BirdCLEF 2024 - **Kesobb_kitalaljuk** team

Bilicki Vilmos (BLU8CF), Goldschmidt Olivér (RXL623), Szabó Tamás János (DN5FXS)

Abstract **- We present a deep learning pipeline for the BirdCLEF 2024 task, aiming to identify bird species from audio recordings. Starting from the provided competition metadata and recordings, we standardize inputs by splitting variable-length audio into 5 s chunks with 1 s overlap, and we remove low-information „silent" segments using an energy threshold. To mitigate severe class imbalance across 182 species, we employ targeted sampling in a custom generator to provide an equalized number of chunks per class per epoch. Robustness is improved via audio and spectrogram augmentation (e.g., time stretching, pitch shifting, Gaussian noise, filtering, and distortion). Our model leverages transfer learning with a pretrained ConvNeXtTiny backbone and a fully connected classification head, trained in two phases (frozen backbone, then fine-tuning with a reduced learning rate) with standard regularization and callbacks.**

Kivonat **- Bemutatunk egy mélytanulási architektúrát a BirdCLEF 2024 feladathoz, amelynek célja madárfajok azonosítása hangfelvételek alapján. A verseny által biztosított metaadatokból és audio fájlokból kiindulva kialakítjuk a tanításhoz használt adathalmazt: a változó hosszúságú hanganyagokat 5 másodperces darabokra bontjuk 1 másodperces átfedéssel, majd az alacsony információtartalmú „csendes" szegmenseket energiaalapú küszöböléssel eltávolítjuk. A 182 faj közötti erős egyenletlen osztály eloszlás kezelésére célzott mintavételezést alkalmazunk egy egyedileg megalkotott generátorban, hogy epochonként fajonként kiegyenlített számú darab kerüljön betanításra. A robusztusságot hang- és spektrogram-alapú augmentációval javítjuk (pl. időbeli nyújtás, hangmagasság-eltolás, Gauss-zaj, szűrés és torzítás). Modellünk transzfer tanulást használ egy előtanított ConvNeXtTiny gerinc hálózattal és egy teljesen összekötött klasszifikációs fejjel, amelyet két fázisban tanítunk (fagyasztott gerincháló, majd finomhangolás csökkentett tanulási rátával) standard regularizáció és callbackek mellett.**

I. INTRODUCTION

CLASSIFICATION problems are tasks in which the elements of a dataset must be assigned to classes based on some characteristic.

In our case, the goal is to determine the species of birds from audio files. This is an interesting and relevant problem: knowing the species can help deduce other important attributes of birds, like endangerment and nocturalism that can help researchers know more about the fauna.

Machine learning, and in particular deep learning techniques [1], are widely used approaches for classification problems. By using neural networks, even highly complex classification tasks can be addressed. In our work, we construct an ensemble model by appropriately combining multiple neural network architectures, thereby leveraging their respective strengths. Convolutional neural networks (CNNs) [2] and feed-forward, fully connected (FC) networks can be effectively applied together for image classification: in our solution, convolutional layers are used to extract relevant features from the images [3] of spectograms of the audio files, which are then utilized by the feed-forward layers to learn the classification decision.

Before training the model, a critical step is data preparation: designing a pipeline that transforms the initial (provided) dataset into inputs suitable for the model. In the original dataset, the class distribution is highly imbalanced, consequently, data augmentation [4] plays a key role. During augmentation, the number of samples (in our case, images) in underrepresented classes is increased using various techniques, thereby balancing them with classes that occur more frequently [5] and reducing the risk that the model's decisions are biased by class representation in the training data. Typical techniques include time stretching, pitch shifting, adding Gaussian noise or using a highpass or bandpass filter.

In addition to selecting an appropriate model architecture, choosing suitable hyperparameters is also essential, as these choices can substantially affect performance.

In this documentation, we present the project's development process as well as the development and architectural decisions made throughout. Each chapter describes a distinct phase of development, detailing the specifics of that phase, its challenges, and the ideas considered.

II. RELATED WORK

Numerous previous studies have been conducted in the area of bird sound recongition, we aim to highlight the ones most relevant for our means:

Zhong, Ming et al. [6] tried to identify 2 rare bird species from Nepal. They cut 2- and 4-seconds long clips from the audios, made spectograms of those, and performed the binary classification of the 2 species by fine-tuning a pretrained ResNet50 model.

Sprengel, Elias et al. [7] used a convolutional neural network-based approach for the BirdCLEF 2016 competition. They separated the data to noise and useful signal by morphologically filtering the spectograms and augmented the data which they used to train their network consisting of 5 convolutional layer and max pooling units, a dense layer of 1024 neurons.

Koh, Chin-Yuan et al. [8] also split chunks of audios, augmented the data and experimented with multiple models for the BirdCLEF 2019 competition: ResNet-18, ResNet-34 and modified Inception-V3.

Based on these works, using convolutional neural networks is a fundamental method to classify bird sounds, and clipping smaller chunks of audios is also a frequently used technique.

III. MODEL

A. Architecture

To solve the task, we will use an architecture based on a convolutional neural network (CNN) and a feed-forward neural network, as this combination is well suited for image classification.

We will implement an existing CNN-based architecture and connect it to a fully connected, feed-forward neural network. The convolutional network highlights image details, the feed-forward network classifies.

For the convolutional component, we will use a pretrained model and fine-tune it together with the attached fully connected network.

First, we will freeze the pretrained model and train only the newly attached layers so that the pretrained weights won't be agressively affected. Afterwards, we will fine-tune the entire model.

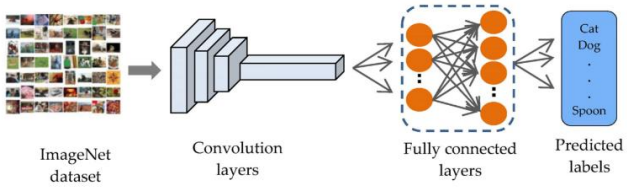

Fig. 1.: Pretraining with ImageNet dataset

During the solution process, we examined multiple network architectures, focusing primarily on the ConvNeXt model family. ConvNeXt is based on modern concepts and high-capacity blocks. We also tried a self-built convolutional network, but it quickly turned out to be not effective enough.

As a starting point, we built a custom convolutional network, which was low on parameters, so it had a fairly quick learning proccess. With this model we could make sure our data preprocessing works well.

After that, we shifted towards ConvNeXt architectures and examined the ConvNeXtTiny variant in more detail, as the smallest member of the ConvNeXt family. Its more modern design and higher parameter count already showed more promising performance at early training stages, suggesting that the model's internal structure better fits the nature of our available data. We built a substantial part of our workflow around this architecture.

B. Hyperparameters

For training, we use mini-batching. The training data is divided into groups, and at each iteration the model receives a number of samples equal to the batch size. This enables processing multiple samples at once, which provides faster and more stable learning and also reduces memory usage.

When choosing the number of epochs, it is advisable to consider that the model may overfit if the value is too high, while if it is too low the model may underfit. To solve this issue, we use early stopping (a callback), which ensures that training stops at an optimal point.

As the optimizer, we use the Adam algorithm. Adam performs well with both small and large batch sizes and typically requires less hyperparameter tuning. In transfer learning tasks it is frequently used because of its simple parametrization and fast convergence, even with relatively limited data.

To prevent overfitting, we apply various regularization techniques. In the classifier layers of the model, for example, we use dropout layers, which deactivate neurons according to a specified probability.

C. Callbacks

To make our teaching more effective, we use the following callbacks:

- Early Stopping callback [10] prevents overfitting by monitoring the validation loss metric. If no improvement occurs over many consecutive epochs, it automatically stops training.

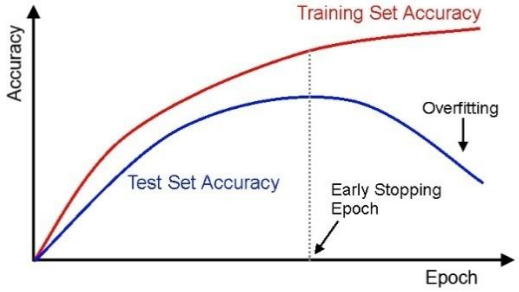

Fig. 2.: Illustration of Early Stopping

- ReduceLROnPlateau callback [11] dynamically decreases the learning rate when training stagnates. If the monitored metric does not improve for a sufficient number of epochs, the learning rate is reduced. This supports finer convergence and is particularly useful during the fine-tuning phase.
- ModelCheckpoint callback saves only the best-performing model according to a selected validation metric. This ensures that we do not lose the optimal weights even if training later degrades due to overfitting.

IV. IMPLEMENTATION

A. Overviewing and Preprocessing Data

We aim to offer a solution for the BirdCLEF 2024 Kaggle competition. The task of the competition is to classify under-studied birds by species, by endangerment and by nocturalism based on audio files containing sounds from birds, meaning that our task will be one multiclass and two binary classifications.

The dataset we use for our task is the one the competition offers us. It contains the following files:

- train_metadata.csv: metadata of the audio files, including primary label, secondary label, type of sound, latitude, longitude, scientific name, common name, author, license, rating, URL and filename

- eBird_Taxonomy_v2021.csv: futher data of bird species, we will not be using this file.
- train_audio: a ZIP file containing the audio files grouped by primary label.

train_metadata.csv contains a lot of useful information, howewer there are some attributes we find less useful for our goal: latitude, longitude, scientific name, common name, author, license, rating, URL.

Primary label, secondary label, type of sound and filename seem like attributes with value, so we need further investigation to decide if we should use them.

Primary label is present for every row record, but secondary label is only present for 1892 records out of the 24459 records of train_metadata.csv, so we decide to drop it. Knowing the type of sound seems helpful for our goal, but there are 918 distinct values of types of sound, which is too many to be used for an effective classification, so we drop that too, meaning that the only attributes we will keep are primary label and filename.

There are 182 distinct primary labels, meaning that there are 182 classes for our classifications. The distribution of classes classes by count of audios is displayed in Fig. 3.

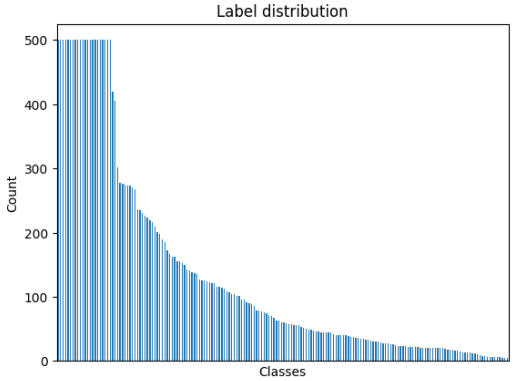

Fig. 3. Distribution of classes in the dataset.

The audio files are of varying lengths, from a few seconds to multiple minutes. Left like this, the data would be inadequate for a model to train on, so we will split the audio files into smaller chunks. We decided to split the data into chunks of 5 seconds with a 1 second overlay. 5 seconds is a smaller, but still relevans amount of information, and a 1 second overlay helps to keep continuous connection of an audio file.

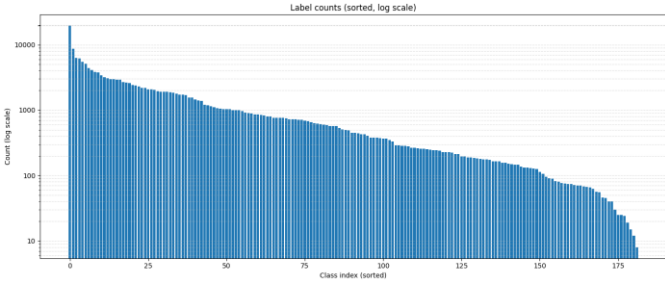

Fig. 4.: distribution of classes in the audio chunks

As shown in Fig. 4., the distribution of classes after splitting is still uneven. To solve this problem, we give the model equivalent chunks of each class in one epoch. Our data generator makes this work. First, we declare a threshold, then our generator reads files from each class, until the chunk number reaches the threshold. After these steps, our model will get threshold number chunks from each class.

We convert the audio chunks into spectograms, that can be used as images for the input of the convolutional layers during the training of the model.

We have to tackle another important issue: removing the „silent" parts of audios. Since most of the bird sounds are relatively short compared to the length of the audio that contains them, a vast portion of the audio chunks lack bird sounds, or contain only an insignificant amount of information. We have to eject these chunks before moving on with creating the train dataset, which we perform by filtering out the chunks that have a maximum energy level of under the given treshold (0.1).

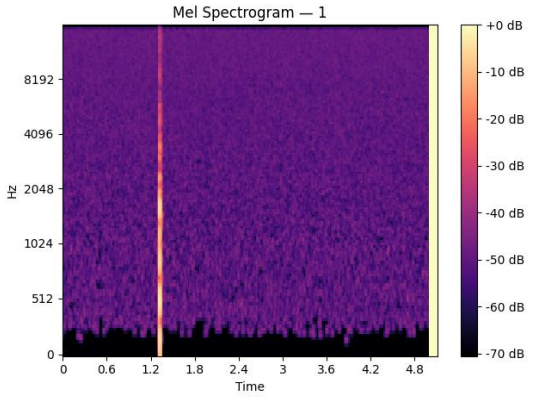

Fig. 5.: example of a silent chunk

To make our model generalize better, we will use augmentation for both the audio and the spectrograms. The audio changes are intended to simulate real-world variations of the sounds.

The parameter values should be chosen such that they do not distort excessively, but instead simulate realistic variations. Overly aggressive augmentation can produce unrealistic images, which would degrade the model's performance on real data.

We augment the audio files using various techniques. These techniques are displayed on a spectogram of an audio chunk.

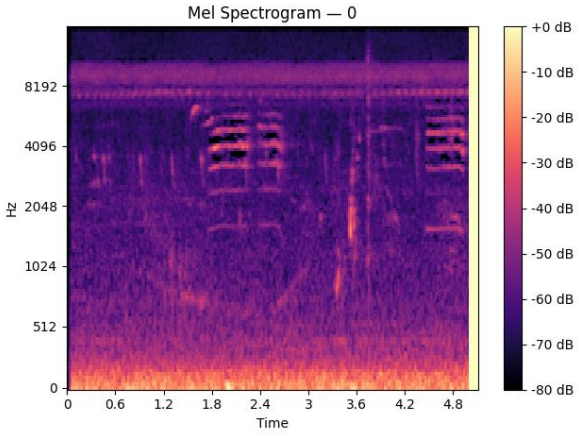

Fig. 6.: spectrogram of an audio chunk

The main augmentation techniques we used were the following:

- time strecth,
- pitch shift,
- adding Gaussian noise,
- high pass and band pass filters (Fig. 7.),
- clipping distortion (Fig. 8.).

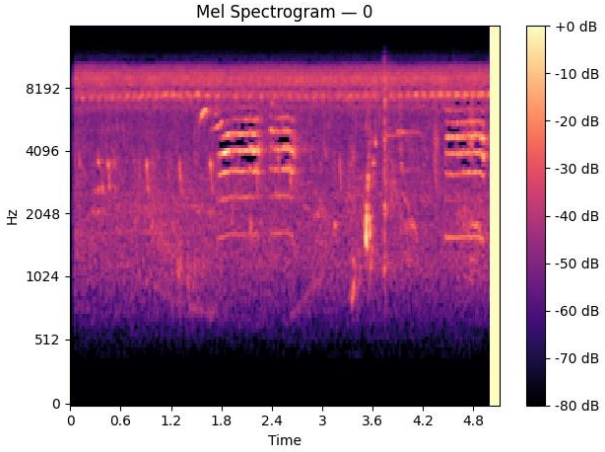

Fig. 7.: high pass filter (200 to 6000 Hz) and band pass filter (5000 to 5000 Hz)

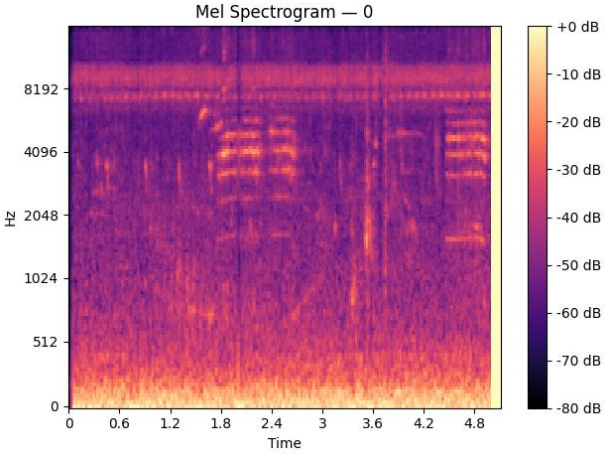

Fig. 8.: clipping distortion of 90%

After augmenting the audio chunk, we also augment the image of the spectogram itself (Fig. 9.). We perform this by clipping out some parts of the image so our model will have a better ability to generalize.

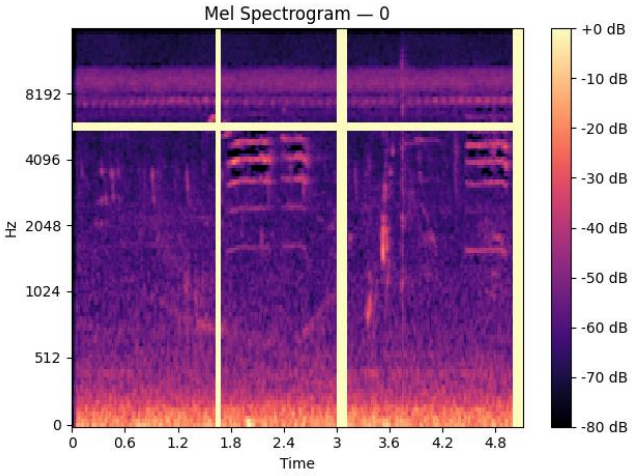

Fig. 9.: augmentation of the spectogram

After these augmentation steps, we have images of identical shape (320x128 pixels) and proper generalization value to be given as input to the convolutional layers of our model.

B. Training

The trainings were conducted both in Google Colab and on our own GPUs. The preprocessing of the audio files made the bottleneck for our training, so the GPU we were training on didn't make a difference there.

The first trainings were on a custom convolutional network, only to make sure our data preprocessing works properly.

Then we changed to ConvNeXtTiny [9]. The first decision in training was to either use ImageNet pretrained weights or train our model from scratch. We made an early training in both cases, shown on the plots below:

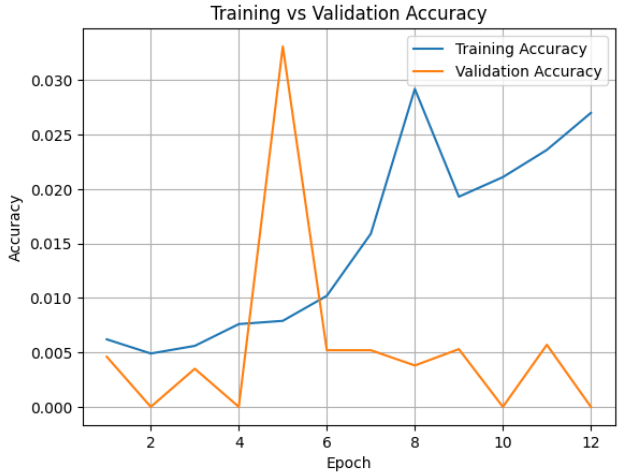

Fig. 10.: results of training from scratch

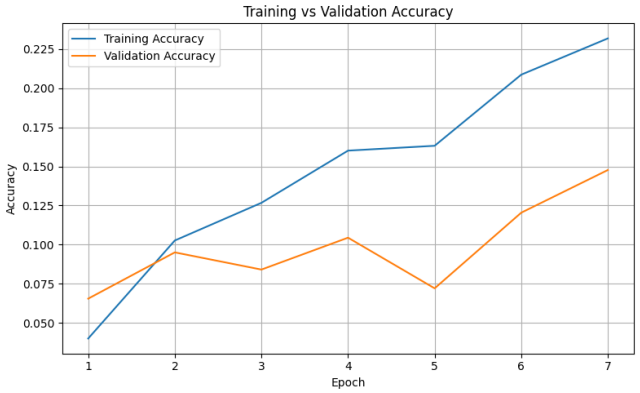

Fig. 11.: results of phase I training with ImageNet weights

We can see that training from scratch is really slow compared to training with pretrained weights, especially with a model this big. Due to lack of resources and time we didn't experiment more with the first case.

In the case of pretrained model, we split our training into 2 phases: phase I training, where we only train the classification head and we save the best weights to start phase II with. In phase II, we unfreeze the whole model and train it with a smaller learning rate.

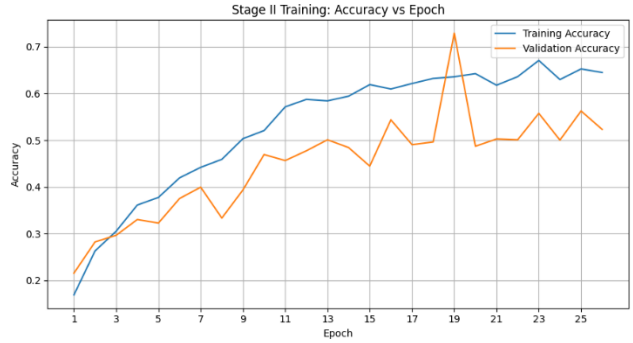

Fig. 12.: early stage II training

C. Hyperoptimalization

Hyperparameter optimization was also carried out in two phases due to limited computational resources. First, we optimized the training setup for Phase I and trained the model accordingly. We saved the weights of our best model (with the best-performing hyperparameters), then continued training in Phase II by loading these weights. Since Phase II required different hyperparameters, we optimized them separately.

During our experiments, we examined the following hyperparameters: learning rate, ReduceLROnPlateau patience, batch size, threshold value (number of samples per class), augmentation strength, the size and depth of the fully connected layers, and dropout rate.

The learning rate is a critical parameter. If it is too high, training can become unstable or even diverge, if it is too low, learning becomes very slow and can easily get stuck in a local minimum. In Phase I, we used a learning rate of 1e-3, which proved stable in our tests. In Phase II, we evaluated multiple values, and found 4e-5 to be the best, we used this value in subsequent experiments. The ReduceLROnPlateau callback halved the learning rate each time the patience parameter was exceeded.

Based on the literature, a larger patience value (e.g., 3–4) would be more optimal so that the learning rate does not decrease too early. However, according to our observations - partly due to the large epoch sizes - even a little drop in validation accuracy could already initiate overfitting. To prevent further training from continuing on an overfitted model, we set patience to 2.

Batch size optimization was constrained by limited resources, so we always used the largest possible batch size. This corresponded to 128 in Phase I and 16 in Phase II.

We made the per-class chunk count after balancing (threshold) a tunable parameter. We tested values of 32, 64, 128, and 256. If the value is too low, in one epoch only a few sound files could be loaded in. A too high value made an epoch really long. We found 128 to be optimal.

We also adjusted augmentation parameters between training runs.

The model's generalization ability is strongly influenced by the number and size of the classifier (fully connected) layers. Initially, we built a single-layer classifier with 1024 neurons. We concluded that the classifier was too small and could not generalize properly. To address this, we added a second hidden layer with 512 neurons after the 1024-neuron layer. This configuration provided seemingly proper generalization, so we continued with these layers.

We inserted dropout and batch normalization layers between the classifier layers. We tested two dropout values: 0.3 and 0.5. The winner was 0.5, as the model was less prone to overfitting.

Our final model contains 29,231,894 parameters. In Phase I, we trained 1,408,694 parameters (the classification head), while in Phase II we trained all parameters.

D. Evaluation

To get certain feedback on the accuracy of our model, we use K-fold cross validation. In our case, K is 4, meaning that 75% of data will be to train our model, and 25% of data will be used to validate our model, and these partitions will iterate through all of our dataset except for the data separated for the test set.

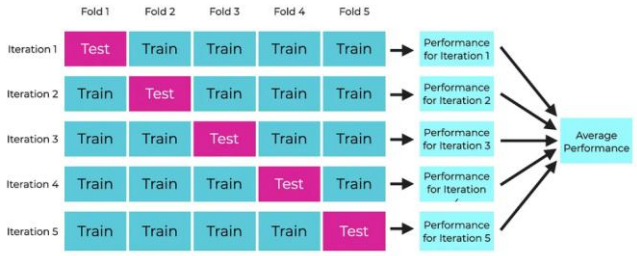

Fig. 13.: the concept of K-fold cross validation (K equals 5 on this figure)

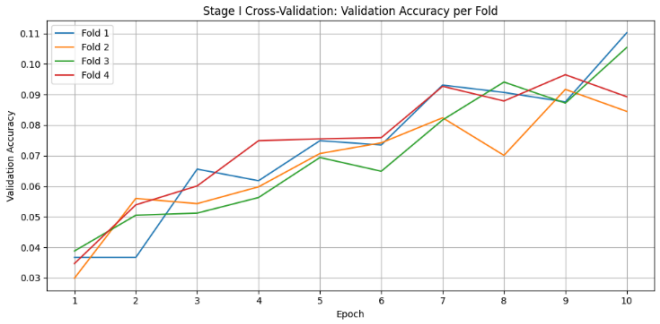

Fig. 14.: Short Stage I training for the 4 folds

On Fig. 14 we can see the validation accuracy in the four train-validation splits. It shows that K-fold works well, it splits the data evenly, and the model doesn't give misleading results.

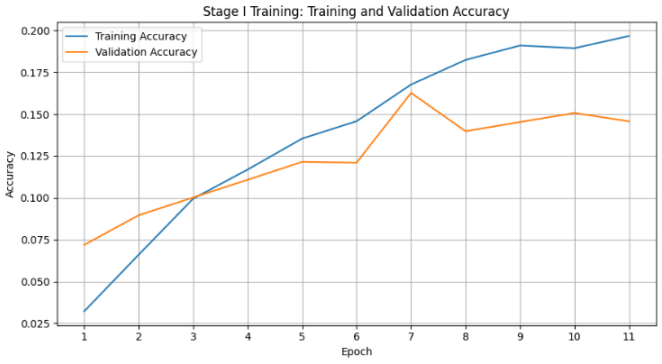

Fig. 15.: Final stage I training accuracy

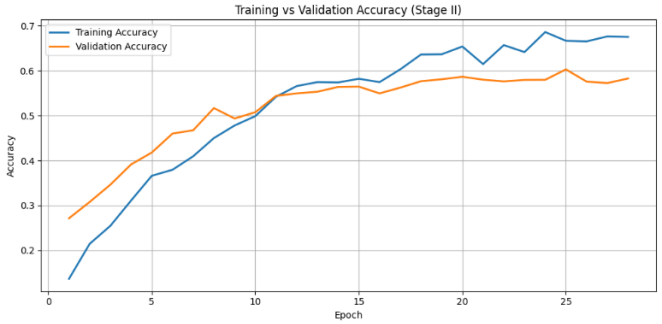

Fig. 16.: Final stage II training accuracy

On Fig. 15 and 16 we can see the two stages of the final training. In the first stage the model reached 16,27% validation accuracy, then in the second, fine-tuning stage it increased to 60,29%.

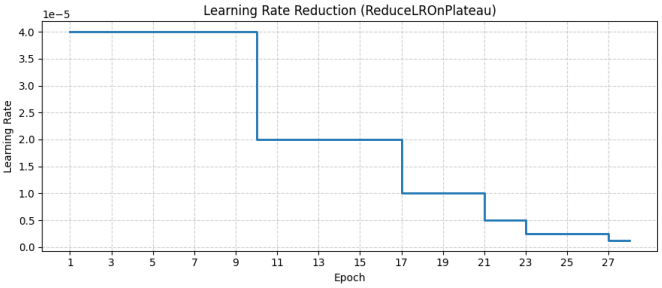

Fig. 17.: Final stage II training - learning rate

Fig. 17 shows the reduction of the learning rate during the stage II training.

E. Testing

Since we couldn't find a test metadata file that could be used as ground truth to test the model, we split the dataset into train (K-fold is used here) and test sets by 85% - 15%.

Based on the model feedback, it achieved 56.82% accuracy on the test dataset, which supports the correctness of the validation accuracy shown in the figure and indicates that the model generalizes well. This is considered particularly good given the nature of the dataset audios, where the bird sounds were often only a short part of the audio file.

This is a relatively strong score compared to random guessing, where the accuracy would be $1/182 \approx 0.005$. Thus, the model performs 103× better than random.

V. CONCLUSION AND FUTURE WORK

Over the course of our work, our objective was to develop a model that classifies the bird classification task as effectively as possible.

First, we explored and became familiar with the provided dataset and its characteristics. We then constructed the training dataset: by applying data augmentation, we balanced the originally uneven class representation. For the model, we employed a composite neural network architecture consisting of convolutional layers followed by feed-forward, fully connected layers. We implemented an existing architecture: we trained the model based on ConvNeXtTiny and evaluated its performance. Another important and non-trivial component of our work was the optimization of our hyperparameters.

As a conclusion, several key lessons can be highlighted. The initial dataset strongly influences subsequent stages of the work: in the presence of class imbalance, augmentation becomes necessary, which is itself a complex and demanding task. The well-known saying that we "stand on the shoulders of giants" is also applicable here, as in this case it proved more effective to implement an existing architecture designed for this purpose than to build a model architecture from scratch. An interesting observation is how substantially the results of different predefined architectures can vary, consequently, identifying the most suitable one for our needs is not a straightforward challenge. Selecting appropriate hyperparameter values also had a major impact on performance. In our case, we worked with a large number of configurable parameters - from augmentation settings through "classical" optimization hyperparameters to the parameters of callbacks.

Overall, the project was a valuable challenge for our team. We all gained considerable experience both professionally and personally through the process of thinking and working as a team. Artificial intelligence - especially deep learning - is of strong interest to all of us, and we therefore plan to continue addressing real-world relevant problems using AI and deep learning approaches in the future. This bird classification task provided an excellent foundation for such continued work.

VI. DISTRIBUTION OF CONTRIBUTIONS

Vilmos explored the scientific field of sound classification with deep learning and collected relevant scientific articles and publications (listed in References).

Tamás downloaded, split and cleaned the data (silent chunks), and Olivér wrote the logic of the lazy data loading.

All team members took part in creating the proper network architecture and training the model.

The same can be said about hyperparameter optimalization and testing, everyone was included.

VII. USAGE OF GENERATIVE AI DURING OUR WORK

During our project, we kept a clear goal to always keep AI usage supervised, and double check every piece of information it provides.

We used the built-in Scholar Labs function of Google Scholar to find relevant scientific works and articles.

We used AI to provide the code needed for efficient and meaningful plotting and data visualization.

We also used AI to write about 70% of the comments in the notebook of the final submission, which were all carefully reviewed by the members of the team.

REFERENCES

[1] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *nature* 521.7553 (2015): 436-444.

[2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).

[3] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (2002): 2278-2324.

[4] Shorten, Connor, and Taghi M. Khoshgoftaar. "A survey on image data augmentation for deep learning." Journal of big data 6.1 (2019): 1-48.

[5] Buda, Mateusz, Atsuto Maki, and Maciej A. Mazurowski. "A systematic study of the class imbalance problem in convolutional neural networks." Neural networks 106 (2018): 249-259.

[6] Zhong, Ming, et al. "Acoustic detection of regionally rare bird species through deep convolutional neural networks." *Ecological Informatics* 64 (2021): 101333.

[7] Sprengel, Elias, et al. "Audio based bird species identification using deep learning techniques." *CLEF (working notes)* 1609 (2016).

[8] Koh, Chih-Yuan, et al. "Bird Sound Classification Using Convolutional Neural Networks." *Clef (working notes)*. 2019.

[9] Xia, Jingsong, Yue Yin, and Xiuhan Li. "An efficient medical image classification method based on a lightweight improved ConvNeXt-Tiny architecture." *arXiv preprint arXiv:2508.11532* (2025).

[10] Prechelt, Lutz. "Early stopping-but when?." Neural Networks: Tricks of the trade. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. 55-69.

[11] Bottou, Léon. "Stochastic gradient descent tricks." Neural networks: tricks of the trade: second edition. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. 421-436.