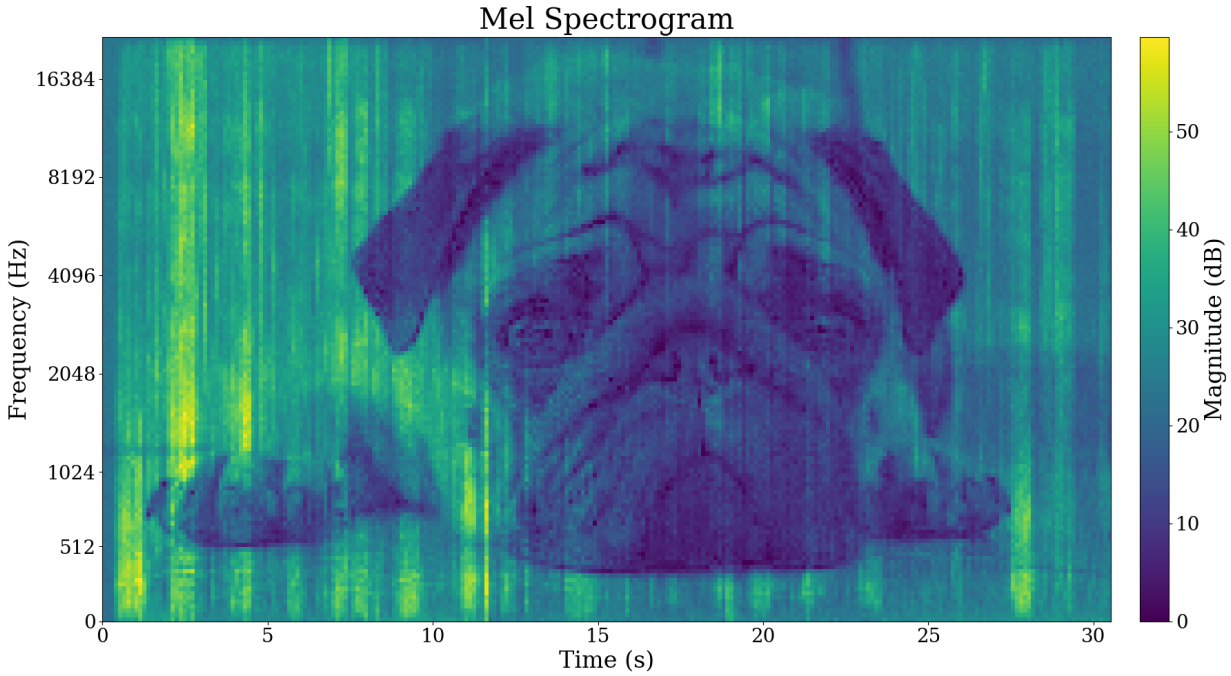

Diagnosing Brachycephalic Obstructive Airway Syndrome in Dogs Using Computer Vision and Machine Learning

Master's thesis in Physics

TIM PAGRELL

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Diagnosing Brachycephalic Obstructive Airway Syndrome in Dogs Using Computer Vision and Machine Learning

TIM PAGRELL

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Division of Material Physics
Chalmers University of Technology
Gothenburg, Sweden 2025

Diagnosing Brachycephalic Obstructive Airway Syndrome in Dogs Using Computer Vision and Machine Learning
TIM PAGRELL

Supervisor and Examiner: Magnus Karlsteen, Department of Physics,
Chalmers University of Technology

Master's Thesis 2025
Department of Physics
Division of Material Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A mel spectrogram combined with an image of a pug (royalty free from https://www.pickpik.com/pug-dog-cute-purebred-portrait-pup-144353) in Python.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Diagnosing Brachycephalic Obstructive Airway Syndrome in Dogs Using Computer Vision and Machine Learning
TIM PAGRELL
Department of Physics
Chalmers University of Technology

Abstract

Brachycephalic Obstructive Airway Syndrome (BOAS) is a breathing disorder that is common among dogs of certain breeds, impairing their quality of life. Diagnosing BOAS requires licensed veterinarians to undergo specialized training, which limits accessibility. There is demand for a more accessible solution, which machine learning could provide. This thesis evaluates two different approaches of machine learning to explore the possibility of classifying BOAS based on audio recorded from Android mobile phones, using Python. The first approach extracts signal features from the audio, which are used to train a simple machine learning model. The second approach relies on computer vision, using spectrograms; visual representations of the audio signals, to train a Convolutional Neural Network (CNN). Due to limited data, I employed augmentation techniques to artificially expand the dataset for training the models. My findings suggest that the spectrogram-based model is better suited for the problem, with a perfect prediction accuracy when the dogs were recorded after a short exercise test, suggesting strong performance under that condition. For dogs at rest, this model achieved an 81.7% accuracy, indicating somewhat promising results even under less favourable conditions. However, due to the limited dataset, the predictive performance was evaluated on few samples, and therefore additional data is needed for a more robust conclusion. Furthermore, one particular augmentation technique designed to account for differences in the recording devices' frequency response enhanced the model's general performance and could be further refined to improve accuracy, especially for dogs at rest.

Keywords: BOAS, machine learning, computer vision, audio, data augmentation, CNN

Acknowledgements

I would first like to express my sincere gratitude to my supervisor and examiner, Magnus Karlsteen, for your continuous guidance and encouragement throughout my project. Your support has undoubtedly contributed to the success of this thesis and made the process as smooth and enjoyable as possible.

I am also very grateful to my colleague, Jennie Berndtson, for your collaboration and positivity throughout the project, as well as for a fruitful opposition. It was a pleasure working alongside you.

A special thank you to Maria Dimopoulou, for your efforts to aid animals in need. Your help has been invaluable for understanding BOAS, and without the audio data that you had gathered this thesis would not have been possible.

Thank you Isabella Sykkö for your willingness to help at the start of my project, making the transition from your work on BOAS painless.

Tim Pagrell, Gothenburg, June2025

Contents

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AUC	Area Under the Curve
BOAS	Brachycephalic Obstructive Airway Syndrome
CNN	Convolutional Neural Network
ET	Exercise Test
FFT	Fast Fourier Transform
FN	False Negatives
FP	False Positives
FPR	False Positive Rate
MFCC	Mel-Frequency Cepstrum Coefficients
ONNX	Open Neural Network Exchange
RFG-S	Respiratory Function Grading Scheme
ROC	Receiver Operating Characteristic
TN	True Negatives
TP	True Positives
TPR	True Positive Rate
ViT	Vision Transformer

List of Figures

List of Tables

1
Introduction

Ever since the wolf was domesticated more than fourteen thousand years ago [1], dogs have been a central part of human societies. In the inception of this companionship, the dog was an ideal hunter-gatherer partner. In more modern times, as chores and priorities have changed, the dog's status in society has adapted. While some still fulfil various practical roles for humans, such as herding farm animals or aiding disabled people, many dogs today are exclusively social pets. As a result, the focus of breeding has shifted from enabling practical traits to forging desirable aesthetics, which in some cases has led to features that impede on the dog's wellbeing. This thesis will focus on detecting a breathing disorder that stems from such selective breeding of dogs, namely Brachycephalic Obstructive Airway Syndrome (BOAS).

This thesis is a collaboration between Chalmers University of Technology and the Swedish University of Agricultural Sciences (SLU). All programming for the project was done in Python [2].

1.1 Background

BOAS is a breathing disorder that affects brachycephalic dogs, which refers to dogs with a short nose and flat face. Some common brachycephalic dog breeds include French and English bulldogs and pugs [3]. The severity of the symptoms of BOAS varies between individuals, and the classification of the disorder is therefore divided into four categories according to the Respiratory Function Grading Scheme (RFG-S) [3], presented in Table 1.1 (though the RFG-S currently only "supports" pugs, French bulldogs and English bulldogs). To further simplify the diagnostication, these can be grouped into two main categories: BOAS negative (grade 0 + grade 1), meaning the dog *is not* in need of medical attention, and BOAS positive (grade 2 + grade 3), meaning the dog cannot breathe properly and *is* in need of medical attention [4]. In severe cases, the dog could even die from the effects of BOAS.

Licensed veterinarian Maria Dimopoulou, specialist on BOAS at SLU, explained during a Zoom meeting on January 23, 2025, that in order to effectively diagnose a dog with a BOAS grade, a veterinarian needs to have undergone special training. As a result, there are few clinics that offer this type of examination, which limits accessibility for dog owners with dogs in need of help. This is central to both the aim and limitations of this thesis, as will be detailed in Sections 1.3 and 1.4.

Table 1.1: Description of the four BOAS classification levels of severity according to the Respiratory Function Grading Scheme (RFG-S) [3].

BOAS grade 0	The dog is clinically unaffected and free of any breathing issues related to BOAS.
BOAS grade 1	The dog is clinically unaffected but suffers mild breathing problems linked to BOAS.
BOAS grade 2	The dog is moderately clinically affected and suffers moderate breathing problems due to BOAS.
BOAS grade 3	The dog is gravely clinically affected and suffers heavy breathing problems due to BOAS.

While not all individuals of the affected breeds suffer from BOAS, a large portion do. A study from Cambridge University, published in 2017, examined a total of 604 different dogs, approximately two hundred each of pugs, French bulldogs and English bulldogs. They found that less than 10% were BOAS free, with more than half of all examined dogs being classified as BOAS positive and therefore were in need of medical care [4]. Combining this with data from the Swedish Board of Agriculture (Jordbruksverket) [5], who state that at the end of 2024 there were 17 321 French bulldogs, 7 038 pugs and 2 135 English bulldogs registered in Sweden, we can approximate that there are likely more than thirteen thousand dogs in Sweden that needs medical attention for BOAS. This number is a very rough approximation, and as such should be used with caution. It does still, however, make it apparent that BOAS is a widespread problem, and that there is a need for easier and more accessible ways to diagnose the syndrome.

On the first of January 2025, *The Swedish Kennel Club* introduced a new rule for dog breeding; stating that dogs are required to have undergone a BOAS examination to be allowed for use in breeding, and that it goes against their guidelines to breed BOAS positive dogs [6]. This further motivates the need for a more accessible solution.

1.2 Related Work

This study is a continuation of a few previous projects. Moa Mårtensson began the efforts at Chalmers University of Technology on the subject with her thesis *Brachycephalic Obstructive Airway Syndrome (BOAS) classification in dogs based on respiratory noise analysis using machine learning* [7]. She used two different audio recording devices: a Littman electronic stethoscope and an Olympus linear PCM recorder Dictaphone. With the help from qualified veterinarians, she gathered audio data on BOAS affected dogs (some with grade 0), before and after a standardized exercise test, with both devices. She then trained a couple of neural networks of different structures. The best results were found using a long short-term memory recurrent neural network paired with the Dictaphone data, with an average accuracy around 86–87% at correctly classifying the BOAS grades.

Henrik Petterson and Olivia Stensöta then built upon Moa Mårtenssons work in their thesis *Data Augmentation for Audio Based Machine Learning Classifying Brachycephalic Obstructive Airway Syndrome (BOAS) in Dogs* [8]. They also considered a simplified approach using only BOAS positive versus BOAS negative and looked at other signal features and networks, as well as using spectrograms for their models. More audio recordings were gathered with the same devices that Moa Mårtensson had used, and data augmentation was utilised to further expand the limited data set. They found some success in optimizing the neural networks and using augmented data, mostly bringing the Littmann results up to par with the Olympus results, which itself only saw a small uplift. They also claim that the approach of only considering BOAS positive versus BOAS negative performed better when using Littmann data, while it performed poorly with Olympus, though the test batches were very small.

Finally, Isabella Sykkö tried a slightly different approach, focusing more on finding relevant signal features to feed to a logistic regression model, for BOAS positive and BOAS negative classification. She also developed an app for Android mobile phones, that acts as an interface for conducting the BOAS classifications in real-time. She found success using Dictaphone recordings but struggled when trying to apply the same solution to audio data from the phone's built-in microphone.

For this thesis, I explored two different approaches, expanding upon the feature-based approach with simpler machine learning models and the spectrogram-based approach with a deep neural network. Data augmentation was also explored and utilized to improve the models' performance due to data scarcity. On top of that, only data recorded from the built-in microphone of a smartphone was used, in contrast to the Dictaphone and stethoscope recordings from most of the previous projects.

This project was carried out in parallel with Jennie Berndtsons master's thesis *Acoustic Signal Analysis and Feature-Based Classification of BOAS: For the Health and Welfare of Brachycephalic Dogs* [9] on the same subject. We took slightly different approaches, as we worked independently and wrote our papers separately, but we continuously shared ideas and results between one another. Certain concepts and results in this thesis are borrowed from her work, where references are made without complete motivation. This thesis is still completely stand-alone, but the reader should consider Jennie's thesis [9] for the full account on such details.

1.3 Aim

The aim of this thesis is to make a working classification model for BOAS on audio recordings from the built-in microphone of Android mobile phones. Some partial objectives are to determine which of two machine learning approaches works best for the problem and to find which augmentation techniques can improve the model's performance.

The long-term goal is for this to develop into an app that makes it significantly

more accessible to examine BOAS in dogs, first for veterinarians but ultimately for dog owners as well, and in turn help give more brachycephalic dogs the medical attention that they need. In the future this could be expanded to Apple's iPhones, so that virtually everyone could access this service.

1.4 Limitations

As is the case for many machine learning problems, the main limiting factor for this thesis was limited data. Since professional diagnosis of BOAS requires specialised veterinarians, the gathering of data is slow. As a result, the access to audio data of BOAS affected dogs with a trustworthy classification was severely limited.

Due to the data scarcity, I limit the models to the binary classification task of diagnosing whether the dog is BOAS negative or BOAS positive. This is largely because BOAS class 3 is very rare in our dataset (only one dog), which makes it impossible for a model to classify with any level of certainty. In the end, re-training the model with more data should yield a better result and could be expanded to the full four BOAS classes. Therefore, the final product should be easily improved without much added development.

I will also only focus on developing the machine learning solution rather than working on any app development. That said, the future use in a different language to Python was always considered and affected decisions in the process, such as how the models were exported.

2
Theory

In this chapter, various core concepts relevant for this thesis are explored and presented. These include two different machine learning approaches (Section 2.1), model evaluation theory (Section 2.2), and data augmentation methods (Section 2.3).

2.1 Machine Learning Approaches

Machine learning is a very broad term, encompassing a large number of methods where a model is trained to make decisions or predictions based on a set of information. Different models naturally have different characteristics and areas in which they excel. However, it is not always straightforward to know which models and methods are best fit for a given problem. As such, testing different approaches to solve a problem with machine learning is a way to find the most appropriate solution [10].

The audio data present in this thesis, and audio data in general, contains more data points than what is feasible to input to most machine learning models: A 30-second-long audio clip with a common sample rate of 44 100 samples per second contains 1 323 000 sampled points. To solve this problem and feed the model material that it can efficiently train on, the number of data points needs to be greatly reduced, without losing important information. In Sections 2.1.1 and 2.1.2, two such approaches to tactically reduce the input size are presented, along with appropriate machine learning models for each.

2.1.1 Feature Extraction

The first approach is feature extraction, where different numerical features are extracted from the raw input signal and fed to the machine learning model [11]. Among such features could, for example, be the mean and standard deviation of the data. Particularly, in the case of audio data, this method can reduce the input size more than a thousandfold, which greatly reduces information redundancy when appropriate features are selected.

The specific features that are of interest to extract vary between problems and datasets and must be chosen accordingly. This process of choosing features can either be manual, which requires a good understanding of the problem to make in-

formed decisions, or automated using specialized algorithms [11]. There are of course benefits and drawbacks to both, but in general it is advisable to use automatic feature extraction for complex problems or when there is little prior knowledge. In either case, it is good practice to test different sets of features to examine which yields the best results.

2.1.1.1 Relevant machine learning models

With the much-reduced size and simple format of the data, there are a heap of supervised machine learning models that can be used, such as logistic regression, random forest algorithms or support vector machines, to name a few (and the ones that performed the best during testing). Deep neural networks can also be considered but are computationally inefficient in comparison to the simpler models and are more complex to use efficiently. One can train and compare multiple of these different models to find the best fit for the problem.

Logistic regression is a specific type of generalized linear model used to classify data, assigning probabilities of the different possible outcomes with the logistic function [12]. Being a linear model, it is limited to only recognising linear decision boundaries between features, which can be problematic if the dependencies in the data are more nuanced.

A random forest classifier is an ensemble of decision tree classifiers. Each decision tree trains on a subset of the data [13] and splits into different "branches" based off a set of decision rules that are inferred from the features found in the data [14]. A tree can then judge new data on which class it is most similar to according to the rules, and the final prediction is the majority vote from the entire ensemble of trees, or "forest".

Support vector machines construct a set of hyperplanes, tuned during training through the support vectors (points closest from both classes to the hyperplane), to separate different classes to the smallest margin (total distance between the hyperplane and support vectors) [15]. By using different kernel functions the shape of the hyperplane can be controlled and as a result the model can work with both linearly and non-linearly separable data. To efficiently choose an appropriate kernel function, however, sufficient knowledge on the trends of the dataset is required.

2.1.2 Conversion to Spectrogram

The second machine learning approach for handling audio data is to convert the waveform to a spectrogram, which is a two-dimensional image of the signal, with time on one axis and frequency on the other [16]. This is in theory also feature extraction, since the image is used to extract features in a way, but the resulting process for the user is fundamentally different which justifies the separation. The spectrogram is created through applying the Fourier transform on small periodic slices of the original high sampled data, mapping out the frequency over time [16]. To better match human hearing, the amplitude is converted to the decibel scale. A

further improvement can be made by using the "mel scale" for the frequency, which accounts for the fact that the human frequency perception is more-so logarithmic than linear [16]. As a result, lower frequencies are given higher resolution compared to higher frequencies. The resulting image is known as a mel spectrogram, see example in Figure 2.1.

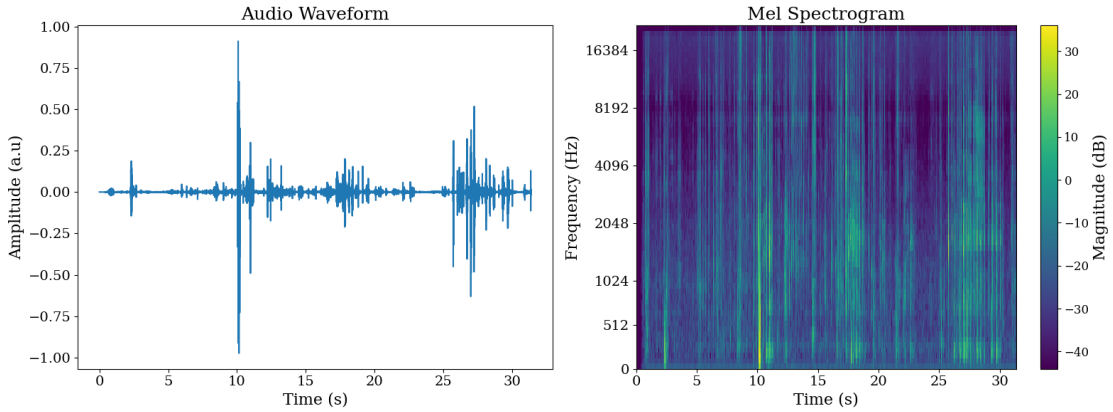

Figure 2.1: A plot of the waveform of an audio signal (left), and an image of the corresponding mel spectrogram (right).

Whether the decibel and mel scales improve the performance of the trained model depends on the nature of the audio signal. Sound profiles that are similar to what humans can differentiate between are a good match for these logarithmic scales [16]. For less familiar waveforms, for example where there is very little difference in peak loudness, or where a small frequency range at very high frequency is of interest, another scale may be more suitable. Similarly to the approach of feature selection, one could test different versions of the input, in this case different scales and resolutions, to find the spectrogram that the machine learning model performs the best with.

Relevant machine learning models

With a two-dimensional image, the size of the input is, in this case, reduced by a factor of around 10–100, depending on the settings of the spectrogram. The task has entered the realm of computer vision, for which two different deep neural network architectures are to be considered for the machine learning model, namely a Convolutional Neural Network (CNN) and a Vision Transformer (ViT). Historically, for a long time (in the perspective of artificial intelligence) CNNs have been dominant in image classification tasks, such as on ImageNet [17]. In more recent times, ViTs have emerged to reach a higher performance ceiling compared to CNNs in most relevant tasks, *when given enough data to train on.* The most crucial factor to consider when choosing between these two models is the number of available data, with a ViT requiring significantly more data than a corresponding CNN- in the order of millions. For this simple reason, a ViT was out of the question for this thesis, since the available data at the start of the project was less than a hundred audio files.

The choice of machine learning model for this approach to the problem is therefore a CNN. Deep neural networks are neural networks with more than one hidden layer, meaning layers that are between the input and output layers and that are therefore neither read nor written to directly [23]. The values of the neurons in the hidden layers are, in a supervised feed-forward network like the CNN, trained through backpropagation, where they are updated according to an optimizer function, to "learn" the correct answer for future predictions. The training consists of several epochs, where all the available data is run through once per epoch, and the learning is limited (or enabled) by a learning rate that must be tuned to balance speed of learning with consistent model results, while avoiding overfitting.

CNNs work by applying filter kernels that downsizes the resolution of an image (or any 2D grid-like data) while increasing the number of data channels, which serves as studied/learned features, multiple times throughout the different deep layers [18]. With this process, during the training the filters "learn" different spatial features like edges, shapes and patterns, which can be used to predict what another image contains. Again, to find the best layout for the CNN, trial and error is the method, adding or removing layers, changing the number of channels or the kernel sizes of the filters. In general, a well working structure is found by applying pooling layers after either a single or multiple convolutional layers, and repeating that substructure, before a final fully connected layer [18]. Regularization components, such as dropout layers and normalization layers, are added once a good structure has been found, to reduce overfitting.

2.2 Model Evaluation

To be able to compare different models and make decisions about which performs better, different ways to evaluate the models can be used. The simplest metric to study is test accuracy, found by feeding the model a set of unseen test data, and comparing the number of correct predictions with the total number of cases. A common train-to-test ratio is holding 80% as training data and leaving the remaining 20% to test on for evaluation. The problem with this simple approach is that not all data is used to train the model, which may hold it back from reaching its potential, and is especially problematic when the data is scarce.

One way to utilize all available data is with K fold cross-validation, where the dataset is split into K different folds (subsets) of data. The data is then used to train K different instances of the same model, where the Kth model trains on all folds except for the Kth fold, which is used as the test set [19]. See Table 2.1 for a reference example where $K = 5$. This way, all data is used in $K - 1$ of the models, and no data is necessarily left out completely (although it can be a good idea to keep another unseen subset of data to test on afterwards if enough data is available). The performance of each model is evaluated and averaged to find the overall score after training. Likewise, when classifying field data, the output of each model is summarized, either averaging over their Boolean classifications or their confidence/probability scores.

Table 2.1: The train-to-test split of a 5 fold cross-validation scheme.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Model 1	**Test**	Train	Train	Train	Train
Model 2	Train	**Test**	Train	Train	Train
Model 3	Train	Train	**Test**	Train	Train
Model 4	Train	Train	Train	**Test**	Train
Model 5	Train	Train	Train	Train	**Test**

Other than simply studying the validation accuracy of a classifier model, there are more nuanced metrics that can be considered. In Sections 2.2.1–2.2.3, three such metrics are introduced.

2.2.1 Confusion Matrix

The confusion matrix is a square matrix of dimensions $n \times n$, where n is the number of classes, that displays every prediction of the model in two dimensions: The predicted label and the true label. The matrix element in row i and column j shows the number of model classifications where the data is of class i, predicted to be of class j [20]. As such, correct predictions are located on the diagonal of the matrix, and the off-diagonal elements are incorrect predictions. An example 3×3 confusion matrix is shown in Figure 2.2. It is fundamentally quite similar to the simple validation accuracy, since the weight on the diagonal represents exactly that, while it expands on the amount of useful information that it contains by showing which classes the model confuses with each other, hence the name.

In the case of a binary classification, the four elements of the resulting matrix represent True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). From these quantities, some statistics can be calculated:

- True Positive Rate (TPR), also known as recall or sensitivity and expressed as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$, is a useful metric when considering imbalanced datasets with few actual positives, where FN have bigger consequences than FP.
- False Positive Rate (FPR), also known as $1-$ specificity and expressed as $\mathrm{FP}/(\mathrm{FP}+\mathrm{TN})$, is useful when FP are particularly undesirable, i.e when reducing the probability of false alarms.
- Precision, expressed as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$, measures the fraction of predicted positives that were actually positive, and is useful when ensuring that positive predictions are to be accurate.

These statistics [21] are used in the two following metrics.

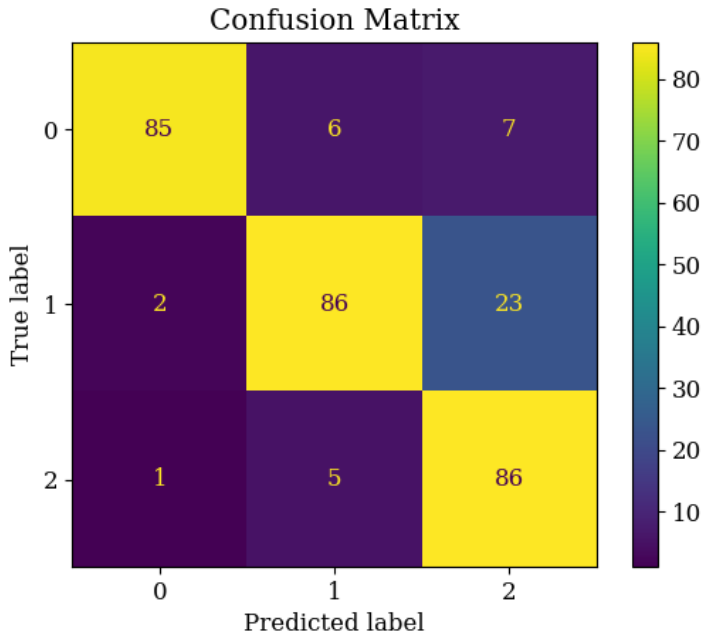

Figure 2.2: An example 3×3 confusion matrix generated from controlled random data, simulating inference of a model with three classes.

2.2.2 F1 Score

The F1 score is the harmonic mean of precision and recall, given mathematically by

$$\mathrm{F1} = 2 \times \frac{\mathrm{precision} \times \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}.$$

It balances the importance of the two metrics and is useful when handling datasets with class-imbalance [21]. For example, on a validation dataset with 80% negative-class data, a model that always predicts negative would have an 80% accuracy, while the F1 score would be zero, which better represents its ability to differentiate between the two classes. In general, a high F1 score corresponds to a good balance between recall and precision, and a well-trained model. Therefore, especially in scenarios with class-imbalanced data, it is preferable over validation accuracy when evaluating a model.

2.2.3 Receiver Operating Characteristic Analysis

The Receiver Operating Characteristic (ROC) analysis gives a different performance metric of a binary classifier, as it shows the models TPR versus its FPR at various discrimination thresholds [20]. The ROC curve is presented as a graphical plot that shows the trade-off between detecting as many positive cases as possible and reducing false alarms, see example in Figure 2.3. The Area Under the Curve (AUC) is used to score the model, with the maximum AUC score of 1.0 being ideal, and a score 0.5 corresponding to random guesses, or coin tossing. The larger the AUC score, the better the performance of the model, with AUC > 0.8 being considered

good [22]. It is commonly employed for diagnostic tests in the medical field, where sensitivity and specificity can be important metrics to consider.

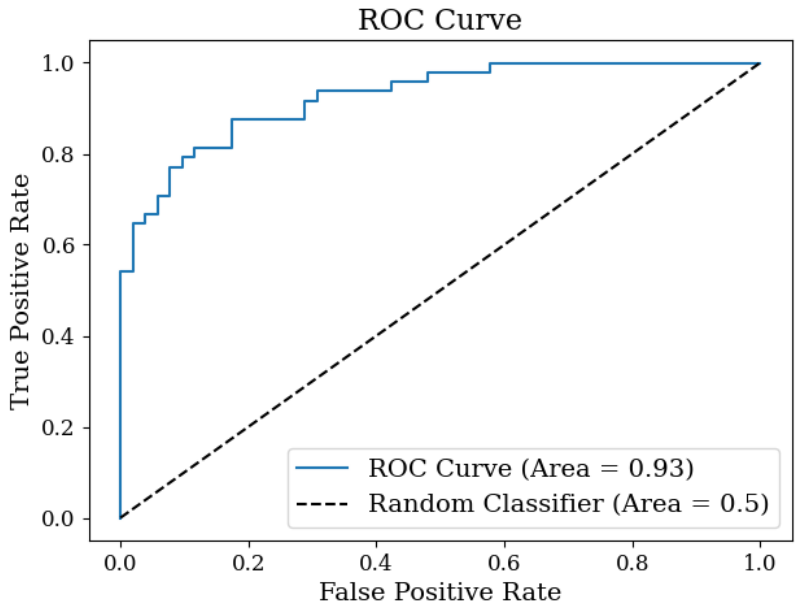

Figure 2.3: An example ROC analysis generated from controlled random data, simulating inference of a binary classification model.

2.3 Data Preprocessing and Augmentation

Preprocessing the input data for a machine learning model is an important part of optimizing its performance. A universal method to preprocess the data is by normalizing to zero mean and unit variance. In the case of the spectrogram input data, this means scaling and shifting the signal data, while in the feature extraction model it is the features that are normalized. This is done to the data while training, which is beneficial in many ways, for example that the gradients in the loss function remain modest in amplitude, ensuring stability [23]. Crucially, the same normalization must be applied to data that is to be classified, for the model to work.

Another way to preprocess data is by applying augmentations. Data augmentation is a regularization technique that is commonly employed in machine learning, with the purpose of artificially generating new data from existing data through small transformations [24]. The benefit of this is twofold: Firstly, it addresses data scarcity by increasing the size of the dataset that the machine learning model can train on, which ensures better performance for some models, especially neural networks. Secondly, it can help mitigate overfitting by helping the trained model generalize over variations in the data, making it more robust. It is, however, still important to note that augmentations cannot replace additional original data gathering, but it can be a cheap way of improving the accuracy of a model when data is scarce.

An important aspect to consider for data augmentation is the strength of the transformations [24]. If the information is altered too much, it may no longer be recognizable as the original class, making it non-credible. This could, for example, be scaling the pitch of an instrument to a register where the instrument cannot operate, removing the credibility of the artificial data. Training a model on such material would not be beneficial for its generalizability since it does not replicate any real data. Meanwhile, the model may put weight to features that are not relevant, reducing its performance on actual data. It is therefore crucial to keep the transformations modest, ensuring that the augmented data is credible.

There are two approaches to data augmentation: online and offline augmentation [24]. During online augmentation, a randomly transformed version of the original data is produced every time the model requests inputs *during* training. This essentially results in an infinitely large dataset (though close copies of the original data), with the downside of requiring extra computational resources every epoch, which increases the training time. With offline augmentation the transformations are instead pre-computed *before* the training of the model. This takes weight off the computational resources during training but requires more storage and means the amount of different augmented versions of the data is limited. Both approaches can be efficiently combined in a transformation chain, using online augmentation for computationally cheap transformations, and offline augmentations for heavier transformations or when definite control is required.

Sections 2.3.1–2.3.8 present a variety of data augmentation techniques that are relevant for this thesis.

2.3.1 Data Slicing

Data slicing is essentially dividing a large piece of information into several smaller pieces of information, such as slicing a ten-second-long audio clip into two five-second-long signals. This is done to increase the size of the dataset, while (for some approaches, such as the spectrogram one) reducing the input size of the model. To efficiently use this technique, it is important to consider the period of possibly significant features, as to not lose information.

Slicing of data can also be done with overlap between the slices, which yields an even larger increase of the dataset size, see example in Figure 2.4. When doing so, it is vital that the data with overlap are placed in the same fold from which the model is trained on, to make sure that there is no overlap between training and validation data. In this project, the data slicing process needs definite control for strict fold management, and is therefore appropriately done *offline*, even though the computation is cheap.

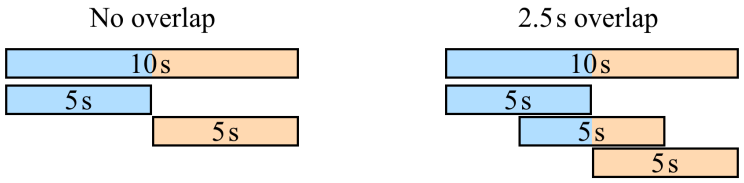

Figure 2.4: A visualization of the principle of slicing a ten-second-long signal into five-second-long segments, with no overlap and 2.5 s overlap respectively.

2.3.2 Noise Addition

The addition of noise is an important transformation in machine learning that entails overlaying the input with randomized noise. There are different types of noise that can be employed for different effects, but Gaussian noise is easy and common to add for model robustness. Adding such white noise helps prevent overfitting, since the exact form of the signal is randomly altered, while the underlying patterns remain [25]. Also, by varying the strength of the noise in combination with normalisation of the input data, different audio levels are emulated, which improves the model's ability to generalize. As the strength of adding noise is by ensuring that every instance of the same original data is slightly different during training, and with it being a computationally cheap transformation, it should be applied *online*.

The random nature of added Gaussian noise ensures that it is distributed uniformly along both the time and frequency domains. A potential problem with this transformation is that this noise can drown out signal features of low amplitude. Adding noise should therefore be done with caution, to make sure important sounds are distinguishable.

Another related method, that falls under the same category of "adding noise", is randomly adding background noises, such as environmental sounds that can occur around the intended area of use [25]. This could in a veterinary context for example be faint sounds of other animals, people talking, footsteps, or medical instruments. Of course, an ideal environment is free of any such sounds, but if that cannot be guaranteed in the general use case, this way of adding noises artificially can "teach" the model to ignore similar unhelpful sounds.

2.3.3 Amplitude Scaling

Amplitude scaling is a simple transformation that multiplies the entire signal with a scaling factor [25]. The result is a signal with an altered loudness; simulating being recorded at different distances from the microphone. This is not useful if the signal is also being normalized, like in the case of the spectrogram input, since the effect of the scaling is nullified. However, for a feature extraction-based model where it is the features that are normalized rather than the input signal itself, amplitude scaling can be used to make the model more robust.

2.3.4 Audio Equalization

Audio equalization is a technique that alters the frequency response of a signal by applying an equalization curve, meaning it changes the amplitude of the components of the waveform differently depending on the frequency. To clarify, the term *frequency response* is often used as a binary frequency range in which a microphone/speaker can record/play sound. However, in this thesis the term is used to describe a device's sensitivity to different frequencies. The equalization curve determines the scaling factor as a function of the frequency; see randomized example in Figure 2.5. This augmentation is useful when handling data from different devices, because different microphones have different frequency responses [26]. It especially appears like the difference is largest at higher frequencies. The use of audio equalization can simulate having data with different frequency responses, which is useful when the number of available devices to gather training data with is limited. The trained model can thereby generalize better over the frequency response and "learn" to handle data from various devices, becoming more robust.

The equalization curve can be generated in a range of ways, for example by combining different filters, or drawing random points spaced out in the frequency domain and interpolating between the points. Depending on which method is used to create the curve, the computational cost will vary, which can affect whether the transformation can be used *online*, or if it should be applied *offline*.

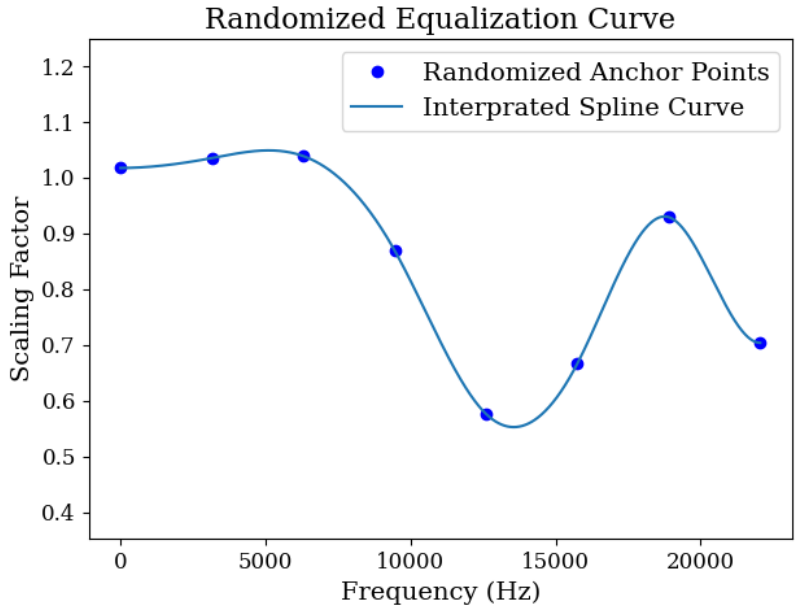

Figure 2.5: Randomized equalization curve, generated from a few randomized anchor points and `scipy.interpolate.make_interp_spline` in Python.

2.3.5 Time Shift

Time shifting is a transformation that simply shifts the signal waveform to the left or right by a random amount in the time domain, with excess signal looping back around to fill the resulting gap. The main use case is for audio data where the time order is not of importance, which is true for this thesis, but not for example in speech recognition [25]. This method alters the input to aid against data scarcity, improving the model's ability to generalize. Time shifting is computationally cheap and can therefore with benefit be applied *online*.

2.3.6 Time Stretch

Time stretching changes the speed of a waveform, **without** changing the frequency of the signal [25]. It should be noted, however, that the amplitude changes as a result of how the transformation is calculated. The method is computationally expensive and is therefore preferably done *offline*.

2.3.7 Pitch Scaling

Conversely to time stretching, pitch scaling changes the frequency (pitch) of the sound **without** changing its speed [25]. Again, the amplitude of the signal is also altered because of the transformation. Like with time stretching, pitch scaling is also computationally expensive and is therefore for efficiency left to *offline* use.

2.3.8 Frequency and Time Masking

Frequency and time masking are two closely related transformations that can be efficiently applied to a spectrogram representation of audio by simply removing data along the frequency and time axis respectively. The data is commonly replaced with either zeros or the mean value of the spectrogram and is done to give the model "blind spots", which makes it more robust after training [27]. Masking in either domain are cheap transformations that should be applied *online*.

3
Methods

This chapter presents the methods adopted to reach the thesis goal of training a machine learning model to diagnose BOAS in affected dog breeds. To begin with, Section 3.1 presents how the data was acquired, including both the new data as well as the data that was gathered during Isabella Sykkö's project. Then, in Section 3.2, the methodology for developing the machine learning models, using the two approaches covered in the Theory, are explained.

3.1 Acquiring data

The process of gathering data is a crucial step in machine learning, because without *good* data to train on, the model's performance and usability will be limited. It is therefore important to be consistent with the methodology and use trustworthy classifications.

The general method was to record two 30 second audio clips per dog, one time during rest, and the second time after a three-minute-long Exercise Test (ET). For some dogs, more than one recording before and after the ET was collected, sometimes using a different phone to record. Also, in a few cases, one of the two files were lost, meaning not all dogs have the same number of files before as after the exercise. Most of the files were recorded into the Waveform Audio File Format (WAVE: `.wav`).

Most of the available data was gathered during Isabella Sykkö's project, with the help of licensed veterinarian and BOAS specialist Maria Dimopoulou from SLU; the extent of this data is presented in Table 3.1. Due to data scarcity, it was decided to continue gathering data during mine and Jennie's projects. Jennie visited different licensed veterinarians conducting BOAS examinations to record audio of dogs along with their diagnosed BOAS grade, sticking to the same format and method to the best of her ability without interfering with the process. Along with recording to WAVE, some additional data was recorded in parallel using a different android device and file format (MPEG-4 Audio: `.m4a`) for analytical and testing purposes. The two devices used were a *Oneplus A5000* and a *Samsung Galaxy Note 20 Ultra*. For a more complete account of Jennie's data gathering, read her thesis [9]. We also sought assistance from Maria to gather even more data, but she did not manage to conduct any BOAS examinations during this time. An updated version of Table 3.1, summarizing all available data by the end of our projects, is presented in the Results chapter.

The sample rate of all the old files were 44 100 Hz, whereas the new files that were recorded by Jennie had a sample rate of 48 000 Hz.

Table 3.1: Information on the audio data available at the start of the thesis work, carried over from Isabella Sykkö's project. The table displays the number of unique dogs as well as the number of files, both before and after the exercise test (ET), per BOAS grade. The total number of dogs were 31, but for two of them the recording before the ET was missing. The same was true for three dogs' recordings after ET.

	Number of Unique Dogs		Number of Files	
	Before ET	After ET	Before ET	After ET
BOAS grade 0	6	4	8	4
BOAS grade 1	13	13	24	22
BOAS grade 2	9	10	11	12
BOAS grade 3	1	1	2	2
Total	29	28	45	40

The MPEG-4 Audio file format was not supported in the Python packages that was used, and therefore the files of this format had to be converted to WAVE first (Convertio was used here, but the conversion process is not important). This was the main reason for using a different format in the first place; to confirm that the audio file format conversion will not destructively change the properties of the files. The files were deemed unaffected after the conversion.

3.2 Machine learning model development

The general development for the two different approaches to the machine learning problem was focused on maintaining similarity, and is presented in Section 3.2.1, while the specifics for both approaches are described separately in Sections 3.2.2 and 3.2.3 respectively. The method for gathering the final performance metrics, that is later presented in the Results Chapter, is explained in Section 3.2.4. All code was written in Python 3.12 [2], with the following central packages: PyTorch 2.5.1 [28] for building and training CNNs, Scikit-learn 1.5.2 [29] for various classifier models and model evaluation, TorchAudio 2.5.1 [30] and librosa 0.10.2 [31] for handling the audio files and signal augmentation/transformation, and openSMILE 2.5.1 [32] for feature extraction. All final code is presented in Appendix B, along with a link to the corresponding GitHub repository.

Since the aim of this thesis is to make a model that can be used inside an Android app, with the ambition of also expanding the reach to Apple and other devices in the future, having the model only work in Python could be a problem. There may be a way to communicate with a Python instance remotely from such an app, but to stay on the safe side: All models were saved using the Open Neural Network Exchange (ONNX) format, which can be loaded into many different programming languages, including for both Android and iOS applications. This was achieved using

SKLEARN-ONNX 1.18.0 [33], and natively in PyTorch, whereas for inference/testing in Python, ONNX 1.17.0 [34] and ONNX Runtime-GPU 1.20.1 [35] were used.

Many classes and functions were created to be utilized across different scripts. It was decided to keep all these utilities in a single lone Python file to be called from all other scripts made for the project: `functions_and_classes.py` (Appendix B.1).

3.2.1 General development

To start off, a metadata file was created to store necessary information about the audio recordings, such as file name and BOAS grade, to be able to efficiently load it into Python with PANDAS [36, 37]. With the intention of using K fold cross-validation, the files were distributed into five folds, as evenly as possible within each BOAS grade, represented by another column in the metadata file. This distribution into folds was initially done manually since there was little data to handle, but a Python script (`distribute_into_folds.py`, Appendix B.2) was later constructed to achieve a similar result automatically with an arbitrary number of files, for any value of K, as futureproofing. The logic behind the fold distribution was to evenly distribute files into the folds with respect to BOAS grade, as well as between BOAS positive and negative. On top of that, extra folds were added for some files when the split could not be done evenly, so that when slicing data, different parts of the original file could go into different folds, without overlap.

Two scripts were then created, the first of which (`slice_data.py`, Appendix B.3) slices data to a specified length, with overlap between the slices if desired. In the case of an uneven number of files to split between all five folds, a file could be split up into two or more folds, crucially without overlap between the folds. The second script (`augment_data.py`, Appendix B.4) transformed the data with *offline* augmentations, composed of time shifting, pitch scaling and equalization. The equalization curve was generated using randomized anchor points, interpolated with splines using `scipy.interpolate.make_interp_spline` [38]. See Figure 2.5 for example, and `AudioUtil.apply_equalization()` in Appendix B.1 for the corresponding Python function. Both scripts generate a new metadata file, containing information about the slice timings and augmentations respectively, for the "new" data.

The training of the models was designed to be easily configurable for different versions of data, to be able to analyse which combinations would yield the best results, all from a single Python script (per approach). The five main parameters for the model configuration were as follows:

- Whether to use data recorded before ET or after ET. These were separated due to the vastly different nature of the audio signals, meaning different features would be relevant in each case.
- Which *offline* augmentations should be used, if any.
- If sliced data or the full signal were to be used.

- If Jennie's adaptive high-pass filtering was applied (details below).
- Whether to use binary classification for BOAS negative and BOAS positive, or the complete set of four BOAS grades.

The first four were explored in this thesis, with the fifth being implemented for a future where more data is available and the full BOAS grade classification becomes viable. Within each configuration of the models, many more parameters naturally exist to specify the training and structure of the model further. These extra parameters were iterated to find the best version of each configuration, before finally comparing the different models with each other using the evaluation techniques covered in the Theory chapter.

The fourth parameter, to apply an adaptive high-pass filter to the signal, was added late in the project, as Jennie had found positive results after inventing it. The high-pass cut-off frequency is determined as the frequency where the cumulative energy of the power spectrum surpasses a given threshold, set to 15% during my utilization of it. The Python function is defined as `AudioUtil.custom_high_pass_filter()` in Appendix B.1, and for the full details on this filter, please consult Jennie's thesis [9]. Because this was added as an afterthought, for simplicity it was included as an online augmentation, despite being quite computationally expensive.

For the *offline* augmentations, three different setups were considered: No *offline* augmentations, only an equalization curve, and a combination of pitch scaling and time stretching. A fourth setup of combining all three *offline* augmentations was planned, but after not finding positive results for pitch scaling and time stretching, this was dismissed in the interest of conserving time and effort. The equalization curve was generated randomly as presented in Figure 2.5, with the gain factor for each point being drawn from a Gaussian distribution with $\mu = 1$ and $\sigma \in [0.05, 0.3]$ increasing with the frequency, to capture the fact that the spectral response appeared to differ more at higher frequencies for different devices. Both the pitch scaling and time stretching was implemented using the corresponding functions in the `librosa.effects` module, with rather modest transformations as to not remove the signals credibility. The pitch scaling factor was drawn uniformly from $[-1, 1]$ semitones, corresponding to a maximal change of $1 - 2^{1/12} \approx 6\%$, and the time stretch factor from $[0.95, 1.05]$, meaning at most 5% change.

The online augmentations were implemented differently for the two approaches and are therefore specified further in their corresponding section below. The specific Python functions defined for the different augmentations, both online and offline, can be found in the utility class called `AudioUtil` in Appendix B.1.

All training in both approaches were carried out with the data represented in Table 3.1, whereas the newly recorded data was kept as a separate test set. The training data was split into five folds using `distribute_into_folds.py` to use in 5 fold cross-validation. This controlled approach to K fold cross-validation has two main benefits over a randomized one: Firstly, the limited amount of data means that randomized fold allocation is likely to yield folds with large class discrepancies, es-

pecially since some classes are already less common than others in the dataset. This is problematic since the models need to train and validate on all classes, and the problem is solved by overseeing the fold distribution. Secondly, removing the cross-validation fold distribution as a factor of uncertainty, ensuring that the folds are the same for all models, is helpful in the search of the best machine learning models for the task. Five folds were chosen since the 80/20-split for training/validation is a common choice in machine learning. Fewer folds would mean even less data to train on, while more would risk a larger class discrepancy than otherwise necessary.

3.2.2 Feature extraction-based model

To begin this branch of the project, a demonstrative tutorial [39] of a feature extraction-based model for audio data was followed. The tutorial was based on a dataset with short audio slices of city sounds (UrbanSounds8K [40]) but had a solid core architecture that could easily be adapted to support our 30 second audio files of dogs breathing. The central structure, that served as the starting point for this half of the project, was mainly a system that read a metadata file, loaded in the audio files mapped in the metadata and extracted a set of predetermined features using LIBROSA. The features were then fed into a machine learning model from SCIKIT-LEARN with *randomized* 10 fold cross-validation.

The structure was modified further, switching out the audio handling to TORCHAUDIO (to use the same functions as the spectrogram model), adding both *online* and *offline* augmentations, and using *controlled* 5 fold cross-validation. Another crucial rework made was of the feature extraction process: Originally, the architecture extracted a total of 82 features, composed of Mel-Frequency Cepstrum Coefficients (MFCC), spectral contrast, and chroma features from a short-term Fourier transform. Since we had limited knowledge of what features are suitable to use, this was exchanged with an automatic approach to feature selection in OPENSMILE. Using the `ComParE_2016` feature set, 6 373 features are swiftly extracted, though many likely lack relevance.

To find the features that were most suited for the BOAS classification task, a separate script was created (`find_opensmile_features.py`, Appendix B.5), in which all 6 373 features were extracted from every audio file in the training dataset and passed through SCIPYS [38] implementation of the Mann-Whitney U-test. I should mention that the OPENSMILE package was a find that Jennie made, along with pairing it with the statistical test, so the credit is hers. The test assesses whether a feature's distribution differs significantly between two classes and assigns a "U statistic" and associated p-value to quantify this difference. A smaller p-value suggests a greater separation between the classes, which indicates that the feature may be informative or discriminative with respect to the class label, making it more suitable for the task. For more information on this, please read Jennie's thesis [9]. The features with a p-value lower than 0.05 were selected, but a maximum of 100 features, in which case the 100 lowest p-value features were chosen. The names of the selected features were saved, to be used in both training and testing of the models.

With the extracted features, a range of different machine learning classifier models from Scikit-learn were trained in a separate script (`sklearn_training.py`, Appendix B.6), to find which models works best for our scenario. However, before feeding the features to the chosen model, they were first normalized to zero mean and unit standard deviation, feature-wise. This normalization data was saved to be available during testing. The models that performed the best initially on the most simple configuration (no augmentations, slicing, filters etcetera.), and that were therefore further used and analysed in the various configurations, were `linear_model.LogisticRegression()`, `ensemble.RandomForestClassifier()` and `svm.SVC()` (Support Vector Classifier) from Scikit-learn.

Due to the nature of the feature-based models, and the enclosed training of the Scikit-learn classifiers works, *online* augmentations cannot be incorporated as intended. To still see if some of those transformations can benefit the models, amplitude scaling and time shifting was implemented similarly to the *offline* augmentations, expanding the dataset before feature extraction rather than permuting it each epoch. This was made as an option for both the feature extraction during training, as well as for the feature selection in `find_opensmile_features.py`.

3.2.3 Spectrogram-based model

Correspondingly to the feature extraction-based model, a different demonstrative example [42], in which a spectrogram-based CNN was trained on the same city sounds dataset (UrbanSounds8K), served as the starting point for this approach. The demonstration had an architecture of loading in the audio data with the help of a metadata file, similar to the feature extraction tutorial, but this time using PyTorch and TorchAudio from the beginning. It featured *some* augmentations and the spectrogram conversion as part of a data-loader, used during training and testing of a CNN with PyTorch. However, it had a few shortcomings, such as incorrect normalization, and most importantly; it did not use any cross-validation structure, despite the dataset being designed for it, resulting in artificially overconfident validation accuracy.

The example was adapted to work with our format of audio data, the catalogue of augmentations was expanded to the extent covered in the Theory Chapter, separating *online* and *offline* augmentations, and model evaluation was added. The faults were addressed, ensuring accurate normalization of each spectrogram to zero mean and unit variance, and implementing a controlled K fold cross-validation scheme. Another important addition was a system for selecting the "best" model from the training stage, to combat over-fitting and getting lost in local minima. After each epoch, the model was run through a set of criteria to judge if it was the "new best" model. The main criterion served to maximize the validation F1-score, with a tiebreaker of maximizing the ROC-AUC score, and minimizing the loss function as the final decider. Together, these criteria chose a model that balanced precision and recall, ensuring good learning, generalization and robustness.

The finalized procedure of loading data, augmenting it, and transforming it into a spectrogram, is handled by the `SoundDS()` dataset class (defined in Appendix B.1). A dataset is created using this class, where each file in the chosen batch of data goes through the following steps:

1. The audio signal is loaded in with `AudioUtil.open()`.
2. The signal is resampled to a set sample rate with `AudioUtil.resampled()`, so that all files have the same rate. The sample rate was set to 44 100 Hz, as that was the lowest sample rate of all data and up-sampling can introduce unwanted signal artifacts. Down-sampling can also introduce artifacts in the form of aliasing, according to the Nyquist theorem [41], though that is easily avoidable by applying a low-pass filter to the signal during the transformation. This was done automatically by the `Resample` transformation function in TorchAudio. 44 100 Hz was also the sample rate of the majority of the data.
3. The signal is then "rechanneled" with `AudioUtil.rechannel()`, setting the number of channels to one, instead of the two channels that all data originally retained. This was done to halve the amount of data per file, as the two channels are close to identical anyway.
4. Then the signal is set to a fixed duration with `AudioUtil.pad_trunc()` to achieve a constant spectrogram size, which the CNN requires. The chosen duration for the sliced and non-sliced data was 6 s and 30 s respectively.
5. If the adaptive filter is requested, the filter is now applied to the signal using `AudioUtil.custom_high_pass_filter()`.
6. If augmentation is requested, `AudioUtil.time_shift()` is used to time shift the signal, and Gaussian noise is added by `AudioUtil.add_white_noise()`. The time shift was by a random amount between 0 – 100%, drawn uniformly to cover all possible "timelines". The Gaussian noise amplitude was set to the noise floor level of the current signal multiplied by a factor drawn from $\mathcal{N}(\mu = 0, \sigma^2 = 1)$. This was done to vary the detail of the noise level by different amounts between each epoch to ensure robustness and prevent overfitting, while sometimes increasing the noise level to generalize over.
7. The signal is transformed to a spectrogram by `AudioUtil.spectro_gram()`, with the size of the Fast Fourier Transform (FFT) set to 1 024 and the number of mel filterbanks set to 64. These parameters define the shape of the spectrogram, along with the duration and sample rate of the signal, and were chosen partly from the demonstrative example [42], and partly because attempts to vary them gave undesired results (either poor performance or computationally more expensive without a performance increase).
8. Finally, if augmentation is requested, frequency and time masking is now applied to the spectrogram using `AudioUtil.spectro_augment()`. 1 – 3 masks per dimension, drawn from $\mathcal{U}[1, 3]$, each replacing up to 10% ($\mathcal{U}[0, 10\%]$) of the total corresponding domain with the spectrogram mean were used.

When called, the output spectrogram from the dataset was, as previously mentioned, normalized to unity variance and zero mean. This would nullify the effect of amplitude scaling, which was hence excluded from this approach. Augmentation was used for training but not for validation and testing, because in the latter cases it is the original data that is of interest.

The training of the CNN models was handled in a separate script (`nn_training.py`, Appendix B.7), creating a PyTorch dataloader from `SoundDS` datasets, and loading the predefined `training()` and `inference()` functions. This training function includes the system for selecting the "best" model mentioned earlier, while the inference function is used to gather the performance metrics used by said system.

Cross-entropy was used as the loss function for all CNN models during training and inference due to its ability to maximize the likelihood of the data, thereby assisting the learning process. The AdamW optimization algorithm was used in training, for its balance between ease of use, speed, and generalizability, along with a cosine learning rate anneal strategy for strong learning early on and more-so fine-tuning towards the final epochs.

The development of the CNN models in PyTorch was guided by an article written by S. Ramesh [18]. In the article, a general and efficient methodology for building neural networks for computer vision is described, the surface of which is covered earlier in Section 2.1.2. Different models were created and iterated using these methods, trying various number of layers, features, kernels etcetera. Furthermore, an attempt was made at implementing ConvNeXt [43], a CNN architecture that famously has found success on ImageNet [44], but this did not yield good results. The final PyTorch CNN models, one optimized for sliced and one for non-sliced data, are defined as `AudioClassifierSlice` and `AudioClassifierNoSlice` respectively, in `pytorch_models.py` (Appendix B.8).

3.2.4 Gathering Results

To extract consistent and comparable performance metrics for the different models, a script (`onnx_inference.py`, Appendix B.9) was created that handles both approaches output models. It loads in the five ONNX model files that corresponds to 5 fold cross-validation of one model configuration and performs inference on a selected dataset. If the dataset is that which the models were trained on, the ith model only predicts the class of data from fold i, meaning no model is inferred on data it was trained on. If the dataset is not distributed into folds, like the newly recorded data that was not included in the training, all models predict the class of all files. In this testing case, the models were fed the files from both phones, meaning the metrics here reflects how the models handle the different frequency response.

The results for all five sub-models are averaged, with the chosen metrics being mean accuracy, ROC-AUC, mean probability of the *correct* class along with the corresponding standard deviation. For the tables presented in the Results Chapter,

each configuration of a model was inferred this way on both the training (validation) data, as well the separate testing dataset.

The previous `onnx_inference.py` script compares data to its true class, but that is only relevant when training the models and is not a real-world use case. A final script (`onnx_file_test.py`, Appendix B.10) was made to fill this gap, to only provide the prediction of the model, without the need for a "true class". Like the inference, it loads in the five ONNX model files, and for each audio file sent into the script (a realistic use case would be a singular file), it gives the prediction and probabilities of all five sub-models, along with an average probability and a vote-based prediction (most frequent binary classification). This is not used anywhere for the Results Chapter, but it is how the model would be implemented in an app and is therefore relevant for the project.

4

Results

In this chapter, the various results of the project are presented, before they are later discussed in the upcoming Discussion Chapter. Below, the result of the data gathering is presented as an updated version of Table 3.1, followed by a concrete demonstration from the newly recorded audio of the frequency response differing between devices. Then, Section 4.1 displays the final models that were determined to work best, along with performance metrics on the different configurations using those models.

As mentioned in the Methods Chapter, all new dog recordings were done in parallel by two different android devices. Five dogs were recorded this way, though only one was BOAS positive, which is seen going from Table 3.1 to Table 4.1. In the case of one of the dogs, the recording was carried out twice before ET, due to environmental noise.

Table 4.1: Information on the audio data available by the end of the thesis work. The table displays the number of unique dogs as well as the number of files, both before and after the exercise test (ET), per BOAS grade. The total number of dogs was now 36, but still two dogs' recordings before ET and three dogs' recordings after ET were missing.

	Number of Unique Dogs		Number of Files	
	Before ET	After ET	Before ET	After ET
BOAS grade 0	7	5	10	6
BOAS grade 1	16	16	32	28
BOAS grade 2	10	11	11	14
BOAS grade 3	1	1	2	2
Total	34	33	57	50

The different frequency response of two devices is showcased in Figure 4.1, where the frequency domain of two audio files recorded in parallel are shown. The frequency spectrum from the *Oneplus A5000* is shown as a solid blue line, whereas that of the *Samsung Galaxy Note 20 Ultra* is a semi-transparent orange line, resulting in the overlap appearing brown.

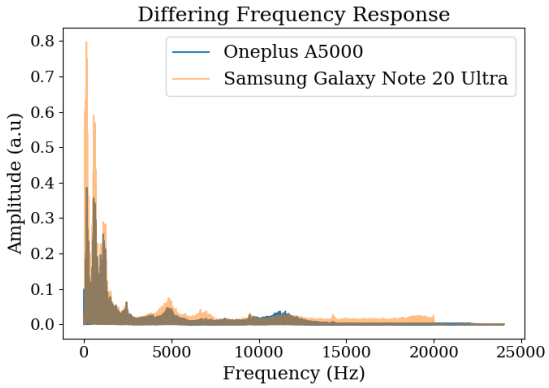

Figure 4.1: A graph that shows the frequency domain for two audio files, recorded in parallel using two different phones.

4.1 Machine Learning Results

This section is further split into two subsections, separating the result of the two approaches into Sections 4.1.1 and 4.1.2. The format is explained here.

First, each model that was deemed the best per approach is explained in detail. For the Scikit-learn models, this means a short code snippet of the model function call with the chosen parameters, while for the PyTorch models this entails the models layer summary from `torchinfo.summary()`.

This is followed by a table showcasing the performance metrics of the model in combinations with the different data configurations. The tables for the feature extraction-based models are only a select few highlight results, as the original tables were excessive and large. Those complete performance metric tables are instead left in Appendix A. For the spectrogram-based CNNs, the table presented in this chapter *is* complete.

The tables are structured such that for each configuration, the resulting accuracy (**Acc**), ROC-AUC score (**ROC**), mean probability of the true classification (**Prob**) and its corresponding standard deviation (**SD**) are shown, both on validation data and the new test data. The statistics are all from a combination of the $K = 5$ models' outputs. In the validation case, each model was inferred on the fold it was **not** trained on, and in the test case all models were inferred on the same testing data. The configuration columns are:

- **Sliced** - Tells whether the model uses sliced data.
- **Offline Aug** - Tells which *offline* augmentations were used during training, if any. **EQ** stands for the equalization curve augmentation, and **PSTS** stands for a combination of pitch scaling and time stretch transformation.
- **Filter** - Tells whether the adaptive high-pass filter is used.

The tables for the feature extraction-based models have an extra column: Since the

online augmentations are handled differently, **Online Aug** is added as a column to keep track of where *online* augmentations were used, if at all. There, **FS** stands for using the augmentations during the feature selection, and **FE** during feature extraction. Also, all the highlighted model configurations used non-sliced data, so that column has been dropped for the tables in Section 4.1.1.

The spectrogram-based CNNs all used the same *online* augmentation, so such a column is not needed for the general case. Also, since the spectrogram-based CNNs performance metric table in this chapter is complete, the best performance number per column for both before and after ET are written in bold font, to highlight the best results.

4.1.1 Feature Extraction-Based Models

Table 4.2 presents the highlighted performance metrics of the logistic regression model, whose function call was:

```
import sklearn.linear_model as lm

model = lm.LogisticRegression(
        max_iter=500,
        fit_intercept=True,
        tol=1e-5,
        solver='lbfgs')
```

Table 4.2: Some highlighted performance metrics of the feature extraction-based logistic regression model trained on different configurations of data.

Logistic Regression Highlighted Results			Before ET							
			Validation				Test			
Online Aug	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
None	None	No	0.978	0.964	0.913	0.181	0.660	0.722	0.607	0.333
FS	None	No	0.928	0.929	0.884	0.224	0.660	0.688	0.616	0.356
FS	EQ	No	0.953	0.945	0.909	0.201	0.720	0.753	0.699	0.365
FE	None	No	0.953	0.970	0.923	0.191	0.740	0.718	0.661	0.367
FE + FS	None	No	0.928	0.945	0.905	0.211	0.760	0.755	0.700	0.374
FE + FS	EQ	No	0.953	0.951	0.931	0.185	0.720	0.725	0.686	0.382

			After ET							
			Validation				Test			
Online Aug	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
None	None	No	0.980	1.000	0.942	0.123	0.650	0.812	0.703	0.337
None	None	Yes	0.935	0.998	0.901	0.181	0.767	0.994	0.710	0.294
None	EQ	No	0.975	0.993	0.931	0.155	0.683	0.916	0.712	0.359
None	EQ	Yes	0.865	0.964	0.876	0.256	0.717	0.950	0.708	0.370
FE	None	No	0.955	0.993	0.930	0.174	0.533	0.362	0.555	0.422
FE	None	Yes	0.910	0.959	0.887	0.236	0.683	0.936	0.670	0.392
FE	EQ	No	0.905	0.974	0.889	0.249	0.600	0.652	0.617	0.451
FE	EQ	Yes	0.935	0.966	0.895	0.249	0.683	0.844	0.688	0.419

The highlighted performance metrics of the random forest model is shown in Table 4.3, and the model function call was:

```
import sklearn.ensemble as en

model = en.RandomForestClassifier(
        n_jobs=-1,
        class_weight="balanced",
        n_estimators=200,
        bootstrap=True)
```

Table 4.3: Some highlighted performance metrics of the feature extraction-based random forest model trained on different configurations of data.

Random Forest Highlighted Results			Before ET							
			Validation				Test			
Online Aug	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
None	None	Yes	0.877	0.926	0.711	0.182	0.740	0.742	0.595	0.142
None	EQ	Yes	0.902	0.926	0.729	0.193	0.660	0.642	0.584	0.174
FS	None	Yes	0.899	0.887	0.697	0.195	0.700	0.670	0.574	0.154
FS	EQ	Yes	0.899	0.919	0.722	0.199	0.640	0.690	0.584	0.179
FE	None	Yes	0.877	0.918	0.724	0.193	0.680	0.750	0.585	0.168
FE	EQ	Yes	0.906	0.926	0.738	0.192	0.640	0.724	0.584	0.186
FE + FS	None	Yes	0.899	0.901	0.712	0.197	0.660	0.765	0.591	0.165
FE + FS	EQ	Yes	0.877	0.901	0.727	0.211	0.660	0.764	0.595	0.200

			After ET							
			Validation				Test			
Online Aug	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
None	None	Yes	0.868	0.913	0.726	0.205	0.817	0.864	0.635	0.132
None	EQ	Yes	0.843	0.868	0.729	0.226	0.833	0.819	0.651	0.161
FS	None	Yes	0.873	0.855	0.720	0.216	0.950	1.000	0.687	0.108
FS	EQ	Yes	0.873	0.796	0.723	0.241	0.900	0.980	0.686	0.137
FE	None	Yes	0.848	0.885	0.730	0.232	0.850	0.936	0.658	0.145
FE	EQ	Yes	0.776	0.800	0.721	0.276	0.833	0.847	0.659	0.190
FE + FS	None	Yes	0.851	0.814	0.710	0.246	0.917	0.954	0.701	0.152
FE + FS	EQ	Yes	0.806	0.822	0.719	0.249	0.867	0.898	0.664	0.166

The support vector machine classifier did not perform well on the testing data, and therefore no results using that model are highlighted, though the performance metrics for the explored configurations are still presented in Appendix A.

4.1.2 Spectrogram-Based Models

The optimal CNN model structure found through trial and error differed between non-sliced and sliced data. Therefore, `AudioClassifierNoSlice` was the final model used for all non-sliced configurations, whereas sliced configurations used `AudioClassifierSlice`, both defined in `pytorch_models.py`, Appendix B.8. The model summaries of both are presented below, followed by Table 4.4 showcasing the performance metrics.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
AudioClassifierNoSlice                   [1, 2]                    65,792
|-Sequential: 1-1                        [1, 256, 2, 40]           --
|    |-Conv2d: 2-1                       [1, 64, 32, 1287]         3,584
|    |-ReLU: 2-2                         [1, 64, 32, 1287]         --
|    |-BatchNorm2d: 2-3                  [1, 64, 32, 1287]         128
|    |-MaxPool2d: 2-4                    [1, 64, 16, 644]          --
|    |-Dropout2d: 2-5                    [1, 64, 16, 644]          --
|    |-Conv2d: 2-6                       [1, 64, 16, 642]          61,504
|    |-ReLU: 2-7                         [1, 64, 16, 642]          --
|    |-BatchNorm2d: 2-8                  [1, 64, 16, 642]          128
|    |-Conv2d: 2-9                       [1, 128, 16, 320]         123,008
|    |-ReLU: 2-10                        [1, 128, 16, 320]         --
|    |-BatchNorm2d: 2-11                 [1, 128, 16, 320]         256
|    |-MaxPool2d: 2-12                   [1, 128, 8, 160]          --
|    |-Dropout2d: 2-13                   [1, 128, 8, 160]          --
|    |-Conv2d: 2-14                      [1, 128, 6, 160]          147,584
|    |-ReLU: 2-15                        [1, 128, 6, 160]          --
|    |-BatchNorm2d: 2-16                 [1, 128, 6, 160]          256
|    |-Conv2d: 2-17                      [1, 256, 4, 80]           295,168
|    |-ReLU: 2-18                        [1, 256, 4, 80]           --
|    |-BatchNorm2d: 2-19                 [1, 256, 4, 80]           512
|    |-MaxPool2d: 2-20                   [1, 256, 2, 40]           --
|    |-Dropout2d: 2-21                   [1, 256, 2, 40]           --
|-AdaptiveAvgPool2d: 1-2                 [1, 256, 1, 1]            --
|-Flatten: 1-3                           [1, 256]                  --
|-Linear: 1-4                            [1, 2]                    514
==========================================================================================
Total params: 698,434
Trainable params: 698,434
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 1.65
==========================================================================================
Input size (MB): 0.66
Forward/backward pass size (MB): 66.45
Params size (MB): 2.53
Estimated Total Size (MB): 69.64
==========================================================================================
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
AudioClassifierSlice                     [1, 2]                    768
|-Sequential: 1-1                        [1, 128, 6, 60]           --
|    |-Conv2d: 2-1                       [1, 16, 62, 504]          1,216
|    |-ReLU: 2-2                         [1, 16, 62, 504]          --
|    |-BatchNorm2d: 2-3                  [1, 16, 62, 504]          32
|    |-MaxPool2d: 2-4                    [1, 16, 30, 251]          --
|    |-Dropout2d: 2-5                    [1, 16, 30, 251]          --
|    |-Conv2d: 2-6                       [1, 64, 30, 247]          21,568
|    |-ReLU: 2-7                         [1, 64, 30, 247]          --
|    |-BatchNorm2d: 2-8                  [1, 64, 30, 247]          128
|    |-MaxPool2d: 2-9                    [1, 64, 14, 123]          --
|    |-Dropout2d: 2-10                   [1, 64, 14, 123]          --
|    |-Conv2d: 2-11                      [1, 128, 14, 121]         123,008
|    |-ReLU: 2-12                        [1, 128, 14, 121]         --
|    |-Conv2d: 2-13                      [1, 256, 14, 121]         295,168
|    |-ReLU: 2-14                        [1, 256, 14, 121]         --
|    |-Conv2d: 2-15                      [1, 128, 14, 121]         295,040
|    |-ReLU: 2-16                        [1, 128, 14, 121]         --
|    |-BatchNorm2d: 2-17                 [1, 128, 14, 121]         256
|    |-MaxPool2d: 2-18                   [1, 128, 6, 60]           --
|    |-Dropout2d: 2-19                   [1, 128, 6, 60]           --
|-AdaptiveAvgPool2d: 1-2                 [1, 128, 1, 1]            --
|-Flatten: 1-3                           [1, 128]                  --
```

```
|-Linear: 1-4                            [1, 128]                  16,512
|-Sequential: 1-5                        --                        (recursive)
|    |-ReLU: 2-20                        [1, 128]                  --
|-Linear: 1-6                            [1, 2]                    258
==========================================================================================
Total params: 753,954
Trainable params: 753,954
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 1.41
==========================================================================================
Input size (MB): 0.13
Forward/backward pass size (MB): 24.26
Params size (MB): 3.01
Estimated Total Size (MB): 27.41
==========================================================================================
```

Table 4.4: Performance metrics of the spectrogram-based CNNs trained on different configurations of data. Some highlight statistics are written in bold font.

Spectrogram-Based CNN			Before ET							
			Validation				Test			
Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	None	No	0.838	0.814	0.753	0.303	0.642	0.770	0.588	0.365
		Yes	**0.886**	**0.817**	**0.792**	**0.264**	**0.833**	0.754	**0.785**	0.344
	EQ	No	0.796	0.800	0.722	0.323	0.650	**0.906**	0.640	0.383
		Yes	**0.866**	0.776	0.719	0.303	**0.817**	**0.866**	**0.786**	**0.313**
	PSTS	No	0.771	0.745	0.700	0.344	0.600	0.842	0.600	0.390
		Yes	0.838	0.800	0.717	0.304	0.717	0.818	0.734	0.367
Yes	None	No	0.803	0.883	0.792	0.251	0.578	0.839	0.572	0.347
		Yes	0.820	0.890	0.802	0.247	0.635	0.796	0.612	0.306
	EQ	No	0.836	0.903	0.830	0.239	0.553	0.806	0.557	0.343
		Yes	0.814	0.879	0.812	0.219	0.658	0.756	0.647	0.344

			After ET							
			Validation				Test			
Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	None	No	**1.000**	**1.000**	**0.955**	**0.075**	**0.980**	**1.000**	0.911	0.133
		Yes	0.924	0.937	0.869	0.245	0.840	0.987	0.849	0.260
	EQ	No	**1.000**	**1.000**	**0.931**	0.131	**1.000**	**1.000**	**0.945**	**0.079**
		Yes	0.924	0.940	0.839	0.250	0.880	0.995	0.856	0.248
	PSTS	No	**1.000**	**1.000**	**0.937**	0.112	0.960	0.990	0.918	0.152
		Yes	0.906	0.926	0.843	0.257	0.940	0.995	0.865	0.194
Yes	None	No	0.916	0.938	0.894	0.158	0.731	0.801	0.713	0.198
		Yes	0.918	0,935	0.896	0,156	0.688	0.762	0.681	0.219
	EQ	No	0.889	0.937	0.879	0.195	0.638	0.751	0.634	0.218
		Yes	0.897	0.923	0.881	0.176	0.719	0.821	0.718	0.226

Finally, to set up for discussion in the following chapter, a couple example loss and accuracy graphs from the training of two of the most promising configurations before and after ET, (**EQ** with filter and **EQ** without filter respectively, both non-sliced) are shown in Figure 4.2. On the same note, Figure 4.3 displays the confusion matrices from the same configuration before ET, for both validation and test data. Also, to add to the validation results: In all scenarios where the model predicted a file incorrectly, the corresponding parallel file recorded on the other device, were predicted correctly with high probability.

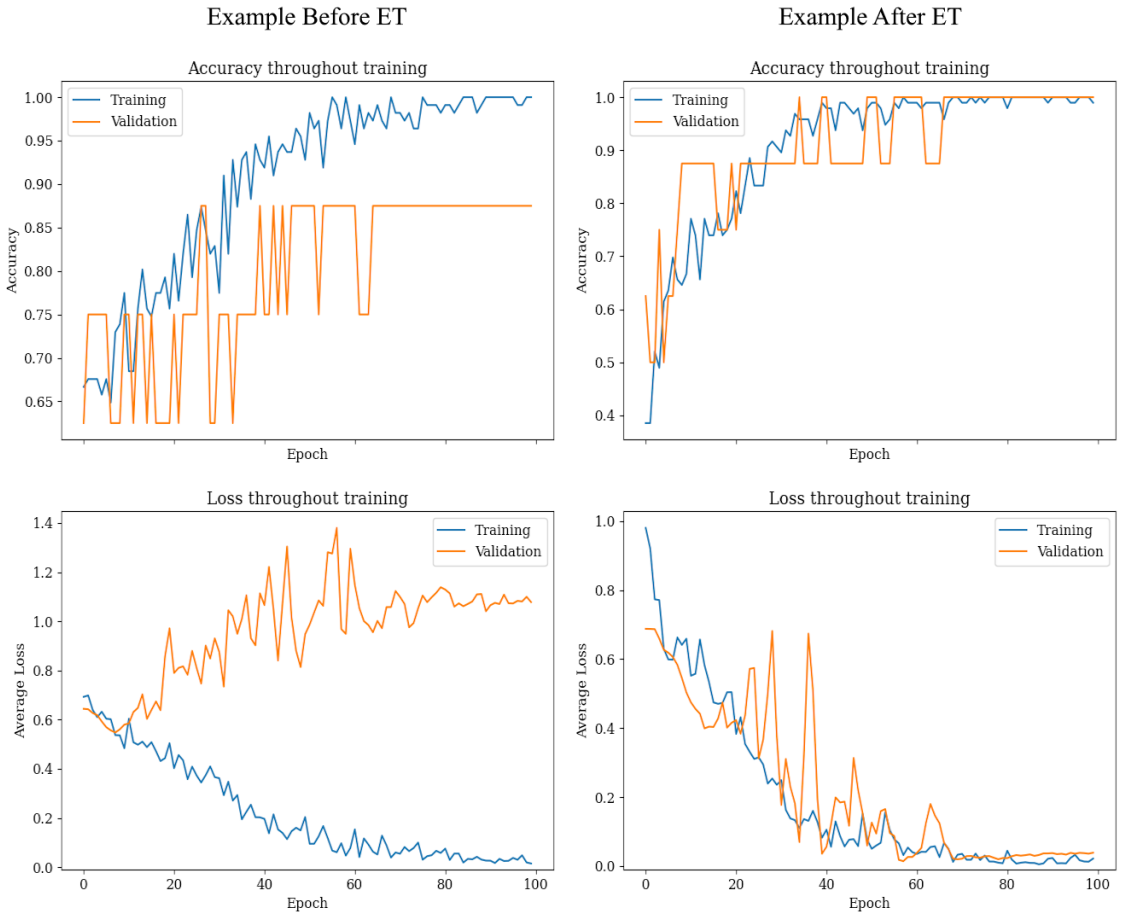

Figure 4.2: Two example training graphs, one each from one of the most promising configurations before and after ET, displaying the loss and accuracy for the training and validation set.

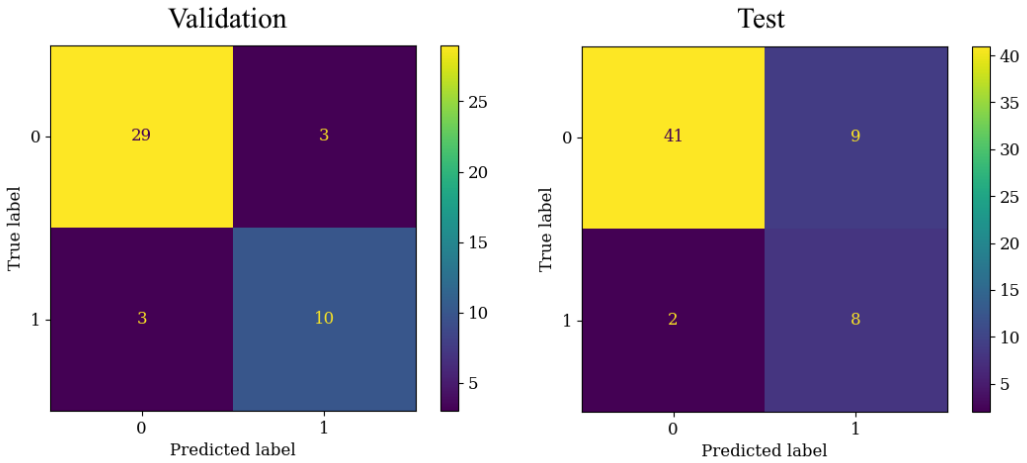

Figure 4.3: Two confusion matrices from inference on both the validation and test data from the most promising configurations before ET.

5
Discussion

In this chapter, the results are analysed and discussed, with the intention of finding the best model setup suitable for the problem of classifying BOAS. Below follows an assessment of the data gathering process and results, while the machine learning models are discussed later in Section 5.1. Some possible implementations are then covered, before possible errors are discussed, in Sections 5.2 and 5.3.

The data used in this project had a class imbalance of roughly two-thirds BOAS negative and one-third BOAS positive. This is not a massive difference, though it may still have left an impact on the final models, in that they might perform better predicting BOAS negative data compared to BOAS positive. Systems were in place to try to prevent this, including using the F1- and ROC-AUC-scores during training to select the best model, but that is unlikely to fix the problem entirely. Another way of handling class imbalance is to multiply the gradient during training by a class imbalance-factor based on which class the data has, to give that class more weight. Data augmentation could also be used here to generate more new BOAS positive data than BOAS negative, until they are equal in number. Of course, the optimal solution would be to gather more data of the minority class, though that is not entirely viable in this case, due to the data already being limited.

The newly recorded data of five dogs was a substantial addition to the dataset. Despite being few in number, the parallel recordings of different phones highlighted a problem that was not present in the previously available data, but that will likely be significant in the real-world use case; the problem being the vastly different frequency response of different devices, as could be observed in Figure 4.1. There are three ways to combat this issue:

1. **Normalize the signal by the device's frequency response**. This is achievable in a controlled setting, where all devices are known. It could be done by recording an invariable sound profile that covers all frequencies in an isolated environment, to use as the normalization function. However, for this project where the goal is to eventually distribute the final product to veterinarians and dog owners, on any perceivable Android-device in a user-friendly implementation, this is not a viable solution.
2. **Gather data using many different devices**. This is achievable and even expected if more BOAS-specialized veterinarians come on board to help gather audio data on BOAS-affected dogs. Though, since the main limitation is the lack of data, it can in the current state of the project not be the sole solution.

3. **Augment the available data by altering the frequency response**. This is achievable in most cases, as compared to gathering real new data, post-processing has no intrinsic resource-limitation. It was implemented in this project through the equalization curve augmentation.

By having the new data at hand during the inference of the models, the augmentation solution that was employed could be evaluated. This was especially useful since the previous data was more similar in that regard.

Furthermore, from the standpoint of the project, it is unfortunate that three out of the five dogs were of BOAS grade 1, while only one dog was classified as BOAS positive, with no BOAS grade 3. This distribution further emphasizes the class discrepancy that is present in the dataset, which for the sake of training the models should preferably be evened out. It also means that there is still only one dog in the entire expanded dataset that had a grade 3 diagnosis, which is bad news for the possibility of classifying all four BOAS grades, rather than negative versus positive.

Another aspect of using different devices, and having different individuals capture the audio data (handling the dog and phone differently), is that the loudness of the files can vary. The general solutions to this are the same three as for the frequency response, although the normalization is now achievable since it is not necessarily device specific. For one, the problem is not present in the spectrogram-based models since the data is normalized to zero mean and unit variance. In the feature extraction-based models, however, the same normalization proved unsuccessful in my attempts and was therefore swapped for amplitude augmentation. It is possible that a different normalization technique could work in this case, but that was not explored further in this thesis.

5.1 Discussing the Models

The models and their configurations are compared to one another based on the performance metrics presented in Tables 4.2, 4.3 and 4.4. The validation data was gathered in consistent and controlled fashion, guided- or directly by a BOAS-specialized veterinarian, while the test data recordings were taken by Jennie- who is not a veterinarian, and crucially, a different person. The test data is also more noisy (veterinary clinic noises), and since it comes from devices different from the validation data, it is more-so comparable with the intended real world use case: A possibly noisy clinic/home with a phone-model that does not match the training data. For this reason, the performance metrics of the test data is valued higher than those of the validation data, though both are of course considered.

In the discussions of the results from the different models in Sections 5.1.1 and 5.1.2, the metrics will be referred to by their shortened name in the tables (Acc, ROC, Prob and SD) for the sake of readability. The specifics of those names were described in the Results Chapter (Section 4.1).

5.1.1 Feature Extraction-Based Models

The logistic regression model in general appears to work well on data that is very similar to the training data, i.e. the validation data, and mostly so for the non-augmented, non-filtered signals. For that configuration both before and after ET, the Acc, ROC and Prob are all close to 1, with a reasonably small SD. This, however, is not true for the test data that deviates from the original data in many ways, some previously discussed. The Acc and Prob are around 0.7 and the standard deviation is large at $\sim 0.3-0.45$, meaning the incorrect guesses are still highly confident. Before ET, the ROC is also around 0.7, but after ET it is higher in most cases, around 0.9 and as high as 0.994 for the non-augmented but filtered configuration. In fact, applying the filter appears to slightly increase the test performance in almost every scenario after ET across the highlighted results, though not in general. All in all, the logistic regressions results are unsatisfactory, despite its strong results for the validation data.

The random forest classifier performs noticeably worse on the validation data than logistic regression, though it is still a decent result with Acc and ROC around 0.9, Prob around 0.7 and a SD of ~ 0.2. The results after ET are slightly worse still on the validation data. At first, the metrics on the testing data before ET appears similar or poorer too, but crucially, in contrast to the logistic regression here, the SD is still rather small. This corresponds to the model being less confident in the incorrect guesses which is a desirable characteristic, though it may also mean that the correct guesses are of lower confidence as well. Moving on to the test data after ET, however, we see a contrasting result where the metrics are comparable to, and in some ways better than, those on the validation data. Another observation is that the filter in almost all cases outperformed the equivalent configuration without filter. The random forest classifier appears to be a better fit for the problem than logistic regression, though it is still far from perfect.

For both logistic regression and random forest, the use of sliced data or the PSTS augmentations proved unfruitful, even destructive, as is hinted by their absence in the highlighted results. The use of an equalization curve did also not perform better than without the augmentation, but in contrast to the former two configurations, it did not significantly reduce the performance of the models.

The standout configuration, among both models, is random forest- without *offline* augmentation- with *online* augmentations only for the feature selection- with the filter applied, especially after ET.

5.1.2 Spectrogram-Based Models

The final CNN models used batch normalization and Kaiming normal initialization (an initialization method to ensure stable and efficient training [45]) for the weights, in a total of five convolutional blocks, with max pooling for the pooling layers. The kernel sizes, strides and paddings of the convolutional layers decreased with the depth, and dropout was added after each pooling layer. The rectified linear unit

was used as the activation function, and an adaptive average pooling layer was implemented before the linear output block. Removing some pooling layers appeared to work to increase the performance of the models, though the effect in each specific position is unclear.

The CNN model for non-sliced data employed a *conv–pool–conv–conv–pool–conv–conv–pool* structure, increasing the number of channels in each convolutional block that precedes a pooling layer, meaning a total of three times: $1 \rightarrow 64 \rightarrow 128 \rightarrow 256$. Meanwhile, the size of the image was reduced from 2579×64 pixels to 40×2 pixels. The dropout layers were all 20%, and the final linear block consisted of a single linear classifier.

For the sliced data, the model instead had a *conv–pool–conv–pool–conv–conv–conv–pool* structure, where the channels were changed in every convolutional block: $1 \rightarrow 16 \rightarrow 64 \rightarrow 128 \rightarrow 256 \rightarrow 128$, while the size of the image went from 516×64 pixels to 60×6 pixels. Why this worked better compared to the previous structure on sliced data is unclear. Perhaps it is partially because fewer features are found in the shorter audio signal, so using less output channels is beneficial. The dropout layers were all 10%, as the original 20% disrupted training too much, the reason for which is uncertain. To speculate, it may have something to do with the fact that dropout is used in back-to-back convolutional blocks, propagating the error that emerges from one dropout through the other. The linear block in this case seemed to work better with two linear classifiers.

In general, the reason why a certain parameter or element in the model performed better than another, or why one structure worked better for sliced data and another for non-sliced data, is difficult for me to explain. The model is like a black box, where you give an input and get an output without seeing what occurs "behind the curtain". In the end, it is all mathematics, but even though you can understand the specifics of what a certain function does in theory, predicting how a change will propagate through 700 000 parameters is non-trivial. It should at the same time be stated that though many configurations were tested, the best model structure will still not have been found. This is largely because training takes significantly longer time in this approach, compared to the feature extraction-based one, and because there are near infinite combinations of parameters. Also, the training is not consistent: different runs will reach different end results due to the stochastic nature of the model initialization and the training process. In a way, this was more about finding the most appropriate approach to the machine learning problem, and finding a decently working model was a part of that.

Moving on to the performance metrics, it can be concluded that the CNNs trained on the sliced data did not perform as well as those trained on non-sliced data. The PSTS augmentations also did not prove useful, though the impact was less negative as for those models. For the data before ET, the use of the adaptive high-pass filter was beneficial in all otherwise relevant (from a performance perspective) configurations, though the opposite is true after ET. This contrasting result can likely be

explained by the difference in features and corresponding frequency that the models "look" at in the respective configuration and would in that case hint at lower frequency components being more important after ET. Also worth mentioning is that the SD is large before ET, which again means overconfident even when correct. The best model configurations after ET, however, have very low SD and a very high Prob, along with perfect Acc and ROC.

Finally, the EQ augmentation gave an interesting yet predictable result; namely that its use improves the test data performance but worsens the inference metrics of the validation data. This was true for before and after ET, though the difference is modest in both cases. It is predictable since the test data uses different devices from the original training data, which is what the augmentation tries to mimic and thereby force the model to generalize over. Meanwhile, the validation data is similar to the training data without transformations, and the augmentation removes some of that similarity.

There is still more to discuss regarding the CNNs results, now turning to Figures 4.2 and 4.3. In the former, the example before ET shows one situation that occurred somewhat frequently when training models on data from before ET, where the validation loss increases rather than decreases. At first this seem to suggest that the model is learning the wrong features, but at the same time, the validation accuracy increases. The explanation is likely partly that the approach is not well suited for the largely featureless input signal, since the dogs are at rest, and partly that the validation set is not aligning well with the training set- that in this case one audio signal is too different from all the training data. A solution to the first-mentioned cause could be to find a better, more optimised model, perhaps using a separate CNN structure for recording before ET, or another approach altogether. A solution to the second-mentioned cause is to gather more data.

The example after ET instead shows a positive result that was rather common for this type of data, though not always to this extent. The validation loss decreases along with the training loss, and the accuracies both reach 100%. This means that the features that the models "learn" from the training data align well with the validation data, and evidently the test data too. After reaching a model result like this, not much more can be improved until there is more data available, or rather: Any true improvement would just yield similar results to the current setup and therefore not appear to improve the model.

The confusion matrices in Figure 4.3 do not reveal much by themselves, other than the fact that for the validation data, the false negative rate is lower than the false positive rate. The test matrix becomes more interesting with the fact that for *all* files that were predicted incorrectly, the parallel file was *always* predicted correctly with high confidence. This suggests that the EQ augmentation is not optimized yet for this data, since the different devices give different predictions on the same original sounds. It is also not entirely consistent with which device corresponds with the correct guess.

5.1.3 Comparing and Summarizing the Two Approaches

To summarize, the best spectrogram-based CNN model configurations found for before ET was non-sliced, with EQ augmentations and the adaptive high-pass filter. The same configuration, but without the filter was the best for after ET. The runner up is the same respective configurations without any *offline* augmentations, since the validation performance is slightly higher and with still competitive test scores. However, as previously mentioned, the test data inference is valued higher.

One big advantage of the feature extraction-based model is the speed of training; in the order of a minute for feature extraction and mere seconds for model training, which enables faster testing and iterations of different models and configurations. With the spectrogram-based CNNs this process takes significantly longer, in the order of an hour per fold, which limits the search for the best model setup. Other than that, suggested by the results obtained in this thesis, the spectrogram-based models outperform the feature extraction approach both before and especially after ET.

I should mention that Jennie found better results with the feature extraction approach, meaning more testing is required before one method can be ruled out completely. Check out her thesis [9] for the details on her method and results.

Slicing the data was not found to have any real advantage over using the full-length signals for any model and instead performed worse. However, it is not enough evidence to completely rule out slicing for future use. Perhaps, the data could be selectively sliced to only consist of audio with sounds that are characteristic for the BOAS grade or even use the slices to forge new 30 s audio clips to expand the dataset. Other slicing settings, such as length and overlap, could also be explored. Pitch scaling and time stretching did not yield any performance increase and can likely be abandoned, at least for the approaches covered in this thesis.

The equalization curve augmentation yielded positive results when used in the otherwise best model configurations, though it might not be necessary once enough data with different devices has been gathered. As suggested by the last paragraph in Section 4.1.2, the equalization curve itself could be improved, especially to work better before ET. The adaptive high-pass filter showed promising results for before ET, and can therefore with benefit be studied further, perhaps trying different thresholds or filter characteristics.

5.2 Real World Implementation

There are a few directions one could take when organizing a real-world implementation of the tool that would result from this thesis work. The first decision to take is whether to train a model on all data, without any validation results, or to use the five models from cross-validation and get an average prediction. The benefit of the former is simplicity, less storage required, and more data used for training.

However, since the performance cannot be validated, I would lean towards the latter alternative, in which case another decision must be made: Should each model prediction be taken as a binary vote, or should its confidence be included, to average over. To me it sounds wiser to incorporate the confidence, since it carries more info of the actual evaluation, though this would need to be tested.

After that, the dynamic between before and after ET must be decided. The simple solution is to keep them separated, and give one diagnosis before ET, and another after ET. A more advanced solution, that in my opinion sounds better and more complete, would be to take both into account, and give a probability based off the combined prediction. If the two align, then the chance is high that the diagnosis is correct (according to the testing results), but if they oppose one another, that should be communicated to the user through a low probability. In such a scenario, I still think the grade from after ET should be the final binary classification, since it performed significantly better for the available data than before ET, but with a message of caution, implying that the certainty is lower.

In a perfect world, an analysis after ET would not be needed, since it demands more of both the dog and the person carrying out the test. However, at the current state of the models, only listening to a dog at rest is not sufficient to make a confident prediction. Also, the exercise test is a key part of a diagnosis, since the physical stress enhances the BOAS symptoms, and could even bring out symptoms that are dormant at rest. Therefore, for a confident prediction of BOAS, such as when used to assist a veterinarian in making an informed decision, both before and after ET recordings should be taken into account.

It is also important to consider the use-cases. The natural first target audience is veterinarians, to assist during BOAS examinations, for which a complete test result with details regarding the prediction could be provided. This is also where an expansion to all four RFG-S BOAS grades would be useful. The second use-case is for dog-owners to download the app to their smartphone and use it to get an *indication* of whether their dog should get a more extensive examination by a licensed veterinarian. This should never be used as proof that the dog is healthy, since it could be manipulated or done incorrectly, so the app could simply recommend appropriate action to the user for their dog, based on the prediction.

5.3 Possible Errors

The biggest source of error is the data scarcity. It makes the results less trustworthy since the inference is done on only $7-12$ files, from $5-8$ dogs, which is simply not enough to make convincing claims about the performance. Limited data to train on also suppresses the models' from reaching their performance potential. On top of that that, it means that the possibility to find the optimal model parameters is limited after predicting all correctly, as was the case for after ET spectrogram-based CNN.

Another possible future error if more data is gathered with different veterinarians, is the fact that the BOAS diagnoses may be slightly biased. The point here is that for a dog that is very close between a BOAS grade 1 and BOAS grade 2, different veterinarians may lean to one side while others could lean to the other. Especially so depending on the day or environment, as the dogs' status, such as how it is feeling, may affect its breathing and subsequently the diagnosis. The effective result is contradictory input data, which could introduce problems for the models. The optimal prediction in such a case would be one with a 50% confidence, though the situation should ultimately be avoided.

6
Conclusion

To conclude, I have shown in this thesis that a spectrogram-based convolutional neural network can correctly classify whether a dog is BOAS positive, based on audio data recorded on Android mobile phones. For dogs recorded after a short exercise test, the best performing model got a perfect score on completely unseen data. However, for recordings of dogs at rest it performed worse, with an accuracy around 80%. I also explored the use of simpler models with a feature extraction approach, but in my results the spectrogram-based CNNs outperformed them across the board.

Despite performing perfectly after exercise, and decently at rest, because of data scarcity, the models' real-world performance could not be extensively tested. The limited access to audio recordings of BOAS-graded dogs meant that the completely unseen testing data consisted of only five dogs. The priority for continuing this work should therefore be to gather more data, not only for being able to evaluate the performance more accurately, but also to train on more data to further improve the models.

I also highlighted a problem with using audio data from arbitrary Android phones, namely that different recording devices can have different frequency responses. I tried to combat this with an augmentation that changes the frequency response randomly during training, to get the models to generalize over this varying feature. This method proved somewhat successful, though the data from dogs at rest showed that the difference is still problematic. I believe the augmentation can be optimized to further improve the models. Gathering more data can also alleviate the need for such an augmentation, if new recordings are carried out with a variety of devices.

6.1 Future work

More data needs to be gathered to be able to confirm that the models are sufficiently accurate, and to further increase their performance. I believe that only at that point should improvements of the model for audio after an exercise test be pursued. For audio of dogs at rest, more could be done before data has been gathered, mainly by improving the frequency response augmentation to better match the effect of using different devices to record. This alone could significantly improve that model and would carry over to that for after exercise as well, though it would not be possible to validate without more data.

While the sliced data did not yield improved model performance, I think it would be interesting to further explore this idea by using the slices to forge new, longer audio signals. That could artificially increase the size of the dataset substantially, while sticking with the 30-second format that worked best.

The filter should be implemented to be applied offline in the future, to drastically speed up training for the related configurations. This would have no effect on the models' predictive performance either, since the filter is not randomized in any way.

Once enough data is available of each class, the models can be expanded to classify all four BOAS grades, rather than just positive and negative. This is not a priority, but it is a natural next step for the project to take.

Of course, what can also be done before more data is gathered is to develop the Android application. I believe the models are at a stage where a combined prediction of audio recorded before and after an exercise test would be sufficiently accurate for a demo-version of the app.

Bibliography

[1] V. Morell, "From Wolf to Dog," *Scientific American*, vol. 313, no. 1, pp. 60–67, Jun. 2015. [Online]. Available: https://doi.org/10.1038/scientificamerican0715-60

[2] Python Software Foundation, *Python Language Reference*, Version 3.12. [Online]. Available: http://www.python.org

[3] Svenska Kennel Klubben, "RFG-Scheme," *Svenska Kennel Klubben*, 2025. https://www.skk.se/uppfodning/halsa/andning/rfg-scheme/ (accessed May 11, 2025).

[4] N.-C. Liu *et al.*, "Conformational Risk Factors of Brachycephalic Obstructive Airway Syndrome (BOAS) in Pugs, French Bulldogs, and Bulldogs," *PLOS ONE*, vol. 12, no. 8, p. e0181928, Aug. 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0181928

[5] Jordbruksverket, "Statistik ur Hundregistret," *Jordbruksverket.se*, Dec. 31, 2024. [Online]. Available: https://jordbruksverket.se/e-tjanster-databaser-och-appar/e-tjanster-och-databaser-djur/hundregistret/statistik-ur-hundregistret (accessed Jan. 29, 2025)

[6] Svenska Kennel Klubben, "RFG blir krav vid avel med mops, fransk- och engelsk bulldogg," *Svenska Kennel Klubben*, 2024. https://www.skk.se/nyheter/2024/06/rfg-blir-krav-vid-avel-med-mops-fransk--och-engelsk-bulldogg/ (accessed May 11, 2025).

[7] M. Mårtensson, "Brachycephalic Obstructive Airway Syndrome (BOAS) Classification in Dogs Based on Respiratory Noise Analysis Using Machine Learning," M.S. thesis, Chalmers Univ. Technol., 2021. [Online]. Available: https://odr.chalmers.se/items/57dd08ef-b832-432e-8f10-336a429de644

[8] H. Pettersson and O. Stensöta, "Data Augmentation for Audio-Based Machine Learning Classifying Brachycephalic Obstructive Airway Syndrome (BOAS) in Dogs," M.S. thesis, Chalmers Univ. Technol., 2021. [Online]. Available: https://odr.chalmers.se/items/fc11f637-e6cb-4750-ba73-9b102e89560f

[9] J. Berndtson, "Acoustic Signal Analysis and Feature-Based Classification of BOAS: For the Health and Welfare of Brachycephalic Dogs," *Chalmers University of Technology*, 2025. [Online]. Available: https://odr.chalmers.se/handle/20.500.12380/1

[10] A. Ashfaq, N. Cronin, and P. Müller, "Recent Advances in Machine Learning for Maximal Oxygen Uptake (VO2 Max) Prediction: A Review," *Informatics in Medicine Unlocked*, vol. 28, p. 100863, Jan. 2022. [Online]. Available: https://doi.org/10.1016/j.imu.2022.100863

[11] MathWorks, "Feature Extraction Explained," *MathWorks.com*, 2025. [Online]. Available: https://mathworks.com/discovery/feature-extraction (accessed Mar. 14, 2025)

[12] scikit-learn developers, "1.1. Linear Models," *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/linear_model (accessed May. 29, 2025)

[13] scikit-learn developers, "RandomForestClassifier," *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier (accessed May. 29, 2025)

[14] scikit-learn developers, "1.10. Decision trees," *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/tree (accessed May. 29, 2025)

[15] R. Singh, "Support Vector Machines (SVM)," *Medium*, Oct. 17, 2024. [Online]. https://medium.com/@RobuRishabh/support-vector-machines-svm-27cd45b74fbb

[16] L. Roberts, "Understanding the Mel Spectrogram," *Medium*, Mar. 14, 2020. [Online]. Available: https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53

[17] T. van der Werff, "CNN vs. Vision Transformer: A Practitioner's Guide to Selecting the Right Model," *Tobias van der Werff*, May 15, 2024. [Online]. Available: https://tobiasvanderwerff.com/2024/05/15/cnn-vs-vit (accessed Mar. 17, 2025)

[18] S. Ramesh, "A Guide to an Efficient Way to Build Neural Network Architectures – Part II: Hyper-Parameter Selection and Tuning for Convolutional Neural Networks Using Hyperas on Fashion-MNIST," *Medium*, May 7, 2018. [Online]. Available: https://medium.com/towards-data-science/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7 (accessed Mar. 13, 2025)

[19] scikit-learn developers, "3.1. Cross-Validation: Evaluating Estimator Performance," *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/cross_validation (accessed Mar. 20, 2025)

[20] scikit-learn developers, "3.4. Metrics and Scoring: Quantifying the Quality of Predictions," *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation#classification-metrics (accessed Mar. 20, 2025)

[21] Google, "Classification: Accuracy, Recall, Precision, and Related Metrics," *Google for Developers*, 2024. [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall

[22] M. P. Muller *et al.*, "Can Routine Laboratory Tests Discriminate between Severe Acute Respiratory Syndrome and Other Causes of Community-Acquired Pneumonia?," *Clinical Infectious Diseases*, vol. 40, no. 8, pp. 1079–1086, Apr. 2005. [Online]. Available: https://doi.org/10.1086/428577

[23] B. Mehlig, "Preprocessing the Input Data," in *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge Univ. Press, 2021.

[24] V. Velardo, "Audio Data Augmentation Is All You Need," *YouTube*, Jan. 10, 2022. [Online]. Available: https://www.youtube.com/watch?v=HH_h52I_Qeg&list=PL-wATfeyAMNoR4aqS-Fv0GRmS6bx5RtTW (accessed Mar. 8, 2025)

[25] V. Velardo, "Audio Data Augmentation Techniques: The Theory," *YouTube*, Jan. 13, 2022. [Online]. Available: https://www.youtube.com/watch?v=bm1cQfb_pLA&ab_channel=ValerioVelardo-TheSoundofAI (accessed Mar. 8, 2025)

[26] V. Hromadova, P. Kasak, R. Jarina, and P. Brida, "Frequency Response of Smartphones at the Upper Limit of the Audible Range," in *Proc. 2022 ELEKTRO*, May 2022. [Online]. Available: https://doi.org/10.1109/elektro53996.2022.9803475

[27] D. S. Park *et al.*, "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition," in *Proc. Interspeech*, pp. 2613–2617, Sep. 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1904.08779

[28] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. NeurIPS*, 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[29] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[30] J. Hwang *et al.*, "TorchAudio 2.1: Advancing Speech Recognition, Self-Supervised Learning, and Audio Processing Components for PyTorch," *arXiv*, Jan. 2023. [Online]. Available: https://doi.org/10.48550/arxiv.2310.17864

[31] B. McFee *et al.*, *librosa/librosa: 0.10.2.post1*. Zenodo, May 2024. [Online]. Available: https://doi.org/10.5281/zenodo.11192913

[32] F. Eyben, M. Wöllmer, and B. Schuller, "openSMILE – The Munich Versatile and Fast Open-Source Audio Feature Extractor," in *Proc. Int. Conf. on Multimedia (MM)*, ACM, Florence, Italy, pp. 1459–1462, Oct. 2010. [Online]. Available: https://doi.org/10.1145/1873951.1874246

[33] ONNX Community, *skl2onnx: Convert scikit-learn Models to ONNX*, Version 1.18.0, 2024. [Online]. Available: https://github.com/onnx/sklearn-onnx

[34] ONNX Community, *ONNX: Open Neural Network Exchange*, Version 1.17.0, 2024. [Online]. Available: https://github.com/onnx/onnx

[35] ONNX Runtime developers, *ONNX Runtime*, Version 1.20.1, 2024. [Online]. Available: https://onnxruntime.ai/

[36] The pandas development team, *pandas-dev/pandas-v2.2.3: Pandas*. Zenodo, 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[37] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proc. 9th Python in Science Conf.*, pp. 56–61, 2010. [Online]. Available: https://doi.org/10.25080/majora-92bf1922-00a

[38] Virtanen, P., et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," in *Nature Methods*, vol. 17, pp. 261–272, 2020. Available: https://doi.org/10.1038/s41592-019-0686-2

[39] M. Kelechava, "Urban Sound Classification with Librosa — Tricky Cross-Validation," *Medium*, Aug. 6, 2020. [Online]. Available: https://medium.c

om/data-science/urban-sound-classification-with-librosa-nuanced-cross-validation-5b5eb3d9ee30 (accessed Feb. 4, 2025)

[40] J. Salamon, C. Jacoby, and J. P. Bello, "A Dataset and Taxonomy for Urban Sound Research," in *Proc. ACM Int. Conf. on Multimedia (MM)*, 2014. [Online]. Available: https://doi.org/10.1145/2647868.2655045

[41] Wikipedia Contributors, "Nyquist–Shannon sampling theorem," *Wikipedia*, Apr. 03, 2025. Available: https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem (accessed May 22, 2025).

[42] K. Doshi, "Audio Deep Learning Made Simple: Sound Classification, Step-by-Step," *Towards Data Science*, Mar. 18, 2021. [Online]. Available: https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5/ (accessed Jan. 21, 2025)

[43] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," *arXiv preprint arXiv:2201.03545*, 2022. [Online]. Available: https://arxiv.org/abs/2201.03545

[44] J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, USA, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

[45] K. He, X. Zhang, S. Ren, and J. Sun, ´´Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *arXiv.org*, 2015. Available: https://arxiv.org/abs/1502.01852

A

Supplementary Results

In this Appendix, the complete results of the feature extraction-based models are presented in tables. Together these tables cover all considered configurations of the few Scikit-learn models that were deemed to work the best on the original data (not sliced, no augmentations, no filters). Tables A.1 and A.2 show the results of the Logistic Regression classifier model, Tables A.3 and A.4 show the results of the Random Forest Classifier model, and Tables A.3 and A.4 show the results of the Support Vector Classifier model.

The format of the tables is similar to those presented in the Results Chapter, with a couple differences:

- The 'Online Aug' column has been replaced with 'FS Aug', which now only represents if the model used online augmentation during Feature Selection or not. The online augmentation during Feature Extraction is now given per table and is specified in the table header and caption, since in order to fit the tables on one page each they needed to be split up.
- Compared to the highlighted results tables of the feature extraction models, these tables include a 'Sliced' column. This column was omitted in the main Results Chapter since none of the highlighted results used sliced data.

Table A.1: Logistic regression, *without* online augmentation for feature extraction.

Logistic Regression Without Online Aug for FE				Before ET							
				Validation				Test			
FS Aug	Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	No	None	No	0.978	0.964	0.913	0.181	0.660	0.722	0.607	0.333
			Yes	0.849	0.918	0.843	0.244	0.400	0.478	0.412	0.415
		EQ	No	0.853	0.923	0.872	0.273	0.620	0.733	0.630	0.408
			Yes	0.878	0.918	0.848	0.253	0.460	0.607	0.471	0.375
		PSTS	No	0.485	0.481	0.475	0.499	0.540	0.450	0.540	0.498
			Yes	0.487	0.418	0.475	0.499	0.680	0.462	0.680	0.466
	Yes	None	No	0.835	0.895	0.828	0.210	0.386	0.186	0.393	0.188
			Yes	0.770	0.846	0.753	0.193	0.425	0.445	0.433	0.191
		EQ	No	0.831	0.879	0.818	0.223	0.425	0.322	0.429	0.162
			Yes	0.765	0.837	0.757	0.187	0.402	0.402	0.405	0.197
Yes	No	None	No	0.928	0.929	0.884	0.224	0.660	0.688	0.616	0.356
			Yes	0.878	0.896	0.821	0.274	0.420	0.440	0.394	0.345
		EQ	No	0.953	0.945	0.909	0.201	0.720	0.753	0.699	0.365
			Yes	0.853	0.863	0.813	0.310	0.480	0.598	0.471	0.384
		PSTS	No	0.394	0.365	0.398	0.488	0.440	0.421	0.440	0.496
			Yes	0.473	0.448	0.475	0.499	0.260	0.463	0.260	0.439
	Yes	None	No	0.832	0.871	0.814	0.222	0.407	0.244	0.427	0.217
			Yes	0.757	0.845	0.753	0.207	0.406	0.377	0.418	0.190
		EQ	No	0.819	0.862	0.814	0.228	0.398	0.249	0.412	0.214
			Yes	0.769	0.835	0.761	0.215	0.416	0.319	0.426	0.208

				After ET							
				Validation				Test			
FS Aug	Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	No	None	No	0.980	1.000	0.942	0.123	0.650	0.812	0.703	0.337
			Yes	0.935	0.998	0.901	0.181	0.767	0.994	0.710	0.294
		EQ	No	0.975	0.993	0.931	0.155	0.683	0.916	0.712	0.359
			Yes	0.865	0.964	0.876	0.256	0.717	0.950	0.708	0.370
		PSTS	No	0.713	0.500	0.711	0.453	0.833	0.500	0.833	0.373
			Yes	0.287	0.500	0.289	0.453	0.183	0.510	0.183	0.387
	Yes	None	No	0.805	0.839	0.786	0.241	0.529	0.007	0.528	0.322
			Yes	0.655	0.778	0.657	0.276	0.314	0.010	0.325	0.219
		EQ	No	0.811	0.845	0.798	0.237	0.523	0.008	0.518	0.325
			Yes	0.677	0.781	0.673	0.270	0.360	0.011	0.372	0.238
Yes	No	None	No	0.890	0.954	0.851	0.254	0.600	0.842	0.628	0.381
			Yes	0.868	0.940	0.838	0.256	0.483	0.228	0.500	0.420
		EQ	No	0.890	0.954	0.859	0.256	0.600	0.854	0.616	0.434
			Yes	0.870	0.942	0.844	0.265	0.533	0.274	0.530	0.423
		PSTS	No	0.391	0.389	0.381	0.478	0.567	0.453	0.567	0.496
			Yes	0.663	0.492	0.667	0.471	0.883	0.730	0.883	0.321
	Yes	None	No	0.813	0.856	0.797	0.239	0.551	0.010	0.540	0.331
			Yes	0.654	0.776	0.653	0.272	0.331	0.008	0.339	0.232
		EQ	No	0.823	0.850	0.801	0.233	0.527	0.013	0.523	0.324
			Yes	0.672	0.788	0.663	0.275	0.361	0.018	0.368	0.256

Table A.2: Logistic regression, *with* online augmentation for feature extraction.

Logistic Regression With Online Aug for FE				Before ET							
				Validation				Test			
FS Aug	Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	No	None	No	0.953	0.970	0.923	0.191	0.740	0.718	0.661	0.367
			Yes	0.824	0.918	0.844	0.292	0.660	0.658	0.658	0.392
		EQ	No	0.903	0.934	0.889	0.264	0.560	0.525	0.528	0.396
			Yes	0.853	0.923	0.853	0.306	0.600	0.640	0.546	0.426
		PSTS	No	0.249	0.341	0.250	0.422	0.320	0.588	0.320	0.466
			Yes	0.541	0.555	0.550	0.497	0.300	0.588	0.300	0.458
	Yes	None	No	0.817	0.874	0.811	0.231	0.383	0.152	0.392	0.212
			Yes	0.837	0.907	0.817	0.191	0.453	0.354	0.459	0.158
		EQ	No	0.802	0.857	0.802	0.243	0.432	0.323	0.436	0.169
			Yes	0.820	0.898	0.811	0.193	0.433	0.329	0.448	0.174
Yes	No	None	No	0.928	0.945	0.905	0.211	0.760	0.755	0.700	0.374
			Yes	0.853	0.882	0.829	0.314	0.480	0.520	0.473	0.391
		EQ	No	0.953	0.951	0.931	0.185	0.720	0.725	0.686	0.382
			Yes	0.878	0.882	0.839	0.310	0.600	0.620	0.583	0.402
		PSTS	No	0.275	0.277	0.275	0.447	0.300	0.381	0.300	0.458
			Yes	0.450	0.560	0.450	0.497	0.200	0.500	0.200	0.400
	Yes	None	No	0.799	0.842	0.794	0.239	0.432	0.253	0.438	0.238
			Yes	0.817	0.879	0.797	0.218	0.435	0.323	0.442	0.186
		EQ	No	0.806	0.844	0.799	0.237	0.420	0.266	0.425	0.216
			Yes	0.810	0.868	0.791	0.218	0.462	0.315	0.457	0.203

				After ET							
				Validation				Test			
FS Aug	Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	No	None	No	0.955	0.993	0.930	0.174	0.533	0.362	0.555	0.422
			Yes	0.910	0.959	0.887	0.236	0.683	0.936	0.670	0.392
		EQ	No	0.905	0.974	0.889	0.249	0.600	0.652	0.617	0.451
			Yes	0.935	0.966	0.895	0.249	0.683	0.844	0.688	0.419
		PSTS	No	0.713	0.500	0.711	0.453	0.833	0.500	0.833	0.373
			Yes	0.287	0.500	0.289	0.453	0.183	0.510	0.183	0.387
	Yes	None	No	0.812	0.844	0.797	0.237	0.551	0.015	0.547	0.324
			Yes	0.831	0.866	0.792	0.213	0.606	0.016	0.580	0.294
		EQ	No	0.823	0.861	0.808	0.236	0.543	0.023	0.538	0.321
			Yes	0.825	0.866	0.795	0.222	0.619	0.004	0.591	0.299
Yes	No	None	No	0.820	0.954	0.859	0.263	0.650	0.714	0.637	0.391
			Yes	0.863	0.945	0.844	0.254	0.600	0.326	0.589	0.459
		EQ	No	0.820	0.952	0.833	0.319	0.633	0.810	0.654	0.435
			Yes	0.840	0.942	0.840	0.284	0.733	0.764	0.726	0.390
		PSTS	No	0.558	0.469	0.545	0.492	0.667	0.516	0.667	0.470
			Yes	0.613	0.438	0.622	0.485	0.867	0.760	0.867	0.340
	Yes	None	No	0.826	0.868	0.811	0.229	0.527	0.021	0.527	0.324
			Yes	0.821	0.858	0.785	0.223	0.625	0.010	0.590	0.299
		EQ	No	0.821	0.864	0.807	0.234	0.534	0.047	0.530	0.317
			Yes	0.827	0.850	0.788	0.219	0.608	0.006	0.581	0.301

Table A.3: Random Forest, *without* online augmentation for feature extraction.

Logistic Regression Without Online Aug for FE				Before ET							
				Validation				Test			
FS Aug	Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	No	None	No	0.906	0.948	0.717	0.169	0.580	0.546	0.553	0.139
			Yes	0.877	0.926	0.711	0.182	0.740	0.742	0.595	0.142
		EQ	No	0.928	0.920	0.738	0.180	0.540	0.554	0.531	0.163
			Yes	0.902	0.926	0.729	0.193	0.660	0.642	0.584	0.174
		PSTS	No	0.651	0.632	0.562	0.160	0.800	0.530	0.583	0.126
			Yes	0.651	0.691	0.565	0.154	0.800	0.581	0.593	0.123
	Yes	None	No	0.852	0.917	0.760	0.183	0.630	0.483	0.572	0.156
			Yes	0.822	0.854	0.699	0.173	0.743	0.657	0.604	0.148
		EQ	No	0.861	0.922	0.775	0.184	0.629	0.497	0.578	0.162
			Yes	0.825	0.852	0.704	0.175	0.743	0.644	0.604	0.150
Yes	No	None	No	0.903	0.875	0.703	0.184	0.560	0.554	0.535	0.133
			Yes	0.899	0.887	0.697	0.195	0.700	0.670	0.574	0.154
		EQ	No	0.903	0.934	0.730	0.179	0.560	0.525	0.524	0.160
			Yes	0.899	0.919	0.722	0.199	0.640	0.690	0.584	0.179
		PSTS	No	0.651	0.523	0.552	0.170	0.828	0.533	0.597	0.148
			Yes	0.651	0.479	0.542	0.147	0.800	0.496	0.582	0.123
	Yes	None	No	0.865	0.925	0.765	0.181	0.632	0.484	0.578	0.172
			Yes	0.824	0.848	0.698	0.176	0.723	0.632	0.598	0.150
		EQ	No	0.867	0.925	0.778	0.184	0.626	0.497	0.581	0.170
			Yes	0.819	0.847	0.705	0.183	0.732	0.638	0.603	0.153

				After ET							
				Validation				Test			
FS Aug	Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	No	None	No	0.818	0.962	0.725	0.192	0.800	0.763	0.621	0.134
			Yes	0.868	0.913	0.726	0.205	0.817	0.864	0.635	0.132
		EQ	No	0.798	0.897	0.723	0.222	0.733	0.750	0.631	0.161
			Yes	0.843	0.868	0.729	0.226	0.833	0.819	0.651	0.161
		PSTS	No	0.713	0.553	0.562	0.132	0.817	0.437	0.588	0.116
			Yes	0.713	0.482	0.549	0.117	0.767	0.548	0.579	0.108
	Yes	None	No	0.751	0.793	0.718	0.256	0.744	0.666	0.612	0.113
			Yes	0.682	0.739	0.659	0.235	0.631	0.501	0.573	0.146
		EQ	No	0.747	0.805	0.726	0.257	0.671	0.577	0.583	0.115
			Yes	0.685	0.738	0.660	0.242	0.655	0.567	0.578	0.138
Yes	No	None	No	0.848	0.910	0.726	0.205	0.867	0.924	0.627	0.120
			Yes	0.873	0.855	0.720	0.216	0.950	1.000	0.687	0.108
		EQ	No	0.848	0.831	0.724	0.243	0.833	0.916	0.649	0.142
			Yes	0.873	0.796	0.723	0.241	0.900	0.980	0.686	0.137
		PSTS	No	0.713	0.623	0.588	0.175	0.833	0.433	0.625	0.147
			Yes	0.713	0.355	0.581	0.190	0.833	0.533	0.629	0.148
	Yes	None	No	0.747	0.793	0.718	0.260	0.733	0.596	0.600	0.123
			Yes	0.680	0.724	0.654	0.242	0.620	0.452	0.567	0.155
		EQ	No	0.753	0.806	0.727	0.256	0.678	0.590	0.589	0.118
			Yes	0.683	0.734	0.657	0.243	0.638	0.554	0.576	0.145

Table A.4: Random Forest, *with* online augmentation for feature extraction.

Logistic Regression With Online Aug for FE				Before ET							
				Validation				Test			
FS Aug	Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	No	None	No	0.903	0.959	0.734	0.173	0.500	0.487	0.530	0.170
			Yes	0.877	0.918	0.724	0.193	0.680	0.750	0.585	0.168
		EQ	No	0.906	0.942	0.740	0.191	0.540	0.517	0.522	0.185
			Yes	0.906	0.926	0.738	0.192	0.640	0.724	0.584	0.186
		PSTS	No	0.651	0.545	0.554	0.163	0.800	0.570	0.587	0.124
			Yes	0.651	0.643	0.541	0.107	0.780	0.539	0.558	0.088
	Yes	None	No	0.862	0.918	0.777	0.197	0.621	0.440	0.577	0.175
			Yes	0.838	0.881	0.740	0.184	0.703	0.653	0.611	0.156
		EQ	No	0.857	0.919	0.781	0.199	0.615	0.433	0.577	0.175
			Yes	0.842	0.880	0.744	0.192	0.717	0.660	0.624	0.163
Yes	No	None	No	0.881	0.909	0.725	0.192	0.640	0.617	0.538	0.167
			Yes	0.899	0.901	0.712	0.197	0.660	0.765	0.591	0.165
		EQ	No	0.881	0.934	0.735	0.184	0.600	0.555	0.525	0.174
			Yes	0.877	0.901	0.727	0.211	0.660	0.764	0.595	0.200
		PSTS	No	0.641	0.525	0.553	0.169	0.800	0.544	0.596	0.133
			Yes	0.651	0.624	0.544	0.120	0.800	0.626	0.577	0.100
	Yes	None	No	0.862	0.923	0.777	0.188	0.629	0.437	0.576	0.183
			Yes	0.836	0.881	0.735	0.184	0.696	0.632	0.610	0.162
		EQ	No	0.869	0.925	0.788	0.194	0.616	0.432	0.578	0.183
			Yes	0.838	0.880	0.743	0.194	0.694	0.652	0.619	0.167

				After ET							
				Validation				Test			
FS Aug	Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	No	None	No	0.798	0.921	0.747	0.221	0.717	0.764	0.605	0.148
			Yes	0.848	0.885	0.730	0.232	0.850	0.936	0.658	0.145
		EQ	No	0.773	0.904	0.748	0.242	0.800	0.794	0.607	0.150
			Yes	0.776	0.800	0.721	0.276	0.833	0.847	0.659	0.190
		PSTS	No	0.713	0.547	0.581	0.171	0.833	0.505	0.624	0.144
			Yes	0.713	0.434	0.585	0.200	0.833	0.384	0.634	0.173
	Yes	None	No	0.752	0.801	0.731	0.262	0.747	0.683	0.618	0.110
			Yes	0.741	0.776	0.704	0.249	0.756	0.681	0.635	0.136
		EQ	No	0.755	0.819	0.737	0.258	0.666	0.620	0.592	0.113
			Yes	0.747	0.781	0.711	0.254	0.780	0.737	0.651	0.127
Yes	No	None	No	0.748	0.864	0.717	0.235	0.85	0.904	0.626	0.133
			Yes	0.851	0.814	0.710	0.246	0.917	0.954	0.701	0.152
		EQ	No	0.803	0.829	0.720	0.261	0.817	0.844	0.619	0.149
			Yes	0.806	0.822	0.719	0.249	0.867	0.898	0.664	0.166
		PSTS	No	0.713	0.680	0.602	0.181	0.833	0.581	0.632	0.144
			Yes	0.713	0.434	0.587	0.195	0.833	0.476	0.629	0.149
	Yes	None	No	0.753	0.809	0.730	0.263	0.749	0.692	0.615	0.111
			Yes	0.745	0.767	0.704	0.250	0.743	0.652	0.635	0.142
		EQ	No	0.753	0.800	0.733	0.265	0.698	0.656	0.602	0.109
			Yes	0.744	0.779	0.709	0.251	0.764	0.716	0.643	0.131

Table A.5: Support Vector Machine, *without* online augmentation for feature extraction.

Support Vector Machine Without Online Aug for FE				Before ET							
				Validation				Test			
FS Aug	Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	No	None	No	0.955	1.000	0.904	0.136	0.650	0.812	0.671	0.305
			Yes	0.890	0.971	0.853	0.198	0.700	0.948	0.619	0.246
		EQ	No	0.910	0.983	0.900	0.190	0.633	0.800	0.653	0.350
			Yes	0.865	0.976	0.893	0.219	0.650	0.674	0.605	0.385
	Yes	None	No	0.721	0.689	0.661	0.223	0.439	0.152	0.534	0.215
			Yes	0.631	0.602	0.607	0.180	0.396	0.258	0.568	0.190
Yes	No	None	No	0.905	0.981	0.899	0.195	0.533	0.568	0.553	0.356
			Yes	0.915	0.976	0.890	0.207	0.633	0.736	0.633	0.408
		EQ	No	0.915	0.976	0.889	0.235	0.617	0.740	0.645	0.403
			Yes	0.915	0.978	0.893	0.220	0.550	0.472	0.555	0.426
	Yes	None	No	0.696	0.670	0.636	0.214	0.409	0.096	0.558	0.224
			Yes	0.645	0.563	0.601	0.212	0.502	0.283	0.637	0.194

				After ET							
				Validation				Test			
FS Aug	Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	No	None	No	0.978	0.959	0.865	0.174	0.580	0.693	0.549	0.266
			Yes	0.903	0.915	0.779	0.220	0.400	0.438	0.383	0.309
		EQ	No	0.878	0.912	0.840	0.284	0.600	0.677	0.551	0.380
			Yes	0.906	0.931	0.823	0.227	0.440	0.530	0.405	0.349
	Yes	None	No	0.771	0.827	0.711	0.194	0.571	0.497	0.570	0.171
			Yes	0.685	0.753	0.637	0.158	0.603	0.626	0.605	0.136
Yes	No	None	No	0.978	0.948	0.894	0.200	0.720	0.715	0.617	0.330
			Yes	0.856	0.926	0.829	0.253	0.480	0.557	0.445	0.388
		EQ	No	0.903	0.931	0.872	0.261	0.600	0.625	0.590	0.391
			Yes	0.853	0.918	0.841	0.278	0.580	0.670	0.506	0.391
	Yes	None	No	0.749	0.810	0.670	0.172	0.473	0.315	0.533	0.169
			Yes	0.698	0.752	0.640	0.191	0.638	0.673	0.635	0.143

Table A.6: Support Vector Machine, *with* online augmentation for feature extraction.

Support Vector Machine With Online Aug for FE				Before ET							
				Validation				Test			
FS Aug	Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	No	None	No	0.890	0.945	0.785	0.204	0.650	0.790	0.623	0.273
			Yes	0.895	0.913	0.742	0.202	0.567	0.530	0.582	0.258
		EQ	No	0.845	0.940	0.837	0.268	0.583	0.638	0.603	0.373
			Yes	0.870	0.916	0.806	0.280	0.550	0.134	0.528	0.432
	Yes	None	No	0.722	0.688	0.658	0.216	0.465	0.268	0.551	0.210
			Yes	0.657	0.667	0.607	0.155	0.438	0.539	0.601	0.138
Yes	No	None	No	0.845	0.966	0.869	0.233	0.617	0.672	0.622	0.374
			Yes	0.895	0.909	0.808	0.277	0.550	0.450	0.542	0.412
		EQ	No	0.825	0.942	0.828	0.275	0.500	0.308	0.533	0.392
			Yes	0.850	0.892	0.817	0.272	0.567	0.276	0.572	0.413
	Yes	None	No	0.692	0.674	0.640	0.209	0.443	0.195	0.571	0.213
			Yes	0.639	0.564	0.601	0.212	0.544	0.357	0.641	0.192

				After ET							
				Validation				Test			
FS Aug	Sliced	Offline Aug	Filter	Acc	ROC	Prob	SD	Acc	ROC	Prob	SD
No	No	None	No	0.953	0.929	0.836	0.191	0.620	0.655	0.562	0.235
			Yes	0.903	0.890	0.743	0.223	0.520	0.570	0.472	0.277
		EQ	No	0.924	0.942	0.873	0.211	0.700	0.752	0.622	0.329
			Yes	0.903	0.907	0.781	0.231	0.400	0.550	0.445	0.295
	Yes	None	No	0.776	0.822	0.710	0.191	0.572	0.497	0.572	0.180
			Yes	0.688	0.740	0.634	0.170	0.607	0.667	0.623	0.139
Yes	No	None	No	0.953	0.940	0.900	0.199	0.700	0.762	0.642	0.316
			Yes	0.878	0.898	0.817	0.266	0.500	0.588	0.483	0.356
		EQ	No	0.978	0.984	0.916	0.188	0.680	0.690	0.585	0.367
			Yes	0.856	0.934	0.831	0.282	0.480	0.588	0.487	0.370
	Yes	None	No	0.774	0.833	0.707	0.186	0.529	0.391	0.561	0.193
			Yes	0.690	0.737	0.634	0.182	0.624	0.674	0.629	0.138

B

Code

All final code that was written for this project is presented here, in sections B.1 through B.10, and can also be found on GitHub. One file is covered per section (with the section being named after the file), where a short description of the script is followed by the code itself. **For this printed version, the code is omitted**.

B.1 `functions_and_classes.py`

This script contains all classes and separate functions that all other scripts use. These include:

- `AudioUtil` (class) for utility regarding audio handling and augmentation.
- `SoundDS` (class) which uses `AudioUtil` to generate a `torch Dataset` from the given data, used for training and validation of PyTorch models.
- `training` (function) and `inference` (function) for training and inference of PyTorch models.
- `preprocess_one_file` (function) that takes a given file and converts it to the correct format of spectrogram that the trained models work with, used when inferring singular files. Also slices the file into multiple spectrograms for the models that uses sliced data.
- `FeaturesModel` (class) for training the models based on feature extraction.
- `augment_for_feature_extraction` (function) handles the online augmentation setup for the feature extraction-based models, utilizing `AudioUtil`.
- `AudioFeature` (class), `parse_metadata` (function) and `extract_audio_features` (function) for extracting audio features from a file, used for extracting the selected openSMILE features.
- `interpret_configuration` (function) for determining which configuration is selected, based on for example model name or selected feature names, to determine how the model should be trained/inferred.
- `calculate_fold_timings` (function) and `calculate_slice_timings` (function) for controlled generation of sliced data into different folds.
- `ROC_analysis` (function) for generating an ROC curve and the corresponding ROC-AUC score.

B.2 distribute_into_folds.py

Distributes files, represented in a `metadata.csv` file, into K different folds, by editing the csv-file. The distribution is based on the files' BOAS classes, attempting to place an equal amount of each class in each fold. If there are not enough files to evenly split into the folds, multiple folds are allocated, into which the files are split without overlap during slicing. When not sliced, the data is placed in the first of their allocated folds.

B.3 slice_data.py

Slices the data given by a `metadata.csv` file and generates a new metadata file with the newly generated data.

B.4 augment_data.py

Augments the data given by a `metadata.csv` file and generates a new metadata file with the newly generated data.

B.5 find_opensmile_features.py

Extracts openSMILE features for all data that is given through a metadata file, with a given configuration, and passes the features through a Mann-Whitney U test, to find the best features. Saves the best features to a csv-file, which is used during training and testing.

B.6 sklearn_training.py

Trains feature extraction-based Scikit-learn models, with different configurations, and saves them as an ONNX model file.

B.7 nn_training.py

Trains spectrogram-based PyTorch models, with different configurations, and saves them as an ONNX model file (and a pytorch model file just in case it may be needed).

B.8 pytorch_models.py

Defines the PyTorch models as classes, defining their structure and forward pass calculations.

B.9 `onnx_inference.py`

Loads in all ONNX model files of a given configuration (all models have the same configuration, but are trained on different folds- per K-fold cross validation), determines whether its Scikit-learn (and thus feature extraction-based) or PyTorch (and thus spectrogram-based), to then infer selected data with it. For validation data, it infers the files from the fold of which the model was not trained on, and for testing data it infers all data on all models.

B.10 `onnx_file_test.py`

Similar to `onnx_inference.py`, but instead of inferring with a known correct label, it just gives the model predictions of the file(s). Is intended as the real-world use where the purpose is to predict the BOAS grade, rather than evaluating the performance of the models.

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY