Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Audio Compression for Acoustic Sensing

Semester Thesis

Martin Lendi
lendim@student.ethz.ch

Computer Engineering and Networks Laboratory
Department of Information Technology and Electrical Engineering
ETH Zürich

Supervisors:
Matthias Meyer
Jan Beutel
Prof. Dr. Lothar Thiele

June 20, 2017

Acknowledgements

I would like to thank my supervisors Jan Beutel and Matthias Meyer for their great support during the semester thesis. Furthermore, I would like to thank the Computer Engineering and Networks Laboratory for the provided workplace and equipment and for giving me the opportunity to work on this thesis.

Abstract

Audio analysis is used in numerous applications such as music classification or environmental monitoring. For latter, data has to be transmitted through a wireless sensor network from the emission sensors to a server. To improve energy efficiency, a prior compression for the data to be sent is suitable. Therefore, a novel data-driven audio compression scheme based on existing work should be designed and analyzed. The concepts used should enable to do a compression for later reconstruction as well as an efficient classification.
For classification, a convolutional neural network is used, where two different inputs to this classification network are compared: a model based on the modified discrete cosine transform combined with a band energy computation and one based on the short time fourier transform followed by Mel scaling. Both models showed similar classification accuracy after training.
For compression, a simple autoencoder is applied on the coefficients of the modified discrete cosine transform and the reconstruction quality is compared to a Mel inversion model and a simplified version of the CELT codec. A better reconstruction quality is reached by the autoencoder than by the Mel inversion model whereas CELT outperformed every other model.

Contents

CHAPTER 1

Introduction

1.1 Problem Description

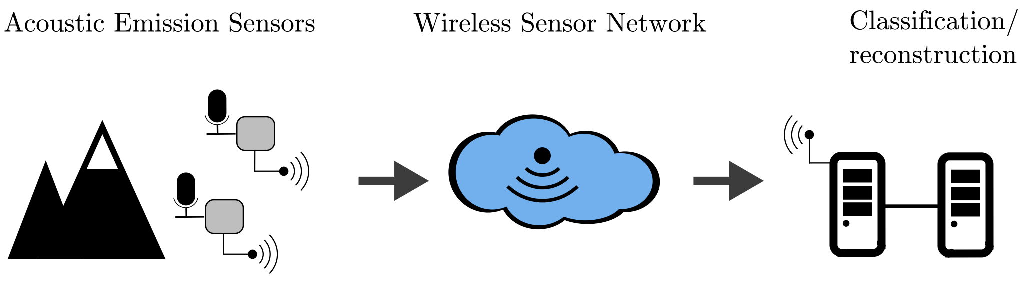

Figure 1.1: Wireless sensor network (WSN) with acoustic sensors. Acoustic data is recorded by the sensors and sent to the server via a wireless sensor network.

Audio analysis can be used for music genre or artist classification, speaker identification [1] or for environmental monitoring such as bird detection or hazard warning systems [2]. For latter, a wireless sensor network (WSN) can be used to transmit data from acoustic sensors which are placed at high distance (e.g. in the mountains). The transmission of audio data through a WSN has a big impact on the energy budget, therefore it would be benefitial to compress the data or to do the classification directly on the sensors. After all, the goal is to compute as less as possible on the sensor nodes but send only as much as needed to the server.
Convolutional neural networks (CNNs) have proved to achieve a high detection accuracy for environmental events. Unfortunately, novel acoustic event detection algorithms based on CNNs are computationally expensive and have high memory requirements [3]. Both can be reduced by structural optimizations to the CNN [4].
In this project, a novel data-driven audio compression scheme based on existing work [4] should be designed and analyzed. This scheme should not only be able to do a classification, but let the possibility to do a compression such that the original data can be reconstructed on the server as good as possible.

1.2 Project Goals

This project includes an evaluation of existing compression schemes and finding audio processing techniques suitable for compression and adapting them for an existing convolutional neural network.
Four major goals are approached in this project:
Audio compression algorithms. The first goal is to find out about existing lossy audio compression schemes and the suitability of reusing components of these schemes for a CNN-based algorithm.
Extending CNN for audio classification. In a second step, an existing convolutional neural network for audio classification should be changed from using a mel-spectrogram to a more suitable signal transformation. This transformation should be chosen such that the original signal can be reconstructed and the classification accuracy of the whole network remains about the same.
Autoencoder. The above mentioned neural network should be extended to an autoencoder, where the input and the output is a raw audio waveform.
Evaluation. The different audio compression schemes should be evaluated regarding the accuracy of the CNN or the reconstruction error between autoencoder input and output. Moreover, the suitability to use the schemes for audio compression on low power embedded systems should be examined.

1.3 Audio Compression Algorithms

Lossy compression schemes are used in a wide range of applications. The most known audio coding format is MP3, which achieves a size reduction of 75% to 95% compared to uncompressed CD quality digital audio. In contrast to lossless audio codecs, lossy formats are not able to reproduce the original sound file exactly. Still, the information loss cannot or can only hardly be heard by the human ear, because data is removed in a perceptual model, which is based on removing or reducing the exactness of the less audible sounds.
In appendix A, the results of a multiformat listening test carried out in July 2014 [5] are shown. In this test, the sound quality of Opus, AAC, Ogg Vorbis and MP3 was evaluated. As a result, the Opus codec performed best with respect to quality at the same bitrate as AAC and Ogg Vorbis.
Opus is built out of two separate codecs, SILK (for speech codec) and CELT (for music codec). In this work, the focus is on the most important parts of CELT, which are the modified discrete cosine transform (Section 2.1) and the preservation of the energy in a given band shape combined with pyramid vector quantization (Section 2.2).

Chapter 2

Theory

In this part, the methods and concepts used in the project are introduced. First, the modified discrete cosine transform, which is used by most lossy audio compression schemes, is shown. Second, band energy computation and pyramid vector quantization and how they work together is introduced. A short introduction to neural networks is given in Section 2.3, followed by the definition of peak signal-to-noise ratio.

2.1 Modified Discrete Cosine Transform

Because of several advantages of the modified discrete cosine transform (MDCT) for tonal signals, well known lossy audio compression codecs like MP3, Ogg Vorbis, AAC and Opus are based on this transform.
The underlying principle of the MDCT is time-domain aliasing cancellation (TDAC) [6]. Furthermore, it is a lapped transform: the first half of the input data is the second half of the input data of the previous call. With the concepts of lapping and TDAC, we can avoid compression artifacts, reach a good frequency selectivity and, what is most valuable, perfect invertibility of the transform.
The modified discrete cosine transform maps $2N$ real inputs to N real outputs and is defined by:

$$X_k = \sum_{n=0}^{2N-1} x_n \cos\left[\frac{\pi}{N}\Big(n + \frac{1}{2} + \frac{N}{2}\Big)\Big(k + \frac{1}{2}\Big)\right]$$

As mentioned, the transform can be inverted. For the inverse modified discrete cosine transform (IMDCT), N real numbers are transformed into $2N$ real numbers according to the formula:

$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cos\left[\frac{\pi}{N}\Big(n + \frac{1}{2} + \frac{N}{2}\Big)\Big(k + \frac{1}{2}\Big)\right]$$

Typically, a symmetric window $w(n)$ of length $2N$ is used before the MDCT and after the IMDCT. In order that aliasing cancellation works and therefore MDCT is invertible, the window has to fulfill the Princen-Bradley condition [6]:

$$w(n)^2 + w(n+N)^2 = 1$$

In Figure 2.1, three windows that fulfill such a condition are plotted, where the framelength is chosen as 160 samples. Opus codec and Ogg Vorbis use a window where the overlapping part is defined by

$$w(n) = \sin\left[\frac{\pi}{2}\sin^2\left(\frac{\pi(n+\frac{1}{2})}{2L}\right)\right].$$

MP3 and AAC use a slightly different expression for the overlapping part:

$$w(n) = \sin\left[\frac{\pi}{2N}\left(n+\frac{1}{2}\right)\right].$$

In this work, a standard cosine window is used as shown in Figure 2.1.

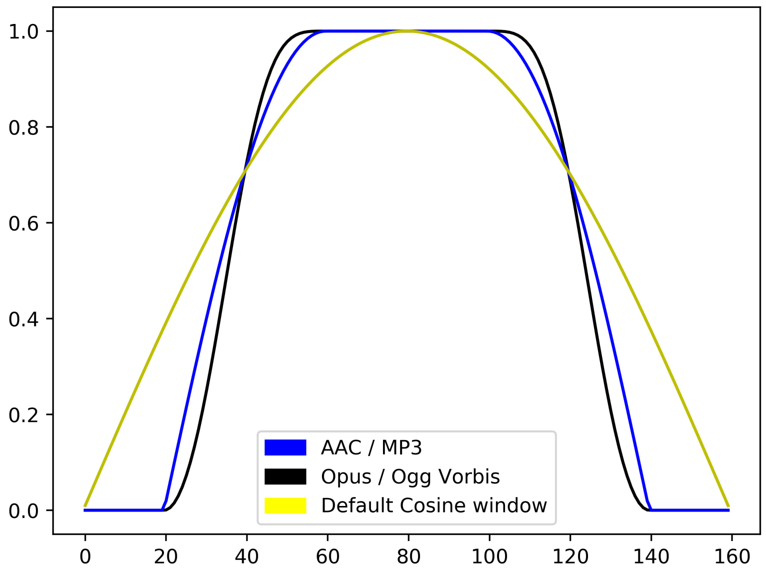

Figure 2.1: Windows used for the MDCT by AAC/MP3 (blue), Opus/Ogg Vorbis (black) and a standard cosine window (yellow)

2.2 Energy Computation and Pyramid Vector Quantization

The main point in CELT is to preserve the spectral envelope by coding the energy of a set of bands. A lot of psychoacoustic knowledge is included in the choice of the format of the bands. Within these bands, psychoacoustic masking is much stronger than between bands. If one dominant frequency component in a band is present, the others with lower contribution that are in the same band can be neglected without hearing any difference, as long as the total energy in the band is preserved.
The band format used is similar to the bark scale, which approximates the critical bands of hearing (Figure 2.2).

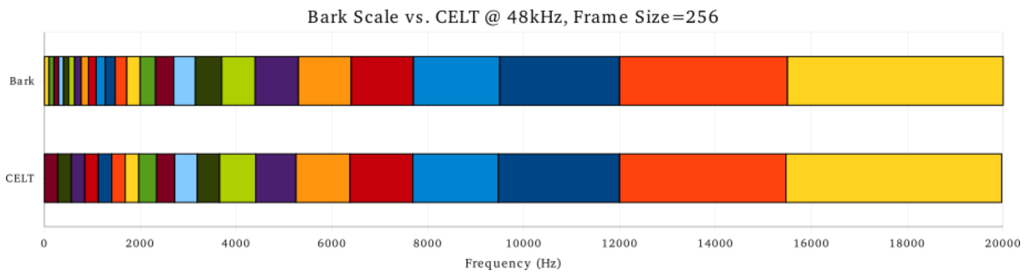

Figure 2.2: Comparison of the Bark scale and the CELT band layout for a sampling frequency of 48 kHz and frame size of 256 [7]. The colors indicate the band areas.

The coefficients of the MDCT are grouped into the bands (Figure 2.2) and then the energy of this produced vector $\boldsymbol{x}$ is computed in each band B:

$$E = \sum_{x_i \in B} |x_i|^2.$$

Each band is then normalized by the square-root of the just computed energy, resulting in an N-dimensional unit vector in each band. Note that the dimension of this unit vector is higher in the high frequencies than in the low frequencies. This vector describes the "shape" of the energy within the band. To sum up, the spectrum is separated from the modified discrete cosine transform to a magnitude (energy) and a unit vector part.

To reach a larger compression ratio, these unit vectors are quantized. Namely, we replace the N-dimensional vectors by a number which indicates the position in a precomputed set of vectors. This set should be evenly distributed on a sphere and is called codebook. The original unit vectors from our spectrum are mapped to the nearest point of this codebook in order to get an error which is as small as possible.

Evenly distributing points on a 2- or 3-dimensional sphere may be easy, but for arbitrary dimension, it is not possible. In Pyramid vector quantization (PVQ), points are evenly distributed on a pyramid instead. Formally, the set of vectors with integer coodinates whose magnitudes sum up to K is computed:

$$S(N, K) = \{\boldsymbol{y} \in \mathbb{Z}^N : \sum_{i=1}^{N} |y_i| = K\}.$$

$S(N, K)$ has the shape of a pyramid around the origin. The vectors in $S(N, K)$ are then normalized to unit norm to get the final codebook.

Example 1 Codebook for $N = K = 2$:

$$S(2,2) = \{(2,0),(-2,0),(1,1),(1,-1),(-1,1),(-1,-1),(0,2),(0,-2)\} \quad \square$$

In Figure 2.3, the red points represent the points in $S(2,2)$, whereas the green points describe the final PVQ codebook for $N = K = 2$.

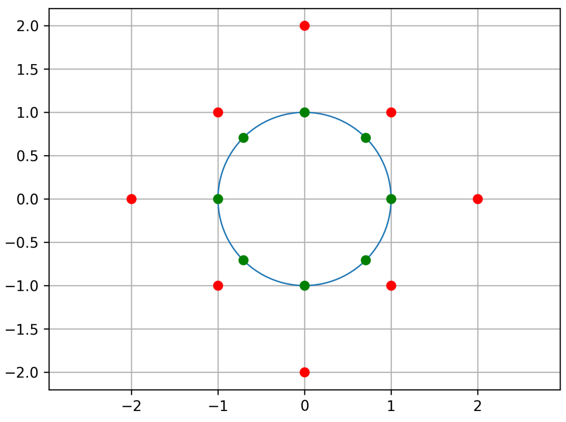

Figure 2.3: PVQ for $N = K = 2$: points in the final codebooks (green points) are the normalized points from $S(2,2)$ (red points).

2.3 Neural Networks

Neural networks are able to approximate an arbitrary function by learning from observation data. They often represent mathematical models that defines a function $f : X \to Y$. Mostly, the networks include several layers with multiple neurons which are connected by weights that can be updated in the learning process. In this project, fully-connected neural networks (dense layers) and convolutional neural networks are used.

2.3.1 Dense Layers

Dense layers consist of one single layer that is fully connected, meaning each input is connected to each output with a certain weight. Dense layers are represented as a matrix vector multiplication, thus it implements the operation:

$$\text{output} = \text{activation}(\text{dot}(\text{input, kernel}) + \text{bias}),$$

where activation is the element-wise activation function, kernel is the weight matrix created by the layer and bias is a bias vector created by the layer. If no activation is specified, it is applied linearly (activation(x)=x). Assuming linear activation, an input vector $\boldsymbol{u} \in \mathbb{R}^n$ and a weight matrix (kernel) $W \in \mathbb{R}^{n \times m}$ result in an output vector $\boldsymbol{y} \in \mathbb{R}^m$.

Example 2 Assume $n = 3$ and $m = 2$, the neural network looks as in Figure 2.4 and can be representented as the weight matrix W. If no activation function and no bias is specified, the output is the matrix vector multiplication of W and the input.

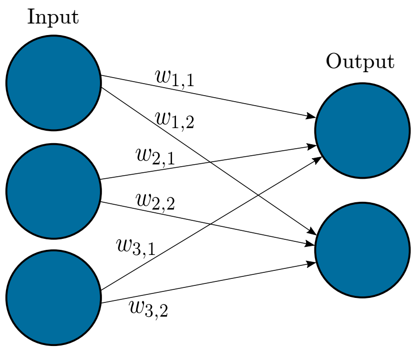

Figure 2.4: Fully connected neural network with three inputs and two outputs

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix}$$

$$\boldsymbol{y} = W\boldsymbol{u} \qquad \square$$

2.3.2 Convolutional Neural Network

An important category of neural networks are convolutional neural networks (CNNs) [8], which are especially suitable for data with a grid-like topology as for example images. The network applies the mathematical operation of a convolution, which is a special linear operation. Instead of the matrix multiplication in Subsection 2.3.1, a convolution is used for a CNN in at least one of the layers. In this work, a convolutional neural network that is similar to the one proposed in [4] for audio classification is changed such that the original signal can be reconstructed and the classification accuracy remains about the same. As in [4], the system is divided into three components: front-end, feature extraction and classification. The front-end, which takes the raw time-domain audio waveform and transforms it to a time-frequency representation, will be introduced in the next chapter.

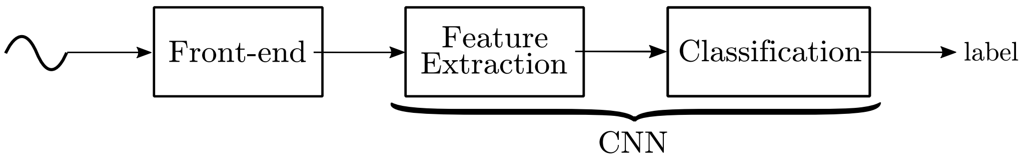

Figure 2.5: Model architecture of proposed CNN: The waveform is passed through the front-end to get a time-frequency representation, then the CNN extracts features and classifies these features to get a label.

In order to get a classification, a CNN first extracts features of the time-frequency representation. The feature extraction of the model used is structured in two sections with two convolutional layers each. With this structure, an adequate number of operations and a moderate number of parameters is used.
These extracted features are then classified. For that, three convolutional layers followed by average pooling is used, which reduces the output of the last convolutional layer to an array of the size that matches the number of labels. For all the convolutional layers except for the first layer which is activated linearly, rectified linear unit (ReLU) is used for activation.
The following table shows all the layers of the CNN used, where FE stands for feature extraction and C stands for classification. The input shape is (256, 68, 1) in this case.

Step	Layer	Filter size	Output Shape	Stride
1	(FE1, Conv2D)	(3,1)	(256, 68, 64)	(1,1)
2	(FE1, Conv2D)	(3,3)	(128, 34, 64)	(2,2)
3	(FE2, Conv2D)	(3,3)	(128, 34, 128)	
4	(FE2, Conv2D)	(3,3)	(64, 17, 128)	(2,2)
5	(C, Conv2D)	(3,3)	(64, 17, 128)	
6	(C, Conv2D)	(1,1)	(64, 17, 128)	
7	(C, Conv2D)	(1,1)	(64, 17, 128)	
8	(C, AvgPool)		(1, 1, 28)	

Table 2.1: Structure of the CNN

2.4 Peak Signal-to-Noise Ratio

Peak signal-to-noise ratio (PSNR) compares the maximum possible power of a signal and the power of corrupting noise and is often used to measure the quality of lossy compression codecs. Let $y^{(i)}$ be the original samples of an audio file and $y_{rec}^{(i)}$ its reconstructed samples where $i = 0, ..., N - 1$ and N is the number of samples. PSNR is defined via the mean squared error (MSE):

$$MSE = \frac{1}{N} \sum_{i=0}^{N-1} \left(y^{(i)} - y_{rec}^{(i)}\right)^2$$

PSNR is defined as:

$$\begin{aligned} \text{PSNR} &= 10 \cdot \log_{10} \left(\frac{MAX_y^2}{MSE} \right) \\ &= 20 \cdot \log_{10} \left(\frac{MAX_y}{\sqrt{MSE}} \right) \\ &= 20 \cdot \log_{10}(MAX_y) - 10 \cdot \log_{10}(MSE) \end{aligned}$$

where MAX_y is the maximum possible value of the original samples. In general, a higher value of the PSNR indicates a better quality of the reconstruction, because higher PSNR means lower mean squared error.

Chapter 3

Models

3.1 Classification Models

In this project, the CNN from Subsection 2.3.2 is evaluated on its accuracy using two different front-ends.

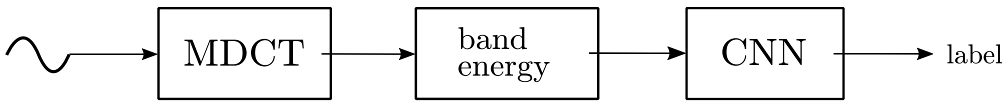

Figure 3.1: Model 1: CNN with a front-end made of MDCT/band energy.

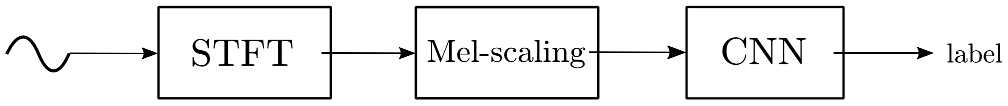

Figure 3.1: Model 2: CNN with a front-end made of STFT/Mel-scaling.

The first model uses the energy computed in a given band scale as an input to the convolutional neural network (Figure 3.1). In this model, the time-frequency representation is obtained by the modified discrete cosine transform. Subsequently, the coefficients of the MDCT are grouped into critical bands of hearing by computing the energy in a given band layout. Like this, the linear frequency scale of the MDCT is removed and a characteristic information of the waveform is obtained.
To reach a better resolution of the energy, the original band format from Figure 2.2 is modified such that each band is divided into several areas of equal length. This results in a total of 34, 68 or 136 energy coefficients per window. The spectrogram, which is the final visual representation of the energy spectrum, consists of 256 such representations. Assuming a framesize of 1280 samples and a hop length of 40 ms, this corresponds to a time span of 10.24 seconds per frame.
This model is compared to the model used in [4], where the front-end is a short

time fourier transform (STFT) followed by Mel scaling (Figure 3.1).

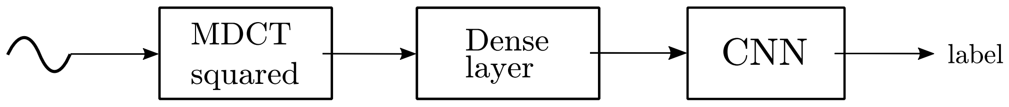

Figure 3.2: Model 3: Dense layer trained simultaneously with the CNN

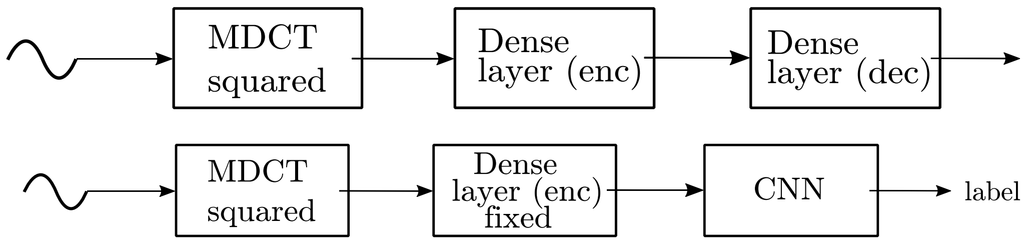

Figure 3.2: Model 4: Train autoencoder formed by two dense layers, then the first layer is fixed and the CNN is trained

In the third and fourth model, the fact that the energy computation from model 1 can be represented as a matrix is used. This matrix is used as initialization for the weights of the dense layer and is trained simultaneously to the CNN in model 3 (Figure 3.2). In the fourth model, an autoencoder that contains two dense layers is first trained and after that, the CNN is trained taking the output the first dense layer as its input. Here, the dense layer before the CNN is not trained anymore with the CNN but is kept fixed with the weights obtained by the autoencoder.

3.2 Compression Models

Three different compression models are compared in Chapter 4, which will be introduced here: CELT, Mel and an autoencoder.
The first model is a simplified version of CELT, containing only the most important elements of this audio compression codec: energy computation and pyramid vector quantization (PVQ). On the encoder side, the coefficients of the MDCT are used to compute the energy in a given band shape and to quantize the normed vectors by comparing them to a precomputed codebook (as described in Section 2.2). The decoder gets back the coded vectors using the same codebook as the encoder, which is denoted as inverse pyramid vector quantization (iPVQ) in Figure 3.3. Then, these vectors are denormalized by multiplying with the energy,

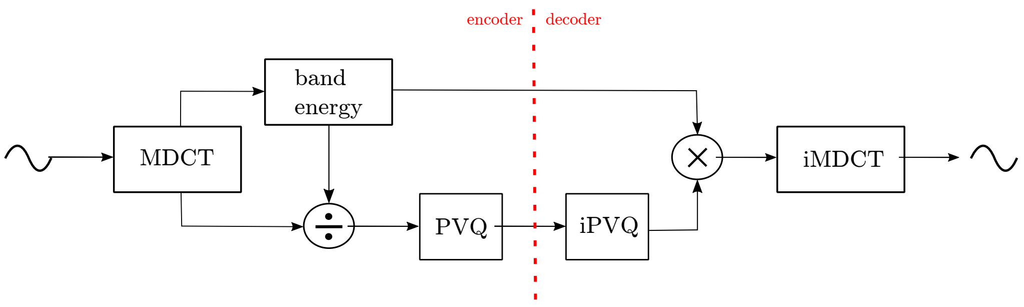

Figure 3.3: Simplified version of the CELT codec. The energy is computed from the MDCT coefficients in different energy band and a PVQ executed on the normed MDCT coefficients.

resulting in a spectrum that can be inverted using the inverse MDCT (iMDCT) to get back a waveform.

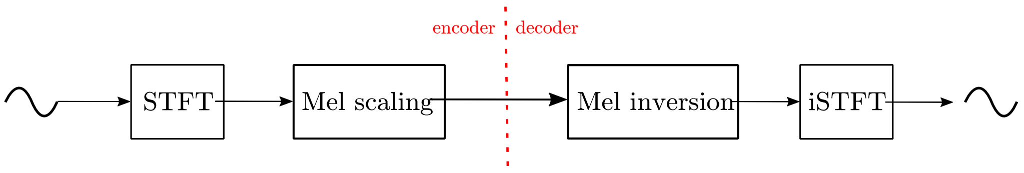

Figure 3.4: Mel encoding and decoding. The coefficients of the STFT are passed through a Mel filter. A Mel inversion filter and inverse STFT is used for decoding.

It is difficult to recover the original signal from a Mel spectrogram. Nevertheless, there is an approach by Tim Sainburg [9] which is used to invert Mel spectrograms back into waveforms. The Mel coefficients are passed through a Mel inversion filter and an inverse short-time fourier transform is carried out to get back the waveform (Figure 3.4) [10].
An autoencoder is the term for a lossy data compression algorithm where compression and decompression are data-specific and learned automatically from training data. The compressing and decompressing functions are implemented with neural networks.

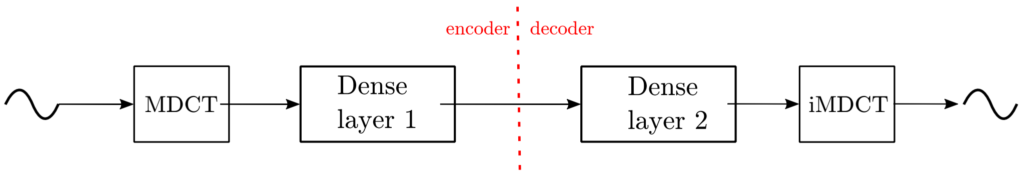

Figure 3.5: Autoencoder based on two dense layers. Dense layer 1 reduces dimension of the input and the dense layer 2 re-establishes original dimension of the spectrum.

In this project, a single fully-connected neural layer is used as encoder as well as decoder (Figure 3.5). The layers are initialized with a matrix that is similar to the energy computation. The coefficients of the MDCT are initially grouped into 136 bands, but the energy is not computed. For decoding, we initially distribute the values of the 136 bands regularly back on the 640 coefficients of the MDCT and the inverse modified discrete cosine transform is carried out.

Chapter 4

Evaluation

4.1 Dataset

All tests are carried out on a dataset which contains various sound files consisting of 28 different event types of variable length[1]. This dataset contains 5223 samples with a total length of 768.4 minutes and is divided into a training set (75%) and a test set (25%). A fourth of the training set is used for validation.

4.2 Classification

First, the CNN using the two different front-ends is compared: band energy and Mel spectrogram (models 1&2). As an input to the CNN, it is distinguished between two different window framelengths used by the MDCT and the STFT. For a window framelength of 1280 samples, the energy and Mels are computed for three different number of bins whereas for a framelength of 640 only 68 energy or Mel bins are used. For both models, the accuracy on a test set after training for 30 epochs is measured. The accuracy indicates the proportion of correctly predicted labels by the CNN. The logarithm of the energy and Mel spectrogram is used as an input to the CNN because a higher accuracy is reached with them.

	(framelength, number of bins)			
Models	(1280, 34)	(1280, 68)	(1280, 136)	(640, 68)
Model 1	0.8653	0.8711	0.8415	0.846
Model 2	0.8713	0.8704	0.8734	0.8561

Table 4.1: Accuracy of CNN trained on the band energy (model 1) and the Mel spectrogram (model 2)

[1] `https://data.vision.ee.ethz.ch/cvl/ae_dataset/`

From Table 4.1, it follows that the two front-ends perform similar with respect to classification accuracy. For model 1, the accuracy for 136 bins is worse compared to 68 bins, which is a bit counter-intuitive, since the resolution of the energy image is higher with 136 bins and thus should perform better because the CNN can learn on more details.
Because the energy model performs best with a window framelength of 1280 and 68 bins, this configuration is used to test models 3 and 4. Dense layers are used here to get an energy representation of the spectrum and training is carried out with (model 3) or before (model 4) the CNN. For validation, the two models are also tested without training the dense layers, which should correspond to the standard energy computation of model 1. For all the tests, a training of 30 epochs is used and the logarithm of the spectrum is taken before the CNN.

Models	Accuracy
Model 3 (without dense training)	0.8603
Model 3 (with dense training)	0.7647
Model 4 (without dense training)	0.8543
Model 4 (with dense training)	0.7128

Table 4.2: Accuracy of CNN trained with model 3 and model 4

From Table 4.2, it is apparent that the models without training perform a bit worse than in the test above where an accuracy of 87.11% is reached, although it should get to the same result. Anyway, both models only reach a low accuracy when the dense layers are trained.

4.3 Autoencoder

For the autoencoder (Figure 3.5), the model when training both dense layers (autoencoder 1) and when training only dense layer 2 (autoencoder 2) for 50 epochs is tested. When training only the second layer, the weights are initialized randomly for layer 2. For all the evaluated models, a compression ratio around 0.21 is used, resulting in 136 bins for the output of dense layer 1, because CELT uses 68 bins for the coded vectors as well as for the energy.
These models have been tested on a guitar, tone, violin and bird audio file from the test set. As a measure to compare the sound quality of the original waveform and the waveform after decoding, peak signal-to-noise ratio is used as introduced in Section 2.4.

PSNR	*guitar*	*tone*	*violin*	*bird*
CELT	43.05	59.82	30.51	30.84
Mel	13.13	2.05	11.12	17.11
Autoencoder 1	31.24	36.75	16.88	19.55
Autoencoder 2	18.75	5.72	14.20	20.39

Table 4.3: Peak signal-to-noise ratio of the models CELT, Mel and Autoencoder for a guitar, tone, violin and bird test file. Autoencoder 1: training both dense layers. Autoencoder 2: training only dense layer 2.

From Table 4.3, it follows that the CELT model performed best for all the test files. It reached the highest PSNR and therefore the lowest mean squared error and sounded indistinguishable to the originals. Both autoencoder configurations performed better than the Mel inversion algorithm.
In Figure 4.1, the MDCT spectrums of a reconstructed guitar sample are plotted for all the tested models, where the left spectrum shows the original.
The CELT spectrum looks very different to the original one. At high frequencies, it is apparent that the whole energy distributes on only a few bins. The ear does not perceive a difference because looking at one frequency band, the less dominant frequency contributions cannot be heard anyways and it is therefore unnecessary to restore their exact value.
The fourth and fifth spectrums are from the autoencoders. When training both dense layers, the model neglects the high frequencies and is focusing in the reconstruction of the lower frequencies. However if only the decoding layer is trained, the high frequencies are more pronounced, but this comes with the cost of less detail in the lower frequencies. Apparently, this is especially disadvantageous for the guitar and tone sample but not for the bird example, where the reconstruction of higher frequencies seem to be more important. This can be nicely seen in the example of the bird, where autoencoder 1 completely neglects the twitter (Figure 4.2). The spectrums of the tone and the violin can be viewed in Appendix B.

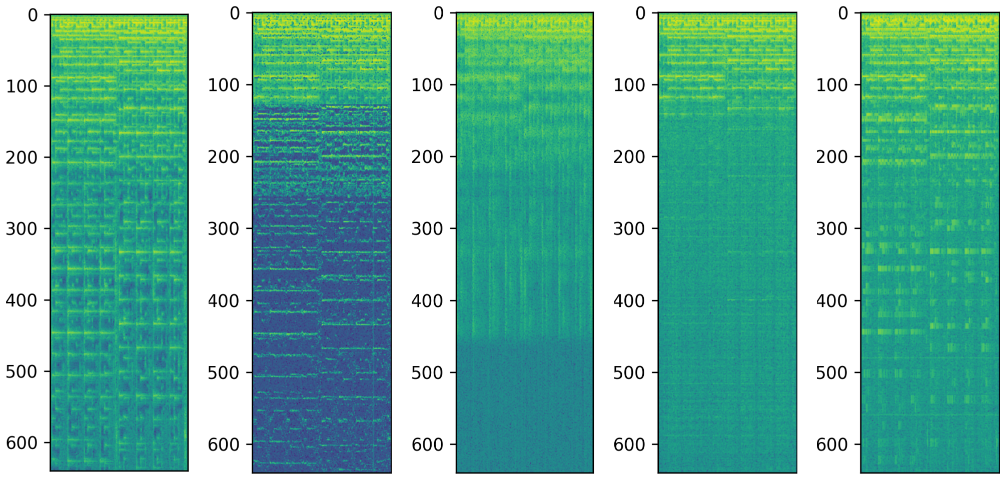

Figure 4.1: Reconstructed MDCT spectrums of a guitar test file. From left to right: original, CELT, Mel, autoencoder 1 (both layers trained), autoencoder 2 (only dense layer 2 trained)

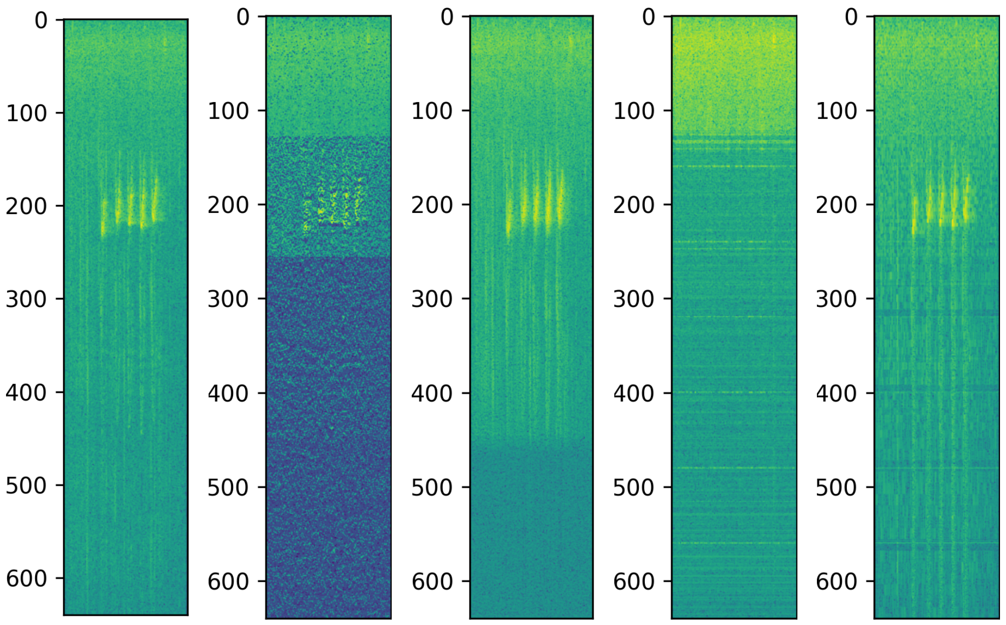

Figure 4.2: Reconstructed MDCT spectrums of a bird test file. From left to right: original, CELT, Mel, autoencoder 1 (both layers trained), autoencoder 2 (only dense layer 2 trained)

Chapter 5

Conclusion

From the results in Chapter 4, it can be concluded that using band energy as a front-end to the CNN is a good alternative to the Mel spectrogram. Although the accuracy is slightly lower, it is possible to recover the waveforms with a very low reconstruction error using the CELT scheme. During this thesis, classification was also tested on the coded vectors from the PVQ, but the CNN was not able to classify from only the coded vectors.
The autoencoders did not perform that well for recovering the MDCT spectrums. This could be due to the diverse dataset used, which includes very different types of data. The spectrum of a bird is very different to the spectrum of a guitar for example. Maybe the autoencoder would perform better if it would be trained only on similar files like a guitar and a violin.
If a low quality reconstruction of the waveforms is sufficient, the autoencoder which only trains the second layer is the most suitable for our purpose. For this model, the sensors only have to compute the modified discrete cosine transform and one matrix multiplication, and most of computation is moved to the server.
A future work could be to extend the neural network on the decoder to a more complex structure to get a better reconstruction of the original waveform. Furthermore, the classification accuracy using the reconstructed MDCT could be evaluated and optimized. Like that,the computationally expensive part of classification would be moved to the server while at the same time sending only a compressed version of the original waveform.

Appendix A

Comparison of Audio Compression Algorithms

As a comparison of quality and compression size, the results of the public multi-format listening test[1] which was carried out in July 2014 are used. In this test, the sound quality of Opus, AAC, Ogg Vorbis and MP3 was evaluated, whearas MP3 was allowed to use a higher bitrate compared to the others.

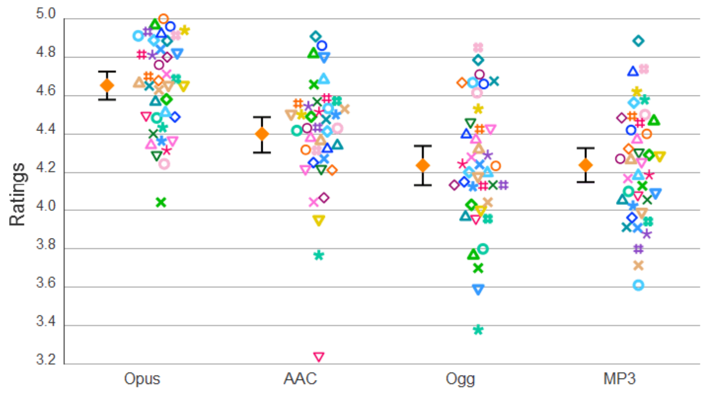

Figure A.1: Scores by tracks

Figure A.1 shows that Opus performs best in terms of quality. AAC follows on the second place and the quality of the tracks compared by Ogg and MP3 was similar in average.
In Figure A.2, the bitrate used by Opus, AAC and Ogg Vorbis and MP3 is shown. Opus, AAC and Ogg Vorbis used a smaller bitrate in average (104-107 kbps) compared to MP3 (136 kbps). Thus, a smaller compression size can be reached by Opus, Ogg Vorbis and AAC while still having the same or even better quality than MP3.

[1] http://listening-test.coresv.net/results.htm

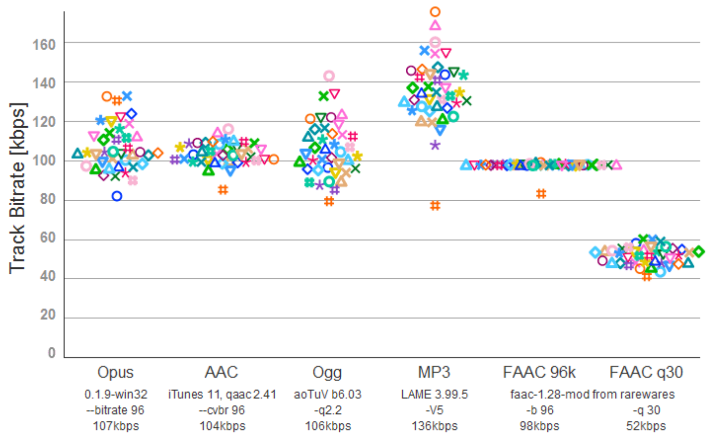

Figure A.2: Bitrate of tracks

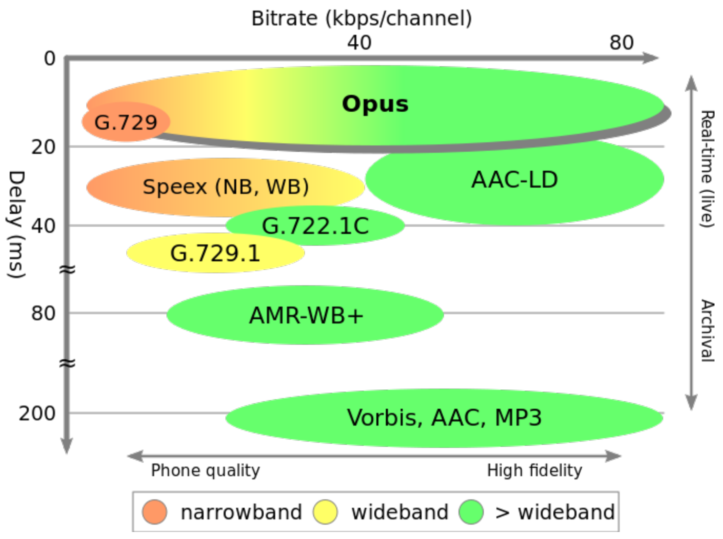

Figure A.3: Bitrate of tracks

Appendix B

Additional plots

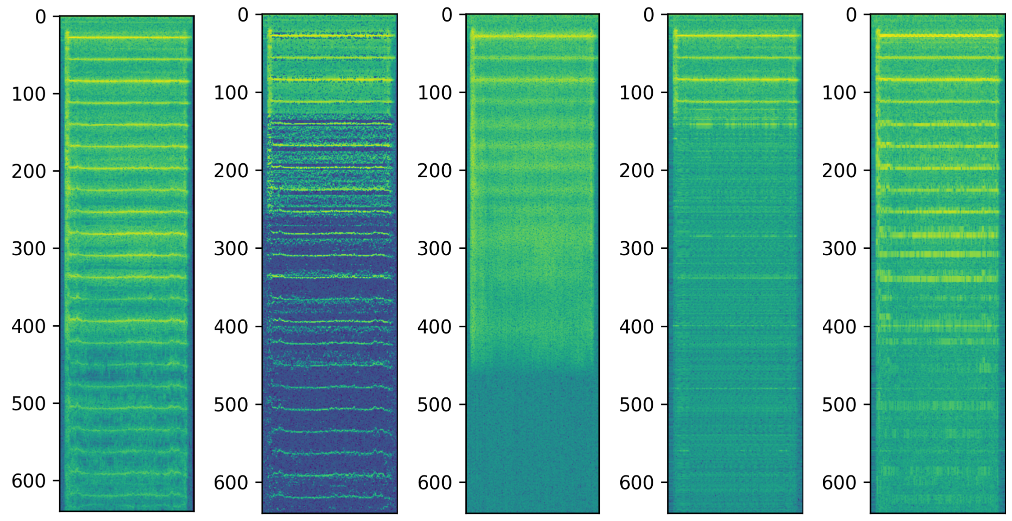

Figure B.1: Reconstructed MDCT spectrums of a violin test file. From left to right: original, CELT, Mel, autoencoder 1 (both layers trained), autoencoder 2 (only dense layer 2 trained)

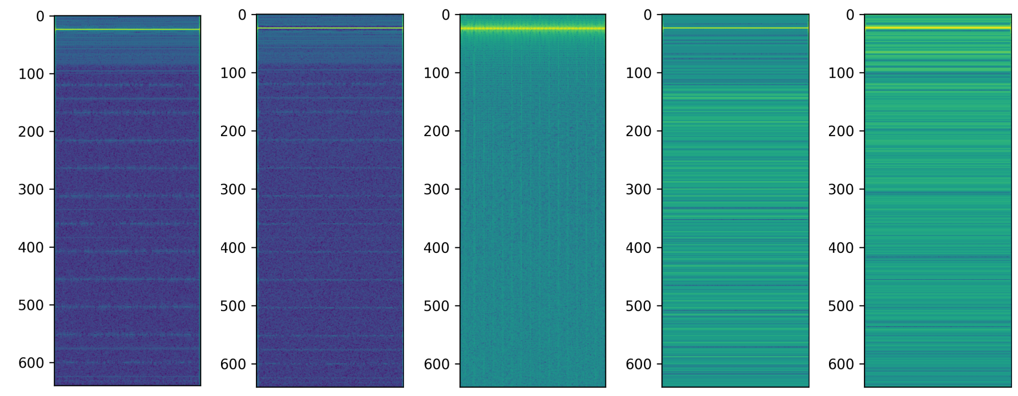

Figure B.2: Reconstructed MDCT spectrums of a tone test file. From left to right: original, CELT, Mel, autoencoder 1 (both layers trained), autoencoder 2 (only dense layer 2 trained)

Bibliography

[1] H. Lee, P. Pham, Y. Largman, and A. Y. Ng, “Unsupervised feature learning for audio classification using convolutional deep belief networks,” in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 1096–1104.

[2] L. Girard, J. Beutel, S. Gruber, J. Hunziker, R. Lim, and S. Weber, “A custom acoustic emission monitoring system for harsh environments: application to freezing-induced damage in alpine rock walls,” *Geoscientific Instrumentation, Methods and Data Systems*, vol. 1, no. 2, pp. 155–167, 2012. [Online]. Available: http://www.geosci-instrum-method-data-syst.net/1/155/2012/

[3] N. Takahashi, M. Gygli, B. Pfister, and L. V. Gool, “Deep convolutional neural networks and data augmentation for acoustic event detection,” *CoRR*, vol. abs/1604.07160, 2016. [Online]. Available: http://arxiv.org/abs/1604.07160

[4] M. Meyer, L. Cavigelli, and L. Thiele, “Efficient convolutional neural network for audio event detection.”

[5] “Results of the public multiformat listening test (july 2014),” http://listening-test.coresv.net/results.htm, accessed: 2017-03-21.

[6] J. Princen, A. Johnson, and A. Bradley, “Subband/transform coding using filter bank designs based on time domain aliasing cancellation,” in *ICASSP ’87. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 12, Apr 1987, pp. 2161–2164.

[7] J. Valin, G. Maxwell, T. B. Terriberry, and K. Vos, “High-quality, low-delay music coding in the opus codec,” *CoRR*, vol. abs/1602.04845, 2016. [Online]. Available: http://arxiv.org/abs/1602.04845

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[9] “Spectrograms, mfccs, and inversion in python,” https://timsainb.github.io/spectrograms-mfccs-and-inversion-in-python.html, accessed: 2010-09-30.

[10] D. Griffin and J. Lim, "Signal estimation from modified short-time fourier transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 2, pp. 236–243, Apr 1984.