# BIRD CLASSIFICATION BY SOUND

TECHNICAL REPORT

**Aleksis Datseris**
Department of Mathematics and Informatics
Sofia University
5 James Bourchier Blvd, 1164 Sofia, Bulgaria
aleksisdatseris@gmail.com

**Boyko Borisov**
Department of Mathematics and Informatics
Sofia University
5 James Bourchier Blvd, 1164 Sofia, Bulgaria
bojkob@uni-sofia.bg

**Kliment S. Berbatov**
Department of Mathematics and Informatics
Sofia University
5 James Bourchier Blvd, 1164 Sofia, Bulgaria
kberbatov@uni-sofia.bg

June 23, 2023

**ABSTRACT**

Birds are excellent indicators of biodiversity change since they are highly mobile and have diverse habitat requirements. Changes in species assemblage and the number of birds can thus indicate the success or failure of a restoration project. We use Convolutional Neural Networks (CNN), as well as transfer learning with EfficientNet to predict bird species based on their calls or songs.

## 1 Introduction

Machine learning algorithms have become more and more popular for identification of different species in the natural environment [5]. They are becoming a huge upgrade, compared to traditional methods, such as manually sifting large amounts of data, generated from Passive acoustic monitoring (PAM), camera traps or biodiversity surveys over large areas. Such surveys usually carry huge logistic and financial burdens and thus are more uncommon as data sets for further analysis. Furthermore, camera traps suffer from a narrow field of view and limited range. Compared to this, Acoustic data provides more opportunities to observe and identify animals. Birds especially have a diverse use of sound in their lives - territory marking, mating calls or to indicate safety or danger. These animals can also act as bio-indicators for the health status of the observed ecosystem. In this paper, we'll be using audio data collected from the Xeno-canto [11] on birds from Eastern Africa.

Our idea is to utilize CNNs and transfer training of EfficientNet[10] with different regularization methods in order to compare their effectiveness.In short, a CNN is a neural network that includes one or more layers using convolution. A convolution is an operation on a matrix, usually performing a weighted sum on a subsection of the matrix. EfficientNet is a pre-trained CNN with generally great performance.

For optimization and regularization, we'll compare the base models with added augmentations, such as Gated Recurring Unit (GRU)[4] layers, Long Short-Term Memory (LSTM)[7] layers, Time-Frequency Masking (TFM) and Siamese network training[3]. LSTMs are a RNN architecture, that can learn long-term dependencies in the input data. A GRU layer is a gating mechanism used in recurrent neural networks, similar to an LSTM, but with fewer parameters. It was introduced in a 2014 paper[4]. Time-frequency masking is the process of applying weights to the time-frequency input data, in order to shift focus to portions of audio.

A Siamese network, first introduced in 1994[1], is a special way of training a neural network. It takes a triplet of images, an anchor, a positive and a negative and passes them through the same model, maximizing the distance between the anchor and negative images and minimizing the distance between the anchor and positive images.

Finally, for scoring, we'll be testing the models on labeled recordings of different bird species. The models will accept 10-second snapshots of audio and give their prediction. We'll compare the top 5 error rates, precision, recall and F1-score.

## 2 Method

### 2.1 Data Formatting

All the data consists of environment recordings of different bird species, collected by users of Xeno-Canto[11]. Unfortunately, some species are over-represented, while others have one or two noisy recordings. Recordings are also different in length - while some are over a minute long, some are shorter than 10 seconds. In order to prevent underfitting, we have entirely excluded species with less than 30 total recordings from the provided data set. Furthermore, some of the remaining classes have gone through upsampling during training, which duplicates their audio recordings, in order to normalize species' distribution. As a result, every class has a minimum of 50 recordings in the training data. After that, each audio recording is broken down into chunks independently of the others. In this way even duplicated audio could result in different audio chunks being cut.

We decided to work with audio chunks of length 10 seconds as input to our models since it is a reasonable time frame to distinguish most birds while also being small enough to allow the models to run relatively fast. Using sampling rate of 32 000 Hz this results in every chunk being a 1D tensor with 320 000 values. This standardization of the model input also has the upside of making the data easy to batch.

Theoretically, each recording can be split into as much as possible 10-second chunks, or padded if shorter than that, but mainly due to hardware memory limitations we decided to sample fewer than possible chunks. Sampling all possible chunks would also convey the risk of over-representation of certain classes since some of the recordings are more than 6 minutes long. We introduce a hyperparameter $N$ which represents the preferred number of 10-second chunks to be split from each audio. If the audio has length of at least $N * 10s$, then it is split into $N$ equal parts, and from each part we randomly sample a 10-second chunk. If the audio is not that long, then we split as many consecutive 10-second chunks as possible. Since the last chunk is usually shorter than 10 seconds, we add padding to the end of it in order to extend it to 10 seconds. We chose a value of $N = 3$ since this is the largest value we could use before hitting hardware memory limitations.

Finally, we transform the raw audio into a Mel-spectrogram of size $128 \times 384$ pixels. It is an image, where the y-axis is the sound frequency and the x-axis is the time. The color intensity corresponds to the sound density at a certain time/frequency. We decided to use this transformation instead of working with the raw audio since there has been reported success with it in the past and it also allows us to use transfer learning with pretrained image classification models.[6][14][8]

### 2.2 Data Augmentation

Each training audio is first modified by adding Noise with a Gaussian distribution with variable standard deviation. The standard deviation value is chosen independently for each audio and for each epoch in order to increase the training data variation.[2]

Then we employ data augmentation and regularization methods MixUp[13] followed by CutMix[12] on the training audio. They are applied over every batch. They are also applied independently for each epoch in order to increase the training data variation.

After the audio is transformed into a Mel-spectrogram, in some of the models we add random time-frequency masks as additional regularisation to the training data. It is again applied independently for each audio and for each epoch. [9]

## 3 Model Architectures

We explore 2 different base neural networks as our predictor tool. Both of them are CNNs, where the second is based on transfer learning using EfficientNet[10]. We also try to combine the two approaches into a single network to get better results. In every iteration of the neural networks we first pass the augmented audio through a Mel Spectrogram layer that outputs a tensor of shape $(128, 384, 3)$, where the first 2 dimensions are the width and length of the resulting spectrogram image, repeated 3 times, to mimic the three channels (Red, Green, and Blue) of a normal image. The 3 channels are also necessary since EfficientNet accepts only images with 3 channels. Every model outputs a vector of 264 numbers, which is interpreted as the logits for each class of birds. After that we also try to use the Siamese

Networks approach to improve the results from the trained models. Also, note that $(p)$ in the name of the models is used to signify that the used component was pre-trained beforehand.

### 3.1 Custom CNN network

The resulting tensor's values are normally standardized and passed through 2 convolutional layers (CL) with a windows size of $(11, 33)$ and stride $(2, 2)$. After which we apply a max pooling operation with a default $(2, 2)$ window, and pass the resulting tensor to another CL with the same window size, but this time with a dilation rate of 2 after which we add a Batch Normalization layer. After each convolutional layer, ReLU is used as an activation function. In some of the tested models, the output is then passed through a Global Average Pooling layer and a Dropout layer with $p = 0.5$ while in others we pass it through a 3-layer bidirectional GRU with a hidden vector size of 128. We also add an intermediate Dropout layer with $p = 0.5$ and an intermediate Batch Normalisation layer before the last GRU. Finally, we have a dense layer that outputs the logits vector of the 264 bird classes.

We compile the model using the Adam optimizer and a Categorical Cross entropy loss function.

$$CrossEntropy = -\sum_{i=1}^{N} y_i * log(\hat{y_i}) \tag{1}$$

#### 3.1.1 CNN

We first trained the described CNN network with a Global Average Pooling after the convolutional layers. We used this model as a baseline to compare later modifications.

#### 3.1.2 CNN(p) + GRU

We then took the pre-trained CNN from the last step and removed the last layers up to the Global Average Pooling. Then we replaced them with the described GRU layers and trained the network again. This gave significant improvement to the results and is also the best modification we got of the custom CNN model.

#### 3.1.3 CNN + GRU

This model has the same architecture as the previous one but we randomly initialized both the CNN and GRU layers and trained them simultaneously. While this gave a small improvement compared to the base model, it did not reach the same performance as the previous model with pre-trained CNN weights.

#### 3.1.4 CNN + LSTM

We decided to test the last approach again but replaced the GRU layers with LSTM ones. This gave significantly worse results as the network became much harder to train.

### 3.2 EfficientNet

For our second base model, we use a pre-trained EfficientNet model from the Keras python library, based on a 2020 paper[10]. We decided to use the EfficientNetB1 model since we had limited memory and computational resources and couldn't afford to train with a more complex one. We set the base models' weight as trainable with the exception of the Batch Normalization ones, though there can be some argumentation for efficiency by ignoring this step and just training the input feedforward layer. The main reason we decided to train even the lower layers is that the EfficientNet models were not trained on spectrogram images and it is necessary for the whole network to be fine-tuned on that task. In some of the tested models, the output is then passed through a Global Average Pooling layer and a dropout layer with $p = 0.5$ while in others we pass it through a 3-layer bidirectional GRU with a hidden vector size of 128 or 256. We also add an intermediate Dropout layer with $p = 0.5$ and an intermediate Batch Normalisation layer before the last GRU. Finally, we have a dense layer that outputs the logits vector of the 264 bird classes.

Same as the CNN model, this one is compiled, using the Adam optimizer and a Categorical Cross entropy loss function.

#### 3.2.1 EfficientNet

First, we trained the described approach of using Global Average Pooling on the EfficientNet outputs. We used this model as a baseline to compare later modifications.

#### 3.2.2 EfficientNet + TFM

First tested the last approach by adding time-frequency mask to the spectrograms in addition to the other augmentations, that are used on the raw audio. This gave some improvement to the results and is also the best modification we got of EfficientNet.

#### 3.2.3 EfficientNet + GRU

We also tried the approach of adding GRU layers after the EfficientNet output and training them simultaneously. We didn't achieve improvement compared to the baseline model.

### 3.3 Combined models

Finally, we tried to combine our custom CNN models with the fine-tuned EfficientNet models. We take a pre-trained model from both approaches and run them in parallel after which we combine the results by concatenation and passing it through a Dense layer.

#### 3.3.1 EfficientNet(p) + (CNN + GRU)

First, we tried to combine the fine-tuned baseline EfficientNet model with the randomly initialized custom CNN + GRU model. This gave a small improvement compared to the baseline fine-tuned EfficientNet model.

#### 3.3.2 (EfficientNet(p) + GRU) + (CNN + GRU)(p) + TFM

First, we took the fine-tuned EfficientNet model with Time-Frequency Mask and replaced the Global Average Pooling with GRU layers. We also took the pre-trained second model from the custom CNN models since it gave the best results. We also apply the Time-Frequency Mask to the spectrogram before giving it as an input to both models. Finally, we added an additional Dropout layer after combining the results from both models and an additional Dense layer after it. This model gave the best results overall.

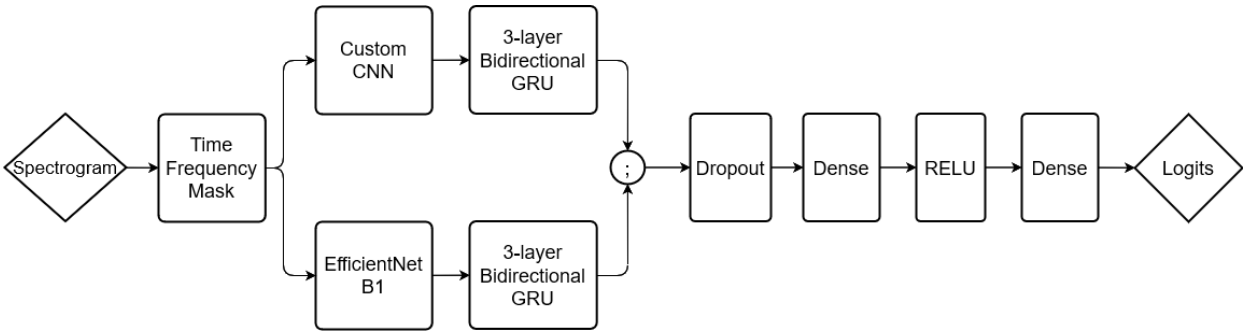


**Figure 1:** Simplified architecture overview of the model

### 3.4 Siamese network

#### 3.4.1 Additional Data Modification

Due to the unique nature of a Siamese Network, we are required to do some extra data modification. Training data no longer requires corresponding labels, but a triplet of entries (anchor, positive and negative). As such, we first create a "modified" training dataset, where we randomly pair each entry with a suitable positive and negative point. Here, suitable means that the anchor and positive image must be different, but of the same class, and a negative image would be of a different class.

#### 3.4.2 Structure

As an input we get a triplet of 3 different audio recordings, which we combine into a batch, passing it through the base model. Afterward, we split the result back into 3 separate vectors, corresponding to the three input audios, and pass that to the custom triplet loss function. This function calculates the distance between the anchor and positive/negative images using cosine distance using the following formula:

$$L(A, N, P) = max(||f(A) - f(P)||_{cos} - ||f(A) - f(N)||_{cos} + margin, 0) \quad (2)$$

Here, the margin signifies a soft margin between positive and negative pairs. The cosine distance between two points is given by:

$$||f(X) - f(Y)||_{cos} = 1 - \frac{X \cdot Y}{||X||_2||Y||_2} \tag{3}$$

#### 3.4.3 Tested models

We use the pre-trained best-performing variations of the previous three classes of models inside our Siamese networks and use their output probability vectors to measure the distance between different inputs. We also tried using the logit vectors instead but that gave worse results.

## 4 Model Training

Training has been done on an up-to-date version of Google Colab running Python 3 on NVIDIA T4 Tensor Core GPUs. These GPUs have around 15GB of available GPU memory and roughly 50GB of disk space.
Initially, the main time bottleneck was loading and decompressing the audio files from memory. To avoid it, we cached the audio inputs on the disk in order to speed up significantly every training epoch after the first. Without caching, training of the models would be unfeasible.
During training the models can use up to around 14.5GB of the GPU memory and nearly all of the available 50GB of disk space for caching. That was the main hardware limitation that prevented us from testing bigger models. Additionally, at the end of each epoch, the best current model is saved into external memory for preservation and later experimentation.

### 4.1 Normal Training

Each model is trained for between 10 and 25 epochs with a batch size of 32, starting with a fixed learning rate. During training, we utilize a learning rate reduction mechanism with factor 0.2 and patience 2 or 3 (if there is no improvement in the last 2 or 3 epochs, the learning rate will be multiplied by the factor). Additionally, we have an early stopping mechanism that stops the training if the learning rate becomes lower than a given threshold.

### 4.2 Siamese Network Training

The Siamese networks training is performed after the normal training, using the best-performing variations of the previous three model classes. As noted above, we use the output probability vectors to calculate the distance between the inputs since it gave better results than using the logit vectors. We train the Siamese network for an extra 16 epochs using the same batch size as before, but this time with a constant learning rate. Finally, we save only the used base model that was fine-tuned by the Siamese network.
We also cut only 1 chunk of 10 seconds from each audio entry instead of the maximum of 3 during normal training. This is because the Siamese networks use triplets of audios as input and already take up 3 times as much memory as the normal inputs.
It is also necessary to note that the MixUp[13] and CutMix[12] augmentations are no longer usable in training Siamese networks so we disabled them.

## 5 Evaluation

All the models so far have been trained to recognize audio recordings with a fixed length of 10 seconds. In order to evaluate their performance, we want to make predictions on audio recordings from the test set that could have variable lengths.
The approach we used is to split every audio from the test set into as many consecutive 10-second chunks as possible. Since the last chunk is usually shorter than 10 seconds, we add padding to the end of it in order to extend it to 10 seconds. Then we run the models on all the 10-second chunks and combine the results to make the final predictions. The way of combining the results and making predictions is different for the normal and Siamese models, so they will be described separately.

### 5.1 Making predictions with normal models

When testing a normal model we take the probability vectors from each 10-second chunk and calculate their mean probability vector. Then the mean probability vector is used as usual to order classes by how preferable they are.
We also tried to combine the logit vectors of every chunk instead by taking their mean or their element-wise maximum and then pass the resulting vector through a softmax to get the resulting probability vector. However, these approaches gave worse results than working with the mean probabilities.

### 5.2 Making predictions with models trained with a Siamese network

The idea of the Siamese networks is to cluster the output vectors together based on their class. Therefore, the standard approach of looking up the probability of a class by taking a certain position in the output vector no longer holds. There are two steps that should be taken in order to predict the class of audio input.
First, after training the model with a Siamese network, we take the output vectors of all the anchor audios that participate in the training set. They will be used as baseline vectors for predicting new inputs.
When we have to predict the class of new audio, we split it into 10-second chunks as described. Then we pass them through the model to get the output vectors and find their mean vector that describes the audio as a whole. Finally, we use the mean vector and the baseline vectors to predict the class of the input. There are 2 approaches we tried to make that prediction.

#### 5.2.1 Centroid Vectors

We have a set of baseline vectors for the different classes. We can calculate the centroid point of each class by taking the mean of all the baseline vectors corresponding to that class and normalizing the result vector. Then we can measure the Euclidean distance from the centroids to the vector we want to predict.
Euclidean distance uses the well-known formula for 2 points, $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$ :

$$dist_{eucl}(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2} \tag{4}$$

Using this distance we calculate the desirability that $X$ is of class $y$ by using the following formula:

$$desirability(X\ is\ of\ class\ y \mid dist(X, y_{centroid})) = \frac{1}{dist(X, y_{centroid})} \tag{5}$$

It should be noted that the calculated desirability has the same functional use as the logits that are used in the standard predictions. In this way, we can order the classes by how preferable they are. This is the method by which we calculate the final results for the models trained with Siamese networks.

#### 5.2.2 K Nearest Neighbours

Another method to predict the class is by using KNN predictor. The baseline vectors and their classes are used as a reference for the predictor. We use a value of $k = 30$ for our predictions and order the classes by how much of their $k$ nearest neighbors are of a given class. This method gave a bit worse results compared to the Centroid Vectors approach.

## 6 Results

The following tables summarise the results of the tested models. Note that $(p)$ in the name of the models is used to signify that the used component was pre-trained beforehand.

**Table 1:** Results from models with normal training

| Model | Top-1 Error | Top-5 Error | Precision | Recall | F1-score |
|---|---|---|---|---|---|
| CNN | 47.79% | 24.20% | 53.49% | 52.21% | 50.46% |
| CNN(p) + GRU | **42.63%** | **22.45%** | **58.27%** | **57.37%** | **56.22%** |
| CNN + GRU | 46.01% | 23.56% | 52.64% | 53.99% | 51.15% |
| CNN + LSTM | 65.43% | 40.25% | 30.75% | 34.57% | 30.26% |
| EfficientNet | 19.86% | 7.03% | 80.78% | 80.14% | 79.79% |
| EfficientNet + TFM | 18.90% | 6.84% | 81.63% | 81.10% | 80.71% |
| EfficientNet + GRU | 19.73% | 8.36% | 80.38% | 80.27% | 79.79% |
| EfficientNet(p) + (CNN + GRU) | 19.03% | 7.16% | 81.32% | 80.97% | 80.60% |
| (EfficientNet(p) + GRU) + (CNN + GRU)(p) + TFM | **17.93%** | **7.25%** | **82.68%** | **82.07%** | **81.80%** |

**Table 2:** Results from models with Siamese network additional training

| Model | Top-1 Error | Top-5 Error | Precision | Recall | F1-score |
|---|---|---|---|---|---|
| (CNN + GRU)(p) | 48.29% | 28.75% | 57.49% | 51.71% | 53.47% |
| (EfficientNet + TFM)(p) | 23.24% | 11.34% | 81.40% | 76.76% | 77.91% |
| ((EfficientNet + GRU) + (CNN + GRU) + TFM)(p) | 24.86% | 12.29% | 78.22% | 75.14% | 75.71% |

## 7 Conclusion

In this work, we build bird audio recognizer models based on CNN architectures and the transformation of raw audio into a Mel-spectrogram. We tried different types of data augmentations in an attempt to improve the models' performance. This includes Gaussian Noise, MixUp and CutMix, which are applied to the raw audio, and Time-Frequency Mask, which is applied to the Mel-spectrogram. We first tested our own custom CNN architecture and then we tested transfer learning with a pre-trained EfficientNet model. In both cases, we tried to add RNN layers after the CNN in an attempt to get better performance. We found out that training the CNN as a classifier first and then adding the RNN layers on top gives better results than training them together from the start. Finally, we combined both custom and EfficientNet architectures into a single network which achieved the best performance from all of our models. We also tried to do additional training on the models by using them as a base model for a Siamese neural network and tested different ways of making predictions with the new models but we did not get an improvement of the results.

## References


[1] Jane Bromley et al. "Signature Verification using a "Siamese" Time Delay Neural Network". In: *Advances in Neural Information Processing Systems*. Ed. by J. Cowan, G. Tesauro, and J. Alspector. Vol. 6. Morgan-Kaufmann, 1993. URL: https://proceedings.neurips.cc/paper_files/paper/1993/file/288cc0ff022877bd3df94bc9360b9c5d-Paper.pdf.

[2] Alexander Camuto et al. *Explicit Regularisation in Gaussian Noise Injections*. 2021. arXiv: 2007.07368 [stat.ML].

[3] Davide Chicco. "Siamese Neural Networks: An Overview". In: *Artificial Neural Networks*. Ed. by Hugh Cartwright. New York, NY: Springer US, 2021, pp. 73–94. DOI: 10.1007/978-1-0716-0826-5_3. URL: https://doi.org/10.1007/978-1-0716-0826-5_3.

[4] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].

[5] Tom Denton, Scott Wisdom, and John R. Hershey. *Improving Bird Classification with Unsupervised Sound Separation*. 2021. arXiv: 2110.03209 [eess.AS].

[6] James L. Flanagan. *Speech Analysis Synthesis and Perception*. Springer Berlin Heidelberg, 1972. DOI: 10.1007/978-3-662-01562-9. URL: https://doi.org/10.1007/978-3-662-01562-9.

[7] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[8] Boyang Zhang Jared Leitner and Samuel Thornton. "Audio Recognition using Mel Spectrograms and Convolution Neural Networks". In: 2019. URL: http://noiselab.ucsd.edu/ECE228_2019/Reports/Report38.pdf.

[9] Soundararajan Srinivasan, Nicoleta Roman, and DeLiang Wang. "Binary and ratio time-frequency masks for robust speech recognition". In: *Speech Communication* 48.11 (2006). Robustness Issues for Conversational Interaction, pp. 1486–1501. ISSN: 0167-6393. DOI: https://doi.org/10.1016/j.specom.2006.09.003. URL: https://www.sciencedirect.com/science/article/pii/S0167639306001129.

[10] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[11] *Xeno-canto birdsounds collection*. https://xeno-canto.org/. Accessed: 2023-06-15.

[12] Sangdoo Yun et al. *CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features*. 2019. arXiv: 1905.04899 [cs.CV].

[13] Hongyi Zhang et al. *mixup: Beyond Empirical Risk Minimization*. 2018. arXiv: 1710.09412 [cs.LG].

[14] Quan Zhou et al. "Cough Recognition Based on Mel-Spectrogram and Convolutional Neural Network". In: *Frontiers in Robotics and AI* 8 (2021). ISSN: 2296-9144. DOI: 10.3389/frobt.2021.580080. URL: https://www.frontiersin.org/articles/10.3389/frobt.2021.580080.