Lund University
Lunds Tekniska Högskola
Computer Science Engineering

# Audio representation for environmental sound classification using convolutional neural networks

Linus Lexfors and Malte Johansson

Submitted in part fulfilment of the requirements for the degree of
Engineer of Computer Science of Lunds University, December 17, 2018

# Abstract

A convolutional neural network (CNN) training framework is described and implemented. The framework is used to train and evaluate an audio classification system, focused on evaluating differences in audio representation. The dataset used is ESC-50, containing 50 different classes of audio. We used SBCNN, a promising architecture suited for embedded systems because of its relatively small size. Several models are trained and evaluated. Linear spectrograms versus mel-scaled spectrograms are compared. Differences in FFT window size and overlap when constructing these spectrograms are evaluated. In addition, models trained on downsampled training data are compared to the models using the original sample rate. In our models, mel-scaled spectrograms outperformed linear spectrograms. The top performing model achieved a top-1 mean accuracy of 74.70%, using mel-scaled spectrograms and a 2048 sample FFT window with 75% overlap, compared linear spectrogram, which achieved a top-1 mean accuracy of 63.35%. The top model was further subjected to two different inference experiments; increasingly noisy data and mixed signals. We show that the model is relatively robust against wind-noise, the accuracy remains above 60% until the SNR between signal and wind-noise approaches 9 dB. The mixed signals test is hard to draw any strong conclusions from.

## Acknowledgements

We would like to thank our supervisors at LTH, Prof. Kalle Åström and Mikael Nilsson. We would also like to thank our deputy supervisor at Axis, Willie Betschart and our assistant supervisor, Andreas Irestål, for their input and ideas. Lastly, we are grateful to have spent our time during the thesis with the people at Axis Core Technologies Media/Graphics, who have given us great input and support.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Neural networks have become increasingly popular for use in commercial applications utilizing machine learning [16, p. 365–366], and a lot of research is being done in the field. Convolutional neural networks (CNN's) in particular have played a key role in the history of deep learning and gained a lot of momentum when *Alex Krizhevsky et al.* [20] won the ImageNet challenge in 2012 [25]. As embedded systems get specialized hardware, such as GPU's, and both computational power and memory size increases, neural networks increasingly become a viable and versatile tool for many different applications.

In the surveillance industry, specialized hardware and increased computational power is utilized in different machine learning applications such as face recognition and scene description. While large amounts of machine learning research is devoted to images and video for these purposes, research on audio classification outside of speech recognition has not been as prevalent. A robust audio classifying system has many applications, ranging from diagnosing illness via coughing analysis [12], to detecting important events in a surveillance system. In a real world surveillance context, robustness and sensitivity to noise would be especially important factors, as false positives would make using analytics of this kind unusable legally or too untrustworthy as aid to security firms.

## 1.1 Problem description

The goal of a audio classifier system is to be able to distinguish between different sources of audio. The system receives some representation of audio data, and outputs a likelihood distribution over all the classes it has been trained to classify. While the requirements on such a system could vary with the specific application, accuracy is always of utmost importance. In a surveillance application for example, the system needs to be robust enough so as to reduce the likelihood of incorrect detection or a false alarm to a minimum.

### 1.1.1 Data representation challenges

Contrasting image classification with audio classification, there are challenges that overlap between the two. One of them is the case of similar classes. If two classes are similar in the data representation used, any system would have more difficulty distinguishing them, compared to two classes that are dissimilar. Figure 1.1 illustrates the frequency content for a subset of the ESC-50 dataset, where it can be seen that many classes have quite similar content.

### 1.1.2 Distance to event in audio

One major difference between image and audio data is the effect that distance to the recorded event has. High frequency components of sound attenuate with distance at a higher rate than lower frequency components. This effect can alter the sound severely. Extra care needs to be taken when constructing datasets for audio, in order to encompass the scenarios of interest. It might be that an algorithm trained on indoor sounds close to the microphone performs badly on sounds recorded at longer ranges outdoors.

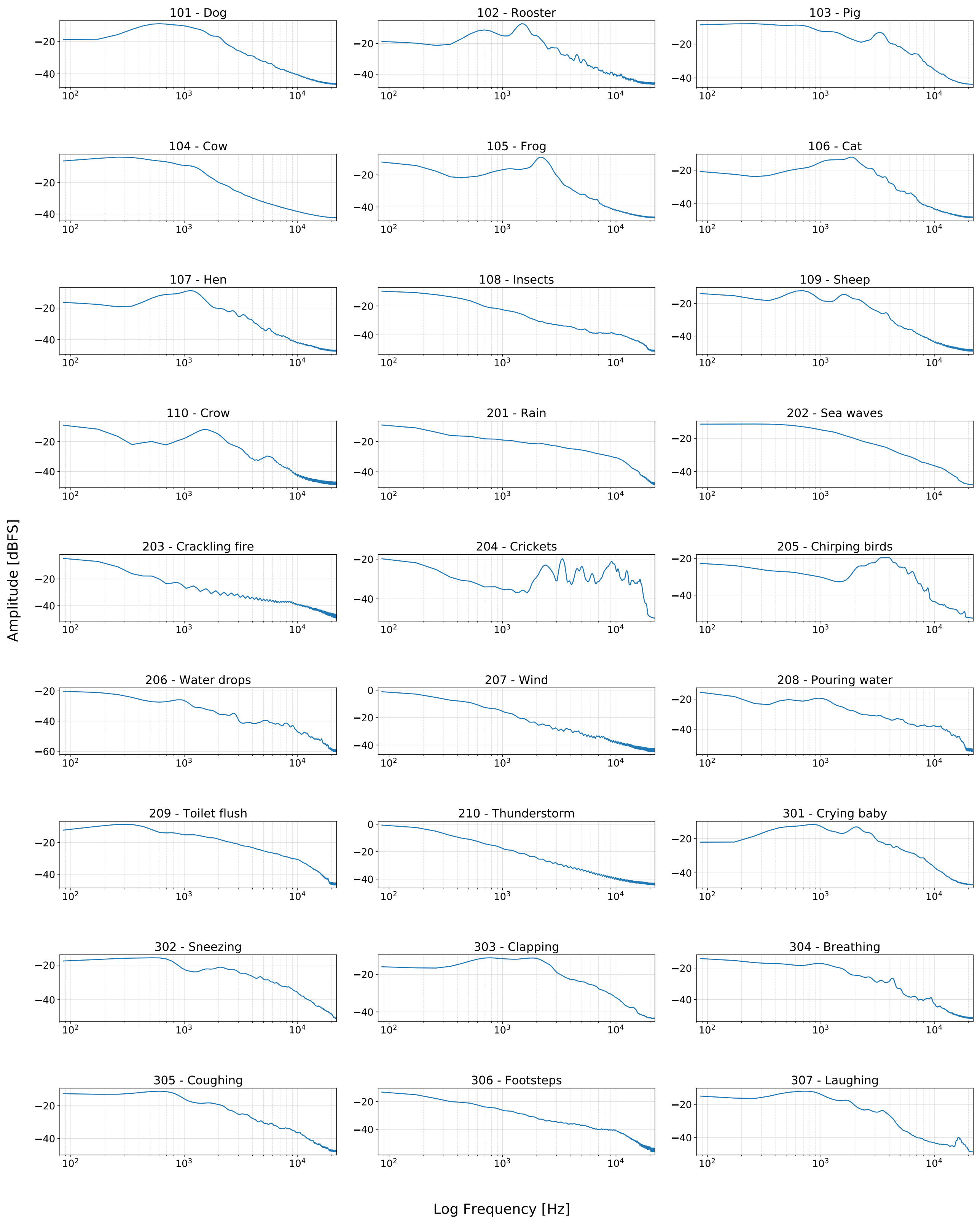


Figure 1.1: Class average spectral density estimation using Welch's method. 27 of the 50 classes in the ESC-50 dataset are featured. Many classes have similar frequency content, making separation with a simple method such as bandpass filtering unfeasible.

## 1.2 Motivation and objectives

This thesis will focus on evaluating convolutional neural networks as a method for audio classification. In particular, the thesis will have embedded platforms in mind when considering different architectures. We aim to implement our own training framework in Python, complete with input pipeline to support different data representations. In order have full control over design and implementation choices, the models we evaluate will be initialized and trained from the beginning, as opposed to using pretrained neural networks. Inventing our own network architectures is not in the scope of this thesis.

After our initial pre-study phase, more detailed thesis questions arose. As CNN's had their initial large scale success on image data [20], data representations such as spectrograms is our primary focus, as they resemble two dimensional images. In this representation, ordering matters, and stronger correlation between nearby data points is assumed, making it plausible as a fitting representation for a CNN.

We investigate choices in frame length and frame overlap in spectrogram construction, as well as normalization and choice of frequency scale. As will be discussed further in the theory section, the inherent trade-off between temporal resolution and frequency resolution is the motivating factor behind comparing various frame length and frame overlap setups. This serves as input to anyone considering embedded implementations of CNN's, as there may be limits in memory and computation resources, restricting the available choices. Another limiting factor is the sample rate of the audio input. Audio codecs in embedded systems might have restrictions in available sample rates, which affects the bandwidth directly. For this reason, we evaluate different sample rate cases and compare their performance.

### 1.2.1 Thesis questions

1. What kind of data input construction provides the best performance to a CNN?
    (a) How does linear frequency scaling compare to nonlinear frequency scaling in spectrograms?
    (b) How much does the balance between frequency resolution and temporal resolution affect performance?
    (c) How much does downsampling audio data to lower sample rates affect performance?
2. The audio signal will probably be attenuated, how much gain boost and normalization is possible to do before the classification performance is too poor?
3. Is it possible to separate two or more superimposed audio sources with sufficient performance?
4. How much does noise from a camera and/or noise from wind affect the accuracy?

## 1.3 Related work

CNN's have grown in popularity after AlexNet [20], won the ImageNet challange in 2012 [25], and its usage is now widespread. Piczak [22] demonstrated the potential of using CNN's for classification of environmental sounds. Using spectrograms with their respective deltas, data pitch-augmentation and dropout, the network's accuracy measured 64.5%. Sigtia et al. [28] showed that deep neural networks, compared to GMM and SVM, gives the best performance to cost ratio, for a range computational costs in embedded systems.
Huzaifah [19], found that Mel-STFT spectrograms performed slightly better than constant-Q transform, continuous wavelet transform, linear spectrogram and baseline MFCC. Mel-STFT spectrograms performed consistently well through the tested variations in model depth, convolution filter size and segment length. Further, Salomon and Bello [27] proposed a deep, high-capacity CNN which in conjunction with data augmentation gave, at the time, state-of-the-art performance on a dataset with 10 classes [7].

# Chapter 2

# Theory

## 2.1 Machine learning

In this section, we go through basic machine learning concepts, and focus specifically on the theory and techniques employed in convolutional neural networks for classification tasks. We start with defining what is known as *supervised classification*, and how it is related to other types of machine learning algorithms. We then move top-down into concepts specific to CNN's. In the last sections, we discuss optimization and regularization and introduce the most common approaches used in state of the art neural networks.

### 2.1.1 What is learning?

There are many different types of machine learning approaches, but what they all have in common is the notion of *learning*. Learning can be loosely defined as iteratively improving performance on a task based on experience, or examples [16, p. 97]. This data driven way of solving a problem stands in contrast to classical programming, where a set of rules is specified by a programmer for the computer to follow and execute.
The most categorical division between learning algorithms that can be made, is one of supervised versus unsupervised algorithms. Supervised algorithms are provided with labeled data,

whereas unsupervised algorithms work with unlabeled data. Another important distinction concerns what type of output the algorithm produces. Regression algorithms produce continuous quantities, and classification algorithms produce discrete values.

### 2.1.2 Supervised classification

The goal of a classification algorithm is to learn to distinguish and categorize input into discrete categories. The output of these algorithms are generally in the form of a score or probability distribution, spanning the categories concerned. The output is usually referred to as *class labels*. As mentioned, the distinction between *supervised* and *unsupervised* learning is important. Unsupervised learning algorithms work with unlabeled data, generally attempting to cluster or separate data based on the learned function. Supervised algorithms are provided with the desired output, or class labels, in order to update the model and reduce the classification error, continuing the learning process.
In summary, a supervised classification algorithm can be formally expressed as attempting to infer the function

$$f(\mathbf{x}, \mathbf{w}) = \hat{\mathbf{y}} \tag{2.1}$$

Where $\mathbf{x}$ is the input to be classified, $\mathbf{w}$ are the learned parameters of the function, and $\hat{\mathbf{y}}$ is the predicted class label. As stated above, the algorithm is additionally provided with the true class label $y$, in order to update the parameters $\mathbf{w}$. This happens in the optimization phase, which we discuss further in subsection 2.1.8.

### 2.1.3 The importance of representation

Supervised classification approaches attempts to learn patterns from observed data. For non-random data, we assume that there exists some underlying phenomena which could explain the observations. The goal of a classification algorithm is to find the patterns which best explains the variation in the data. If the patterns the algorithm finds and employs to distinguish between different categories generalize well, the model might be able to predict and explain new, unobserved data. This is the main goal of classification algorithms, to reduce what is known as the *generalization error*, the error on unseen data. How the data is represented is therefore crucial to the performance of the algorithm. Finding the proper representation can make or break the possibility of finding a good model. A good example of the importance of representation would be in training a model to be able to linearly separate two types of data points, see Figure 2.1 [16, p. 3–5].

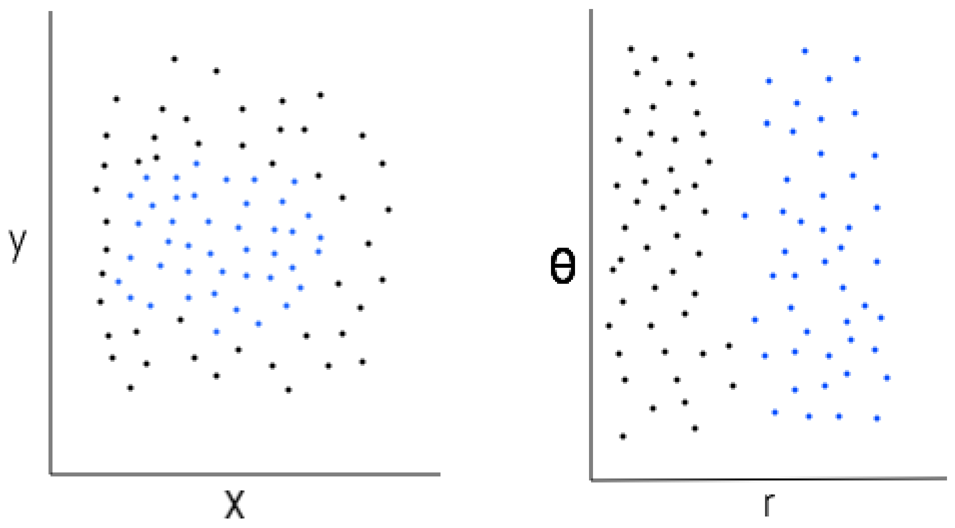


Figure 2.1: Two scatter plot sketches illustrating the same data. In the left plot, representing the data with cartesian coordinates, linearly separating the blue points from the black ones is impossible. With polar coordinates in the right plot, a vertical line can separate the data.

### 2.1.4 Handcrafted vs. learned features

A *feature* is any piece of information that the machine learning algorithm uses as part of the representation of the data. In some cases so called handcrafted features, i.e., features chosen by the system designer, can give good results. A commonly referenced dataset using handcrafted features is the *Iris flower dataset*, where four different features have been measured in order to distinguish between three types of Iris flower, one of them being the petal width [3]. In this type of setting, the algorithm models a mapping from representation to output; the features for it to use are already defined.

Although this approach can be effective for some tasks, where the space of possible observations is relatively small, it can be very time consuming and difficult for complex tasks, such as image or audio recognition. As an example, consider choosing a set of features that accurately describe a face. In terms of pixel data, this requires capturing an immense amount of variation of angles, shadows and light settings. A solution to this problem is to let the algorithm learn the features as well, with what is known as *representation learning* [16, p. 4]. Deep neural networks does this in a layered, hierarchical manner, which is one of the reasons why they have proven so successful in image recognition. These networks are able to learn higher level, abstract representations expressed as combinations of lower level representations. These layers are chained together so that an initial, low level layer might learn to represent and find edges. These edge representations gets fed to the next layer where they are combined to represent corners and contours. At the end of several layers, features representing eyes and mouths might emerge.

### 2.1.5 Convolutional neural networks

In this section, the most common layer types and operations that are applied in a convolutional neural network will be defined and explained. We use the case of a two dimensional matrix input when defining the different operations, but note that a third dimension can be introduced to the input making it a 3D tensor (representing color channels in an RGB image, for example). In the case of convolution this would imply that the filter kernel introduced in the next section is a 3D tensor as well, spanning the extra dimension. See Figure 2.2 for a overview illustration of a typical architecture.

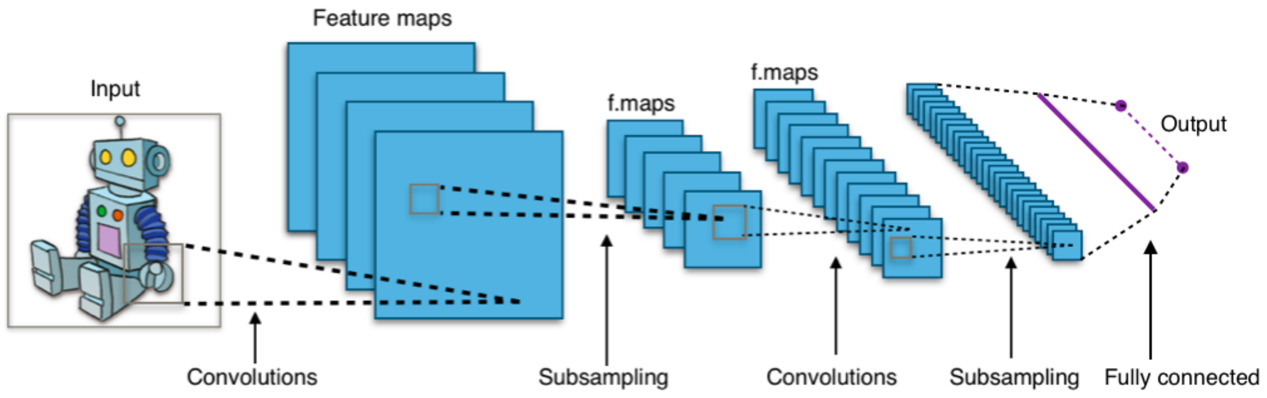


Figure 2.2: A typical CNN structure. The output of one layer is used as input to the next. Retrieved from `https://commons.wikimedia.org/wiki/File:Typical_cnn.png`

#### 2.1.5.1 The convolutional layer

A convolutional layer performs filtering of the input data, producing what is commonly referred to as an activation (or feature) map. The filter itself, the filter *kernel*, is represented as a matrix $\boldsymbol{W}$ of size $w \times h$. The filter kernel is usually quadratic, so that $w = h$, but can also have other dimensions depending on how the surrounding pixels correlate. Most machine-learning systems implement cross-correlation instead of true convolution, the difference being that the filter-kernel is not "flipped" with respect to the input dimensions. This filter is swept across the input, calculating the entry wise product sum $\boldsymbol{W}_{i,j}x_{i,j}$ for each window $x_{i,j}$ of the input, where the input window is of the same size as the filter kernel. The scalar output of one such operation is recorded in the activation map, the weighted sum of neighboring entries in the input window $x_{i,j}$.

A bias term is commonly added to the output as well, so that the final operation can be summarized as:

$$y = \mathbf{W} * x + b \tag{2.2}$$

where (*) is the discrete convolution operation. Note that the bias is a learned parameter as well, and commonly one bias per filter is employed.

The *stride* $s$ defines how many columns the filter kernel is shifted in each step of the convolution, as well as how many rows once the filter reaches the end of the first input dimension.

*Padding* of the input is sometimes employed in order to ensure a certain output size. This is commonly done by adding zeros along the edges of the input in order to increase both the width and height by $p$. See Figure 2.3 and Figure 2.4 for examples of different convolution settings and their resulting outputs.

In summary, the following relationship (assuming a quadratic kernel so that $k = w = h$) defines the output size of a 2D convolution [13]:

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1 \tag{2.3}$$

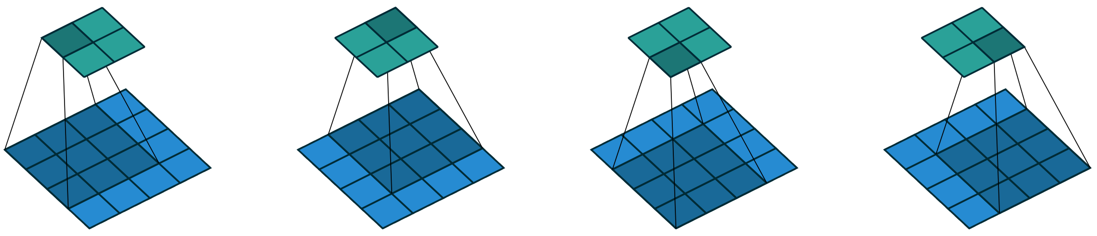


Figure 2.3: Convolution with kernel size $k = 3$, input size $i = 4$, no padding $p = 0$ and unit strides $s = (1, 1)$. Blue pixels are input, green pixels are the output feature map. Reproduced from Dumoulin and Visin under license [13].

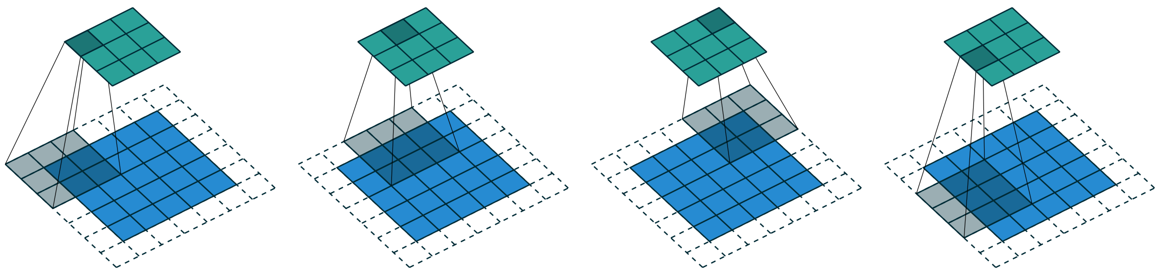


Figure 2.4: Strided convolution with $s = (2, 2)$. The input has been padded with $p = 1$. Reproduced from Dumoulin and Visin under license [13].

In practice, several filters are used in one layer, is this case producing an activation/feature *volume* of depth $d$, where $d$ is the number of filters applied to the input.

The convolutional approach assumes that data points in the input, that are closer together, are more strongly correlated, and vice versa. This is very important to consider when choosing how to construct the input to a CNN.

#### 2.1.5.2 The receptive field

An output pixel's *receptive field* is the number of input pixels its value depends on. In Figure 2.3, each of the green pixels see a $3 \times 3$ area, which is the receptive field for that output. Adding another convolution layer using the green pixels as input would linearly increase the receptive field of that layers output. Because an output pixel is unaffected by changes done outside its receptive field, it's a useful concept helping to understand what patterns are possible for a CNN model to learn [21].

#### 2.1.5.3 The pooling layer

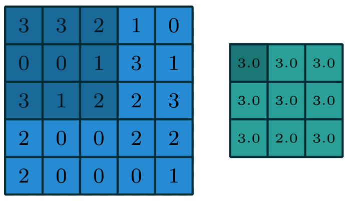


Figure 2.5: Max pooling the dark region in the blue input, results in the dark pixel in the green output. Reproduced from Dumoulin and Visin under license [13].

Pooling layers perform subsampling on the input data, replacing a region of activations from the previous layer with a summary statistic of those outputs [16, p. 335–337]. This is useful for computational efficiency, but in addition it has another property which can improve a convolutional networks robustness; the output of the pooling operation is invariant to small translations of the input data. In other words, if a convolution layer is followed by a pooling layer, a learned feature will still get picked up by the network even if it is slightly translated in the input. Since the exact location of a feature is not commonly as important as the presence of the feature in the first place, pooling is widely used in CNN architectures. A common type of pooling is max-pooling, where the maximum value in a window is taken as the output representing that region. As in the convolution case, a window of size $w \times h$ is swept over the input, producing the output. See Figure 2.5 for an illustration of max pooling.

#### 2.1.5.4 The fully connected layer

The last layers in a convolutional network are commonly referred to as *fully connected* layers. They provide the network with the capability to learn non-linear combinations of the features learned earlier in the network. They are fully connected or dense, in the sense that no parameter sharing takes place. Every parameter in the fully connected layer interacts with its own part of the input, in contrast with convolutional layers, where a parameter in one kernel filter interacts with many different data points in the input. They also usually serve to condense the output of the network into a final vector of the same size as the number of classes the model can distinguish between. This operation can be expressed as:

$$y = WX + b \tag{2.4}$$

As in the convolution layer, a learned bias term is added to the result of the matrix multiplication. The so called hidden layer represents one matrix multiplication, and yet another one is employed to compute the final output. See Figure 2.6 for an illustration of the structure.

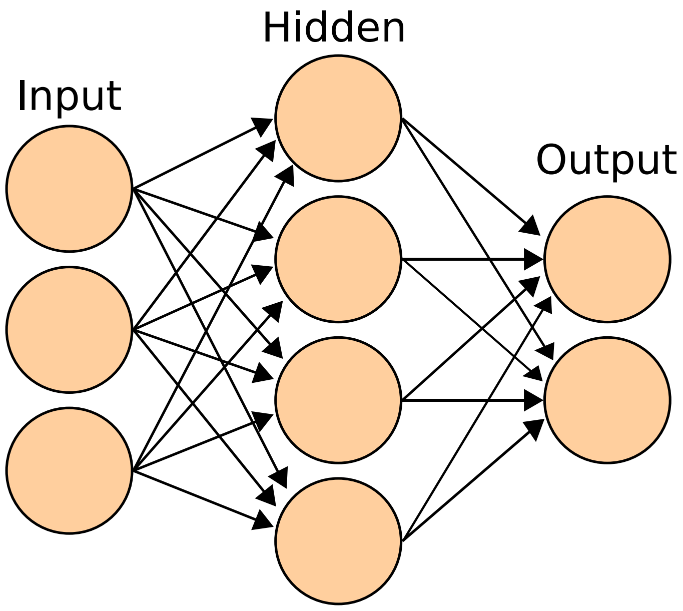


Figure 2.6: Illustration of the fully connected structure. Edges represent learned parameters which are multiplied with the input and summed together at the nodes. Retrieved from `https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg`

### 2.1.6 Rectified Linear Units

In order to be able to model complex relationships, it is not enough to simply composite several linear transformations on top of each other, since they then collapse into a single layer. Since convolution is a linear operation, and the matrix multiplication in a fully connected layer is also linear (the bias terms can also be included by adding a bias dimension of ones to the input), we need to add nonlinearities to our models. By applying an *activation function* to each output of a layer, we avoid collapsing the model and hence make it much more powerful.
As of spring 2018, the most widely used activation function is the *rectified linear unit* (ReLU) [24]. See Figure 2.7 for an illustration. It is defined as:

$$f(x) = max(0, x) \tag{2.5}$$

ReLU's have some advantageous properties, such as efficient computation and efficient gradient propagation.

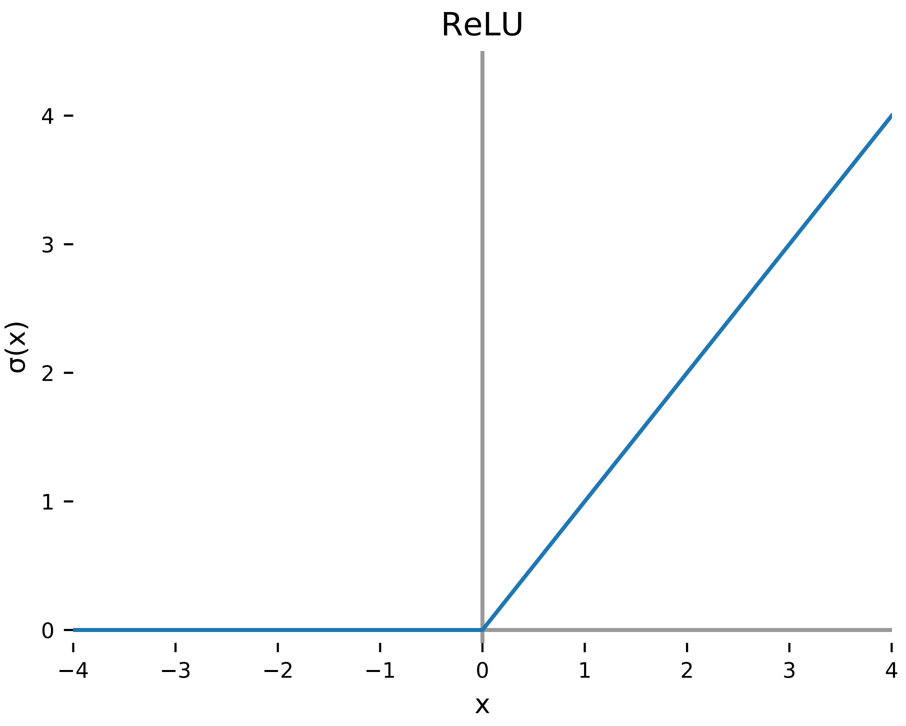


Figure 2.7: Plot of a rectified linear unit.

### 2.1.7 Parameter initialization

All trainable parameters need to be initialized in some manner. Bias variables are commonly initialized to zero, since they express shifting of a function this means an initial shift of zero. Glorot & Bengio [15], suggests the following distribution for initialization of other layer-parameters $W_{ij}$:

$$W_{i,j} \sim U[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}] \tag{2.6}$$

where $n$ is the number of columns/rows in $W$ and $U$ is the uniform distribution.

### 2.1.8 Optimization

Finding the optimal parameter settings for a given model, requires a quantitative definition of the *error* of each predicted class during training. This is achieved by defining a loss function $L(\mathbf{x}, y)$, where $\mathbf{x}$ is the input to be classified, and y is the correct class label. Optimization is then done in terms of minimizing this function by adjusting the parameters of the model. There are many loss functions to choose from. We have used *cross entropy loss* since it is the most common and intuitive one for classification tasks.
As mentioned, the final output of a classifier network is a vector of size $K$, where $K$ is the number of categories to classify. The values in the output vector are referred to as *logits*, or unscaled log probabilities, and can be interpreted as class scores. A common approach is to normalize these values in some manner. The *softmax* function takes a vector $\mathbf{x}$ of arbitrary values and squashes them to a vector of values in the range [0,1] that sums to 1.
It is defined as such:

$$\sigma(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \quad for \;\; j = 1, ..., K \tag{2.7}$$

The values can now represent a probability distribution over $K$ different outcomes.

#### 2.1.8.1 Cross Entropy Loss

Cross entropy can be used to measure the distance between two probability vectors:

$$D(S, L) = -\sum_{j=1}^{K} L_j \log(S_j) \tag{2.8}$$

where $S$ is the softmax normalized output vector of the model, and $L$ is a *one-hot encoded* vector representing the true class of the current input. Both vectors are of size $K$, the total number of classes. One hot encoded means that the vector representing the true class distribution has exactly one entry with the value 1, the rest being zero. The vector indices $j$ correspond to the class labels.

#### 2.1.8.2 Stochastic gradient descent

Gradient descent, or steepest descent, is the bread and butter of most neural network optimization. After a loss value has been computed from input flowing through the network, the parameters of the network need to be updated properly so as to minimize the loss value. Since the number of parameters in a network can be in the order of millions [20], an exhaustive, brute force approach is intractable. This is commonly referred to as the *curse of dimensionality.*
Stochastic gradient descent works by numerically approximating the gradient of the loss function with respect to the model parameters, and then taking a small step in the negative direction of the gradient [16, p. 83–84]:

$$\mathbf{x}' = \mathbf{x} - \epsilon \bigtriangledown_x f(\mathbf{x}) \tag{2.9}$$

where $\epsilon$ is commonly referred to as the *learning rate*, a hyper parameter that the designer of the network chooses.
In training CNN models, stochastic gradient descent is commonly done with *batches* of training examples. This simply means that a gradient approximation is done using an average of the gradient from several examples. This approach tends to speed up computation and provide a less varying gradient between iterations, making the training more consistent [16, p. 274–276].

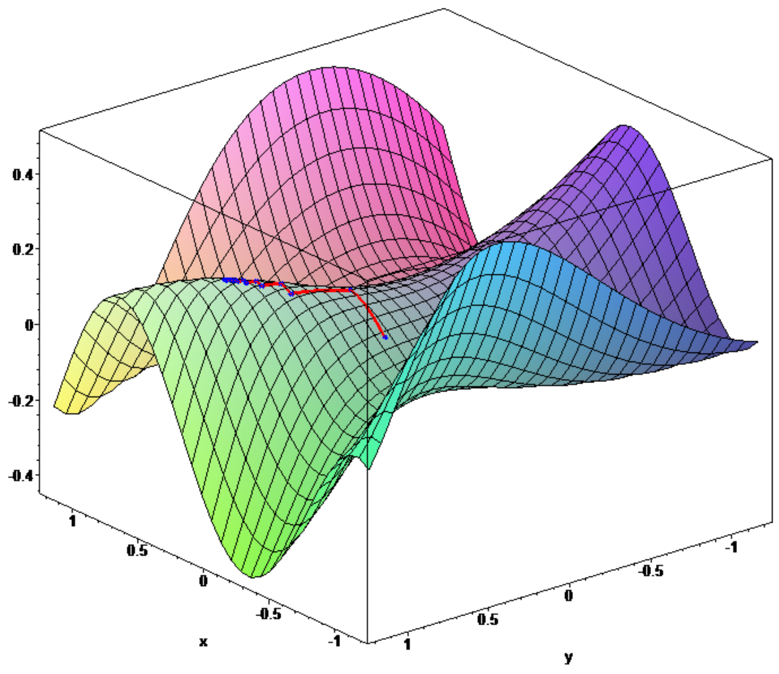


Figure 2.8: An illustration of the gradient descent algorithm. The red lines represent an update step of the parameters in the model.

## 2.1.9 Regularization

Overfitting occurs when the gap between the *training error*, i.e., the performance on training data, and the *generalization error* (the performance on unseen test data) is too large. If the hypothesis space of available functions is unrestricted, the algorithm could possibly keep reducing the training error towards zero, while the generalization error increases. Regularization is a technique employed to express a preference against certain types of functions [16, p. 116–117]. Parameter regularization is done by adding a regularization term to the loss function:

$$L(\mathbf{x}, y, \mathbf{w}) = L(\mathbf{x}, y) + \lambda R(\mathbf{w}) \tag{2.10}$$

where $L(\mathbf{x}, y)$ is any loss function as defined earlier in this chapter, $\mathbf{w}$ are the weights of the model (or the weights in a certain layer) and $\lambda$ is a scaling penalty term chosen by the designer. A $\lambda$ of 0 imposes no regularization.

The most commonly used weight regularizer is the L2-norm (euclidian norm):

$$R(\mathbf{w}) = \sqrt{w_1^2 + ... + w_n^2} \tag{2.11}$$

Another possible weight regularizer is the L1-norm (”taxicab norm”):

$$R(\mathbf{w}) = |w_1 + ... + w_n| \tag{2.12}$$

Both of these regularizers penalize large weights, with slightly different trends. The L-2 norm tends towards dispersing the total weight mass across all the weights, while L-1 favors sparsity. In Figure 2.9 we see an example of overfitting, the line does not perfectly fit all the data, still it models the general trend in the data better than the more complex function.

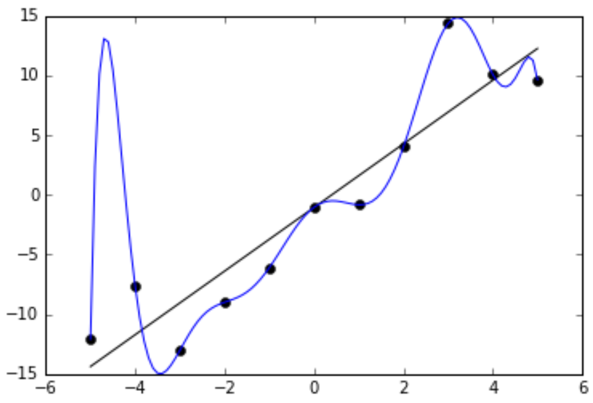


Figure 2.9: A typical example of overfitting. The simpler linear model generalize better than the more complex function that perfectly fits the points. Retrieved from `https://commons.wikimedia.org/wiki/File:Overfitted_Data.png`

### 2.1.9.1 Dropout

In 2012, a regularization technique known as dropout [18], was introduced, which yielded general improvements in benchmark tests and was used in Alexnet [20], to great success. Since then it has become a staple in neural network architectures. The idea is to randomly omit neurons in the network (generally 50% of them) for every batch or training case, during training. This has shown to reduce overfitting [18], by preventing neurons from becoming too dependent on neighboring neurons.

## 2.2 Audio representation

In subsection 2.1.3, the concept of data representation was introduced in general. An example was shown, illustrating how two different representations of the same data points could impact the results of an algorithm. In this section, we discuss various ways of representing audio data in particular. After a brief description of the perhaps most intuitive audio representation, the waveform, we shift to frequency domain representations. We focus on the *spectrogram*, which combines both time and frequency information. Different options and their trade offs in spectrogram creation are illustrated and discussed. We end with introducing the mel-scale, a way to re-scale frequencies that uses a model of human hearing.

### 2.2.1 The waveform

The waveform representation depicts how the amplitude of sample values vary over time. Representing audio as a waveform is not very complex nor computational expensive. However, it only contains two dimensions: amplitude and time. The general shape of the audio loudness can be observed in the waveform and some sound event periodicity of the sound. Waveform is a valuable representation in order to locate where sound events starts and ends, particularly how the sample values are distributed over time. However, trying to imaging what the audio sounds like from a waveform, such as in Figure 2.10, let alone to deduce what kind of event the audio represents is not trivial since its frequency contents is not explicit in this form.

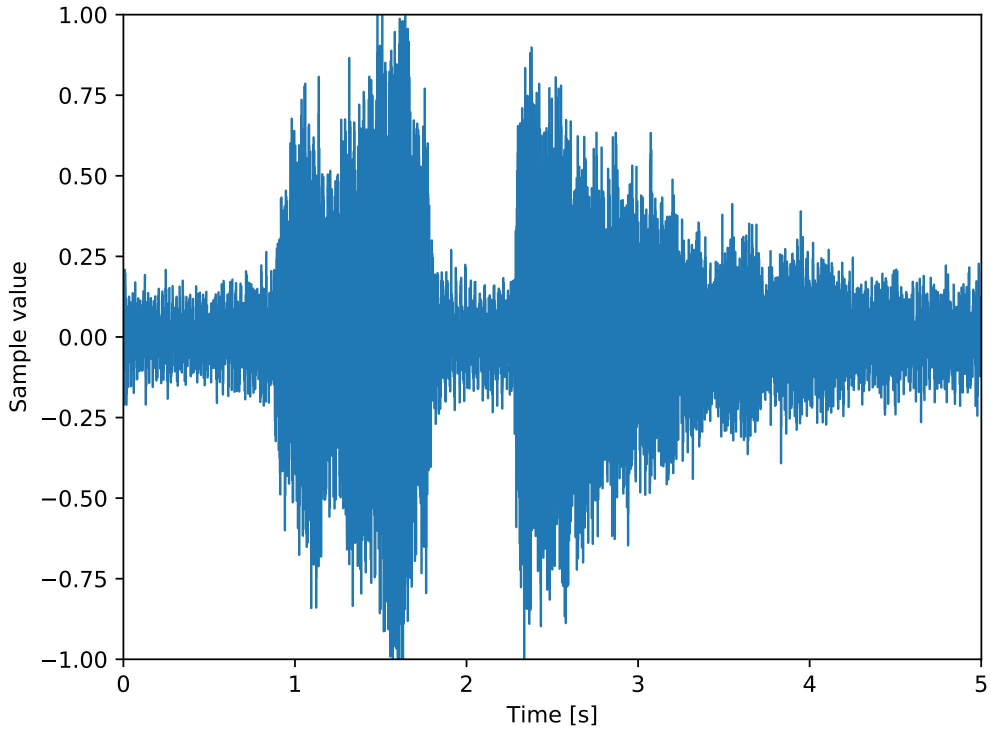


Figure 2.10: Waveform of a car horn. The sound event begins just before the first second and ends around 4.5 s. A pause of approximately 0.5 s can also be seen around 2 s. The leading and trailing parts are noise from the recording.

### 2.2.2 The spectrogram

The Fourier transform is a useful tool for spectral analysis of audio. It converts signals from the time-domain to the frequency domain, such that the magnitude and phase of each frequency component can be obtained. Fast Fourier Transform(FFT) is an optimized implementation of the Discrete Fourier Transform(DFT)[1] which can effectively be used for real-time analysis on discrete sequences, such as sample values in audio. In short, FFT correlates frequencies contained in the signal and bins them together in discrete steps. Since environmental audio contains momentary as well as stationary frequencies, the Short-Time Fourier Transform (STFT) is preferred as it depicts the spectrum changing over time. STFT uses a sliding FFT window to obtain a spectra for each segment in time of the original signal. The squared magnitude of each spectra is then stacked together to form a power spectrogram estimate. The result is a spectra

[1]http://www.nti-audio.com/en/functions/fast-fourier-transform-fft.aspx

changing over time, with frequency on one axis and time on the other. The values contained in this spectra describe the intensity of a certain frequency at a certain point in time. Logarithmic compression, namely decibels, of these values is common practice to balance regions of low and high energy, compare Figure 2.11 with Figure 2.12. Decibel is a relative scale and as such a reference value is needed, e.g. maximum loudness that can be encoded.

A spectrogram reveals interesting information about the audio. *Harmonics* (frequencies that are multiples of a fundamental frequency, the 1st harmonic), time variant events, periodic events and temporal localization of events can easily be identified from a spectrogram, some of which can be seen in Figure 2.12. As mentioned in subsection 2.1.5, convolutions assume tighter correlation between data points that are closer together in the input. This makes spectrograms especially apt as input to a CNN, since the convolutional filters can find patterns both in time and in frequency.

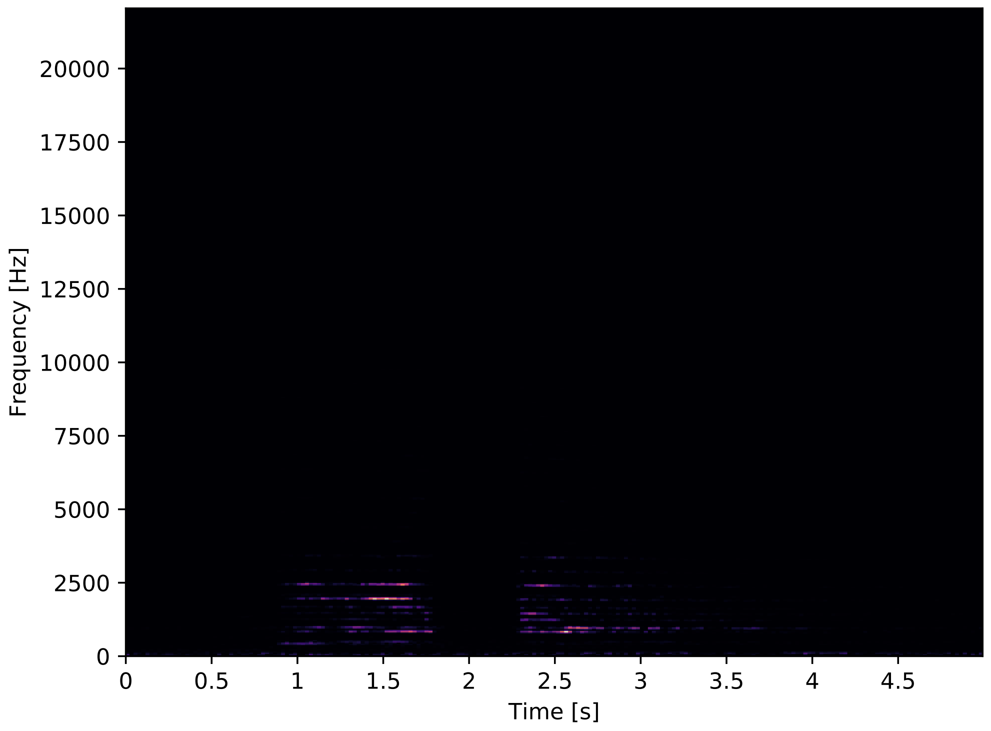


Figure 2.11: Uncompressed spectrogram of a car horn. 1024 $N_{fft}$, 0 overlap and 44.1 kHz sampling rate.

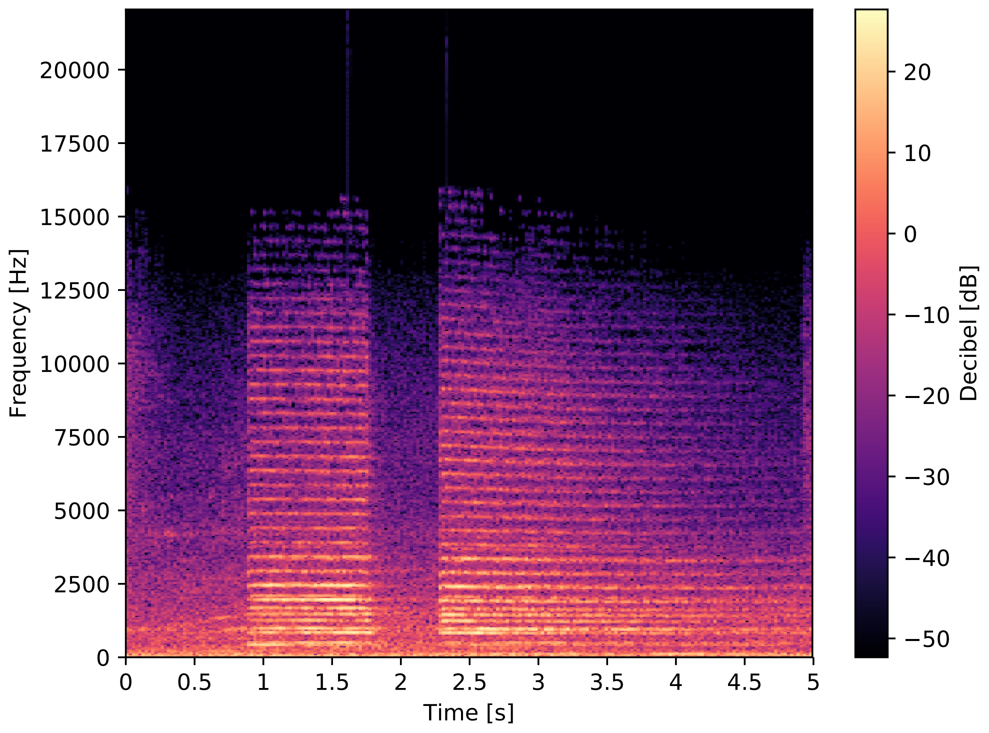


Figure 2.12: Spectrogram of a car horn. Decibels are calculated using the mean value of the spectrogram as reference. 1024 $N_{fft}$, 0 overlap and 44.1 kHz sampling rate. Location of events in time and frequency content are clearly visible, compared to Figure 2.10.

### 2.2.3 Window functions and spectral leakage

Frequencies that are not periodic within a selection of a finite set of samples, a *segment*, exhibits projection onto other frequencies in the basis set (the frequency bins) of the FFT. This occurs due to discontinuities of the signal's period at the boundaries of the segment in the periodic extension of that segment. This effect is called spectral leakage and appears in a spectrogram as blurring, i.e., leakage, around frequencies with high energy or as vertical lines. This is not to be confused with *aliasing*, which is caused by the nature of discrete sampling of continuous signals. Aliasing becomes apparent when the sampling rate is too low and new frequencies emerge, so called aliases. The amount of spectral leakage is affected by the sampling period, it is not, however, provoked by the sampling itself [17].

*Tapering* functions, also called *window* functions, are multiplicatively applied over a segment in the time-domain in order to minimize spectral leakage. There is a trade-off between frequency resolution and amplitude accuracy when using tapering functions. As the width of the window's main lobe narrows the ability to distinguish two closely spaced frequencies of similar strength increases. However, a narrower main lobe results in larger side lobes which increases the amount of spectral leakage. With a wider main lobe the ability to separate closely spaced frequencies decreases but less energy is leaked into other frequency bins. This relationship between the width of the main lobe and the height of the sidelobes can be seen in Figure 2.13.

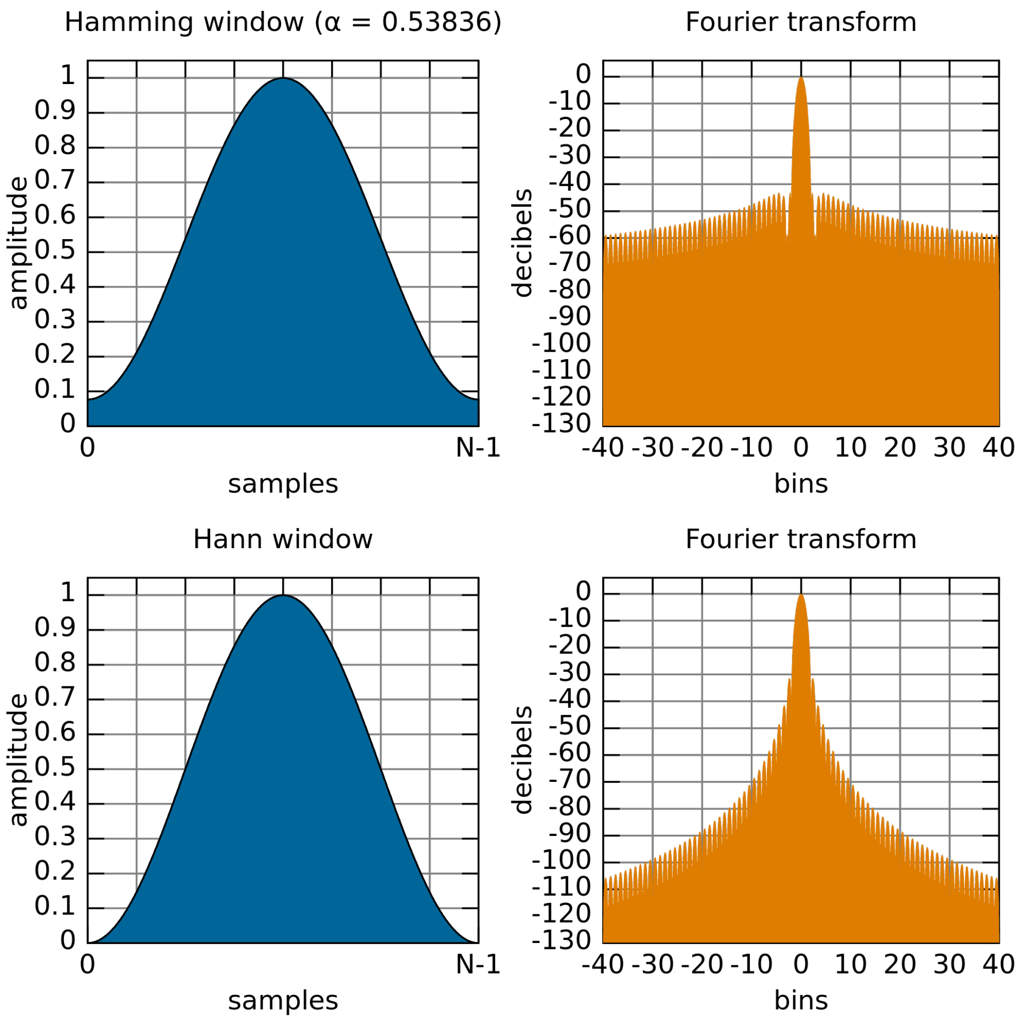


Figure 2.13: Window functions and their Fourier transform[2]. The Hamming window has a more narrow main lobe compared to Hann which correlates to the height and rolloff rate of the sidelobes.

[2]Created by Olli Niemitalo, taken from `https://commons.wikimedia.org/wiki/File:Window_function_and_frequency_response_-_Hamming_(alpha_%3D_0.53836).svg` and `https://commons.wikimedia.org/wiki/File:Window_function_and_frequency_response_-_Hann.svg` respectively.

#### 2.2.3.1 Choosing tapering function

Determining a good tapering function for the STFT always depends on the given signal and there is no right answer. However, there are some general guidelines for choosing a suitable window depending on the category of the signal. Because of the trade-offs discussed above, each tapering function has its advantages and disadvantages. Environmental sounds are composed of droning sounds, harmonics, momentary and non-stationary signals. Using Table 2.1 as a rough guideline, the Hann window is favorable in this case since it is good for unknown content, combinations of sine waves and narrowband random signals.

Table 2.1: Different types of signals and appropriate windows in order to distinguish the contents of a signal [10].

| **Signal Content** | **Window** |
|---|---:|
| Sine wave or combination of sine waves | Hann |
| Sine wave (amplitude accuracy is important) | Flat Top |
| Narrowband random signal (vibration data) | Hann |
| Broadband random (white noise) | Uniform |
| Closely spaced sine waves | Uniform, Hamming |
| Excitation signals (Hammer blow) | Force |
| Response signals | Exponential |
| Unknown content | Hann |

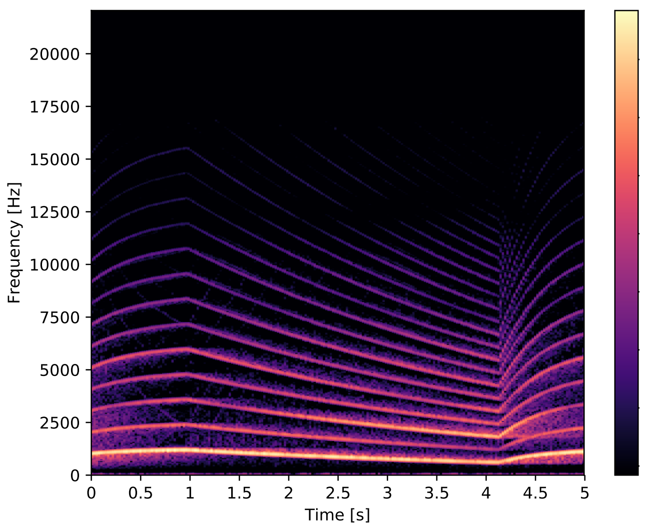
As seen in the Table 2.2, the Hann window have the highest side lobe level compared to both Hamming and Blackman-Harris, however it also has a higher side lobe falloff rate. This implies that the Hann window has more spectral leakage projecting onto frequencies close to the desired basis vector, but its leakage diminishes quickly further away from that basis. Comparing the Hamming window in the same fashion, it also has high side lobe levels but a much slower side lobe falloff rate, meaning that the spectral leakage continues its projection onto distant frequencies throughout the basis set. This is undesirable in the context of environmental sounds. These sounds can contain a range of interesting frequencies which spectral leakage will obscure.

Table 2.2: Comparison of window functions and their characteristics. Taken from [17], cross-validated with [2].

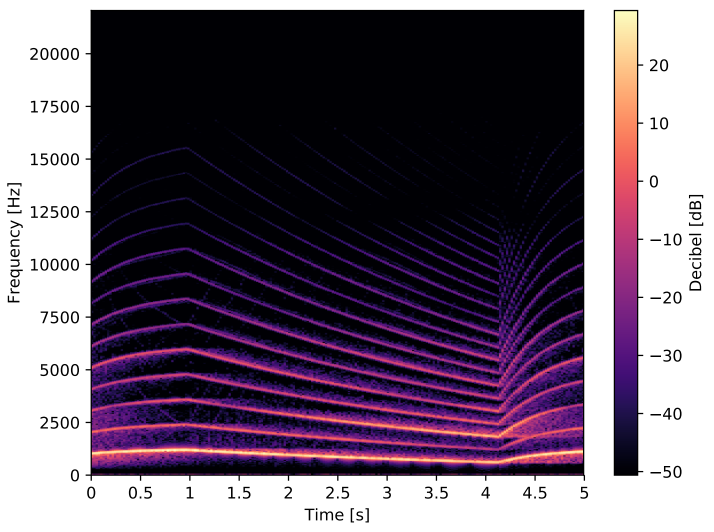
| **Window** | **-3 dB Main lobe width** (bins) | **-6 dB Main lobe width** (bins) | **Highest sidelobe level** (dB) | **Sidelobe rolloff** (dB/OCT) |
|---|---|---|---|---|
| Hanning ($\alpha = 2.0$) | 1.44 | 2.00 | -32 | -18 |
| Hamming | 1.30 | 1.81 | -43 | -6 |
| Blackman | 1.68 | 2.35 | -58 | -18 |
| Minimum 4-term Blackman-Harris | 1.90 | 2.72 | -92 | -6 |
| 4-sample Kaiser Bessel ($\alpha = 3.0$) | 1.74 | 2.44 | -69 | -6 |

Blackman-Harris is also a common window function, the minimum 4-term variant in case of Table 2.2, starts with a very low side lobe level hence compensating for the low falloff rate. Despite these good metrics, it has one disadvantage when applied to environmental sounds, which is the main lobe width. Compared to both Hann and Hamming, it has a much wider main lobe. Not many windows compare to the Hann window in terms of frequency resolution and limited spectral leakage range. This is probably why the Hann window is so commonly used in spectral analysis, particularly with unknown audio content.

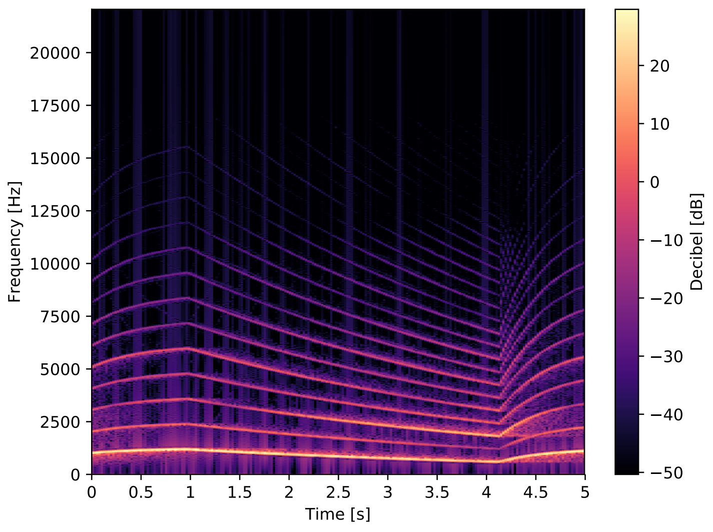
Figure 2.14 illustrates some of the effects of different window functions, where Hann and Blackman-Harris have much less spectral leakage appearing as vertical lines and blurring. The Hann window have slightly more concentrated lines which is best seen in the base frequency of the harmonics. This correlates to the width of the main lobe.



(a) Blackman-Harris window



(b) Hann window



(c) Hamming window

Figure 2.14: Spectrogram plots of a siren, using different tapering functions. All the plots are made using 1024 FFT points with same sized windows, no overlap, 44.1 kHz sampling rate. Decibels are calculated using the mean value of the spectrogram as reference.

### 2.2.4 Specifics of STFT

In addition to choosing a tapering function, there are more parameters and trade-offs. One trade-off is between frequency and time resolution. The *Gabor limit* states that a function and its Fourier transform cannot be limited in both time and frequency. This limits the frequency resolution of the FFT since the recorded signal is limited in time. Figure 2.15 illustrates this trade-off.

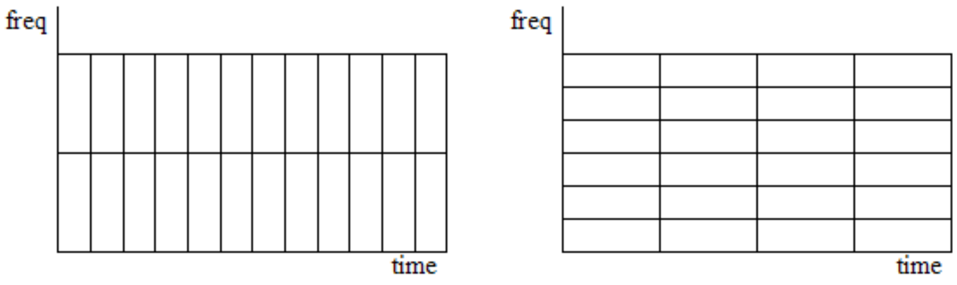


Figure 2.15: Simple illustration of the Gabor limit showing the trade-off between time and frequency resolution.

Segment length, also called FFT points or $N_{fft}$, plays an important role for the resolution of the spectrogram due to the Gabor limit. The segment length is inverse proportional to the frequency resolution of the FFT, as seen in Equation 2.13, and relates to the ability to resolve closely spaced frequencies at a given sampling rate[3]. Time resolution is important for resolving temporal spacing between different events. Table 2.3 shows frequency resolution corresponding to some common numbers for $N_{fft}$, given a sampling rate. One way to increase the frequency resolution is to increase the sample frequency. Another way is to keep $N_{fft}$ constant, but lowering the sample rate. This might be undesirable, since the Nyquist-Shannon sampling theorem would impose a lower max frequency that can be resolved. Ultimately, the sampling rate should be chosen to accommodate the maximum occurring frequency in accordance to the Nyquist frequency, Equation 2.14, and any frequency resolution requirements.

$$\Delta R_{fft} = \frac{f_s}{N_{fft}} \tag{2.13}$$

[3]http://www.bitweenie.com/listings/fft-zero-padding/

$$f_s \geq 2f_c \tag{2.14}$$

Table 2.3: Common power of two values of $N_{fft}$ and corresponding frequency resolution for three different sampling rates. As expected from Equation 2.13, bigger segment length or lower sampling rate results in higher frequency resolution

| $\mathbf{N_{fft}}$ | $\mathbf{\Delta R_{N_{fft}}}$ **[Hz]** | | |
|---|---|---|---|
| | **44.1 kHz** | **32 kHz** | **16 kHz** |
| **128** | 344.53 | 250.0 | 125 |
| **256** | 172.27 | 125 | 62.5 |
| **512** | 86.13 | 62.5 | 31.25 |
| **1024** | 43.07 | 31.25 | 15.63 |
| **2048** | 21.53 | 15.63 | 7.81 |
| **4096** | 10.77 | 7.81 | 3.91 |

Just as the Nyquist-Shannon sampling theorem limits the maximum frequency to be analyzed without aliasing, there are limits to the smallest spacing between two frequencies that can be resolved in a spectrogram when analyzing a finite sequence. This spacing depends on the segment duration as per Equation 2.15, where $t$ is the duration of segment. Overlapping the segments increases the smoothness of the spectrogram. It also allows for separation of events shorter than the segment length. With no overlap, events that are shorter than the duration of the segment can appear to happen simultaneously [9].

$$\Delta R_w = \frac{1}{t} \tag{2.15}$$

### 2.2.5 Mel-scaled spectrogram

Human hearing is not perfect and the perception of pitch is one of its imperfections. At higher frequencies, progressively larger intervals are perceived to produce the same increment in pitch.

The mel-scale is a perceptual scale modeling this phenomena. Empirical measurements results in different mel-scales which tends to be of a linear nature up to a certain breakpoint around 1000 Hz, after which the curve tends to a logarithmic nature. Each version may differ in where the breakpoint is and the shape of the curve, see Figure 2.16 for an example of such a curve. However, most mel-scales are defined such that 1000 mels is exactly 1000 Hz. See Equation 2.16 and Equation 2.17 for one example of how to compute mel values, and its inverse. Values of Equation 2.16 is illustrated in Figure 2.16. Using the mel-scale for spectrograms increases the area of lower frequency patterns in the figure while diminishing the area of higher frequencies, as can be seen in Figure 2.17 compared with Figure 2.12.

$$M(f) = 2595\log_{10}(1 + f/700) \tag{2.16}$$

$$M^{-1}(m) = 700(exp(\frac{m}{1125}) - 1)) \tag{2.17}$$

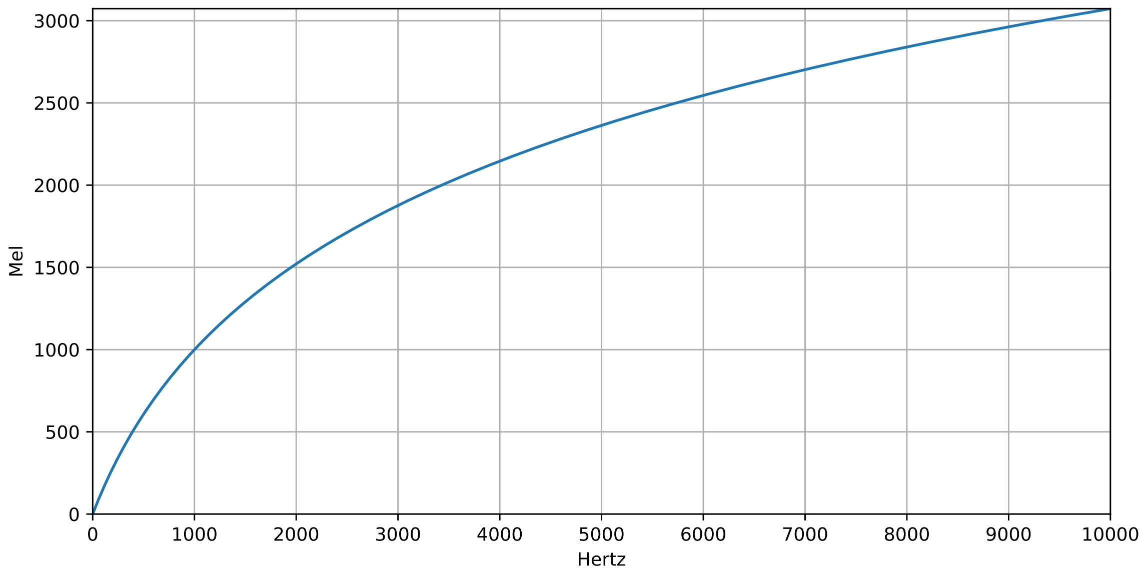


Figure 2.16: Mel values as per Equation 2.16

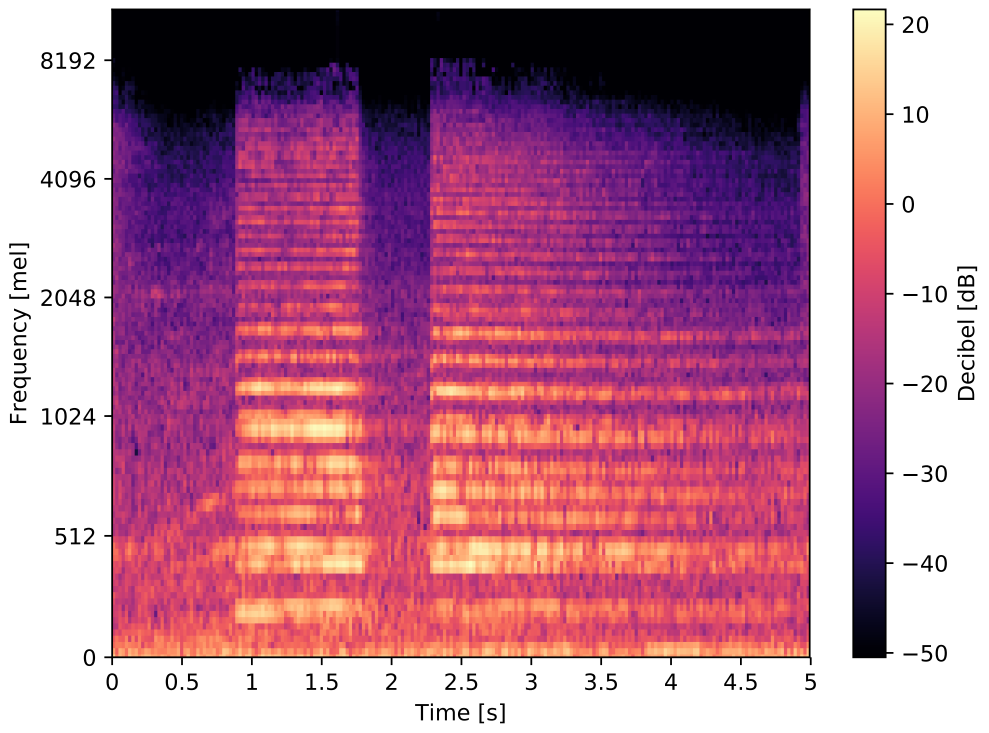


Figure 2.17: Mel-scaled spectrogram of a car horn, using 1024 points Hann window and a 1024 points FFT. Decibels are calculated using the mean value of the spectrogram as reference. Lower band frequencies take up more area and higher frequencies less, compare with 2.12

### 2.2.5.1 Mel filterbank

Filterbanks are used in signal processing for applying an array of band pass filters to a signal. To generate a mel filterbank the sampling rate, $N_{fft}$, and the desired number of mel bins is needed. First the minimum and maximum frequency is set, for example 0 Hz and $f_s/2$ respectively. These frequencies define the boundaries and are converted into mel using Equation 2.16. Creating, for example, 10 mel bins require 12 points on the mel-scale, therefore another 10 points are taken linearly spaced between the lower and upper mel frequencies. These mel frequencies are then converted back into Hertz using Equation 2.17 and rounded off to nearest frequency bin, using Equation 2.18 below:

$$f(i) = floor\left(\frac{(N_{fft}+1) * h(i)}{f_s}\right) \tag{2.18}$$

Where $h$ is the mel spaced frequencies in Hertz. The mel filterbank is then obtained using Equation 2.19, where $m$ is the number of filters. In short, the first filter starts at the first bin, the lower boundary, reaches its max at the second bin and stop at zero again at third bin. This is repeated for each filter, with the last filter ending at the last mel bin. Optionally, area normalization by dividing each filter with the width of its mel band can be used. A simple illustration of a normalized mel filterbank can be seen in 2.18.

$$H_m(k) = \begin{cases} 0 & , k < f(m-1) \\ \frac{k-f(m+1)}{f(m)-f(m-1)} & , f(m-1) \leq k \leq f(m) \\ \frac{f(m01)-k}{f(m+1)-f(m)} & , f(m) \leq k \leq) f(m+1) \\ 0 & , k > f(m+1) \end{cases} \tag{2.19}$$

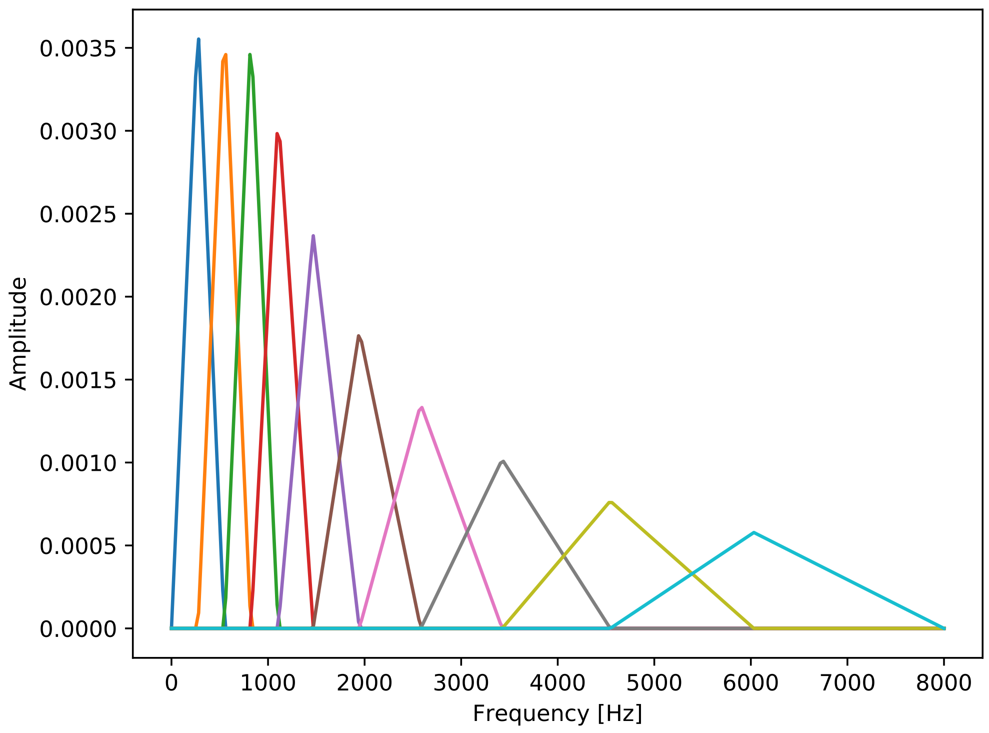


Figure 2.18: Mel filterbank illustration using mel values as per Equation 2.16, sampling frequency of 16000, 512 $N_{fft}$ to produce 10 mel bins. The filterbank is area normalized by dividing each filter with the width of the mel band.

## 2.3 Dataset

One of the key components for good performance in machine learning is good training and validation data. Deep neural networks, are generally more data hungry than other machine learning algorithms. Large amounts of data is however not enough, in order for a model to generalize well the data also needs to encompass some variance. One kind of variance in audio is the presence of different acoustic effects such as sound interference, echo and reverberation. These effects are likely present in datasets containing recordings from different scenes, as each scene most likely has a unique acoustic setting. Another type of variance is the diversity of audio sources. Capturing audio from multiple sources, each with their own characteristics, increases the variance in the dataset. This is logical, since we know from real world experience that, for example, different car engines have different sound. Although they have similar characteristics, they still sound different.

### 2.3.1 ESC-50

Environmental Sound Classification [23] is a labeled dataset containing 2000 short sound clips from 50 different classes. The classes roughly belong to five different categories: animals, natural soundscapes, human, non-speech sounds, interior/domestic sounds and exterior/urban sounds. The clips are circa 5 seconds long, and come arranged into 50 folders containing 40 clips each. The original recordings were gathered by the Freesound.org project [14]. The clips are prearranged into 5 folds based on the original recordings they were extracted from. A survey was conducted by crowd sourcing through CrowdFlower and tested human accuracy on the dataset, which measured 81.3%.

# Chapter 3

# Implementation and methodology

There are many available frameworks for implementing neural networks; Caffe [1], Torch [6] and Tensorflow [5] are commonly referenced in the machine learning literature. In this section, we go through our implementation of a training framework, how the input pipeline is constructed and the evaluation strategy used to compare different models. This is followed by a more detailed explanation of the CNN architecture evaluated and how the training input was preprocessed.

## 3.1 Tools and implementation

### 3.1.1 Tensorflow

We implemented a CNN training framework in TensorFlow, using their Python frontend inside a virtual environment known as *virtualenv* [8].
TensorFlow is a framework for implementing machine learning algorithms. The computation model is based on directed graphs, consisting of nodes which specify a certain operation [11]. A node can have zero or more inputs, and zero or more outputs. Data flows through graphs in the form of *tensors*. Tensors in TensorFlow are implemented as typed, multidimensional arrays [11], the shape of which may not be known until run time. Operations may be performed on CPU as well as GPU devices.

Design of a computation graph is done through defining operation nodes and connecting their input to the previous nodes output. An operation node is only executed when its output is required by any node it is connected to, deeper in the graph. See Figure 3.1 for an illustration of such a graph. The loss function and gradient descent update rules are implemented as operation nodes as well.

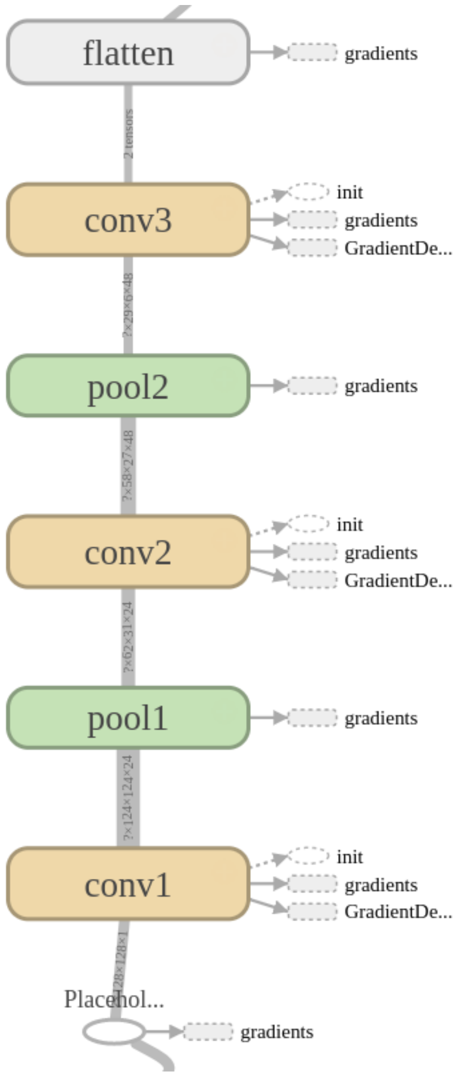


Figure 3.1: Excerpt from a computation graph, illustrated with TensorBoard, TensorFlow's visualization tool. Data flows from an input pipeline through the different layers. The first "conv1" node will only compute an output when the next node, "pool1", requires it.

### 3.1.2 Training and input framework

TensorFlow allows for operation nodes to be provided with data in a variety of ways [4]. For training data, we have followed the convention of specifying special input operation nodes at the beginning of the computation graph.
We preprocessed training data into TensorFlows proprietary format, TF-records, that are stored on disk. These records contain three fields; a training example, the correct label, and a filepath for debugging purposes. Only the first two fields are needed during training.
This allowed us to create various input pipelines depending on what type of input a specific model's architecture requires.
The input node points to a TF-record on disk, and describes a set of operations for reading out the serialized data from it. These include reshaping the data into the correct number of dimensions, buffering a number of training examples, shuffling and collecting a batch of the chosen size. The input node is connected to the first computation node in the network.

A complete training step can be summarized as such:

1. A batch of training examples and ground truth labels is read from the input pipeline.
2. The training data forward-propagates through the network.
3. Loss is calculated between ground truth labels and network output.
4. This loss or error is propagated backwards through the network, updating the weights via the chain-rule.

#### 3.1.2.1 Evaluation step

**Definition:** one *epoch* is defined as one full iteration through the training data.
After every epoch of training, the computation graph is fed data from an evaluation set, kept separate from the training set. It is crucial that the network only perform inference on this set, the parameters of the model must not be updated, as this would corrupt the model.

A second method to provide input to a operation node is through so called *feeding.* This method consists of providing a node with tensor input data directly though Python code. The evaluation set is preloaded into memory at startup. In order to prevent training on evaluation data, the evaluation routine is provided a handle to the softmax-node.
This node only depends on the network output; hence the loss node and gradient descent update node are never executed during this step.

### 3.1.3 Evaluation strategy

Our main goal was to compare the relative performance of a chosen network, when varying input construction and sample-rates. We use 5-fold cross validation to evaluate model performance, as it is a relatively robust way of evaluation, considering the relatively small size of the ESC-50 dataset. As mentioned before, the ESC-50 dataset is prearranged into 5 folds, making it a fitting approach for the dataset.
See Figure 3.2 for an illustration of k-fold cross validation. This method consists of training a model using 80%(four folds) of the data as training data, and the last 20%(one fold) as evaluation data. This is repeated a total of 5 times, with a different fold as the evaluation set every time. We then record the **mean** performance across all evaluation folds as representative for that model.
This method, while increasing training time, ensures that all the data in the dataset is used for evaluation exactly once, and avoids the risk of arbitrarily choosing a evaluation subset that happens to be non-representative (i.e., lucky or unlucky).
The variance in fourier transform frame length and overlap between the different models, changes the amount of TF-patches(explained in the following section) that comprise one example. Due to this, the size of one epoch varies between our different models. We therefore chose the number of epochs to train each model in beforehand, so as to ensure that the amount of parameter-updating steps were approximately the same for each model.

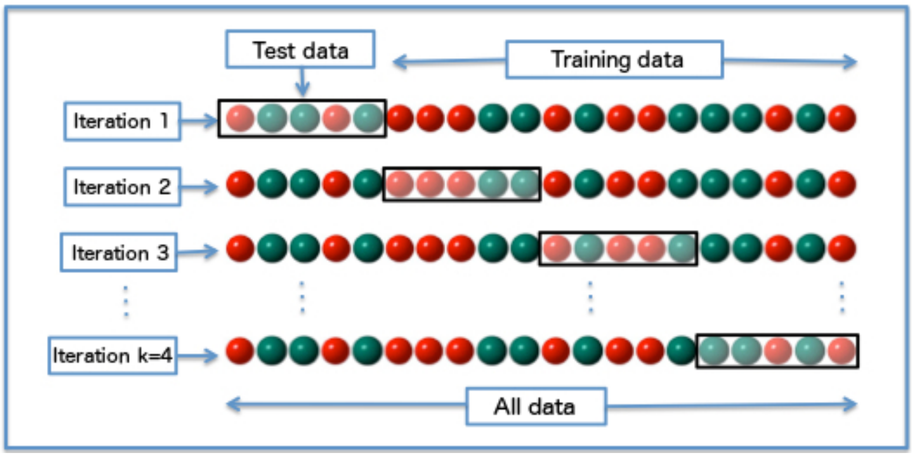


Figure 3.2: Illustration of K-fold cross-validation. Each fold is used as evaluation data exactly once, removing the chance of accidentally choosing a particular evaluation subset that may misrepresent model performance. Obtained from `https://commons.wikimedia.org/wiki/File:K-fold_cross_validation_EN.jpg`

### 3.1.4 Network architecture

#### 3.1.4.1 SB-CNN

The network proposed by Justin Salamon and Juan Pablo Bello, showed great results [27], on a smaller dataset of 10 classes [7]. Their architecture consists of three convolutional layers, interspersed with two max-pooling layers and two fully connected layers at the end, see Figure 3.3. Pooling more in time than in frequency in the max-pooling layers is an interesting design choice, as it prioritizes sensitivity to frequency content over temporal location. Another interesting aspect of this model is the input format which consists of slices excerpted from the mel-scaled spectrograms, so called TF-patches(Time-Frequency patches). The dimensions of these patches are $128 \times 128$, with 128 frequency bands and 128 steps in time. With a sampling rate of 44.1 kHz and using $1024 N_{fft}$ (the number of samples in one STFT window) with no overlap, this corresponds to approximately $3s$ per patch, see Figure 3.4. TF-patches are taken randomly in time from audio clips in the training set without repetition. It is a novel way to augment the dataset and an attempt to make the network less sensitive to where in a sound event a recording happens. Evaluation is done by taking all possible TF-patches for each audio file and averaging all the resulting accuracy measurements. The number of trainable parameters for SB-CNN is 244034. This is a small amount of parameters compared to many architectures, where the number can often be in the millions, like the venerated AlexNet [20]. In Tensorflow, weights are 32 bit floating point numbers, which results in a model size on disk of approximately 950 kilobytes.

#### Parameter initialization

All bias variables are initialized to zero. Other layer parameters are initialized from the distribution suggested by Glorot & Bengio [15], as per Equation 2.6 in the Theory section.

#### Parameter regularization

The parameters in the fully connected layers have L2-regularization applied to them, as per Equation 2.11 in the Theory section, with a penalty factor of 0.001 as in the original SB-CNN.

#### Learning rate

We use a static learning rate of 0.01, as in the original SB-CNN.

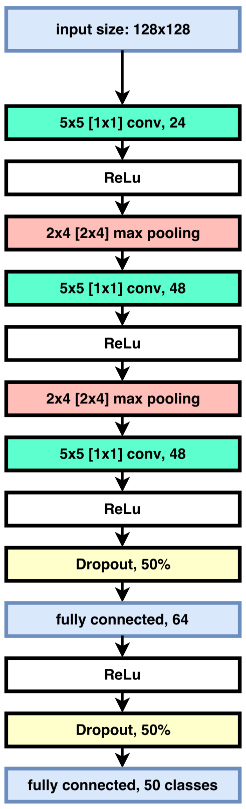


Figure 3.3: Overview of the layers in Salamon and Bellos network. The first numbers in a layer are the size of the convolution or pooling kernel, which is then followed by the stride in brackets. The last number is the amount of filters in a convolution layer, or the how many hidden units there are in a fully connected layer.

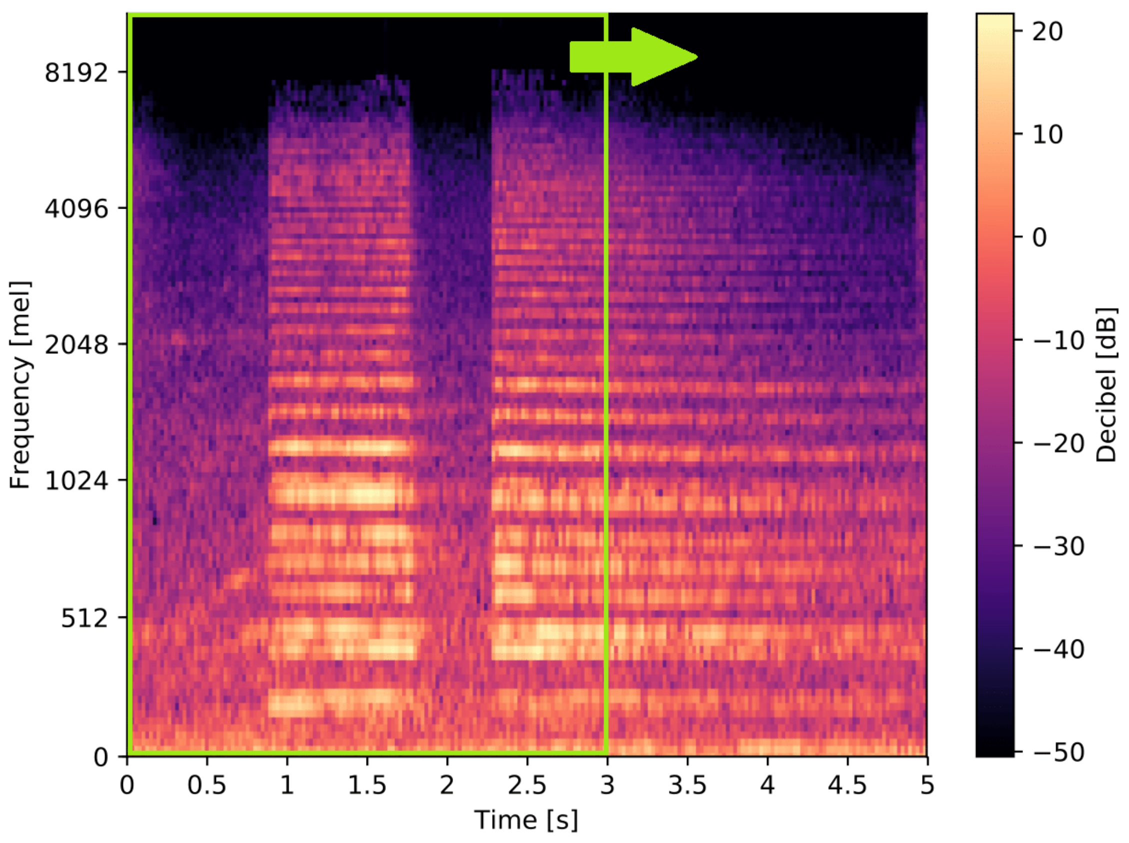


Figure 3.4: Illustration of a TF-patch. As the arrow indicates, all possible TF-patches are used throughout training. They are extracted from each member of the training set, shuffled and then stored in a TF-record. A batch of 100 random TF-patches are used in every training step.

### 3.1.5 Pre-processing of input data

Before data is written into TF-records it needs to be processed and converted into the required data representation and time-frequency dimensions. For these purposes, we wrote a Python script that reads through the whole database and processes it.

We use the SoundFile[1] library to read sample values from audio files. Soundfile utilize the NumPy[2] library which provides powerful N-dimensional array objects that we use throughout the script whenever possible. Some files in the datasets are in stereo but for our purposes we

[1]https://pypi.python.org/pypi/SoundFile/0.8.1
[2]http://www.numpy.org/

only need one audio channel. For those in stereo, we convert them to mono by taking the average over the two channels. For some trainings we downsampled the audio using a wrapper[3] for the AudioSegment class in Pydub[4], which utilized an anti-aliasing filter implemented in Sox[5]. The SciPy[6] library provides a flexible implementation of spectrogram creation using STFT, which we use to calculate all power spectrograms. We use Librosa for calculating the mel-scaled spectrograms by using the spectrogram output from SciPy.

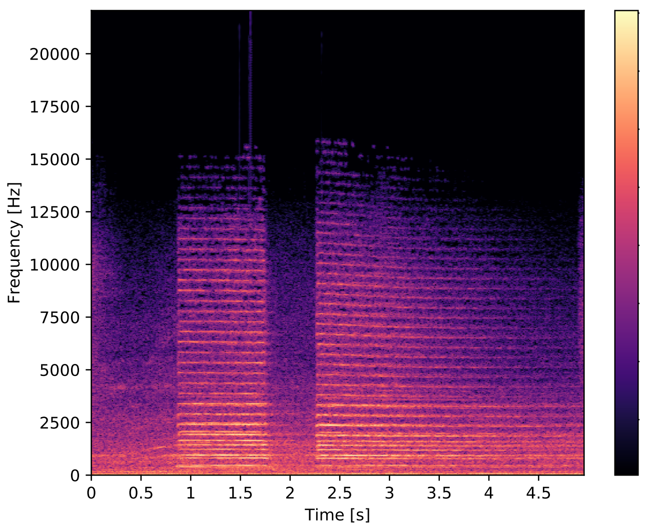
Librosa internally generate and use mel filterbanks, converting Hertz to mel, to compute the mel-scaled spectrograms. By default these filterbanks are generated with Librosa's own replication of the well-established Matlab Auditory Toolbox of Slaney[7], not the HTK formula as presented in subsection 2.2.5 and subsubsection 2.2.5.1. The difference being that in Slaney's implementation the frequency bins are linear-spaced up to 1000 Hz, after which it becomes log-spaced. We use 128 frequency bins as specified in the work by Salamon and Bello [27].

Resizing is needed for the linear spectrograms when the frequency dimension is bigger than the required dimensions, 128 as with mel-scaled spectrograms. We use Pillow's[8] implementation of Lanczos resampling to resize the spectrograms. Figure 3.5 illustrates the Lanczos resampling of a spectrogram along the frequency axis.

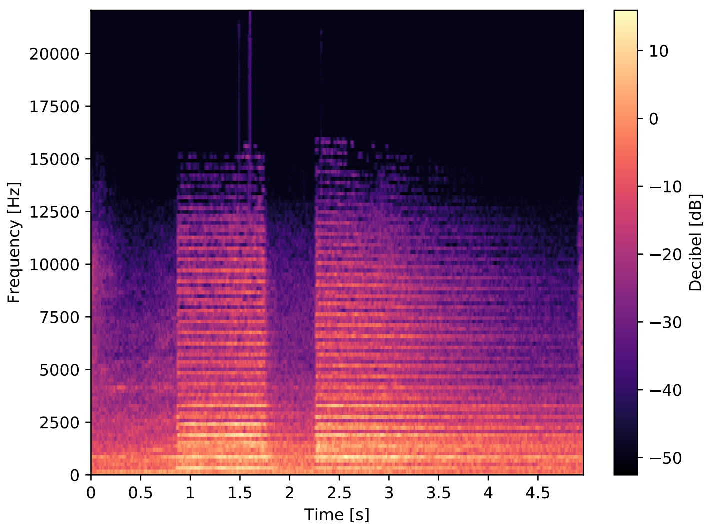
Logarithmic compression, i.e., decibel, is applied to both spectrograms and mel-scaled spectrograms using the **power_to_db**[9] function in Librosa.

Each TF-patch is normalized individually with respect to its own mean power. This is done so that the filter kernel of the CNN learns to distinguish and generalize patterns of relative intensity rather than the absolute loudness in the spectrograms.

[3]`https://github.com/jiaaro/pydub`
[4]`https://github.com/jiaaro/pydub`
[5]`http://sox.sourceforge.net/SoX/Resampling`
[6]`https://www.scipy.org/`
[7]`https://engineering.purdue.edu/~malcolm/interval/1998-010/AuditoryToolboxTechReport.pdf`
[8]`https://pillow.readthedocs.io/en/3.1.x/index.html`
[9]`https://librosa.github.io/librosa/generated/librosa.core.power_to_db.html`



(a) Original



(b) Resampled with Lanczos

Figure 3.5: Spectrogram (a) of a car horn using 2048 $N_{fft}$ and 75% overlap. (b) shows Lanczos resampling along the frequency axis with 128 bins.

# Chapter 4

# Evaluation

## 4.1 Data representation comparisons on SB-CNN

Here, we present the results from our models. Data representation varies between the models, but every model were trained and evaluated with 5-fold cross validation on the Salamon-Bello CNN, defined in section 3.1.4.1.

As seen in Table 4.1, the best performing model uses mel-scaled spectrograms with 2048 $N_{fft}$ and 75% overlap. It achieved an accuracy of 74.70% and 88.35% in top-1 and top-3 respectively. Comparing mel-scaled spectrograms with linear spectrograms, both with 2048 $N_{fft}$ and 50% overlap at 44.1 kHz sampling rate, we see that the model using mel-scaled spectrograms outperforms the model using spectrograms, with a top-1 accuracy of 71.85% against 63.35%. This is a significant difference of 8.5 percentage points in top-1 accuracy. The model using mel-scaled spectrograms with 2048 $N_{fft}$ with 75% overlap perform better than the model with 50% overlap, with an increase in accuracy of 2.85 percentage points. Comparing the mel spectrogram model with zero overlap with the one using 2048 $N_{fft}$ and 50% overlap, we see a smaller improvement of 1.65 percentage points.

Table 4.1: Results table for models trained on 44.1 kHz data. $S$ and $M$ in the type column is for spectrogram and mel-scaled spectrogram respectively. The top performer used mel-scaled spectrograms with 2048 $N_{fft}$ and 75% overlap. This shows that higher resolution and smoothness in frequency compared to temporal resolution, gave the best results.

| **Type** | $\boldsymbol{N_{fft}}$ | **Overlap** | **Sampling rate** | **Top-1** | **Top-3** |
|---|---|---|---|---|---|
| | **(samples)** | **(%)** | **(kHz)** | **(%)** | **(%)** |
| **S** | 2048 | 50 | 44.1 | **63.35** | 81.25 |
| **M** | 1024 | 0 | 44.1 | 70.20 | 86.90 |
| **M** | 2048 | 50 | 44.1 | 71.85 | 87.10 |
| **M** | 2048 | 75 | 44.1 | **74.70** | 88.35 |

### 4.1.1 Downsampled data

To test how sampling rate impacts the neural network's performance we resampled the dataset into 16 and 32 kHz. The 5-fold cross-validation results are shown in Table 4.2. Here, again, the model using mel-scaled spectrograms performs better than the model using spectrograms, with 66.04% vs. 63.43% in top-1 accuracy. All models trained and evaluated on downsampled data performed equal or worse than their counterpart, shown in Table 4.1. This result is expected, since information in the original data at higher frequencies is lost at these lower sampling rates. In addition, lower sampling rate means that the STFT window covers a larger time span with the same size of $N_{fft}$. The lower temporal resolution is a probable cause of the decreased accuracy.

Comparing the model using mel-scaled spectrograms with 50% overlap with its downsampled counterpart, we see a loss of in accuracy of 2.61 percentage points. However, the models using linear spectrograms were not impacted by downsampling, where we see an increase of 0.08% accuracy in the 50% overlap case, which we regard as insignificant.

Table 4.2: Results table for downsampled audio. $S$ and $M$ in the type column is for spectrogram and mel-scaled spectrogram respectively. Again the mel-scaled spectrogram model outperformed linear spectrograms. The model trained on 16 kHz data performed the worst. This was expected, since a sample rate of 16 kHz implies a Nyquist frequency of 8 kHz, see Equation 2.14. Our analysis of the ESC-50 data shows that there is the frequency content in the data above 8 kHz, all this information is missed in the downsampled data.

| **Type** | $\boldsymbol{N_{fft}}$ | **Overlap** | **Sampling rate** | **Top-1** | **Top-3** |
|---|---|---|---|---|---|
| | **(samples)** | **(%)** | **(kHz)** | **(%)** | **(%)** |
| **S** | **2048** | 75 | 16 | 62.43 | 80.79 |
| **S** | **2048** | 50 | 32 | **63.43** | 81.74 |
| **M** | **2048** | 50 | 32 | **66.04** | 84.19 |

## 4.2 Further evaluation of the top performing model

### 4.2.1 Confusion matrix

The confusion matrix serves as a further tool to analyze the results, in addition to the accuracy. Figure 4.1 illustrates the predictions on the evaluation set from all five folds, superimposed into one matrix. The rows correspond to the true class and the column to the prediction made by the network. An accuracy of 100% would yield non-zero values along the diagonal only.

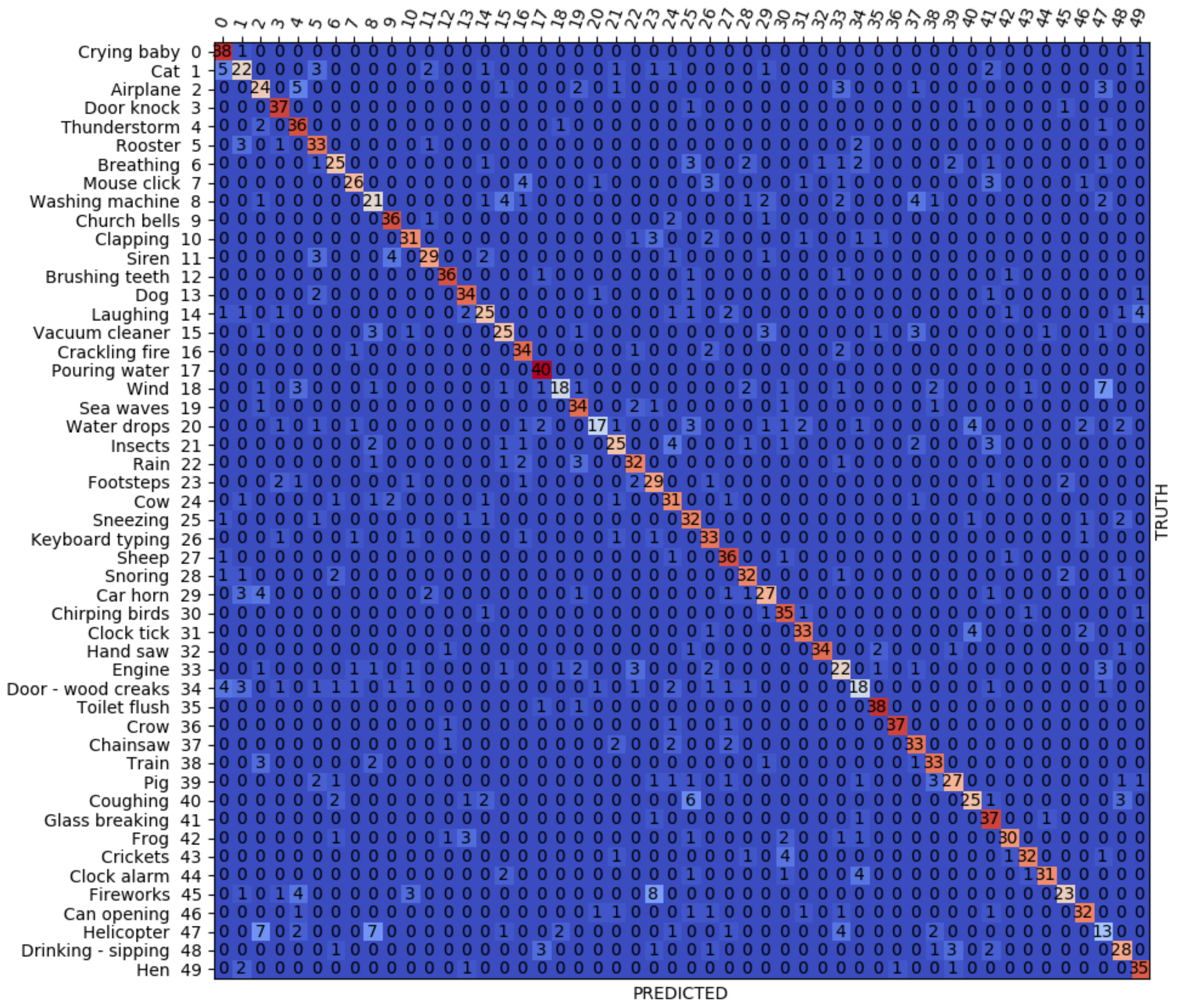


Figure 4.1: Confusion matrix from the top performing model. This figure shows the predictions on the evaluation set for every fold in one matrix. A noticeable pattern is the confusion between the noisier classes such as Helicopter, Wind, Thunderstorm and Airplane. Helicopter was the hardest class to predict for the model, with only 13/40 correct predictions, as it was mainly confused with other noisy classes like Washing machine and Airplane. Pouring water was especially salient to the network, with every example correctly classified.

From the confusion matrix, we observe that many of the mispredictions highest in number are between classes that resemble each other more closely than other classes to humans as well. As mentioned in the Dataset section, a survey was performed to measure human performance on the ESC-50 dataset[23]. These predictions had an overall accuracy of 81.3%, compared to our top model with an accuracy of 74.7%. Common mispredictions in this survey have significant overlap with our results, such as predicting *Helicopter* to be *Airplane*. Other examples of overlapping mispredictions include predicting *Fireworks* as *Footsteps* and *Cat* as *Crying baby*.

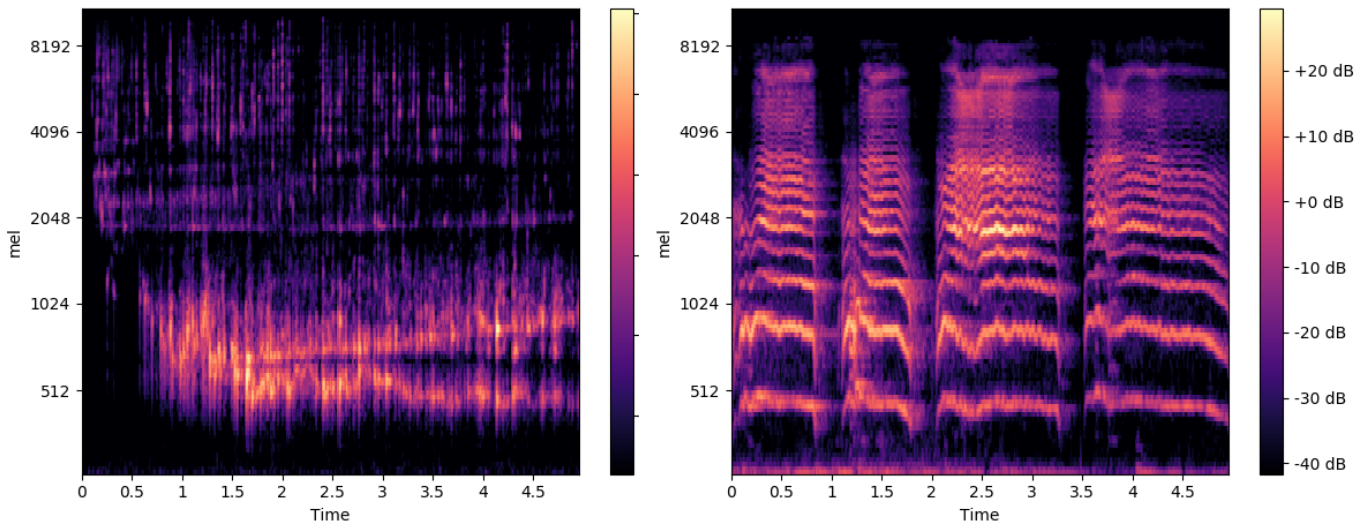
The classes most distinct to the network:

- **Pouring water**, 40/40 correct predictions
- **Toilet flush**, **Crying baby**, 38/40 correct predictions
- **Door knock**, **Crow**, **Glass breaking**, 37/40 correct predictions

The classes least distinct to the network:

- **Helicopter**, 13/40 correct predictions
- **Water drops**, 17/40 correct predictions
- **Door - wood creaks**, **Wind**, 18/40 correct predictions

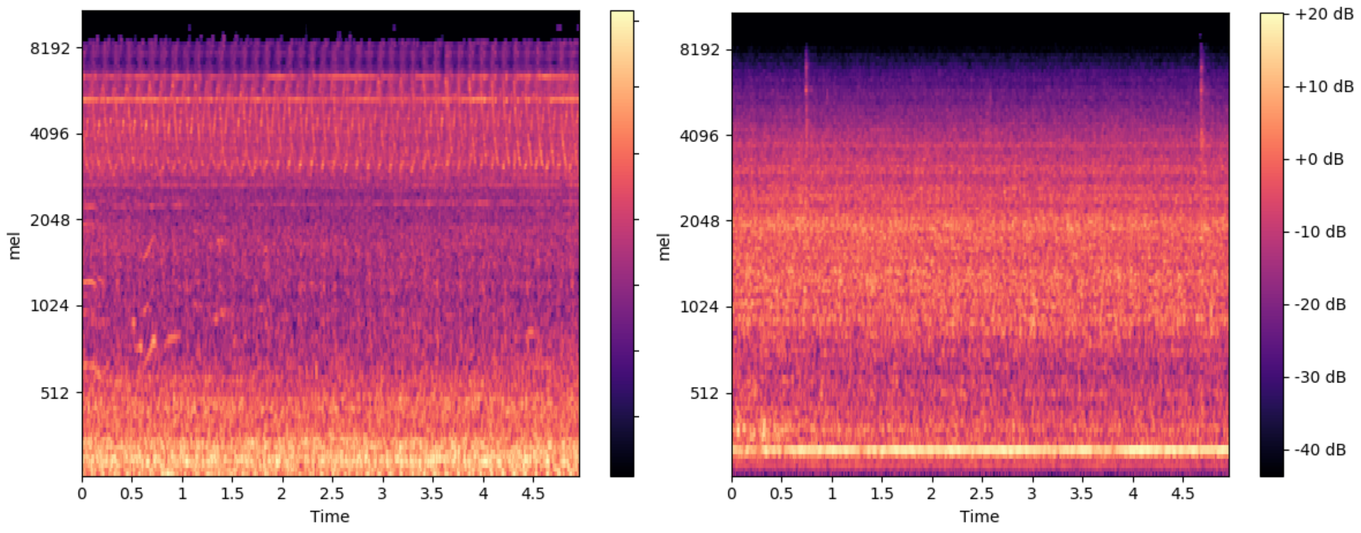
Figure 4.2 shows two mel-scaled spectrogram examples of classes that the model were very good at predicting, while Figure 4.3 shows examples of classes that were mispredicted as a variety of noisy, droning sounds. Classes that the model predicted correctly contain unique temporal patterns or patterns in frequency. Noisy classes, however, we believe are mispredicted due to the fact that they share a wide band frequency content and a relatively continuous intensity across the duration of the entire audio clip. This makes it harder to find patterns and learn features for the convolutional filters.



(a) Pouring water.

(b) Crying baby.

Figure 4.2: A commonality among the top classes is that they contain unique spectral lines and harmonics, such as in these two examples. Our theory is that these spectral patterns are 'unique enough' between classes so as to provide the convolutional filters with room to learn discriminating features.



(a) Helicopter.

(b) Washing machine.

Figure 4.3: Noisier classes were more often mispredicted as each other. Our explanation is that because their frequency content is spread more evenly from 0 Hz up to the Nyquist frequency, the convolutional filters have a harder time finding discriminating features to separate these classes.

### 4.2.2 Evaluation of data with added noise

Surveillance cameras are expected to be subjected to noise from different sources. Static noise from circuits, noise generated from motors in the camera and wind are common sources of noise, all of which influence the recorded audio. Testing the models tolerance to noise is essential when considering real world applications since input data may be contaminated with noise.

To test this we performed inference with increasing proportions of wind noise added to the evaluation data. The wind was scaled in power to a set SNR using Equation 4.1.

$$SNR_{dB} = 10 \times log_{10}\left(\frac{P_{signal}}{P_{noise}}\right) \tag{4.1}$$

Figure 4.4 shows the accuracy for different values of SNR. At 50 dB SNR there is virtually no wind noise in the signal, at 0 dB SNR the wind and signal are equal in power. The accuracy declines somewhat linearly until 22 dB SNR, after which the decline accelerates heavily. At 0 dB SNR the accuracy is reduced by circa 14 percentage points. At 0 dB SNR the accuracy is down to about 45%, a reduction nearly 30 percentage points. From these results it is clear that noise has a adverse effect on accuracy, while still retaining higher accuracy than expected, as the training did not incorporate data augmented with noise. We expect that a model trained on randomly denoised data would outperform this model in a test such as this.

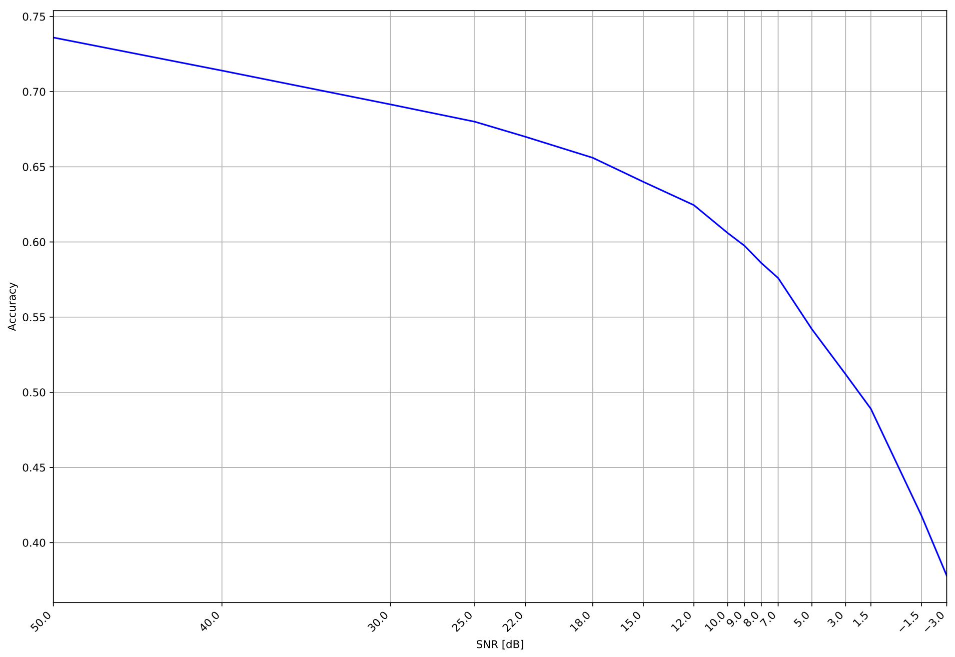
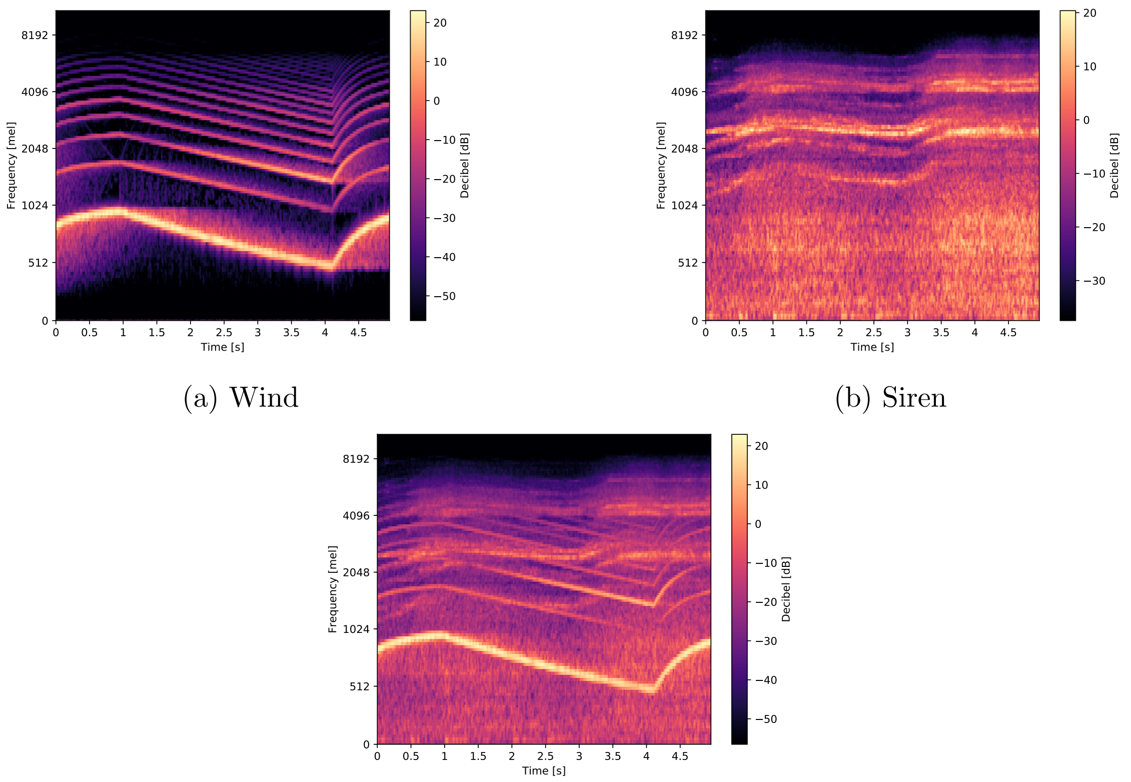


Figure 4.4: The mean accuracy on the evaluation set for all folds, as the signal to noise ratio increases. The falloff in accuracy is linear until circa 25 dB SNR, where it starts rapidly declining.



(a) Wind

(b) Siren

(c) Siren and wind

Figure 4.5: Siren(a) and wind(b) mixed with 10 dB SNR, demonstrating what a real world scenario could look like.

### 4.2.3 Mixed signals

We subjected the best performing model to a mixed signals test. This test was performed on one of the 5 folds, with specific clips chosen from the evaluation set of that fold. In each test, two clips from different classes, which the model correctly identified on their own, were mixed together at three different ratio levels. The clips mixed were not normalized before mixing, they were simply attenuated at various ratios and then superimposed.

The mixed signal was then put through the same pre-processing pipeline as in regular evaluation, which means that loudness normalization with the mean power as reference was applied to the already mixed signal. The interest in the test was two-fold. We wanted to see whether the less salient clip would disappear from top three predictions at equal ratios. We also wanted to see if both clips, regardless of power level, showed up in the top three predictions at the two other ratios, or if some other completely different class emerged at the top.

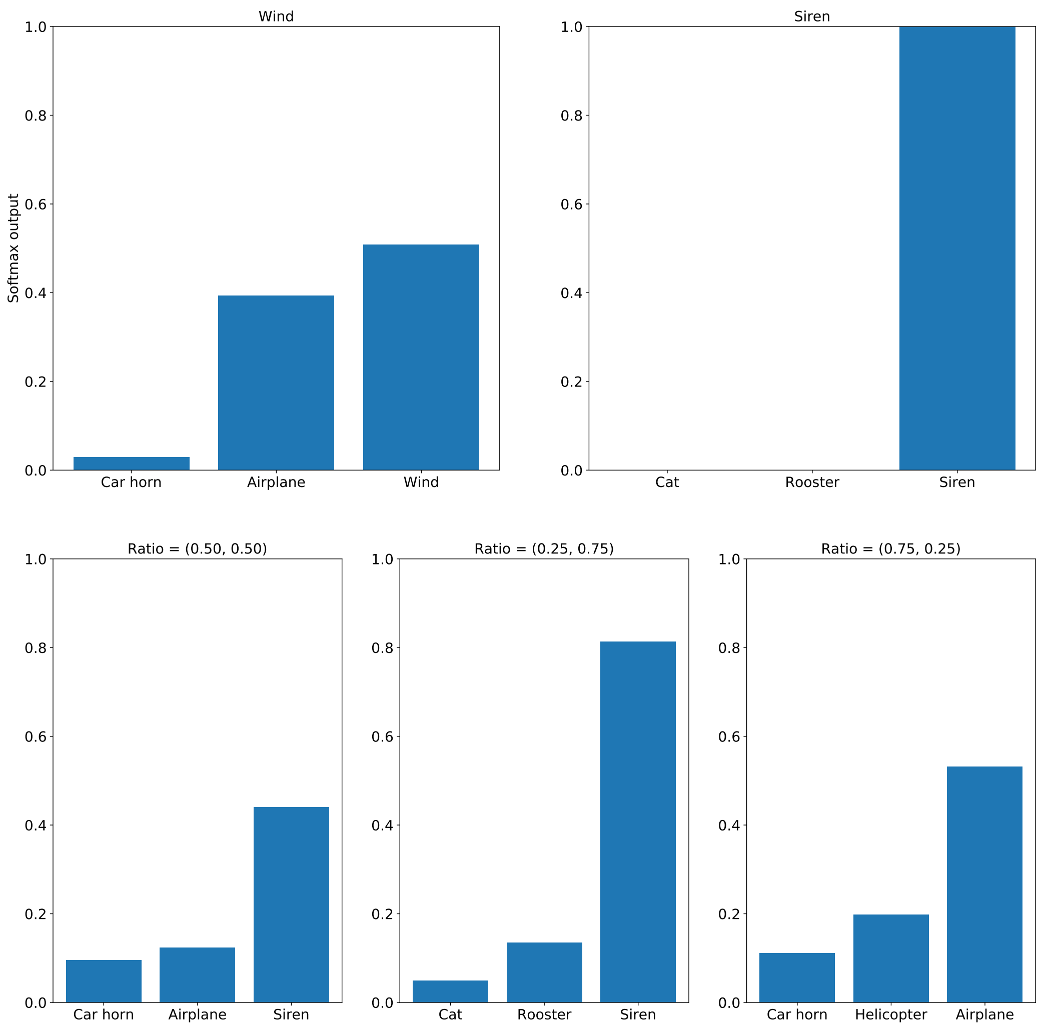


Figure 4.6: In the top part: top 3 predictions for each clip, in the bottom part: top 3 predictions for a signal mixing the two clips at different ratios. The Wind class is absent from every mix, while the salient Siren class only disappears in the 25/75 case.

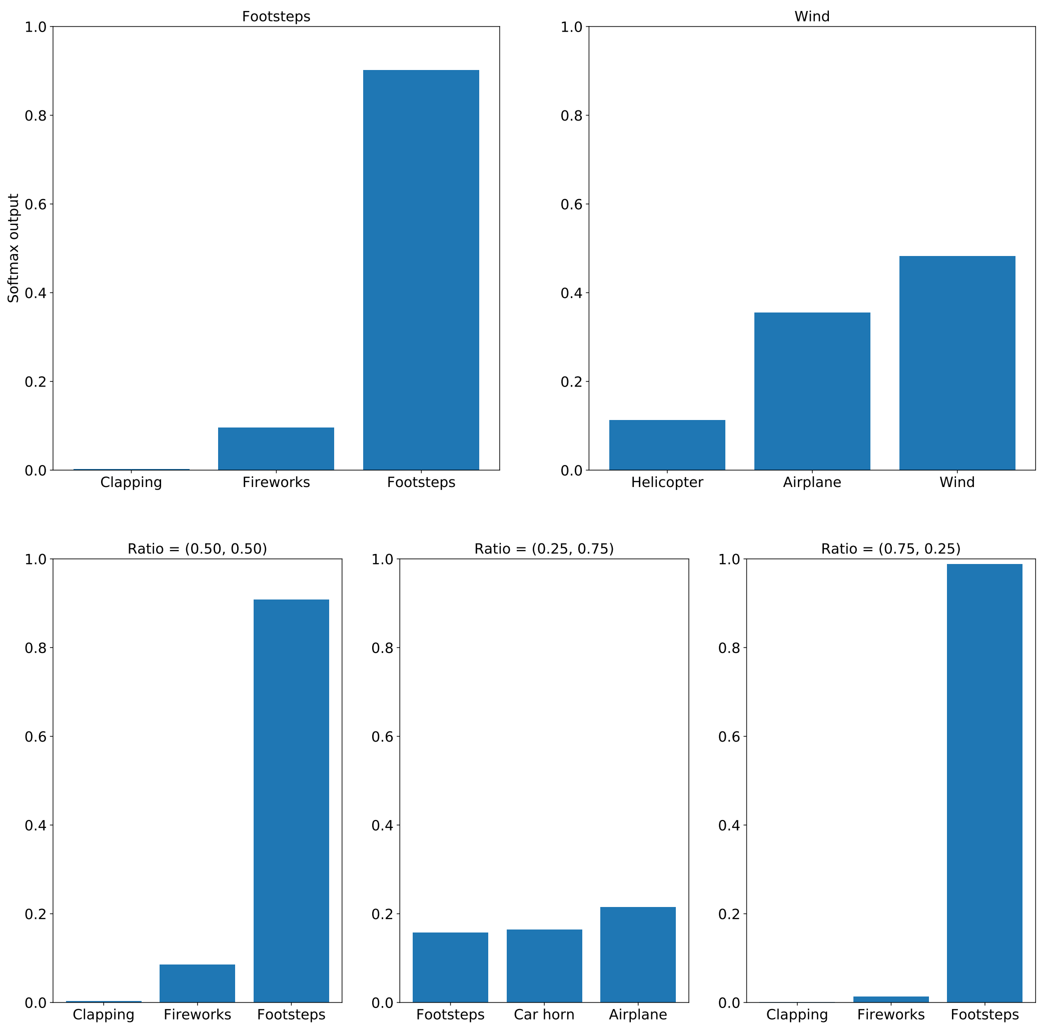


Figure 4.7: In the top part: top 3 predictions for each clip, in the bottom part: top 3 predictions for a signal mixing the two clips at different ratios. The Footsteps class remains in every mix, while the Wind disappears from the top 3 predictions.

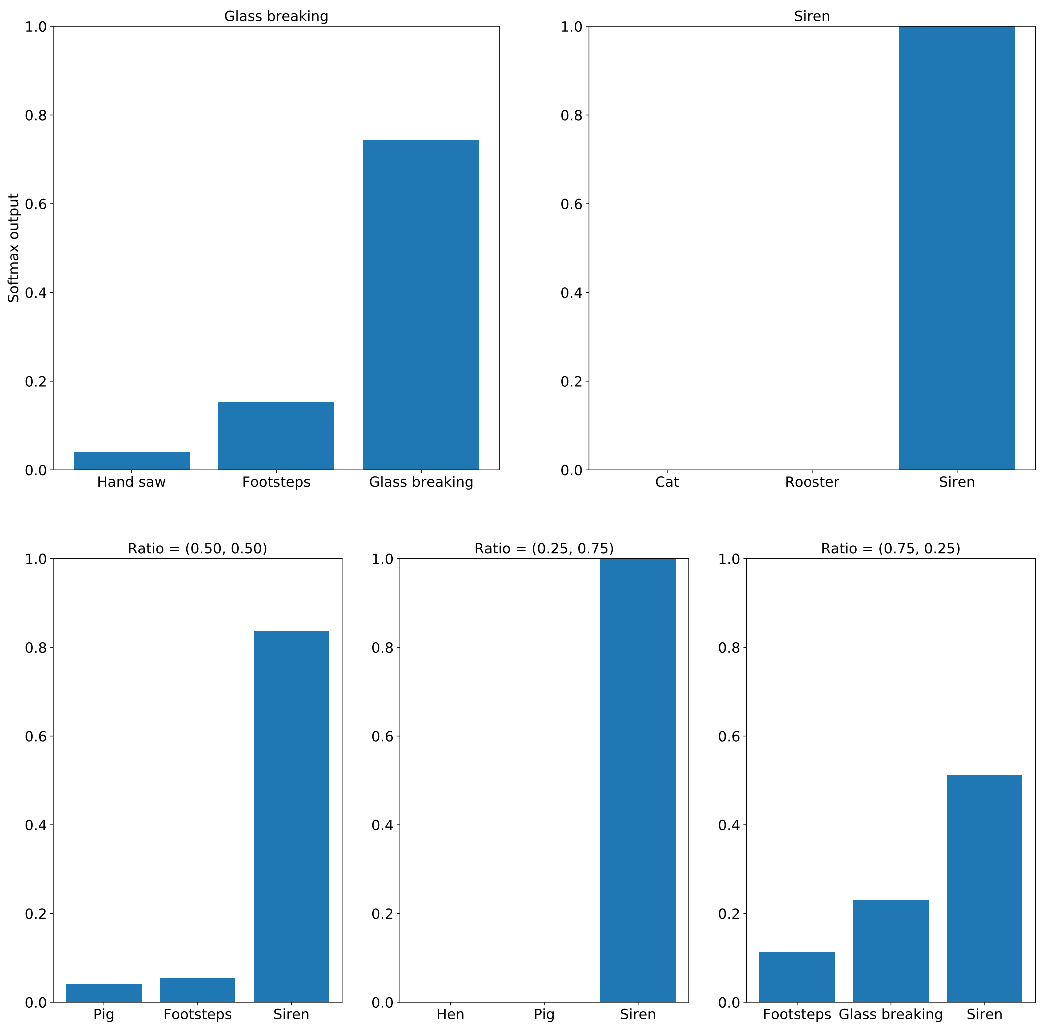


Figure 4.8: In the top part: top 3 predictions for each clip, in the bottom part: top 3 predictions for a signal mixing the two clips at different ratios. As the Siren class is very salient, it remains the top predicted class regardless of mixing ratio, while Glass breaking only remains when it is dominating the mix.

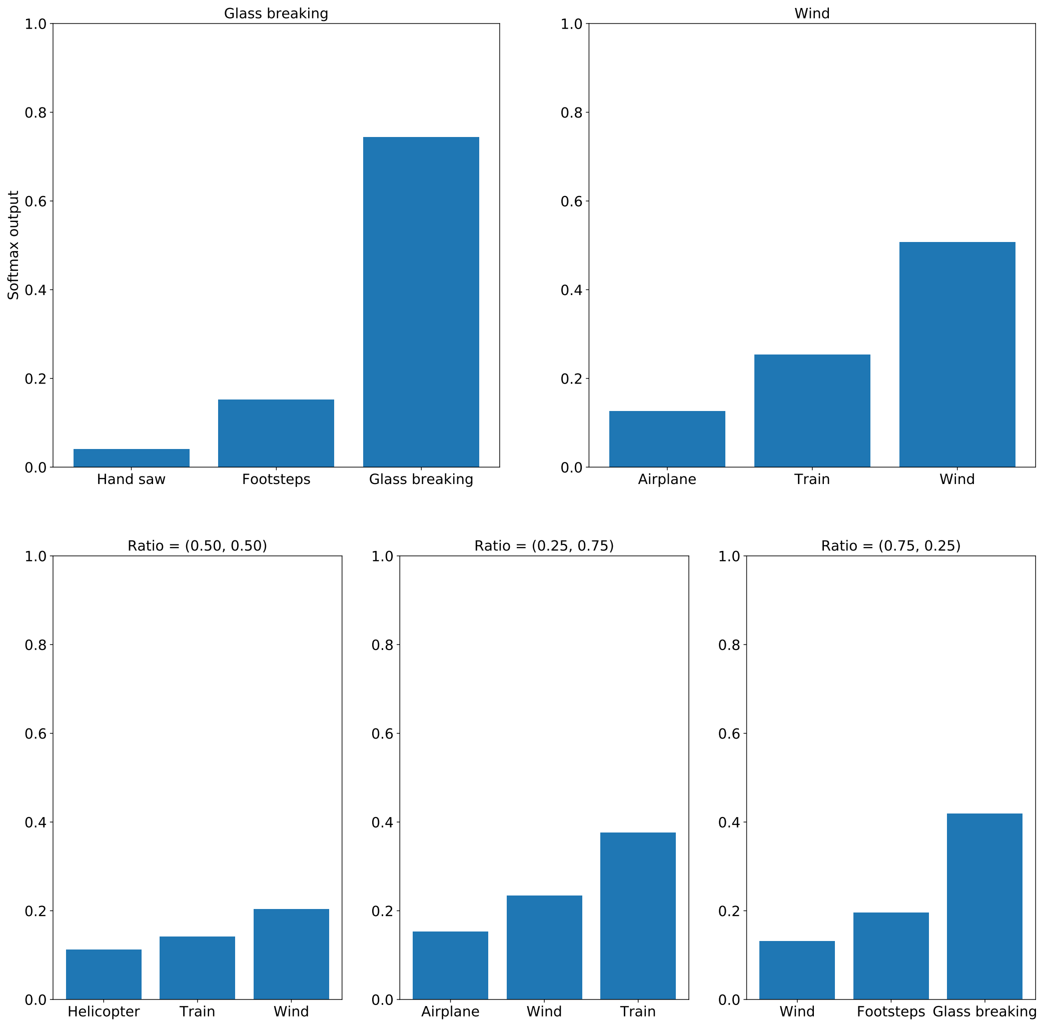
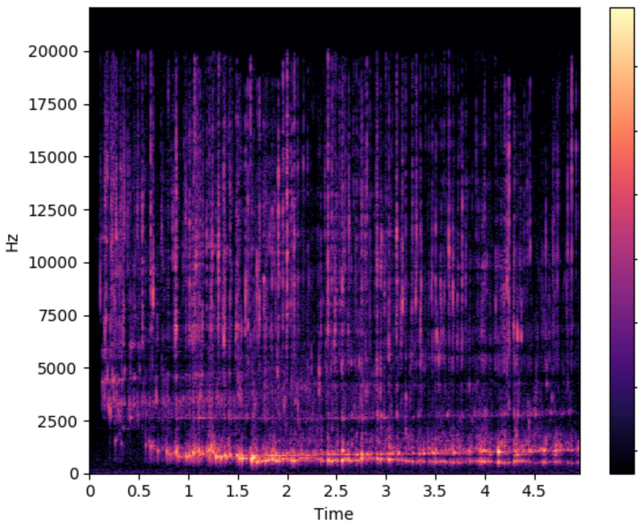


Figure 4.9: In the top part: top 3 predictions for each clip, in the bottom part: top 3 predictions for a signal mixing the two clips at different ratios. In the 50/50 and 25/75 case, Glass breaking disappears from the top 3 predictions.

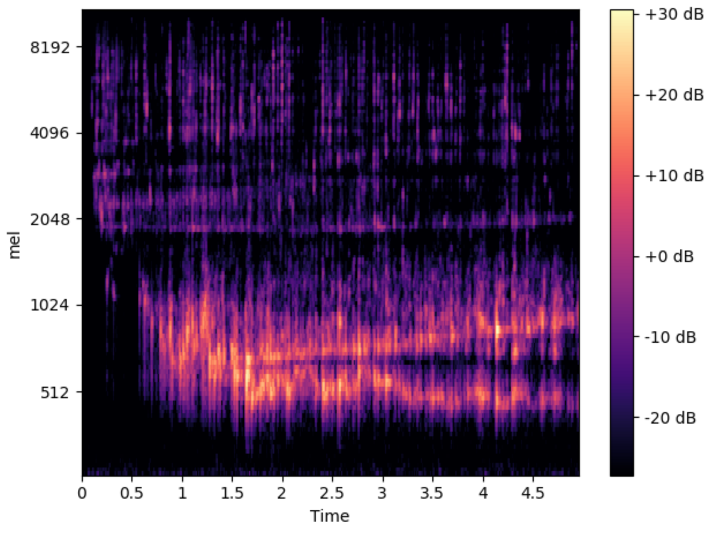
The softmax output shows that a correctly predicted class could vary a lot in confidence level. As seen in Figure 4.6, even though both the Wind class and the Siren class were correctly predicted, the softmax for Wind is much lower than the softmax for Siren. This could be important when evaluating a system that is deployed, as one would want a system with a high activation on only the true class preferably. We see that in cases such as in Figure 4.9, a completely different class emerges to the top prediction when mixed at a 25/75 ratio and in the 50/50 case the Glass breaking class has disappeared from the top 3 predictions. As this test was performed on only 4 pairs of signals, we do not draw any strong conclusions from the results, except that the model appears to not handle mixed signals well with any consistency.

## 4.3 Mel-scaled spectrograms compared to linear spectrograms

As mentioned in subsection 2.1.3 in the Theory section, how data is represented can impact performance a lot when training a machine learning model. If the representation of that data accentuates any existing patterns more, it is reasonable to expect it to perform better. In our models, mel-scaled spectrograms consistently outperformed linear spectrograms. We believe that this is due to the fact that a majority of sounds in the ESC-50 dataset have their distinct frequency content in the 0-10000 Hz range and much less content in the 10000-20000 Hz range. Whenever this is the case, a representation that accentuates the lower bands of frequencies will provide the filters in the neural network with more variation and detail to learn features from. Figure 4.10 illustrates this with two examples of the same audio clip. The linear spectrograms also consistently lack frequency content in the upper rows of the input data. Due to the nature of the sounds in the dataset, many classes have similar content (zeroes) in the upper rows, and space that doesn't provide unique information is still trained on. The mel-scaled spectrograms on the other hand, since they compress the higher frequencies, generally fill the entire input matrix more fully. This spreads information more evenly in the input, making it easier for the network filters to find features.



(a) Linear spectrogram of pouring water.



(b) Mel-scaled spectrogram of pouring water.

Figure 4.10: The same audio clip as a linear spectrogram (left hand picture) and as a mel-scaled spectrogram (right hand picture). The mel-scaled spectrogram has higher resolution in the lower bands of frequencies, making any patterns in that range more distinct to the network. The two spectral lines at circa 1000 Hz in the linear spectrogram are much more distinct in the mel-scaled spectrogram.

# Chapter 5

# Conclusion

## 5.1 Summary of Thesis Achievements

We implemented a training framework with Python and Tensorflow. We used SB-CNN[27] and the ESC-50 dataset[23] to train and evaluate our models.

### 5.1.1 Answers to thesis questions

**What kind of data input construction provides the best performance?**
The mel-scaled spectrograms models yielded higher accuracy than the linear spectrograms. The top performer using mel-scaled spectrograms had 74.7% accuracy compared to 63.35% for the best model using linear spectrograms, beating it by 11.35 percentage points.
**How does linear frequency scaling compare to nonlinear frequency scaling in spectrograms?**
Our models trained on nonlinear data reached higher accuracy than those on linear data. The nonlinear mel scaled spectrograms gives far more resolution in the lower band of frequencies, where a lot of classes contain unique patterns.

**How much does the balance between frequency resolution and temporal resolution affect performance?**

Reduction in temporal resolution, in terms of downsampling, does not affect the accuracy in a significant way for models using linear spectrograms. In case of mel-scaled spectrograms, it is hard to draw any strong conclusions of how time-frequency resolution affect the performance of the model. We observe that the model with the largest amount of overlap performed best in the mel-scaled case.

**How much does downsampling the audio data affect performance?**

Accuracy was impacted by downsampling, but models with linear spectrograms were much less impacted than those using mel scaled spectrograms. We get an accuracy drop of 7.32 percentage points when downsampling from 44.1 kHz to 32 kHz when using mel scaled spectrograms. In the linear spectrogram case, the different is only 0.08 percentage points, which we regard as insignificant.

**The audio signal will probably be attenuated, how much gain boost and normalization is possible to do before the classification performance is too poor?**

We did not gather our own evaluation data, as this would have been too time consuming and would make it harder to benchmark against existing results. Therefore this question remains unanswered.

**Is it possible to separate two or more superimposed audio sources with sufficient performance?**

With our top model, it was not possible to separate two superimposed audio sources with any consistency. However, we note that any class with a high softmax output more often remained in the top 3 prediction in the mixed signals.

**How much does noise from a camera and/or noise from wind affect the accuracy?**

When evaluating on data mixed with wind noise, we observe a steep drop in accuracy after circa 25 dB SNR. The accuracy had dropped from 74.7% to circa 45% at 0 dB SNR, where the noise is equal to the signal in power.

## 5.2 Sources of error

### 5.2.1 Parameter changes affect the epoch size when using TF-patches

One of the goals with our approach in this thesis, was to attempt to change as few parameters as possible between the different models trained. This in order to make comparisons of model performance easier to reason about and explore.
However, using the TF-patch approach described in the Implementation section, while varying $N_{fft}$ or the sampling rate, changes the number of TF-patches that one training example consists of. One value of $N_{fft}$ may for example yield 427 TF-patches per audio clip in one model, where another value would yield 300 TF-patches per audio clip. This changes the size of one epoch, and can be interpreted as a variation in data augmentation. This variation is then in addition to changes in time resolution, frequency resolution and sampling rate.
As an attempt to make model comparisons as fair as possible still, we chose to train all the models a number of epochs, so as to have approximately the same amount of update steps before the final evaluation. This means that epoch size varies between models with varying $N_{fft}$ and overlap, which makes it harder to compare them to each other. Note that this does not apply to the comparison of mel-scaled spectrogram and linear spectrogram models where these parameters are equal.

### 5.2.2 ESC-50 dataset is small

The ESC-50 dataset contains only 2000 unique audio clips. In order to make stronger claims of model performance and its chances of performing in a real world scenario, a lot more training and evaluation data would be needed.

## 5.3 Future Work

**Dataset augmentation**. In this paper we use the ESC-50 dataset, which is a relatively small data set. Augmenting the training set could further improve the accuracy performance, such as using:

- Dataset augmentation with noise such as wind. This can increase robustness against noisy input data, as described by Salamon and Bello[27].
- Other augmentations such as varying pitch and attenuation increase performance and generalization[27].

**New, larger datasets** There is no dataset for audio that compares to ImageNet[25], which contains several million images, spanning thousands of categories. New collaborative efforts to create larger datasets for audio would provide new challenges and better benchmarking. A larger dataset would also have to contain data from more varied recording scenarios. Since higher frequencies attenuate faster with distance than lower frequencies, this needs to be taken into account when creating a varied dataset.

**Data representation**. There are several parameters that are not explored in this paper which could impact the model's accuracy. Some trade-offs between accuracy and computational costs are unavoidable and for embedded systems these should be further investigated:

- The number of points in the filterbank for mel-scaled spectrograms.
- Non-normalized filterbanks or other types of normalization.
- Tapering functions. There are many variants and depending on what classes is to be classified other tapering functions could improve the performance.
- Other scales. The mel-scale proved to increase performance compared to linear spectrograms and there might be other scales that aid the model to discriminate between classes better.

**Raw audio and other architectures**.

One team of researchers trained on raw audio data, providing an end-to-end solution[29]. Other researchers successfully used a model for filterbank learning[26], which also uses raw audio. These developments suggests that pre-processing of audio data could perhaps be avoided, but the impact on overall computational costs is not stated.

# Bibliography

[1] Caffe, deep learning framework. http://caffe.berkeleyvision.org.

[2] Characteristics of different smoothing windows. National Instruments. http://zone.ni.com/reference/en-XX/help/370051V-01/cvi/libref/analysisconcepts/characteristics_of_different_smoothing_windows.

[3] The iris flower dataset. https://en.wikipedia.org/wiki/Iris_flower_data_set.

[4] Methods for reading data into tensorflow graphs. https://www.tensorflow.org/versions/r1.1/programmers_guide/reading_data.

[5] Tensorflow, open source machine learning framework. https://www.tensorflow.org.

[6] Torch, scientific computing framework. http://torch.ch.

[7] Urban sound dataset. https://urbansounddataset.weebly.com.

[8] Virtualenv, a tool to create isolated python environments. https://virtualenv.pypa.io/en/stable/.

[9] Understanding fft overlap processing. Primer, Tektronix, January 15 2014. https://www.tek.com/document/primer/understanding-fft-overlap-processing-fundamentals-0.

[10] Understanding ffts and windowing. National Instruments, December 30, 2016. http://www.ni.com/white-paper/4844/en/.

[11] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[12] J. Amoh and K. Odame. Deepcough: A deep convolutional neural network in A wearable cough detection system. *CoRR*, abs/1509.02512, 2015.

[13] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, Mar. 2016.

[14] F. Font, G. Roma, and X. Serra. Freesound technical demo. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13, pages 411–412, New York, NY, USA, 2013. ACM.

[15] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

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

[17] F. J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, Jan 1978.

[18] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[19] M. Huzaifah. Comparison of time-frequency representations for environmental sound classification using convolutional neural networks. *CoRR*, abs/1706.07156, 2017.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[21] W. Luo, Y. Li, R. Urtasun, and R. Zemel. Understanding the effective receptive field in deep convolutional neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4898–4906. Curran Associates, Inc., 2016.

[22] K. J. Piczak. Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, Sept 2015.

[23] K. J. Piczak. Esc: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, pages 1015–1018, New York, NY, USA, 2015. ACM.

[24] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017.

[25] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[26] H. Sailor, D. Agrawal, and H. Patil. Unsupervised filterbank learning using convolutional restricted boltzmann machine for environmental sound classification. pages 3107–3111, 08 2017.

[27] J. Salamon and J. P. Bello. Deep convolutional neural networks and data augmentation for environmental sound classification. *CoRR*, abs/1608.04363, 2016.

[28] S. Sigtia, A. M. Stark, S. Krstulovi, and M. D. Plumbley. Automatic environmental sound recognition: Performance versus computational cost. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(11):2096–2107, Nov 2016.

[29] B. Zhu, K. Xu, D. Wang, L. Zhang, B. Li, and Y. Peng. Environmental sound classification based on multi-temporal resolution CNN network combining with multi-level features. *CoRR*, abs/1805.09752, 2018.