Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Østen Finnes Holkestad

A deep learning based approach to detect the common spadefoot toad

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Guillaume Dutilleux
June 2021

NTNU
Norwegian University of
Science and Technology

Østen Finnes Holkestad

A deep learning based approach to detect the common spadefoot toad

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Guillaume Dutilleux
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU

Kunnskap for en bedre verden

Abstract

The EU's Habitats Directive states that the common spadefoot toad (*Pelobates fuscus fuscus (P. fuscus))* is in need of strict protection, being placed in the category known as Annex IV species (European Council 1992) (European Council 2006). This category states that species must be placed under a strict protection regime (European Council 2021). Therefore non-invasive methods such as acoustic monitoring could be a possible way to keep count of species in its habitats.

Deep learning is presented as a possible way to develop a detector of the common spadefoot toad based on its advertisement calls. The data that is used to train the detector was collected over two years at two sites (Dutilleux and Curé 2020). The two sites act as breeding grounds for the spadefoot toad as well as a few other anuran species and the data was as such collected during the breeding periods. The data is transformed to spectrogram and is used as input to a neural network. Using the convolutional neural network architecture EfficientNet (introduced in (Tan and Le 2020)) the detector developed in this paper achieves a precision of 93.47% at the expense of getting a true positive rate of 67.28%. In addition to this, the detector achieves a false positive rate of 0.55%. The low false positive rate comes from the way the post-processing is done when testing the model. A comparison with the software detector developed in (Dutilleux and Curé 2020) is also done. The software detector achieved true positive rates ranging between 53% and 73% and a false positive rate of 1.5%.

A discussion is made on the detection of the common spadefoot toad, and examples of true and false positives as well as false negatives are presented. The way ground truth times are extracted from label files is found to not give an exact insight into how the model actually performs, and possible ways to avoid this in the future is presented.

In addition to a detector used on the advertisement call of the adult specimens, the report also aims to find if the juvenile specimens vocalize underwater. It has been found earlier that the juvenile vocalizes in its terrestrial phase (Hagen et al. 2016). Using this knowledge, a study of whether a deep learning model can find if it vocalizes underwater or not is performed. A concrete conclusion on whether the juvenile vocalizes underwater or not is not made, as the model only predicted sounds that were found to be false positives. The reason for this can come from a few different reasons which are discussed in greater detail. Juvenile vocalizations underwater should therefore be studied in the future.

This report shows that it is possible to get reliable results both with regards to false positive and false negative predictions when using a deep learning-based detector on the advertisement call of the common spadefoot. Therefore the detector can be applied to long-term recordings in habitats for conservation purposes.

Samandrag

EU sitt Habitat-direktiv seier at løkfrosken (eng. common spadefoot toad, lat. *Pelobates fuscus fuscus (P. fuscus)*) treng streng beskytting, og er plassert i direktivet sitt Anneks IV (European Council 1992) (European Council 2006). Denne kategorien seier nettopp at arten må plasserast under eit strengt program for å beskytte den (European Council 2021). Derfor kan ikkje-innvaderande metoder som akustisk monitorering vere ein mogleg måte å overvake arten i sitt habitat.

Djup læring (eng. deep learning) vert brukt som ein mogleg måte å utvikle ein detektor av løkfrosken (basert på paringsrop til arten). Dataen som er brukt til å trene detektoren er samla inn over to år på to forskjellige område (Dutilleux og Curé 2020). Dei to områda er paringsområde for arten i tillegg til nokre andre anura, så all data var teken opp i heile paringsperioden. Dataen vart så transformert til spektrogram og brukt som inn-data til eit nevralt nettverk. Ved å bruke det konvolusjonelle nevrale nettverket EfficientNet (først introdusert i (Tan og Le 2020)) klarte detektoren å oppnå ein presisjon (eng. precision) på 93.47% på kostnad av ein sann positiv-rate (eng. true positive rate) på 67.28%. I tillegg til dette oppnår detektoren ein falsk positiv-rate (eng. false positive rate) på 0.55%. Den låge falske positiv-raten er eit resultat av post-prosesseringa som vert gjort under testinga av modellen. Ei samanlikning av detektoren utvikla i denne masteroppgåva og software-detektoren utvikla i (Dutilleux og Curé 2020) vert også gjort. Software-detektoren oppnår sanne positiv-rater på mellom 53% og 73% og ein falsk positiv rate på 1.5%.

Ein diskusjon vert gjort på deteksjonen av løkfrosken, og eksempel på sanne positive, falske positive og falske negative prediksjonar vert presentert. Måten dei faktiske tidene (eng. ground truth) frosken lagar lydar vert henta ut er funnen til å ikkje gi eit heilt eksakt innblikk i korleis modellen faktisk presterer, og moglege måtar å unngå dette i framtida vert presentert.

I tillegg til ein detektor som kan brukast på paringsropet til vaksne individ, vil rapporten også sjå om ungdomsindivid av frosken kan verte funnen under vatn. Det har vorte funne tidligare at desse individa vokaliserer i si landbaserte fase av livet (Hagen mfl. 2016). På bakgrunn av denne kunnskapen vert det gjort ei studie på om ein djup læring-modell kan finne ut om den vokaliserer under vatn. Det vert ikkje gjort ein konkret konklusjon på om ungdomsindivid vokaliserer under vatn eller ikkje då modellen berre predikerer falske positive. Grunnen til dette kan kome frå nokre grunnar som vert diskutert i rapporten. Vokaliseringane til ungdomsindivida burde derfor undersøkast nærare i framtida.

Denne rapporten viser at det er mogleg å få pålitelege resultat både med tanke på falske positive og falske negative prediksjonar ved å bruke ein djup læring-basert detektor på paringsropet til løkfrosken. Derfor vil denne detektoren kunne brukast på langtidsopptak i forskjellige habitat for vern av arten.

Preface

This report is my master thesis written at the Department of Electronic Systems (IES) at NTNU, Trondheim. It is the final work in the 5 year study programme Electronics Systems Design and Innovation (MTELSYS). It has been a rewarding experience working on this and I have learned a lot that I believe I will have use for later in my career as an electronics engineer.

I would like to thank my supervisor Guillaume Dutilleux for letting me choose this thesis. Through good discussions he has given me great insight into the field that is bioacoustics and I have learned a lot about bioacoustic monitoring. I would also like to thank some of my fellow co-students for rewarding discussions regarding the technical solutions used in this report and for giving a greater understanding of the machine learning field.

Contents

1 Introduction

1.1 Bioacoustic monitoring

The International Union for Conservation of Nature (IUCN) Red List is a list of all the endangered species in the world (IUCN 2021), currently containing more than 37400 (known) species. They list among others that 41% of all known anuran speices and 26% of all known mammals are endangered. Therefore monitoring species could be used in gaining knowledge on an endangered species and detect possible unexpected changes in an environment (Jones et al. 2013, p. 330). Jones and colleagues also points out that monitoring is costly. Recently-available solutions like the AudioMoth (Hill et al. 2019) could help in making bioacoustic monitoring available for everyone (Welz 2019). The AudioMoth has for example been used in research to develop detection algorithms (Prince et al. 2019) and to monitor specific species such as the New Forest Cicada native to the UK (Rogers and Zilli 2021).

Monitoring anurans acoustically by their advertisement calls could be an efficient way to gain insight into how the breeding output is affected by environmental change (Teixeira, Maron and Rensburg 2019). Setting out recording equipment at breeding sites of anurans (or any other taxa) allows for more long-time monitoring as human observers can not be present at these site indefinitely (Höbel 2017). Recording long-time audio will in turn reduce the probability of missing the presence of the species, as human observers are there only for a short time. The process of recording for longer periods is also of little to no disturbance to the species (Bridges and Dorcas 2000).

1.2 Earlier reasearch on classification of bioacoustic data

A vocalization identification tool (RIBBIT) based on the periodic structure found in anuran calls was developed by Lapp and colleagues (Lapp et al. 2021). They applied this tool on vocalizations made by the boreal chorus frog (*Pseudacris maculata*) and the harlequin frog (*Atelopus varius*). The tool achieved a precision (true positives over the sum of true and false positives) of 90% on the boreal chorus frog for a given threshold of what they call RIBBIT score. The trade-off in this case was that a precision of 90% means getting a recall of 56% meaning the number of false negatives is almost the same as number of true positives. (Noda, Travieso and Sánchez-Rodríguez 2015) used a combination of Mel and Linear Frequency Cepstral Coefficients (MFCC & LFCC) with Support Vector Machines, Hidden Markov Models and random forests to achieve classification rate of $95.38\% \pm 5.05$ on some anuran species. (Strout et al. 2017) used a spectrogram as input to a convolutional neural network. The network acts as feature extraction, and these features are then fed into a Support Vector Machine, achieving a mean classification accuracy of 73.57%. (Huzaifah 2017) tried different time-frequency representations of sound events in the UrbanSound8k dataset and found that Mel-STFT spectrograms performed generally better than linear-STFT ones. (Alonso et al. 2017) developed an automatic classification system for 17 different anuran species based on MFCC and a Gaussian Mixture Model, resulting in accuracies between 96.1% and 100% for the species. No false positive rates were presented in these papers.

1.3 Earlier research on the common spadefoot toad

The EU's Habitats Directive states that the common spadefoot toad (*Pelobates fuscus fuscus (P. fuscus)*) is in need of strict protection, being placed in the category known as Annex IV species (European Council 1992) (European Council 2006). This category states that species must be placed under a strict protection regime (European Council 2021). Therefore non-invasive methods such as acoustic monitoring could be a possible way to keep count of species in their habitats. Nyström and colleagues noted that the toad is in decline in some of its habitats and performed a classification of 72 ponds in southern Sweden, thereof 33 inhabiting the species (Nyström et al. 2002). The ponds where the P. fuscus was detected were found to share a few characteristics in that they were large, permanent (i.e. does not dry out), eutrophic with high concentrations of oxygen and having high spring temperatures.

In (Rannap et al. 2015) some Northern European habitats of the P. fuscus are discussed. 407 waterbodies in the Netherlands, Denmark and Estonia were examined by measuring 23 different habitat characteristics. It was discussed that intensive agriculture in the former two countries were one of the main threats. Whereas in the latter an overgrowing of open habitats and small freshwater bodies have negatively affected the species. The Danish consulting company AmphiConsult successfully secured some of the Northern habitats of the P. fuscus in Denmark and Estonia with its DRAGONLIFE project (Amphiconsult 2015). The project restored the Annex IV status of the toad in Estonia to *favorable*.

Ten Hagen and colleagues recorded and researched juvenile specimens of the toad and found that these individuals vocalize before sexual maturity in their terrestrial phase (Hagen et al. 2016). They found that the juvenile calls could be split into three distinct groups, and discussed that juvenile vocalizations is a natural trait for P. fuscus. (Dutilleux and Curé 2020) developed a software detector for the adult common spadefoot toad. The detector uses peak detection on a pre-processed signal in the time domain, achieving true positive rates ranging from 53% to 73% and a lower than 1.5% false positive rate.

1.4 Aim of the thesis

As shown in Sections 1.2 and 1.3 the manual monitoring of habitats of the P. fuscus happens over several years, and only needing to deploy recording equipment and automatically analyzing the data could therefore be of great help in monitoring species. As the P. fuscus is also threatened in some of its habitats, monitoring this species is of utmost importance. The aim of this thesis is therefore to develop a deep learning-based detector of the toad so as to help in automating the process.

It would therefore be interesting to see if deep learning could improve on the results achieved in (Dutilleux and Curé 2020), where more traditional signal processing methods were used to detect the toad. A detector like the one presented in this paper could then be applied to recordings from different sites in the species' entire habitat for conservation purposes. In addition to developing a deep learning-based model for detecting adult individuals, the report will also present results from a deep learning-based model for classifying several classes. An exploration to see if such a classifier could be used to detect juvenile individuals of the spadefoot toad will be presented. Due to the fact that the juvenile has been found to vocalize during its terrestrial phase (Hagen et al. 2016), an additional study will be made to see if deep learning-based methods could be used to find potential underwater vocalizations.

NOTE: As some of the theory is similar to the one used in the author's own (unpublished) specialization project report, any reuse of the text from that report will be colour-coded in blue and will look like this sample text.

2 Background

The necessary background to understand the results is given in this Section. First a look into some of the bioacoustic background is given in Section 2.1 followed by the background needed for the deep learning in Section 2.2.

2.1 Bioacoustics

Performing passive acoustic monitoring of shallow ponds introduces a few challenges; not all biological life vocalize, high frequencies are attenuated quickly in water, sound propagation is complex in shallow water and very little is known of particular sounds produced by individual species at these sites (Linke et al. 2018). Recently however, monitoring species by passive acoustic monitoring has taken off following the growing trend of automated data collection and big data (Sugai et al. 2018).

Masking occurs when noise or other sounds interferes with an animal's ability to produce or perceive sounds. As other acoustic sources gets louder it will become more difficult for animals to perceive conspecifics. The degree of masking is dependent on sound level, frequency band and duration of the sound. It has the greatest impact to species when in the same frequency band as important communication signals, like the advertisement call. Some species can either increase the intensity of the call, while others increase/decrease frequency or some even stops vocalizing which is a great threat to smaller populations (Discovery of Sound in the Sea (DOSITS) 2020). For example, in ponds where P. fuscus is present, sounds like rain or other vocalizing species were found to be some sources of masking. The common spadefoot toad is special in that it does not vocalize in chorus, and that the vocalization is not broadband (energy mainly being in the range 700Hz-1200Hz) (AmphibiaWeb 2020; Dutilleux and Curé 2020).

2.1.1 The different call types of the P. fuscus

The most common of the call types of the spadefoot toad is its advertisement call. An example of a stereotypical advertisement call of the common spadefoot toad is shown in Figure 1. As seen the call usually consists of two or three (sometimes even more) distinct "notes" of pulsed vocalizations. The notes can be seen at 0.2 seconds and 0.4 seconds.

The mean length of the advertisement call of the *Pelobates fuscus insubricus* has been reported to be 0.506 s (Seglie, Gauna and Giacoma 2013, p. 61). Depending on the water temperature the average length of the advertisement call of the *Pelobates fuscus fuscus* is anywhere between 368.10 ms (24°C) and 881.46 ms (4°C) (Müller 1984, p. 128). (Schneider 1966, p. 124) found that the call had a length between 310 ms and 336 ms (with a water temperature of 15°C), with a mean of 318 ms.

The juvenile individuals of the common spadefoot toad have also been found to vocalize in their terrestrial phase (Hagen et al. 2016). Three distinct call types of different durations and frequencies were found when analyzing the juvenile and they were named call type, S, E and P. Call type S were found to be somewhat similar to the adult advertisement call, as it contains two-three separate notes. However the notes of this call type were shorter and not clearly pulsed. Examples of the call types on individuals form the Fürstenkuhle Nature Reserve in Germany can be found in (Hagen et al. 2016, p. 4 of 8).

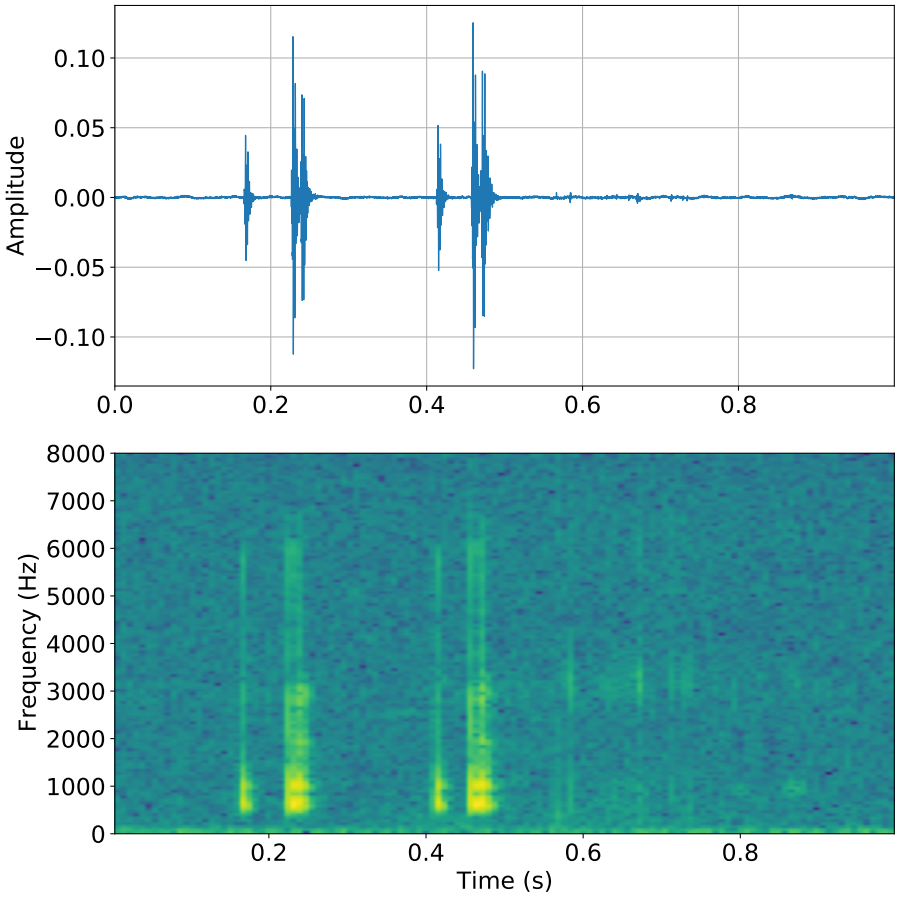


Figure 1: Waveform and spectrogram of a stereotypical P. fuscus advertisement call. Time on x-axis, amplitude on y-axis of waveform and frequency in Hz on spectrogram.

2.2 Deep learning

Most of the theory about deep learning and neural networks is taken from the books *Deep Learning book* by Ian Goodfellow and colleagues (Goodfellow, Bengio and Courville 2016) as well as *Dive into Deep learning* by Aston Zhang and colleagues (Zhang et al. 2020) unless otherwise stated. Both books are available for free online.

2.2.1 Deep learning and neural networks

Feedforward neural networks are the quintessential deep learning models (Goodfellow, Bengio and Courville 2016, p. 164). The goal of these types of networks is to map a function $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$ of some input $\mathbf{x}$ to an output $\mathbf{y}$. Assuming that a set of parameters is named $\boldsymbol{\theta}$ then the feedforward network will learn the parameter set that results in the best function approximation. The process of learning and updating these parameters is called training. A complete iteration over the complete set of input (dataset) used for training is called an epoch (Zhang et al. 2020, ch. 3.2.7).

When connecting several of these function consecutively the resulting network is said to be deeper. If a model learns features in the data and can perform well on previously unobserved data, it is said to have an ability to generalize. This ability can be measured with a cost function, which is calculated from a test set collected separately from the training dataset. This is a function chosen by the user, as it is dependent on use-case.

A fully-connected feedforward neural networks consists of several nodes (or neurons) in a structure as the simple one in Figure 2. This network in particular contains an input layer, two hidden layers (layers "hidden" form the user during training) and an output layer. A vector $\mathbf{x} = [x_1, x_2]$ is fed into the network. The "node" containing a single data point is connected to all nodes in the next layer with connections holding weights.

For example the first node in the first hidden layer h_{11} is connected to both inputs of the previous layer. This node input can then be calculated as the linear combination of the weights connected to the node and the output from the previous layer. Assuming the connections (weights) to this node can be written as w_1 and w_2 (shown in Figure 2), then Equation (1) shows the linear combination of the input and weights. Such a linear combination is then calculated all the way to the output $\mathbf{y} = [y_1, y_2]$.

$$h_{11} = x_1 \cdot w_1 + x_2 \cdot w_2 \tag{1}$$

Classification tasks are usually associated with supervised learning (Mohri, Rostamizadeh and Talwalkar 2018, p. 6). What this means is that a network is given training examples with already defined labels, learns the mapping between input and output and makes predictions on unseen data. The predictions are then compared to the true labels and a measure of (un)certainty is then calculated. More on this measure can be found in Section 2.2.5.

A binary classification problem is a problem where the desired output is either 1 or 0. The actual output of a model designed to predict binary classes is then seen as a probability, meaning the closer the output is to 1, the more confident the model is that the given class is present (Amazon 2021). A threshold score is chosen to find all predictions that are above or below it. All predictions with output higher than the threshold is returned as a detection while all predictions below it is returned as a non-detection.

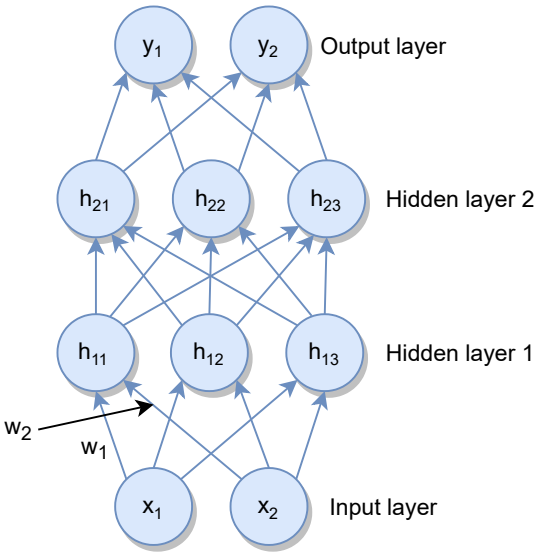


Figure 2: A simple structure of a fully-connected network containing an input and output layer and two hidden layers. The input x_1 and x_2 is fed through the network layers. Each layer is a linear combination of weights and inputs of the previous layer.

It is also possible to perform multi-label classification. This is used when a model should give multiple predictions at the same time (Brownlee 2020). For example if a model should predict the common spadefoot toad at the same time as a bird sings, it would be possible to map that specific input with an output that corresponds to the P. fuscus and bird simultaneously. A bit more on the implementation of this is found in Section 3.2.3.

Training a model bears the risk of over- and underfitting. Goodfellow defines underfitting as the model not being able to obtain a sufficiently low error value on the training set. Overfitting occurs when there is a too large gap between the training and testing errors (Goodfellow, Bengio and Courville 2016, p. 109-110). In other words the model is overfit when it learns the input-to-output mapping and is not able to generalize for new input.

Hyper-parameters are parameters that configure a model and their values can not be found during training of the model. These kinds of parameters can for example be set by inferring from earlier similar problems, performing a search for the best values or by trial and error. Examples of hyper-parameters that the user can control is the number of epochs a model is trained for, learning rates in optimization algorithms or the number of layers/nodes in a layer in a neural network (Brownlee 2017).

Brian Ripley defines training data, validation data and test data in the following way. Training data is defined as “a set of examples used for learning, that is to fit the parameters of the classifier”, validation data as “a set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network” and test data as “a set of examples used only to assess the performance of a fully-specified classifier” (Ripley 1996, p. 354). The process of making a model is in other words split into three separate steps; training, validation and testing.

2.2.2 Convolutional Neural Networks

As the name suggests, the main operation performed in Convolutional Neural Networks (CNNs) is convolution. Generally the convolutional part’s goal in a neural network is

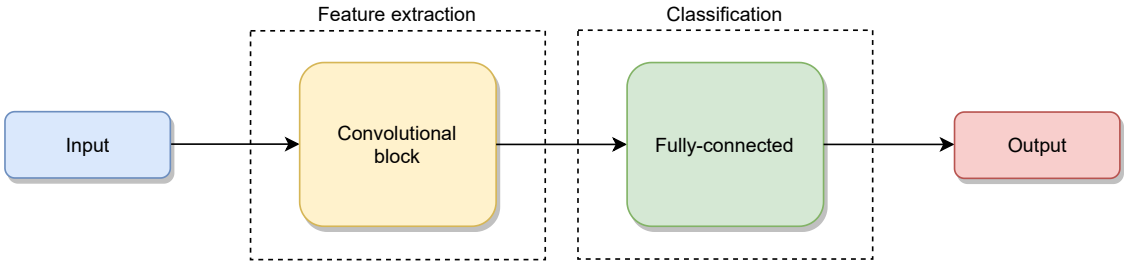


Figure 3: General structure of a convolutional neural network. Input (e.g. an image) is fed through a convolutional neural network for feature extracted, classified in a fully-connected network and then the output is returned.

feature extraction and is then followed by a fully-connected network that does the classification (Khoshdeli, Cong and Parvin 2017). A general block diagram showing this structure is shown in Figure 3, where input in the form of an image-like structure is fed through a convolutional neural network for feature extraction and then that information is used in a fully-connected network for classification. The network predictions are then returned at the output.

Assuming a two-dimensional image can be annotated $\mathbf{I}$ and a two-dimensional filter (also called *kernel*) $\mathbf{K}$ the resulting convolution of these two, $\mathbf{S}$, can be defined as in Equation (2). Notice that the operation is commutative, in that the penultimate term can be equally written as the final term.

$$\mathbf{S}(i,j) = (\mathbf{I} * \mathbf{K})(i,j) = \sum_{m,n} \mathbf{I}(m,n)\mathbf{K}(i-m, j-n) = \sum_{m,n} \mathbf{I}(i-m, j-n)\mathbf{K}(m,n) \quad (2)$$

Figure 4 shows an example of a simple 5-by-5 matrix containing an arbitrary set of binary-valued data (colored in blue). The figure also shows a 3-by-3 kernel (color red), also with binary values, which is to be convolved with the data matrix. Figure 5 shows the first step in the convolution where the purple 3-by-3 area inside the data matrix is the kernel applied to the data. The resulting 3-by-3 matrix (colored in green) contains the sum of multiplied values of the kernel and data matrix. Figure 6 shows one of the following steps where a few more data points have been calculated. The kernel is said to be "striding" with a value of 1 through the data matrix in this example. If the stride was 2 for example, the resulting convolved matrix would have the shape 2×2. The convolution continues in this fashion until the green matrix is filled out.

The example explained above contains a simplification of convolution in applied convolutional neural networks. This is because images usually contains more than one channel. A channel can for example be the red colored channel in an RGB image. In the hidden layers of a convolutional neural network the channels are often called *feature maps* as they contain the learned "features" produced by different kernels applied to the same image. Kernels can both be applied in the spatial (height and width) dimension and in the channel dimension. If the input data contains several input channels c_i, then it is needed as many kernels as there are channels. Concatenating the kernels together is then necessary (Zhang et al. 2020, ch. 6.4.1).

Depthwise separable convolution takes a normal convolution operation and splits it into two separate operations. First is the depth-wise convolution which is a convolution performed on a per-channel basis (each channel is kept separate). This can be seen as the

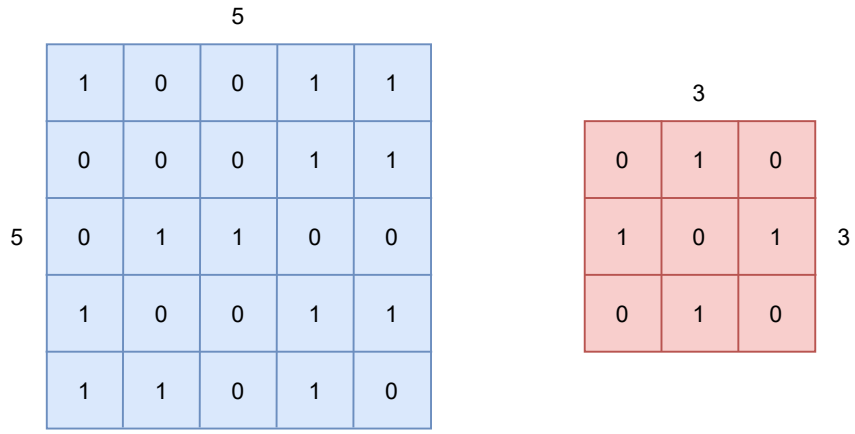


Figure 4: Example of a 5x5 matrix (blue) and a 3x3 kernel (red) that are to be convolved with each other.

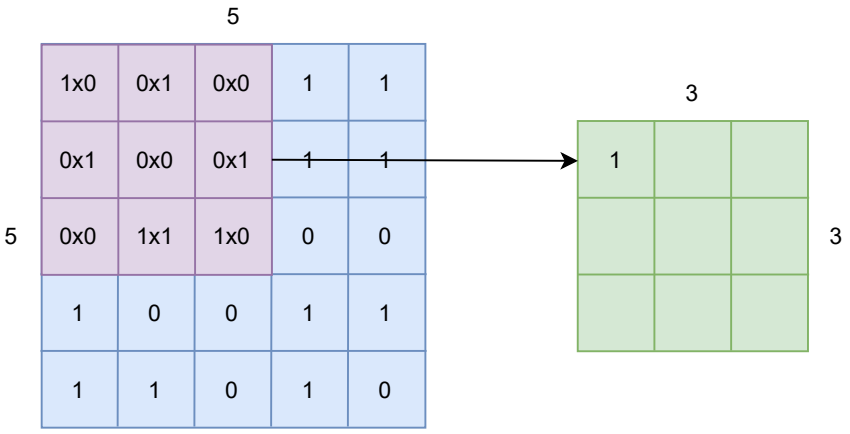


Figure 5: Applying the kernel to the top left 3x3 elements in the data matrix. The resulting convolution is then added to a new 3x3 matrix (green).

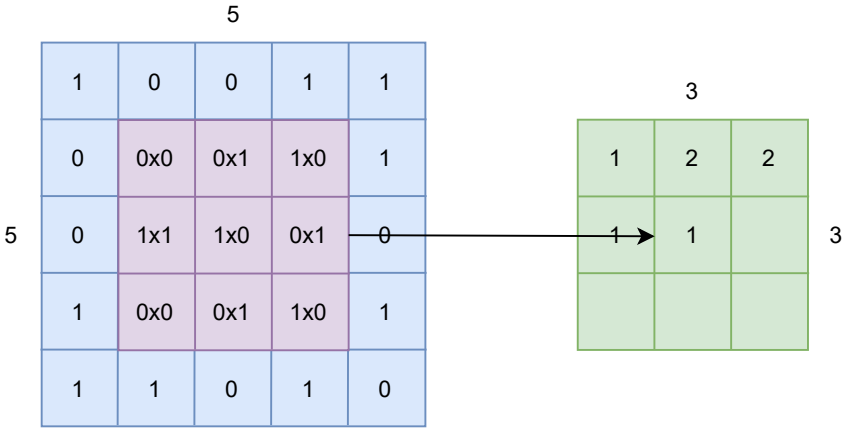


Figure 6: Continuation of the convolution of the kernel and data matrix. The kernel "strides" through the data matrix.

163	149	172
90	111	106
63	86	99

Figure 7: Example of a simple 3x3 image channel with values in the range 0-255 (as for example the red color channel in an RGB).

convolution shown in Figures 4-6, as that example only shows one image channel. Such an operation is then applied to all channels, and a point-wise convolution is then performed on the resulting image. This operation performs a convolution on every single point/pixel in the image but doing so over all channels at the same time. The depthwise separable convolution operation is computationally cheaper than normal convolutions (Bendersky 2018).

2.2.3 Pooling layers

Pooling layers are intermediate layers inbetween convolutional layers, and their purpose is to mitigate the sensitivity of the convolutional layers and to spatially downsample representations (Zhang et al. 2020, ch. 6.5). For the sensitivity, imagine that an image is represented by a matrix X of shape $1024 \times 1024 \times 3$. If the image is moved a single pixel to the right then the output of a network will be completely different for this picture. Therefore the pooling layers will detect nearby spatial similarities. Pooling layers acts as a summary statistic of nearby outputs (Goodfellow, Bengio and Courville 2016, p. 335).

Usually the pooling layers are either a maximum or an average layer. As an example assume there is a 3x3 image channel as in Figure 7 (for example the red channel in an RGB image). The first step of applying a 2x2 pooling operation to this image is shown in Figure 8. The resulting average pooling is shown as the blue matrix and is the average of all elements in the 2x2 pooling operation, while the maximum pooling operation returns the largest element in the 2x2 matrix. Figure 9 shows the finished pooling operation for both average and maximum pooling. For the first step the average of the elements is $(163 + 149 + 90 + 111)/4 = 128.25$, which is then promptly added to the first element in the avg. pooling layer. The maximum in the same is 163.

2.2.4 Backpropagation

Backpropagating in a neuron-like structure was proposed by Rumelhart in 1986 (Rumelhart, G. E. Hinton and Williams 1986). What this algorithm aims to do is to calculate the gradients in the network so that the weights and biases can be updated.

To calculate the backwards pass through a network, the forwards pass must first generate an output in the final layer of the network. Rumelhart, Hinton and Williams shows that defining the total error E, the gradient of this error with respect to the input can be used to propagate the gradient towards the input. The total error E of a layer is defined as in Equation (3), where c is an index over input-output pairs, j is an index over output units,

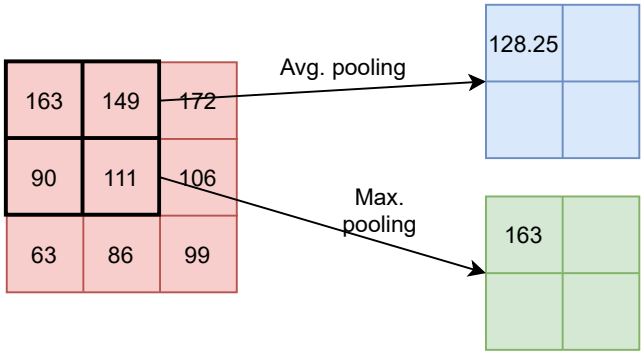


Figure 8: The first step of applying a 2x2 pooling operation to the 3x3 image. Average pooling is shown in the resulting blue 2x2 matrix, and maximum pooling is shown in the green 2x2 matrix.

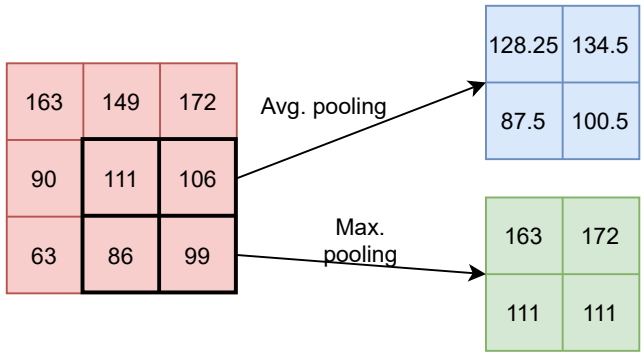


Figure 9: The last step of applying a 2x2 pooling operation to the 3x3 image. Average pooling is shown in the resulting blue 2x2 matrix, and maximum pooling is shown in the green 2x2 matrix.

y_j is the output of node j and d_j is the desired output.

$$E = \frac{1}{2} \sum_c \sum_j (y_{j,c} - d_{j,c})^2 \tag{3}$$

The output of the node is generally fed through a non-linear function, called activation functions. More on this in Section 2.2.8. If non-linear activation functions are not used, the whole feed-forward neural network will just be a linear function of its input (Goodfellow, Bengio and Courville 2016, p. 168). In the original paper for backpropagation, the sigmoid activation function is used as an example function, and is shown in Equation (4), where the output of the function for node j is y_j and the input is x_j. This could be any other activation as well, as the sigmoid is only used as an example.

$$y_j = \sigma(x_j) = \frac{1}{1 + e^{-x_j}} \tag{4}$$

Using sigmoid the gradient of the error E with respect to the input x_j is shown to be expressed as in Equation (5).

$$\frac{\delta E}{\delta x_j} = \frac{\delta E}{\delta y_j} \cdot y_j(1 - y_j) \tag{5}$$

They show that for a weight from layer i to layer j, w_{ji}, the gradient can be expressed as in Equation (6).

$$\frac{\delta E}{\delta w_{ji}} = \frac{\delta E}{\delta x_j} \cdot y_j \tag{6}$$

Lastly the paper shows that calculating the gradient of the error with respect to the outputs of the penultimate layer i can be found as in Equation (7). This equation is then used to propagate the error from the output layer towards the input layer.

$$\frac{\delta E}{\delta y_i} = \sum_j \frac{\delta E}{\delta x_j} \cdot w_{ji} \tag{7}$$

2.2.5 Loss function

The loss (or cost) function is as mentioned in Section 2.2.1 a measure of (un)certainty in a machine learning model. For a binary problem the Binary Cross-Entropy (BCE) loss function can be used. This function aims to penalize bad predictions. It is defined as in Equation 8, where x_n is the nth predicted element in the batch, of total size N (more on batch size in Section 2.2.6), and y_n is the corresponding desired output.

$$l(x, y) = L = [l_1, ..., l_N],\ l_n = [y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)] \tag{8}$$

If the desired target $y_n = 1$ and the prediction goes to zero, the loss will become large due to $\log x_n$ in the first term approaching infinity. If $y_n = 0$ and $x_n = 1$ then the second term

goes towards infinity. If however $x_n = y_n = 1$ or $x_n = y_n = 0$ then the first or second term goes to zero respectively.

2.2.6 Optimization algorithm

The aim of an optimization algorithm is to minimize the loss of a model. Assuming we have a set of parameters θ, the loss function will quantify the quality of the model wrt. these parameters. The goal of the optimization algorithm is then to find the parameters θ (weights and biases) that minimizes the loss function (Gilon et al. 2021). For a convolutional neural network the weights that are updated in each backwards pass are the kernel elements. For a fully-connected network the weights between nodes are updated.

The most used optimization algorithms for deep neural networks are the ones based on stochastic gradient descent (SGD) (Goodfellow, Bengio and Courville 2016, p. 149). These work by calculating the gradient and moving in the direction of the negative gradient of the loss function wrt. the weights/parameters (Chen 2020).

An example of an SGD-based optimizing algorithm is Adam (Adaptive moment estimation) (Kingma and Ba 2017). This optimizer combines the gradient descent algorithm AdaGrad (Duchi, Hazan and Singer 2011) with RMSProp (Tieleman and G. Hinton 2012). The Adam algorithm updates exponential moving averages of the gradient m_t at time step t and the squared gradient v_t where the hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ controls the decay rates of the gradients' moving averages. In the original paper $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The moving averages are estimates of the 1st moment (mean) and the 2nd moment (the uncentered variance). The moments m_t and v_t are defined respectively in Equations (9) and (10), where g_t is the gradient at time step t (Ruder 2016).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{9}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{10}$$

The original paper introduces bias-corrected moments, $\hat{m}_t$ and $\hat{v}_t$, to counteract initialization bias pulling the moments towards zero. These moments are defined in Equations (11) and (12) respectively.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{11}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{12}$$

Lastly the parameter update is done as in Equation (13). The term η is known as the learning rate and acts as the size of the step between each parameter ($\eta = 0.001$ in the original paper). The ϵ term is chosen to be very small (10^{-8} in the original paper) and prevents the denominator term to ever becoming zero (Kingma and Ba 2017).

$$\theta_t = \theta_{t-1} - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{13}$$

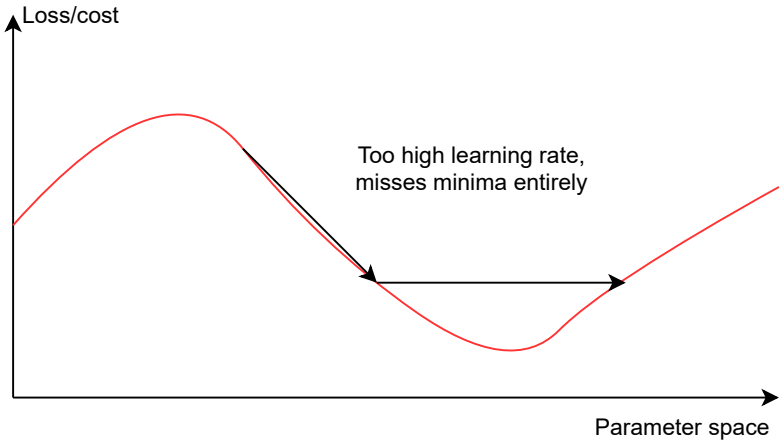


Figure 10: Parameter updates with a learning rate that is set too high. The updates to the parameters (illustrated with black arrows) misses the minima of the cost function (in red) entirely due to the large updates of the parameters.

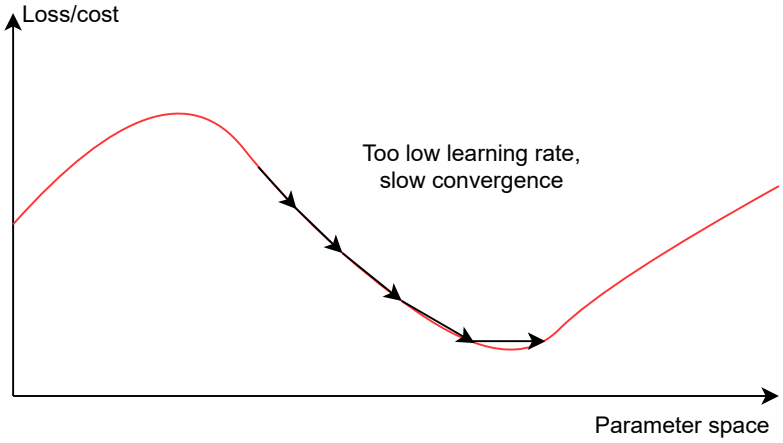


Figure 11: Parameter updates with a learning rate that is set too low. The updates of the parameters (illustrated with black arrows) slowly converges towards the minima of the cost function (in red).

To avoid missing the minima of the parameter search, it is possible to introduce a learning rate scheduler. Having a too large learning rate will lead to the parameter update jumping over the minima, while a too small learning rate will lead to slow training and sub-optimal performance (Zhang et al. 2020, ch. 11.11). A too high learning rate is illustrated in Figure 10 and a too low learning rate is illustrated in Figure 11. The red curve illustrates some cost function as a function of a given parameter space and the arrows shows the parameter updates as arrows along the curve. Therefore it could be beneficial to introduce a scheduling of the learning rate so that the learning slows down as the parameter optimization closes in on the minima. A parameter γ is introduced so that every N number of epochs (chosen by the user) the learning rate is reduced by a factor of γ.

Since optimizers based on SGD relies on the stochastic approximations, it is common to feed mini-batches of data into a deep learning model (Masters and Luschi 2018). Feeding a single data point through a network while optimizing with SGD-based algorithms will lead to the search being influenced by noise. Using mini-batches gives the optimizers a more generalized view of the data, avoiding noisy input.

2.2.7 Regularization in neural networks

Dropout is a method to introduce regularization and avoid overfitting in neural networks (Srivastava et al. 2014). Trying to deal with overfitting at test time in large networks (which are slow due to many operations at run-time) by combining the predictions of many networks simultaneously is difficult, which is why dropout was introduced. The method is called dropout because neurons in a network are actually "dropped out" of the network during training, i.e., a neuron with its connections is removed.

Batch normalization is also a common method to introduce regularization in a neural network (Ioffe and Szegedy 2015). Using this method, the input to the batch normalization layer is standardized. First the mean and standard deviation of the current batch is computed and the input is then standardized. The standardized input is then scaled by a factor α and a bias μ is added. α and μ are the parameters that are learned during training in this layer.

2.2.8 Activation functions

As mentioned in Section 2.2.4 the non-linearity introduced with activation functions will stop a feed-forward networks from only being a linear function of its input. Following will be a presentation of the relevant activation functions used in the implementation of the models in this thesis. The input to an activation function is the summed input of all nodes in the previous layer.

The linear unit (or identity unit) lets the input pass through as-is. It is simply defined as $y = x$ (Brownlee 2021).

The Rectified Linear Unit (ReLU) (Goodfellow, Bengio and Courville 2016, p. 171) is defined mathematically as in Equation (14), where output is y and input is x. This function is non-linear, and has well-defined derivatives for input values below and over 0. Different implementations may choose to define the derivative at 0 differently as it is not well-defined in itself. Applying an activation function can be called the detector stage because the non-linear nature of the functions only "detects" a specific input (for example only positive input for the ReLU function) (Goodfellow, Bengio and Courville 2016, p. 335).

$$y = \max(0, x) \tag{14}$$

The ReLU6 activation function can be first found in (Krizhevsky and Geoffrey Hinton 2010) and is defined as in Equation (15) for an input x and output y. This is very similar to a regular ReLU but differs from the fact that it returns x only in the interval $[0, 6]$ and is flat elsewhere. The reason for choosing 6 as a limit in the ReLU was due to its ability to learn sparse features earlier.

$$y = \min(\max(0, x), 6) \tag{15}$$

The sigmoid (or logistic) activation function has already been defined as in Equation (4). This function is used to set a boundary on the output of a neuron, squashing the input which can have a range $(-\infty, \infty)$ to the range $(0, 1)$ (Zhang et al. 2020, ch. 4.1.2.2). Applying this will then make the output more numerically stable. The sigmoid activation

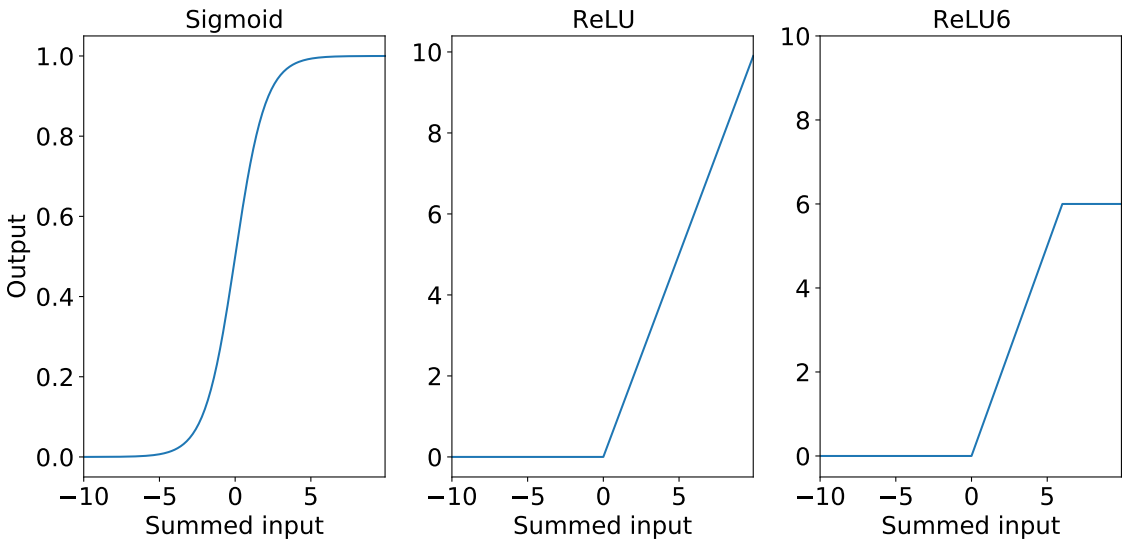


Figure 12: Activation functions as explained in Section 2.2.8. The sigmoid function on the right squeezes input between 0 and 1. ReLU returns the input for input > 0 and zero for input < 0. ReLU6 returns the same as ReLU for input < 6 but returns 6 for input > 6. Output of the activation functions is shown on the y-axis, and summed input into the function on the x-axis.

can be interpreted as a probability of the output units of a network in binary classification problems. Figure 12 shows the activation functions as explained above.

2.2.9 Transfer learning

Transfer learning is a re-purposing of already trained models (along with its weights and biases) of one problem and applying it to another problem. Assuming a problem P_1 has a similar distribution as problem P_2 it is possible to exploit what has been learned for the first problem and then improve generalization in the other problem (Goodfellow, Bengio and Courville 2016, p. 534). As an example Goodfellow explains that this can be understood as the input being of the same nature in both problems, but the targets being of different types. The input can for example be images of animals, but in one case the targets are cats and dogs whereas in another case the targets can be elephants and giraffes.

2.2.10 Metrics

Different metrics will be used to present the results, and following is a clarification of them. The metrics stated below applies to a binary problem in which there is either presence or absence of a case.

If the model predicts presence and there is also presence in the ground truth, this is a true positive (TP). Predicting presence of a case when it is actually absent in the ground truth is a false positive (FP). True negative (TN) is a prediction of absence when there is nothing present and lastly false negative (FN) is a negative prediction when the case is present in the ground truth.

Precision can be interpreted as a metric of how many of the predictions that were pos-

itive are actually true positives. Recall (also called true positive rate) is the portion of how many of the relevant cases (e.g. wanted vocalizations) are being correctly predicted (Google 2020b) (Ghoneim 2019). Precision and recall can be expressed mathematically as in Equations (16) and (17) respectively.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{16}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{17}$$

For a detection problem it is interesting to look at the false positive rate (FPR), which is defined as the total number of false positives divided by the total number of negative cases (Pico.net 2021). It is the rate of how many negative cases that were falsely predicted as positive and is shown in Equation (18).

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{18}$$

Accuracy is defined as the total number of correct predictions divided by the total number of predictions made (Google 2020a) (Ghoneim 2019). Mathematically it can be stated as in Equation (19).

$$\text{Accuracy} = \frac{\text{true predictions}}{\text{total \# of predictions}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{19}$$

The total number of predictions and true negatives are expected to be relatively large compared to the rest of the numbers because the toad does not vocalize continuously and not in chorus (AmphibiaWeb 2020). It is therefore beneficial to mainly look at precision, recall and the FPR as indicators of the model performance.

3 Methodology

This Section includes all the methods and implementations that are used to produce the results that can be found in Section 4. In Section 3.1 a short introduction to the equipment and code libraries used in the implementation is given. Following this, an in-depth look into the data is presented in Section 3.2. A short walk-through of the pre-processing is found in Section 3.3, while a detailed description of the neural network used is given in Section 3.4. Lastly the process of training and testing the model is described in Sections 3.5 and 3.6 respectively.

3.1 Equipment and code libraries

Both a desktop computer and laptop were used during the course of this thesis. The desktop computer used (mainly) for training and testing has an ASUS Nvidia GTX 1070 with 8 GB of VRAM, AMD Ryzen 5 1600X processor, 16GB RAM with Windows 10. A HUAWEI Matebook X Pro was also used both for some training, testing and pre-processing. The laptop has Debian 10 Buster and is equipped with an Intel Core i5-8250U, 8GB RAM, integrated graphics from Intel (UHD Graphics 620) as well as an NVIDIA MX150 which was not used for testing and pre-processing.

Both computers are running Anaconda's Spyder version 4.2.1. For all deep learning scripts the Pytorch library (Paszke et al. 2019) for Python (Python Software Foundation 2020) were used. Both torchaudio and torchvision of the Pytorch framework was used for pre-processing of the audio. A comprehensive list of the main Python libraries and versions used follows below.

- Python 3.8.5 (Python Software Foundation 2020)
- Pytorch 1.7.1 (Paszke et al. 2019)
- torchaudio 0.8.0 (Paszke et al. 2019)
- torchvision 0.9.0 (Paszke et al. 2019)
- librosa 0.8.0 (McFee et al. 2020)
- pandas 1.2.3 (Reback et al. 2021)
- numpy 1.19.2 (Harris et al. 2020)
- scipy 1.6.1 (P. Virtanen et al. 2020)
- efficientnet-pytorch 0.7.1 (https://github.com/lukemelas/EfficientNet-PyTorch)
- scikit-learn 0.24.1 (Pedregosa et al. 2011)

3.2 About the data

This section will mainly be about the data that was used in this thesis. An introduction of the sites where the recordings were made will follow, as well as any relevant equipment used when it was collected. A short discussion on the soundscape found at the sites will follow and lastly a bit about the labeling process of the data will be explained.

3.2.1 Data collection

The data used in training and validating the deep learning-based models were collected over a two year period (2015 and 2016) on two different sites in Northeastern France. Recordings were taken in the breeding season of the P. fuscus, spanning from (at the earliest) mid-March to (at the latest) mid-July. Two sites were chosen for the recordings, Mothern and Sauer's Delta (named Sauer from here on). They are located on the river Rhine's floodplain and can be found 4 kilometers apart (Dutilleux and Curé 2020).

Programmable SM2 Songmeter programmable audio field recorders connected to an HTI-96 hydrophone were used to monitor the Pelobates Fuscus (both from Wildlife Acoustics, Maynard, U.S. (Wildlife Acoustics, Inc. 2021)). To ensure coverage of the whole breeding season, the recorders operated continuously from late March to late June. For the four campaigns (from here on named Mothern 2015, Sauer 2015, Mothern 2016 and Sauer 2016), SM2 recorders were programmed to record for 5 min every half hour. Audio was stored at a sampling rate of 16 kHz at 16-bit resolution in WAV format (Dutilleux and Curé 2020).

Vocalizations of the juvenile Pelobates fuscus were collected from the animal sounds library of The Museum für Naturkunde in Berlin (The Museum für Naturkunde 2021a) (The Museum für Naturkunde 2021b). The juvenile toad recordings that were retrieved from this archive are all from the paper by (Hagen et al. 2016). The recordings were downloaded in MP3 format at a sample rate of 44.1kHz, and are openly available on a Creative Commons license (CC BY-NC-SA 3.0 DE) (Creative Commons 2021). A full list of the sounds used can be found in Appendix A.

3.2.2 Soundscape at the sites

The Mothern and Sauer sites contains a rich soundscape. Anurans other than the Pelobates fuscus can be found in these sites. The Sauer site is a breeding site for species like the agile frog (*Rana dalmatina*), the marsh frog (*Pelophylax sp.*) and the European tree frog (*Hyla Arborea*). Mothern contains both the agile frog and the European tree frog, but not the marsh frog (Dutilleux and Curé 2020).

At least six species of passerine birds were identified in the original paper; the common blackbird (*Turdus merula*), the song thrush (*Turdus philomelos*), the common chaffinch (*Fringilla coelebs*), Phylloscopus collybita (Phylloscopidae), Acrocephalus scirpaceus (Acrocephalidae) and Erithacus rubecula (Muscicapidae), as well as the Common cuckoo (Cuculus canorus (Cuculidae)) (Dutilleux and Curé 2020).

Invertebrates can also be heard making sound in the later dates of the campaigns. Low frequency noises found in the data can be attributed to factors such as water turbulence, wind and even creatures moving and touching the equipment.

Due to the sites being relatively close to urban sites, a fair bit of anthropogenic noise can be heard. Everything from faint human talking, to dogs barking, revving motorcycles and even train horns were found in the recordings. It is pointed out in (Dutilleux and Curé 2020) that there is no significant road or rail infrastructure present in a 400 meter distance from both sites, and that most of the anthropogenic sounds might be heard due to an atmospheric temperature inversion. Abiotic sounds like rain and droplets hitting the hydrophone are also heard regularly throughout the campaigns.

3.2.3 Labeling of the data

Labeling of the data was done manually in Audacity version 2.4.2 (Audacity Team 2020). Labeling a file in Audacity exports .TXT-files with three columns separated by a tabulator; one for the label, one for start time of the sample and the last for the ending time of the sample. This was promptly read with the pandas (Reback et al. 2021) *read_csv* function, with the tabulator separator specified in the function call.

A general structure of a label file can be seen in Table 1, where the first column contains the start time, the second contains end time and the last one contains the label. For detection of the adult Pelobates fuscus binary values were chosen to indicate presence and absence. For example if a Pelobates fuscus is found in the recordings it was labeled a '1' while absence of the toad was labeled '0'. This resulted in .TXT files that can remind of the structure seen in Table 1.

Table 1: A simple structure of how a label file can look.

Start time (sec)	End time (sec)	Label
0.523	1.263	1
4.249	4.910	1
...	...	...

The Mothern 2015 campaign was used for labeling the training and validation data for adult P. fuscus. The early days of the Sauer 2015 campaign were also used to label the agile frog. It was ensured that the files used from Sauer 2015 were not included in the ground truth. Training and testing on the same data will not give an actual insight into the performance of the model as the trained model will already know the output of the validation data. A full list of dates and times used to create both the detector and the multi-label classifier can be found in Appendix B in Table 9 and Table 8 respectively. A ground truth dataset provided the supervisor of this thesis was also labeled and prepared for testing the detection model, which is expanded more upon in Section 3.6.1.

Multi-label classification

For detection of the juvenile specimens a multi-label classifier was designed. What this means is that the prediction of the model can be several of the classes at the same time. The solution is chosen to be a multi-label classification one due to the fact that many of the species vocalize at the same time.

For the multi-label classifier a few classes were initially chosen, as they are more present at the sites than others (Dutilleux and Curé 2020). Table 2 shows the classes that are used in the final system with a short description of what they are.

Both the agile frog and European tree frog are present at both sites and were therefore chosen to be included in the classifier. The adult and juvenile specimens of the P. fuscus were also annotated separately.

It was found unnecessary to label separate bird species as including separate cases would complicate the number of classes. As the recordings were made under the pond surfaces with a hydrophone it is not seen as necessary to include separate classes for bird songs. Thus a general "*bird*" label was used when annotating the data.

The "*other*" label refers to all sounds that are not the other classes that were chosen and that deviates from the general background sounds. This includes other wildlife,

Table 2: Annotations used in the multi-label classifier with short description of what they mean.

Index	Label	Description
0	**agile**	Agile frog (*Rana dalmatina*)
1	**bird**	General class containing bird song
2	**inv**	Invertebrates
3	**other**	Class containing any other non-background sounds
4	**pfa**	Adult common spadefoot toad
5	**pfj**	Juvenile P. fuscus
6	**rain**	Any and all precipitation in a sound clip
7	**tree**	European tree frog (*Hyla arborea*)

anthropogenic sounds like the ones explained in Section 3.2.2 and abiotic sources like shock/scratching sounds, clicks etc. Invertebrates were also included in its own class, as the later months of the recordings contains many cases were for example the adult P. fuscus vocalizes while an invertebrate is making sounds.

When going through the data, instances were labeled with multiple labels at the same time. For example a bird could be singing at the same time as an adult P. fuscus vocalizes, while it is raining. Then this instance is labeled as *bird,pfa,rain*. An array of length 8 is instantiated with all values set to zero. For all labels present in a sound clip, the value at the corresponding index is then set to 1 in the array (refer to Table 2 to see which labels correspond to the different indices). For the example mentioned before, an array with the values `[0,1,0,0,1,0,1,0]` is created.

All instances are then sliced into 1 second clips (with 50% overlap if the file is longer than 1.5 seconds) and subsequently references to the files with labels, both in array and text form, are added to a .CSV file for easier reading when training. An example of the structure of such a file is shown in Table 3.

Table 3: A simple overview of what a .CSV file for multi-label classification can end up looking like.

Filename	Labels	Labels (text form)
filename_1_1	[0,1,0,0,1,0,1,0]	bird,pfa,rain
filename_1_2	[0,0,0,0,0,1,0,0]	pfj
filename_2_1	[0,0,1,1,1,0,0,1]	inv,tree,pfa,other
filename_3_1	[1,0,0,1,0,0,0,0]	agile,other
...	...	...

The final number of data points for the adult detector contained 2257 positive cases and 4576 negative cases. For the classifier, 280 cases of the juvenile were used and the other classes were spread out over the remaining 3741 files. The total number of each class used in the final model is shown in Figure 13. The sum of all the occurrences are not equal to the number of files, as some sounds are labeled with multiple classes. The total number of each class is found in Table 4.

Table 4: Count for each of the classifier's 8 classes.

Class	**agile**	**bird**	**inv**	**other**	**pfa**	**pfj**	**rain**	**tree**
Count	534	1213	560	896	1018	280	703	338

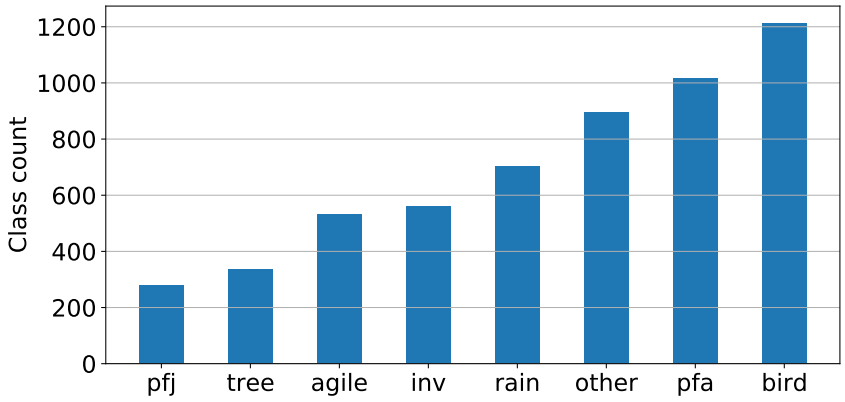


Figure 13: The count of each of the 8 classes used in the final classifier model. Labels used can be found in Table 2.

3.3 Pre-processing

3.3.1 Audio preparations

As discussed in Section 2.1.1 the call of the P. fuscus can last on average up to 0.9 seconds. Due to this fact the audio length chosen to make the dataset is *1 second*. After labeling each case of the P. fuscus in Audacity, the label files are loaded in as pandas (Reback et al. 2021) DataFrames in Python. The corresponding sections in the .WAV files are then padded randomly on each side (if it is shorter than one second) or sliced into several consecutive 1-second files (if longer than one second). A comma-separated values (.CSV) file is then created containing reference to the audio file name and its corresponding label. .CSV files were created for both the detection of adult and juvenile specimens.

The audio is loaded with torchaudio's load function which normalizes the audio in the range $[-1, 1]$ by default. The spectrogram transform is initialized using torchaudio's functionality. Since the input size to the network used is known $(224, 224, 3)$ (3 for number of channels), it is possible to calculate the number of Fourier bins and hop length needed when transforming to spectrogram. Since the number of bins produced by the spectrogram transform is $\#bins = (N\_FFT/2) + 1$, it can be rearranged to $N\_FFT = (\#bins - 1) \cdot 2 = (224 - 1) \cdot 2 = 446$. By default, in torchaudio, the window size is equal to the number of Fourier points.

Since the sampling rate is constant for all audio (16 kHz = 16000 samples per second) and all audio is the same length (1 second), the hop length can also be calculated by using the desired image size. Since the number of wanted windows is known (224), it is possible to use this information. Multiplying the number of windows (224) with the hop length almost gives the sample rate. The end of the last window will miss the end of the second by a length equal to the overlay between two consecutive STFT windows (which in this case is (446 − hop length)/446)). The hop length needed to get the wanted image size is found in Equation (20).

$$\text{hop length} \cdot 224 - (\frac{446 - \text{hop length}}{446}) = 16000 \Rightarrow \text{hop length} = \frac{16000}{224 + \frac{1}{446}} \approx 71.43 \quad (20)$$

By pre-computing the number of Fourier points and hop length any potential major alter-

ations done to the image by resizing can be avoided. A hop length of 71 was used in the spectrogram transform, so that the resulting number of points in the time dimension is 226. After transforming the audio to spectrogram the amplitude was transformed to dB.

The audio is bandpass-filtered with a Butterworth filter of order 5. For detection of the adult P. fuscus the audio is bandpassed between 100Hz and 3500Hz, and concatenated into three channels as this is the number of input channels needed in the network used. This frequency range was chosen with trial and error, but it was found that having a too low highcut frequency resulted in worse predictions. Having a too high frequency might include bird song or other sounds like rain which masks the vocalization.

For the multi-label classification and possible detection of the juvenile only frequencies lower than 100Hz were removed, which was achieved with an order 5 Butterworth high-pass filter. This frequency was chosen due to the juvenile recordings having low frequency background noise. It is chosen not to remove any of the high frequencies as some of the vocalizations of the juvenile are broadband. The low-passed spectrogram is, as for the detection, concatenated into three channels due to the network input needing to be three channels. More on the network can be found in Section 3.4.

Lastly a resize is done to the "image" of the spectrogram to ensure that it has shape $224 \times 224 \times 3$.

3.4 Architecture: EfficientNet

For training a model, the EfficientNet architecture by Mingxing Tan and Quoc V. Le of Google Brain was used (Tan and Le 2020). The building blocks used in EfficientNet are based on a combination of MobileNetV2's inverted bottlenecks (Sandler et al. 2019) as well as squeeze-and-excitation blocks (Hu, Shen and Sun 2019) which will be presented in this section.

The inverted bottleneck architecture can be seen in Figure 14 and in Table 5 (where input and output are written with their respective shapes). First is an input of shape $h \times w \times c$ (h is height, w width and containing c channels) which is convolved point-wise. The resulting output is an expanded image in the channel dimension by a factor of t. This expansion will result in the intermediate layers having more channels than the input and output, acting as a kind of inverted bottleneck. Following is a depth-wise convolution with a 3-by-3 kernel and a stride of 1. This means the image will keep its height and weight dimensions and also keep its expanded number of channels tc. Lastly a 2D point-wise convolution, the number of channels will be reduced from tc to c'. What this achieves is that instead of letting a single kernel find both cross-channel and spatial features in an image at the same time, the operation is split into two. The cross-channel correlations and spatial correlations are then done separately to gain more knowledge while at the same time using less processing power (Sandler et al. 2019).

Note that the shortcut connection (the connection from 'Input' to 'Add' in Figure 14) is only used if the number of input channels c is equal to the chosen number of output channels c'. This is because the shapes of the input and output must be identical in order to add them together. If $c \neq c'$ the shortcut connection is removed and only the output of the final 2D point-wise convolution is passed on in the network.

Table 5: Inverted bottleneck residual block as seen in (Sandler et al. 2019). Here k is the number of input channels, h and w is the height and width of the input respectively, t is the expansion factor, s is stride and k' is the wanted output channels.

Input shape	Operator	Output shape
$h \times w \times c$	1x1 Conv2d, ReLU6	$h \times w \times (tc)$
$h \times w \times (tc)$	3x3 dwise stride=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tc)$
$\frac{h}{s} \times \frac{w}{s} \times (tc)$	Linear activation 1x1 Conv2d	$\frac{h}{s} \times \frac{w}{s} \times c'$

In addition to the inverted bottleneck, EfficientNet uses squeeze-and-excitation optimization as seen in (Hu, Shen and Sun 2019). The aim of this block is to weight each channel adaptively. It works as follows: A squeeze-excitation block takes a convolutional block as input, where each of these channels are *squeezed* into a single numeric value by applying a global average pooling operation on a per-channel basis. To fully capture the channel-wise dependencies the *excitation* operation is introduced. This is achieved by introducing two fully connected layers, where the first one is applied with the ReLU activation function and the second is applied with the sigmoid activation function. The first fully connected layers is also applied with a reduction ratio r so as to reduce the dimensionality. The resulting matrix contains per-channel weights and it is then multiplied with the input matrix. Figure 15 shows the general squeeze-excitation block as explained.

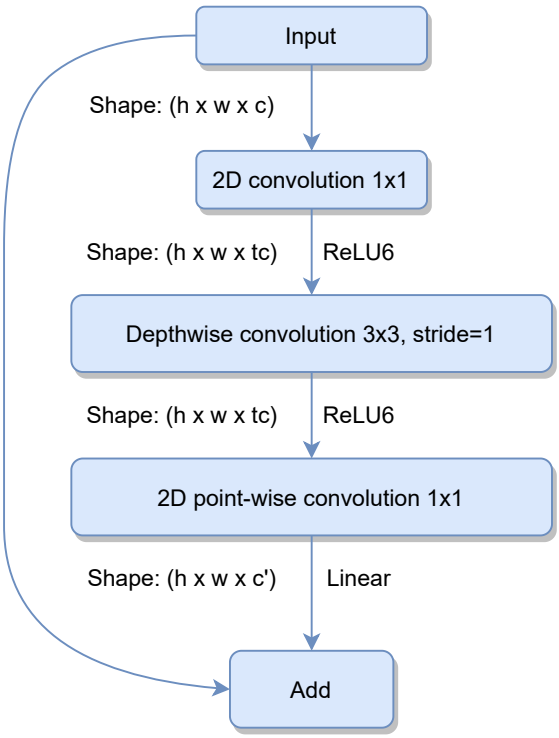


Figure 14: Inverted bottleneck residual block (from (Sandler et al. 2019)) with stride $s = 1$. An image of input shape (h, w, c) (height, width, channels) is fed through the bottleneck block, with a given expansion t in the channel dimension. A depth-wise convolution is performed on a per-channel basis, keeping the number at the number of expanded channels. A point-wise convolution is then performed to reduce the number of channels to c'.

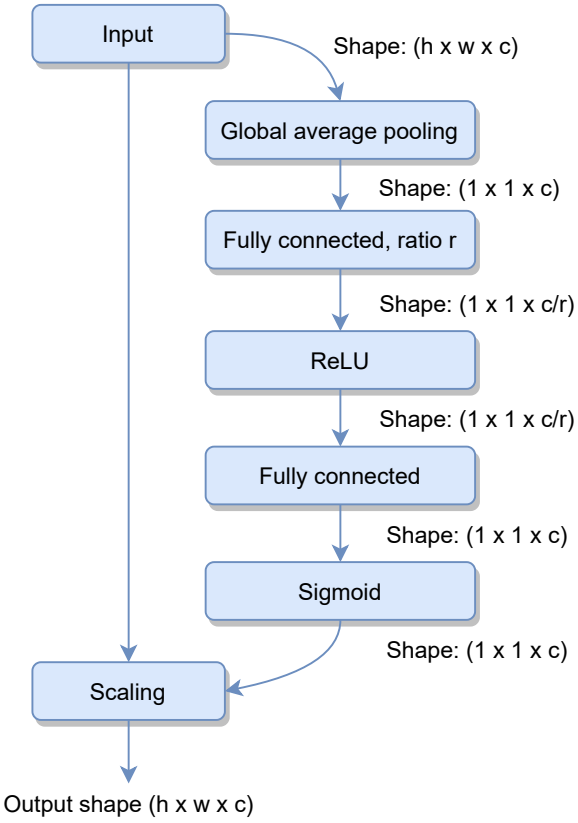


Figure 15: Squeeze-excitation block from (Hu, Shen and Sun 2019).

Using the building blocks as explained earlier the simplest EfficientNet, EfficientNetB0, can be introduced. The structure of the convolutional part of the network is shown in Table 6. Input images to EfficientNet are defined to have shape $(224 \times 224 \times 3)$, where height and width are the two first elements and the number of channels are last. Each MBConv block contains a squeeze-and-excitation block.

Table 6: EfficientNetB0 structure as introduced in (Tan and Le 2020). The number behind MBConv in stages 2-8 means the expansion factor t. FC in the last stage stands for fully-connected.

Stage i	Operation	Resolution	# of channels	# of layers
1	Conv 3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1, pooling and FC	7×7	1280	1

This network can be made larger (deeper, wider and higher resolution) following a method the authors calls compound scaling. Using a compound coefficient ϕ, Equation (21) can be used to scale the network. Here α, β and γ are constants that can be found with a grid search, so that $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ (i.e. for any ϕ the number of FLOPS (computational power) will approximately increase by 2^ϕ) (Tan and Le 2020). The best constants for EfficientNetB0 are found to be $\alpha = 1.2$, $\beta = 1.1$ and $\gamma = 1.15$ (under the constraint mentioned above). To increase B0 to the larger models, choosing for example $\phi = 1$ will yield the EfficientNetB1 model while $\phi = 6$ will yield the EfficientNetB6 model.

$$\begin{aligned} \text{depth} : d &= \alpha^\phi \\ \text{width} : w &= \beta^\phi \\ \text{resolution} : r &= \gamma^\phi \\ \text{so that } \alpha \cdot \beta^2 \cdot \gamma^2 &\approx 2, \\ \alpha \geq 1, \beta \geq 1, \gamma &\geq 1 \end{aligned} \tag{21}$$

The model also contains batch normalization in each of the MBConv blocks, and dropout with a probability of 0.3 used in the layer before the fully connected layer at the end of the network.

For a Pytorch implementation of EfficientNet the one by Luke Melas was used which is available on https://github.com/lukemelas/EfficientNet-PyTorch under an Apache 2.0 License (Apache Software Foundation 2004).

Pre-trained models of EfficientNetB0 and EfficientNetB3 were used, where weights and biases were trained on ImageNet. Due to the pre-trained network having 1000 outputs, the final fully-connected layer had to be modified to have the correct number of outputs. EfficientNetB0 was used as an initial test of the library and the final model used was EfficientNetB3 due to the fact that it has higher accuracies without the need of much

more processing power (about twice the parameters). The desktop could not handle any larger models due to running out of memory. Therefore the B3 was chosen as a middle ground. EfficientNetB3 has a Top-1 accuracy of 81.6% and a Top-5 accuracy of 95.7% with 12 million parameters, whereas a model with 7 times the parameters (84 million parameters, ResNeXt-101 found in (Xie et al. 2017)) has comparable accuracies.

3.5 Training

3.5.1 Data preparations

A custom Pytorch DataSet class is implemented for the detection of the P. fuscus, and the data is subsequently loaded into memory with the DataLoader class. All pre-processing of audio explained in Section 3.3 is implemented in the DataSet. The data folder is needed as an input, with references to all files in the folder with corresponding labels.

3.5.2 Training on EfficientNet

A simple class was made for using the pre-trained EfficientNetB3 model. It is customized to either have an output of 1 node (for the adult detection) or 8 nodes (number of classes for juvenile classification). Pre-trained weights for EfficientNet were simply loaded into the model with the `from_pretrained` function of the library. The last fully connected layer of the model is replaced to contain the right number of outputs. Originally the output of the models contains 1000 ouptut nodes, so some more fully-connected layers were added to get the right amount of output nodes.

More accurately a `torch.nn.Sequential` replaced the last EfficientNetB3 layer, containing a fully-connected layer with 1536 input nodes to 512. This is followed by a ReLU activation and a 0.25 probability dropout, and then another fully-connected layer with 512 input nodes and 128 output nodes. The ReLU activation is then used and a 0.5 probability dropout layer is added before finishing the model with a fully-connected layer with 128 input nodes and either 1 or 8 output nodes depending on the use-case.

As introduced in Section 2.2.5 a loss function is needed to calculate the certainty of the model during training. Due to the binary nature of the labels, BCEWithLogitsLoss from Pytorch is used. This is a variation of the binary cross-entropy loss function described in Section 2.2.5 (Equation (8)), in that the predictions x_n are squeezed with a sigmoid. Mathematically it is described as in Equation 22. It includes a sigmoid for numeric stability.

$$l(x, y) = L = [l_1, ..., l_N],\ l_n = [y_n \cdot \log(\sigma(x_n)) + (1 - y_n) \cdot \log(1 - \sigma(x_n))] \tag{22}$$

A batch size of $m = 32$ were chosen when training the EfficientNetB3 models, as any larger batch sizes did not work due to lack of computational resources. (Masters and Luschi 2018) found that batch sizes between $m = 2$ and $m = 32$ performed consistently better than larger batch sizes.

In every model trained the manual seed is set to 42 with Pytorch, for the sake of reproducibility. To ensure that results can be reproduced, some other flags were set with Pytorch as for example the `torch.backends.cudnn.deterministic = True` which avoids non-deterministic factors across platforms and Pytorch versions. A random 80-20 train-test

split (80% train data, 20% validation data) is done when loading the data to ensure the model trains with randomly chosen data containing all classes. The training data is then shuffled to ensure that it contains examples from all classes.

Training time is dependent on model complexity and number of epochs. The final model of the adult P. fuscus (based on a pre-trained EfficientNetB3) is trained on 40 epochs. Running that on the desktop computer (see Section 3.1 for specs) took about 1 hour and 15 minutes, at around 2 minutes per epoch for a batch size of 32. 40 epochs were found to yield the best results, and it was found through numerous trials. The learning rate for the Adam optimizer is chosen to be 0.001 with a learning rate scheduler step size of 15 and a reduction parameter $\gamma = 0.2$. Default values of all other parameters are used in the final model, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.

During training of the model, a part of the complete dataset is also set away for validation during this process. This is to assess the performance of the model during training. When iterating through the validation all gradients are stopped from updating because this part of the dataset should only be used to assess performance and not actually train the model. The `model.eval()` flag in Pytorch is set during validation to stop the network from using regularization methods like dropout and batch normalization.

Classifier for juvenile vocalizations

The juvenile classifier is trained for 60 epochs, and at about 2 minutes per epoch for a batch size of 32 on EfficientNetB3 this resulted in about 2 hours of training time. The number of output nodes in the network is changed to 8. No other changes were made to the network than this. The same settings were used in the training as for the adult and the same hyper-parameters were set for optimizers, schedulers and loss. However the model is trained for longer due to the additional complexity of the output of the data.

3.6 Testing

3.6.1 Ground truth/testing data

600 30-second ground truth files was prepared for detection of adult specimens (Dutilleux and Curé 2020). The files used to test were taken from the Mothern 2016, Sauer 2015 and Sauer 2016 campaigns. There are also ground truth data available from Mothern 2015, but this was not used due to the fact that the detector was trained on data from this campaign. This ground truth was pre-labeled with a binary value of whether or not the P. fuscus vocalizes in that file as well as how many vocalizations were present. As the solution developed in this thesis works on a second-to-second basis, it was found necessary to manually label the times where vocalizations were found. This was done so that a more robust ground truth could be used when evaluating the models. Every occurrence of the P. fuscus was labeled with a '1'. The labeled instances' start times were rounded to the nearest 0.5 second due to the jumping window solution having an overlap of 50%.

However this can lead to some problems which are solvable. Imagine a toad is vocalizing as in Figure 16. At the first of the two notes it can be seen that the beginning of it is not included in the time that was rounded to the nearest 0.5 second. The model is predicting with 0.5 second intervals, so the time step $t = 2$ in the figure is likely not to be the only predicted time. The windows that starts at $t = 1.5$ and $t = 2.5$ also includes significant parts of the vocalization, so including these windows in generating the ground truth was done in the final test script. This is illustrated in Figure 17. So for a single ground truth time, both the previous and next time steps are added.

Another type of testing is also done where only the rounded ground truth time is compared to the predicted times, i.e. the previous and next windows are not added as prediction times.

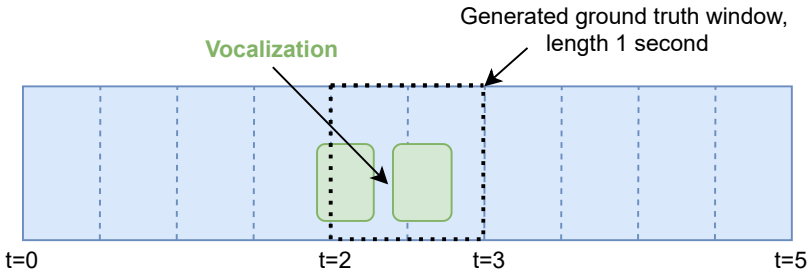


Figure 16: A toad vocalizing close to a time step, but rounding the time to the closest 0.5 seconds will lead to some of the vocalization not being included as a positive in the ground truth. Time is along the x-axis, and the y-axis is only used to illustrate vocalizations.

The same pre-processing that was done on the training data was also done on the testing data. This includes creating spectrogram with the same parameters as explained as in Section 3.3.1.

3.6.2 Testing procedure

Adult individuals

The ground truth audio is sliced into 1 second "chunks" with 50% overlap. For 30 second files this gives 59 unique predictions per file (due to the window starting at 29.5 not

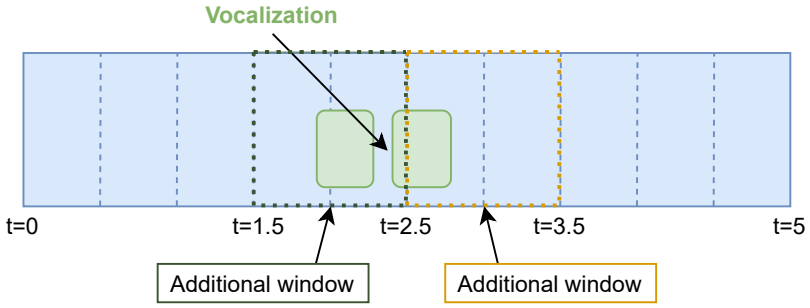


Figure 17: The time steps before and after the rounded ground truth time are added as ground truth times, as they also contain significant parts of the vocalization.

being included), for a total of $600 \cdot 59 = 35400$ predicitons. If, and only if, two or more consecutive chunks give positive predictions a boolean is set to true and the prediction times of the current and previous windows are kept to compare it with the ground truth. This way false positives were avoided more frequently because single predictions (which can come from e.g. rain or other clicking sounds) are not included.

The final way that is tested is to use no overlap between consecutive windows, meaning there is a 1 second hop between predictions. This will lead to a less robust model in the case of false positives, but will act as a middle ground in the trade-off between false positives and false negatives. This leads to a total number of 30 predictions per file times 600 files = 18000 predictions.

A sigmoid is applied to the predictions made by the model after training to get it in the range $[0, 1]$. It is then possible to interpret the output as a "probability" or a measure of confidence. The larger the input is into the sigmoid function the closer the output is to 1. The value returned from the sigmoid is checked against a threshold, where 0.95 was used in the final tests.

Performance of the adult detection model is mainly assessed with precision, recall and the false positive rate (Equations (16), (17) and (18) respectively). This is because of the assumption that the number of true negatives will outnumber the number of true positives, false positives and false negatives leading to a high accuracy.

Juvenile individuals

The performance of the juvenile classifier is not assessed with any of the mentioned metrics. This is due to the fact that it should be used to explore the possibility that it could find vocalizations underwater without any prior annotation. Therefore testing of the juvenile was done on the entire Sauer 2015 campaign. It wouldn't be possible with the time given to go through the whole campaign to find and label potential vocalizations. All predictions the model made of the juvenile were instead appended to a .TXT file with the name of the file it made predictions on as well as the time it predicted potential vocalizations. A single test run on the complete campaign took around 14 hours.

No overlap was used in the audio windows used when testing the classifier for the juvenile. This is due to the fact that the classifier should be able to detect all three types of juvenile calls, which varies in length. Using overlap and checking if two consecutive windows are positive predictions would lead to potential misses of the vocalizations due to their short length.

4 Results

This section will introduce the results from the training and testing of a deep learning-based detector and a classifier for the adult and juvenile common spadefoot toad respectively. The main results presented will be from the testing, as applying the trained machine learning models to actual field data is the most interesting. Different metrics for the detector will be presented and will be discussed in greater detail in Section 5.

4.1 Results from the training

Following is a collection of the results gained from training a detector on the adult P. fuscus vocalizations and a classifier that will try to detect the juvenile toad. The main results presented from training will be accuracy and loss plots, with a discussion of the detection of adults and juveniles following in Sections 5.1 and 5.2 respectively.

4.1.1 Detection of adult P. fuscus

The accuracy and loss plots of the model used in detection of the adult common spadefoot toad are shown in Figures 18 and 19 respectively. Epochs are plotted along the x-axis and accuracy and loss are plotted on the y-axis. It can be seen that the loss reaches a minimum on epoch 12, and slowly stabilizes at a slightly higher value after this. The accuracy of the model approaches 100% for the training and stabilizes at around 98.5% in the validation.

The large spike in validation accuracy and loss at epoch 8 in Figures 18 and 19 might come from the fact that the batches in that specific epoch contained several mis-classifications. It could also come from the learning rate being initialized to being too large, but this is not seen as a problem for the remainder of the report as the model stabilizes later in the training.

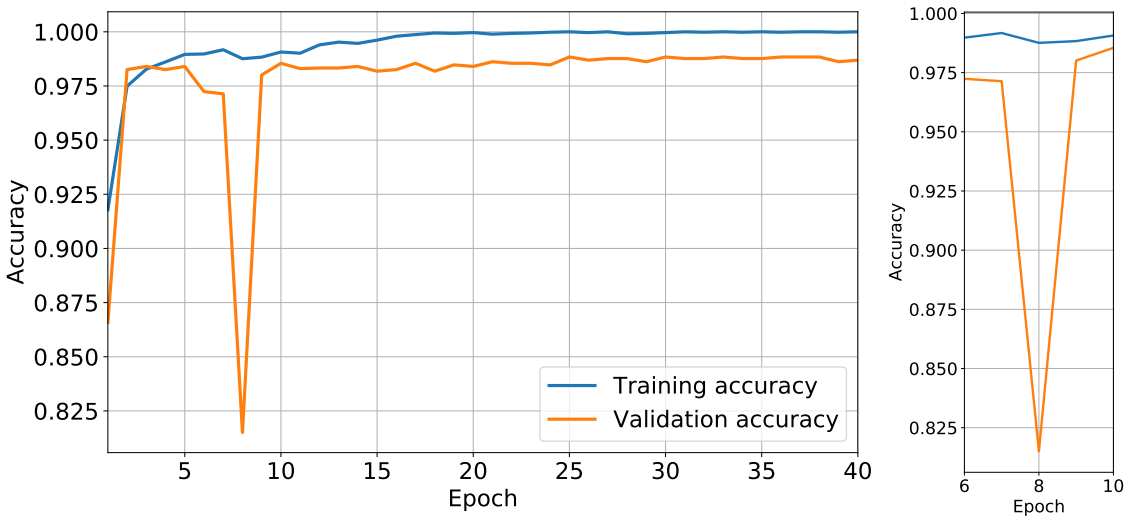


Figure 18: Accuracy from training a model of P. fuscus detection for 40 epochs on the left. The figure on the right shows epochs 6 to 10 where the model dipped in accuracy.

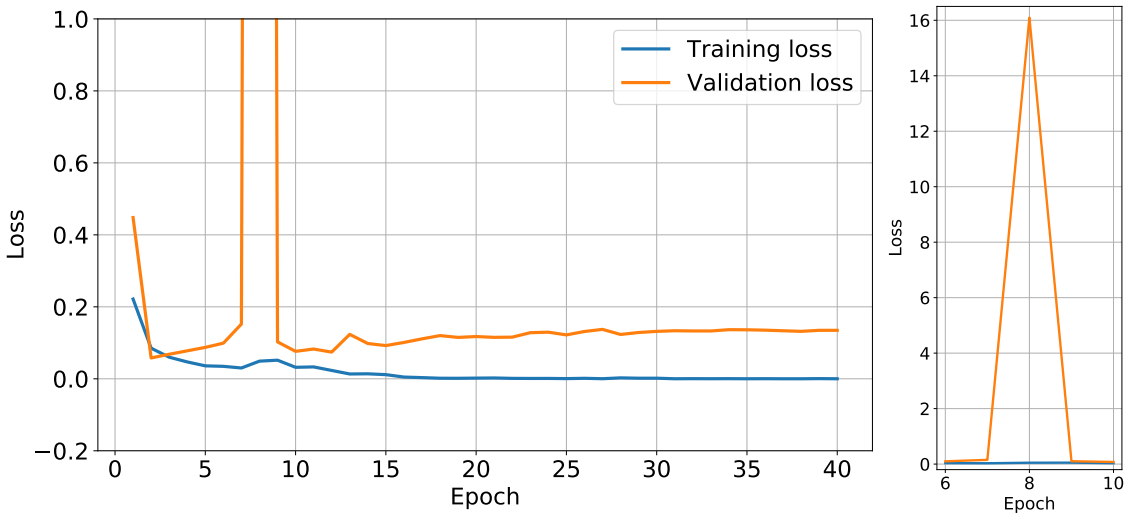


Figure 19: Loss from training a model of adult P. fuscus detection for 40 epochs on the left. The figure on the right shows epochs 6 to 10 where the model spiked in loss.

4.1.2 Detection of juvenile P. fuscus

Figures 20 and 21 shows the accuracy and loss for the classifier trained on juvenile vocalizations respectively. It seems that the training was much more stable when training the multi-label classifier than for the detector as there are no random spikes or drops in neither the loss nor accuracy. Figure 21 shows that the validation loss (in orange) lies close to the training loss and then gradually becomes flatter and flatter while the training loss keeps decreasing. The training accuracy approaches 100% while the validation accuracy stabilizes at around $97\% - 98\%$.

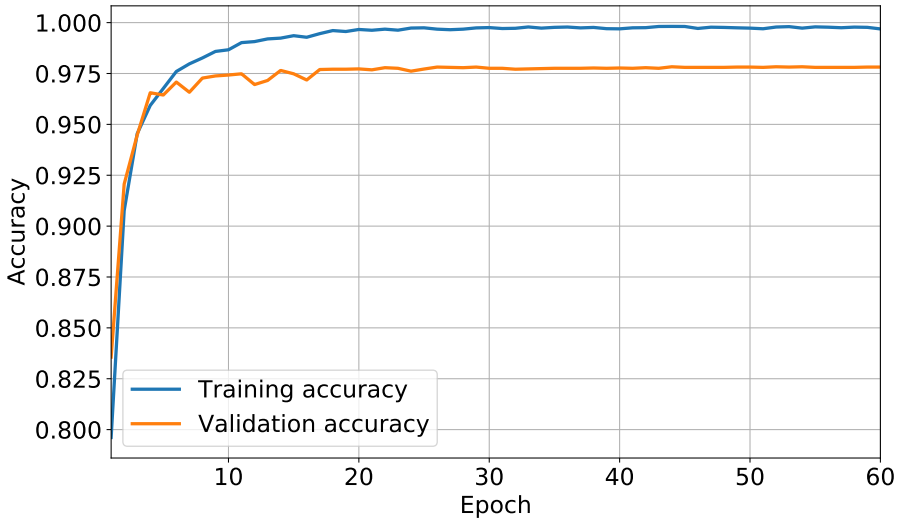


Figure 20: Accuracy from training a model of juvenile P. fuscus classification for 60 epochs.

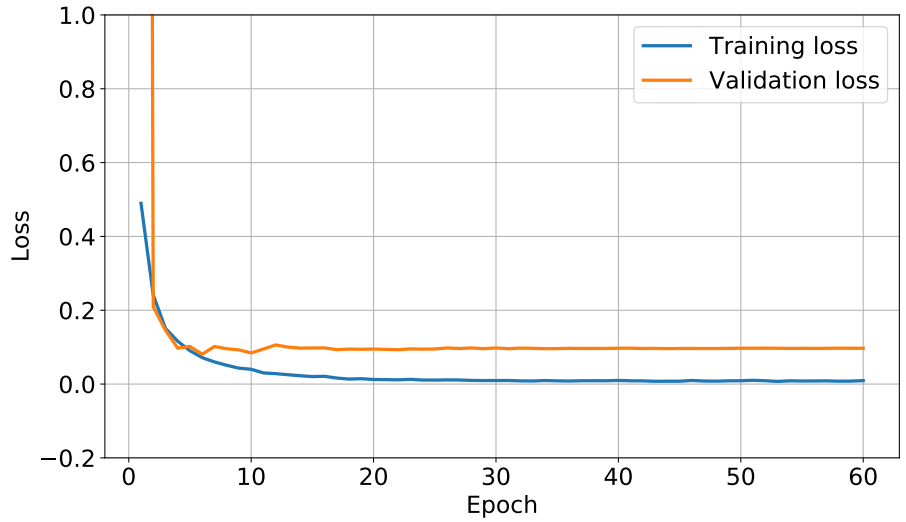


Figure 21: Loss from training a model of juvenile P. fuscus classification for 60 epochs.

4.2 Results from the testing

Any times mentioned when presenting examples from testing are presented in French Winter time (UTC+01). All spectrograms are plotted with time in seconds on the x-axis and frequency in Hz on the y-axis unless otherwise specified. The range of the time is $[0, 1]$ second and frequency range is $[0, 8000]$Hz. In this section the findings of applying the trained models will be presented. Some of the metrics gained from the detector, as well as a collection of false positives and false negatives will be presented. Section 4.3 will present some of the results from testing the detector trained on terrestrial sounds on the underwater recordings to see if it could be possible to detect the juvenile.

4.2.1 Detection of adult P. fuscus

Three different detectors were tested for the P. fuscus, and they differ in the way the testing is done. In the first detector (Test 1), only the labeled instances in the ground truth time were rounded to the nearest 0.5 seconds and added. In the second detector (Test 2), both the time before and after the labeled instance were added as discussed in Section 3.6.1. In the third test (Test 3) there is no overlap between the windows, meaning the test is done on a second-to-second basis. Table 7 shows the different metrics for the three tests. A more detailed discussion on these results will follow in Section 5.1.

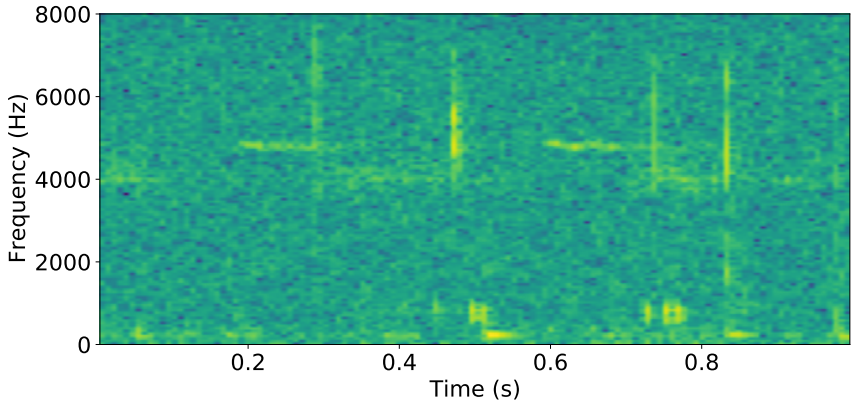
Table 7: The metrics resulting from testing the same model in three different ways. Test 1 was tested with only the labeled ground truth time rounded to the nearest 0.5 seconds. Test 2 was tested with the times both before and after added as discussed in Section 3.6.1. Test 3 was done with no overlap between the windows. TP rate is the rate of true positives (recall).

Test	# TP	# TN	# FP	# FN	Precision	TP rate	FP rate
Test 1	1211	32425	1470	294	45.17%	80.47%	4.34%
Test 2	2506	31500	175	1219	93.47%	67.28%	0.55%
Test 3	1084	16246	395	275	73.29%	79.76%	2.37%

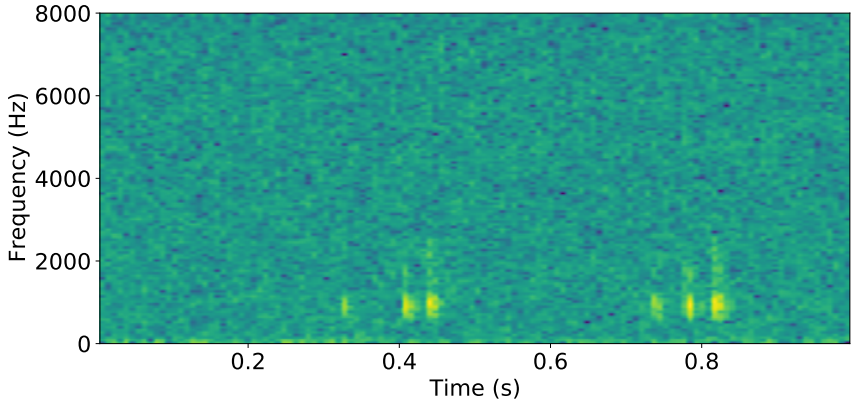
It is apparent from the results in Table 7 that Test 2, using both the time before and after the ground truth time, yields the lowest number of false positives as well as the highest number of true positives. However, there is a trade-off between the number of false positives and the number of false negatives. Test 1 performs better with regards to the number of false negatives and true negatives with a true positive rate of 80.47%, where Test 2 has 67.28%. Test 3 seems to perform somewhere between the other two tests. Unless otherwise stated Test 2 will be the one examples of true/false positives and false negatives are taken from.

4.2.2 True positives in adult detection

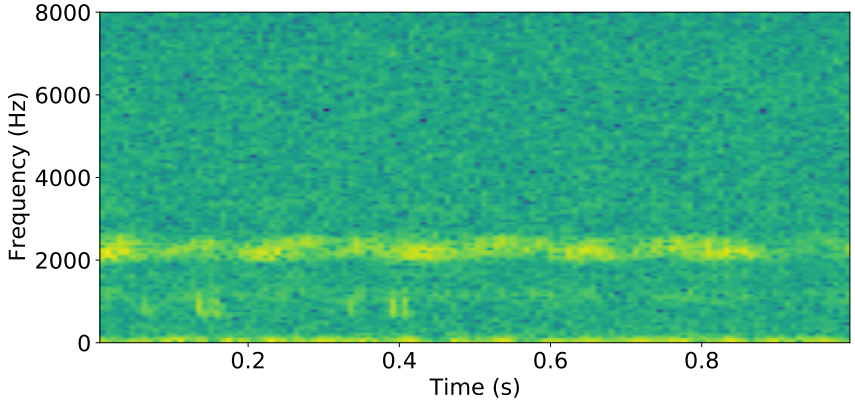
True positives means that the model predicted presence of a vocalization when there are indeed presence of the toad. Examples of some of the true positives in Test 2 are shown in Figure 22. Figure 22a shows an example of a true positive where the model detected the toad where more complex sounds can be seen around it (the notes can be seen at 0.5 and 0.7 seconds around 1kHz). Figure 22b shows true positive predictions where only the toad vocalized, while Figure 22c shows a correct prediction while the European tree frog is vocalizing at around 2kHz.



(a) TP in Mothern 2016, May 16th at 06:00



(b) TP in Sauer 2015, April 7th at 01:00

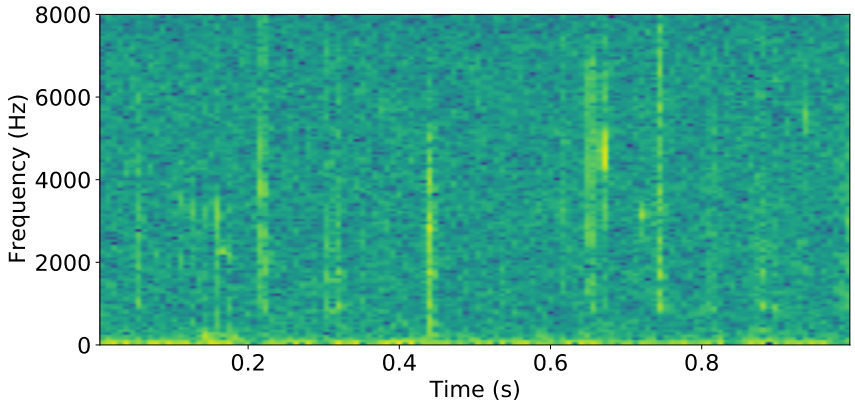


(c) TP in Sauer 2016, May 3rd at 03:00

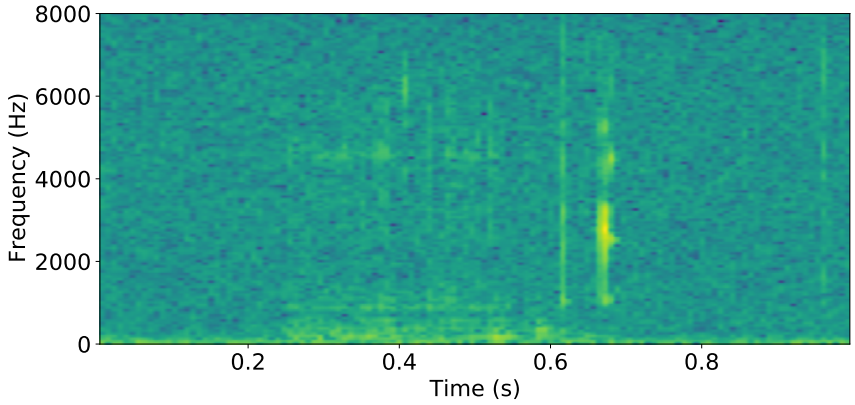
Figure 22: Examples of some of the true positives in Test 2. (a) shows an example of the toad vocalizing with more complex surrounding sounds, (b) shows a prediction where only the toad vocalized, while (c) shows a correct prediction while the European tree frog is vocalizing.

4.2.3 False positives in adult detection

False positives means that the model predicted a vocalization when in fact there were none. Some examples of false positives will be presented, and a discussion of them will follow in Section 5.1. Figure 23 shows examples of false positives, where Figure 23a is from the Sauer 2015 campaign on April 30th at 15:00 and Figure 23b is from the Sauer 2016 campaign on May 11th at 02:00. Theses are clear false positives as they do not contain any vocalizations. Both Figure 23a and Figure 23b shows rain drops hitting the surface of the pond or the hydrophone which gives false positives.



(a) FP in Sauer 2015, April 30th at 15:00.



(b) FP in Sauer 2016, May 11th at 02:00.

Figure 23: Examples of false positives from Test 2. Both Figures (a) and (b) shows rain drops hitting the surface of the pond or the hydrophone which gives false positives.

There are also times when the detector predicts presence, but it is difficult to say whether it is an actual vocalization or just a sound that resembles the advertisement call. An example of this is shown in Figure 24, where a potential vocalization is seen at around 0.3 seconds with frequencies close to 1000Hz. The European tree frog can also be seen vocalizing continuously between 2000Hz and 3000Hz.

It also seems that the detector reports false positives when in fact it should be a true positive. Examples of this can be seen in Figure 25. Both Figure 25a and Figure 25b shows a single note of the advertisement call in the last 0.2 seconds. This might come from the way ground truth times are prepared. More on this can be found in Section 5.1.4.

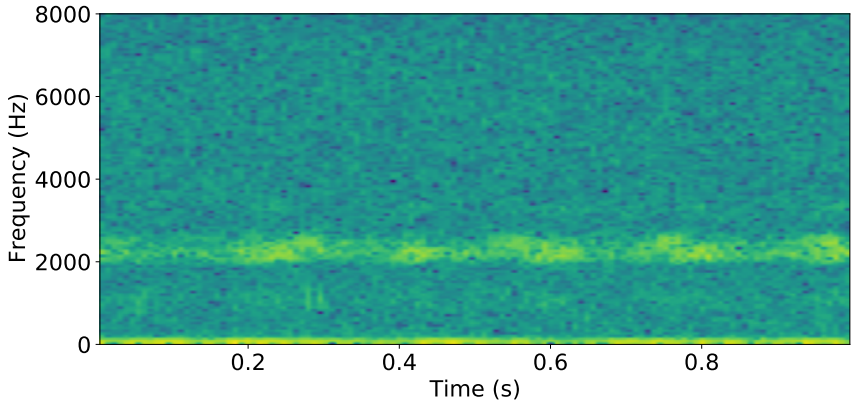
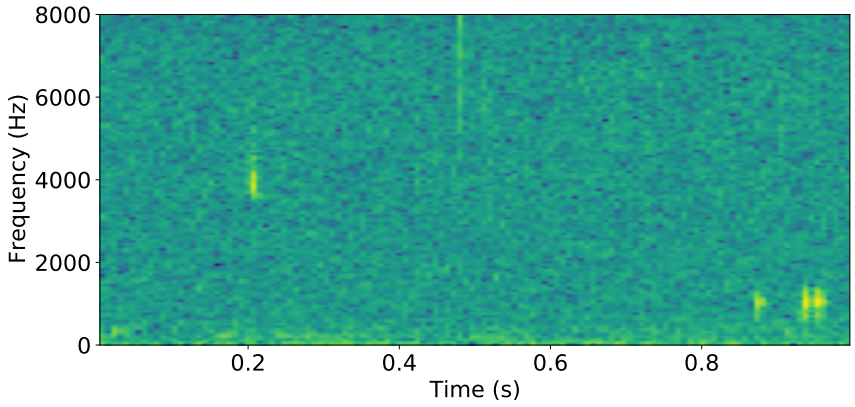
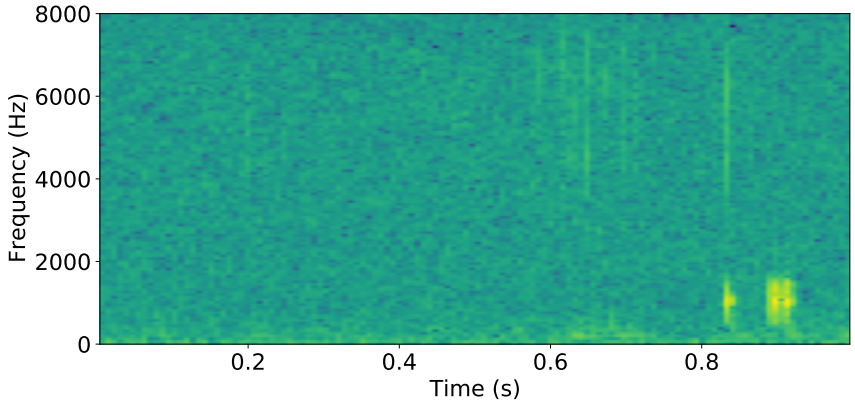


Figure 24: Example of a reported false positive that is difficult to say if it actually is a vocalization or just a sound that resembles the advertisement call. The potential vocalization can be seen at around 0.3 seconds with frequencies around 1000Hz.



(a) FP in Mothern 2016, May 1st at 02:30



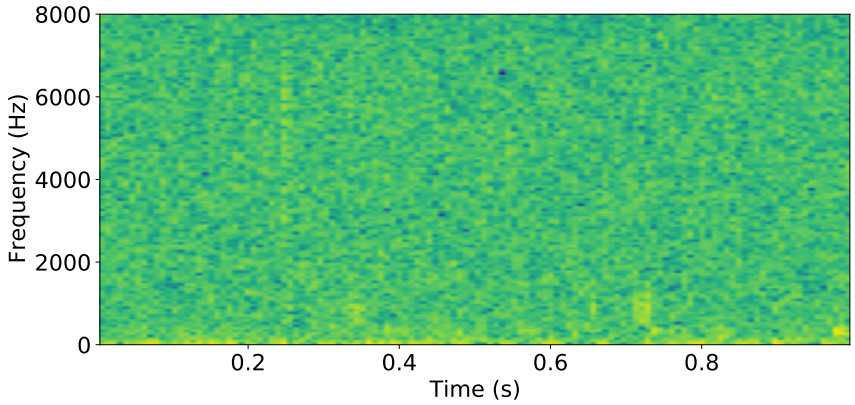
(b) FP in Mothern 2016, May 2nd at 04:00

Figure 25: Examples of times where the model reported a false positive, when in fact clear vocalizations can be seen at the end of the spectrograms of both (a) and (b).

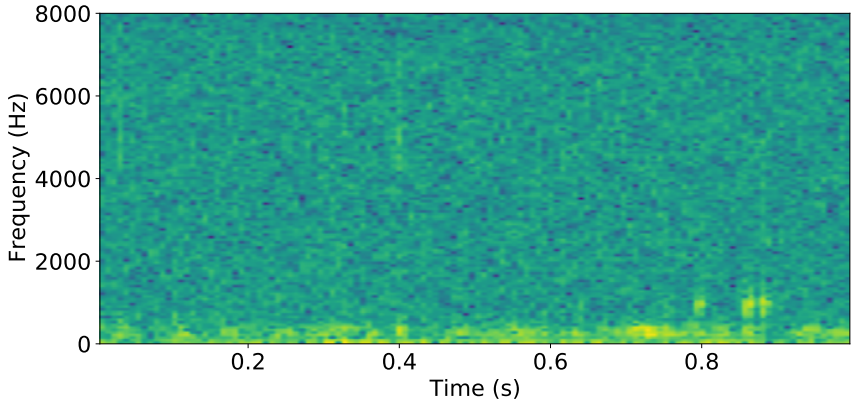
4.2.4 False negatives in the adult detector

A false negative means that the model did not predict a vocalization when in fact there was one. It means that the model missed a vocalization, so going through the false negative predictions should contain some sort of note of the toad. Some examples of false negatives will be presented, and a discussion of them will follow in Section 5.1. Examples of missed vocalizations can be seen in Figure 26. Figure 26a shows a weak vocalization at about 0.7 seconds with frequencies around 1000Hz, while Figure 26b shows a clearer vocalization being missed. The low frequency noise in the latter is found to be scratching noise.

Figure 27 shows some examples of reported false negatives that should in fact be a true negative. These clips contain rain drops and vocalizations of the European tree frog (*Hyla arborea*) at around 2kHz. No clear vocalizations of the P. fuscus can be seen in these windows. Looking closer at the mis-classifications for both these examples there were vocalizations either in the window directly before (Figure 27b) or both before and after (Figure 27a).

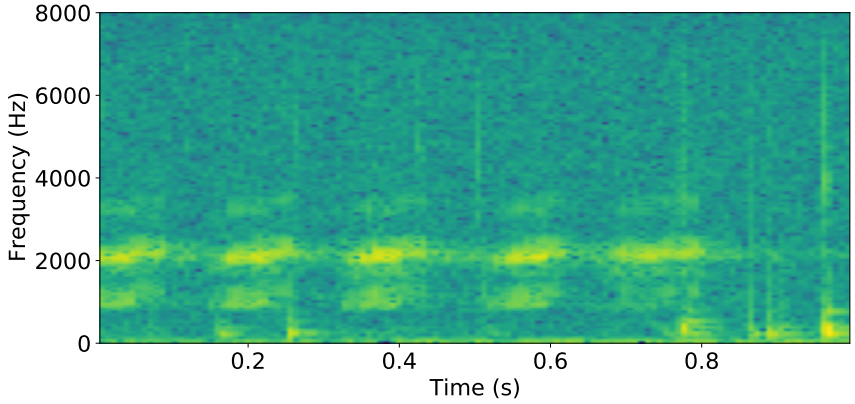


(a) FN in Mothern 2016, Apr 20th at 01:00

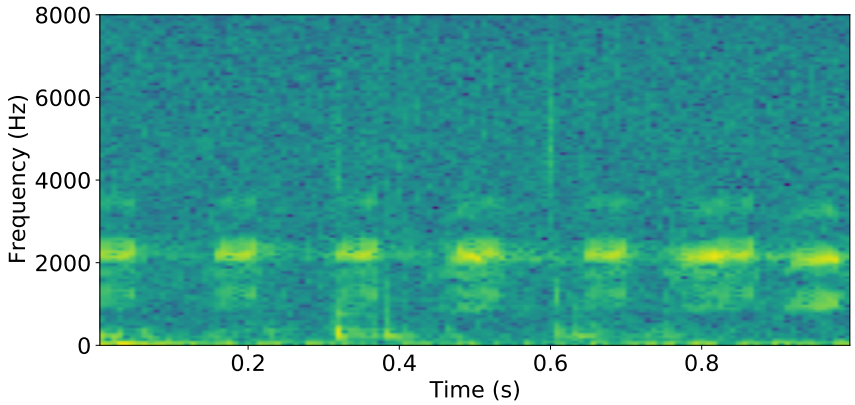


(b) FN in Mothern 2016, May 2nd at 01:00

Figure 26: Examples of missed vocalizations. (a) shows a weak vocalization at about 0.7 second with frequencies around 1000Hz, while (b) shows a clearer vocalization at 0.8 second being missed. The low frequency noise in (b) is found to be scratching noise.



(a) FN in Mothern 2016, May 14th at 00:00



(b) FN in Mothern 2016, May 14th at 00:00

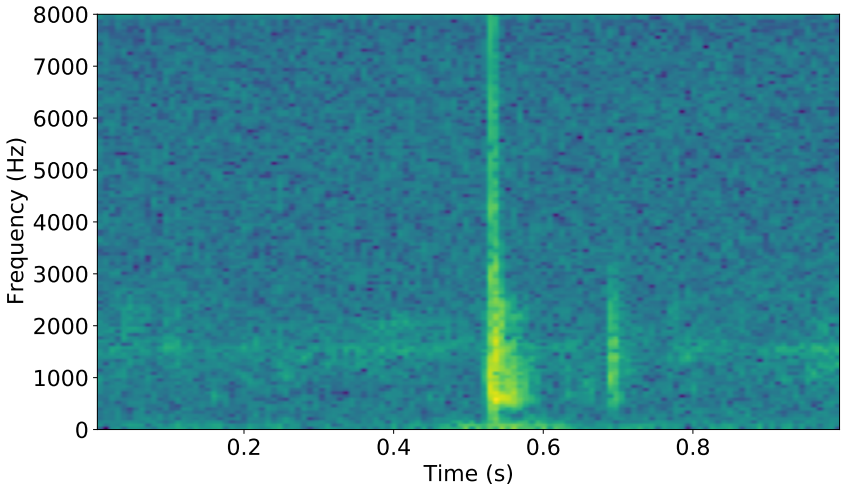
Figure 27: Examples of reported false negatives that should in fact be true negatives. The European tree frog is seen vocalizing at 2kHz.

4.3 Detection of juvenile P. fuscus

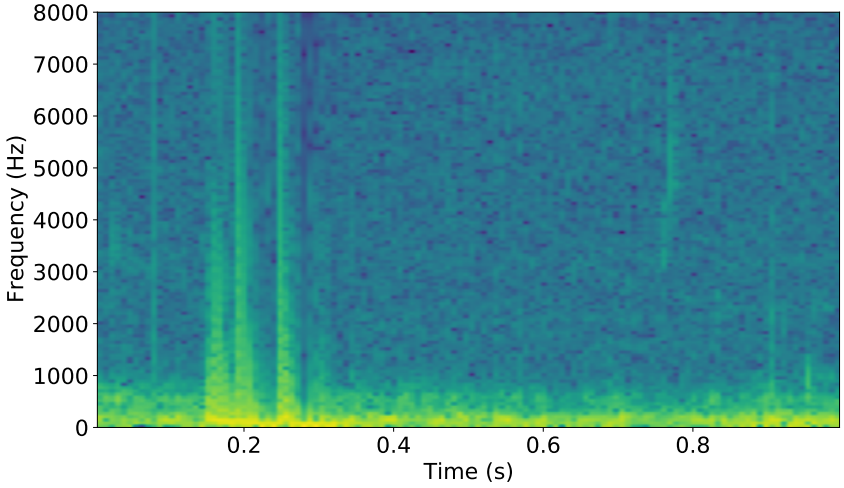
As mentioned in Section 3.6.2 a detection test of the juvenile was done over entire Sauer 2015 campaign. Due to the recordings being recorded during the juvenile's terrestrial phase it was difficult to say whether such a test would return any positive predictions. Therefore any predictions the model made was written to a .TXT file, and consequently checked at the end of the 14 hour testing period.

In the final model it was made 2448 predictions over all 5470 5-minute files in the Sauer 2015 campaign. This means the model did $5470 \cdot 5 \cdot 60 = 1641000$ predictions. In turn this gives a juvenile prediction rate of $2448/1641000 \approx 0.149\%$.

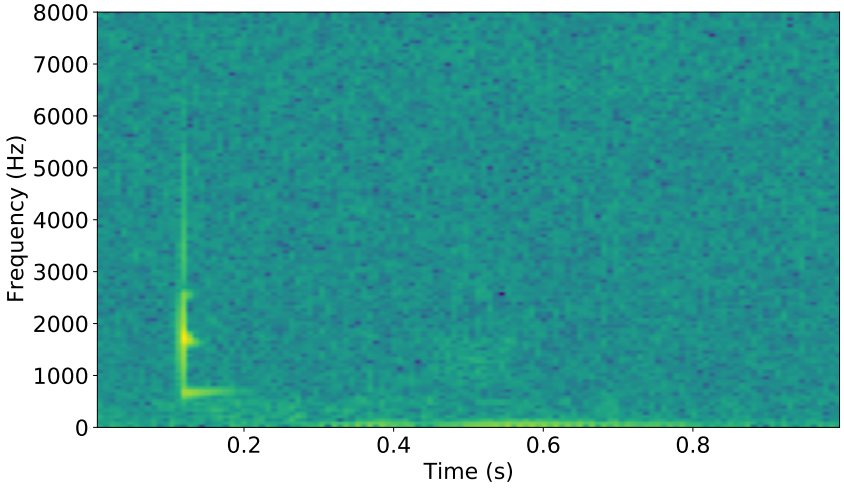
Going through the positive predictions it looks like they are all false positives. Some examples of the false positives are shown in Figure 28. Figure 28a shows a spectrogram of what sounds like something heavy hitting the water surface, Figure 28b sounds like shock noise of some sort while Figure 28c is a water drop hitting the hydrophone.



(a) Falsely classified as juvenile on May 7th at 18:30



(b) Falsely classified as juvenile on May 9th at 11:00



(c) Falsely classified as juvenile on June 30th at 21:30

Figure 28: Examples of some of the false positives in the juvenile test.

5 Discussion

This section will be a discussion of the results presented in Section 4. In Sections 5.1 and 5.2 a discussion on the findings of the detector of adults and classifier of the juvenile is presented respectively. Section 5.1.5 will present a comparison of the results achieved in this thesis with the software detector presented in (Dutilleux and Curé 2020). Lastly in Section 5.3 a discussion on potential future improvements is presented.

The EfficientNet architecture was chosen due to achieving much better performance than earlier ConvNets with the same amount of parameters (such as ResNet and Inception) at a much lower computational cost. The EfficientNetB3 architecture specifically was chosen as the final model due to lack of processing power for the larger ones. As the models get larger there are diminishing returns on the accuracy, so the B3 was deemed sufficient for the models developed in this report.

5.1 Testing of the adult detection

The discussion of the detection of the adult P. fuscus will be focused on the testing of the finished model. Section 5.1.1 will be a short discussion of the true positives of the tests, Section 5.1.2 is a discussion on the false positives and Section 5.1.3 will present a discussion of the false negatives. In Section 5.1.4 a reflection on the way the ground truth is extracted from the label files follows, and lastly a comparison with the software detector from (Dutilleux and Curé 2020) is presented in Section 5.1.5.

Test 1 and Test 2 of the detector was done on the ground truth files with the an overlap of 0.5 seconds. This overlap choice comes from the fact that normal P. fuscus vocalizations on average lasts about 0.5 seconds as discussed in Section 2.1.1. If two overlapping windows then returns positive prediction it would mean there is a higher probability that it was actually a vocalization. In this way false positives were avoided in a much higher rate if this was not the case.

Test 3 was done with no overlap between predictions. This gave rise to a larger proportion of false positives than for Test 2, but proved to be a middle ground for precision and recall. Since the ground truth in Test 3 was only rounded to the nearest second, it would mean that some of the vocalization might fall outside of the ground truth time. This could then in turn lead to the model counting false positives when there are in fact a vocalization in the prediction. The final number of false positives in this test could in reality be a bit lower than reported.

5.1.1 True positives in the adult detector

Figure 22 shows some examples of true positive predictions of the adult specimen detector. Figure 22a shows a vocalization occurring while droplets or other clicking sounds are present. Figure 22b shows a more "clean" prediction of the detector as only the P. fuscus can be seen vocalizing. The last example, shown in Figure 22c, contains the European tree frog vocalizing at the same time as the P. fuscus. From these examples, it seems the detector is robust against different kinds of shock noises and other species vocalizing.

5.1.2 False positives in the adult detector

Test 1 performed the worst when it comes to false positives, as seen in Table 7, having 8.4 times the number of false positives that Test 2 has.

Figure 23 shows false positives that does not contain vocalizations as expected. Listening to the first clip (Figure 23a) and looking at the spectrogram it seems that the detector gave a prediction for rain since only droplets hitting the pond surface/hydrophone could be heard. In the second clip (Figure 23b) it seems that this prediction was also made due to a water droplet hitting the hydrophone.

Sometimes it was found difficult to conclude whether a detection should have been counted as a false positive, as seen in Figure 24. In this case the amplitude of the European tree frog's call is larger than that of the potential vocalization, so it is both difficult to hear and see in the spectrogram whether this was a vocalization or not.

As seen in Figure 25 the model made predictions of the P. fuscus that are counted as false, when they are in fact true. This comes from the way the ground truth times are extracted. As an example, look to Figure 16 on the left side of the vocalization. The window that starts at $t = 1$ stops right after the toad has started vocalizing. This means that the model might predict a positive, but the way the model is tested will lead to this counting as a false positive. What this could mean for the model is that it in reality it performs slightly better with regards to false positives than actually reported.

5.1.3 False negatives in the adult detector

Test 2 gave the most false negative predictions as seen in Table 7, while Test 3 gave the fewest (this is seen as a consequence of only making about half the number of predictions compared to the first two tests).

Sometimes the toad only makes a single note. Due to the model only keeping track of the prediction times if two or more consecutive windows are positive this could give rise to false negatives. As this is relatively uncommon however, this is not believed to be the main cause of false negatives. Only 3 cases of this was found in the ground truth data.

Figure 26 shows some clear examples of vocalizations that were missed. Figure 26a contained a weak vocalization, which might have been the main cause for the model to miss it. It could be that the dataset contains few weak vocalizations, so that when performing a train-test split the training data does not contain enough weak vocalizations to be able to generalize well for weak signals. Figure 26b shows a clearer vocalization than the previous one, but this signal contains low frequency noise. The missed vocalization can come from a lack of training examples containing vocalizations with low frequency noise. But it could also come from the training examples that does *not* contain vocalizations.

As false negatives are defined as missed detections, it would be expected that all false negatives should contain part of or entire vocalizations. This is not the case, as is evident from Figure 27. These windows did not contain the P. fuscus, so they should be counted as a true negative and not a false negative. This comes as a consequence of adding the previous and following window of the ground truth one. In some cases the time of the vocalization might be rounded down. Adding the previous window will then lead to that window not containing part of the vocalization, thus being counted as a false negative when in fact it is a true negative. The other case of rounding up and the following window

not containing a vocalization can also happen with this way of testing. Studying the false positives in Figure 27 confirms that adding both the previous and next window to the ground truth added windows that did not contain any vocalizations. This is a clear limitation of doing the testing this way.

5.1.4 Extraction of the ground truth

As discussed above it seems from both the false negatives and false positives that the way the ground truth times are extracted can be one of the main reasons that the detector behaves so differently in the two tests (presented in Table 7). This is not surprising due to the way the actual vocalizations were labeled. Rounding to the nearest discrete time step will result in some information getting lost, and adding both the previous and next time steps leads to too many windows being produced in turn increasing the number of false negatives.

5.1.5 Comparison to software detection

The software detector presented in (Dutilleux and Curé 2020) achieved reliable results in detecting the adult common spadefoot toad. The aim of that detector was to have a low enough false positive rate so as to minimize human post-processing time and a high enough true positive rate so that it is impossible to miss presence during a complete breeding season. With this in mind the resulting false positive rates were lower than 1.5% and true positive rates ranging from 53% to 73%.

The detector presented in this thesis achieved different results based on the way it was tested. It was tested on the same ground truth files that the software detector was tested on. The software detector was tested on a file-to-file basis, while the detector presented in this report were tested on a second-to-second basis. The resulting precision ranged from 45.17% to 93.47% while the true positive rates ranged from 67.28% to 80.47%, outperforming the software detector for the largest TP rate. The best model with regards to false positive rates achieved 0.55%, meaning that of all predictions that should have been negative, only 0.55% were classified as positive. But the FP rate could in fact be even lower in reality as discussed in Section 5.1.2, due to the fact that some of the false positives were actually true positives.

From the results presented in Table 7 it can seem that the detector with the lowest false positive rate (only 1/3 of the FP rate in the software detector) performed almost as well as the software detector with regards to true positives. The software detector achieved 73% while the deep learning-based one achieved 67.28%.

5.2 Detection of juvenile P. fuscus

5.2.1 Training of juvenile classifier

As seen in Figure 13 and Table 4 it is apparent that the data used in training the classifier is imbalanced. The juvenile only had a total count of 280 cases, while the largest class containing bird songs had 1213 cases. This could lead to the juvenile not being represented enough while training and the model might not learn to generalize the juvenile calls.

5.2.2 Testing of juvenile classifier

Testing possible detections of the juvenile specimen yielded no correct predictions. This might come from a few reasons. From studying the false positives of the juvenile classifier, it can look like it mistakes loud shock sounds, turbulent sounds or even rain droplets hitting the hydrophone for the juvenile vocalizations as shown in Figure 28. It could therefore be the case that the classifier has learned that the juvenile means loud and sudden noises.

Due to the fact that no true positive predictions were made, it might come from the fact that the data that was used was recorded on land during the juvenile's terrestrial phase. The recordings of the juvenile used to train the classifier were taken in a controlled environment and the specimens vocalized when fed (Hagen et al. 2016). The original recordings were also taken at 44.1kHz, so to make it so that the correct frequency range was used the signal was downsampled to 16kHz, so the spectrograms might have lost some information that is crucial to the vocalizations. In addition not much pre-processing was done to the audio other than resampling, transforming to spectrogram and resizing.

Due to the different vocalizations being spread over a large frequency span, it might be that the vocalizations have been masked in other sounds in the Sauer 2015 campaign recordings.

It could even be that the juvenile spadefoot toad does not even vocalize underwater at all, and that is the reason that the model made no true positive predictions. However it should be noted that the results presented here is not a conclusion that the juvenile does not vocalize underwater, as it still might be the case.

The aim of the juvenile detection was to explore the possibility of using deep learning-based methods to find if it vocalizes underwater. Since the detection did not predict any true positive cases, it is not possible to make a conclusion on whether this is the case or not.

5.3 Future work

In this section a discussion on possible future work and improvements of the results will follow.

5.3.1 Possible improvements of the data

The performance of the models are heavily dependent on how good it is to generalize for the data. Adding more data points would lead to the model getting a better statistical understanding of the vocalizations. Expanding the dataset can be done in a couple of different ways. The first is to label more data, while the second one would be to perform data augmentation. Labeling more data is very straightforward, but time-consuming. The other way is to artificially expand the dataset with more points by adding augmented versions of some of the already labeled data. Augmentations that could be relevant for audio is slight time and frequency shifts, changing the speed of the audio slightly up or down and adding Gaussian noise.

Loading the data in different ways could be a potential improvement. Because the input to the pre-trained EfficientNet are RGB images, it could be beneficial to first save the spectrograms as for example PNG images and then loading the images in the DataLoader. If this is done then a normalization transform must be applied to the image as the images that trained the model are expected to be normalized in each of the three color channels. It could also be interesting to see how a detector model would perform on a stream of data being loaded, i.e. applying the model to real-time applications.

The image size might also be a point worth looking into. EfficientNet uses a size of 224×224 so that was the one chosen in the final training. However, increasing the size could be a possible improvement to the system as the vocalizations of the P. fuscus has a very fine structure in the time dimension. This would however give a trade-off in performance, as doubling the image size in both directions would lead to a quadrupling in the amount of data. Sufficient processing power is then needed, and due to a lack of computer memory it was not tried out for this thesis.

Other types of pre-processing could also be worth trying out. For example (Noda, Travieso and Sánchez-Rodríguez 2015) used a combination of Mel and Linear Frequency Cepstral Coefficients (MFCC & LFCC) with Support Vector Machines, Hidden Markov Models and random forests to achieve classification rate of $95.38\% \pm 5.05$. (Strout et al. 2017) used a spectrogram as input to a convolutional neural network. The network acts as feature extraction, and these features are then fed into a Support Vector Machine, achieving a mean classification accuracy of 73.57%. (Huzaifah 2017) tried different time-frequency representations of sound events and found that Mel-STFT spectrograms performed generally better than linear-STFT ones.

Specifically for the classifier other lengths of input data could be tried. This is due to some of the classes making sounds for longer than a second. Potentially Recurrent Neural Networks could be used for a dynamic-length input, which have been used in speech applications and sound event detection (see for example (Adavanne and T. Virtanen 2017; Li and Wu 2014; Lim et al. 2018)).

5.3.2 Adult detection

Because some the false positives seemingly should be true positives it could mean the detector should perform better than reported. Therefore it could be beneficial to re-label the ground truth on a window-to-window basis (1 second windows with 0.5 second hops) to get an even better estimate of the actual performance of the model. The current way the ground truth is extracted from the label files leads to a trade-off between false negatives and false positives. As mentioned in Section 5.1, the rounding of the ground truth time to the nearest 0.5 second and then adding both the previous and following window as ground truth times leads to potential false negatives actually being true negatives and false positives actually being true positives. If the data is not re-labeled it could be beneficial to take a closer look into the way the ground truth times are extracted when testing the model.

Due to the fact that the tests of the detector were only performed on similar audio data as the data the model was trained on, it is difficult to say how it would perform on data recorded at other sites with other equipment. The difference in background noise could give rise to different activations in a neural network, so a possible solution to this could be to

5.3.3 Juvenile detection

The juvenile detection could be improved quite a lot, given that the data used is recorded on land. A possible way to achieve more reliable results could be to mask the terrestrial sounds with the background noise present at the Mothern and Sauer sites. Other kinds of augmentations could be made to make the sounds closer to the test set. The amplitude in some of the clips were found to be a lot larger than others, so scaling the amplitudes could make the sounds closer to the soundscape at Mothern and Sauer.

The bad performance could also come from the fact that the amount of data used for the juvenile class is underrepresented when training, so that only a few examples were used per training batch. As mentioned at the start of this section increasing the number of data points would help the model generalize better for the given class.

As the adult P. fuscus detector was designed as a single-output classifier, this could also be done for the juvenile. This means that a new dataset could be designed with a 1-0 output. If the design of the juvenile detector is to be remade, adding more data would be beneficial.

6 Conclusion

This master thesis has presented the results from developing a deep learning-based detector of the adult common spadefoot toad (*Pelobates fuscus*) based on its advertisement call. Using a pre-trained architecture, EfficientNet (Tan and Le 2020), the detector with the fewest false positives achieved a precision of 93.47% with a corresponding recall/true positive rate of 67.28% and a false positive rate of 0.55%.

Three different tests were made on a prepared ground truth dataset to see how the model performed. The first test were made with overlapping 1-second windows (50% overlap) and rounding the ground truth time to the nearest 0.5 second- This test achieved a precision of 45.17%, recall of 80.47% and a false positive rate of 4.34%. A second test was made with 50% overlap, but this time both the time before and after the ground truth time were added. The test achieved a precision of 93.47%, recall of 67.28% and a false positive rate of 0.55%. Both the first two test were made more robust against false positives by checking if two consecutive predictions were positive. A last test was tried out with no overlap in windows (one and one second separately at a time) and the ground truth time was rounded to the nearest second. This test achieved a precision of 73.29%, recall of 79.76% and a false positive rate of 2.37%, being a middle ground of false positives and false negatives to the two other tests. A comparison with the software detector developed by (Dutilleux and Curé 2020) was performed, finding that the deep learning-based detector outperformed the software one with a three times lower false positive rate for comparable true positive rates.

In addition to a detector of the adult, a method to detect the juvenile individual was also tried out. It has been shown in (Hagen et al. 2016) that the juvenile vocalizes during its terrestrial phase, so training a deep learning model to see if it could detect underwater vocalizations was done. The resulting predictions that were made were all found to be false positives, so it is concluded that the model trained on terrestrial vocalizations can *not* be used to find juveniles vocalizing underwater. The audio clips of the juvenile used to train the model have different background noise levels than the recordings used to test, and it seems that loud shock noises gives some of the same activations in the neural network that the vocalizations do. However, the results in this thesis is not a definite conclusion that the juvenile specimen does not vocalize underwater. More research should therefore be done in the potential juvenile underwater vocalizations.

This thesis has shown that it is possible to get reliable results both with regards to false positives and false negatives when using a deep learning-based detector on the advertisement call of the common spadefoot. Therefore it is possible to apply the detector to long-term recordings in possible habitats for conservation purposes to find the toad.

References

Adavanne, Sharath and Tuomas Virtanen (2017). 'Sound event detection using weakly labeled dataset with stacked convolutional and recurrent neural network'. In: *CoRR* abs/1710.02998. URL: http://arxiv.org/abs/1710.02998.

Alonso, Jesús B. et al. (2017). 'Automatic anuran identification using noise removal and audio activity detection'. In: *Expert Systems with Applications* 72, pp. 83–92. DOI: 10.1016/j.eswa.2016.12.019.

Amazon (2021). *Amazon Machine Learning: Binary classification*. URL: https://docs.aws.amazon.com/machine-learning/latest/dg/binary-classification.html (visited on 8th June 2021).

AmphibiaWeb (2020). *Pelobates Fuscus on AmphibiaWeb*. University of California, Berkeley, CA, USA. URL: https://amphibiaweb.org/species/5270 (visited on 31st May 2021).

Amphiconsult (2015). *DRAGONLIFE – Securing Leucorrhinia pectoralis and Pelobates fuscus in the northern distribution area in Estonia and Denmark (2010-2015) LIFE08 NAT/EE/000257*. URL: https://amphi.dk/projekter/eu-projekter/dragonlife-securing-leucorrhinia-pectoralis-and-pelobates-fuscus-in-the-northern-distribution-area-in-estonia-and-denmark/ (visited on 25th May 2021).

Apache Software Foundation (2004). *Apache License 2.0*. URL: https://www.apache.org/licenses/LICENSE-2.0 (visited on 30th May 2021).

Audacity Team (2020). *Audacity(R): Free Audio Editor and Recorder [Computer application]. Version 2.4.2*. URL: https://audacityteam.org/ (visited on 31st May 2021).

Bendersky, Eli (2018). *Depthwise separable convolutions for machine learning*. URL: https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/ (visited on 30th May 2021).

Bridges, Andrew S. and Michael F. Dorcas (2000). 'Temporal Variation in Anuran Calling Behavior: Implications for Surveys and Monitoring Programs'. In: *Copeia* 2000.2, pp. 587–592. DOI: https://doi.org/10.1643/0045-8511(2000)000[0587:TVIACB]2.0.CO;2.

Brownlee, Jason (2017). *What is the Difference Between a Parameter and a Hyperparameter?* URL: https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/ (visited on 5th June 2021).

— (2020). *Multi-Label Classification with Deep Learning*. URL: https://machinelearningmastery.com/multi-label-classification-with-deep-learning/ (visited on 13th June 2021).

— (2021). *How to Choose an Activation Function for Deep Learning*. URL: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/ (visited on 5th June 2021).

Chen, John (2020). *An updated overview of recent gradient descent algorithms*. URL: https://johnchenresearch.github.io/demon/ (visited on 5th June 2021).

Creative Commons (2021). *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Germany (CC BY-NC-SA 3.0 DE) License*. URL: https://creativecommons.org/licenses/by-nc-sa/3.0/de/deed.en (visited on 28th May 2021).

Discovery of Sound in the Sea (DOSITS) (2020). *Masking in Mammals*. URL: https://dosits.org/animals/effects-of-sound/potential-effects-of-sound-on-marine-mammals/masking-in-mammals/ (visited on 9th June 2021).

Duchi, John, Elad Hazan and Yoram Singer (2011). 'Adaptive subgradient methods for online learning and stochastic optimization'. In: *Journal of Machine Learning Research* 12.Jul, pp. 2121–2159.

Dutilleux, Guillaume and Charlotte Curé (2020). 'Automated acoustic monitoring of endangered common spadefoot toad populations reveals patterns of vocal activity'. In: *Freshwater Biology* 65.1, pp. 20–36. DOI: https://doi.org/10.1111/fwb.13111.

European Council (1992). *COUNCIL DIRECTIVE 92/43/EEC*. URL: https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CONSLEG:1992L0043:20070101:EN:pdf (visited on 21st May 2021).

— (2006). *Guidance document on the strict protection of animal species of Community interest under the Habitats Directive 92/43/EEC*. URL: https://ec.europa.eu/environment/nature/conservation/species/guidance/pdf/guidance_en.pdf (visited on 21st May 2021).

— (2021). *The Habitats Directive*. URL: https://ec.europa.eu/environment/nature/legislation/habitatsdirective/index_en.htm (visited on 21st May 2021).

Ghoneim, Salma (2019). *Accuracy, Recall, Precision, F-Score & Specificity, which to optimize on?* URL: https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124 (visited on 27th May 2021).

Gilon, Yosefa et al. (2021). 'Optimization of neural networks'. In: *CS231n: Convolutional Neural Networks for Visual Recognition*. Stanford course on Convolutional Neural Networks. URL: https://cs231n.github.io/optimization-1/.

Goodfellow, Ian, Yoshua Bengio and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Google (2020a). *Classification: Accuracy*. URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy (visited on 27th May 2021).

— (2020b). *Classification: Precision and Recall*. URL: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall (visited on 26th May 2021).

Hagen, Leonie ten et al. (Sept. 2016). 'Vocalizations in juvenile anurans: Common spadefoot toads (Pelobates fuscus) regularly emit calls before sexual maturity'. In: *Die Naturwissenschaften* 103, p. 75. DOI: 10.1007/s00114-016-1401-0.

Harris, Charles R. et al. (Sept. 2020). 'Array programming with NumPy'. In: *Nature* 585.7825, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

Hill, Andrew P. et al. (2019). 'AudioMoth: A low-cost acoustic device for monitoring biodiversity and the environment'. In: *HardwareX* 6, e00073. ISSN: 2468-0672. DOI: https://doi.org/10.1016/j.ohx.2019.e00073. URL: https://www.sciencedirect.com/science/article/pii/S2468067219300306.

Höbel, Gerlinde (2017). 'Social facilitation is a better predictor of frog reproductive activity than environmental factors'. In: *Biotropica* 49.3, pp. 372–381. DOI: https://doi.org/10.1111/btp.12437. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/btp.12437.

Hu, Jie, Li Shen and Gang Sun (2019). 'Squeeze-and-Excitation Networks'. In: *CoRR* abs/1709.01507. arXiv: 1709.01507. URL: https://arxiv.org/abs/1709.01507.

Huzaifah, Muhammad (2017). 'Comparison of Time-Frequency Representations for Environmental Sound Classification using Convolutional Neural Networks'. In: *CoRR* abs/1706.07156. arXiv: 1706.07156. URL: http://arxiv.org/abs/1706.07156.

Ioffe, Sergey and Christian Szegedy (2015). 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: *CoRR* abs/1502.03167. arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

IUCN (2021). *IUCN Red List: List of endangered species*. URL: https://www.iucnredlist.org/ (visited on 6th June 2021).

Jones, Julia et al. (Feb. 2013). 'The 'why', 'what' and 'how' of monitoring for conservation'. In: *Key Topics in Conservation Biology 2*, pp. 327–343. DOI: 10.1002/9781118520178.ch18.

Khoshdeli, Mina, Richard Cong and Bahram Parvin (Feb. 2017). 'Detection of Nuclei in H&E Stained Sections Using Convolutional Neural Networks'. In: *IEEE International Conference on Biomedical Health Informatics*.

Kingma, Diederik P. and Jimmy Ba (2017). 'Adam: A Method for Stochastic Optimization'. In: *CoRR* abs/1412.6980. arXiv: 1412.6980. URL: https://arxiv.org/abs/1412.6980.

Krizhevsky, Alex and Geoffrey Hinton (2010). 'Convolutional Deep Belief Networks on CIFAR-10'. In: URL: http://www.cs.utoronto.ca/~kriz/conv-cifar10-aug2010.pdf.

Lapp, Sam et al. (Feb. 2021). 'Automated detection of frog calls and choruses by pulse repetition rate'. In: *Conservation Biology* n/a.n/a. DOI: https://doi.org/10.1111/cobi.13718. URL: https://conbio.onlinelibrary.wiley.com/doi/abs/10.1111/cobi.13718.

Li, Xiangang and Xihong Wu (2014). 'Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition'. In: *CoRR* abs/1410.4281. URL: http://arxiv.org/abs/1410.4281.

Lim, Hyungui et al. (May 2018). 'RARE SOUND EVENT DETECTION USING 1D CONVOLUTIONAL RECURRENT NEURAL NETWORKS'. In: *Detection and Classification of Acoustic Scenes and Events (DCASE)*.

Linke, Simon et al. (Mar. 2018). 'Freshwater ecoacoustics as a tool for continuous ecosystem monitoring'. In: *Frontiers in Ecology and the Environment* 16. DOI: 10.1002/fee.1779.

Masters, Dominic and Carlo Luschi (2018). 'Revisiting Small Batch Training for Deep Neural Networks'. In: *CoRR*. eprint: 1804.07612. URL: http://arxiv.org/abs/1804.07612.

McFee, Brian et al. (July 2020). *librosa/librosa: 0.8.0*. Version 0.8.0. DOI: 10.5281/zenodo.3955228. URL: https://doi.org/10.5281/zenodo.3955228.

Mohri, Mehryar, Afshin Rostamizadeh and Ameet Talwalkar (2018). *Foundations of Machine Learning*. 2nd ed. MIT Press.

Müller, Burkhard (1984). 'Bio-akustische und endokrinologische Untersuchungen an der Knoblauchkröte Pelobates fuscus fuscus (Laurenti, 1768)'. In: *SALAMANDRA - German Journal of Herpetology* 20, pp. 121–142.

Noda, Juan J., Carlos M. Travieso and David Sánchez-Rodríguez (2015). 'Methodology for automatic bioacoustic classification of anurans based on feature fusion'. In: *Expert Systems with Applications* 50, pp. 100–106. DOI: 10.1016/j.eswa.2015.12.020.

Nyström, Per et al. (2002). 'The declining spadefoot toad Pelobates fuscus: calling site choice and conservation'. In: *Ecography* 25, pp. 488–498. DOI: 10.1034/j.1600-0587.2002.250411.x.

Paszke, Adam et al. (2019). 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pedregosa, F. et al. (2011). 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Pico.net (2021). *What is a False Positive Rate?* URL: https://www.pico.net/kb/what-is-a-false-positive-rate (visited on 27th May 2021).

Prince, Peter et al. (2019). 'Deploying Acoustic Detection Algorithms on Low-Cost, Open-Source Acoustic Sensors for Environmental Monitoring'. In: *Sensors* 19.3. ISSN: 1424-8220. DOI: 10.3390/s19030553. URL: https://www.mdpi.com/1424-8220/19/3/553.

Python Software Foundation (2020). *Python Language, version 3.8.5*. URL: http://www.python.org (visited on 31st May 2021).

Rannap, Riinu et al. (Dec. 2015). 'Geographically varying habitat characteristics of a wide-ranging amphibian, the Common Spadefoot Toad (Pelobates fuscus), in Northern Europe'. In: *Herpetological Conservation and Biology* 10, pp. 904–916.

Reback, Jeff et al. (Apr. 2021). *pandas-dev/pandas: Pandas 1.2.4*. Version v1.2.4. DOI: 10.5281/zenodo.4681666. URL: https://doi.org/10.5281/zenodo.4681666.

Ripley, Brian D. (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press. DOI: 10.1017/CBO9780511812651.

Rogers, Alex and Davide Zilli (2021). *The New Forest Cicada Project*. URL: http://www.newforestcicada.info/ (visited on 6th June 2021).

Ruder, Sebastian (Jan. 2016). *An overview of gradient descent optimization algorithms*. URL: https://ruder.io/optimizing-gradient-descent/index.html (visited on 31st May 2021).

Rumelhart, David E., Geoffrey E. Hinton and Ronald J. Williams (1986). 'Learning representations by back-propagating errors'. In: *Nature* 323, pp. 533–536.

Sandler, Mark et al. (2019). 'MobileNetV2: Inverted Residuals and Linear Bottlenecks'. In: *CoRR* abs/1801.04381. arXiv: 1801.04381. URL: https://arxiv.org/abs/1801.04381.

Schneider, Hans (1966). 'Die Paarungsrufe Einheimischer Froschlurche *Mating calls of the native frogs* (DISCOGLOSSIDAE, PELOBATIDAE, BUFONIDAE, HYLIDAE)'. In: *Zeitschrift für Morphologie und Ökologie der Tiere* 57, pp. 119–136.

Seglie, Daniele, Andrea Gauna and Cristina Giacoma (Jan. 2013). 'Description of the male advertisement call of Pelobates fuscus insubricus (Anura, Pelobatidae), with general notes on its acoustic repertoire'. In: *Bulletin de la Société Herpétologique de France* 145-146, pp. 61–72.

Srivastava, Nitish et al. (2014). 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting'. In: *Journal of Machine Learning Research* 15, pp. 1929–1958. URL: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf.

Strout, J. et al. (2017). 'Anuran call classification with deep learning'. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2662–2665. DOI: 10.1109/ICASSP.2017.7952639.

Sugai, Larissa Sayuri Moreira et al. (Nov. 2018). 'Terrestrial Passive Acoustic Monitoring: Review and Perspectives'. In: *BioScience* 69.1, pp. 15–25. ISSN: 0006-3568. DOI: 10.1093/biosci/biy147. eprint: https://academic.oup.com/bioscience/article-pdf/69/1/15/27503065/biy147.pdf. URL: https://doi.org/10.1093/biosci/biy147.

Tan, Mingxing and Quoc V. Le (2020). 'EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks'. In: *CoRR* abs/1905.11946. arXiv: 1905.11946. URL: https://arxiv.org/abs/1905.11946.

Teixeira, Daniella, Martine Maron and Berndt J. van Rensburg (2019). 'Bioacoustic monitoring of animal vocal behavior for conservation'. In: *Conservation Science and Practice* 1.8, e72. DOI: https://doi.org/10.1111/csp2.72. URL: https://conbio.onlinelibrary.wiley.com/doi/abs/10.1111/csp2.72.

The Museum für Naturkunde (2021a). *Archive of animal sounds from The Museum für Naturkunde*. URL: https://www.tierstimmenarchiv.de/ (visited on 12th May 2021).

— (2021b). *The Museum für Naturkunde - Leibniz Institute for Evolution and Biodiversity Science*. URL: https://www.museumfuernaturkunde.berlin/en/about/the-museum (visited on 12th May 2021).

Tieleman, T. and G. Hinton (2012). *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning.

Virtanen, Pauli et al. (2020). 'SciPy 1.5.2: Fundamental Algorithms for Scientific Computing in Python'. In: *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Welz, Adam (2019). *Listening to Nature: The Emerging Field of Bioacoustics*. YaleEnvironment360. URL: https://e360.yale.edu/features/listening-to-nature-the-emerging-field-of-bioacoustics (visited on 6th June 2021).

Wildlife Acoustics, Inc. (2021). *Wildlife Acoustics Home Page*. URL: https://www.wildlifeacoustics.com/ (visited on 27th May 2021).

Xie, Saining et al. (2017). *Aggregated Residual Transformations for Deep Neural Networks*. arXiv: 1611.05431. URL: https://arxiv.org/abs/1611.05431.

Zhang, Aston et al. (2020). *Dive into Deep Learning*. https://d2l.ai.

Appendix

A Juvenile sounds used

The juvenile P. fuscus sounds that were used from animal sounds archive of the Berlin Museum für Naturkunde (The Museum für Naturkunde 2021a).

files Pelobates_fuscus_juvenil_LtH_0001_short.mp3 to
Pelobates_fuscus_juvenil_LtH_0005_short.mp3

files Pelobates_fuscus_juvenil_LtH_0007_short.mp3 to
Pelobates_fuscus_juvenil_LtH_0008_short.mp3

files Pelobates_fuscus_juvenil_LtH_0013_short.mp3 to
Pelobates_fuscus_juvenil_LtH_0029_short.mp3

files Pelobates_fuscus_juvenil_LtH_0039_short.mp3 to
Pelobates_fuscus_juvenil_LtH_0114_short.mp3

files Pelobates_fuscus_juvenil_LtH_0116_short.mp3 to
Pelobates_fuscus_juvenil_LtH_0195_short.mp3

files Pelobates_fuscus_juvenil_LtH_0197_short.mp3 to
Pelobates_fuscus_juvenil_LtH_0211_short.mp3

files Pelobates_fuscus_juvenil_LtH_0213_short.mp3 to
Pelobates_fuscus_juvenil_LtH_0214_short.mp3

file Pelobates_fuscus_juvenil_LtH_0218_short.mp3

files Pelobates_fuscus_juvenil_LtH_0224_short.mp3 to
Pelobates_fuscus_juvenil_LtH_0267_short.mp3

B Annotated times and dates

Table 8 shows the times used for multi-label classification. The table is read left to right and downwards.

Table 8: Times that were used to create the multi-label classifier dataset.

Site	Date	Time	Site	Date	Time
Mothern	11.03.2015	13:00:00	Mothern	12.03.2015	09:00:00
Mothern	13.03.2015	05:30:00	Mothern	13.03.2015	23:00:00
Mothern	14.03.2015	11:00:00	Mothern	15.03.2015	08:00:00
Mothern	18.03.2015	09:30:00	Mothern	19.03.2015	06:00:00
Mothern	20.03.2015	17:00:00	Mothern	27.03.2015	05:00:00
Mothern	28.03.2015	07:30:00	Mothern	29.03.2015	09:30:00
Mothern	31.03.2015	20:00:00	Mothern	01.04.2015	08:00:00
Mothern	01.04.2015	15:00:00	Mothern	03.04.2015	07:00:00
Mothern	04.04.2015	17:30:00	Mothern	05.04.2015	09:00:00
Mothern	06.04.2015	10:00:00	Mothern	07.04.2015	19:00:00
Mothern	08.04.2015	12:00:00	Mothern	08.04.2015	20:00:00
Mothern	09.04.2015	13:00:00	Mothern	10.04.2015	07:00:00
Mothern	11.04.2015	19:00:00	Mothern	12.04.2015	18:00:00
Mothern	13.04.2015	08:00:00	Mothern	13.04.2015	10:30:00
Mothern	14.04.2015	06:00:00	Mothern	15.04.2015	09:00:00
Mothern	17.04.2015	11:00:00	Mothern	18.04.2015	15:00:00
Mothern	19.04.2015	12:00:00	Mothern	20.04.2015	16:30:00
Mothern	21.04.2015	09:00:00	Mothern	22.04.2015	15:00:00
Mothern	23.04.2015	11:00:00	Mothern	23.04.2015	14:30:00
Mothern	24.04.2015	12:00:00	Mothern	25.04.2015	17:30:00
Mothern	26.04.2015	12:30:00	Mothern	27.04.2015	06:00:00
Mothern	28.04.2015	09:30:00	Mothern	28.04.2015	12:00:00
Mothern	29.04.2015	13:00:00	Mothern	01.05.2015	07:30:00
Mothern	02.05.2015	08:30:00	Mothern	03.05.2015	19:00:00
Mothern	04.05.2015	11:00:00	Mothern	05.05.2015	07:00:00
Mothern	06.05.2015	08:00:00	Mothern	07.05.2015	06:00:00
Mothern	08.05.2015	07:00:00	Mothern	09.05.2015	10:00:00
Mothern	10.05.2015	06:30:00	Mothern	11.05.2015	07:00:00
Mothern	12.05.2015	11:00:00	Mothern	14.05.2015	08:00:00
Mothern	15.05.2015	06:00:00	Mothern	16.05.2015	12:00:00
Mothern	02.04.2015	10:00:00	Mothern	16.04.2015	04:30:00
Mothern	30.04.2015	09:30:00	Mothern	17.05.2015	06:00:00
Mothern	02.06.2015	21:00:00	Mothern	20.06.2015	10:00:00
Mothern	18.05.2015	14:00:00	Mothern	19.05.2015	12:30:00
Mothern	20.05.2015	11:00:00	Mothern	21.05.2015	16:00:00
Mothern	22.05.2015	10:00:00	Mothern	23.05.2015	09:00:00
Mothern	24.05.2015	17:30:00	Mothern	25.05.2015	23:00:00
Mothern	26.05.2015	22:00:00	Mothern	27.05.2015	03:00:00
Mothern	28.05.2015	04:00:00	Mothern	29.05.2015	05:00:00
Mothern	30.05.2015	17:00:00	Mothern	31.05.2015	22:30:00
Mothern	01.06.2015	16:30:00	Mothern	03.06.2015	22:00:00
Mothern	04.06.2015	09:00:00	Mothern	05.06.2015	20:00:00

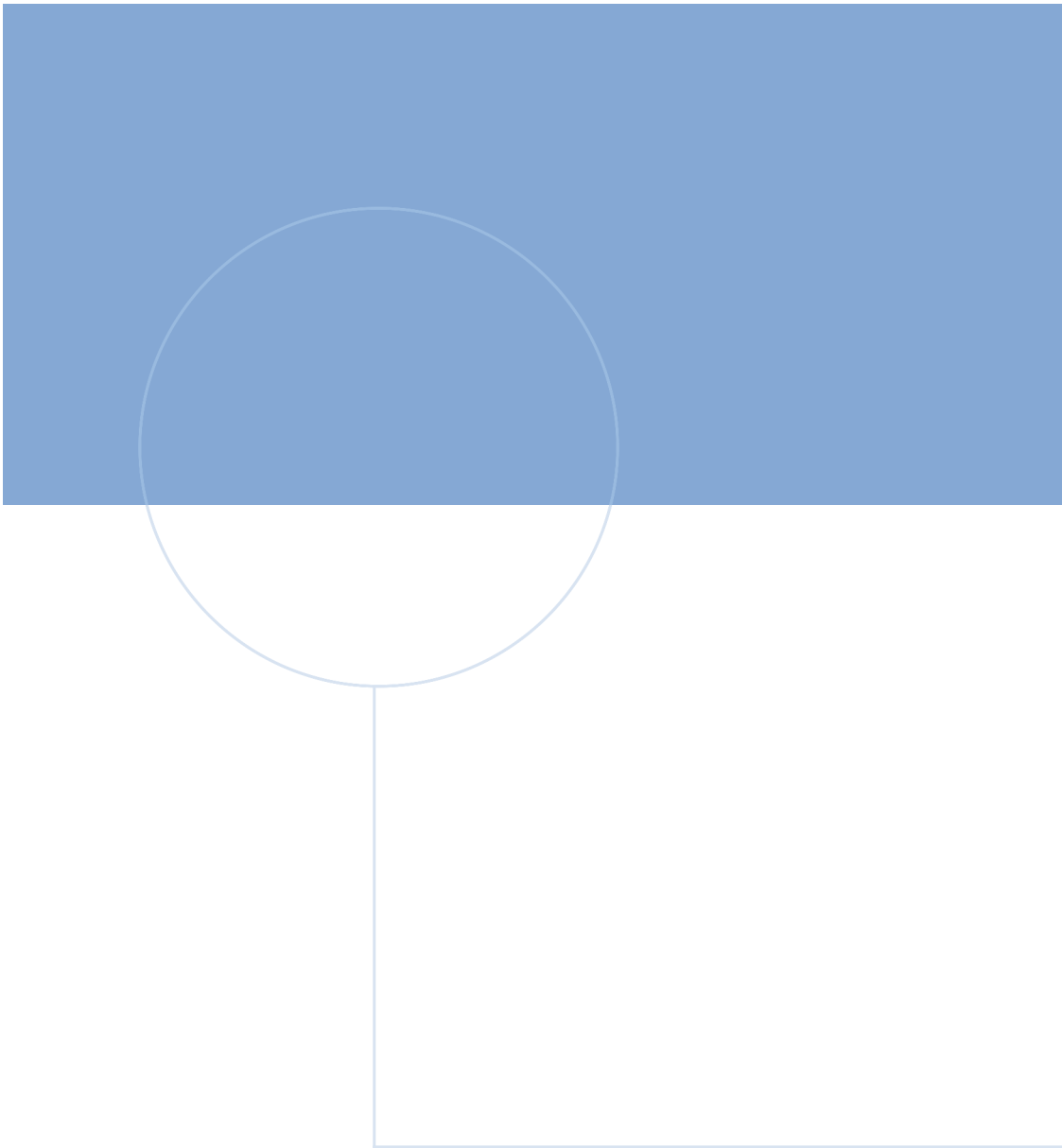
Mothern	06.06.2015	22:00:00	Mothern	07.06.2015	06:30:00
Mothern	08.06.2015	08:30:00	Mothern	09.06.2015	15:30:00
Mothern	10.06.2015	09:30:00	Mothern	11.06.2015	15:30:00
Mothern	11.06.2015	18:30:00	Mothern	12.06.2015	11:30:00
Mothern	13.06.2015	10:30:00	Mothern	14.06.2015	13:00:00
Mothern	15.06.2015	15:00:00	Mothern	16.06.2015	20:30:00
Mothern	17.06.2015	14:00:00	Mothern	18.06.2015	14:30:00
Mothern	19.06.2015	13:00:00	Mothern	21.06.2015	12:30:00
Mothern	22.06.2015	14:30:00	Sauer	11.03.2015	19:30:00
Sauer	11.03.2015	20:00:00	Sauer	17.03.2015	20:30:00
Sauer	17.03.2015	21:00:00	Sauer	17.03.2015	21:30:00
Sauer	17.03.2015	22:00:00	Sauer	17.03.2015	22:30:00
Sauer	17.03.2015	23:00:00	Sauer	17.03.2015	23:30:00
Sauer	18.03.2015	01:00:00	Sauer	18.03.2015	01:30:00
Sauer	18.03.2015	19:00:00	Sauer	18.03.2015	20:30:00

Table 9 shows the times that were used to create the detector dataset. The times are shown in no particular order.

Table 9: Times that were used to create the detector dataset.

Site	**Date**	**Time**	**Site**	**Date**	**Time**
Mothern	14.05.2015	06:00:00	Mothern	13.04.2015	10:30:00
Mothern	11.04.2015	18:30:00	Mothern	22.04.2015	15:00:00
Mothern	14.04.2015	13:00:00	Mothern	15.04.2015	09:00:00
Mothern	17.04.2015	11:00:00	Mothern	21.04.2015	10:00:00
Mothern	05.05.2015	05:30:00	Mothern	07.05.2015	06:00:00
Mothern	09.05.2015	10:00:00	Mothern	11.05.2015	14:30:00
Mothern	19.03.2015	06:00:00	Mothern	04.04.2015	12:30:00
Mothern	04.04.2015	17:30:00	Mothern	03.05.2015	07:00:00
Mothern	15.05.2015	06:00:00	Mothern	12.04.2015	07:30:00
Mothern	12.04.2015	18:00:00	Mothern	13.04.2015	08:00:00
Mothern	03.05.2015	08:30:00	Mothern	04.05.2015	11:00:00
Mothern	23.05.2015	08:00:00	Mothern	23.05.2015	09:00:00
Mothern	24.05.2015	12:00:00	Mothern	14.04.2015	06:00:00
Mothern	12.05.2015	11:00:00	Mothern	05.05.2015	07:00:00
Mothern	14.05.2015	08:00:00	Mothern	11.03.2015	13:00:00
Mothern	12.03.2015	09:00:00	Mothern	13.03.2015	05:30:00
Mothern	14.03.2015	11:00:00	Mothern	15.03.2015	08:00:00
Mothern	18.03.2015	09:30:00	Mothern	04.04.2015	15:00:00
Mothern	11.04.2015	19:00:00	Mothern	18.04.2015	15:00:00
Mothern	19.04.2015	12:00:00	Mothern	20.04.2015	16:30:00
Mothern	21.04.2015	09:00:00	Mothern	13.03.2015	23:00:00
Mothern	20.03.2015	17:00:00	Mothern	27.03.2015	05:00:00
Mothern	23.04.2015	11:00:00	Mothern	23.04.2015	14:30:00
Mothern	24.04.2015	12:00:00	Mothern	25.04.2015	17:30:00
Mothern	26.04.2015	12:30:00	Mothern	27.04.2015	06:00:00
Mothern	30.04.2015	09:30:00	Mothern	01.05.2015	07:30:00
Mothern	02.05.2015	08:30:00	Mothern	28.03.2015	07:30:00

Mothern	29.03.2015	09:30:00	Mothern	31.03.2015	20:00:00
Mothern	01.04.2015	08:00:00	Mothern	01.04.2015	15:00:00
Mothern	02.04.2015	10:00:00	Mothern	03.04.2015	07:00:00
Mothern	05.04.2015	09:00:00	Mothern	06.04.2015	10:00:00
Mothern	07.04.2015	19:00:00	Mothern	08.04.2015	12:00:00
Mothern	08.04.2015	20:00:00	Mothern	09.04.2015	13:00:00
Mothern	10.04.2015	07:00:00	Mothern	03.05.2015	19:00:00
Mothern	28.04.2015	09:30:00	Mothern	28.04.2015	12:00:00
Mothern	29.04.2015	13:00:00	Mothern	16.05.2015	12:00:00
Mothern	17.05.2015	06:00:00	Mothern	18.05.2015	14:00:00
Mothern	19.05.2015	12:30:00	Mothern	20.05.2015	11:00:00
Mothern	21.05.2015	16:00:00	Mothern	22.05.2015	10:00:00
Mothern	24.05.2015	17:30:00	Mothern	25.05.2015	23:00:00
Mothern	26.05.2015	22:00:00	Mothern	28.05.2015	04:00:00
Mothern	30.05.2015	17:00:00	Mothern	31.05.2015	22:30:00
Mothern	01.06.2015	16:30:00	Mothern	02.06.2015	21:00:00
Mothern	29.05.2015	05:00:00	Mothern	03.06.2015	22:00:00
Mothern	04.06.2015	09:00:00	Mothern	05.06.2015	20:00:00
Mothern	07.06.2015	06:30:00	Mothern	06.06.2015	22:00:00
Mothern	08.06.2015	08:30:00	Mothern	09.06.2015	15:30:00
Mothern	10.06.2015	09:30:00	Mothern	11.06.2015	15:30:00
Mothern	12.06.2015	11:30:00	Mothern	14.06.2015	13:00:00
Mothern	15.06.2015	15:00:00	Mothern	16.06.2015	20:30:00
Mothern	11.06.2015	18:30:00	Mothern	13.06.2015	10:30:00
Mothern	17.06.2015	14:00:00	Mothern	18.06.2015	14:30:00
Mothern	19.06.2015	13:00:00	Mothern	20.06.2015	10:00:00
Mothern	21.06.2015	12:30:00	Mothern	22.06.2015	14:30:00
Mothern	16.04.2015	04:30:00	Mothern	06.05.2015	08:00:00
Mothern	08.05.2015	07:00:00	Mothern	10.05.2015	06:30:00
Mothern	11.05.2015	07:00:00	Mothern	27.05.2015	03:00:00
Sauer	11.03.2015	19:30:00	Sauer	11.03.2015	20:00:00
Sauer	17.03.2015	20:30:00	Sauer	17.03.2015	21:00:00
Sauer	17.03.2015	21:30:00	Sauer	17.03.2015	22:00:00
Sauer	17.03.2015	22:30:00	Sauer	17.03.2015	23:00:00
Sauer	17.03.2015	23:30:00	Sauer	18.03.2015	01:00:00
Sauer	18.03.2015	01:30:00	Sauer	18.03.2015	19:00:00
Sauer	18.03.2015	20:30:00	-	-	-



NTNU

Norwegian University of
Science and Technology