Milestone 5: Classifying Bird Song in Europe

Group 4: Sandra Forro, Imran Naskani, Erin Rebholz

Table of Contents

0. Imports and Initialization

[ ] ↳ *8 cells hidden*

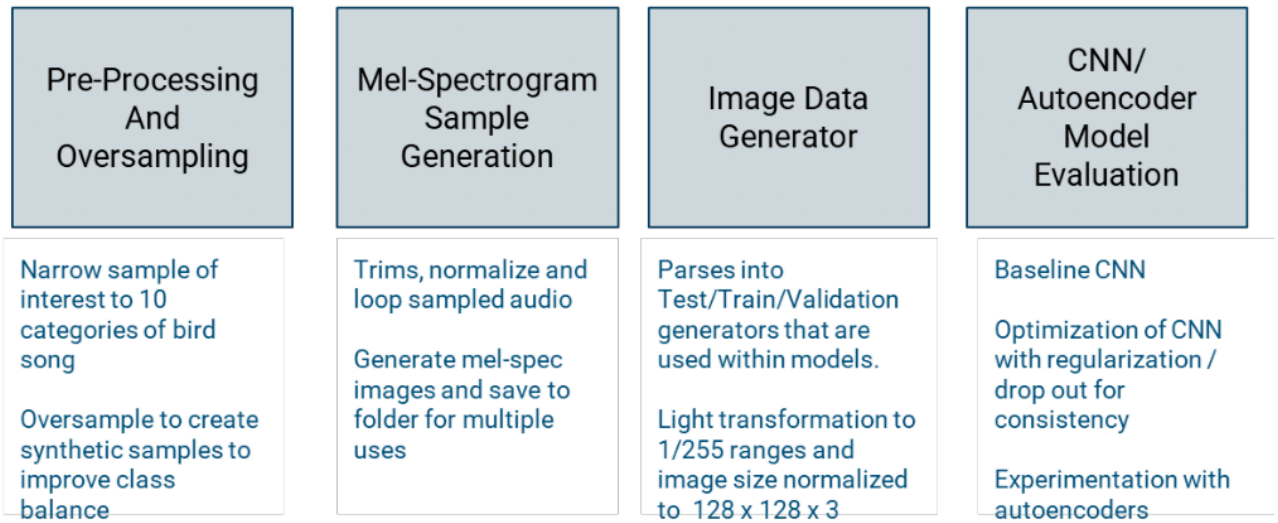
1. Problem Statement

The aim of this project is to **Classify Top Bird Songs in Europe**. 2,901 15 second song recordings of the top 10 European bird species were subsetted from the Xeno-canto (XC), the Foundation for Nature Sounds in the Netherlands. The number of samples are approximately 200 per bird type, with small upward or downward deviations. This project is inspired by the Nature and Biodiversity Conservation Union (NABU)'s paid bird song detection app, which is being foregone by users for the slower, manual process of sending recordings to a bird expert via WhatsApp instead. Success is defined as surpassing the 85% accuracy of the app.

The analysis below follows the following general process:

```
img = mpimg.imread(os.path.join(base_path,'figures/eval_pipe.png'))
plt.figure(figsize=(14, 10))
plt.axis('off')
imgplot = plt.imshow(img)
plt.show()
```

Evaluation Pipeline



2. Dataset Background and Sample Selection

2.1 Data Acquisition

The dataset is acquired from Global Biodiversity Information Facility's website: https://www.gbif.org/dataset/b1047888-ae52-4179-9dd5-5448ea342a24

This data shared by GBIF is infact a subset of the entire collection of the Bird sound collection of Xeno-canto (XC), the Foundation for Nature Sounds in the Netherlands, available at https://xeno-canto.org/?gid=1. There are more than 700k occurences of bird sound recordings, verified and shared by Xeno-canto. This particular subset only comprises recordings whose recordists have given explicit sharing permission. Though recordings with a 'questioned' status (to indicate quality review) are included in the data set, the corresponding audio file is not available. We will remove those accordingly.

Data acquisition is done in two steps. GBIF has publised several text files which contain information about the bird species/subspecies/generic names, the location and time of occurence, and the download link of their audio recordings (amongst others). In the first step, we acquired these text files from the GBIF's website and then downloaded the audios using the links provided in the text files.

However, the total data is more than 700k, which is huge and cannot be downloaded in bulk since each audio file has its separate downlaod link, making the download sequential and more time consuming. ***Therefore, in the interest of scalability of this project, we have decided to use audio occurences only from the EUROPE region and the duration of audio clip to be less than or equal to 15 seconds.*** EDA confirmed that this is a valid scope: longer recordings start to form a long tail in the duration distribution, and Europe has the most data.

For collaboration, we have set up this notebook as well as all the acquired data on Google Colab, but for the purpose of submission, we are running this notebook on the same data stored locally.

The main files of our interest which were acquired from GBIF's website are:

- multimedia.txt
- occurrence.txt
- meta.xml

2.a.1. multimedia.txt

This file contains information about the occurence and the link to download the audio. It has 15 fields; 7 fields are blank but are not needed (metadata). The main fields which we selected from this file are unique ID, type, format, identifier and description of the occurences. These fields are merged with selected fields of occurrence.txt later to use as a final dataframe for exploration. Since we are modeling audio clips, we have applied a filter on the 'type' field to select ony 'Sound' based observations. A look at the images revealed that they are not sufficiently pre-processed or standardized enough for our needs.

[ ] ↳ *2 cells hidden*

2.a.2 occurence.txt

This file mainly contains demographic, species' scientific identification, licensing and miscellaneous information with a lot of blank fields. After reviewing this file, we decided to use only selected fields which were useful for our purpose and objective.

We see missing data in some of these fields; details are shown below in the output of .info(). However, this missingness is not a roadblock. No imputation is used - enough data is available and filtered accordingly. Since we will be focusing on audio data rather than tabular data, columns with high missingness, such as as `sex`, can just serve as additional information when available. The column `associatedTaxa` has a high missing rate, yet it denotes occurences with background noise/other birds. Thus, the missingness in `associatedTaxa` shown below signifies that 33% of recordings have multiple birds singing. The only other concerning missing value rate is for level3Name, which characterizes cities. We can use geographic coordinates instead (if applicable).

There are some fields that will not be kept but are necessary as filters. These are `year` (the data ranges back to the 1800s) and `identificationRemarks`, which characterize observations where the classification is subject to review. According to the GBIF website, these should not have any associated multimedia, but we will filter them out to be certain.

[ ] ↳ *6 cells hidden*

2.a.3. meta.xml

This file provides a link to a description for each column (a lookup for the data). It was used to understand the values in each column, and which need to be considered for the analysis. The main list was extracted from the meta.xml file, thus this file does not need to be loaded anymore. It is accessible at: https://dwc.tdwg.org/list/

2.b. Basic Exploration and Data Selection

In this section, we performed some basic exploration of the data to understand the distribution of audio files as a function of duration in seconds, and different regions. The purpose of this exploration is to scale the magnitude of this project and select a subset of these 700k observations logically.

2.b.1. Distribution of audio files as a function of duration in seconds

On the raw data from multimedia dataframe, we saw an elbow plot showing audio files going all the way up to 2000 seconds, however the majority of the count of audio files was in lower durations (approximately <= 50 seconds).

In the next step we applied filtering on continent and duration to filter a reasonable size of data to work on.

[ ] ↳ *1 cell hidden*

2.b.2. Data Selection based on Region and Duration Filtering

- GBIF Region Present and == Europe
- Sound Files only
- files 15 sec or less

Applying these filters reduced the overall data from 700K to 80K (which roughly took 20 hours to download), which is large but more manageable than the original size.

Futher, we tried to plot count of audio samples per generic names, species, genus and vernacular names to identify if there is any opportunity to further filter the data while keeping the consistency among samples.

[ ] ↳ *8 cells hidden*

2.c. Final Sample Selection for POC Model

- Review different classification names (generic, vernacular, species or genus) to select our class labels for modeling.
- We noticed that Vernacular Names (common sense names) with at least 1000 audio samples, would give us 200 prediction labels (bird types).

- Limiting the number of samples per vernacular name to 1000 would reduce the number of samples we processed, but may be enough for good quality predictions (to be tested further)
- Apart from the bird species taxonomy, there is also the background birds (noise, if we want only one bird output) field and bird behavior to consider. As outlined below, additional noise can be a challenge, and whether to filter those observations out or use them will have to be weighed. Regarding the type of behavior, an overwhelming majority of data is for bird calls or songs. We may seperately classify both. Noteably, a review of the Sun Bird song identification app critisizes that it does not discriminate between these two behavior types. It is therefore a valuable improvement.

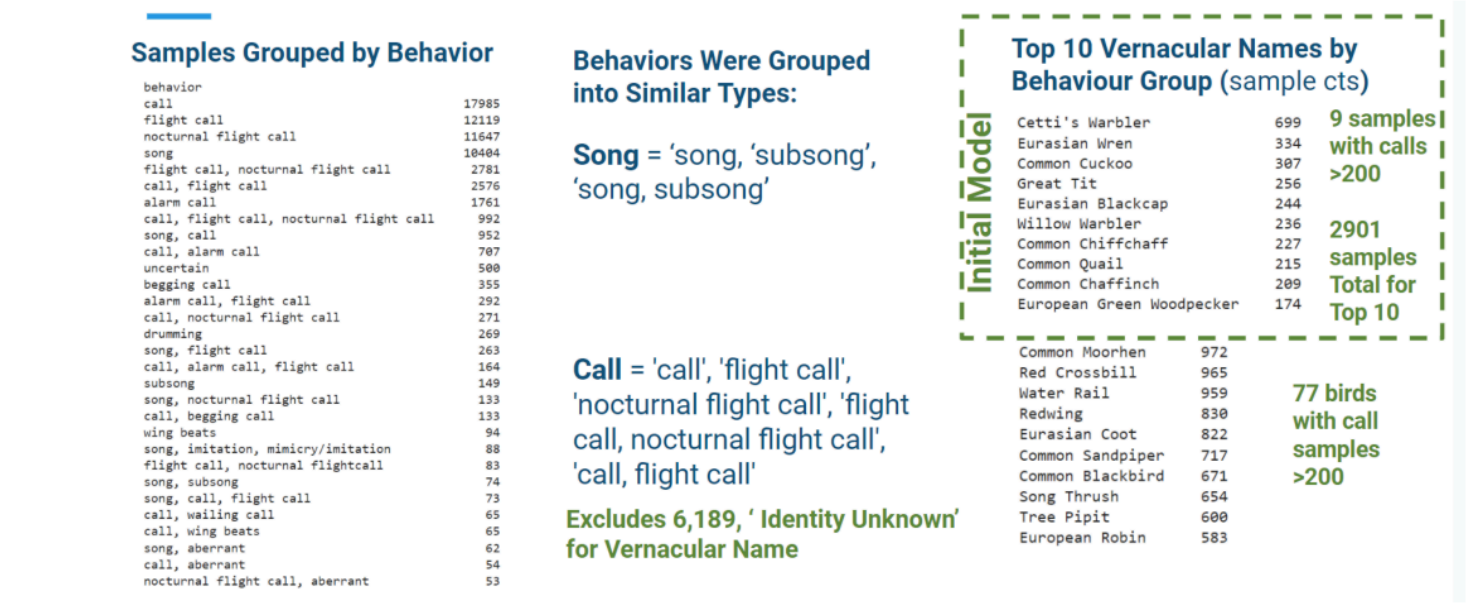
2.c.1.Further Data Analysis and Sample Narrowed to Top 10 Birds with Data Availability for Song and Behaviors

In order to account for bird behaviors that may vary within the sample, we segmented out data into CALL and SONG type behaviors and determined the top 10 birds by vernacular name for each behavior type. The SONG category was selected for our first prediction set, given it was smaller (easier to process) but also provided an additional challenge through higher class imbalance and lower samples. The Calls behavior has more samples that surpass the 200 sample mark, so we also created a 80 bird_call samples on top of the 10 bird_song samples that could be used for future expanded predictions.

[ ] ↳ *11 cells hidden*

2.c.2. Final Cohort Selection

```
img = mpimg.imread(os.path.join(base_path,'figures/sample.png'))
plt.figure(figsize=(14, 10))
plt.axis('off')
imgplot = plt.imshow(img)
plt.show()
```



3. Audio Processing Refinement

The project first took individual audio files and processed them to review best practices in using TensorFlow I/O audio functionatlity. These findings were used to generate a more robust data processing function to generate image samples for our selectected bird subset.

Tensorflow I/O Documentation

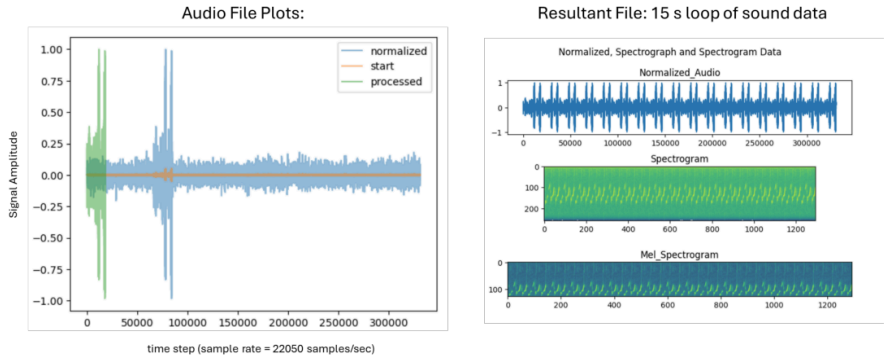
Key Findings

1. Audio must be converted into a tensor for effective processing
2. Each audio has its own sampling rate, which needs to be standardized across all audios. This is done through resampling.

3. Audios can come with one or two channels. All audios must therefore must be standardized into one channel by taking the mean of the two channels.
4. Some audios are quiet/far away from the bird. In order to compare audios effectively, the files must be normalized in a range of -1 to 1, where the maximum amplitude is used as basis.
5. Long tails of silence can occur both at the start and/or end of audios. This causes discrepencies in the appearance of signals; trimming is applied to keep only the parts where the bird sings.
6. Adding padding to reach the specified length of 15 seconds across all audios equally results in long tails of silence. Instead, audio looping is utilized. Padding also requires resizing (padding) of images at the time of model input, which is an extra pre-processing step that results in a sub-optimal visualization. Looping eliminates this need as all image acquire the same dimensions.
7. Final image generation: a mel spectrogram in decibels is the most aligned with the human auditory system's perception of sound and creates the clearest image compared to the other alternatives tested (spectrogram, spectrogram in decibels, mel spectrogram not in decibels).

These findings are represented in the visualizations below, where the first image shows the trimmed and normalized audio in green, and the second illustrates the effectiveness of mel spectrograms vs. spectrograms and how they compare to an audio signal that is only in the time domain (Normalized_Audio) instead of both time and frequency.

```
img = mpimg.imread(os.path.join(base_path,'figures/preprocessing.png'))
plt.figure(figsize=(14, 10))
plt.axis('off')
imgplot = plt.imshow(img)
plt.show()
```



3a. Audio Processing: Single Sample Example

In this section, the effect of audio processing is tested in order to build a pipeline. This entails loading the audio into tensors and observing the effects of different pre-processing techniques that are applied. Finally, spectrograms, mel spectrograms and mel spectrograms in decibel scale are compared to choose a final input format for the model.

https://www.tensorflow.org/io/tutorials/audio
https://www.geeksforgeeks.org/audio-data-preparation-and-augmentation-in-tensorflow/

[ ] ↳ *3 cells hidden*

3b. Audio and Image: Multiple Sample Generation

The final audio and image sampling pipeline leverages the TensorFLow I/O audio package to normalize, reduce background noise, trim samples to isolate bird sound, and then loops them to a full 15 seconds to ensure consistent file lengths. The sound files are then converted to mel spectrogram image files.

Prior to the pipeline, we also created an 'oversample' function that calculates the largest class and then randomly samples images and varies some of the image generation settings in order to create a balanced number of samples for each class. The oversampling function was introduced after testing the baseline model and realizing that there was a strong majority class that may be creating class imbalance.

While more robust transformations using the ImageDataGenerator function in our next pipeline step were also attempted, we found that those settings created noise that lowered model accuracy and found that it was more effective to perturb the mel-spectrogram generation factors, such as frequnecy and decibel filters and noise reduction factors, to generate incremental image samples. The final frequency, decibel levels, mels, noise cancellation were driven by visual inspection of images and the number of exceptions generated. If the noise reduction was too high or if frequencies were narrowed too much, an image sample would fail to generate or and all or majority white image would be created. Note that the image processing pipeline sometimes fails to generate a small number of the samples, however the settings were optimized to minimize these exceptions.

While this sample generation function was run numerous times to perfect sample generation, the logic remaining in this write up includes processing the 2901 original samples, and then 6990 'oversampled' images. We also experimented with image generation on the fly, in conjunction with generating batches to feed into our model. However in our remaining approach, we directly generated images and reused those images in our subsequent steps. This way the mel-spectrograms would need to be generated only once and after they are stored in the directory that portion of the code need not be run again and resources are used more efficiently to train the model from stored images, perform several iterations as well as try different models.

```
#Import meta data file
song_top10_df = pd.read_csv(os.path.join(base_path,"song_top10_df.csv"), index_col = 0)
```

3.b.1.Oversampling logic to increase under representated classes

[ ] ↳ *1 cell hidden*

3.b.2. Image / audio sampling function (LONG LEAD TIME 1-3 sec/sample)

[ ] ↳ *8 cells hidden*

4. Image Data Generator Functions

The Image Data Generator function within TensorFlow is used to develop our image data pipeline. Mel-spectrogram images are split into 3 dataframes for training, validation and testing and then using flow_from_df() method to get these images in a batch size of 16. The validation set is used during model training to avoid data leakage and we preserve the test data until we have a final model ready to test. The `class_mode` in the pipeline is set to 'categorical' since it is a classification problem. We are shuffling the data for training to reduce any bias.

The Image Data Generator also gives us flexibility to perform data augmentation using various parameter for rotation, shifting, noise addition, flipping etc. At this step, images are rescaled to a range of 1/255 to normalize the images, but other generic transformations are not used. While testing the baseline CNN model, these generic transformations showed lower accuracy scores than when the image pipeline omitted generic transformations. One hypothesis for this result is that very sophisticated information is encoded within our mel-spectrogram images and the generic preturbations destroy valuable information for the model learning process. With this insight, we instead conduct our image variance within our sample processing step, as discussed in Section 3.

[ ] ↳ *8 cells hidden*

5. Model Selection and Testing

In this section we design and test several neural network architectures on the imbalanced as well as balanced (oversampled) dataset. We will use the data pipeline which we created in the previous section using the cnn_prep_pipeline() function. Since our data is now in the form of images, we will mainly use a Convolution Neural Network (CNN) and experiment with the architecture based on the observations, i.e. train and validation accuracies, loss and F1-score for all the ten classes of birds. For this prupose, we have created two helper funtions as shown below:

plot_hist()

- This funtion plots the training and validation accuracies and losses as a function of epochs.

model_eval()

- This function takes several arguments including the model, training and validation data pipeline, number of epocs to run, and name of the model, etc.

- It defines two callback fuctions. First one for early stopping with patience of 5. This is done to preserve energy and unnecessary running of epochs in case if model's performance is not improved for several epochs (5 in this case).
- Second callback function is model check point to save the best weights achieved during the training process.
- Model fitting is performed in model_eval()
- At the end of training, model evalation is performed training and validation data using the best weights.
- Lastly, this function calculates the F1-score for each of the top 10 birds and plots them on a horizontal bar graph.

Helper Functions

[ ] ↳ *2 cells hidden*

5.a. Baseline Model with No Oversampling

The first model we designed is our naive baseline CNN model with a basic design of 3 convolution layers with 64, 128 and 32 filters respectively. Our input dimension will remain consistent at 128 x 128 x 3 throughout the testing. Since our images are mel spectrograms of bird songs, where the audio features are concentrated only in certain areas of the image, we started with a small kernel size of 3 to extract the small and localized features from the image. To preserve symmetry and information in the convolution step, we used padding = 'same' to ensure consitent dimensioning at the step of convolution and the size reduction is performed after each convolutional layer using maximum pooling of size 2, i.e. reducing the dimensions by half. After flattening the output from the convolution section of the model, we pass it through only one dense layer of 256 units. The last output layer has 10 units based on 10 classes in the data sample, for which we applied softmax as activation function. The output layer will also remain consistent throughout our testing.

We have choosen 30 epochs as default which will be used for all the models we will test. But with that we are using early stopping as well to ensure that the model doesn't keep running unnecessarily.

In the first step we ran the baseline model on original imbalanced data. The basline model stopped training within the first ten epochs. We saw the training and validation losses diverging very quickly after the first few epochs. The model gave us training accuracy of around 80% and validation accuracy of around 66%, suggesting severe overfitting. The average F1-score was around 65% with some birds having low F1-scores due to imbalanced data. Please note that these numbers may change upon next execution of the code due to randomness in tensorflow.

In the next step we will train the same baseline model on balanced (oversampled) data to compare the performance difference just by data augmentation.

[ ] ↳ *1 cell hidden*

5.b. Baseline Model with Oversampled Data

Upon training the same baseline model on oversampled balanced data, we saw that the model trained for more epochs and gave a boost to train and validation accuracies. We got a training accuracy of around 98% with a validation accuracy of around 91%. We still see a lot of overfitting in the results. The F1-score jumped up to low 90% with only one bird falling behind the average. The results were quite encouraging and suggested to experiment with model architectures and see if different design choices can help us improve further on accuracy, F1-score and overfitting.

[ ] ↳ *1 cell hidden*

5.c. Optimized Model 1 with Oversampling

From this point onwards, all of our model training will be performed only on balanced data.

In the next iteration, we only added one more layer in the convolution network, having filters as 64, 128, 64, 32, to make the network more robust and extract more features. We also addethe d one more dense layer of 128 units on top of the previous layer of 256 units. We noticed that this network significantly reduced the total number of trainable parametrs compared to the baseline model; baseline model parameters were around 2.2 millions vs optimized model 1 of roughly 720K, which is one third of the baseline model's parameters. Upon examining the model summary, we notced that reduction in total parameters is due to size reduction after adding one more convolutional and max pooling layer which in turn reduces the number of parameters going into the dense layer.

This was an interesting observation as we found similar performance as the baseline model but at a significantly less number of trainable parameters. Therefore we decide to use the same architecture for all further enhancements in next steps.

We got a train accuracy around 97% and validation accuracy around 91%, making it slightly less overfitting than the baseline model. The average F1-score stayed above 90%, close to the F1-score from the baseline model, this time a different bird falling behind the average,

suggesting this could be due to randomness in the training process. In the next steps, we will apply various regularization techniques to reduce the gap between training and validation accuracies.

[ ] ↳ *1 cell hidden*

5.d. Optimized Model 2 - Oversampling & Regularization

In this step we experimented with two enhancements in our design:

1. We increased the kernel size in the first 2 layers, inspired by the AlexNet architecture, hoping this would enhance high level feature extraction from the mel spectrogram.
2. Regularization: We used L1 regularization (0.0001) and dropouts (0.5) in the dense layers. We tried various values for both but these values performed best.

The number of parameters slightly increased to about 878K due to increase in kernel size, but it is still significantly lower than the baseline model. This model took more number of epochs to train compared to previous models and gave us training accuracy of around 98% but the validation accuracy increased to around 93-94% which is the highest we have achieved so far. The average F1-score also increased to mid 90%, with all the birds above 90% individually.

These are the best results we have seen so far in our modeling journey. There is still some overfitting but it is less than what we have seen in previous steps. To overcome this further, we tried to increase the regularization parameter value from 0.0001 to 0.0005 and 0.001, as well as batch normalization but neither of these gave us any additional gains.

In the next step we tried to incorporate Autoencoders as our last attempt to reduce overfitting.

[ ] ↳ *1 cell hidden*

5.e. Optimized Model 3 - Oversampling & Regularization & Autoencoders

In this section, we tries to add autoencoders with our optimized model 2 from the previous step as an attempt to reduce overfitting. This is going to be a two-step process, with first step to train the autoencoders. For this purpose, we extracted the images portion from our train and validation pipeline and combined them both into one dataset to train the autoencoders.

Due to time limitation, we did not experiment too much with the architecture and adopted the design as shown in the Lab on CNN. After training the autoencoders, we tested their result on the mel spectrograms and found them to make our mel spectrograms more smooth and denoised as expected (this test was done separately and not included in this notebook).

Once we finisned training the autoencoders, we froze its layers and added a layer of these autoencoders in our optimized model 2. The total number of parameters increased by approximately 121K coming from autoencoders, making the total parameters to be about 1 million. This model took longest to train out of all the models so far and gave us similar results as previous models; training accuracy of about 98% and validation accuracy of about 92%. The F1-score dropped back to low 90%.

Overall, We did not see any gain by utilizing autoencoders to help in reducing overfitting.

So far we have seen that our optimized model 2 has outperformed all of our other attempts. It gave us the highest validation accuracy with relatively lower number of trainable parameters. Therefore it will be our choice as a final model to be applied on unseen test data.

[ ] ↳ *5 cells hidden*

5.f. Overview of Models

```
img = mpimg.imread(os.path.join(base_path,'figures/model_arch.png'))
plt.figure(figsize=(12, 8))
plt.axis('off')
imgplot = plt.imshow(img)
plt.show()
```

CNN Models	Baseline Model	Optimized Model 1	Optimized Model 2	Optimized Model 3 with Autoencoders
Input	128 x 128 x 3	128 x 128 x 3	128 x 128 x 3	128 x 128 x 3
Conv2D layers	3	4	4	4
Conv2D channels	64, 128, 32	64, 128, 64, 32	64, 128, 64, 32	64, 128, 64, 32
Kernel size	3x3, 3x3, 3x3	3x3, 3x3, 3x3, 3x3	11x3, 9x3, 3x3, 3x3	11x3, 9x3, 3x3, 3x3
Padding	Same	same	same	same
MaxPooling2D	(2,2), (2,2), (2,2)	(2,2), (2,2), (2,2), (2,2)	(2,2), (2,2), (2,2), (2,2)	(2,2), (2,2), (2,2), (2,2)
Stride	1	1	1	1
Dense layers	1	2	2	2
Dense units	256	256, 128	256, 128	256, 128
Dropouts	-	-	0.5, 0.5	0.5, 0.5
L1 regularization	-	-	0.0001, 0.0001	0.0001, 0.0001
Output layer	1	1	1	1
Output units	10	10	10	10
learning rate (Adam)	default	default	0.0005	0.0005
Trainable Parameters	2,212,522	726,634	878,698	125,315 (autoencoder) + 878,698 (model)

Autoencoder Architecture		
Encoder	Input	128 x 128 x 3
	Conv2D layers	3
	Conv2D channels	128, 64, 32
	Kernel size	3x3, 3x3, 3x3
	Padding	same
	MaxPooling2D	(2,2), - , (2,2)
	Stride	1
Decoder	Conv2D layers	3
	Conv2D channels	32, 64, 3
	Kernel size	3x3, 3x3, 3x3
	Padding	same
	UpSampling2D	(2,2), (2,2), -
	Stride	1
Trainable Parameters	125,315	

6. Results Interpretation and Analysis

In summary, our Optimized Model 2 provides the highest overall validation accuracy. It has one of the lowest numbers of parameters, however the model ran for 30 epochs before achieving the best model predictions/weights.

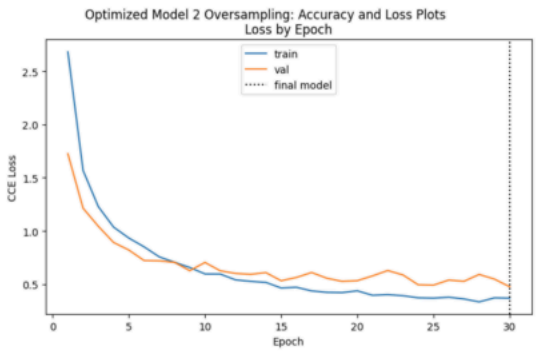
```
img = mpimg.imread(os.path.join(base_path,'figures/model_eval.png'))
plt.figure(figsize=(12, 8))
plt.axis('off')
imgplot = plt.imshow(img)
plt.show()
```

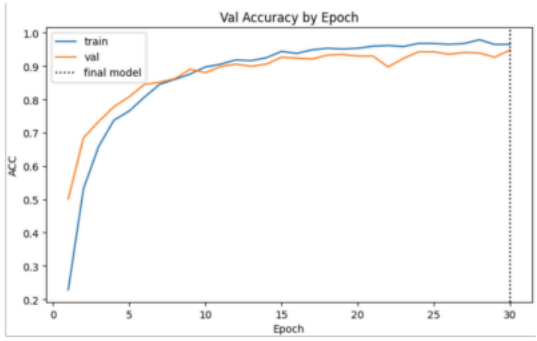
Best Model based on performance*

CNN Models	Baseline Model	Baseline Model	Optimized Model 1	Optimized Model 2	Optimized Model 2 with Autoencoders
Data	Imbalanced (No oversampling)	Balanced (with oversampling)	Balanced (with oversampling)	Balanced (with oversampling)	Balanced (with oversampling)
Trainable Parameters	2,212,522	2,212,522	726,634	878,698	1,004,013 [125,315 (autoencoder) + 878,698 (model)]
Epochs	30	30	30	30	30
Patience	5	5	5	5	5
Best Epoch	4	8	8	30	15
Train Accuracy	0.8002	0.9878	0.9701	0.9955	0.9837
Validation Accuracy	0.6674	0.9158	0.9112	0.9484	0.9212

The train and validation loss and accuracy curves follow closely, without significant overfitting of the train data. Model parameters were optimized to ensure more consistent results when model is applied to novel data in the inference phase.

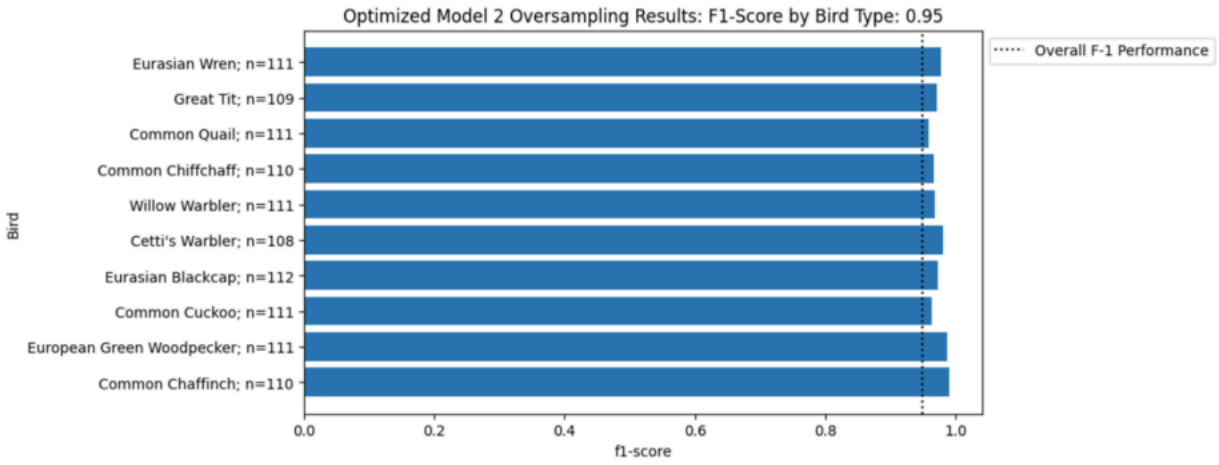
```
img = mpimg.imread(os.path.join(base_path,'figures/opt_2.png'))
plt.figure(figsize=(12, 8))
plt.axis('off')
imgplot = plt.imshow(img)
plt.show()
```





Accuracy of the predictions by bird type were also explored by calculating the f1-score for each prediction category. In the case of our Optimized Model 2, we see a balanced prediction accuracy across all 10 classes. Note that the oversampling of the data was one of our biggest imporvements in prediction accuracy, as underperforming classes were improved, raising overall accuracy scores.

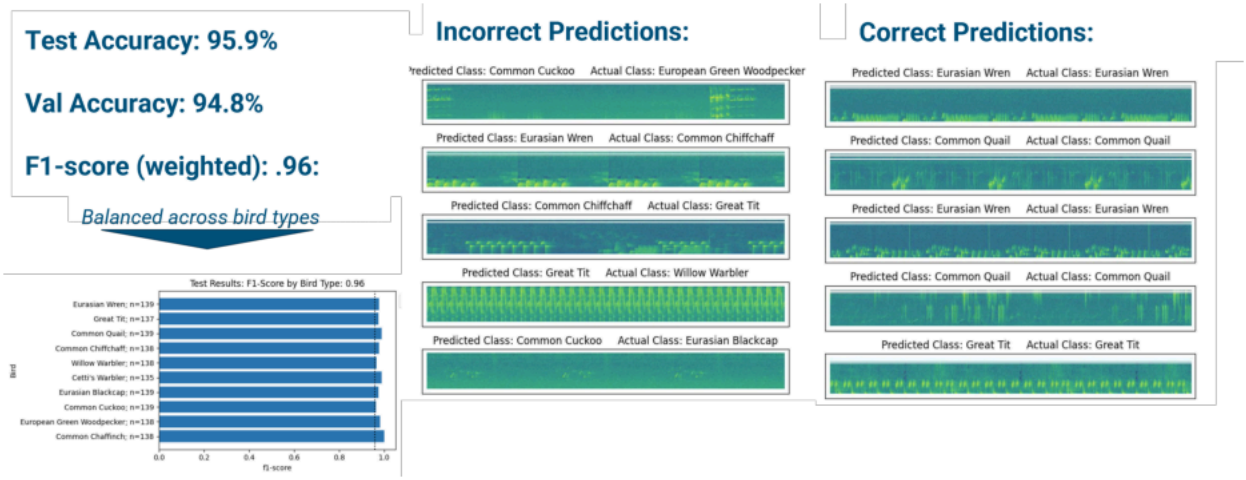
```
img = mpimg.imread(os.path.join(base_path,'figures/f1_score_opt_2.png'))
plt.figure(figsize=(12, 8))
plt.axis('off')
imgplot = plt.imshow(img)
plt.show()
```



6.b. Final Test Model Run & Ongoing Inference Expectations

Our test sample was used to conduct our final inference on the Top 10 Bird Song cohort. The final test accuracy of 95% was in line with our validation accuracy. Similarly the F1-score for the 10 classes also were balanced with all classes over 90%. Test accuracy results validated the consistency of our model performance on novel data.

```
img = mpimg.imread(os.path.join(base_path,'figures/final_inference.png'))
plt.figure(figsize=(12, 8))
plt.axis('off')
imgplot = plt.imshow(img)
plt.show()
```



Inference Results

[ ] ↳ *6 cells hidden*

7. Summary of Findings and Future Next Steps

Our 10 Bird Song model acheived 95% in test accuracy using our final selected model, which is greater that the 85% benchmark that we defined at the start of our project. While the model was tested on a narrow range of samples in order to perfect the sample generation pipeline and test model, the model also provides a foundation that can be expanded to broaden the model's prediction capabilities.

Additional extensions of the model are possible by incorporating additional samples to expand our problem statement to include additional species, bird behaviors, geographic locations. For instance, Appendix 1 contains a model that expands the 10 Bird Song model to an additional 80 Bird Calls, still within Europe.

Further, the foundational model is a base prediction of which of 10 birds are present in the recording. This model could also be expanded to not only predict the vernacular name of the bird but also the type of sound that a bird is making, using the same learning process. For instance, Appendix 2 contains an adjustment to our Optimized Model architecture to include predictions of call and song, alongside the 10 bird vernacular names in our base model.

Finally, while a relatively simplistic CNN-based model was capable of achieving a relative high accuracy, with additional data and classes, exploration of more robust architectures and transfer learning may be required to acheive similar accuracy results.

APPENDIX 1: Expansion to Top 80 CALLS and Top 10 SONGS

Objective: Test our final model and data pipeline on both Top 10 Bird_song combinations as well as Top 80 Bird_call behaviors within Europe.

Conclusions: Application of our final model from the Top 10 Bird Song data yields accuracy rates of 70% with a new sample size of additional 68 Bird Call combinations. Sample parameter tuning was required to cover ranges within call data to expand to appropriate frequency and decibel ranges. Additional tuning is required to ensure a balanced pool across all 90 samples, as 12 Bird_Call samples omitted entirely and several had lower sample sizes than our 700 cap.

Sample Generation Pipeline

35K original samples were oversampled with the goal of creating consistent groups of at most 700 samples. This cohort used the same hyperparameters to generate samples as our 10 Bird_song model. This model resulted in ~49K image samples, and ommitted data from 12 of the 90 bird_behavior combos being tested. Samples fail to generate if the noise filtering threshold is too high, or if the sound frequency or decible ranges are outside the paramters for the model. In research, bird calls occur at lower frequencies and may be higher decibles.

Further adjustment is required as there is still significant sample omission for several bird_behavior groups and a small portion of resulting images have some white banding at the top. Both of these challenges can be overcome by further paramter tuning within the sample generation process.

[ ] ↳ *2 cells hidden*

Model Results

The samples from our 78 remaining bird-behavior combinations were run through the Optimized Model 2, under similar specifications as our inital 10 Bird-song samples. Additonal parameter tuning will be required in order to optimize model to additional samples.

Initial Validation Accuracy: 70%

Initial Train Accuracy: 74%

[ ] ↳ *1 cell hidden*

F1-Score Performance

F1-Score performance varies widely across bird_behavior types. Sample irregularity may be part of the issue given many of the samples generated have white horizontal banding that may interfere with the learning process. Model accuracy should improve as image quality improves through further tuning and data coverage becomes more consistent across the bird_behavior categories.

[ ] ↳ *2 cells hidden*

A1.A:Oversample 80 call and 10 song data frames

[ ] ↳ *1 cell hidden*

A1.B: Generate New Samples for Additional Categories

[ ] ↳ *3 cells hidden*

A1.C: Datagen Pipeline for Model

[ ] ↳ *2 cells hidden*

A1.D: Call Final Optimized Model and Use on New Samples

[ ] ↳ *2 cells hidden*

APPENDIX 2: Additional Sample Augmentation with Call and Song for Top 10 Song bird

Objective: expand the project scope to both song and call behaviors within Europe for the top 10 song birds. Increase the number of samples for both behaviors by including additional samples from the original data. Aim for 700 samples per bird type for both song and call (which is the amount of samples the majority song category, Cetti's Warbler, has). Oversampling will be used to make up differences where not enough data is obtained.

Finally, modify the pipeline and models to incorporate 2 targets: bird type and behavior (song/call). By observing the impact of adding more data and increasing the classification task, we will get valuable insights on future improvements.

Conclusion: An average F1 score of 86% and 99% for bird types and behaviors could be achieved, respectively (accuracies: 85% and 99% for test without noteable overfitting). However, this not only required pulling additional data from our data set (by expanding filters), but also ensuring that both song and call data have enough extra augmented data. In adition, the model architecture had to be modified and the mel frequency ranges and had to be tuned precisely for calls, especially as even small variations (when augmenting data) could lead to no distorted images. As concluded in Appendix 1, already further research on audio parameters would improve results and perhaps eliminate the need to use a more complex model architecture as done here.

1. Gain additional data

The data is filtered to obtain new bird samples:

- Data after 2010 with duration > 15 seconds and <= 60 seconds

- Data between 2007-2010 (not including the latter) with length <= 60 seconds
- Data after 2010 with length <= 15 seconds with non-exact matching (as opposed to only exact matches defined previously, e.g. 'songsong') - gives a few extra samples

Below is the distribution breakdown and the amount of additional samples that are required for song per class. Cetti's Warbler is the dominating song class and therefore needs no additional samples. In terms of calls, 700 need to be added for all bird types, to match the maximum number of songs.

vernacularName (sample numbers are approximate, may differ by a few which can be ignored since oversampling will be used to make up the numbers)

- Cetti's Warbler
  - approx. 700 song data, maximum amount per bird type
- Eurasian Wren
  - 332 song data, requires an increase of ±368 song samples
- Common Cuckoo
  - 308 song data, requires an increase of ±392 song samples
- Great Tit
  - 260 song data, requires an increase of ±440 song samples
- Willow Warbler
  - 233 song data, requires an increase of ±467 song samples
- Common Chiffchaff
  - 230 song data, requires an increase of ±470 song samples
- Eurasian Blackcap
  - 229 song data, requires an increase of ±471 song samples
- Common Quail
  - 215 song data, requires an increase of ±485 song samples
- Common Chaffinch
  - 211 song data, requires an increase of ±489 song samples
- European Green Woodpecker
  - 169 song data, requires an increase of ±531 song samples

[ ] ↳ *31 cells hidden*

2. Oversample the difference that remains after adding the new data. Generate spectrograms for both the new data and additional oversampled data.

The code below comes from the data pipeline, with small modifications to allow for double classification. With it, we arrive at the final sample numbers in train, test and validation shown below.

Expand the code below to see an image summarizing the final train, validation and test sample sizes.

[ ] ↳ *16 cells hidden*

3. Modify the data generator to handle double classification

[ ] ↳ *5 cells hidden*

4. Modify the final chosen model to handle double classification

[ ] ↳ *3 cells hidden*

5. Results using the final chosen model, optimized_model2

[ ]

↳ *2 cells hidden*

5.a Trying a different architecture to attain an 85% accuracy in bird type classification

Note that though the model outperforms the previous, there are a lot more parameters. The model was trained with additional 50 samples of augmented call and song data for all birds and behaviors. As suspected, this evened out the f1 score distribution and raised the performance for all classes. Using the original final model architecture (`optimized_model2`), the accuracy reached 80% with the additional samples. In order to breach the 85% accuracy threshold, the architecture had to be adjusted.

[ ] ↳ *3 cells hidden*

6. Summarize Findings & Future Work

In the process of adding more data, parameters had to be adjusted for call data, which exhibits lower sound frequencies. Using the default settings of song data on the call samples resulted in spectrograms with white, horizontal lines that even caused completely blank images in some cases. This also occurred for the top call and song bird classification task in the previous section. Examples are shown below. The settings were now updated for call to:

```
mel_fmin = 1500
mel_fmax = 7000
```

When oversampling, small variations in the `mel_fmin` and `mel_fmax` could already result in white horizontal lines, thus the allowed deviation range for creating new samples was reduced to `mel_range = 0.01`. Instead, the Epsilon value beneath which sound is considered noise was varied more aggressively (the current oversampled data used `epsilon_range = 0.5`, yet 0.9 produced even more distinct results, but with more exceptions that could not render). It is an effective method to create new samples and is more fitting than the options provided by the data generator. Further research on the possible frequency ranges per bird type and behavior, as well as other hyperparameters in mel spectrograms and audio processing, would improve the oversampling/augmenting phase. In order to remove the white lines, very fine tuning was necessary for `mel_fmin` and `mel_fmax`; a `mel_fmin` of 10 with `mel_fmax = 8000` resulted in only one or two lines, while increasing `mel_fmin` to 1500 and decreasing `mel_fmax` to 7000 finally removed them. Nevertheless, due to varying the `mel_range`, there are still a few smaples with a white line. The image below shows how effective increasing the `epsilon_range` is for sample generation and the issue with `mel_range`.

Regarding the model performance for double classification, the original results are:

- Train Accuracy for Bird: 0.81
- Train Accuracy for Behavior: 0.98
- Validation Accuracy for Bird: 0.7
- Validation Accuracy for Behavior: 0.96
- Test Accuracy for Bird: 0.72
- Test Accuracy for Behavior: 0.96

while the behavior type is classified with high accuracy, the bird type classification performs worse. There is a trade-off in performance between classes when using one model for different tasks. There is also more overfitting than our previous results, though this may be because not enough augmentation was used (since using new data was the primary objective).

It appears that using two seperate networks would give the best results. An argument against using seperate networks would be that, since calls and songs differ in frequencies, their spectrograms are different for each bird type and require both labels. Due to time constraints, the model could not be tested on both call and song data without behavior labelling, yet this would be a valuable next step to verify whether a second class is necessary in the first place. However,the model has nigh perfect f1 scores for the behaviors; there is likely a distinct difference in them and a model without additional behavior information would struggle to classify both song and call.

Fine-tuning this model would be more resource efficient than separate models. For instance, having seperate fully connected layers instead of shared ones may achieve a similar effect to seperate models, without increasing the parameters as much as a second model.

A positive takeaway for the bird type classification in this joint classification task is that, while the F1 score decreased for bird types, it is still above 50% for all bird categories, meaning the model has a better than chance success at identifying positive cases correctly and avoiding false positives. The Common Quail and European Green Woodpecker have the highest F1 scores. While these bird classes do not have the highest absolute number of oversampled data for either call or song, they have the highest joint number of oversampled data. This indicates that oversampling works well as an augmentation technique if it is done for both behaviors equally. For instance, the Eurasian Blackcap has no oversampling for song (only call), and it has the lowest F1 score. To test this, we increased all call and song samples by 50 extra augmented samples. As a result, the F1 score imrpoved to 86% with the same accuracy scores for train, test and validation of 85%, which just manages to hit our specified target accuracy. The F1 scores are more uniformly distributed across bird types compared to the initial results.

Apart from the above, the research direction of this project would also benefit from 1) further researched pre-processing/augmentation methods to account for more bird classes in the future, where the imbalance will only grow and samples will be few, 2) additional inputs for geographic information or other variables in the data, 3) making use of the bird background noise column to furnish the top 3 likeliest birds instead, and 4) fast, efficient processing and real-time inference capabilities (many Gigabytes of storage were required for audio and spectrograms, much was needed in compute resources; being able to retrain the model frequently with new Xeno Canto data would be useful, etc.). Scalability and cost efficiency should be considered.

```
img = mpimg.imread(os.path.join(base_path_109b_folder,'figures/augmentation.png'))
plt.figure(figsize=(14, 10))
plt.axis('off')
imgplot = plt.imshow(img)
plt.show()
```