

Democritus University of Thrace

Department of Electrical & Computer Engineering

Indoor Sound Detection & Classification

Papadopoulos Aristeidis

Supervising Professor: Mitianoudis Nikolaos, Associate Professor DUTH

October 2022, Xanthi

Thanks

I would like to thank my family and my friends for their unwavering support, as well as Dr. Mitianoudis for his advices and his invaluable assistance during the creation of this thesis.

Abstract

Audio is one of the most typical ways with the help of which humans perceive their surrounding environment, alongside visual stimuli. These two concepts comprise audio and visual information respectively. Using this information, humans can, with very high precision, recognize an activity that is happening in their environment. However, computers do not possess this ability inherently.

Using machine learning, the research community attempts to transfer this ability to computers and machines. Its applications vary, from their integration to the more and more popular home assistants, independent audio surveillance systems or complementary to visual surveillance systems, activity recognition, based exclusively on sound signals.

In this thesis, the use of machine learning techniques is examined and tested, using a dataset, comprising of audio signals, recorded using microphones inside a house. More specifically, this thesis examines the use of indoor sound signals, in contrast to other projects using outdoor sound signals. The target of this thesis is to build a machine learning architecture that achieves the most accurate classification of the available audio excerpts in the test database.

Contents

1 Introduction to Sound Classification **1**
1.1 Sounds representation in maths 1
1.2 Sound Classification 2
1.3 Brief historical review 2
1.4 Thesis Structure 3

2 Audio processing **4**
2.1 Pre-processing 4
2.1.1 Trimming 4
2.1.2 Filtering 4
2.1.3 Normalization 4
2.1.4 Beamforming 4
2.2 Feature Extraction 5
2.2.1 Chromagram 5
2.2.2 Gammatone Frequency Cepstral Coefficients 5
2.2.3 Constant Q Transform 6
2.2.4 Spectrograms/Mel-Spectrograms 6
2.2.5 Mel-Frequency Cepstral Coefficients 8

3 Neural Networks **10**
3.1 Introduction 10
3.2 Historical retrospect 10
3.3 General Elements of Neural Networks and Deep Learning 14
3.3.1 Simple Neural Networks 14
3.3.2 Convolutional Neural Networks - CNNs 14
3.3.3 Recurrent Neural Networks - RNNs 15
3.3.4 Transformers 16
3.4 Neural Network Layers 18
3.4.1 Convolution Layer 18
3.4.2 Pooling Layer 20
3.4.3 Fully Connected Layer 21
3.5 Activation Functions 22
3.5.1 Sigmoid Function 22
3.5.2 Hyperbolic Tangent Function (Tanh) 22
3.5.3 Rectified Linear Unit(RELU) 23
3.5.4 Leaky RELU 24
3.5.5 Exponential Linear Unit (ELU) 24
3.5.6 Gaussian Error Linear Unit (GELU) 25
3.5.7 Softmax 25
3.6 Neural Network's Training 25

3.6.1 Epochs . . . 25
3.6.2 Batch Size . . . 25
3.6.3 Learning Rate . . . 26
3.6.4 Cost Function . . . 26
3.6.5 Momentum . . . 26
3.6.6 Optimization Algorithms . . . 27
3.6.7 Class Weights . . . 28
3.7 Generalization techniques . . . 29
3.7.1 Dropout layer . . . 29
3.7.2 Normalization . . . 29
3.7.3 Data Augmentation . . . 30
3.7.4 Regularization . . . 30
3.7.5 Learning rate Decay . . . 31
3.7.6 Callback . . . 33
3.8 Models Evaluation Metrics . . . 33
3.8.1 Metrics . . . 33
3.8.2 Confusion Matrix . . . 34

4 Indoor Sound Classification **35**
4.1 Software development environment . . . 35
4.2 Hardware used . . . 35
4.3 Dataset . . . 36
4.3.1 Data overview . . . 36
4.3.2 Audio features . . . 38
4.3.3 Data augmentation . . . 39
4.4 Image Pre-processing . . . 40

5 Deep Learning Architectures For Indoor Sound Classification **41**
5.1 Hyper-parameter Tuning . . . 41
5.2 1D Convolutional Networks . . . 42
5.3 2D Convolution Networks . . . 44
5.3.1 Symmetrical kernels . . . 44
5.3.2 Non Symmetrical Kernels . . . 46
5.4 Two CNNs in parallel . . . 48
5.5 Ensemble of two convolutional models . . . 48
5.6 Vision Transformers . . . 50

6 Experimental Results **52**
6.1 Competition Results . . . 52
6.1.1 Inoue et. al. [1] . . . 52
6.1.2 Tanabe et. al. [2] . . . 54
6.1.3 Liu et. al. [3] . . . 55
6.2 Thesis Results . . . 56
6.2.1 1D Convolution . . . 56
6.2.2 2D Convolution . . . 57
6.2.3 Model Ensemble . . . 59
6.2.4 Vision Transformers . . . 60
6.3 Results Comparison . . . 61

7 Summary and Conclusion **63**

Chapter 1

Introduction to Sound Classification

In this chapter, a small introduction is made on sounds, their classifications, what kind of application can be made from their classification and how the research community has dealt with this problem.

1.1 Sounds representation in maths

A sound (or sound wave) is a sum of vibrations that are being transmitted, using a medium. In other words, a sound is a waveform, where the independent variable is time and the dependent variable is the amplitude of said waveform.

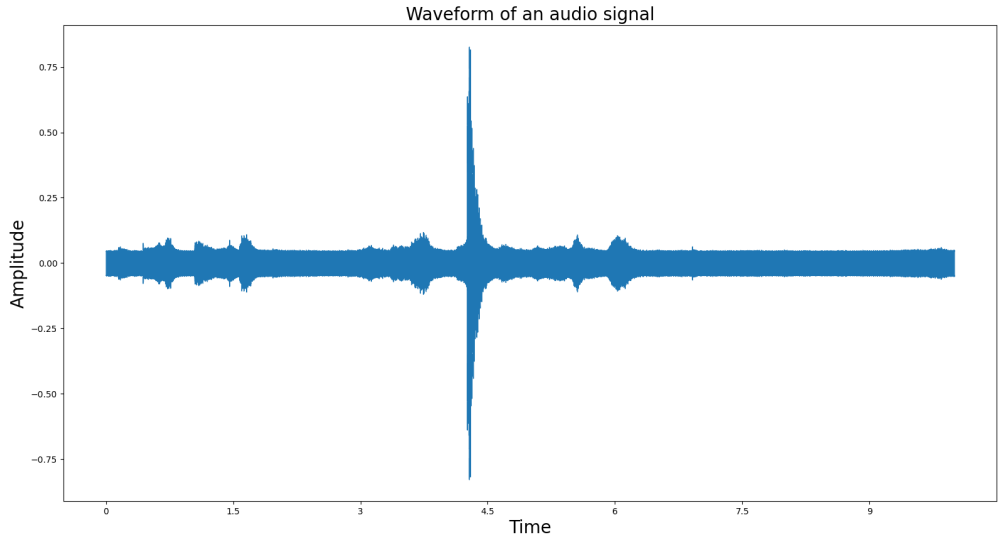


Figure 1.1: A representation of an audio signal

One of the most basic features of an audio is its frequency, i.e., how common those vibrations are in the unit of time. However, more audio signals are comprised of multiple frequencies, not by a single one. All of those frequencies are defined as the frequency spectrum of the signal. Since the amplitude of a sound wave changes throughout time, so do the frequencies as well. The visualization of frequency changes in the duration of time is the spectrogram.

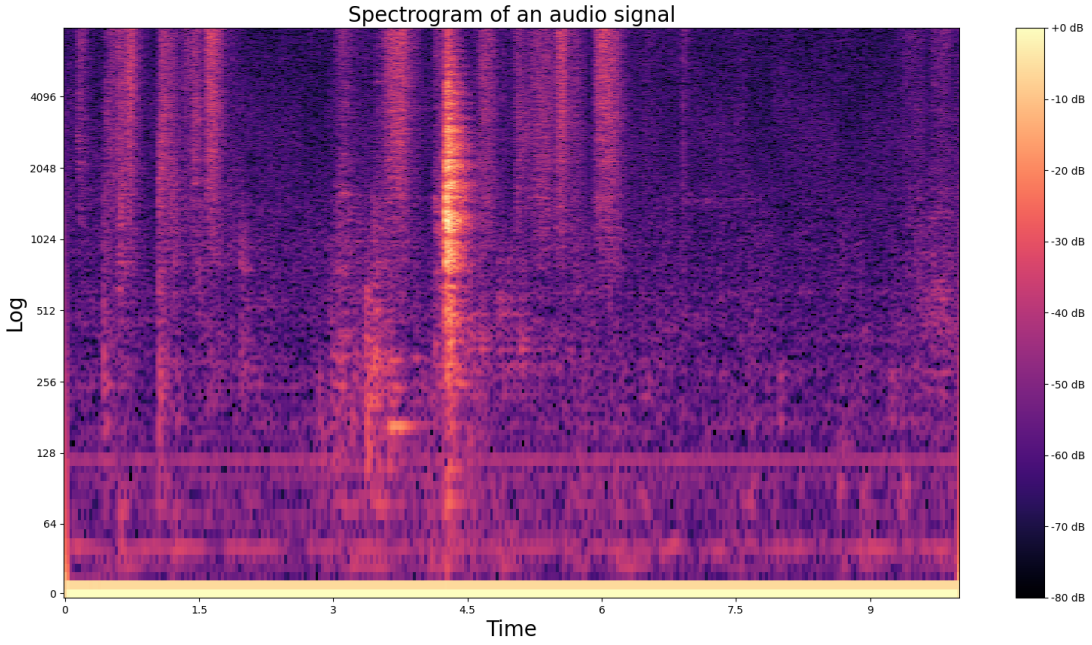


Figure 1.2: Spectrogram of the previous waveform

Humans can differentiate the different sounds they hear based to a great extend to their frequency and can also detect the direction of those sounds to a sufficient degree [4]. However, in certain applications, this functionality must be utilized by using artificial means, such as observing a person using a home assistant or by the use of wearables [5], in the construction of smart homes [6], in the creation of detection systems (alarms) and in other uses [7]. In the case of artificial means, this is achieved by the use of neural networks, where these problems are part of a variety of problems including sound classification, sound source detection, speaker separation and voice detection.

1.2 Sound Classification

Sound classification is a method for automatic event detection, through finding and learning sound patterns, trying to simulate the information retrieval that the human ear system does [8, 9]. The research community, to deal with this problem, looked for solutions in machine learning and artificial intelligence.So, whether actual audio clips or features extracted from them are used, the problem is considered to be a classification problem. These problems are separated mainly into two categories, the outdoor sound detection and the indoor sound detection. In both categories, the target is the right detection (and classification) of the audio scenes that the audio clips used are hinting. For example, if an sound is about a person working, then the audio scene can be set to work.

1.3 Brief historical review

Initially, the techniques used were classic classification methods, such as Support Vector Machines (SVM), k-nearest neighbors(knn) etc [10]. Then, the use of convolu-

tional neural networks (CNN) in image classification problems prevailed, resulting in their usage in audio classification as well [11], culminating in a variation of VGG, VGGish [12], which is trained in spectrograms of the Audioset dataset Audioset[1] [13]. Audioset contains audio clips for 632 possible classes. Recently, Vision Transformers were used with greater success than CNNs [14].

In any case, before these methods are used, the original sounds are either pre-processed or features are extracted from them or both. The main pre-processing methods, along with the different features are presented further in this thesis.

1.4 Thesis Structure

In Chapter 2 information extraction methods from sounds will be presented, while in the next chapter the basic principles of neural networks are discussed, along with their different parameters (Chapter 3). Then, an overview of the dataset and its features used is made (Chapter 4). In Chapter 5 the neural networks studied and designed for this thesis is shown, while in Chapter 6 a comparison is made and their results are commented upon. Finally, in Chapter 7 an overview of this thesis and its results is done.

[1] https://research.google.com/audioset/

Chapter 2

Audio processing

In this chapter, some pre-processing techniques are analysed, along with certain features that can be extracted from sound clips.

2.1 Pre-processing

Audio signals that are about to be used are probably going to be pre-processed, to either increase the accuracy of the final model and reduce the computational costs, or to remove unwanted elements in them, such as noise. [9].

2.1.1 Trimming

By Trimming, the total duration of the sound data is reduced, while usually the silence found in their beginning and their end is removed. Also, noise can be removed. By removing non-useful information, the computational costs can be reduced.

2.1.2 Filtering

With Filtering, the removal of unwanted frequencies is achieved. This can be utilized with many ways, such as with the use of Low-Pass Filters (LPF), High-Pass Filters(HPF), Band-Pass Filters(BPF) and others.

2.1.3 Normalization

Normalization transforms an audio file in a way that all the contained acoustic scenes are in the same decibel level, without one overlapping the other, or them being overlapped by noise. So, each moment in the sound signal have values within the same range.

2.1.4 Beamforming

This technique can be used with two ways. As filtering, to remove noise from an audio file. Otherwise, a multi-channel audio clip can be transformed into single-channel audio clip, as long as the locations of the microphones used during the recording are known [15]. As such, the end file is comprised of a weighted average of these channels, instead of the non-weighted average used more commonly, to convert multi-channel audio to single-channel. The weighted averaged is used based on the fact that different microphones record, up to a degree, differently the same activity, depending on their position.

2.2 Feature Extraction

Different kinds of features have been proposed which can be used depending on the problem. The main features found are:

- Chromagram [16]
- Gammatone Frequency Cepstral Coefficients (GFCC) [17]
- Constant Q-Transform (CQT) [18]
- Spectrogram or MelSpectrogram [18]
- Mel-Frequency Cepstral Coefficients (MFCC) [19]

2.2.1 Chromagram

Chromagram is the display of the changes of the pitch (or the fundamental frequency) in time. Usually, it is used in problems related to music, as they capture the harmonics and the melodies of a sound, while being robust to the changes of timbre.

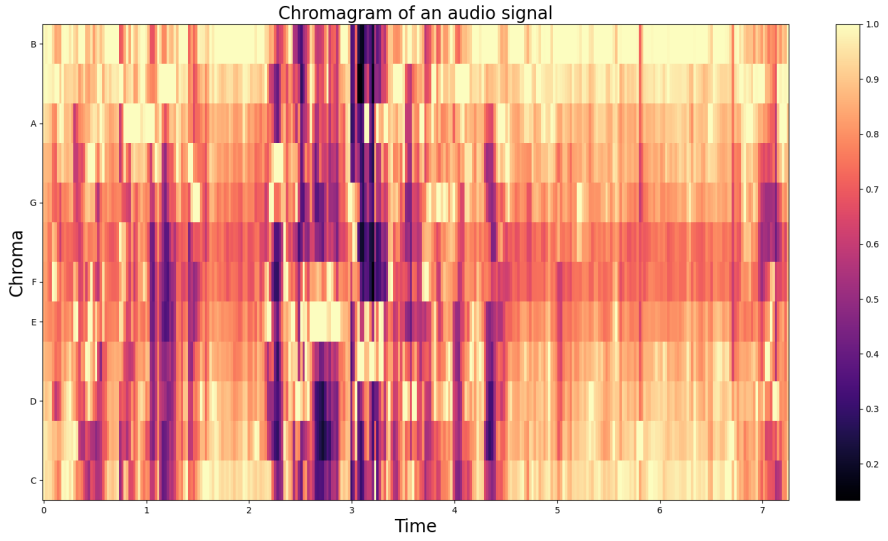


Figure 2.1: Chromagram of a sound

2.2.2 Gammatone Frequency Cepstral Coefficients

Gammatone Frequency Cepstral Coefficients (GFCC) are used in speech recognition. Calculating these coefficients is based on a representation of frequency with time called cochleogram.

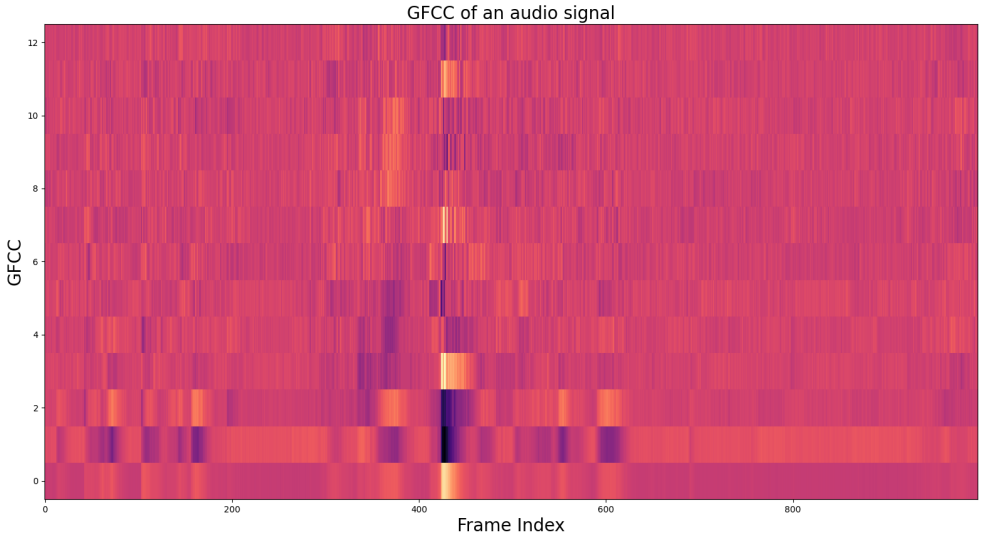


Figure 2.2: GFCC of an audio

2.2.3 Constant Q Transform

It is about a representation of a signal in the time and frequency domains. Frequency bands are geometrically placed, while the Q coefficients of all bands are equal. Mainly, this feature is used in problems in the music domain.

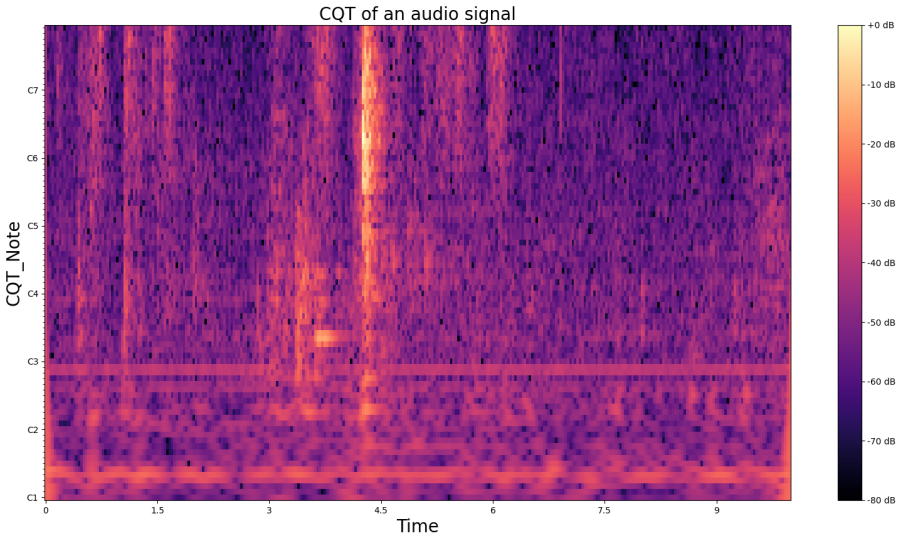
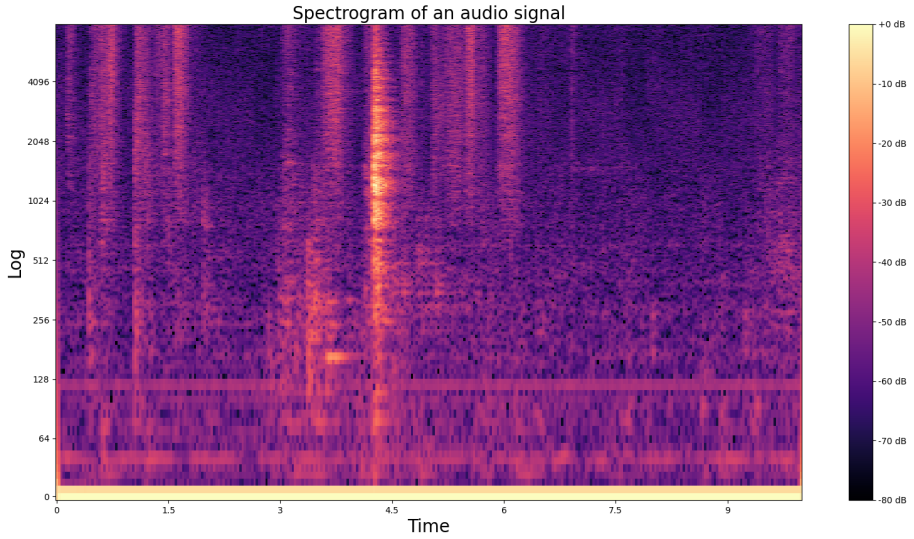


Figure 2.3: CQT of an audio

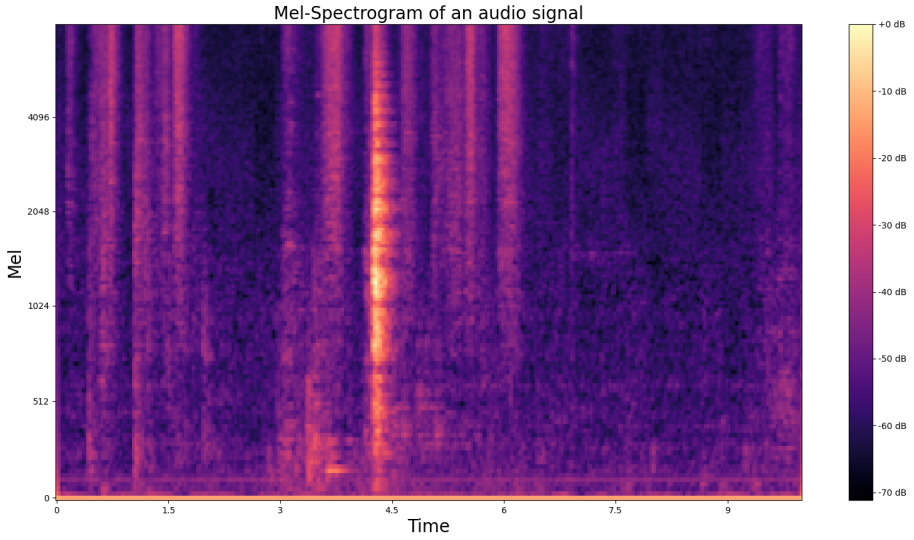
2.2.4 Spectrograms/Mel-Spectrograms

Spectrogram, as mentioned previously, is a representation of an audio signal in the time-frequency domain. What is represented in a spectrogram is the distribution of energy into the different frequencies as the signal progresses in the time unit. One

variation of spectrograms, is mel-spectrograms, where frequencies are placed in logarithmic method, simulating the way the human ear works.



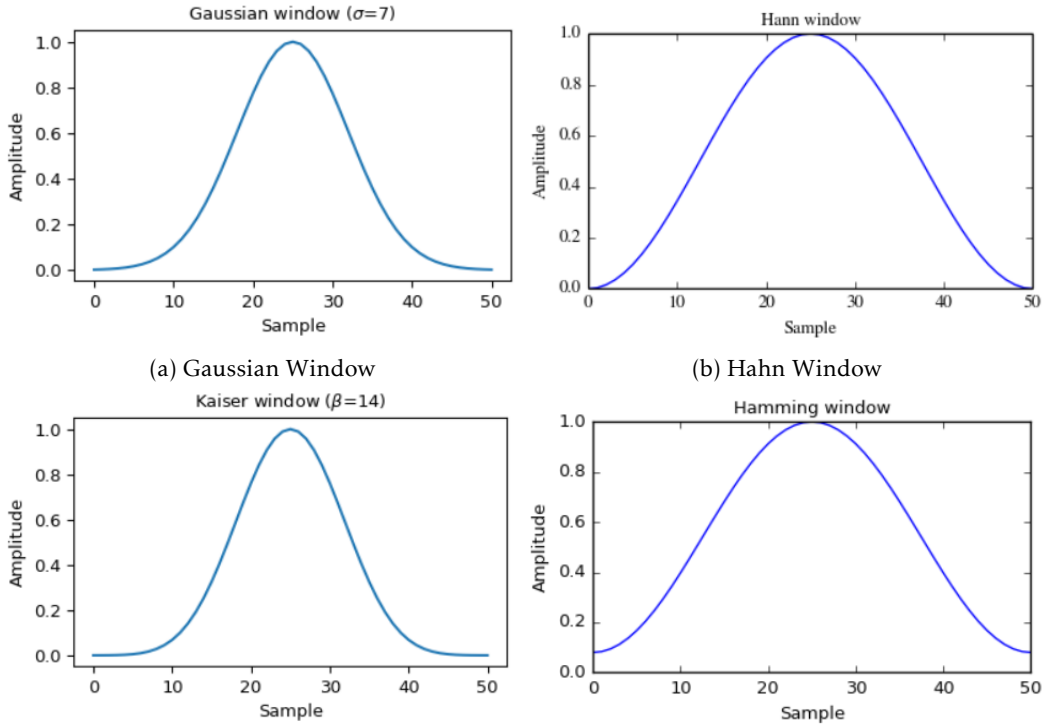
(a) Spectrogram



(b) Mel-Spectrogram

Figure 2.4: Spectrogram variations

These spectrograms are essentially visualizing the result of applying STFT (Short Time Fourier Transform) to the waveform of the sound. More specifically, STFT is an application of Fast Fourier Transform (FFT) onto the original signal, where the signal is being split into equally sized pieces, which sometimes are overlapping with eachother. Then, a window function is applied with its result being the spectrogram. Usually, the user defines the window length, the number of Fourier Transform Coefficients, the number of samples between sliding windows hop length and the kind of window to use. Indicatively, some window functions are the kaiser, the hamming, the hahn and the gaussian and these are displayed below at figure .



(a) Gaussian Window

(b) Hahn Window

(c) Kaiser Window

(d) Hamming Window

Figure 2.5: Commonly used window functions for STFT [20].

2.2.5 Mel-Frequency Cepstral Coefficients

Finally, the MFCC are one of the most commonly used features to be extracted from audio signals. Their computation is based on spectrograms and it has been noted that the coefficients that are computed are very effective in capturing the main characteristics of audio signals.

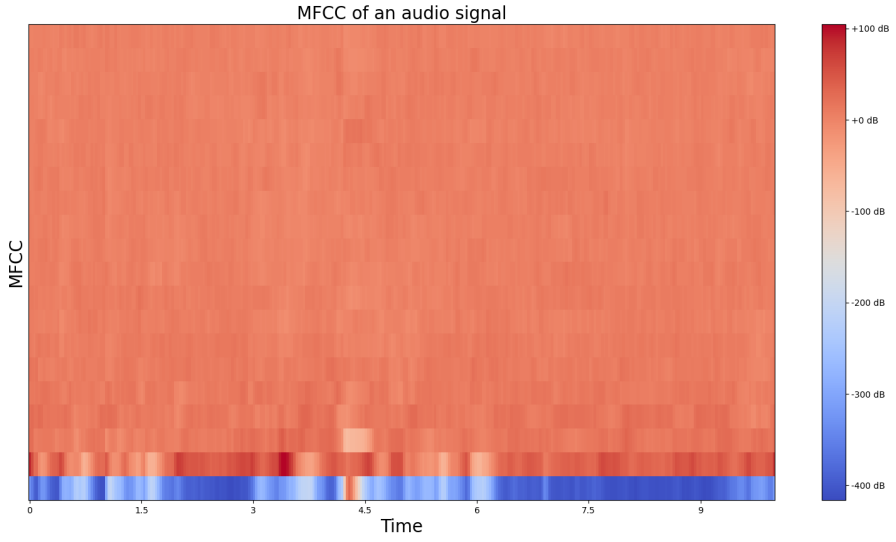


Figure 2.6: MFCC of a sound

Their computation consists of five basic steps [21]. Initially, the original signal is

splitted into overlapping parts, where Fourier transform is applied. Then, by using triangular windows, the spectrum values are matched to the logarithmic mel scale and a coefficient is calculated for each triangular window. Afterwards, discrete sine transform is applied to these coefficients in order to further compress the energy of the first coefficients. The end result is the MFCC.

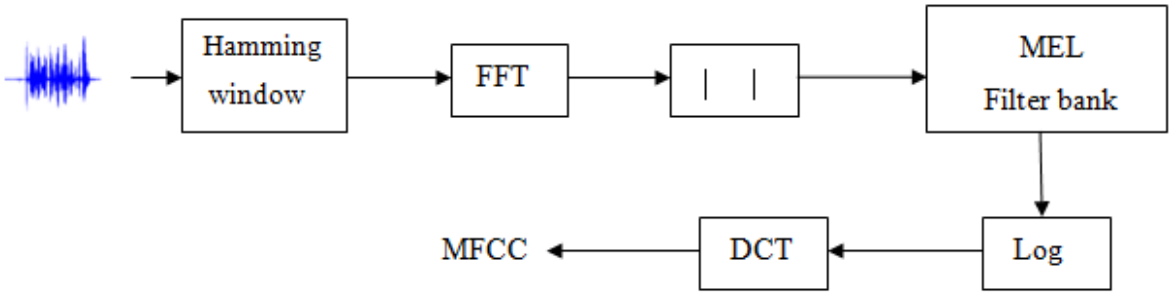


Figure 2.7: Computational steps of MFCC [22].

Chapter 3

Neural Networks

3.1 Introduction

Artificial neural networks (or neural networks) were created in an attempt to simulate the biological neural network. They are powerful computational systems and have the ability to solve or to be applied in various problems and fields of interest. Some examples are:

- Image or Sound Classification
- Object Detection
- Data mining
- Robotics, Automatic Control applications
- Optical character recognition
- Cybersecurity

3.2 Historical retrospect

The start was made in 1943, with the model designed by McCulloch and Pitts, which is based on the way biological neural networks operate [23]. More specifically, they designed a model which resembles the biological neuron and is possible to solve mathematical and logic equations, even though they did not suggest a learning method. The answer to this was given by D.O. Hebb [24] a few years later, who suggested a learning rule, commonly known as Hebbian rule and the learning that uses this rule is known as Hebbian learning. Then, Frank Rosenblatt in 1958, suggested the Perceptron Model [25], with applications in pattern recognition, and the Perceptron convergence theorem. However, the main disadvantage, among others, is that the problems Perceptron can solve must be linearly separable, as noted by Minsky and Papert in 1968 [26].

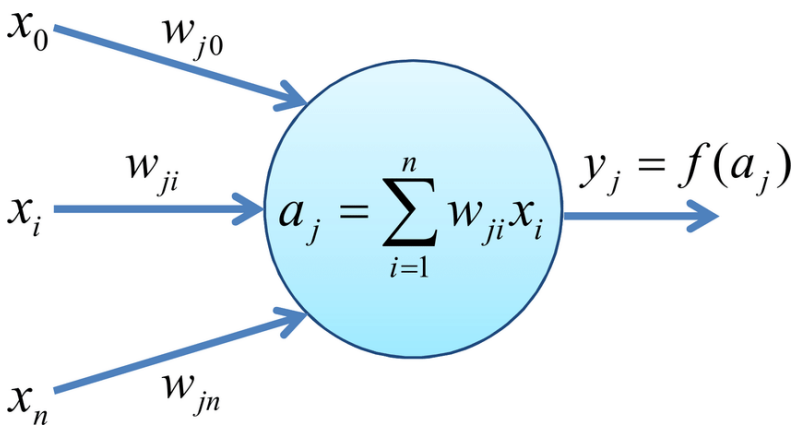


Figure 3.1: Perceptron visual representation [27]

Due to the disadvantages presented by Minsky and Papert and due to the low computational power of those times, the interest around neural networks gradually was reduced and research remained stale for the next few years. Interest was renewed by the Back-Propagation algorithm, relative to training multi-layered neural networks, by Rumelhart, Hinton and Williams [28]. Furthermore, the rapid development of computer systems was crucial to the spread of neural networks and their applications. In 1998, Yann LeCun and his team proposed a neural network structure that became known as LeNet, based on consecutive convolutions between the inputs of the network and a set of predetermined kernels. This structure was named Convolutional Neural Networks or CNNs [29].

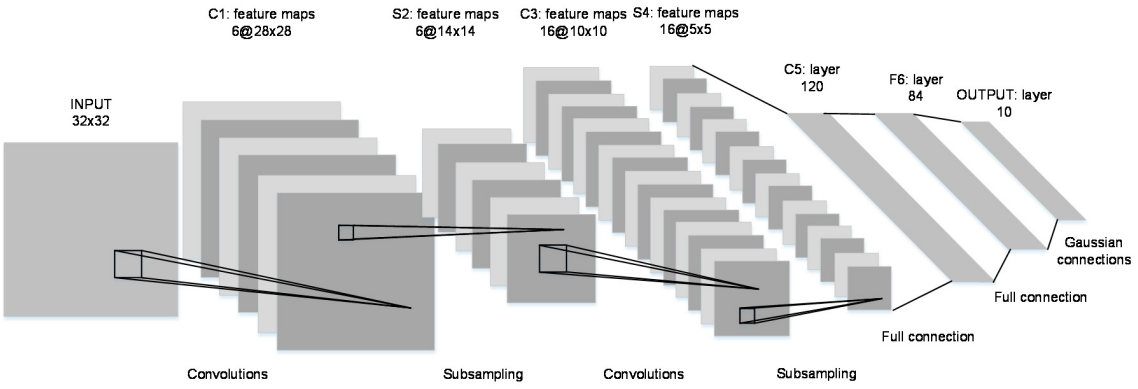


Figure 3.2: LeNet5 Visual Representation [30].

The success of these networks enlarged the research community's interest and led to their almost universal use, rendering existing networks obsolete. The increasing interest and research into neural networks lead to the organization of many contests, such as ImageNet [31], with models Alexnet [32], GoogleNet [33], VGGNet [34] achieving notable results.

Afterwards, new kinds of networks appeared, the so called Recurrent Neural Networks (RNNs) [35].In this network topology, the concept of time is introduced, as these networks have the ability to remember their previous input and find possible time correlation between inputs. These networks are used in problems where previous input is important or the time relationship between samples, such as time series or stock prediction.

For all their success, both convolutional neural networks as well as recurrent neu-

ral networks are plagued by the vanishing gradient problem, which intensifies as the network gets deeper. The solution is given by two new kinds of networks, Long-Short Term Memory (LSTM) networks [36] and ResNet networks. LSTM networks use activation cells, limiting the inputs and the outputs of neurons, resulting in the spread of gradient throughout time and not disappearing as in RNNs.

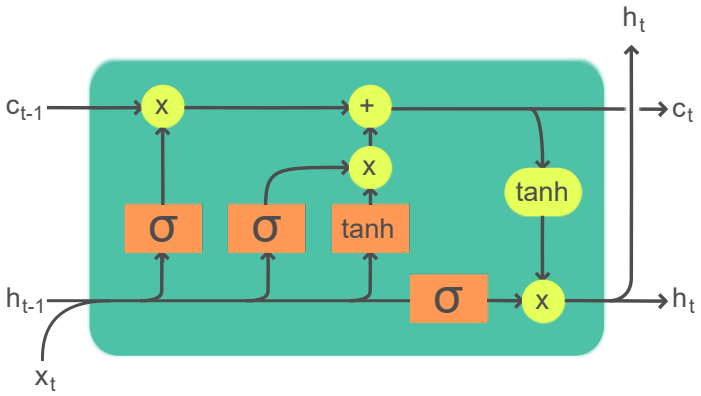


Figure 3.3: LSTM cell.

Furhermore, the creation of ResNet, by Kaiming He, Xiangyu Zhang, Shaoqing Ren Jian Sun [37], introduces the concept of skip connection, a solution for very deep convolutional neural networks. As displayed in f 3.4, a previous output of one layer reappears and is added to the output of a next layer.

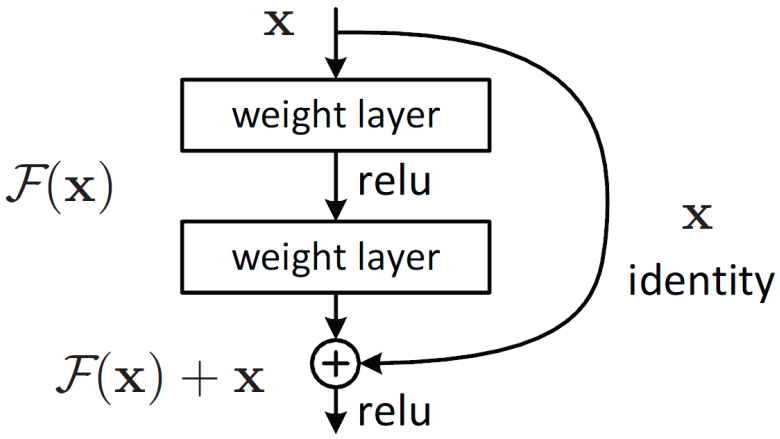


Figure 3.4: Skip connection visualization [37].

In the last year, applications initially in Natural Language Processing and in applications using text, one commonly used network is the Transformer Network. In this case of networks, one noteworthy parameter is the notion of Attention [38]. By utilizing this mechanism, a model can weigh its inputs depending on their relevancy.

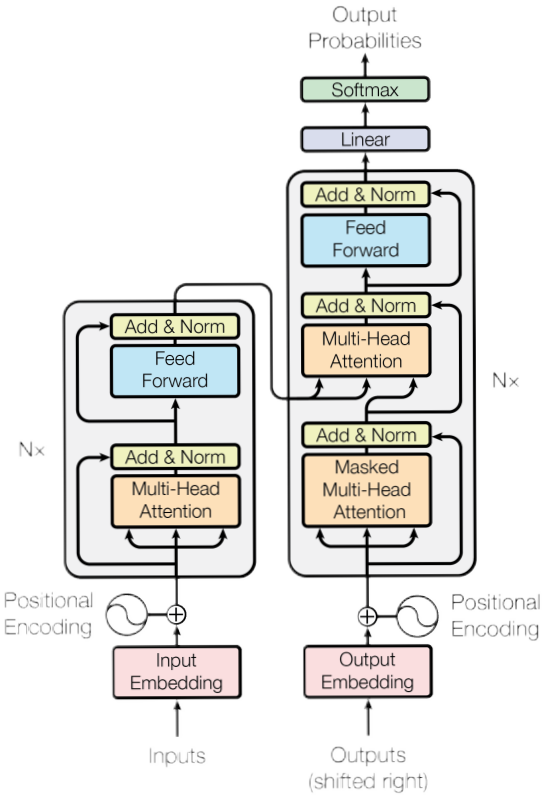


Figure 3.5: Transformer Network [38].

Lastly, recently the use of transformers has expanded into computer vision, with the use of Vision Trnasformers (ViT), where these models achieve better results than those of ResNet [39]. Vision Transformers' input is splitted into sub-images or patches, which are then used as input into the network.

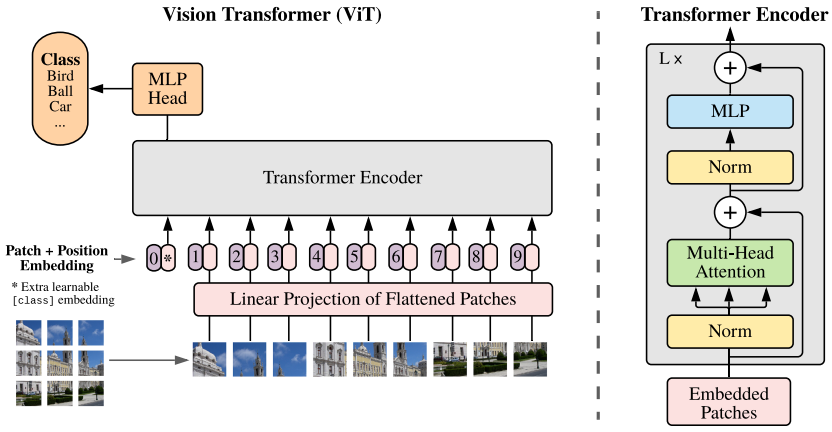


Figure 3.6: ViT network [39].

However, as noted by the authors, in order to achieve those results, the training must be first completed to other larger datasets(transfer learning).Seung Hoon Lee, Seunghyun Lee and Byung Cheol Song suggested modifications to the way attention is applied and to the way patches are created [40], in order to reduce the size of data used for training.

3.3 General Elements of Neural Networks and Deep Learning

Each kind of neural network is consisted by multiple units, organized into layers. The first layer is called input layer, the inbetween layers are called hidden layers while the last layer is called output layer. Each layer has a function, named activation function. During training, a cost function is used, which expressed the deviation of the network's output from the actual ones, while an optimization algorithm is used in order to minimize the value of the cost function. Thus, the lower the value of the cost function is, the more samples are going to be correctly classified. In the following units, an analysis is made on the most common choice for each mentioned concept, as well as common network architectures.

3.3.1 Simple Neural Networks

Multi-Layered Perceptrons or Feedforward neural networks, as they consist of multiple perceptron layers connected with each other and because the information is forwarded. Each layer can have any number of simple perceptron units, while common practise is as the network deepens, to increase the number of perceptron units per layer.

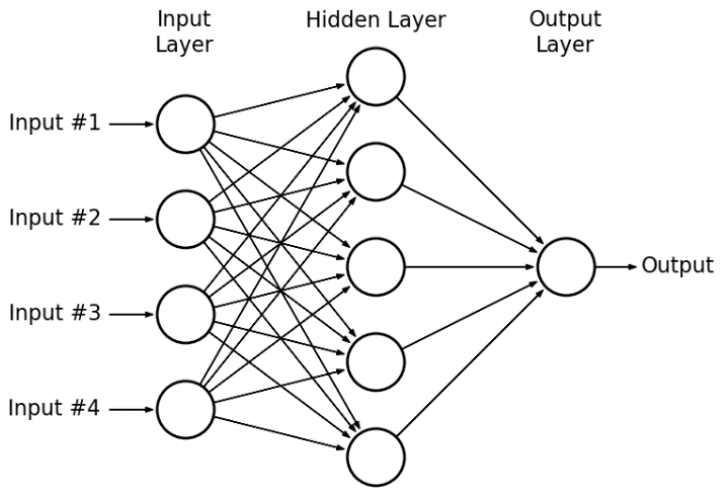


Figure 3.7: A Multi-Layered Perceptron Network [41].

However, as the network deepens, the training becomes more complicated and the probability of trapping the training process in some local minima is increased [42].

3.3.2 Convolutional Neural Networks - CNNs

In order to deal with the ever increasing problems in the machine learning field, a new architecture needed to be implemented. Thus, having as foundation the functionality of optical cells, [43], working as spatial filters into an area called receptive fields, the convolutional neural networks were created. As such, optical cells take advantage of the local correlation of information into optical signals.

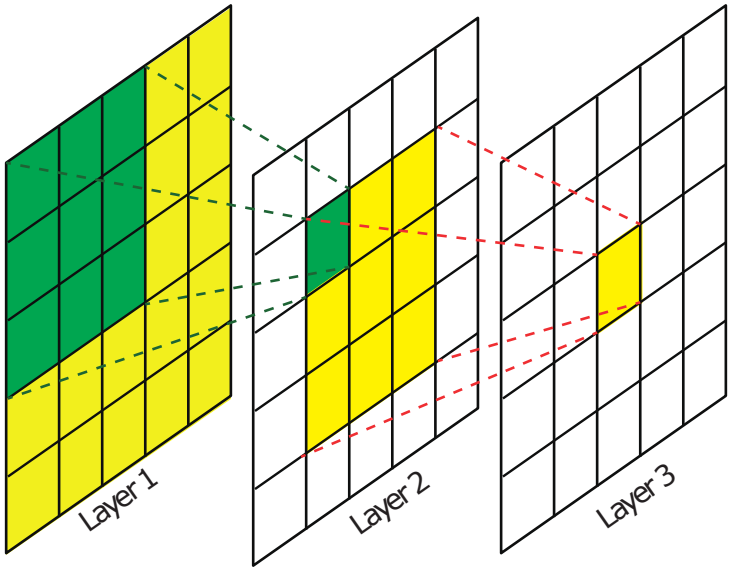


Figure 3.8: Receptive fields visualization [44].

Therefore, by using the convolution operation and by properly defining in different convolution layers it is possible to emulate this function into computer systems. More information can now be extracted that the network can use to solve more complex classification problems.

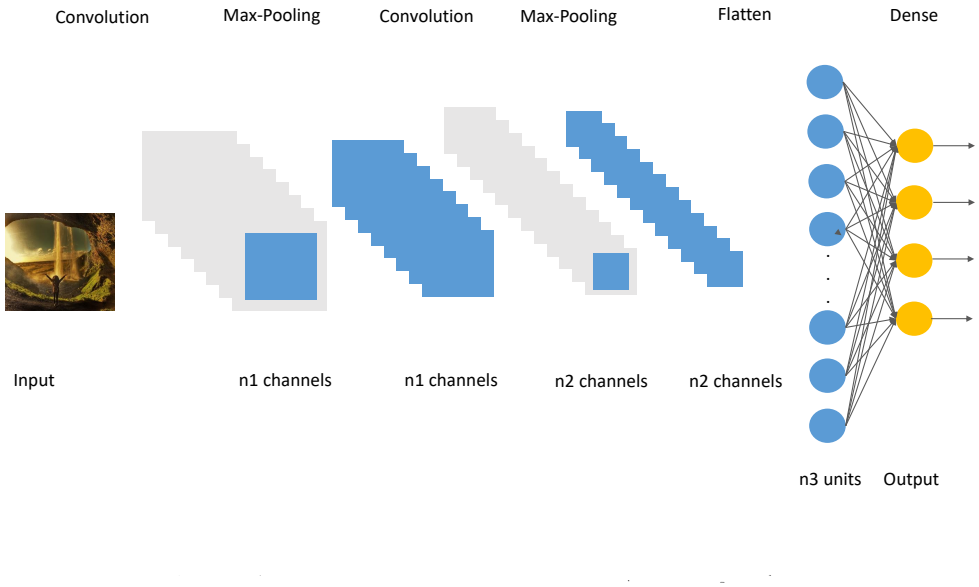


Figure 3.9: Typical CNN structure [45].

A usual network includes sequential convolution layers with sub-sampling layers, resulting in a dimension-reducting layer, followed by some fully connected layers.

3.3.3 Recurrent Neural Networks - RNNs

Recurrent Neural Networks are used for data that are time correlated, for example a timeseries. In contrast with CNNs, recurrent networks' input contain the activation of the previous layer. Thus, it is possible to find a time connection between them.

Considering this, their use is highly advisable when correlation between inputs is of interest, such as texts.

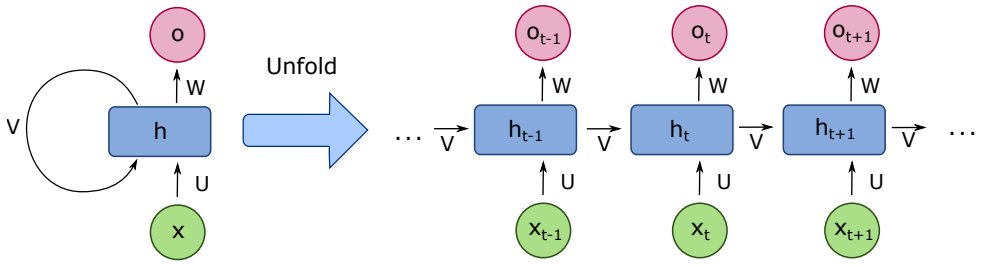


Figure 3.10: A RNN unit (left) and its unraveling in time (right).

3.3.4 Transformers

Transformers were proposed as the answer to the search of new architectures that model time correlation and pay attention to useful structures of the timeseries, for the problem of classification [38]. They originally were applied to Natural Language Problems (NLP) and Machine Translation. However, it was apparent by now that their use could be expanded into other problems, as well as in computer vision, where a variation of the orignal network was created, the Vision Transformer [39].

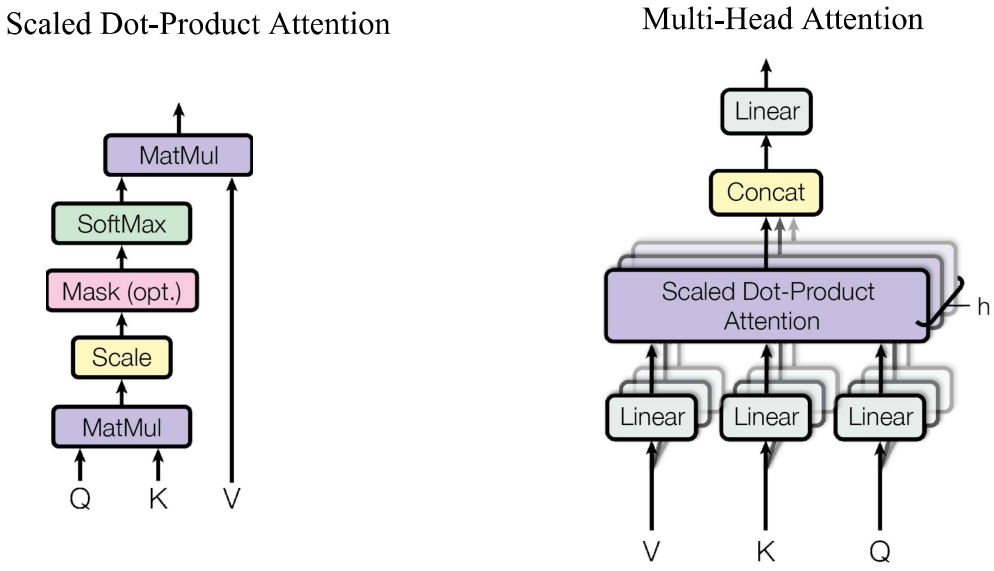


Figure 3.11: Attention mechanism types, on left the simple version and on the right the multi-head one [38].

Basic principle of operation is the attention mechanism, where a match is made between a query and a pair of key-value to an output. Essentially, it is a compatibility function, where for every sample a weight is computed that is attributed to a corresponding key. This operation can be paralleled, referred to in bibliography as multi-head attention. In order to make the head concept understandable, it is enough to assume that for a problem, n people are asked. Those people represent the heads.

Furthermore, due to the absence of convolutional and recurrent layers, information that my be included into the data stream is lost. As such, the mechanism of positional encoding is introduced, in order to have the notion of sequence. Finally, the input is split into tokens, where in conjunction with positional encoding are then forwarded to the network. In the case of vision transformers, input images are split into pieces called patches. The network's output, in any case, is the prediction of the network, either with probability or with logits.

However, despite their innovations, transformers have large demands data-wise, in order to compete with convolutional networks and their variations such as ResNet. This is due to the fact that spatial information found in convolution, in transformers is either in less extend or not at all. In order to deal with these problems, two variations into the transformers mechanism were proposed, Shifted Patch Tokenization [40] and Locality Self-Attention Mechanism [40].

Shifted Patch Tokenization

Each image is shifted by half patch size into every diagonal direction, i.e. upper right, upper left, lower right and lower left. Thus, four new images are created, where they are cropped to have the same size as the original. Then, all the images are put back together and are split into not overlapping patches. In the end, visual tokens are created by using linear projection and Layer Normalization.

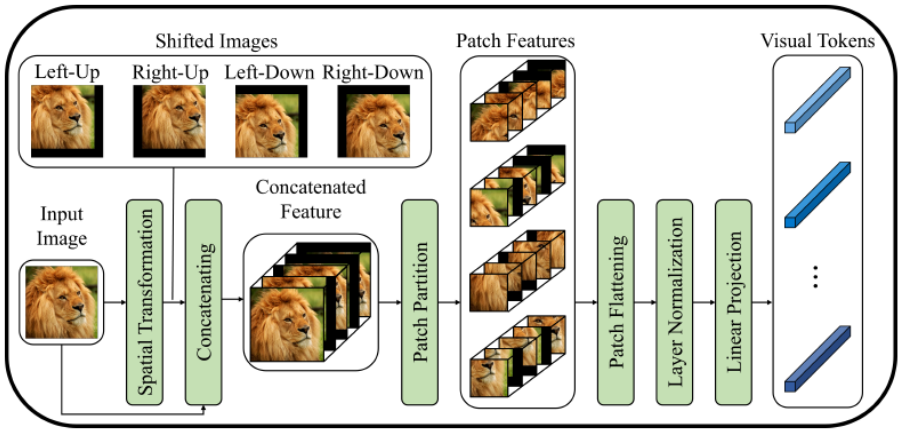


Figure 3.12: Shifted Patch Tokenization Visualization [40].

Locality Self-Attention Mechanism

It is consisted by two parts, Diagonal Masking and Learnable Temperature Scaling. In Diagonal Masking, higher score is given when softmax function is applied to new tokes, instead of the tokens already found by ViT. In Learnable Temperature Scaling, the ability to find the temperature on its own is given to ViT, which is then used by softmax functions of the attention mechanism.

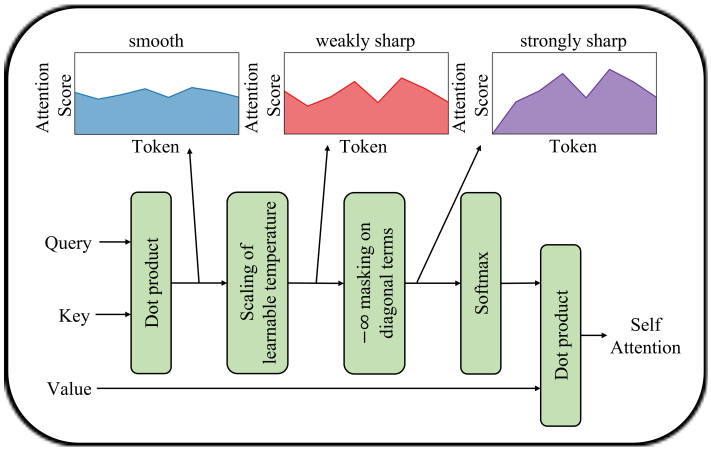


Figure 3.13: Visualization of Locality Self-Attention Mechanism [40].

3.4 Neural Network Layers

3.4.1 Convolution Layer

Convolution layers are one of the most basic parts of convolutional neural networks. Feature extraction is made on these layers. Features are found by the convolution operation, where the input is convoluted with a matrix named kernel. Furthermore, before the kernel operations take place, a number of filters are shifted upon the image, to enable the kernel to extract different features. The convolution results are called feature maps and are equal to the number of filters. Depending on the input dimensions, convolution can be 1D, 2D or 3D. Other important variables that must be defined are padding and stride. Padding defines the convolution operation in boundary conditions, i.e. the image's edges. Many choices are available, with the two most coming being the use of zeros to fill the elements missing, also known as zero padding, to copy edge elements or to not perform convolution there. Stride defines how long the shift will be between two successive convolutions, with high value performing less operations, speeding up the process but with possible information loss.

1D Convolution

One dimension convolution is used when the data used is timeseries, such as audio, sensor data, ECGs or text. The kernel in this case is one dimensioned and is shifted across the input in one direction.

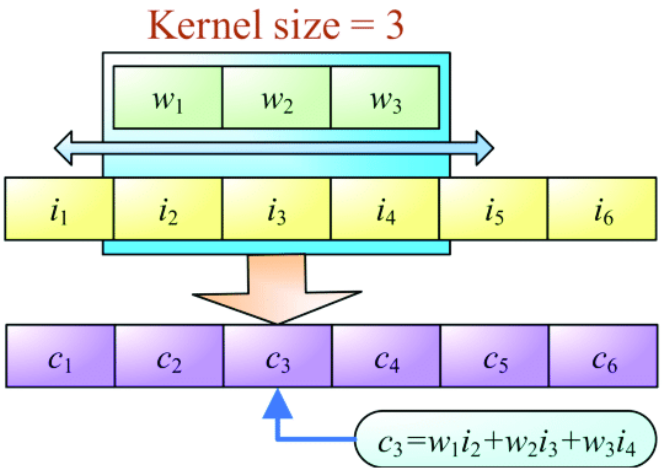


Figure 3.14: 1D Convolution Example [46].

It is a simple mathematical operation with small requiremnts in computational costs.

2D Convolution

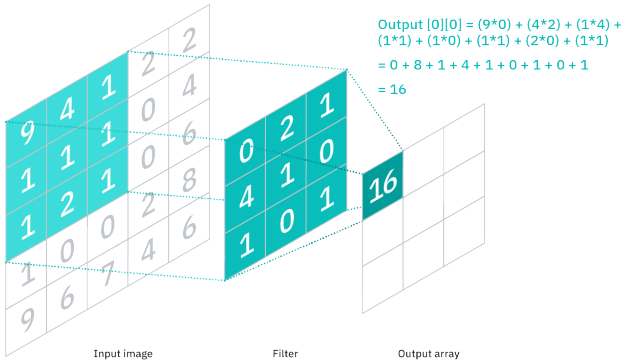


Figure 3.15: 2D Convolution example [47].

Usually, kernels are square-shaped, for example 3×3, 5×5, where the increase of the kernels dimensions, results in higher, more intense features found. There are cases of non square shaped kernels, where essentially one dimensioned convolution is computed [1].

Finally, it is worth mentioning that in order to reduce training time, separable convolution and depthwise separable convolution [48] have been proposed, computing convolution with less mathematical operation.

3D Convolution

It a convolution made on 3D data, such as video and MRI images. Due to the dimensions, higher requirements in hardware occur in contrast with previous convolutions. Lastly, kernel also has 3 dimensions as well.

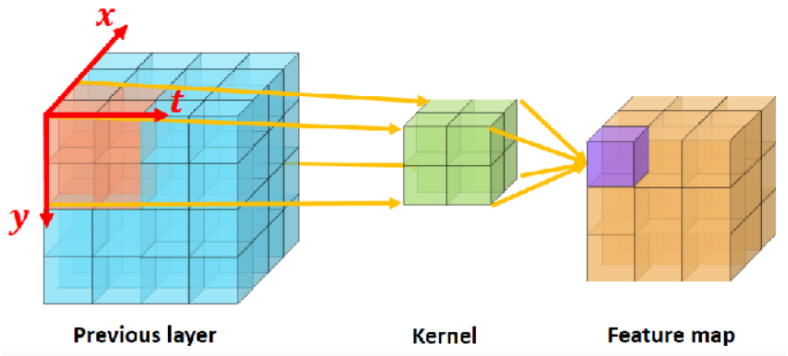


Figure 3.16: 3D Convolution example [49].

3.4.2 Pooling Layer

These layers reduce the size of the feature map, reducing in this way the complexity of the network and the chance of overfitting. These are divided into global [50], max and average pooling. It has been noted that using max-pooling, stronger features are selected, while by using average-pooling a selection of more smooth features occurs. Which layer to use is dependent on the case. Further division can be made regarding dimensions, with 1D pooling, 2D pooling and 3D pooling.

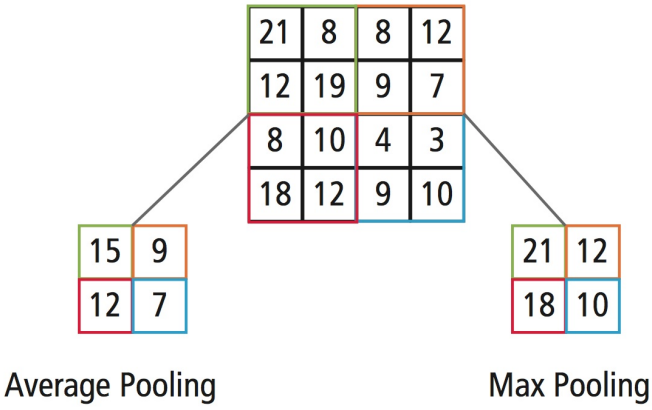


Figure 3.17: Max Pooling and Average Pooling 2D examples [51].

In the case of simple pooling, from each feature map either the maximum or the average values are selected, depending on the layer used. In contrast, by using global pooling, a value representative of each feature map is selected, resulting in significant parameter reduction. Furthermore, global pooling layers can be used in place of the output layer, where classification is made. This means overfitting can be avoided, in regards to that layer [50].

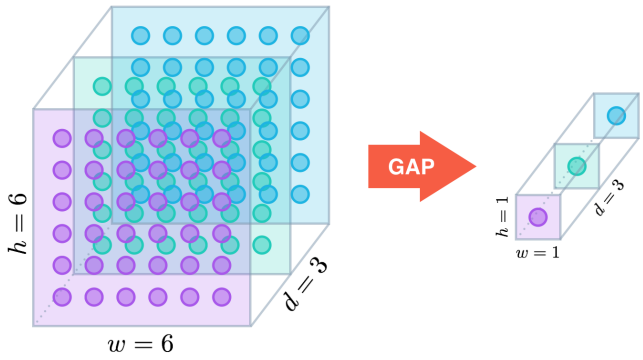


Figure 3.18: Global Average Pooling example[52].

Two parameters must be defined for this kind of layer. Initially, the window size, which will extract the value. The higher the value is, the stronger the sub-sampling will be, while the reverse is true for lower value. The second parameter that must be defined is stride, how much the window will shift after one sub-sampling. In this case as well, higher value results in intenser sub-sampling.

3.4.3 Fully Connected Layer

Essentially, fully connected layer consists of multiple percetron units. Each neuron of this layer is connected to each neuron of the previous layers, resulting in substancial increase of trainable parameters and in danger of overfitting, if their use is not careful. Usually, they are placed in the last layers of CNNs, although proposals have been made to replace them with a combination of Convolution layers with Global Average Pooling Layers [50].

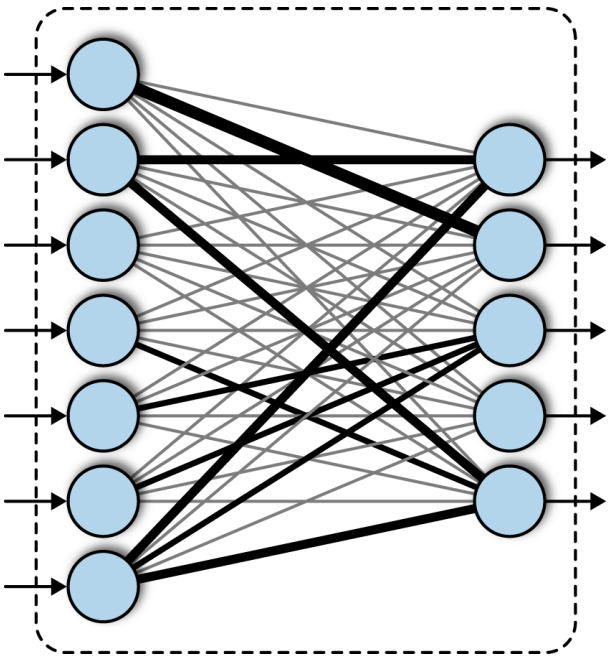


Figure 3.19: Fully connected Layer Visualization [53].

3.5 Activation Functions

Activation Functions are used to help the network learn complicated patterns and features. It applies a non-linear transformation to the output of each neuron. An important requirement for activation functions is that they need to be differentiable, in order for the back-propagation algorithm to be applied. Some of the most important ones are analysed in the following sections.

3.5.1 Sigmoid Function

Sigmoid function is a mathematical function that when plotted seems a lot like the "S" letter [54]. Depending on how it is defined, the neuron output is adjusted to values from 0 to 1 or from -1 to 1. The foundation of this function is the biological neuron function. Especially when sigmoid is defined from 0 to 1, the function is expressing probabilities.

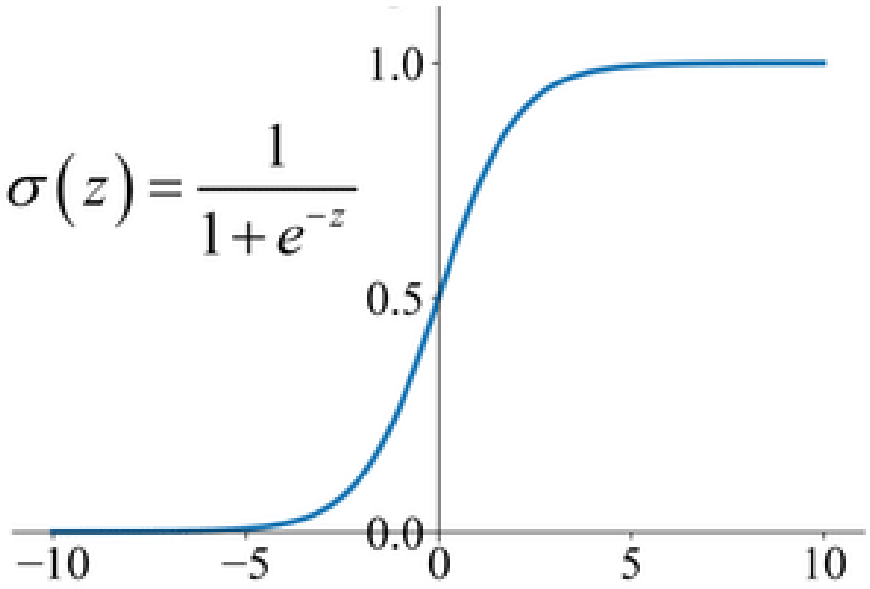


Figure 3.20: Sigmoid function plot [55].

3.5.2 Hyperbolic Tangent Function (Tanh)

It is similar to the sigmoid function, with the difference being that the output is in the range of -1 to 1 and centered around zero. It is more efficient than sigmoid in multi-layered perceptrons, but as the sigmoid, the vanishing gradient problem can appear.Furthermore, for both activation functions an exponential element must be computed, resulting in higher computational costs. Usually, in binary classification tanh is used for hidden layers and sigmoid at the output layer.

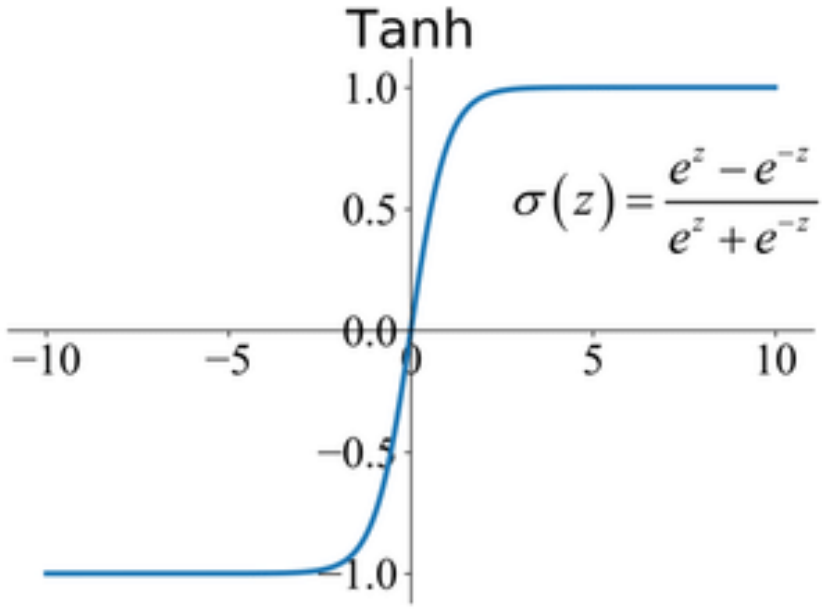


Figure 3.21: Tanh Plot [55].

3.5.3 Rectified Linear Unit(RELU)

In order to deal with the disadvantages of previous functions, the Rectified Linear Unit or RELU was proposed [56]. It is arguably the most popular activation functions, with its output being matched to its input for positive values, while the negative values are set equal to zero.

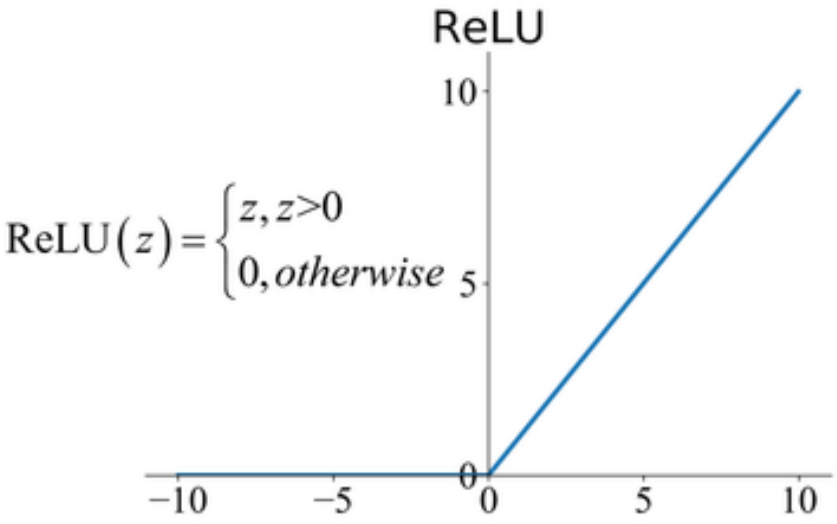


Figure 3.22: RELU Plot [55].

Due to the linearity of this function, its computation is quick and easy, in contrast with the exponential elements of Sigmoid and Tanh functions. However, new disadvantages appeared, with the most important one being the Dead Neuron Problem (or Dead RELU problem). As it can be deduced from the plot above, if the neuron's input is negative, then its output is zero. If the input is constantly negative, then the neuron is never activated and as such never trained.

3.5.4 Leaky RELU

Leaky RELU was exclusively designed to deal with the dead neuron problem caused by RELU, while keeping all the advantages of its use. Thus, a small gradient is used in the negative area, with the use of a coefficient a. As such, negative values are matched to very small ones, instead of zero so there are non-zero activations.

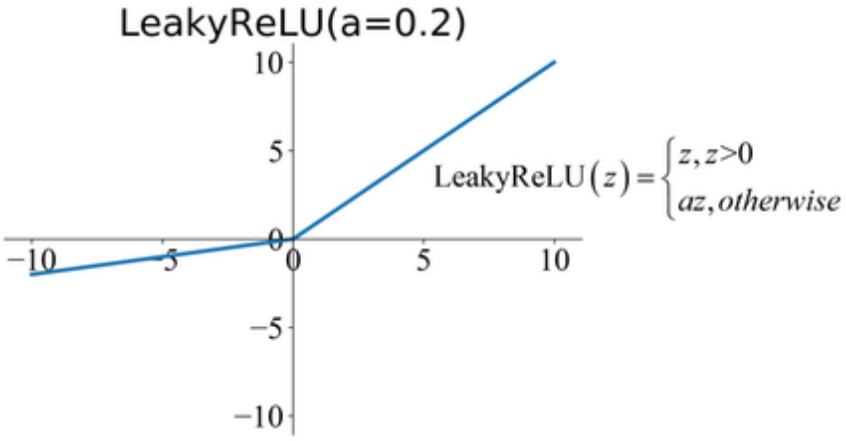


Figure 3.23: Leaky RELU plot [55].

3.5.5 Exponential Linear Unit (ELU)

ELU [57] results in faster minimization of the activation function and because of its value range, the average of the activations is near zero faster, i.e. faster learning process. Also, adjustment to negative activations, due to the exponential part, is made more smoothly than RELU and Leaky RELU. The parameter a influences how steeply or smooth the transition of negative activations will be.

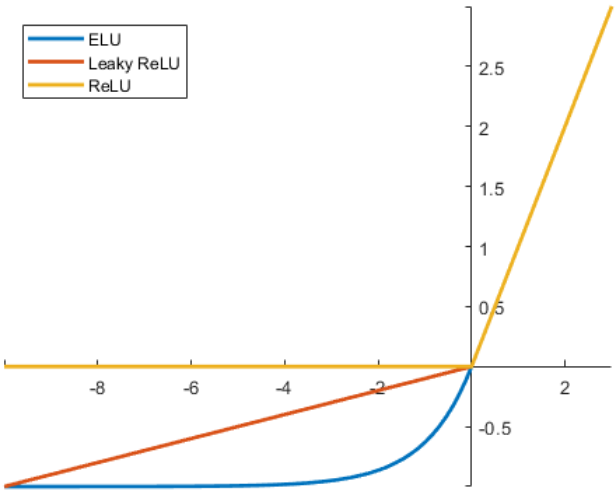


Figure 3.24: Comparison plot of ELU, RELU and Leaky RELU. [58].

Of course, due to the exponential part, this function is computationally heavier than the rest RELU activation functions.

3.5.6 Gaussian Error Linear Unit (GELU)

This function was recently used with transformers in NLP applications and achieved the best results compared to other architectures.

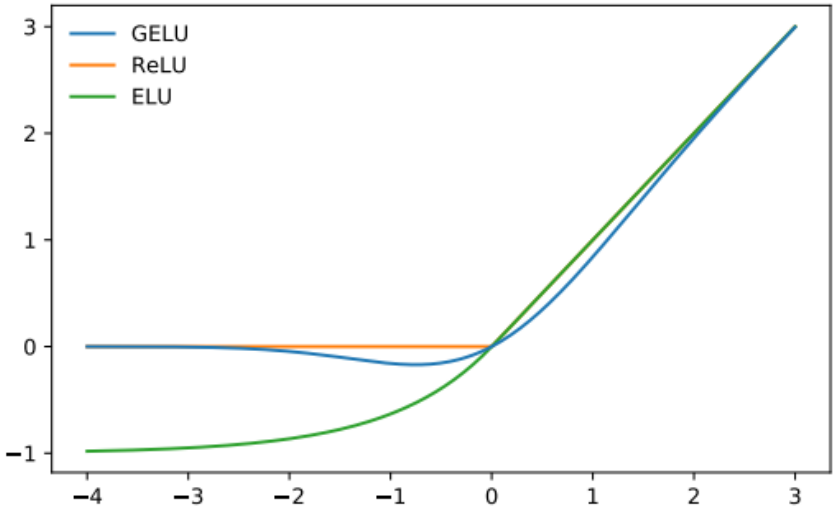


Figure 3.25: Comparative plot of GELU, RELU and ELU [59].

It is one of the most recent activation functions and there is not enough research yet supporting its superiority compared to other activation functions.

3.5.7 Softmax

The most common usage of softmax is in the output layer of neural networks, when multi-class classification is the network's objective. This is due to the fact that a vector of any dimensions is compressed to a vector of same dimensions, with the sum of its values equal to 1, representing probabilities. Also, it is used in transformer layers, to smooth and normalize the result of the attention mechanism.

3.6 Neural Network's Training

3.6.1 Epochs

An epoch is defined as the passing of all the training samples through the network once. Small epoch number will propably result in underfitting, i.e. the inability of the network to correctly classify the training samples. Conversely, large epoch number increases the chances of overfitting, the successful classification of the training samples only and the inability of generalization to new data.

3.6.2 Batch Size

Ideally, training data should pass the network together, in order for the weight changes to be more representative. However, due to hardware limitations, mainly from RAM and GPU cards, where data is loaded and processed, this is not possible. Thus, the number of samples that can be passed through the network must be defined. Furthermore, the change of the weight's value is done after the pass of one batch. So, if batch

size is small, then the weights change often, which may not be an optimal choice, especially when outliers or noise is inside the data. Usually, batch size is set as a power of 2, for example 32, 64 and so on, depending on the size of RAM.

3.6.3 Learning Rate

Learning rate probably is the most important hyper-parameter of neural networks. It defines how large or minuscule the changes in weights per epoch are. High value causes sudden changes in weights and may speed up training, but can also lead into a state of non convergence. In contrast, low values result in smooth changes of weights, but larger training time and the possibility of trapping learning in some local minimum. Its value varies, depending on the problem, typically it is set equal somewhere in the range of 0.1 to 0.0001. Finally, it is possible to have a changing learning rate, computed by some equation or some other method.

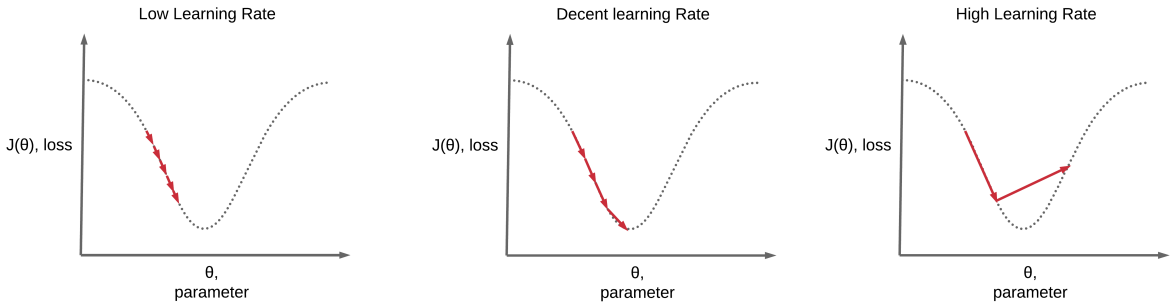


Figure 3.26: Cost function behavior depending on learning rate value [60]

3.6.4 Cost Function

Cost function is the mathematical equation that is dependent on the trainable parameters of the networks and its inputs, and is an indication of training. The objective is to minimize its value, while generally, as long as its value is decreased, the network is better trained. In classification problems, the main function used is cross-entropy [61], whereas if the problem is binary, then its binary counterpart is used (binary cross-entropy), while for more classes categorical cross-entropy is used. Cross-entropy's equation is:

$$J = -\sum_{i=1}^{N} y_i * \log \hat{y}_i \tag{3.1}$$

where N is the number of classes, y_i is the actual class of the n-th sample and $\hat{y}_i$ is the network's prediction for the n-th sample. Finally, depending on the number of classes, one-hot encoding can be used to represent the classes or the classes can be represent by a unique integer.

3.6.5 Momentum

With momentum, the acceleration of minimizing the cost function is achieved, along with smoother training process. Its value is usually set to 0.9 and it is placed to the weight changing equation. Therefore, as long as cost function value is reduced, the use of momentum speeds up this decrease in a way. Thus fluctuations are also decreased and a smoother training is obtained.

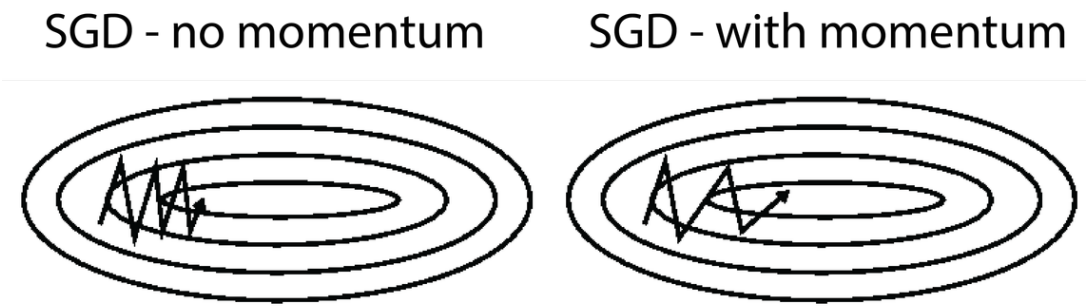


Figure 3.27: Visual representation of SGD algorithm without and with momentum [62].

Momentum is defined as:

$$u_{t+1} = \mu * u_t - \eta * \nabla f(\theta_t) \tag{3.2}$$

$$\theta_{t+1} = \theta_t + u_{t+1} \tag{3.3}$$

where μ is the momentum variable, i.e. to what extend the gradients of previous steps remain , θ_t are the training parameters and η is the learning rate. u_t amasses the gradients of the previous steps and applies the change to the θ_{t+1} parameters.

Quite recently the Nesterov momentum [63] was proposed as a variation of Adam algorithm, named Nadam [64]. There, instead of using the previous equation, Nesterov momentum is used, computed by:

$$u_{t+1} = \mu * u_t - \eta * \nabla f(\theta_t + \mu * u_t) \tag{3.4}$$

$$\theta_{t+1} = \theta_t + u_{t+1} \tag{3.5}$$

where the $\mu * u_t$ is now added to the equation calculating u_{t+1}.

3.6.6 Optimization Algorithms

The objective of optimization algorithms is the minimization of a cost function by means of finding a local minimum and as such the convergence of learning. Thus, by the decay of the cost function, the difference between the network's prediction and the actual labels is minimizing and the network successfully classifies its input. The two most common optimization algorithms are Adam [65] and Stochastic Gradient Descent(SGD) [66], while quite recently AdaBound [67] was proposed, an algorithm trying to merge the benefits of Adam and SGD.

Stochastic Gradient Descent (SGD)

It is one of the oldest optimization algorithms. The weight values change with each sample, which may contribute to less stable training, especially if the data are noisy. Generally, it leads to better results than Adam. What is more, sufficient adjustment must be made to the algorithm's parameters, to yield optimal results. The equation of weight renewal is:

$$\theta_{t+1} = \theta_t - \eta * \nabla J \tag{3.6}$$

where θ_{t+1} and θ_t are the next and previous weight values, η is the learning rate and ∇J is the gradient of the cost function.

Adam

Adam is the more commonly used algorithm, with its name derving from Adaptive Momentum. Its advantage is the fact that learning rate is changed, to find the proper value of learning rate for each parameter separately. It was designed in an effort to combine the positive aspects of two other algorithms, Adagrad and RMSprop. The equation for this algorithm is:

$$\theta_{t+1} = \theta_t - \eta * \frac{\hat{m}}{\sqrt{\hat{u}} + \varepsilon} \tag{3.7}$$

where θ_{t+1} and θ_t are the following and previous weight values accordingly and η is the learning rate. Also, $\hat{m}$ and $\hat{u}$ are non-polarized values of exponential averages and deviations, with ε being added to avoid a possible division by zero.

Even though learning time is drastically reduced, depending on the case, lower generalization ability is achieved than SGD. In order to solve this, a modification to the algorithm was proposed, named amsgrad [68], while previously the use of weight decay [69] was also proposed.

Nadam

It is an Adam variation, with the difference being the use of Nesterov momentum [64]. Therefore, the weight update rule is converted to:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{u}_t} + \epsilon} \left(\beta_1 \hat{m}_t + \frac{(1-\beta_t) g_t}{1-\beta_1^t} \right) \tag{3.8}$$

The variables $\theta_t, \eta, \hat{m}_t$ were explained previously, with variables β_1^t and β_1 being the new parameters, while g_t expresses the gradient.

AdaBound

The solution to the problems of SGD and Adam is given by the recent Adabound algorithm. In this algorithm, dynamic limits are set to the values of learning rate, where initially the lowest limit is zero and the highest is infinity. While training progresses, the two limits are converging to a value. So, when training begins, the algorithm is similar to adaptive methods such as Adam, while near the end it is similar to SGD. Thus, a faster initial convergence is achieved, along with the generalization ability provided by SGD. Weight renewal is computed according to:

$$x_{t+1} = \Pi_{\mathcal{F}, \mathrm{diag}(\eta_t^{-1})} \left(x_t - \eta_t \odot m_t \right) \tag{3.9}$$

Finally, the authors suggest the use of Adabound to models with high depth and not for simple architectures, while they also suggest the amsbound algorithm, in accordance to the Adam variation.

3.6.7 Class Weights

When dealing with class imbalance in the training dataset, i.e. in unequal distribution of samples for each class, a weight can be given to each miss represented class. For example, when a wrong classification is made for the class with the least samples, this mistake is more costly than a mistake being made on a class with more samples.

3.7 Generalization techniques

Often during the training of neural networks, over-fitting occurs, i.e. the network learns too well the training samples. So, if new input is given, then the network fails to correctly classify them. There are different ways to combat this problem, analysed in the following subsections.

3.7.1 Dropout layer

By using dropout layer, during training, a random number of neurons is deactivated, not altering their weights, therefore not being trained [70]. User set the probability p with which neurons are deactivated.

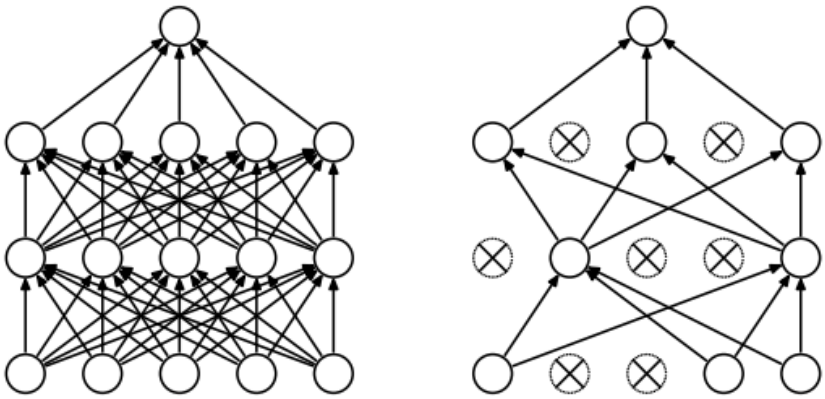


Figure 3.28: Network before (left) and after (right) the use of dropout [70].

3.7.2 Normalization

The inputs of the hidden layers constantly change, resulting in their distribution changing as well. With normalization, this problem is dealt with, as the behavior and stability of the network is improved. Different forms of normalization exist, with the most common being batch normalization [71],layer normalization [72], instance normalization [73], group normalization [74] and weight normalization [75].

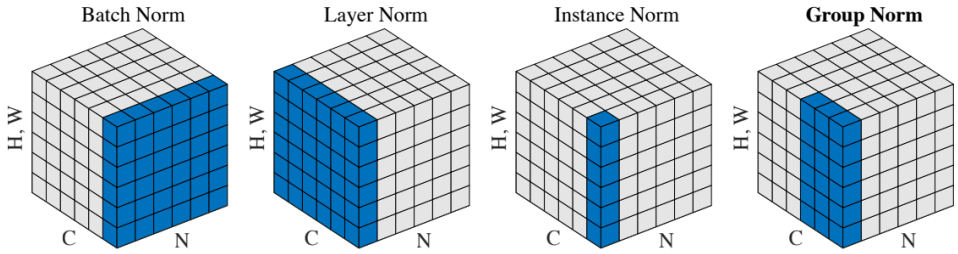


Figure 3.29: Different kinds of normalization [74]

Batch Normalization

According to this method, for each batch during forward propagation, normalization is applied in a way that the average is equal to zero and standard deviation is equal to one.

Layer Normalization

In contrast with the previous normalization, layer normalization applies the aforementioned calculation to the responses of each layer, instead of each batch. It is used mostly on recurrent neural networks and on the transformer blocks of the transformer networks.

Instance Normalization

It is an expansion of layer normalization concepts, with the difference being that normalization is applied on each channel individually. Someone can easily deduce that this is used when the network's input is three dimensional.

Group Normalization

This case of normalization is achieved by applying the computation to groups of channels. It is an intermediate method between layer normalization and instance normalization.

Weight Normalization

Totally different is weight normalization, which redefines the weight vectors, decoupling the direction from these vectors. As such, convergence is achieved as well as a reduction in computational complexity.

3.7.3 Data Augmentation

More often that not, results of neural networks are not satisfying, as the data used are not enough to sufficiently train them. Thus, new data are needed to be produced as inputs during training, to help the network respond effectively to any kind of new inputs. Some examples are rotation, vertical or horizontal flip, cropping and noise addition. Their use is depedent on the problem.

3.7.4 Regularization

Regularization is used during training and is applied to the cost function. More specifically, the objective is to the fluctuations caused by certain training samples. So, a factor is placed to the cost function, evaluating the model as to not receive immense values. Three types of regularizations are most commonly used, L1 regularization, L2 Regularization and their combined use.

L1 Regularization

L1 Regularization or LASSO [76] is calculated as:

$$L_{new} = L_{old} + \lambda * |w|_1 \tag{3.10}$$

where L_{new} is the new value of the cost function, L_{old} is the former value, λ is a parameter which determines how strong the regularization will be and $|w|_1$ is the L1 norm of the weight vector, or otherwise the sum of absolute values.

L2 Regularization

L2 Regularization or Ridge regularization [76] is defined as:

$$L_{new} = L_{old} + \lambda * |w|_2^2 \tag{3.11}$$

where $|w|_2$ is the L2 norm or the Euclidean measure of the weight vector. In both cases, λ values range from 0.01 to 0.001.

3.7.5 Learning rate Decay

In the case where learning rate is decreased with specific pace, the methods of exponential decay, inverse time decay and cosine delay or restart are used [77]. In this way, a smoother search of local minima is sought as the weight change is smaller during training. In case the process is trapped in a non local optima point, restart of learning rate can be used to search for a local optima.

Exponential Decay

Exponential decay is computed as:

$$\eta = \eta_0 * e^{-k*t} \tag{3.12}$$

where η_0 is the initial learning rate, η is the new learning rate, k is the decay rate and t is the number of epochs. As such, when this decay is used, learning rate follows a similar trajectory as the plot in figure 3.30.

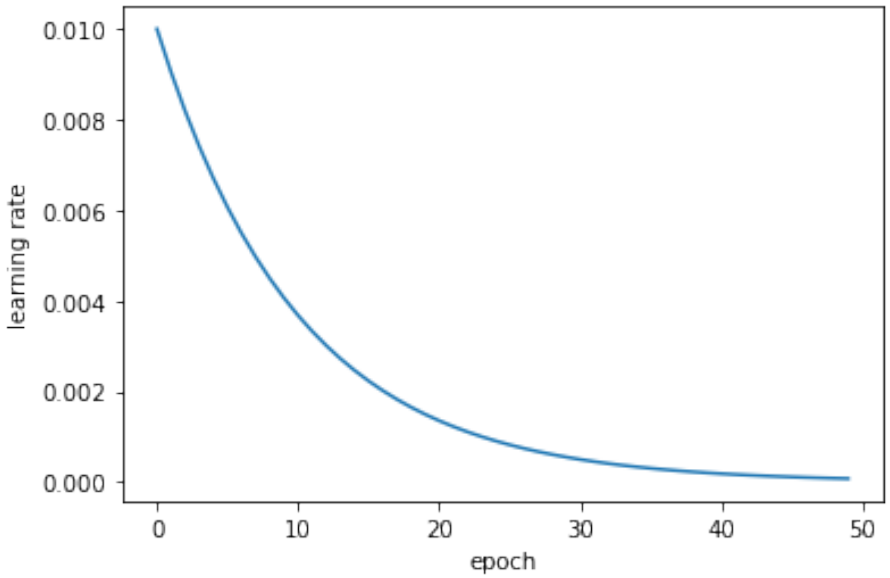


Figure 3.30: Plot of exponential decay of learning rate.

Inverse-time Decay

In this simple case, the new learning rate is calculated as:

$$\eta = \frac{\eta_0}{1 + k * \frac{t}{step}} \tag{3.13}$$

The plot has the following look, as depicted in figure 3.31.

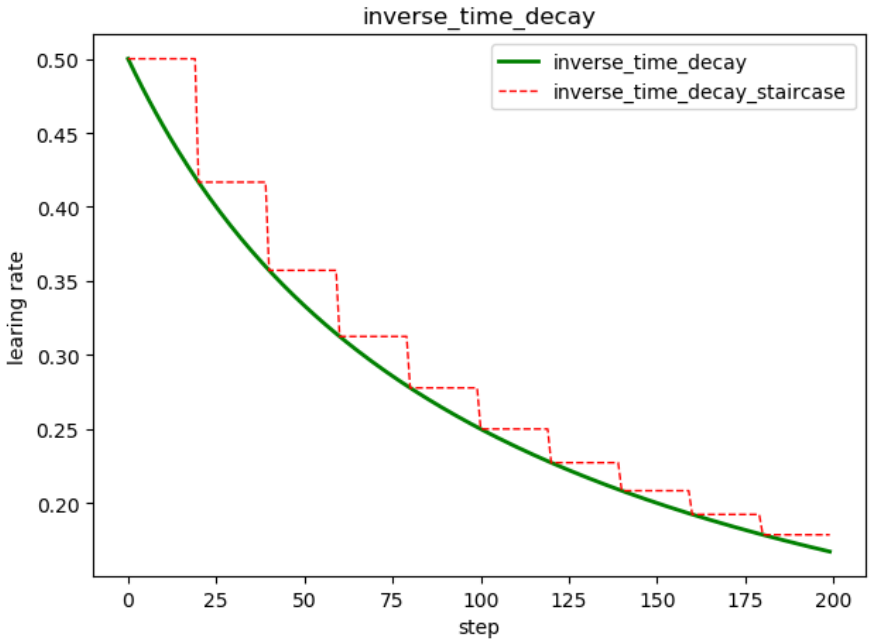


Figure 3.31: Plot of inverse time decrease of learning rate.

Cosine Delay - Restart

It is a method, initially proposed to be used with SGD algorithm, but then its use was generalized to other algorithms. According to the following equation:

$$\eta_t = \eta^i_{min} + \frac{1}{2}\left(\eta^i_{max} - \eta^i_{min}\right)\left(1 + \cos\left(\frac{T_{cur}}{T_i}\pi\right)\right) \tag{3.14}$$

where η^i_{min} and η^i_{max} are the minimum and the maximum value of learning rate and T_{cur} refers to the epoch number from last restart. Restart is defined the reversal of learning rate to its initial value. In that way, a relatively high learning rate is used initially, where it quickly decreases to the minimum value set and then increases again to the maximum set value. This process is repeated under the predetermined number of cycle is completed.

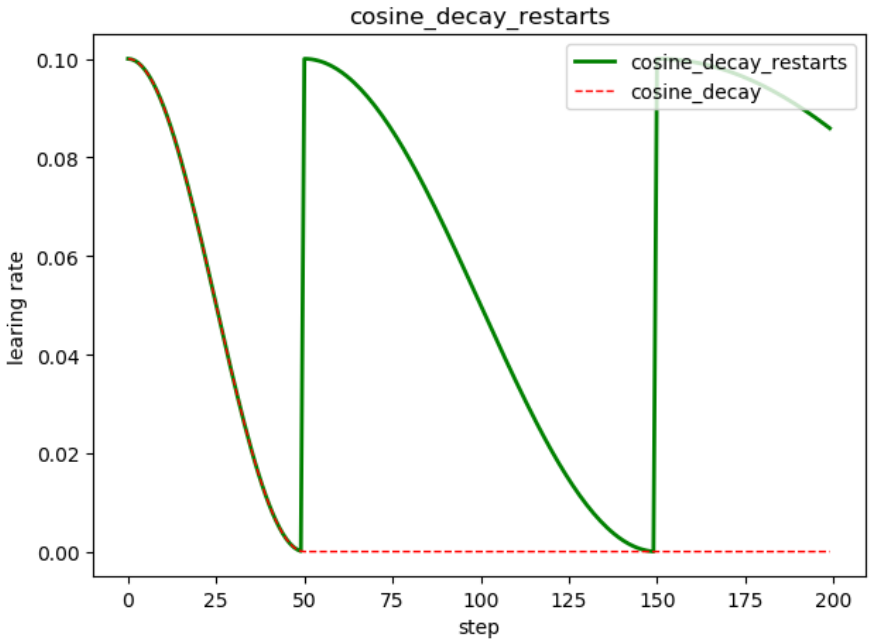


Figure 3.32: Comparison of cosine decrease of learning rate (green) and without restart(red).

3.7.6 Callback

Using callbacks, the effect of over-fitting can be minimized. There are different kinds of callbacks, with the most common implementing a gradual decrease in learning rate, either by a specific equation, or by certain epochs, even if no improvement is noticed in a metric. Also, if no improvement is achieved, there is the option to restore weights of a previous epoch, where the optimal efficiency of the model was achieved.

3.8 Models Evaluation Metrics

3.8.1 Metrics

There are numerous ways to count the ability of one network to correctly classify its input. These ways are called metrics. The easiest method is accuracy, i.e., how many correctly classified samples there are divided by the total number of samples, expressing the efficiency of the model for all classes.

$$accuracy = \frac{correct\ classifications}{total\ classifications} \tag{3.15}$$

Another metric is precision [78], defined as:

$$precision = \frac{tp}{tp + fp} \tag{3.16}$$

As in the ability of the model to correctly classify a sample of the x class. Finally, the recall and f1-score metrics can be found, calculated as:

$$recall = \frac{tp}{tp + fn} \tag{3.17}$$

and

$$f_1 = 2 * \frac{precision * recall}{precision + recall} \tag{3.18}$$

where tp are the correctly classified samples of n-th class and fn are all the samples labeled as n-th class but actually are not. F1-score basically is the harmonic mean of precision and recall.

3.8.2 Confusion Matrix

Confusion Matrix is the visualization of the classification results. In one axis, the network's predictions are placed, while on the other the actual labels of the samples can be seen. The main diagonal displays the correct classifications, while the other spots on the matrix are the wrong classification per class. As such, it can be deduced whether the model correctly predicts for each class or not and if it is partial in its prediction(because its data is unbalanced).

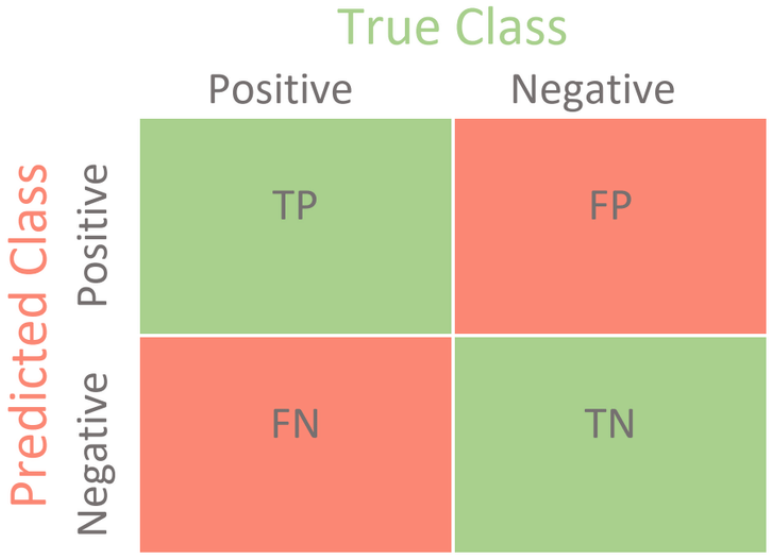


Figure 3.33: Confusion Matrix example for binary classification [79].

Chapter 4

Indoor Sound Classification

There is an increasing interest in surveilling and improving life at home by using different types of sensors, including microphones. In the context of Ambient Assisted Living (AAL) people are being watched, for example in support of elderly people or individuals with chronic diseases, having their activities monitored. When examining a way of audio detection, an activity can be considered as an acoustical scene. Recently, audio detection and classification of acoustic scenes has captured the interest of research community and has led to numerous public databases for a wide range of applications.

In this chapter, an analysis is made on the datased used.Also, the pre-processing methods used will be under the microscope, along other techniques that could optimize the results.

4.1 Software development environment

The programming language used is Python (version 3.9.0) [80]. For reading and extracting features from the audio files library librosa was used [81]. The neural networks were developed using Tensorflow [82] - Keras [83], while for different extra capabilities around machine learning sklearn [84] was utilized. Furthermore, reading csv files was made available by pandas [85] library and for visualization purposes matplotlib [86] was used. Finally, arithmetic operations were computed with the help of numpy [87].

4.2 Hardware used

The main development environment was as follows:

- 2x8GB CORSAIR RAM clocked at 2993 MHz
- NVIDIA GTX 1660 OC 6GB GPU
- SSD 500GB Western Digital
- M2.NVE SAMSUNG EVO PLUS 250GB
- AMD Ryzen 5 2400G with frequency equal to 3.6 GHz
- Gigabyte Aorus Gaming Plus Motherboard

Furthermore, models were developed also in Google Collaboratory [88], where RAM is equal to 12GB with the option to expand to 26GB and the GPU is either TESLA P100 16GB or TESLA T4 16GB.

4.3 Dataset

4.3.1 Data overview

The dataset used derives from a processed version of SINS dataset and was also used in DCASE 2018 [89] competition. It is consists of 10 second recordings, from the stay of a person in a house. 7 microphone arrays were used, with each one comprising of 4 microphones.

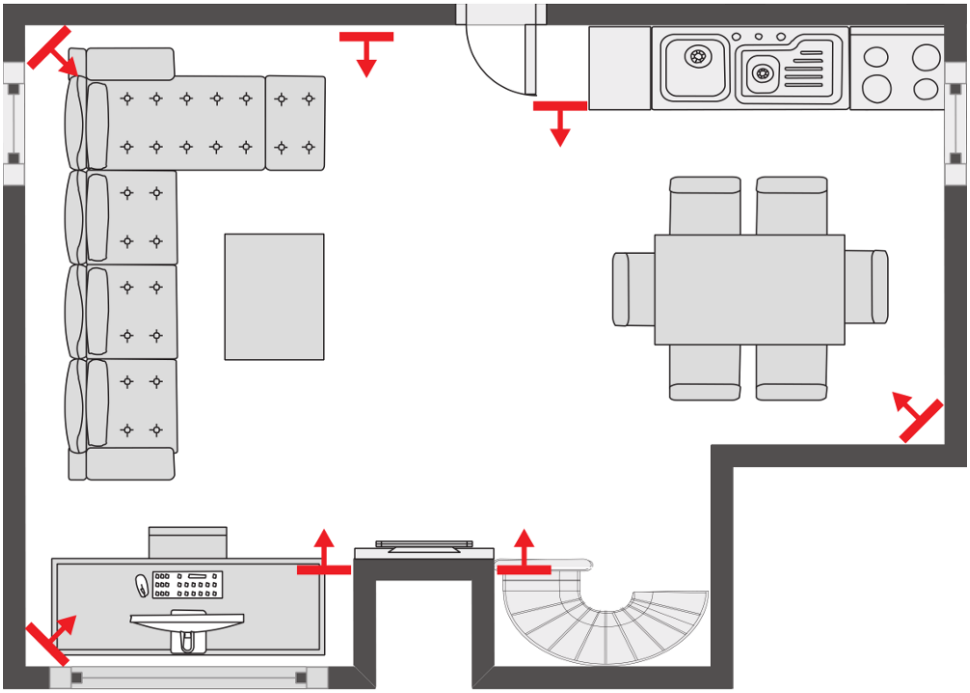


Figure 4.1: Top view of the room, with visible markings of microphone positions.

All of the sound clips describe one activity exclusively. These activities are one of the following:

- Absence
- Cooking
- Dishwashing
- Eating
- Other
- Social Activity
- Vacuum cleaning
- Watching TV
- Working

The duration of each audio file is 10 seconds, with sampling frequency equal to 16 KHz. Also, each audio file, due to the four microphones in each array, is four channeled. The dataset is split between a development set, used for training the network and an evaluation set, used for evaluation purposes. The number of samples is 72.984, with their distribution per activity as follows:

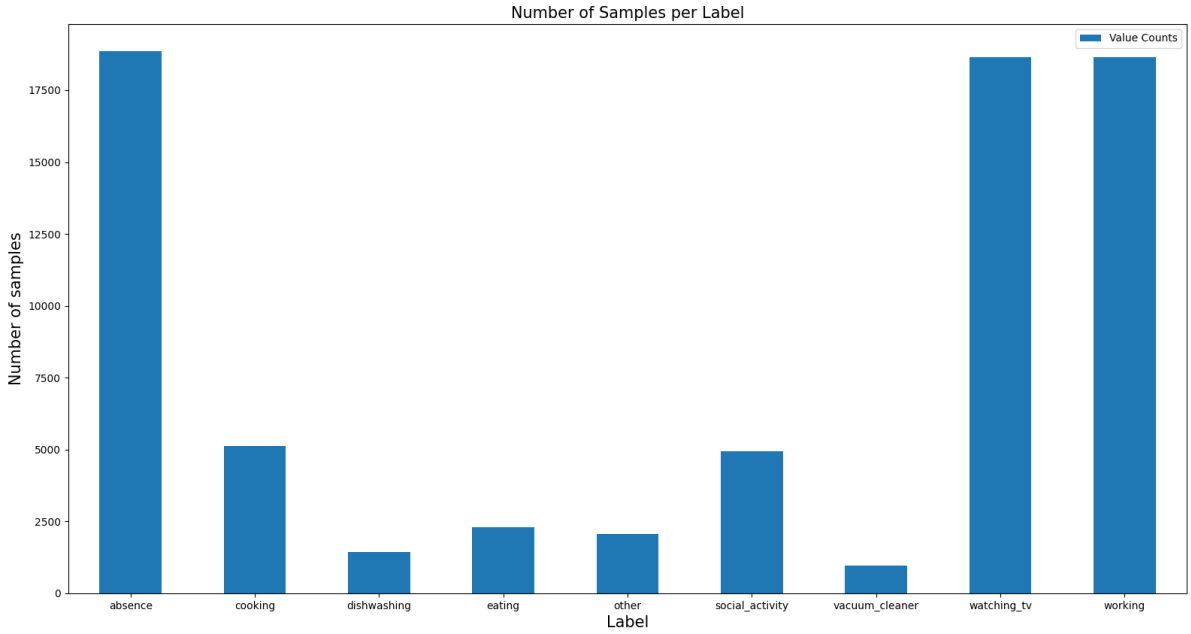


Figure 4.2: Class-based distribution of the dataset.

By randomly selecting a sample from each class and creating their spectrograms, some conclusions can be made, which can possibly be the basis of the features used by the neural networks.

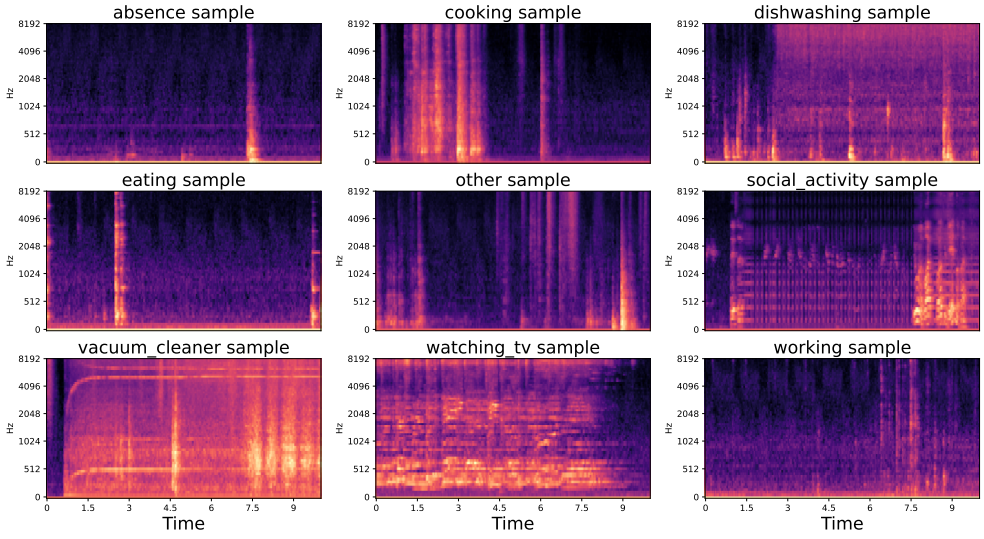


Figure 4.3: Sample spectrograms for each class.

During the absence of activity, at random intervals a sound can be noticed, as in a recording of any object, while during cooking different sounds can be seen such as

water boiling or stirring, explaining the frequency activity in random moments and frequencies. In addition, it can also be surmised that washing the dishes displays activity in random times, while when eating peaks in spectrogram could be sounds of eating utensils. Examples containing social activity and television watching are similar, due to the fact that people are talking in them, so frequential information can be found in different frequencies with varying volume. Recordings contain repeating sounds, such as vacuuming are expected to contain information in higher frequencies with repeating duration, from the vacuum cleaner's engine. Finally, samples with various activities are placed under the other label.

4.3.2 Audio features

The audio files were initially read in the frequency they were recorded, i.e. 16KHz. The features that were used are the MFCC and the mel-spectrograms. Various coefficients were tried, most notably 32, 64, 128, 256 to study their effect, while for MFCC 20 coeffiecints were also used.Hop length, window size and the number of FFT coefficients were equal to 512, 1024, 2048 samples respectively, translating in to the time domain to 32ms, 128ms, 128ms accordingly. These values were set arbitrarily and are not based on previous research.

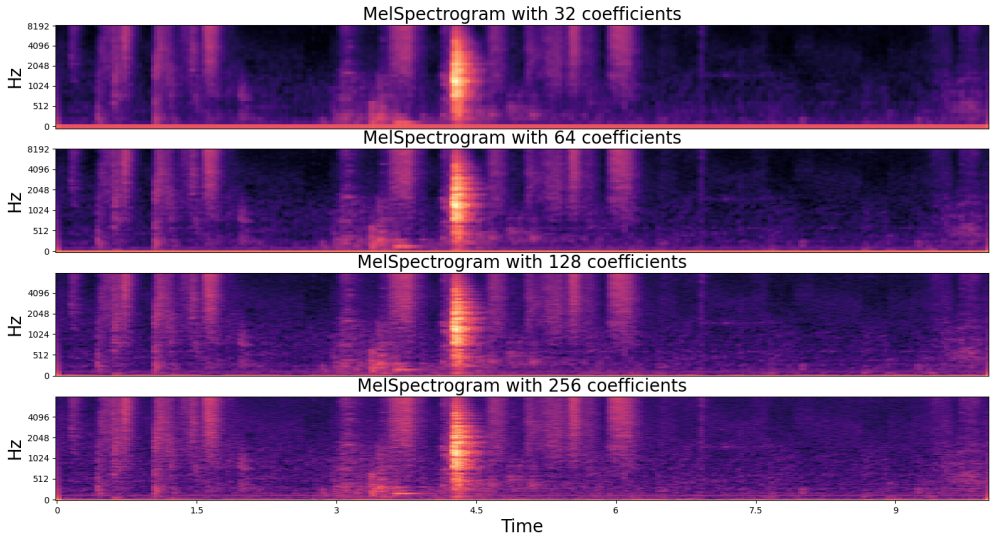


Figure 4.4: Mel-spectrograms for each number of coefficients.

As it can be observed, increasing the number of coefficients increases the frequency discretion, at the cost of icreased computational costs in order to produce these images.

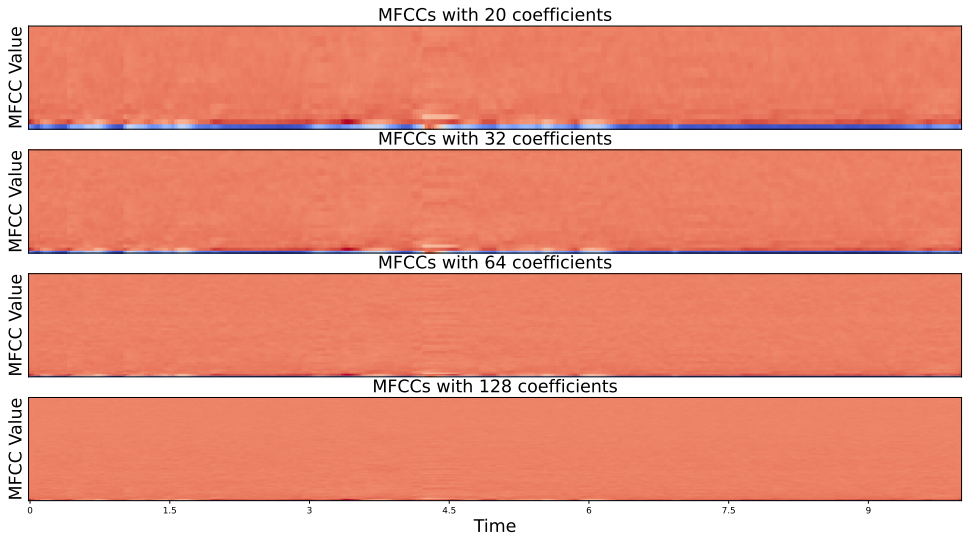


Figure 4.5: MFCCs for each number of coefficients.

Now, the problem of audio classification has been transformed into image classification, where images used comprise of spectrograms and MFCC visualizations. Grayscale coloring has been selected, because using RGB is not information worthy and because more channels would increase the complexity of the neural networks.

What must be noted is that no pre-processing was applied to the audio files, as it was deemed that it would not help, while the audio files do not have any noise recorded, as it was a controlled recording environment. However, beamforming was tried, without any benefits, so their use was rejected.

4.3.3 Data augmentation

Not a lot of augmentations can be made on spectrogram visualizations, however recently three were proposed: time masking, frequency masking and time warping [90]. Time masking transformation essentially masks frequency changes throughout time, giving the chance to the network to fill in the gaps, while frequency masking works similarly, with frequencies in a small duration being masked. Finally,with time warping, a piece of the original spectrogram is randomly selected and is placed onto another place of the spectrogram. All of the three methods contain the random parameter, whether it is the size or the placement of the mask.

Due to hardware limitations, input images were not able to be produced during the training of each network, as the speed of the training would also increase dramatically. As such, the input images were stored and then read as batches when training occurred.

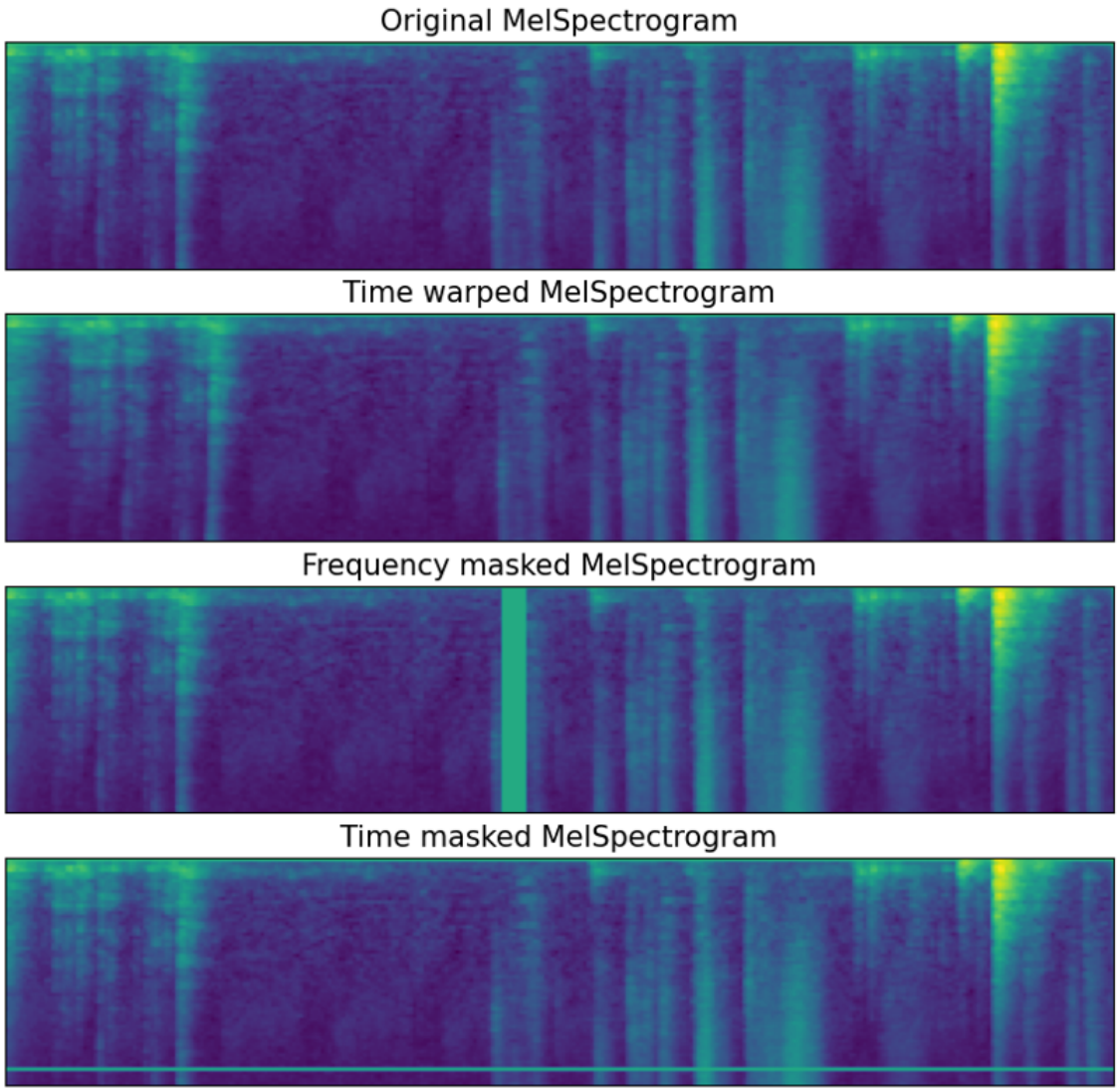


Figure 4.6: Augmentations used. Top to bottom: initial spectrogram, time-warped spectrogram, frequency-masked spectrogram, time-masked spectrogram.

4.4 Image Pre-processing

The only method of preprocessing that was used, was rescaling from $[0, 255]$ to $[0, 1]$. This is computed by dividing with $\frac{1}{255}$ each pixel and is applied as a network layer. Commonly used transformations, such as mirroring, do not have any physical meaning to be applied to audio-based images.

Chapter 5

Deep Learning Architectures For Indoor Sound Classification

In this chapter a thorough analysis is made on different architectures that were researched, trying to achieve the highest macro F1-score possible, the metric used in DCASE 2018 Task 5. As secondary target, the creation of a model with the minimum parameters possible was set, without comprising the model's accuracy. Creating and testing the networks needed a period of 7 months. Finally, the images used, with coefficient number equal to 64, were used to compromise between information accuracy and computational costs.

5.1 Hyper-parameter Tuning

Batch size was set to its largest value possible, a value that did not overflow RAM or GPU's VRAM. Depending on image dimensions, values 64, 128, 256 were used. GELU was used as the activation function, because it outperformed RELU and Leaky RELU in my experiments.

Learning rate's value was 0.0004 in any case, as it led to satisfying results. Larger values led to unstable learning process and worse results. Furthermore, callbacks were used tracking validation loss, in a way that when validation loss was not imporving in 6 or 8 epochs, training was halted and the weights were restored to their values when validation loss was minimum. Also, learning rate decay was used, when validation loss did not improve for 2 consecutive epochs, decreasing learning rate to a percentage of the starting value.

Epochs were equal to 50, 75 ή 100, depending on whether overfitting occured in 100 epochs. What is more, experiments using regularization were conducted, where when it was used, L2 norm was equal to 0.001. Weights were not randomly initialized, but instead he function [91] was used. It has been proven that when using GELU or RELU as activation functions, using He functions leads to better performance. Also, in every network, a *Rescaling* layer is used, converting the pixels' values from 0 to 255 to a new range of 0 to 1. Finally, to deal with class imbalance, class weights were initialized in accordance with:

$$class_weight_i = \frac{n_i}{n_{total} * N} \tag{5.1}$$

where n_i are the samples of the i-th class, n_{total} the total samples in the dataset and N the number of classes.

5.2 1D Convolutional Networks

Usually 1D Convolution is used in time series. However, it is possible to utilize them in signals with two dimensions, if one considers them constant in one direction, void of information, with the convolution operation being calculated in the other dimension which contains the needed information. Thus, in the case of MFCCs, where the vertical axis contains their values and in the horizontal axis is the time unit, 1D Convolution can be used in the vertical axis [92]. The advantages of this method is the fact that significantly less computational resources are used and less complex mathematical operations are computed.

This model contains some layers that are pre-processing the inputs given. The images, in grayscale format, are converted to matrices, from dimensions $HxWx1$ to HxW. Then, axis are inverted, to apply convolution to the time matrix, leaving the information intact, the information being MFCC. Afterwards, instead of the common practise of using Batch Normalization, *Group Normalization* is used, with grouping parameter equal to 16, after experimentation. This value is preserved throughout the whole architecture. Then, the first *1D Convolution* layer is placed, with kernel of 3 and 32 filters. It must be noted that in all convolution and fully connected layers *Weight Normalization* is used. Thus, after the first convolution operation, the sequence of *Group Normalization*, *1D Convolution*, *Dropout & 1D MaxPooling* is placed, with the filters of convolution being 64, the kernel having a size of 3 and the dropout probability being 0.5. This layer chain is used 3 times, with the difference being the filter number, which are increased to 128, 256 and 512 each time. Finally, *Flatten* is applied and a sequence of *Dropout & Dense* layer follows, with the probability being 0.5 and the neurons being 128 and 9.

The optimization algorithm used is Adam, with amsgrad. Learning rate is $\eta = 0.0004$ and decay can be applied, if validation loss is not decreased for 2 epochs, in 93% of the its current value, while batch size is 256. Also, regularizer is used, using L2 and with $l2 = 0.001$. Epoch number is 75, with the ability to stop training if for 8 epochs no improvement is made.

In figure 5.1 the architecture desrcibed can be seen. It must be noted that *Weight Normalization* layers can not be seen, as they are included in the convolution and fully connected layers.

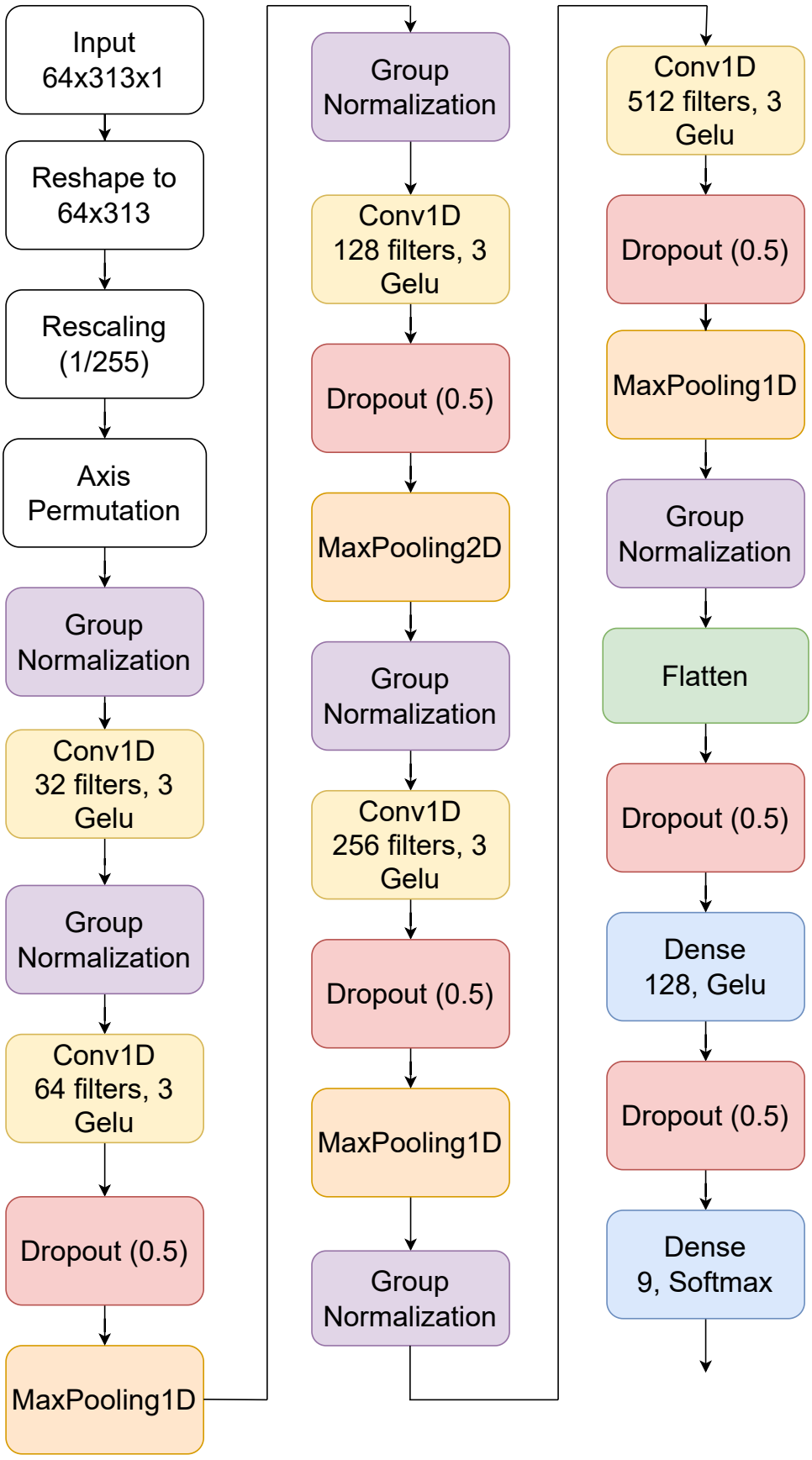


Figure 5.1: 1D Convolution Network.

5.3 2D Convolution Networks

Using 2D Convolution, information that lies in two dimensions can be found, for example in image data. In this thesis, the inputs that were used as images were the mel-spectrograms, containing information in both dimensions, as in the vertical axis frequencies can be found, while in the horizontal axis their development in time can be found. Of course, as the MFCC are stored in 2D matrices, they can be used in 2D Convolution as well.

5.3.1 Symmetrical kernels

Common practise is the use of symmetrical kernels in the convolution layers. In this implementation, kernels 3×3 are used. Convolution is performed in both axes, extracting information concurrently from them.

This model is following the common architecture of a CNN. As such, after *Rescaling* is a *Batch Normalization* layer, followed by a sequence of *2D Convolution - Max Pooling - Dropout*, with the number of filters in the convolutional being 64, 64, 128, 256 accordingly. Kernels' dimensions are 3×3, except for the last convolutional layer which uses a 5×5 kernel. Pool size in *Max Pooling* layers is 2×2, while the *Dropout* probabilities are 0.4 , 0.4 and 0.5. In order to use a fully connected layer at the end of the network, usually a Flattening layer is used to concatenate all the feature maps to a 1D vector. However, here a *2D Global Max Pooling* layer was used, as it outperformed the flattening layer and led to substantial decrease in trainable parameters. Then, a *Batch Normalization* layer is placed followed by a (*Fully Connected*) with 64 neurons, with the number chosen empirically, a *Dropout* layer with probability 0.5 and finally the classification layer with 9 neurons.

The network can be seen in figure 5.2 and uses the optimization algorithm AdaBound with amsbound and learning rate $\eta = 0.0004$ that is reduced to 90% of its value if for 3 epochs validation loss does not improve, and if for 8 epochs validation does not improve, training is halted. Epochs were set to 50 and batch size to 64. These values were selected after testing to achieve optimal F1-Score.

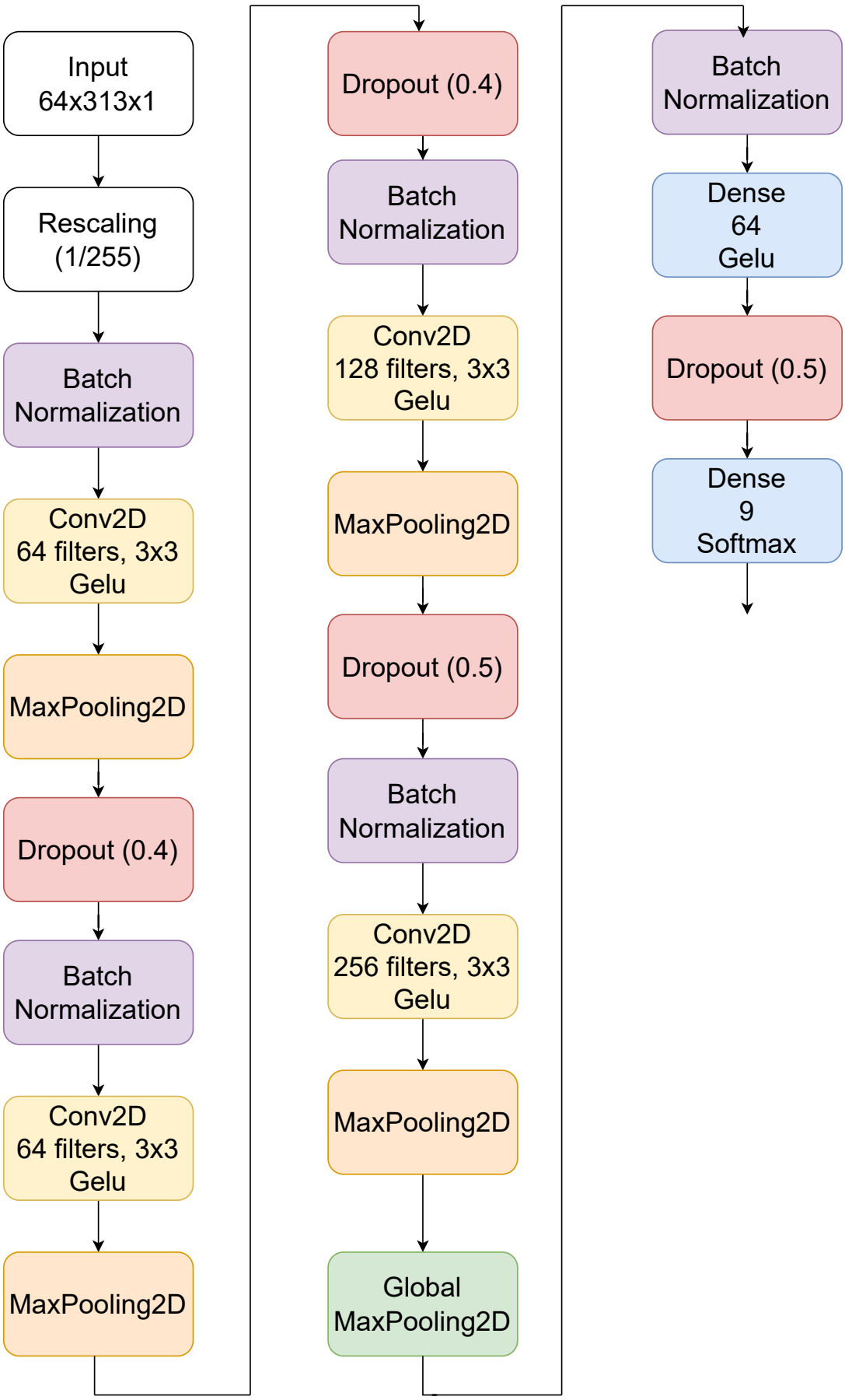


Figure 5.2: 2D Convolutional Network using symmetrical kernels.

5.3.2 Non Symmetrical Kernels

By using non symmetrical kernels, for example 3x1, convolution operation is computed in one axis only, keeping the other axis constant. Thus, for instance in the case of spectrograms, convolution can be calculated only in the frequency axis (vertical axis) and then in the time axis (horizontal axis). Therefore, the network can be first trained using information contained into the frequency spectrum and then in the time one. So when for example a kernel 3x1, convolution is performed on the frequency axis, while the reverse is true for a kernel of 1x3, where convolution is calculated for the time axis.

The network developed in this case is partially inspired by Innoue et. al.[1]. More specifically, after the Rescaling layer, *Batch Normalization* is applied and afterward the first *2D Convolution* layer is placed with 64 filters and kernel 5x1. These layers are followed by a *Dropout* layer with probability 0.2 and a *Max Pooling* layer with pool size 3x1. Afterwards, a *Batch Normalization* layers is used and a *2D Convolution* with kernel 10x1 and 128 filters. The convolution layers are finished with a sequence of *Batch Normalization*, *Dropout* with probability 0.3 and *2D Convolution* with 256 filters and kernel 1x6. Before *fully connected* layers are used, *Batch Normalization* is applied followed by a *Global Max Pooling 2D*. Finally, the network ends in *fully connected* layer with 128 neurons, *Dropout* with probability 0.5 and the classification layer with 9 neurons. The architecture can be seen in figure 5.3.

Hyper-parameter tuning results into learning rate having a starting value equal to 0.0004, while callbacks were used, with epoch number 8 and learning rate decay at 2 epochs, to a percentage of 0.92 of its previous value. Batch size is 256 samples, epoch number is 50 and as optimization algorithm Adabound with amsbound was chosen, with learning rate for SGD parts equal to 0.1.

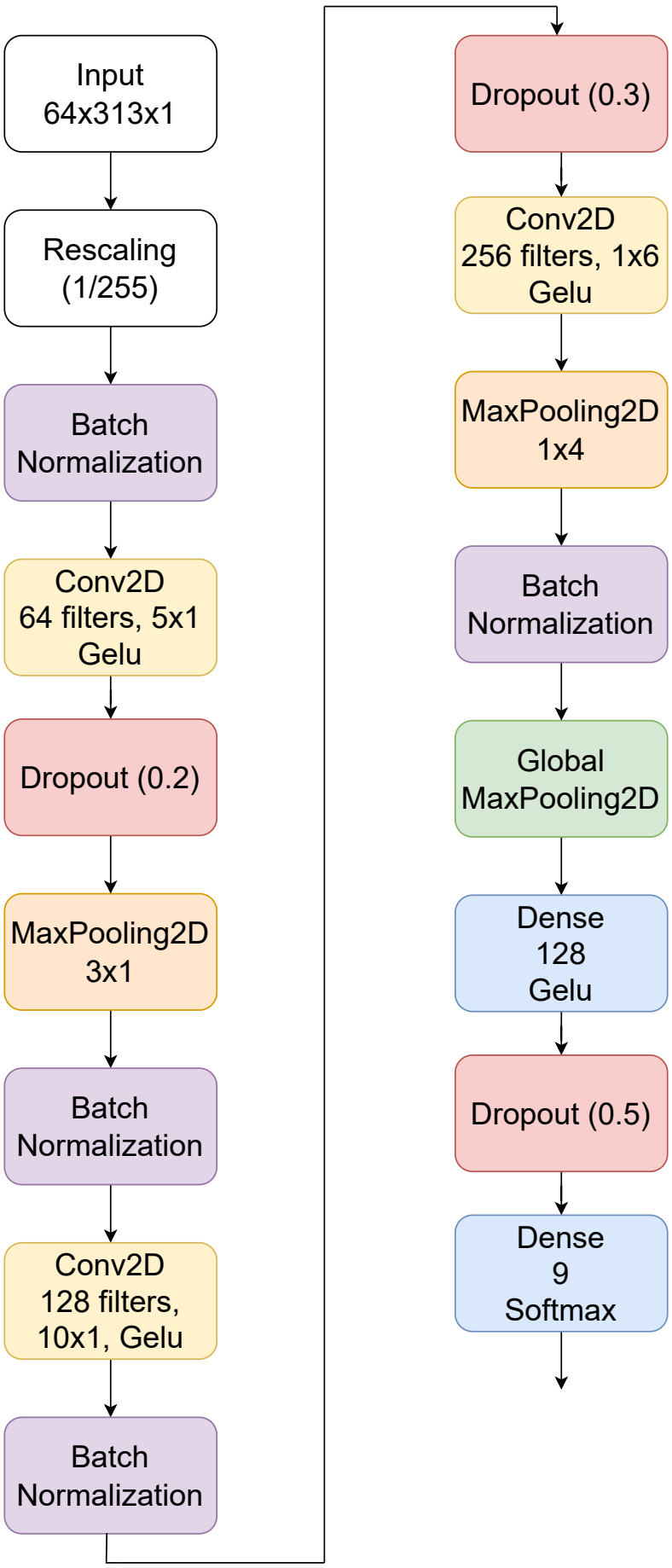


Figure 5.3: 2D Convolutional Network using non symmetrical kernels.

5.4 Two CNNs in parallel

Training two networks in parallel is based in the combined use of features detected by each network, to increase the information extracted and to classify with more accuracy. However, computational resources and training time are increased significantly. This parallel model consists of two parts, where the one part is the 1D Convolutional Network mentioned before and the second one is the 2D Convolutional model with the square kernels. To make a prediction, the average of each network is used. The network can be seen in figure 5.4.

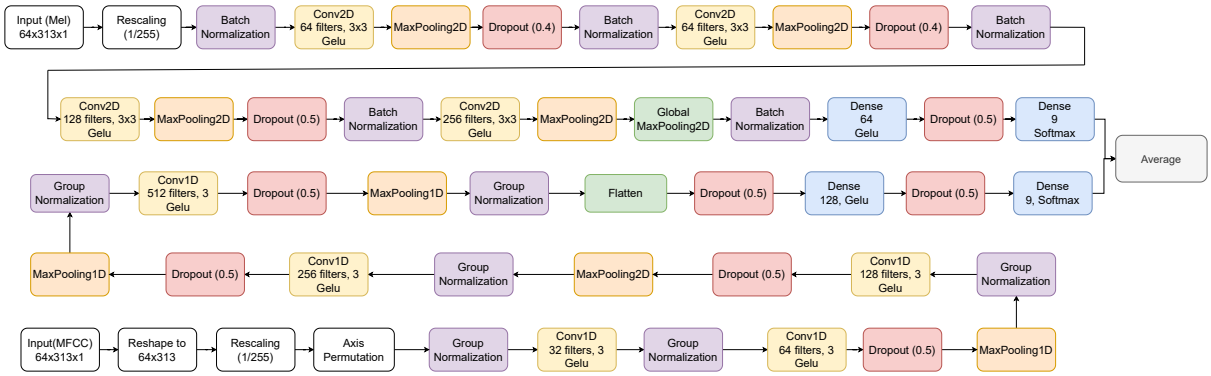


Figure 5.4: Parallel netwrok structure.

Training variables were set to be the same as previous model, using Adabound with amsgrad as optimization algorithm and learning rate decay with initial value $\eta = 0.0004$, then decreasing depending on validation loss. More specifically, if for 6 epochs the validation loss is not reduced, training stops and if it is for 2 epochs, learning rate decreases to 92% of its previous value. Lastly, batch size is equal to 256 samples and epochs are set to 50.

Despite these things, this network did not achieve the same results as the rest of the networks, so it will not be discussed further.

5.5 Ensemble of two convolutional models

It is similar to parallel networks, but in this case each model is trained separately and then the trained networks are combined to form the final model, during evaluation. Thus, each model is not connected to some disadvantages that the other models may have, but subsequently the training time is increased, as training occurs for each model separately. The proposed architecture can be seen in figure 5.5.

The models chosen for the ensemble are the two 2D Convolution models previously analysed. Thus, the models with the largest F1-scores were selected and their output was combined to use the average value of their predictions. Adabound with amsbound was selected to be the optimization algorithm with learning rate $\eta = 0.0004$ and batch size equal to 64 samples. Other combinations were tried from the other models that were created, but in the end the one that can be seen in the image below was the most successful.

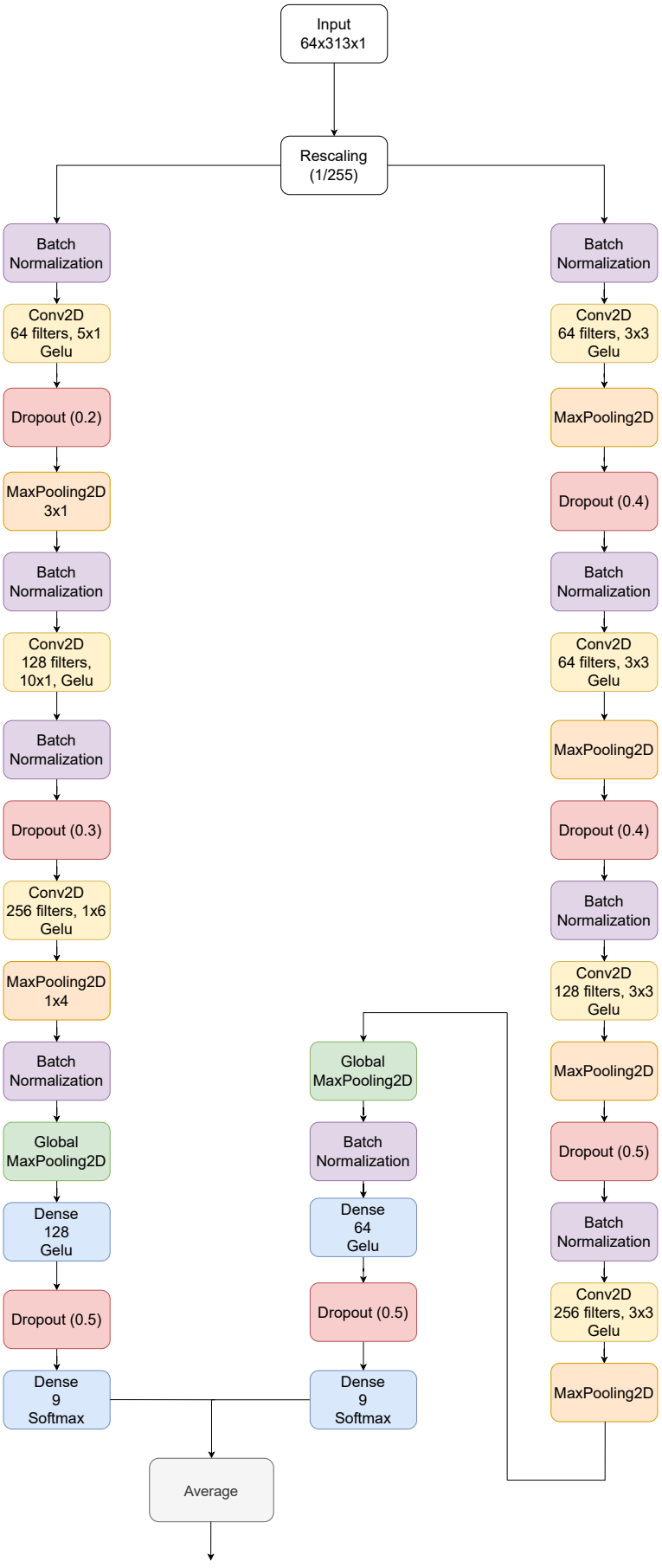


Figure 5.5: Ensemble network architecture.

5.6 Vision Transformers

After the transformers' success (and more specifically the Vision Transformers') in classification problems, their use in the topic of this thesis was tried. As of the moment of writing, the most commonly used network is the Audio Spectrogram Transformer (AST) [14]. Thus, a similar architecture was created, having as inputs the spectrograms and using the transformer mechanisms to achieve the best possible classification. If h and w the height and the width of an image accordingly, P the size of the patches, then following the original Vision Transformer paper:

$$h * w \,\%\, P == 0 \tag{5.2}$$

Therefore, images are expanded or shrinked to abide by this rule. Furthermore, the dimensionality of the linear projection is set to 128, the number of heads for multi-head attention to 4 and the patches number to 16. Thus, patches with dimensions 16×16 will be produced. Then, after rescaling layer, a *Batch Normalization* layer is placed, followed by the creation of patches, according to *Shifted Patch Tokenization*. Afterwards, encoding and linear projection is applied, followed by a recurrence of the next structural element:

- *Layer Normalization*
- *Multi-head Attention* layer, according to *Learning Scaling Attention*, while here is embedded and *Dropout* with probability 0.5.
- Skip Connection) between the output of the *Encoding* layer and the *Multi-head Attention* layer
- *Layer Normalization*
- Skip Connections between the output of the *Encoding* layer and the previous *Layer Normalization*

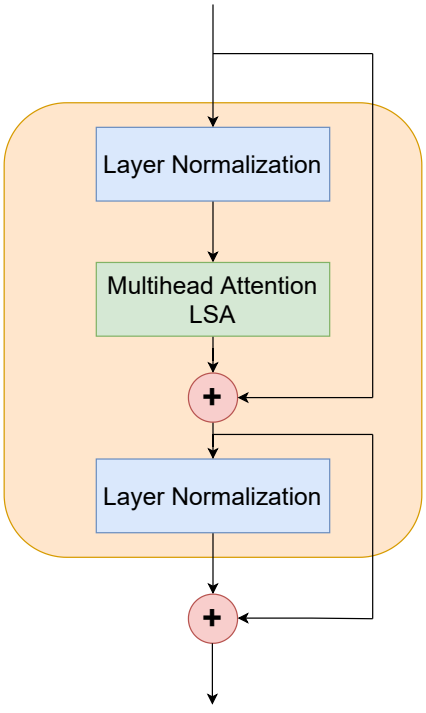


Figure 5.6: Transformer block used.

The structural block (or element) described, in the original ViT model contains fully connected layers which were removed, as their use increased the overfitting effect and incremented significantly the trainable parameters of the models. This element is repeated 12 times, followed by *Layer Normalization*, *Flatten & Dropout* layers in this order, with *Dropout* probability set to 0.5. Then, a *Batch Normalization* layer is placed, followed by a fully connected layer of 64 units, a *Dropout* with probability 0.5 and finally the classification layer of 9 units.

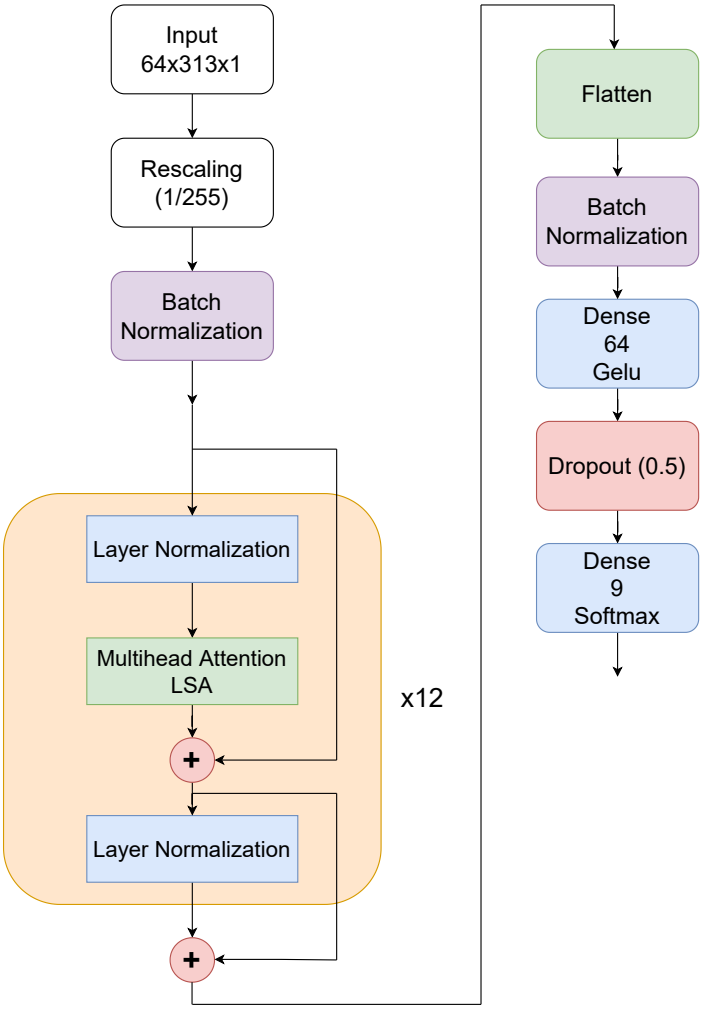


Figure 5.7: Vision Transformer Network.

The optimization algorithm used in this case is Nadam, with learning rate $\eta = 0.0004$, batch size equal to 256 and epoch number set to 50. Finally, if there is no improvement in validation loss for 6 epochs training is stopped, while if this happens for 2 epochs, learning rate is reduced to 93% of its starting value.

Chapter 6

Experimental Results

In this section, the best two networks will be shortly presented, as those were defined in DCASE 2018 competition, while in the end the results of this thesis will be compared with those of the competition.

This subtask of DCASE 2018 competition is about the classification of every day activities in a household, such as cooking. The given samples are pieces of multi-channel recordings, which are recorded from multiple microphone arrays in different locations. This means that spatial properties can be utilized to be used as inputs in a classification problem.

The dataset comprises of audio recordings recorded from 4 microphone arrays in different positions. The dataset contains audio from 7 microphone placements, each with an array of 4 microphones, available, with 3 unknown microphone arrays. The objective of this competition was to find the architecture that could achieve the highest F1-score.

6.1 Competition Results

The first two teams achieved the same macro F1-score, equal to 88.4%. However, their models are vastly different, as it can be deduced from the following subsections.

6.1.1 Inoue et. al. [1]

This model by Inoue et. al. [1] creates new data, by mixing the samples given and then by splitting the audio into pieces and changing their order to create new samples. With this method, the writers try to deal with the dataset imbalance and furthermore no normalization is applied. Lastly, the writers use 40 coeffiecients for the mel-spectrograms they create.

The network has the appearance of a simple CNN with non symmetrical kernels. Going into more details, in the first layers, a convolutional one can be found with kernel 7×1, 64 filters and RELU as activation function followed by, a Batch Normalization layer and by a max-pooling layer with pool size 4×1. Then, Dropout is used with probability equal to 0.2. Finally, two convolution layers are used with kernels 10×1, 1×7 and filters 128 and 256 accordingly , with batch normalization and RELU functions completing the first part of the network. A Global Max Pooling layer with dropout and two fully connected layers, with 128 and 9 neurons end the suggested network [1].

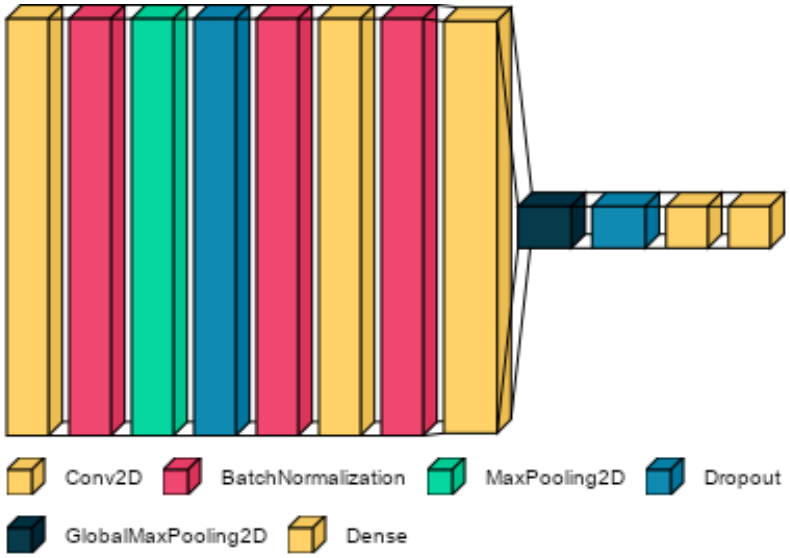


Figure 6.1: The network of Inoue et. al.

For training, Adam algorithm is used, with learning rate $\eta = 0.0001$ and batch size 256. Training is made in 500 epochs and the weights that lead to optimal validation accuracy are selected. Each sample is four-channeled, so the authors decided to use four copies of their network, one for each channel and select the average of their prediction. With this model, they achieved macro F1-score of 88.4%. More details about their results can be seen in figure 6.2.

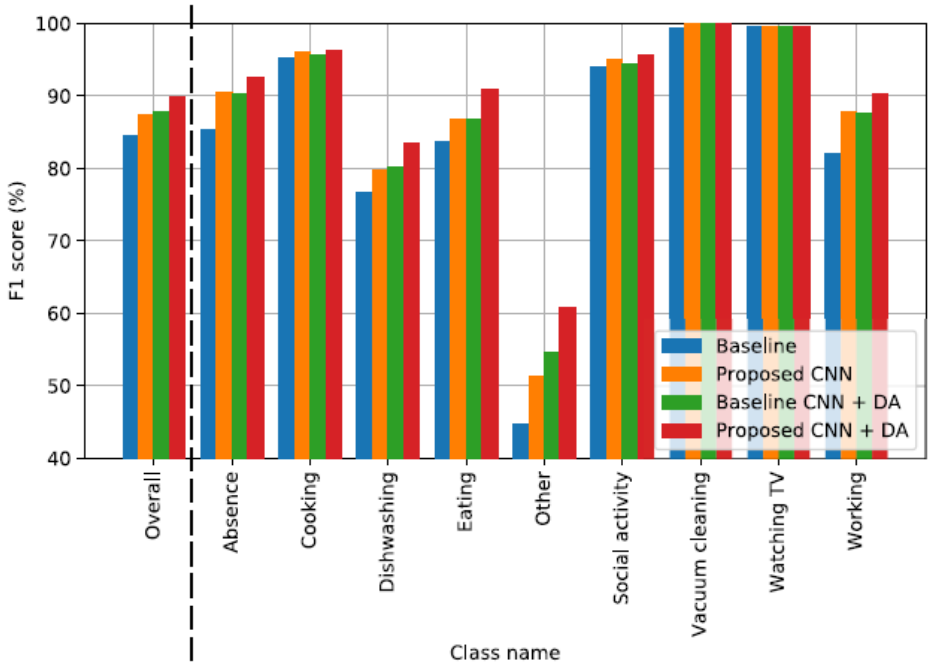


Figure 6.2: Inoue et. al. results for their model per class [1].

Based on the above results, it can be deduced that their solution has difficulty identifying the other class, with significant lower percentages that the other classes.

6.1.2 Tanabe et. al. [2]

Tanabe and his team split their network in two, front-end and back-end. Front-end consists of pre-processing techniques, such as noise reduction, sound source separation and echo removal. Thus, they want to minimize overfitting in the training of the back-end part, which is used for feature extraction and classification. They use MFCC and mel-spectrograms, which are then fed to 1D Convolution architectures and networks based on VGG16. Classification is based on network ensemble.

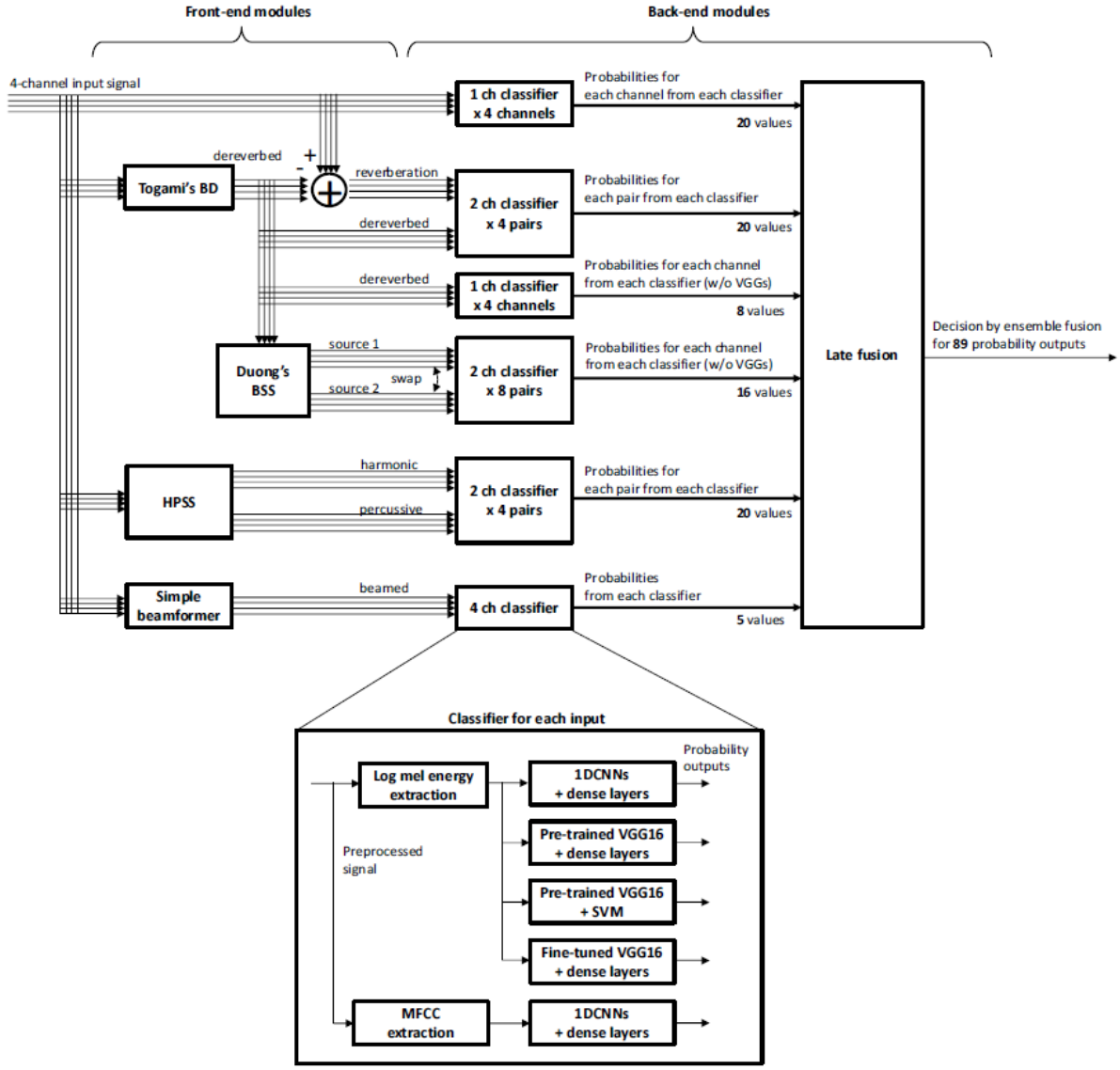


Figure 6.3: Tanabe et. al. network [2].

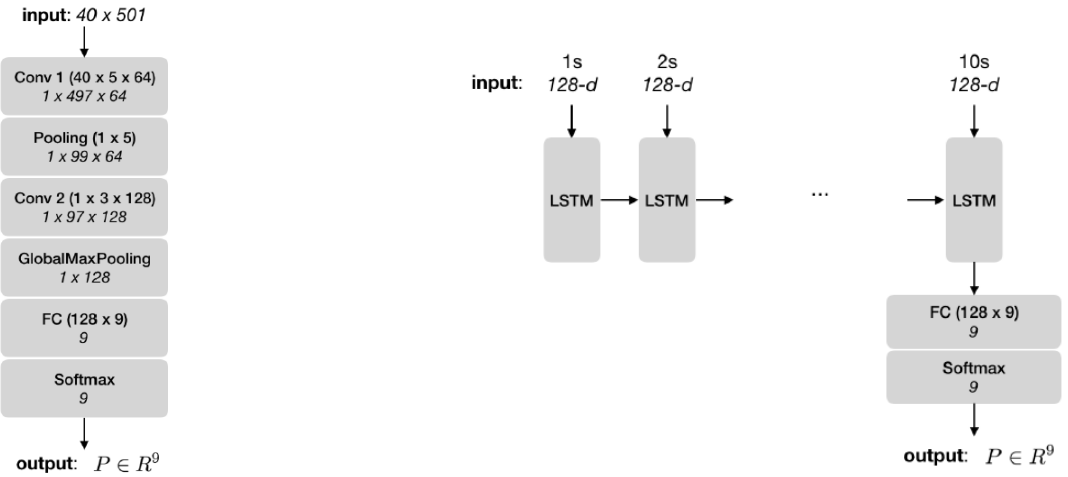
Four different versions are submitted, where ensemble classifies based on average, use of SVM and weighted F1-Score, with details on their results in the following table.

Class	Baseline	Proposed (mean prob.)	Proposed (random forest)	Proposed (SVM)	Proposed (mean F1-weighed prob.)
Absence	85.41	90.63	87.91	87.36	90.49
Cooking	95.14	96.28	96.54	96.58	96.37
Dishwashing	76.73	83.78	86.43	86.82	84.54
Eating	83.64	92.92	93.93	94.40	93.15
Other	44.76	60.29	62.29	64.76	61.01
Social activity	93.92	96.23	96.26	95.61	95.52
Vacuum cleaning	99.31	100.00	100.00	100.00	100.00
Watching TV	99.59	99.59	99.70	99.54	99.38
Working	82.03	88.06	87.34	87.26	88.07
Average	84.50	89.75	90.04	90.26	89.84

Figure 6.4: Tanabe et. al. results.

6.1.3 Liu et. al. [3]

Finally,Liu's team [3], ranked in the third place, was based on a mixture of MFCC, mel-spectrograms and features extracted by VGGish. Using these, they train a CNN using mel-spectrograms and MFCC, while with VGGish extracted features they train an LSTM network. Classification is made using the average of the predictions from the three networks. The authors believe that the average of the four channels is the best way to extract the audio features, thus each network is trained alone and only during evaluation they use their ensemble.



(a) The CNN Network

(b) The LSTM Network

Figure 6.5: Liu et. al. model.

According to their results, the most commonly miss-classified classes are other, abscence and working, with the other samples being classified either as absence or as working. In contrast, the most successful classifications are made for the samples of vacuum cleaning and watching TV, resulting in a macro F1-Score of 87.5%.

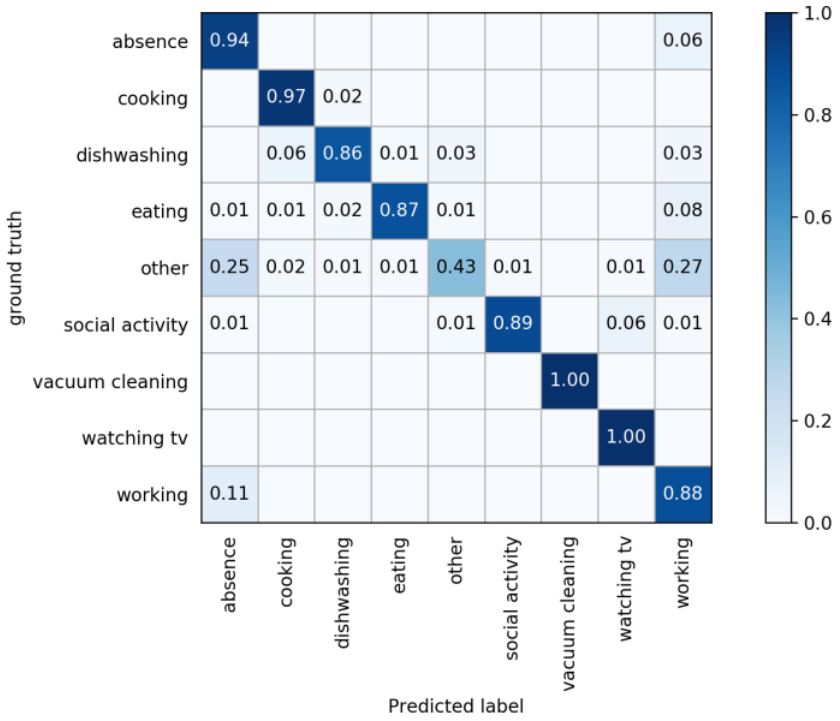


Figure 6.6: Liu et. al. results in confusion matrix format [3].

6.2 Thesis Results

The results described are a product of ten repeated training processes, for each model separately, in order to examine the stability of each model's accuracy. Thus, in the following subsections a deeper analysis is made on each model's results, with the best result being the one with highest F1-score. The Average line for each table is the result of averaging the results of the ten training phases.

6.2.1 1D Convolution

Metric	Training Accuracy	Validation Accuracy	Test Accuracy	F1-Score
Average	0.93122	0.94392	0.90923	0.85945
Best F1 Score	0.9321	0.9444	0.916	0.8665

Table 6.1: A table summarizing the results of the 1D Convolution network

In the following confusion matrix, the predictions of the model are displayed. In the main diagonal the correct predictions can be seem with the rest being the wrong predictions for each class.

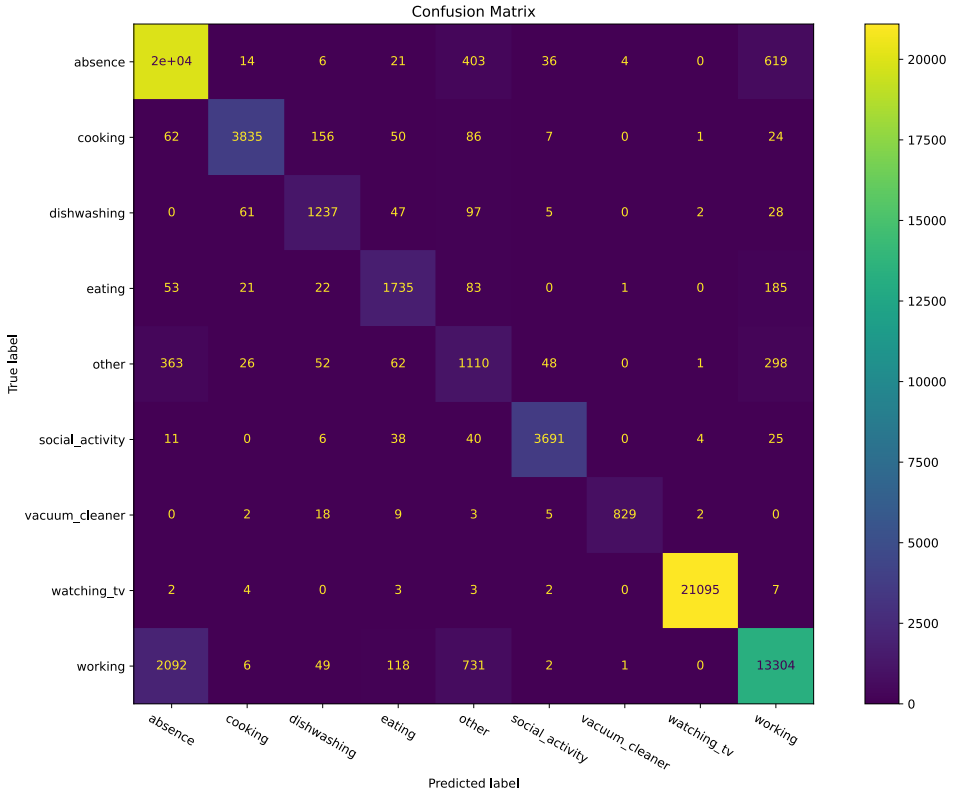


Figure 6.7: Confusion matrix of the 1D Convolution network

Generally, it seems that this model avoided overfitting, since the numbers for training, validation and testing accuracy are similar, with testing accuracy being smaller, as expected. This is probably due to the relatively high probability used in the Dropout layers. Despite this, most miss-classifications happen for the samples belonging to other, absence and working classes. This is due to the fact that these three classes describe repeating sounds, especially absence and working samples. Also, samples belonging in other class, as it is already known, can contain any sound that was recorded but is not relevant with any other class, such as the transition between two activities. Therefore, it is reasonable that the least successfully classified class is other. Also, classes with a specific feature, such as vacuum cleaner, are identified correctly with high percentages, even though they have less samples.

6.2.2 2D Convolution

Symmetrical Kernels

Metric	Training Accuracy	Validation Accuracy	Test Accuracy	F1-Score
Average	0.9416	0.9632	0.9136	0.8642
Best F1 Score	0.9401	0.962	0.9183	0.8706

Table 6.2: A table summarizing the results of the 2D Convolution network with symmetrical kernels

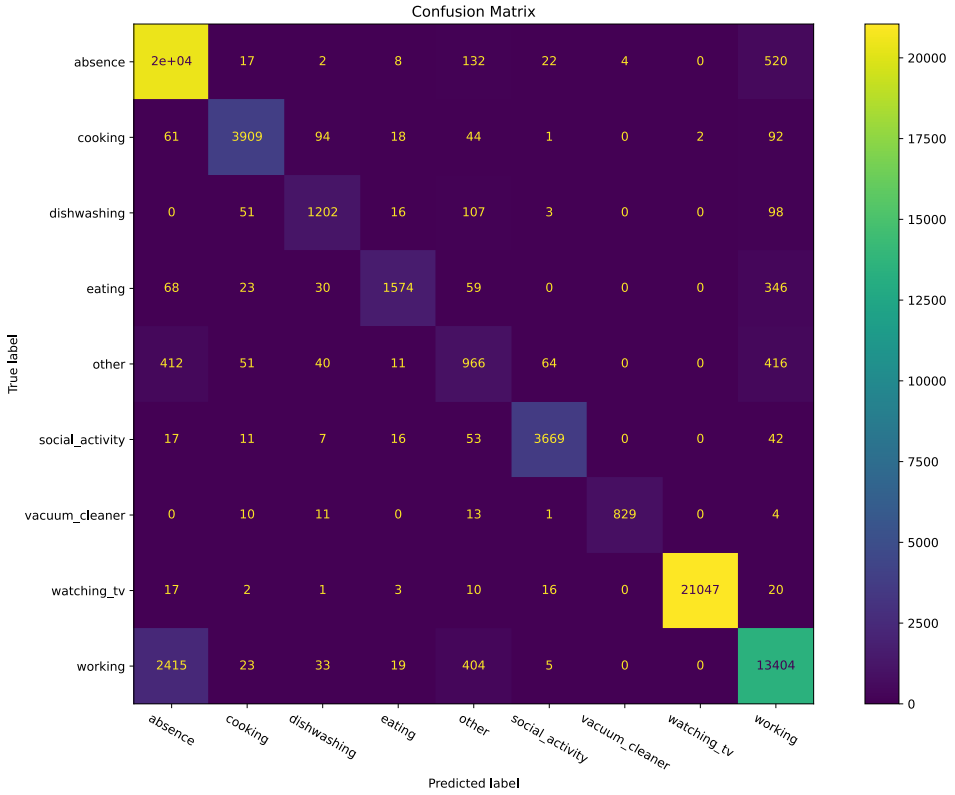


Figure 6.8: Confusion matrix of the 2D Convolution network using symmetrical kernels.

Based on the results, this model is a model void of overfitting, with the classifications following the pattern of the previous model, as in a lot of wrong predictions for the samples belonging to absence, other and working class. Despite this, this model outperforms the previous one, partly due to the fact that information in mel-spectrograms is more helpful.

Non symmetrical kernels

Metric	Training Accuracy	Validation Accuracy	Test Accuracy	F1-Score
Average	0.912	0.94	0.8912	0.8478
Best F1 Score	0.9304	0.9515	0.9008	0.8545

Table 6.3: A table summarizing the results of the 2D Convolution network with non-symmetrical kernels.

In this case, smaller percentages are achieved when compared to the two previous models, even though no overfitting seems to be in effect here. As such, apart from the aforementioned problems of the classes absence, other and working, the low efficiency of this model may be due to the use of non-symmetrical kernels, missing a high amount of information.

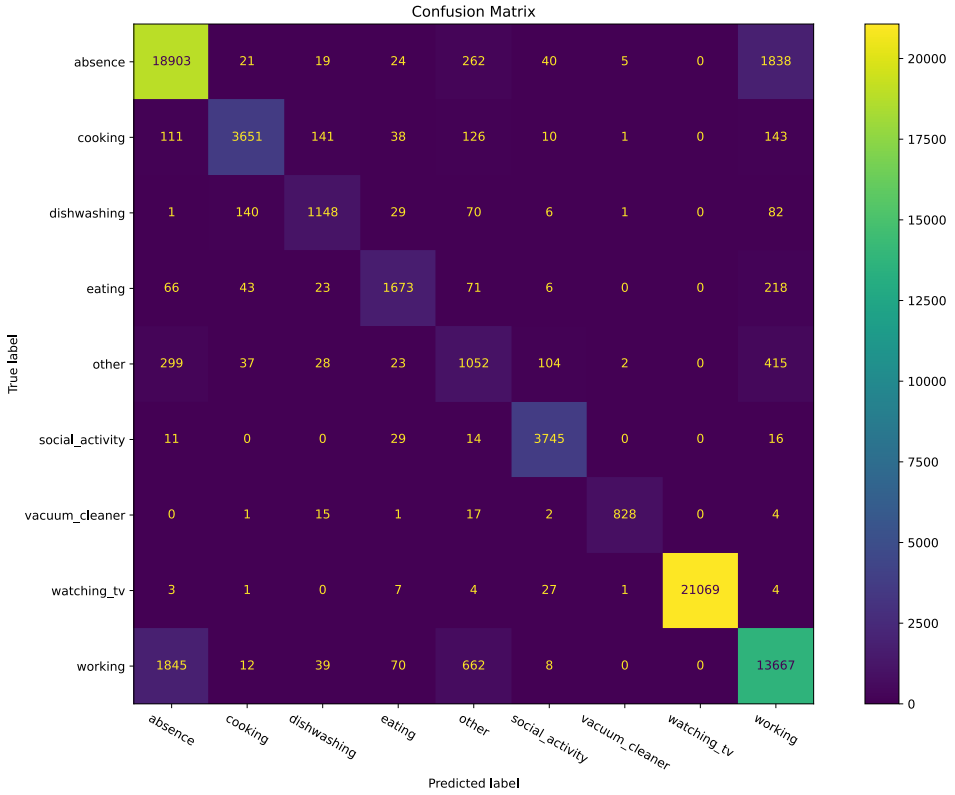


Figure 6.9: Confusion matrix of the 2D Convolution network using non-symmetrical kernels.

6.2.3 Model Ensemble

In the case of model ensemble the average can not be computed, as already trained models are used. Therefore, by choosing the best trained models, the result is the following:

Training Accuracy	Validation Accuracy	Test Accuracy	F1-Score
-	-	0.9271	**0.8871**

Table 6.4: Ensemble network results.

Based on the results, it can be deduced that the ensemble network is the best solution, confirming the use of similar concepts in the contest. The similarity problem between absence, other and working remains still, as the wrong classifications are largely attributed to samples from these classes.

Visualizing the above results using confusion matrix:

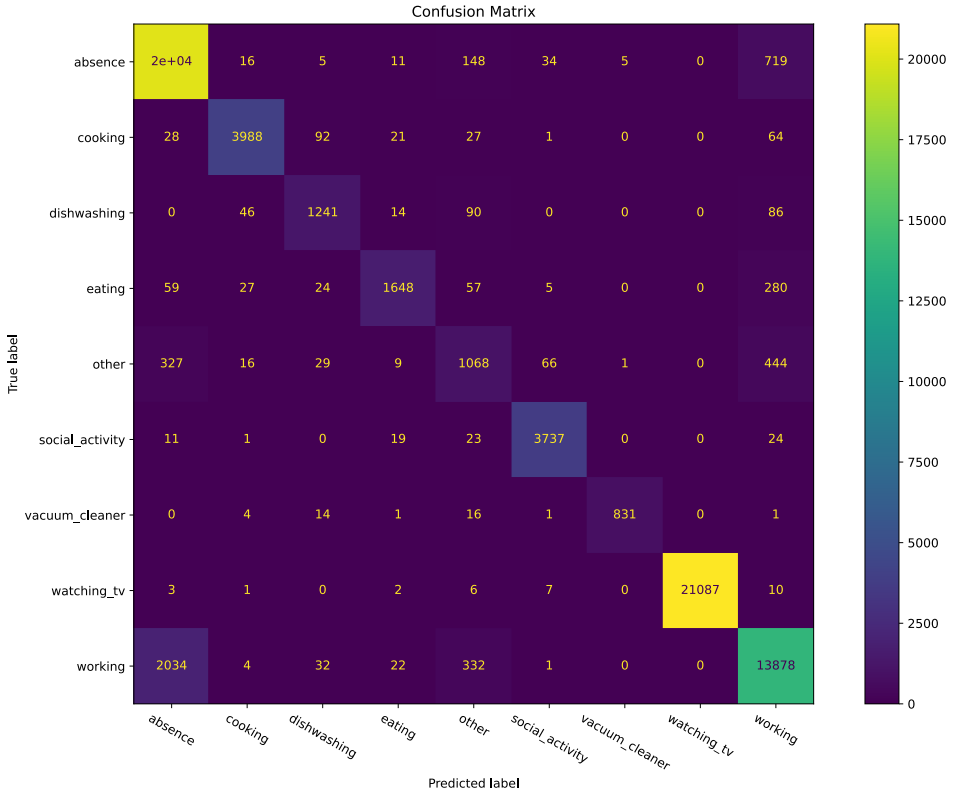


Figure 6.10: Confusion matrix of the ensemble model

6.2.4 Vision Transformers

In the following table, the average and the best scores can be seen, stemming from the ten distinct training phases for this model.

Metric	Training Accuracy	Validation Accuracy	Test Accuracy	F1-Score
Average	0.9670	0.9676	0.8627	0.82074
Best F1 Score	0.9549	0.9619	0.8729	0.8298

Table 6.5: Table containing the results of the Vision Transformer model.

Unfortunately, the disadvantages of vision transformers can be confirmed in this implementation as well. Specifically, overfitting occurs, because the models achieve high accuracy percentages in both training and validation sets, but fail to reproduce this accuracy in the test set, as significantly lower accuracy is achieved. This failure is apparent and in the F1-Score, with low values as well. Thus, even though several generalization techniques are used and new data are created, these attempts are not enough to improve the generalization capabilities of the trained models.

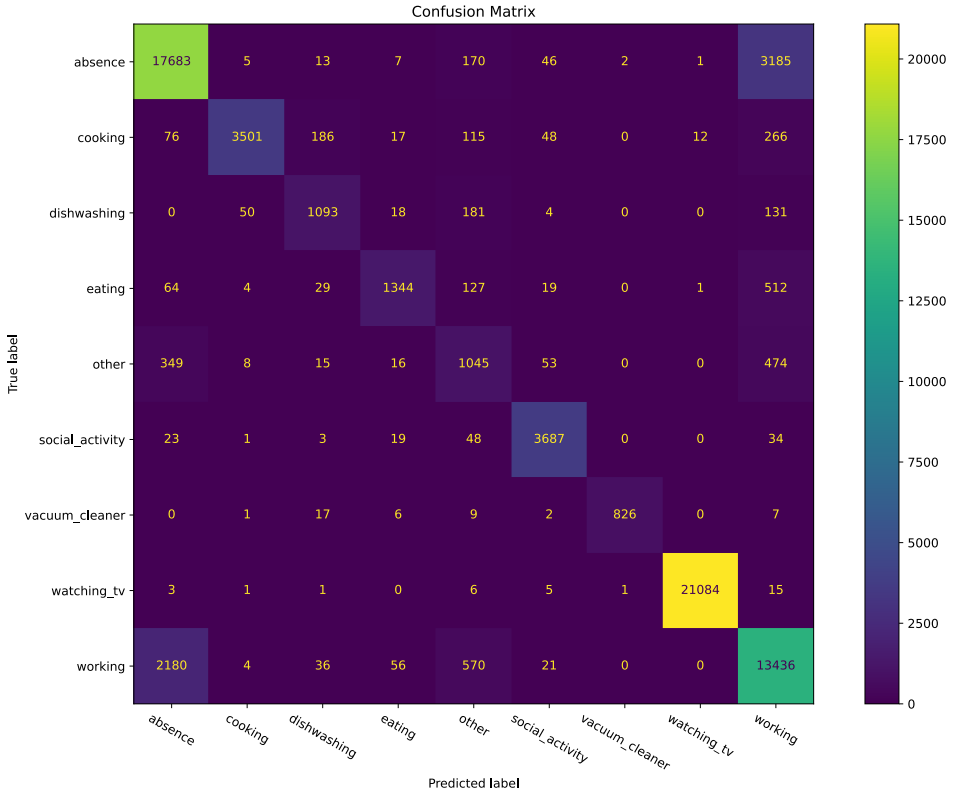


Figure 6.11: Confusion matrix for Vision Transformer network.

6.3 Results Comparison

In the next table the best results from each model can be seen, as those were expanded upon in the previous sections.

Architecture	Train Accuracy	Val Accuracy	Test Accuracy	F1-Score
1D Convolution	0.9321	0.9444	0.916	0.8665
2D Conv, symmetrical kernels	0.9401	0.962	0.9183	0.8706
2D Conv, non symmetrical kernels	0.9304	0.9515	0.9008	0.8545
Vision Transformer	0.9549	0.9619	0.8729	0.8298
Ensemble	-	-	0.9271	**0.8871**

Table 6.6: Table comparing the results from each model

In the following box plot the stability of each model can be displayed, with the *orange* line denoting the average.

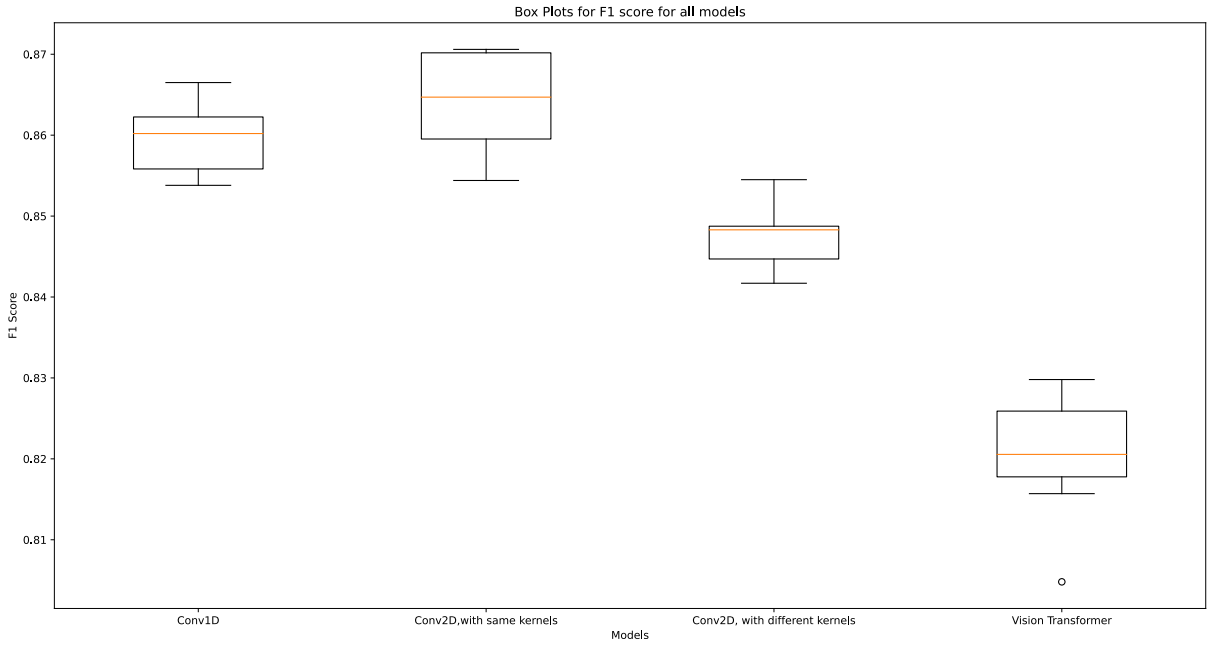


Figure 6.12: F1 Score for each model in box plot format.

Based on the box plot, more consistent performance can be found in the 2D Convolution model with symmetrical kernels. Nevertheless, the ensemble network achieves the best results, as it would appear that the combination of the two 2D Convolution models lead to better generalization ability. It is possible this logic was followed by the winning submissions of the competition, as those networks are model ensemble as well. It is worth noting, that exluding the Vision Transformer model, no other network seems to be affected by overfitting. Also, most of these networks have relatively less trainable parameters, so their use in real situations could be possible, as these models do not require a lot of computational resources.

Despite this, the weaknesses of this dataset could not be overcome with any model from this thesis. Therefore, classes absence, other and working, due to their samples' similarity undermine the general perfomance of the networks. Sure enough, these are the classes that troubled the contestants as well.

Chapter 7

Summary and Conclusion

In conclusion, the object of this thesis was indoor sound classification, such as an audio of an electric vacuum, the sound of TV and others. Initially, a synopsis was made into general pre-processing techniques and into features extracted from audio data. Afterwards, the backbone of neural networks and machine learning was viewed and analyzed.

Having as foundation the varying and multilayered research already conducted for this problem, MFCC features and mel-spectrograms were used, them being the most common and popular choices of data, rather than using the original audio samples. As such, usage of simple architectures initially was used (one and two dimensional convolution) with commonly used values for their hyperparameters, such as choosing optimization algorithm and proper activation function. Then, more complex architectures were implemented (parallel networks, ensemble and transformers), having in mind to use newer ideas in the machine learning domain, while also bearing in mind the particularities of the data used.

Afterwards, the dataset used in this thesis was overviewed and analysis was made into the architectures created for this thesis. After countless tries, the MFCC/Mel coefficient number was set to 64 as a compromise between the low accuracy of 32 and the high computational complexity of 128. A small synopsis followed of the winning networks in the original contest and a comparison was made between those results and this theses' results.

In summation, during this thesis a variaty of networks architectures and parameters was used in order to achieve optimal results. From all the networks created, the ensemble network yielded optimal results, with F1 Score equal to **0.8871**. As such, it is apparent the reason most architectures in the DCASE2018 competition were also ensemble networks.

My hope is that this thesis can be a guide for future works on this topic of interest and to be the starting point in using Vision Transformers in audio problems, because as of the time this is written, research is limited. Finally, what is worth to further research is the effect on the end result the usage of different parameters when creating mel-spectrogram has, for example the window size.

Bibliography

[1] Tadanobu Inoue, Phongtharin Vinayavekhin, Shiqiang Wang, David Wood, Nancy Greco, and Ryuki Tachibana. Domestic activities classification based on CNN using shuffling and mixing data augmentation. Technical report, DCASE2018 Challenge, September 2018.

[2] Ryo Tanabe, Takashi Endo, Yuki Nikaido, Takeshi Ichige, Phong Nguyen, Yohei Kawaguchi, and Koichi Hamada. Multichannel acoustic scene classification by blind dereverberation, blind source separation, data augmentation, and model ensembling. Technical report, DCASE2018 Challenge, September 2018.

[3] Huaping Liu, Feng Wang, Xinzhu Liu, and Di Guo. An ensemble system for domestic activity recognition. Technical report, DCASE2018 Challenge, September 2018.

[4] Minhyuk Jung & Seokho Chi. Human activity classification based on sound recognition and residual convolutional neural network. *Automation in Construction*, 114:103177, 2020.

[5] Xueyi Wang, Estefanía Talavera, Dimka Karastoyanova, and George Azzopardi. Fall detection and recognition from egocentric visual data: A case study. In *Pattern Recognition. ICPR International Workshops and Challenges: Virtual Event, January 10–15, 2021, Proceedings, Part I*, page 431–443, Berlin, Heidelberg, 2021. Springer-Verlag.

[6] Panagiotis Giannoulis, Gerasimos Potamianos, and Petros Maragos. Overlapped sound event classification via multi-channel sound separation network. In *2021 29th European Signal Processing Conference (EUSIPCO)*, pages 571–575, 2021.

[7] Pattana Intani and Teerapong Orachon. Crime warning system using image and sound processing. *2013 13th International Conference on Control, Automation and Systems (ICCAS 2013)*, pages 1751–1753, 2013.

[8] Daniele Barchiesi, Dimitrios Giannoulis, Dan Stowell, and Mark D. Plumbley. Acoustic scene classification: Classifying environments from the sounds they produce. *IEEE Signal Processing Magazine*, 32(3):16–34, 2015.

[9] Tuomas Virtanen, Mark D. Plumbley, and Dan Ellis, editors. *Computational Analysis of Sound Scenes and Events*. Springer International Publishing, 2018.

[10] Vasileios Bountourakis, Lazaros Vrysis, and George Papanikolaou. Machine learning algorithms for environmental sound recognition: Towards soundscape semantics. In *AM '15*, 2015.

[11] Juncheng Billy Li, Wei Dai, Florian Metze, Shuhui Qu, and Samarjit Das. A comparison of deep learning methods for environmental sound detection. *2017 IEEE*

International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 126–130, 2017.

[12] Shawn Hershey, Sourish Chaudhuri, Daniel P. W. Ellis, Jort F. Gemmeke, Aren Jansen, R. Channing Moore, Manoj Plakal, Devin Platt, Rif A. Saurous, Bryan Seybold, Malcolm Slaney, Ron J. Weiss, and Kevin Wilson. Cnn architectures for large-scale audio classification, 2016.

[13] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.

[14] Yuan Gong, Yu-An Chung, and James Glass. Ast: Audio spectrogram transformer, 2021.

[15] B.D. Van Veen and K.M. Buckley. Beamforming: a versatile approach to spatial filtering. *IEEE ASSP Magazine*, 5(2):4–24, 1988.

[16] Roger N. Shepard. Circularity in judgments of relative pitch. *The Journal of the Acoustical Society of America*, 36(12):2346–2353, 1964.

[17] Yang Shao, Zhaozhang Jin, Deliang Wang, and Soundararajan Srinivasan. An auditory-based feature for robust speech recognition. *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4625–4628, 2009.

[18] Mark French and Rod Handy. Spectrograms: Turning signals into pictures. *Journal of Engineering Technology*, 24:32–35, 03 2007.

[19] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366, 1980.

[20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, SciPy 1.0 Contributors Reddy, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[21] Md. Sahidullah and Goutam Saha. Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition. *Speech Communication*, 54(4):543–565, 2012.

[22] Kevin Brunet, Karim Taam, Estelle Cherrier, Ndiaga Faye, and Christophe Rosenberger. Speaker recognition for mobile user authentication: An android solution. *8ème Conférence sur la Sécurité des Architectures Réseaux et Systèmes d'Information (SAR SSI)*, 09 2013.

[23] Frederic B. Fitch. Warren s. mcculloch and walter pitts. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. *Journal of Symbolic Logic*, 9(2):49–50, 1944.

[24] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[25] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[26] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 09 2017.

[27] Robert Coop. *Mitigation of Catastrophic Interference in Neural Networks and Ensembles using a Fixed Expansion Layer*. PhD thesis, 01 2013.

[28] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.

[29] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 12 1989.

[30] Viet Tra, Jaeyoung Kim, Sheraz Khan, and Jongmyon Kim. Bearing fault diagnosis under variable speed using convolutional neural networks and the stochastic diagonal levenberg-marquardt algorithm. *Sensors*, 17:2834, 12 2017.

[31] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[32] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.

[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[35] L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications*. CRC Press, Inc., USA, 1st edition, 1999.

[36] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[39] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[40] Seung Hoon Lee, Seunghyun Lee, and Byung Cheol Song. Vision transformer for small-size datasets, 2021.

[41] Hassan Hassan, Abdelazim Negm, Mohamed Zahran, and Oliver Saavedra. Assessment of artificial neural network for bathymetry estimation using high resolution satellite imagery in shallow lakes: Case study el burullus lake. *International Water Technology Journal*, 5, 12 2015.

[42] S. Theodoridis. *Machine Learning: A Bayesian and Optimization Perspective*. 05 2015.

[43] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[44] Haoning Lin, Zhenwei Shi, and Zhengxia Zou. Maritime semantic labeling of optical remote sensing images with multi-scale fully convolutional network. *Remote Sensing*, 9:480, 05 2017.

[45] María García-Ordás, José Benítez-Andrades, Isaías García, Carmen Benavides, and Hector Alaiz Moreton. Detecting respiratory pathologies using convolutional neural networks and variational autoencoders for unbalancing data. *Sensors*, 20, 02 2020.

[46] Ping-Huan Kuo and Chiou-Jye Huang. A green energy application in energy management systems by an artificial intelligence-based solar radiation forecasting model. *Energies*, 11:819, 04 2018.

[47] IBM Cloud Education. Convolutional neural networks. https://www.ibm.com/cloud/learn/convolutional-neural-networks, 2020.

[48] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2016.

[49] Jie Huang, Wengang Zhou, Houqiang Li, and Weiping Li. Sign language recognition using 3d convolutional neural networks. In *2015 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2015.

[50] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2013.

[51] Vardan Papyan David Donoho and Yiqiao Zhong. Analyses of deep learning stanford university course notes. https://stats385.github.io, 2019.

[52] Cook Alexis. Global average pooling layers for object localization. https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/, 2017.

[53] M Hollemans. CNN on the iphone with VGGNet. https://machinethink.net/blog/convolutional-neural-networks-on-the-iphone-with-vggnet/, 2016.

[54] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In José Mira and Francisco Sandoval, editors, *From Natural to Artificial Neural Computation*, pages 195–201, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[55] Junxi Feng, Xiaohai He, Qizhi Teng, Chao Ren, Honggang Chen, and Yang Li. Reconstruction of porous media from extremely limited information using conditional generative adversarial networks. *Physical Review E*, 100, 09 2019.

[56] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[57] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.

[58] Aditi Shenoy. *Feature optimization of contact map predictions based on inter-residue distances and U-Net++ architecture*. PhD thesis, 07 2019.

[59] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2016.

[60] Jie Fu Ritchie Ng. Deep learning wizard, April 2019.

[61] Reuven Y. Rubinstein and Dirk P. Kroese. *The Cross-Entropy Method*. Springer New York, 2004.

[62] Lavanya Shukla. Designing your neural networks. https://www.kdnuggets.com/2019/11/designing-neural-networks.html, 2019.

[63] Yurii Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

[64] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[65] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[66] Claude Lemaréchal. Cauchy and the gradient method, 2012.

[67] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. In *Proceedings of the 7th International Conference on Learning Representations*, New Orleans, Louisiana, May 2019.

[68] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond, 2019.

[69] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2017.

[70] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[71] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[72] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[73] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2016.

[74] Yuxin Wu and Kaiming He. Group normalization. *CoRR*, abs/1803.08494, 2018.

[75] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016.

[76] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 78, New York, NY, USA, 2004. Association for Computing Machinery.

[77] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2016.

[78] David Olson and Dursun Delen. *Advanced Data Mining Techniques*. 01 2008.

[79] Arda Aras. Explaining what learned models predict: In which cases can we trust machine learning models and when is caution required ?, 03 2020.

[80] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[81] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, 2015.

[82] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[83] François Chollet et al. Keras. https://keras.io, 2015.

[84] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[85] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[86] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[87] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[88] Ekaba Bisong. *Google Colaboratory*, pages 59–64. Apress, Berkeley, CA, 2019.

[89] Gert Dekkers, Steven Lauwereins, Bart Thoen, Mulu Weldegebreal Adhana, Henk Brouckxon, Toon van Waterschoot, Bart Vanrumste, Marian Verhelst, and Peter Karsmakers. The SINS database for detection of daily activities in a home environment using an acoustic sensor network. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop (DCASE2017)*, pages 32–36, November 2017.

[90] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. SpecAugment: A simple data augmentation method for automatic speech recognition. In *Interspeech 2019*. ISCA, sep 2019.

[91] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[92] Md. Shamim Hussain and Mohammad Ariful Haque. Swishnet: A fast convolutional neural network for speech, music and noise classification and segmentation, 2018.