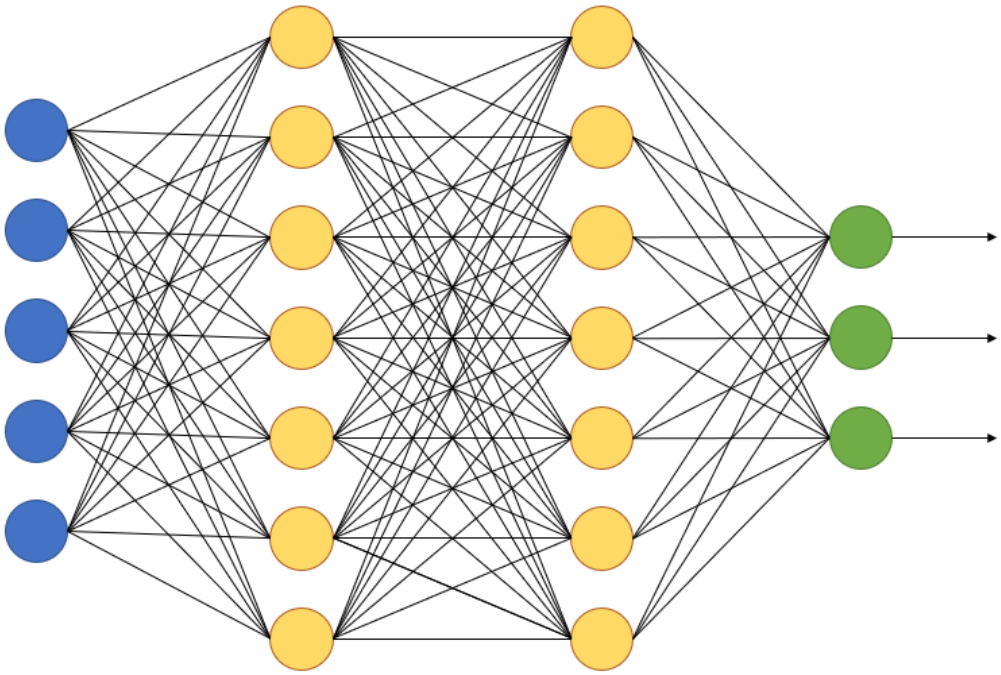
CHALMERS
UNIVERSITY OF TECHNOLOGY



Single and Multi-Label Environmental Sound Classification Using Convolutional Neural Networks

Master's thesis in the Programme Sound and Vibration

SANTIAGO ALVAREZ-BUYLLA PUENTE

Division of Applied Acoustics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018: ACEX30-18-65

Single and Multi-Label Environmental Sound Classification Using Convolutional Neural Networks

SANTIAGO ALVAREZ-BUYLLA PUENTE

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Civil and Environmental Engineering
Division of Applied Acoustics
Audio Technology Group
Chalmers University of Technology
Gothenburg, Sweden 2018

Single and Multi-Label Environmental Sound Classification Using Convolutional Neural Networks

SANTIAGO ALVAREZ-BUYLLA PUENTE

Supervisor: Jens Ahrens, Department of Architecture and Civil Engineering
Examiner: Jens Ahrens, Department of Architecture and Civil Engineering

Master's Thesis 2018: ACEX30-18-65
Department of Civil and Environmental Engineering
Division of Applied Acoustics
Audio Technology Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visual representation of a feedforward neural network.

Typeset in LaTeX

Gothenburg, Sweden 2017

Single and Multi-Label Environmental Sound Classification Using Convolutional Neural Networks

SANTIAGO ÁLVAREZ-BUYLLA PUENTE
Department of Civil and Environmental Engineering
Chalmers University of Technology

Abstract

Artificial neural networks are computational systems made up of simple processing units that have a natural propensity for storing experiential knowledge and making it available for use. In the recent years this technology has seen an exponential growth in the fields of image recognition, natural language processing or speech recognition. However, there is a dearth of research on environmental sound analysis. In combination with IoT and wireless sensor networks, artificial neural networks could help to characterize and therefore better address noise issues present in urban environments.

This master thesis investigates the theory and construction of artificial neural networks for single-label and multi-label multiclass classification of environmental sounds like dog bark, street music or jackhammer. Evaluation to different corruptions of the sounds are studied, as well as methods to increase robustness to these variations.

A convolutional neural network arquitecture is proposed for both tasks. The inputs to the networks are time-frequency patches extracted from the computed mel-spectrogram of the signals. Dropout and weight decay regularization methods are applied and the cross-entropy loss is optimized using Adam algorithm.

Results show that these systems are very sensitive to noise and level corruptions of the inputs. Techniques like data augmentation and amplitude scaling are needed to avoid these issues. Results to the multi-label classification task show that it is still possible for a neural network to learn in a complicated mixed environment. However there is still room for improvement regarding prediction accuracy. Since no previous benchmarks are available for comparison, this study sets the stage for the multi-label classification task using UrbanSound8K dataset.

Keywords: Deep Learning, Environmental Sound Classification, Convolutional Neural Networks, Mel-spectrogram, UrbanSound8k.

Acknowledgements

I want to thank Jens Ahrens for his help and for his confidence in the idea of this project, despite not being a common area of research in the division of applied acoustics. I also want to thank Lennart Svensson, who kindly helped me to set the planning of the project. I want also to express my gratitude to the people from ÅF: Andreas Colebring, Martin Elmcrona, Åsa Collet, Victor Diez and Karl Vilén, for their confidence in the project and their valuable feedback. Many thanks also to Justin Salomon and Karol Piczak, who kindly provided all requested information about their papers.

Santiago Álvarez-Buylla Puente, Gothenburg, June 2018

Contents

List of Figures

List of Tables

1

Introduction

This chapter will give an introduction to the master thesis, starting with a background to the subject and a literature review, followed by aim and research questions. Thereafter the outline of the document will be presented to provide a clearer understanding of the structure of the thesis.

1.1 Background

In the recent past many technological revolutions have been observed. Cloud computing, the Internet of Things (IoT) or big data analytics are now, among many others, common terms and common technologies used in people's daily lives. Additionally, the advent of low-cost and low-power transceivers has made possible the deployment of wireless sensor networks, allowing cities to monitor all kinds of events and trends in real time.[1] Smart cities aim to make use of all these technologies, obtaining large amounts of information that can then be processed using computation technologies, in order to better understand the problems present in twenty-first century cities, with the primary objective of improving citizen's welfare. Sadly enough, many citizens still suffer nowadays from exposure to inadequate noise levels that can lead to annoyance, sleep disturbance, and related increases in the risk of hypertension and cardiovascular disease.[2]

This thesis originated from the idea of extracting valuable information from street audio recordings. Even though it is clear that noise levels are a useful measure to obtain from any audio signal, being able to detect which are the sound sources present in the signals would further increase the usefulness of any audio recording. This would allow to characterize every single sound source present in the recorded signals, and consequently, in the city. Having this type of information, city agencies could more efficiently address issues related to undesired sound pressure levels and propose corrective measures grounded on solid evidence.

Artificial neural networks are computing systems that have been widely used for image recognition problems. Recent research[3–6] has proved that these systems are also suitable for audio recognition problems. This master thesis aims to contribute to the task of mitigating this overlooked but widespread noise problem, exploring how the use of artificial neural networks can help to identify the sources present in recorded environmental sounds.

1.2 Literature review

The topic of environmental sound classification using artificial neural networks has received increasing interest over the last few years. In [4] and in [5], the authors used convolutional neural networks for environmental sound classification, making use of the public Urbandsound8K dataset, which is also the dataset used for study in the present document. Using different architectures they achieved average test accuracy results of 73% and 79% respectively, the latter using different data augmentation techniques. The authors of [6] used two well-known convolutional deep neural networks for image recognition, AlexNet and GoogleNet, using the spectrogram of the audio signals as input feature to the networks. In [7] the authors used deep recurrent neural networks for the same purposes of audio scene classification achieving state-of-the-art accuracy results in the LITIS Rouen dataset, confirming the validity of this type of arquitecture for the audio classification task. In [8] the authors approached the same problem from a multi-task point of view, suggesting that recognition accuracy may improve when using a multi-task model rather than multiple task-specific models.
Although it can be seen that extensive research has been carried out on single-label audio classification, little or none attention has been paid to the task of multi-label classification, where the goal is to predict more than one possible class label. Robustness evaluation of the models to input corruptions has not received much attention either, despite being a clear problem to face in real-world applications.

1.3 Aim

This master thesis has three main goals. Firstly, investigate the theory and construction of artificial neural networks for single-label audio classification. Secondly, study the robustness to noise and level differences of the model when trained on the UrbanSound8K dataset, and test different techniques to improve this robustness. Lastly, study the reliability of multi-label environmental classification using a generated version of UrbanSound8K.

1.4 Research questions

In order to attain these aims, a different number of research questions will be answered:

- How do different noise levels in the input signals affect the classification prediction accuracy of the trained neural network?
- How do level differences in the input signals affect the classification prediction accuracy of the trained neural network?
- How do data augmentation and data preprocessing techniques help to improve network robustness to these input corruptions?
- Is it possible to solve the multi-label classification task for environmental sounds using the UrbanSound8K dataset?

1.5 Outline

This master thesis is divided into five chapters. After the introduction, Chapter 2 presents the general theory regarding artificial neural networks and feature generation methods for audio signals. In Chapter 3 the method is described, explaining how every step of the project was carried out, providing all necessary information for their possible recreation. In Chapter 4 the results are presented, explained and interpreted. Chapter 5 presents the conclusions derived from this academic work and suggesting future research directions.

2
Theory

In this chapter, a study on the theory about artificial neural networks will be presented. Thereafter, the theory necessary to understand the feature generation from the audio files will be explained.

2.1 Artificial neural networks

An artificial neural network is a massively parallel distributed processor composed of simple processing units (neurons) that is able to store experiential knowledge and make it available for use.[9] Artificial neural networks are computing systems based on a biological model: the organization of the human nervous system.[10] These computational systems resemble the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.
2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

These systems are not a recent discovery. In the formative years of neural networks (1943-1958), several researchers stand out for their pioneering contributions:[9]

- McCulloch and Pitts (1943) for introducing the idea of neural networks as computing machines.
- Hebb (1949) for postulating the first rule for self-organized learning.
- Rosenblatt (1958) for proposing the perceptron as the first model for learning with a teacher (i.e., supervised learning).

2.1.1 Relation to biological neurons and human nervous system

The fundamental processing unit in the human brain is a *neuron*, or nerve cell.[10] Neurons in the human brain are composed of a cell body, or *soma*, along with fibers called *dendrites* that receive electrical impulses from other neurons, and other, long fibers known as *axons* that conduct impulses away from the cell. Put in computer terminology, dendrites act as input devices for a neuron, while output to other neurons occurs via axons. The interface between an axon and another neuron to which it transmits information occurs across a tiny gap called a *synapse*. An electrical impulse sent down an axon causes the release of certain chemicals, called *neurotransmitters*, into the synapse. Depending on the nature and amount of these chemicals that are released, the receiving neuron is either *excited* or *inhibited*. Each

individual dendrite may receive excitory or inhibitory stimuly, and the overall effect on these stimuli on the neuron is algebraically additive. If the net effect of the excitory neurotransmitters minus the net effect of the inhibitory ones exceeds a certain electrical threshold called the *action potential*, then the neuron will fire, otherwise, it will remain inactive. While the functionality of an individual neuron is simple, the connections between neurons are very complex and organized into hierarchical layers. It is these connections that define the functionality of the nervous system.
In Figures 2.1 and 2.2 the anatomy of a biologic neuron and the synapse between two neurons can be seen.

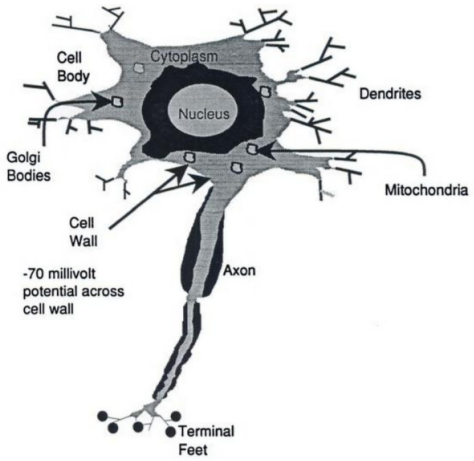


Figure 2.1: Anatomy of a neuron. Picture taken from [11].

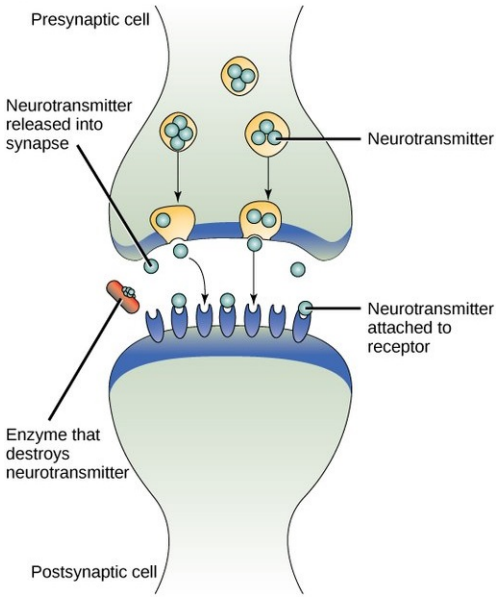


Figure 2.2: Simplified sketch of the synapse between two neurons. Picture taken from `https://commons.wikimedia.org/wiki/File:Figure_09_01_02.jpg`.

2.1.2 Feedforward neural networks

The goal of a feedforward neural network is to find a non linear function that maps the space of the inputs x to the space of the outputs y.[12] In other words, to learn the function

$$f^* : \mathbb{R}^m \to \mathbb{R}^n,\ f^*(x;\theta) \tag{2.1}$$

that can approximate:

$$y \approx f^*(x;\theta) \tag{2.2}$$

where θ are the parameters of the network. A feedforward network learns the value of the parameteres θ that result in the best function approximation, by solving the equation:

$$\theta \leftarrow \arg\min_{\theta} L(y, f^*(x;\theta)) \tag{2.3}$$

where L is a loss function chosed for the particular task. These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f and finally to the output y. Feedforward neural networks are called *networks* because they are usually represented by composing together many different functions. For instance, there might be three functions $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$ connected in a chain to form:

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x))) \tag{2.4}$$

In this case, $f^{(1)}$ is called the first layer of the network, $f^{(2)}$ is called the second layer, and so on. The final layer of a feedforward network is called the output layer. During neural network training, $f(x)$ is driven to match $f^*(x)$. Each training example x is accompanied by a label $y \approx f^*(x)$. The training examples specify directly what the output layer must do at each point x; it must produce a value that is close to y. The behavior of the other layers is not specified by the training data, but the learning algorithm must decide how to use those layers in order to produce the desired output. It is for this reason that these layers are called hidden layers.[12] In Figure 2.3 , an image of a four-layer feedforward neural network with two hidden layers can be seen:

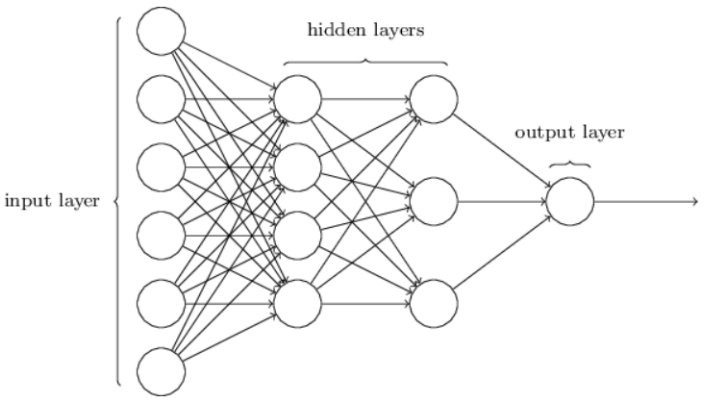


Figure 2.3: Sketch of a four-layer feedforward neural network.

These types of networks are densely connected, meaning that to each node of layer $[l]$ in the network correspond as many weights as nodes are on the previous layer $[l-1]$. These weights are multiplied elementwise by the outputs of the previous layer (or the inputs to the network if a node from the first hidden layer is being considered), and the result is arithmetically added. A bias term is added to the result and then a non-linear activation function is applied. The output of this activation function is then fed to the next layer or taken as the prediction if a node of the last layer is being considered. This procedure is depicted in Figure 2.4:

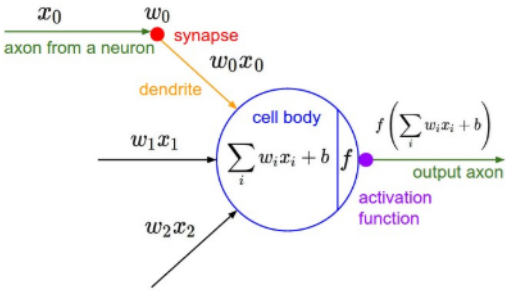


Figure 2.4: Sketch of the operations taking place in one node of a feedforward neural network. Picture taken from [13].

Expressed in a mathematical form, the activation (or output) a of certain node i belonging to certain layer l is computed as:

$$a_i^{[l]} = h(W_i^{[l]T} a^{[l-1]} + b_i^{[l]}) \tag{2.5}$$

where W is the vector of weights of that specific node, b the bias term and h the activation function operation. The role of the bias is to add more flexibility to the network by allowing shifts of the activation function. By looking at the simple example shown in Figure 2.5 it can be seen that without the bias term it would not be possible to fit a line that would approximate as good to the data as with the bias term.

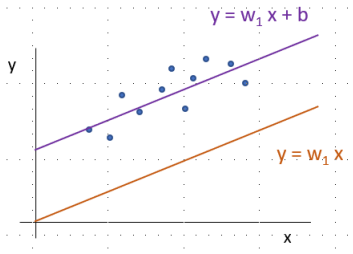


Figure 2.5: An example of how introducing a bias term helps performing linear regression.

2.1.3 Activation functions

Activation functions are used to introduce non-linearity in the system. This is done in order to increase the hypothesis space, i.e. the set of functions that the learning algorithm is allowed to select as being the solution.[12]

A linear function could be chosen as activation function. However, if that would be the case, then the feedforward network as a whole would remain a linear function of its input. It would not make sense then having more than one neuron, since it could learn the same function f^* as a feedforward network with ten thousand neurons having linear activation functions.
In modern neural networks, the default recommendation is to use the rectified linear unit or ReLU[12] defined by the activation function $g(z) = max\{0, z\}$ and depicted in Figure 2.6.

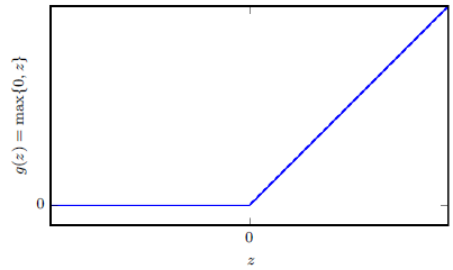


Figure 2.6: The rectified linear activation function.

Applying this function to the output of a linear transformation yields a nonlinear transformation. However, the function remains very close to linear, in the sense that is a piecewise linear function with two linear pieces. It is for this reason that they preserve many of the properties that make linear models easy to optimize with gradient based methods.[12]
The activation function plays a crucial role in the output layer of the network. For tasks that require predicting the value of a binary variable y, like classification problems with two classes, a logistic sigmoid function is usually chosen as activation function since this function squashes the output values to the range (0,1), i.e. to probabilities. By using this output activation function, the network is modelling a probability distribution over a binary variable, where the goal is to produce a single number

$$\hat{y} = P(y = 1|x) \tag{2.6}$$

that quantifies the conditional probability of y being equal to 1 given x. It follows then that

$$1 - \hat{y} = P(y = 0|x) \tag{2.7}$$

The logistic sigmoid function is defined as

$$h(z) = \frac{1}{1 + e^{-z}} \tag{2.8}$$

and it is depicted as

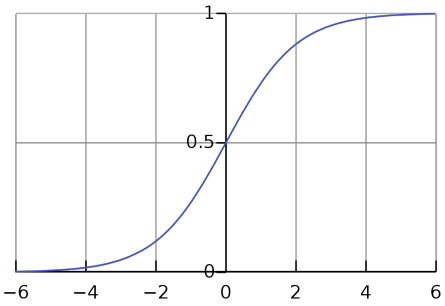


Figure 2.7: Logistic sigmoid function.

When the goal is to represent a probability distribution over a discrete variable with K possible values, i.e. a multinouilli distribution, the softmax function is used. This function can be seen as a generalization of the sigmoid function. It is defined as:

$$h(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.9}$$

where $j = 1, 2, ...K$ indexes the different output units. It can be seen that the softmax function normalizes a K dimensional vector z of arbitrary real values into a K dimensional vector $h(z)$ whose components add up to 1 (in other words, a probability vector).[14] For this reason this is the default activation function used in classification tasks where the classes are mutually exclusive, i.e. where the goal is to predict one single label.

2.1.4 Loss functions

The central goal in network training is not to memorize the training data, but rather to model the underlying generator of the data, so that the best possible predictions for the output vector y can be made when the trained network is subsequently presented with a new value for the input vector x.[15]
The error function or loss function is a function that the network wants to minimize. This is often represented as the difference between the target and the network's output. Different loss functions implement different notions of "distance" between the target and the network output, but the general idea remains the same: the cost function gives a distance metric, and the network tries to make the distance as small as possible.
One of the most common loss functions when it comes to classification tasks is the logarithmic loss function or simply *log loss*. It can also be found under the name *logistic loss* or *cross-entropy loss*. When dealing with classification problems for more than two classes, this loss function is often termed *categorical cross entropy* or *multi-class logarithmic loss*[16]. It is defined as:

$$L(y, p) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{c=1}^{C} y_{i,c} \cdot \log\left(p_{i,c}\right) \tag{2.10}$$

where m is the number of instances, C is the number of possible labels (classes), $y_{i,c}$ is a binary indicator of whether or not label c is the correct classification for

instance i, and $p_{i,c}$ is the model probability of assigning label c to instance i.
For a binary classification problem, equation 2.10 simplifies to:

$$L(y,p) = -\frac{1}{m}\sum_{i=1}^{m}(y_i \log p_i + (1-y_i)\log(1-p_i)) \tag{2.11}$$

By looking at the plot of this function an intuition of its behaviour can be gained. The plot shown in Figure 2.8 shows the logarithmic loss contribution from a single positive instance where the predicted probability ranges from 0 (the completely wrong prediction) to 1 (the correct prediction). It is apparent from the gentle downward slope towards the right that the loss gradually declines as the predicted probability improves. Moving in the opposite direction though, the log loss ramps up very rapidly as the predicted probability approaches 0. This means that the log loss heavily penalizes classifiers that are confident about an incorrect classification. For example, if for a particular observation, the classifier assigns a very small probability to the correct class then the corresponding contribution to the log loss will be very large indeed. Naturally this is going to have a significant impact on the overall loss for the classifier.[17]

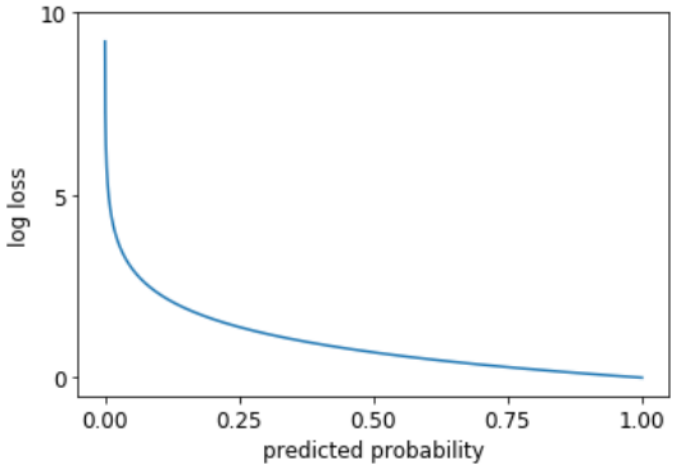


Figure 2.8: Log loss function for a single example for different predicted probabilities when the true label is 1.

2.1.5 Gradient descent

Most deep learning algorithms, such as artificial neural networks, involve optimization of some sort. If the optimization goal is to minimize a function, then this function is called the cost or the loss function.[15]
Suppose the goal is to optimize a function $y = f(x)$, where both x and y are real numbers. The derivative of this function is denoted as $f'(x)$ and it gives the slope of $f(x)$ at the point x. In other words, it specifies how to scale a small change in the input in order to obtain the corresponding change in the output: $f(x+\epsilon) \approx f(x) + \epsilon f'(x)$. It is possible thus to reduce $f(x)$ by moving x in small steps with opposite sign of the derivative. This technique is called gradient descent. For higher dimensional functions, each parameter is updated independently and the

steps lead in the direction of the steepest descent. This is how each of the parameters θ_i of the learning algorithm is updated in order to find the solution to Equation 2.3:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial L}{\partial \theta_i} \tag{2.12}$$

An example of this technique is shown in Figure 2.9.

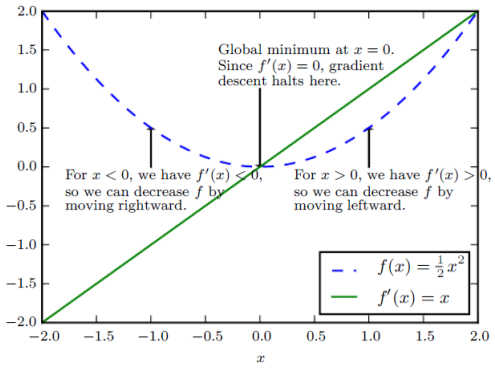


Figure 2.9: An illustration of how the gradient descent algorithm uses the derivatives of a function can be used to follow the function downhill to a minimum. Picture taken from [12].

When $f'(x) = 0$ the derivative provides no information about which direction to move. These points where $f'(x) = 0$ are called critical points. A local minimum is a point where $f(x)$ is lower than at all neighbouring points, so it is no longer possible to decrease $f(x)$ by making infinitesimal steps. A local maximum is a poing where $f(x)$ is higher than at all neighbouring points, thus is not possible to increase $f(x)$ by making infinitesimal steps. Critical points that are neigher maxima nor minima are called saddle points.[12] These three types of critical points are illustrated in Figure 2.10.

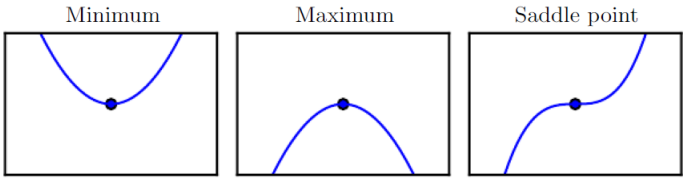


Figure 2.10: Examples of the three types of critical points, in 1-D. Picture taken from [12].

It is normally the case to optimize functions that may have many local minima that are not optimal, and many saddle points surrounded by very flat regions. All of this makes optimization very difficult, especially when the input to the function is multidimensional. It is therefore usually settled to find a value of f that is very low, but not necessarily the global minimum, as illustrated in Figure 2.11.

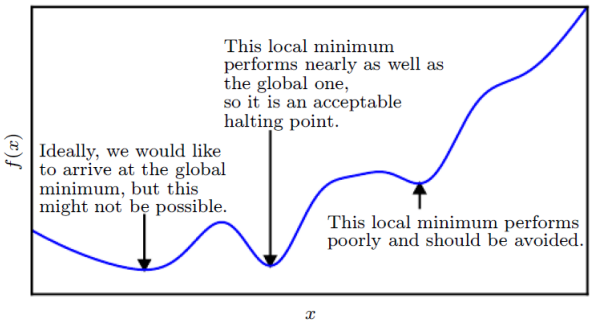


Figure 2.11: Optimization algorithms may fail to find a global minimum when there are multiple local minima or plateaus present. Solutions are generally accepted even though they are not truly minimal, so long as they correspond to significantly low values of the cost function. Picture taken from [12].

2.1.6 Convolutional neural networks

Convolutional neural networks (hereafter CNNs or Convnet) are very similar to the already presented feedforward neural networks. They are made up of neurons that have adjustable weights and biases. Each neuron receives some inputs, performs a elementwise product, adds a bias term and performs a non-linear activation. A loss function is computed and the parameters are updated using gradient descent optimization on this function. However, CNNs present some new characteristics that make them more suitable arquitectures when working with images as inputs. These characteristics will be described in the following subsections. In Figure 2.12 a typical arquitecture of a CNN is depicted:

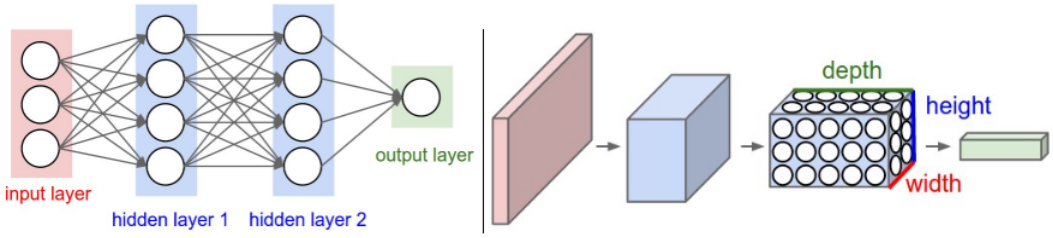


Figure 2.12: Left: A regular 3-layer neural network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. Picture taken from [13].

Another typical ConvNet arquitecture is shown in Figure 2.13, which shows the use of two new types of layers: convolutional and pooling layers. The depth of the 3D volumes corresponds in this image to the dimension going into the plane.

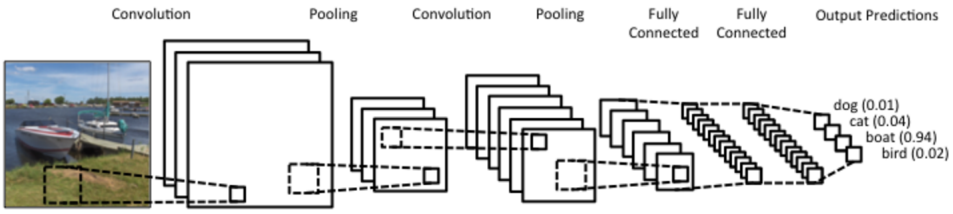


Figure 2.13: Arquitecture of a CNN having convolutional, pooling and fully connected layers.

2.1.6.1 Convolution and cross-correlation operation

CNNs introduce a new layer operation, called convolution. Convolution is a formal mathematical operation, just as multiplication, addition, and integration. Addition takes two numbers and produces a third number, while convolution takes two signals and produces a third signal.

In 2-D, the discrete convolution of two images I and K is defined as:

$$F(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.13}$$

where m and n are the dimensions of the kernel (or filter), and i and j are the indexes for each pixel. In convolutional network terminology, the first argument to the convolution is often referred to as the input (I) and the second argument as the kernel (K). The output is referred to as the feature map (F). The commutative property of convolution arises because of the flipped kernel relative to the input, in the sense that as m increases, the index into the input increases, but the index into the kernel decreases. However, this is not usually an important property of a neural network implementation.

Cross-correlation is the same as convolution but without flipping the kernel:

$$F(i,j) = (I * K)(i,j) \sum_m \sum_n = I(m,n)K(i+m, j+n) \tag{2.14}$$

Technically speaking, cross-correlation is the operation applied in convolutional layers, and not convolution. An example is shown in Figure 2.14. An input I is convolved with a kernel K and a feature map is obtained. The number of feature maps will depend on the number of kernels that the designer chooses for each particular convolution layer. The last layers of a convolutional network are regular fully connected layers.

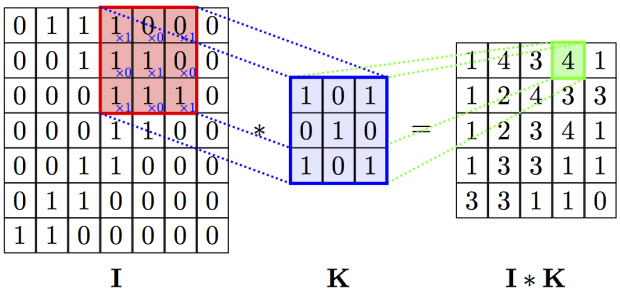


Figure 2.14: An example of 2-D cross-correlation between a kernel K and an input I, creating a feature map. The area highlighted in red is called the receptive field of the input. Picture taken from `https://github.com/PetarV-/TikZ/tree/master/2D`.

2.1.6.2 Why convolutional neural networks?

CNN architectures make the assumption that the inputs are images, which allows the encoding of certain properties into the architecture that make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. In particular, CNNs leverage two important ideas:[12]

- Sparse connectivity
- Parameter sharing

When processing an image, it might have thousands or millions of pixels, but it is possible to detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels. This means that fewer parameters are needed, which both reduces the memory requirements of the model and improves its statistical efficiency. This is what is meant by sparse connectivity.

Parameter sharing refers to the use of the same parameters (filters) for more than one function in a model. It is based on the assumption that if one feature is useful to detect at some spatial position, then it should also be useful to detect at a different position on the input.[13] In a traditional neural network, each element of the weight matrix is used exactly once when computing the output of a layer. It is multiplied by one element of the input and then never revisited. In a convolutional neural network, each member of the kernel is used at every position of the input. The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, the network learns only one set.

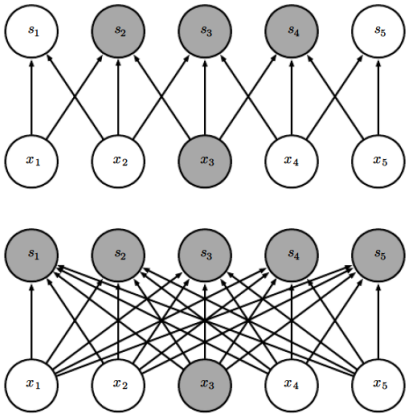


Figure 2.15: Sparse connectivity, viewed from below: One input unit is highlighted, x_3, and also the output units in s that are affected by this unit. (Top)When s is formed by convolution with a kernel of width 3, only three outputs are affected by x_3. (Bottom)When s is formed by matrix multiplication, connectivity is no longer sparse, so all of the outputs are affected by x_3. Picture taken from [12].

2.1.6.3 Pooling layers

After each convolution operation, it is common to periodically insert a Pooling layer in a ConvNet architecture. The Pooling layer operates independently on every depth slice of the input and its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. A pooling function replaces the output of the network at a certain location with a summary statistic of the nearby outputs. For example the max function, which reports the maximum output within a rectangular neighborhood. Other functions like average can also be applied.

The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 along both width and height, discarding 75% of the activations of each depth slice. Stride is by how many input units the pooling filter is shifted for each pooling operation. Every max operation would in this case be taking a max over 4 numbers. This process is depicted in Figure 2.16.

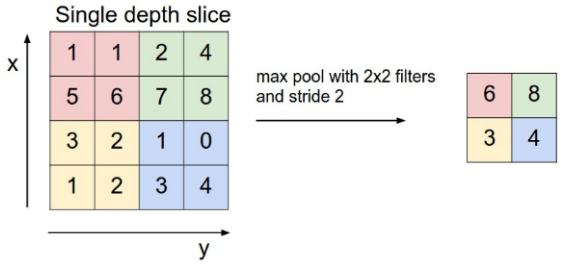


Figure 2.16: The most common downsampling operation is max, giving rise to max pooling, shown in this picture with a stride of 2. That is, each max is taken over 4 numbers.

2.1.7 Capacity, Over-fitting and Underfitting

The main challenge in machine learning is to perform well on new, previously unseen inputs, not just those on which the model was trained. The ability to perform well on previously unobserved inputs is called generalization.[12]
When training a model, an error measure on the training set called the training error is computed, and the goal is to reduce this training error. This may look simply like an optimization problem. What separates machine learning from optimization is that the generalization error, also called the test error, is desired to be low as well. The generalization error is defined as the expected value of the error on a new input. Here the expectation is taken across different possible inputs, drawn from the distribution of inputs it is expected for the system to encounter in practice.
The factors determining how well a machine learning algorithm will perform are its ability to:[12]

- Make the training error small, which correspond to a model with low *bias*.
- Make the gap between training and test error small, which corresponds to a model with low *variance*.

These two factors correspond to the two central challenges in machine learning: underfitting and overfitting. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large. It is possible to control whether a model is more likely to overfit or underfit by altering its capacity. A model's capacity is its ability to fit a wide variety of functions. Models with low capacity may struggle to fit the training set. Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.
In Figure 2.17 these two phenomena are illustrated for different model capacities.

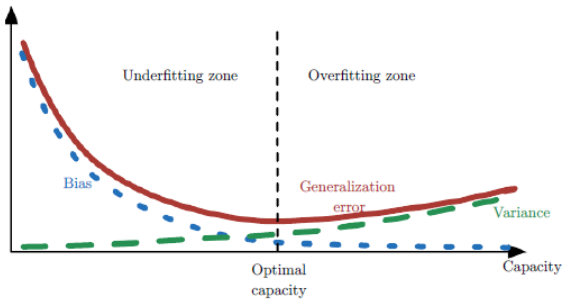


Figure 2.17: As capacity increases, bias tends to decrease and variance tends to increase. Picture taken from [12].

One way to control the capacity of a learning algorithm is by choosing its hypothesis space, i.e. the set of functions that the learning algorithm is allowed to select as being the solution. For example, a linear regression algorithm has the set of all linear functions of its input as its hypothesis space. If linear regression is generalized to also include polynomials, rather than just linear functions in its hypothesis space, the model's capacity would be increased.

Machine learning algorithms perform best when their capacity is appropriate for the true complexity of the task they need to perform and the amount of training data they are provided with. Models with insufficient capacity are unable to solve complex tasks. Models with high capacity can solve complex tasks, but when their capacity is higher than needed to solve the present task they may overfit.
In Figure 2.18 this principle is shown. The picture shows a linear, quadratic and degree-9 predictor attempting to fit a problem where the true underlying function is quadratic. The linear function is unable to capture the curvature in the true underlying problem, so it underfits. The degree-9 predictor is capable of representing the correct function, but it is also capable of representing infinitely many other functions that pass exactly through the training points. There is little chance of choosing a solution that generalizes well when so many different solutions exist. In this example, the quadratic model is perfectly matched to the true structure of the task so it generalizes well to new data.

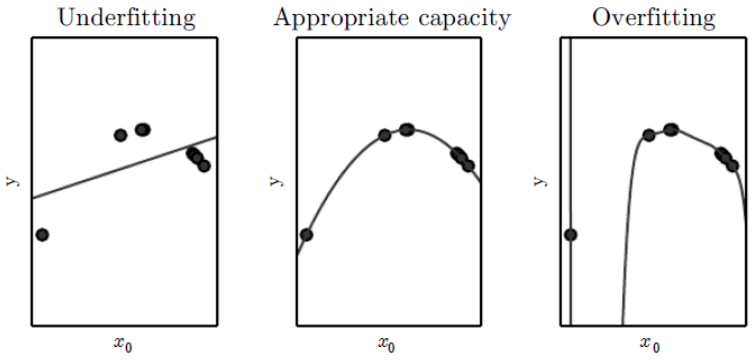


Figure 2.18: Three models with different capacities being fitted to a training set. Picture taken from [12].

2.1.8 Regularization methods

Deep neural networks contain multiple non-linear hidden layers and this makes them very expressive models that can learn very complicated relationships between their inputs and outputs. With limited training data, however, many of these complicated relationships will be the result of sampling noise, so they will exist in the training set but not in real test data, even if it is drawn from the same distribution. However, as it was already mentioned, the goal of network training is not to learn an exact representation of the training data itself, but rather to build a statistical model of the process which generates the data. This is important if the network is to exhibit good generalization.[15] Regularization can be thought of as any modification made to a learning algorithm that is intended to reduce its generalization error but not its training error.[12]
In Figure 2.18 three situations with different model's capacities were illustrated. Each of them, ordered from left to right, was characterized by:

1. Excluding the true data generating process (corresponding to underfitting and inducing bias).
2. Matching the true data generating process.

3. Including the generating process but also many other possible generating processes (the overfitting regime where variance rather than bias dominates the estimation error.

The goal of regularization is to take a model from the third regime into the second regime.[12]

2.1.8.1 Weight decay

Many regularization approaches are based on limiting the capacity of the models by adding a parameter norm penalty $\Omega(\theta)$ to the loss or the cost function J:

$$\widetilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta) \tag{2.15}$$

One of the simplest forms of regularizer is called *weight decay* and consists of the sum of the squares of the adaptive parameters in the network

$$\Omega = \frac{1}{2}\sum_i w_i^2 \tag{2.16}$$

where the sum runs over all weights and biases.
As it was shown in Figure 2.18, to produce an over-fitted mapping having regions of large curvature requires relatively large values for the weights. A network mapping having small values of the weights is less prone to overfitting.
A weight decay regularized model has the following total objective function:

$$\widetilde{J}(w; X, y) = \frac{\alpha}{2}w^T w + J(w; X, y) \tag{2.17}$$

with the corresponding parameter gradient

$$\nabla_w \widetilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y) \tag{2.18}$$

where α is a tunable parameter than controls the amount of regularization to be included in the model. To take a single gradient step to update the weights, the following update is performed:

$$w \leftarrow w - \epsilon\left(\alpha w + \nabla_w J(w; X, y)\right) \tag{2.19}$$

which can also be written as

$$w \leftarrow (1 - \epsilon\alpha)w - \epsilon\,\nabla_w J(w; X, y)) \tag{2.20}$$

It can be seen from the previous equation that the addition of the weight decay term has modified the learning rule to multiplicatively shrink the weight vector by a constant factor on each step, just before performing the usual gradient update. Therefore, by using a regularizer of the form Eq.2.16 the weights are encouraged to be small.

2.1.8.2 Dropout

The term *dropout* refers to dropping out some of the units (neurons) in a neural network. Dropping a unit out means temporarily removing it from the network, along with all its incoming and outgoing connections, as shown in Figure 2.19. The choice of which units to drop is random. In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks.[18]

Applying dropout to a neural network corresponds to sampling a *thinned* network from it. The thinned network consists of all the units that survived the dropout operation (Figure 2.19b). A neural network with n units can be seen as a collection of 2^n possible thinned neural networks which share weights. For each presentation of each training case, a new thinned network is sampled and trained. So training a neural network with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely, if at all.

Recall that the feed-forward operation of a standard neural network can be described as

$$z_i^{[l+1]} = w_i^{[l+1]} a^{[l]} + b_i^{[l+1]}, \tag{2.21}$$

$$a_i^{[l+1]} = h(z_i^{[l+1]}), \tag{2.22}$$

where $a^{[l]}$ denotes the vector of activations from the previous layer, $w^{[l+1]}$ and $b^{[l+1]}$ are the weights and biases of layer $[l+1]$ and h denotes the activation function operation. With dropout, the feed-forward operation becomes:

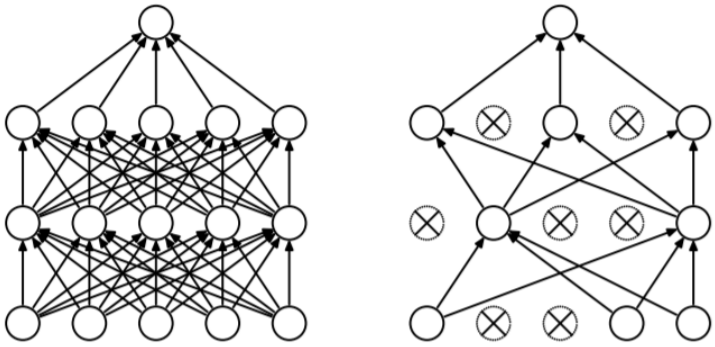
$$r_j^{[l]} \sim \text{Bernoulli}(p), \tag{2.23}$$

$$\widetilde{a}^{[l]} = r^{[l]} * a^{[l]}, \tag{2.24}$$

$$z_i^{[l+1]} = w_i^{[l+1]} \widetilde{a}^{[l]} + b_i^{[l+1]}, \tag{2.25}$$

$$a_i^{[l+1]} = h(z_i^{[l+1]}). \tag{2.26}$$

where, for any layer l, $r^{[l]}$ is a vector of independent Bernoulli random variables each of which has probability p of being 1. This vector is multiplied elementwise with the outputs of that layer, $a^{[l]}$, to create the thinned outputs $\widetilde{a}^{(l)}$. The thinned outputs are then used as input to the next layer.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.19: Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. Picture taken from [18].

2.2 Feature generation

In this section the main theory behind the process of feature generation for the audio files used for the training of the models will be presented.
Feature generation is the process of taking raw, unstructured data and defining features, i.e. variables, for potential use in the statistical analysis. This is a really important step in order to provide the machine learning algorithm with the best possible data to perform in the best possible way. An example showing the importance of this step is shown in Figure 2.20.

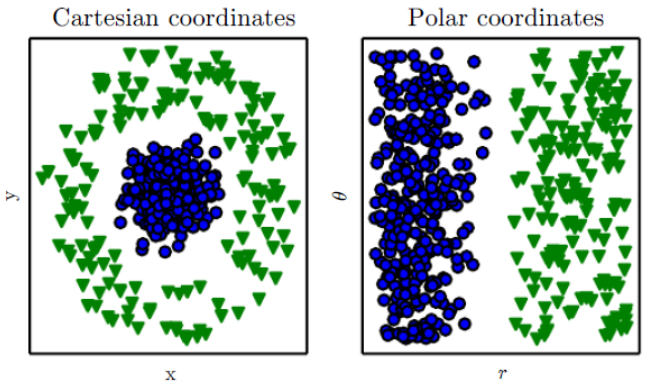


Figure 2.20: Example of how different representations may turn an impossible task (separating the two categories drawing a straight line) into an easy task for a computer to solve. Picture taken from [12].

2.2.1 Fourier transform

The Fourier transform allows the conversion of signals between the time and the frequency domains. The continuous Fourier transform of a signal $s(t)$ is defined as

$$S(\omega) = \int_{-\infty}^{\infty} s(t)e^{-j\omega t}dt \tag{2.27}$$

The discrete counterpart, i.e. the Discrete Fourier Transform (DFT) is defined as

$$S(k) = \sum_{n=0}^{N-1} s(n)e^{-j\frac{2\pi}{N}kn} \tag{2.28}$$

where k refers to the Fourier component number, or frequency bin, and n to the sample number.[19] In general, $S(k)$ is complex valued, and usually the only interesting part is the magnitude of it. This is called a magnitude spectrum $|S(\omega)|$. A power spectrum is calculated by squaring the magnitude spectrum, i.e. $|S(\omega)|^2$.
The frequency step Δf of a DFT is calculated as the sampling frequency f_s divided by the number of frequency components N_f in the spectrum.

$$\Delta f = \frac{f_s}{N_f} = \frac{1}{\Delta t \cdot N_f} \tag{2.29}$$

The number of frequency components N_f is identical to the number of samples of the signal used in the transform. However, the FT returns a double sided spectrum containing negative frequencies which are usually discarded. This means that the single sided spectrum contains half the number of frequency bins than the number of samples in the time domain.

The product $\Delta t \cdot N_f$ is equal to the period time T, and hence:

$$\Delta f = \frac{1}{T} \tag{2.30}$$

An important conclusion from this last formula is that the sampling rate of the signal has no direct influence in the frequency resolution of the corresponding FT. However, the sampling rate does have an influence in the frequency coverage of the FT. Higher sampling frequency allows for higher frequency coverage as established by the Nyquist-Shannon sampling theorem. In Figure 2.21 a picture showing the fourier transform of a sine wave can be seen.

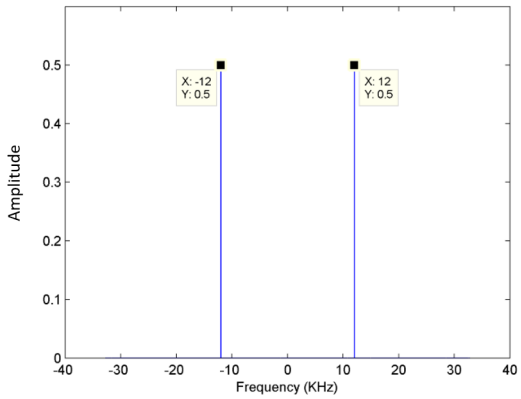


Figure 2.21: Fourier transform of a sine wave of frequency 12 kHz, showing both negative and positive frequencies.

2.2.2 Spectrogram and STFT

The DFT assumes the signal to be transformed as being periodic. That is, the end of the signal is connected to the beginning of the signal. This can cause discontinuities at the signal edges, that would show up in the spectrum as non-zero amplitudes at other frequencies than the ones present in the signal.[19] This spread of the amplitude caused by the assumed periodicity of the signal is commonly referred to as *leakage*.

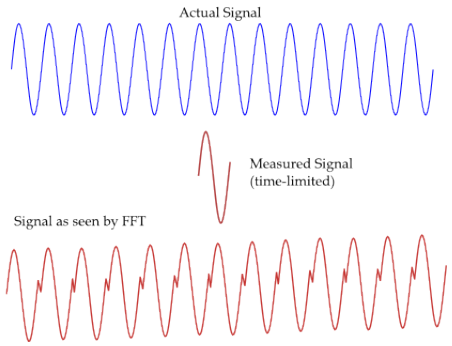


Figure 2.22: An example of how the DFT periodicity assumption might lead to spectral leakage. Picture taken from [20].

One solution to avoid this *leakage* is to window the signal before taking the DFT. One commonly used window is the *Hanning window*, which is defined as:

$$h(n) = \frac{1 - \cos\left(\frac{2\pi n}{N+1}\right)}{2} \qquad n = 1, 2, ..., N \tag{2.31}$$

where n is the sample number and N the total number of samples. A Hanning window function is depicted in figure 2.23.

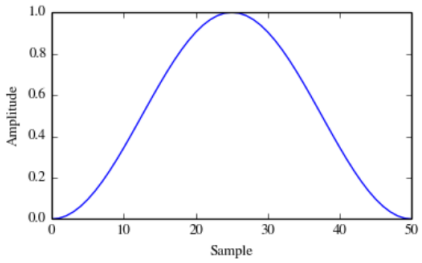


Figure 2.23: Hanning window function, N = 50

The spectrogram is nothing but several DFT stacked together, which allows to visualize how frequencies change over time. In order to do so, a time window needs to be chosen for which to compute the DFT. This time window is usually chosen to overlap with the following time frame. This is done to preserve all the information of the signal that would otherwise be lost due to two consecutive windowing operations.

2.2.3 Mel-frequency spectrogram

The mel scale is a logarithmic frequency scale that tries to better adapt to human hearing. It was developed by experimenting with the human interpretation of pitch in 1940's with the sole purpose of describing the human auditory system on a linear scale.[21] The experiment showed that the pitch is linearly perceived in the frequency range 0-1000 Hz. Above 1000 Hz, the scale becomes logarithmic. An approximated formula widely used for mel-scale is shown below:

$$f_{mel} = 2595 \log_{10}\left(1 + \frac{f_{Hz}}{700}\right) \tag{2.32}$$

where f_{mel} is the resulting frequency on the mel-scale measured in mels and f_{Hz} is the frequency measured in Hz. The plot of this function can be seen in Figure 2.24.

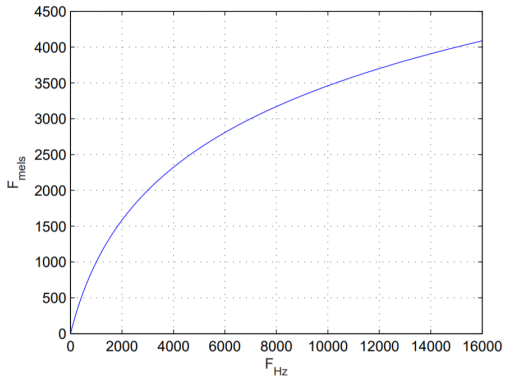


Figure 2.24: Mel frequency scale vs linear frequency scale

Basically the mel-frequency spectrogram is the regular spectrogram of a signal, mapped onto the mel-frequency scale. In addition to that, the mel-spectrogram is usually grouped into frequency bands. This grouping is obtained by multiplying the discrete spectrogram with a mel-scaled filterbank made up of several overlapping triangular windows. The discrete frequency bins are therefore mapped into a predefined number of mel-frequency bands.
Since the frequency range of the signal, determined by its sampling rate, is distributed in uniformly-spaced bands along the mel-scale, this has the consequence that low frequencies are emphasized over high frequencies, for which a more coarse frequency resolution is obtained. In Figure 2.25 the distribution of the different triangular filterbanks over the two frequency scales can be seen.

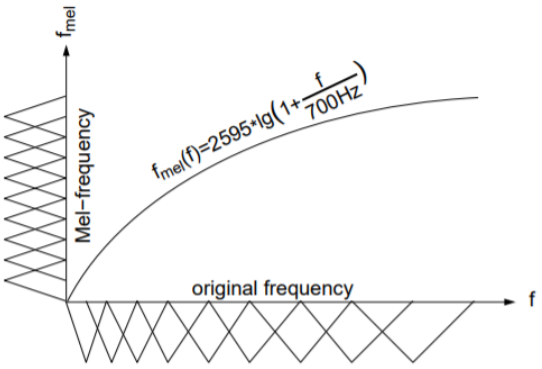


Figure 2.25: Schematic plot of different triangular filterbank implementations. The filters are either uniformly distributed on the mel-warped spectrum, or non uniformly at the original spectrum. Picture taken from [21].

3
Methods

In this section, all the crucial steps performed for the development of this master thesis will be described. With the information presented in this chapter, the reader will have enough information to understand the procedure and to potentially apply it to their own project.

3.1 Software

All the programming was performed in Python, version 3.6.3. Besides all the general libraries used for data processing and analysis in Python such as Numpy or Matplotlib, four specific libraries were used in this project: The library SoundFile was used for reading and writting audio files. The audio analysis library Librosa[22] was used for the resampling of the audio files as well as for generating the mel-frequency spectrograms that were fed as training data to the network. For the neural network programming part, Google's library Tensorflow was used, version 1.1.0. The library Scikit-learn was used for calculating the confusion matrixes shown in Chapter 4, Results.
All the code used in this project can be found in `https://github.com/santigijon/Audio-event-classification-using-neural-networks`.

3.2 The dataset: UrbanSound8K

UrbanSound8K is the dataset used in this project. It is an open source dataset, published by Justin Salomon, Christopher Jacoby and Juan Pablo Bello.[3] The dataset contains 8732 labeled sound excerpts of less than or equal to four seconds duration of urban sounds from 10 classes: air conditioner, car horn, children playing, dog bark, drilling, enginge idling, gun shot, jackhammer, siren, and street music. All excerpts are taken from field recordings uploaded to www.freesound.org. The files are pre-sorted into ten folds to help in the reproduction of and comparison with classification results reported in other works.
All 8732 audio files of urban sounds are in WAV format. The sampling rate, bit depth, and number of channels are the same as those of the original file uploaded to Freesound, and hence, they vary from file to file.
In Figure 3.1 a graph showing the distribution of foreground and background slices per class is shown. This characterization of the audio files is a subjective judgment by the authors of the dataset.

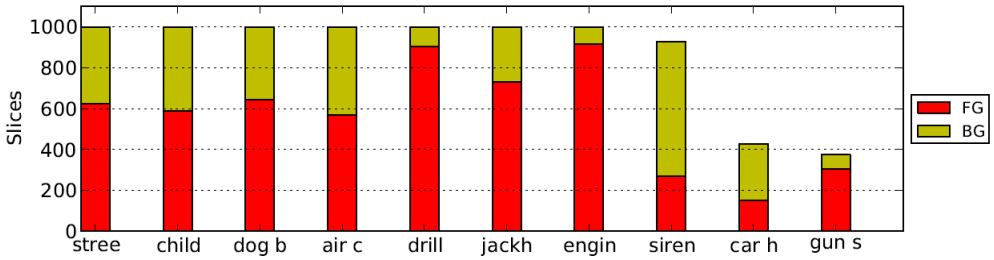


Figure 3.1: Slices per class in UrbanSound8K, breakdown by foreground (FG) / background (BG). Picture taken from [3].

The total values of the histogram shown in Figure 3.1 can be seen in table 3.1.

Class	Air conditioner	Car horn	Children playing	Dog bark	Drilling
Slices	1000	429	1000	1000	1000
Class	Engine idling	Gun shot	Jackhammer	Siren	Street music
Slices	1000	374	1000	929	1000

Table 3.1: Number of total audio slices for the different classes in UrbanSound8K dataset

A more thorough study of the dataset was performed by analyzing the different file lengths present in the dataset, for the different classes. The files were splitted into one, two and three seconds duration clips. The number of obtained clips resulted in the distributions shown in Tables 3.2, 3.3 and 3.4.

Class	Air conditioner	Car horn	Children playing	Dog bark	Drilling
Slices	3994	953	3951	2989	3449
Class	Engine idling	Gun shot	Jackhammer	Siren	Street music
Slices	3915	444	3513	3617	4000

Table 3.2: Number of one second duration slices for the different classes in UrbanSound8K dataset

Class	Air conditioner	Car horn	Children playing	Dog bark	Drilling
Slices	1997	439	1971	1427	1676
Class	Engine idling	Gun shot	Jackhammer	Siren	Street music
Slices	1946	129	1697	1801	2000

Table 3.3: Number of two second duration slices for the different classes in UrbanSound8K dataset

Class	Air conditioner	Car horn	Children playing	Dog bark	Drilling
Slices	997	218	980	697	829
Class	Engine idling	Gun shot	Jackhammer	Siren	Street music
Slices	973	34	838	901	1000

Table 3.4: Number of three second duration slices for the different classes in UrbanSound8K dataset

It can be observed that there are clear differences in the duration of the audio files between classes. For example, all the audio excerpts belonging to the class *Street music* are longer than three seconds, whereas only 34 audio excerpts of the class *Gun shot* are longer than three seconds.

3.3 Audio files processing

In order to have all the audio files with the same characteristics, they were all resampled to a sampling rate of 22050 Hz and converted to mono channel. This sampling rate allows for a frequency coverage up to 10 kHz, which is more than necessary, as shown in [23]. The amplitude range of most of the audio files is (-1,1), although there are certain floating point file formats that contain values greater than (-1,1). However, this was observed just for very few of the examples.

The network arquitectures considered in the present study do not accept inputs of different sizes, therefore it is required that all inputs have the same duration. This duration was chosen to be three seconds. Therefore a last step in the processing of the audio files was to zero-pad all the files that did not reach the three seconds duration. Comparing tables 3.4 and 3.1 it can be seen that a different proportion of zero-padded audio files per class is obtained with this method. The authors of [4] used a different technique for extending the audio files. They appended the same file as many times as needed until reaching the desired three seconds file duration.

3.4 Multi-label dataset generation

UrbanSound8K is a dataset of individual environmental sounds. In order to study the multi-label classification task, a dataset where sounds are combined was needed. This dataset was obtained by combining audio samples of the UrbanSound8K dataset. This was done in such a way that combinations of two classes are present in each of the new audio samples. The distribution of files per folder for the different classes is shown in Figure 3.5. The lowest number of instances of one class across folders was chosen as the number of instances of each class and folder considered for the new dataset generation.

Folder / Class name	fold0	fold1	fold2	fold3	fold4	fold5	fold6	fold7	fold8	fold9
Air conditioner	100	100	100	100	100	100	100	100	100	100
Car horn	36	42	43	59	98	28	28	30	32	33
Children playing	100	100	100	100	100	100	100	100	100	100
Dog bark	100	100	100	100	100	100	100	100	100	100
Drilling	100	100	100	100	100	100	100	100	100	100
Engine idling	96	100	107	107	107	107	106	88	89	93
Gun shot	35	35	36	38	40	46	51	30	31	32
Jackhammer	120	120	120	120	120	68	76	78	82	96
Siren	86	91	119	166	71	74	77	80	82	83
Street music	100	100	100	100	100	100	100	100	100	100

Table 3.5: Number of audio files per folder. The lowest number of files of one class in a folder is highlighted in green.

The organizational structure of the 10 folders was maintained. A combination of 28 audio files per class per folder was created where, for each of the 28 elements of one class in one folder, a combined file was created by combining this element with one of the elements of each other class in that same folder. Thus, since there are ten classes, for a set of one example per class (ten examples), $\sum_{n=1}^{9} n = 45$ combinations are created. In Figure 3.2 a picture summarizing the process is provided, where A to J represent the ten different classes in the dataset. The combination of the 28 files per folder for the 10 different folders led to a dataset of 12600 audio files, which keeps the size of the dataset within a reasonable size compared to the original dataset and therefore does not vastly increment computational requirements.

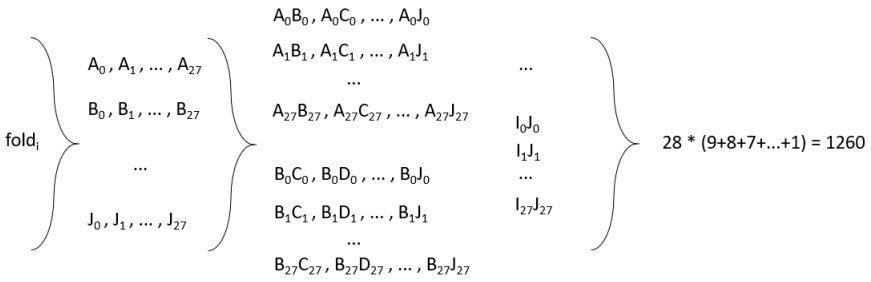


Figure 3.2: Sketch showing how the multi-label dataset generation was performed.

3.5 Input features to the networks

For both the single-label and multi-label tasks the input features to the network are mel-frequency spectrograms of 60 frequency bands and three seconds duration. The spectrograms were calculated with a window (Hanning) size of 1024 samples and a hop size of 512 samples. The mel-spectrogram was chosen since it provides a finer resolution at lower frequencies, where most of the relevant acoustic information is present.[23]

In Figure 3.3 some of the images that were fed to the network for training can be seen. The units are in dB relative to the maximum value of the power spectrogram of each sample.

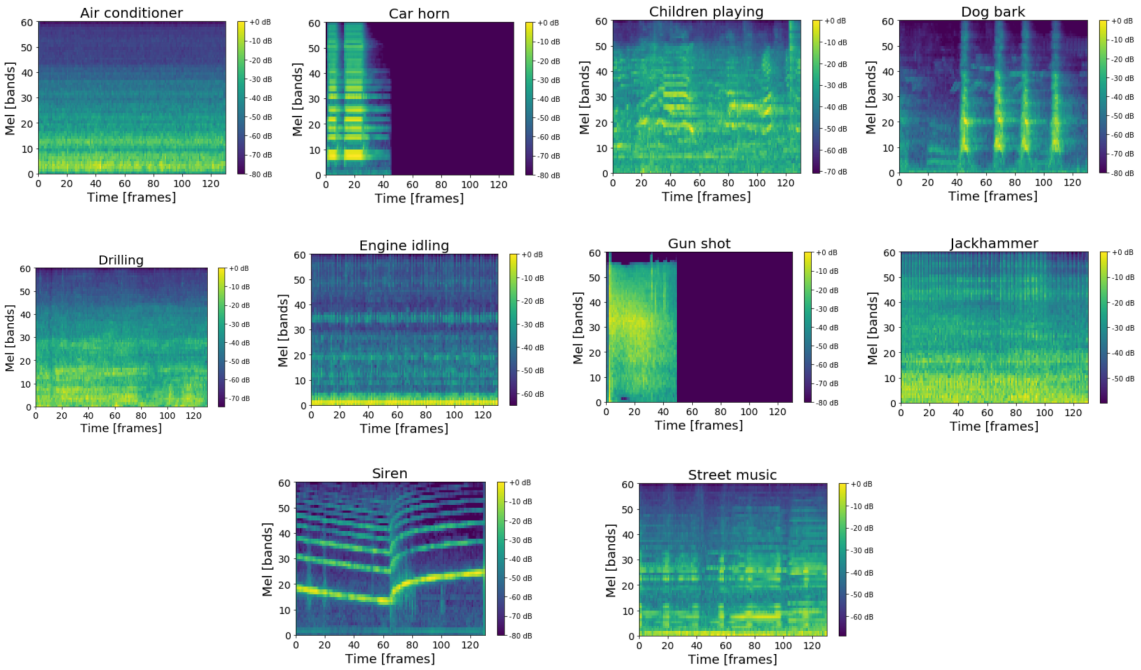


Figure 3.3: Mel-frequency spectrograms of ten random samples belonging to the ten different classes of UrbanSound8K.

In Figure 3.4 two mel-frequency spectrograms of two random samples belonging to the dataset generated for the multi-label task can be seen. This was the type of training data used for the multi-label classification task.

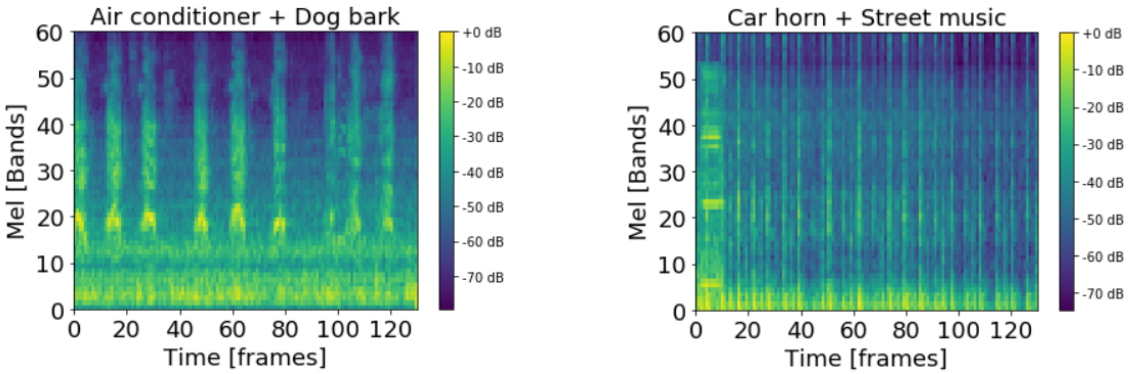


Figure 3.4: Mel-frequency spectrograms of two mixed samples.

3.6 Network arquitectures

In this section the two network arquitectures used in the study are presented.

3.6.1 Single-label classification

In Figure 3.5 a sketch of the network arquitecture used for the single-label classification task can be seen. In the figure, *Conv* stands for convolutional layer, *Pool*

stands for pooling layer, and *FC* stands for fully connected layer. The letters *f* and *s* stand for filter size and stride. And *ReLU* and *Softmax* are the activation functions of the specified layer. The model has a total of 151290 parameters.

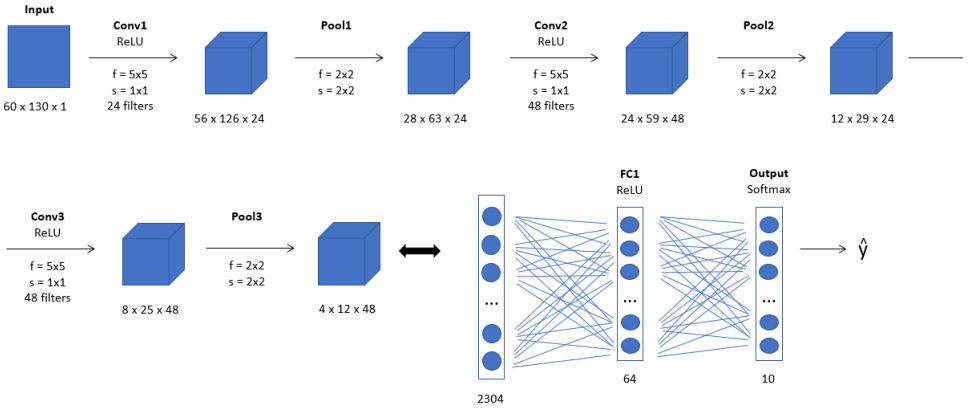


Figure 3.5: Network arquitecture for the single-label classification task.

3.6.2 Multi-label classification

In Figure 3.6 a sketch of the arquitecture used for the multi-label classification task can be seen. The model has a total of 299386 parameters.

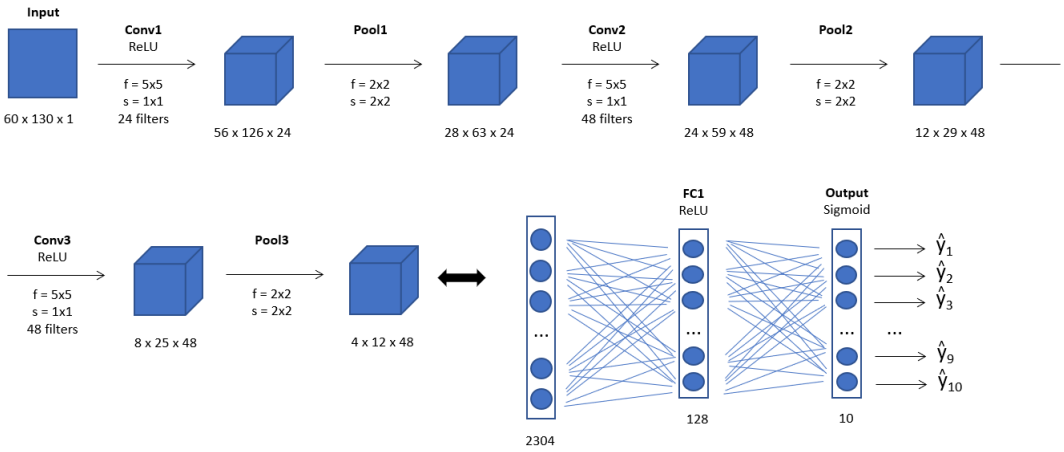


Figure 3.6: Network arquitecture for the multi-label classification task.

3.7 Dataset variations for the single-label classification task

In this section three variations of the UrbanSound8K dataset are explained. These variations were considered after observing the first results of the baseline model and the results after evaluating it on noise and amplitude corrupted inputs. The model was then trained on these variations in order to study different behaviours:

- Undersampling: To test whether reducing the number of instances per class improves accuracy or not.
- Data augmentation with BG noise: To find out how does data augmentation with noise affect robustness to noise intrusions.
- Random amplitude scaling: To find out if the network is capable of learning relative pattern differences in the audio files, or it can just rely on absolute values. To find out if it is possible to force the network to learn patterns in the audio signals.

3.7.1 Undersampling

A first attempt to reduce the effect of overfitting was to reduce the number of training examples in the classes where overfitting was more present, i.e. where the difference between training and test accuracy is the greatest. According to the results illustrated in Figure 4.1, these classes were judged to be *Air conditioner*, *Engine idling* and *Jackhammer*. Additionally, two other variations were implemented.
The A-E versions were obtained as follows: A is a modified version of UrbanSound8K where 651 audio files of the class *Air conditioner* were discarded. In B, 677 files of the class *Engine indling* were discarded. In C, 261 files of the class *Gun shot* were discarded. In D, 660 files of the class *Jackhammer* were discarded. In all A, B, C and D the files were discarded using a 66% random discard rule.[1]
In version E, a reduction of all the classes was performed. This was made to get an approximately balanced dataset and study the consequences. The resulting total number of audio examples per class can be seen in table 3.6. The total number of examples in the dataset was reduced from 8732 to 3355 examples.

Class	Air conditioner	Car horn	Children playing	Dog bark	Drilling
Slices	335	295	342	333	339
Class	Engine idling	Gun shot	Jackhammer	Siren	Street music
Slices	330	374	344	321	342

Table 3.6: Number of audio examples for the different classes in version E of the UrbanSound8K dataset

3.7.2 Data augmentation with noise

The dataset was augmented with pink noise levels of SNR +0 dB, i.e. for every three-seconds audio sample, another sample was created by adding pink noise to it. Thus the resulting dataset had 17464 audio samples.

3.7.3 Amplitude scaling

Two different scaling versions of the inputs to the network were considered:

[1] In Python: if random.random > 0.33, then discard example.

- Version 1: the samples were scaled so that their average mel-spectrogram magnitudes per class lied within the same amplitude range. This was done according to Figure 3.7.
- Version 2: the same first step as in Version 1 was applied, and then a second step was performed, where the samples were rescaled by a random factor $f \sim U(1, 16)$. This was done to increase amplitude variance in the dataset, in an attempt to encourage the network to learn relative amplitude patterns.

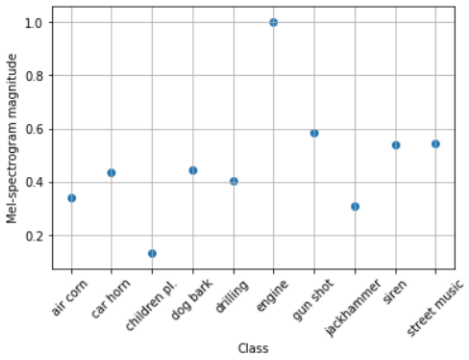


Figure 3.7: Original mel-spectrogram amplitudes averaged across instances of the same class, relative to class *Engine*.

3.8 Input corruption for the single-label classification task

In this section the two considered variations of the original dataset for a posterior evaluation to these variations are explained.

3.8.1 Noise addition

In order to study how different noise levels influence the accuracy prediction of the network, the original audio samples were combined with noise.
Pink noise was generated using the Voss-McCartney algorithm, and the SNR (signal to noise ratio) differences were calculated with respect to rms values as in:

$$\mathrm{SNR} = 10 \log_{10} \left(\frac{\mathrm{rms}_{\mathrm{signal}}}{\mathrm{rms}_{\mathrm{noise}}} \right) \tag{3.1}$$

This was done for six different SNR values: -5 dB, -2 dB, 0 dB, +2 dB, +5 dB and +8 dB. The lowest SNR was -5 dB, which was a very strong noise level in the signal as judged by the writer.
The noise was added to the files in two different ways. Firstly it was added only to the duration of the original file. In the second version the noise was added to the whole three seconds of the file.

3.8.2 Amplitude scaling

The second input corruption considered was amplitude scaling of the audio signals, in order to study how robust the network is to level differences. Four different level differences with respect to the original signal were considered: -12, -6, +6 and +12 dB, as in:

$$\text{Level}_{\text{difference}} = 20\log_{10}\left(\frac{\text{scaled}_{\text{signal}}}{\text{original}_{\text{signal}}}\right) \tag{3.2}$$

3.9 Training procedure and hyperparameter search

In this section, the details corresponding to the training procedure of the networks are explained. Adam[24] algorithm was used for the optimization of both tasks.

3.9.1 Single-label classification task

In Table 3.7, the final hyperparameter choice used for training of the baseline model is shown.The hyperparameter search was performed on a single folder due to computational reasons. A more thorough search of hyperparameters could be done by 10-fold cross validating for every choice of hyperparameters, but this was not the case in this work.

Learning rate	Batch size	Epochs	Weight decay factor	Dropout probability
0.001	50	50	0.0025	0.5

Table 3.7: Final choice of hyperparameters used during training.

Once the final set of hyperparameteres was chosen, the model was trained using 10-fold cross validation. During training one folder was always discarded in order to be consistent with previous research studies where they used this folder for validation purposes. The training was performed as shown in Table 3.8:

Discarded folder	fold0
Test folder	fold1
Training folders	fold2-fold9

Table 3.8: Folder arrangement for training, validation and testing.

For each new step of the cross-validation procedure, every folder index of Table 3.8 was incremented by one.
The dataset augmentation with background noise was the only variation where the hyperparameters were changed from the baseline model. A limited search of hyperparameters was performed and the resulting parameters are shown in Table 3.9.

Learning rate	Batch size	Epochs	Weight decay factor	Dropout probability
0.001	50	40	0.005	0.5

Table 3.9: Hyperparameters used in training for the model with noise augmented dataset.

3.9.2 Multi-label classification task

The final hyperparameter choice used for training of the model is shown in Table 3.10.

Learning rate	Batch size	Epochs	Weight decay factor	Dropout probability
0.001	40	120	0.00	0.5

Table 3.10: Final choice of hyperparameters used during training

The 1260 files corresponding to the combined files from folder 9 were used as test set, and those corresponding to folder 8 were used as validation set. 10080 files were therefore used as training set.

4
Results

In this chapter the results to the study will be presented. Each result is accompanied by a posterior discussion where the results are interpreted.

4.1 Network classification performance

In this section the prediction accuracy results for the baseline model and the three different variations are presented.

4.1.1 Baseline

The obtained results for the single-label classification task using the UrbanSound8K dataset are shown in Table 4.1. Following the same procedure as [4], accuracy was calculated as the average of the individual prediction accuracies across the 10 folders. The results reveal a high degree of overfitting to the training data, measured by the difference between training and test accuracy. The different explicit regularization techniques tried out during the hyperparameter search, i.e. dropout and weight decay, did not help to reduce this generalization gap, suggesting that generalization is a difficult task on this particular dataset. It is prone to overfitting due to the low intra-class variance of some of the classes. The main reason for this low variance is the fact that many of the sound excerpts of each class were extracted from the same audio file when the dataset was created.[3]
This bias is more pronounced for those classes whose original sound recordings had longer duration. It is less likely to find a sound recording of 20 seconds duration for the class *Gun shot*, whereas for the classes *Jackhammer* or *Air conditioner* that is many times the case.

Training accuracy	Test accuracy
90.60 %	64.82 %

Table 4.1: Training and test accuracy results (10-fold cross validated).

In Figure 4.1 the normalized confusion matrices for the training and test sets can be seen. This gives a clearer picture about the overfitting distribution over the different classes. The differences between training and test accuracies reveal the degree of overfitting present in the model for each class. A big difference means that the network is memorizing the training samples, but is not able to generalize well to

new unseen test samples. *Air conditioner*, *Engine idling* and *Jackhammer* are the classes where the highest degree of overfitting is observed. *Gun shot*, *Dog bark* and *Street music* are the classes where the lowest degree of overfitting is observed.

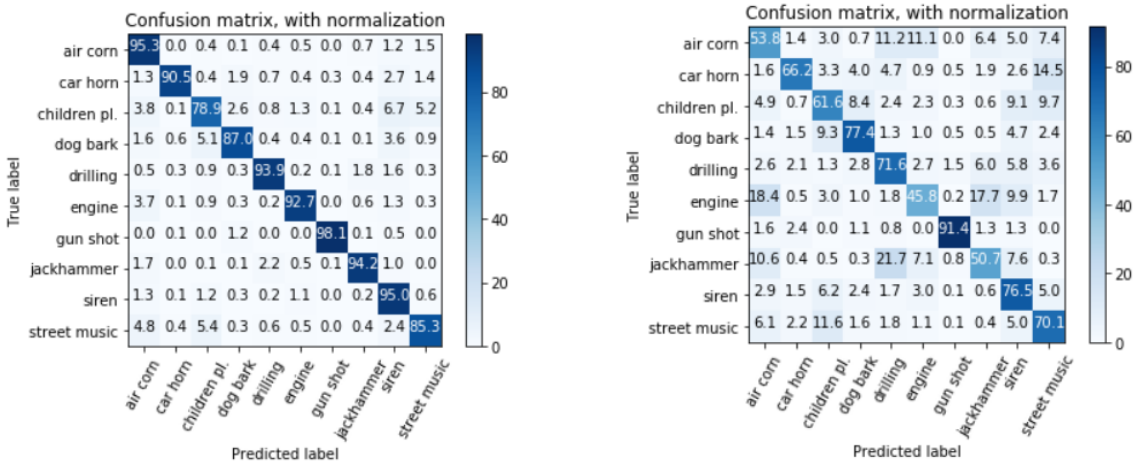


Figure 4.1: Training (left) and test (right) confusion matrices of the baseline model.

The following observations can be drawn from these results. For the three classes with the highest degree of overfitting:

- According to the aforementioned observations about the original duration of the audio files, these three are the classes presenting the lowest intra-class variance.
- All of them present a noisy sound nature (see Figure 3.3), making it difficult for the network to learn meaningful patterns.
- The highest confusions when classifying each of these classes occur between these three classes and also the class *Drilling*. This is a proof of the similarity of these four classes and the consequent difficulty of differentiating between them.

As for the classes with the lowest degree of overfitting:

- These are either transient sounds or classes with high degree of intra-class variance, such as *Street music*.
- The instances of the class *Street music* might have been extracted from the same audio file. However, they will still present a high degree of variance since music is a non-stationary process.
- A high variance among class instances forces the network to learn meaningful features of each class, since it cannot rely on any particular characteristic common only to the training data. Therefore the lower degree of overfitting for the *Street music* class.

By looking at Figure 3.1 it can be seen that the classes *Siren* and *Car horn* are the only classes where a higher number of background instances are present, compared to foreground instances. This explains why, besides being a transient (*Car horn*) and, in theory (see Figure 3.3), an easily identifiable sound (*Siren*), the network occasionally confuses them with sounds like *Street music* or *Children playing*, which are common background city noises.

4.1.2 Dataset variations

The results obtained when training the model on the three dataset variations introduced in Section 3.7 are presented in this subsection.

4.1.2.1 Undersampling

In Table 4.2 the changes in prediction accuracy per class when performing undersampling of the dataset can be seen. The values shadowed in color were calculated as: $accuracy\ version_i - baseline\ accuracy$, where $i \in (A, E)$.

Class / Version	AI	CA	CH	DO	DR	EN	GU	JA	SI	ST
Baseline	53.8	66.2	61.6	77.4	71.6	45.8	91.4	50.7	76.5	70.1
A	-34.6	-10.3	6.4	-2.2	2.2	7.0	-2.1	-0.6	0.2	2.5
B	-1.9	-0.9	3.1	-2.4	0.7	-17.6	-1.9	1.9	-0.2	4.8
C	-10.1	3.7	4.5	-1.0	4.2	-0.8	-14.5	4.2	-0.4	-1.7
D	-10.0	0.9	10.4	-1.5	1.5	5.6	0.5	-13.6	-4.4	-1.6
E	-23.7	-16.2	10.6	1.2	-3.4	0.5	0.0	4.1	-1.9	1.3

Table 4.2: Change in prediction accuracy (%) per class for different variations of the training set.

The study reveals that undersampling is not a good procedure for this particular dataset. Some classes remained unaltered to the variations, like *Dog bark*, *Siren* or *Street music*, whereas other classes like *Air conditioner* were singnificantly influenced. None of the variations had an overall positive influence in prediction accuracy.

An example of how reducing the training data can have dangerous consequences is shown in Figure 4.2. When the amount of samples of the class *Engine* is reduced, the accuracy of the class *Air conditioner* remains unaltered. However, it can be seen than now the network is confusing the *Engine* sounds with the class *Air conditioner*. This gives an intuition about how close some classes are to others and the consequent difficulty for the network to differentiate between them. Thus, reducing the amount of information about one of these classes makes the network more likely to get confused when trying to identify other classes that are similar to the class whose samples were reduced.

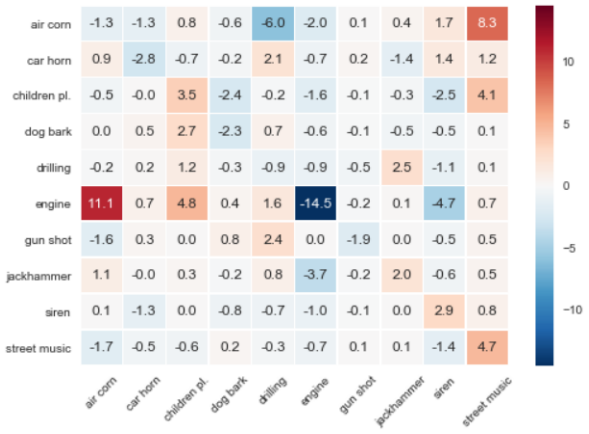


Figure 4.2: Change in prediction accuracy (%) when performing undersampling of class *Engine*. Calculated as: cnf matrix version B - cnf matrix baseline, normalized.

A remarkable observation is that in all versions B, C and D, reducing the samples of the corresponding class had a negative consequence on the prediction accuracy of that class. But when the reduction is done proportionally with other classes (version E), the accuracy of these classes did not decrease with respect to the baseline, suggesting that many of the audio slices are redundant and the network does not learn additional information from them. In fact, all these samples are doing is contributing to the overfitting of the network.

4.1.2.2 Data augmentation with noise

The results of the model when training the network on the augmented dataset are shown in table 4.3.

Training accuracy	Overall test accuracy	Test accuracy on clean inputs	Test accuracy on noisy inputs
91.06 %	63.24 %	64.23 %	62.06 %

Table 4.3: Training and test accuracy results for the noise augmented model (10-fold cross validated).

Comparing this table to table 4.1 it can be seen that a reduction in the generalization gap has not been achieved, suggesting that this kind of data augmentation is not favorable for this particular dataset when applied to all classes. In Figure 4.3 the confusion matrix comparison between baseline and background noise augmented can be observed. It can be seen that the prediction accuracy of some classes such as *Air conditioner* was negatively influenced by the noise augmentation, whereas for other classes like *Children playing* or *Drilling* it was improved. This suggests that applying class-conditional data augmentation would improve the results. A reason for the no improvement of the overall predictions may be that this particular noise has a similar structure to the noise present in certain classes, consequently biasing the predictions towards those classes, like *Drilling* or *Children playing*. Therefore

applying data augmentation with different noise types and levels would most likely improve the results.

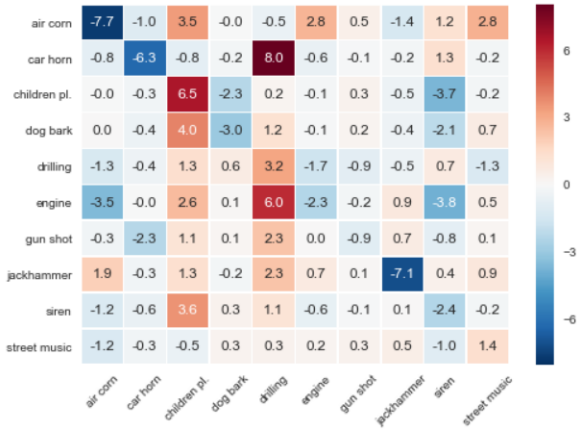


Figure 4.3: Change in prediction accuracy (%) per class when performing data augmentation with noise. Calculated as: cnf matrix augmented - cnf matrix baseline, normalized.

4.1.2.3 Amplitude scaling

The results after training on the scaled dataset are shown in Table 4.4 for the two versions presented in Section 3.7.3.

Version	Training accuracy	Test accuracy
1	90.46 %	66.56 %
2	86.36 %	63.57 %

Table 4.4: Training and test accuracy results (10-fold cross validated) for the amplitude scaled model.

The results to Version1 show that the generalization gap was reduced by 2% with respect to baseline (see Table 4.1). This suggests that having the input scales in a smaller range helps the learning process of the network. The results to Version2 do not bring any significant conclusion in terms of accuracy, but they do in terms of robustness to amplitude corruption of the inputs as it will be shown in the next section.

4.2 Network robustness to input corruption

In this section the results obtained after evaluating the different models with two different types of input corruptions are presented. These two types of input corruption are noise addition and amplitude scaling.

4.2.1 Noise addition

In Table 4.5 the different accuracy drops when noise is added to the inputs are shown for both noise configurations.

SNR (dB)	-5	-2	+0	+2	+5	+8
Noise applied to original sound duration	44.82	24.04	11.85	4.91	0.58	0.38
Noise applied to the 3 seconds extended audio file	46.58	26.48	14.24	6.45	1.41	0.73

Table 4.5: Decrease in prediction accuracy (%) when the audio files are corrupted with pink noise. Differences with respect to baseline (see Table 4.1).

Looking at the results, it can be concluded that SNR levels from +5 dB and above have no meaningful effect in the prediction accuracy of the network. On the contrary, SNR levels below +2 dB have a very pronounced effect in the prediction accuracy. To get a wider picture of the effect that these noise additions have into the classification accuracy, the confusion matrix for one of the considered noise levels is shown in Figure 4.4.

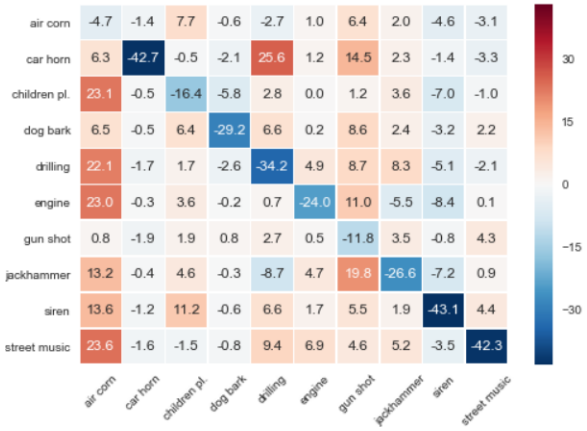


Figure 4.4: Normalized confusion matrix evaluated on noise corrupted inputs of SNR -5 dB. Calculated as the difference between the evaluated-on-corrupted-input confusion matrix and the baseline confusion matrix, normalized.

Looking at the confusion matrix, a clear decrease in accuracy for all the classes is observed. However, it can be seen that not all classes are influenced alike. *Air conditioner* is the class attracting the majority of confusions, which indicates that the network has learned to detect some features corresponding to this class that are common to the noise used in the input corruption. This explains also why this class is the least affected by the noise corruption, its recall has increased but its precision has decreased.
In Table 4.6 the prediction accuracy results of the data augmented model when evaluated on noise corrupted inputs are shown. In order to make the results comparable to those in Table 4.5, the network was evaluated on noise corrupted inputs only.

Comparing the results from Tables 4.5 and 4.6, it is clear that performing data augmentation with noise substantially helps dealing with noise corrupted inputs. As expected, the network has become more invariant to this type of inputs, being able to correctly identify them, no matter the presence of the noise. And it has not only helped to improve robustness to the noise level used in the data augmentation, but also to other noise levels.
The negative values observed on the table are due to the comparison with the overall accuracy test of the model. Since the model predicts with more accuracy clean inputs (see Table 4.3), as the corrupted audio files approach clean inputs (higher values of SNR) the 64.23 % accuracy observed on clean inputs is approached, and thus the difference $63.24\,\% - 64.23\,\% = -0.99\,\%$.

SNR (dB)	-5	-2	+0	+2	+5	+8
Noise applied to original sound duration	25.98	4.94	1.18	-0.45	-0.59	-0.75

Table 4.6: Decrease in prediction accuracy (%) when the audio files are corrupted with pink noise. Differences with respect to augmented baseline (see Table 4.3, column 2).

4.2.2 Amplitude scaling

In Table 4.7 the decrease in prediction accuracy when the network is evaluated on amplitude scaled versions of the inputs is shown. The variations in the prediction accuracies when the network is evaluated on amplitude corrupted inputs indicate that the network is sensitive to level differences. Meaning that the network is not learning relative amplitude patterns . However, it might have learned that certain class A usually have a strong energy band at 1 kHz for example. And now when increasing the levels of all the inputs, some other class B with previously medium-level energy at 1 kHz gets amplified, and then the network confuses it with the class A. It does not necessarily mean that the network only learned absolute amplitude patterns. It might have learned to localize where those patterns occur, even if it has not learned to identify relative amplitude patterns, which would be the ideal case.

Level difference	-12 dB	-6 dB	+6 dB	+12 dB
Accuracy decrease	17.06 %	4.76 %	1.68 %	8.46 %

Table 4.7: Decrease in prediction accuracy when the audio files were scaled in amplitude. Differences with respect to baseline (see Table 4.1).

In Figure 4.5 the confusion matrices when the network was evaluated on amplitude corrupted inputs can be seen. By looking at the +12 dB level differences, the two most pronounced changes happen for the class *Jackhammer* getting confused with the class *Engine*, and the class *Children playing* getting confused with the class *Street music*. Looking at Figure 4.6 it can be seen that on average, the class *Children playing* has lower amplitude values than the class *Street music*, and both

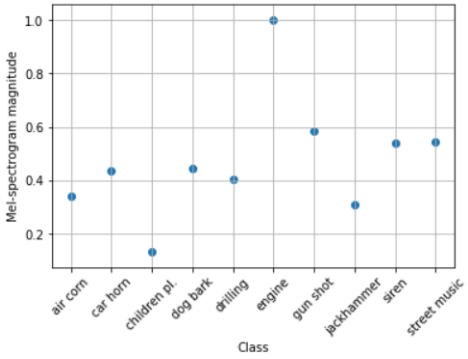


Figure 4.6: Mel-spectrum magnitudes averaged across instances of the same class, relative to class *Engine*, from the original dataset.

classes have a similar sound structure. This has probably lead the network to confuse between these two classes. The same confusion might have happened between *Jackhammer* and *Engine*, confirming that the network has learned some absolute amplitude frequency related features.

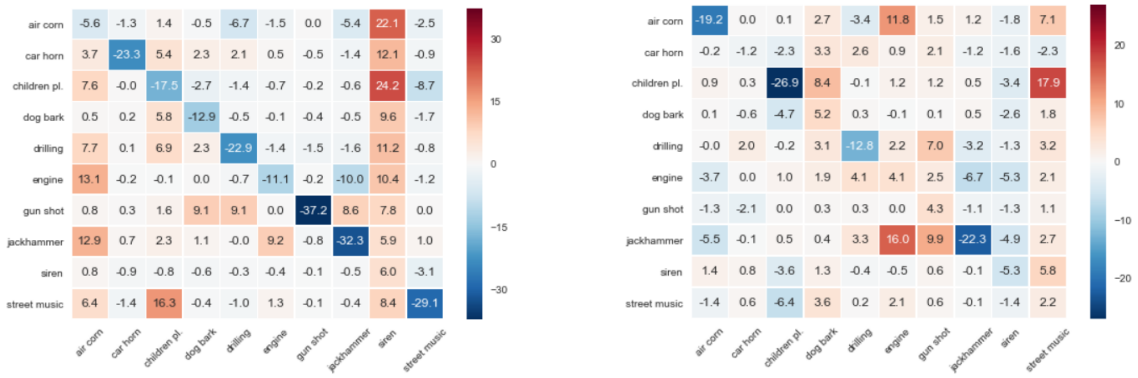


Figure 4.5: Normalized confusion matrices with respect to baseline when the baseline model is evaluated on amplitude corrupted inputs for different levels, -12 dB (left) and +12 dB (right).

In Table 4.8 the accuracy drops when evaluating the two models on amplitude corrupted inputs are shown. Version1 had no overall difference regarding robustness to amplitude corruption of the inputs with respect to the drops in the baseline model. However, Version2 presented improved robustness to amplitude corruptions when the amplitude was increased. On the contrary, it performs worse than the baseline when the scaling factor is <1.

In Figure 4.7 two confusion matrices when evaluating the model Version2 on corrupted inputs are shown. It can be seen that, when evaluating the model on higher levels corrupted inputs, the only class whose accuracy noticeably decreases is *Air conditioner*. The model is really sensitive to reductions in the input levels, as there is a clear bias towards the class *Siren* whose origin is hard to explain.

Level difference	-12 dB	-6 dB	+6 dB	+12 dB
Accuracy decrease Version1	17.90 %	5.58 %	1.99 %	8.25 %
Accuracy decrease Version2	30.73 %	16.48 %	1.11 %	0.40 %

Table 4.8: Decrease in prediction accuracy when the audio files were scaled in amplitude. Differences with respect to corresponding test results (see Table 4.4).

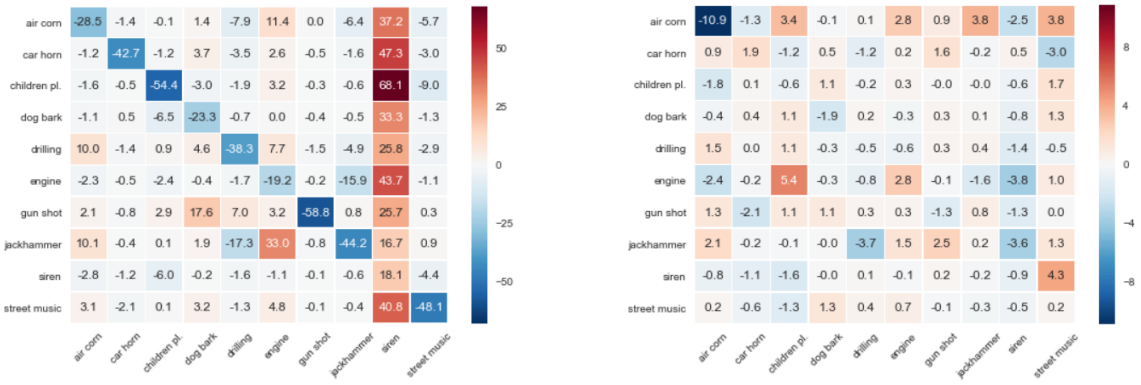


Figure 4.7: Normalized confusion matrices with respect to Version2 (see Table 4.4) when the model Version2 is evaluated on amplitude corrupted inputs for different levels, -12 dB (left) and +12 dB (right).

4.3 Multi-label classification

The multi-label prediction results are shown in Table 4.9. The results show that the task of multi-label classification of environmental sounds is clearly feasible. Neural networks are capable of learning representations in complicated mixed environments where more than one sound source is present. This is a powerful and important ability of neural networks because having more than one sound source would always be the case in real environmental sound classification scenarios.

Due to the nonexistence of previous studies on this task using the UrbanSound8K dataset, a benchmark for result comparison was generated considering the accuracy metrics for a random classifier that would indicate the presence of each class, 50 % of the time. This gives the results shown in Table 4.10.

Further research is needed to draw any conclusions on this but, the positive results on the multi-label task may suggest that, even when using a limited dataset where only 280 instances of each class were considered, the multi-label task is likely to introduce some inherent regularization that encourages the network to learn more robust and features of each class, since some of the input features are masked at times.

Precision	Recall	F1 Score
70.56 %	58.37 %	63.89 %

Table 4.9: Classification results for the multi-label task.

Precision	Recall	F1 Score
20.00 %	50.00 %	29.00 %

Table 4.10: Multi-label classification benchmark generated with a random classifier following a Bernoulli distribution with $p = 0.5$.

5
Conclusion

A convolutional neural network arquitecture was developed for the single-label multiclass classification task using the UrbanSound8K dataset, achieving an overall prediction accuracy of 65%. Studies on undersampling of the dataset did not show any improvement in overall performance of the classifier, but revealed that a high proportion of the samples for most of the classes are not contributing to the learning of the network, which confirms the low degree of intra-class variance of this particular dataset and the consequent high overfitting. The confusion matrix of the baseline results revealed that foreground and background instances have a big impact on the classifier confusion of certain classes with other common environmental sounds, like *Street music* or *Children playing*.

Studies on noise corruption and amplitude scaling of the inputs revealed how sensitive these systems are to input variations, showing significant prediction accuracy drops for SNR levels lower than 5 dB. Noise data augmentation techniques and amplitude scaling of the training set showed to significantly improve the robustness of the classifier to these input variations.

A multi-label environmental sound classifier was developed using a variation of the UrbanSoun8K dataset, in which 280 instances per class were combined to give a dataset of 12600 samples. Despite the reduced number of samples, a recall of 58 % was obtained, leaving generous room for improvement in this task.

Further research is needed to draw any conclusions on this but, the positive results on the multi-label task suggest that, even when using a limited dataset where only 280 instances of each class were considered, the multi-label task is likely to introduce some inherent regularization that encourages the network to learn more robust and features of each class, since some of the input features are masked at times.

Since no previous benchmark is available at the time of writing for the multi-label classification task using the UrbanSound8K dataset, this study sets the stage for further research in this field.

Bibliography

[1] Charlie Mydlarz, Justin Salamon, and Juan Pablo Bello. "The implementation of low-cost urban acoustic monitoring devices". In: *Applied Acoustics* 117 (2017), pp. 207–218.

[2] EEA. *Noise in Europe*. Tech. rep. ISSN 1977-8449. Kongens Nytorv 6, 1050 Copenhagen K, Denmark: European Environment Agency, 2014.

[3] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. "A dataset and taxonomy for urban sound research". In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, pp. 1041–1044.

[4] Justin Salamon and Juan Pablo Bello. "Deep convolutional neural networks and data augmentation for environmental sound classification". In: *IEEE Signal Processing Letters* 24.3 (2017), pp. 279–283.

[5] Karol J Piczak. "Environmental sound classification with convolutional neural networks". In: *Machine Learning for Signal Processing (MLSP), 2015 IEEE 25th International Workshop on*. IEEE. 2015, pp. 1–6.

[6] Venkatesh Boddapati et al. "Classifying environmental sounds using image recognition networks". In: *Procedia Computer Science* 112 (2017), pp. 2048–2056.

[7] Huy Phan et al. "Audio scene classification with deep recurrent neural networks". In: *arXiv preprint arXiv:1703.04770* (2017).

[8] Yuni Zeng et al. "Spectrogram based multi-task audio classification". In: *Multimedia Tools and Applications* (2017), pp. 1–18.

[9] Simon S Haykin et al. *Neural networks and learning machines*. Vol. 3. Pearson Upper Saddle River, NJ, USA: 2009.

[10] Joseph D Dumas II. *Computer architecture: fundamentals and principles of computer design*. CRC Press, 2016.

[11] Steven K Rogers and Matthew Kabrisky. *An introduction to biological and artificial neural networks for pattern recognition*. Vol. 4. SPIE Press, 1991.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[13] *Convolutional Neural Networks for Visual Recognition*. `http://cs231n.github.io/convolutional-networks/`. Accessed: 2018-06-01.

[14] *The Softmax Function, Neural Net Outputs as Probabilities, and Ensemble Classifiers*. `https://towardsdatascience.com/the-softmax-function-neural-net-outputs-as-probabilities-and-ensemble-classifiers-9bd94d75932`. Accessed: 2018-06-03.

[15] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[16] *Improving the way neural networks learn.* http://neuralnetworksanddeeplearning.com/chap3.html. Accessed: 2018-04-04.

[17] *Making sense of Logarithmic Loss.* http://www.exegetic.biz/blog/2015/12/making-sense-logarithmic-loss/. Accessed: 2018-04-04.

[18] Nitish Srivastava et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[19] Patrik Andersson. *Lab 4: Signal Analysis Task. Sound and Vibration Measurements.* Oxford university press, 2005.

[20] John G Proakis and Dimitris G Manolakis. *Digital signal processing.* Pearson Education, 2013.

[21] *The mel frequency scale and coefficients.* http://kom.aau.dk/group/04gr742/pdf/MFCC_worksheet.pdf. Accessed: 2018-05-25.

[22] Brian McFee et al. "librosa: Audio and music signal analysis in python". In: *Proceedings of the 14th python in science conference.* 2015, pp. 18–25.

[23] Karol J. Piczak. "The details that matter: Frequency resolution of spectrograms in acoustic scene classification". In: *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop.* Munich, Germany, 2017.

[24] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).