there exists an equal class balance in the testing and training splits, so that all classes are represented equally.

This is especially important given the small number of representative samples for each class. Advanced methods exist in order to further improve our results, such as bootstrapping, but this has not been pursued directly. Random forests use bootstrap sampling internally however, where a portion of the data is not used for training. This produces a measurable result similar in effect to cross-validation, referred the OOB error, discussed in Section 4.2.2.

For each fold, only features from the selected training samples are used for training and validation.

A total of 3 batches have been computed and merged, giving 12 species in total. Selection was performed using the mechanisms outlined in Section 2.3, and feature vectors constructed as described in Section 3.3. Table 4.1 shows the selection of species including spectrogram and template counts per species. In total 4292760 cross correlation mappings were performed.

Table 4.1: Selected species with spectrogram and template counts

Species Label	Spectrograms	Templates
Common Blackbird	20	6132
Great Reed Warbler	20	5111
Common Rosefinch	20	3191
Common Cuckoo	20	2909
Common Chiffchaff	20	2886
European Greenfinch	20	2759
Pale-breasted Spinetail	20	2501
Ortolan Bunting	20	2362
Common Reed Bunting	20	2233
Chestnut-breasted Wren	20	2203
Corn Bunting	20	2082
Rufous-browed Peppershrike	20	1404
Total	240	35773

4.2.2 Evaluation Metrics

Each of the five following evaluation metrics are computed in a fold. They are then stored and averaged at the end of the cross validation.

- **Accuracy** Is the ability of the classifier to correctly label samples.

$$\frac{TP+TN}{P+N} \tag{4.1}$$

- **Precision**, Sometimes referred to as the positive predictive value, is the ability of the classifier to label all predicted values correctly.

$$\frac{TP}{TP+FP} \tag{4.2}$$

- **Recall**, Sometimes referred to as sensitivity, is the ability of the classifier to label all positive samples correctly.

$$\frac{TP}{P} \tag{4.3}$$

- **F-beta score** Is the weighted harmonic mean of the precision and recall, where higher values indicate better performance. This makes it simpler to compare the performance of different parameter sets.

$$\frac{(1+\beta^2)TP}{(1+\beta^2)TP+\beta^2 FN+FP} \tag{4.4}$$

We consider precision and recall as equally important, in which case the beta value is set to 1:

$$\frac{2TP}{2TP+FN+FP} \tag{4.5}$$

- **OOB error rate** Is the out-of-bag (OOB) samples are those which were not selected during the construction of a specific tree. Given the most common label from classifying a sample with its respective tree, the proportion of incorrect classifications is averaged over all OOB samples is taken as the OOB error rate on the training data.

The Scikit Learn `precision_recall_fscore_support` function was used to compute these metrics.

Because this is a multi-class classification task, each metric is computed on a per-label basis. The average is then computed using "macro" averaging, in which the mean of each score is computed with equal weighting for all labels. This is appropriate considering we use stratified cross-validation, and each species is considered equally important to all the others.

Results

Results are promising, good performance has been achieved using the initial parameters, with observed improvements after tuning as is shown in Table 4.2.

Table 4.2: Comparison of accuracies using default and tuned parameters

	Metric Scores	
Metric	Default Params.	Tuned Params.
Accuracy	79.6 (7.9)	88.7 (6.2)
F1 Score	78.0 (8.6)	87.9 (6.7)
Recall	79.6 (7.9)	88.7 (6.2)
Precision	82.0 (9.1)	90.9 (6.5)
OOB Error	31.5 (2.7)	12.6 (1.4)

* Values are shown as percentages.
* Standard deviations are shown in parentheses.

4.2.3 Confusion Matrix

A confusion matrix plots the rate at which each label is predicted as any other label, where the rows represent the true label, and columns represent the classifier's predictions.

We construct a confusion matrix to evaluate the performance of the classifier and the features used to discriminate between species. In this case the matrix allows us to visualise the performance distribution per class, and give some insight into how the classifier responds to the selected features.

Figure 4.1 shows the resulting confusion matrix from classifying the selected data with the optimal parameters found through tuning, averaged over the 10-run 10-fold cross validation. Row-wise normalisation was performed.

The confusion matrix shows good classification performance for most species. A high (90%+) true-positive rate is observed for half of the species.

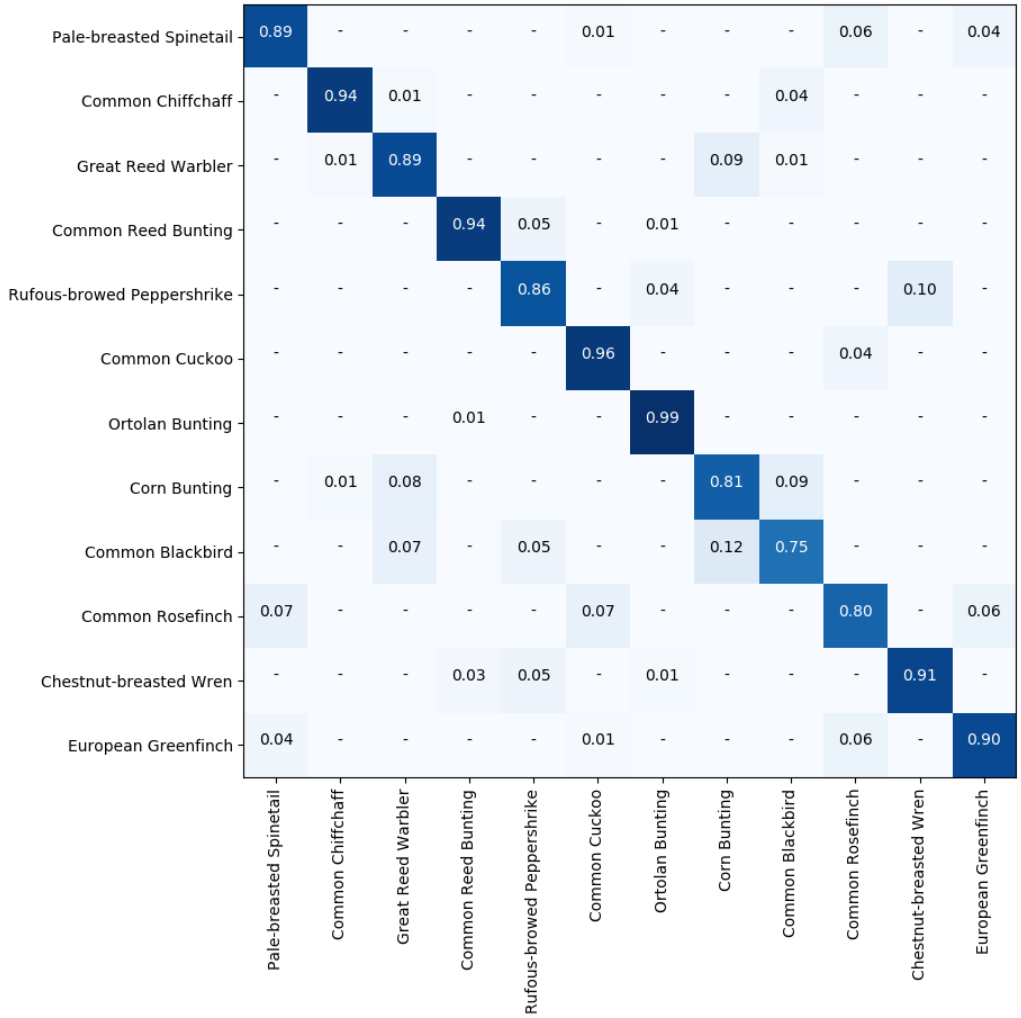


Figure 4.1: Row-Normalised Confusion Matrix

We see that the Common Blackbird is the worst performing classification with a 24% false negative rate, and often mistaken for a Corn Bunting. The inverse is also true to a lesser extent. This suggests weakness in the precision of Common Blackbird or Corn Bunting templates.

Aside from computing the metrics shown in Section 4.2.2, a confusion matrix can only offer a speculative insight into template performance. Nonetheless, it offers a starting point into further template analysis as discussed in Section 4.4.2.

4.3 Tuning

There exists no default set of parameters or rule set which will guarantee optimal performance. The only conclusive method to find optimal parameter values is to evaluate each by comparison. This can quickly lead to over-fitting if the results are misinterpreted or incorrectly measured.

The primary cause of over-fitting is tuning on the entire data-set. To reduce the chance of this, data-sets should be split into completely independent development and evaluation sets, on top of cross validation. Although we don't do this here, we do use 10-times 10-fold cross validation as described in Section 4.2.

The OOB error rate gives us an estimate of real-world performance, and we may gain some insight into whether or not we are actually gaining performance in other metrics or over-fitting by comparing this with the OOB error.

In order to confidently produce unbiased statistics, and ensure that tuning is not guided to over-fitting, validation curves can be computed. These compare the performance of the classifier on separate development and validation data-sets. This has not been done because we have not collected sufficient data to be able to split it further on top of the cross validation used.

We could also plot the learning rate, which shows the performance of the classifier with a varying number of samples, allowing us to gauge whether adding more samples will improve classification scores or not. This has also not been done.

4.3.1 Gridsearch

The most basic form of parameter search is gridsearch. A set of influential parameters are chosen, as well as a range of discrete values to test. An exhaustive enumeration of all possible parameter value combinations is then run.

We test each combination using the usual evaluation mechanism with the following search space:

- Number of Estimators: 10, 35, 50, 75, 90, 100, 110, 250, 300, 500, 5000, 10000
- Max features: $sqrt(n)$, $log2(n)$, $0.33 * n$

The effect of each parameter is described in Section 4.1.3. We have not tested other parameters due to the time cost of doing so.

Results

The OOB error rate (Figure 4.2a) shows that the random forest is predicted to achieve peak performance with $0.33 * n.features$ at around 300 estimators, rising outside the stable range of $]150, 500[$. The error rate plateaus at about 500 estimators for all max feature functions.

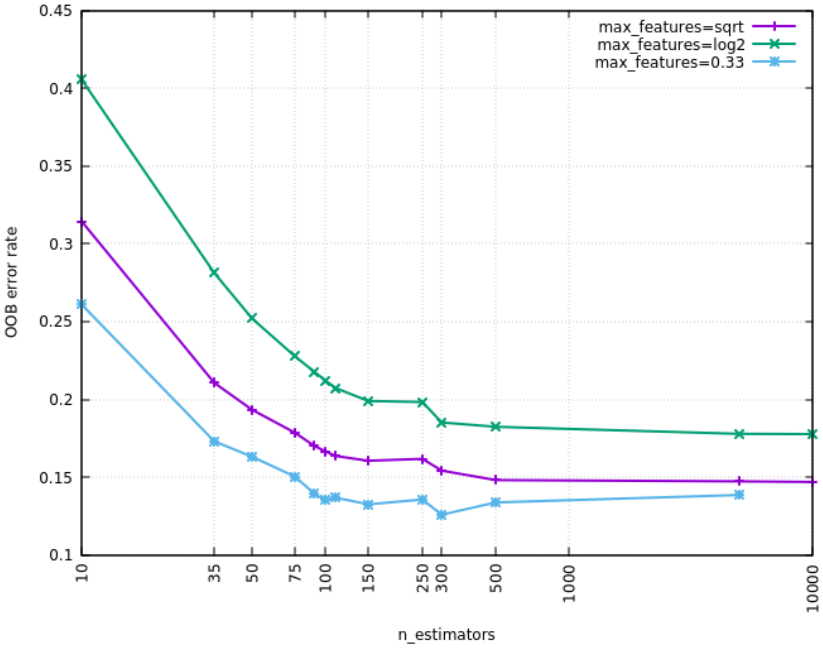
Accuracy (Figure 4.2b) and F1-score (Figure 4.2c) show that peak performance is achieved somewhere between $[75, 300[$ estimators for all max feature functions, peaking at 89% accuracy for $0.33 * n$. This correlates with the global minimum OOB error at 300 estimators.

Both metrics quickly drop somewhere between $[300, 500]$, where the random forest is most likely beginning to over-fit and lose generalisation. After this point, as the number of trees increases, we observe over-fitting as accuracy, F1-score, and the OOB error rate increases.

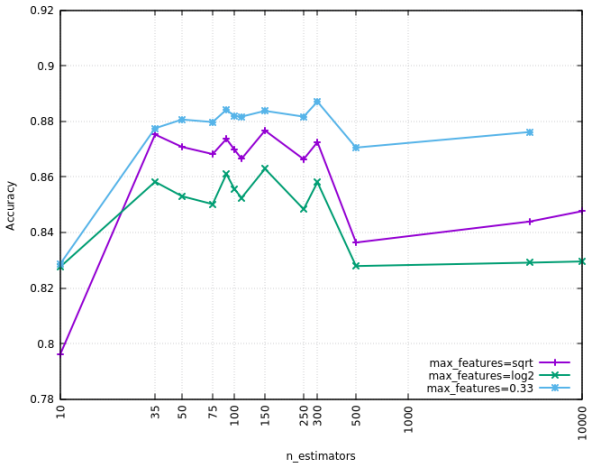
Because the OOB error begins to stabilise at 100, it is likely that peak real-world performance will lie within $[100, 300[$ for most max feature functions.

Performance improvements are generally observed with the increase in max features. The linear max features function shows a smoother response and better performance overall.

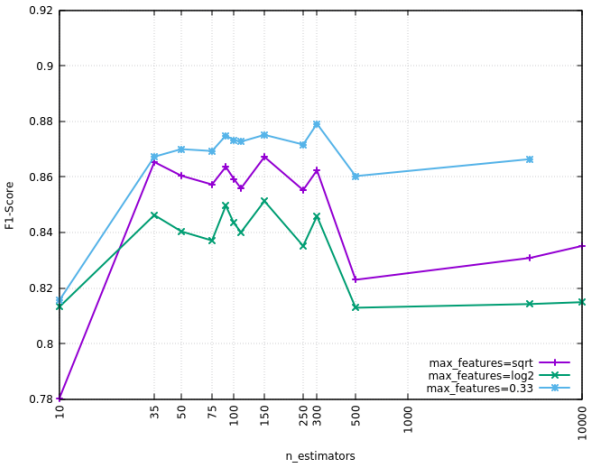
Without a completely independent validation data set it is impossible to perform an evaluation which is truly representative of real-world performance.



(a) OOB error



(b) Accuracy



(c) F1-Score

Figure 4.2: Metrics over n. estimators for each max feature function

4.4 Feature Importance

Because our feature vector is directly correlated with the templates of each species, it is possible to determine their impact and quality by computing feature importances from our classifier. This allows us to reduce the number of templates involved in classifying a given sample, significantly cutting down the cost of template matching per species. Feature importances also help us analyse the preprocessing performance in detail, as well as view inter-species correlations in cases where there exists a high level of confusion.

4.4.1 In Random Forests

The two most common methods for retrieving feature importance from a random are detailed in this section.

Mean decrease accuracy

A simple method for measuring the impact of each feature is through the mean decrease in accuracy. With this method, the impact of removing each feature is measured one-by-one. This method may be sensitive to the random nature of the classifier, so multiple runs may be required to reduce variance. This can be prohibitively expensive.

Mean decrease impurity

Because tree nodes in a random forest correlate directly with a specific feature, it is possible to measure or estimate the importance of each feature by determining the probability of a node in a tree being traversed over the number of nodes in the forest. This is known as the mean decrease impurity, or Gini importance, which is what we use in Sckikit's random forest classifier implementation.

4.4.2 Results and Analysis

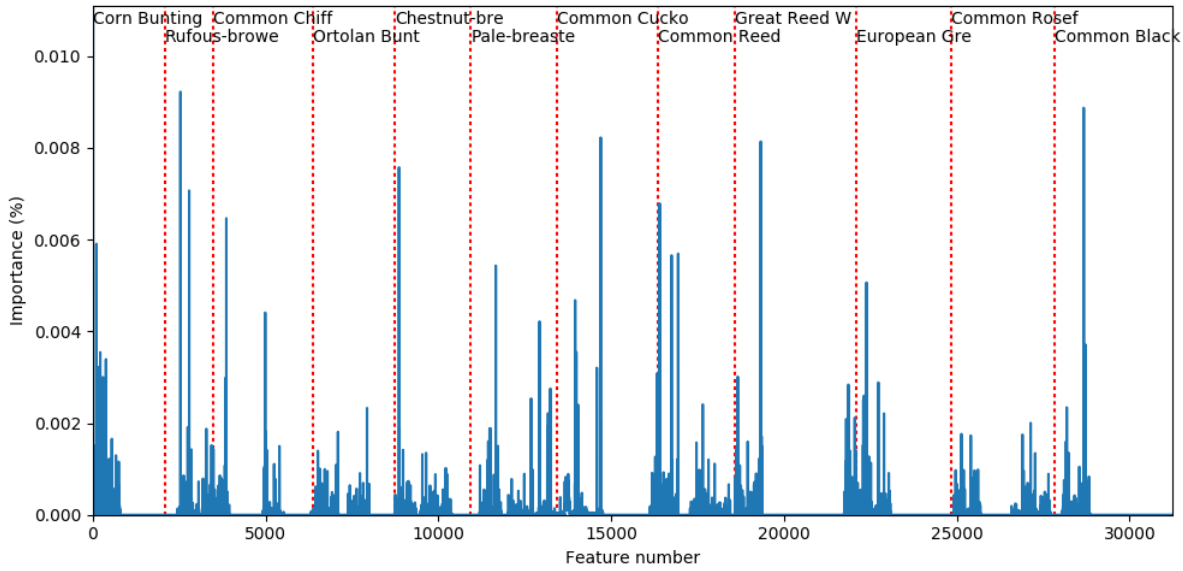
Measuring feature importances exposes 43% (15343 out of 35773) templates as completely irrelevant for classification within our data-set (Figure 4.3a). Removing these features shows no reduction in accuracy, and a significant speedup during template matching is potentially gained.

There exist 35773 templates distributed across 20 spectrograms for each of the 12 species selected. When classifying a new sample, its spectrogram is be cross-correlated with each of the templates. This operation is extremely expensive, taking up to 70 hours, or approximately 5.8 hours per sample, 0.6 seconds per template on average.

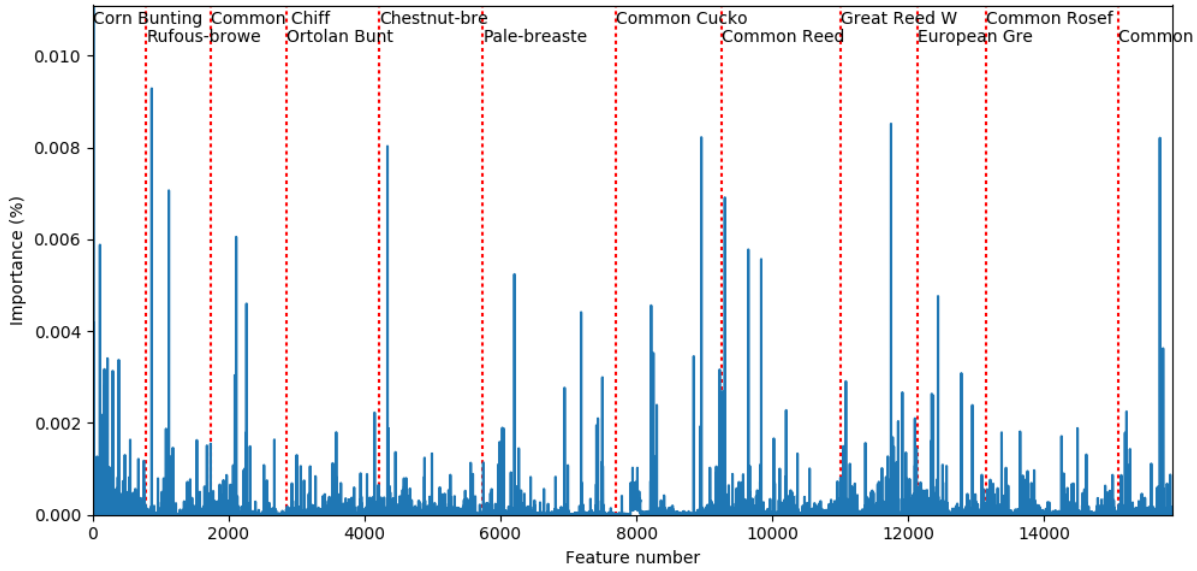
The aim is therefore to reduce the feature count as much as possible while retaining the highest possible accuracy by determining an appropriate importance cutoff. Removing features and reevaluating the classifier takes no significant time in contrast to template extraction and may therefore be done by iteratively removing and checking classifier accuracy, adjusting the importance cutoff accordingly. This course only reduces the time required to classify new samples. We have not done this.

Feature reduction is beneficial to feature extraction, specifically when merging the results of two batches. Templates with importance scores equalling or approaching zero can be safely removed from the set before merging with another batch, saving a considerable amount of time. Although we have not done this in practice the estimated amount of time saved is in the order of 2 and a half hours per sample, assuming a consistent 43% template reduction rate.

Because we are using a single algorithm to classify the entire label set, it is difficult to determine individual label-specific importances of each feature. The importance figures extracted here represent the overall impact of each template in classifying *all* labels.



(a)



(b)

Figure 4.3: Visualisation of feature importances before (a) and after (b) reduction

Survey

To gain some insight into the preferable structure of a template, we have extracted the top ten most influential templates from the best (Acc. 99%) and worst performing (Acc. 75%) label. From this selection the Common Blackbird's templates (Figure 4.5 show slightly less inter-correlation than those of the Ortolan Bunting (Figure 4.4). It is somewhat surprising that some of these features, although appearing to be overly generic, are the most influential.

It may be possible that these high importance features score well because they fit only to very specific samples, or perhaps they are excellent discriminators for other species. It is difficult to make further analysis without label-specific importances and measuring the effect

of modifying the feature vector further.

Figure 4.4: Top 5 most influential features from the Ortolan Bunting

Figure 4.5: Top 5 most influential features from the Common Blackbird

CONCLUSIONS

5.1 Summary

This report describes the solution developed for automatic birdsong recognition. Existing approaches to relevant problems were used as a starting point. The approach reduces the initial problem to image recognition using spectrograms of the 2211 field recordings sourced from Xeno-canto. Image processing techniques were investigated to maximise template quality. A random forest classifier was selected to classify 240 of the samples using results from cross-correlation. After tuning, the model was evaluated using 10-times 10-fold cross validation, achieving a peak accuracy of 89%, and F1 score of 88%. Finally, feature importances were analysed to rank templates and perform further feature reductions to improve scalability.

Several speculative improvements were discussed throughout the report.

5.2 Reflection

An open-ended project such as this is highly exploratory in nature. Regardless of the approach taken, it is likely to involve on many facets of computer science. Our approach focuses on computer vision and machine learning, touching on performance computing with (relatively) large amounts of data.

Although good results were achieved, these could have been improved. Model performance is highly dependent on the features used, and this area lacks in sophistication and validation. Although we learned about a few computer vision techniques for this, we could have increased the breadth and depth of our investigations in this area, specifically for template evaluation.

We are confident that the choice of classifier was a good one. A lot was learned about machine learning practices and validation methodologies, and although there exists a lot of contradictory findings and advice, it is the general consensus that with whatever application, it is the data that shapes the approach and the only conclusive evaluation to be made is directly based on this.

Given more time, other classifiers may be evaluated as well, however validation was limited immensely by the lack of data processed data. A total of 20 samples per label was not enough to confidently measure the performance of the classifier. Given more data, a further separation between development and validation data would have been possible, leading to a more representative evaluation.

The lack of data stems from the immense processing power required to perform over 4,000,000 cross-correlations. This makes it extremely expensive to test new template extrac-

tions. With this in mind it would have been justifiable to seek external computing resources early on, and to spend more time on optimisations. Also, Python was not the right choice in terms of performance.

On the topic of software engineering, recommended practices were not observed. The exploratory nature of the project has lead to many quick iterations and hacks in the software. Some effort was expended to restructure large portions of the code to make it more robust to future changes, however this quickly deteriorated. This was due to the fact that no strict requirements were produced for the program's functionality, as it was intended to be a test-bed for the explored methods. There exists a trade-off between developing the framework and developing the solution.

We have learned instead that simplicity is key: equal performance could have been achieved by splitting the program into discrete artefacts, and transferring data through files instead of in-memory. This pattern ensures separation of concern, and code alterations would not influence the operation of irrelevant parts of the code.

5.3 Future Work

The approach taken here shows great promise for further development.

There is much work to be done for preprocessing. Improving this is non trivial, but there are many possible starting points (Sections 3.2.1 and 3.2.3).

Once a healthy number of samples have been processed, of least 30 species, with 50 spectrograms each as a starting point, we can concentrate on improving the machine learning and feature extraction mechanisms.

Before the method can be comfortably scaled, performance optimisations must be considered. Immediately obvious improvements are migrating to a more performant language, such as C++ or Java, and utilising GPU acceleration for any heavy computations. A machine with more notable specifications would be advantageous, but a distributed approach would be better. Many of the mechanisms used here can easily be parallelised, and therefore distributed.

Bibliography

(N.d.). http://www.ffmpeg.org.

All About Birds (n.d.). https://www.allaboutbirds.org/do-bird-songs-have-frequencies-higher-than-humans-can-hear/.

Bouckaert, Remco R. (2003). "Choosing Between Two Learning Algorithms Based on Calibrated Tests." In: *ICML*. Ed. by Tom Fawcett and Nina Mishra. AAAI Press, pp. 51–58. ISBN: 1-57735-189-4. URL: http://dblp.uni-trier.de/db/conf/icml/icml2003.html#Bouckaert03.

Bradski, G. (2000). "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools*.

Breiman, Leo (2001). "Random Forests". In: *Mach. Learn.* 45.1, pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: http://dx.doi.org/10.1023/A:1010933404324.

Briechle, Kai and Uwe D Hanebeck (2001). "Template matching using fast normalized cross correlation". In: *Aerospace/Defense Sensing, Simulation, and Controls*. International Society for Optics and Photonics, pp. 95–102.

Brumm, Henrik (2004). "Causes and consequences of song amplitude adjustment in a territorial bird: a case study in nightingales". In: *Anais da Academia Brasileira de Ciências* 76.2, pp. 289–295. DOI: 10.1590/s0001-37652004000200017.

Catchpole, Clive and 1942- Slater P. J. B. (Peter James Bramwell) (1995). *Bird song : biological themes and variations*. English. Includes bibliographical references (p. [221]-240) and index. Cambridge [England] : New York, NY, USA : Cambridge University Press. ISBN: 0521417996 (hardback).

Chirpomatic (n.d.). http://www.chirpomatic.com. Accessed: 2017-04-29.

Fodor, Gábor (2013). "The Ninth Annual MLSP Competition: First Place". In:

Geurts, Pierre, Damien Ernst, and Louis Wehenkel (2006). "Extremely randomized trees". In: *Machine learning* 63.1, pp. 3–42.

Goëau, Hervé et al. (2014). "LifeCLEF bird identification task 2014". In: *CLEF2014*.

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

Kaggle (n.d.). *MLSP 2013 Bird Classification Challenge*. https://www.kaggle.com/c/mlsp-2013-birds. Accessed: 2016-06-14.

Kroodsma, Donald (1983). *Acoustic Communication in Birds, Vol. 2: Song Learning and Its Consequences*.

Lasseck, Mario (2013). "Bird Song Classification in Field Recordings: Winning Solution for NIPS4B 2013 Competition". In:

— (2014). "Large-scale Identification of Birds in Audio Recordings." In:

Otsu, N. (1979). "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1, pp. 62–66. ISSN: 0018-9472. DOI: 10.1109/TSMC.1979.4310076.

Palmer, Jake (2016). “Bird Song Recognition”. 3rd year project report. School of Computer Science, University of Manchester.

Podos, Jeffrey and Paige S. Warren (2007). “The Evolution of Geographic Variation in Birdsong”. In: *Advances In The Study Of Behaviour* 37, pp. 403–458.

Robnik-Šikonja, Marko (2004). “Improving Random Forests”. In: *Machine Learning: ECML 2004: 15th European Conference on Machine Learning, Pisa, Italy, September 20-24, 2004. Proceedings*. Ed. by Jean-François Boulicaut et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 359–370. ISBN: 978-3-540-30115-8. DOI: 10.1007/978-3-540-30115-8_34. URL: http://dx.doi.org/10.1007/978-3-540-30115-8_34.

Warblr (n.d.). http://warblr.net. Accessed: 2017-04-29.

Xeno Canto (n.d.). http://www.xeno-canto.org. Accessed: 2016-06-14.

Appendices

SAMPLE AND TEMPLATE COUNTS PER SPECIES

Species	Spectrograms	Templates
Common Blackbird	101	38818
Eurasian Blackcap	99	42274
Eurasian Wren	94	17431
Common Chiffchaff	66	10273
Grey-breasted Wood Wren	66	5603
Spotted Towhee	53	3583
Garden Warbler	48	18258
Eurasian Blue Tit	44	4228
Common Whitethroat	42	9858
Great Reed Warbler	40	12762
Rufous-browed Peppershrike	39	3491
River Warbler	38	11291
Yellowhammer	38	6442
White-breasted Wood Wren	38	4123
Eurasian Reed Warbler	37	30421
Common Grasshopper Warbler	36	3035
Corn Bunting	32	3177
Common Reed Bunting	31	2850
Northern Mockingbird	29	8333
Common Cuckoo	29	4854
Common Redstart	27	10774
Common Rosefinch	27	3733
Ortolan Bunting	24	2686
Chestnut-breasted Wren	24	2524
Western Meadowlark	23	8256
Pale-breasted Spinetail	22	2717
White-throated Toucan	21	6814
European Greenfinch	21	2844
Pin-striped Tit-Babbler	20	2575
Ferruginous Pygmy Owl	18	5000
American Robin	18	3515
Barred Antshrike	16	2478
Black-striped Sparrow	15	2510
Lesser Shortwing	15	1656
Green-winged Saltator	15	1644

Black-faced Antbird	15	1514
Pale-breasted Thrush	14	2955
Plain-tailed Wren	14	2847
Channel-billed Toucan	14	1619
Plumbeous Vireo	14	1094
Common Potoo	13	3690
Grace's Warbler	13	2069
Swainson's Thrush	13	2044
Rufous-capped Warbler	13	1692
Greenish Warbler	13	1668
White-breasted Tapaculo	13	1525
Tawny Owl	13	1133
Loggerhead Shrike	12	3281
Undulated Tinamou	11	4184
Collared Owlet	11	3642
Great-tailed Grackle	11	2424
Japanese Bush Warbler	11	1628
Giant Antshrike	11	605
Rattling Cisticola	10	0
Bell's Vireo	10	0
White-throated Sparrow	10	0
Scrub Greenlet	10	0
Olive Warbler	10	0
Laughing Falcon	10	0
Red-winged Blackbird	9	0
Common Hawk-Cuckoo	9	0
Yellow-browed Sparrow	9	0
Eastern Meadowlark	9	0
Slaty-breasted Wood Rail	9	0
Sumichrast's Wren	9	0
Plain-crowned Spinetail	9	0
Yellow-eyed Junco	9	0
Spotted Nightingale-Thrush	9	0
Japanese White-eye	9	0
Painted Bunting	8	0
Southern Yellowthroat	8	0
Great Tinamou	8	0
Sooty Antbird	8	0
Styan's Grasshopper Warbler	8	0
Yellow-throated Vireo	7	0
Connecticut Warbler	7	0
Hooded Siskin	7	0
Fire-tufted Barbet	7	0
Duida Woodcreeper	7	0
Gartered Trogon	7	0
Eurasian Treecreeper	7	0
Buff-browed Foliage-gleaner	7	0

Eurasian Golden Oriole	7	0
White-bellied Antpitta	7	0
Grey Antwren	7	0
Southern Boubou	7	0
Western Orphean Warbler	6	55
Bay Wren	6	10
Recurve-billed Bushbird	6	0
Golden-crowned Sparrow	6	0
Splendid Sunbird	6	0
Moustached Wren	6	0
Collared Antshrike	6	0
Violaceous Euphonia	6	0
Siberian Blue Robin	6	0
Henna-hooded Foliage-gleaner	6	0
Baillon's Crake	6	0
Paradise Jacamar	6	0
Rufous-vented Tapaculo	6	0
Ashy-headed Greenlet	6	0
Spotless Starling	6	0
Eastern Woodhaunter	6	0
White-throated Screech Owl	6	0
Lineated Foliage-gleaner	6	0
Orange-breasted Bushshrike	5	10
Yellow Grosbeak	5	0
Tanager Finch	5	0
Foothill Schiffornis	5	0
Singing Quail	5	0
Esmeraldas Antbird	5	0
Spectacled Owl	5	0
Dark-capped Bulbul	5	0
Marico Sunbird	5	0
Yellow-bellied Elaenia	5	0
Buff-cheeked Greenlet	5	0
Audubon's Oriole	5	0
Curve-billed Scythebill	5	0
Wedge-billed Woodcreeper	4	341
California Quail	4	126
White-throated Toucanet	4	0
Red-fan Parrot	4	0
Black-cowled Saltator	4	0
Western Wood Pewee	4	0
Variegated Antpitta	4	0
Zigzag Heron	4	0
Fan-tailed Cuckoo	4	0
Greater Thornbird	4	0
Blue-chested Hummingbird	4	0
Green-backed Sparrow	4	0

Varied Bunting	4	0
Ferruginous Partridge	4	0
White-vented Violetear	4	0
Golden-rumped Euphonia	3	379
White-fronted Honeyeater	3	336
White-necked Puffbird	3	82
White-spectacled Warbler	3	53
White-shouldered Antbird	3	0
Rufous-tailed Hummingbird	3	0
Blue-winged Pitta	3	0
Jerdon's Leafbird	3	0
Olive Thrush	3	0
Azure Jay	3	0
Johannes's Tody-Tyrant	3	0
Short-billed Leaftosser	3	0
Japanese Quail	3	0
Chestnut-bellied Thrush	3	0
Crested Lark	3	0
Emerald Toucanet	3	0
Greater Hoopoe-Lark	3	0
Chestnut-bellied Nuthatch	3	0
Forest Elaenia	3	0
Long-billed Pipit	3	0
Moustached Puffbird	3	0
Choco Tapaculo	3	0
Wedge-tailed Grass Finch	3	0
Pearly Antshrike	2	173
Little Cuckoo-Dove	2	139
Bahama Oriole	2	109
White-eared Brown Dove	2	97
Plush-crested Jay	2	96
Grey-throated Warbler	2	92
White-naped Jay	2	56
Pririt Batis	2	35
Black-goggled Tanager	2	33
Scarlet-rumped Trogon	2	5
Arafura Fantail	2	0
Yellow-crowned Elaenia	2	0
Bronzy Jacamar	2	0
Philippine Coucal	2	0
Flammulated Bamboo Tyrant	2	0
Russet Nightingale-Thrush	2	0
Hepatic Tanager	2	0
Solitary Tinamou	2	0
Jerdon's Nightjar	2	0
Cuban Vireo	2	0
Streak-breasted Treehunter	1	1118

Emei Shan Liocichla	1	350
Crimson-backed Tanager	1	261
Curve-winged Sabrewing	1	236
Eastern Nicator	1	169
Yellow-breasted Boatbill	1	160
Tiny Tyrant-Manakin	1	149
Mountain Wren-Babbler	1	119
Winifred's Warbler	1	116
Negros Scops Owl	1	109
Green-chinned Euphonia	1	98
Sombre Rock Chat	1	83
Cherrie's Antwren	1	39
Sand Martin	1	17
Yellow-breasted Pipit	1	13

MACHINE SPECIFICATIONS

CPU Model	Intel(R) Core(TM) i5-5300U CPU
CPU Architecture	x86_64
CPU MHz	2700
CPU Cores	4
RAM Type	DDR3
RAM MHz	1600
RAM GB	16
OS	Arch Linux 4.10.4

LIBRARY MODULES LISTING

Python		2.7
	Numpy	1.12.1
	SciPy	0.19.0
	Scikit-Learn	0.18.1
	Matplotlib	2.0.0
	OpenCV	3.2.0
	LXML	3.7.2
	Requests	2.13.0
Misc		
	FFmpeg	3.2.4