Identifying Birds from audio recordings using Deep Neural Networks

Gernot Zöcklein

March 1, 2022

Abstract

The following report is a write-up for a project done in scope of the course *Practical Project in AI* at Johannes Kepler University, Linz. The goal of my project was to study different Neural Network architectures for classifying birds from audio recordings. The report aims to be a short summary of what I have done and to describe excerpts of the most interesting experiments I have performed. The project was done under supervision of Dr. Jan Schlüter. The structure of this report is also based on his paper [1]. Code for the project can be found on *github*.

Introduction

Identifying which birds are present in an area can provide important information about the habitat. As birds are high up in the food chain, they are excellent indicators of deteriorating environmental quality and pollution. [2] However, manually labeling sound files is a very tedious task and requires much time from experts. Hence, it would be very valuable to automate, or at least support, the labeling using machine learning approaches. Deep Neural Networks have proven to perform quite well at classifying audio data, which is why they were the focus of this project. The goal of the project was to explore different Neural Architectures and Hyperparameters and to compare and combine these different models.

Data

In order to train Neural Networks, labeled training data has to be available. Fortunately, the organizers of the BirdCLEF-2021 Kaggle Challenge [2] collected more than 60000 labeled short audio snippets of 397 different bird species from the xeno canto website [3]. These data are the main resource for the project.

For each audio recording, both both primary and secondary labels were provided. The primary label denotes the bird which is most present in a recording, and the secondary labels denote other birds present in the background. The length of these recordings is quite short, ranging from a few seconds to a few minutes at most. Moreover, the recordings are weakly supervised, i.e the only information we are given is that the mentioned birds do occur in the recording, but we do not have any information on when they occur.

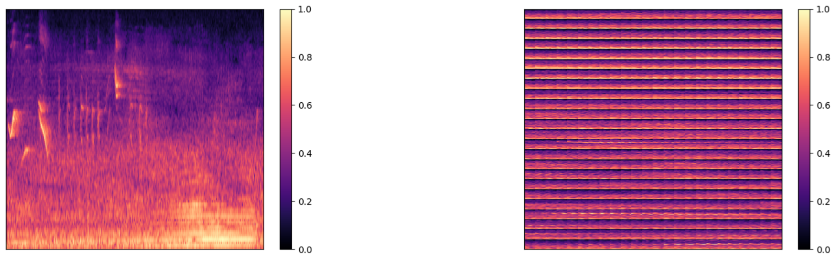
The format of the test data differs quite a lot from the train data, which proves to be a major difficulty. In the test setting, 10 minute snippets are provided, where the user has to provide labels for which birds occur in each 5 second window of the recording.

Models

As already described in the abstract, only Neural Network models were used, as these have proven to be state of the art in bird call classification. In order to keep this report short, the general model architecture for sound event detection will not be presented here. We refer e.g to [1], section 2, for more details.

PaSST: Transformers with Patchout

The first model class we tried were the pretrained PaSST models, as described in [4]. These models are vision transformer models and fine-tuned on the publicly available audioset dataset. These models take standard mel spectrograms as input, as can be seen in Figure 1 (a).



(a) Standard (for PASSts, CNNs) (b) Reshaped (for STFT Transformer)

Figure 1: The different model inputs

STFT Transformer

This model type was first termed and introduced by a participant in the BirdCLEF2021 challenge [5]. The only difference to the previously discussed PaSST's is the way the input is fed into the model. While in PaSST's, the spectrogram is split up into 16×16 2d patches, in this model class, each "patch" consists of just one time slice of the spectrogram, meaning each patch is in fact a 256×1 slice. In order to achieve this, the spectrogram was reshaped, as can be seen in Figure 1 (b). Since using 256 mel bins can lead to unnecessary computation, we only used 128 or 64 mel bins, and then interpolated the spectrogram vertically in order to have 256 dimensional patches.

Convolutional Neural Networks

The last model class we tried were Convolutional Neural Networks, which have been known to achieve state of the art performance for sound event detection tasks. We used models that were pretrained on Imagenet. Similar to PaSSTs, the CNN's also take the basic mel spectrogram as input features.

Training

Excerpts

As already explained previously, not all train recordings have the same length. Moreover, the predictions on the test recordings have to be in 5 seconds windows. Additionally, some model architectures require different input sizes to work with. For all these reasons, excerpts of the given recordings have to be used during the training procedure. That means that during training we used random crops of the recordings. The randomness also brings the benefit that during training, the same exact signal will never be seen twice by a model.

While Convolutional Neural Networks can in theory handle all input sizes due to a pooling layer at the end, the Transformer models are limited in input length. In fact, PaSST models expect 10 second snippets, and as they were also pretrained on audio recordings of that length, the performance drops significantly if they are fed shorter audio. Hence, we had to train the PaSST models on 10 second snippets. As the predictions on the test data will be in 5 second windows, we decided to use 5 second snippets as the input length during training for the CNNs and STFT Transformer models we trained.

However, the train data are weakly supervised. Hence, it might be possible that we introduce a lot of noise to our models if the 5 second cuts we are choosing do not actually contain any bird. In order to solve this issue, we used longer excerpts during training. As the expected input of the model still stays 5 seconds, we cut up the longer snippets into multiple 5 second snippets, fed each of those snippets separately into the model, and then performed a max pooling on the logits. We saw a tremendous performance boost using this approach. We also tried to perform pooling before a final feed forward layer, however in the limited experiments we performed in that setting, this approach seemed to lead to worse performance during inference, and hence we abandoned it early on.

Augmentation

Using data augmentation has been a well known trick for training Deep Neural Networks, as it serves to aid generalization. We used the following basic methods in general:

- Randomly adding Gaussian noise
- Randomly adding Pink noise
- Randomly adding gain
- Randomly performing a slight pitch shift
- Mixup audio in a batch 50% of the time with $\alpha = 0.4$
- Superimpose the recordings with nocall data from the freefield challenge [6].

Optimization

First of all, we split the data into a train and validation set, with a 90% and 10% stratified split, respectively. We used both Adam and AdamW optimizers, though in most experiments we used AdamW with weight decay set to 10^{-4}. In most cases, we set the initial learning rate to $2 \cdot 10^{-5}$. Moreover, we decreased the learning rate by 10^{-1} whenever the validation loss didn't improve significantly for 10 or more mini-epochs. Because of memory requirements, we often had to stick with a batch size of 8 or 16, though we saw most success with higher batch sizes, such as 32. In order to train with larger batch sizes on machines with a small GPU memory, we used smaller batch sizes and accumulated gradients of multiple mini batches.

Other Tricks

We also used the following techniques in some of our models, mainly to have a less homogeneous model base for better ensemble results:

- Label Smoothing, i.e setting even the labels of classes not occuring in a given recording to 1/396

- As the loss is computed as the mean of the binary cross entropy loss of each one out of the 397 birds that could possibly occur, we sometimes changed the weight of loss terms associated with birds that were actually present in a recording by a factor between 3 and 10.

Inference and Validation

As already mentioned previously, the inference task is quite different to the training scenario, as we have to automatically annotate multiple 10 minute recordings.

Getting good results on 5 second windows with PaSST models, which expect 10 second windows, proved to be a major challenge. The first thing we tried was to window predictions, i.e to cut overlapping 10 second windows from the test sound scapes and then average the logits in order to get predictions for 5 second windows. The next thing we tried was to pad 5 second audio snippets with 5 seconds of no signal whatsoever. However, what finally worked best was to simply cut out 5 second windows, and duplicate and concatenate them in order to receive 10 second inputs for the PaSST models. Unfortunately, we never quite managed to come close to the extraordinary performance that PaSST models had on the validation set, and consider it an interesting question to further pursue the reason for this. Of course the CNN and STFT Transformer models could directly predict on 5 second snippets by design.

In order to validate our models on data more closely resembling the test data, and in order to choose good thresholds for bird predictions, we used the long train audio, also made available from the BirdCLEF 2021 challenge hosts. These are 20 labeled audio files that have the same format as the test data. We used these data to select the classification thresholds we used during inference on the challenge data. Moreover, we computed the F1 score for segments with birds and with no birds separately, and then weighted them by the percentage of call vs. nocall labels in the public challenge data, as described in [5], section 4.3.

Experiments

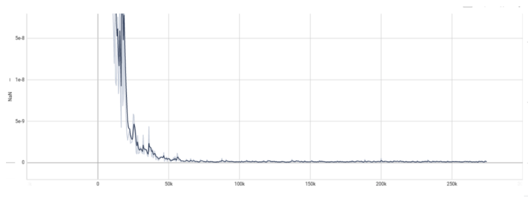
We describe some experiments we did with the different model classes, and report the main results.

PaSST

As there is no straightforward way to perform predictions on 5 second windows for a PaSST model that was trained on 10 second audio, the first thing we tried to do was to train only on 5 second snippets to make the model more suited for our inference task. However this did not work at all. Hence from then on we decided to focus on 10 second snippets. We tried two different pretrained model architectures provided by the PaSST repository [7], the only difference in the two models being the stride that is used when generating patches. In the first model, which we call PaSSTs10, a stride of 10 is used when generating patches from a mel spectrogram, meaning there is overlap between the patches. In the second model, which we call PaSSTs16, a stride of 16 is used when generating patches, and hence there is no overlap. Of course, the former architecture takes a substantial amount more GPU memory and computation effort. Both model classes perform exceedingly well on the local validation data, achieving F1 scores of about 0.69. That is better than any other model we have tried. Note that the predictions on the validation data was done on 10 second snippets for PaSSTs and on 5 second snippets for the other model classes, and hence the comparison is not quite fair. Unfortunately, the impressive results on the local validation set do not translate to the inference data. We conjecture that one reason for this may be that PaSST models are overconfident, leading to a lot of false positives. The main results are summarized in the table below.

Model Type	Validation F1	Long Train F1	Public LB F1
PaSSTs10	0.689	0.6876	0.6505
PaSSTs16	0.685	0.6947	0.6638
ensemble	-	0.7154	0.6774

Table 1: Comparison of the best PaSST models


(a) Gradients before tweaking

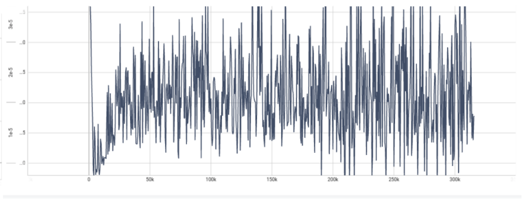

(b) Gradients after tweaking

Figure 2: Average gradients of first layer over time

STFT Transformers

It took us quite a while for the STFT Transformer models to produce any meaningful results. In fact, the majority of time we invested into this model type was simply to get it to work. Initially the models simply did not seem to learn anything at all. After an initial investigation, we found out that the gradients were dying our really quickly. We initially tried to make the models converge by using different learning rate schedulers, such as CosineAnnealing scheduler and OnceCycleLR scheduler, however this did not yield the desired results. We also tweaked the parameters for generating the mel spectrograms. We found that using 128 mel bins and then linearly interpolating them to dimensionality 256 worked best, but this barely improved the main issue of vanishing gradients.
What finally led us to achieve reasonable results was to normalize the mel spectrograms before training and to use a very small´ initital learning rate. We found that an initial learning rate between $2 \cdot 10^{-5}$ and 10^{-6} worked best. We tried different normalization schemes, such as simply substracting the sample mean and dividing by the sample variance and doing a min-max scaling. We found that there was not a large difference between the normalization methods, though min-max scaling worked slightly better. After all this tweaking, the gradients finally stayed stable during training and our models started to learn, as can be seen in figure 2.

Once we figured out how to make STFT Transformers converge, we started to experiment with different architectures and other settings. We tried ViT-tiny, ViT-small, Vit-base [9] and the destilled variant of ViT-base [10]. We used the timm backend for training all of those models [11]. As expected with Transformers, the larger the model is, the better the results are. While the training setup for the above four models initially stayed identical, we also tried one ViT-base model where we used label smoothing and changed the weights for the secondary labels to 0.8. We call this model ViT-BaseLS. Interestingly enough, this model had the best performance on the validation set, but worse performance on the challenge data than a standart ViT-Base model. We conjecture that this model is good at classifying bird call, but overconfident, which leads to a lot of false positives during inference. The main results are summarized in the table below.

Model Type	Validation F1	Long Train F1	Public LB F1
ViT-tiny	0.51	0.6936	0.6760
ViT-base	0.563	0.7174	0.7124
ViT-BaseLS	0.586	0.6858	0.6922
Vit-base-distilled	0.535	0.7160	0.6938
ensemble	-	0.7315	0.7303

Table 2: Comparison of the best STFT Transformer models

Convolutional Neural Networks

In stark difference to the other two model types, pretrained Convolutional Neural Networks immediately worked very well. This was to be expected, as CNNs have been the state of the art for this task for quite some time. We initially experimented with the pretrained model architectures, switching between a resnet with 50 layers and an efficient net version, though we quickly found the resnet to be superior and hence stuck with it for most other experiments. The best single model we achieved was a basic resnet with 50 layers, without any deviations from our standard training procedure, we label this model resnet50b. Other models that performed well were a resnet with 50 layers, where we weighted the loss for bird occurences and also set the weight for the secondary labels to 0.8 instead of 1, we call this model resnet50w. We also trained one similar resnet, but instead of manual normalization of the mel spectrograms, we used batchnorm, this model is denoted by resnet50n. Finally, we also trained one resnet with only 34 layers. The results are summarized in the table below.

Model Type	Validation F1	Long Train F1	Public LB F1
resnet50b	0.6	0.7038	0.7253
resnet50w	0.53	0.6841	0.7130
resnet50n	0.54	0.7027	0.7117
resnet34	0.515	0.6820	0.7044
ensemble	-	0.7290	0.7403

Table 3: Comparison of the best CNN models

Binary bird call classifier

As 54% of segments of the public test set contained no bird calls at all, we figured that a classifier that only predicts whether a bird or not is present at a given recording could be helpful. In order to train the classifier, we used the three datasets containing labelled call and nocall recordings [8], which led us to roughly 54000 labeled 10 second segments of bird calls, on which we trained a standard resnet, with the same optimization setting as for the ¸multilabel classifiers. The model achieved an accuracy of 92.5% on the validation data.

Ensembling

It is already well known from previous machine learning challenges that ensembling multiple models will lead to better results on the leaderboard. Hence, we performed inference on the test data with 10 different models separately, and then averaged the logits of all models to get final predictions for each 5 second window. We also tried to perform a windowing of the final predictions, however this did not lead to improvements in performance. In order to also use the binary birdcall classifier in our ensemble, we tried two different approaches. The first one was to find a separate threshold for the binary classifier, and label all snippets as nocall that exceeded the threshold of the binary classifier's predictions. The second approach was to perform a weighting of the probabilities of the ensemble classifier by the predicted probability of a bird presence from the binary classifier. This approach also meant that we only had to tune one threshold instead of two. In the end, this method worked best. The results are summarized in the table below.

Ensemble Type	Long Train F1	Public LB F1	Private LB F1
No binary information	0.7512	0.7569	0.6420
Separate threshold	0.7574	0.7243	0.6527
Weighted	0.7607	0.7366	0.6617

Table 4: Comparison of ensemble types

Conclusion

We conclude that Transformer models can catch up to the performance of Convolutional Neural Networks, although they require a lot more fine tuning. In any case, they prove valuable in a final ensemble, and hence we think that they should be considered in future bird call call classification challenges. Moreover, we believe that these type of Transformer models form an interesting line of inquiry for future competitions and we conjecture that it might be possible to achieve even better results when using Transformers that are pretrained on audioset, like PaSSTs. While unfortunately PaSSTs did not work that well for the given inference problem, we believe that the problem could be solved by considering the requirements for the length of the snippets already during pre training. Moreover, we believe that Transformers with sparse attention mechanisms, such as BigBird [12], could prove to be very well suited for such a task. Unfortunately there were no straightforward implementations of BigBird for Pytorch yet, and hence we did not have time to include it in our experiments.

Acknowledgmenets

First and foremost, I want to thank Dr. Jan Schlüter for suggesting me this interesting topic and for the countless tips and bits of advice he has given me. Moreover, I want to thank him for providing me with some code snippets that made my training process more efficient. I also want to thank all participants of the BirdCLEF-2021 Kaggle challenge that contributed to the discussion and shared their ideas at the end of the competition, I truly learnt a lot from them. Lastly, I want to thank the Institute of Computational Perception at the Johannes Kepler University in Linz for providing me with GPU's, as without them I could not have performed the numerous experiments.

References

[1] Schlüter, Jan, Learning to Monitor Birdcalls From Weakly-Labeled Focused Recordings, `http://ceur-ws.org/Vol-2936/paper-139.pdf`

[2] BirdCLEF 2021, `https://www.kaggle.com/c/birdclef-2021/overview`, last accessed on 26.02.2022

[3] `https://xeno-canto.org/`, last accessed on 26.02.2022

[4] Koutini et. al, Efficient Training of Audio Transformers with Patchout, `https://arxiv.org/pdf/2110.05069.pdf`

[5] Puget, Jean-Francois, STFT Transformers for Bird Song Recoginition, *https://github.com/jfpuget/STFTTransformer*, last accessed on 26.02.2022

[6] Stowell et. al, An open dataset for research on audio field recording archives: freefield1010, `https://arxiv.org/abs/1309.5275`

[7] `https://github.com/kkoutini/`, last accessed on 26.02.2022

[8] `https://dcase.community/challenge2018/task-bird-audio-detection`, last accessed on 26.02.2022

[9] Dosovitskiy et. al, An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, `https://arxiv.org/abs/2010.11929`

[10] Touvron et. al, Training data-efficient image transformers & distillation through attention, `https://arxiv.org/abs/2012.12877v2`

[11] Wightman, Ross, PyTorch Image Models, `https://github.com/rwightman/pytorch-image-models`, last accessed on 26.02.2022

[12] Zaheer et. al, Big Bird: Transformers for Longer Sequences, `https://arxiv.org/pdf/2007.14062.pdf`