BirdClef - Deep Dream Team

1. Introduction of the Task

The number of birds and their species contain valuable information for conservationists about the results of their actions. However monitoring them with the traditional ways are limited.Passive acoustic monitoring with ML makes it more maintainable and greater in spatial scales.

Our task was identifying easter Kenyan bird species by their sounds using deep learning. The main goal was to determine which of the 264 bird species can be heard in a 10 minutes audio file. The solution should be a table containing the predicted probabilities to all bird species in the first and the second half of the audio file.

For the training of the neural network a dataset was given with audio files containing only one species' sound paired with the appropriate labels.

1.1 Approach and Mindset

The idea was making the predictions on the 10 minutes file by separating the chirps in it, then making the classification on each chirp. Then for each bird species we search the maximum probability the neural network predicted and we use these maximum values as the probabilities of a species to be in the audio file. To achieve this goal we made a convolutional neural network trained on the dataset.

To train the neural network we used the melspectrograms of the audio files as images and we created a 2D convolutional neural network. To achieve more accuracy we tried creating more types of networks from our individual network to transfer learning with ResNet50.

In our opinion the key to achieve successful teaching progress and making a good prediction is the quality of the input data. This is why we spent more time on making high quality inputs by perfecting the preprocess.

2. Solution

2.1. Preprocessing

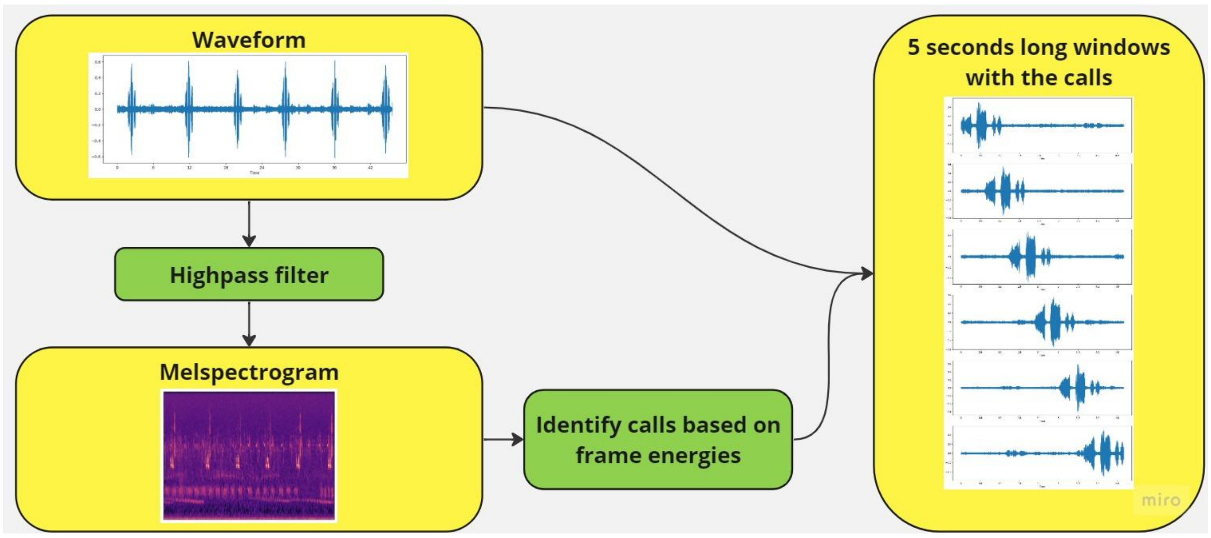
Having a well-curated, high-quality dataset with sufficient diversity is a key factor for success when training a neural network. We conducted our data preparation and preprocessing with this in mind. For the project, we utilized the data provided by Kaggle for the challenge, which consisted of approximately 17,000 audio files in Ogg format along with their metadata in a CSV file. The CSV file contained ratings for all the audio files. Our first step was to remove all the files with low ratings from the dataset. Additionally, we excluded files that were too short and of low quality.

Since the length of the audio files varied significantly throughout the dataset, we decided to extract 5-second-long clips from the files. These clips could later be fed to the neural network as spectrograms. Rather than choosing segments randomly from the files, our goal was to capture parts where actual bird calls were present. To achieve this, we first applied a highpass filter with a frequency cutoff below 1000 Hz to eliminate any low-frequency sounds that were certainly not bird calls.

Following this, we generated mel spectrograms from the audio. Mel spectrograms are created by calculating the short-time Fourier transform of a signal in the time domain with a moving window. This results in a 2-D array where one axis represents time, and the other represents the frequency domain. The values in this array represent the amplitudes of the signals at each frequency and every time step. By summing these values at every time step, we obtained "frame energies" that describe the loudness of the noise at each time step.

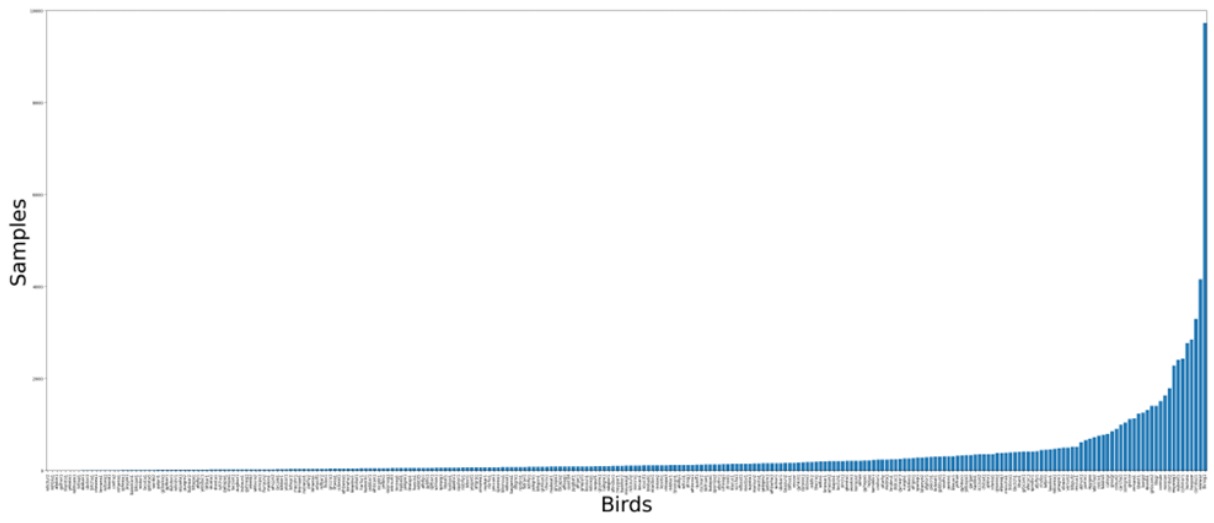
We assumed that bird calls are present when the frame energy is significantly high compared to the rest of the audio. We chose the threshold value to be ten times the median value of all the frame energies in the audio file. We then concatenated the frames that were within 5 seconds of each other and contained bird calls. This way, they would later be treated as one sequence of calls rather than separate calls. We converted their spectrogram indices to time indices and proceeded to cut out 5-second-long windows around the calls.

In cases where the call sequence was shorter than five seconds, we moved the window around the call so that it was always placed differently within the window, introducing some variation to the data. Where the entire audio clip was shorter than 5 seconds, we padded it with random noise. Finally, we saved the 5-second-long clips as Ogg files.


Waveform
Highpass filter
Melspectrogram
Identify calls based on frame energies
5 seconds long windows with the calls
miro

2.2 Selecting the training examples and data generation during training

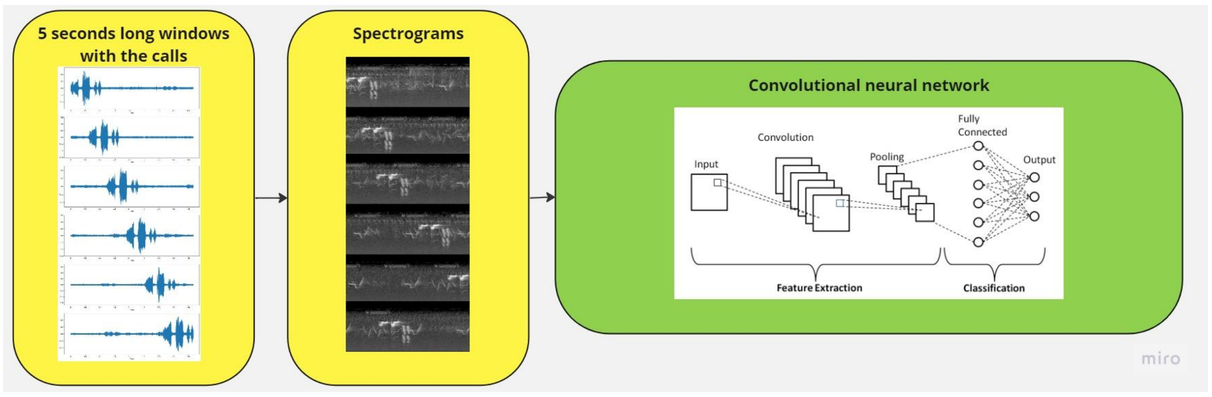
The number of samples extracted for bird species varied significantly. We decided not to use data from species where the number of calls was insufficient because, despite our thorough data preparation, sometimes we misidentified other noises as bird calls. With enough samples, this should not be a problem, but when the sample count is too low, it can be a serious issue and can significantly hinder training. Hence, we chose to only use data from birds from which we could extract at least 50 samples. This meant that we will only give predictions for 192 of the 264 classes; however, the samples from these made up roughly 90% of all the samples.



It is important to note that we did our initial training on even smaller subsets of the data. First, we began training on the samples from birds with over 500 samples, which meant only 30 classes but about half of the whole dataset. The models trained on this subset have reached the best accuracy at about 70%.

Loading the data while training was also a major issue, since given the size of the dataset, loading all data at once was not an option. We experimented with two approaches. One of the approaches was to save the samples as Ogg files and then load them as spectrograms during training with a custom data generator. This allowed for on-the-fly data augmentation but was very CPU heavy and slowed down training to such a degree that we only used this approach for training on the smaller dataset with 30 classes.

The other approach was to save the spectrograms as pictures and load them with the Keras ImageDataGenerator class. This approach is significantly faster; however, it comes at the cost of giving up augmentation because image augmentation techniques do not work well with spectrograms (we have experimented with this as well, but it did not yield satisfactory results). We loaded the data like this during all the trainings with the larger dataset. We saved the spectrograms as RGB images, however with the same values on all the 3 channels, which were the original numbers in the spectrogram array just rescaled to the 0-255 range, so the information content was unchanged. This format was very useful when training pretrained networks, namely ResNet50, which was trained on image data and expected 3-channel imports.



2.3. Training

2.3.1. Transfer Learning with ResNet50 architecture

One of our approaches was building our model on ResNet50 which is a pretrained convolutional network used for computer vision. The expectation was that this neural network can give a more precise classification due to the features it has in their pretrained weights. Because the ResNet50 was trained on RGB images we had to upsample the number of channels of the melspectrograms from 1 to 3. We did this using a simple convolutional layer. After the ResNet50 layer we flattened the data and we made our classification with 2 dense layers.

With this architecture we managed to get 62% accuracy on a database containing those bird species whose number of chirps was higher than 500. This means a database of 30 species. While this result isn't bad at all, our simpler convolutional neural network was able to achieve the same accuracy with much fewer parameters to store and train. For this reason the simpler network is computationally more effective thus we used this simpler model later in our task.

2.3.2. Simple Convolutional Neural Networks

We have experimented with a number of simpler CNN architectures, first only as sanity tests besides the ResNet50 trainings, however they always outperformed it. The reason for this might be the relatively small dataset, on which simpler models are able to generalize better. After an automated hyperparameter-optimization with keras tuner, we have found the model architecture that performed the best on the large dataset. After 10 epochs, it predicted with roughly 51,5% accuracy on both the testing and validation datasets, both of which consisted of roughly 8000 randomly chosen examples from all of the classes. This was, however, the best accuracy we could achieve on this dataset, because the model started overfitting upon further training. This could be improved upon by introducing audio augmentation techniques, which we have experimented with previously.

It is also important to note that when training on the large dataset with 192 classes, we had to modify the loss function. In order to achieve that, we calculated class weights to all the classes and fed them to the fit function.

3. Conclusion and room for improvement

Finding a more efficient way for on-the fly data augmentation would certainly improve accuracy by dampening the effects of overfitting. While image augmentation techniques are an option, and we have experimented with them as well, in combination with a method called curriculum learning, during which the model was first trained on unaltered data, and after reaching certain level of accuracy, image augmentation techniques were introduced, however it did not yield satisfactory results. Applying real audio augmentation techniques during data loading would be the best approach to address overfitting. Finding a more efficient way to achieve that would most likely improve model performance significantly.