Deep learning-based algorithm for violence detection in audio data

Eduardo Dias Gomes

UMinho |

Universidade do Minho
Escola de Engenharia

Eduardo Dias Gomes

Deep learning-based algorithm for violence detection in audio data

October 2022

Universidade do Minho
Escola de Engenharia

Eduardo Dias Gomes
(A85686)

Deep Learning-based algorithm for violence detection in audio data

MSc Dissertation
[integrated] Master's in Engineering and Management of Information Systems

Dissertation performed under supervision of
Vaibhav Hemantkumar Shah

José Luís Mota Pereira

October 2022

DECLARATION OF INTEGRITY

ABSTRACT

Deep learning-based algorithm for violence detection in audio data

Currently, the mobility services industry lacks a component that guarantees the safety of both its drivers and customers, which is not in line with the constant evolution of the technological sector and the exponential increase of IoT devices capable of capturing data from both the external and internal environment of the vehicle. In this context, Bosch Car Multimedia introduced the RideCare project, which aims to monitor various data in real time from the vehicles of a mobility service provider's fleet, being the root where this dissertation is inserted.

The present work aims to study and develop a Deep Learning-based algorithm capable of detecting violent scenarios only using audio data as its input. In the experimental phase of this project, the CRoss Industry Standard Process for Data Mining (CRISP-DM) methodology was implemented in order to ensure that all project requirements were met in the most efficient way. Special attention was given to the data preparation phase as well as the modeling phase to ensure greater accuracy in terms of classification capability. Subsequently, the models were trained in several test scenarios, composed of several different audio representations, allowing to carry out a comparative analysis in order to extract the most competent model, which is comprised by the MobileNet architecture using the Mel-frequency cepstral coefficients audio feature as input, being able to achieve an accuracy of 81%.

Keywords: Audio Event Classification; Audio Features; CRISP-DM; Deep Learning; Violence Detection

RESUMO

Algoritmo baseado em deep learning para deteção de violência em dados de áduio

Atualmente, a indústria de serviços de mobilidade carece de uma componente que garanta a segurança tanto dos seus condutores como dos seus clientes, o que é algo que não se alinha com a constante evolução do setor tecnológico e com o exponencial aumento de dispositivos *IoT* capazes de capturar dados tanto do meio externo como interno do veículo. Desta forma, a Bosch Car Multimedia introduziu o projeto *RideCare* que tem vista a monitorização de vários dados em tempo real dos veículos de uma frota dum prestador de serviços de mobilidade, sendo a raíz onde esta dissertação se enquadra.

O presente trabalho tem como objetivo estudar e desenvolver um algoritmo baseado em *Deep Learning* capaz de detetar cenários violentos apenas usando dados de áudio como *input.* Na fase experimental deste projeto a metodologia *CRoss Industry Standard Process for Data Mining* (*CRISP-DM*) foi implementada no sentido de garantir que todos os requisitos do projeto fossem cumpridos da forma mais eficiente. Foi dada uma especial atenção à fase de preparação dos dados bem como a fase de modelação para certificar uma maior precisão a nível de capacidade de classificação. Posteriormente, os modelos foram treinados em vários cenários de teste, compostos por várias representações de áudio diferentes, permitindo fazer uma análise comparativa de modo a retirar o modelo mais competente, sendo este constituído pela arquitetura *MobileNet* utilizando como input o *audio feature Mel-frequency cepstral coefficients*, atingindo uma acuidade de 81%.

Palavras chave: *Audio Features*; Classificação de eventos de audio; *CRISP-DM*; *Deep Learning*; Deteção de violência

INDEX

LIST OF ABBREVIATIONS/ACRONYMS

NLP	Natural Language Processing
DL	Deep Learning
ANN	Artificial Neural Network
CNN/ConvNet	Convolutional Neural Network
IoT	Internet of Things
ReLU	Rectified Linear Unit
FC	Fully Connected Layer
ADC	Analog Digital Conversion
FFT	Fast Fourier Transformation
STFT	Short-Time Fourier Transformation
MFCC	Mel Frequency Cepstral Coefficient
SVM	Support Machine Vector
AE	Auto Encoder
RNN	Recurrent Neural Network
WAV	Waveform Audio File Format
SR	Sample rate
CQT	Constant-Q Transformation

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Framework and motivation

Deep learning is a machine learning subset based on Artificial Neural Networks (ANN), meaning they try to simulate the behavior of the human brain, which can perform very well when trained with large quantities of data. These type of algorithms have multiple use cases such as image classification and segmentation, which can be used for analyzing medical images more accurately [1], or even self-driving vehicles [2]. Moreover, these have been applied very successfully on object detection and tracking [3], translating to huge advances on face recognition and identification [4]. Speech and music are also a popular type of data used to train DL models, in forms of acoustic research. There are several studies related to speech generation, and speech recognition which are deeply related to Natural Language Processing (NLP). On the other hand, music research includes, music generation, genre classification and beat tracking.

When it comes to violence detection, these types of algorithms are very dependent on visual data, or a combination of both video and audio. With the constant growth of IoT and devices, the amount of data available rises exponentially with time [5], auditory data included, and these signals overflow with relevant information that can easily be processed and used in a DL system. Although these architectures have been used in the field of environmental sound classification and anomaly detection, the approach taken in this dissertation aims to explore an uncommon method in which only auditory data and its features will be used to fit a Deep Neural Network (DNN) with the purpose of detecting violence, more specifically in-car violence.

This dissertation project will be developed as part of an Academic Internship at Bosch Car Multimedia unit in Portugal. This partnership between Bosch and University of Minho allowed a support combination from my dissertation tutors and the monitoring of my team, which consequently, helps this project to be developed in an efficient and effective way. Moreover, Bosch provides a cluster that allowed model training to be way faster, for a quicker development and comparison of results.

1.2 Objectives and Expected results

Objective definition and interpreting expected results are crucial for the successful development of this dissertation. The main and most important objective for this master dissertation was the development of a DL model that will be able to detect violent activity with the highest accuracy possible, solely based on auditory data. It is important to refer that the dataset used for the development of this architecture was provided by Bosch. It was also mandatory to research and define the best tools that were meant to be used for the practical development of this dissertation, as well as task planning for a more organized and efficient work. The development of technical articles was also expected. Finally, the last defined objective was the porting of the algorithm to a target device.

1.3 Document structure

The present dissertation is divided in six distinct chapters. Firstly, the Chapter 1 (Introduction) is comprised by the Framework and motivation, Objectives and Expected results, the Document structure of this dissertation and the Dissertation work plan. Chapter 2 (State of the art) is the most extensive one given that all the relevant concepts for this master dissertation are described. It starts by explaining the concept behind Deep Learning, followed by a deep dive into Artificial Neural Networks and its variants. Furthermore, Regularization methods were studied, and different audio representations were also described. Data Augmentation is also discussed given that it’s an important step of the Deep Learning pipeline and the nature of the provided dataset. In order to finish this chapter, Common approaches on audio classification are also described, and a study regarding Related work on violence detection using auditory data is conducted.

On Chapter 3 (Methodology, Technologies, and Tools), the methodology that guided the development of this dissertation is also mentioned, this being CRoss Industry Standard Process for Data Mining (CRISP-DM). In addition, this chapter also details the Python libraries and Tools that were used in order to develop the practical aspect of this master dissertation. Chapter 4 (Use case – In-car violence detection) the Deep Learning architectures and the testing scenarios results are documented at the light of the adopted methodology, presenting the business understanding data understanding and preparation, modelling and evaluation.

Lastly, in the sixth and last chapter, a conclusion for this dissertation is presented, summarizing all the work done and its contributions, alongside the project limitations and plans for future work.

1.4 Dissertation work plan

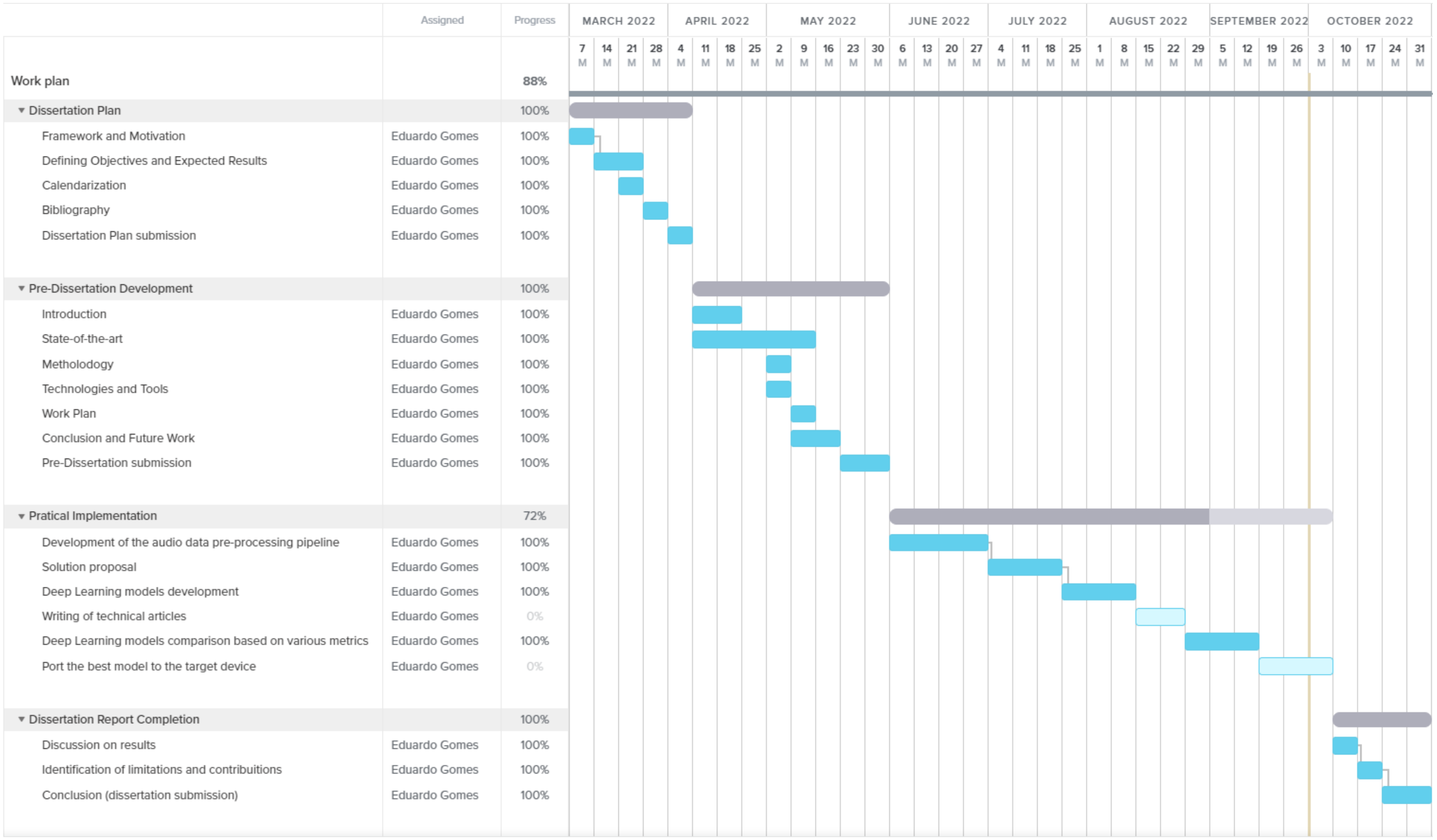
On this sub-section, a project timeline is presented as well as the tasks that were developed throughout this dissertation. The planning phase was a crucial step for this master dissertation since it specifies what tasks and activities that should have been in place, as well as time resources that were allocated to them. The time requirements for each task were assigned based on the project objectives as well as the stipulated dates by the authors' course direction. All the group tasks are represented on the Figure 1 Gantt Diagram and the final date refers to the submission of this dissertation report on October 31st.

The main group of tasks for this dissertation were:

- **Dissertation plan** (March 7th to April 5th): The purpose behind this task was to detail the project subject following the structure: Framework and Motivation, Defining Objectives and Expected Results, Calendarization and Bibliography. This document was submitted on April 5th.
- **Pre-dissertation development** (April 5th to May 31st): The pre-dissertation report aims to take a deeper dive into the contents defined on the previously described document. In this report an introduction to the subject was made as well as the expected results and contributions of this dissertation. A literature review was also conducted as well as the tools and technologies that were going to be used in the implementation phase. A work plan was presented alongside the conclusions and the future work for the practical implementation of the project.
- **Practical Implementation** (May 31st to October 3rd): This task refers to the practical implementation phase of this dissertation. Here is where the metadata and audio preprocessing pipeline were developed in order to apply the necessary transformations to the audio labels and signals respectively. The Deep Learning architectures were also defined in this phase alongside its

training and result comparison based on various metrics. It is important to note that this was developed based on the adopted methodology. It was expected that the writing of the final report was done throughout this phase as well as the development of scientific documentation.

- **Dissertation Report Completion** (October 3rd to October 31st): This final group of tasks were meant to finalize the Dissertation Report. Here, all the results were discussed in the light of the previously defined objectives, as well as the identification of limitations (and possible future work) and what were the contributions of this project. The final sub-task, the Dissertation Submission refers to the submission of this report which is scheduled for October 31st.



	Assigned	Progress
Work plan		88%
▾ Dissertation Plan		100%
Framework and Motivation	Eduardo Gomes	100%
Defining Objectives and Expected Results	Eduardo Gomes	100%
Calendarization	Eduardo Gomes	100%
Bibliography	Eduardo Gomes	100%
Dissertation Plan submission	Eduardo Gomes	100%
▾ Pre-Dissertation Development		100%
Introduction	Eduardo Gomes	100%
State-of-the-art	Eduardo Gomes	100%
Methholodogy	Eduardo Gomes	100%
Technologies and Tools	Eduardo Gomes	100%
Work Plan	Eduardo Gomes	100%
Conclusion and Future Work	Eduardo Gomes	100%
Pre-Dissertation submission	Eduardo Gomes	100%
▾ Pratical Implementation		72%
Development of the audio data pre-processing pipeline	Eduardo Gomes	100%
Solution proposal	Eduardo Gomes	100%
Deep Learning models development	Eduardo Gomes	100%
Writing of technical articles	Eduardo Gomes	0%
Deep Learning models comparison based on various metrics	Eduardo Gomes	100%
Port the best model to the target device	Eduardo Gomes	0%
▾ Dissertation Report Completion		100%
Discussion on results	Eduardo Gomes	100%
Identification of limitations and contribuitions	Eduardo Gomes	100%
Conclusion (dissertation submission)	Eduardo Gomes	100%

Figure 1 - Dissertation activities Gantt Chart

2. STATE OF THE ART

2.1 Deep Learning

Deep learning is a Machine Learning subset which combines powerful learning techniques with knowledge about how the human brain works, statistics and applied mathematics [6]. Although Deep Learning is a technology created in the mid-40s, this technique has a higher computational power and a higher capability on dealing with larger volumes of data than regular Machine Learning approaches, making this concept a big highlight through the years with constant advances. This particular case of ML is able to learn and represent data as an hierarchy of concepts, and each of these concepts are divided into much simpler ones and abstract representations that are a product of more discrete ones [6].

The first DL models were created with the objective of recreating how the human brain learns at a computational level, which originated the concept of Artificial Neural Network, which then evolved to the current designation. These type of networks, also known by Representation Learning, are able to create high level abstractions from the input data, utilizing an arbitrary number of layers for processing [6]. The layers are responsible for the most part of the input processing, learning its features so it can output value for tasks with higher complexity.

As it was previously mentioned, Artificial Neural Networks are computational models based on the processing and learning capability of the human brain, which makes it viable to learn non-linear variable relationships, easily identifying patterns making it a powerful alternative to traditional Machine Learning methods [7]. There are multiple forms of Neural Networks such as Feed Forward Networks, Convolutional Neural Networks, Recurrent Neural Networks and Residual Neural Networks, that will be described in the following sub-sections.

2.2 Artificial Neural Networks

Artificial Neural Networks are a computational learning system that is composed by a network of functions used to understand and translate some sort of data input and output a desired value, usually in another form. In much simpler words, these set of algorithms are deeply inspired in how the human brain operates, they have the ability to recognize patterns, information and relationships in the given data.

The simplest form of an ANN is formed by three components:

1. An input layer – this layer corresponds to the input nodes meaning the information from the "outside world" is provided here for the model to learn and later, output a value based on the learned features. These nodes are also responsive to pass to the next connected layer in a left to right manner.
2. A hidden layer – this component is made of a set of neurons that are meant to perform all the computations on the data received from the input layer. The simplest form of an ANN is made of a single hidden layer but there can be as many as desired although this might have performance implications.
3. An output layer – the concept behind this layer is self-explanatory, it corresponds to the output of the model derived from the calculations performed on the previous layers. The number of nodes in this layer depends on the number of possible classes. For example, in a binary classification problem, only one neuron is needed on this layer, since the output can only be either 1 or 0.

2.2.1 Multi-layer Perceptron

One of the simplest and oldest forms of Artificial Neural Networks is called Multi-layer Perception (MLP). The main characteristic of this model is that all of its layers are fully connected, meaning a neuron has one weighted connection between itself and the neurons of the next layer. For a better understanding of how these types of networks function, a step-by-step approach was conducted, using Figure 2 as an example of a MLP network.

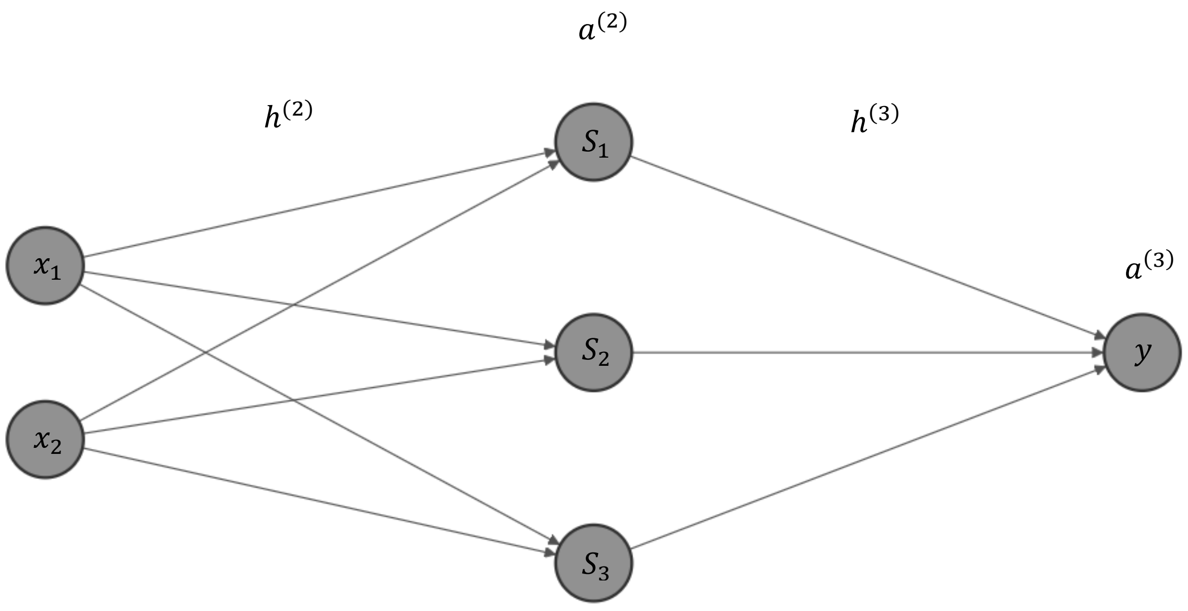


Figure 2 - MLP network example

Firstly, all the inputs are multiplied by their weights. Weights are associated to inputs in order to identify its coefficient, translating into how impactful a particular input will be (which can be a positive or negative reinforcement). After weight assignment a bias variable b is added. This is a constant that is used to help the model fit in the most effective way. In other words, before the information is passed from the input layer to the hidden layer, a matrix multiplication is performed between vector input x with the weight matrix $W^{(1)}$ and later adding the bias variable which returns the net input $h^{(2)} = xW^{(1)} + b$. On the hidden layer, the next step is to apply the activation function to the net input previously calculated $a^{(2)} = f(h^{(2)})$ which returns the activation vector of the second layer. This concept may appear to be abstract, but it will be later explained on sub-section 2.2.3. On the third layer and final layer, the net input is again calculated using $h^{(2)} = a^{(2)}W^{(2)} + b$ and apply the activation function in order to get the predicted output: $y = f(h^{(3)})$

2.2.2 Back propagation

The process described on the sub-section 2.2.1 is known as forward propagation. When training an *NN* the final step of the forward propagation is the evaluation of the predicted output $\hat{y}$ against the expected value y. This is done by using a cost function E, also known as a loss function. This function is set as a parameter when fitting the model and depends on the use case. It is based on this function’s value that the model works its way to adjust its parameters in order to get to a closer value to the real one which is present on the dataset. This is known as back propagation and was created in order to achieve the minimum value of the cost function by adjusting the *NN* weights and biases (in a positive or negative manner) based on the gradients of the cost function [8]. The gradient calculation is done to each weight and bias based on the chain rule, and it is done layer by layer in an iterative backwards way with the objective of avoiding redundant computations.

2.2.3 Activation Functions

As it is described on section 2.2, all the connections have weights associated which are taken into account when processing the information and passing it through the different layers. In order to recognize patterns and relationships on the data received from the outside world, activation functions must be used between layers. These mathematical equations determine if a neuron will be fired or not, and it’s done based on the if the inputs are important for final model prediction. Activation functions introduce non-linearity to the model which is a requirement to learn and recognize complex mappings from data [9], [10]. Let’s say the following classification problem is proposed: classify if a customer will buy a product based on its age, education level and marital status. A simple linear classifier wouldn’t be able to predict very accurately this binary classification problem, simply because the pattern/relationship that defines whether the customer will buy a product or not is not linear.

Furthermore, activation functions are also used to keep the value of the output from a neuron restricted to a certain value that depends on the function used. This reduces the amplitude of the output signal into a finite value. This operation is known as squashing.

There are multiple studies regarding activation functions, not only in performance comparison [10], [11] but also in ways to find new and more viable ones [12]. In order to keep this section short, only four activation functions will be described, these being Rectified Linear Unit, Sigmoid, SoftMax and Hyperbolic Tangent Activation Function.

Rectified Linear Unit (ReLU)

As mentioned in [13], the ReLU is the most popular activation function in the world right now. This function makes a simple calculation that returns the value provided as input or 0 if the input is lower than 0. Because the ReLU function is linear for half the input and nonlinear for the other half, this function is often referred to as a piecewise linear function. This implies that on the layers that use this function not all the neurons will be activated, which translates to a better performance and efficiency. This function can be mathematically defined as $f(x) = \max(0, x)$

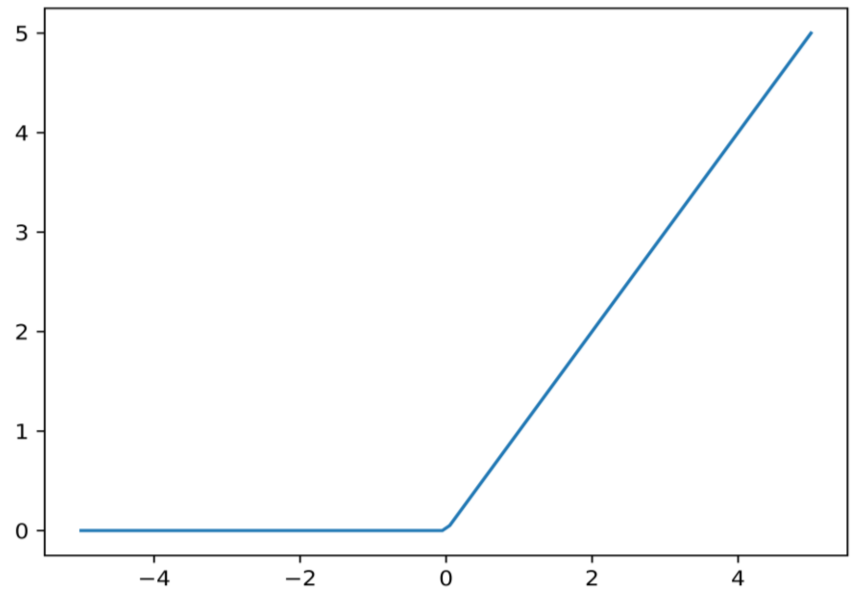


Figure 3 - ReLU activation function plot

Sigmoid Activation Function

The Sigmoid activation function can be defined as $\sigma(x) = (1 + e^{-x})^{-1}$. It takes an input x and squashes it between 0.0 and 1.0 which can be interpreted as a probability for that specific input. This activation function is often used in the last layer for two-class (binary) classification problems.

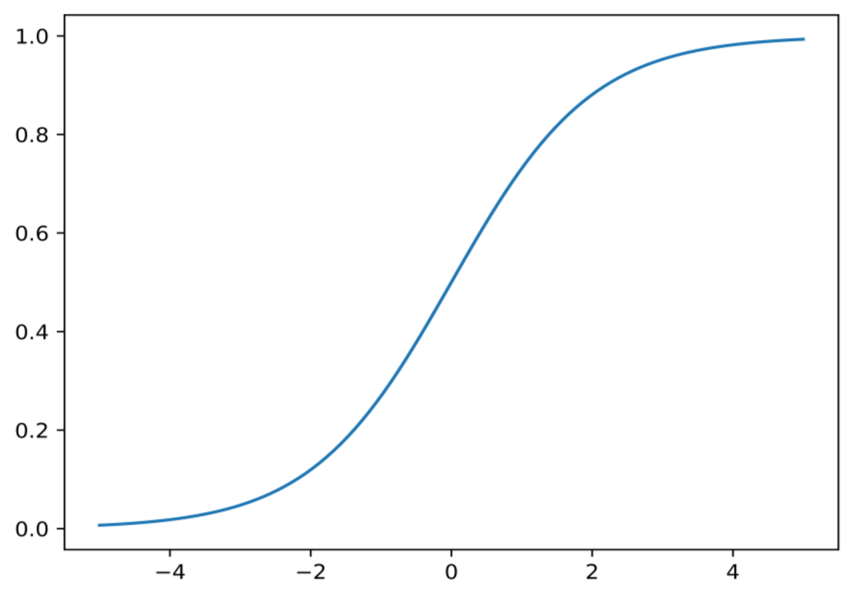


Figure 4 - Sigmoid Activation Function Plot

Hyperbolic Tangent Function (Tanh)

This function is very similar to the previous one, but it is symmetric around the origin of the graph, it's a zero centered activation function. Just as the sigmoid, it also squashes the input values, but into a bigger interval, between -1.0 and 1.0. The advantage the Tanh function has over the Sigmoid function is that the negative inputs will be mapped in a strongly negative way and positive inputs will be mapped strongly positive. This property makes this function perfect for binary classification. It can be represented by $f(x) = 2S(2x) - 1$, S standing for the sigmoid function.

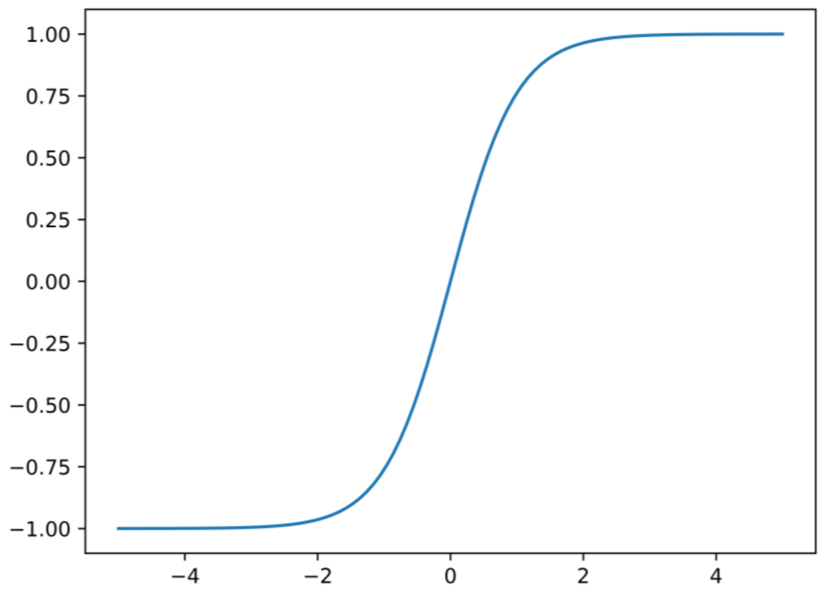


Figure 5 - Tanh Activation function plot

Softmax activation function

The *Softmax* activation function is the generalization of the Sigmoid function, meaning it's a combination of multiple Sigmoid functions [9]. As mentioned earlier, the Sigmoid function returns values between 0 to 1, which can be perceived as the probability for each class, in a binary classification situation. This function, however, can be used for multiclass classification problems, returning a probability for each of the values present on the input vector. It can be mathematically expressed by:

$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$ for i = 1, ..., k. Where k is the number of classes.

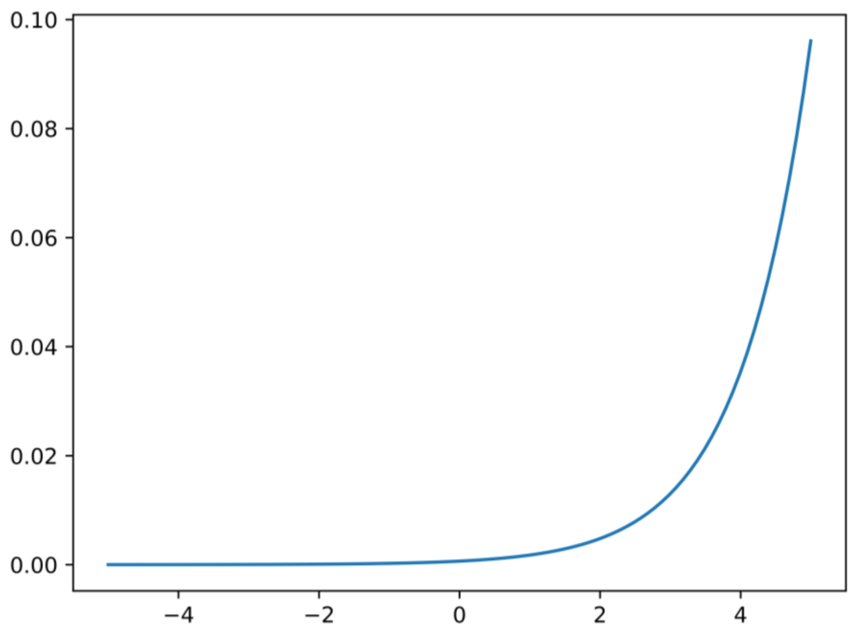


Figure 6 - Sofmax Activation Function Plot

2.3 Convolutional Neural Networks

Convolutional Neural Networks is a Deep Learning model that takes data that can be represented in a grid pattern [14], such as images. It assigns learnable and improvable weights and biases to various features in the data, in order to be able to classify them based on feature learning. This type of architecture is noted for using a mathematical operation, in at least one of its layers, called convolution instead of the previously explained matrix multiplication. This specific characteristic is the reason behind the successful implementations of image recognition while using CNNs [15]. It allows the model to understand the spatial and temporal features of an image through the application of filters.

The simplest version of a *ConvNet* is usually composed by three main layers: Convolutional Layer, Pooling Layer and Fully Connected layer. In this section, all three will be described.

2.3.1 Convolutional Layer

Convolutional layers are the center of attention in CNN architectures. This layer receives a learnable filter that handles the feature detection. This is often referred as a kernel. A kernel is a two-dimensional array of weights that as a parameterized size and when it is applied to an area of a grid like data, such as an image, a matrix multiplication is calculated between the input pixels and the filter which is latter outputted to an array. The next step is shifting this filter by a stride and repeat the same process till the kernel has been applied to the entirety of the grid data. On an image, the stride defines how many pixels the kernel will shift, meaning the lower the stride, the higher overlapping that will occur, translating to a dimensionally larger output [16]. The output of this process is often referred as an activation map or a convolved feature.

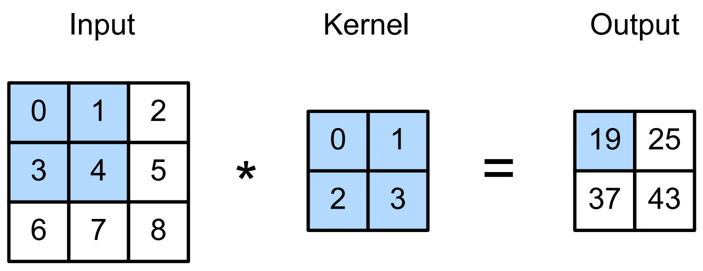


Figure 7 - Example of a kernel applied to a 2D input, source: https://bit.ly/3vVnEUu

The main objective of the convolutional operation is to extract high-level features, such as edges from an input image. The number of convolutional layers is arbitrary, where the closer they are to the input layer, they will detect more low-level features such has color or gradient orientation. On the other end, the closer these layers are to the output, the features will be severely more broken down which means they are responsible for the capture of the high level spatial and temporal dependencies.

As mentioned in section 2.2.1, each neuron in a MLP is connected to the previous ones. If an Artificial Neural Network is trained using images as inputs, it would result on a huge model that could not be trained effectively. On CNNs, each filter of a convolutional layer is connected to a certain part of the input, this is called sparse interaction and can be controlled by adjusting the filter size. Another major difference is that weights on ANNs are independent of each other, in CNNs rather, each filter applies the same weights at each local region of the input image. This is called parameter sharing, it presupposes that if one region filter can compute a certain feature, then there is a chance that it can be useful in another area, meaning it forces each kernel to detect the same feature across the input.

In addition, convolutional layers also reduce or maintain the dimensionality of their outputs compared to the inputs, this operation is called padding. Taking a 6x6x1 image as an example (number of width pixels x number of height pixels x number of channels), if a 3x3x1 kernel is applied and the same padding parameter is used, the output matrix will have a 6x6x1 dimensionality, meaning it ensures that the output has the same size as the input. On the other hand, if valid padding is used, a matrix with the dimensions of 3x3x1 is returned. Lastly, full padding can be used, meaning the output dimensions are increased by adding zeros around the input matrix. After the convolutional operation, an activation function is applied to the output matrix.

2.3.2 Pooling layer

Pooling layers are used to apply further changes to the output of the convolutional operation. These layers are also known as down samplers given that their goal is to reduce spatial size of its inputs culminating in a lower computational load. This procedure is very similar to what happens in the convolutional layer, but instead of using weights, the filter applies an aggregation function to the values within it. There are a few variations of this procedure: Max-pooling, Average Pooling, L1-normalization, Weighted Average Pooling. In this section, Max and Average Pooling will be the focus point given that they show to be the most commonly used in Convolutional Neural Networks.

When Max-Pooling is applied, the filter moves across the input, selects the pixel with highest value and uses it on the output array. This pooling approach is generally the most used on CNN architectures. On the other hand, when the average pooling algorithm is used, as the filter shifts on the input it calculates the average value within the receptive field.

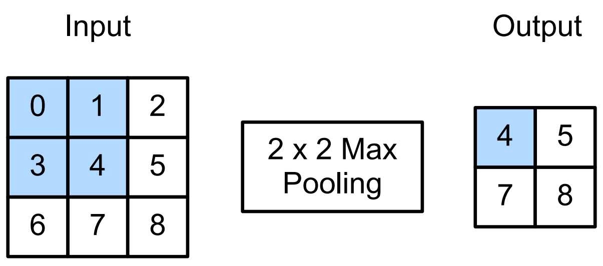


Figure 8 - Example of Max Pooling, source: bit.ly/3vWzy0o

Although a lot of information is lost when pooling is applied, the computational cost is lowered significantly, as well as helping the network become invariant to translations. This means if the input is slightly translated, the pooling operation will produce the same output. This property, although it doesn't help detecting the feature location, it does influence on identifying if the feature is present or not.

2.3.3 Fully-Connected Layer

After the convolutional and pooling operations, the output is generally flattened into a column vector, a one-dimensional array of numbers, and connected to one, or more fully-connected layers.

This layer, also referred to as a dense layer, is used in order to classify the inputs based on the features extracted on the previous layers. This is just a feed-forward network that is used to learn in a non-linear way, the patterns of the output features of the convolutional layers. While the layers described in sub-section 2.2.1 usually use ReLU functions, Fully-Connected Layers often use a SoftMax or Sigmoid activation function depending on the classification problem, returning a output for each of the input vector values, which can be interpreted as the probability of each class.

2.4 Recurrent Neural Network

Recurrent Neural Networks (*RNN*) are a type of Artificial Neural Networks specially designed to treat sequential data or time series data. The main characteristic of this network is that they are comprised of neurons with one or more feedback loop [17]. This allows the network to remember prior inputs, influencing current inputs and outputs. In CNNs and other types of Neural Networks, it is presupposed that the inputs and outputs are independent of each other, but in RNNs the outputs depend on prior elements within the temporal sequence. It is also important to note that these networks, much like CNNs, also use parameter sharing.

Because of the "memory" property of Recurrent Neural Networks, they are often used in the fields of NLP [18], speech recognition [19] and can also be applied in audio classification tasks by combining this architecture with convolutional layers [20]. Although this is a very interesting feature, if long-term memory is needed this type of network is no longer adequate, which introduces the concept of Long short-term Memory networks which will be described in the next sub-section.

2.4.1 Long Short-term Memory

Long Short-term Memory is a type of Recurrent Neural Network that aims to solve the short-term memory issue by using a gate mechanism, responsible to control input information. This architecture was motivated by the inaccessibility to long time lags when base RNNs are used, meaning that these simpler networks were unable to capture long-term temporal dependencies [21].

There are three types of gates on these networks: the forget gate, the input gate, and the output gate. The forget gate is responsible for the identification of what data should be forgotten. The input gate is able to decide what input data should be stored and finally, the output gate passes the updated information from the current index on the time series to the next one [22]. All three gates receive two input vectors x_t and H_{t-1} which refers to the input vector on the instant t and the output vector of the input gate on the instant $t-1$, respectivily.

Mathematically the operation that occurs on the forget gate can be define as:

$$f_t = \sigma(x_t * U_f + H_{t-1} * W_f)$$

Where x_t is the input to the current timestamp, U_f is the weight associated to the input, H_{t-1} is the hidden state of the previous timestamp and W_f is the weight matrix associated with that hidden state. A sigmoid function is then applied to the information that will return a number between 0 and 1. If the value is 0 then the network will forget this information and if the value is 1 it will forget nothing.

The operation on the input gate can be defined as:

$$i_t = \sigma(x_t * U_f + H_{t-1} * W_f)$$

After the sigmoid function is applied, a *tanh* function is also executed returning a value between 1 and -1. If the value is negative the information is subtracted from the cell state and if it is negative the value is then added to the cell state.

Finally, on the output gate the following operation is executed:

$$o_t = \sigma(x_t * U_o + H_{t-1} * W_o)$$

Again, the same functions as the previous gate are calculated in order to get the output for the next cell.

2.5 Residual Neural Networks

One of the biggest problems in training deeper architectures is the existence of exploding and vanishing gradients. Back propagation is applied to compute the gradients using the chain rule which can inevitably lead to an exponential growth or vanishment of the gradients, preventing the weights from updating and thus not allowing the architecture to

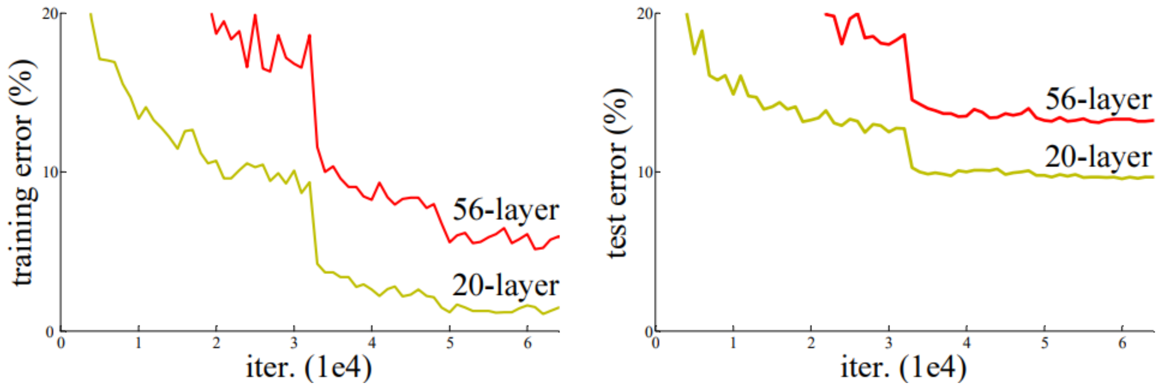


Figure 9 - Training error and test error on CIFAR-10 with 20-layer and 56-layer "plain" networks. Source: Adapted from [22]

perform better. In [23], He et al. show an example of a 20 layer CNN against a much deeper CNN one. This plot shows a greater training and test error on the deeper network against its shorter counterpart.

Residual Neural Networks (*ResNet*) were created to solve the training of very deep neural networks by using residual blocks. These residual blocks use a type of connection called skip connection also known as shortcut connections. This type of mapping doesn't add additional parameters but allows to add a layers' output to the following layer.

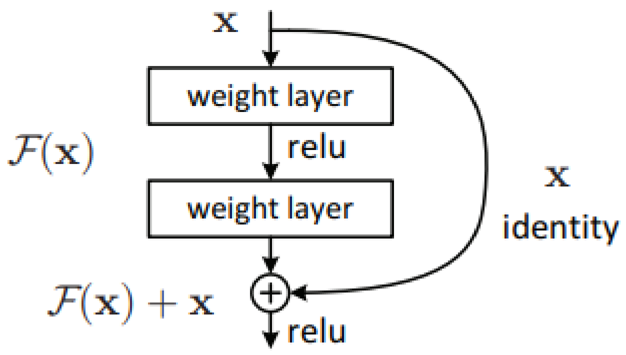

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 10 - Residual block. Source: Adapted from [22]

However, a layer's output dimension x can differ from the spatial dimension of the output of the following layer $F(x)$. In order to solve this, the authors propose that the identity mapping (the output of the previous layer) is multiplied by a linear projection W to increase its dimensions, matching the following layer (also called the residual), allowing both outputs to be combined as the input for the next layer. This procedure can be expressed by the following math function:

$$y = F(x, \{W_i\}) + W_i x$$

In the previously discussed article, He et al. show that by adding residual blocks and increasing the network depth, networks achieve higher accuracy against its less deep counter parts.

2.6 Regularization

To classify a machine learning model as successful, it needs to achieve a low training error while minimizing the difference between training and test error. Training error is a metric that defines the error over the data used for training the model, while the test error is measured on the predictions made on unseen data, in other words, a validation set. Commonly, when training a new DL system, a very high training accuracy will be seen, but it will fall short when predicting on new data, meaning the generalization error is rather high. This is known as overfitting, and it happens when the model is too complex for the given task or simply there isn't enough data to train on.

Regularization methods were introduced to deep learning systems, as a way to reduce overfitting as well as keeping the training error at a minimum. In this section, the most commonly used techniques will be described.

2.6.1 L1 and L2 Regularization

L1 and L2 regularization methods are both very common techniques of model regularization [13; 14; 15]. These procedures update the general cost function by adding another term designated as the regularization or penalty term. Due to the addition of this regularization variable to the general cost equation, both approaches try to penalize bigger weights [27], making the values of the weight matrices tend to decrease. The key difference between these methods is the term that they add to the cost function. In L1 it can be expressed as:

$$C_\lambda = C + \frac{\lambda}{2m} * \sum\|w\|$$

The main purpose of this model is to turn the less important feature's coefficient into zero, and consequently remove some irrelevant features altogether. This operation can work particularly well when a large number of features are present, making it useful to compress the model. On the other hand, L2 Regularization can be represented as:

$$C_\lambda = C + \frac{\lambda}{2m} * \sum\|w\|^2$$

This means that it forces the weights to decay towards zero, making the model prefer smaller weights. This method is generally preferred over L1 when model compression is not required.

2.6.2 Dropout

The dropout regularization method is a very simple method that, at every iteration, randomly selects a set of nodes and disables them, and consequently, removes all their incoming and outgoing connections. This means that each iteration has a different set of nodes resulting in different outputs. This is the same as sampling a sub NN from a larger network [27]. This emulates a different model architecture at each iteration, translating to a very computationally cheap and effective way of applying regularization, which makes this method the most common among data scientists.

The Dropout technique uses a hyperparameter *p* which sets the probability for the units to be disconnected and it can be tuned in order to achieve better results.

2.6.3 Batch normalization

When supervised machine learning is applied, a model will learn the patterns or relationships between the input and its labels through training datasets. A common problem that happens with these models is that the input distribution can differ from the real-word or test data, meaning that the model may make wrong predictions. This obstacle is often referred to as covariance shift [28]. Furthermore, internal covariate shift can also occur, given that during neural network training, as the parameters of a preceding layer change, the input distribution of a layer also changes. This negatively affects the training speed and requires a more careful parameter initialization.

In order to tackle this problem, Batch Normalization [29] was introduced. It's a layer that applies standardizing and normalizing operations on the input received from the precedent layer, without altering its shape. This not only handles internal covariate shift but also makes for a faster learning rate and more care-free initialization of parameters.

2.6.4 Early Stopping

Early stopping is the simplest method out of regularization techniques. In this specific case a monitor is used to keep track of a certain metric, generally being an error metric such as the validation loss. When the validation error is getting worse the model simply stops training, hence the name, early stopping. Besides the monitor parameter, early stopping also takes the number of epochs which sets the interval with no improvement after which training will be stopped. Also, a minimum delta can be set which defines the minimum variance change that needs to be seen on the monitored metric in order to qualify it as an improvement.

2.7 Sound and Audio

In simple terms, sound is a pressure wave which is created by a vibrating object and is transmitted through a medium that can be solid, a gas or a liquid. These variations in pressure can be represented over time which is often referred to as a sound signal.

There are periodic and aperiodic sound signals, where in the first, the sound wave repeats itself at a period T, where a phenomenon of compression and rarefaction are seen represented by the height of the wave. This height represents the intensity of the sound, and it is known as the amplitude (A). On the latter, the same occurrences are also seen, but the wave does not oscillate on a repeated pattern. Most sounds fall on this sound signal category, like the human voice, a bird chirping or an instrument being played. In order to represent these sounds, sound signals with different frequencies can be added together creating composite signals. Frequency represents the number of waves that pass a fixed point in time.

2.7.1 Audio Digital Conversion

Converting analog audio to its digital form is done by the process called ADC (Analog Digital Conversion). There are multiple complex components present on this method, but on this section only the two main concepts will be covered, these being Sampling, and Quantization or Resolution.

In order to convert analog audio, the ADC module takes measures of amplitude of the sound at a fixed interval of time. Each of these measurements is called a sample and the number of samples taken per interval of time is called sampling rate. So, during sampling, if a 22050 Hz sampling rate is used on a 30 second audio clip 661500 samples are returned. Quantization refers to the number of bits used to store each sample amplitude point. The greater the bit depth, the more accurate the representation of a sound will be, given that a wider interval is used in which amplitude can be described.

2.8 Audio Data preparation for DL architectures

Historically, audio classification problems were approached with machine learning models, depending heavily on digital signal processing techniques to extract audio features that are used as inputs [30]. For example, features like zero crossing rate and short time energy would be extracted in order to classify environmental sounds [31]. For emotion detection, timbral, tonal and rhythmic features would be extracted [32], [33] . This requires a lot of audio domain expertise to solve these problems alongside being tuning dependable to reach useful results.

However, with Deep Learning development, these architectures have demonstrated a huge success in handling audio. Traditional audio data preparation is no longer needed, and it is surpassed by standard data preparation without the need of hand-crafted specific features. Although the audio signal can be used as an input, it is very common to transform it into its visual representations (also known as audio features) such as spectrograms and later feed it into the DL network.

2.8.1 Spectrum

As mentioned in previous sections, natural sounds cannot be represented by a periodic single sine function. The Fourier transformation equation allows the decomposition of complex sound waves into a sum of sine waves oscillating at different frequencies. There are multiple variations of this method with different outputs, but only Fast Fourier Transformation will be covered on this subsection. After FFT is performed on a sound signal, a spectrum is returned, moving from the time domain to the frequency domain. It can be seen as a snapshot of all the frequencies that represent the sound but losing the time aspect of it.

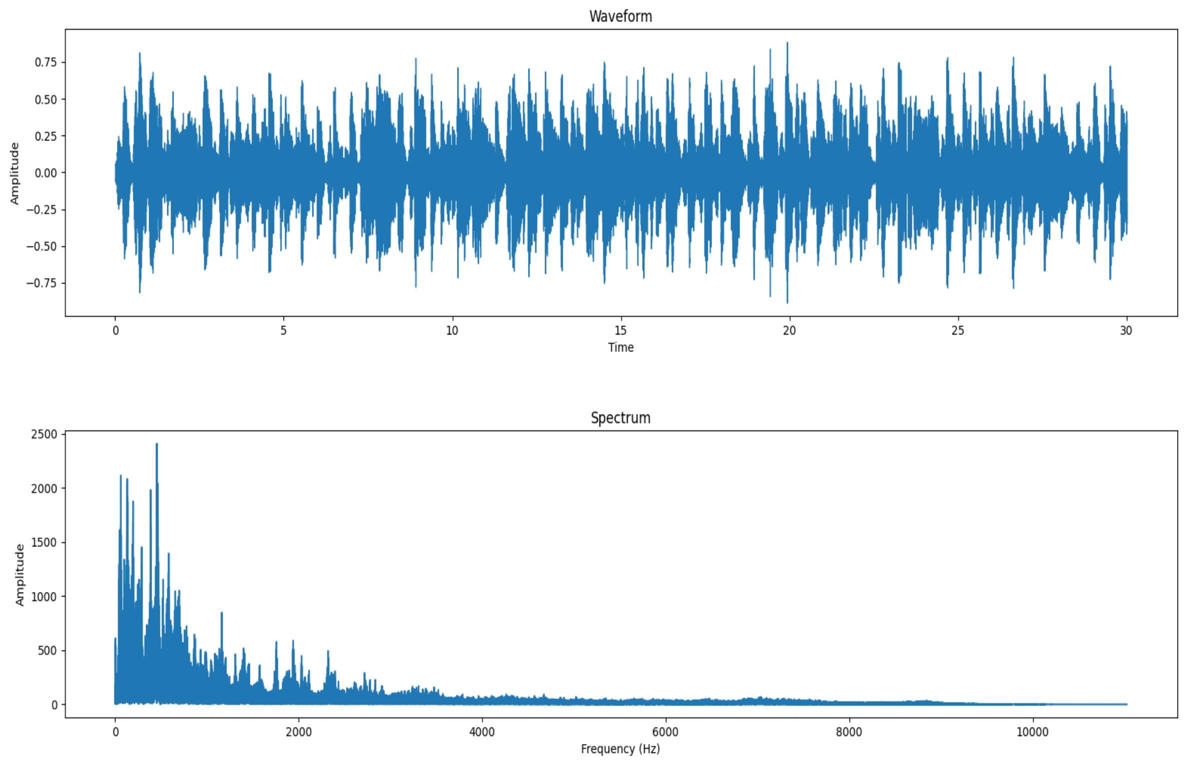


Figure 11 - Example of a Spectrum obtained using FFT

Although this type of audio representation is not really used on Deep Learning models as an input, it is useful to study how energy level of each frequency is distributed on the audio sample. It is often used to calculate amplitude thresholds in order to remove noise from the sampled data.

2.8.2 Spectrograms

When FFT is performed, a simple representation that averages the presence of the frequency components across the whole duration of a signal, is returned. So, the frequencies' magnitude are known but not when they are more or less present. Audio data is characterized by the evolution of its frequency components over time, hence the use of spectrograms on Deep Learning architectures.

Spectrograms are a feature that can be extracted by applying the Short Time Fourier Transformation. In order to preserve the important time aspect of sound, FFT is computed at different intervals with a fixed frame size (e.g., 2048 samples) returning a spectrogram, where the magnitude is represented as a function of time and frequency. The magnitude dimension is described by color, allowing us to understand the presence of that frequency on a given time.

In other words, a Short-time Fourier Transformation is a series of FFTs performed on a windowed signal, providing time-localized frequency information for data like audio signals, in which its frequency components vary over time [34].

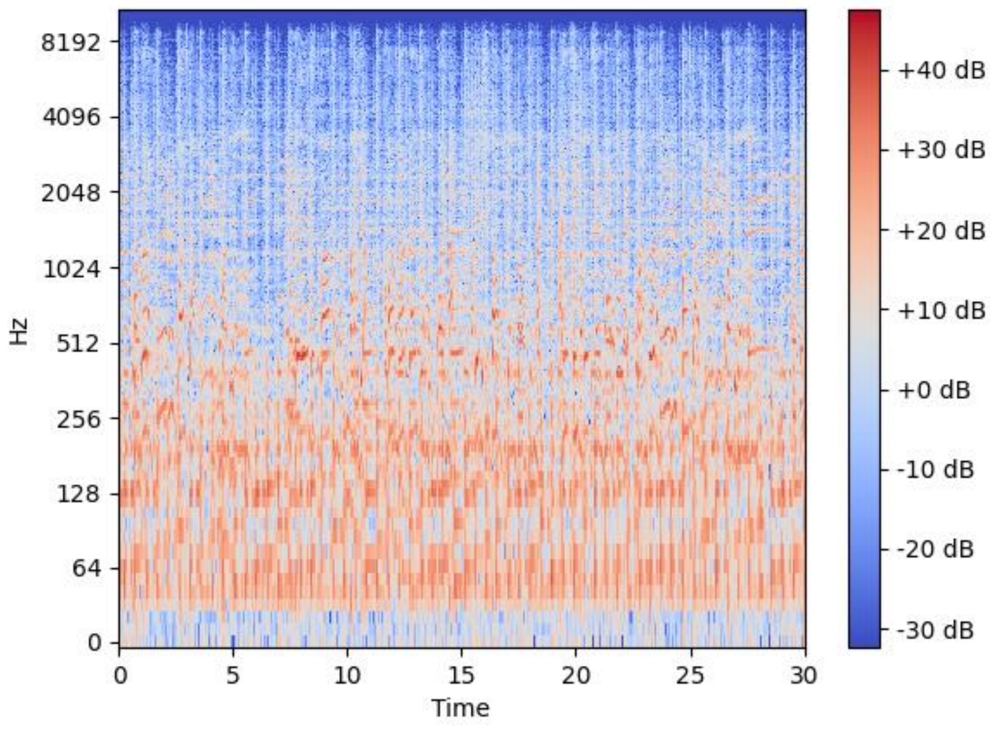


Figure 12 - Example of a Logarithmic Scaled Spectrogram

Mathematically, the STFT equation can be represented as:

$$X(m,k) = \sum_{n=0}^{N-1} x(n+mH) \cdot w(n) \cdot e^{-i2\pi n\frac{k}{N}}$$

This function is characterized by its windowing function $w(n)$ that its multiplied by the number of samples present on the frame $x(n + mH)$ where m is the current frame and H is the hop length, which describes how many samples the window shifts.

Most of the time, the endpoints of a signal are discontinuous, because they're not an integer number of periods. This translates to spectral leakage on the spectrogram, meaning that these discontinuities appear as high-frequency components that are not present on the original signal. In order to minimize this problem, the standard procedure is having a smaller hop length than the frame size, originating frame overlapping which accounts for the information that is lost on the endpoints of the framed signal.

Moreover, when extracting spectrograms using STFT, there is an important time-frequency tradeoff that is related to the parametrized frame size. The larger the frame size, higher the frequency resolution and lower resolution on the time domain, and vice-versa [35]. There are some heuristics that can help choosing the number of samples for the frame size, but it is mostly dependent on the problem that is being investigated.

2.8.3 Mel-Frequency Cepstral Coefficients

Mel-Frequency Cepstral Coefficients are a very widely used feature for audio related Machine Learning tasks [36], especially on speech recognition [37], [38]. A Mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound signal, based on a linear cosine transformation of a log power spectrum on a Mel-scaled frequency. The MFCCs are coefficients that collectively form an MFC. The Mel scale is related to how humans perceive frequency or pitch of a tone to its actual measured frequency. The steps in order to retrieve MFCCs from an audio signal are:

1. Use a windowing function to break the signal into overlapping frames
2. Perform the Discrete Fourier Transformation to the framed signal.
3. Map the powers of the spectrum obtained on the previous step to the Mel-scale by using triangular overlapping filters
4. Calculate the log of each of the energies returned on the previous step.
5. Perform the Discrete Cosine Transformation to each of the log *filterbank* energies in order to obtain the cepstral coefficients.

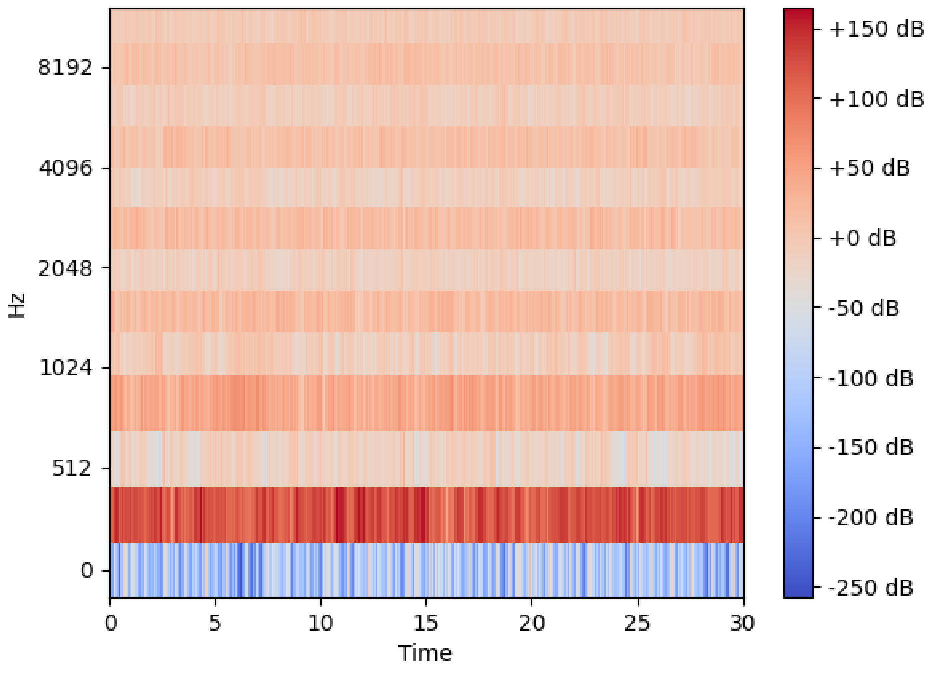


Figure 13 - Example of MFCCs representation

2.8.4 Mel-Spectrograms

In simple words, Mel-Spectrograms are Spectrograms converted to the Mel-scale. The Mel-Spectrogram is a very common form of Spectrogram used as input to Convolutional Neural Networks designed for audio classification. The way these spectrograms are obtained is quite similar to the MFCC feature extraction. First the signal is divided into windows and the FFT is computed to each frame. Then the Mel scale is generated by taking the entire frequency spectrum and separating it into an arbitrary number of *mels*, becoming evenly spaced frequencies, in the sense as how humans perceive those frequencies. Finally, for each window the magnitude is decomposed into its components and the Mel-Spectrogram is obtained.

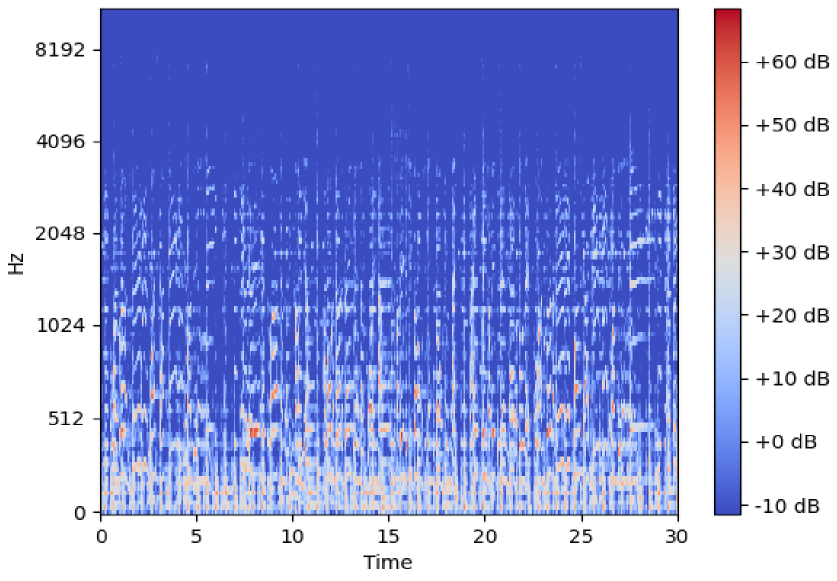


Figure 14 - Example of a Mel-Spectrogram

2.8.5 Constant-Q Transform

Constant Q Transforms or CQT for short, are a type of audio signal representation commonly used in audio classification as well [39]. Its calculation is very similar to the previously discussed STFT, however the spacing between the first harmonics are based on an increasing logarithmic space [40]. By increasing its window size for lower frequencies and increasing this window for higher frequencies a lower computational power is needed, which is why CQTs are considered an optimized version of the STFT. It also has its draw backs since reducing the window size for the last harmonics reduces the detail on the upper frequencies.

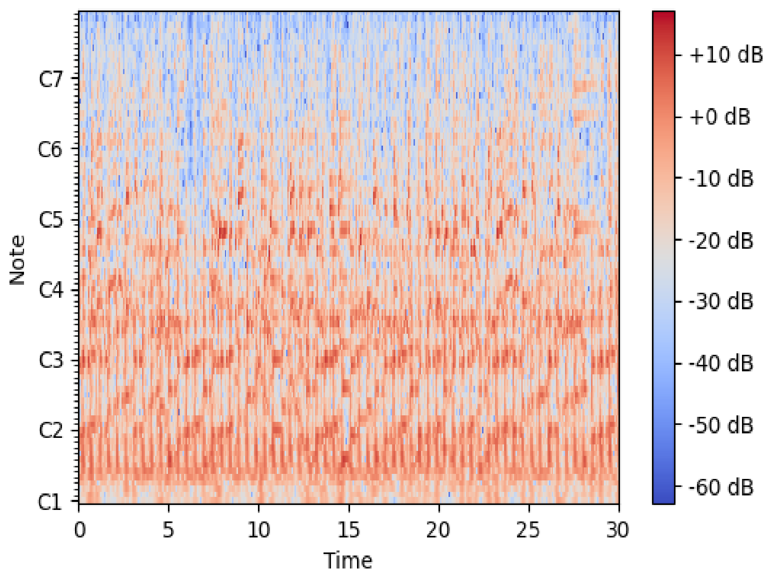


Figure 15 - Example of a Constant-Q Transformation

2.8.6 Chroma vector

The human ear perceives sound pitch in a periodic way, meaning two pitches can be perceived in the same way if they differ by one or multiple octaves. A chroma vector, or chromogram is pitch scaled STFT, meaning it shows a 12-element vector for each pitch class. A pitch class means every pitch that is separated by one octave.

This type of feature can be a useful tool to analyze sound waves from a music file when pitch is an important tuning feature, being a powerful descriptor of the tonal content of a musical audio signal.

This type of feature is also very commonly used for audio classification tasks. Research found that chroma vectors are popular among algorithms for thumbnailing music [41], [42].

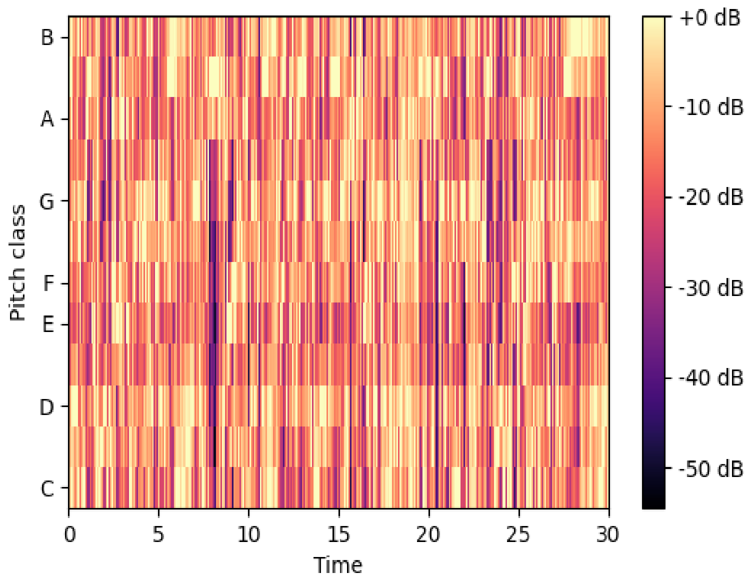


Figure 16 – Example of a Chroma vector

2.9 Data Augmentation

Data augmentation is a common technique that can be applied to a dataset in order to increase its diversity, especially when not enough data is present. This is done by artificially modifying existing data in small ways.

In image recognition there are some basic approaches to augment the data that are meant to be used as input to a DL model, these being: image cropping or scaling, image rotation, color modification or adding noise to the image [43]. Although the new images are a modification of the original ones, the semantics have not been touched. For instance, a rotated image of a cat is still considered as a cat, but for a Deep Learning model is a new data sample that will help the models' generalization capability.

On audio, the previously discussed augmentations can't be applied since they change what the audio features represent, meaning semantics are altered which introduces a lot of noise on the training data, increasing training error and decreasing its prediction capability. However, there are also multiple ways to insert data augmentation either on the spectrogram representation of the signal or on the signal itself [44]. In this section only Spectrogram Augmentation, Time Shift, Pitch Shift, Time Stretch and Noise Addition will be covered.

2.9.1 Spectrogram Augmentation

Normal transformations such as rotation or color alteration cannot be applied to spectrograms since it would alter the sound that it represents. In 2019 Park et al. introduced a data augmentation method for automatic speech recognition [45] known as *SpecAugment*.

They experimented in two ways:

- Frequency Masking: this procedure randomly masks a range of consecutive frequencies, translating to a horizontal bar on the spectrogram.

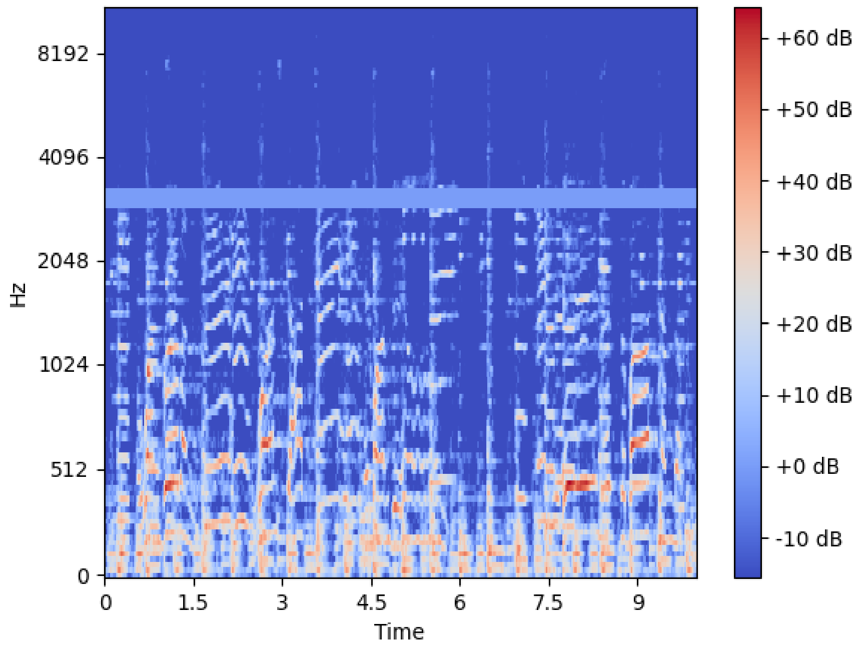


Figure 17 - Frequency Masked Mel-Spectrogram

- Time Masking: very similar procedure to the frequency mask method, but instead random ranges of time are blocked from the original spectrogram using vertical bars.

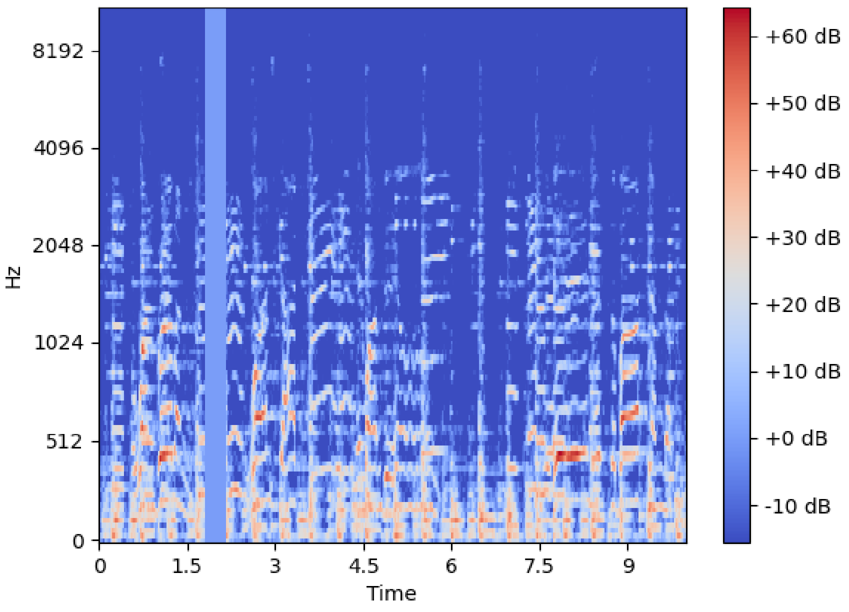


Figure 18 - Time Masked Mel-Spectrogram

2.9.2 Time Shift

Time shifting is simply shifting the audio to the right or the left by a random amount. For sounds with a repeated pattern the audio can be repeated when shifted. In human speech however, the order of the sound is a must for speech recognition so the gaps created when shifting can be padded with silence.

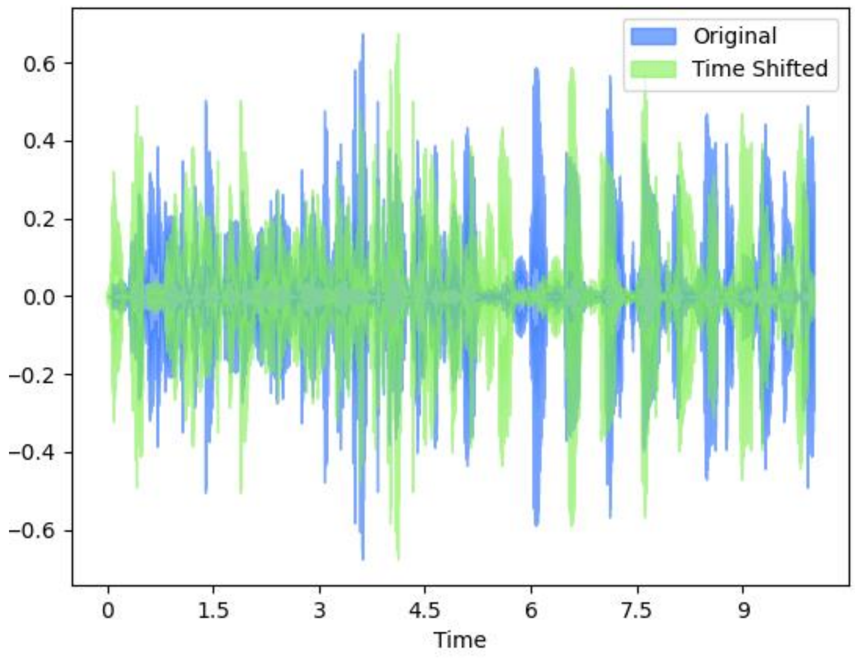


Figure 19 - Example of a time-shifted signal

2.9.3 Pitch Shift

Pitch shifting changes the pitch of a signal by an arbitrary number of semitones without altering its tempo.

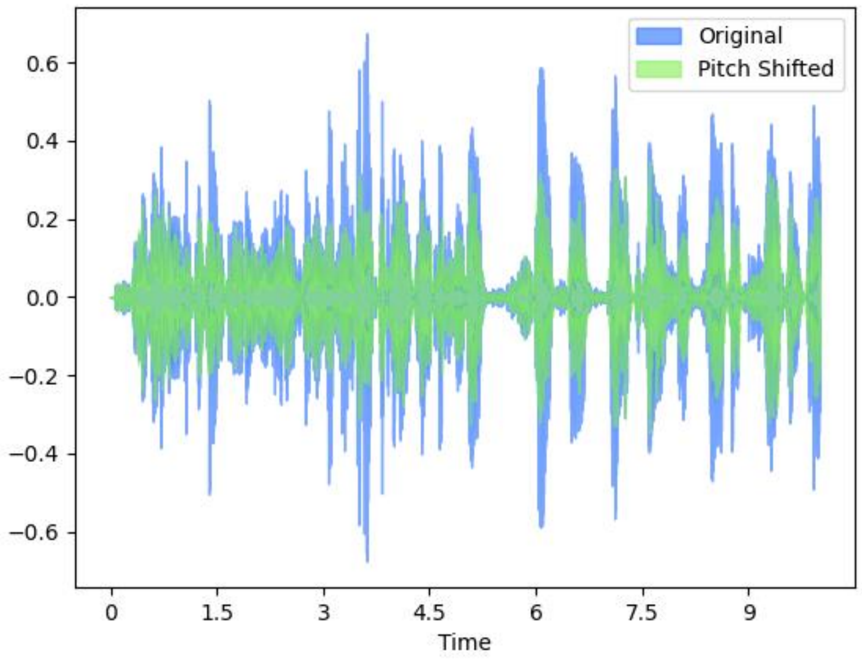


Figure 20 - Example of a pitch-shifted signal

2.9.4 Time Stretch

Time stretching alters the tempo of an audio clip, meaning the speed of the audio can be altered without changing its pitch. Consequently, the length of the signal will also change, so padding needs to be added in case the time stretched signal has a smaller input shape and clipping in case the signal dimensionality becomes too long.

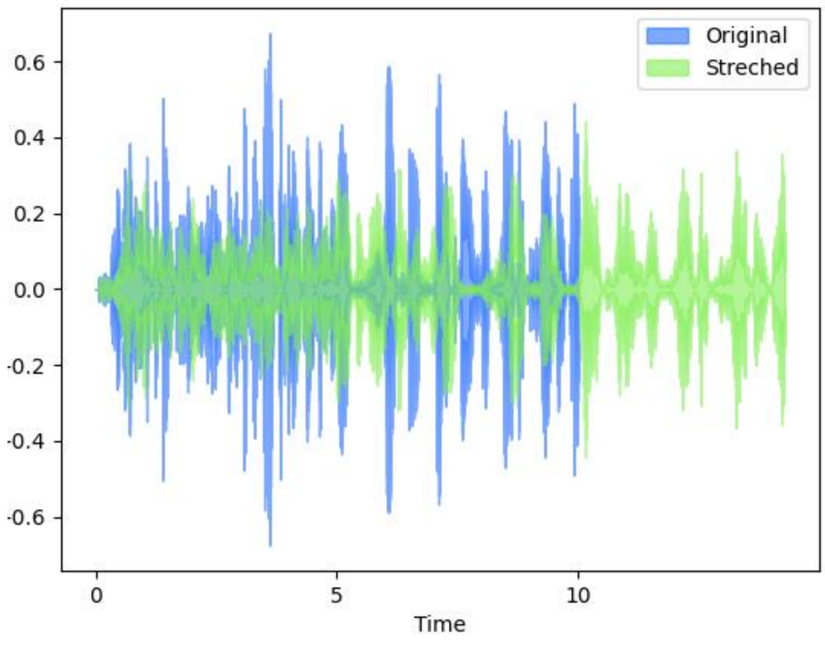


Figure 21 - Example of a time-stretched signal

2.9.5 Noise Addition

Gaussian noise can be added to the audio signal which makes the input smoother and easier to learn. It's a very simple but effective way of applying data augmentation to audio data.

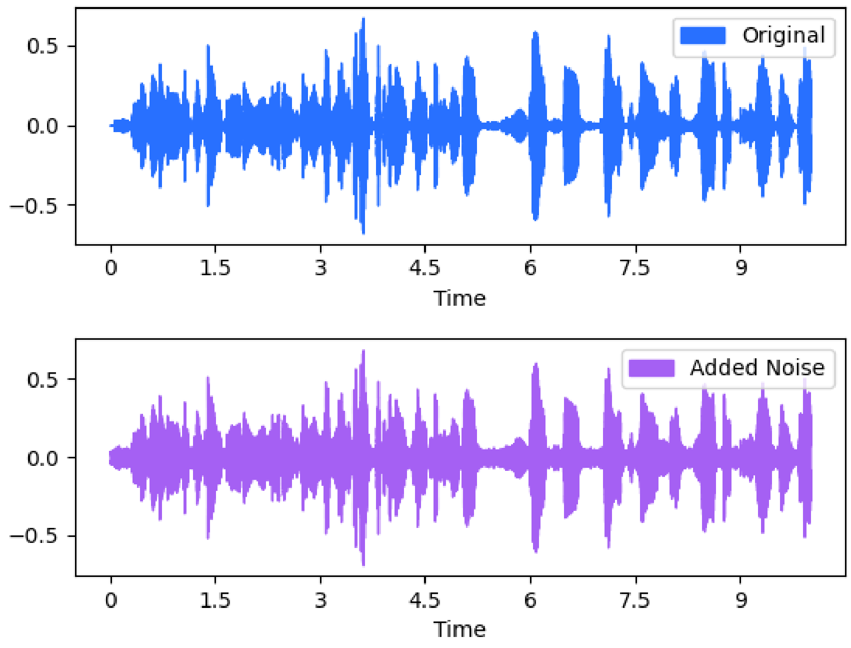


Figure 22 - Example of Gaussian noise added to a signal

2.10 Common approaches

Since the study object of this dissertation is Violence Detection using audio data, it also falls into the categories of Sound Event Detection (SED) and Environmental Sound Classification (ESC), thus, in this section, common approaches for SED and ESC will be described as well.

Historically, many approaches on Sound Event Classification and Environmental Sound Classification have heavily relied on speech recognition techniques. This means the feature extraction was a method of choice to use as inputs. The most common type of features were Mel Frequency Cepstral Coefficients (MFCCs) [46], [47], [48], Average Zero Crossing Rate [49], Constant-Q Transformations [39], Mel-band Energy Features [50] and hand-crafted specific features [51]. These were widely used in combination with Machine Learning classifiers such as the Non-Negative Matrix Factorization [52].

The state of the art for these classification problems is based on Deep Neural Networks, including Feed Forward Networks [32; 33], Convolutional Neural Networks [55]–[57], Residual Neural Networks [58] and the more sophisticated Recurrent Neural Networks with transfer learning [59]. These allowed to overcome the domain specific knowledge needed to perform machine learning on audio data, by simply using images as input, such as spectrograms, and managing to learn high-level features from them.

As this master dissertation will focus on Violence Detection using Deep Convolutional Neural Networks as a medium for feature extraction and sound classification, it is important to dive into more detail about this type of approach.

In [55], Salamon and Bello use logarithmic scaled mel-spectrograms as inputs to a CNN in order to classify environmental sounds present on the Urban8K dataset. They introduce a problem related to the scarcity of labeled audio event data and propose data augmentation as a method to solve this issue. This stage included the following augmentations: time stretching, pitch shifting, dynamic range compression and the addition of background noise. Although data augmentation should help on creating a larger dataset and thus converging to a higher accuracy model, this has not been the case for this specific dataset given that the training time greatly increased and the difference on model accuracy was negligible. The authors' model is comprised of 3 convolutional layers with 2 pooling layers between them followed by 2 dense layers. All the layers were using ReLU as the activation function except for the output layer which used SoftMax. The categorical accuracy reported for this method was an average of 79%.

Using logarithmic mel-spectrograms and delta feature spectrograms as input features and the same dataset, Zhang and Zhou in [57] approached this problem with a different architecture. They identified a research gap, this being that most approaches on Environmental Sound Classification would use smaller filters on their convolutional layers which translates to a need of a deeper convolutional network in order to learn log contextual information. In order to resolve this issue, the authors propose the use of dilated convolution filters in order to increase the receptive field of the CNN without introducing more parameters and layers. The first part of the network comprises of 2 dilated convolutional layers running LeakyReLU as the activation function, and 2 max-pooling layers between them. These were divided into two channels, one receiving log mel-spectrograms and the other delta feature spectrograms. The previously described network is then connected to 2 fully connected layers and these to an output layer with SoftMax as the activation function. It is important to mention that the authors also used a data augmentation module that applied time stretching and noise adding to the inputs. The proposed architecture got a solid 81.9% categorical test accuracy.

In [58] Palanisamy et al. use state-of-the-art techniques to compare sound classification performance on single and ensemble models using the GTZAN, UrbanSound8K and ESC-50 as their datasets. They start by extracting multiple audio features and feed them to base-line model (SoundNet) to understand which type of audio feature would perform best, founding that the Logarithmic Mel-Spectrogram was the best audio representation for this specific problem. To fine tune the models' performance, multiple hop lengths and window sizes were used to extract the feature and then fed to the models while also comparing accuracy when using pretrained and random weights. It is also necessary to note that data augmentation has been implemented, specifically pitch shifting and time stretching. On the final experiment, the pretrained DenseNet, ResNet and Inception models were tested on all three datasets in a single and ensemble matter. The DenseNet achieved the higher validation accuracy across the board, losing only to the ResNet in its ensemble version, specifically on the GTZAN dataset.

Much like SED and ESC, historically, violence detection has been based on feature extraction from video and images like spatial-temporal features, optical flow, motion information, and acceleration [60] and feeding them to a Machine Learning algorithm [61] were the most common way to approach this problem. Auditory data has often been neglected for this type of problem, although some researchers include it, like in [62] Mahadevan, Li, Bhalodia and Vasconcelos proposed an system that tried to identify violent scenes based on blood and flame detection combined with motion and sound.

There are multiple approaches that were studied regarding violence detection, but in this section only state-of-the-art Deep Learning based approaches will be covered since they are known for easier integrations and better results.

In [63] Abdali and Al-Tuma propose a architecture to classify hockey videos as violent or non-violent, using a CNN as a spatial feature extractor that feeds its output to a Long Short-Term Memory cells that then extracts temporal patterns from the inputs which are fed to a dense layer using sigmoid as the activation function since it’s a binary classification problem.

They use a pre-trained model, the VGG-19 and since the dataset was rather small to get the best results, they needed to use transfer learning. Firstly, the architecture receives a 4d tensor, a sequence of frames with the shape (40x160x160x3) corresponding to (frame, height, width, RGB color channels) where the pre-trained VGG19 processes each one. The output of the previous step is grouped and flattened into a 2d vector representing a spatial feature for one frame. Each of the outputs are then processed by the LSTM, global average pooling is applied in order to get a 1d vector and finally its fed to a fully connected layer which will be used to get the probability of violence in the given video. On the test set the model achieved a 98% accuracy.

2.11 Related work

Regarding violence detection tasks using only audio data, only two articles related to the issue were found [64], [65].

In [64] Theodoros, Dimitirios, Andreas and Sergios described an approach that aims to be a contribution for automated characterization of multimedia content with respect to violence. The authors start by extracting six audio features from each segment of an audio file from the time and frequency domain. These are energy entropy, signal amplitude, short time energy, zero crossing rate, spectral flux, and spectral roll off. In order to classify these audio signals as violent or non-violent content they use a Support Vector Machine using the previously described features as normalized inputs. The dataset was divided in half, obtaining the training and testing sets.

The authors present the classification error rates for each individual feature as well as the error rates when all the features were used together (8-D case), this was meant for a better understanding in future feature selection. On the 8-D case, the model recall was 90.5%, a precision of 82.4% and 85.5% accuracy.

In [65] Giannakopoulos and Pikrakis present a very uncommon approach on classifying violent content using audio from movies. They use a multi-class classification scheme, dividing the labels into 6 classes, where 3 of them are considered as violent. Each audio file is broken into segments and each segment is divided into frames and for each of them, twelve feature sequences are calculated, these being: zero crossing rate, spectrogram, chroma vector features, energy entropy, spectral rolloff, pitch, and MFCCs. All these features go through different statistical calculations except the chroma vector features and MFCCs that go through multiple equations. Using 30 movies hand-labeled by the authors and feeding them to a Bayesian Network the authors got a 90.8% recall, 86.6% precision and 89% accuracy for the violence classification.

2.11.1 Anomaly Detection

Anomaly Detection using audio sources is a fairly common investigated subject, and since violent scenarios can be considered as anomalies, this subsection will be used in order to describe these types of work.

Historically, the most common way to perform anomaly detection is using Auto Encoders [66] which are a form of a feedforward, fully connected neural where the output layer has the same dimensionality as the input layer. This network compresses the input, encoding it in a reduced dimension, known as the latent space, learning the most important relationships. This distorted version of the input is later decoded, and it is reconstructed back to the original dimensions in an unsupervised manner. The difference between the output vector and the input is called the reconstruction error and by using a threshold it is possible to detect anomalies.

Although auto encoders are known by their low computational needs, new architectures are also being explored with better performances such as transfer learning techniques, features can be extracted from different audio representations by Deep Neural Networks like Recurrent Neural Networks and then fed into an AE [59].

Another example is the use of a LSTM neural network with 10 units in order to detect anomalies in 3D-printers [67]. MFCCs and mel filter banks were the author's choice as inputs and since acquiring printing anomaly recordings is very time-consuming, data augmentation techniques were also used to increase training data.

Rushe and Namee did a research on using raw audio signals paired with a convolutional autoregressive architecture [68] and presenting significant performance gains over deep autoencoders when it comes to anomaly detection. They used the *WaveNet* architecture, training it to predict the next sequence using non-anomalous samples, meaning the network will learn the conditional distribution across normal data that anomalous sequences won't follow. The model then predicts the next sequence, compares it with the subsequent actual value and if the mean squared error is high then it is indicative of an anomaly.

This type of architecture has benefits over Recurrent Neural Networks by using dilated causal filters in order to increase the receptive field and ReLU units are replaced by gated units in order to obtain the benefits of a LTSM without the need of a recurrent algorithm, since these are known for their difficulty on parallelizing backpropagation through time, slowing the network training process.

3. METHODOLOGY, TECHNOLOGIES, AND TOOLS

With the purpose of understanding the fundamental concepts for this master dissertation, deep research for article pre-selection was conducted. The platforms used were *ScienceDirect*, *Google Scholar*, *Semantic Scholar*, *Google and Web of Science.* Title, abstract and number of citations for each article were the properties chosen by the author in order to identify relevant work for this dissertation.

A methodology is a study of the best methods that are used in a specific domain in order to achieve a certain goal of knowledge. In this dissertation the CRoss-Industry Standard Process for Data Mining was the adopted methodology to structure the plan of action. The reason behind this choice is that this methodology is more complete than, for example, SEMMA or PMML. It offers superior advantages over them, such as: greater project feasibility, greater project viability and faster development and lower development costs [69]. Moreover, this methodology has previously been studied and applied by the author in previous projects.

In addition, to contextualize the practical development of this dissertation there is a need to identify the technologies and tools that are meant to be used for its implementation, thus, a summary of all the chosen technologies, tools, and libraries were also conducted in this chapter.

3.1 CRISP-DM

CRISP-DM, Cross-Industry Standard Process for Data Mining, was developed in 1996 and later published in 1999, with the purpose of orienting the development of Data Mining projects [70], guarantying a lower project complexity, lower development costs and a management ease.

This methodology, presented on Figure 18, is comprised by six different steps and each of them is defined by a second level series of tasks that can either be applied or not. This methodology can be applied to the process of Machine Learning implementation, providing a lower project complexity, and ensuring that the requirements are fulfilled.

Ideally, the transition between the six phases of the CRISP-DM methodology should be done in a linear form. However, Machine Learning projects suffer from great complexity and the transition between phases can, and most likely will be done in a circular motion, meaning that it is possible to backtrack to previous steps.

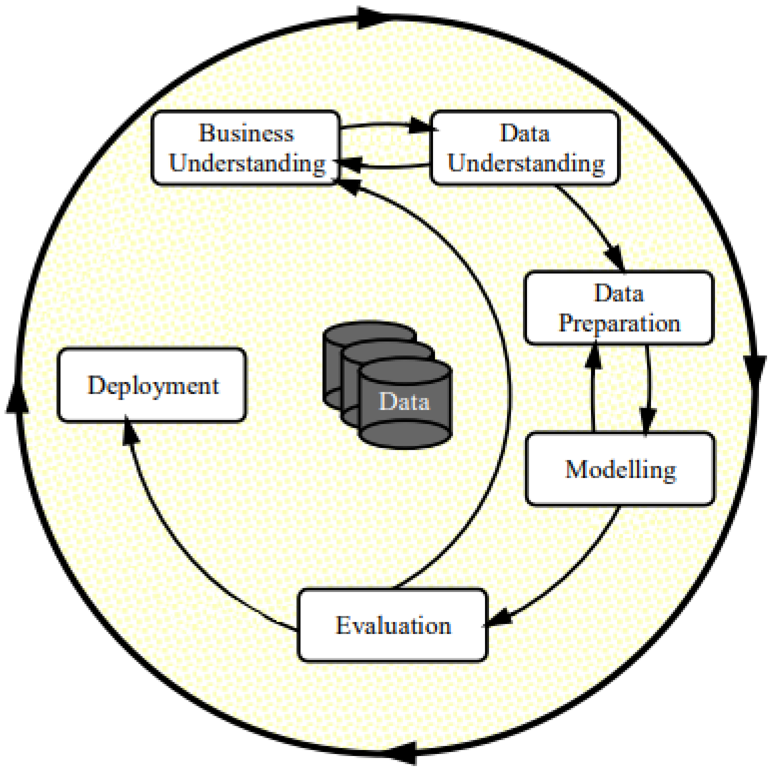


Figure 23 - CRISP-DM Methodology (adapted from Wirth and Hipp, 2000)

1. *Business Understanding*: The output of this step is the necessities and business objectives. It is also expected an objective definition in order to establish a plan of action for the different requirements. It is very common to backtrack to this step when the project is in an advanced state, given that the requirements can change over time.
2. *Data Understanding*: Data collection and data analysis is performed on this step. This is meant to detect data incoherence and understand the relationships between the variables. It is always possible to go back to this step, regardless of the project state, either because the dataset has been altered or new data has appeared.

3. *Data Preparation*: A very important phase where the problems detected on the previous step are dealt with. This involves data processing and treatment, and when new issues are found on *modelling* and *deployment* it is possible to go back to this step.
4. *Modelling*: In this phase is where the modelling techniques are chosen in order to be applied to the problem at hand. It is mandatory to learn the different model requirements regarding the project limitations. The parameters of the model should also be adjusted in order to obtain the best results.
5. *Evaluation*: On this step is where all the previous models will be assessed in order to determine which model will be used in the future implementation. There is a common back and forth between this phase and *modelling*, this is known as hyperparameter optimization.
6. *Deployment*: Lastly, a result assessment is planned, including the steps and how to execute them. A final report is developed, and a project revision is done. In certain cases, the final model can be implemented on a real environment.

3.2 Technologies and Tools

In this section, the different technologies and tools that were used on the practical development of this dissertation are described. Given the nature of this dissertation, it was developed entirely in the programming language Python. Python is a programming language created in 1991 by Guido van Rossum and it is often used to build websites, software, task automation and data analysis. There are multiple languages that could be used for Machine Learning purposes, such as R Programming Language, C++ or Java.

The main reason this language was chosen in detriment of others, is because the amount of experience the author has with it and the wide number of libraries that can help with pre-processing tasks and modelling. Furthermore, Python is the most used programming language for Machine Learning, meaning there is a vast support available online. In Table 1, there is a brief description about the libraries and packages that were relevant in the practical development of this dissertation.

Table 1 - Tools and Python Libraries

Name	Description	Version
Pandas	Pandas is a package made on top of NumPy and is most widely used for data science and data analysis tasks. It uses a tabular data structure known as *dataframe*, that allows to execute functions over the data that it contains. Because Pandas is so widely used, it works very well with most Machine Learning libraries.	1.4.1
NumPy	NumPy is the library of choice to perform mathematical operations on multidimensional python arrays. It utilizes powerful data structures that allows efficient and fast calculations with matrices being essential for Machine Learning tasks.	1.22.4
Matplotlib	Matplotlib is a library that allows the user to create data visualizations, using NumPy. This tool also as an API that permits the integration of its plots in real applications, allowing to create animated and interactive visualizations. In this dissertation it will be used to plot signals and its respective spectrograms. It will be important to plot training and validation metrics to evaluate the models' performance.	3.1
Librosa	Librosa is a python package used for music and audio analysis. This package is the starting point for this dissertation. It will allow to analyze and transform audio signals into its derivates. It allows to extract spectrograms, mel-spectrograms, and various audio features.	0.9.1
Audiomentations	Audiomentations is a Python library that is used for audio data augmentation. Its execution is very fast and supports a wide variety of audio augmentation methods which is perfect for increasing the amount of training samples.	0.24.0
Scikit-Learn	Scikit-Learn is a Python library that includes mathematical and statistical algorithms that are meant for Machine Learning tasks. It also has modelling modules for regression, classification, and clustering.	1.1.0
TensorFlow	TensorFlow is Python and JavaScript library, created by Google, that allows to build Machine Learning pipelines. It also takes advantage of data augmentation algorithms. It has embedded state-of-the-art techniques such as RNNs, CNNs and LTSMs that can be used for end-to-end audio classification.	2.9.0

Visual Studio Code	Visual Studio Code is a code editor with support for development operations like debugging, task running and version control. It includes syntax highlighting, code completion, snippets and code refactoring. It is also a very modular IDE since it allows to install extensions to further increase its features.	1.67
MobaXterm	MobileXterm is a software that allows for remote computing. It will be used in order to connect to Bosch's cluster in order to have dataset access and train the developed deep learning model.	22.0

4. Use case – In-car violence detection

This chapter includes all the practical work developed in all the CRISP-DM methodology phases which aids a better understanding of the project requirements. As a consequence, this chapter is divided in all of the CRISP-DM steps: business comprehension, data understanding, data preparation, modelling, evaluation and implementation.

On the Business understanding section the use-case is contextualized with the purpose of defining the business objectives, data mining objectives as well as the success criteria. The following section describes the dataset provided by Bosch and how it was obtained. A deep data analysis is also conducted along a data quality report of the chosen metadata which is used for the modeling phase. In the Data preparation section, all the data transformations to the WAV files and metadata are included and a data preparation architecture is also presented. On the Modelling phase, all the models used are described along the alterations done to its architectures, following by a description of all the test scenarios and their evaluation. Finally, on the Evaluation section the best model of each test scenario were compared in order to obtain the best combination of audio features and model architecture.

Furthermore, since there is an increasing amount of devices capable of audio recording in and processing in real time, all the code and additional information has been included in multiple appendices so that this investigation can be easily replicated in similar contexts.

4.1 Business understanding

For mobility service providers, vehicle safety must be ensured at all times, any problem regarding the condition of a car must be reported in a transparent way to ensure drivers and passengers safety as well as reducing vehicle down time.

In addition, there are multiple reports regarding assaults in mobility service providers, for example, Uber mentioned in a safety report 3824 sexual assaults reported in its US platform in 2019 and 2020 while 20 people were killed in assaults. These types of companies heavily rely on costumer safety and satisfaction as well as its drivers which cannot be measured in real-time. In addition, mobility service providers often offer a driver rating system which can most definitely support the costumer choice, but nothing can ensure that violence related incidents between driver and passenger or vice-versa won't happen. Furthermore, these types of events can only be addressed after the fact, instead they should be communicated to the fleet manager in real-time.

In this context, Bosch's RideCare project aims to provide a device capable of car damage detection using a sensor to classify damages into different categories (as well as storing GPS coordinates and timestamp). Smoke detection is another feature that reports the smoking event and its duration in real-time as well as anomaly driving detection which aims to identify irresponsible drivers and ultimately reduce the risk of car accidents.

This dissertation project is developed with the purpose of improving the current device by adding a new feature: in-car violence detection. This investigation aimed to achieve Bosch's business objective, it being whether in-car violent scenarios can be detected by only using auditory data as an input to a DL algorithm. Following this business objective, the data mining objective can easily be defined: classifying auditory data into non-violent or violent labels with the highest accuracy possible, meaning it's a binary classification problem.

4.1.1 Success criteria

In order to measure the investigation results it is essential to define metrics and criteria that can evaluate the results obtained by the different models proposed on the modeling phase. The selected metrics for the models' evaluation were accuracy, recall, precision, area under curve, F1-score and Binary Cross Entropy as the loss function. The mathematic formulas of the previously mentioned metrics are described in this sub-section. On Table 2 all the metrics are contextualized in this use-case as well as its utility and why they have been chosen.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{F1-Score} = 2 * \frac{Precision*Recall}{Precision+Recall}$$

$$\text{Binary Cross Entropy} = \frac{1}{N}\sum_{i=1}^{N} -(y_i * \log{(p_i)} + (1-y_i) * \log{(1-p_i)})$$

Where p_i is the probability of the violent class (1) and $(1-p_i)$ is the probability of the non-violent class (0)

In order to understand the previously described formulas it is mandatory to understand that:

- TP means true positive, this corresponds to the number of samples from the positive class that were classified by the model as positive. In this dissertation use-case a TP means that the model classified a violent scenario correctly.
- TN means true negative, thus, it describes the number of negative samples classified as negative. In this use-case, it describes the number of non-violent scenarios classified as such.
- FP means false positive. It corresponds to the number of negative examples classified as positive (incorrectly). Thus, in this use-case it refers to the number of non-violent scenarios classified as violent.
- FN means false negative. This corresponds to the number of examples from the positive class that were classified as negative. In this use-case it describes the number of violent scenarios classified as non-violent.

Table 2 - Model Assessment metrics

Metric	Description	Justification
Accuracy	Proportion of correctly classified records (positive or negative).	It's a very good base-line metric as well as defining very well how a model performs. It is important to have a high accuracy with a low error rate.
Recall	Proportion of positive records correctly classified as such.	It measures the capacity of the model to identify correctly true positives, in this case: violent scenarios. This metric is very important since an imbalanced dataset is

		used which describes the real world quite well.
AUC	Measure of the model's discriminatory capacity.	Metric that identifies the model's capability to differentiate the possible targets. For example, if in 100 scenarios where 80 are non-violent and 20 are violent and a random model predicts that all of them are non-violent it would have an 80% accuracy, however its capability to distinguish violent and non-violent scenarios is very low, hence the importance of this metric
Precision	Proportion of classified positive records that are true positives.	The same principle as the previous metric.
F1-score	Reliability measure of the classifier.	In this use case, recall and precision are both very important metrics. F1-score is a combination of both which allows to understand the balance between them.
Binary Cross Entropy	The binary cross entropy loss function compares the models' output probabilities to the actual real values. It penalizes the probabilities based on the distance between the output and the expected real value. Since it is a measure of "distance", this metric should be minimized.	Since this use-case is a binary classification problem, this makes it the perfect loss function for the models training, since all the probabilities lie between 0 and 1.

4.2 Data understanding

The dataset used in this master's dissertation is provided by Bosch and it was created by recording different scenarios using the camera of the target device and interpreted by paid actors. The metadata present for each audio file were manually annotated following a specific label scheme structured by Bosch.

Although this dissertation aims to study about violence detection using auditory data, the dataset has both audio and video files and the metadata is divided in six different categories, these being: *"speaker"*, *"background_noise"*, *"scene_acitivities_noises"*, *"physical_violence_noises"*, *"scene_classification"* and "silence" each of them having different labels that can be found on Table 3.

Moreover, each of these categories were described by plotting its label durations and complementing it with additional information about each attribute, such as its description, examples and important observations.

Table 3 - Dataset Labels

Type	**Possible Labels**	
speaker	Speaker events	Speaker aggressiveness
	laughing; shouting; sigh; whispering; screaming; singing; activated_speech; crying; coughing; cheering; struggle; vocalization	no_agg; agg_low; agg_medium; agg_high
background_noise	buckle_off; buckle_on; fan; door_closing; laughing; talking; coughing; shouting; music; other; door_opening; radio_voice; radio_music; dropping_something; radio_voice; radio_music; animal_noise; traffic; signal-horn; construction_work; brakes; phone_ringing; children; train; rain; wind; thunder; bicycle; car_driving; engine; window_opening; window_closing; car_trunk; car_hood; bumping_intocar; uncertain; undefined	
scene_activities_noises	anomaly_arguing; anomaly_conversation; anomaly_driving; anomaly_interaction; anomaly_violence; anomaly_uncertain; normal_no_interaction; normal_conversation; normal_arguing; normal_driving; normal_interaction; normal_radio_music; normal_uncertain	
physical_violence_noises	struggle; slapping; hitting; punching; kicking; uncertain	
scene_classification	neutral_emotion_backseat; negative_emotion_backseat; neutral_emotion_fronseat_ positivite_emotion_backseat; negative_emotion_frontseat; positive_emotion_front_seat	
silence	silence	

4.2.1 Speaker data

On the speaker set each is composed by two sub-categories: speaker aggressiveness and speaker events. For the speaker aggressiveness sub-set, each entry represents an actor, containing its seat position, gender, and the level of aggressiveness as well as the interval of time this entry occurred. On Figure 24 the duration of each label regarding the speaker aggressiveness, is demonstrated. Moreover, on Figure 25 the same described properties are shown but for the speaker events.

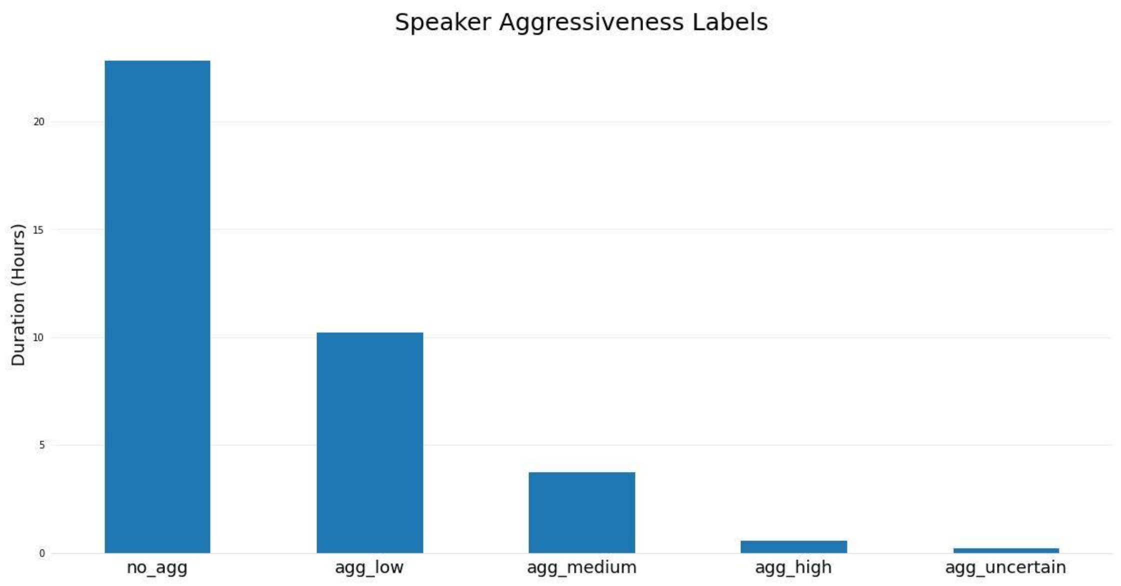


Figure 24 - Duration, in hours, of the speaker aggressiveness labels

Table 4 - Speaker aggressiveness data description

Attribute	Description	Type	Observations	Examples
actor_id	Identifies which actor is speaking	Integer	-	1; 4
gender	Identifies actors' gender	Char	-	'm' or 'f'
seat_info	Specifies what seat the actor is in	Integer	-	1; 3
aggressiveness	Identifies actors' level of aggressiveness	String	-	"no_agg", "agg_high"
time_start	Specifies when the label sarts	Float	-	0.0; 12.2

time_end	Specifies when the label ends	Float	-	17.3 ; 64.2

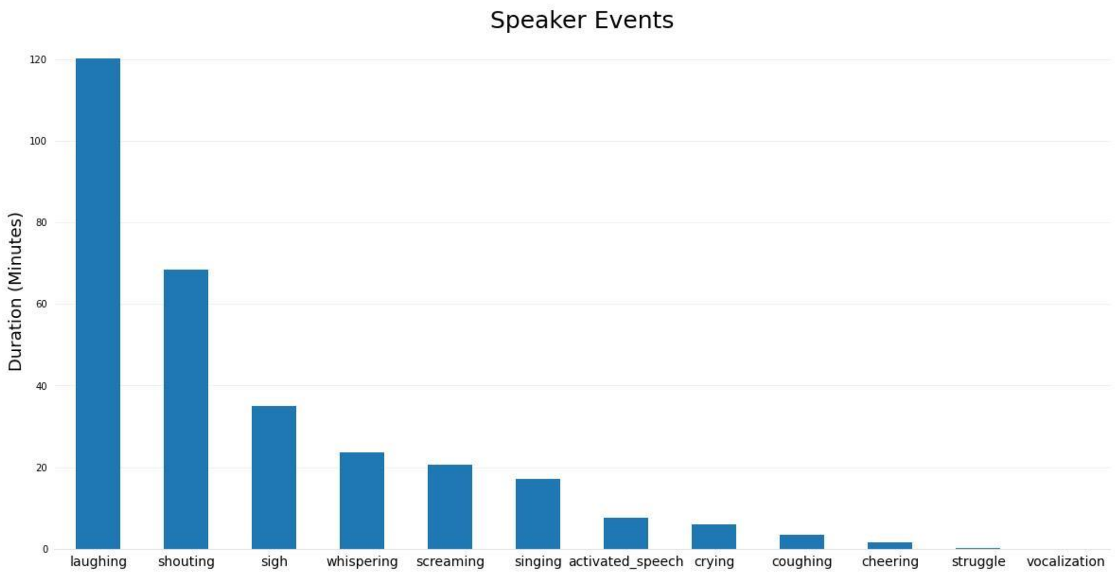


Figure 25 - Duration, in minutes, regarding the labeled speaker events

Table 5 - Speaker events data description

Attribute	Description	Type	Observations	Examples
actor_id	Identifies which actor is speaking	Integer	-	1; 3
gender	Identifies actors' gender	Char	-	'm' or 'f'
seat_info	Specifies what seat the actor is in	Integer	-	1; 4
event	Identifies actors' type of action	String	Contains 2 misplaced labels ("*struggle*" and "*vocalization*")	"*singing*"; "*laughing*"
time_start	Specifies when the label sarts	Float	-	0.0; 15.4
time_end	Specifies when the label ends	Float	-	11.7; 33.1

Although speaker data contains a lot of information regarding the level of aggressiveness of each speaker, it was not chosen as the metadata for the modeling phase. The reason behind this, is that each entry represents an actor which can create overlapping of aggressiveness labels making it difficult to understand what label represents best a scene interval which would create a lot of outliers. The main objective is to have an output regarding the overall scenario and not pinpoint which person is being labeled as aggressive, which would be extremely hard to do using only auditory data and probably result in a very inaccurate model.

4.2.2 Physical violence data

Physical violence noises describe the noise generated by a physical encounter between the actors inside the car. This would be usable to identify what is happening in a violent scenario using computer vision thus not suitable for this dissertation use case. Furthermore, a few events were misplaced such as "*fan*" and "*radio_music*" which are not considered as physical violence noises.

Table 6 - Physical violence noises data description

Attribute	Description	Type	Observations	Examples
event	Identifies what physical noise is playing	String	Multiple misplaced labels	"*slapping*", "*hitting*"
time_start	Specifies when the label sarts	Float	-	0.0; 15.4
time_end	Specifies when the label ends	Float	-	11.7; 33.1

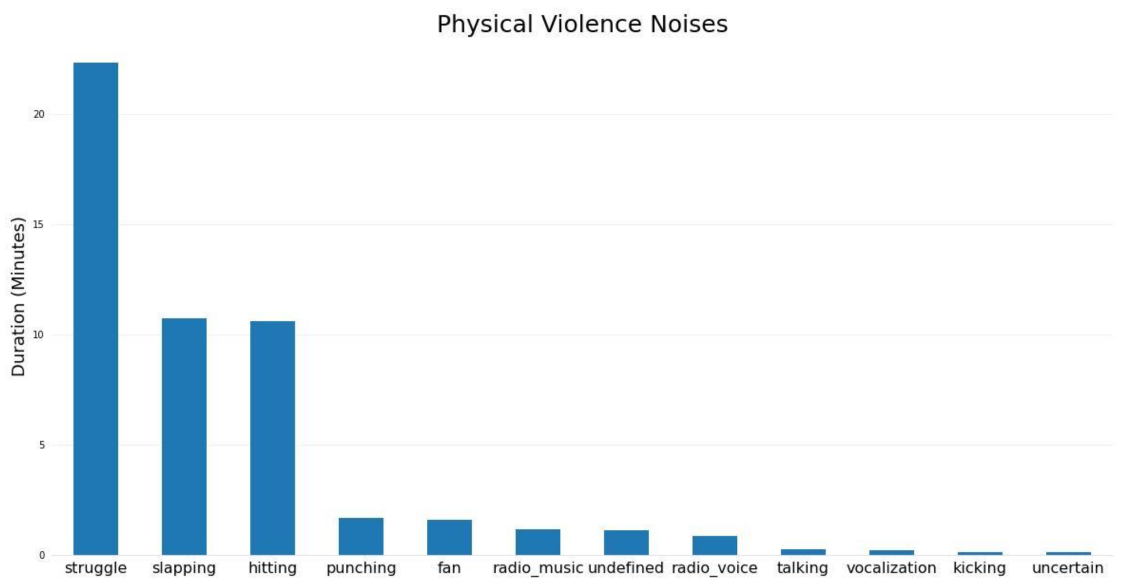


Figure 26 - Duration, in minutes, of the labeled physical violence noises

4.2.3 Background noise data

Background noise and silence labels are self-explanatory, describing the overall background noise on each time interval and where silence is present which make for a big part of this dataset. The silence set has about 2 hours of silence labels, hence it isn't plotted in this sub-section. Although the highest background noise label in terms of duration is "*undefined*" and a few misplaced labels such as "*screaming*" and "*hitting*", with the right data preparation this set could be useful in a sound event classification use case.

Table 7 - Background noises data description

Attribute	Description	Type	Observations	Examples
event	Identifies what background noise is playing	String	Multiple misplaced labels	"*fan*"; "*radio_music*"
time_start	Specifies when the label starts	Float	-	0.0; 15.4
time_end	Specifies when the label ends	Float	-	11.7; 33.1

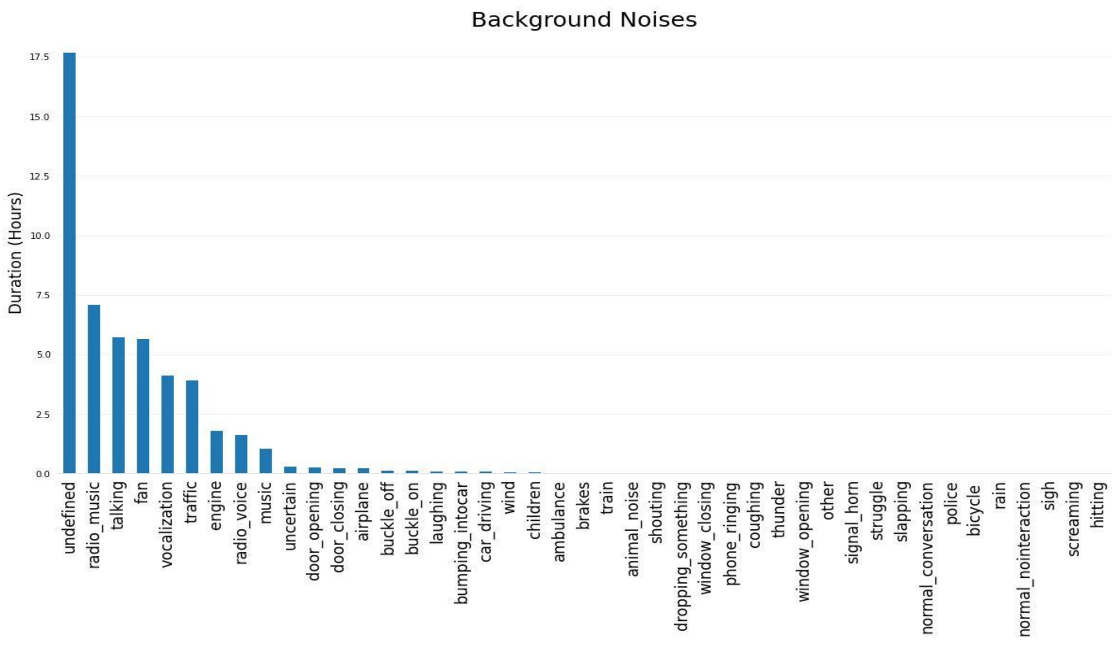


Figure 27 - Duration, in hours, regarding the background noises events

4.2.4 Scene classification data

Scene classification represents the overall emotion in a specific time interval either on the front seat or back seat. Although it could be used for this master dissertation use case it suffers from the same problem as the speaker labels, meaning that there are overlapping labels between negative and positive emotion which would create outliers for this specific problem. It would be more appropriate for an emotion classification problem when location is also a prediction variable.

Table 8 - Scene classification data description

Attribute	**Description**	**Type**	**Observations**	**Examples**
events	Identifies the overall emotion from each seat	String	Multiple misplaced labels	*"neutral_emotion_backseat"*; *"negative_emotion_frontseat"*
time_start	Specifies when the label sarts	Float	-	0.0; 15.4
time_end	Specifies when the label ends	Float	-	11.7; 33.1

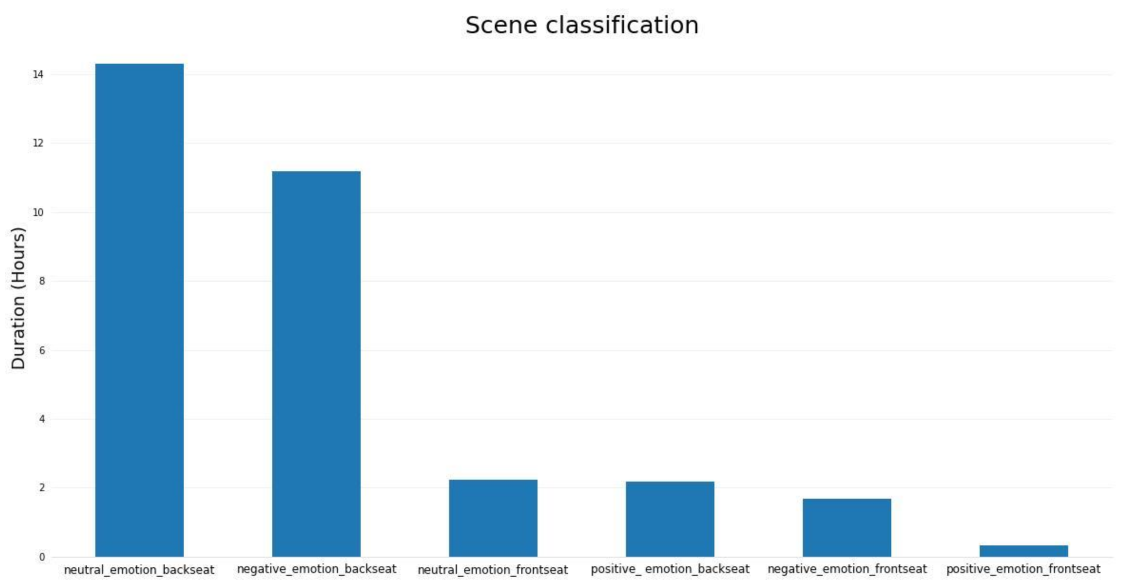


Figure 28 - Duration, in hours, regarding the emotional scene classification for each seat in the car

4.2.5 Scene activity noises data

Scene activity noises were the category chosen as the target variable for the future model. It has no overlapping intervals and generalizes each one into a label making the perfect data for a binary classification problem. It also divides into "normal" and "anomaly" labels that can be considered as nonviolent and violent respectively.

Table 9 - Scene activity noises data description

Attribute	Description	Type	Observations	Examples
event	Identifies the overall scene activity	String	A few labels with the same meaning	*"normal_conversation"*; *"anomaly_violence"*
time_start	Specifies when the event starts	Float	-	0.0; 15.4
time_end	Specifies when the event ends	Float	-	11.7; 33.1

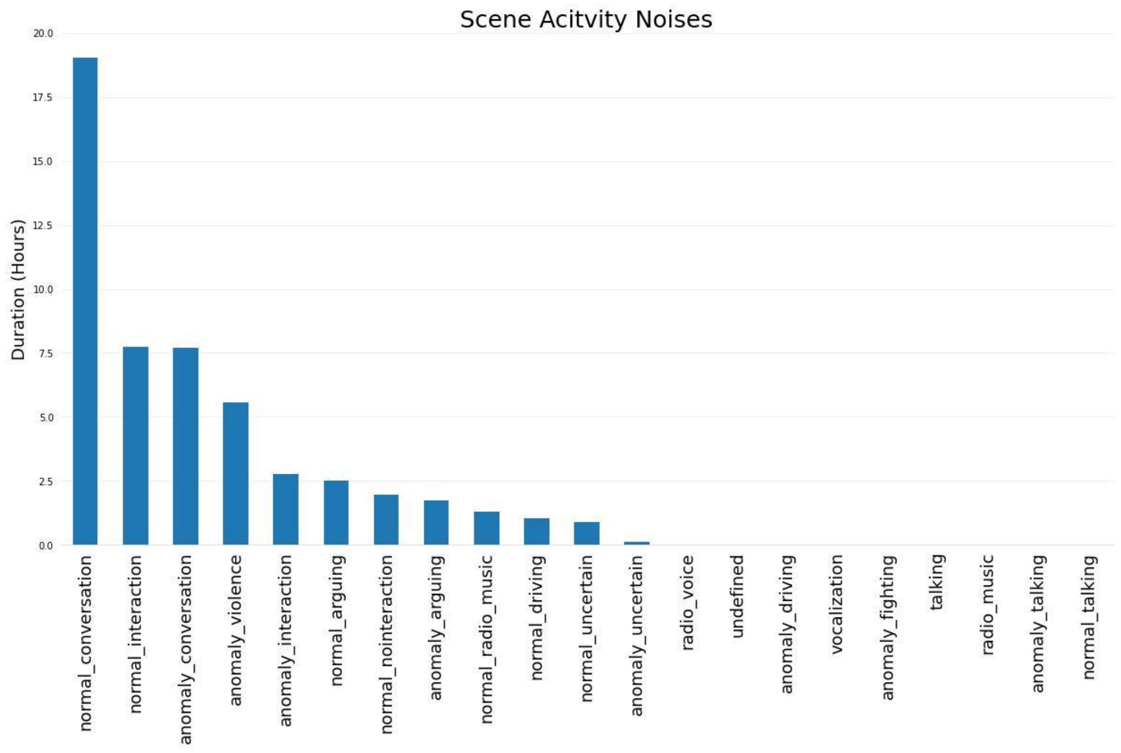


Figure 29 - Duration, in hours, regarding the scene activity labels

Since this set has been chosen in detriment of others to create the final dataset, a quality analysis was also conducted with the objective of identifying possible incoherencies and missing values on the data.

Table 10 - Scene activity noises data quality analysis

Attribute	Data type	Row Count	Missing values	Unique values	mean	std	min	max
event	String	34678	0	21	-	-	-	-
time_start	Float64	34678	0	1697	47.36	47.54	0	355.2
time_end	Float64	34678	0	3273	52.80	48.61	0.1	356.91

4.2.6 WAV Files

This dataset is composed by 2169 WAV files with a combined duration of near 52 hours. It was also necessary to conduct an analysis on sample rate since all the inputs for the models had to have the same length and dimensions. In Figure 30 is presented the sample rate count of the WAV files which shows that they had to be resampled to a fixed SR. Not only that but 3 corrupted files were found so they weren't used on the final dataset.

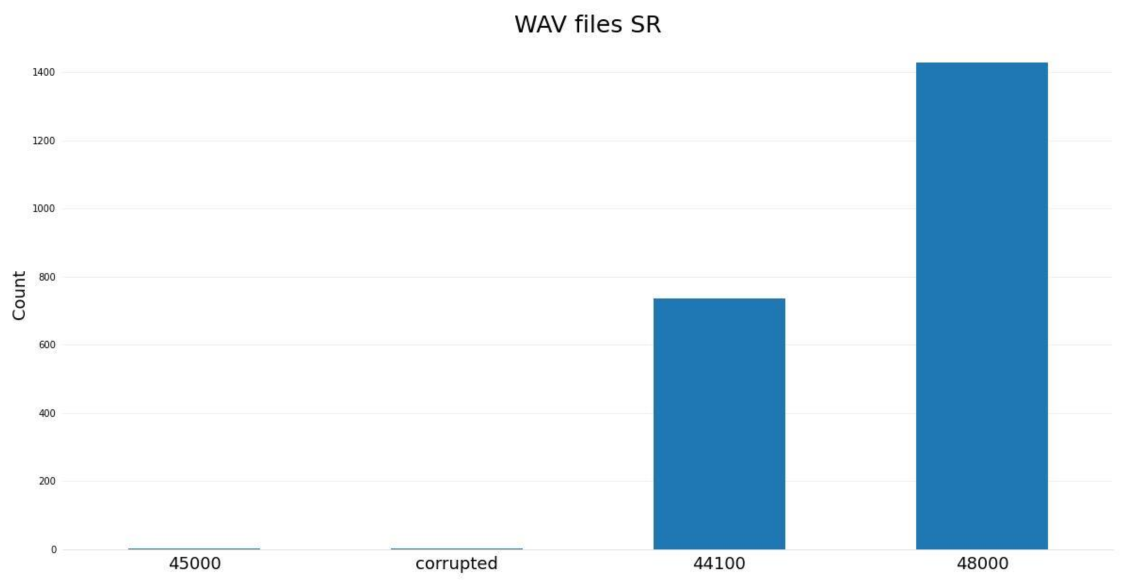


Figure 30 - WAV files sample rate plot

4.3 Data preparation

Data preparation is a very important CRISP-DM phase which aims to prepare data before it is fed to the ML model which can heavily affect performance. The most common activities are attribute selection which, in this dissertation, was done in the prior phase since another dataset will be created in the current adopted methodology phase. In a classic machine learning problem, data cleaning is a very important step to improve data quality, however, no inconsistencies were found in the scene activity noises data during the Data understanding step of the CRISP-DM methodology when it comes to missing values and misplaced labels. However, a few labels do have the same meaning which had to be mapped to existing ones.

Another way to increase data quality is to check whether the events are correctly labeled or not, unfortunately the only way to execute this, is to manually check each recording and confirm its authenticity, which would be very time consuming and not cost-effective.

In Figure 31 the data preparation pipeline is presented which effectively shows the data flow needed for this project. This pipeline is divided into metadata preparation and audio transformations. The first one is responsible for the creation of the new dataset with the violent related labels, and the latter applies all the necessary transformations to the audio files for later input to the model.

On the metadata preparation, for each metadata file, scene activity noises entries were extracted and then mapped to violent and nonviolent labels. The way that these labels were mapped is described on Table 11. The next step all the labels are transformed into fixed sized windows with a fixed time step, in this case a 3 second window with a 3 second step was chosen. This means that labels with a higher duration than 3 seconds are divided into 3 second windows and the ones with a lower duration are later padded with silence to guarantee that all the inputs have the same size. After this process is done for each metadata file, the nonviolent and violent labels are encoded to 0 and 1 respectively.

Table 11 - Map of scene activity metadata to violent labels

Labels	Mapped label	Description
"normal_*"	*non_violent*	All the labels starting with *"normal"* were mapped to *"non_violent"*
"anomaly_*"	*violent*	All the labels starting with *"anomaly"* were mapped to *"violent"*
"talking"; *"normal_talking"*; *"vocalization"*	*non_violent*	These labels are considered as *"normal_conversation"* which maps to *"non_violent"*
"radio_voice"; "radio_music";	*non_violent*	These labels are all considered as "normal_radio" which maps to *"non_violent"*
"undefined"; *"anomaly_uncertain"*; *"normal_uncertain"*; *"anomaly_driving"*; *"normal_driving"*	-	Undefined and uncertain labels were not considered and removed from the dataset. Driving labels were also removed since they do not describe the problem at hand.

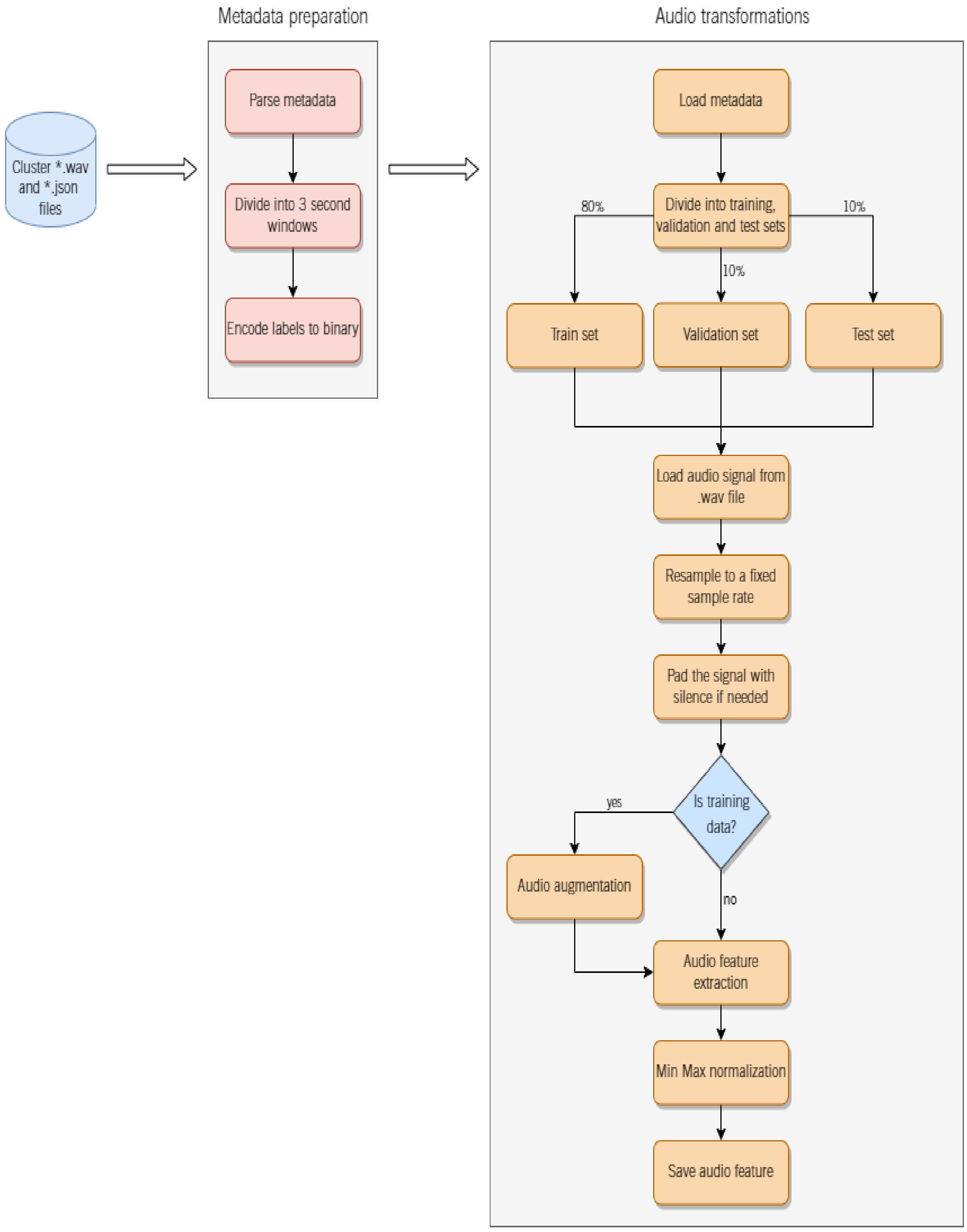


Figure 31 - Data preparation architecture

This code, that can be found in Appendix A.1, returns a new dataset containing the WAV file path, label, start and end times as well as the duration. The previously discussed dataset has a label distribution described on Figure 32.

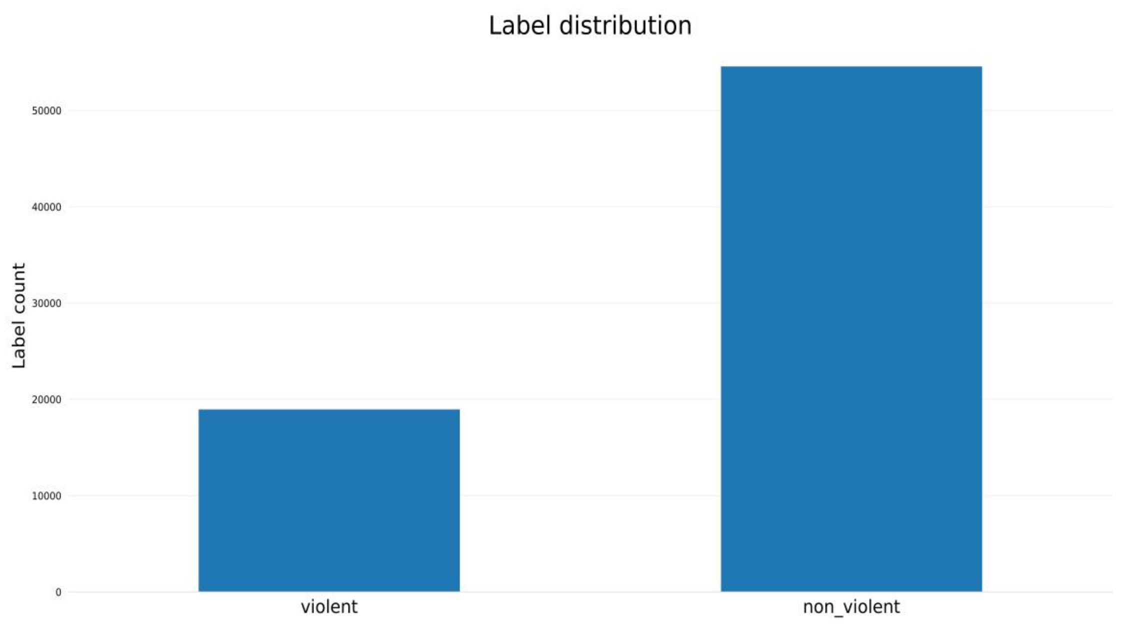


Figure 32 - Label distribution of the metadata dataset

On the audio transformation pipeline, the previously discussed dataset is split into train, validation and test sets using a very standard 80% of the data for training 10% for validation and the final 10% for testing data. For each WAV file, a window signal is loaded, resampled to 22050Hz, and then padded with zeros (silence) to correspond to a 3 second window if needed. Since the dataset used is quite imbalanced, which truly describes the real world, data augmentation is applied by either randomly time stretching the signal or pitch shifting. This is done to only 50% of the train dataset, which is divided in a stratified manner to maintain the class weights. The reason behind only training data being augmented is that artificially augmenting the validation and test sets can lead to overly confident performance which would describe poorly how the model would perform in a real word use case.

Finally, using the signal, the desired audio features are extracted, min max normalized and saved as images for later usage. In Figure 33 the final label distribution for all the extracted datasets is presented. The code used for the audio processing and data augmentation can be found in Appendix A.2.

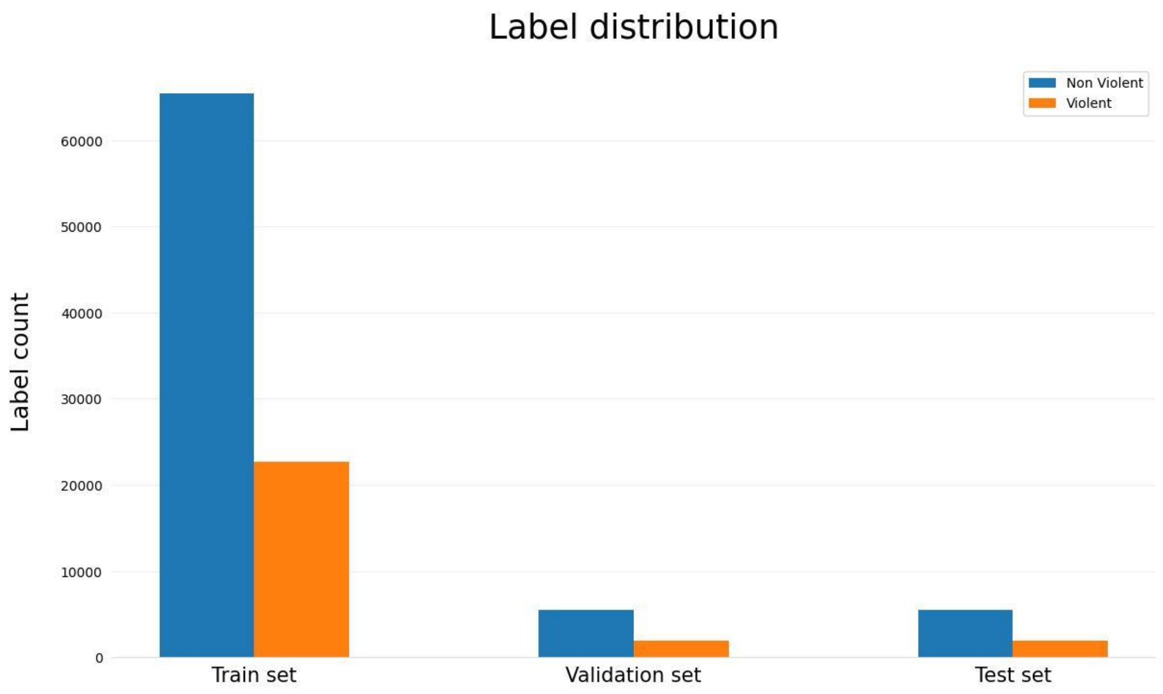


Figure 33 - Label distribution of all datasets

4.4 Modelling

Following the data preparation phase the modelling stage was executed. Here is where a set of models were selected as well as the creation of various scenarios as a way to translate the business objectives.

4.4.1 Model architecture and alterations

In this sub-section the reason behind the model's choice is explained as well as the specific architecture alterations that were required and executed to solve overfitting problems. These possible alterations can be considered regularization methods and they were also mentioned in section 2.6. However, network depth and width were concepts introduced in this sub-section since they refer specifically to network layers and neurons distribution.

In Table 12 a crossing matrix is presented between the models and the alterations done to its architectures as well as if the early stopping callback was used but an in-depth choice justification and architecture alterations were also conducted. All the models' architectures can be found in Appendix B.1 as well as its implementation code.

Table 12 - Crossing matrix between models and possible alterations

Alterations / Architecture	Reduce network depth	Reduce network width	Added dropout
VGG-16	X	X	X
MobileNet			X
ResNet-18			X

VGG-16

This model architecture was chosen due to it being a "true" CNN architecture. It is composed by 13 convolutional layers and two fully connected layers which produced a good baseline of how CNNs can perform when fed audio features as an input to solve a violence detection problem. Moreover, it has previously been shown that this architecture presents good results in an audio classification problem. In [71] Hershey got a 0.911 AUC using 70 million audio clips with 3000 labels which is very promising.

For this dissertation use case, a smaller variation of this model is used. The reason behind this is that the dataset provided by Bosch is considerably smaller and by using the regular VGG-16 network with 138 million parameters would lead to overfitting.

To avoid this problem, a narrower and shorter version of this network is implemented, meaning the number of filters and layers were changed. The last 6 convolutional layers were removed and the number of filters present on the dense layers were lowered to 256, 128, 1 in this specific order. The last activation is also altered to a Sigmoid function since it's a binary classification problem. Multiple dropout layers were also introduced as a regularization method.

MobileNet

Since the target device is a very computational limited platform and the violence detection algorithm has to be executed in very short intervals of time, an efficient neural network is needed, hence the choice behind the MobileNet architecture.

This model is designed to be used on mobile devices being super lightweight in regards of computational power requirements making it a suitable choice for the nature of this dissertation. It uses depthwise separable convolution which splits a kernel into 2 separate kernels that perform two convolutions: depthwise and pointwise convolution [72]. On the depthwise convolution, a spatial feature map of all the inputs channels is returned and then the pointwise convolution with a 1 by 1 kernel is applied in order to change its dimension which makes it way more efficient with a low impact on accuracy.

Alterations were done regarding network depth as a way to introduce the dropout regularization method. In order to achieve this the Global Average Pooling layer has been removed and five fully connected layers were added with three dropout layers with a 0.4 dropout rate. The last layer activation function was also changed to Sigmoid and the number of neurons to 1.

ResNet-18

Previous tests showed a clear early overfitting when using deeper networks thus the reason behind choosing the ResNet architecture. The state-of-art section also revealed that *NN* architectures can heavily benefit from an increasing number of layers when used alongside short connections. Moreover, in the Common approaches section the Residual Neural Network architecture showed an impressive performance regarding sound classification.

To introduce the dropout regularization method, the last fully connected layer was removed and four were added with 128, 64, 32, 1 as the number of filters in this order. Furthermore, the last activation function was changed to Sigmoid.

4.4.2 Testing scenarios

To test model performance, multiple testing scenarios were created. A scenario is comprised by a combination of input variables, models, and validation techniques. As inputs, the Constant-Q Transformations, Mel-Spectrograms, MFCCs and Chroma vector were chosen since this master dissertation literature review shows that these features are very commonly used in audio classification tasks. The Short-Fourier Transformation was also tested since all the previous features apply the STFT equation in order to be extracted, meaning that the STFT is a way faster feature to extract making it a suitable input for this use case.

Table 13 - Scenario description

Scenario	Inputs	Architectures	Validation techniques
A	Mel-Spectrograms	All	Train-Test Split
B	Constant-Q Transformation	All	Train-Test Split
C	Chroma vector	All	Train-Test Split
D	MFCCs	All	Train-Test Split
E	STFT	All	Train-Test Split

4.4.3 Model training parameterization

On Table 14 all the parameters used for each training are described. In addition to these parameters a TensorFlow callback was also used which is known as model checkpoint. Similarly, to the early stopping callback, this function allows to monitor any chosen metric which, saving model weights or the entire model when a new optimized value is achieved on an epoch-by-epoch basis. In this case, the chosen metric is the validation F1-score since the dataset is heavily unbalanced which makes it a better metric than accuracy to evaluate the models. After each model training, the optimized weights for F1-score are loaded and the architecture is evaluated on the test set. All the code for the parametrization and the callback function can be found in Appendix B.2.

Table 14 - Model training parameters and its respective values

Parameter	Value
Epochs	100
Batch size	32
Learning rate	0.005
Decay Rate	Learning rate / Epochs (=) 0.005 / 100 = 0.00005
Optimizer	SGD
Loss function	Binary Cross entropy
Class weights	{0: 0.6742, 1: 1.9357}

4.4.4 Model Assessment

After selecting the model architectures and defining the training parameters, the ANNs were trained using each scenario input and then evaluated using the respective test set. To understand model performance, the metrics defined on Success criteria subsection were registered.

It is important to note that the results presented on the following tables were obtained by using the test set and applying the model weights optimized for the highest validation F1-Score which were obtained by using TensorFlow's model checkpoint callback during training.

The training plots regarding each test scenario can be found in Appendix C.

Scenario A

Regarding scenario A, in which Mel-Spectrograms were used as each model's input, the best performing architecture was the MobileNet since it has the highest values for accuracy, precision, F1-Score and AUC and the lowest registered value for the used loss function.

Table 15 - Validation of scenario A

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**	**Binary Cross entropy**
VGG-16	0.7561	0.5190	0.7873	0.6256	0.8609	0.4736
MobileNet	0.7972	0.5810	0.7724	0.6632	0.8843	0.3733
ResNet-18	0.7936	0.5839	0.7012	0.6372	0.8670	0.4019

Table 16 - Scenario A metrics explanation

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**	**Binary Cross entropy**
VGG-16	The model correctly classified 75.61% of all the samples	This means that when the model classified a scenario as	This means that the model correctly classified	This means that the ability of the model to identify	86.09% represents the probability of the model	The model had an average of corrected probabilities

	in the test set (non-violent and violent).	violent from the test set it was correct 51.90% of the time.	78.73% of all the violent scenarios in the test set.	violent scenarios and to be accurate with those cases is 62.56%.	to distinguish violent scenarios from non-violent ones.	of 47.36%, meaning it was on average 47.36% far off from the real expected values (violent and non-violent),
MobileNet	The model correctly classified 79.72% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 58.10% of the time.	This means that the model correctly classified 77.24% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those cases is 66.32%.	88.43% represents the probability of the model to distinguish violent scenarios from non-violent ones.	The model had an average of corrected probabilities of 37.33%, meaning it was on average 37.33% far off from the real expected values (violent and non-violent),
ResNet-18	The model correctly classified 79.36% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 58.39% of the time.	This means that the model correctly classified 70.12% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those	86.70% represents the probability of the model to distinguish violent scenarios	The model had an average of corrected probabilities of 40.19%, meaning it was on average 40.19% far

				cases is 63.72%.	from non-violent ones.	off from the real expected values (violent and non-violent),

Scenario B

On scenario B, Constant-Q Transformations were used as inputs and again, the MobileNet architecture was the best performing model. It had the best metric values, losing on Recall to the VGG-16 model and having a higher loss function than the ResNet architecture.

Table 17 - Validation of scenario B

Model	Accuracy	Precision	Recall	F1-Score	AUC	Binary Cross entropy
VGG-16	0.7879	0.5681	0.7469	0.6454	0.8723	0.4352
MobileNet	0.8101	0.6120	0.7246	0.6636	0.8824	0.4147
ResNet-18	0.8074	0.6247	0.6380	0.6312	0.8737	0.3941

Table 18 - Scenario B metrics explanation

Model	Accuracy	Precision	Recall	F1-Score	AUC	Binary Cross entropy
VGG-16	The model correctly classified 78.79% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 56.81% of the time.	This means that the model correctly classified 74.69% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those	86.09% represents the probability of the model to distinguish violent scenarios	The model had an average of corrected probabilities of 43.52%, meaning it was on average 43.52% far

				cases is 64.54%.	from non-violent ones.	off from the real expected values (violent and non-violent),
MobileNet	The model correctly classified 81.01% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 61.20% of the time.	This means that the model correctly classified 72.46% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those cases is 66.36%.	88.24% represents the probability of the model to distinguish violent scenarios from non-violent ones.	The model had an average of corrected probabilities of 41.47%, meaning it was on average 41.47% far off from the real expected values (violent and non-violent),
ResNet-18	The model correctly classified 80.74% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 62.47% of the time.	This means that the model correctly classified 63.80% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those cases is 63.12%.	87.37% represents the probability of the model to distinguish violent scenarios from non-violent ones.	The model had an average of corrected probabilities of 39.41%, meaning it was on average 39.41% far off from the real expected values

						(violent and non-violent),

Scenario C

Scenario C was the worst out of all testing scenarios which shows that Chroma vectors are not suitable for this use case. Nonetheless, the MobileNet architecture proved again to be the best performer out of the 3 models, only losing in the recall metric to the VGG-16 architecture.

Table 19 - Validation of scenario C

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**	**Binary Cross entropy**
VGG-16	0.6844	0.4440	0.8767	0.5895	0.8070	0.4772
MobileNet	0.7108	0.4663	0.8246	0.5957	0.8230	0.4487
ResNet-18	0.7007	0.4556	0.8102	0.5832	0.8103	0.4725

Table 20 - Scenario C metrics explanation

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**	**Binary Cross entropy**
VGG-16	The model correctly classified 68.44% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 44.40% of the time.	This means that the model correctly classified 87.67% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those cases is 58.95%.	80.70% represents the probability of the model to distinguish violent scenarios from non-violent ones.	The model had an average of corrected probabilities of 47.72%, meaning it was on average 47.72% far off from the real expected

						values (violent and non-violent),
MobileNet	The model correctly classified 71.08% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 46.63% of the time.	This means that the model correctly classified 82.46% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those cases is 59.57%.	82.30% represents the probability of the model to distinguish violent scenarios from non-violent ones.	The model had an average of corrected probabilities of 44.87%, meaning it was on average 44.87% far off from the real expected values (violent and non-violent),
ResNet-18	The model correctly classified 70.07% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 45.56% of the time.	This means that the model correctly classified 81.02% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those cases is 58.32%.	81.03% represents the probability of the model to distinguish violent scenarios from non-violent ones.	The model had an average of corrected probabilities of 47.25%, meaning it was on average 47.25% far off from the real expected values (violent and non-violent),

Scenario D

The MFCC scenario was clearly the best audio feature out of all testing scenarios concluding in a very close call between all the models. In this scenario the VGG-16 wins in both accuracy and precision, MobileNet has better AUC and loss function values and lastly ResNet has the highest recall and F1-score. Since the difference on metric values is quite low, the inference time was considered in order to decide which model performs best in which MobileNet performs best against the other two architectures.

Table 21 - Validation of scenario D

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**	**Binary Cross entropy**
VGG-16	0.8160	0.6219	0.7352	0.6738	0.8906	0.3723
MobileNet	0.8142	0.6174	0.7395	0.6730	0.8902	0.3625
ResNet-18	0.8103	0.6052	0.7645	0.6756	0.8901	0.3872

Table 22 - Scenario D metrics explanation

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**	**Binary Cross entropy**
VGG-16	The model correctly classified 81.60% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 62.19% of the time.	This means that the model correctly classified 73.52% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those cases is 67.38%.	89.06% represents the probability of the model to distinguish violent scenarios from non-violent ones.	The model had an average of corrected probabilities of 37.23%, meaning it was on average 37.23% far off from the real expected values

						(violent and non-violent),
MobileNet	The model correctly classified 81.42% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 61.74% of the time.	This means that the model correctly classified 73.95% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those cases is 67.30%.	89.02% represents the probability of the model to distinguish violent scenarios from non-violent ones.	The model had an average of corrected probabilities of 36.25%, meaning it was on average 36.25% far off from the real expected values (violent and non-violent),
ResNet-18	The model correctly classified 81.03% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 60.52% of the time.	This means that the model correctly classified 76.45% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those cases is 67.56%.	89.01% represents the probability of the model to distinguish violent scenarios from non-violent ones.	The model had an average of corrected probabilities of 38.72%, meaning it was on average 38.72% far off from the real expected values (violent and non-violent),

Scenario E

For the last scenario, the STFT audio feature was used, and the registered metrics show that, again, the MobileNet architecture outperformed the other two models having better results for recall, F1-Score, AUC and loss function values.

Table 23 - Validation of scenario E

Model	Accuracy	Precision	Recall	F1-Score	AUC	Binary Cross entropy
VGG-16	0.8013	0.5911	0.7501	0.6612	0.8834	0.4176
MobileNet	0.8027	0.5897	0.7778	0.6708	0.8891	0.3735
ResNet-18	0.8101	0.6151	0.7087	0.6586	0.8810	0.3745

Table 24 - Scenario E metrics explanation

Model	Accuracy	Precision	Recall	F1-Score	AUC	Binary Cross entropy
VGG-16	The model correctly classified 80.13% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 59.11% of the time.	This means that the model correctly classified 75.01% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those cases is 66.12%.	88.34% represents the probability of the model to distinguish violent scenarios from non-violent ones.	The model had an average of corrected probabilities of 41.76%, meaning it was on average 41.76% far off from the real expected values (violent and non-violent),
MobileNet	The model correctly	This means that when	This means that the	This means that the	88.91% represents	The model had an

	classified 80.27% of all the samples in the test set (non-violent and violent).	the model classified a scenario as violent from the test set it was correct 58.97% of the time.	model correctly classified 77.78% of all the violent scenarios in the test set.	ability of the model to identify violent scenarios and to be accurate with those cases is 67.08%.	the probability of the model to distinguish violent scenarios from non-violent ones.	average of corrected probabilities of 37.35%, meaning it was on average 37.35% far off from the real expected values (violent and non-violent),
ResNet-18	The model correctly classified 81.01% of all the samples in the test set (non-violent and violent).	This means that when the model classified a scenario as violent from the test set it was correct 61.51% of the time.	This means that the model correctly classified 70.87% of all the violent scenarios in the test set.	This means that the ability of the model to identify violent scenarios and to be accurate with those cases is 65.86%.	88.10% represents the probability of the model to distinguish violent scenarios from non-violent ones.	The model had an average of corrected probabilities of 37.45%, meaning it was on average 37.45% far off from the real expected values (violent and non-violent),

4.5 Evaluation

The evaluation phase from the CRISP-DM methodology is one of the most important stages of the project since it aggregates all the models and scenarios as well as the obtained results. This type of analysis generates a greater view on what model performs best regarding this master dissertation use case, meaning the most suitable model for in-car violence detection only using auditory data with the highest predictive capability was extracted from this step.

Table 25 - Table with the best model results per scenario

Metric / Scenario	Accuracy	Precision	Recall	F1-Score	AUC	Binary Cross entropy	Model
A	0.7972	0.5810	0.7724	0.6632	0.8843	0.3733	MobileNet
B	0.8101	0.6120	0.7246	0.6636	0.8824	0.4147	MobileNet
C	0.7108	0.4663	0.8246	0.5957	0.8230	0.4487	MobileNet
D	0.8142	0.6174	0.7395	0.6730	0.8902	0.3625	MobileNet
E	0.8027	0.5897	0.7778	0.6708	0.8891	0.3735	MobileNet

The Table 25 is created by extracting the best model per scenario which are explained on the Model Assessment sub-phase. All the scenarios share the same best model, which means the MobileNet architecture is the one which will be used for the CRISP-DM's deployment phase.

Scenario D achieved the highest values when it comes to the success criteria which allows to conclude that from all the audio features tested, the MFCC audio feature is the best data to use as an input for in-car violence detection. In other words, the MobileNet architecture and MFCC image representation should be the chosen model architecture and audio feature, respectively, for the following phase of the CRISP-DM methodology.

4.6 Deployment

Although it was anticipated that the proposed architecture would be ported to a target device the results did not meet Bosch's success criteria. Moreover, due to time constraints it was impossible to perform real-life experiments using the final model in the RideCare use-case context. Despite this fact, the present work presented valid contributions when it comes to understanding how these architectures work and how well they perform on violence detection, using audio data (in its visual representation form) related tasks.

In the context of the RideCare project, the deployment phase would correspond to the integration of the extracted model on the evaluation phase on the target device. This implies that a similar but new pipeline would have to be integrated as well, meaning transforming the audio signals into its MFCC representation and then feed it to the trained MobileNet architecture which in turn would predict whether it represents a violent scenario or not. Finally, this pipeline would then alert the mobility service provider of the incident which allows the person in charge to take action, guaranteeing a security improvement of all parties involved.

5. CONCLUSIONS

This last chapter presents an overview of all the work done with a special focus on the extracted conclusions from its practical implementation. Therefore, it is divided into three main sections, the first one being a summary of all the work done throughout this dissertation, the second aims to evaluate all the contributions of this master's dissertation by comparing the previously defined objectives to the completed ones and finally, the third section describes these projects limitations as well as future work that can be developed to improve it.

5.1 Work summary

Deep Learning has been used in various subjects such as image recognition, fraud detection, natural language processing and audio classification. This dissertation aimed to develop a DL architecture that can detect violent scenarios based on auditory data. It also contributes to a better understanding on if audio inputs in combination with a Deep Neural Network is enough to detect in-car violent scenarios which has never been researched before.

Initially, in the Introduction section, the problem was contextualized in order to identify the research gap and what the project requirements are. The objectives and expected result are also discussed in this phase. A work plan was also defined in order to specify the different project milestones and to temporally organize the different tasks. Additionally, the specified tasks were also described. This work plan was a visual reference designed for an efficient implementation of this project, guarantying an easier time allocation to reach the project requirements.

A literature review was also conducted in order to describe and explain the various concepts associated with the task at hand, framing the issue in a conceptual form with the explaining of Deep Learning, Artificial Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks, Residual Neural Networks and the different representations of audio features, alongside its possible augmentations for future use on the DL architecture. Artificial Neural Networks and its variants were the focus of this chapter since it was the machine learning variant associated to the practical implementation of this master dissertation. In this chapter, regularization methods were also discussed since they are a very important option to explore on the practical development of this dissertation in order to minimize overfitting and improve model generalization, helping with predictions on real-word and unseen data.

Finally, a deep dive in related work and common approaches was taken, which was extremely important since it lighted the path on how these networks are implemented and how can they perform on acoustic event detection and sound classification, which highly influenced the final solution architecture.

In order to lower the project complexity and organize it in different steps, the methodologic approach CRISP-DM was also discussed. Furthermore, the reason behind using Python is explained alongside the libraries and tools that were used for the practical development of this master dissertation.

In chapter 4 the use case was contextualized as well as the practical execution of the CRISP-DM methodology, assuring that every requirement is met. Here the business understanding is described and all the steps for the meta and audio data extraction are represented. Finally, in the end of this chapter the results are presented as well as a comparing table and the best performing model is selected.

5.2 Contributions

Concluding the practical work of this dissertation it is possible to summarize the current situation status regarding the initially defined objectives and the actual acquired results. It is safe to say that most of the expectations were met and that the present work fulfilled with the main goal of developing a deep learning-based model that is able to classify violent scenarios using only auditory data with more than 80% accuracy.

Although it was expected that the model will be implemented on a target device, this requirement has not been met due to the results not meeting Bosch’s success criteria. As per the criteria, in the given use case, a model should be capable of near perfect violent scenario detections, but the solution architecture doesn’t allow that, given that it uses audio features which makes it very sensitive to frequency and amplitude and not semantics or even physical altercations. Another pre-defined objective was the development of a scientific article which was not met due to time constraints and the need to allocate more time resources to other tasks such as implementation and testing of scenarios.

For the violence detection using auditory data domain, the state-of-the-art review allowed to understand that deep learning solutions are very uncommon approach for this problem, making this investigation a baseline for future ones. The practical development of this dissertation also contributed in a scientific way, allowing a higher understanding of how well a deep learning architecture, while using audio features as inputs, can perform on a violence classification task. Not only that, but the MFCC testing scenario showed the highest metric values which implies that it is the best option for this use case.

5.3 Limitations and future work

The solution presented in this Master's dissertation and the results obtained alongside it, can be considered a solid baseline for a future machine learning implementation on the mobile service industry specifically for in-car violence detection. Nonetheless, there are a few limitations and aspects that should be considered as future work which will be explained in this section.

The presented solution, considering it's a deep learning approach, is computationally inexpensive. On the other hand, the usage of audio feature representations as input makes it highly dependent on frequency and amplitude, which means that in a situation where a violent discussion or altercation is in place and these two variables are not very much present the model could classify the scenario incorrectly. This opens a possible new approach, by combining a speech to text model and feeding its outputs to a DL architecture capable of text classification.

Furthermore, for the practical implementation of this master dissertation a dataset was needed to be created from the provided one, potentially introducing data noise which can heavily influence the models' classification capability. In addition, for deep learning standards the number of samples was rather low. Hence, in the future, a more suitable and increasing set of data could possibly improve the models results, in this context or in others.

In the practical development of this investigation, only three deep learning model architectures were tested, meaning that in the future more variations of Neural Networks should be tested as well as the inclusion of more audio features. Furthermore, audio features can also be stacked, meaning multiple inputs can be fed to the models. This means that feature selection algorithms could also be used in order to improve the final architecture performance.

Finally, the solution should be deployed and followed by a business validation. This means that the developed prototype should be first tested in a controlled environment where an analysis should be conducted in order to understand the solution classification capability against real world data, its ability to learn from the said information and its computational performance in real-time.

BIBLIOGRAPHY

[1] L. Cai, J. Gao, and D. Zhao, “A review of the application of deep learning in medical image classification and segmentation,” *Ann Transl Med*, vol. 8, no. 11, p. 713, Jun. 2020, doi: 10.21037/atm.2020.02.44.

[2] M. Daily, S. Medasani, R. Behringer, and M. Trivedi, “Self-Driving Cars,” *Computer*, vol. 50, no. 12, pp. 18–23, Dec. 2017, doi: 10.1109/MC.2017.4451204.

[3] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, “Object Detection With Deep Learning: A Review,” *IEEE Trans. Neural Netw. Learning Syst.*, vol. 30, no. 11, pp. 3212–3232, Nov. 2019, doi: 10.1109/TNNLS.2018.2876865.

[4] S. Balaban, “Deep learning and face recognition: the state of the art,” Baltimore, Maryland, United States, May 2015, p. 94570B. doi: 10.1117/12.2181526.

[5] A. Sheth, “Internet of Things to Smart IoT Through Semantic, Cognitive, and Perceptual Computing,” *IEEE Intell. Syst.*, vol. 31, no. 2, pp. 108–112, Mar. 2016, doi: 10.1109/MIS.2016.34.

[6] I. Goodfellow and Y. Bengio, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[7] A. Esteva *et al.*, “A guide to deep learning in healthcare,” *Nat Med*, vol. 25, no. 1, pp. 24–29, Jan. 2019, doi: 10.1038/s41591-018-0316-z.

[8] B. J. Wythoff, “Backpropagation neural networks,” *Chemometrics and Intelligent Laboratory Systems*, vol. 18, no. 2, pp. 115–155, Feb. 1993, doi: 10.1016/0169-7439(93)80052-J.

[9] S. Sharma, S. Sharma, and A. Athaiya, “ACTIVATION FUNCTIONS IN NEURAL NETWORKS,” *IJEAST*, vol. 04, no. 12, pp. 310–316, May 2020, doi: 10.33564/IJEAST.2020.v04i12.054.

[10] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation Functions: Comparison of trends in Practice and Research for Deep Learning,” 2018, doi: 10.48550/ARXIV.1811.03378.

[11] B. Karlik and A. V. Olgac, “Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks,” *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[12] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for Activation Functions,” 2017, doi: 10.48550/ARXIV.1710.05941.

[13] S. Sharma, “Activation Functions in Neural Networks,” *https://towardsdatascience.com/*. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (accessed Mar. 25, 2022).

[14] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging*, vol. 9, no. 4, pp. 611–629, Aug. 2018, doi: 10.1007/s13244-018-0639-9.

[15] M. Pak and S. Kim, “A review of deep learning in image recognition,” in *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, Kuta Bali, Aug. 2017, pp. 1–3. doi: 10.1109/CAIPT.2017.8320684.

[16] K. O’Shea and R. Nash, “An Introduction to Convolutional Neural Networks,” 2015, doi: 10.48550/ARXIV.1511.08458.

[17] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, “Recent Advances in Recurrent Neural Networks.” arXiv, Feb. 22, 2018. Accessed: May 22, 2022. [Online]. Available: http://arxiv.org/abs/1801.01078

[18] W. Yin, K. Kann, M. Yu, and H. Schütze, “Comparative Study of CNN and RNN for Natural Language Processing.” arXiv, Feb. 07, 2017. Accessed: May 22, 2022. [Online]. Available: http://arxiv.org/abs/1702.01923

[19] A. Graves, A. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, BC, Canada, May 2013, pp. 6645–6649. doi: 10.1109/ICASSP.2013.6638947.

[20] K. Choi, G. Fazekas, M. Sandler, and K. Cho, “Convolutional recurrent neural networks for music classification,” in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, LA, Mar. 2017, pp. 2392–2396. doi: 10.1109/ICASSP.2017.7952585.

[21] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A Search Space Odyssey,” *IEEE Trans. Neural Netw. Learning Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017, doi: 10.1109/TNNLS.2016.2582924.

[22] S. Siami-Namini, N. Tavakoli, and A. Siami Namin, “A Comparison of ARIMA and LSTM in Forecasting Time Series,” in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Orlando, FL, Dec. 2018, pp. 1394–1401. doi: 10.1109/ICMLA.2018.00227.

[23] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition.” arXiv, Dec. 10, 2015. Accessed: Sep. 28, 2022. [Online]. Available: http://arxiv.org/abs/1512.03385

[24] M. Q. Pham, B. Oudompheng, J. I. Mars, and B. Nicolas, “A Noise-Robust Method with Smoothed ℓ 1 / ℓ 2 Regularization for Sparse Moving-Source Mapping,” *Signal Processing*, vol. 135, pp. 96–106, Jun. 2017, doi: 10.1016/j.sigpro.2016.12.022.

[25] B. Bilgic *et al.*, “Fast image reconstruction with L2-regularization: Fast Reconstruction With L2-Regularization,” *J. Magn. Reson. Imaging*, vol. 40, no. 1, pp. 181–191, Jul. 2014, doi: 10.1002/jmri.24365.

[26] O. Demir-Kavuk, M. Kamada, T. Akutsu, and E.-W. Knapp, “Prediction using step-wise L1, L2 regularization and feature selection for small data sets with large number of features,” *BMC Bioinformatics*, vol. 12, no. 1, p. 412, Dec. 2011, doi: 10.1186/1471-2105-12-412.

[27] H. Soumare, A. Benkahla, and N. Gmati, “Deep learning regularization techniques to genomics data,” *Array*, vol. 11, p. 100068, Sep. 2021, doi: 10.1016/j.array.2021.100068.

[28] N. Tripuraneni, B. Adlam, and J. Pennington, “Covariate Shift in High-Dimensional Random Feature Regression,” *arXiv:2111.08234 [cs, stat]*, Nov. 2021, Accessed: May 10, 2022. [Online]. Available: http://arxiv.org/abs/2111.08234

[29] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *arXiv:1502.03167 [cs]*, Mar. 2015, Accessed: May 10, 2022. [Online]. Available: http://arxiv.org/abs/1502.03167

[30] Guodong Guo and S. Z. Li, “Content-based audio classification and retrieval by support vector machines,” *IEEE Trans. Neural Netw.*, vol. 14, no. 1, pp. 209–215, Jan. 2003, doi: 10.1109/TNN.2002.806626.

[31] F. Rong, “Audio Classification Method Based on Machine Learning,” in *2016 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, Changsha, China, Dec. 2016, pp. 81–84. doi: 10.1109/ICITBS.2016.98.

[32] S. Cunningham, H. Ridley, J. Weinel, and R. Picking, “Supervised machine learning for audio emotion recognition: Enhancing film sound design using audio features, regression models and artificial neural networks,” *Pers Ubiquit Comput*, vol. 25, no. 4, pp. 637–650, Aug. 2021, doi: 10.1007/s00779-020-01389-0.

[33] T. Petri, “Exploring relationships between audio features and emotion in music,” *Front. Hum. Neurosci.*, vol. 3, 2009, doi: 10.3389/conf.neuro.09.2009.02.033.

[34] B. Rajoub, “Characterization of biomedical signals: Feature engineering and extraction,” in *Biomedical Signal Processing and Artificial Intelligence in Healthcare*, Elsevier, 2020, pp. 29–50. doi: 10.1016/B978-0-12-818946-7.00002-0.

[35] N. Kehtarnavaz, “Frequency Domain Processing,” in *Digital Signal Processing System Design*, Elsevier, 2008, pp. 175–196. doi: 10.1016/B978-0-12-374490-6.00007-6.

[36] B. Logan, “Mel Frequency Cepstral Coefficients for Music Modelling,” *Proc. 1st Int. Symposium Music Information Retrieval*, 2000.

[37] R. Hasan, M. Jamil, G. Rabbani, and S. Rahman, “Speaker Indentification Using Mel Frequency Cepstral Coefficients,” *Proceedings of the 3rd International Conference on Eletrical and Computer Engineering (ICECE 2004)*, Dec. 2004.

[38] L. Muda, M. Begam, and I. Elamvazuthi, “Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques.” arXiv, Mar. 22, 2010. Accessed: May 21, 2022. [Online]. Available: http://arxiv.org/abs/1003.4083

[39] M. Henaff, K. Jarrett, K. Kavukcuoglu, and Y. LeCun, “UNSUPERVISED LEARNING OF SPARSE FEATURES FOR SCALABLE AUDIO CLASSIFICATION,” *ISMIR*, p. 6, 2011.

[40] J. C. Brown, “Calculation of a constant *Q* spectral transform,” *The Journal of the Acoustical Society of America*, vol. 89, no. 1, pp. 425–434, Jan. 1991, doi: 10.1121/1.400476.

[41] M. A. Bartsch and G. H. Wakefield, “Audio thumbnailing of popular music using chroma-based representations,” *IEEE Trans. Multimedia*, vol. 7, no. 1, pp. 96–104, Feb. 2005, doi: 10.1109/TMM.2004.840597.

[42] D. P. W. Ellis and G. E. Poliner, “Identifying `Cover Songs’ with Chroma Features and Dynamic Programming Beat Tracking,” in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP ’07*, Honolulu, HI, Apr. 2007, p. IV-1429-IV–1432. doi: 10.1109/ICASSP.2007.367348.

[43] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *J Big Data*, vol. 6, no. 1, p. 60, Dec. 2019, doi: 10.1186/s40537-019-0197-0.

[44] S. Wei, S. Zou, F. Liao, and weimin lang, “A Comparison on Data Augmentation Methods Based on Deep Learning for Audio Classification,” *J. Phys.: Conf. Ser.*, vol. 1453, no. 1, p. 012085, Jan. 2020, doi: 10.1088/1742-6596/1453/1/012085.

[45] D. S. Park *et al.*, “SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition,” 2019, doi: 10.48550/ARXIV.1904.08779.

[46] T. Heittola, A. Mesaros, T. Virtanen, and A. Eronen, “Sound Event Detection in Multisource Environments Using Source Separation,” *Machine Listening in Multisource Environments*, p. 5, 2011.

[47] T. K. Chan and C. S. Chin, “A Comprehensive Review of Polyphonic Sound Event Detection,” *IEEE Access*, vol. 8, pp. 103339–103373, 2020, doi: 10.1109/ACCESS.2020.2999388.

[48] J. Salamon, C. Jacoby, and J. P. Bello, “A Dataset and Taxonomy for Urban Sound Research,” in *Proceedings of the 22nd ACM international conference on Multimedia*, Orlando Florida USA, Nov. 2014, pp. 1041–1044. doi: 10.1145/2647868.2655045.

[49] T. Zhang and C.-C. J. Kuo, “Audio content analysis for online audiovisual data segmentation and classification,” *IEEE Trans. Speech Audio Process.*, vol. 9, no. 4, pp. 441–457, May 2001, doi: 10.1109/89.917689.

[50] S. Chu, S. Narayanan, and C.-C. J. Kuo, “Environmental Sound Recognition With Time–Frequency Audio Features,” *IEEE Trans. Audio Speech Lang. Process.*, vol. 17, no. 6, pp. 1142–1158, Aug. 2009, doi: 10.1109/TASL.2009.2017438.

[51] M. Papakostas *et al.*, “Deep Visual Attributes vs. Hand-Crafted Audio Features on Multidomain Speech Emotion Recognition,” *Computation*, vol. 5, no. 4, p. 26, Jun. 2017, doi: 10.3390/computation5020026.

[52] J. F. Gemmeke, L. Vuegen, P. Karsmakers, B. Vanrumste, and H. Van hamme, “An exemplar-based NMF approach to audio event detection,” in *2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, NY, USA, Oct. 2013, pp. 1–4. doi: 10.1109/WASPAA.2013.6701847.

[53] P. Khunarsal, C. Lursinsap, and T. Raicharoen, “Very short time environmental sound classification based on spectrogram pattern matching,” *Information Sciences*, vol. 243, pp. 57–74, Sep. 2013, doi: 10.1016/j.ins.2013.04.014.

[54] Z. Kons and O. Toledo-Ronen, “Audio event classification using deep neural networks,” in *Interspeech 2013*, Aug. 2013, pp. 1482–1486. doi: 10.21437/Interspeech.2013-384.

[55] J. Salamon and J. P. Bello, “Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification,” *IEEE Signal Process. Lett.*, vol. 24, no. 3, pp. 279–283, Mar. 2017, doi: 10.1109/LSP.2017.2657381.

[56] K. J. Piczak, “Environmental sound classification with convolutional neural networks,” in *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, Boston, MA, USA, Sep. 2015, pp. 1–6. doi: 10.1109/MLSP.2015.7324337.

[57] X. Zhang, Y. Zou, and W. Shi, “Dilated convolution neural network with LeakyReLU for environmental sound classification,” in *2017 22nd International Conference on Digital Signal Processing (DSP)*, London, Aug. 2017, pp. 1–5. doi: 10.1109/ICDSP.2017.8096153.

[58] K. Palanisamy, D. Singhania, and A. Yao, “Rethinking CNN Models for Audio Classification,” 2020, doi: 10.48550/ARXIV.2007.11154.

[59] R. Müller, F. Ritz, S. Illium, and C. Linnhoff-Popien, “Acoustic Anomaly Detection for Machine Sounds based on Image Transfer Learning,” 2020, doi: 10.48550/ARXIV.2006.03429.

[60] M. Ramzan *et al.*, “A Review on State-of-the-Art Violence Detection Techniques,” *IEEE Access*, vol. 7, pp. 107560–107575, 2019, doi: 10.1109/ACCESS.2019.2932114.

[61] T. Hassner, Y. Itcher, and O. Kliper-Gross, “Violent flows: Real-time detection of violent crowd behavior,” in *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, Providence, RI, USA, Jun. 2012, pp. 1–6. doi: 10.1109/CVPRW.2012.6239348.

[62] V. Mahadevan, W. Li, V. Bhalodia, and N. Vasconcelos, “Anomaly detection in crowded scenes,” in *2010 IEEE Computer Society Conference on Computer Vision and Pattern*

Recognition, San Francisco, CA, USA, Jun. 2010, pp. 1975–1981. doi: 10.1109/CVPR.2010.5539872.

[63] A.-M. R. Abdali and R. F. Al-Tuma, “Robust Real-Time Violence Detection in Video Using CNN And LSTM,” in *2019 2nd Scientific Conference of Computer Sciences (SCCS)*, Baghdad, Iraq, Mar. 2019, pp. 104–108. doi: 10.1109/SCCS.2019.8852616.

[64] T. Giannakopoulos, D. Kosmopoulos, A. Aristidou, and S. Theodoridis, “Violence Content Classification Using Audio Features,” in *Advances in Artificial Intelligence*, vol. 3955, G. Antoniou, G. Potamias, C. Spyropoulos, and D. Plexousakis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 502–507. doi: 10.1007/11752912_55.

[65] T. Giannakopoulos, A. Pikrakis, and S. Theodoridis, “A Multi-Class Audio Classification Method With Respect To Violent Content In Movies Using Bayesian Networks,” in *2007 IEEE 9th Workshop on Multimedia Signal Processing*, Chania, Crete, Greece, 2007, pp. 90–93. doi: 10.1109/MMSP.2007.4412825.

[66] M. Meire and P. Karsmakers, “Comparison of Deep Autoencoder Architectures for Real-time Acoustic Based Anomaly Detection in Assets,” in *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Metz, France, Sep. 2019, pp. 786–790. doi: 10.1109/IDAACS.2019.8924301.

[67] P. Becker, C. Roth, A. Roennau, and R. Dillmann, “Acoustic Anomaly Detection in Additive Manufacturing with Long Short-Term Memory Neural Networks,” in *2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA)*, Bangkok, Thailand, Apr. 2020, pp. 921–926. doi: 10.1109/ICIEA49774.2020.9102002.

[68] E. Rushe and B. M. Namee, “Anomaly Detection in Raw Audio Using Deep Autoregressive Networks,” in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, United Kingdom, May 2019, pp. 3597–3601. doi: 10.1109/ICASSP.2019.8683414.

[69] A. Azevedo and M. Santos, “KDD, semma and CRISP-DM: A parallel overview,” Jun. 2008.

[70] R. Wirth and H. Jochen, “CRISP-DM: Towards a standard process model for data mining,” *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, pp. 29–39, 2000.

[71] S. Hershey *et al.*, “CNN architectures for large-scale audio classification,” in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, LA, Mar. 2017, pp. 131–135. doi: 10.1109/ICASSP.2017.7952132.

[72] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” 2017, doi: 10.48550/ARXIV.1704.04861.

Appendix A

Appendix A.1 Code used for the metadata preparation

```
VIOLENT_LABELS = ["anomaly_arguing", "anomaly_conversation",
"anomaly_fighting", "anomaly_interaction", "anomaly_talking",
"anomaly_violence"]
NON_VIOLENT_LABELS = ["normal_arguing", "normal_conversation",
"normal_interaction", "normal_nointeraction", "normal_talking",
                        "normal_radio_music","radio_music", "talking"]


def get_label(label):
    if label in VIOLENT_LABELS:
        return "violent"
    elif label in NON_VIOLENT_LABELS:
        return "non_violent"
    else:
        return None

def load_json(file_path):
    with open(file_path, "r") as f:
        data = json.load(f)
        if "scene_activities_noises" in data['audio_metadata']:
            return data["audio_metadata"]["scene_activities_noises"]
        else:
            return []

def get_data(json_path, wav_path):
    scene_data = load_json(json_path)

    data = {
        "File_path" : [],
        "Time_start" : [],
        "Time_end": [],
        "Duration": [],
        "Label" : [],

    }
    for scene_noise in scene_data:
        if "event" in scene_noise:
            label = get_label(scene_noise["event"])
            if label == None:
                continue
            data["Label"].append(label)
            data["Time_start"].append(float(scene_noise["time_start"]))
            data["Time_end"].append(float(scene_noise["time_end"]))
            data["Duration"].append(round(float(scene_noise["time_end"])
- float(scene_noise["time_start"]), 2))
            data["File_path"].append(wav_path)
        else:
            continue

    df = pd.DataFrame(data)
    df = df.sort_values(['Time_start', 'Time_end'], ascending=[True,
True])
    return df

def prepare_dataset(dataset_path, window_size, step_size, output_dir):
    df_structure = {
        "File_path" : [],
        "Time_start" : [],
        "Time_end" : [],
        "Duration" :  [],
        "Label" : []
    }
    df = pd.DataFrame(df_structure)
    for root, dirnames, filenames in os.walk(dataset_path):
            for f in filenames:
                if f.endswith('json'):
                    json_path = os.path.join(root, f)
                    wav_file = f.replace(".json",
"_center_top_wav_audio_ros.wav")
                    wav_file_path = os.path.join(root,
wav_file).replace(os.sep, '/')
                    data = get_data(json_path, wav_file_path)
                    for row in data.itertuples():
                        start_time = row.Time_start
                        end_time = row.Time_end
                        while start_time < end_time:
                            if start_time + step_size < end_time:
                                window_time_end = start_time + step_size
                                duration = window_size
                            else:
                                window_time_end = end_time
                                duration = end_time - start_time
                            duration = round(duration, 2)
                            data_df = pd.DataFrame({'File_path':
[row.File_path], 'Time_start': [start_time],
                                        'Time_end':
[window_time_end],'Duration': [duration],
                                        'Label': [row.Label]})

                            df = pd.concat([df,data_df],axis=0)
                            start_time += step_size

    label_encoder = preprocessing.LabelEncoder()
    df['LabelID'] = label_encoder.fit_transform(df["Label"])
    df.to_csv(output_dir)
    print(df[["Label", "LabelID"]].value_counts())
    print(df["Duration"].mean())
    print(df["Duration"].min())
    print(df["Duration"].max())


if __name__ == "__main__":
    RANDOM_JSON_FILE = 
"fs/datasets/av/dataset_main/Hanau02/i3/Hanau02_i3_027.json"
    DATASET_PATH = "fs/datasets/av/dataset_main/"
    STEP_SIZE = 3
    WIN_SIZE = 3
    OUTPUT_DIR = 
"data_preparation/metadata_preprocessing/results/dataset_san_{}win_{}step
.csv".format(WIN_SIZE, STEP_SIZE)

    prepare_dataset(DATASET_PATH,WIN_SIZE, STEP_SIZE, OUTPUT_DIR)
```

Listing 1 - Metadata preparation code

Appendix A.2 Code used for the audio data preparation

```
class Loader:

    def __init__(self, sample_rate, mono):  # offset being time_start and 
duration is the window_size(seconds)
        self.sample_rate = sample_rate
        self.mono = mono

    def load(self, file_path, offset, time_end):
        duration = time_end - offset
        signal, sr = librosa.load(file_path, offset=offset, 
duration=duration, mono=self.mono, res_type="kaiser_fast")
        return signal, sr

    def get_sample(self, signal, offset, time_end, original_sample_rate):
        offset_samples = offset * original_sample_rate
        duration_samples = (time_end - offset) * original_sample_rate
        return signal[offset_samples:offset_samples + duration_samples]

    def resample(self, signal, original_sr):
        if self.sample_rate != original_sr:
            signal = librosa.resample(signal, original_sr, 
self.sample_rate, res_type="kaiser_fast")
        return signal


class Padder:  #
    def __init__(self, num_expected_samples, mode = "constant"):
        self.num_expected_samples = num_expected_samples
        self.mode = mode

    def is_padding_needed(self, len_arr):
        return True if self.num_expected_samples > len_arr else False

    def pad(self, array):  # padding on the end of the original array
        if self.is_padding_needed(len(array)):
            num_missing_samples = self.num_expected_samples - len(array)
            array = np.pad(array, (0, num_missing_samples),
mode=self.mode)
        return array


class MelSpecExtractor:

    def __init__(self, sample_rate, type_feature_name = "mel", type_yaxis
= "mel"):
        self.sample_rate = sample_rate
        self.type_feature_name = type_feature_name
        self.type_yaxis = type_yaxis

    def extract(self, signal):
        mel_signal = librosa.feature.melspectrogram(y=signal,
sr=self.sample_rate)[:-1]
        spectogram = np.abs(mel_signal)
        log_spec = librosa.amplitude_to_db(spectogram)
        return log_spec, self.type_feature_name, self.type_yaxis

class STFT_Extractor:

    def __init__(self, sample_rate, type_feature_name = "stft",
type_yaxis = "log"):
        self.sample_rate = sample_rate
        self.type_feature_name = type_feature_name
        self.type_yaxis = type_yaxis

    def extract(self, signal):
        stft = librosa.stft(y=signal)
        stft = np.abs(stft)
        log_spec = librosa.amplitude_to_db(stft)
        return log_spec, self.type_feature_name, self.type_yaxis


class MFCCExtractor:

    def __init__(self, sample_rate, type_feature_name = "mfcc",
type_yaxis = "mel"):
        self.sample_rate = sample_rate
        self.type_feature_name = type_feature_name
        self.type_yaxis = type_yaxis

    def extract(self, signal):
        mfccs_features = librosa.feature.mfcc(y=signal,
sr=self.sample_rate, n_mfcc=40)
        return mfccs_features, self.type_feature_name, self.type_yaxis

class CQT_Extractor:

    def __init__(self, sample_rate, type_feature_name = "cqt", type_yaxis
= "cqt_note"):
        self.sample_rate = sample_rate
        self.type_feature_name = type_feature_name
        self.type_yaxis = type_yaxis
    def extract(self, signal):
        cqt = np.abs(librosa.cqt(signal, sr=self.sample_rate))
        cqt = librosa.amplitude_to_db(cqt)
        return cqt, self.type_feature_name, self.type_yaxis

class Chroma_Extractor:

    def __init__(self, sample_rate, type_feature_name = "chroma",
type_yaxis = "chroma"):
        self.sample_rate = sample_rate
        self.type_feature_name = type_feature_name
        self.type_yaxis = type_yaxis

    def extract(self, signal):
        chroma = np.abs(librosa.feature.chroma_stft(signal,
sr=self.sample_rate))
        chroma = librosa.amplitude_to_db(chroma)
        return chroma, self.type_feature_name, self.type_yaxis

class MinMaxNormaliser:

    def __init__(self, min_val, max_val):
        self.min = min_val
        self.max = max_val

    def normalise(self, array):
        a = (array - array.min())
        b = (array.max() - array.min())
        norm_array = np.divide(a, b, out=np.zeros_like(a), where=b!=0)
        norm_array = norm_array * (self.max - self.min) + self.min
        return norm_array


class Saver:

    def __init__(self, base_feature_save_dir, duration, step_size):
        self.base_feature_save_dir = base_feature_save_dir
        self.duration = duration
        self.step_size = step_size

    def save_feature(self, feature, type_feature, file_path, offset,
time_end, label, type_df, format_type, y_axis="mel"):
        feature_save_dir =
"{}/{}_{}_{}win_{}step/{}/{}/".format(self.base_feature_save_dir,
type_feature, format_type, self.duration, self.step_size, type_df, label)
        save_path = self._generate_save_path(feature_save_dir, file_path,
offset, time_end, format_type)
        if format_type == "img":
            self.save_img(feature, save_path, y_axis)
        else :
            self.save_npy(feature, save_path)

    def save_img(self, feature, save_path, y_axis):
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.axes.get_xaxis().set_visible(False)
        ax.axes.get_yaxis().set_visible(False)
        ax.set_frame_on(False)
        librosa.display.specshow(feature, x_axis = "time", y_axis=y_axis)
        plt.savefig(save_path, bbox_inches='tight', pad_inches = 0)
        plt.clf()
        plt.close("all")

    def save_npy(self, feature, save_path):
        feature = feature[..., np.newaxis]
        np.save(save_path, feature)

    def _generate_save_path(self, feature_save_dir, file_path, offset,
time_end, format_type):
        format = "png" if format_type == "img" else "npy"
        ending_str = "_{}_{}.{}".format(offset, time_end, format)
        file_name = os.path.split(file_path)[1][:-4] + ending_str
        save_path = feature_save_dir + file_name
        return save_path


class PreProcessingPipeline:

    def __init__(self, loader, padder, feature_extractors, saver,
normaliser, format_type):
        self.loader = loader
        self.padder = padder
        self.feature_extractors = feature_extractors
        self.normaliser = normaliser
        self.saver = saver
        self.format_type = format_type

    def _extract_feature(self, feature_extractor, signal, file_path,
offset, time_end, label, type_df):
        feature, type_feature_name, type_yaxis =
feature_extractor.extract(signal)
        feature = self.normaliser.normalise(feature)
        self.saver.save_feature(feature, type_feature_name, file_path,
offset, time_end, label, type_df, self.format_type, type_yaxis)

    def _process_file(self, file_path, offset, time_end, label, type_df):
        signal, sr = self.loader.load(file_path, offset, time_end)
        signal = self.loader.resample(signal, sr)
        signal = self.padder.pad(signal)
        for feature_extactor in self.feature_extractors:
            self._extract_feature(feature_extactor, signal, file_path,
offset, time_end, label, type_df)

    def get_train_val_df(self, df_input, stratify_colname='LabelID',
frac_train = 0.8, frac_val = 0.1, frac_test = 0.1, random_state = None):

        X = df_input
        y = df_input[[stratify_colname]]


        df_train, df_temp, y_train, y_temp = train_test_split(X,
                                                            y,
                                                            stratify=y,

test_size=(1.0 - frac_train),

random_state=random_state)

        relative_frac_test = frac_test / (frac_val + frac_test)

        df_val, df_test, y_val, y_test = train_test_split(df_temp,
                                                            y_temp,
                                                            stratify =
y_temp,
                                                            test_size =
relative_frac_test,
                                                            random_state
= random_state)
        return df_train, df_val, df_test


    def process(self, dataframe, type):
        for row in dataframe.itertuples():
            try:
                self._process_file(row.File_path, row.Time_start,
row.Time_end, row.Label, type)
            except:
                print(row.File_path, row.Time_start, row.Time_end)
                continue


if __name__ == "__main__":
    DURATION = 3
    STEP_SIZE = 3
    SAMPLE_RATE = 22050
    NUM_EXPECTED_SAMPLES = DURATION * SAMPLE_RATE
    MONO = True
    FORMAT_TYPE = "img"
    DATASET_DIR =
"/home/goe2brg/DL_Violence_Detection_v7/data_preparation/metadata_preproc
essing/results/dataset_san_{}win_{}step_mnt.csv".format(DURATION,
STEP_SIZE)
    BASE_FEATURE_SAVE_DIR = "/home/goe2brg/DL_Violence_Detection_v7/data"
    TRAIN_DATASET_OUTPUT_DIR =
"/home/goe2brg/DL_Violence_Detection_v7/data_preparation/audio_preprocess
ing/datasets/processing_datasets/"
    TRAIN_DATASET_NAME =
"train_dataset_san_{}win_{}step_mnt.csv".format(DURATION, STEP_SIZE)

    df = pd.read_csv(DATASET_DIR, index_col=0)
    loader = Loader(SAMPLE_RATE, MONO)
    padder = Padder(NUM_EXPECTED_SAMPLES)
    feature_extractors = [MelSpecExtractor(SAMPLE_RATE),
MFCCExtractor(SAMPLE_RATE), CQT_Extractor(SAMPLE_RATE),
STFT_Extractor(SAMPLE_RATE), Chroma_Extractor(SAMPLE_RATE)]
    min_max_normaliser = MinMaxNormaliser(0, 1)
    saver = Saver(BASE_FEATURE_SAVE_DIR, DURATION ,STEP_SIZE)
    pipeline = PreProcessingPipeline(loader, padder, feature_extractors, 
saver, min_max_normaliser, FORMAT_TYPE)
    df_train, df_val, df_test = pipeline.get_train_val_df(df, "LabelID", 
frac_train = 0.8, frac_val=0.1, frac_test = 0.1,random_state=691)
    df_train.to_csv(TRAIN_DATASET_OUTPUT_DIR + TRAIN_DATASET_NAME)

    print(DATASET_DIR)
    print(BASE_FEATURE_SAVE_DIR)

    pipeline.process(df_train, "training")
    pipeline.process(df_val, "validation")
    pipeline.process(df_test, "testing")
```

Listing 2 - Audio data preparation code

```
class DataAugmentation:

    def __init__(self, sr):
        self.sr = sr


    def add_white_noise(self, signal, noise_percentage_factor = 0.2):
        noise = np.random.normal(0, signal.std(), signal.size)
        augmented_signal = signal + noise * noise_percentage_factor
        return augmented_signal

    def random_gain(self, signal, min_factor=0.1, max_factor=0.12):
        gain_rate = random.uniform(min_factor, max_factor)
        augmented_signal = signal * gain_rate
        return augmented_signal

    def time_strecth(self, signal, strech_rate = 0.2):
        return librosa.effects.time_stretch(signal, strech_rate)

    def pitch_scale(self, signal, num_semitones = 4):
        return librosa.effects.pitch_shift(signal, self.sr, 
num_semitones)



class PreProcessingPipeline:

    def __init__(self, loader, padder, feature_extractors, saver, 
normaliser, data_augmentation, format_type):
        self.loader = loader
        self.padder = padder
        self.feature_extractors = feature_extractors
        self.normaliser = normaliser
        self.saver = saver
        self.format_type = format_type
        self.data_augmentation = data_augmentation
    def _extract_feature(self, feature_extractor, signal, file_path,
offset, time_end, label, type_df):
        feature, type_feature_name, type_yaxis =
feature_extractor.extract(signal)
        feature = self.normaliser.normalise(feature)
        self.saver.save_feature(feature, type_feature_name, file_path,
offset, time_end, label, type_df, self.format_type, type_yaxis)

    def get_dataset(self, df_input, frac = 0.5, random_state = None):
        df_aug, df_trash = train_test_split(df_input,
                                        stratify=df["LabelID"],
                                        test_size=frac,
                                        random_state=random_state)
        return df_aug

    def _process_file(self, file_path, offset, time_end,label, type_df):
        signal,sr = self.loader.load(file_path, offset, time_end)
        signal = self.loader.resample(signal, sr)
        signal = self.padder.pad(signal)
        augmentation = random.randint(0,1)
        if augmentation == 0:
            signal = self.data_augmentation.add_white_noise(signal)
        else:
            signal = self.data_augmentation.time_strecth(signal)

        for feature_extractor in self.feature_extractors:
            self._extract_feature(feature_extactor, signal, file_path,
offset, time_end, label, type_df)

    def process(self, dataframe, type):
        for row in dataframe.itertuples():
            try:
                self._process_file(row.File_path, row.Time_start,
row.Time_end, row.Label, type)
            except:
                 print(row.File_path, row.Time_start, row.Time_end)
                 continue


if __name__ == "__main__":
    DURATION = 3
    STEP_SIZE = 3
    SAMPLE_RATE = 22050
    NUM_EXPECTED_SAMPLES = DURATION * SAMPLE_RATE
    MONO = True
    FORMAT_TYPE = "img"
    DATASET_DIR =
"/home/goe2brg/DL_Violence_Detection_v7/data_preparation/audio_preprocess
ing/datasets/processing_datasets/train_dataset_san_{}win_{}step_mnt.csv".
format(DURATION, STEP_SIZE) #CHANGE THIS ON CLUSTER
    BASE_FEATURE_SAVE_DIR = "/home/goe2brg/DL_Violence_Detection_v7/data"
    df = pd.read_csv(DATASET_DIR, index_col=0)


    loader = Loader(SAMPLE_RATE, MONO)
    padder = Padder(NUM_EXPECTED_SAMPLES)
    min_max_normaliser = MinMaxNormaliser(0, 1)
    feature_extractors = [MelSpecExtractor(SAMPLE_RATE),
MFCCExtractor(SAMPLE_RATE), CQT_Extractor(SAMPLE_RATE),
STFT_Extractor(SAMPLE_RATE), Chroma_Extractor(SAMPLE_RATE)]
    data_augmentation = DataAugmentation(SAMPLE_RATE)
    saver = Saver(BASE_FEATURE_SAVE_DIR, DURATION ,STEP_SIZE)
    pipeline = PreProcessingPipeline(loader, padder, feature_extractors,
saver, min_max_normaliser,data_augmentation, FORMAT_TYPE)

    df = pipeline.get_dataset(df, frac=0.50, random_state=691)

    pipeline.process(df, "training")
```

Listing 3 - Data augmentation code

Appendix B

Appendix B.1 Models' code implementation and architecture plot

```
def create_vgg(input_shape, n_classes, last_activation = "sigmoid"):
    model = Sequential()
    model.add(layers.Conv2D(32, (3, 3), input_shape=input_shape,
activation = "relu"))
    model.add(layers.MaxPool2D((2, 2)))

    model.add(layers.Conv2D(64, (3, 3), activation = "relu"))
    model.add(layers.MaxPool2D((2, 2)))

    model.add(layers.Conv2D(128, (3, 3), activation = "relu", ))
    model.add(layers.MaxPool2D((2, 2)))

    model.add(layers.Conv2D(256, (3, 3), activation = "relu", ))
    model.add(layers.MaxPool2D((2, 2)))

    model.add(layers.Flatten())

    model.add(layers.Dense(128, activation = "relu"))
    model.add(layers.Dropout(0.4))

    model.add(layers.Dense(64, activation = "relu"))
    model.add(layers.Dropout(0.4))

    model.add(layers.Dense(32, activation = "relu"))
    model.add(layers.Dropout(0.4))

    model.add(layers.Dense(n_classes, activation=last_activation))
```

```
return model
```

Listing 4 - VGG-16 implementation code

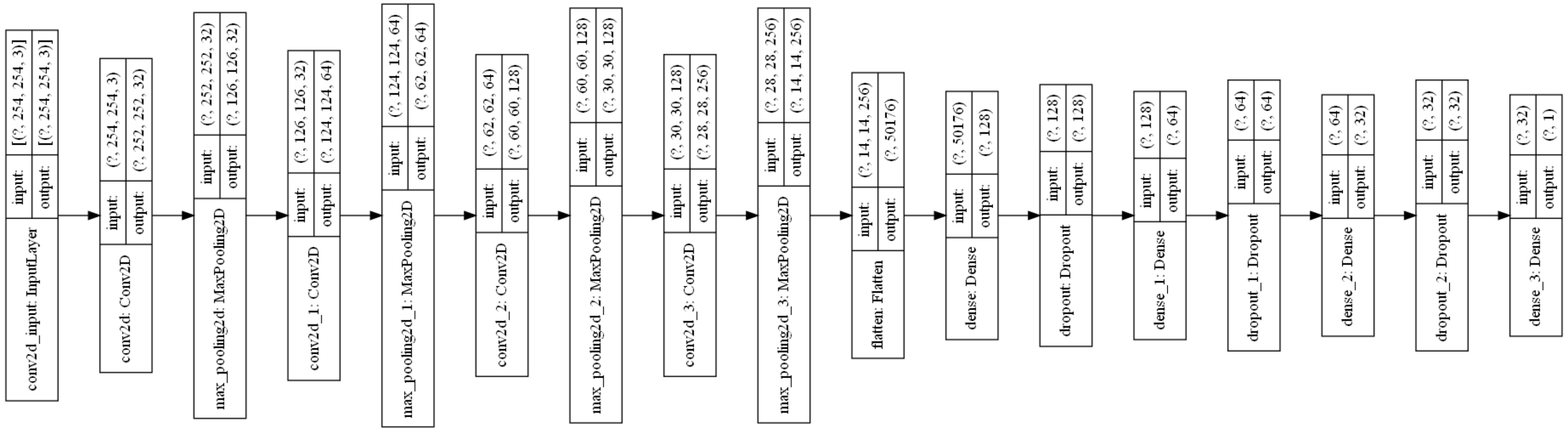


Figure 34 - VGG-16 architecture

```
def depth_block(x, strides):
    x = DepthwiseConv2D(3,strides=strides,padding='same',
use_bias=False)(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)
    return x

def single_conv_block(x,filters):
    x = Conv2D(filters, 1,use_bias=False)(x)
    x= BatchNormalization()(x)
    x = ReLU()(x)
    return x

def combo_layer(x,filters, strides):
    x = depth_block(x,strides)
    x = single_conv_block(x, filters)
    return x

def MobileNet(input_shape=(224,224,3),n_classes = 1,
activation="sigmoid"):
    input = Input ( input_shape)

    x = Conv2D(32,3,strides=(2,2),padding = 'same', use_bias=False)
(input)
    x =  BatchNormalization()(x)
    x = ReLU()(x)

    x = combo_layer(x,64, strides=(1,1))

    x = combo_layer(x,128,strides=(2,2))
    x = combo_layer(x,128,strides=(1,1))

    x = combo_layer(x,256,strides=(2,2))
    x = combo_layer(x,256,strides=(1,1))

    x = combo_layer(x,512,strides=(2,2))
    for _ in range(5):
        x = combo_layer(x,512,strides=(1,1))

    x = combo_layer(x,1024,strides=(2,2))
    x = combo_layer(x,1024,strides=(1,1))

    x = Flatten()(x)

    x = Dense(512, activation='relu')(x)
    x = Dropout(0.4)(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.4)(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.4)(x)
    x = Dense(64, activation='relu')(x)

    output = Dense(n_classes,activation=activation)(x)
    model = Model(input, output)
    return model
```

Listing 5 - MobileNet implementation code

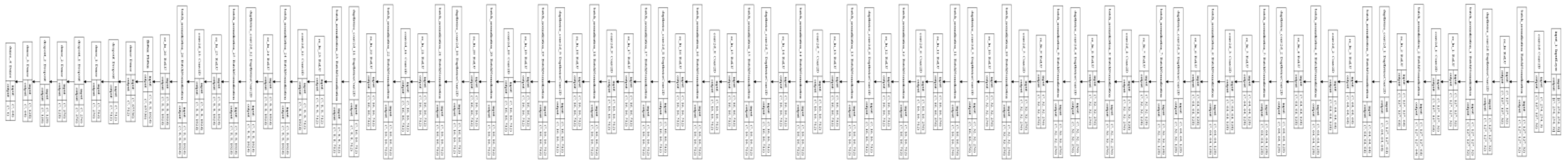


Figure 35 - MobileNet architecture

```
class ResnetBlock(Model):
    def __init__(self, channels: int, down_sample=False):
        super().__init__()

        self.__channels = channels
        self.__down_sample = down_sample
        self.__strides = [2, 1] if down_sample else [1, 1]

        KERNEL_SIZE = (3, 3)
        INIT_SCHEME = "he_normal"

        self.conv_1 = Conv2D(self.__channels, strides=self.__strides[0],
                             kernel_size=KERNEL_SIZE, padding="same", )
        self.conv_2 = Conv2D(self.__channels, strides=self.__strides[1],
                             kernel_size=KERNEL_SIZE, padding="same", )

        self.merge = Add()

        if self.__down_sample:
            self.res_conv = Conv2D(
                self.__channels, strides=2, kernel_size=(1, 1),
kernel_initializer=INIT_SCHEME, padding="same")


    def call(self, inputs):
        res = inputs

        x = self.conv_1(inputs)
        x = tf.nn.relu(x)
        x = self.conv_2(x)


        if self.__down_sample:
            res = self.res_conv(res)

        x = self.merge([x, res])
        out = tf.nn.relu(x)
        return out


class ResNet18(Model):

    def __init__(self, num_classes, last_activation="sigmoid", **kwargs):
        super().__init__(**kwargs)
        self.conv_1 = Conv2D(64, (7, 7), strides=2,
                             padding="same",
kernel_initializer="he_normal")
        self.pool_2 = MaxPool2D(pool_size=(2, 2), strides=2,
padding="same")
        self.res_1_1 = ResnetBlock(64)
        self.res_1_2 = ResnetBlock(64)
        self.res_2_1 = ResnetBlock(128, down_sample=True)
        self.res_2_2 = ResnetBlock(128)
        self.res_3_1 = ResnetBlock(256, down_sample=True)
        self.res_3_2 = ResnetBlock(256)
        self.res_4_1 = ResnetBlock(512, down_sample=True)
        self.res_4_2 = ResnetBlock(512)
        self.flat = Flatten()
        self.fc_1 = Dense(128, activation='relu')
        self.drop_out_1 = Dropout(0.5)
        self.fc_2 = Dense(64, activation='relu')
        self.drop_out_2 = Dropout(0.5)
        self.fc_3 = Dense(32, activation='relu')
        self.fc_4 = Dense(num_classes, activation=last_activation)

    def call(self, inputs):
        out = self.conv_1(inputs)
        out = tf.nn.relu(out)
        out = self.pool_2(out)
        for res_block in [self.res_1_1, self.res_1_2, self.res_2_1,
self.res_2_2, self.res_3_1, self.res_3_2, self.res_4_1, self.res_4_2]:
            out = res_block(out)
        out = self.flat(out)
        out = self.fc_1(out)
        out = self.drop_out_1(out)
        out = self.fc_2(out)
        out = self.drop_out_2(out)
        out = self.fc_3(out)
        out = self.fc_4(out)
        return out
```

Listing 6 - ResNet-16 implementation code

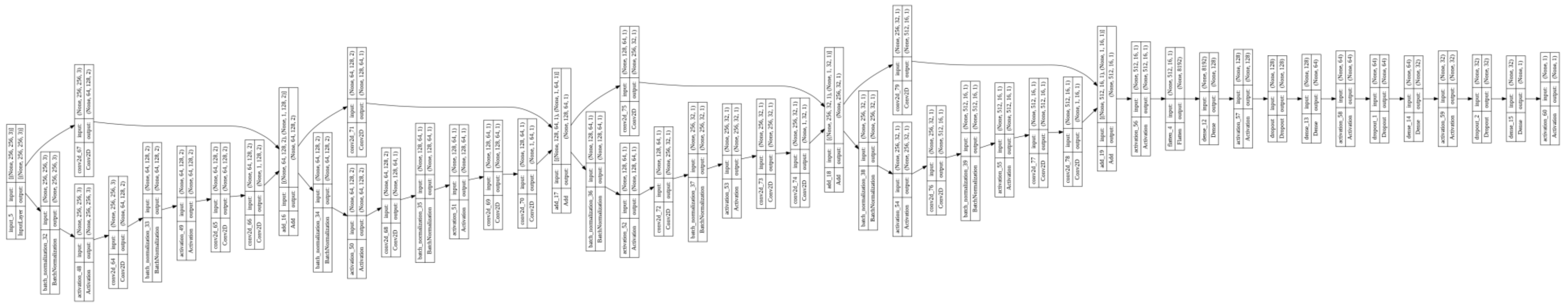


Figure 36 - ResNet-18 architecture

Appendix B.2 Model parametrization

```
if __name__ == "__main__":
    RUN_NUMBER = 2
    DURATION = 3
    STEP_SIZE = 3
    FORMAT_TYPE = "img"
    INPUT_TYPE = "mel"
    INPUT_SHAPE = (254, 254, 3)
    TRAINING_DIR =
"/home/goe2brg/DL_Violence_Detection_v7/data/{}_{}_{}win_{}step/training/
".format(INPUT_TYPE, FORMAT_TYPE, DURATION, STEP_SIZE)
    VALIDATION_DIR =
"/home/goe2brg/DL_Violence_Detection_v7/data/{}_{}_{}win_{}step/validatio
n/".format(INPUT_TYPE, FORMAT_TYPE, DURATION, STEP_SIZE)
    TESTING_DIR =
"/home/goe2brg/DL_Violence_Detection_v7/data/{}_{}_{}win_{}step/testing/"
.format(INPUT_TYPE, FORMAT_TYPE, DURATION, STEP_SIZE)

    METRICS = [
        metrics.BinaryAccuracy(name="acc", threshold=0.5),
        metrics.Precision(name='precision'),
        metrics.Recall(name='recall'),
        f1_score,
        metrics.AUC(name='auc'),
        metrics.TrueNegatives(name="tn"),
        metrics.TruePositives(name="tp"),
        metrics.FalseNegatives(name="fn"),
        metrics.FalsePositives(name="fp"),
    ]
    BATCH_SIZE = 32
    EPOCHS = 100
    LEARNING_RATE = 5e-3
    DECAY_RATE = LEARNING_RATE / EPOCHS
    LOSS = BinaryCrossentropy()

    model_checkpoint_callback_f1_score = ModelCheckpoint(
    filepath=MODEL_PATH_F1SCORE,
    save_best_only = True,
    save_weights_only = True,
    monitor='val_f1_score',
    verbose = 1,
    mode='max',
    )

    CALLBACKS = [model_checkpoint_callback_f1_score]
    image_gen = ImageDataGenerator(rescale=1./255)
    train_gen = image_gen.flow_from_directory(TRAINING_DIR,
                                            target_size =
(INPUT_SHAPE[0], INPUT_SHAPE[1]),
                                            batch_size = BATCH_SIZE,
                                            class_mode = 'binary',
                                            shuffle = True
                                            )

    val_gen = image_gen.flow_from_directory(VALIDATION_DIR,
                                            target_size =
(INPUT_SHAPE[0], INPUT_SHAPE[1]),
                                            batch_size = BATCH_SIZE,
                                            class_mode = 'binary',
                                            shuffle = True
                                            )
    test_gen = image_gen.flow_from_directory(TESTING_DIR,
                                            target_size =
(INPUT_SHAPE[0], INPUT_SHAPE[1]),
                                            batch_size = BATCH_SIZE,
                                            class_mode = 'binary',
                                            )


    class_weights = class_weight.compute_class_weight(
                class_weight = 'balanced',
                classes = np.unique(train_gen.classes),
                y = train_gen.classes)

    CLASS_WEIGHTS = {
        0: class_weights[0],
        1: class_weights[1],
    }


    optimizer = SGD(learning_rate=LEARNING_RATE,
decay=LEARNING_RATE/EPOCHS)
```

Listing 7 - Models' parameter initialization

Appendix C

Appendix C.1 Training and validation plots for the VGG-16 model

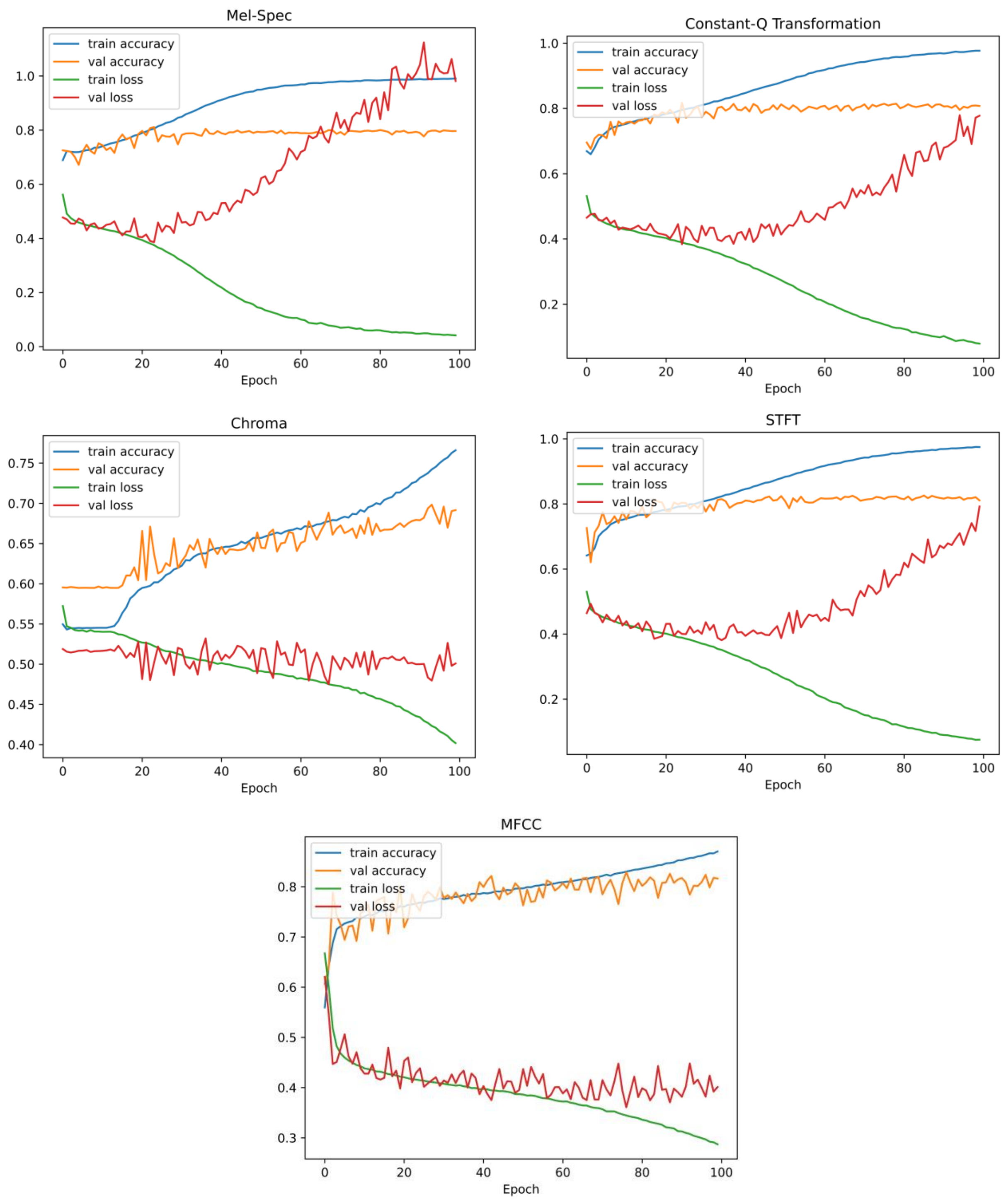


Figure 37 - VGG model training plot of all audio features

Appendix C.2 Training and validation plots for the MobileNet model

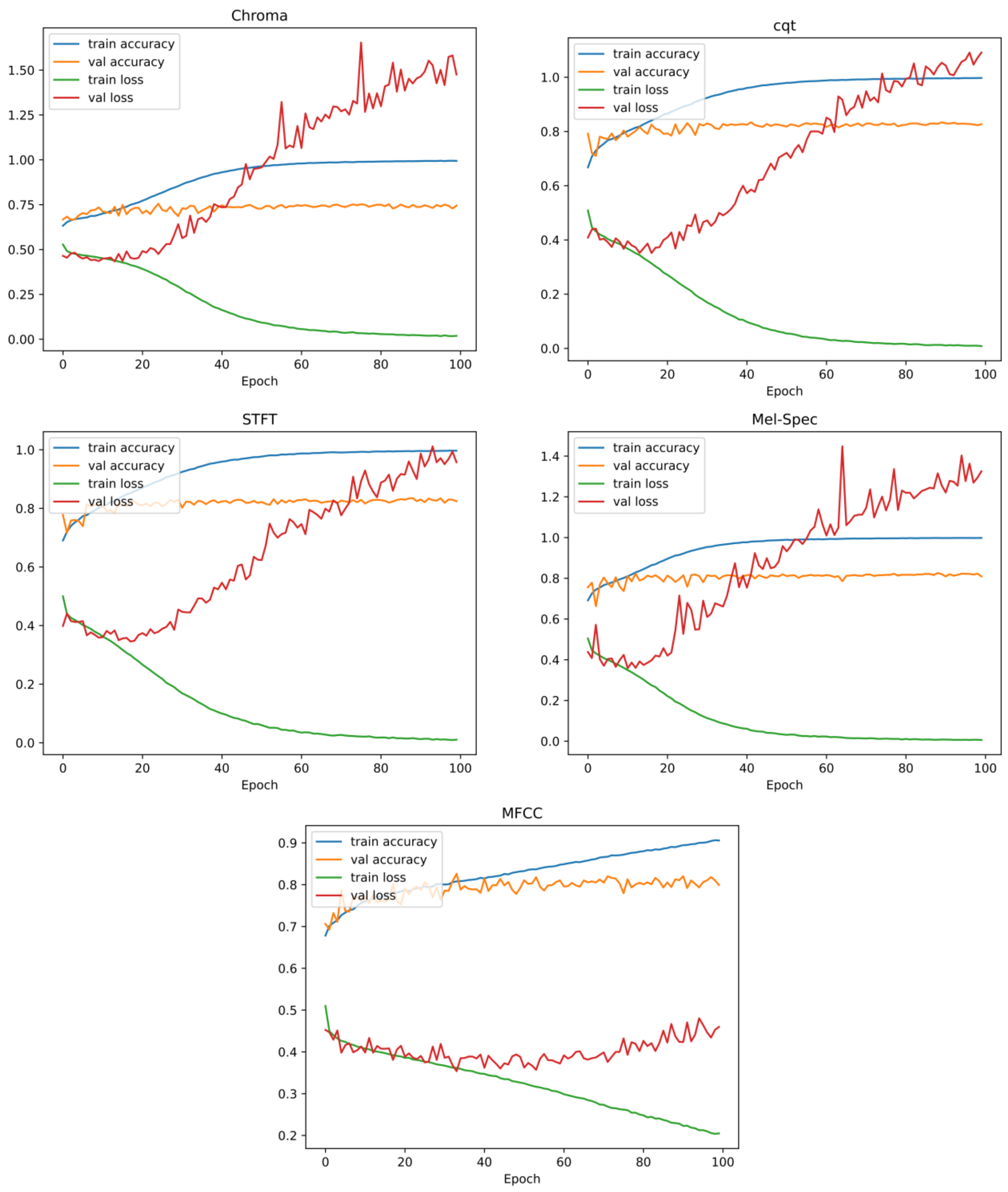


Figure 38 - MobileNet model training plot of all audio features

Appendix C.3 Training and validation plots for the ResNet-18 model

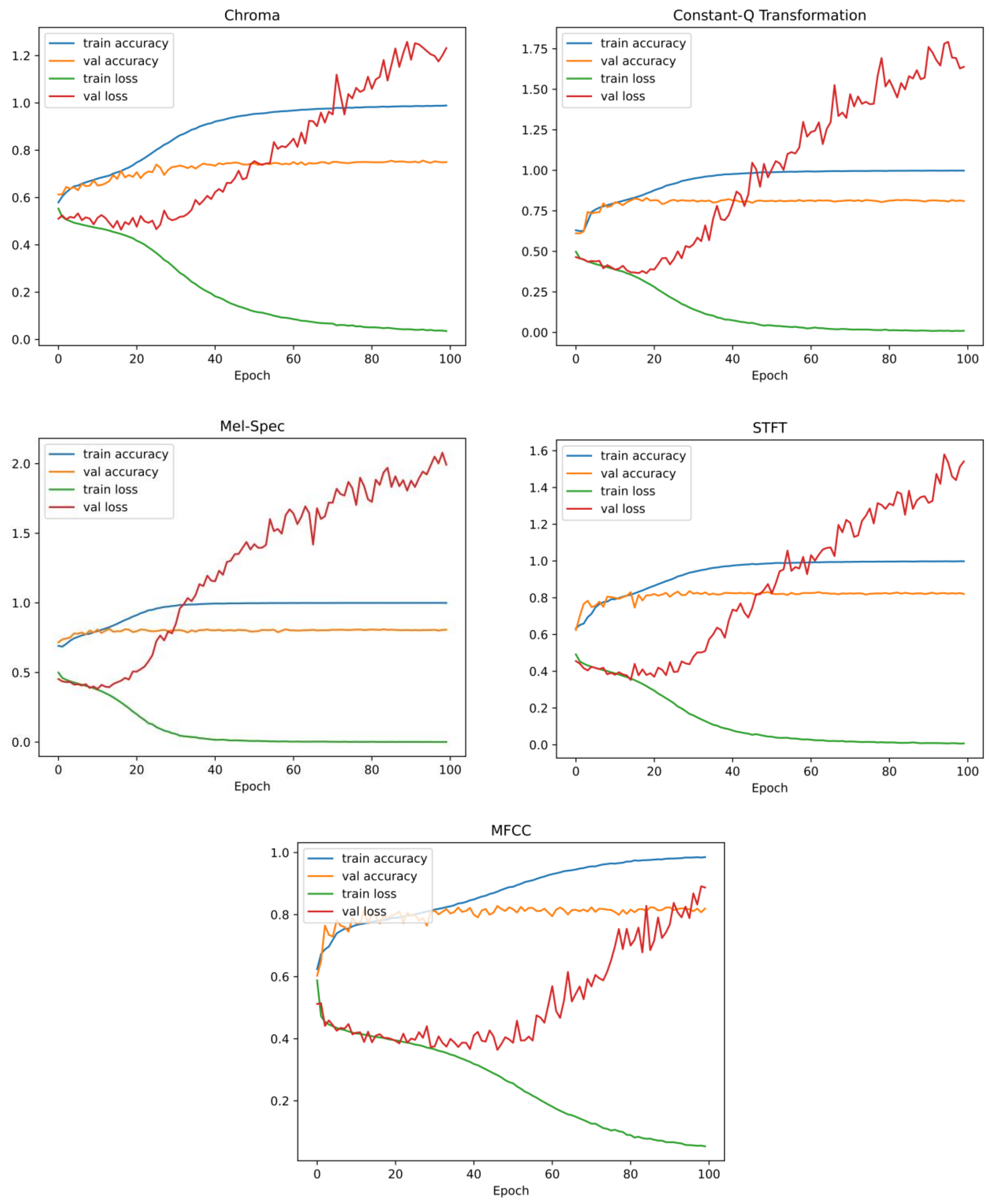


Figure 39 – ResNet-18 model training plot of all audio features