



CHALMERS
UNIVERSITY OF TECHNOLOGY

**Brachycephalic Obstructive Airway Syndrome (BOAS)
classification in dogs based on respiratory noise analysis using
machine learning**

Moa Mårtensson

Master Thesis in Biomedical Engineering

February 2021

Brachycephalic Obstructive Airway Syndrome (BOAS) classification in dogs based on respiratory noise analysis using machine learning

Moa Mårtensson

Master Thesis in Biomedical Engineering

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2021



CHALMERS
UNIVERSITY OF TECHNOLOGY

Brachycephalic Obstructive Airway Syndrome (BOAS) classification in dogs based on respiratory noise analysis using machine learning

Moa Mårtensson

Master Thesis in Biomedical Engineering

Supervisors:

Magnus Karlsteen, Department of Physics, Chalmers University of Technology

Eva Skiöldebrand, Swedish University of Agricultural Sciences

Examiner:

Magnus Karlsteen, Department of Physics, Chalmers University of Technology

© Moa Mårtensson, 2021

Typeset in L^AT_EX

Printed by Chalmers Reproservice

Gothenburg, Sweden 2021

Department of Physics

Chalmers University of Technology

SE-412 96 Gothenburg

+46 31 772 1000

Abstract

Brachycephalic Obstructive Airway Syndrome (BOAS) is a problem in several dog breeds due to a compressed shape of the skull. It is classified as BOAS grade 0-3, where 0 is normal breathing and 3 is the most severe grade of the syndrome. Grade 2-3 can cause great suffering for the affected dogs and needs treatment. This study aimed to find a method using machine learning to classify the BOAS grade based on audio recordings of respiratory noise. The recordings were converted into Mel-Frequency Cepstral Coefficients (MFCCs) to be processed as images by the network. The results proved that Recurrent Neural Network - Long Short-Term Memory (RNN-LSTM) was a successful method to classify the four different BOAS grades with an accuracy of about 86-87% for dictaphone recordings and about 62-66% for stethoscope recordings. Convolutional Neural Networks (CNN) also managed to classify the BOAS grades but this method was less accurate, with an accuracy of approximately 74-76% for dictaphone recordings and 50-54% for stethoscope recordings. The study was a collaboration between Chalmers University of Technology and Swedish University of Agricultural Sciences.

Keywords: Brachycephalic Obstructive Airway Syndrome, BOAS, Machine learning, Convolutional Neural Network, CNN, Mel-Frequency Cepstral Coefficients, MFCC, Recurrent Neural Network, Long Short-Term Memory, RNN-LSTM, Respiratory noise analysis

Acknowledgements

This study is a collaboration between Chalmers University of Technology and Swedish University of Agricultural Sciences.

I would like to thank the team at the Swedish University of Agricultural Sciences; Eva, Ingrid and Maria. Thank you for your expertise in the field of veterinary medicine and for a fun cooperation!

Thanks to all dog owners and dogs who were willing to take the time and effort to participate in this study. Without you this project would not be possible!

The greatest thanks possible I want to give to Magnus who has been the best supervisor anyone could ask for! Thank you for helping me with Python, the report and everything else, and for being available at all hours of the day. Thank you for your support, dedication and patience!

Abbreviations

BOAS - Brachycephalic Obstructive Airway Syndrome

CNN - Convolutional Neural Network

ET - Exercise Test

LSTM - Long Short-Term Memory

MFCCs - Mel-Frequency Cepstral Coefficients

RNN - Recurrent Neural Network

Medical terms

Apnoea - respiratory arrest

Brachycephalic - short skull

Dyspnoea - difficulty breathing

Hyperplasia - overgrowth of tissue

Hypoplasia - underdevelopment of tissue

Larynx - voice box, part of respiratory tract above the trachea

Nasopharynx - the rear part of the nasal cavity above the soft palate

Regurgitation - reflux

Rhinoplasty - surgery to change the shape of the nose

Staphylectomy - removal of a part of the soft palate

Stenosis - narrowing

Stertor - a low pitched snoring sound during inspiration

Stridor - a high pitch wheezing sound from the laryngeal area

Trachea - windpipe

Contents

1	Introduction	1
1.1	Aim	1
1.2	Brachycephalic Obstructive Airway Syndrome (BOAS)	1
2	Methods	3
2.1	Audio recording	3
2.2	Preprocessing the signal	3
2.2.1	Mel-Frequency Cepstral Coefficients (MFCCs)	3
2.3	Machine Learning	5
2.3.1	Overfitting	6
2.3.2	Convolutional Neural Networks (CNN)	7
2.3.3	Recurrent Neural Networks (RNN)	7
2.3.4	Recurrent Neural Network - Long Short-Term Memory (RNN-LSTM)	7
3	Results	9
3.1	CNN	9
3.2	RNN-LSTM	9
3.2.1	Four BOAS classes	10
3.2.2	Three BOAS classes	12
3.2.3	Two BOAS classes	15
4	Discussion	16
4.1	Potential sources of error	16
4.2	Future work	17
5	Conclusions	18

1 Introduction

All dog breeds descend from the wolf [1]. It can be hard to imagine from the variety of shapes and sizes of dogs today. But what consequences does excessive breeding to promote a certain physical attribute instead of good health and mentality have for the affected dogs?

1.1 Aim

The aim of this project is to find a method using machine learning to classify Brachycephalic Obstructive Airway Syndrome (BOAS) grades in brachycephalic dogs based on respiratory noises.

1.2 Brachycephalic Obstructive Airway Syndrome (BOAS)

Dog breeds with flat faces and short noses such as pugs and bulldogs are called brachycephalic dogs and refer to the compressed shape of the skull [2]. Many of these dogs suffer from Brachycephalic Obstructive Airway Syndrome (BOAS). Dogs with BOAS have anatomical deviations such as nostril stenosis, nasopharyngeal hyperplasia, tracheal hypoplasia and elongated and thickened soft palate that may obstruct parts of the airways [3]. Figure 1 presents normal nostrils and nostril stenosis.

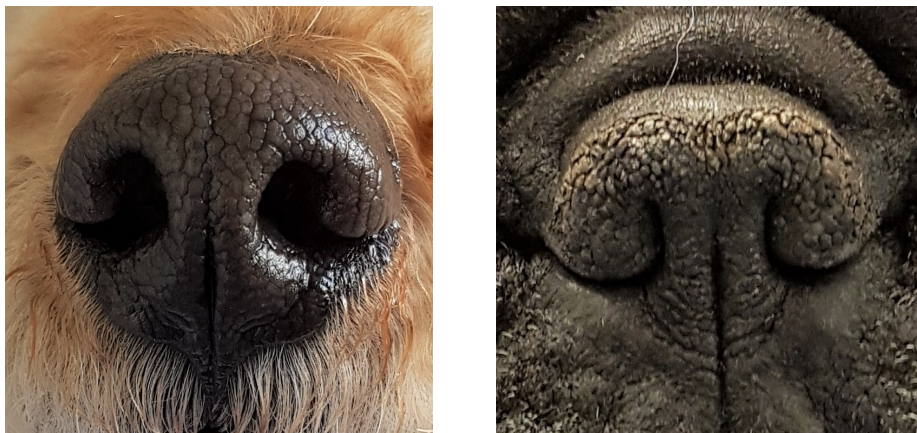


Figure 1: Normal nostrils to the left and nostril stenosis to the right.

Photo: Moa Mårtensson

Figure 2 presents normal upper airway anatomy in dogs.

BOAS results in symptoms such as snoring, inspiratory dyspnoea, sleep apnoea, regurgitation/vomiting and complications related to anesthesia [3]. The symptoms from the upper

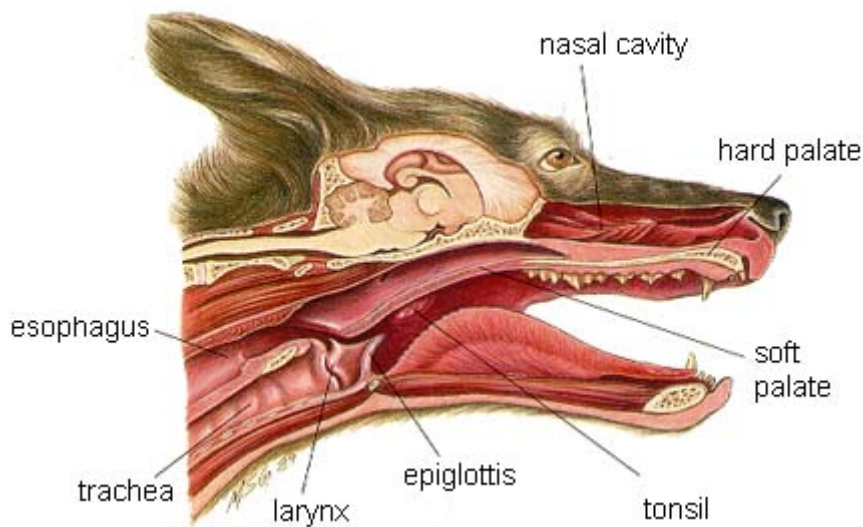


Figure 2: Normal upper airway anatomy in dogs.
Image from Hill's Atlas of Veterinary Clinical Anatomy, used with permission.

airways increase during and after exercise, with high temperatures, when stressed or excited and with overweight [3].

With a stethoscope a veterinarian can auscultate abnormal respiratory sounds such as stertor and stridor. Stertor is a low pitched noise usually caused by an elongated soft palate or nasopharyngeal obstruction. Stridor is a high pitched noise caused by compromised or collapsed laryngeal area. The presence of stridor is associated with a higher BOAS grade [3].

BOAS is graded from 0-3, where 0 is no respiratory noises and 3 is severe respiratory noises. Generally grade 0-1 does not affect the dog, whereas grade 2-3 is a problem that affects the dogs quality of life and requires treatment [4].

Treatment for BOAS is usually weight loss and/or surgery [4]. Surgery aims to correct some of the airway malformations in order to breath better. Airway surgery may include procedures like staphylectomy (shortening of the soft palate) and/or rhinoplasty (nostril enlargement) [3] [2].

Extreme brachycephalic breeds have a shorter life expectancy than other breeds the same size. They also have a higher risk of dying from respiratory related causes [4].

2 Methods

The programming language used is Python version 3.8 with additional libraries such as Tensorflow, Librosa and Keras. The code is inspired from Valerio Velardo's series of tutorials on YouTube called "The sound of AI", which is used for classifying music genres. The code has been modified to accommodate BOAS-grading of respiratory noises from dogs.

2.1 Audio recording

Recording of the dogs respiratory noises was performed during September 21st-25th 2020 at the University Animal Hospital in Uppsala, Sweden. All participants needed to be at least one year old and their owners had to sign a consent form and answer some questions, including medical history. There were 41 dogs participating, 24 of them were brachycephalic dogs and 17 were reference dogs. The brachycephalic dogs consisted of eleven pugs, eleven French bulldogs, one English bulldog and one Boston terrier. All dogs were examined by veterinarian Maria Dimopoulou, an EBVS European Specialist in Small Animal Surgery, and graded according to the BOAS scale from 0 to 3. The nostrils were photographed and respiratory patterns video recorded if needed during the analysis process. The respiratory sounds at rest were recorded with a 3M Littmann electronic stethoscope model 3200 placed on the side of the larynx. Simultaneously an Olympus linear PCM recorder LS-P1 dictaphone was placed 10-15cm from the dogs mouth and nose. The recordings were carried out for 30s. Thereafter the dogs performed a three-minute exercise test (ET) where they ran back and forth with a handler. Immediately after, they were recorded again in the same way for another 30s. The Littmann stethoscope was connected via Bluetooth to the software Stethassist, where the recordings could be saved and exported.

2.2 Preprocessing the signal

The audio recordings have been manually edited into 30s wav-files and major distortions and errors have been removed. The software used for this was BandLab. An attempt was made to amplify the signal from the stethoscope recordings due to low volume. The amplification created a lot of distortion and was hence not used.

2.2.1 Mel-Frequency Cepstral Coefficients (MFCCs)

To convert the audio signal into a time-frequency domain representative image for the machine learning network to interpret, MFCCs are used [5] [6]. MFCCs are widely used in signal processing for sound recognition [7] [6] and classification to find similarities in or between signals. Human hearing sensitivity is different for different frequencies. MFCCs can mimic the non-linear frequency characteristics of human hearing [6]. MFCCs are the

inverse discrete cosine transform of the logarithmic energy in mel-frequency bands of the signal [7] [5] which is presented in equation 1:

$$mfcc = \sqrt{\frac{2}{M_{mfcc}}} \sum_{m=1}^{M_{mfcc}} \log(X_m(t)) \cos\left(\frac{c(m - \frac{1}{2})\pi}{M_{mfcc}}\right) \quad (1)$$

where M_{mfcc} is the number of mel-frequency bands, m is the index of the mel-frequency band, $X_m(t)$ is the energy of the m^{th} band and c is the index of the coefficient [5].

Figure 3 presents the waveform of a 30s stethoscope recording graded BOAS 3. The x-axis represents time and the y-axis represents amplitude. This waveform was transformed into a MFCC spectrogram for the machine learning process.

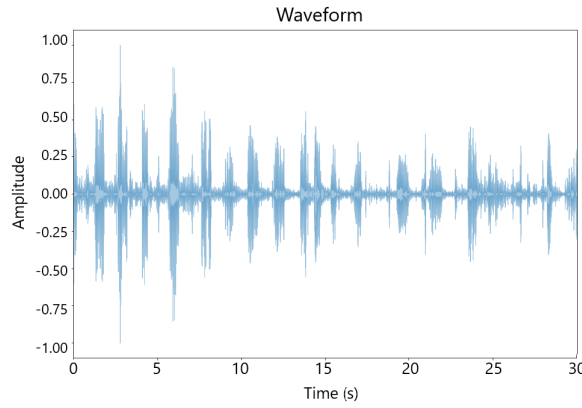


Figure 3: The waveform of a 30s stethoscope recording graded BOAS 3.

Image created by Moa Mårtensson.

For visualisation purposes, Figure 4 presents a Mel spectrogram. The x-axis represents the time and the y-axis represents frequency. The colorbar represents the amplitude in dB. The image suggests that the frequency as well as the amplitude are relatively low.

Figure 5 presents a MFCC spectrogram with 39 coefficients. The axes represent time on the x-axis and coefficients on the y-axis. The color bar to the right represents the values of the coefficients.

A Python code divided the audio files into 3s segments and created 39 MFCCs for each segment and saved them all into a json-file that was later used for machine learning. The code for creating the MFCCs can be found in Appendix I.

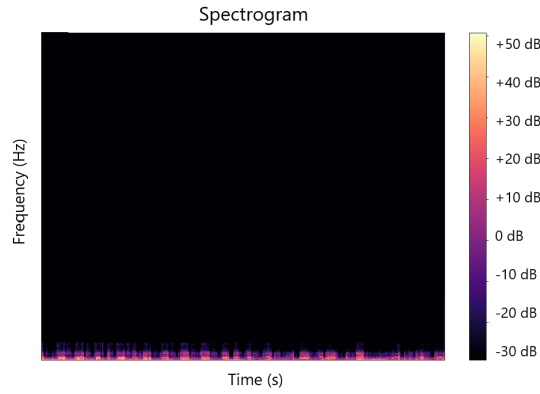


Figure 4: A Mel spectrogram of a 30s stethoscope recording graded BOAS 3.
Image created by Moa Mårtensson.

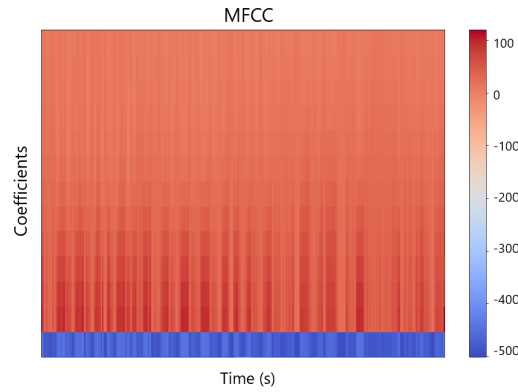


Figure 5: The MFCCs of a 30s stethoscope recording graded BOAS 3 and 39 coefficients.
Image created by Moa Mårtensson.

2.3 Machine Learning

Deep learning is a type of machine learning method that contains multiple levels of nonlinear operations with hidden levels in a neural network. Deep learning can discover complex relationships within datasets using algorithms that keep relevant information from previous layers. It is used for classification purposes since its advanced functions can learn to differ between different response classes [8].

The initial task of machine learning was to transform relevant information from the audio recordings into representative images. In this project that was the MFCC images. Then

a model was created and trained using the MFCCs. After the training, the model needed to be evaluated to know how well the model worked. Unbalanced number of outcomes between the class labels while training can cause problems measuring the accuracy for the model [9].

There are different types of machine learning problems. If the output is numeric it is called a regression problem. If the output is a class label it is a classification problem. The number of labels decides the subcategory. If it is a yes or no problem, with only two possible options, it is called a binary classification problem. More than two classes are called multi-class classification problems. There is also something called multi-label classification, when a sample belongs to multiple classes [9].

2.3.1 Overfitting

Overfitting is when the model is overtrained on the training dataset. It is unable to generalize the test dataset and therefore performs poorly [10] [11]. It can be suspected if the training dataset has a significantly higher accuracy than the test dataset. This can be a result of a too small dataset [10].

Common techniques for limiting overfitting are simplifying the network architecture, regularization, collecting more data and data augmentation. Unfortunately, there is no universal solution for overfitting, hence testing different techniques and evaluating the result is the way to handle it [11].

Simplifying the model can consist of reducing the number of layers and the number of neurons in the layers. This is a time-consuming strategy since there are countless combinations to try and evaluate [11].

There are dozens of regularization techniques. The most common ones are L1, L2, dropout and early stopping. L1 and L2 are weights that reduce the network's capacity to adapt to the dataset. This is done by adding a term to the cost function [11]. Dropout randomly removes a neuron from the network in each epoch of training the model [10] [11]. Randomly removing units creates a smaller network where the weights are smaller and distributed across the predictors in the model [8]. Early stopping reduces the number of iterations/epochs during training so the model stops before it has overtrained on the dataset [10].

Collecting more data is probably the easiest way to reduce overfitting. In reality, it may not always be possible [11].

Data augmentation has the goal of generating additional data. It is the same data only modified and added, but it has been made unrecognizable for the model. Augmentation for

an image can consist of for example rotating, shifting and stretching [11].

To address the overfitting in this project, simplifying the model and the regularization methods L1 and L2, dropouts and early stopping were implemented. These actions had inadequate results and the overfitting persisted. Gathering more data was unfortunately not an option. Data augmentation was not implemented due to time limitations, but would be very interesting as a part of future work.

2.3.2 Convolutional Neural Networks (CNN)

CNNs are a type of artificial neural network that are inspired by the neurons in the visual cortex of the human brain. This makes them suitable for artificial vision and image analysis. CNNs mainly consist of convolutional and pooling layers. The convolutional layers perform a mathematical operation called convolution, which practically means to apply a filter matrix to the input. This makes it possible to detect and learn shapes, patterns and other features in the input. The pooling layers down-sample the input by reducing the dimension, which makes the network more robust [9]. The code for the CNN used in this project is presented in Appendix II.

2.3.3 Recurrent Neural Networks (RNN)

RNN is a method for processing sequential data. It passes forward what was learned from the previous time step (the output) as an additional input to the next time step. This way it attempts to predict the output from the history of previous input data [12] [8]. It uses feed forward neural networks with cyclic connections. In the network there are three main connections types that are very important; input to hidden layer, hidden to hidden layer and hidden to output layer. The weights from these connections are represented by different matrices that are processed into a scalar value that is classified as a binary variable. The loss function then compares the predicted binary variable to the actual label [8].

RNN architecture has some limitations. Even though it was created to learn long-term dependencies, it does not do it very well due to problems called vanishing gradient and exploding gradient. It causes the weights in the network to become extremely small or large during network training due to the error signal only can be traced back a few steps. The weight deviations hence increase exponentially over time. To circumvent this problem a variant of the RNN model was created, Long Short-Term Memory (LSTM) [8] [12].

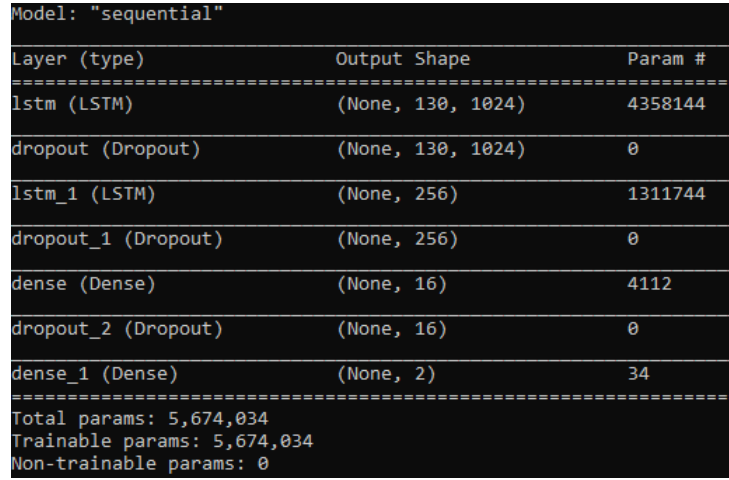
2.3.4 Recurrent Neural Network - Long Short-Term Memory (RNN-LSTM)

RNN-LSTM avoids vanishing and exploding gradients by remembering important information from previous steps while eliminating information that is unnecessary from being

used in future steps. The model is able to remember context and dependencies over long periods of time [12].

RNN-LSTM has an architecture built on connected sub-networks called memory blocks. The memory blocks recall input [8]. They contain accumulator cells and three different types of gates; input gate, forget gate and output gate. The gates are multiplication units that can store and access information [8] [12]. Each gate can learn what inputs are useful for predicting the outputs. It passes input information forward and back-propagates the error and adjusts the weights [12].

In Figure 6 the architecture of the RNN-LSTM network used in this project is presented. The complete code is presented in Appendix III.



```
Model: "sequential"
Layer (type)                Output Shape              Param #
-----
lstm (LSTM)                  (None, 130, 1024)        4358144
dropout (Dropout)            (None, 130, 1024)        0
lstm_1 (LSTM)                (None, 256)              1311744
dropout_1 (Dropout)          (None, 256)              0
dense (Dense)                 (None, 16)               4112
dropout_2 (Dropout)          (None, 16)               0
dense_1 (Dense)              (None, 2)                34
-----
Total params: 5,674,034
Trainable params: 5,674,034
Non-trainable params: 0
```

Figure 6: Architecture of the RNN-LSTM network.
Image created by Magnus Karlsteen.

To find the optimum number of neurons in each layer, a programming loop was created that tested different combinations. The same method was used to evaluate the most suitable learning rate and the best combinations of activation functions. To find other optimum hyperparameters, such as number of layers and mini-batch size, some different combinations were tested manually. Based on the results from these procedures, the activation functions hard sigmoid and softmax were used. The number of neurons were set to 1024, 256 and 16 in the different layers. The dataset was divided into 55% training data, 25% test data and 20% validation data. For the dropout layers the number of units dropped were set to 0.3. The learning rate optimizer used for compiling the model was called adaptive moments or Adam. The learning rate was set to $1 * 10^{-5}$. Mini-batch size is the number of training set samples that are processed in each iteration. A large mini-batch size takes a long time for each iteration while a small mini-batch size may not reach the local minimum

[8]. The mini-batch size used in this network was 32 and the number of epochs/iterations was 1000.

3 Results

The 41 participants were graded by veterinarian Maria Dimopoulou, an EBVS European Specialist in Small Animal Surgery, into BOAS classifications and the distribution is presented in Table 1.

Table 1: The distribution of BOAS grades among the participants.

	BOAS 0	BOAS 1	BOAS 2	BOAS 3
Pugs	2	5	4	0
French bulldogs	0	5	2	4
English bulldogs	0	0	1	0
Boston terriers	0	1	0	0
Reference dogs	17	0	0	0
Total number of dogs	19	11	7	4

3.1 CNN

When the CNN method is used, the model has an average accuracy ranging from about 50% to 76% for four BOAS-classes, as can be seen in Table 2.

Table 2: Classification accuracy for the different recording types using four BOAS classes and CNN.

	Littmann Before ET	Littmann After ET	Olympus Before ET	Olympus After ET
Highest accuracy	57.10%	62.90%	80.50%	81.30%
Lowest accuracy	41.00%	42.90%	66.90%	69.10%
Average accuracy	49.53%	53.67%	73.73%	75.57%

When BOAS 2 and 3 are combined into one class, because of limited data in these classes, three BOAS classes can be evaluated. Three BOAS classes are used in Table 3. A slight improvement can be seen as the models average accuracy varies from about 53% to 79%.

3.2 RNN-LSTM

The RNN-LSTM method is used in the following sections.

Table 3: Classification accuracy for the different recording types using three BOAS classes and CNN.

	Littmann Before ET	Littmann After ET	Olympus Before ET	Olympus After ET
Highest accuracy	58.10%	68.60%	78.00%	86.20%
Lowest accuracy	46.70%	57.10%	68.60%	69.90%
Average accuracy	53.33%	64.13%	73.88%	78.85%

3.2.1 Four BOAS classes

In Table 4 the models classification accuracy is presented after 20 tries in each recording category and with four BOAS-classes; 0, 1, 2 and 3. The average accuracy varies from 62% for Littmann After ET to almost 87% for Olympus After ET, which is a significant improvement compared to using CNN.

Table 4: Classification accuracy for the different recording types using four BOAS classes and RNN-LSTM.

	Littmann Before ET	Littmann After ET	Olympus Before ET	Olympus After ET
Highest accuracy	77.10%	69.50%	92.40%	94.30%
Lowest accuracy	59.00%	52.40%	81.40%	81.30%
Average accuracy	66.39%	61.80%	86.28%	86.92%

In Figures 7-8, 11-12 and 14-15 confusion matrices illustrates the distribution of classifications for each audio segment, 3s in length. The observant reader may notice that the matrices have a total of 101 test samples, which is due to Python crashing if 100 was used, which would have corresponded to percent otherwise. The matrices present the classification accuracy for each BOAS class, as opposed to the tables that show the classification accuracy of the entire model. The x-axis represents the predicted BOAS class and the y-axis represents the true BOAS class. The colorbar shows lighter colors for higher values. The trend of a light diagonal from the top left to the bottom right with dark sides indicates a successful classification. The accuracy for each BOAS class is calculated from the following equation:

$$accuracy = \frac{TruePositive + TrueNegative}{TruePositive + TrueNegative + FalsePositive + FalseNegative} \quad (2)$$

and the weighted overall precision for each matrix is calculated using:

$$precision = \sum_{i=1}^C \left(\frac{Samples\ i}{TotalSamples} * \frac{TruePositive\ i}{TruePositive\ i + FalsePositive\ i} \right) \quad (3)$$

where i represents the rows of the matrix and C are the number of classes.

In Figure 7 a confusion matrix for Littmann with four BOAS classes is presented. For Littmann Before ET the accuracy is 81% for BOAS 0, 86% for BOAS 1, 86% for BOAS 2 and 93% for BOAS 3. The weighted overall precision is 72.7%. For Littmann After ET the accuracy is 80% for BOAS 0, 90% for BOAS 1, 87% for BOAS 2 and 96% for BOAS 3. The precision is 69.9%.

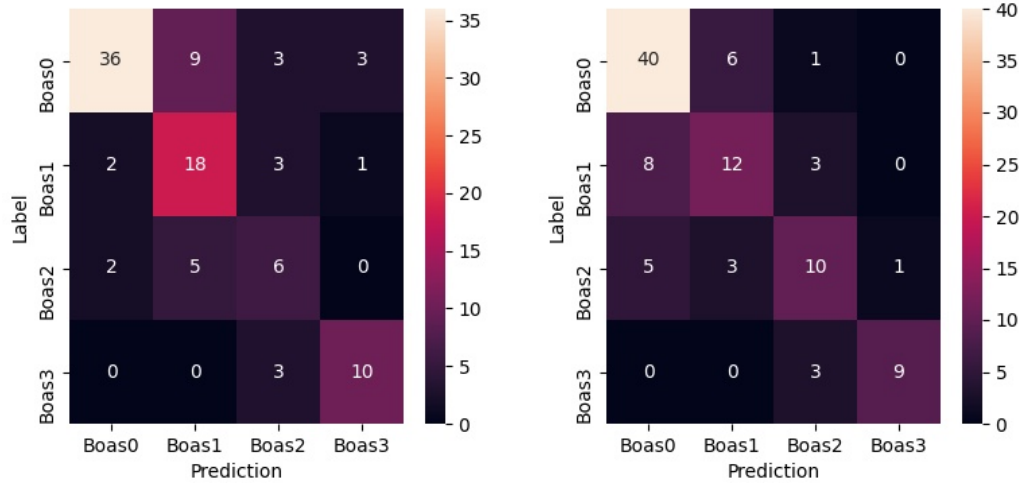


Figure 7: Confusion matrix for RNN-LSTM with four BOAS classes for Littmann Before ET on the left and Littmann After ET on the right.

Images created by Magnus Karlsteen.

In Figure 8 a confusion matrix for Olympus with four BOAS classes is presented. A clear lighter diagonal with darker sides can be seen indicating a successful result. Olympus Before ET for BOAS 0 has 93% accuracy, BOAS 1 96%, BOAS 2 93% and BOAS 3 98%. The weighted overall precision is 90.4%. Olympus After ET for BOAS 0 has 93% accuracy, BOAS 1 93%, BOAS 2 96% and BOAS 3 100%. The precision is 91.2%.

A problem with overfitting was discovered when comparing the training accuracy and the test accuracy. In Figure 9 a problem with overfitting for Littmann After ET with four BOAS classes has been visualized to the left. The train accuracy is significantly higher than the test accuracy, and the test error/validation loss increases over time. To the right a graph without overfitting is presented for comparison. The train and test graphs follow each other well.

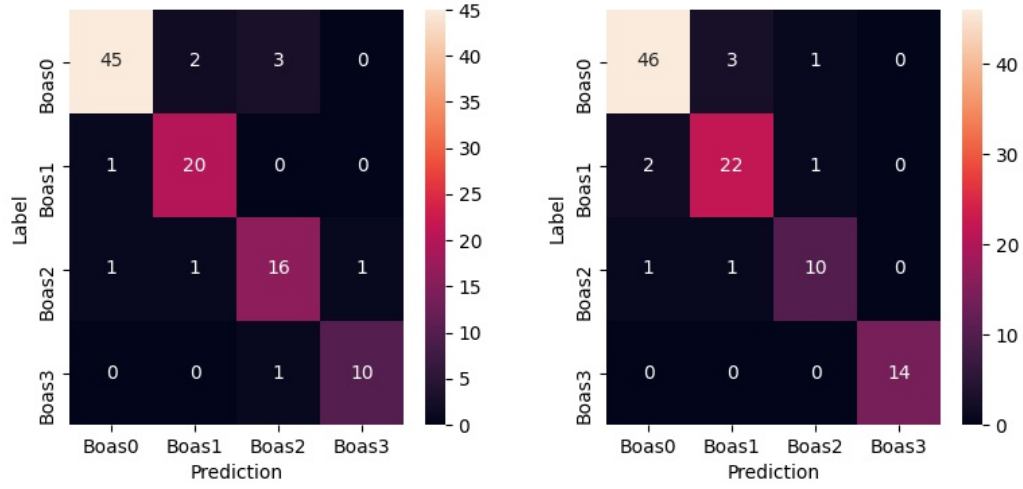


Figure 8: Confusion matrix for RNN-LSTM with four BOAS classes for Olympus Before ET on the left and Olympus After ET on the right.
Images created by Magnus Karlsteen.

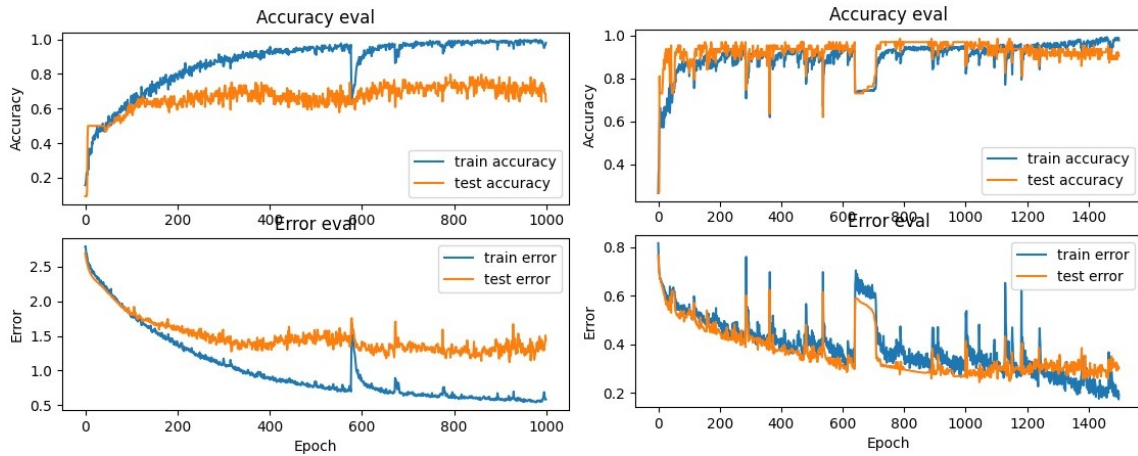


Figure 9: Overfitting to the left and no overfitting to the right.
Images created by Magnus Karlsteen.

3.2.2 Three BOAS classes

If BOAS grade 2 and 3 are combined into one group, the overall result for the model improved as is presented in Table 5. Here 12 tries for each recording type were used. The average accuracy varies from 69-88% for the different recording types.

Table 5: Classification accuracy for the different recording types using three BOAS classes and RNN-LSTM.

	Littmann Before ET	Littmann After ET	Olympus Before ET	Olympus After ET
Highest accuracy	73.30%	79.00%	90.70%	92.70%
Lowest accuracy	60.00%	64.80%	78.80%	84.60%
Average accuracy	68.56%	70.95%	85.81%	88.48%

In Figure 10 two matrices for Littmann Before and After ET with three BOAS classes are presented. For Littmann Before ET on the left the accuracy for BOAS 0 is 81%, BOAS 1 is 75% and BOAS 2 and 3 is 80%. The precision is 69.7%. For Littmann After ET on the right the accuracy is 81% for BOAS 0, 78% for BOAS 1 and 83% for BOAS 2 and 3. The precision is 70.8%.

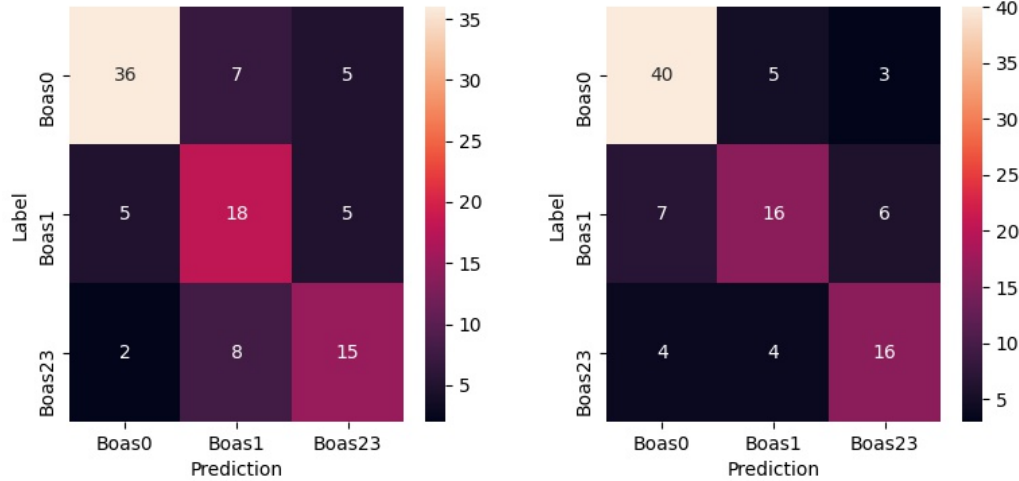


Figure 10: Confusion matrices with three BOAS-classes. Littmann Before ET on the left and Littmann After ET on the right.

Images created by Magnus Karlsteen.

In Figure 11 two matrices for Olympus Before and After ET with three BOAS classes are presented. For Olympus Before ET BOAS 0 has 96% accuracy, BOAS 1 has 94% and BOAS 2 and 3 combined has 94%. The precision is 92.1%. Olympus After ET has identical accuracy and precision as Olympus Before ET.

Four Littmann After ET recordings were then excluded from the training data in an attempt to circumvent overfitting. The whole 30s recordings are divided into 3s segments and each segments classification as well as the whole recordings classification are presented in

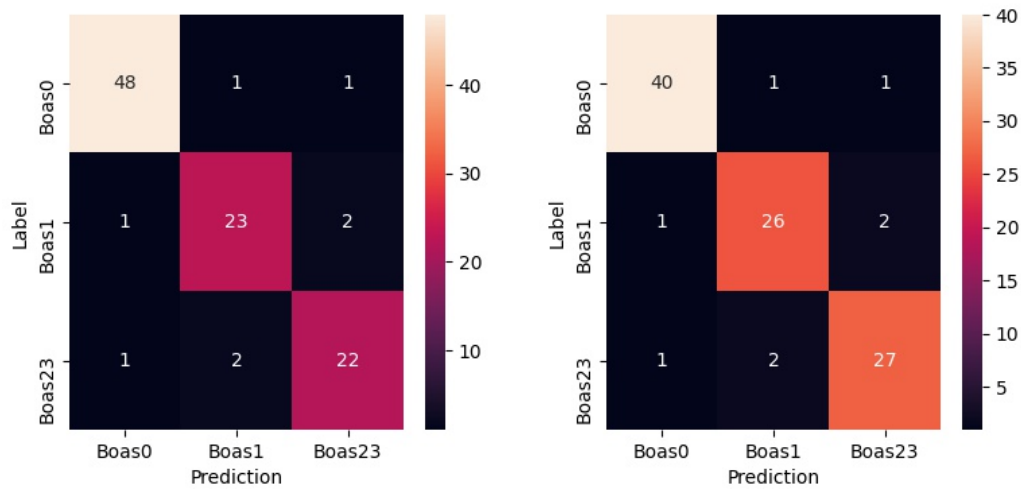


Figure 11: Confusion matrices with three BOAS-classes. Olympus Before ET on the left and Olympus After ET on the right.
Images created by Magnus Karlsteen.

Figure 12. The same procedure was performed for the other recording types, but with a significantly less accurate result.

File	0-3s	3-6s	6-9s	9-12s	12-15s	15-18s	18-21s	21-24s	24-27s	27-30s
BOAS 0	0	0	0	1	0	0	0	0	0	0
BOAS 1	1	1	1	23	23	23	23	1	1	1
BOAS 23	23	23	23	23	23	23	23	23	1	23
BOAS 23	23	1	23	23	23	1	23	23	23	23

File	Predict 0	Predict 1	Predict 23	True label	Result	Probability
BOAS 0	90%	10%	0	0	0	90%
BOAS 1	0	56%	44%	1	1	56%
BOAS 23	0	14%	86%	23	23	86%
BOAS 23	0	25%	75%	23	23	75%

Figure 12: Four Littmann after ET files successfully BOAS-graded.
Image created by Magnus Karlsteen and Moa Mårtensson.

3.2.3 Two BOAS classes

In Table 6 BOAS 0 and 1 are combined into one class and BOAS 2 and 3 are one class. 25 tries for each recording type were performed. The average accuracy of the model varies from 79-93% for the different recording types.

Table 6: Classification accuracy for the different recording types using two BOAS classes and RNN-LSTM.

	Littmann Before ET	Littmann After ET	Olympus Before ET	Olympus After ET
Highest accuracy	86.70%	92.40%	94.90%	97.60%
Lowest accuracy	63.80%	77.10%	86.40%	86.20%
Average accuracy	79.16%	85.23%	90.97%	93.04%

In Figure 13 two confusion matrices for Littmann Before and After ET with two BOAS classes are presented. Littmann Before ET has an accuracy of 84% for BOAS 0 and 1, and 84% for BOAS 2 and 3. The precision is 83.8%. Littmann After ET has an accuracy of 92% for BOAS 0 and 1, and 92% for BOAS 2 and 3. The precision is 92.4%.

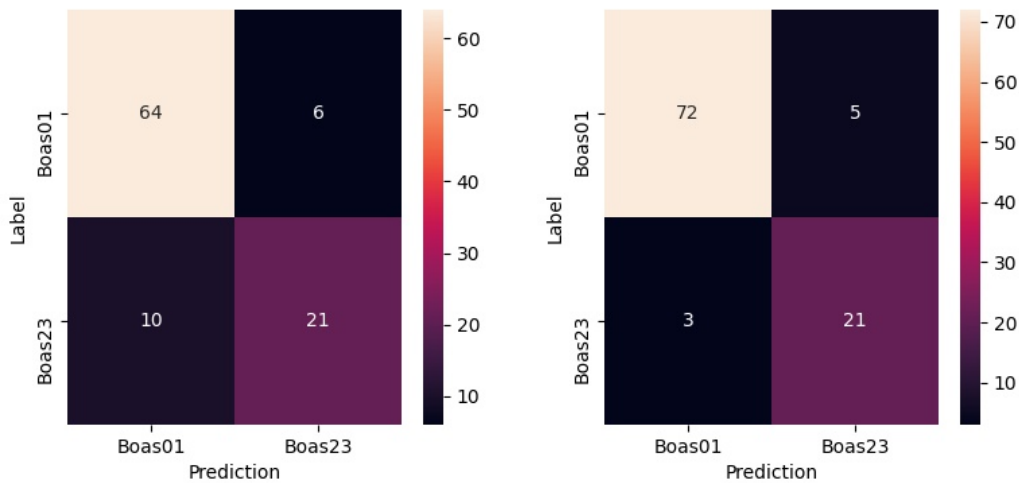


Figure 13: Confusion matrices with two BOAS-classes. Littmann Before ET on the left and Littmann After ET on the right.
Images created by Magnus Karlsteen.

In Figure 14 two matrices for Olympus Before and After ET with two BOAS classes are presented. Olympus before ET has an accuracy of 93% for BOAS 0 and 1 and 93% for BOAS 2 and 3. The precision is 93.1%. Olympus after ET has an accuracy of 96% for BOAS 0 and 1 and 96% for BOAS 2 and 3. The precision is 96.2%.

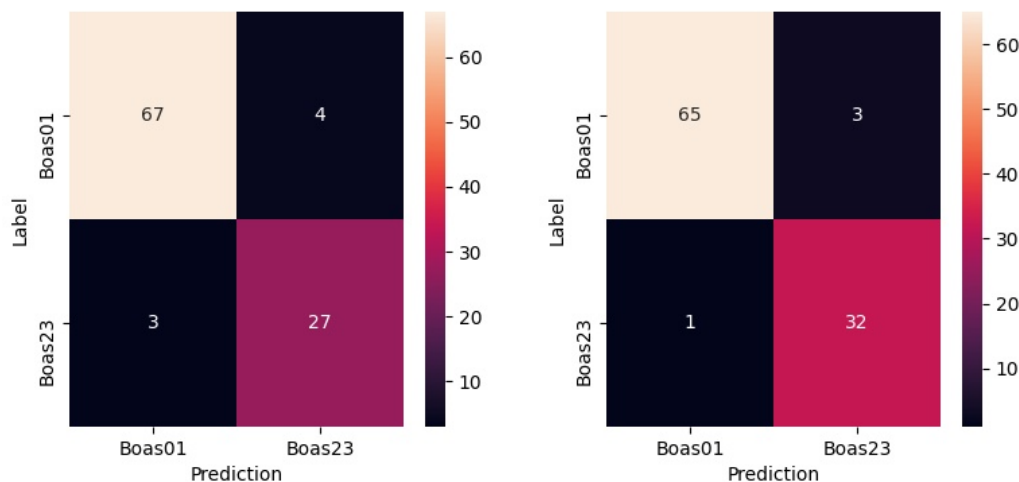


Figure 14: Confusion matrices with two BOAS-classes. Olympus Before ET on the left and Olympus After ET on the right.
Images created by Magnus Karlsteen.

4 Discussion

RNN-LSTM has a significantly improved classification accuracy compared to CNN. This was discovered early on and the number of tests with CNN was reduced because of it, but kept for comparison. Focus has been on RNN-LSTM.

The general trend is that Olympus has a better classification accuracy than Littmann, and that fewer BOAS classes have better accuracy than many BOAS classes.

Overall, Littmann results are significantly lower than Olympus. A possible reason for this is that the Littmann recordings have a lower sound volume. This was tried to circumvent by amplifying the signal, but the sound quality became too poor with extensive noise and could hence not be used. Generally, the overfitting is more prominent for Littmann than Olympus, and higher for many BOAS classes than for few BOAS classes, which may be another reason why Olympus and few BOAS classes perform better.

4.1 Potential sources of error

A limited number of recordings as well as fewer recordings in some BOAS classes than others lead to limited training data, especially for BOAS 3. This is a cause for overfitting. Actions such as different regularization methods and simplifying the model has been taken,

but with inadequate results. If more time had been available, further attempts to address this issue would have been executed, mainly using data augmentation. A larger number of recordings and a more even distribution between the BOAS grades would probably be very beneficial to decrease overfitting, and hence increase the accuracy for every recording type.

The recordings are not 100% free from disturbances, but it may not be possible when working with animals. Many of the dogs were panting, which may mask the respiratory sounds of interest. In some cases the handler who ran with the dog during the exercise test were panting during the after ET recording. At some point doors closed and people walked or talked in adjacent rooms.

4.2 Future work

For future work, more recordings from a larger number of dogs would be useful. Gathering more data would probably limit the overfitting, which would improve the classification accuracy. The training data can be augmented with pitch shift, time stretch and background noise if applied to the audio files. If applied to the MFCC images, rotating, shift and stretch can be used. Augmentation was not performed in this project because of time limitations. This method would increase the number of audio files since the augmented files would be added to the existing dataset.

The network can be further developed to possibly achieve a better model for classifying BOAS.

The method may be used for development of a mobile phone application for dog owners to have an assessment of their dogs breathing. If classified as BOAS 2 or 3 it would be recommended to see a veterinarian for further assessment and possible treatment.

It could also be a tool for smaller veterinary clinics to do a first assessment of a dog to see if it needs a referral to a specialist on BOAS.

A tool based on this technique could also benefit brachycephalic breed buyers when visiting a breeder with the intention of buying a dog. The tool could be used to obtain the BOAS grade for the dog of interest, if the dog is at least one year old, or for the parents of the puppy of interest. Hopefully, most people would refrain from buying a dog that is proven to struggle with breathing or has parents who do. It may require expensive and risky surgery, as well as have a compromised quality of life.

Best case scenario would be if a tool using this technique could improve the guidelines and laws for the breeding industry. A tool could decide which brachycephalic dogs are suitable and not for breeding purposes regarding their airways. Other factors such as overall health

and temper should of course also be taken into account. Making the airway problems easily measurable would in this situation be a huge benefit compared to the current arbitrary opinion of the dog owner, who may not be aware of existing breathing difficulties.

5 Conclusions

The general trends are that Olympus has a better accuracy than Littmann, fewer BOAS classes have better accuracy than many BOAS classes and RNN-LSTM performs better than CNN, although both methods manage to classify BOAS grades.

RNN-LSTM has proven to be an efficient method to classify BOAS grades from 0-3. The average accuracy is 86% and the precision is 90% for the Olympus dictaphone Before ET, and the accuracy is 87% and the precision is 91% for Olympus After ET. The method is also efficient for Littmann stethoscope recordings but with an accuracy of 66% and a precision of 73% for Littmann Before ET, and the accuracy is 62% and the precision is 70% for Littmann After ET.

The accuracy can probably be further improved as a part of future work using more data and data augmentation. Ideally the additional data would be evenly distributed between the BOAS classes. More data, data augmentation and evenly distributed data would probably address the overfitting of training data and hence improve the classification accuracy.

The results could be used as a tool such as a mobile phone application to easily measure the BOAS grades of dogs using the mobile phones microphone. The dictaphone results are more interesting than the stethoscope results if the next goal is to develop a mobile phone application, since it is more similar to the phones microphone.

References

- [1] P. Jouventin, Y. Christen, and F. S. Dobson, “Altruism in wolves explains the coevolution of dogs and humans,” *Ideas in Ecology and Evolution 2016*, vol. 9, no. 9, pp. 4–11, may 2016.
- [2] S. J. Ettinger, E. C. Feldman, and E. Côté, *Textbook of veterinary internal medicine*, 8th ed. USA: Elsevier, 2017.
- [3] J. Riggs, N.-C. Liu, D. R. Sutton, D. Sargan, and J. F. Ladlow, “Validation of exercise testing and laryngeal auscultation for grading brachycephalic obstructive airway syndrome in pugs, french bulldogs, and english bulldogs by using whole-body barometric plethysmography,” *Veterinary Surgery*, vol. 48, no. 48, pp. 488–496, 2019.
- [4] J. Ladlow, N.-C. Liu, L. Kalmar, and D. Sargan, “Brachycephalic obstructive airway syndrome,” *Veterinary record*, vol. -, no. -, pp. 375–378, 2018.
- [5] T. Virtanen, M. D. Plumbley, and D. Ellis, *Computational analysis of sound scenes and events*, 1st ed. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer International publishing, 2018.
- [6] S. Jin, X. Wang, L. Du, and D. He, “Evaluation and modeling of automotive transmission whine noise quality based on mfcc and cnn,” *Applied Acoustics*, vol. 172, no. 107562, 2021.
- [7] I. D. Jokić, S. D. Jokić, V. D. Delić, and Z. H. Perić, “One solution of extension of mel-frequency cepstral coefficients feature vector for automatic speaker recognition,” *Information Technology and Control*, vol. 49, no. 1, pp. 224–236, 2020.
- [8] B. K. Reddya and D. Delenb, “Predicting hospital readmission for lupus patients: An rnn-lstm-based deep-learning methodology,” *Computers in Biology and Medicine*, vol. 101, no. 101, pp. 199–209, 2018.
- [9] J. Ramírez and M. Flores, “Machine learning for music genre: multifaceted review and experimentation with audioset,” *J Intell Inf Syst*, vol. 55, pp. 469–499, 2020.
- [10] H. H. Aghdam and E. J. Heravi, *Guide to convolutional neural networks - A practical application to traffic-sign detection and classification*, 1st ed. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer International publishing, 2017.
- [11] U. Michelucci, *Applied Deep Learning: A Case-Based Approach to Understanding Deep Neural Networks*, 1st ed. Switzerland: Apress, 2018.

- [12] B. D. Bowes, J. M. Sadler, M. M. Morsy, M. Behl, and J. L. Goodall, “Forecasting groundwater table in a flood prone coastal city with long short-term memory and recurrent neural networks,” *Water*, vol. 11, no. 1098, 2019.

Appendix I

Code for MFCC extraction from audio recordings

```
import json
import os
import math
import librosa
import numpy

DATASET_PATH = "C:\Users\moama\Desktop\boaslibrary_z\littmann_after\"
JSON_PATH = "C:\Users\moama\Desktop\boaslibrary_z\littmann_after\data_littmann_after.json"
SAMPLE_RATE = 22050
TRACK_DURATION = 30 # measured in seconds
SAMPLES_PER_TRACK = SAMPLE_RATE * TRACK_DURATION

def save_mfcc(dataset_path, json_path, num_mfcc=39, n_fft=2048, hop_length=512, num_segments=5):
    """Extracts MFCCs from music dataset and saves them into a json file along with genre
    labels.
    :param dataset_path (str): Path to dataset
    :param json_path (str): Path to json file used to save MFCCs
    :param num_mfcc (int): Number of coefficients to extract
    :param n_fft (int): Interval we consider to apply FFT. Measured in # of samples
    :param hop_length (int): Sliding window for FFT. Measured in # of samples
    :param num_segments (int): Number of segments we want to divide sample tracks into
    :return:
    """

    # dictionary to store mapping, labels, and MFCCs
    data = {
        "mapping": [], #names of genres, ex classical, blues
        "labels": [], #output, ex 0=classical, 1=blues
        "mfcc": [] #input
    }

    samples_per_segment = int(SAMPLES_PER_TRACK / num_segments)
    num_mfcc_vectors_per_segment = math.ceil(samples_per_segment / hop_length)

    # loop through all genre sub-folder
    for i, (dirpath, dirnames, filenames) in enumerate(os.walk(dataset_path)):
```

```

# ensure we're processing a genre sub-folder level
if dirpath is not dataset_path:

# save genre label (i.e., sub-folder name) in the mapping
semantic_label = dirpath.split("/")[-1]
data["mapping"].append(semantic_label)
print("\nProcessing: { }".format(semantic_label))

# process all audio files in genre sub-dir
for f in filenames:

# load audio file
file_path = os.path.join(dirpath, f)
signal, sample_rate = librosa.load(file_path, sr=SAMPLE_RATE)

# process all segments of audio file
for d in range(num_segments):

# calculate start and finish sample for current segment
start = samples_per_segment * d
finish = start + samples_per_segment #number of samples per segment

# extract mfcc
mfcc = librosa.feature.mfcc(signal[start:finish], sample_rate, n_mfcc=num_mfcc, n_fft=n_fft,
hop_length=hop_length)
mfcc = mfcc.T

# store only mfcc feature with expected number of vectors
if len(mfcc) == num_mfcc_vectors_per_segment:
data["mfcc"].append(mfcc.tolist())
data["labels"].append(i-1)
print("{} {}, segment: {}".format(file_path, d+1))

# save MFCCs to json file
with open(json_path, "w") as fp:
json.dump(data, fp, indent=4)

```

```
if __name__ == "__main__":  
    save_mfcc(DATASET_PATH, JSON_PATH, num_segments=10)
```

Appendix II

Code and architecture for CNN

```
import json
import numpy as np
from sklearn.model_selection import
train_test_split
import tensorflow.keras as keras
import matplotlib.pyplot as plt

DATA_PATH = "C:\\Users\\moama\\Desktop\\boaslibrary_z\\littmann_after\\data_littmann_after.json"

def load_data(data_path):
    """Loads training dataset from json file.
    :param data_path (str): Path to json file containing data
    :return X (ndarray): Inputs
    :return y (ndarray): Targets
    """

    with open(data_path, "r") as fp:
        data = json.load(fp)
        X = np.array(data["mfcc"])
        y = np.array(data["labels"])
        return X, y

def plot_history(history):
    """Plots accuracy/loss for training/validation set as a function of the epochs
    :param history: Training history of model
    :return:
    """

    fig, axs = plt.subplots(2)

    # create accuracy subplot
    axs[0].plot(history.history["accuracy"], label="train accuracy")
    axs[0].plot(history.history["val_accuracy"], label="test accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy eval")
```



```

# create error subplot
axs[1].plot(history.history["loss"], label="train error")
axs[1].plot(history.history["val_loss"], label="test error")
axs[1].set_ylabel("Error")
axs[1].set_xlabel("Epoch")
axs[1].legend(loc="upper right")
axs[1].set_title("Error eval")
plt.show()

def prepare_datasets(test_size, validation_size):
    """Loads data and splits it into train, validation and test sets.
    :param test_size (float): Value in [0, 1] indicating percentage of data set to allocate to test
    split
    :param validation_size (float): Value in [0, 1] indicating percentage of train set to allocate
    to validation split
    :return X_train (ndarray): Input training set
    :return X_validation (ndarray): Input validation set
    :return X_test (ndarray): Input test set
    :return y_train (ndarray): Target training set
    :return y_validation (ndarray): Target validation set
    :return y_test (ndarray): Target test set
    """

    # load data
    X, y = load_data(DATA_PATH)

    # create train, validation and test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
    X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train,
        test_size=validation_size)

    # add an axis to input sets
    X_train = X_train[..., np.newaxis]
    X_validation = X_validation[..., np.newaxis]
    X_test = X_test[..., np.newaxis]

    return X_train, X_validation, X_test, y_train, y_validation, y_test

def build_model(input_shape):

```

```

"""Generates CNN model
:param input_shape (tuple): Shape of input set
:return model: CNN model
"""

# build network topology
model = keras.Sequential()

# 1st conv layer
model.add(keras.layers.Conv2D(1024, (3, 3), activation='relu', input_shape=input_shape))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(1, 1), padding='same'))
model.add(keras.layers.BatchNormalization())

# 2nd conv layer
model.add(keras.layers.Conv2D(256, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(1, 1), padding='same'))
model.add(keras.layers.BatchNormalization())

# 3rd conv layer
model.add(keras.layers.Conv2D(16, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(1, 1), padding='same'))
model.add(keras.layers.BatchNormalization())

# flatten output and feed it into dense layer
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512, activation='relu'))
model.add(keras.layers.Dropout(0.3))

# output layer
model.add(keras.layers.Dense(4, activation='softmax'))

return model

def predict(model, X, y):
    """Predict a single sample using the trained model
    :param model: Trained classifier
    :param X: Input data
    :param y (int): Target
    """

```

```

# add a dimension to input data for sample - model.predict() expects a 4d array in this
case
X = X[np.newaxis, ...] # array shape (1, 130, 13, 1)

# perform prediction
prediction = model.predict(X)

# get index with max value
predicted_index = np.argmax(prediction, axis=1)

print("Target: { }, Predicted label: { }".format(y, predicted_index))

if __name__ == "__main__":

# get train, validation, test splits
X_train, X_validation, X_test, y_train, y_validation, y_test = prepare_datasets(0.25, 0.2)

# create network
input_shape = (X_train.shape[1], X_train.shape[2], 1)
model = build_model(input_shape)

# compile model
optimiser = keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=optimiser,
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

model.summary()

# train model
history = model.fit(X_train, y_train, validation_data=(X_validation, y_validation), batch_size=32,
epochs=1000)

# plot accuracy/error for training and validation plot_history(history)

# evaluate model on test set
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)

# pick a sample to predict from the test set

```

```
X_to_predict = X_test[40]
y_to_predict = y_test[40]

# predict sample
predict(model, X_to_predict, y_to_predict)
```

Appendix III

Code and architecture for RNN-LSTM

```
import json
import tensorflow as tf
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import matplotlib.pyplot as plt

# import tensorflow as tf
physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], enable=True)
# gpus = tf.config.experimental.list_physical_devices(device_type='GPU')
# tf.config.experimental.set_memory_growth(device=gpus[0], enable=True)

DATA_PATH = "data.olympus_before.json"

def load_data(data_path):
    """Loads training dataset from json file.
    :param data_path (str): Path to json file containing data
    :return X (ndarray): Inputs
    :return y (ndarray): Targets
    """

    with open(data_path, "r") as fp:
        data = json.load(fp)

    X = np.array(data["mfcc"])
    y = np.array(data["labels"])
    return X, y

def plot_history(history, way):
    """Plots accuracy/loss for training/validation set as a function of the epochs
    :param history: Training history of model
    :return:
    """
```

```

fig, axs = plt.subplots(2)

# create accuracy subplot
axs[0].plot(history.history["accuracy"], label="train accuracy")
axs[0].plot(history.history["val_accuracy"], label="test accuracy")
axs[0].set_ylabel("Accuracy")
axs[0].legend(loc="lower right")
axs[0].set_title("Accuracy eval")

# create error subplot
axs[1].plot(history.history["loss"], label="train error")
axs[1].plot(history.history["val_loss"], label="test error")
axs[1].set_ylabel("Error")
axs[1].set_xlabel("Epoch")
axs[1].legend(loc="upper right")
axs[1].set_title("Error eval")
plt.savefig('images/error_accuracy'+way)
#plt.show()

def prepare_datasets(test_size, validation_size):
    """Loads data and splits it into train, validation and test sets.
    :param test_size (float): Value in [0, 1] indicating percentage of data set to allocate to test
    split
    :param validation_size (float): Value in [0, 1] indicating percentage of train set to allocate
    to validation split
    :return X_train (ndarray): Input training set
    :return X_validation (ndarray): Input validation set
    :return X_test (ndarray): Input test set
    :return y_train (ndarray): Target training set
    :return y_validation (ndarray): Target validation set
    :return y_test (ndarray): Target test set
    """

# load data
X, y = load_data(DATA_PATH)

# create train, validation and test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=validation_size)

```

```

return X_train, X_validation, X_test, y_train, y_validation, y_test

def build_model(batch_input_shape):
    """Generates RNN-LSTM model
    :param batch_input_shape (tuple): Shape of input set
    :return model: RNN-LSTM model
    """

    aktiv='hard_sigmoid'
    aktiv1='softmax'
    N1=1024
    N2=256
    N3=32

    # build network topology
    model = keras.Sequential()

    # 2 LSTM layers
    model.add(keras.layers.LSTM(N1, input_shape=batch_input_shape, return_sequences=True,
    kernel_regularizer=keras.regularizers.l2(0.001)))
    model.add(keras.layers.Dropout(0.3))
    model.add(keras.layers.LSTM(N2, kernel_regularizer=keras.regularizers.l2(0.001)))
    model.add(keras.layers.Dropout(0.3))

    # dense layer
    model.add(keras.layers.Dense(N3, activation=aktiv, kernel_regularizer=keras.regularizers.l2(0.001)))
    model.add(keras.layers.Dropout(0.3))

    # output layer
    model.add(keras.layers.Dense(4, activation=aktiv1))

    return model

def draw(way):
    # get train, validation, test splits
    X_train, X_validation, X_test, y_train, y_validation, y_test = prepare_datasets(101/470, 0.2)

    # create network
    batch_input_shape = (X_train.shape[1], X_train.shape[2]) # 130, 13
    model = build_model(batch_input_shape)

```

```

# compile model
optimiser = keras.optimizers.Adam(learning_rate=1E-5)
model.compile(optimizer=optimiser,
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

model.summary()

# train model
history = model.fit(X_train, y_train,
validation_data=(X_validation, y_validation),
batch_size=32, epochs=1000)

# plot accuracy/error for training and validation
plot_history(history, way)

# evaluate model on test set
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)

metrics = history.history
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.savefig('images/loss_valloss'+way)

#plt.show()
print(y_test)

#predictions = model.predict(X_test[:7])
predictions = model.predict(X_test)
print("predictions shape:", predictions.shape)
#np.set_printoptions(precision=3)
#[print(*line) for line in predictions]
#print(predictions)

y_pred=np.argmax(predictions, axis=1)
print("y_pred shape:", y_pred.shape)
print(y_pred)

```



```

confusion_mtx = tf.math.confusion_matrix(y_test, y_pred)
plt.figure(figsize=(4, 4))
commands=("Boas0","Boas1","Boas2","Boas3")
sns.heatmap(confusion_mtx, xticklabels=commands, yticklabels=commands, annot=True,
fmt='g')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.savefig('images/matrix'+way)
#plt.show()
return

if __name__ == "__main__":
way=('OB4_grafer.jpg')
draw(way)

```