Large-Scale Bird Sound Classification using Convolutional Neural Networks

Stefan Kahl[1], Thomas Wilhelm-Stein[1], Hussein Hussein[1], Holger Klinck[2], Danny Kowerko[1], Marc Ritter[3], and Maximilian Eibl[1]

[1] Technische Universität Chemnitz, Straße der Nationen 62, 09111 Chemnitz, Germany
[2] Bioacoustics Research Program, Cornell Lab of Ornithology,
Cornell University, 159 Sapsucker Woods Road, Ithaca, NY 14850, USA
[3] Hochschule Mittweida, Technikumplatz 17, 09648 Mittweida, Germany
{stefan.kahl, thomas.wilhelm-stein, hussein.hussein,
danny.kowerko, maximilian.eibl}@informatik.tu-chemnitz.de,
holger.klinck@cornell.edu, ritter@hs-mittweida.de

Abstract. Identifying bird species in audio recordings is a challenging field of research. In this paper, we summarize a method for large-scale bird sound classification in the context of the LifeCLEF 2017 bird identification task. We used a variety of convolutional neural networks to generate features extracted from visual representations of field recordings. The BirdCLEF 2017 training dataset consist of 36.496 audio recordings containing 1500 different bird species. Our approach achieved a mean average precision of 0,605 (official score) and 0,687 considering only foreground species.

Keywords: Bioacoustics, Large-Scale Classification, Convolutional Neural Networks, Audio Features, Bird Sound Identification, BirdCLEF 2017

Source code of this project: https://github.com/kahst/BirdCLEF2017

1 Introduction

1.1 Motivation

Identifying bird species based on their calls, songs and sounds in audio recordings is an important task in wildlife monitoring for which the annotation is time consuming if done manually. With the arrival of convolutional neural networks (CNNs, ConvNets), automated processing of field recordings made a huge leap forward [1]. Nonetheless, processing large datasets containing hundreds of different classes is still very challenging. In the past years, many ground breaking CNN architectures evolved from evaluation campaigns such as TREC, CLEF or the ILSVRC [2][3][4]. Adapting those architectures for the purpose of audio event detection has become a common practice despite the very different domains of image and audio inputs. Generating deep features based on visual representations of audio recordings has proven to be very effective when applied to the classification of audio events such as bird sounds [5][6].

1.2 Dataset

The BirdCLEF 2017 [7][8] training data is built from the Xeno-Canto collaborative database[1] and contains 36.496 sound recordings with a total number of 1500 species (50% increase from the 2016 dataset). Most audio files are sampled at 44.1 kHz, 16 bits, mono and show a wide variety of recording quality, run length, bird count and background noise. The training set has a massive class imbalance with a minimum of four recordings for *Laniocera rufescens* and a maximum of 160 recordings for *Henicorhina leucophrys*. The training data is complemented with XML-files containing metadata such as foreground and background species, user quality ratings, time and location of the recording and author name and notes. We did not make use of any of the additional metadata except for the class id of foreground species. The presence of numerous background species distorts the training data and makes single label training particularly challenging.

2 Workflow

Our workflow consists of four main steps. First, we extract spectrograms from all audio recordings. Secondly, we extend our training set through extensive dataset augmentation. Next, we try to find the best CNN architecture with respect to number of classes, sample count and data diversity. Finally, we train our models using consumer hardware and Open Source toolkits and frameworks.

2.1 Generating Spectrograms

We decided to use magnitude spectrograms with a resolution of 512x256 pixels, which represent five-second chunks of audio signal. This (relatively large) input size is computationally expensive when training ConvNets but our experiments showed that high resolution spectrograms contain more valuable details and the overall classification performance benefits from larger inputs.

We extracted five-second spectrograms for each sound recording using a four-second overlap, which resulted in 940.740 images. We implemented a heuristic to decide whether a signal chunk contains bird sounds or background noise only. We mainly adapted the approach of [1] and [9] and removed spectrograms with improper signal to noise ratio. Figure 1-3 visualize this process. We selected 869 spectrograms containing heavy background noise and no bird sounds for our dataset augmentation process.

Despite this method for signal and noise separation, the training data remains distorted. The classification error depends on clean, distinct classes, which is almost impossible to achieve if done automatically. Species present in the audio recordings are not time coded. Therefore, background species might interfere with feature learn-

[1] http://www.xeno-canto.org

ing, especially for species with only few training samples. The amount of training samples greatly influences the generalization error. More samples significantly improve the detection rate; we noticed that classification of species with more than 1000 spectrograms performed best. Class imbalance affects generalization as well. We tried different techniques like cost-sensitive learning to counter this circumstance but noticed that those methods did not lead to a higher mean average precision. However, reducing class imbalances seems to benefit real world applications focused on rare species.

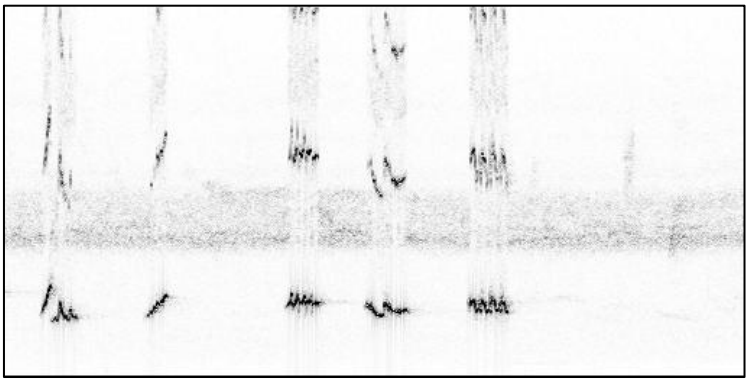

Fig. 1. Magnitude spectrogram after signal preemphasis and value normalization. We use the framework "python_speech_features"[2] for FFT with a window length of 0.05 and step size of 0.0097 for five-second chunks of the signal. We use a FFT length of 840 and crop high frequencies, which reduces the input size and is sufficient for most bird species. We did not perform any noise reduction.

Fig. 2. Processed spectrogram after median blur, median threshold, spot removal and morphological closing. This approach suppresses background noise and highlights actual bird sounds.

[2] http://python-speech-features.readthedocs.io

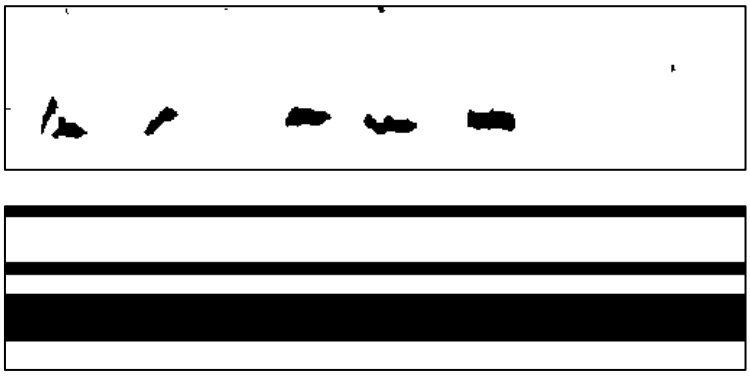

Fig. 3. We only analyze the low frequency crop of the processed spectrogram (top) and decide whether it contains signal or noise (bottom). We use the sum of all rows containing signal (black) as threshold for bird presence or absence.

2.2 Dataset Augmentation

Dataset augmentation is vital to reduce the generalization error. However, most established augmentation methods like horizontal flip and random crop are not suitable for spectrograms as they might mask the original signal. Dataset augmentation should always target the properties of the test data, which are underrepresented or missing in the training data. We evaluated different augmentation methods using a local validation set consisting of ~50.000 samples from 100 species. We incorporated the following augmentations into our final runs:

Vertical Roll: Following [1] we implemented a random, pitch shifting vertical roll of maximum five percent, which had great impact on the generalization error. This seems to be by far the most beneficial dataset augmentation. We tried time shifting horizontal roll as well, but found that this augmentation harms generalization. This might be because we generated overlapping spectrograms and with that, already used time shifted spectrograms.

Gaussian Noise: Synthetic noise often helps convolutional neural networks to focus on salient image features. Most models will learn to ignore the noise over time, which makes them robust even against other (more realistic) noise sources. We simply added Gaussian noise of random intensity to our spectrograms and re-normalized the resulting images.

Noise Samples: In addition to random Gaussian noise, we added noise samples (spectrograms our heuristic rejected as bird sounds) which significantly improves the classification result and speeds up the entire training process. Most of the sound recordings show similar noise patterns; we tried to counter these patterns with the selection of 869 noisy spectrograms and randomly added them to our training images.

Batch Augmentation: Most sound recordings contain more than one bird species, which may vocalize at the same time. We tried to simulate this by randomly combining spectrograms of the same batch. Combining samples of the same class will not affect label distribution, whereas the combination of samples of different species results in multi-label targets that can be used to train sigmoid outputs.

We applied all augmentations at runtime, during training using CPU idle time. We implemented a multi-threaded batch loader, which significantly speeds up training. Our batch loader operates during a forward-backward pass iteration executed on the GPU.

2.3 CNN Architecture

Finding the best CNN architecture is a time consuming task and often done purely by intuition. Current state-of-the-art approaches try to tackle this issue with automated hyperparameter search [10]. We decided to reduce the amount of possible design decisions and relied on current best practices for CNN layouts. All weighted layers (except for input and output layers) use Batch Normalization [11], Exponential Linear Units (ELU) for unit activation [12] and are initialized using He-initialization [13]. We wanted large receptive fields in our first convolutional layers, which have proven to be very effective for spectrograms during our experiments. We use filter sizes of 7x7 and 5x5 for larger inputs and 3x3 kernels for smaller input sizes in deeper layers. **Table 1** provides an overview of the three model designs we used for our submission.

Although the BirdCLEF classification task with 1500 classes, class imbalances and a distorted dataset is rather complex, shallow CNN architectures with classic layouts and only a few layers seem to be more effective than more complex highway networks with multiple tens of layers like DenseNet [14] or ResNet [15]. We tried different implementations of state-of-the-art convolutional networks but found them inferior to our simple CNN architectures. This might be due to the fact, that the image domain of spectrograms is very homogenous despite more than 1500 different signal types. Most spectrograms contain only little information, leaving most pixels blank. This observation is backed by the works of [1][5][6].

Large input sizes are not common in current image classification publications. Most approaches reduce the input size to a maximum of 256x256 pixels. Current consumer GPUs are well suited for larger inputs. On the other hand, models with large input sizes are considerably harder to train and tune, training takes significantly more time and larger inputs do not always benefit generalization. Our experiments showed that non-square, high-resolution inputs of spectrograms do indeed achieve better classification results especially for large and diverse datasets. We used strided convolutions and pooling layers to cope with large inputs.

Additionally, a larger number of filters seems to be more effective than a larger number of hidden units. We found that 512 units per dense layer is sufficient even for

1500 classes. Determining the right amount of network parameters is crucial to avoid under- and overfitting. This process is also very time consuming considering the fact that less parameters might work well on small validation sets but usually underfit on larger datasets. Validation experiments should always show slight overfitting in order to have good generalization capacity when trained on more classes. Even though, models with a large number of weights did eventually overfit during our experiments with 1500 classes, so we decided to dial down the weight count.

Table 1. Model architectures used for our submitted runs. Every convolutional and every dense layer (except the final output layer) has batch normalization before its ELU activations, is He-initialized and has shape preserving padding. Model 1 is our main model, used most in our runs. Model 2 and 3 are part of our CNN ensemble; Model 2 showed best generalization after few epochs but needs significantly more time to train than Model 1; Model 3 is used for the classification of subsets of the training data with a maximum of 500 classes. Shallow models perform better on small class selections and can be combined in ensembles if trained with sigmoid outputs.

Model 1 8 weighted Layers ~4000s per Epoch	Model 2 9 weighted Layers ~5500s per Epoch	Model 3 8 weighted Layers ~1000s per Epoch
Conv1, 64x7x7, Stride 2	**Conv1, 32x7x7, Stride 1**	**Conv1, 32x7x7, Stride 2**
MaxPooling, Size 2	MaxPooling, Size 2	MaxPooling, Size 2
	Conv2, 32x5x5, Stride 1	
	MaxPooling, Size 2	
Conv2, 128x5x5, Stride 1	**Conv3, 64x5x5, Stride 1**	**Conv2, 128x5x5, Stride 1**
MaxPooling, Size 2	MaxPooling, Size 2	MaxPooling, Size 2
Conv3, 256x3x3, Stride 1	**Conv4, 128x3x3, Stride1**	**Conv3, 256x3x3, Stride 1**
MaxPooling, Size 2	MaxPooling, Size 2	MaxPooling, Size 2
Conv4, 512x3x3, Stride 1	**Conv5, 512x3x3, Stride 1**	**Conv4, 512x3x3, Stride 1**
MaxPooling, Size 2	MaxPooling, Size 2	MaxPooling, Size 2
Conv5, 1024x3x3, Stride 1	**Conv6, 1024x3x3, Stride 1**	**Conv5, 512x3x3, Stride 1**
MaxPooling, Size 2	MaxPooling, Size 2	MaxPooling, Size 2
DenseLayer, 512 Units	**DenseLayer, 512 Units**	**DenseLayer, 512 Units**
Dropout, p=0,5	Dropout, p=0,5	Dropout, p=0,5
DenseLayer, 512 Units	**DenseLayer, 512 Units**	**DenseLayer, 512 Units**
Dropout, p=0,5	Dropout, p=0,5	Dropout, p=0,5
DenseLayer, 1500 Units	**DenseLayer, 1500 Units**	**DenseLayer, <500 Units**
Softmax Output	Softmax Output	Sigmoid Output

Most recent approaches at well-known evaluation campaigns use CNN ensembles to achieve their best classification results. Separate predictions are combined (bagging and boosting) to form the final ranking. We trained 19 convolutional neural networks and selected seven of them for our ensemble submission. Ensembles may not be ap-

plicable for real world tasks such as real-time wildlife monitoring but effectively boost the overall classification performance.

2.4 Training

Time efficient training becomes crucial when training on 1500 classes with more than 940.000 samples. We tried to optimize our training process in order to save computation time and maintain a good overall performance at the same time. We evaluated different kinds of parameter settings and found the following to be very effective:

Learning Rate Schedule: The learning rate is one of the most important hyperparamters when training ConvNets. Fixed learning rates may hinder the optimization process from converging. Common practice uses learning rate steps, which reduce the learning rate on various occasions during training. Although batch normalization allows for larger learning rates, in order to achieve full convergence of the learning process, parameter changes have to be minimal near the end of training. We found that linear interpolation of the learning rate during training, with changes applied after each epoch, are very effective and can dramatically improve the classification result. We started our training process with a learning rate of 0.01 and decreased it over 55 epochs to a value of 0.00001.

Optimizer: Choosing the best optimizer for stochastic gradient descent parameter updates is vital for fast optimization convergence. We decided to use ADAM updates [16] (with the beta1 parameter set to 0.5) because of the high convergence speed the algorithm provides. In combination with our dynamic learning rate (which is still beneficial despite the adaptive nature of the optimizer), we achieved a significant speed-up compared to the Nesterov momentum.

Loss function: We use categorical cross entropy and binary cross entropy as loss functions for single and multi-label scenarios. We applied L2 regularization with a weight of 0.0001. Additionally, we experimented with different kinds of cost-sensitive loss functions, which increase the loss for misclassifications of rare species. Massive class imbalances may lead to a good overall classification accuracy just because of the dominance of single species. Incorporating class probability distributions into the loss function counters this effect if added to the loss alongside cross entropy and L2 distance (higher penalty if class is less probable). For the BirdCLEF 2017 challenge, this method turned out to be ineffective, but we observed a very clean confusion matrix for rare species, which might indicate a real world application of this approach.

Pre-trained Models: Re-using already trained models for new training iterations can cut the computation time needed until convergence by a great margin. Softmax classifier tend to be much more efficient when training ConvNets. Therefore, we trained models with single label outputs and used these pre-trained models as starting point for our multi-label scenarios with sigmoid outputs. Doing that, we were able to

skip 20-30 epochs of training time per model. Some of our ensemble models were trained on different subsets of the training data. We made use of a pre-trained model every time we switched to new subsets.

Batch Size: Increasing the size of batches for the training process is beneficial mostly due to the use of batch normalization. Smaller batches lead to more iterations per epoch and tend to perform better after the first few epochs. In the end, larger batches seem to provide better generalization. Choosing the best batch size always depends on the amount of VRAM the GPU provides. We had to set the batch size to 128, which was the largest we could fit in memory for all models, mainly constraint by the large receptive fields we used in the first layers of our ConvNets.

The implementation of our code is done purely in Python using *NumPy*, *Theano* [17] and *Lasagne* [18] for models, objectives and solvers, *OpenCV* for image processing, *scikit-learn* for metrics and *Matplotlib* for visualization. We did all of our experiments on a single PC with a NVIDIA Titan X graphics card. We switched to a NVIDIA P6000 GPU for the training of our final models, which provides 24GB of VRAM and two times faster training.

We used a local validation split of five percent of the training spectrograms to monitor the training process and limited the total number of samples per class to 1500. Training took between 15h and 80h per model on all 1500 classes and ~4h for our 100 class experimental models. We trained every model for 55 epochs and used early stopping to find the best parameter setting. Some models showed their best performance after 55 epochs, which indicates that longer training periods may have been beneficial. However, we did not proceed training these models due to time constraints.

3 Evaluation

3.1 Performance on Local Test Set

We used a local test split of the given training data to evaluate our ConvNets after training. Therefore, we randomly separated 10% of all recordings (at least one file per species) for our test set. Our test set reflects the dataset distribution of species relatively well and results are comparable to the official scores. The local test data contains 3557 recordings of varying recording quality and length.

We used fixed random seeds and trained every CNN for 55 epochs, selecting the best performing snapshot according to the validation loss on a 5% validation split of the input spectrograms. **Table 2** shows selected results of more than 100 different experiments, which we conducted. Due to time constraints, we did not manage to test all possible hyperparameter and dataset augmentation combinations, especially for our DenseNet and ResNet architectures. However, our experiments indicated that classic, carefully tuned CNN layouts outperform highway networks (our most com-

petitive model was a DenseNet-32). On the other hand, CNNs with shortcuts need significantly less parameters and usually scale with increasing parameter count. There might still be a lot of potential laying in those architectures if carefully crafted.

Table 2. Model evaluation on local test split which contains at least one sample per species. Input size was 512x256 pixels (if not stated otherwise) of five-second chunks of each recording with no overlap taking ~800ms for each sample prediction. Overlapping consecutive spectrograms boost the MAP by ~2% but takes considerably more time to process. We were not able to evaluate DenseNet or ResNet architectures with more than 50 layers due to limited resources.

CNN Type	Description	MAP FG+BG	MAP FG
Model 1	No dataset augmentation	0,481	0,553
Model 2	No dataset augmentation	0,468	0,537
Model 3	No dataset augmentation	0,455	0,527
Model 1*	Dataset augmentation	0,583	0,671
Model 1**	Dataset augmentation, sigmoid activations	0,559	0,643
Model 1	Dataset augmentation, 10s spectrograms	0,554	0,645
Model 1	Dataset augmentation, 256x128px specs	0,576	0,663
Model 2	Dataset augmentation	0,573	0,661
Ensemble***	Seven models, average pooling	**0,629**	**0,711**
DenseNet-32	Dataset augmentation	0,558	0,642

*used as model for Run 1 **used as model for Run 2 ***used for Run 3

We selected the best models based on the results of our local test set evaluation for our submission. Additionally, we selected seven ConvNets for an ensemble. Predictions were pooled only by averaging the probabilities for every species based on the prediction for all five-second spectrograms of every recording. We tried numerous prediction pooling strategies like linear interpolation, thresholds or dilation but found none of them outperforming simple average pooling. However, fine-tuning the prediction process can lead to significantly better results for the same tested model.

3.2 Official Scores

We submitted four runs, each of them pursuing a different strategy. Our submission contains the result of two single models (Run 1&2) and the predictions of two ensembles (Run 3&4). All runs are fully automatic with no manual interference. Only Run 4 uses additional metadata.

TUCMI Run 1: This run was composed of the predictions of a single model (Model 1, see **Table 1**) with softmax activations to demonstrate our best performing model on a single label task. Prediction took an average of 833ms per sample recording (on a P6000 GPU).

TUCMI Run 2: We used the fully trained net from our first run as pre-trained model for this attempt of a multi label predictive CNN with sigmoid activations. We used batch augmentation with an average of two labels per sample of each batch to simulate simultaneously vocalizing bird species. Expectedly, this net did not score as good as our first model due to the distorted training set, which makes multi label predictions very challenging. It performed slightly better in the soundscape domain, which was the focus of this attempt. However, we expected a significant difference between both runs for the soundscape recordings, which was not the case. Prediction took an average of 950ms per sample recording.

TUCMI Run 3: Ensembles of CNNs are widely used in evaluation campaigns such as TREC or CLEF. Despite their lack of real world application, ensembles often score best, which is also the case for our seven-model ensemble. Bagging predictions benefits from models trained on different portions of the training data. We decided to train four models on species containing up to 300, 500, 1000 and 2000 training samples (Model 3), one model trained on 256x128 pixel spectrograms (Model 2) and both models of our first two runs. This run is our best performing attempt; prediction took an average of 6s per sample recording due to sequential testing.

TUCMI Run 4: Dedicated models tend to perform better if the number of expected audio events is fixed. We tried to estimate the most probable bird species present in the soundscape recordings based on the given geo-coordinates and the corresponding eBird frequency bar charts for the months of June, July and August. We ranked species based on the probability of occurrence in the Loreto/Peru area and trained a second ensemble for 100 selected species with different CNN layouts and multi label predictions. This is our only metadata assisted run, focused solely on soundscape prediction and performed similar to our models trained on 1500 species. Prediction took an average of 4s per sample recording due to sequential testing.

Table 3. Official scores (Mean average precision) of our submitted runs. Our ensemble scored better in all categories than our other runs; our dedicated model trained on selected species performed nearly as good as the models trained on all 1500 species for soundscape recordings.

Run	FG+BG	Only FG	Soundscapes 2017*	Soundscapes 2016
1	0,564	0,644	0,111	0,063
2	0,547	0,652	0,131	0,064
3	**0,605**	**0,678**	**0,162**	**0,079**
4	0,064	0,068	0,091	0,062

*only the 2017 soundscapes were time-coded (predictions every five seconds)

3.3 Additional Scores

The soundscape domain with multiple birds vocalizing at the same time, diverse and noisy backgrounds and, most importantly, no explicit training data is by far the most challenging test set. We tried to tackle these difficulties with a dedicated model

ensemble, trained on specific bird species only. Considering the overall results for the 2017 time-coded soundscapes, our Run 4 did not perform as expected. However, additional evaluation results kindly provided by the organizers (**Table 4**) show, how important the selection of the right bird species for neural net training can be. This is important for future real world applications of wildlife monitoring. The results verify that dedicated models specialized for the identification of bird species of a specific region outperform general models trained for the detection of a wide variety of bird species.

Table 4. Additional results (MAP) provided by the organizers for time-coded soundscape recordings detailed by country. Our dedicated Run 4 (trained on samples for bird species most probable for Loreto/Peru) outperforms all other runs by a great margin for soundscapes recorded in Peru.

Run	All	Colombia	Peru
1	0,099	0,101	0,003
2	0,119	0,121	0,007
3	**0,144**	**0,146**	0,026
4	0,061	0,059	**0,158**

This basically implies two detection strategies: Either limiting the bird species during the training of dedicated models or using probability measures based on species appearance for general models to refine classification results. The second option seems to be the most flexible, allowing for the adaption of one model to multiple scenarios such as changing seasons or the relocation of monitoring systems without the need to train a new model. Future experiments will have to show whether both methods perform equally good.

4 Source Code

We made a refined and commented version of our source code alongside detailed instructions publicly available on GitHub[3]. This repository enables everyone to reproduce our submissions, to train own models and evaluate results. We added our selected noise samples and a pre-trained model from our first run. We will keep the repository updated and will add some functionality for demo applications in the future. If you have any questions or remarks regarding the source code, please do not hesitate to contact us.

[3] https://github.com/kahst/BirdCLEF2017

5 Future Work

There are a number of techniques that we think might help to improve bird sound classification upon our current results. Aside from better-crafted and tuned ConvNet architectures, extensive dataset augmentation and more training time, we would like to assess the following methods:

Reducing dataset distortion: A clean dataset with sharp classes is vital especially for multi label tasks like soundscape recordings. While this task could be done manually, we propose a more efficient way using neural nets trained to distinguish between noise and bird sounds. The excellent Warblr and FreeSound datasets[4] provide several tens of thousands of samples for training.

3D-Convolutions: Mapping audio signal chunks to images via FFT is very effective but does not fully account for the sequential nature of continuous signals. With the rise of 3D-Convolutions [19], we could think of sequence preserving image inputs like sequential stacks of spectrograms. Every input signal is split into chunks of 30 or more seconds, each second encoded as spectrum. All spectrograms of such a chunk form a 3D input (actually 5D: batch size, channels, stack size, width, height) which contains valuable information concerning bird sound occurrences over time. This approach will likely reduce the dataset distortion for birds with single calls in long time spans as well.

Snapshot Ensembles: Pooling the predictions of multiple CNNs is important for top scoring results in evaluation campaigns. Training ensembles is very time consuming and requires different datasets and/or network architectures. Snapshot Ensembles [20] try to reduce the amount of training time needed for an ensemble by using repeating learning rate cycles, which lead to independently converged models using the same dataset and architecture. Benchmarks show that those ensembles outperform state-of-the-art model architectures.

6 Conclusion

We provided insights into our attempt of large-scale bird sound classification using various convolutional neural networks. After we conducted numerous experiments to identify the best techniques of dataset augmentation, training methods and network architectures, our best submission to the 2017 BirdCLEF challenge achieved a score of 0,605 MAP ranking second of all submissions. The results show that there is still a lot of room for improvements especially for the soundscape domain, which likely is the most important real-world application. Additionally, we provide a GitHub repository for the free use of our code base and with that, hope to offer a baseline for future BirdCLEF tasks.

[4] http://machine-listening.eecs.qmul.ac.uk/bird-audio-detection-challenge/

Acknowledgement

The European Union and the European Social Fund for Germany partially funded this research. This work was also partially funded by the German Federal Ministry of Education and Research in the program of Entrepreneurial Regions InnoProfile-Transfer in the project group localizeIT (funding code 03IPT608X). We like to thank Matt Medler, Tom Schulenberg, and Chris Wood from the Cornell Lab of Ornithology for their kind assistance and advice.

References

1. Sprengel, E., Martin Jaggi, Y. K., & Hofmann, T. (2016). Audio based bird species identification using deep learning techniques. Working notes of CLEF.
2. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).
3. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1-9).
4. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).
5. Takahashi, N., Gygli, M., & Van Gool, L. (2017). AENet: Learning Deep Audio Features for Video Analysis. arXiv preprint arXiv:1701.00599.
6. Piczak, K. (2016). Recognizing bird species in audio recordings using deep convolutional neural networks. In Working notes of CLEF 2016 conference.
7. Joly, A., Goëau, H., Glotin, H., Spampinato, C., Bonnet, P., Vellinga, W., Lombardo, J., Planquè, R., Palazzo, S., & Müller, H. (2017). LifeCLEF 2017 Lab Overview: multimedia species identification challenges. In Proceedings of CLEF 2017.
8. Goëau, H., Glotin, H., Planquè, R., Vellinga, W., & Joly, A. (2017). LifeCLEF Bird Identification Task 2017. In CLEF working notes 2017.
9. Lasseck, M. (2013, December). Bird song classification in field recordings: winning solution for NIPS4B 2013 competition. In Proc. of int. symp. Neural Information Scaled for Bioacoustics, sabiod. org/nips4b, joint to NIPS, Nevada (pp. 176-181).
10. Smithson, S. C., Yang, G., Gross, W. J., & Meyer, B. H. (2016). Neural networks designing neural networks: multi-objective hyper-parameter optimization. arXiv preprint arXiv:1611.02120.
11. Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.
12. Clevert, D. A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289.
13. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).
14. Huang, G., Liu, Z., Weinberger, K. Q., & van der Maaten, L. (2016). Densely connected convolutional networks. arXiv preprint arXiv:1608.06993.

15. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 770-778).
16. Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
17. Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., ... & Bengio, Y. (2016). Theano: A Python framework for fast computation of mathematical expressions. arXiv preprint arXiv:1605.02688.
18. Dieleman, S., Schlüter, J., Raffel, C., Olson, E., Sønderby, S. K., Nouri, D., ... & Kelly, J. (2015). Lasagne: First release. Zenodo: Geneva, Switzerland.
19. Tran, D., Bourdev, L., Fergus, R., Torresani, L., & Paluri, M. (2015). Learning spatiotemporal features with 3d convolutional networks. In Proceedings of the IEEE International Conference on Computer Vision (pp. 4489-4497).
20. Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., & Weinberger, K. Q. (2017). Snapshot ensembles: Train 1, get m for free. arXiv preprint arXiv:1704.00109.