Designing a Lightweight Convolutional Neural Network for Bird Audio Detection

Christos Tsompos
Department of ECE
Aristotle University of Thessaloniki
Thessaloniki, Greece
ctsompos@ece.auth.gr

Vasilis F. Pavlidis
Department of ECE
Aristotle University of Thessaloniki
Thessaloniki, Greece
vpavlid@ece.auth.gr

Kostas Siozios
Department of Physics
Aristotle University of Thessaloniki
Thessaloniki, Greece
ksiop@auth.gr

2022 Panhellenic Conference on Electronics & Telecommunications (PACET) | 979-8-3503-9958-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/PACET56979.2022.9976355

***Abstract*—State-of-the-art machine learning models have been used for a plethora of Sound Event Detection (SED) applications, including bird vocalization detection, which plays a significant role in monitoring the overall ecosystem health. However, many popular models are characterized by high complexity and, therefore, are rather unsuitable for operating on IoT edge devices. To address this limitation, this paper proposes a lightweight Convolutional Neural Network with emphasis on edge computing. The proposed architecture achieves an accuracy measure of 86.42% on a bird audio detection task with only 73,377 trainable parameters.**

***Index Terms*—Convolutional Neural Network, Lightweight CNN, Machine Learning, Sound Event Detection, Bird Audio Detection**

I. Introduction

Sound event detection is a task that involves locating and classifying specific types of sound in audio data from real-life environments [9]. Examples of sound detection could be speech recognition, monitoring animal sounds or music classification. There is a wide variety of sound detection tasks, such as the simplistic binary presence (absence) problem or the more complex polyphonic segmentation where different types of sound in the same audio clip are identified. Considering, however, that diverse sounds can overlap in a real-life environment, even binary classification can be challenging. This observation is the primary motivation of this work relating to bird audio detection.

Bird audio detection is important for both scientific and environmental purposes. The last few years bioacoustics has become a Big Data research area, providing huge amounts of audio data that cannot be inspected manually, and addresses the need for this process to be fully automated [1]. Bird detection usually aims to monitor population densities, migration patterns, and provides valuable information about the health of the overall ecosystem. Additionally, examining the presence/absence of birds in the environment can be used as filtering in order to reduce the amount of data for further species identification.

Machine learning models constitute a very helpful tool in audio detection [1]. State-of-the-art models such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Support Vector Machines (SVMs), Gaussian Mixture Models (GMMs) etc., can produce very accurate results in sound detection and classification projects. Unfortunately, deploying complex Machine Learning models, especially Deep Learning models that have millions of trainable parameters, in embedded devices that operate at the edge of Internet of Things (IoT) networks is not straightforward. Low-power embedded devices are very limited in terms of storage space, RAM, and computational resources [6]. In order to exploit the benefits of IoT edge computing, including performance, energy efficiency and security, it is important to design lightweight Machine Learning (ML) models that do not require considerable resources to operate.

This paper describes the procedure of designing a low complexity CNN for bird sound binary classification. The procedure starts with the necessary data preprocessing, continues with the step-by-step selection of the optimal parameters[1] and concludes with the presentation of the final CNN architecture. The proposed CNN achieved 86.42% accuracy with less than 100 thousand trainable parameters.

The rest of the paper is organized, as follows: Section II describes succinctly previous works related to the domain of bird audio detection. Section III describes our approach to solve a bird detection problem. The proposed framework for deploying an edge-IoT platform for this task is also introduced in Section III. Experimental results that highlight the superiority of the proposed solution are given in Section IV. Finally, Section V concludes the paper.

II. Related Work

CNNs have been used to investigate the problem of animal sound classification [13]. By using a collection of 875 animal sound samples containing 10 types of animals, they obtained a maximum accuracy of 75% with Nadam optimizer. In 2016, Stowell *et al.* started a Bird Audio Detection (BAD) challenge [1]. The participants were asked to design different algorithms that predict the presence or absence of bird chirps in audio files. The main goal of the challenge was to achieve an Area Under Curve (AUC) measure as high as possible on a hidden

[1]Network parameters include number of layers, kernels etc. Trainable parameters are the weights.

dataset. The provided training data came from freefield1010 [14] and Warblr [15] (development set), while the unseen testing dataset came mostly from the TREE research project[2].

The contestants presented a wide variety of Machine Learning Algorithms. Recently Thomas Grill and Jan Schlüter presented two independent feedforward CNNs applied to mel spectrograms, named *bulbul* and *sparrow*, achieving 88.7% AUC and 83.7% AUC, respectively [2]. Cakir *et al.* combined convolutional layers with recurrent layers to create a CRNN applied to mel band energies, thereby obtaining an AUC measure of 88.5% [3]. Thomas Pellegrini proposed a DenseNet architecture and managed to achieve 88.22% AUC score on the unknown data [17]. Additionally, the DenseNet obtained an accuracy of $\sim$90% on the validation dataset. Another novel approach was based on Archetypal Analysis [4]. The classification was performed using SVMs with a dynamic kernel and a variant of probabilistic sequence kernel, achieving an accuracy of 85.2% on the development dataset. Thakur *et al.* also worked with SVMs with a dynamic kernel and demonstrated a procedure that leads to a 70% speedup for the proposed method with a very small drop in accuracy [5].

III. The Proposed Framework

Inspired by the BAD challenge, we propose a new bird detection model with emphasis on edge devices. Thus, the introduced machine learning model aims at low complexity and storage requirements. However, as we highlight at the experimental results section, the impact of these architectural selections is negligible (about 4–5%) as compared to the most accurate prediction models. Since the *bulbul* architecture was the winning submission, we choose to work with CNNs applied to mel spectrograms. However, our approach differs in the sense that in contrast to the challenge demands, we do not aim to construct an as accurate as possible CNN. The objective is to create a lightweight CNN that has a small number of layers and trainable parameters tailored to real-time inference on embedded devices characterized by limited computational resources while obtaining a prediction accuracy above 85%. Indeed, the important trade-offs among accuracy, power, and computing and storage resources for this problem have not been explored. In this paper, therefore, we present a structured step by step methodology with the aim of designing a CNN architecture from the ground up that highlights these trade-offs.

A. Proposed Decision-Making Algorithm

Each audio clip is sampled with a sample rate of 22.05 KHz and Short-Time Fourier Transform (STFT) with a window size of 2,048 samples and a hop size of 512 samples (75% overlapping) is applied, generating frames that last 93 msec. Subsequently, frequencies are converted onto the mel scale and finally, we apply 128 mel scaled filter banks to capture the energy of 128 frequency bands. This way, 431×128 shaped mel spectrograms are extracted. Mel spectrograms are image representations of sound, where every pixel contains the value of sound intensity (in dB) at a specific mel frequency band. Such a spectrogram is illustrated in Figure 1. Since bird call intensity can vary on different sound samples, it is important to normalize the dataset such that the intensity values are within the same range.

[2]https://wiki.ceh.ac.uk/display/NRT/NERC+RATE+TREE+Home

Each convolutional layer consists of a number of kernels. A kernel is a matrix of weights that scans the input data (in our case images) in order to extract certain features through convolution. When all the kernels move over the data, they forward a stack of images to the next layer. Since the desired architecture is a lightweight CNN, we initially choose a relatively small number of 32 kernels per layer with a size of 3×3. The stride value describes the number of rows or columns the kernels move by while scanning the data. We select a stride value of 2 for the purpose of scanning the image faster and reducing train and inference time.

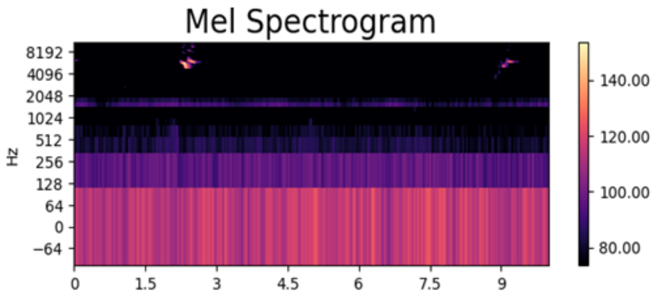

Fig. 1. Mel spectrogram showing bird chirps at 2s and 9s.

Additionally, the size of the image decreases by $\sim$75% after each layer, leading to a smaller number of trainable weights where the output of the last convolutional layer is connected to a dense layer used for classification. Each convolutional layer uses a ReLU activation function, while the output layer uses a Sigmoid function for classification. The Sigmoid function output has a value range between 0 and 1 and can be considered as the probability of bird presence in a data instance. During training, the weights are adjusted by a function or algorithm called optimizer. The optimizer modifies the weights and determines the optimal values that minimize a loss function. A loss function computes the error between the expected value and the output model value. Adam optimizer is a good solution, suitable for a variety of machine learning models and converges very quickly [7]. The initial learning rate is 0.001. The learning rate determines how quickly the CNN weights adjust. As a loss function, we pick the binary cross-entropy, which is the standard function for binary classification. Although adding more layers usually improves accuracy, it has a negative impact on training and inference time due to increasing number of trainable parameters and computational cost. Therefore, we limit our solution space to network architectures with a maximum of 5 convolutional layers. We also add a fully connected layer with 128 neurons to enhance classification accuracy. Afterwards, we resize the images and test our CNNs in order to explore how the input shape affects the aforementioned tradeoffs for models with a different number of convolutional layers.

A common issue with machine learning models is overfitting, meaning that the model memorizes the training data. As a result, the model is not able to generalize well on new examples, leading to deteriorating accuracy. Regularization techniques address the issue of overtraining by reducing the generalization error. Dropout is a widely used technique that temporarily removes units from the network, along with all its connections [8]. The choice of the eliminated units is random and depends on a fixed probability p. We choose a value of 0.5, since it is close to optimal for a variety of tasks [8].

Convolutional layers are frequently preceded by pooling layers. These layers downsample the map features with the use of a filter while extracting the most valuable information and eliminating irrelevant features that can be considered as noise. By reducing the spatial dimension of the input, it lowers the number of trainable parameters and minimizes computational cost. At the same time, it improves the CNN's performance by preventing overfitting [10]. There is a wide variety of popular methods e.g., max and average pooling. We choose max pooling, since it is extensively used in CNNs and is more effective on simple classifiers. The starting filter size is 2×2 and scans the data with a stride value of 2.

The next step of designing the algorithm is to try out using a larger kernel size for some layers in order to examine if large-scale features contain useful information. We have to take into consideration that using 5×5 kernels will increase the number of the network's trainable parameters. Another kernel adjustment would be changing the number of kernels per layer. For instance, after consecutive downsampling of the input data, the image size is much smaller and, hence, it may be preferable to decrease the number of kernels used in the last convolutional layer to avoid irrelevant information. Those modifications could be a part of a fine tuning process. Keras [11] and Tensorflow [12] offer tuning algorithms that select the optimal values for a wide variety of network parameters, such as different optimizers and learning rates, training and testing batch sizes, stride values or dropout probabilities p. In case the parameters we want to adjust are too many to be examined manually, fine tuning algorithms speed up the process of ironing out the last details to achieve the best accuracy performance. However, we have to ensure that these changes do not have a negative effect on the desired tradeoffs.

For the last step of the design process, we use 5-fold cross validation. Through this procedure, 20% of the data is held for later testing. The rest of the dataset is split randomly into 5 equally sized folds. We train our CNN five times, using 1 fold as the validation set and the remaining 4 folds as the training set. This way, we can choose the model that demonstrates the highest accuracy as the final CNN to be deployed for inference on an embedded device. In Figure 2, the entire procedure of designing the CNN can be seen in the form of a flowchart.

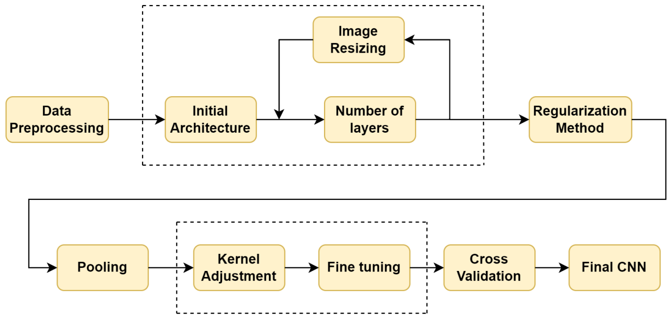

Fig. 2. The proposed method in the form of a flowchart.

IV. Experimental Results and Discussion

The dataset comes from merging the freefield10 [14] and Warblr [15] and consists of 15,690 audio files, most of which are 10 seconds long, except from a minority that slightly differs in duration. Raw audio data, however, is not suitable for CNNs. Hence, some preprocessing of the data is required, where we recreate the dataset using a fixed length of 10 sec in order to extract mel spectrograms. Before feeding the data into the CNN for training, we normalize the data using scaling and change the range of the distribution of pixel values to (0,1). An alternative solution would be to apply Batch Normalization [16] on the convolutional layers. We avoid this approach due to additional computational cost. After preprocessing, we split the data into training (80%) and testing (20%) sets.

The initial series of experiments explores the effect of adding convolutional layers to the network. We train 5 different CNNs with an ascending number of convolutional layers for 30 epochs, keeping 20% of the training set for validation. After each training epoch, we store the value of the loss function in order to detect how many epochs have passed until the error is minimum and retrain the model for the same amount of epochs. By decreasing the number of training epochs, overfitting is reduced.

The results reported in Table I show that more layers do not necessarily lead to improved performance. Convolution with stride value of 2 decreases the input size rapidly. As a result, after adding a certain number of layers, the image no longer contains valuable information for feature extraction, leading to a significant drop in accuracy. However, having a small output size at the end of the convolution implies fewer connections with the dense layers used for classification. Furthermore, due to a larger amount of computations, the training time increases along with the number of layers. The networks comprising less than 3 convolutional layers contain millions of trainable parameters. Due to the limited storage space, they are considered unsuitable for deployment on edge devices. The CNN that has 4 convolutional layers is considered acceptable, since it has fewer than 200K trainable parameters. However, we keep exploring the solution space to see if we can further improve our architecture.

In order to decide upon the optimal number of layers for our network, we repeat the experiments for different input

TABLE I
THE EFFECT OF LAYER NUMBER ON CNNS

Conv. Layers	Accuracy	Training time (sec)	Trainable parameters
1	77.50%	1841.56	13,587,009
2	79.22%	2295.72	3,266,145
3	79.92%	2521.46	764,545
4	81.99%	2713.91	188,065
5	79.73%	2767.64	62,145

sizes. We change the number of mel frequency bands of the spectrogram from 128 to 80. Since we move towards reducing the image size, we eliminate the possibility of using more than 4 convolutional layers. The CNNs demonstrate significant improvements with an input shape of 431×80. Apart from the fact that the number of trainable parameters and the training time are reduced, every CNN exhibits an increase in accuracy by ∼2-2.5%. The networks with 3 and 4 convolutional layers yield an accuracy of 81.45% and 84.67%, respectively. The former consists of 445,057 trainable parameters and requires ∼1580 sec for training and the latter has 134,817 trainable parameters and completes training in ∼1601 sec. The amount of trainable weights for the models with 1 and 2 convolutional layers remains prohibitively high to fulfill our requirements. Conducting the same experiments with 431×64 and 216×80 shaped images results in lower accuracy such that we conclude that the optimal combination is a CNN with 4 convolutional layers (CNN-4) and an input size of 431×80. Nevertheless, we also test CNNs with 3 convolutional layers (CNN-3), in case we can further reduce the weight number and improve accuracy.

After limiting the space solution, we use regularization to reduce overfitting. We mentioned before that we used optimal epoch selection and retrained the networks for the same reason. However, due to the stochastic nature of CNNs, the epoch selected as optimal can differ during retraining and, hence, it may not always prevent overfitting. Using regularization solves this problem because the loss function value changes more slowly and reduces the overfitting probability. By applying dropout with 0.5 probability on every convolutional layer, the CNN-3 performs better and achieves an accuracy of 83.46%, but requires 4668 seconds to train, meaning that the network becomes 3 times slower. The accuracy of CNN-4 remains approximately the same and its training time triples as well (5246 sec). At a later stage of the designing process, we realize that applying dropout only on the last convolutional layer and the fully connected layer has a negligible impact on the training time and helps with overfitting prevention.

In order to further improve the CNN accuracy, we apply max pooling after each convolutional layer. Since pooling filters scan the images similarly to the convolution kernels, using a stride value of 2 for pooling layers as well would decrease the image size significantly, leading to information loss. To exploit the advantages of max pooling, we rearrange the kernel strides. The key is to modify the network layers in such a way that the output size of the last convolutional layer minimizes connections with the dense layers used for classification, without causing a drop in accuracy. Table II shows the enhancement in performance after applying max pooling layers and dropout regularization with a probability of 0.5 to the networks.

TABLE II
RESULTS AFTER APPLYING MAX POOLING AND DROPOUT.

Conv. Layers	Accuracy	Training time (sec)	Trainable parameters
3	85.95%	1712.52	68,225
4	86.42%	2042.82	73,377

As we can see, the two models perform very closely. Since the difference in trainable parameters and training time is insignificant, we choose CNN-4 as our final CNN. Inference time is measured to be 4 sec for the classification of 3,138 images.

Before concluding our designing procedure, we perform manual fine tuning. Replacing the 3×3 kernels of the first convolutional layer with 5×5 kernels does not improve accuracy. Furthermore, we retrain the model with different learning rate values. Using a learning rate of 0.01 leads to a ∼3% drop in accuracy, while a value of 0.0001 results in 85% accuracy and increased training time. Thus, the initial learning rate turns out to be optimal. The final CNN architecture is depicted in Figure 3.

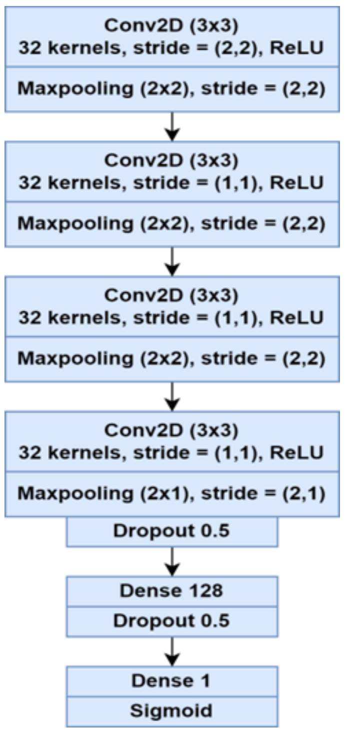
Conv2D (3x3)
32 kernels, stride = (2,2), ReLU
Maxpooling (2x2), stride = (2,2)
Conv2D (3x3)
32 kernels, stride = (1,1), ReLU
Maxpooling (2x2), stride = (2,2)
Conv2D (3x3)
32 kernels, stride = (1,1), ReLU
Maxpooling (2x2), stride = (2,2)
Conv2D (3x3)
32 kernels, stride = (1,1), ReLU
Maxpooling (2x1), stride = (2,1)
Dropout 0.5
Dense 128
Dropout 0.5
Dense 1
Sigmoid

Fig. 3. The proposed CNN architecture.

Finally, we apply 5-fold cross validation on the proposed CNN to observe the effect different data partitions have on the model's performance. The CNN is trained for 5 different combinations of training and validation data. The evaluation results on the testing set are listed in Table III.

TABLE III
RESULTS EVALUATION WITH CROSS VALIDATION.

	Fold 1	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**
Accuracy	85.91%	86.17%	85.56%	86.20%	85.66%

It is clear that the way the dataset is split plays a significant role in accuracy performance. Since the audio data is collected from real life environments, it is possible that there are significant dissimilarities among data instances due to vocalizations of various species, background noise or weather conditions. Thus, different features are extracted from separate data samples, leading to various results in accuracy measurements. It is important to mention that even retraining a machine learning model with the exact same data can lead to different results, due to random initialization of the trainable weights. Nevertheless, we have not explored optimal data partition and weight initialization where these tasks are left as future work. The procedure we follow turns out to be very efficient, since it leads to a low complexity architecture that achieves an accuracy measure above 86%.

Before concluding the paper we compare our final model to the DenseNet architecture proposed by Thomas Pellegrini [17], which ranked 3rd among 30 Bird Audio Detection challenge participants. The DenseNet consists of 74 layers with a total of 328K parameters and leads to an accuracy measure of $\sim$90% on the validation dataset. Our CNN shows a $\sim$3.5% drop in accuracy measured on the same dataset by using only 22.3% of the DenseNet parameters and comprises 5 layers in total, leading to much lower computational complexity. Unfortunately, we do not have access to the hidden test dataset used in the Bird Audio Detection challenge and, hence, we are unable to do a proper comparison with the other participants' work.

V. CONCLUSIONS

This paper presents a lightweight CNN architecture for bird audio detection using features that are extracted from mel spectrograms. The primary focus of this research is to design a low complexity network that is suitable for deployment on IoT edge devices. The limited number of layers and kernels results in low computational cost. Moreover, max pooling layers with a stride value of 2 decrease the input size while preserving important features and, hence, thereby reducing the number of trainable parameters without causing a drop in accuracy. The proposed CNN achieved up to 86.42% accuracy score with only 73,377 trainable parameters.

ACKNOWLEDGMENT

This research has been co-financed by the European Regional Development Fund of the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE (project code: T2EDK-01681).

REFERENCES

[1] D. Stowell, M. Wood, Y. Stylianou, and H. Glotin, "Bird detection in audio: a survey and a challenge," IEEE Int. Workshop Mach. Learn. Sig. Process., 2016, pp. 1-–6.

[2] T. Grill and J. Schlüter, "Two convolutional neural networks for bird detection in audio signals," European Signal Processing Conf., 2017, pp. 1764–1768.

[3] E. Cakir, S. Adavanne, G. Parascandolo, K. Drossos and T. Virtanen, "Convolutional recurrent neural networks for bird audio detection," European Signal Processing Conf., 2017, pp. 1744–1748.

[4] V. Abrol, P. Sharma, A. Thakur, P. Rajan, A. D. Dileep and A. K. Sao, "Archetypal analysis based sparse convex sequence kernel for bird activity detection," European Signal Processing Conf., 2017, pp. 1774–1778.

[5] A. Thakur, R. Jyothi, P. Rajan and A. D. Dileep, "Rapid bird activity detection using probabilistic sequence kernels," European Signal Processing Conf., 2017, pp. 1754–1758.

[6] F. Tsimpourlas, L. Papadopoulos, A. Bartsokas and D. Soudris, "A Design Space Exploration Framework for Convolutional Neural Networks Implemented on Edge Devices," in IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, pp. 2212–2221, Nov. 2018.

[7] Diederik P. Kingma, Jimmy Lei Ba "Adam: A Method for Stochastic Optimization," in Int. Conf. for Learning Representations, 2015.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", in Journal of Machine Learning Research, Vol. 15, pp. 1929–1958, 2014.

[9] X. Xia, R. Togneri, F. Sohel, Y. Zhao and D. D. Huang, "Sound Event Detection Using Multiple Optimized Kernels," IEEE/ACM Trans. on Audio, Speech, and Language Processing, vol. 28, pp. 1745–1754, 2020.

[10] A. Zafar, M. Aamir, N. Mohd Nawi, A. Arshad, S. Riaz, A. Alruban, A. Dutta, S. Almotairi, "A Comparison of Pooling Methods for Convolutional Neural Networks", Applied Sciences 2022, Vol. 12, No. 17, pp. 8643.

[11] Chollet, Franois. (2015) Keras: Deep learning library for theano and tensorflow [Online]. Available: https://keras.io

[12] M. Abadi, A. Agarwal, P. Barham *et al.*, "Tensorflow: Large scale machine learning on heterogeneous distributed systems," arXiv preprint arXiv:1603.04467.

[13] E. Şaşmaz and F. B. Tek, "Animal Sound Classification Using A Convolutional Neural Network," Int. Conf. on Computer Science and Engineering, 2018, pp. 625–629.

[14] D. Stowell, and M. Plumbley, "An open dataset for research on audio field recording archives: freefield1010", (https://arxiv.org/abs/1309.5275)

[15] [Online]. Available: http://warblr.net (accessed on 21/09/22).

[16] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", Proc. of Int. Conf. on Machine Learning, Vol. 37, pp. 448—456, July 2015.

[17] T. Pellegrini, "Densely connected CNNs for bird audio detection," European Signal Proc. Conf., 2017, pp. 1734–1738.