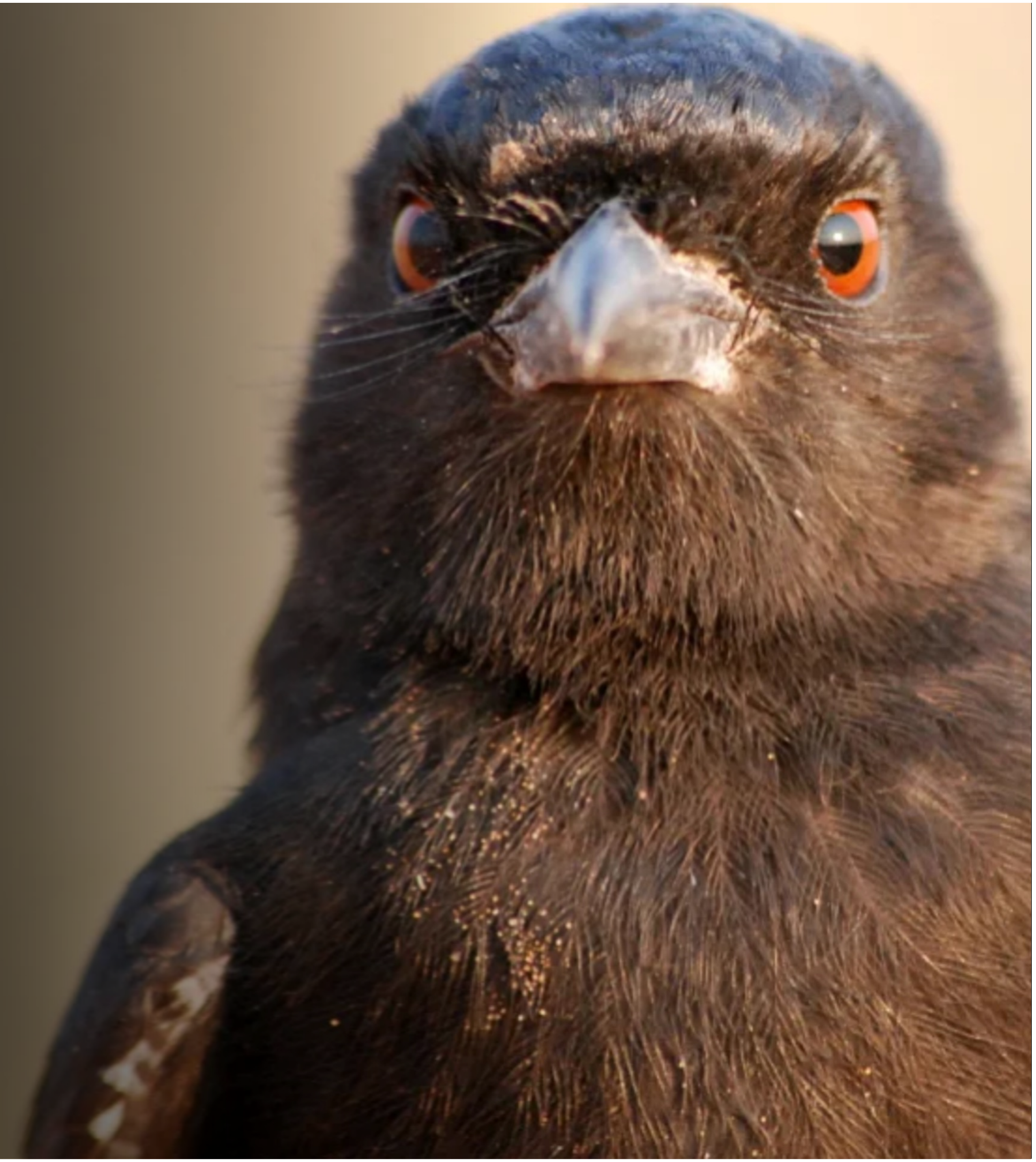
WHAT DOES THE BIRD SAY?

JAKOB D. OLSEN, BIRK DISSING
& MALTHE A. M. NIELSEN

12.06.2023



OUTLINE

PROBLEM

DATA VISUALIZATION

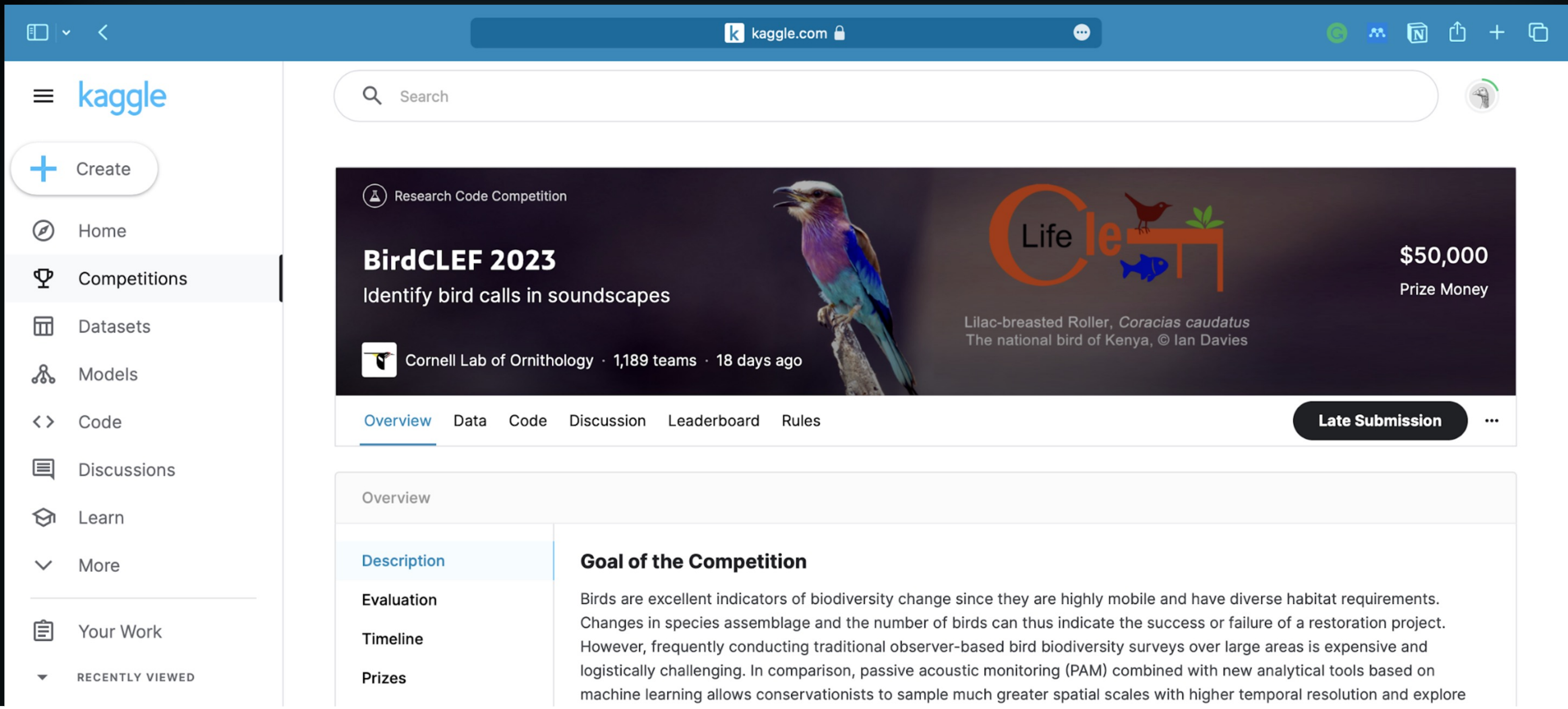
CNN HOMEBREW

CNN PRETRAINED

CONCLUSION

KAGGLE

GOAL OF THE COMPETITION



KAGGLE

RULES & GOALS

KAGGLE RULES

- MAX 2 HOURS CPU RUNTIME
- NO GPU
- NO PRIVATE DATA

GOAL

- IDENTIFY 264 AFRICAN BIRDS

THE DATA

AND METADATA

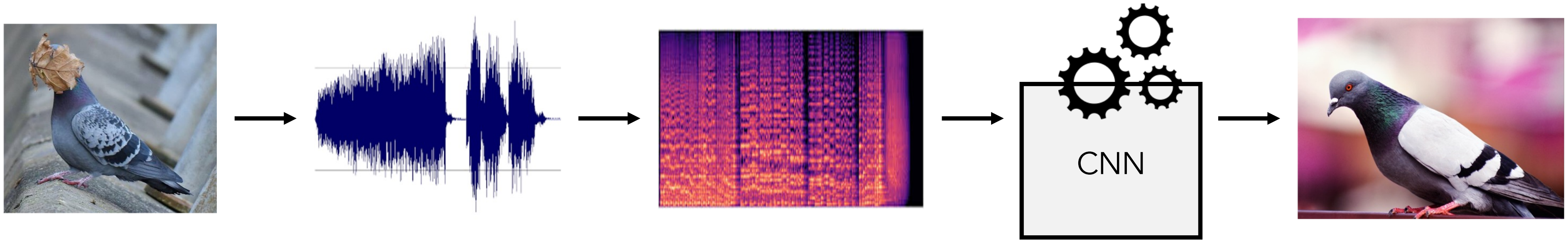
AMOUNT OF DATA

- 264 DIFFERENT BIRDS
- 1087 AUTHORS
- 193 HOURS AUDIO FILES (4.92 GB)
- 16974 FILES

	primary_label	secondary_labels	type	latitude	longitude	scientific_name	common_name	author	license	rating	url	filename
0	abethr1	[]	['song']	4.3906	38.2788	Turdus tephronotus	African Bare-eyed Thrush	Rolf A. de By	Creative Commons Attribution-NonCommercial-Sha...	4.0	https://www.xeno-canto.org/128013	abethr1/XC128013.ogg
1	abethr1	[]	['call']	-2.9524	38.2921	Turdus tephronotus	African Bare-eyed Thrush	James Bradley	Creative Commons Attribution-NonCommercial-Sha...	3.5	https://www.xeno-canto.org/363501	abethr1/XC363501.ogg
2	abethr1	[]	['song']	-2.9524	38.2921	Turdus tephronotus	African Bare-eyed Thrush	James Bradley	Creative Commons Attribution-NonCommercial-Sha...	3.5	https://www.xeno-canto.org/363502	abethr1/XC363502.ogg
3	abethr1	[]	['song']	-2.9524	38.2921	Turdus tephronotus	African Bare-eyed Thrush	James Bradley	Creative Commons Attribution-NonCommercial-Sha...	5.0	https://www.xeno-canto.org/363503	abethr1/XC363503.ogg
4	abethr1	[]	['call', 'song']	-2.9524	38.2921	Turdus tephronotus	African Bare-eyed Thrush	James Bradley	Creative Commons Attribution-NonCommercial-Sha...	4.5	https://www.xeno-canto.org/363504	abethr1/XC363504.ogg

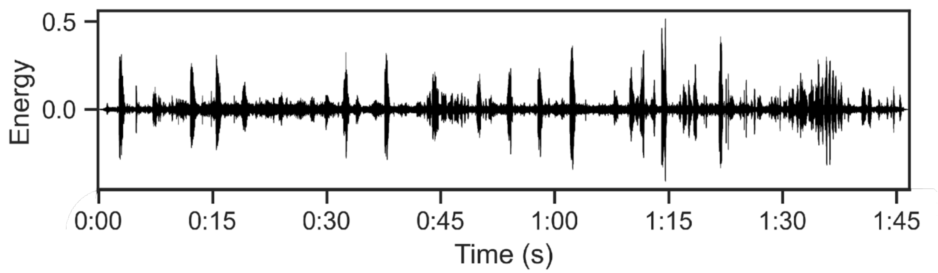
WORKFLOW

What idea did we have?



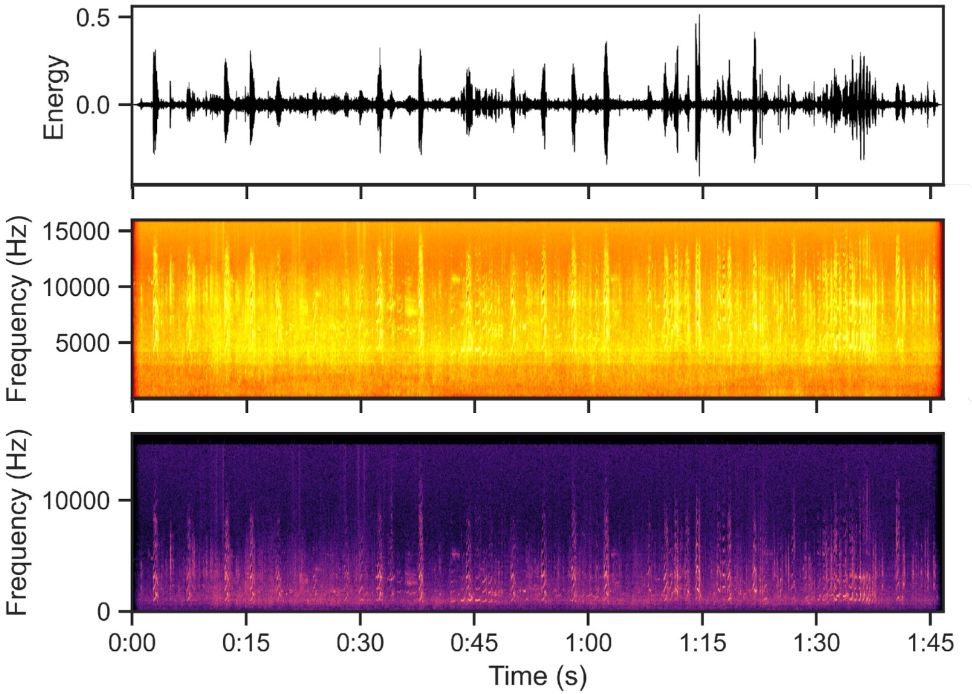
TRANSFORMATION

FROM AUDIO TO PICTURE



TRANSFORMATION

FROM AUDIO TO PICTURE

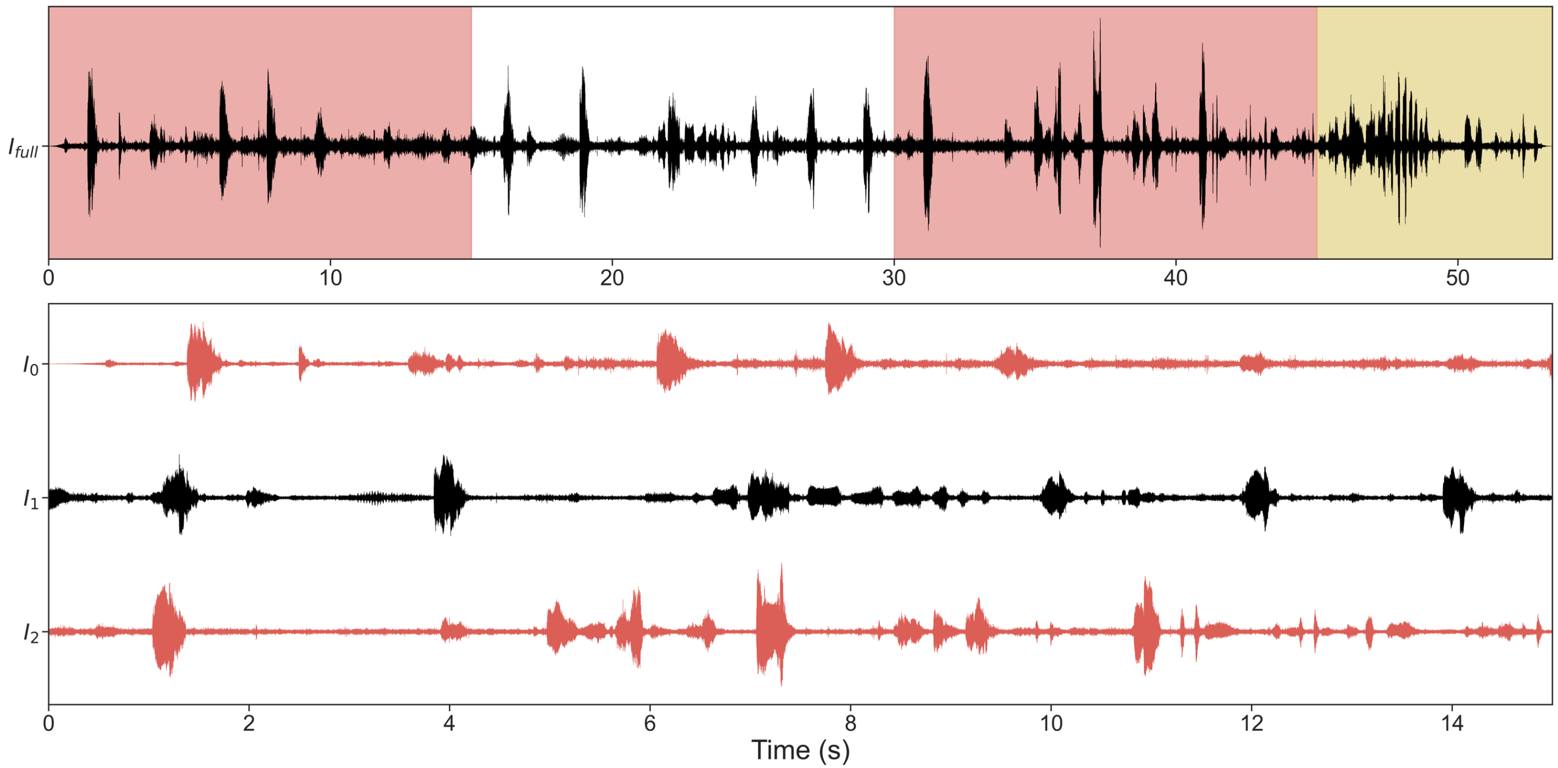


MELSPECTROGRAM

```
torchaudio.transforms.MelSpectrogram(
    sample_rate=32000,
    n_mels=128,
    n_fft=2028,
    hop_length=512,
    f_max=16000,
    f_min=20,
    power=2,
    window_fn=torch.hann_window,
)
```

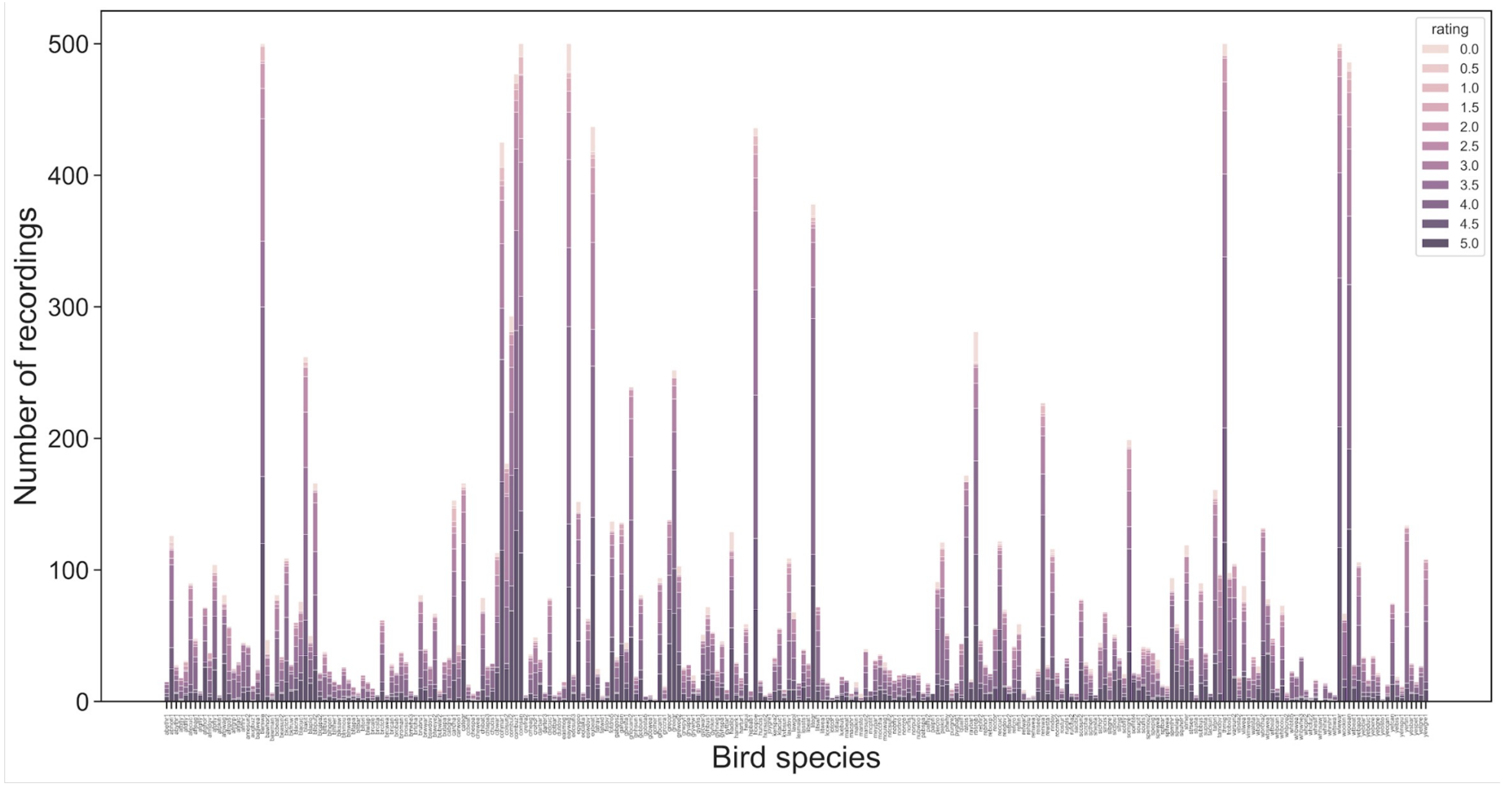
CUTTING SAMPLES

SEPERATED IN SHORT TIME INTERVALES



OF RECORDING

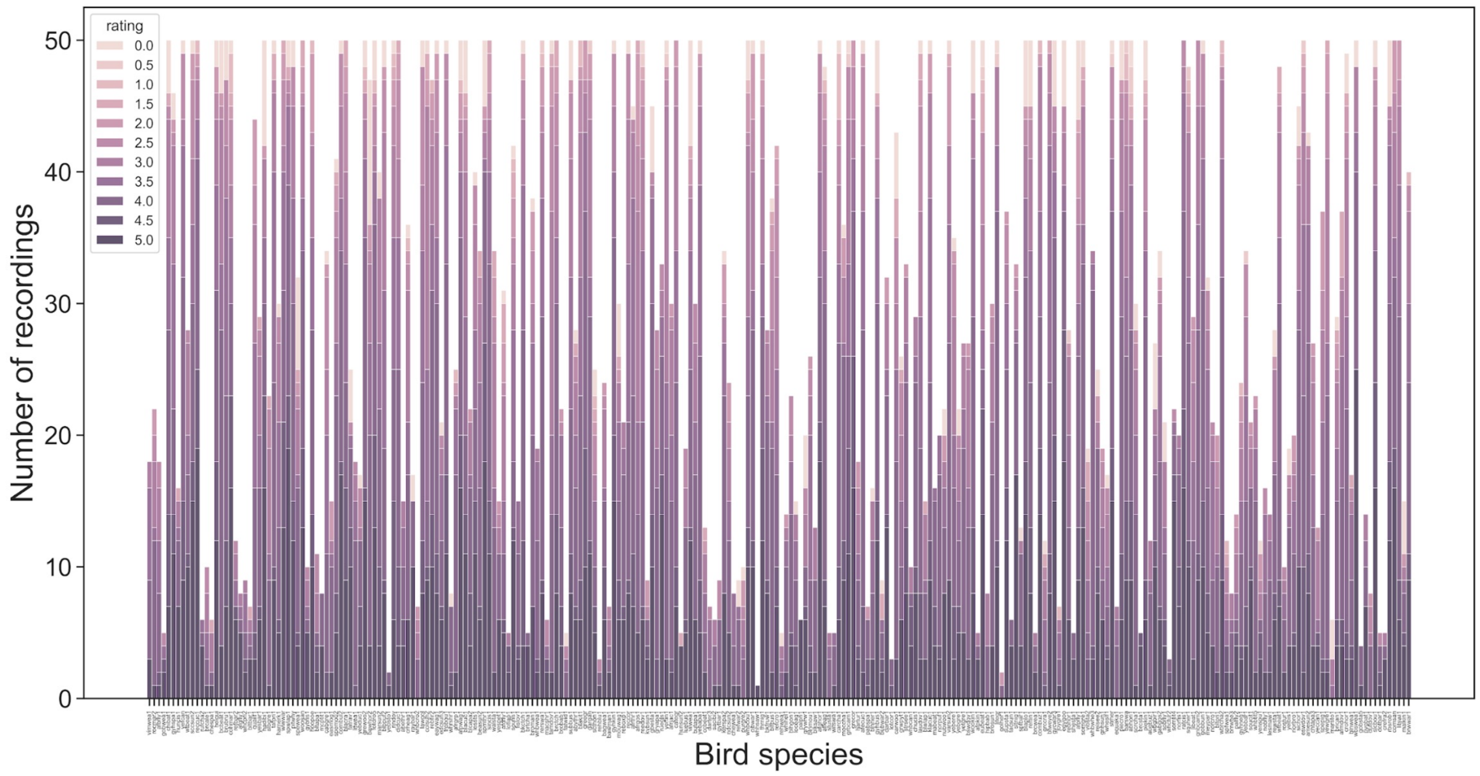
PER BIRD (STARTING OUT)



OF RECORDING

PER BIRD (WITH CUTOFF)

16974 ➔ 7966 FILES



OUTLINE

PROBLEM

DATA VISUALIZATION

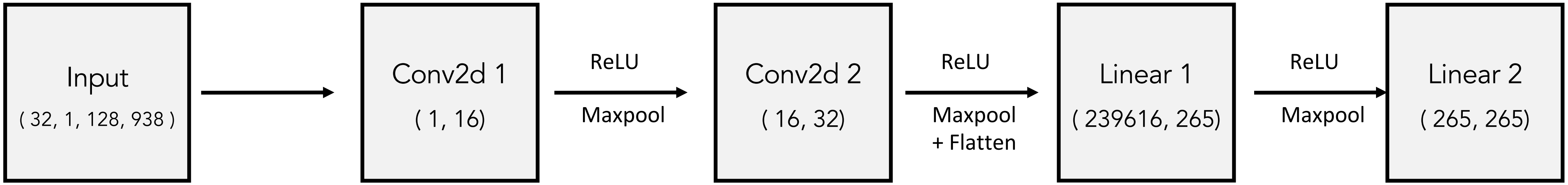
CNN HOMEBREW

CNN PRETRAINED

CONCLUSION

CNN SIMPLE

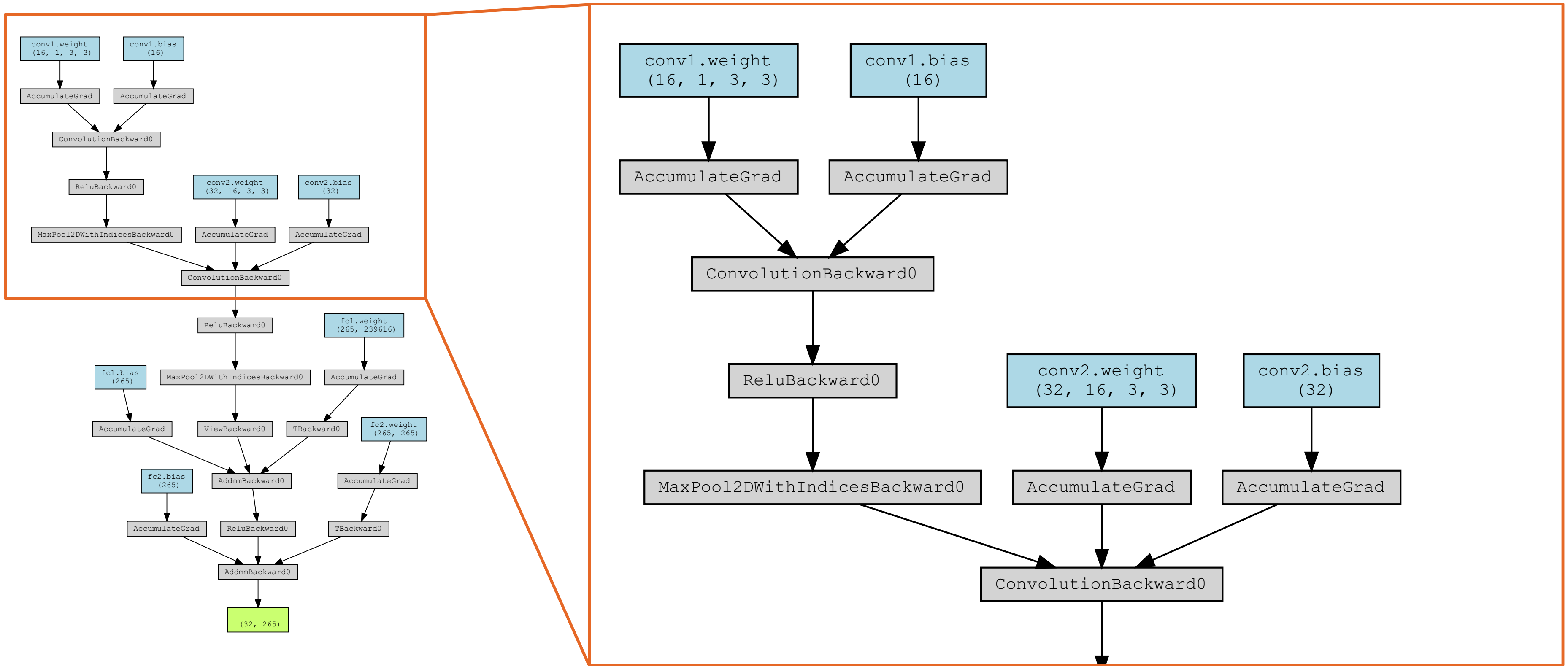
STRUCTURE



LR = 0.001
CROSS ENTROPY LOSS
ADAM OPTIMIZER

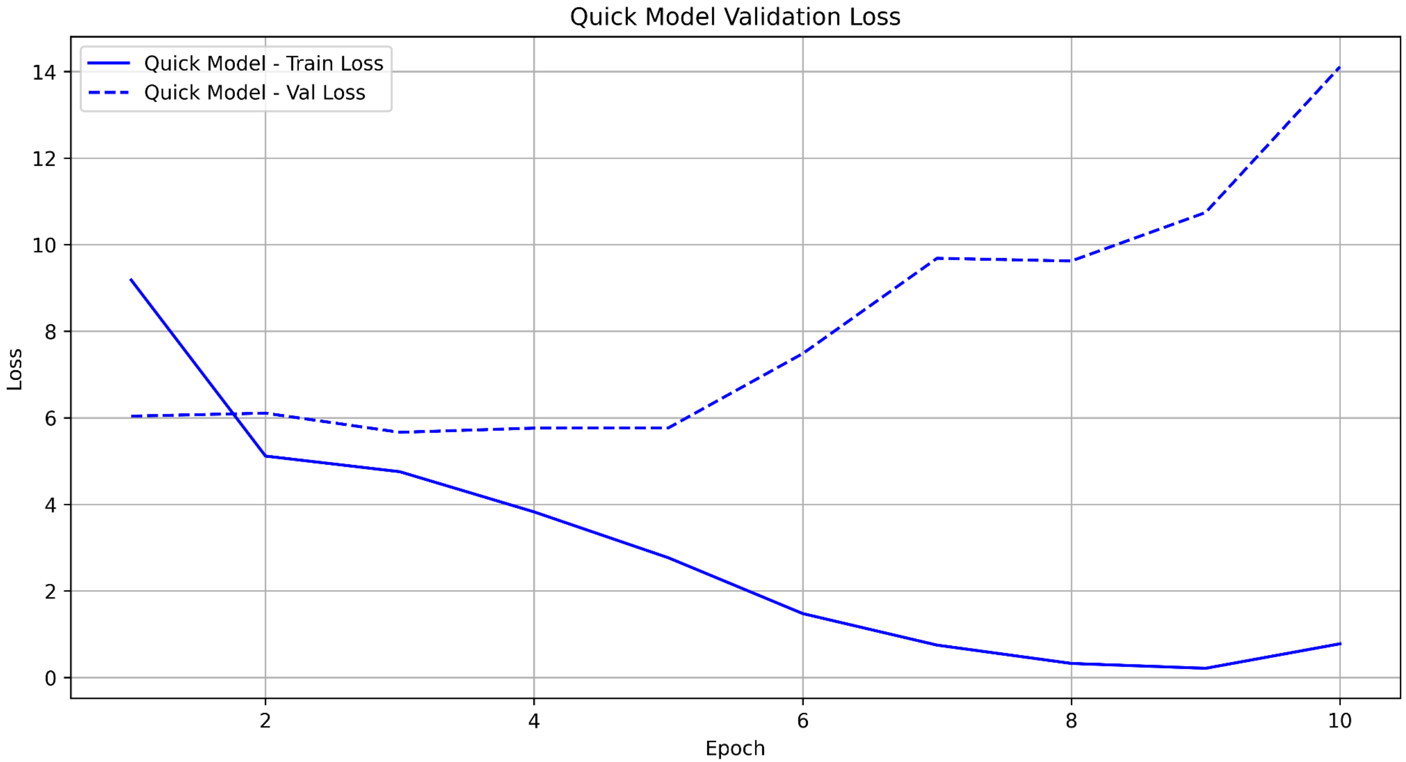
CNN SIMPLE

STRUCTURE



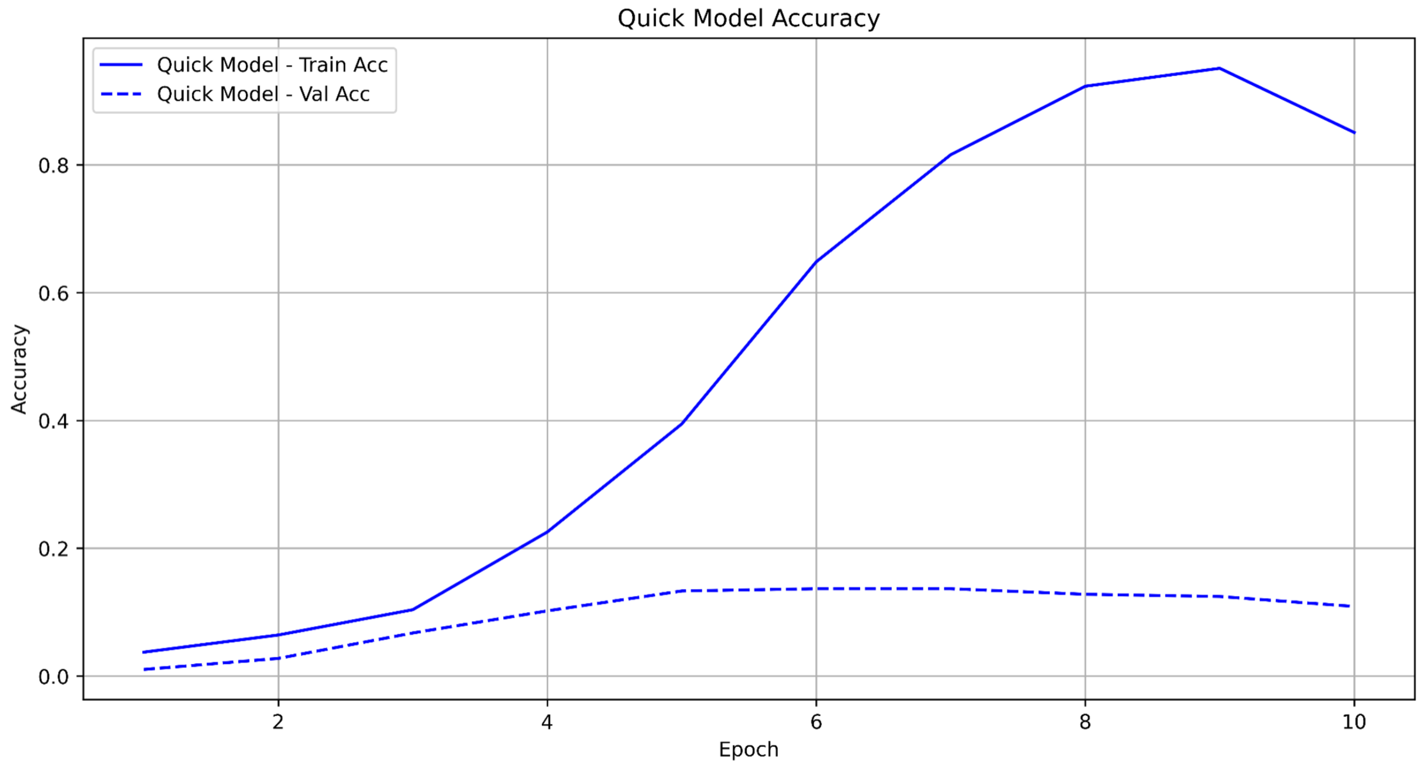
MODEL LOSS

LOSS VS. EPOCH



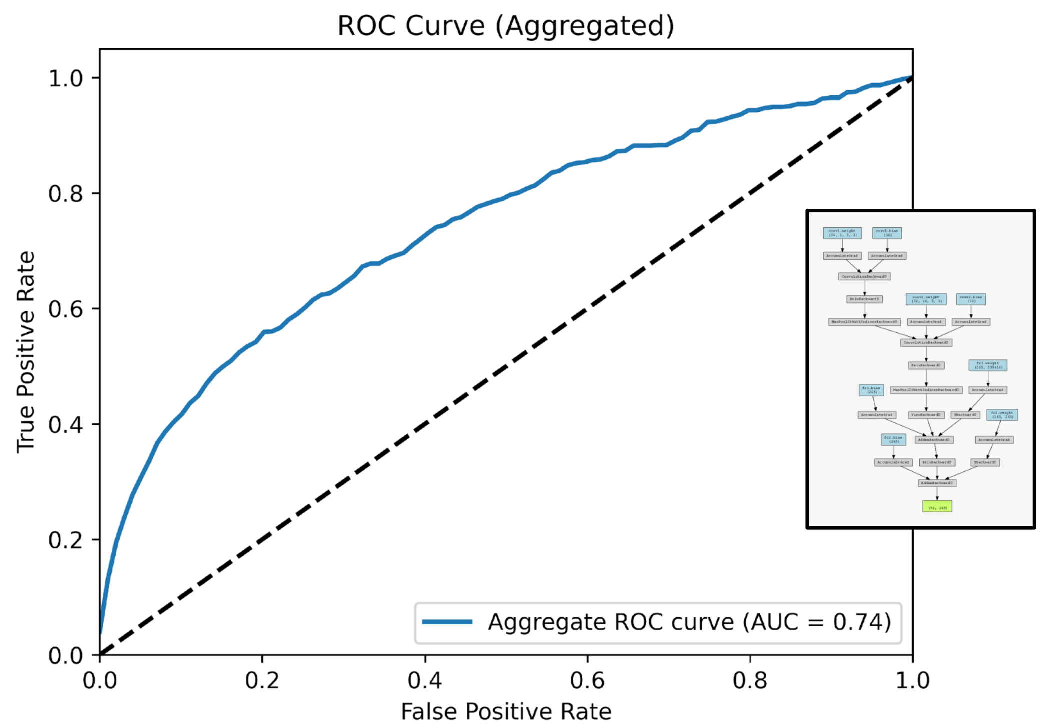
MODEL ACC.

ACCURACY VS. EPOCH



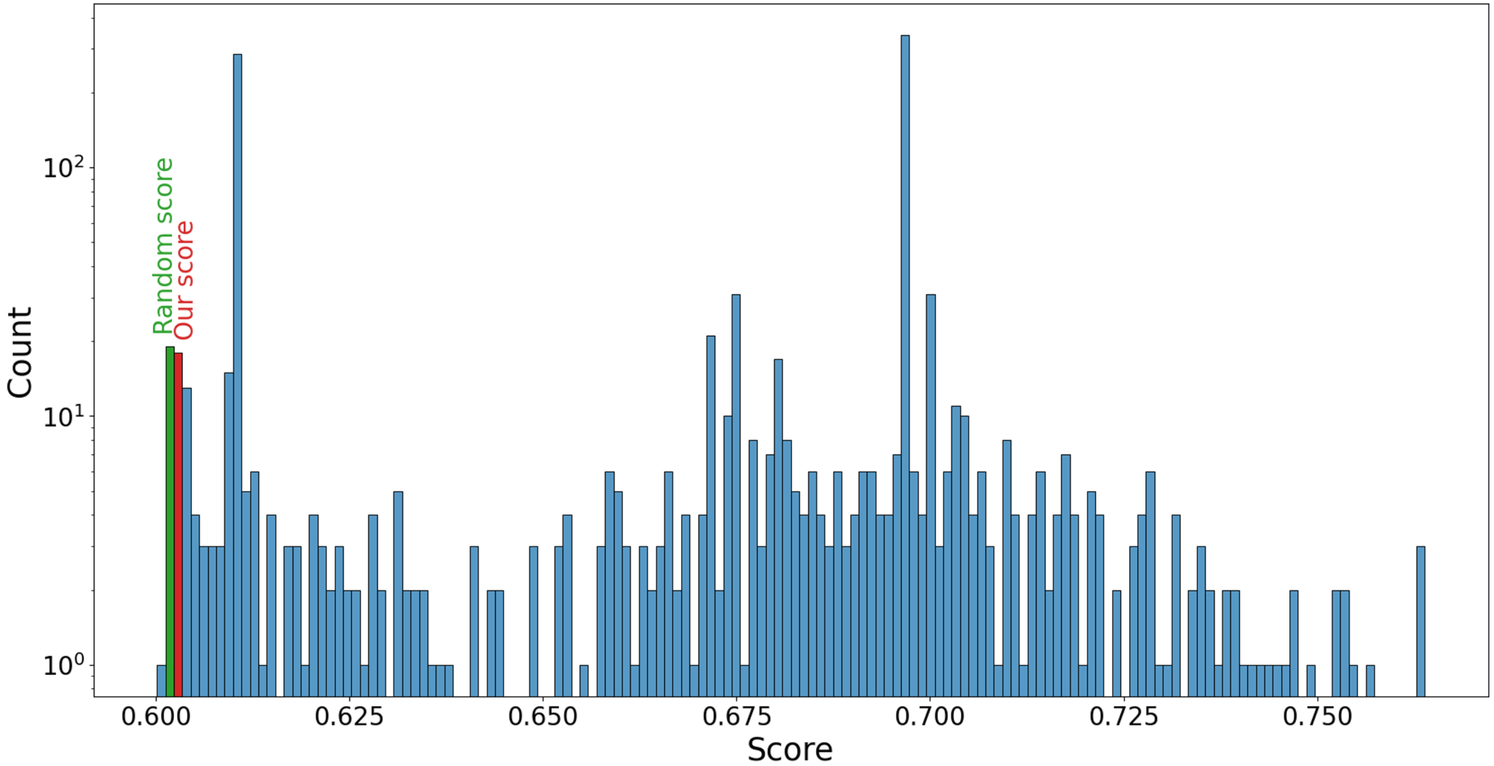
CNN SIMPLE

BETTER THAN RANDOM PERFORMANCE



CNN SIMPLE

DID WE WIN ON KAGGLE? No.



OUTLINE

PROBLEM

DATA VISUALIZATION

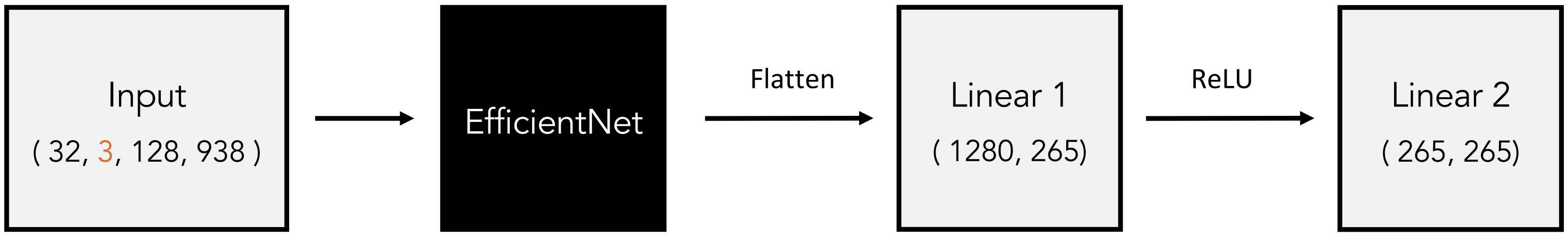
CNN HOMEBREW

CNN PRETRAINED

CONCLUSION

CNN PRETRAINED

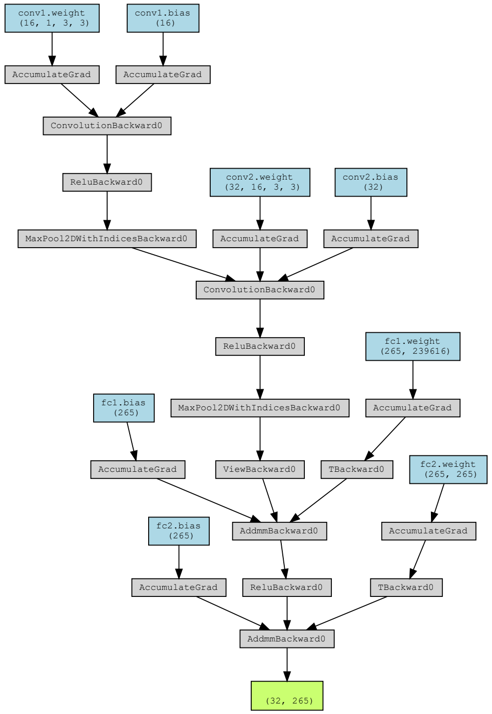
WITH EfficientNet



LR = 0.001
CROSS ENTROPY LOSS
ADAM OPTIMIZER

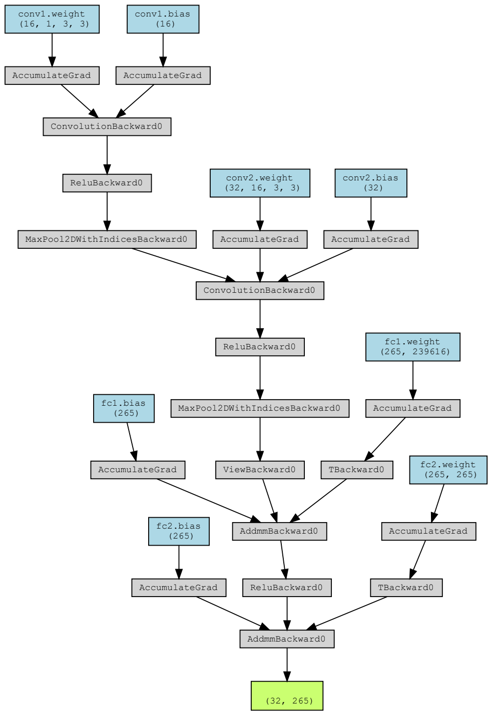
CNN COMPARISON

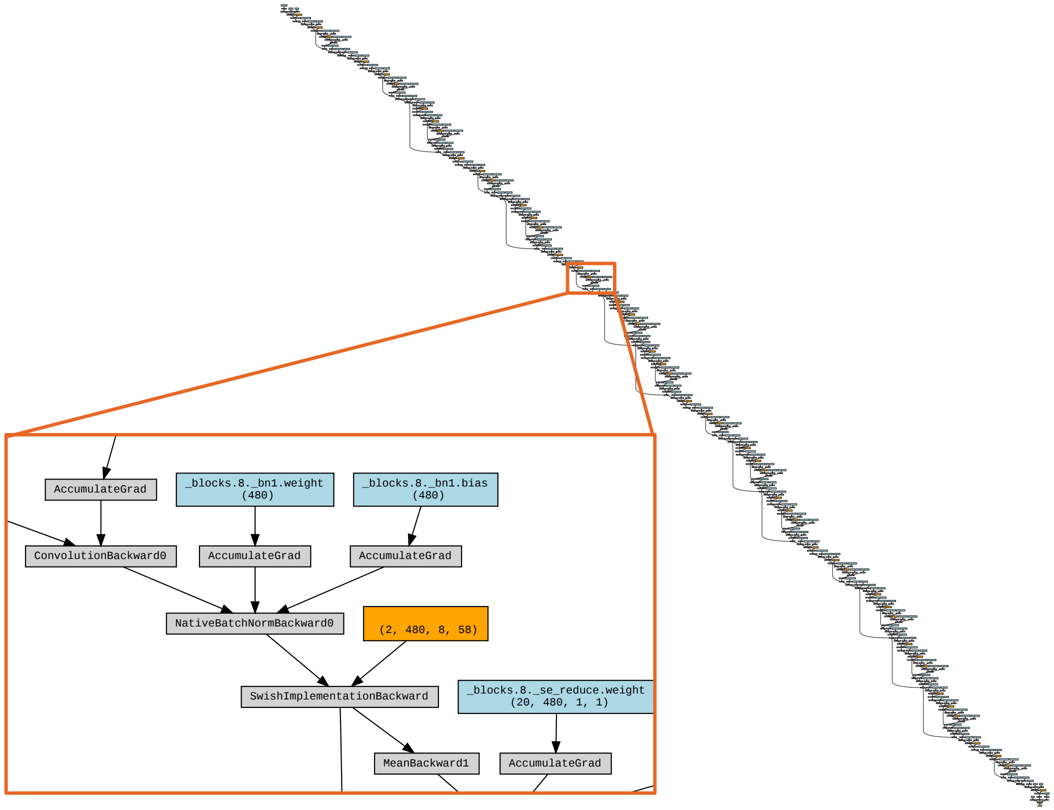
HOMEMADE VS. PRETRAINED



CNN COMPARISON

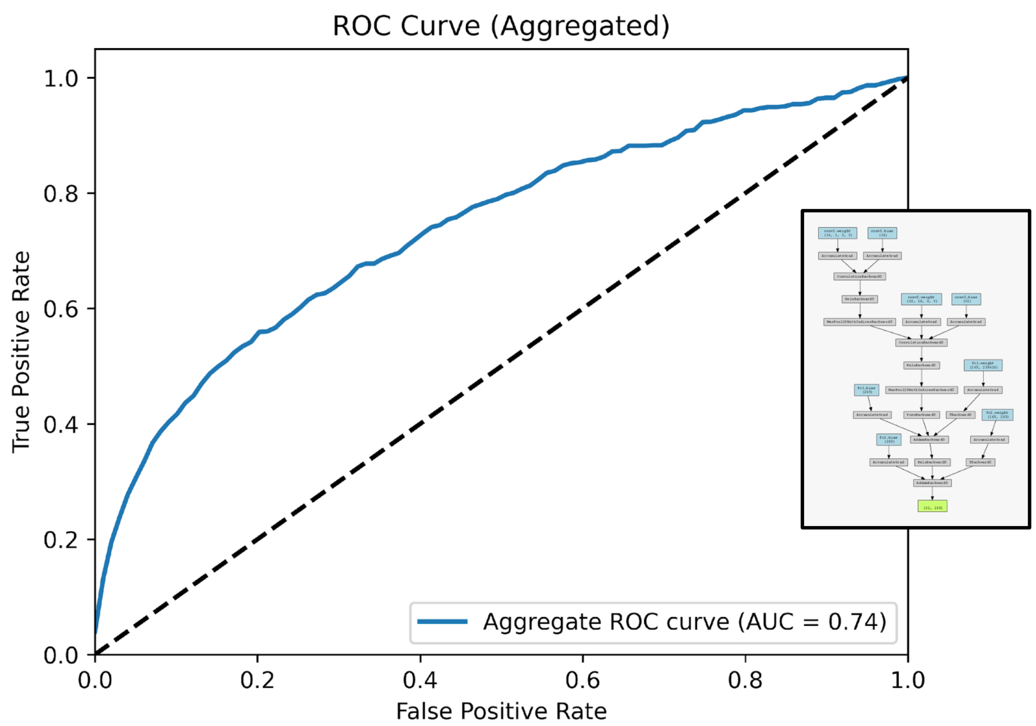
HOMEMADE VS. PRETRAINED





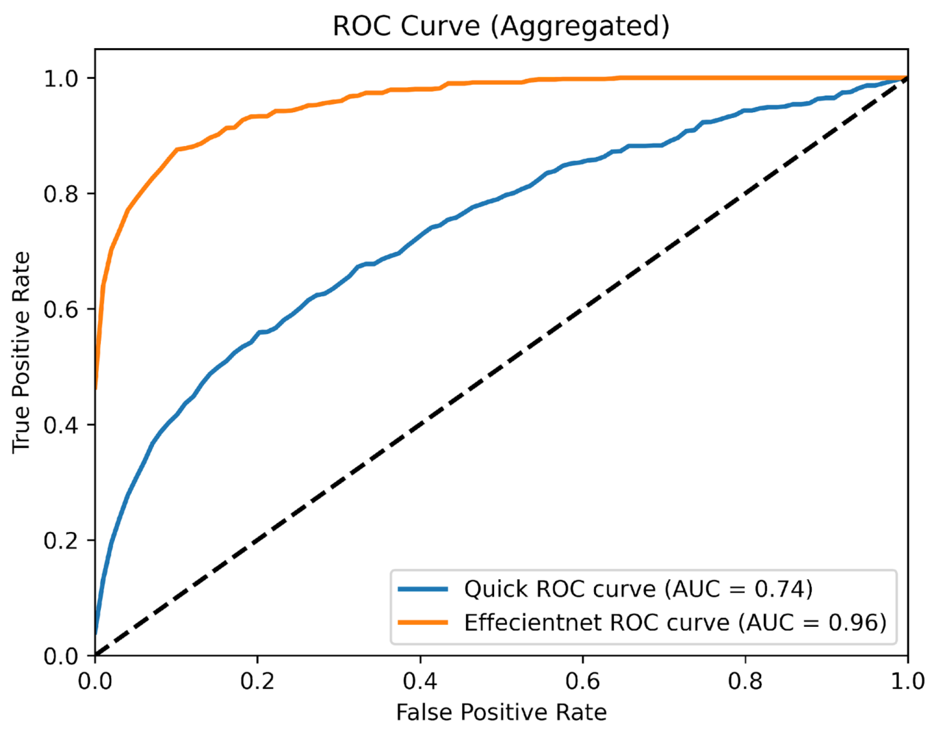
CNN PERFORMANCE

HOMEMADE



CNN PERFORMANCE

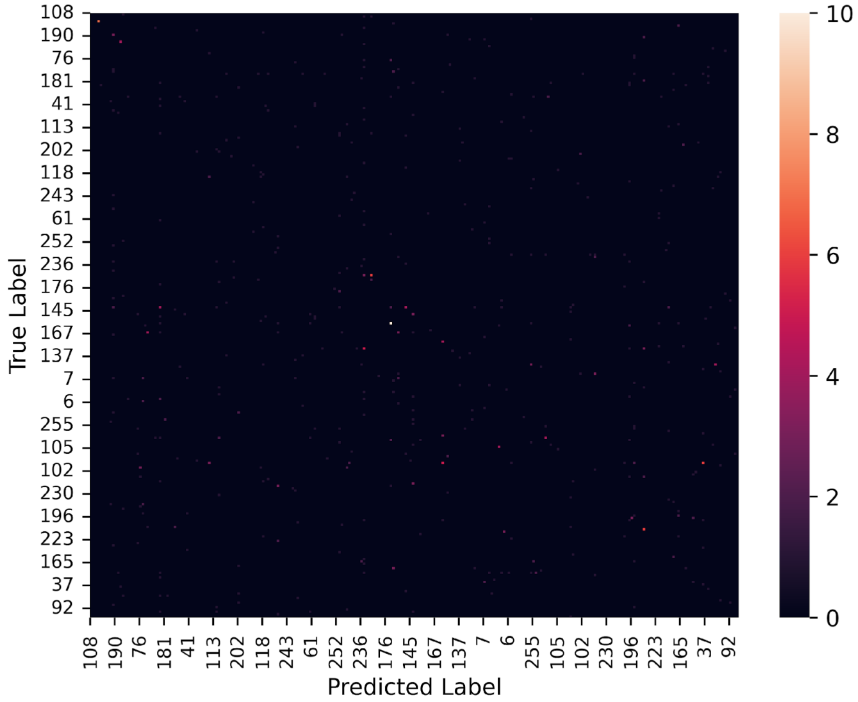
HOMEMADE VS. PRETRAINED



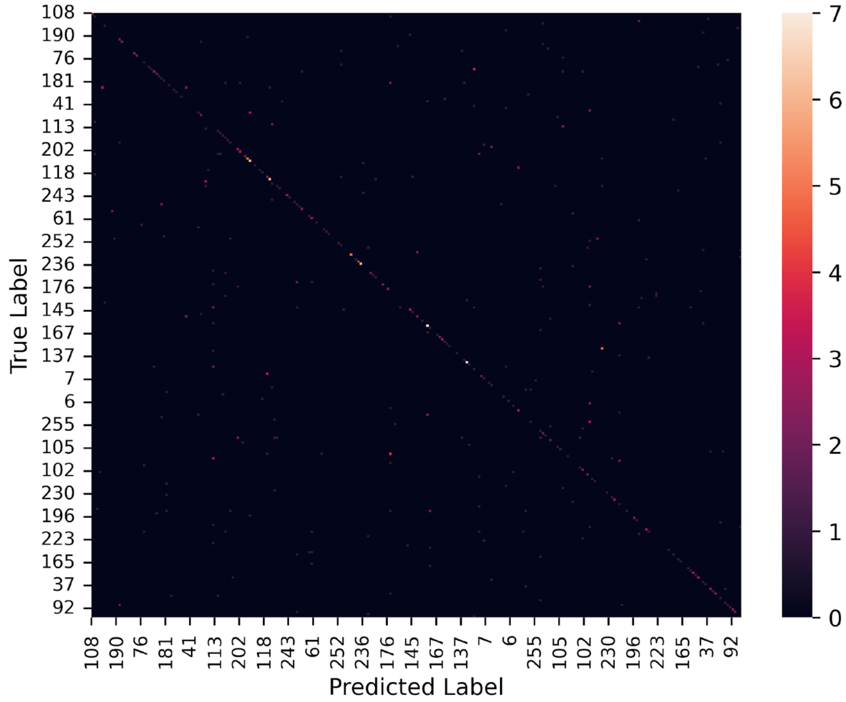
CNN PERFORMANCE

CONFUSION MATRIX

HOMEBREW

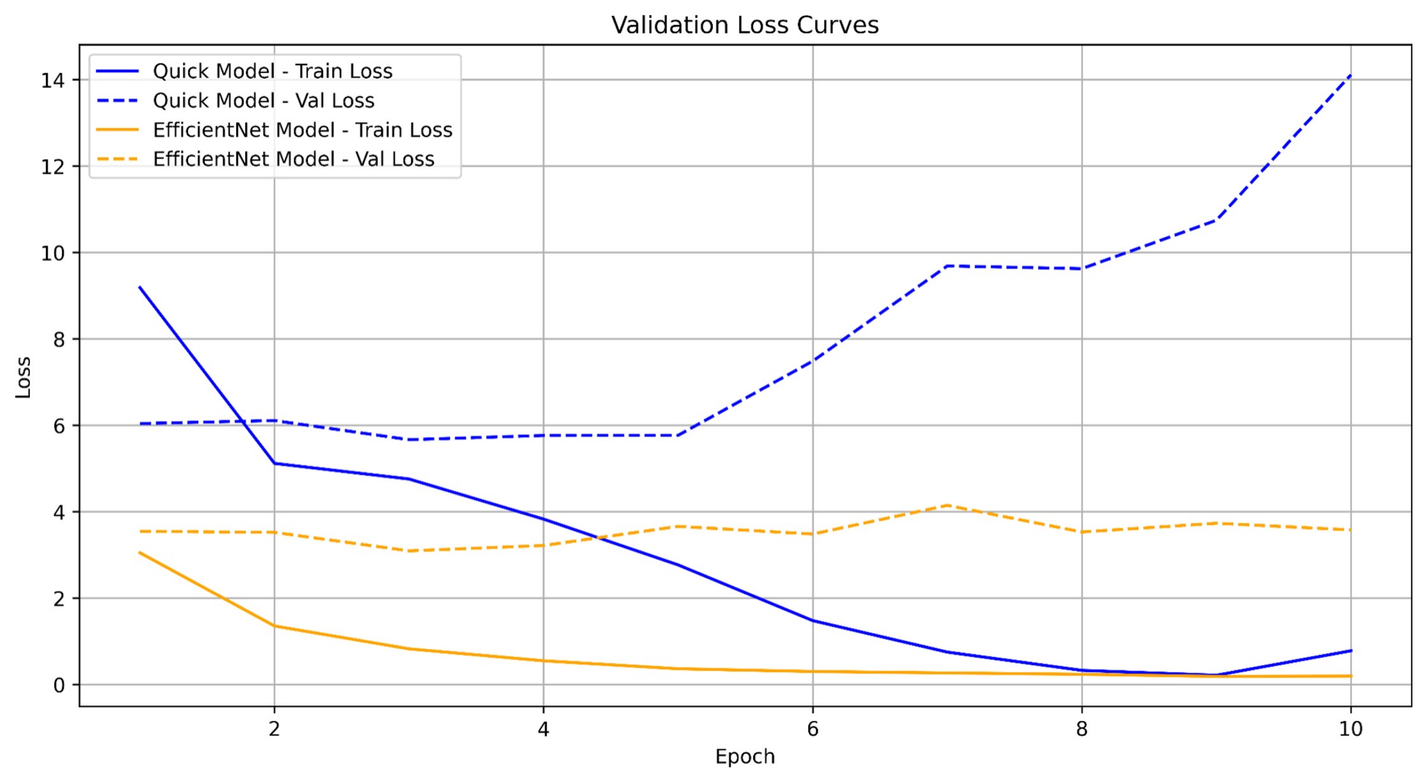


PRETRAINED



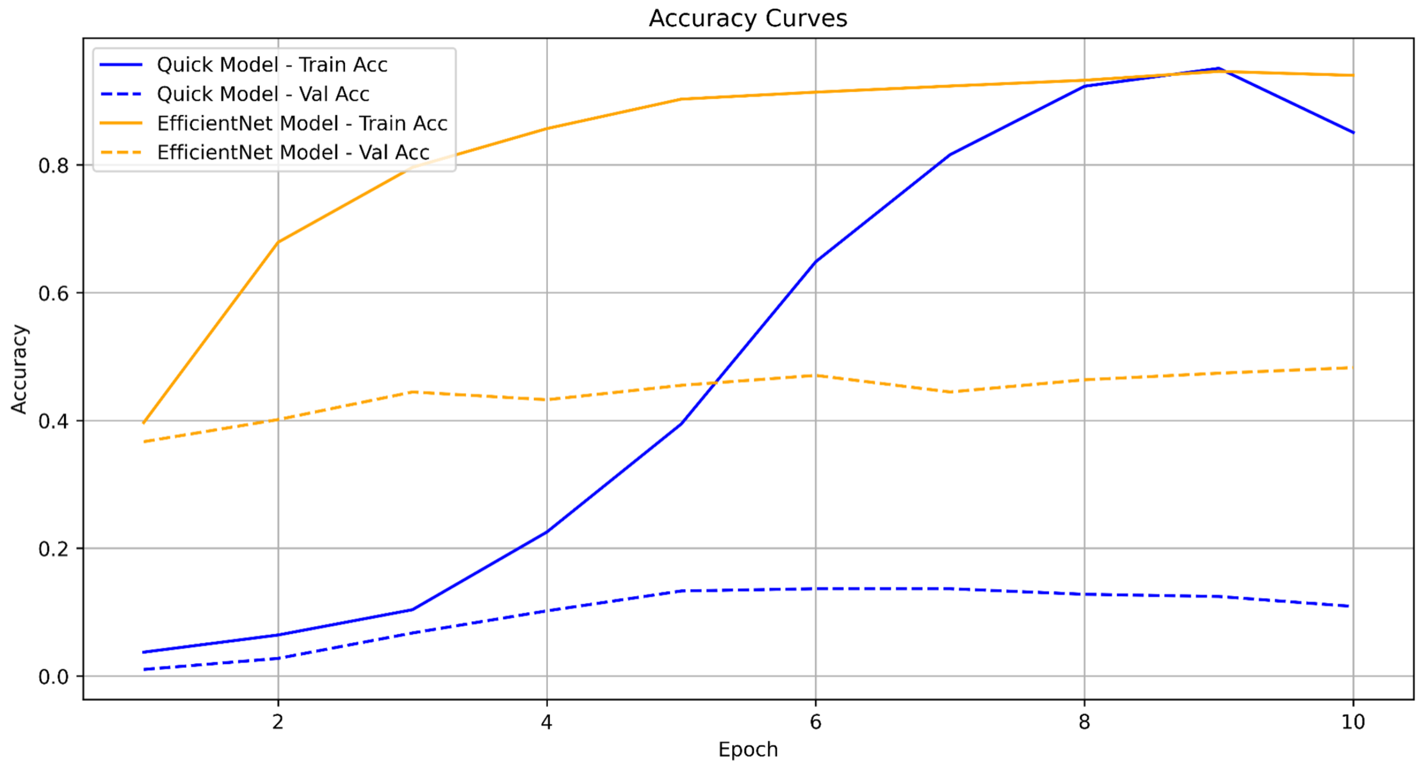
MODEL LOSS

LOSS VS. EPOCH



MODEL ACC.

ACCURACY VS. EPOCH



OUTLINE

PROBLEM

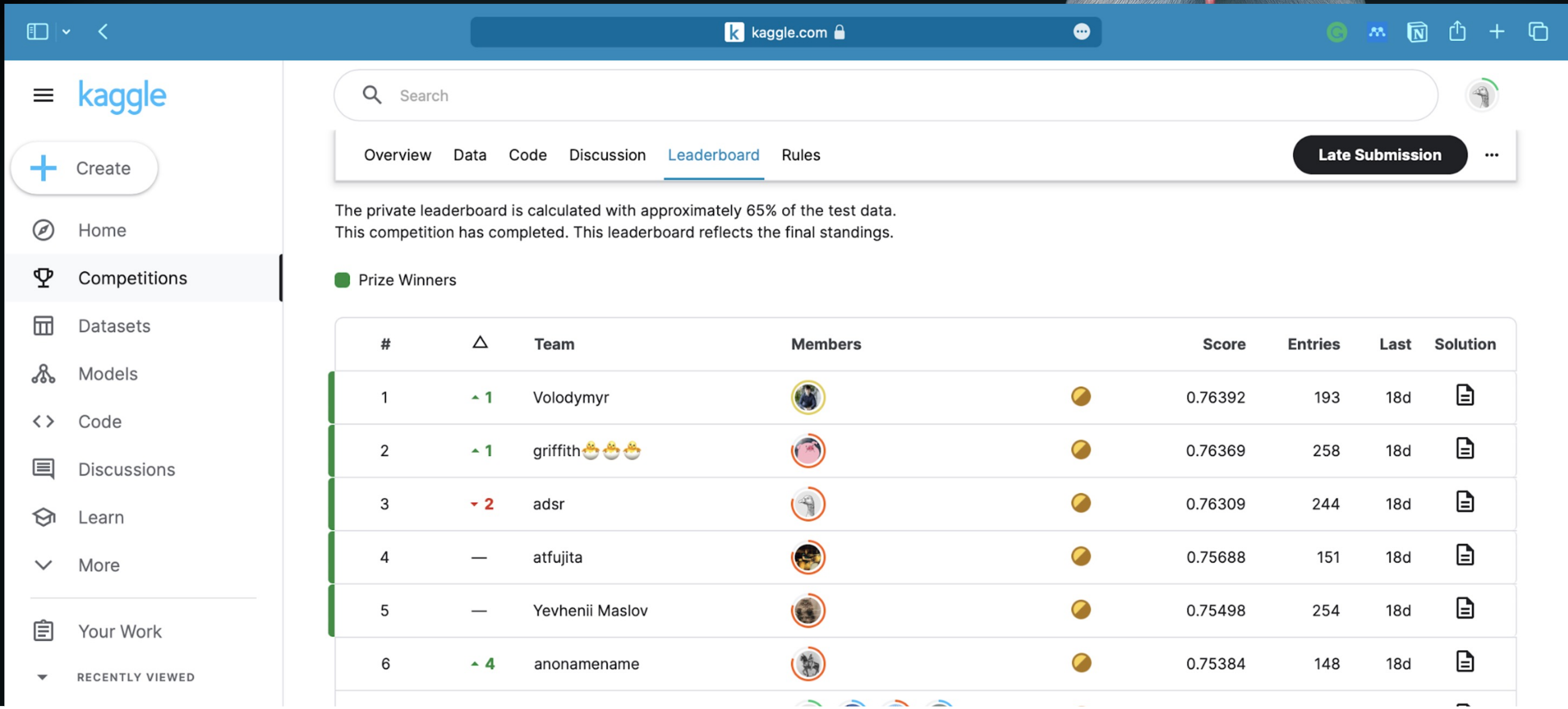
DATA VISUALIZATION

CNN HOMEBREW

CNN PRETRAINED

CONCLUSION

REVISITING
KAGGLE LEADERBOARD



The private leaderboard is calculated with approximately 65% of the test data.
This competition has completed. This leaderboard reflects the final standings.

Prize Winners

#	△	Team	Members	Score	Entries	Last	Solution
1	▲ 1	Volodymyr		0.76392	193	18d	
2	▲ 1	griffith🐣🐣🐣		0.76369	258	18d	
3	▼ 2	adsr		0.76309	244	18d	
4	—	atfujita		0.75688	151	18d	
5	—	Yevhenii Maslov		0.75498	254	18d	
6	▲ 4	anonamename		0.75384	148	18d	

REVISITING

KAGGLE LEADERBOARD

ACCURACY

- HOMEBREW – 14%
- PRETRAINED – 48%
- NO CROSS VALIDATION

IMPROVEMENTS

- LOAD AND TRAIN ON ALL DATA
- PRETRAIN ON OTHER BIRD AUDIO
- OTHER PRETRAINED MODELS


THANK YOU
FOR YOUR ATTENTION

APPENDIX

APPENDIX
CHOICES AND METHODS

Workflow

Steps and ideas

We first created functions to load and convert the audio data to mel spectrograms. We tried to get a boosted decision tree to work with data from feature analysis of the data, however, this did not work. We then created our first neural network to have a proof of concept to work on, for this we first created a lass of network that we could expand on quickly (1st CNN slide 66), this was too complex to start with so we decided to just make a simple CNN that we hardcoded the dimension (2nd CNN slide 67).

We also ran into memory issues with the data and the data was trimmed to a maximum of 50 files per bird. Some birds also had very few audio files and we tried to supplement them with audio files from xenocanto.com aiming to get at least 5 files per bird where possible. We then made our convolutional network using Pytorch to see if it was possible to predict the bird from audio. We then wanted to try and see if we could train and predict on different dimensions (3rd CNN slide 68), but this attempt failed and we chose to ignore it to focus on other parts. Our last CNN was our most successful one, which used a pre-trained one (4th CNN slide 69). The reason for creating this was to improve the performance by creating a new network using a pre-trained network based on EffecientNet, where we used 'EfficientNet b0'.

Memory problems

Loading data to CPU and GPU

One of our major problems was dealing with memory issues, either on the CPU or GPU. Our final solution was a compromise between the total computation time for loading in data, as well as the amount of data used. While not being the most optimal way to solve our problem it worked, loading the current dataset in segments of 15 seconds results in a memory usage of around 14-15GB RAM, while the data we load onto the GPU is around 8GB.

A better solution to solving our memory issues would be to change our pipeline for how and when we load the data. A solution we wanted to implement but didn't have time for was to first create a separate validation set, and then load in training data at the start of each epoch. This would allow us to load the entire dataset, without trimming it.

CNN Input

Data dimensionality

The input to the CNN needed to be the same for all audio files regardless of their length. This was ensured by using the same parameters in the pytorch function converting the audio waveform to a mel spectrogram for all audio files. For the dimensionality to be the same across all inputs to the CNN the audio which was converted to mel spectrogram also needed to always be the same length. This was done by cutting the audio files longer than 15s into 15s segments and discarding the rest. If the audio file was less than 15s the audio was instead looped to be 15s.

Choosing models

HOMEMADE VS. PRETRAINED

First, a homemade model was made to get a simple model to work and classify the audio clips. This consisted of 2 convolutional layers and 2 linear layers. The performance for the homemade model was better than random but far from good. To get a better performance it was decided to create a model which uses a pretrained model. It was thought that this could result in better performance by having a more complex model which could have undergone training on more data than we could do. The pre-trained model chosen was 'EfficientNet b0'. After the pre-trained model, there is a linear layer. The performance of the pre-trained model was much better than the simple model.

Dynamic CNN

Handling different training and prediction dimension.

Since the conditions for the birdCLEF competition were to guess on 5-second segments we tried making a dynamic CNN that could train on samples with a longer duration and still predict on ones of 5-second duration. The reason for this was that the 5-second segment duration had a worse performance compared to longer durations (15-30 seconds), and it makes sense since we would be training more on background noise instead of segments with actual bird audio in it. But we did not manage to get this CNN to work with good accuracy, best attempts were >1% accuracy so the idea was down-prioritised so we could focus on other parts of the project.

Segment duration

HOMEMADE VS. PRETRAINED

In order for the input to the CNN to be uniform each audio clip was chosen to have the same duration. This was achieved by splitting audio files into 15s durations and discarding the leftover. For the audiofile under 15s, the audio file was looped until it the duration was 15s. 15s duration was chosen as a longer duration resulted in better training, as for shorter segments there could be segments with no bird calls. For durations over 15s, there were memory problems and 15s were therefore chosen.

Trust in performance

No cross validation

In the validation set only one file per bird was used. For some birds, the number and length of recordings were small which resulted in difficulty in evaluating the accuracy of individual birds. As the validation set was of limited size the performance varies slightly for each training instance of the model. The performance seemed to be accurate to +-1% but cross-validation was not performed to verify the performance of the models.

Stereo-Mono

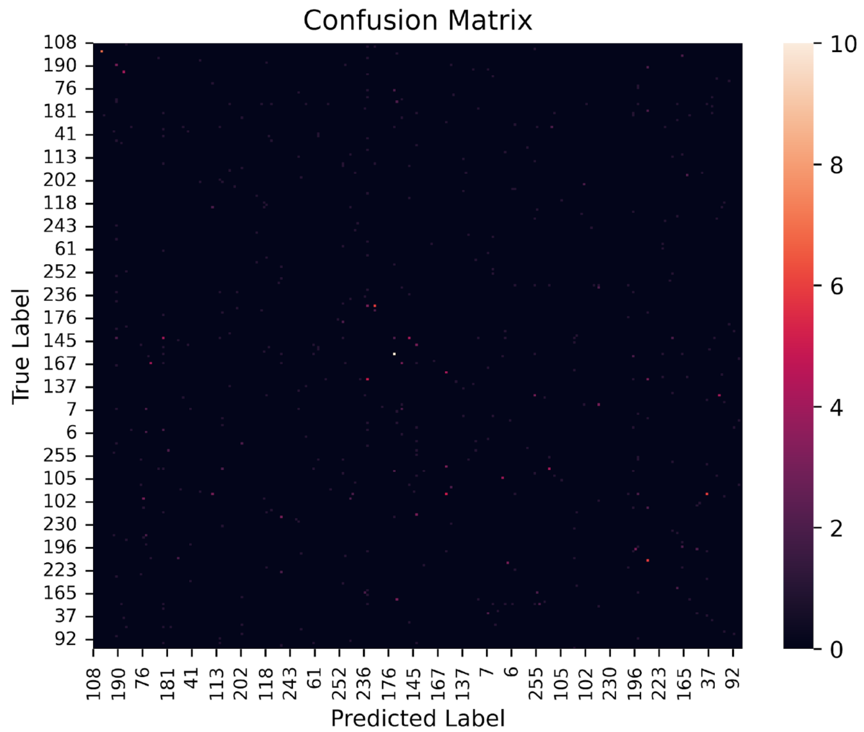
Audio file formatting

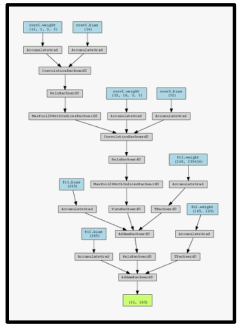
The audio files we downloaded from xenocanto.com to supplement birds with low amount of audio were stereo files, while the files from birdCLEF were mono. The stereo files were converted to mono.

APPENDIX
OTHER FIGURES

CNN SIMPLE

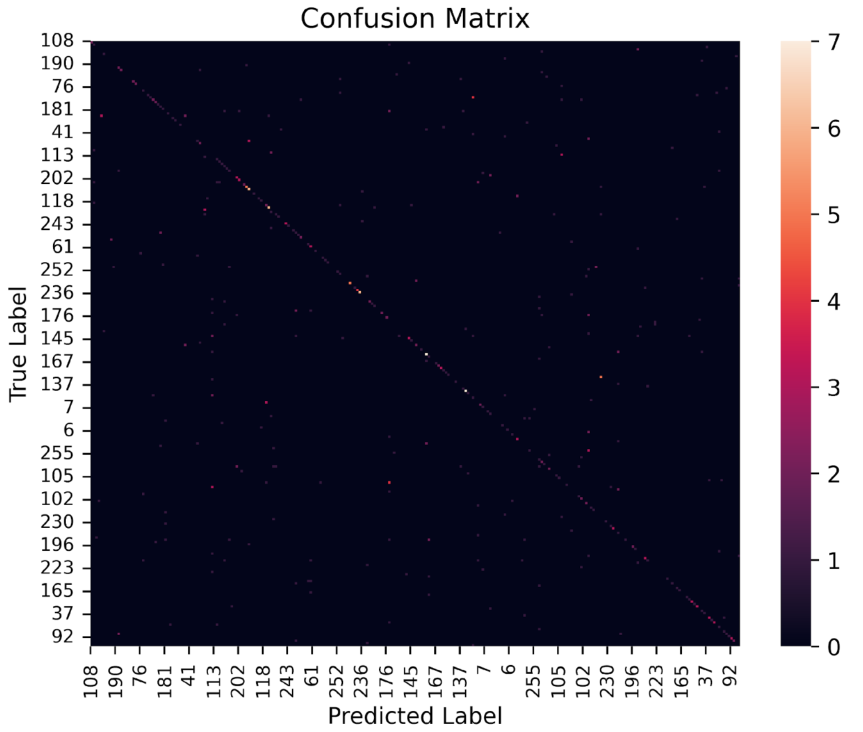
CONFUSION MATRIX





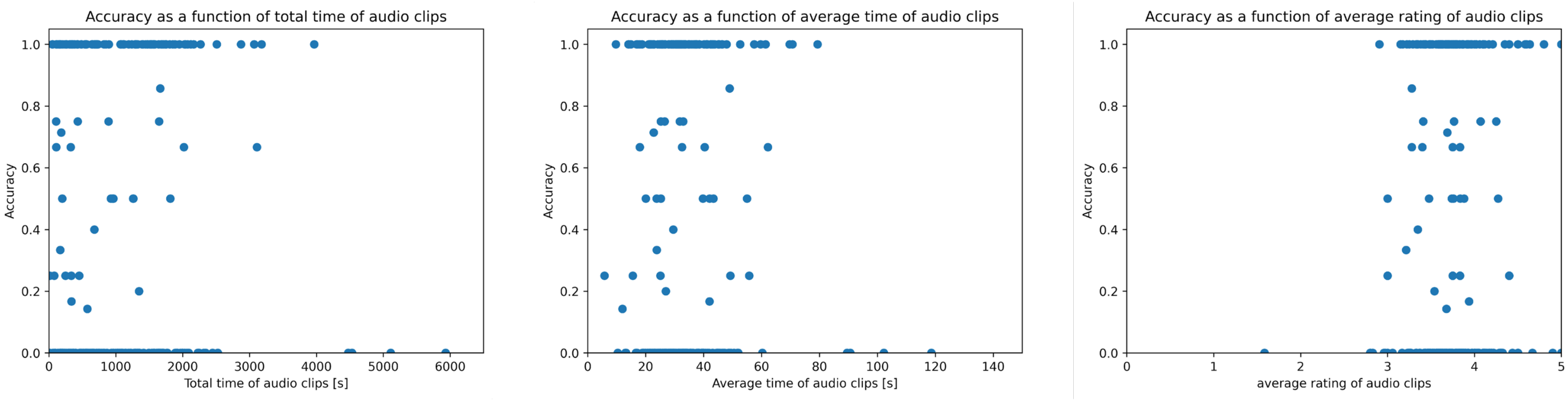
CNN PRETRAINED

CONFUSION MATRIX



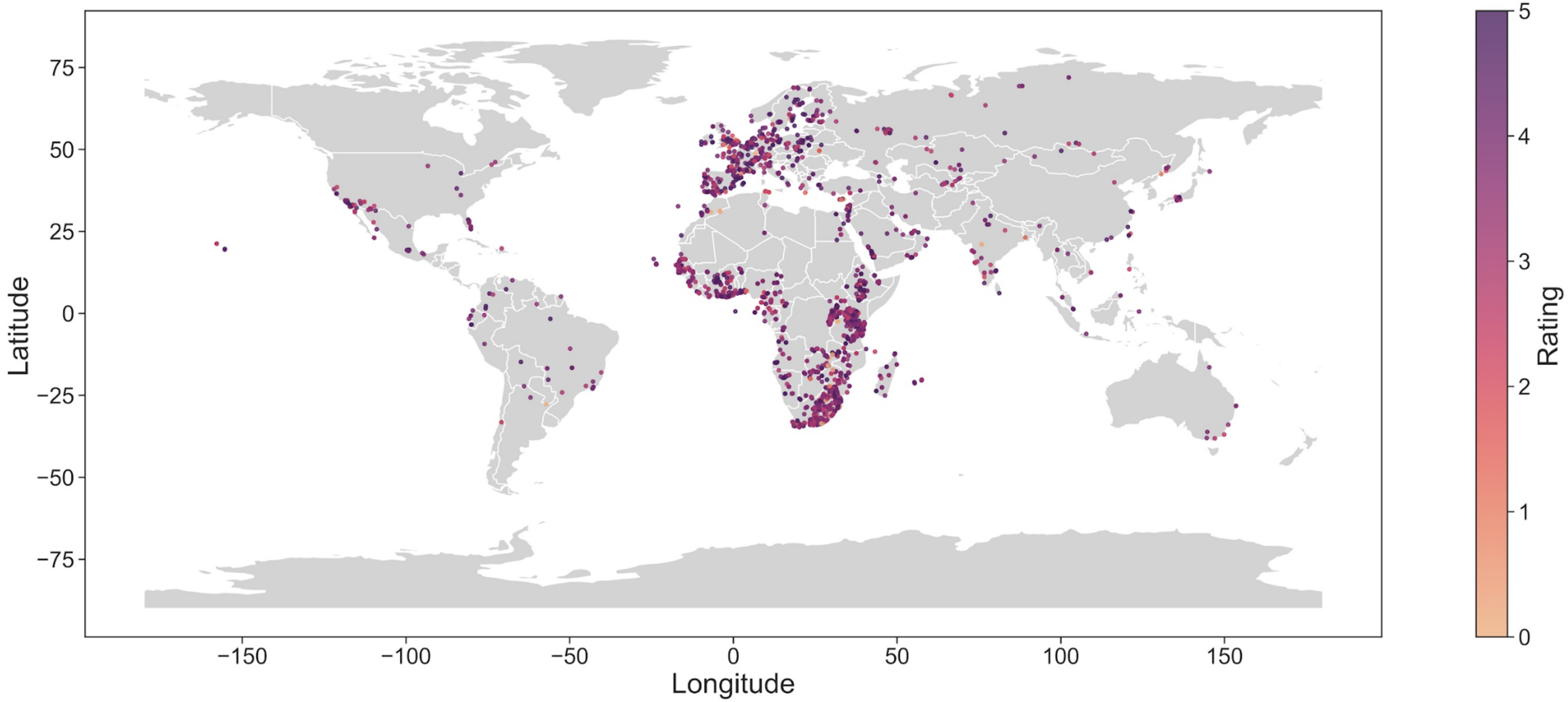
CNN ACCURACY

VS. AMOUNT OF DATA



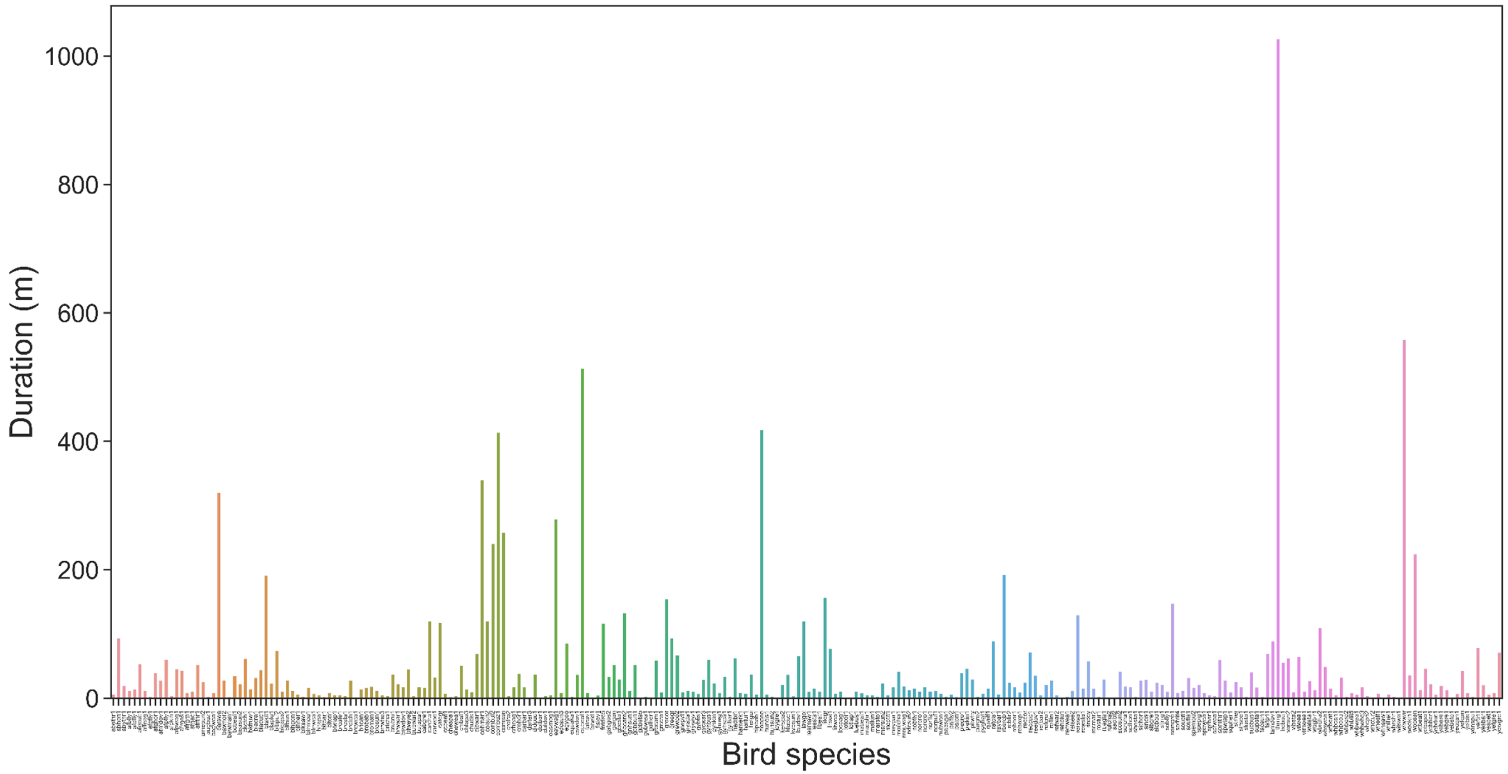
WHERE

ARE THE BIRDS FROM?



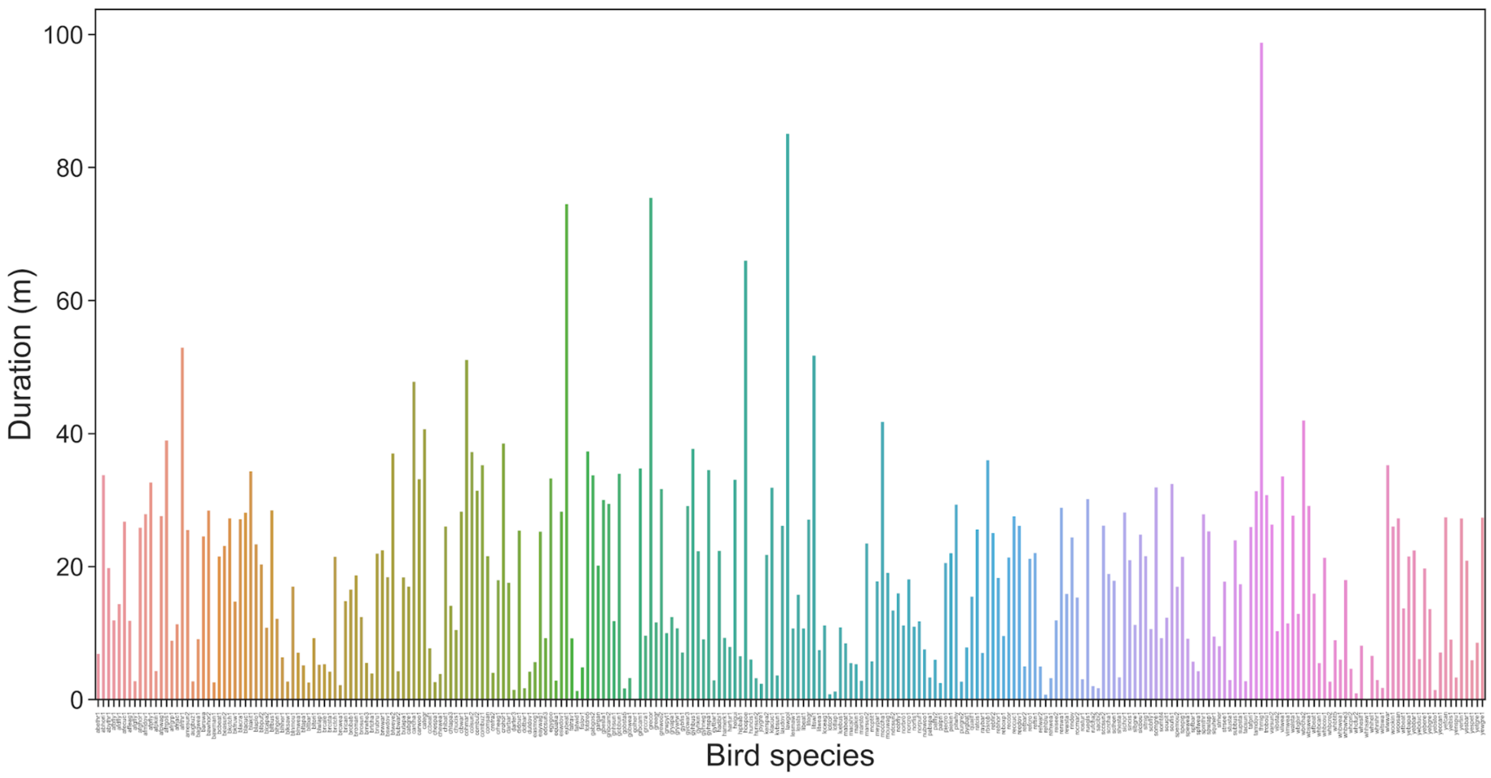
MINUTES OF AUDIO

PER BIRD (NOT TRIMMED)



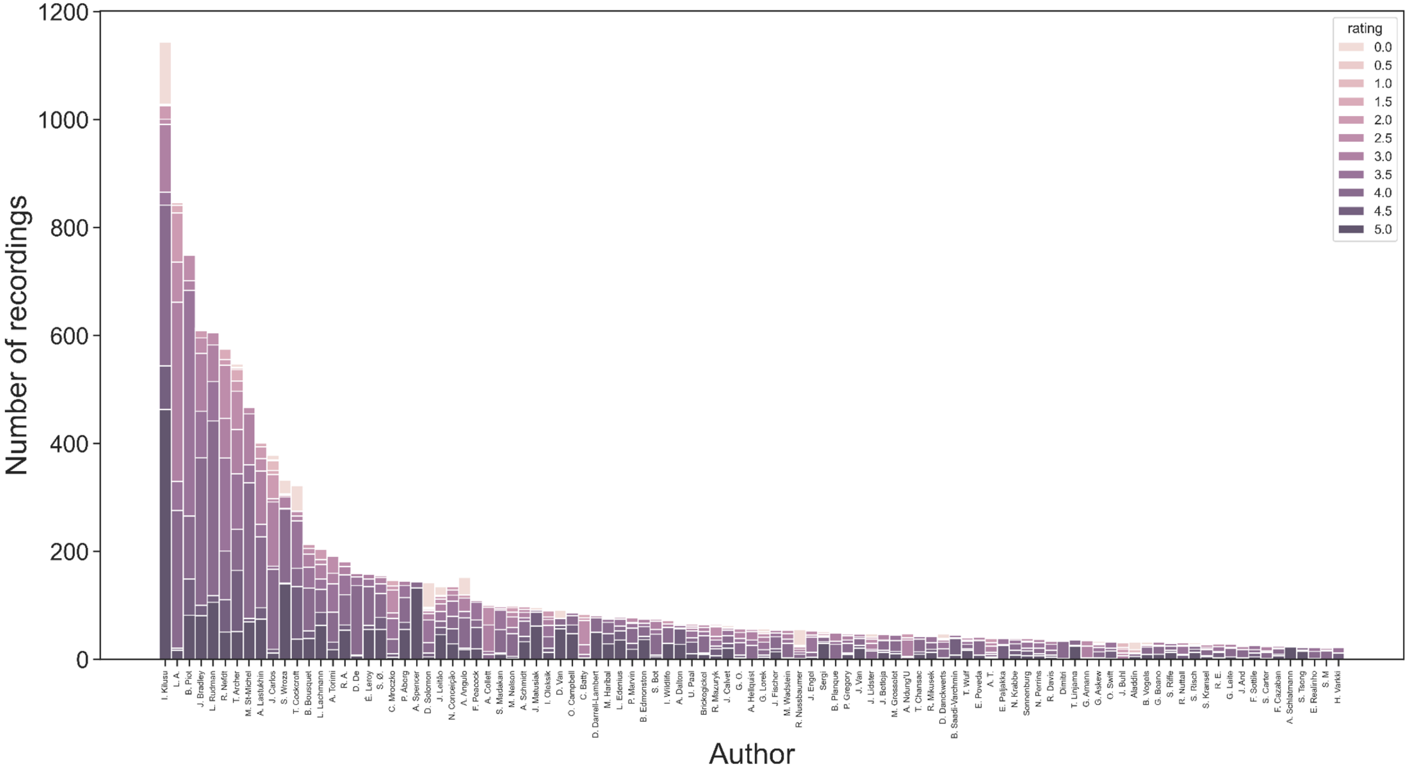
MINUTES OF AUDIO

PER BIRD (TRIMMED)



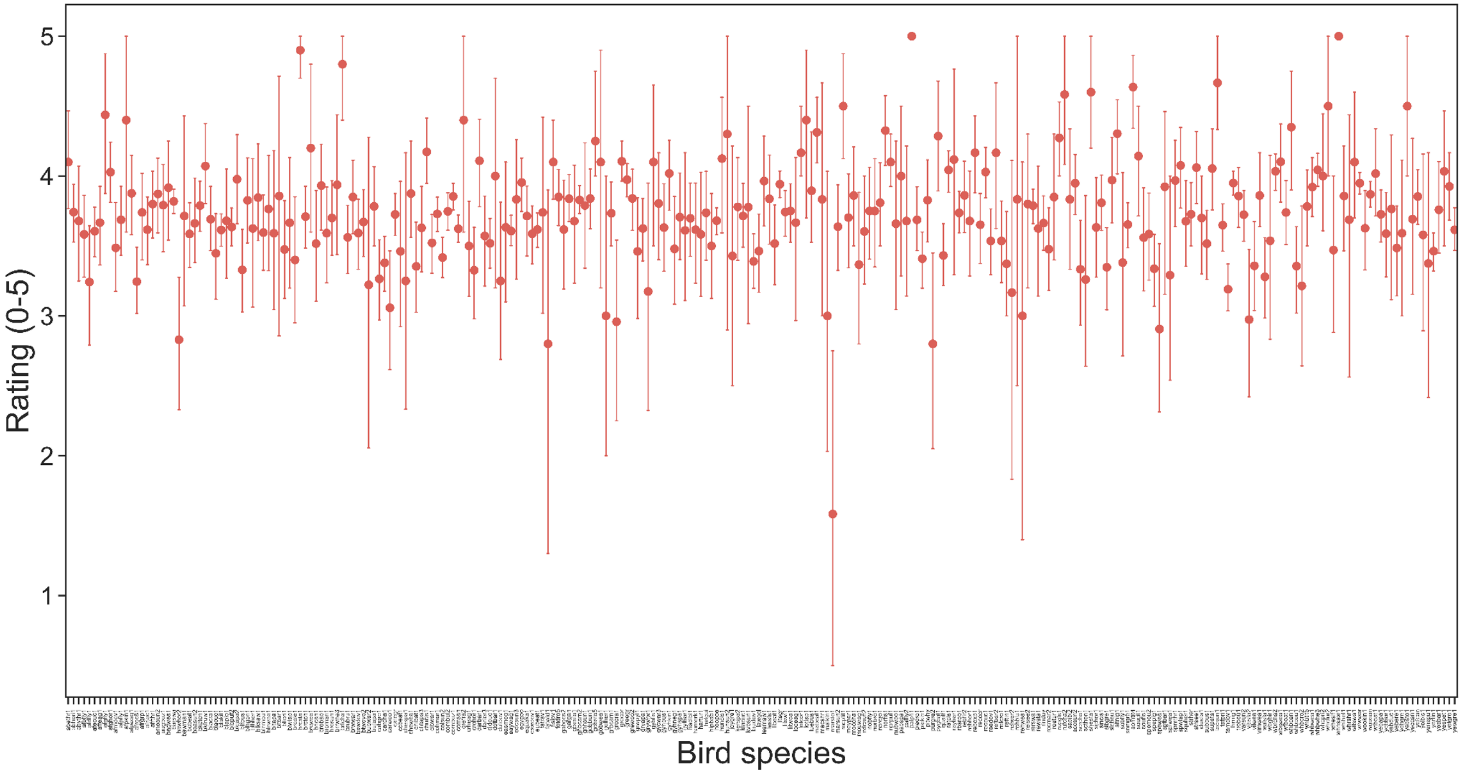
OF RECORDINGS

PER AUTHOR



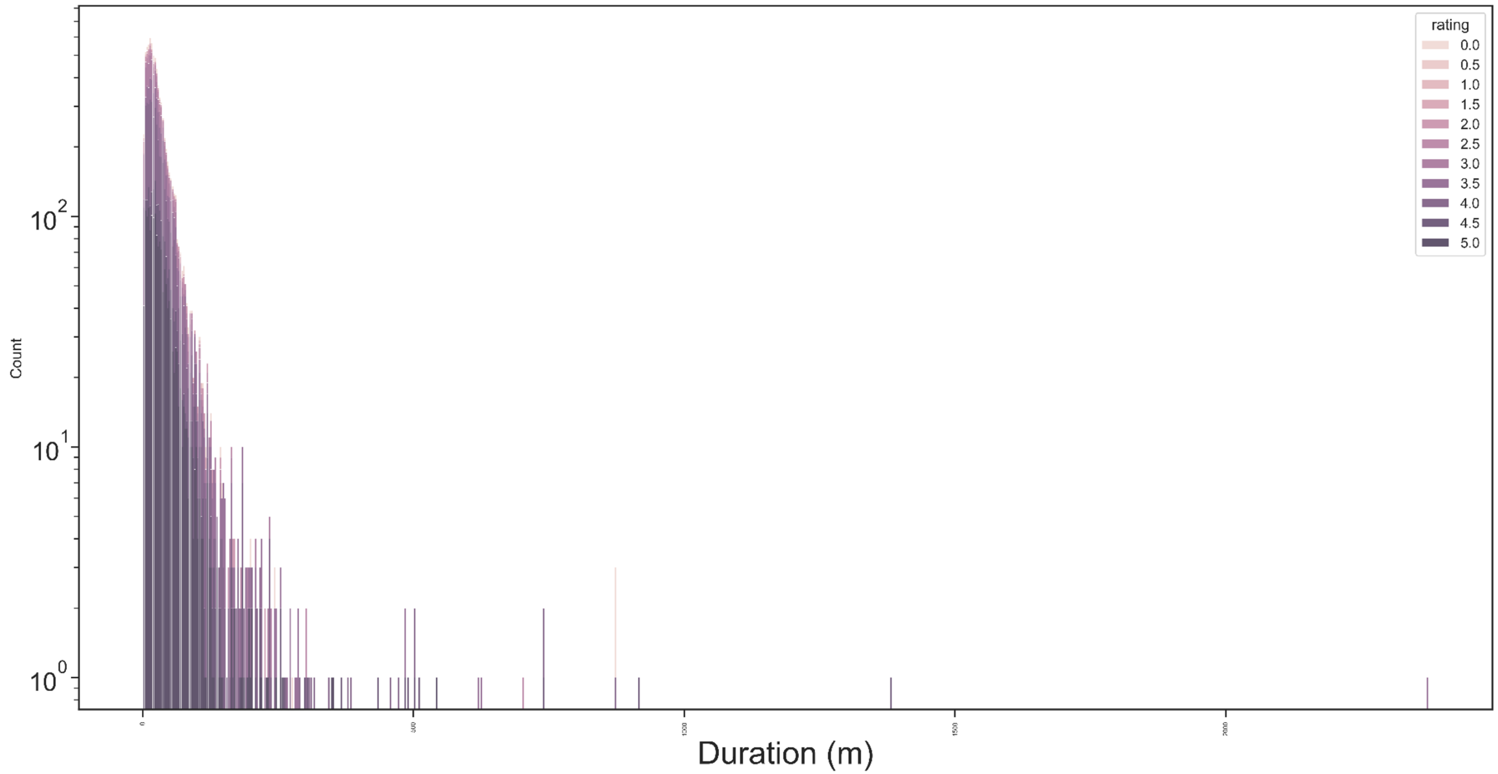
MEAN RATING

PER BIRD



DURATION

HISTOGRAM



APPENDIX
FEATURE EXTRACTION

FEATURE EXTRACTION

CORROLATION PLOT

We wanted to extract features from the audio files. This was done by using a multible of different librosa functions. The functions were:

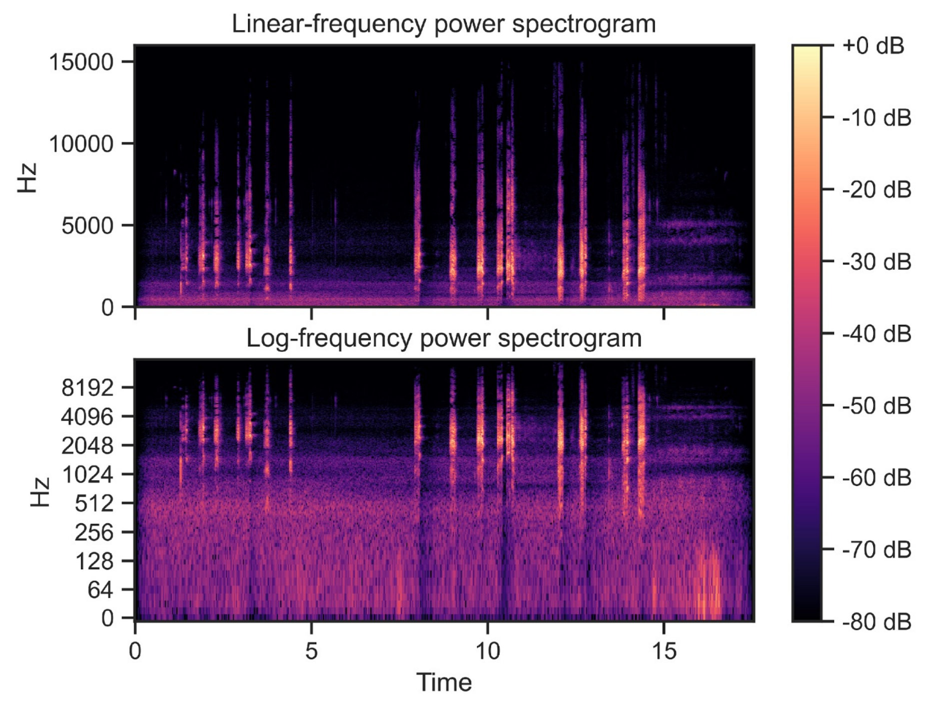
- Spectral bandwidth
- Spectral centroid
- Zero crossing rate
- Spectral rolloff
- MFCC
- Delta
- Chroma stft
- Chroma cqt
- Chroma cens
- Tonnetz
- Melspectrogram

From the result of all these feature analysis we determined the following statistics:

- Standard deviation
- Mean
- Min
- Max
- Median
- 1st quartile
- 3rd quartile
- Variance
- Mean absolute deviation
- Root mean squared

This resulted in a total of 110 features

FEATURE EXTRACTION
SHORT-TIME FOURIER TRANSFORM

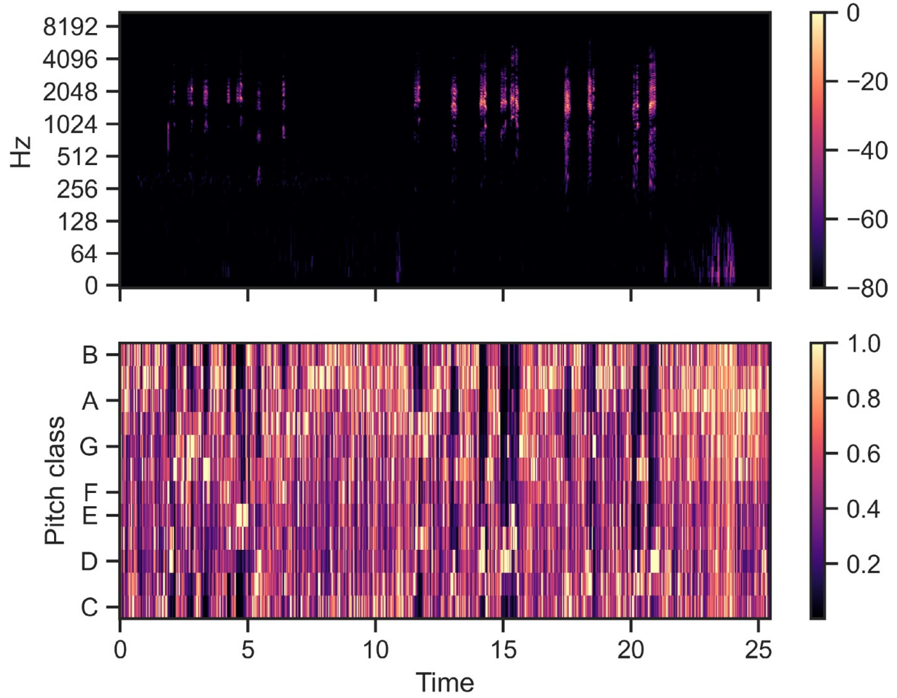


Here we plot the Short-time Fourier transform (STFT) of the audio file. Using the following equation:

$$\mathbf{STFT}\{x(t)\}(\tau,\omega) \equiv X(\tau,\omega) = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-i\omega t}\,dt$$

The STFT is a complex-valued function of two real variables, conventionally called time and frequency, representing the frequency content of the signal as it changes over time.

FEATURE EXTRACTION
CHROMA STFT

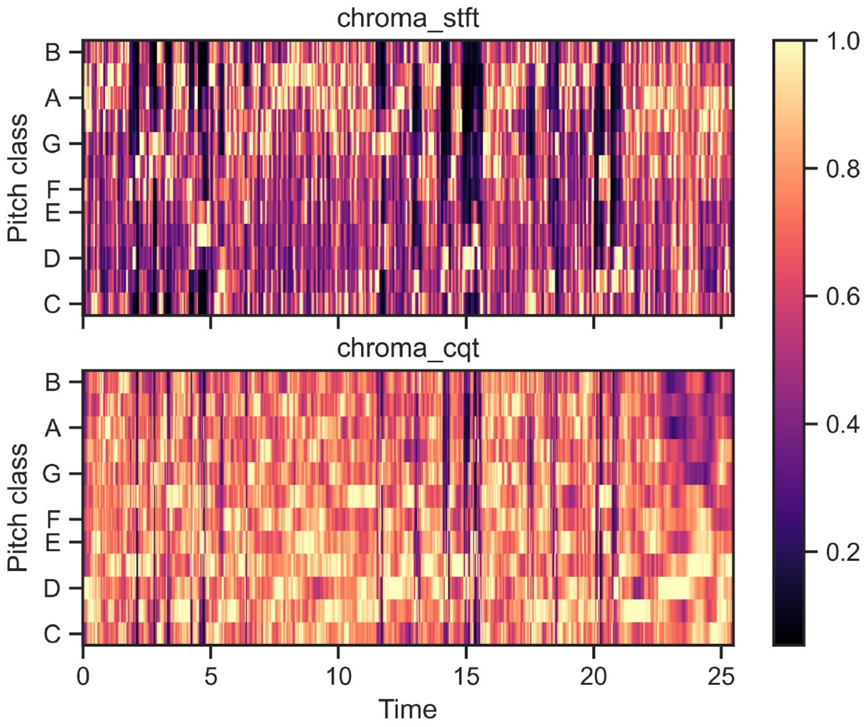


Compute a chromagram from a waveform or power spectrogram. This is closely related to the twelve different pitch classes.

Here we only see small structures in the data.

FEATURE EXTRACTION

LONG-WINDOWED SFTF



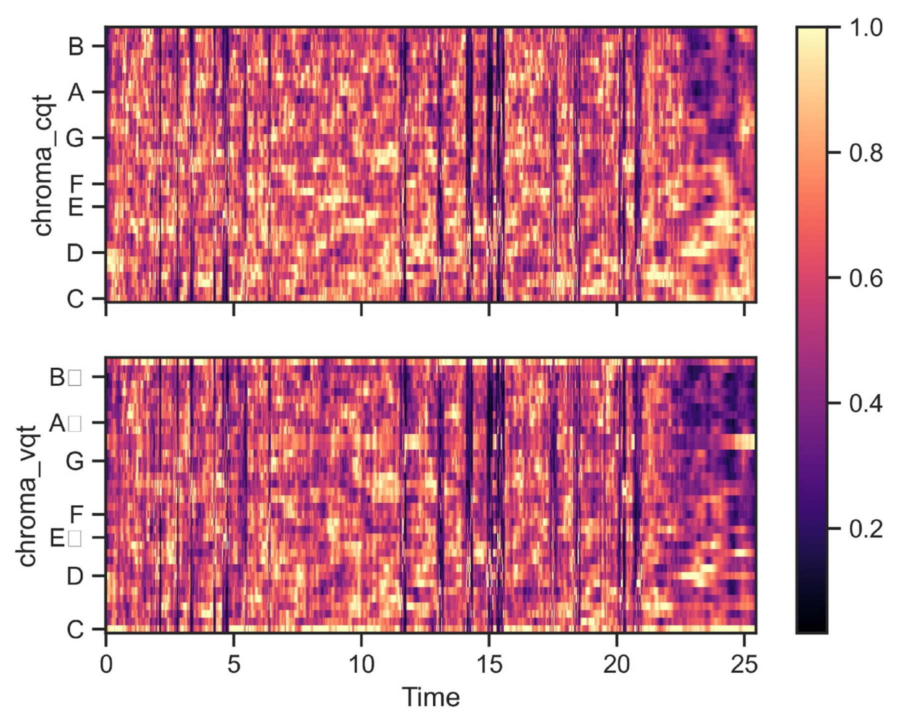
Compute a chromagram from a waveform or power spectrogram. This is closely related to the twelve different pitch classes.

Here we only see small structures in the data.

Furthermore, we calculate the Constant-Q chromagram.

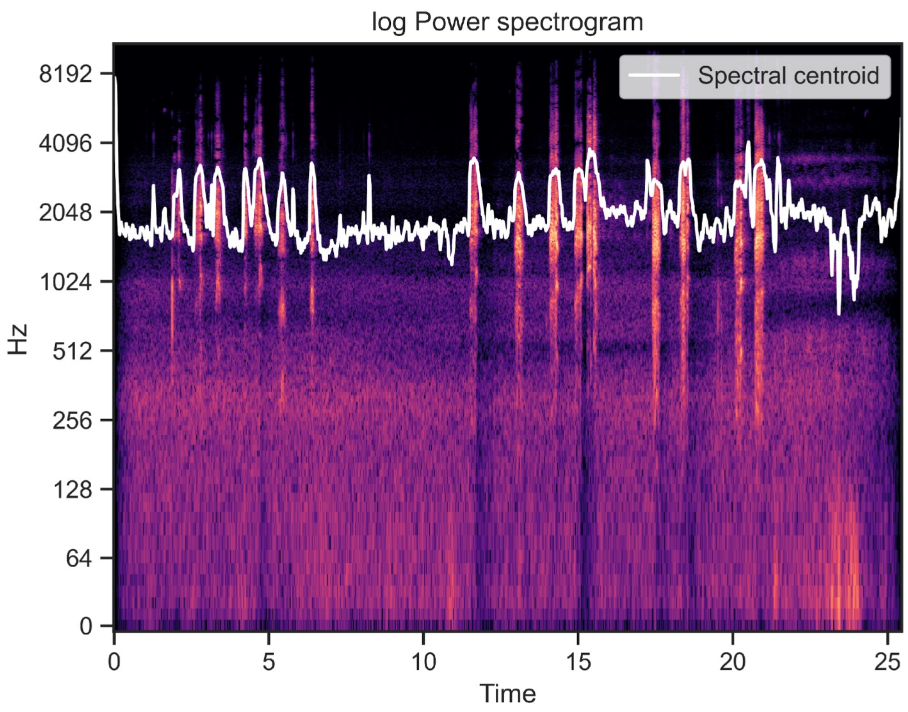
FEATURE EXTRACTION

CQT VS. VQT



We calculate the Constant-Q chromagram and Variable_Q chromagram. VQT differs from CQT, by not aggregating energy from neighbouring frequency bands.

FEATURE EXTRACTION
SPECTRAL CENTRIOD

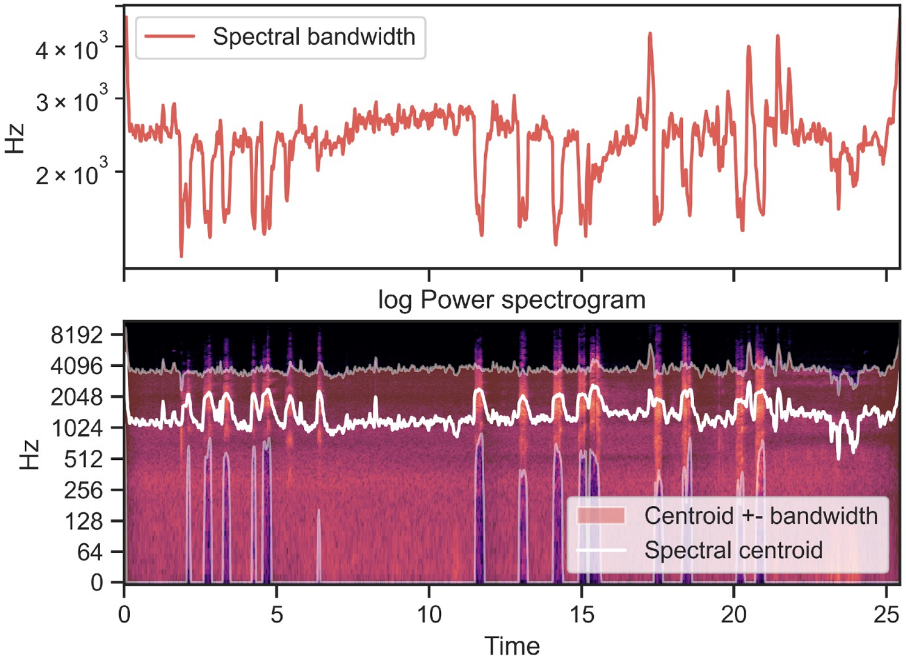


We compute the spectral centroid as the mean of each frame of the magnitude spectrogram when normalized.

```
centroid[t] = sum_k S[k, t] * freq[k] / (sum_j S[j, t])
```

FEATURE EXTRACTION

SPECTRAL BANDWIDTH

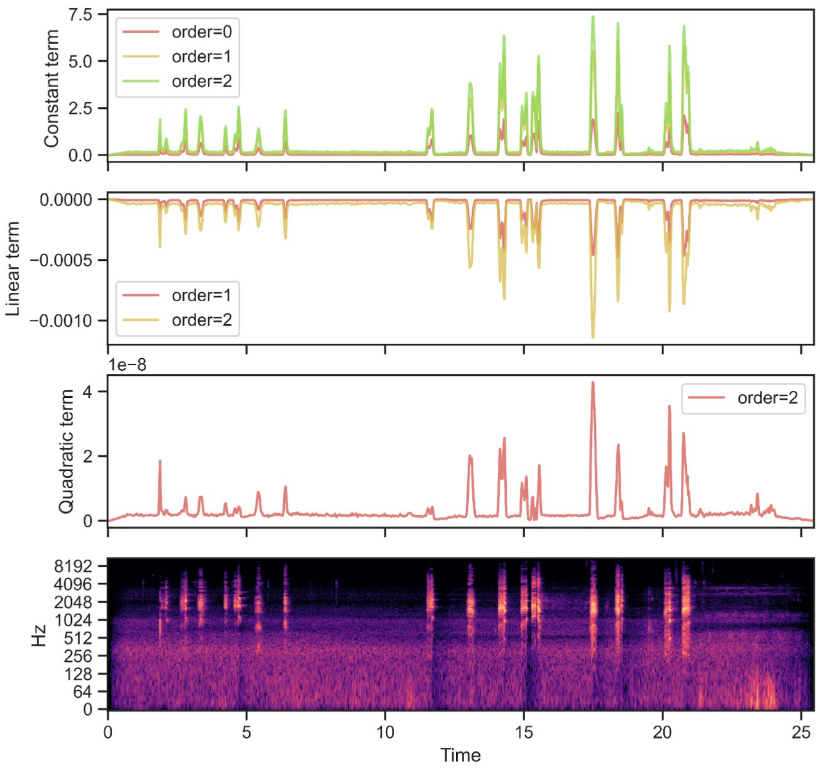


We compute the spectral centroid as the mean of each frame of the magnitude spectrogram when normalised. Furthermore, we calculate the spectral bandwidth.

```
(sum_k S[k, t] * (freq[k, t] - centroid[t])**p)**(1/p)
```

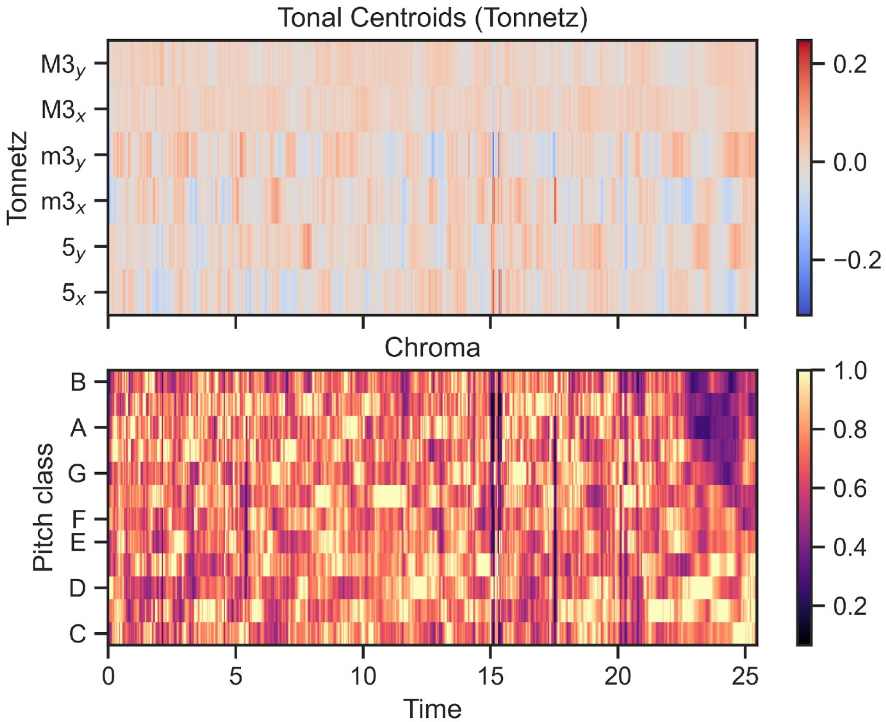
FEATURE EXTRACTION

HOMEMADE VS. PRETRAINED



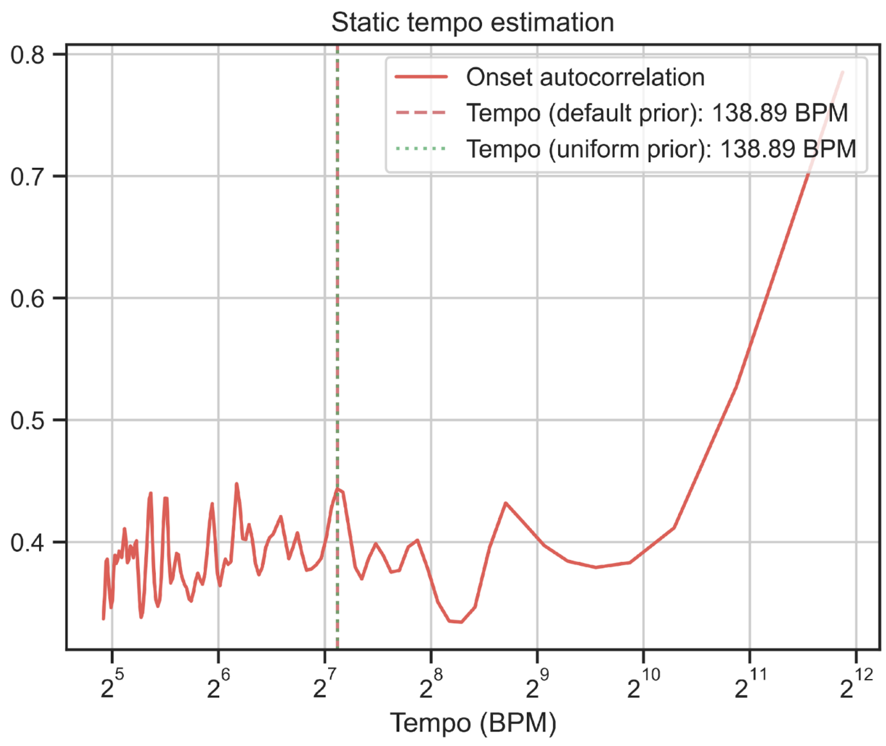
Fitting nth-order polynomial to the columns of the spectrogram.

FEATURE EXTRACTION
TONNETZ



Here we calculate the tonal centroid features (tonntz).

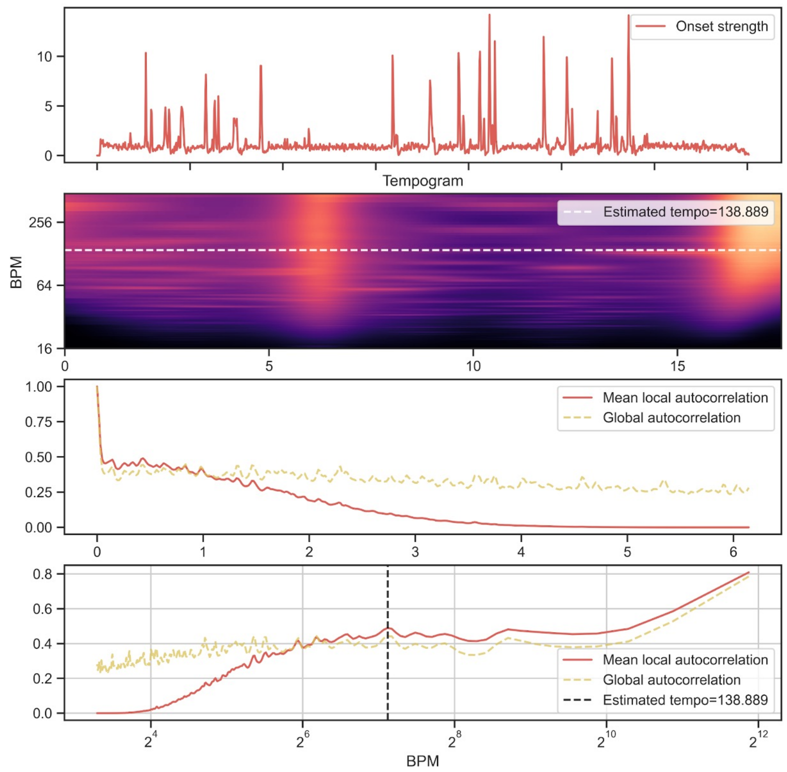
FEATURE EXTRACTION
TEMPO (BPM)



Estimation the tempo of the bird audio files.

FEATURE EXTRACTION

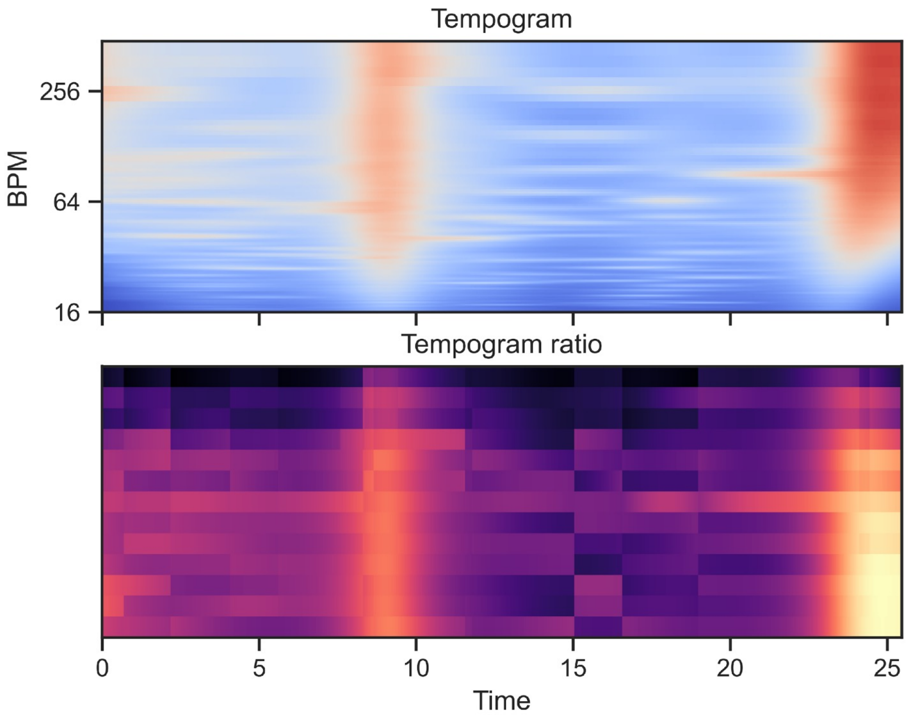
TEMPOGRAM



Estimation the full tempogram of the bird audio files.

FEATURE EXTRACTION

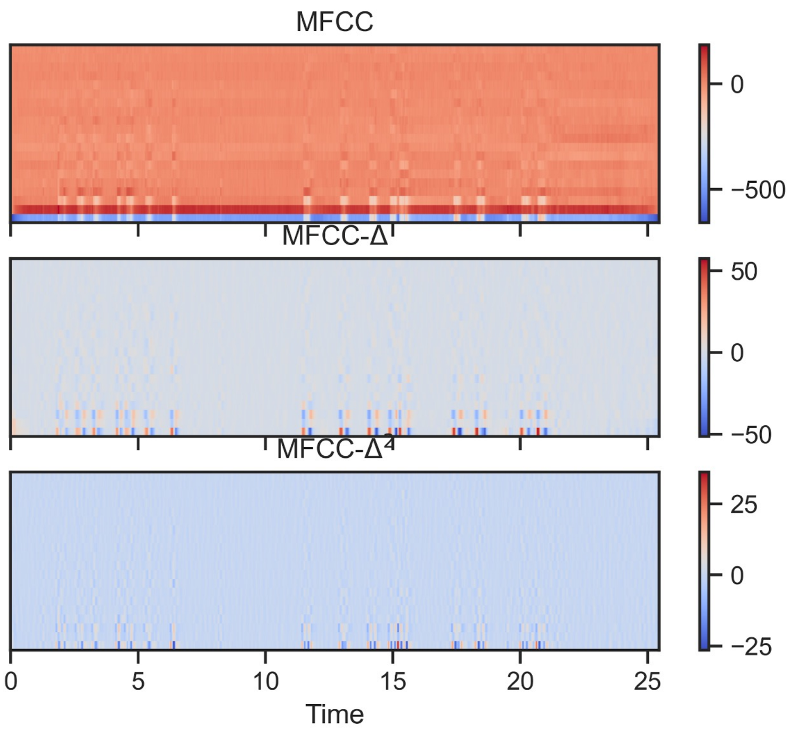
TEMPOGRAM RATIO



Estimation the full tempogram ratio of the bird audio files.

FEATURE EXTRACTION

DELTA



We determine the delta features, the local estimate of the derivative of the data.

APPENDIX
CODE SNIPPETS

1st CNN VERSION

HOMEMADE - Modular/quick extended

```
class Dynamic_CNN(nn.Module):
    def __init__(self, in_dim, out_dim, layers):
        super(Dynamic_CNN, self).__init__()
        self.in_dim = in_dim
        self.out_dim = out_dim
        self.layers = layers

        self.layers = nn.ModuleList()
        in_chan = in_dim

        for out_chan, kernel_size in layers:
            conv_layer = nn.Sequential(
                            nn.Conv2d(in_chan, out_chan, kernel_size),
                            nn.ReLU(),
                            nn.MaxPool2d(kernel_size=kernel_size))
            self.layers.append(conv_layer)
            in_chan = out_chan

        self.fc = nn.Linear(in_chan, out_dim)

    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
            x = nn.functional.relu(x)

        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

2nd CNN VERSION

HOMEMADE - Simple fallback

```
class CNN(nn.Module):
    def __init__(self, num_classes):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, stride=1, padding=1)
        self.relu = nn.LeakyReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(239616, num_classes)  # Initialize with size 0
        self.fc2 = nn.Linear(num_classes, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.maxpool(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.maxpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

3rd CNN VERSION

Dynamically changing input

```
lass CNN(nn.Module):
   def __init__(self, num_classes):
       super(CNN, self).__init__()
       self.conv1 = nn.Conv2d(1, 10, kernel_size=3, stride=1, padding=1)
       self.relu = nn.ReLU()
       self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
       self.conv2 = nn.Conv2d(10, 20, kernel_size=3, stride=1, padding=1)
       self.fc1 = nn.Linear(0, num_classes)  # Initialize with size 0
       self.fc2 = nn.Linear(num_classes, num_classes)

   def forward(self, x):
       x = self.conv1(x)
       x = self.relu(x)
       x = self.maxpool(x)
       x = self.conv2(x)
       x = self.relu(x)
       x = self.maxpool(x)
       size = x.shape[1] * x.shape[2] * x.shape[3]
       x = x.view(x.size(0), -1)
       self.fc1 = nn.Linear(size, num_classes)
       x = self.fc1(x)
       x = self.relu(x)
       x = self.fc2(x)
       return x
```

4th CNN VERSION

PRETRAINED

```
class CNN(nn.Module):
    def __init__(self, num_classes):
        super(CNN, self).__init__()
        self.model = EfficientNet.from_pretrained('efficientnet-b0')
        self.fc1 = nn.Linear(1280, num_classes)
        self.relu = nn.LeakyReLU()
        self.fc2 = nn.Linear(num_classes, num_classes)

    def forward(self, x):
        x = self.model(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

LOAD AUDIO

AND METADATA

```python
# Takes filepath from metadata dataframe and returns audio file
def load_audiofile(filepath):
    audio, sr = sf.read(filepath)
    if len(audio.shape) > 1:
        audio = np.mean(audio, axis=1)
    return audio.astype(np.float32), sr
```

```python
# Takes the directory with the data and returns pandas with metadata
def load_metadata(directory,datadir, trim=False):
    if trim:
        df = pandas.read_csv(directory+'/train_metadata_trim.csv')
    else:
        df = pandas.read_csv(directory+'/train_metadata.csv')
    df['filename'] = datadir+"/train_audio/"+df['filename']
    chosen_coloumns = ['latitude', 'longitude', 'common_name', 'rating', 'filename']
    return df[chosen_coloumns]
```

GET MELSPECTGRAM
AND STFT

```
def get_melspectrogram(audio, sr=32000, n_mels=128, n_fft=2028, hop_length=512, fmax=
16000, fmin=20,power=2.0,top_db=100):
    if type(audio) is str:
        audio, sr = load_audiofile(audio)
    waveform = torch.from_numpy(audio)
    transform = torchaudio.transforms.MelSpectrogram(
                                    sample_rate=sr,
                                    n_mels=n_mels,
                                    n_fft=n_fft,
                                    hop_length=hop_length,
                                    f_max=fmax,
                                    f_min=fmin,
                                    power=2.0
                                    )
    melspectrogram = transform(waveform)

    melspectrogram = torchaudio.transforms.AmplitudeToDB()(melspectrogram)
    melspectrogram = torch.nn.functional.normalize(melspectrogram, p=2, dim=0)

    melspectrogram = (melspectrogram * 255)

    return melspectrogram
```

```
#Calculates Short Time Fourier Transformation of an audio file
# audio -- Can be filepath from metadata dataframe or numpy array with ogg data
def get_STFT(audio, sr=32000, n_fft=2028, nperseg=512):
    if type(audio) is str:
        audio, sr = load_audiofile(audio)
    stft_audio = stft(audio, nfft=n_fft, nperseg=nperseg)
    return stft_audio
```