

Maja Sofie Stava

ANN for classification of Jack snipe

Master's thesis in Electronics Systems Design and Innovation

Supervisor: Guillaume Dutilleux

June 2020

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



NTNU

Norwegian University of
Science and Technology

Maja Sofie Stava

ANN for classification of Jack snipe

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Guillaume Dutilleux
June 2020

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

ABSTRACT

Machine learning used in biodiversity conservation is an expanding field of study, as it provides non-intrusive monitoring of events in the wild. For the purpose of this thesis, the event of interest is to classify jack snipe in a recording of several hours. A binary CNN- and LSTM model was built and trained on an annotated dataset consisting of 400 clips. Each clip has a duration of 4 seconds, where 130 of the clips are of jack snipe vocalization, and 270 is non-jack snipe vocalization, like wind, grouse, and crow. The models are then tested on the sound recording of 1 h and 49 min, containing 22 vocalizations of jack snipe. The recording included clean vocalizations and vocalizations polluted with environmental sounds such as wind and river noise. The best result came from using the CNN model with input shape of (90, 126, 1) and added DTW and bandpass filter with frequency range 400-2000 Hz on the clips to be predicted, with a true positive rate (TPR) of $TPR = 0.88$ and false positive rate (FPR) of $FPR = 0.086$. The total length of all the 196 positive predicted clips is 13 min and 4 sec. When testing on full recordings of 9 h and 15 h length, the predictions proved more unstable depending on how windy the recording was.

Maskinl ring som brukes i biomangfoldisk bevarelse er et fagomr de i utvikling, da det tilbyr en ikke-ingripende overv kning av hendelser i naturen. For formålet til denne avhandlingen s  er vi interesert i   klassifisere kvartsbekkasin i opptak p  flere timer. En bin r CNN -og LSTM modell ble bygd og trent p  et merket datasett best ende av 400 klipp, hvor hvert klipp varte i 4 sekunder. Av disse var 130 klipp av kvartsbekkasin, mens 270 var av ikke-kvartsbekkasin. Modelene blir testet med et umerket lydopptak p  1 t og 49 min, som inneholder 22 vokaliseringer av kvartsbekkasin. Lydklippet inneholder klare vokaliseringer uten mye forstyrrelse, og vokaliseringer som er forurenset med lyder fra omgivelsen, slik som vind og elvbrus. Det beste resultatet ble oppn dd ved   bruke en CNN model med inngangsform (90, 126, 1) og et p f rt b ndpassfilter med frekvensomr de 400-2000 Hz p  lydklippet. Dette ga en sann positive testrate (TPR) p  $TPR = 0.88$, og en falsk positiv testrate (FPR) p  $FPR = 0.086$. Den totale lengden p  de predikert positive klippene er 13 min og 4 s.

ABBREVIATIONS

ANN Artificial Neural Network

CNN Convolutional Neural Network

DNN Deep Neural Network

FPR False positive rate

JS Jack snipe

LSTM Long Short-Term Memory

MFCC Mel-frequency Cepstrum Coefficient

ML Machine Learning

NN Neural Network

RNN Recurrent Neural Network

TPR True positive rate

AKNOWLEDGMENT

I would like to thank my supervisor, Guillaume Dutilleux, for providing guidance and advice on how to proceed with the task. I would also like to thank Eline Stenwig and Kristina Andersen Thue for discussing different approaches with me, and giving moral support when needed.

I would like to thank NINA and John Atle Kålsås for providing the annotated files and the sound files used to create the dataset.

Last but not least, I would like to thank my family for support, encouragement and brain storming around different ways to solve the problems at hand during the writing of this thesis.

1	Introduction	1
1.1	Motivation	1
1.2	Scope	2
1.3	Previous work	2
1.4	Outline of the report	3
1.5	Background	4
2	Theory	7
2.1	Recurrent neural networks, Long short-term memory	7
2.2	Convolutional neural network	8
2.3	Mel-frequency Cepstrum Coefficient	10
2.4	Implementation	11
2.5	Validation and improvements of the models	14
3	Method	17
3.1	Pre-processing	17
3.2	Implementation	19
3.3	Testing of the trained models	22
4	Results	25
4.1	Training of model	25
4.1.1	CNN model	25
4.1.2	LSTM model	26
4.2	Testing of the models using test recording JS22	26
4.2.1	CNN model	27
4.2.2	LSTM model	28
4.3	Vocalizations detected fully or partly	29
4.4	Testing on long recording	29
5	Discussion	31

5.1	The training of the models	31
5.2	Validation of the models	31
5.3	CNN vs LSTM	33
5.4	Testing on full length recording	34
6	Conclusion	35
6.1	Future work	36
A	Timestamp for jack snipe vocalization in test clip JS22	41
B	How to set up and run the training/classification script	45
B.1	Packages to install	45
B.2	Training file	46
B.3	Running the trained model	46
C	Trained models	47
C.1	Trained CNN graph and architecture	47
C.2	LSTM	51
D	Confusion matrix, TPR and FPR for the CNN model	53
D.1	CNN with 45 mel bands	53
D.2	CNN with 90 mel bands	54
D.3	CNN with 128 mel bands	55
E	confusion matrix, TPR and FPR for the LSTM model	57
E.1	LSTM with 45 mel bands	57
E.2	LSTM with 90 mel bands	58
E.3	LSTM with 128 mel bands	59
F	List of bird species	61

The dataset used in this project was recorded in Kautokeino between 7-14th of June 2017. It is not representative for all birds in Kautokeino, as it is recorded at one location and in a limited time window of 8 days in June. Each day two sound files are generated, one large containing 18 h and 38 min of audio and one small with 5h and 20min of audio. It was decided to focus mainly on jack snipe, from here referred to as JS, as this bird is red-listed and difficult to spot visually. The goal is to identify JS vocalization in the recordings using machine learning methods and reduce the time it would take to look through the full recording manually to find the vocalizations. A benchmark code using Mel-frequency cepstrum coefficient and convolutional neural network from the Urban sound challenge [1] were used as inspiration.

1.1 Motivation

The artificial neural network is an exciting field in fast development, becoming more important in modern society. Examples of use are in medical technology, biodiversity monitoring, automated cars, and in financial institutions[2]. For biodiversity conservation, non-inverse bioacoustic tracking is a growing field of interest[3]. As I like spending time in the outdoors, I found the idea of using machine learning to classify birds interesting, and an excellent opportunity to learn more about machine learning.

1.2 Scope

The scope of this report is to look into the following

- Using machine learning to train a binary NN model on a dataset made up of 400 clips with a length of 4 seconds, where 130 is of Jack snipe vocalization, and 270 is of any other sound than jack snipe.
- Using machine learning to identify Jack snipe by running a trained model on long recordings from a recorder placed in the wild. The results are presented as a sound clip containing the jack snipe's vocalization, where the name gives the start time of the vocalization, the length of the clip, and the confidence interval. The timestamp is also stored in a text file to be viewed in Audacity.

1.3 Previous work

Convolutional neural networks (CNN) are popular to use in audio classification, and have been used for classifying bird species[4], environmental sounds[1], fish[5] and music genre[6]. In 2004, Dan Stowell and Mark D. Plumbley [7] used unsupervised learning with the implementation of spherical k-mean, on four large and diverse datasets of bird vocalization. The results gave a strongly improved bird classification. They also suggested that raw Mel spectra might be better for benchmark compared to the MFCCs, as it outperformed the MFCCs in their tests.

Stefan Kahl et al. [8] used CNNs for large scale birds sound classification for the BirdCLEF 2017. The dataset used consisted of 36 496 audio recordings with 1500 different bird species. The training set consisted of unbalanced classes, with the smallest class having four recordings, while the maximum class has 160 recordings. They separated the sound file into five seconds sound clips with four seconds overlap and transformed these clips into 940 740 spectrograms. They modeled three different CNNs, where the runtime during training was measured to a maximum of 5500 s per Epoch. They achieved a mean average precision of 0.605.

In *Bird sounds classification by large scale acoustic features and extreme learning machine* [9] by Kun Quian et al., they implement p-center for segmentation, before extracting a large scale of acoustic features from bird sound syllables as units. The classifier is an extreme learning machine and achieved an accuracy of 86.57 % with 54 bird species.

Shawn Hershey et al. [10] looked closer into the processing time for large datasets by using 70 million training videos from Youtube, generating a total of 5.24 million hours of audio labeled with 30 871 audio tags. The training time for the different models varied between 35h to 119h.

The lack of annotated datasets has been a challenge when trying to improve the accuracy of bird song classification. In 2019, Dan Stowell et al. [11] introduced NIPS4Bplus in *NIPS4Bplus: a richly annotated birdsong audio dataset*. NIPS4Bplus consists of recordings of bird vocalization with species tags and temporal annotations. The intent was to provide a fully annotated dataset with bird vocalization to improve detection and classification further.

Dan Stowell launched the app *Warblr* the 13th august 2015 [12]. Wablr is a bird recognition app that is aimed for British birds. It uses unsupervised learning, with spherical k-means in combination with CNN [7]. From the launch back in 2015 to today, Wablr has become increasingly popular, and thus the sound database for the app increases as user world wild record birds and saves the clip in the database. In 2017 a Norwegian app similar to Wablr, *Whatbird*, launched by a company based in Trondheim. Today the app has over 30 000 daily users [13], and the database expands as the users store their clip in the database. In order for the apps to give a result, it needs to have access to a mobile network or Wi-Fi. It analyses each clip and can have problems identifying multiple birds or if there is much noise.

1.4 Outline of the report

First, some theory on machine learning and the neural network will be presented in Chapter 2. Important layers making up a neural network model in Keras is explained in the same chapter. In Chapter 3, the method for pre-processing the training dataset and training the models is explained. The making of a test clip to run trough the trained models is also explained, together with methods to help validate the models. Then follows Chapter 4 where all the results from the running and testing on the models are presented before a follow-up discussion is done in Chapter 5 followed by the conclusion in Chapter 6

1.5 Background

Jack snipe

Jack snipe is a shy bird, that tends to crunch down when approached, making it hard to detect visually. The preferred habitat is thick vegetation, wet grassland, marshes, and reedbeds. It is a small bird with a long beak. Jack snipe vocalizes in the lower frequency band, with a frequency range between 500-1500 Hz. The vocalization sounds a bit like a galloping horse in the distance, with one vocalization lasting around 8-12 seconds.

Machine learning

Machine learning (ML) focuses on the development of computer programs and allows the machine to learn from a large amount of data, and predict the most likely output based on the observed patterns picked up during training. There are four different machine learning methods: Supervised ML -, unsupervised ML-, Semi-supervised ML -and Reinforcement ML algorithms. For the purpose of this thesis, supervised ML is used.

Supervised learning

According to Mark Ryan M.Talabis et al. [14], supervised learning uses labeled data to train the model. The prediction of unlabeled data is made with the labeled data as the basis of the prediction. It is still used a lot today and is easy to implement. A drawback is the performance, as it is time consuming when dealing with large datasets. Supervised learning demands more work from the developer, as one has to label the training data manually.

Artificial neural network

An artificial neural network (ANN) is a network of nodes developed to imitate the human brain. The nodes in the neural network (NN) works as neurons in the brain. The neuron nodes are interconnected with each other, analyzing and processing information and is capable of learning as more input is given. An illustration of the natural neuron is shown in Fig.1.1. Easily explained, the

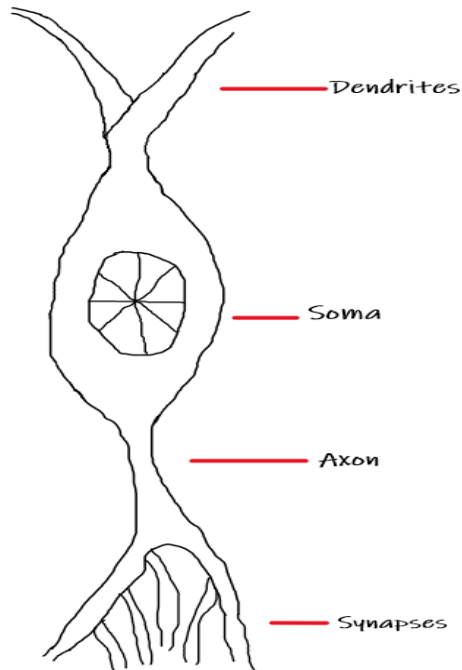


Figure 1.1: Illustration of a neuron

neuron takes in input (dendrites), process the input (soma), does some non-linear operation on the processed input (axon) before outputting the results to the other connected neurons (synapses). The artificial neurons are not as complex as the natural neurons, but the function of these four basic components are simulated in the artificial neurons [15].

2.1 Recurrent neural networks, Long short-term memory

In *The Basics of Recurrent Neural Networks (RNNs)* by Ben Khuong [16], a recurrent neural network (RNN) is a model that gives the output at time t back to the input in the same network layer at time $t+1$. A drawback when using RNNs is that RNNs are not capable of memorizing long term dependency of an input time sequence since the gradient will vanish when using this model. To solve the shortcoming of RNNs, the long short-term memory (LSTM) network was introduced. LSTM is a RNN, with feedback connections. The LSTM network is well suited to classify, process, and make predictions based on time series data.

A LSTM network consists of a set of recurrently connected subnets, called memory blocks, where each block contains a minimum of one self-connected memory cell and three multiplicative units: The input, output and forget gates. See Fig. 2.1 for a graphical overview of the cell. By looking at the figure, one can see that the new input to the model (x_t) is combined with the previous output from h_{t-1} . The input signal is then processed through the input gate, The first order of business in the LSTM cell is to decide what information to keep in the cell state. This is done in the forget gate (f_t), and it outputs a number between 0 and 1, where 0 means throw all away and 1 means keep all. Next step is to decide what information to store in the cell state. This is done in the input gate (i_t), which is a layer of activated sigmoid nodes. The combined input signal is then squashed by the tanh layer, and multiplied with the output from the input gate is then multiplied by the squashed input signal. The result of this operation is that the sigmoids get rid of any elements of the input vector that is not required.

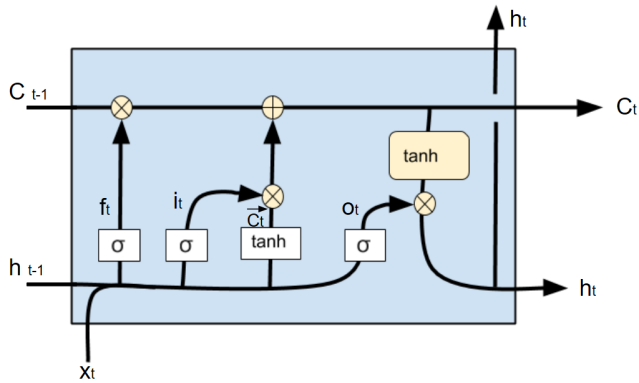


Figure 2.1: Illustration of a LSTM cell

This gives the network a continuous series of write, read, and reset operations. The set-up of an LSTM network is the same as for a simple RNN, with the only difference being that the non-linear units in the hidden layers of the RNN are replaced by the memory blocks of LSTM [16].

2.2 Convolutional neural network

CNN is often used for machine learning on images, as it identifies spatial patterns from images efficiently. CNN consists of an input and output layer, with multiply hidden layers between them. The hidden layers of a CNN model have a minimum of one convolution layer and then one or more fully connected layers. A CNN model with three convolutional layers and one fully connected layer can be seen in Fig. 2.2.

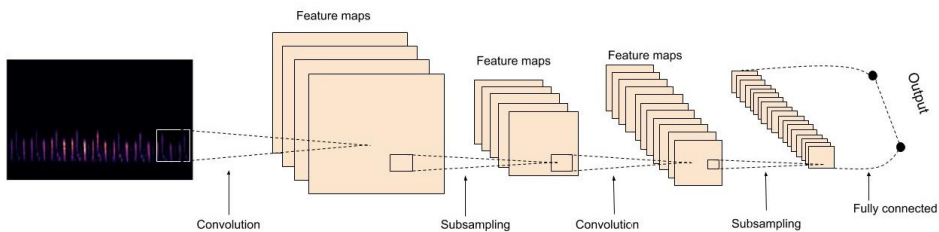


Figure 2.2: Illustration of a CNN model with 3 convolution layers.

The convolutional layer - Conv2D

The convolutional layer is the first layer to extract features from the input image, learning the image features by using small squares of input data. The convolution layer preserves the relationship between the pixels [17].

The input to a convolution layer is an image of the form $m \times m \times q$, where m is the height and the width of the image given in pixels, and q is the number of channels. The convolution layer will consist of k kernels of size $n \times n$, where $n < m$. the kernel size represents the height and width of the 2D convolution window and can vary for each kernel. The kernel size decides the locally connected structure, where each of these is convoluted with the input image to produce k feature maps of the size $m-n+1$. As a general rule, the kernel size is often two odd numbers, e.g. (3, 3) or (5, 5). The reason is simple; when the kernel is odd-sized, the output pixel will be centered in all the layers, see Fig.2.3. The

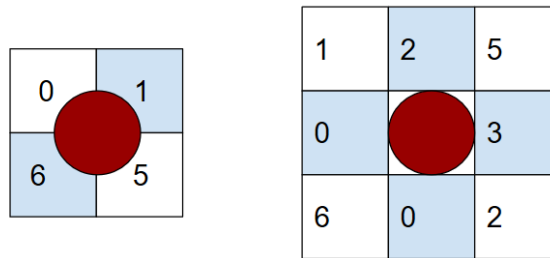


Figure 2.3: Illustration of two different kernel sizes, with kernel size (2, 2) to the left and (3, 3) to the right.

first Conv2D layer needs to be given the input_shape, e.g. input_shape = (128, 126, 1), which gives a 128×126 greyscale picture. This is followed by a pooling layer, where each map is downsampled by pooling[18]. Pooling reduces the time and use of resources in the training stage[19]. It reduces the feature map size without a loss in information, thus reduces the amount of processing needed in the training of the model. There are three common variants of pooling in CNN: Max pooling, which takes the maximum pixel value within the filter; Average pooling, which takes the average pixel value within the square and sum pooling, which sums the pixel values in the filter. The pooling is done over $p \times p$ contiguous regions, where the size of the pooling operator is smaller than the size of the feature map. The size of the feature map is given as the output after each Conv2D layer. The most commonly used is a 2×2 pixel window applied

with a stride of two pixels. This gives that each feature map will be reduced by a factor of two, reducing the number of pixels or values in each feature map to one quarter the size. In recent years, max-pooling has become the most common to use, as it extracts the maximum value for each patch of the feature map, see Fig.2.4 for illustration.

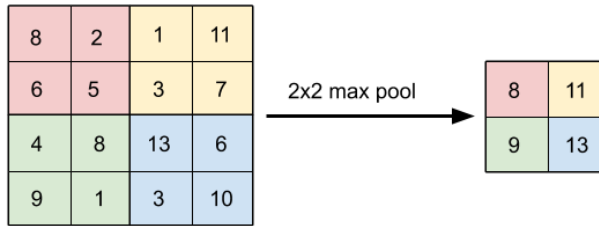


Figure 2.4: Illustration of max pooling with kernel size 2x2 and stride = 2.

2.3 Mel-frequency Cepstrum Coefficient

Mel-Frequency Cepstral coefficient (MFCC) is explained by Golam Rabbani et al. [20] as the information on the rate of change in spectral bands. It is based on the fact that that variation in the human ear's critical bandwidth with frequency is known. MFCCs uses two types of filters, linearly spaced filters, and logarithmically spaced filters. The signal is then expressed in the Mel frequency scale in order to capture the important characteristics of the signal. See Fig. 2.5 for a block diagram of the MFCC processor. The triangular shapes in the mel

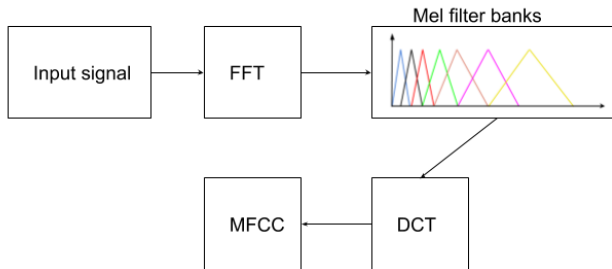


Figure 2.5: A block diagram of the MFCC process.

filter banks is called mel bands. From [20], the Mel-frequency scale is a linear frequency spacing below 1000Hz and a logarithmic spacing above 1000Hz. The

pitch of a 1 kHz tone, which is 40 dB over the perceptual hearing threshold, is used as a reference point and defined as 1000 mels. The formula for converting frequency f in Hz to the Mel scale can be seen in Eq.(2.1).

$$Mel(f) = 2595 * \log_{10}(1 + \frac{f}{700}) \quad (2.1)$$

The Cepstral coefficient is obtained by taking the log mel spectrum $Mel(f)$ and convert it back to time. The mel spectrum coefficients, as well as their logarithms, are real numbers, so the conversion back to the time domain can be done by using Discrete Cosine Transform (DCT). The resulting spectrum is not in the frequency domain nor in the time domain; it is in the quefrequency domain and is referred to as the Cepstral[21]. The MFCC can be found as its amplitude. When plotting, the mel spectrogram shows the time on its x-axis and the mel scale on the y-axis. In Fig.2.6, the mel spectrogram of a JS vocalization is displayed. In the Fig. 2.6, the y-axis is the frequency in log scale.

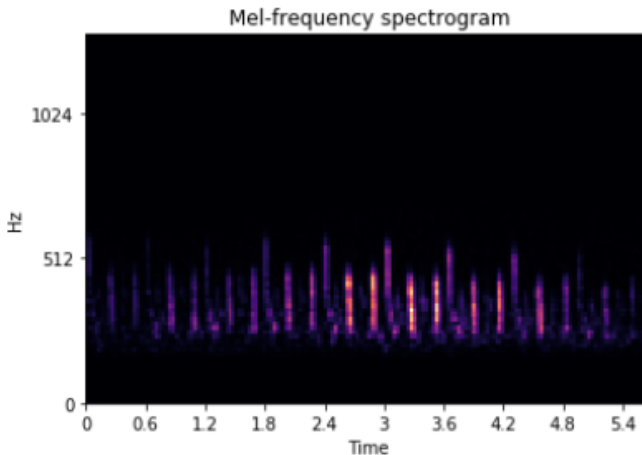


Figure 2.6: Mel spectrogram of a clean jack snipe vocalization, the x-axis shows the time in [s] and the y-axis is the frequency in log scale.

2.4 Implementation

Keras is an API used on the TensorFlow backend to make the implementation of machine learning simpler. It is compatible with Python, and have support in many libraries.

Before the training starts, some pre-defined variables are set: epoch and batch

size. Epochs are the number of times the algorithm runs through the entire dataset. In general, the epoch size is decided depending on if the dataset contains very similar or different data, e.g., only white dogs or black cats will need fewer epochs than if the data is different. Batch size is equal to the number of iterations for one epoch. The model is trained using gradient descent, where there are three common types: Stochastic-, Batch -and mini-batch gradient descent, see list below.

- Batch gradient descent: Batch size = size of the training set
- Stochastic gradient descent: Batch size = 1
- Mini-batch gradient descent: $1 < \text{Batch size} < \text{size of the training set}$

Batch gradient descent is time-consuming and can have only one update per epoch. On the positive side, it is guaranteed to converge. Stochastic gradient descent is usually much faster than batch gradient descent and can be used to learn online. Mini-batch gradient descent uses the best of both worlds, as it reduces the variance of the parameter updates and has support in deep learning libraries, which makes the computing of gradient with respect to a mini-batch very efficient[22].

As the training dataset often is extensive, the batch size is generally not set equal to training size. Small batch size is often preferred, as it makes it easier to fit one batch of training data in memory, and because it gives more noise. This offers a regularizing effect and reduces the generalization error[23].

Important layers in Keras for machine learning

From [24], the following are some of the important layers in Keras that needs or is beneficial to implement when building a NN model.

- **Dropout():** The dropout layer is used to prevent overfitting. The dropout rate is a float between 0 and 1. The dropout layer sets random input nodes to 0 and scales the other nodes up by $\frac{1}{1-\text{rate}}$. To activate the Dropout() layer, the training parameter must be set to True.
- **Dense():** The deeply connected NN layer. It implements the operation: $\text{output} = \text{activation}(\text{dot}(\text{input}, \text{kernel}) + \text{bias})$ where **activation** is the activation function passed as the activation argument (like ReLU or softmax, see below for explanation of both) and **kernel** is a weights matrix created by the layer. If use bias is set to true, the **bias** is a bias vector created by the layer.
- **Flatten():** It simply flattens the input without affecting the batch size. For example, if the input to the flatten layer is on the form (None, 3, 30, 16), the output shape is (None, 1440), as $3 \times 30 \times 16 = 1440$.

- **Activation layer:** The activation functions are either linear or non-linear, and is applied to the linear input before the output is passed on to the next layer. In machine learning it is most popular to chose a non-linear activation function, as it can learn from highly complex, non-linear data. When the non-linear activation function is applied to all neurons in the NN, it makes the whole network non-linear, making it easier for the model to generalize the data. Three popular activation functions are the rely, -tanh -and softmax activation function.
 - **Rectified Linear Unit activation (relu)** function, returns the element-wise maximum of 0 and the input tensor, $\max(x,0)$ unless another max return value is specified.
 - **The tanh activation** fuction is a hyperbolic tangent activation function with range $[-1, 1]$, where a negative inut is mapped strongly negative and zero inputs will be maped near zero. The tanh activation function is mainly used for binary classifiers.
 - **Softmax activation** function. It converts a real vector to a vector of probabilities in the range 0-1 and sum to 1. This activation function is often used for the last layer in the NN so that the result could be used as a probability distribution.

Compilation

When compiling the model in Keras, the two arguments for optimizer and loss needs to be specified. There are nine different options for optimizers, which are all explained in [24]. For deep neural networks, the *adam* optimizer is a popular choice. It requires little memory and works well without changing the hyperparameters to much. For shallow neural networks, the *Stochastic gradient descent* (SGD) optimizer works well.

For the other argument, the loss function, there are three main types of loss: Probabilistic losses, Regression losses, and hinge losses. Each type has multiple subclasses, see [24] for more details. Probability losses give the cross-entropy loss between a true label and the predicted label and are used when there are only to classes to predict. Regression loss calculates the mean square error between true labels and predicted labels. Hinge losses lie in the name, as it computes the hinge loss between a true label and the predicted label. The hinge loss is widely used for support vector machines.

2.5 Validation and improvements of the models

Overfitting and underfitting

In machine learning, there are two outcomes one wishes to avoid when training a model; underfitting and overfitting. Underfitting is described by WMP van der Aalst et al. [25] when the model over-generalizes the data, meaning that the model will have problems to assign specific characteristics to a class. As can be seen from Fig.2.7, the model has a high error value when its underfitted, meaning it has poor performance on the training data.

Overfitting is more commonly discussed and can cause the developer to think that the model works better than it does. An overfitted model will give good results on the training data, so much that it can have an issue with generalizing data and thus not perform well on new data[25]. In Fig.2.7, it can be seen that overfitting occurs when the model gets more complex. There are some

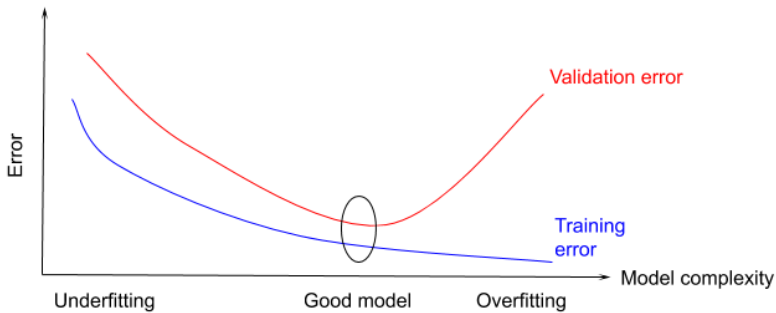


Figure 2.7: An illustration showing underfitting and overfitting in a model when under training.

techniques that can be used to reduce overfitting when evaluating machine learning algorithms. As explained by Nitish Srivastava et al. [26], dropout is a way of reducing overfitting. It provides a method for approximating combining exponentially many different neural networks efficiently. Dropout refers to the fact that when used, it drops out nodes (hidden and visible) in a NN. Another method is to reduce the complexity of the model, i.e., reduce the number of layers in the model.

Dynamic time warping

Dynamic time warping (DTW) is a technique that is used to find an alignment between two given, time-dependent sequences [27]. It finds the best match between signals even if the signals are out of phase. The `dtw()` function in Python gives the best alignment of two signals and returns the distance. The distance is small for similar clips, and DTW is often used in vocal and speech recognition where a word could be spoken in different tones and at different speeds.

3.1 Pre-processing

The sound files for the training dataset was made from sound clips taken in a national park in Kautokeino in 2017 by the Norwegian Institute for Nature Research (NINA). It consists of 400 annotated sound clips, 130 of which is JS and 230 if of wind, rivers, crickets, and a variety of birds suck as cocko, chaffinch, crow, grouse a.s.o. A list of the birds possibly present is presented in Appendix F. The annotation is done in *Audacity* before the sound files are exported. The clips containing JS are filtered through a bandpass filter with a frequency range between 400-2000 Hz as the frequency range for the vocalization of JS is between 500-1500 Hz, see Fig.3.1. This is done to make the

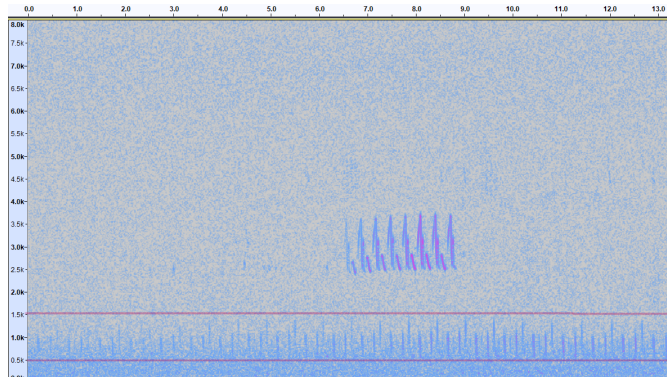


Figure 3.1: Illustration of a vocalization of jack snipe, showing between the two lines and with a duration of a littler over 13 seconds.

vocalization as clean as possible. No filtering is used on the samples without JS vocalization.

Training and validation dataset

The sound clips are then cut to a specific length and sampling frequency. Each clip is 4 seconds long, with a samplings frequency of 16 kHz. The number of detected vocalization varied a lot for each file, where the highest had almost twice as many detected vocalizations than the one with least. The number of vocalization in each file can be seen from Tab.3.1 The detection of vocalization

Table 3.1: Table showing the number of detected vocalization in the recordings provided from John Atle Kålås at NINA done between 07-14th of June. Each recording is divided into to two sub-recordings, containing 24h of sound when summarized.

Recording	Date	Detected vocalization
1	07/06	51
2	08/06	56
3	09/06	58
4	10/06	64
5	11/06	69
6	12/06	101
7	13/06	81
8	14/06	93

of JS was marked as seen Tab.3.1 in an additional .csv file also provided from NINA.

- 1: Not visible on the sonogram, but can be heard weakly
- 2: Weakly visible on the sonogram
- 3: Partly covered by noise from rivers in the sonogram
- 4: Can be seen clearly in the sonogram
- 5: Two males vocalizes almost at the same time, making the vocalization longer and clearer
- 9: Clearly visible in the sonogram, but the sound quality is not checked closer
- Detected: the vocalization was detected, but the quality of the detection was not further commented

The annotated dataset used to train the model contains mainly clips from re-

ording 8 and 3, but also from recording 1 and 2. In total, the annotated dataset contains 130 clips with jack snipe vocalization, annotated as JS, and 270 clips containing wind, river noise, a variation of bird calls such as chaffinch, crow, and cuckoo, annotated as NJS. Out of the 130 clips with JS vocalization, the majority are marked as 9 in the .csv file, but also vocalizations marked as 1, 2, 4, and 5 are collected to give a better range of vocalizations in the dataset for the model to train on. Before building and training the model, the dataset is pre-processed. When getting the spectrogram for the sounds, the number of mel bands used in the filterbank, `n_mels`, is specified as 128 by default. Models with 45, 90, and 128 mel bands are trained and tested to check how different values affected the classification. The input shape of the picture is set to be `(n_mels, 126, 1)` for the CNN model and `(n_mels, 126)` for the LSTM. 126 corresponds to a sound clip of length 4 seconds with $f_s = 16$ kHz.

The training dataset is unbalanced, with 140 more sound samples in the NJS class. To balance this out, class weight is added by running the following command:

```
Class_Weights = class_weight.compute_class_weight('balanced',
    np.unique(y_train_integer_encoded), y_train_integer_encoded)
```

which is then called upon when using `fit()` on the compiled model

```
history = model.fit(x=X_train, y=y_train,
    epochs=MAX_EPOCHS,
    batch_size=MAX_BATCH_SIZE,
    verbose=0,
    validation_data= (X_val, y_val),
    callbacks=callback,
    class_weight=Class_Weights )
```

The number of epochs is changed to avoid underfitting and overfitting. The batch size is kept more constant, and is set to 32.

3.2 Implementation

The code is written in Python 3.7.4, in the online compiler jupyter notebook. An artificial environment is created using Anaconda Navigator version 2019.10. All additional packages used is installed in the environment to prevent root access problems. See Appendix B for a step-by-step explanation of how to run the program.

The following packages with the used version are listed below, along with a short explanation.

- **Keras version 2.3.1:** A high-level API for machine learning running on top of Tensorflow, which is an end-to-end open platform for machine learning[24]. With Keras, the different layers and models are imported.
- **Librosa version 0.7.2:** Librosa is used for audio analysis and signal processing, and is used to load the sound recordings and clips with desired sampling frequency. Librosa is also used to normalize and fix the length of the training sound samples.
- **Numpy v1.18.0:** NumPy is a library that is used for handling large, multi-dimensional arrays and matrices, with support to easily operate these arrays with mathematical operations.
- **Scikit-learn version 9.23:** Scikit-learn is a Python module for machine learning, both supervised and unsupervised. It focuses on making the machine learning algorithms easy to use with a high-level language such as Python [28]. It is used to split the test and validation set by using `test_train_split()` and add weight to balance the two classes as the NJS class is outweighing the JS class with 140 sound samples.
- **h5py version 2.9.0 :** Used to save the finished trained model.

The models are trained for sampling frequency $f_s = 16$ kHz. As mentioned above, the number of filters used to take the mel spectrogram can be specified and changed and is 128 by default. The models are tested with three different numbers of filters, 45, 90, and 128 filters. Both were trained with the annotated dataset, with different input shapes depending on the number of filters/mel bands. The dataset consists of 400 clips, and is a small dataset, meaning that the NN should not be too complex in order to avoid overfitting. The CNN model was constructed with two to three layers, depending on the number of filters used in the mel spectrogram. A `MaxPooling()` layer is added, followed by a flattening layer to prepare the vector to be passed to the fully connected layer.

Both the models are compiled with the following

```
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])
```

where the loss arguments `binary_crossentropy` is used as there is only two classes to predict. The *adam* algorithm is chosen as optimizer as it is efficient and demands little memory.

The code from the Urban sound challenge [1] was used as a benchmark code.

It started as a three-layer CNN model, where 10% of the training set was split into a test set. For this classification task, the number of layers was altered, and the test set was made by using 30 % of the training dataset.

Building and training of CNN model

The code below is used to build the CNN model with input shape (45, 126, 1).

```

model = Sequential()
input_shape= X_train.shape[1:]

model.add(Conv2D(16, (3, 3), strides=(1, 1), input_shape=input_shape))
model.add(MaxPooling2D(2))
model.add(Activation('relu'))

model.add(Conv2D(24, (3, 3), strides=(1, 1), padding="valid"))
model.add(Activation('relu'))
model.add(Dropout(rate=0.5))

model.add(Flatten())
model.add(Dropout(rate=0.5))

model.add(Dense(24))
model.add(Activation('relu'))
model.add(Dropout(rate=0.5))

model.add(Dense(len(list_labels)))
model.add(Activation('softmax'))

model.summary()

```

After the build, the models end up with an output of (5, 15, 24) after the last Conv2D layer, giving a 5×15 pixel window as output. The kernel size is kept at (3, 3) as it is preferable with a small, odd-sized kernel due to reasons explained in chapter 2.2. The strides are set to (1, 1) after some testing. The activation functions are ReLU and softmax to get the probability distribution when testing as the output is in vector form.

Building and training of LSTM model

The code for building the LSTM model with input shape (90, 126) is shown below.

```

input_shape = X_train.shape[1:]

```

```

model = Sequential()

model.add(LSTM(8, input_shape=input_shape, return_sequences=True))
model.add(Activation('tanh'))

model.add(LSTM(16, return_sequences=True))
model.add(Activation('tanh'))

model.add(Flatten())
model.add(Dropout(rate=0.5))

model.add(Dense(16))
model.add(Activation('tanh'))
model.add(Dropout(rate=0.5))

model.add(Dense(len(list_labels)))
model.add(Activation('softmax'))

model.summary()

```

For the LSTM model, the activation layer is changed to tanh.

3.3 Testing of the trained models

The model is tested with different values of mel bands when computing the mel-spectrogram used to classify the sounds. It is also tested with different frequency bands in the bandpass filter applied to the pre-process clip. The upper frequency limit being 2 kHz and 4 kHz, and the lower frequency limit is 400 Hz. JS vocalizes in the range of 500-1500 Hz, and the lower limit is set to reduce the impact of wind to some degree. The upper limit is 2 kHz so that all the tops are in the spectrogram, but it will also remove some of the other bird vocalizations, e.g., chaffinch who vocalizes frequently and in the frequency range 2000-8000 Hz. The upper limit is changed to 4 kHz to see if it could give a better sound picture for the model to predict.

The best model for the different filters is stored for further testing on the recordings from Alta. Recording 6 consists of one sub-recording 6.1 containing 15h of sound recordings, and one sub-recording 6.2 that contains 9h of sound recording. Sub-recording 6.2 is cut to a smaller clip of a total length of 01:49:00, with a total of 22 vocalizations of JS. This clip will be referred to as 22JS in the following text. The vocalizations in 22JS are a mix of strong and weakly visible

vocalizations in the spectrogram. Sound clips polluted with wind of different degrees is also included. The code file `CNN_classifier.py` and `LSTM_classifier.py` shows the code used to classify JS22. The code snip for the bandpass filter and the making of the Mel spectrogram is shown below. The parameter `nr_mel` is, as mentioned, changed depending on the desired mel bands in the filterbank.

```
f_sound = butter_bandpass_filter(sound, 400, 2000, fs, order = 5)

ps = lb.feature.melspectrogram(y=f_sound, sr=fs, window = 'hann',
                               n_fft = 2048, n_mels = nr_mel)
```

For the `melspectrogram()`, an *hann* window is chosen as it has good frequency resolution and reduces spectral leakage. The length of the FFT window, `n_fft`, decides how detailed the analysis is. A narrow FFT window means a more detailed analysis of the signal, while a wider FFT window is used when there is no quick changes or disturbances in the signal [29]. For this classification, the `n_fft` was first set to 1024 to get a detailed analysis of the input signal `f_sound`. This gave some error in detecting the JS pattern, and thus indicated a too thoroughly detailed analysis. When looking at a JS vocalization in Audacity, the minimum period (T) in the signal is measured to approx. $T = 250$ ms. Setting `n_fft = 1024` corresponds to a window size of 64 ms, which is narrow. Changing to `n_fft = 2048` gives a width of the window corresponding to 128 ms, which is still smaller than the period. However, the increased width improves the spectral resolution and make the model more adaptable to recognize the correct pattern.

To find the false positive rate (FPR) and true positive rate (TPR) of 22JS, all the 4 seconds clips are stored and viewed to find the number of clips containing JS vocalization. The clip needed a minimum of 1 second of JS's vocalization to be counted as a JS positive clip. In the classification, no clips less than 1 second was classified as JS; thus, this was set as the limit. The 22JS file contained 1635 sound clips of 4 seconds, where 69 clips contain JS vocalization, and 1566 clips are negative of JS. The timestamp of all the JS vocalizations of JS22 can be found in appendix A. The Fig.3.2 illustrates one vocalization of JS lasting 12 seconds being split into three chunks of 4-second clips. This splitting is done on the total length of the recording, meaning that in 'worst-case' scenario, a vocalization lasting 12 seconds could be split into 4 chunks, which all need to be predicted to cover the whole vocalization.

A script that stores the timestamp in a .txt file to be imported to Audacity is created. It stores the start time of the clip and the end time, with tab set as

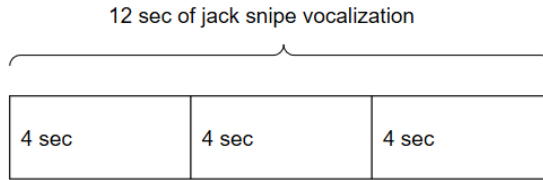


Figure 3.2: Illustration of how one vocalization of jack snipe is split into 3 chunks of 4 second clip

the delimiter. This will make it easier to see which vocalizations the model will have some challenges with classifying and provide a better understanding of how well the model works.

To further improve the classification, dynamic time warping was used on the predicted clips. The limit for maximum value for the distance was set to 50 after some testing on clips with JS to see what the typical value would be to avoid cutting out positive vocalizations. The new predicted clips are stored in a separate .txt file.

Test on full 24h recording

After finding the best classification model based on the TPR and the FPR, the total length of recording 6.1 and 6.2 is run through the best model. Recording 6.1 has a total length of approximately 15 hours of sound recordings, and recording 6.2 has approximately 9 hours. The first classification run is done without using DTW before applying DTW in the second run.

4.1 Training of model

The models are trained with the training dataset to find the combination of the variables that will give the best results. The loading of the dataset takes 9-10 seconds, and the compiling of the CNN model and the LSTM model takes between 10-15 seconds. The trained model is presented below, followed by the testing of both models on the testing sound file JS22.

4.1.1 CNN model

In Fig.4.1 the accuracy and loss function of a CNN model with input shape (128, 126, 1) is displayed. Models with input shape (90, 126, 1) and (45, 126, 1) are trained as well, where 128, 90, and 45 represent the mel bands/filters in the filter bank used in the computation of the MFCCs. For the accuracy - and

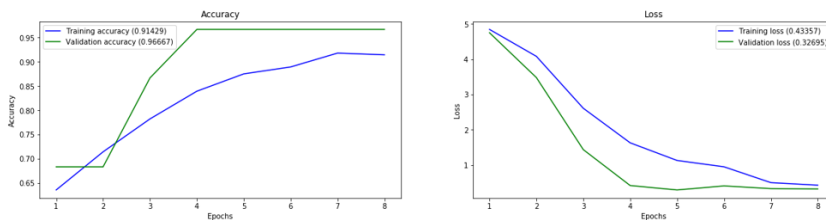


Figure 4.1: The loss and accuracy of the CNN model where 128 mel bands/- filters is used in the computation of the MFCC

loss graph of the models with input shape (45, 126, 1) and (90, 126, 1), see

Appendix C.1.

4.1.2 LSTM model

The LSTM model is also trained with three the different input sizes. In Fig. 4.2 a LSTM model with input shape (90, 126) is shown. The number of epochs is

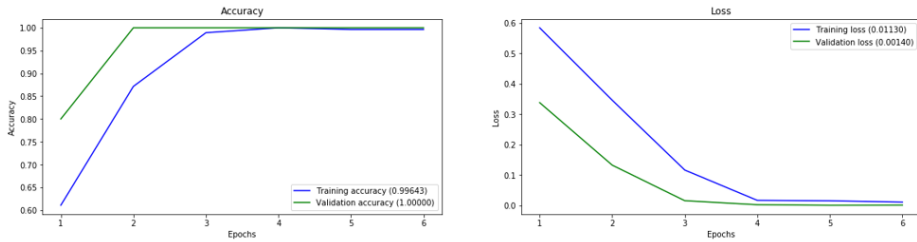
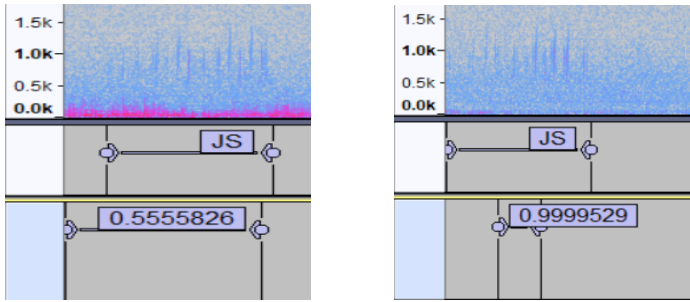


Figure 4.2: The loss and accuracy of the LSTM model where 90 mel bands/- filters is used in the computation of the MFCC

set to 6 and batch size is set to 32, achieving an accuracy of 0.98 for the test set. For the accuracy - and loss graph of the models with input shape (45, 126) and (128, 126), see Appendix C.2.

4.2 Testing of the models using test recording JS22

As mentioned in 3.2, a test recording with 22 vocalizations of JS is created to test the trained models. Six tests were conducted for each model; the first three with the number of mel bands changed between 45, 90, and 128, and then again for the same variations in mel bands but with added DTW to the positive classified clips. All the results for the CNN - and LSTM model can be viewed in Appendix D and E respectively. In Fig.4.3, two annotated vocalizations with the corresponding predicted clips are displayed. The predicted clip is annotated with the confidence interval.



(a) Annotated vocalization of jack (b) Annotated vocalization of jack snipe with the belonging confidence interval of the predicted clip.

Figure 4.3: Both (a) and (b) shows the predicted clips with belonging confidence interval generated by the CNN model using 90 mel bands.

4.2.1 CNN model

For the CNN model, the best TPR is achieved when there are 90 mel bands in the frequency bank, and the frequency range is set to 400-2000 Hz for the band pass filter. This gives the model the input shape (90, 126, 1). In Tab. 4.1 the confusion matrix with TPR and FPR is shown for the CNN model with applied band-pass filter with frequency range 400-2000 Hz. Here, one can see that the

Table 4.1: Confusion matrix for CNN model with 90 mel bands, and an applied band pass filter with range 400-2000 Hz.

	Actually positive	Actually negative
Predicted positive	67	1465
Predicted negative	2	101
TPR	0.97	
FPR	0.94	

FPR is very high, and when viewing the clip in Audacity it shows that many of the classified positive clips are minutes long. When calculating the total length of the all the positive clips, the length is 01:45:16 (hh:mm:sec), meaning that the model classifies almost the total length of JS22 as positive. DTW is then applied to the classifications, with limit < 50 . The results of this can be seen from Tab.4.2. Here, one can see that the FPR is reduced. The total length of the clips classified as positive is reduced to 00:13:04.

Table 4.2: Confusion matrix for CNN model with 90 mel bands, an applied band pass filter with range 400-2000 Hz. Additionally, DTW is used on the classified clips.

	Actually positive	Actually negative
Predicted positive	61	135
Predicted negative	8	1431
TPR	0.88	
FPR	0.086	

4.2.2 LSTM model

The LSTM model with input shape (45, 126), where the bandpass filter used on the clip has frequency range 400-2000 Hz, gives the results best results. This is listed in Tab. 4.3. The FPR is high, and the total length of the predicted positive

Table 4.3: Confusion matrix for LSTM model with 45 mel bands, and an applied band pass filter with range 400-2000 Hz.

	Actually positive	Actually negative
Predicted positive	51	1225
Predicted negative	18	341
TPR	0.74	
FPR	0.78	

clips is 01:25:04. Adding DTW to the classified clips with limit <50, gives the results shown in Tab. 4.4.

Table 4.4: Confusion matrix for LSTM model with 45 mel bands, and an applied band pass filter with range 400-2000 Hz. Additionally, DTW is used on the classified clips.

	Actually positive	Actually negative
Predicted positive	3	46
Predicted negative	66	1521
TPR	0.043	
FPR	0.029	

4.3 Vocalizations detected fully or partly

The JS22 file consists of 22 clips of JS vocalization of different length. By looking only at the vocalizations without the dividing of 4 second chunks, the positive class is reduced to the number of JS vocalization, which is 22. The CNN model with mel bands equal to 90 and a frequency range 400-4000 Hz for the band-pass filter gives the best result in this scenario, where 16/22 of the clips are recognised. This gives a FP = 142, and the total length of the predicted positive clips are 00:47:04. In Tab. 4.5 the results from partly detected clips are listed. Note that the length of the predicted clips for 90 mel bands with out DTW

Table 4.5: Vocalizations detected partly or fully by the different trained CNN models.

CNN	2 kHz		4 kHz	
	wo. DTW	DTW	wo. DTW	DTW
45 mel bands	14	13	9	8
90 mel bands	22	22	16	16
128 mel bands	20	13	10	9

is 01:42:08.

4.4 Testing on long recording

The best model, the CNN model with 90 mel bands and frequency range 400-2000 Hz, is run on the full length of recording 6.1 and 6.2 to see the process time and the number of predicted positive clips. The 6.1 recording is tested first. The wall time without DTW is 00:08:48, and gives 629 predictions. The total length for all the predictions is 13:33:28. When DTW is added to the classification, the wall time is 00:56:58, and it produces 42 predictions with summarized length of 13:33:12. When viewed in Audacity, the predicted clips were long, on average 00:18:22, and gave very high likelihood for these clips. The longest clips appeared in the parts of the recording that was heavily influenced by wind, see Fig. 4.4

Recording 6.2 is tested after 6.1. This recording (6.2) is the recording in which the JS22 is made out of. Without DTW, the wall time is 00:04:14 and results in 5733 positive predicted clips, collected to 61 clips in the directory where the predicted sound files are stored. The total length for the predicted clips was 06:22:12. After using DTW, the wall time is 00:54:17 and 2177 positive pre-

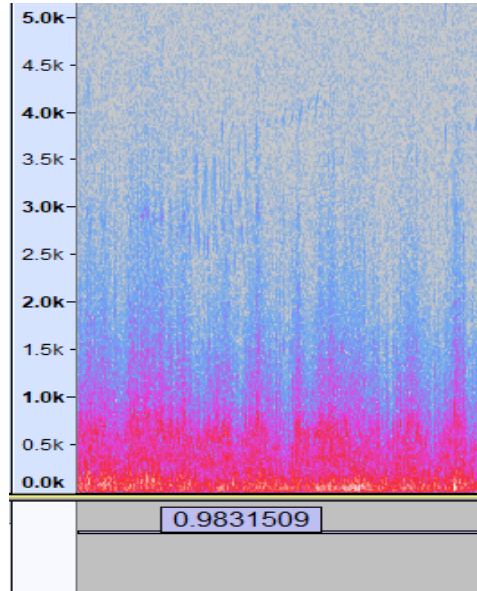


Figure 4.4: A prediction done on recording 6.1 with much wind. The y-axis shows the frequency, while the tag indicated the confidence interval.

dicted clips are detected, with a total of 469 clips being stored in the directory. Total length of the predictions is 02:25:08.

By testing different mel bands' values, it was interesting to see how well the classification models worked for each model.

5.1 The training of the models

The dataset used to train the models is very small, consisting of only 400 samples. The small dataset makes it hard to train a model, as it gets overfitted easily. A model with few hidden layers and nodes is used to minimize the overfitting, but either way, the models reach between 0.96-1.0 on accuracy for the training and validation and dataset. A model with many layers or a large number of nodes in the layers results in a poor model with a high loss value.

5.2 Validation of the models

As can be seen from the results, the classification with applied $f_u = 2$ kHz, gave a better TP for all tests. When looking closer into the clips classified here, the clips were often very long and covered more than one vocalization of JS in one positive classification. The most extended clip is 17 min and 36 s and contains six vocalizations of JS, which correspond to a quarter of the total length of 22JS.

The classification model gives large clusters of positive classifications in areas where the wind are noticeable. JS vocalizes in the range 500-1500 Hz, meaning that wind present in this frequency range would make it hard for the model to identify the vocalizations of JS. The idea of taking in the probability of vocal-

ization of JS in the predicted sound file name was to easier prioritize the clip that likely contains JS. When testing on the JS22, and importing the labels to Audacity, it became clear that the given probability of a predicted clip does not necessarily give a good picture of the likelihood of a positive vocalization. For the models with mel bands equal 45, the length of the predicted clips is long. For the vocalizations with the presence of wind, the probability was lower than for the clean clips. The vocalizations that had wind in the 400 Hz frequency proved more challenging to predict, and was part of larger predicted clips with length of minutes. To reduce the number of FP, the DTW function is applied to the positive predicted clips. When trying to set a minimum distance limit,

The model classifies river noise and crickets as JS with a high probability (over 0.9) at first. Two actions were done to improve this, one in the training of the model and the other in the pre-processing of the input signal to be classified. In training, clips of crickets and river noise are added to the model's training dataset before the model is compiled again. In the pre-process of the signal, where the Mel spectrogram is generated, the width of the FFT window is altered to improve the spectral resolution and changed from 1024 to 2048. These actions reduced the probability score and the number of FP on these clips. The challenges when trying to improve on the river noise, is that is in the same frequency range as JS.

As the model have challenges when there is a high presence of wind and river noise, it would be beneficial to try to minimize these types of environmental sounds when setting up the recorder.

True positive and False positives

After importing the labeled predicted clips to Audacity, some similarity with the false positive represents themselves. The majority of the false positive clips contain wind, river noise and crickets. The cricket gave a very high probability of JS, coming out in the upper 90 percentile when the model is run at a frequency range of 400-2000 Hz and a low number of mel bands. As the frequency range is changed to 400-4000 Hz, these clips are reduced or removed from the models with the different mel bands.

As the TPR is calculated from the total number of positive clips, some models detected a higher amount of the vocalizations than the rate indicates. When counting vocalizations detected, some of the models with lower TPR recognized a part of many of the vocalizations, i.e., recognized one clip out of the three that made up the vocalization as displayed in Fig. 3.1. If one only looks

at how many of the vocalizations that were partly or fully detected, the CNN model with 90 mel bands in the filterbank, applied DTW and bandpass filter with frequency range 400-2000 Hz, find all the 22 vocalizations. Also, for 45 mel bands (without DTW), the results are better, where the model finds 14/22 vocalizations, and the total length of the positive predictions is 11 min and 24 s. For the 45 mel band CNN model, the hits decreased when DTW was applied. The limit was increased, but it proved hard to find a limit that would keep the FP down and the TP on a constant level. As DTW compares two signals as explained in Ch. 2.5, it could mean that the SNR in the predicted clip is low and thus makes it difficult to recognize as JS vocalization.

5.3 CNN vs LSTM

When viewing the positive classifications in Audacity, it shows that, on average, the clips classified by the LSTM are longer and have a much poorer FPR. The LSTM model gets overfitted easily, and the results could indicate that the trained model generalizes badly as the predicted clips often stretch over long time intervals. The duration of one JS vocalization is around 8-12 seconds on average, so the predicted clip's duration should be between 8-16 seconds if all the clips contain JS.

The CNN model trained with input size (90, 126, 1) and with applied bandpass filter with frequency range 400-2000 Hz detects the most JS positive clips, with a TPR = 0.88 after applying DTW. This model gets a high hit rate for both frequency ranges, and this could indicate that more mel bands are efficient when trying to predict clips with the presence of wind in them. The LSTM model had a TPR = 0.74 before DTW, and FPR = 0.78. The predicted clips were minutes long when viewed in Audacity, and thus the FPR is high as the total length of all the predicted clips is 01:24:04 long, meaning that 78 % of JS22 was predicted to contain JS. These results could indicate that the model is bad at generalizing for JS vocalization.

When the model is trained with fewer filters in the Mel spectrogram, many of the predicted clips are longer than for models trained with 90 and 128 filters. The long clips could indicate that the trained model is underfitted, as that would make it hard to recognize patterns. At the same time, it is a few clips of grouse, which vocalizes in the frequency range 400-1000 Hz, that are classified as JS. There are more than one vocalizations from grouse in the training dataset. The grouse vocalization signal has longer periods than the JS vocalization signal and lasts for approx. 2 s, indicating that the model does recognize

a pattern.

5.4 Testing on full length recording

The CNN model that uses 90 mel bands gave the best result and is used on recording 6. Recording 6 is split into two sub-recordings, where recording 6.1 lasting 14:57:57 and recording 6.2 lasting 08:56:32. When viewed in Audacity, recording 6.1 shows much wind in the spectrogram; thus, it might be hard to predict on this sub-recording as the SNR value is low. Recording 6.2 does also has a high presence of wind in it, but it is not persistent through the whole recording. Thus, it would be reasonable to think that predictions on 6.2 will be more accurate than for 6.1.

Recording 6.1 is tested first. The first run with no DTW and frequency range 400-2000 Hz returned predictions with a total length of almost the same length as the recording, lasting in a total of 13 hours and 43 minutes, and returning 673 positive clips. As the goal for the classification is to reduce the time used on viewing and listening to hours of recordings in Audacity, this would be a poor outcome. DTW is applied, and reduces the total length of the predicted sound considerably to 13:33:12, with 42 positive predictions. The positive recordings were long and the confidence interval was very high in parts with a lot of wind.

Then recording 6.2 is tested. This recording had less wind in the beginning, and JS22 is made out of this part. The wall time for this recording was 4 min and 14 s and gave a total of 61 positive predicted clips with a total length of 6 hours 22 min and 12 s. With 61 predicted positives, this gives an average length for each prediction of 6 min and 16 s, meaning that the model has some challenges recognizing the patterns for JS vocalization when there is a presence of wind and/or river noise.

When using DTW, the wall time is 00:54:17, and it returned 469 positive predicted clips that are stored. The total length of the predictions is 02:25:08, which is a reduction of 55 % compared to the original 6.2 recording. After viewing the predicted clips in Audacity, the model has predicted minutes long clip with high probability.

CHAPTER 6

CONCLUSION

The presence of jack snipe is detected when running the trained model on a sound recording. The detected clips are stored in a directory and the start and end time is transmitted to a .txt file with the confusion interval as the tag for the clip to be imported to Audacity. By viewing these files, it was easier to detect patterns to where the classifier failed and find the TPR and FPS. It was found that a CNN model with mel bands = 90 gave the best result, but with a high FPR = 0.94, where many of these clips contained wind and river noise. This indicates that the training of the model worked, but the classification could be improved as it has some challenges detecting samples where wind or river noise is present (see sec ??). After applying DTW on the predicted clips, the FPR was reduced significantly, while the TPR was still high with TPR = 0.88. The total length of all the predicted samples is 00:13:04, reducing the data from its original 01:49:00.

When starting, the goal was to reduce the time a person needs to use to identify JS vocalization by automating as much as possible of the work flow. The model with the best results, CNN with 90 mel bands and using DTW function, had a TPR = 0.88 and FPR = 0.086, and reduced the recording to around 13 min. As such there is an improvement compared to the starting point (which was a recording of length 01:49:00) and this program developed will be beneficial to the end user when the problem with high presence of wind and river noise in the FP is solved.

6.1 Future work

Some learning points from this project, is that results could be further improved by following up on the wind and river noise problematic as discussed in Ch. 5.2. As the width of the FFT window decides the spectral resolution of the spectrogram, wavelet analysis could be interesting to look into. Wavelet analysis adapts to the changes in the signal, meaning that it could provide a detailed analyses (narrower FFT window) when a change in the signal is detected, and run through the signal fast otherwise by using a wider FFT window. This could reduce the length of the predicted clips, and possibly remove some of the FP predictions on clips containing wind and river noise.

The annotated training dataset is small, making it easy to overfit and harder to make a robust model. For future work, the training dataset should be made larger, containing more vocalizations of jack snipe. It would also be interesting to include more species in the training dataset, and use unsupervised learning. The model would learn without demanding further input from the developer and give out more data for the end user to use.

LSTM CNN model could be an interesting model to try out, as the LSTM model got many hits on the positive clips, but struggled with many FP and very long predictions. The models could also be trained on a higher sampling frequency, as this would improve the resolution for the spectrogram and thus may make the detection of the pattern easier for the model.

BIBLIOGRAPHY

- [1] Smartsheet, *Real-life and buissness application of neural network*, <https://github.com/praveendhaked/Urban-Sound-Classification/blob/master/CNN.ipynb>.
- [2] Smartsheet, *Real-life and buissness application of neural network*, <https://www.smartsheet.com/neural-network-applications>.
- [3] Z. Zhao, S.-h. Zhang, Z.-y. Xu, K. Bellisario, N.-h. Dai, H. Omrani and B. C. Pijanowski, 'Automated bird acoustic event detection and robust species classification', *Ecological Informatics*, vol. 39, pp. 99–108, 2017, ISSN: 1574-9541. DOI: <https://doi.org/10.1016/j.ecoinf.2017.04.003>. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S157495411630231X>.
- [4] D. Stowell, M. Wood, H. Pamua, Y. Stylianou and H. Glotin, 'Automatic acoustic detection of birds through deep learning: The first bird audio detection challenge', *Methods in Ecology and Evolution*, vol. 10, Oct. 2018. DOI: 10.1111/2041-210X.13103.
- [5] M. Malfante, J. Mars, M. Dalla Mura and C. Gervaise, 'Automatic fish sounds classification', *The Journal of the Acoustical Society of America*, vol. 143, pp. 2834–2846, May 2018. DOI: 10.1121/1.5036628.
- [6] G. Tzanetakis and P Cook, 'Musical genre classification of audio signals', *IEEE Transactions on Speech and Audio Processing*, vol. 10, pp. 293–302, Jan. 2002.
- [7] D. Stowell and M. Plumbley, 'Automatic large-scale classification of bird sounds is strongly improved by unsupervised feature learning', *PeerJ*, vol. 2, e488, Jul. 2014. DOI: 10.7717/peerj.488.

- [8] S. Kahl, T. Wilhelm-Stein, H. Hussein, H. Klinck, D. Kowerko, M. Ritter and M. Eibl, ‘Large-scale bird sound classification using convolutional neural networks’, Sep. 2017.
- [9] K. Qian, Z. Zhang, F. Ringeval and B. Schuller, ‘Bird sounds classification by large scale acoustic features and extreme learning machine’, Dec. 2015, pp. 1317–1321. DOI: 10.1109/GlobalSIP.2015.7418412.
- [10] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. J. Weiss and K. Wilson, ‘Cnn architectures for large-scale audio classification’, in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2017, pp. 131–135. DOI: 10.1109/ICASSP.2017.7952132.
- [11] V. Morfi, Y. Bas, H. Pamua, H. Glotin and D. Stowell, ‘Nips4bplus: A richly annotated birdsong audio dataset’, *PeerJ Computer Science*, vol. 5, e223, Oct. 2019. DOI: 10.7717/peerj-cs.223.
- [12] D. Stowell and F. Wilkinson, *Wablr*, <https://www.wablr.co.uk/>, 2015.
- [13] abelmagic, *Whatbird*, <http://whatbird.no>, 2018.
- [14] M. R. M. Talabis, R. McPherson, I. Miyamoto, J. L. Martin and D. Kaye, ‘Chapter 1 - analytics defined’, in *Information Security Analytics*, M. R. M. Talabis, R. McPherson, I. Miyamoto, J. L. Martin and D. Kaye, Eds., Boston: Syngress, 2015, p. 2, ISBN: 978-0-12-800207-0. DOI: <https://doi.org/10.1016/B978-0-12-800207-0.00001-0>. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/B9780128002070000010>.
- [15] D. Anderson and G. McNeill, ‘Artificial neural networks technology’, *Kaman Sciences Corporation*, vol. 258, no. 6, pp. 1–83, 1992.
- [16] B. Khuong, *The basics of recurrent neural networks (rnns)*, <https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-alleffb7808f>, 2019.
- [17] D. Tomè, F. Monti, L. Baroffio, L. Bondi, M. Tagliasacchi and S. Tubaro, ‘Deep convolutional neural networks for pedestrian detection’, *Signal Processing: Image Communication*, vol. 47, pp. 482–489, 2016, ISSN: 0923-5965. DOI: <https://doi.org/10.1016/j.image.2016.05.007>. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0923596516300637>.
- [18] Stanford, *Convolutional neural network*, <http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/>.

- [19] M. West, *Convolutional neural network: The theory*, <https://www.bouvet.no/bouvet-deler/understanding-convolutional-neural-networks-part-1>, 2019.
- [20] M. Hasan, M. Jamil, G. Rabbani and M. S. Rahman, 'Speaker identification using mel frequency cepstral coefficients', *Proceedings of the 3rd International Conference on Electrical and Computer Engineering (ICECE 2004)*, Dec. 2004.
- [21] A. V. Oppenheim and R. W. Schaffer, 'From frequency to quefrequency: A history of the cepstrum', *IEEE Signal Processing Magazine*, vol. 21, no. 5, pp. 95–106, Sep. 2004, ISSN: 1558-0792. DOI: 10.1109/MSP.2004.1328092.
- [22] S. Ruder, 'An overview of gradient descent optimization algorithms', pp. 2–3, Sep. 2016.
- [23] J. Brownlee, *How to improve deep learning model robustness by adding noise*, <https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/>.
- [24] F. Chollet *et al.*, *Keras*, <https://keras.io>, 2015.
- [25] J. Brownlee, *Overfitting and underfitting with machine learning algorithms*, <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>.
- [26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, 'Dropout: A simple way to prevent neural networks from overfitting', *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, Jun. 2014.
- [27] 'Dynamic time warping', in *Information Retrieval for Music and Motion*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 69–84, ISBN: 978-3-540-74048-3. DOI: 10.1007/978-3-540-74048-3_4. [Online]. Available: https://doi.org/10.1007/978-3-540-74048-3_4.
- [28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and É. Duchesnay, 'Scikit-learn: Machine learning in python', *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: <http://jmlr.org/papers/v12/pedregosa11a.html>.
- [29] M. LLC. (1999). *Librosa.feature.melspectrogram*, [Online]. Available: <https://librosa.org/librosa/master/generated/librosa.feature.melspectrogram.html> (visited on 25/06/2020).

APPENDIX **A**

TIMESTAMP FOR JACK SNIPE VOCALIZATION IN TEST
CLIP JS22

Sound clips containing jack snipe vocalization	hh:min:sec
	00:01:40
	00:01:44
clip 1	00:01:48
	00:06:32
	00:06:36
	00:06:40
clip 2	00:06:44
	00:16:52
	00:16:56
	00:17:00
clip 3	00:17:04
	00:24:04
	00:24:08
clip 4	00:24:12
	00:41:40
	00:41:44
	00:41:48
clip 5	00:41:52
	00:46:40
	00:46:44
	00:46:48
clip 6	00:46:52
	00:53:44
	00:53:48
	00:53:52
clip 7	00:53:56
	00:55:52
clip 8	00:55:56
	00:59:08
	00:59:12
clip 9	00:59:16
	01:00:32
	01:00:36
clip 10	01:00:40
	01:04:12
	01:04:16
clip 11	01:04:20
	01:08:28
	01:08:32
	01:08:36
clip 12	01:08:40
	01:13:40
	01:13:44
clip 13	01:13:48
	01:15:16
clip 14	01:15:20
	01:16:52
	01:16:56

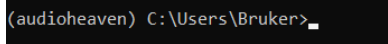
clip 15	01:17:00
	01:20:36
	01:20:40
clip 17	01:20:44
	01:22:12
	01:22:16
clip 18	01:22:20
	01:23:48
	01:23:52
clip 19	01:23:56
	01:28:56
	01:29:00
	01:29:04
clip 20	01:29:08
	01:30:36
	01:30:40
clip 21	01:30:44
	01:47:00
	01:47:04
clip 22	01:47:08
Total number of 4 seconds clip	69

APPENDIX B

HOW TO SET UP AND RUN THE TRAINING/CLASSIFICATION SCRIPT

B.1 Packages to install

Before installing the required packages, make an artificial environment on your computer and activate it in terminal. An easy way to do this is by using Anaconda navigator and create a new environment. When this is done, you open Terminal in this environment. It should then look like this: and install



```
(audioheaven) C:\Users\Bruker>_
```

Figure B.1: How the terminal window should look when the environment *audioheaven* is activated.

the following packages by typing *pip install PACKAGE NAME* in the Terminal window:

- numpy
- librosa
- scikit-learn
- h5py
- panda
- os
- matplotlib

B.2 Training file

A .csv file containing the file ID and the associated class. In the .csv file, the filename of the sound file and the class of prediction (either Jack snipe or NJS) is listed in the same column, separated by , and no white space. See Fig.B.2. The dataset is stored in a sub-directory in the same directory as the .csv file is

A
file_ID,Class
filter_sample1,Jack Snipe
filter_sample3,Jack Snipe
filter_sample4,Jack Snipe
filter_sample2,Jack Snipe
filter_sample5,Jack Snipe
filter_sample7,Jack Snipe
filter_sample12,Jack Snipe
filter_sample13,Jack Snipe

Figure B.2: How the training sound files are listed in the .csv file. file_ID and Class are the two headers. Jack snipe is one of the classes, the other one being NJS.

stored. When running *trained_model_16000.py*, change the path to where the .csv file and the directory storing the dataset are stored on your computer.

B.3 Running the trained model

To run the model on a recording, use *CNN_classifier.py*. Make sure the recording is stored in the same directory as the script, or enter the full path. Depending on how many *n_mels* the model is trained on (45, 90 or 128), make sure *n_mels* matches this. You can choose what to call the directory to where the sound files will be saved and the corresponding .txt file that will be used to import the timestamps to Audacity.

APPENDIX C TRAINED MODELS

C.1 Trained CNN graph and architecture

CNN with mel bands = 45

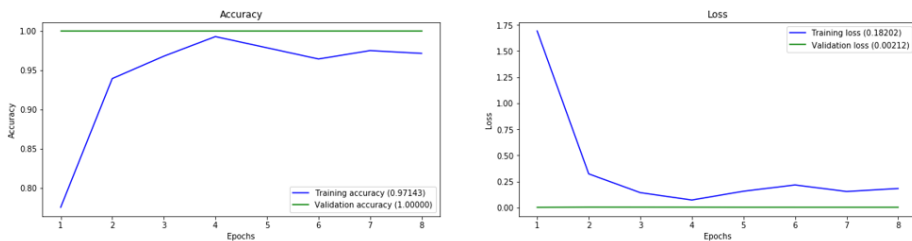


Figure C.1: Trained model with input shape (45, 126, 1)

With the below building architecture

```
model = Sequential()
input_shape= X_train.shape[1:]

model.add(Conv2D(16, (3, 3), strides=(1, 1), input_shape=input_shape))
model.add(MaxPooling2D(2))
model.add(Activation('relu'))

model.add(Conv2D(24, (3, 3), strides=(1, 1), padding="valid"))
model.add(Activation('relu'))
model.add(Dropout(rate=0.5))
```

```

model.add(Flatten())
model.add(Dropout(rate=0.5))

model.add(Dense(24))
model.add(Activation('relu'))
model.add(Dropout(rate=0.5))

model.add(Dense(len(list_labels)))
model.add(Activation('softmax'))

model.summary()

```

CNN with mel bands = 90

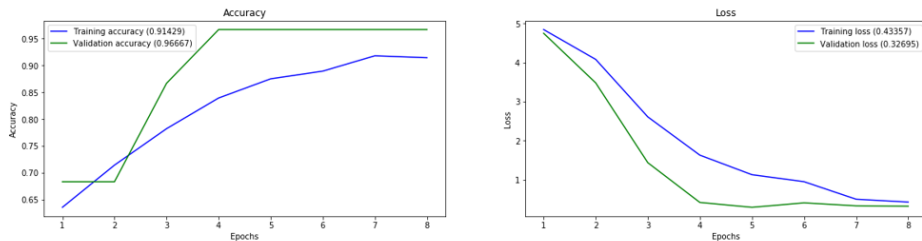


Figure C.2: LEGG INN RIKTIG FIGUR(90, 126, 1)

```

model = Sequential()
input_shape= X_train.shape[1:]

model.add(Conv2D(16, (3, 3), strides=(2, 2), input_shape= input_shape))
model.add(MaxPooling2D(2))
model.add(Activation('relu'))

model.add(Conv2D(24, (3, 3), strides=(2, 2), padding="valid"))
model.add(MaxPooling2D(2))
model.add(Activation('relu'))
model.add(Dropout(rate=0.3))

model.add(Flatten())
model.add(Dropout(rate=0.3))

model.add(Dense(24))
model.add(Activation('relu'))

```

```

model.add(Dropout(rate=0.3))

model.add(Dense(len(list_labels)))
model.add(Activation('softmax'))

model.summary()

```

CNN with mel bands = 128

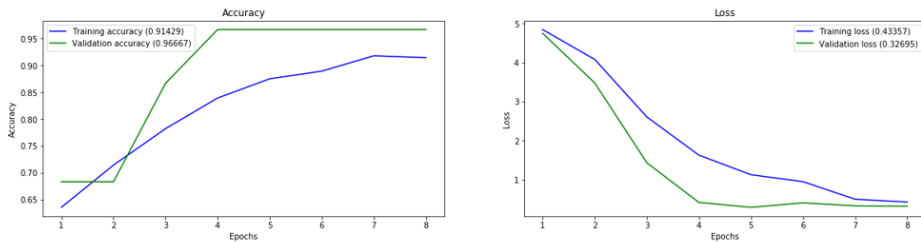


Figure C.3: Trained model with input shape (128, 126, 1)

```

model = Sequential()
input_shape= X_train.shape[1:]

model.add(Conv2D(16, (3, 3), strides=(1, 1), input_shape=input_shape))
model.add(MaxPooling2D(2))
model.add(Activation('relu'))

model.add(Conv2D(24, (3, 3), strides=(1, 1), padding="valid"))
model.add(MaxPooling2D(2))
model.add(Activation('relu'))
model.add(Dropout(rate=0.5))

model.add(Conv2D(36, (3, 3), strides=(1, 1), padding="valid"))
model.add(MaxPooling2D(2))
model.add(Activation('relu'))
model.add(Dropout(rate=0.5))

model.add(Flatten())
model.add(Dropout(rate=0.5))

model.add(Dense(24))
model.add(Activation('relu'))

```

```
model.add(Dropout(rate=0.5))  
  
model.add(Dense(len(list_labels)))  
model.add(Activation('softmax'))  
  
model.summary()
```

C.2 LSTM

LSTM with mel bands = 45

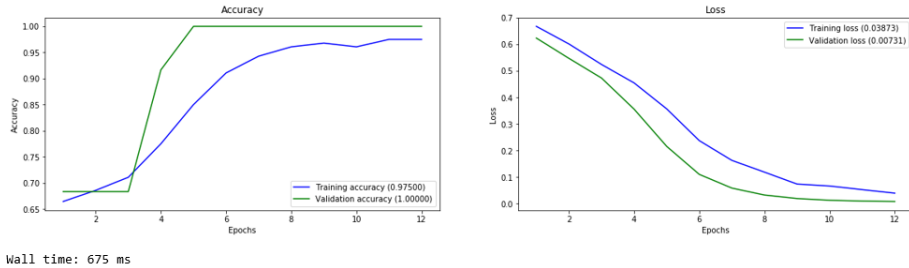


Figure C.4: Trained model with input shape (45, 126)

with the following building architecture:

```

model = Sequential()

model.add(LSTM(8, input_shape=input_shape, return_sequences=True))
model.add(Activation('tanh'))

model.add(LSTM(16, return_sequences=True))
model.add(Activation('tanh'))

model.add(Flatten())
model.add(Dropout(rate=0.5))

model.add(Dense(16))
model.add(Activation('tanh'))
model.add(Dropout(rate=0.5))

model.add(Dense(len(list_labels)))
model.add(Activation('softmax'))

model.summary()

```

LSTM with mel bands = 90

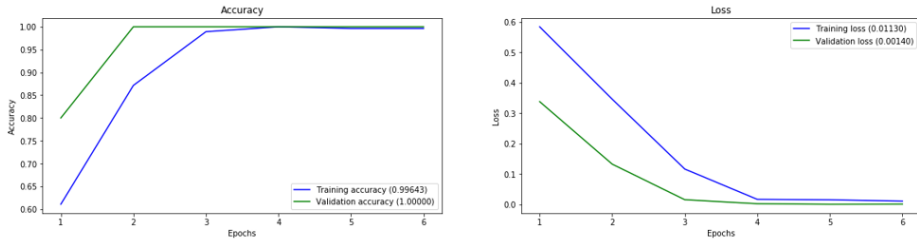


Figure C.5: Trained model with input shape (90, 126)

The building architecture is equal for all the LSTM models.

LSTM with mel bands = 128

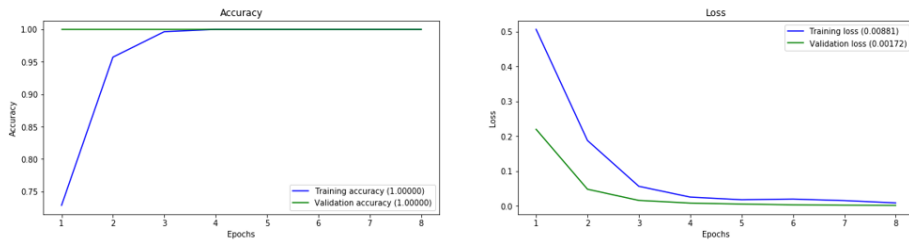


Figure C.6: Trained model with input shape (128, 126)

The building architecture is equal for all the LSTM models.

APPENDIX D

CONFUSION MATRIX, TPR AND FPR FOR THE CNN MODEL

D.1 CNN with 45 mel bands

With frequency range 400-2000 Hz

Table D.1: Confusion matrix for CNN model with 45 mel bands, and an applied band pass filter with range 400-2000 Hz.

Predicted positive	Actually positive	Actually negative
	41	130
Predicted negative	28	1436
TPR	0.594	
FPR	0.083	

Table D.2: Confusion matrix for CNN model with 45 mel bands, and an applied band pass filter with range 400-2000 Hz. Additionally, DTW is used on the classified clips.

Predicted positive	Actually positive	Actually negative
	37	125
Predicted negative	32	1441
TPR	0.536	
FPR	0.080	

With frequency range 400-4000 Hz

Table D.3: Confusion matrix for CNN model with 45 mel bands, and an applied band pass filter with range 400-4000 Hz.

Predicted positive	Actually positive 11	Actually negative 3
Predicted negative	58	1563
TPR	0.159	
FPR	0.002	

Table D.4: Confusion matrix for CNN model with 45 mel bands, and an applied band pass filter with range 400-4000 Hz. Additionally, DTW is used on the classified clips.

Predicted positive	Actually positive 7	Actually negative 7
Predicted negative	62	1559
TPR	0.101	
FPR	0.004	

D.2 CNN with 90 mel bands

With frequency range 400-2000 Hz

Table D.5: Confusion matrix for CNN model with 90 mel bands, and an applied band pass filter with range 400-2000 Hz.

Predicted positive	Actually positive 67	Actually negative 1465
Predicted negative	2	101
TPR	0.971	
FPR	0.936	

With frequency range 400-4000 Hz

Table D.6: Confusion matrix for CNN model with 90 mel bands, an applied band pass filter with range 400-2000 Hz. Additionally, DTW is used on the classified clips.

Predicted positive	Actually positive 61	Actually negative 135
Predicted negative	8	1431
TPR	0.884	
FPR	0.086	

Table D.7: Confusion matrix for CNN model with 90 mel bands, and an applied band pass filter with range 400-4000 Hz.

Predicted positive	Actually positive 39	Actually negative 667
Predicted negative	30	899
TPR	0.565	
FPR	0.426	

Table D.8: Confusion matrix for CNN model with 90 mel bands, an applied band pass filter with range 400-4000 Hz. Additionally, DTW is used on the classified clips.

Predicted positive	Actually positive 18	Actually negative 682
Predicted negative	51	884
TPR	0.261	
FPR	0.436	

D.3 CNN with 128 mel bands

With frequency range 400-2000 Hz

Table D.9: Confusion matrix for CNN model with 128 mel bands, and an applied band pass filter with range 400-2000 Hz.

Predicted positive	Actually positive 45	Actually negative 904
Predicted negative	24	662
TPR	0.652	
FPR	0.577	

Table D.10: Confusion matrix for CNN model with 128 mel bands, an applied band pass filter with range 400-2000 Hz. Additionally, DTW is used on the classified clips.

	Actually positive	Actually negative
Predicted positive	43	158
Predicted negative	26	1408
TPR	0.623	
FPR	0.100	

With frequency range 400-4000 Hz

Table D.11: Confusion matrix for CNN model with 128 mel bands, and an applied band pass filter with range 400-4000 Hz.

	Actually positive	Actually negative
Predicted positive	27	453
Predicted negative	42	1113
TPR	0.391	
FPR	0.289	

Table D.12: Confusion matrix for CNN model with 128 mel bands, an applied band pass filter with range 400-4000 Hz. Additionally, DTW is used on the classified clips.

	Actually positive	Actually negative
Predicted positive	21	343
Predicted negative	48	1223
TPR	0.304	
FPR	0.219	

APPENDIX E

CONFUSION MATRIX, TPR AND FPR FOR THE LSTM MODEL

E.1 LSTM with 45 mel bands

With frequency range 400-2000 Hz

Table E.1: Confusion matrix for LSTM model with 45 mel bands, and an applied band pass filter with range 400-2000 Hz.

Predicted positive	Actually positive	Actually negative
	51	1225
Predicted negative	18	341
TPR	0.739	
FPR	0.782	

Table E.2: Confusion matrix for LSTM model with 45 mel bands, and an applied band pass filter with range 400-2000 Hz. Additionally, DTW is used on the classified clips.

Predicted positive	Actually positive	Actually negative
	3	46
Predicted negative	66	1521
TPR	0.043	
FPR	0.029	

With frequency range 400-4000 Hz

Table E.3: Confusion matrix for LSTM model with 45 mel bands, and an applied band pass filter with range 400-4000 Hz.

Predicted positive	Actually positive	Actually negative
Predicted negative	24	358
	45	1208
TPR	0.347	
FPR	0.229	

Table E.4: Confusion matrix for LSTM model with 45 mel bands, and an applied band pass filter with range 400-4000 Hz. Additionally, DTW is used on the classified clips.

Predicted positive	Actually positive	Actually negative
Predicted negative	7	7
	62	1559
TPR	0.101	
FPR	0.004	

E.2 LSTM with 90 mel bands

With frequency range 400-2000 Hz

Table E.5: Confusion matrix for CNN model with 90 mel bands, and an applied band pass filter with range 400-2000 Hz.

Predicted positive	Actually positive	Actually negative
Predicted negative	41	933
	28	633
TPR	0.594	
FPR	0.596	

With frequency range 400-4000 Hz

Table E.6: Confusion matrix for LSTM model with 90 mel bands, and an applied band pass filter with range 400-2000 Hz. Additionally, DTW is used on the classified clips.

Predicted positive	Actually positive 3	Actually negative 42
Predicted negative	66	1524
TPR	0.043	
FPR	0.027	

Table E.7: Confusion matrix for LSTM model with 90 mel bands, and an applied band pass filter with range 400-4000 Hz.

Predicted positive	Actually positive 16	Actually negative 300
Predicted negative	53	1266
TPR	0.231	
FPR	0.192	

Table E.8: Confusion matrix for LSTM model with 90 mel bands, and an applied band pass filter with range 400-4000 Hz. Additionally, DTW is used on the classified clips.

Predicted positive	Actually positive 1	Actually negative 20
Predicted negative	68	1546
TPR	0.014	
FPR	0.013	

E.3 LSTM with 128 mel bands

With frequency range 400-2000 Hz

Table E.9: Confusion matrix for LSTM model with 128 mel bands, and an applied band pass filter with range 400-2000 Hz.

Predicted positive	Actually positive 38	Actually negative 972
Predicted negative	31	663
TPR	0.550	
FPR	0.594	

Table E.10: Confusion matrix for LSTM model with 128 mel bands, and an applied band pass filter with range 400-2000 Hz. Additionally, DTW is used on the classified clips.

	Actually positive	Actually negative
Predicted positive	3	43
Predicted negative	66	1524
TPR	0.043	
FPR	0.027	

With frequency range 400-4000 Hz

Table E.11: Confusion matrix for LSTM model with 128 mel bands, and an applied band pass filter with range 400-4000 Hz.

	Actually positive	Actually negative
Predicted positive	10	38
Predicted negative	59	1528
TPR	0.145	
FPR	0.024	

Table E.12: Confusion matrix for LSTM model with 128 mel bands, and an applied band pass filter with range 400-4000 Hz. Additionally, DTW is used on the classified clips.

	Actually positive	Actually negative
Predicted positive	2	1
Predicted negative	67	1565
TPR	0.029	
FPR	0.0006	

APPENDIX F

LIST OF BIRD SPECIES

Norsk	English
Kvartbekkasin	Jack snipe
Dverggås	Lesser white
Stokkand	Mallard
Stjertand	Pintail
Brunnakke	Wigeon
Krikkand	Common teal
Toppand	Tufted duck
Havelle	Long-tailed duck
Kvinand	Goldeneye duck
Lirype	Grouse
Storlom	black-throated diver
Fjellvåk	Rough-legged hawk
Dvergfalk	Merlin
Sandlo	Killdeer
Heilo	Plovers
Temmincksnipe	Temminck's stint
Brushane	Ruff
Grønnstilk	Wood Sandpiper
Strandsnipe	Common Sandpiper
Sotsnipe	Spotted Redshank
Lappspove	Bar-tailed godwit
Småspove	Eurasian Whimbrel
Enkeltbekkasin	Common Snipe
Svømmesnipe	Red-necked Phalarope

Fjelljo	Long-tailed skua
Fiskemåke	Common gull
Rødnebbterne	Artic Tern
Taksvale	Common House Martin
Sandsvale	Sand Martin
Heipiplerka	Meadow pipit
Gulerlen	Western Yellow Wagtail
Blåstrupe	Bluethroat
Steinkvetten	Northern wheatear
Måltrost	Song Thrush
Rødvingetrost	Redwing
Gråtrost	Fieldfare
Løvsanger	Willow Warbler
Varsler	Great Grey Shrike
Skjære	Eurasian Magpie
Kråke	Crow
Ravn	Raven
Bjørkefink	Brambling
Gråsisik	Common Redpoll
Sivspurv	Common Reed Bunting
Lappspurv	Lapland Longspur
Sangsvane	Whooper Swan
Gjøk	Common Cuckoo
Bokfink	Chaffinch

Table F.1

