UNIVERSIDADE FEDERAL DO PARANÁ

ABDUL MOHAMAD KADRI HOFFMANN

A DEEP LEARNING APPROACH FOR EMERGENCY VEHICLES CLASSIFICATION AND LOCALIZATION USING ACOUSTIC SENSORS

CURITIBA

2021

ABDUL MOHAMAD KADRI HOFFMANN

A DEEP LEARNING APPROACH FOR EMERGENCY VEHICLES CLASSIFICATION AND LOCALIZATION USING ACOUSTIC SENSORS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Engenharia Elétrica, pelo Programa de Pós-Graduação em Engenharia Elétrica, Setor de Tecnologia, Universidade Federal do Paraná.

Orientador: Prof. Dr. Eduardo Parente Ribeiro

Co-Orientador: Prof. Dr. Werner Huber

CURITIBA

2021

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

H699d Hoffmann, Abdul Mohamad Kadri
A deep learning approach for emergency vehicles Classification and localization using acoustic sensors [recurso eletrônico] / Abdul Mohamad Kadri Hoffmann – Curitiba, 2021.

Dissertação - Universidade Federal do Paraná, Setor de Tecnologia, Programa de Pós-graduação em Engenharia Elétrica.

Orientador: Prof. Dr. Eduardo Parente Ribeiro
Coorientador: Prof. Dr. Werner Huber

1. Redes Neurais Convolucionais. 2. Sensores Automotivos. I. Universidade Federal do Paraná. II. Ribeiro, Eduardo Parente. III. Hube,r Werner. IV. Título.

. CDD: 004.36

Bibliotecária: Roseny Rivelini Morciani CRB-9/1585

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em ENGENHARIA ELÉTRICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **ABDUL MOHAMAD KADRI HOFFMANN** intitulada: **A Deep Learning Approach for Emergency Vehicles Classification and Localization using Acoustic Sensors**, sob orientação do Prof. Dr. EDUARDO PARENTE RIBEIRO, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 27 de Agosto de 2020.

Assinatura Eletrônica
28/08/2020 10:28:53.0
EDUARDO PARENTE RIBEIRO
Presidente da Banca Examinadora

Assinatura Eletrônica
27/08/2020 19:12:12.0
JULIO CÉSAR NIEVOLA
Avaliador Externo (PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ)

Assinatura Eletrônica
28/08/2020 14:10:34.0
LUIS HENRIQUE ASSUMPÇÃO LOLIS
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
28/08/2020 10:44:00.0
GIDEON VILLAR LEANDRO
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

AGRADECIMENTOS

Gostaria de agradecer imensamente ao meu professor orientador, professor Dr. Eduardo Parente Ribeiro. Sem seu apoio técnico e administrativo o caminho já desfiador da dupla diplomação seria simplesmente intransponível. Obrigado por me acreditar e me apoiar em direção de meus sonhos — o de desenvolver uma pesquisa com desdobramentos práticos em ambiente internacional.

Gostaria de agradecer profundamente ao doutorando Fabio Reway por seu suporte e paciência. Obrigado por doar seu tempo profissional e pessoal a mim. Sem isso, um dos momentos de mais fragilidade em minha vida teria sido ainda muito mais complicado.

Meu mais sincero obrigado ao professor Dr. Werner Huber e ao time do CARISSMA, o qual nunca falhou em providenciar um ambiente frutífero para desenvolver ideias e ensaios. Foram nas dependências do CARISSMA que tive um dos período mais intensos de aprendizado que já tiver a oportunidade de experimentar.

Agradeço também o Departamento de Engenharia Elétrica da UFPR. Os professores e funcionários os quais impactaram a minha vida positivamente foram agentes essenciais para que a minha curiosidade e ambição técnica se desenvolvesse.

Finalmente, agradeço à minha mãe e ao meu pai por terem me proporcionado todo o amor e carinho para eu poder me aprimorar intelectualmente. Sem a nossa estrutura familiar, seria impossível ter alçado qualquer voo. Obrigado por todos os seus sacrifícios.

RESUMO

A indústria automotiva tem como uma de suas tecnologias mais notáveis e promissoras as funções de sistemas avançados de assistência ao condutor — Advanced Driver Assistance System (ADAS) e consequentemente a direção autônoma. Isto não só deve afetar positivamente a segurança atualmente encontrada no tráfego urbano, mas deve revolucionar todo o modelo de negócios dessa indústria, como já se observa. Este projeto objetiva investigar se sensores acústicos automotivos podem ser considerados uma contribuição tecnológica viável ao atual grupo de sensores usualmente utilizados. Uma rede neural convolucional (CNN) foi treinada como modelo de classificação binário para detecção de veículos de emergência. O classificador que teve melhor desempenho foi obtido utilizando uma versão modificada da arquitetura AlexNet, treinada com uma variedade de sons de tráfego urbano e sirenes de veículos de emergência de diversas regiões do mundo. Foi utilizado um total de 23 horas de gravação. Os segmentos de áudio foram tratados e pré-processados até chegar à forma de espectrogramas-mel, os quais foram utilizados na camada de entrada da CNN. Isto permitiu ao modelo uma melhor capacidade de generalização sobre o conceito da presença, ou não, de um veículo de emergência. Obteve-se um score-f1 médio de 0,935 e 0,895 no treinamento e validação cruzada, respectivamente, e um valor de 98% de área sob a curva ROC (AUC). Também se avaliou a localização da fonte sonora utilizando a técnica de correlação-cruzada generalizada — Generalized Cross-Correlation (GCC) para estimar a diferença de tempo de chegada — *time difference of arrival* (TDOA) das frentes de ondas, e um conjunto de regras heurísticas para obter a direção de chegada — direction of arrival (DOA). Assim se obteve o azimute relativo da fonte, com acurácia de 89,89% em cenários estáticos, mas somente 4,88% em cenários dinâmicos. O conjunto de treinamento utilizado foi uma mistura entre o banco de áudios AudioSet and ensaios gravados nas premissas Center of Automotive Research on Integrated Safety Systems and Measurement Area (CARISSMA). Os resultados da classificação foram melhores do que os encontrados em outros estudos de classificação de áudio em situações de trânsito, enquanto os resultados de localização de fonte sonoras só se mostraram eficientes em cenários estáticos.

Palavras-chave: Processamento Digital de Áudio, Classificação de Cenários de Áudio, Redes Neurais Convolucionais, Sensores Automotivos.

ABSTRACT

One of the most prominent and promising technologies in the modern automotive industry is the advent of *Advanced Driver Assistance System* (ADAS) and the autonomous driving. It may not only change safety levels currently found in traffic, but also revolutionize the whole automotive industry business model, as we can see hints of it happening already. This project investigates whether audio sensors can be a technological viable addition to the current sensor set vehicle commonly use. A Convolutional Neural Network (CNN) was trained to classify auditory scenarios as containing emergency vehicles or not containing emergency vehicles. The best performing classifier was obtained using a slightly modified AlexNet architecture, trained with audio excerpts of various urban and traffic scenarios as well as emergency vehicle sirens from all over the world, totaling a duration of almost 23 hours. These excerpts went through a series pre-processing steps, and transformation to mel-spectrograms that were fed to CNN's input layer. That granted the model better generalization over the class of emergency vehicles, allowing the model to achieve class-averaged f1-scores for the training and validation sets of 0.935 and 0.895, respectively. Using the latter dataset, a 98% Area Under the ROC Curve (AUC) was achieved. Moreover, a sound source localization algorithm was employed using generalized cross-correlation (GCC) to provide the source's audio wavefront Time-Difference of Arrival (TDOA) on the microphone array. A group of heuristic rules were applied to disambiguate these values, transforming to Direction of Arrival (DOA). The obtained relative azimuth, presented 89.89% accuracy in static scenarios, but only 4.88% accuracy in dynamic scenarios. The training data was assembled using a mixture between AudioSet dataset and tests recorded in enter of Automotive Research on Integrated Safety Systems and Measurement Area (CARISSMA) facilities, and the latter source was also used for source localization estimation and could be validated against data recorded using the Automative Dynamic Motion Analyzer (ADMA). Results show that classification results were better than the ones found in closely correlated works, but source localization results only showed efficiency in static scenarios.

Keywords: Digital Audio Processing, Auditory Classification, Convolutional Neural Networks, Automotive Sensors.

FIGURES LIST

TABLES LIST

ACRONYMS

ADMA	Automative Dynamic Motion Analyzer
ANN	Artificial Neural Networks
ASL	Acoustic Source Localization
AUC	Area Under the ROC Curve
ADAS	Advanced Driver Assistance System
CAPS	Combined Active & Passive Safety
CARISSMA	Center of Automotive Research on Integrated Safety Systems and Measurement Area
CLI	Command Line Interface
CNN	Convolutional Neural Network
DCASE	Detection and Classification of Acoustic Scenes and Events
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DL	Deep Learning
DOA	Direction of Arrival
DNN	Deep Neural Network
FAD	Fully Automated Driving
FFT	Fast Fourier Transform
GCC	Generalized Cross-Correlation
GMM	Gaussian Mixture Model
HAD	Highly Automated Driving
HMM	Hidden Markov Model
kNN	k-Nearest Neighbors
MEMS	Microelectro-Mechanical Systems

MFCC	Mel-Frequency Cepstral Coeficient
ML	Machine Learning
MLP	Multilayer Perceptron
NHTSA	National Highway Traffic Safety Administration
NN	Neural Networks
OCGPC	One-Class Gaussian Process Classification
OEM	Original Equipment Manufacturer
PCM	Pulse-Code Modulation
PHAT	Phase Transform
RNN	Recurrent Neural Networks
ROC	Receiver Operating Characteristic
SAE	Society of Automotive Engineers
SNR	Signal-to-Noise Ratio
SR	Sampling Rate
STFT	Short-Time Fourier transform
SVM	Support Vector Machine
TDOA	Time Difference of Arrival
TaaS	Transport-as-a-Service

TABLE OF CONTENTS

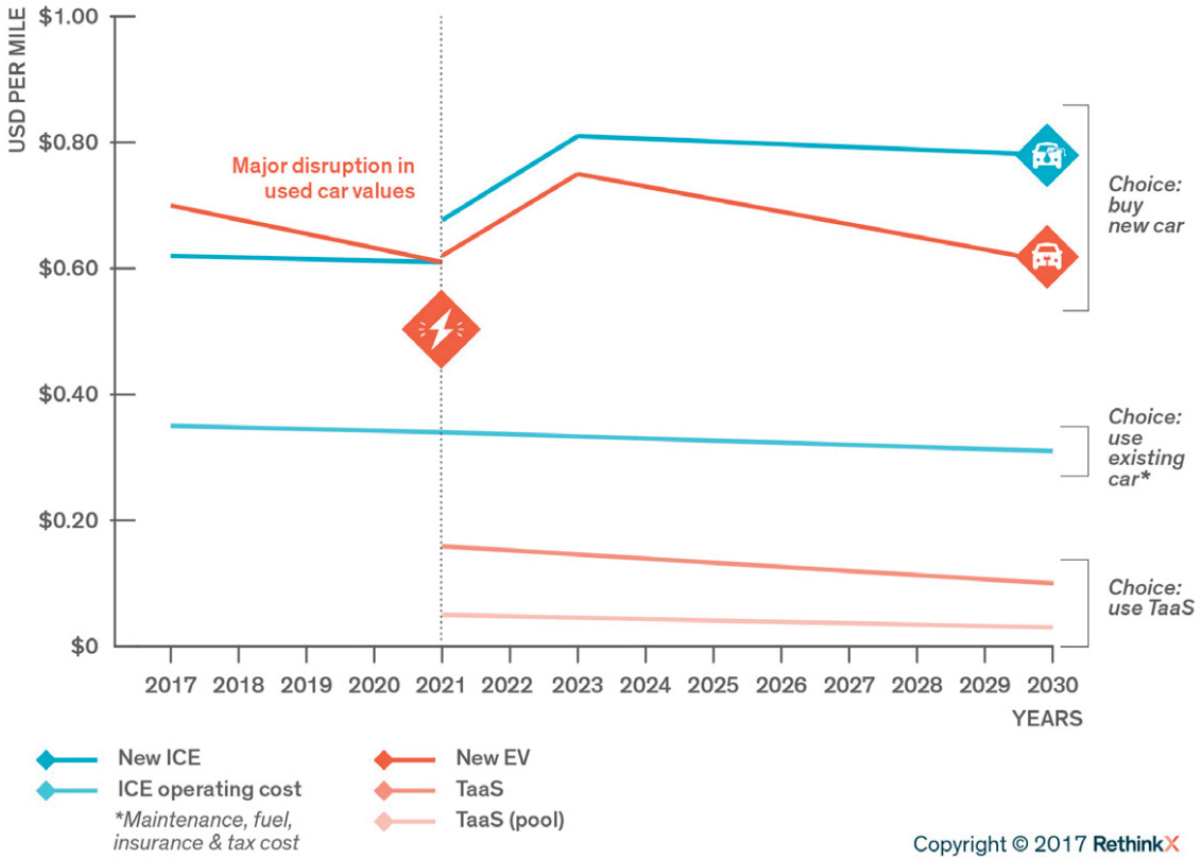
1 Introduction

1.1 The Automotive Business Model Revolution

Huge shifts towards the development and usage of autonomous driving are forcing the whole automotive industry to quickly adapt, as example the German Original Equipment Manufacturer (OEM) Volkswagen, BMW and Daimler cooperation geared towards automated driving. This technology impacted the whole business model in the automotive field. Predictions point towards a transformation from the current business model to transport-as-a-service (TaaS). This means that instead of each private entity owning a vehicle in order to commute around, the means of transportation are shared between them. In this model, the car is perceived as an asset which is shared between users, while its property remain tied to the company offering the service. This resembles — to some extent — the current services offered by companies such as Uber and Lyft, also called pre-TaaS platforms, whereas the major difference between these two category of services lie in the absence of a human driver in front of the wheel. That way, robo-taxis will operate under lower operation costs for the provider. It is possible mainly due to fleet optimization (instead of having a privately owned vehicle with a time usage rate of 4%), that TaaS vehicles would be available on-demand 24 hours per day (ARBIB; SEBA, 2017). That is, once a person is dropped at their destiny the can be immediately summoned by a different user. It is estimated that costs would be cut from four to ten times if a person would adopt TaaS instead of buying a new car in 2021 (ARBIB; SEBA, 2017), as seen in figure 1.

Robo-taxis are self-driving cars (so called self-driving taxis) with a Society of Automotive Engineers (SAE) autonomous level of 4 or 5, which is operated by an on-demand mobility service. The concepts of autonomous levels aggregate how a feature or set of features when engaged interact with the human driver, from no automation (level 0 — Systems in place to issue warnings, but no vehicle control) to full automation (level 5 — No human intervention required) (SAE, 2018). These levels are depicted in figure 2.

Figure 1: Comparison of costs between individually owned Internal Combustion Engine (ICE) and Electric Vehicles (EV), and TaaS. (ARBIB; SEBA, 2017)

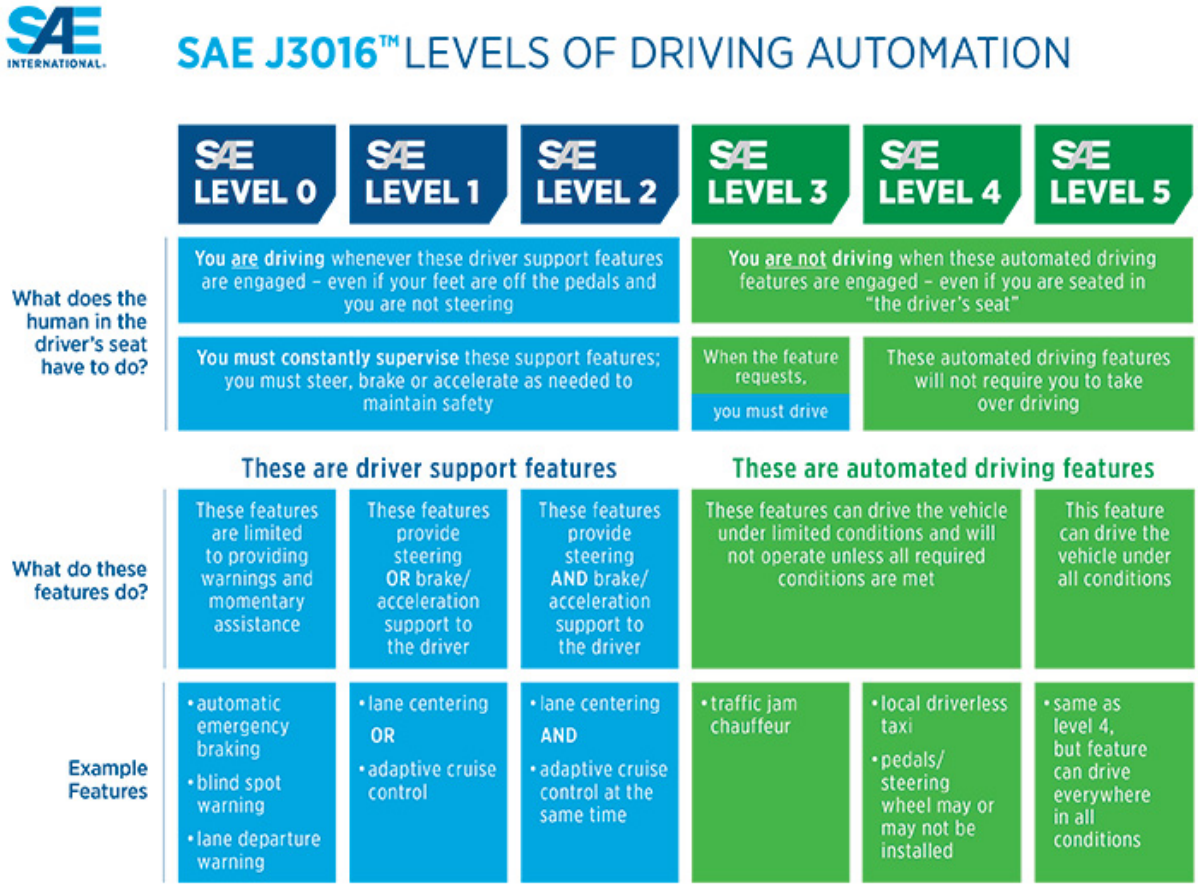


This business model transformation is predicted to happen within 10 years, which juxtapose with the expected full self-driving arrival (ARBIB; SEBA, 2017).

1.2 Computers on-board

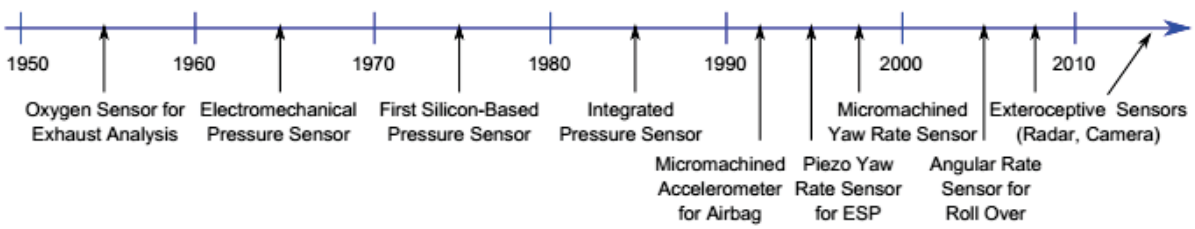
The automotive industry is going through a major overhaul, with the global autonomous driving market expected to grow up to $173.15 billions by 2030, with shared mobility services contributing to approximately 65% (CORPORATION, 2018). However, even though cars are a relatively new advent, it is not the first time this industry has gone through prominent changes. May it be due to technology pull or push, the field has historically demanded high investments in research and development. Some of the biggest milestones achieved during the automotive sensor industry history can be seen in figure 3. Many mechanical and electromechanical parts in a car have been replaced by digital electronic devices. Nowadays these parts and systems are mostly controlled by Electronic Control Unit (ECU), which convert and process signals output by the sensors.

Figure 2: Summary of the interactions between automatic features and human responsibilities among the levels of driving automation. (SAE, 2018)



There is a trend of embedding technology onto vehicles, that may have started with analog parts, but recent higher efficiency and safety demands — often times imposed by legislation — started charging heavier computational tasks. Evidently, Advanced Driving Assistance Systems (ADAS) and Fully Automated Driving (FAD) are one biggest focus of today's attention in the industry, comprising one of the four key parts of the Combined Active & Passive Safety (CAPS) systems. The main objective of this category of functions is to sense the surroundings of ego and provide appropriate information or automatic actuation (PFÄFFLE, 2006), such as done in the Adaptive Cruise Control.

Figure 3: Milestones in automotive sensor development. (BOTSCH, 2018)



These functions created a huge demand for specialized hardware capable of analyzing images from cameras, rasters from radars, and point clouds from lidars.

1.3 Automotive Safety as a Priority

The trend for 2030 indicates that not only Highly Automated Driving (HAD) functions will become more popular in terms of adoption in mid-range vehicles, but also that some of high-end cars are going to be totally autonomous, rather than restraining automation just to certain functions of the vehicle (Corporate Partnership Board, 2015).

Within this context, above comfort and convenience that these autonomous vehicles may deliver, the main motivation for wide adoption of these technologies is the safety factor. In 2015, the US National Highway Traffic Safety Administration (NHTSA) asserted that 94% of accidents are caused by human error in American traffic (General Motors, 2019). The number ascertained by the German Federal Statistical Office in 2018 was even higher, crediting about 98% of the accidents causes to human error (STATISTISCHESBUNDESAMT, 2017). More responsibility now lies on the shoulders of sensors of a vehicle than ever before, aiding or even replacing the human senses in order to provide better traffic safety. This is why providing world-class object detection and tracking reliability is pivotal for the vehicular application.

1.4 Project Outline

An autonomous driving system conceptually consists of three independent high level subsystems: The sensor subsystem, the data fusion subsystem, and the actuation subsystem. The first one acts as an input interface for the whole architecture. The data fusion layer processes all data and acts as a middleware. It is able to manage and transform the input data and also synthesize new data from the numerous raw inputs as needed to serve the output layer. The actuators use all data previously collected and processed to effectively perform driving functions such as steering, accelerating, and breaking.

This work unfolds the first subsystem, looking to provide exploratory analysis on an unconventional sensor for ADAS and FAD functions. To sense the environment autonomous and automated driving classically relies mainly on radar, sonar, lidar and camera sets (MARCHEGIANI; POSNER, 2017). However, human basic senses are the ones which society has most relied on for driving so far — namely, sight and hearing. Sight can be compared to computer vision, which is already widely applied in automated driving functions. Humans

use hearing in traffic constantly for being aware of the presence of cars, trucks, sirens, bicycles, pedestrians, animals and so on. These acoustic cues definitely improve surroundings awareness, Thorslund (2014) shows that people suffering from hearing loss tend not to get a driver's license, despite showing similar performance under baseline conditions and Gordon e Pearson (2016) shows that deaf and hard-of-hearing drivers on Rochester Institute of Technology campus are 1.5 to 3.1 times more likely to get involved in motor vehicle accidents. In the other hand, Thorslund (2014) also showed that persons in such conditions apply compensatory strategies, i.e., they look more often at the mirrors, drive slower and scan the environment in a more general way before looking away from the road. An interesting conclusion is that the performance difference between deaf and non-deaf drivers is reduced when the scenario is more demanding than the baseline driving — another cognitive task is simultaneously requested, as using a navigation system. This leads us to the conclusion that lack of hearing diminishes the overall confidence for the driving activity. For automated driving functions, this information means that interpreting the surroundings soundscape will help improve the system performance if this analogy holds true.

This opening has already been recognized and approached by other authors using classical digital signal processing to detect characteristic emergency vehicles' sirens (MEUCCI et al., 2008), but with limited generalization capabilities. To solve this problem, many different algorithms can be employed, or even a combination of these. Since audio related classification problems have started being better addressed during the last decade (GUBBI et al., 2013), the number of classifier categories employed and analyzed in this context has greatly risen. This can also be said for the increasing number of audio applications such as auditory scene classification (MESAROS; HEITTOLA; VIRTANEN, 2016), acoustic event detection (HRABINA; SIGMUND, 2018), automatic speech recognition (O'Shaughnessy, 2015), sound source separation (Slizovskaia et al., 2019), musical genre classification (TZANETAKIS; COOK, 2002), among others.

This project comprises a set of microphones giving the vehicle the ability to interpret acoustic cues in order to provide yet another input to the data fusion layer. Related works such as (MARCHEGIANI; POSNER, 2017), will serve as a baseline of comparison. The scope of both works is virtually identical where (MARCHEGIANI; POSNER, 2017) provides a model for multi-class classification using the k-nearest neighbors (kNN) algorithm for classifying audio excerpts recorded in traffic scenarios, whilst no proper localization implementation is provided by the reference.

Figure 4: Schematic of expected driver's behavior for forming the *Rettungsgasse*. (DEFEUG, 2013) .

Microphones may excel in situations in which the others would not be able to provide reliable information about the objects in the scene, such as corners — situations where neither cameras or radars present good performance due to its geometrical characteristics.

In section 2.1.3, we will further discuss related works and also state-of-the-art techniques which are being currently employed to solve the auditory classification problem.

What is desired with this project is to assert the performance of acoustic sensors in automated driving applications. This will be accomplished by developing a prototype pipeline for providing an object list and basic localization.

For simplicity's sake, the scope will be restricted to classify whether emergency vehicles are present in the auditory scenario, which accompanied by sirens stand out more clearly in the urban soundscapes. This fact should be taken in account, as our main comparison work (MARCHEGIANI; POSNER, 2017) provides multi-class classification, while we provide a binary one. To validate the prototype a use case consisting of roadways coping with traffic jams where emergency vehicles with the sirens turned on go through a corridor made by the other drivers. This structure called *Rettungsgasse* is represented in figures 4 and 5. This work lastly strives for laying the foundations for automating this task, providing a basic setup for future development by providing elementary classification and localization features to the integration layer. Such function would certainly be necessary for achieving SAE levels 4 and 5 of autonomous driving in German roadways.

1.5 Project Structure

This document is comprised of an bibliographic review, where we shall provide a handful of closely related works and what they affirm and what can be taken from these statements to provide more land coverage to this project. It will cover the core concepts in machine learning in general, from its conceptual employment and which problems it looks to

Figure 5: Example of an actual *Rettungsgasse*. (WIKIPEDIA, 2005) .

solve to the mathematical framework. Later on, we will dive in how Deep Neural Network (DNN) works, providing better understanding on what must be done in order to retrieve a good model from the training process. We will initially cover image processing as it is much more intuitive and then translate these concepts to audio processing. We will also cover the main parameters of digital audio, how it reproduces a analog signal in much more compressed space using not only digital signals but also encoding and decoding to gain in compression while losing the smallest amount of information we possibly can. Lastly, an overview on the main sound source localization algorithms are going to be provided, along with a pick justification, following the mathematical framework introduction.

In section 3, a detailed overview on the tooling built around the problem is going to be described. This is an important part of the project, consuming the biggest amount of time as many tools were home-brewed and thereafter verified by the author. It will go through the methodology employed to build the datasets, and the software engineering behind the classifier and the localization algorithms.

In section 4, we will analyze the results obtained from applying all the tools together in a cooperative way that culminates in a series of performance indicators. These performance indicators provide insights on how good the classifier is, so we can start drawing conclusive statements over the viability of such an algorithm on the automotive area, provided this prototype offers a satisfactory performance for the project's scope. A conclusion gathers all the analysis results on a consolidated manner and provides future improvements for the current work.

2 Bibliographic Review

2.1 Sound Source Classification

With the plethora of algorithms designed for classification, detection, and segmentation, it is simpler to shape a parallel between image processing as this domain is easier for humans to visualize — due to the media being used — what are the actual implications in the signal and then return to the acoustic domain.

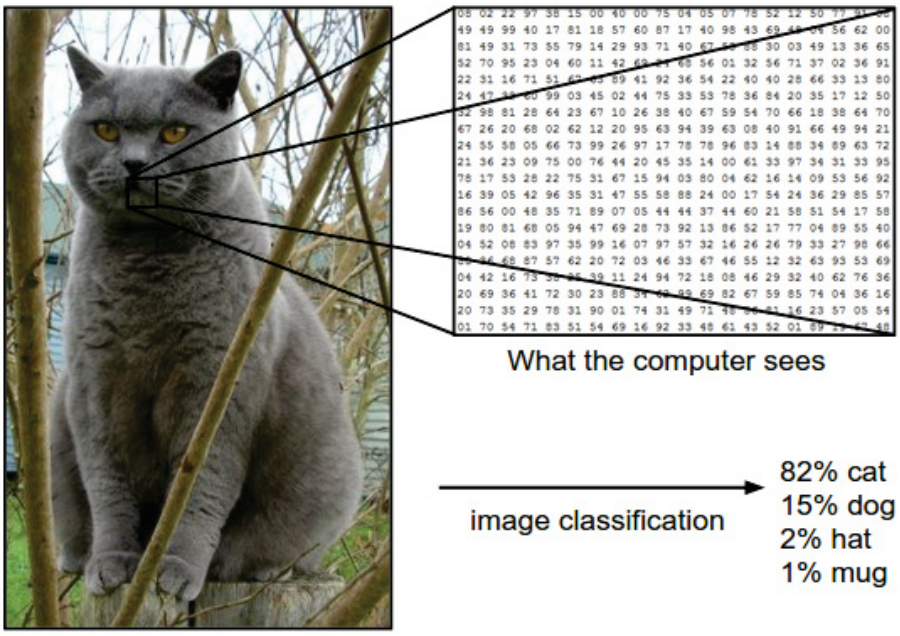
2.1.1 Tasks Taxonomy within Computer Vision

The course of Convolutional Neural Networks for Visual Recognition offered by the Stanford University (NG; LI; KARPATHY,) offers great insight on the subject — as said before, the field of computer vision works with the very same types of algorithms, since both audio and images are nothing more than digital signals arranged in different forms.

The most fundamental problem computer vision looks to solve is image classification. Labels can be interpreted as tags given to each image in a dataset. Images consist of of pixels with values associated to them, e.g., an image composed of 248 horizontal pixels and 400 vertical pixels, will accommodate 99200 values between 0 and 255 per channel. Monochromatic images possess only a single channel, but we usually look at full color pictures and it means three channels are present: Red, Green and Blue. This combination is also known as RGB. Considering that, we get 297,600 values ranging from 0 to 255, illustrated by figure 6 . Hence, the problem can be mathematically summarized as finding a good model that maps these 297,600 values to a single set of characters or integer number.

Often called 'Classification + Localization' and Object detection, they differ by the number of instances being analyzed, as localization only looks for the semantic meaning of the concept, while object detection looks for the actual objects. Both of these techniques utilize bounding boxes, which are a type of labeling where a box can be created, edited and deleted, and the box is assigned concepts. It surrounds the concept and is described by its position in

Figure 6: Example of how an image is interpreted at lower level. (NG; LI; KARPATHY,)



the image or in a grid within the image. The comparison between both tasks can be seen in figure 7.

Lastly, the task of segmentation is approached. Segmentation also be done in regard to both semantics and objects, as showed in figure 8. The objective here is to find exact bounds which describe a given concept, i.e., associate labels with areas in the image. In that way, the limits between all objects are strictly described under coordinates, not allowing overlaps between these segments.

Independent of number of classes present and the way data is associated or arranged, every problem is a linear combination of the data aspects, such as pixel position, value and neighbor pixels relation.

2.1.2 Digital Audio and Auditory Classification Tasks

For images, we have values associated to each pixels in each color channel. For digital audio, we have different values associated to discrete amplitude levels across the discrete time. Videos can be nicely compared to audio, since both are arranged along time. Generically processes can be described by n independent variables (Weinstein; Feder; Oppenheim, 1993). For three channels we would have:

Figure 7: Comparison between the output of Classification+Localization and Object Detection Algorithms. (NG; LI; KARPATHY,)

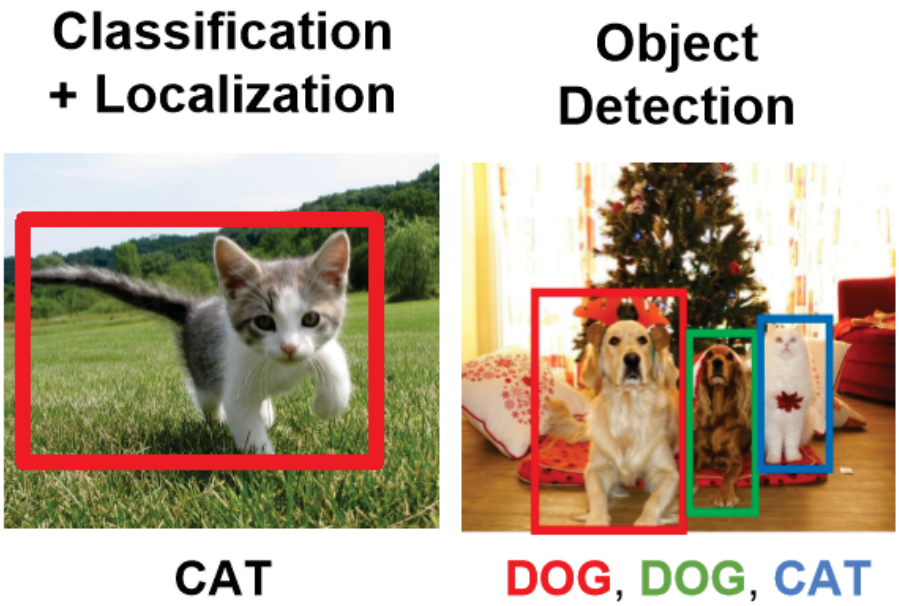
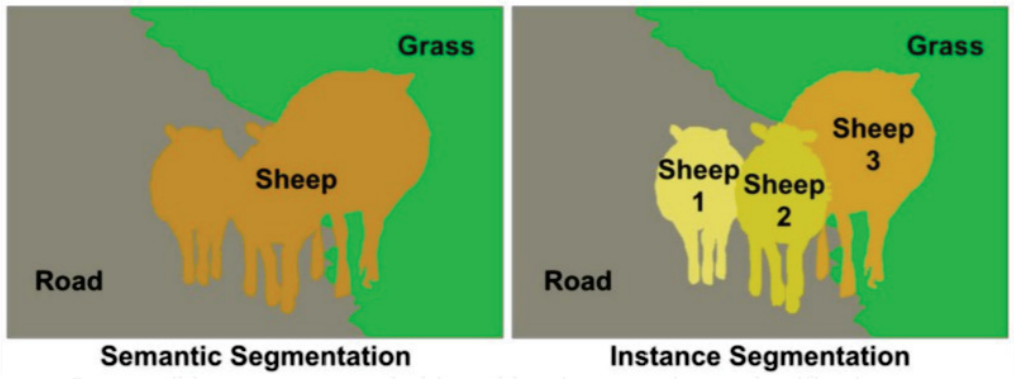


Figure 8: Examples of segmentation on a road alongside a grass field with sheep. (NG; DROR, 2018)



$$H(\omega) = \begin{bmatrix} H_{11}(\omega) & H_{12}(\omega) & H_{13}(\omega) \\ H_{21}(\omega) & H_{22}(\omega) & H_{23}(\omega) \\ H_{31}(\omega) & H_{32}(\omega) & H_{33}(\omega) \end{bmatrix} \tag{2.1}$$

where $H_{xx}(\omega)$ are the transfer functions of each channel separately and $H_{xy}(\omega)$ represent the cross-coupling effects between them. Thus, for three completely decorrelated independent variables, we would have:

$$H(\omega) = \begin{bmatrix} H_{11}(\omega) & 1 & 1 \\ 1 & H_{22}(\omega) & 1 \\ 1 & 1 & H_{33}(\omega) \end{bmatrix} \tag{2.2}$$

We can describe video signals as a 3-dimensional matrix containing information for the red, green, and blue channel — where each transfer function carries information about the pixel position matrix as well, height and width. Stereo audios are described on a 2-dimensional matrix, left and right channel — where each transfer function carries information about the instant wave amplitude.

The last step is to stack these transfer function matrixes along the time axis. For digital signals, the time axis is discrete, so the number of stacked matrixes represent the number of samples.

Electric audio signals can be analog, using varying voltage to represent amplitude against a continuous time axis. However, most modern digital audio formats use Pulse-Code Modulation (PCM). This is the process of digitizing signal, and consists of three stages:

1. Sampling
2. Quantization
3. Encoding

Sampling is the process of discretizing in time, taking regular samples of the instantaneous amplitude at a constant Sampling Rate (SR). The most commonly used SR for audio signals are 44.1kHz and 48kHz.

These values are used to comply with the Nyquist theorem. Given the signal in figure 9, ω_M is the highest frequency component for an analog signal, which in turn determines the bandwidth of that signal. The theorem states that a sampling rate ω_s must be at least $2\omega_M$ in order to adequately reproduce a signal without producing any aliasing effects (OPPENHEIM; WILLSKY; NAWAB, 1996), as shown in comparison of figures 10 and 11. Since human hearing can detect noise up to 20 kHz, this is meant to avoid aliasing.

Quantization is the process of discretizing amplitude, where the values are bounded to a maximum value and any sound wave that stimulates a sensor's membrane may clip to this maximum value. On the other hand, sounds with low intensity will not be detected as it will

Figure 9: Analog periodic function of $\omega\ X(j\omega)$. (OPPENHEIM; WILLSKY; NAWAB, 1996)

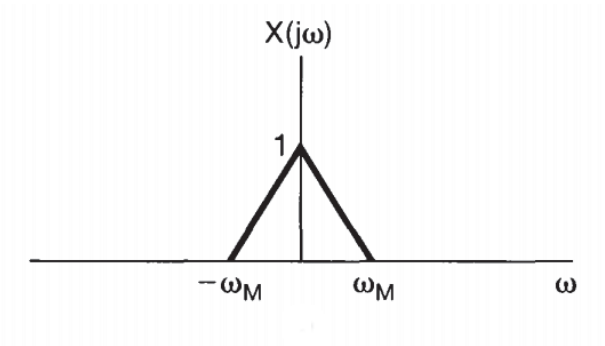
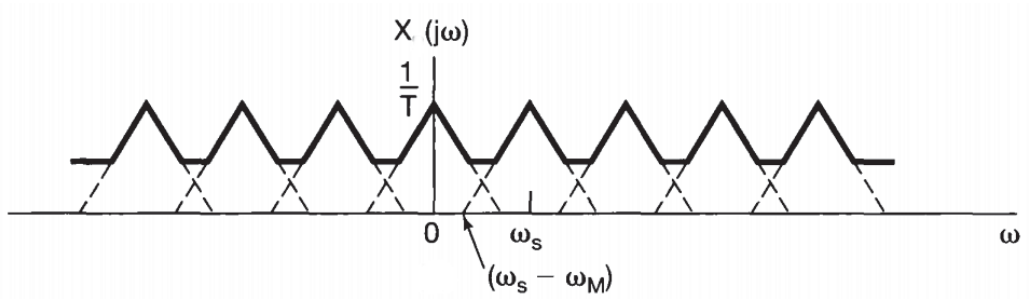


Figure 10: Spectrum of sampled signal with $\omega_s < 2\omega_M$. (OPPENHEIM; WILLSKY; NAWAB, 1996)



be below the very first value different from zero. Bit-Depth is the number of different discrete levels of amplitude the sound may assume, and therefore higher storage space is needed for representing higher resolutions, e.g. 16 bits can store up to 65,536 different levels. Sensitivity is used in tandem with the Bit-Depth to choose optimal results, since increasing the number of possible levels, decreases the smallest value it can detect. Figure 12 shows the digitization process.

Encoding, in turn, is the process transforming this stream of data to a more compact and efficient format. This transformation can be done to lossy formats such as *.ogg* or lossless formats such as *.wav*. Also different codecs may use encode the signal not even using the

Figure 11: Spectrum of sampled signal with $\omega_s \geqslant 2\omega_M$. (OPPENHEIM; WILLSKY; NAWAB, 1996)

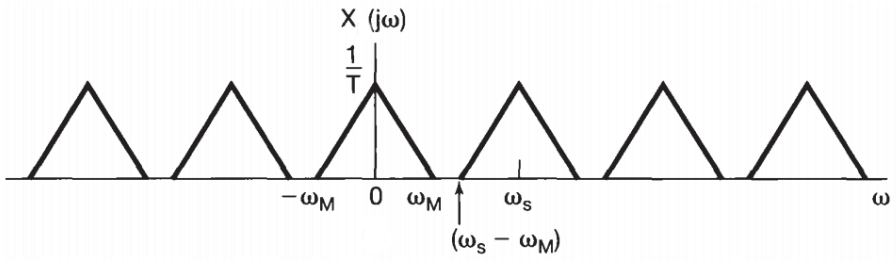
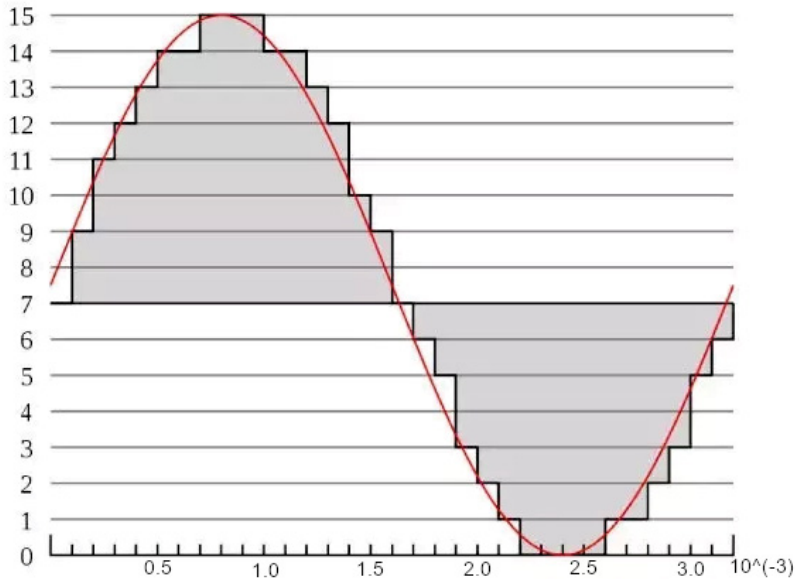


Figure 12: Juxtaposition between an analog audio signal and its digital counterpart. Y axis represents the Amplitude in discrete levels, and X axis represents Time in seconds. (FANDOM, 2014)



PCM process. Some of them use perceptual codecs, which apply Fourier Transform to store data frequency-wise instead of amplitude-wise.

Finally, returning to the classification tasks context, object localization and object detection can be associated to the Acoustic Event Detection task, which looks for timeframes that get an positive association to a given sound source which is taken as the interest event, but it restrains itself to classifying and triggering an event of detection, not actually estimating the relative localization of the interest sound source. Thus, what has to be designed is a classifier to employ the segmentation task in the audio stream. This stream has to be buffered into an optimum buffer size to provide good precision and real-time processing. This is critical for automotive applications due to the nature of its dynamic.

2.1.3 Related works in the field of audio classification models

The classification of sound sources task in an audio signal has already been proposed in the literature using features such as Mel-Frequency Cepstral Coeficient (MFCC) and classifiers such as Hidden Markov Model (HMM) in Durey e Clements (2012) and Chan e Eric (2010) or GMM in Mesaros et al. (2010). (Sigtia et al., 2016) provides a thorough analysis in compromises between performance and computational cost for the most successful and commonly used algorithms for completing the task we are interested in. Sigtia et al. (2016) compares results between Gaussian mixture model (GMM), Support Vector Machine (SVM), and Deep Neural

Network (DNN). The comparison is run on the Acoustic Event Detection task investigating whether DNN are able to outperform classic models when provided large data sets. Two tasks were evaluated using the distinct models: Detecting baby cries and detecting smoke alarms against a large number of impostor sounds. The results from their analysis suggest that GMM provide a good baseline compromise between computational cost and performance, while SVM presented similar performance while having worse computational costs, and since SVM are not parametric models, the employment of kernels are defined by the data, which is not a consistent and reliable way of scaling this solution. Lastly, DNN consistently outperformed both previous models, and since these networks posses a very high quantity of parameters one can employ an optimal model for each specific task and hardware.

Moreover, in recent years a solution that has stood out in the scientific environment is the use of Convolutional Neural Networks (CNN) to accomplish all sorts of acoustic classification tasks. DNN have earned a lot of prominence in competitions for the detection and classification of acoustic scenes and events, such as the Detection and Classification of Acoustic Scenes and Events (DCASE) competitions. The DCASE editions 2013 (STOWELL et al., 2015) and DCASE 2017 (MESAROS et al., 2017), for example, built their baseline algorithms on GMM using Mel-Frequency Cepstral Coeficient (MFCC). However, the winner model in the acoustic scene category Mun et al. (2017) used log banks along with a combination of Recurrent Neural Networks (RNN) and CNN. This shows that currently the best results in the core application aimed by this project really lie in the use of DNN. The question that follows now would be which exact DNN algorithm or combination of algorithms should be used to this application. This shall be discussed in section 3.2.2. Furthermore, there is no consensus on the academic community whether pure CNN or RNN, o mixed models are better suited for audio classification applications. This situation seems to become clearer as we take Choi et al. (2016) in account, stating that it seems like more complex architecture seem to have better performance general, like a parallel CNN+RNN one, we see a compromise between speed and accuracy. This trade-off is even more critical for automotive applications, where real-time is demanding. What we can see as a consesus, however, is the use of Log-amplitude mel-spectrograms as the main feature provider, as can be asserted by Huzaifah (2017) and Cowling e Sitte (2003). These works compared many domain knowledge representations which can be used to provide features for environmental sound classification model training — such as MFCC, Perceptual Linear Prediction (PLP) coefficients, bare short-time Fourier transform (STFT), fast Wavelet transform (FWT) and continuous Wavelet transform (CWT), between many others. The outcome from the afore mentioned works is that for deep learning, differently than classic models using GMMs for example, de-correlation is not pivotal anymore,

presenting Mel-frequency spectrograms consistently performing best in such applications at the time being. This comes with some caveats as window size tuning, as (HUZAIFAH, 2017) reinforces it is dataset dependant and is a parameter to be tuned per dataset. Filter size is not as data dependant but rather representation dependant, as depending on the scale for amplitude and pitch, a representation might be more or less tolerant to parameter invariance. This holds true to mel-spectrograms as both scales are logarithmic.

Related works such as Marchegiani e Posner (2017) and Tadjine et al. (2015) already show results for audio detection applied to the automotive area. In the first work, Marchegiani presents a similar architecture to the current proposal, based on a two-fold system, classifying events (the presence of the emergency vehicle for the given case) on stage one and then performing the location of the sound source on stage two. The first stage greatly differs from the one being proposed by this project, basing their event identification on One-Class Gaussian Process Classification (OCGPC), extracting the signal of interest using a mask, and then performing the classification through Nearest Neighbors, without implementing the of the second stage, where the sound source should be properly localized. Tadjine et al. (2015) approaches the related instrumentation and localization aspects, where it utilizes Microelectro-Mechanical Systems (MEMS) microphones from two Kinect sensors from Microsoft, applying techniques such as Generalized Cross-Correlation (GCC) to perform the localization of the sound source, and this approach will be of great importance for basing the development of the localization contraption in this project.

2.1.4 Basic concepts in Machine Learning

The simple underlying concept of a Machine Learning (ML) algorithm is simply that for any given task T, the performance P is continuously enhanced proportionally to experience E (NG; DROR, 2018). Simply put, similarly to humans, as more good-quality examples are fed to the algorithm, better it performs on the task it has been designed to do.

Thus, this project focuses on applying ML to find patterns in audio signals structure that correlate with classes of objects in the real world.

2.1.4.1 Classes of Machine Learning Algorithms

ML algorithms can be roughly divided into three categories:

1. Supervised Learning

2. Unsupervised Learning

3. Semi-Supervised Learning

Classifiers and Regression algorithms fall on the first category. An algorithm is said to be supervised when the input is built by labeled examples. On the other hand, there is unsupervised learning, which does not relies on labels, but simply groups data that shows similar features in clusters. Lastly, there is semi-supervised learning algorithms which only have partially labeled data, in quantities which may vary. Data points can be correlated to each other and then inferred as belonging to a class depending on its nearest neighbors.

2.1.4.2 Examples and Features

Examples and features relate to each other in a very important way for machine learning algorithms. A good example would be trying to predict houses prices, where we have *m* examples and *n* features. Examples are instances of houses we have obtained information about and Features are concepts that allow us to compare them, as number of bedrooms, total size in squared meters and etc.

$$
\begin{aligned}
x_j^{(i)} &= \text{value of feature } j \text{ in the } i^{th} \text{ training example} \\
\boldsymbol{x}^{(i)} &= \text{the input (features) of the } i^{th} \text{ training example} \\
m &= \text{the number of training examples} \\
n &= \text{the number of features}
\end{aligned}
$$

At the beginning of the training phase a model would know nothing about predicting houses prices based on the house features. We would then provide examples gathered from the real world — newspaper ads, real state agencies online sites, apartments listing sites and so on. Each of these examples contain features from the each house but also they contain the price which is being asked for them. The training process consists of showing to the model each and every example along with its features and price and allowing the model to tweak its parameters to best map features to price prediction. After training, we would be able to use this model to predict a house's price y (the prediction output by the model) of an example never seen before by it based on the features x_j^i where $j = 0$ to $j = n$ (number of rooms and total house size and so on) of each example i it saw at the training phase.

2.1.4.3 Model Representation and Fitting

That said, the core ML framework utilizes another statistical concept for providing information about the subject: Hypothesis testing. Hypotheses strive to fit a model to the data available, and each of them is a candidate model that approximates a target function for mapping inputs to outputs (NILSSON, 1998). For simplicity's sake, an example of hypothesis $\hat{y}$ to be considered is a linear first order relationship for the regression task:

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x \tag{2.3}$$

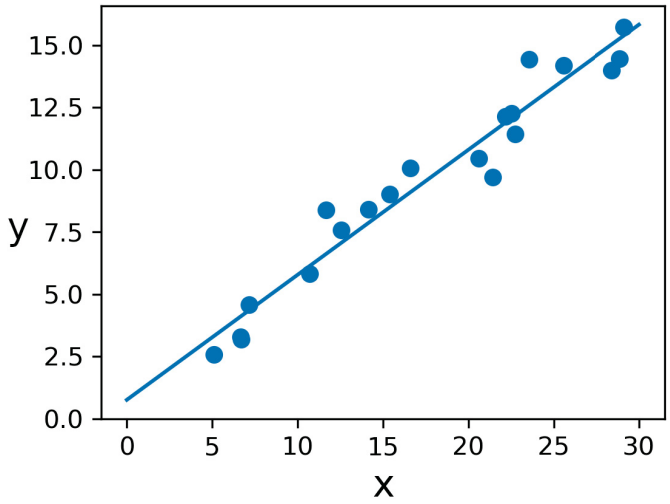
This relationship describes a linear curve on a plane with θ_0 and $\theta_1 x$ as the linear and angular parameters. It is important to note that this curve is generically called a hyper-plane, as in higher dimension spaces, it may assume other geometrical characteristics. In regressions, the model looks for parameters θ — also called weights — that best fit the data presented. This could be useful for predicting future realizations of similar data obeying a similar distribution as the sample used. This tuning is done by using the Cost Function $J(\theta_0, \theta_1)$ using m examples:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \tag{2.4}$$

Literature sometimes uses the nomenclature Loss Function when referring to data points, but these concepts are both highly correlated, and often used interchangeably. The cost function evaluates if a hypothesis is the most suitable one between the many being tried. In equation 2.4 the result of the sum figure is called the sum of the residues, which is the difference between the prediction $\hat{y}$ and the realization y. The Euclidean distance between a data point and its orthogonal projection against the curve is the value which composes the residue. It could also be described as the difference between the ground truth points and the estimated points. Now, it is possible to measure the accuracy of the hypothesis function by using the Cost Function and assess which hypothesis minimizes the outcome of this value, as can be seen in figure 13.

Classical classifiers such as SVM utilizes a kernel — which is still being represented by a hyper-plane which looks to optimally separate two groups of data points. This means that this model will look for parameters θ that maximizes separation between the groups through the minimization of its Cost Function. In figure 14, the curve describes what is the limit between the two classes. Consequently, both θ values have to be tuned to best fit the curve that best segregates the two groups.

Figure 13: Linear Regression example using Gaussian distributed random data. $(x,\ y)$ pair was arbitrarily generated and has no meaning nor unit.



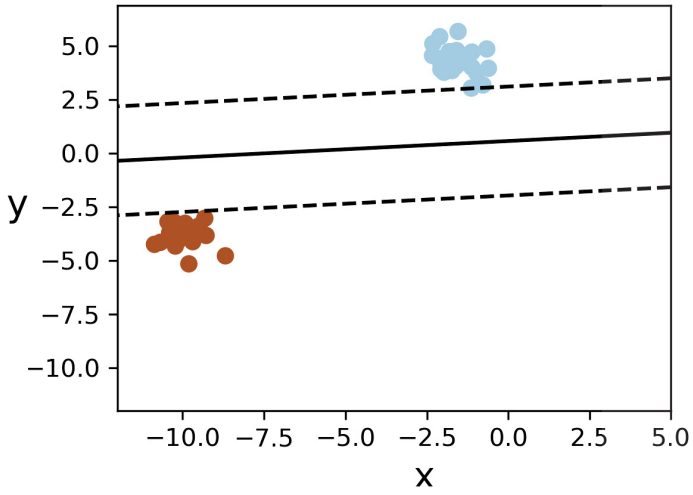
Not only the Euclidean distance has to be used as the Loss Function, but a whole family of error measurement functions. The mean squared error shown in equation 2.5 is usually the standard one used in ML applications, but many other algorithms such as the Cross Entropy can also be employed.

$$MSE = \frac{\sum_{i=1}^{n}(\hat{y}(i) - y(i))^2}{n} \tag{2.5}$$

where n corresponds to the number of examples.

The Learning process iterates over the data set of examples and minimizes the loss function. The loss function alone, however, only evaluates the hypothesis. A new one has to be proposed which would ideally better fit the data points. The algorithm used for this task is named the optimization algorithm, e.g., the gradient descent.

Figure 14: Linear Classification example using Gaussian distributed random data. (x, y) pair was arbitrarily generated and has no meaning nor unit.



This method consists taking a moving differentiation from the last loss function results, and as soon as the derivative plateaus and approximates zero the minimum value for the loss function will have been achieved with all the updated theta values which compose the polynomial expression describing the model. The gradient descent depends on the learning rate α. This is a hyperparamter. It describes how big of a jump the algorithm should take on each iteration. If it is too small, it will take very long to converge, and if too expressive, may overstep the minima and instead of converge, start to diverge unsteadily. That is, each iteration would take the form of

$$
\begin{aligned}
&\text{repeat until convergence: } \{ \\
&\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)} \\
&\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)} \\
&\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)} \\
&\dots \\
&\}
\end{aligned}
$$

with all parameters θ simultaneously updated.

Using vectorized operations are much more computationally efficient than element-wise operations (NG; DROR, 2018):

repeat until convergence: {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j := 0...n}$$

}

2.1.4.4 Bias and Variance Trade-Off

For a number of reasons, a model may overfit or underfit.

Underfitting occurs when a model is under-trained (left side of figure 15). In cases of underfit models they are considered biased model. While bias still high, the model has not learned enough about the data, and will struggle to get accurate results, however it will still be precise.

An algorithm can also overfit data (right side of figure 15). Even if intuitively it would make sense to train as much as possible, it would be a problem if the model would simply memorize the data, and lose the capacity to generalize what the underlying concept the data really has. This problem is represented by high variance. In this case, accuracy would near 100%, although with low capacity of generalization precision would be low.

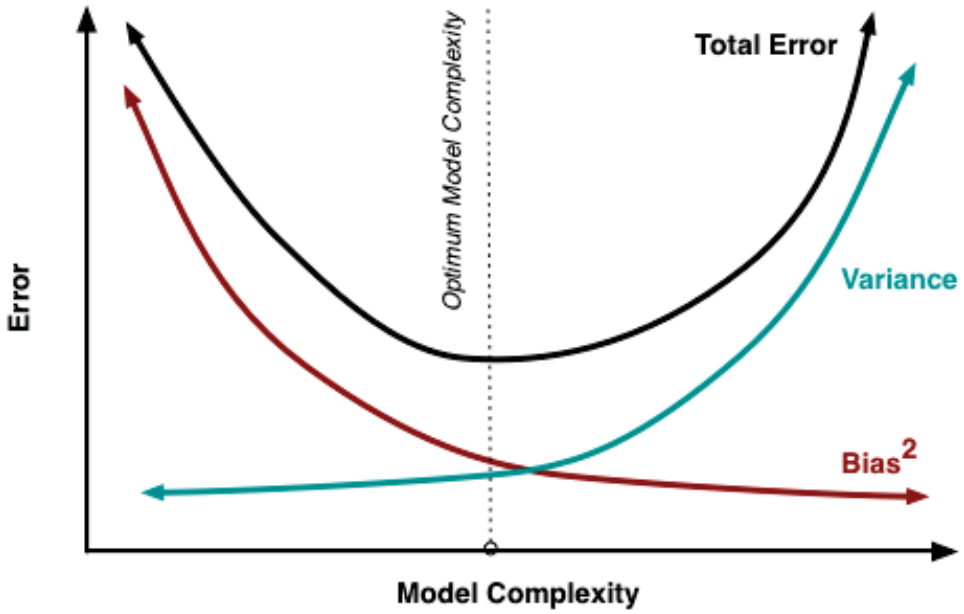
2.1.4.5 Cross-validation

A hypothesis may have a low error for the training examples but still be inaccurate (because of overfitting). Thus, to evaluate a hypothesis, given a dataset of training examples, we can split up the data into two sets: a training set and a test set.

Typically, the training set consists of 70 % of data and the test set consists of the remaining 30 % (NG; DROR, 2018).

Just because a learning algorithm fits a training set well, that does not mean it is a good hypothesis. It could overfit and as a result the predictions on the test set would be poor. Given that, the error of the hypothesis as measured on the data set with which we trained the parameters would be lower than the error on any other data set. Understanding this behavior helps to identify situations when the model is too complex to be manually analyzed or even inaccessible to the user.

Figure 15: Representation of the principle of the Variance versus Bias Trade-off (NG; DROR, 2018)



Given many models with different polynomial degrees, we can use a systematic approach to identify the most suitable function. A good methodology for choosing the best model complexity for our hypothesis would be test each polynomial degree and thereafter look at the resulting error.

Another way to break down our dataset into the three sets would be:

- Training set: 60 %
- Cross validation set: 20 %
- Test set: 20 %

For cross-validation there are many different approaches available. We could try different θ values (weights or polynomials, depending on the algorithm). Also, try different learning rates, regularization or any other hyperparameter.

However, a very important approach using the validation set and the cross-validation approach is to effectively utilize all data instead of only train on 70 %, for example.

We could achieve that using the k-fold or the *Leave One Out* techniques. The former consists in shuffling and then dividing the dataset in k different parts. If we consider $k = 10$, we would divide the data in 10 parts, then train using 9 parts out 10. Afterwards, with the

model accordingly trained, we test in against the one part left out and write down the results. The technique then requires it to be repeated for all 10 parts, distinctly leaving one out each time.

In the end, we would have a great sum of results against these 10 different development sets used in each iteration, depicted by figure 16. A recommended course of actions is to synthesize the results as they were the output of one single training and validation routine (NG; DROR, 2018). That way, it would be possible to assembly a confusion matrix that used all the data, and can be used for performance measurement the same way as in conventional validation routines.

Figure 16: 10-fold cross-validation illustration. (NG; DROR, 2018)

2.1.4.6 Classification Evaluation

Since classification models are validated applying labels to data which is already labeled, it is possible to measure how good or bad a model is. However, this measurement has to be quantifiable, so analytically it would be possible to choose the best one from the many being tested. This is the role of performance metrics, which are fundamental for gaining important insights on a model behavior when validated or tested.

The most fundamental concept in evaluation is the assembly of a confusion matrix (KOHAVI; PROVOST, 1998).

For binary classification, a confusion matrix would be composed of True Positives (TP), False Positives (FP), True Negative (TN), and False Negatives (FN) (KOHAVI; PROVOST,

Table 1: Confusion matrix structure.

		Predicted Class	
		Negative	Positive
Actual Class	Negative	**a**	**b**
	Positive	**c**	**d**

1998). For contextualization, let us use a fire alarm example: If there is a fire detected by the system the alarm should go off, otherwise the alarm should remain silent. When there is a fire, it is a positive event: The event of a fire happened. That means there are a total of 4 combinations possible, which can more generally be described as:

- True Positives: The event happened and it was detected
- True Negatives: The event did not happen and it was not detected
- False Positives: The event did not happen but it was detected as it had happened
- False Negatives: The event happened but it was not detected as it had happened

This allow us to then assemble the confusion matrix, which takes the form seen in table 1. **a** represents the TN values, **b** represents the FP values (also known as Type-1 errors), **c** represents FN values (also known as Type-2 errors), and finally **d** represents TP values.

Metrics are very important in both business and engineering world, since they actually allows this measure the gap between the real world objective and the proxy training objective, set by the Cost Function (NILSSON, 1998).

The most important performance metrics are (ZHENG, 2015):

- *Accuracy* is the proportion of the total number of predictions that were actually correct:

$$AC = \frac{a + d}{a + b + c + d} \tag{2.6}$$

- *Sensitivity* (also known as both Recall or True Positive Rate) is the proportion of positive cases that were correctly identified:

$$TP = \frac{d}{c + d} \tag{2.7}$$

Table 2: Confusion matrix structure for 3 class classification.

		Predicted Class		
		Class 1	Class 2	Class 3
Actual Class	Class 1	**a**	**b**	**c**
	Class 2	**d**	**e**	**f**
	Class 3	**g**	**h**	**i**

- *Specificity* (also known as True Negative Rate) is defined as the proportion of negatives cases that were classified correctly:

$$TN = \frac{a}{a+b} \tag{2.8}$$

- *Precision* is the proportion of predicted positive cases that were correct:

$$P = \frac{d}{b+d} \tag{2.9}$$

If more than two classes were available, not only the training procedure would change, but also the confusion matrix itself would as well. An example of a three classes classification confusion matrix can be seen in table 2. For these cases each of the metrics would have to be extracted for each single class, comparing correct predictions against wrong prediction, i.e., any other prediction that was not the given class being currently analyzed.

2.1.5 Artificial Neural Networks and Deep Learning

2.1.5.1 Artificial Neurons

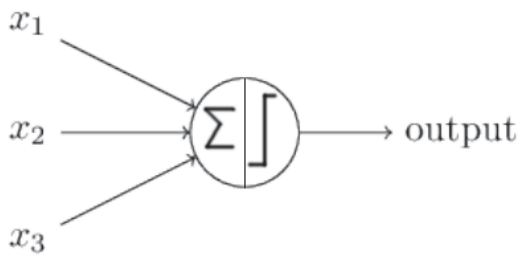
A Artificial Neural Networks (ANN) is a mathematical model, and also a computational model which is composed of a set of neurons organized in layers. This is a biologically inspired model to mimic our brain structure. Simplifying the composition of neurons, they have dendrites, a cell body, and an axon.

The idea of combining neurons to form a network has already been long established now (dating back to 1959, with Bernard Widrow and Marcian Hoff in Stanford). Making a parallel between the biological and the artificial neuron we have the following:

1. The dendrites: The model input connections
2. The cell body (or Soma): The linear calculation and the activation functions
3. The axon: The output connections

On a high level, these are the only parts composing artificial neurons, or as they are also often called — perceptrons. In order to summarize these concepts through a illustration, perceptrons use their dendrites to take in inputs (x_1, x_2, x_3 in figure 17), use their cell body to apply the linear combination operations to all these inputs and then applies a non-linear activation function, and lastly use their axon to output the results (NIELSEN, 2015).

Figure 17: A simple perceptron representation. (NIELSEN, 2015)



The operations carried through the cell body are the multiplication of each input by a respective weight, then all the results are summed and submitted to a function. This function has a particular name – the activation function. This name comes after the biological neuron counterpart, where during the transmission of processing of information in our brains, some cells are not activated and therefore will not pass forward the electrical pulse they might receive. Therefore, to mimic this behavior we apply an activation function. So, the mathematical operations for a single perceptron are described by equation 2.10.

$$z = \boldsymbol{w}^T \boldsymbol{x} + b \tag{2.10}$$

where z is the output, x is the input vector, w is the weight vector, and b is the bias. Bias is a special weight that require no input, and this it corresponds to the output when there is zero input. It represents an extra neuron included with each pre-output layer and stores the value of '1', helping the model to represent patterns that do not necessarily pass through the origin. That way if all features would have value 0 associated to them, it offsets the model to assume some base value. Bias is not part of the perceptron itself, but a unit from the network layer. Despite having output z we still have to process this input through the activation function. The most commonly used activation function is the ReLu function, described by equation 2.11.

$$g(z) = \max(0, x) \tag{2.11}$$

The function itself is quite straight forward, if the sum output z is a positive value, the ReLu output g responds linearly (NG; DROR, 2018). The reason why this function is so

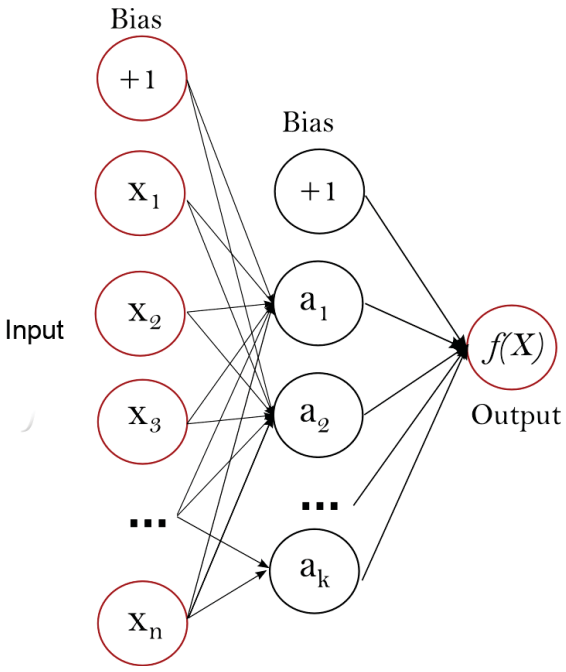
commonly used is because it adds non-linearity to the input output mapping. The output h from the single perceptron is described by equation 2.12, which is the output value seen in figure 17. It is worth noting different activation functions work better for different problems and architectures, as a classical example being the sigmoid function, which is the classical activation function used in logistic regression.

$$h = g(z) \tag{2.12}$$

2.1.5.2 Multi-layer Perceptrons

When stacking neurons, each connection that provides inputs for z configure a layer. A layer may not consist of a single neuron, but have j units with each one of them connecting to all the units in the following layer. However, the concept of a single-layer Neural Network (NN) is established by the presence of three layers: An Input layer, a Hidden layer, and an Output layer. This structure can be seen in figure 18.

Figure 18: Neural Network representation. (NIELSEN, 2015)

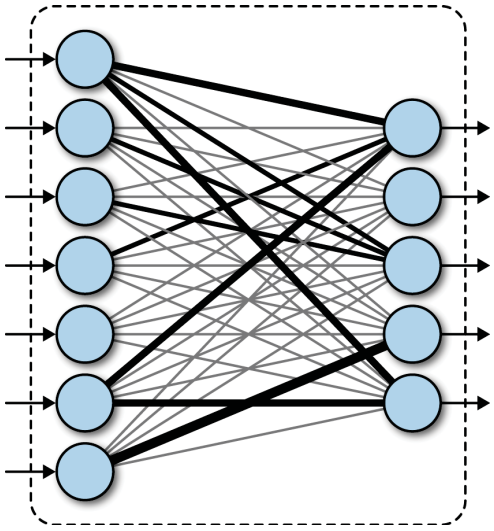


A single perceptron is able to output predictions, however the complexity of problems it can solve are really low. If these functions were linear, we would be able to simplify all the mathematical operations to a single linear combination (GOODFELLOW; BENGIO; COURVILLE, 2016). This way, more layers and more units in a network actually translate into model complexity.

The problem is we are limited to an input layer, a hidden layer and an output layer. Well, this concept can be manipulated unfolding the hidden layer — a hidden layer has this name because we are not able to understand what is the logic behind the weight vector, taking it as a black box. So what we can do is further develop this black box, since we only care about its input and output, and place as many hidden layers in the model as the problem to be solved needs. Furthermore, this is exactly what taking figure 18 and replacing its hidden layer with one or more layers shown in figure 19 would achieve. This operation is what distinguish a regular NN and a DNN — when all units are connected to all the units in the previous layer this is called a Multilayer Perceptron (MLP). This greatly improves our capacity to better fit the training data with much more complex models, but computational complexity also increases. This is why Deep Learning (DL) has seen a big increase in popularity in the last years in many different ML applications, and this project is no exception to that. Equation 2.10 was used for a single perceptron, but for more general units description within the DNN, equation 2.13 is used to relate the vector $\boldsymbol{w}$ of weights, the vector $\boldsymbol{x}$ of outputs incoming from the last layer $i-1$, the bias from last layer $i-1$ and the unit output z. Note that subscripts j indicates the j^{th} element of the hidden unit, while superscripts i indicate the i^{th} layer of the network.

$$z_j^{[i]} = \boldsymbol{w}_j^{[i]^T} \boldsymbol{x} + b_j^{[i]} \tag{2.13}$$

Figure 19: Two Fully Connected Layers with different line thicknesses representing different weights (RAMSUNDAR; ZADEH, 2018).



Furthermore, it is possible to consider bias units function once again. They are not tied to any previous layer in the network, so they don't represent any form of activity, but are treated the same as any other weight. Without a bias node, no layer would be able to produce an output for the next layer that differs from 0 if the feature values were 0 (NG; DROR, 2018).

After all, what bias nodes do is help networks solve more types of problems by allowing them to employ more complex logic gates.

NNs map input to output through forward propagation, and after that compares the output with the ground truth provided during the training phase to employ back-propagation (GOODFELLOW; BENGIO; COURVILLE, 2016).

A NN makes a prediction propagating the input data forward through the network, layer by layer, until it reaches the final layer which outputs a prediction. The mathematical model is a combination of successive operations applied through the entire network. In simple input-hidden-output layers networks equation 2.14 shows how forward propagation is executed layer-wise, providing prediction $Pred.$ $A()$ is the application of the activation function, $\boldsymbol{x}$ is the vector of inputs, $\boldsymbol{w}_h$ is the weights vector applied to the hidden layer, and $\boldsymbol{w}_o$ is the weights vector applied to the output layer. With each additional layer, more additional activation functions would be applied and also more weight vectors would be added to each one of those (NIELSEN, 2015).

$$Pred = A(\ A(\ \boldsymbol{x}\boldsymbol{w}_h\ )\boldsymbol{w}_o\ ) \tag{2.14}$$

To learn, a DNN adjusts its weights once per batch during the training phase, where additionally to forward propagation also back-propagation takes place. After the network has finished training, it does not perform the latter. Back-propagation is carried comparing the actual output and the desired output (NG; DROR, 2018). As seen in section 2.1.4 equation 2.4 used the mean squared error in equation 2.5 to make this comparison. In DL instead of using equation 2.4 and the MSE (which would only make sense for solving problems unrelated to Classification) though, cross-entropy usually perform better as a Loss Function, which is expressed as:

$$L(z, y) = -\Big[y \log(z) + (1 - y) \log(1 - z)\Big] \tag{2.15}$$

With this updated Cost Function, we start by applying the chain rule, where:

$$f(x) = A(B(C(x))) \tag{2.16}$$

can be expressed as:

$$f'(B) = f'(A) \cdot A'(B) \tag{2.17}$$

and using this property we can take:

$$Cost = L(A(z(\boldsymbol{x}\boldsymbol{w}))) \tag{2.18}$$

and obtain:

$$\begin{aligned} L'(\boldsymbol{w}) &= L'(A) \cdot A'(z) \cdot z'(\boldsymbol{w_i}) \\ &= L'(\boldsymbol{w}) \cdot A'(z) \cdot \boldsymbol{H}_{i-1} \end{aligned} \tag{2.19}$$

where $\boldsymbol{H}_{i-1} = A(\boldsymbol{w}_{i-1}\boldsymbol{x}_{i-1})$ is the output from layer $i-1$ (also known as the layer $i-1$ current hypothesis). Since from equation 2.19, we get the term leading to the $\boldsymbol{H}_{i-1}$ we could differentiate in relation to even farther weight vectors towards the beginning of the network, even getting to layer $i = 1$. Therefore, generalizing this idea we obtain:

$$\frac{\partial L(z,y)}{\partial w} = \frac{\partial L(z,y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w} \tag{2.20}$$

This way, we would have access to the contribution each weight has regarding the Loss Function through the gradient being back propagated. That means if we only wanted to obtain the corresponding loss for weight vector corresponding to the last layer it would be much easier on the computational demand. In the other hand, for obtaining the corresponding loss for the weights of the first layer I would have to calculate the gradient of every single layer that came after it. So we would have to calculate the entire gradient either way, which is really computational demanding. Equation 2.21 summarizes the update routine, with α being the learning rate parameter which has been explained in section 2.1.4.3 (NG; DROR, 2018).

$$w := w - \alpha \frac{\partial L(z,y)}{\partial w} \tag{2.21}$$

From equation 2.21, it is possible to note that both the Loss Function and the Learning Rate are hyper-parameters that can and directly impact how training is carried on DL frameworks.

2.1.5.3 Convolutional Neural Networks

CNN are an especialized form of DNN. They assume special spatial structure in its input. In particular, it assumes that inputs that are close to each other spatially are semantically related. This assumption makes most sense for images, since pixels close to one another are likely semantically linked. As a result, convolutional layers have found wide use in deep architectures for image processing (RAMSUNDAR; ZADEH, 2018). That is to say the biggest differences between MLP and CNN are the implementation of convolutional layers and the connections of units between two layers.

Mathematically, CNN provide tools for exploiting the local structure of data effectively (RAMSUNDAR; ZADEH, 2018). Parts of an image that are close to one other in the pixel grid are likely to vary together (for example, all pixels corresponding to a table in the image are probably brown). These networks learn to exploit this natural covariance structure in order to learn effectively. Therefore, CNN are not fully connected throughout the hidden layers (PATTANAYAK, 2017). Lastly, the final output will be reduced to a single vector of probability scores called logits, allocated along the depth dimension (RAMSUNDAR; ZADEH, 2018).

Input An image is composed of the dimensions $height \times width \times channels$. $Height \times Width$ is the Resolution of the image, and the number of channels (usually Red, Green and Blue channels) can be 3 (R,G,B), 1 (grayscale), and etc. For other applications other than images we could call Channels "Depth". To our application, the analogy would be represented by the dimensions frequency (or timeseries), amplitude and audio channels (stereo would have two channels, for example).

Local Receptive Fields The local receptive field concept originates in neuroscience, where the receptive field of a neuron is the part of the body's sensory perception that affects the neuron's firing. Neurons have a certain field of view as they process sensory input that the brain sees. This field of view is traditionally called the local receptive field, and could be interpreted as a patch of skin or to a segment of a person's visual field.

Convolutional Kernels (or Filters) A convolutional layer applies nonlinear function to a local receptive field in its input. We have the *stride size* of the kernel which controls how the receptive field is moved over the input. It could move one, two or N tiles at a time (it's like a matt moving under a fixed format, and how much the matt moves is the stride). the filter size

f is usually odd, that is because otherwise the padding would be assymetric and it wouldn't have a centered value.

Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters (Kernels).

Pooling Through empirical use the scientific community realized max pooling helps because it extracts the sharpest features of an image (RAMSUNDAR; ZADEH, 2018). So given an image, the sharpest features are the best lower-level representation of an image. But even according to Andrew Ng, max pooling works well but no one knows why (LI; JOHNSON; YEUNG, 2018).

Softmax Function Softmax is useful because it maps [-inf, +inf] to [0, 1] similar as a Sigmoid Function would. But Softmax also normalizes the sum of the values(output vector) to be 1. This function consumes the last layer of weights being propagated forward, which are called logits. This name comes from the fact we are mapping weight values without bound to probabilities.

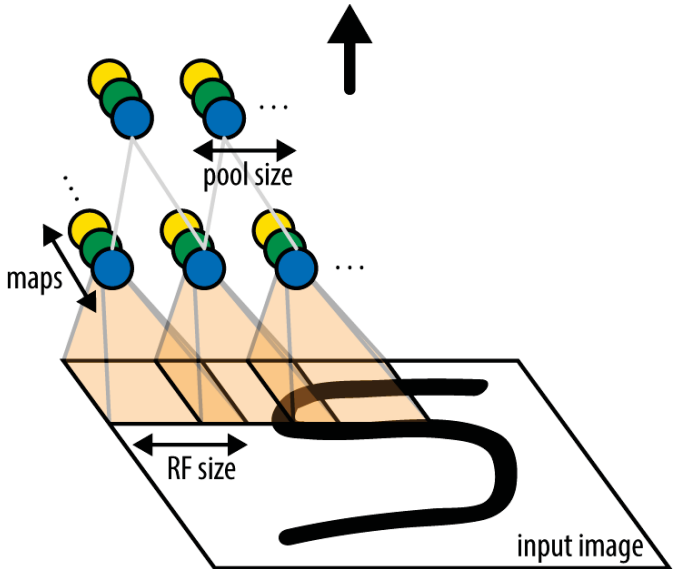
Architecture

1. Feature extraction: In this part, the network will perform a series of convolutions and pooling operations during which the features are detected. If we had a picture of a zebra, this is the part where the network would recognise its stripes, two ears, and four legs (its lower level features).

   We retrieve the Local Receptive Field — each of these image patches will be handled by a separate perceptron. A nonlinear transformation is applied to incoming data through the neuron (which originates from the local receptive image patch) (PATTANAYAK, 2017).

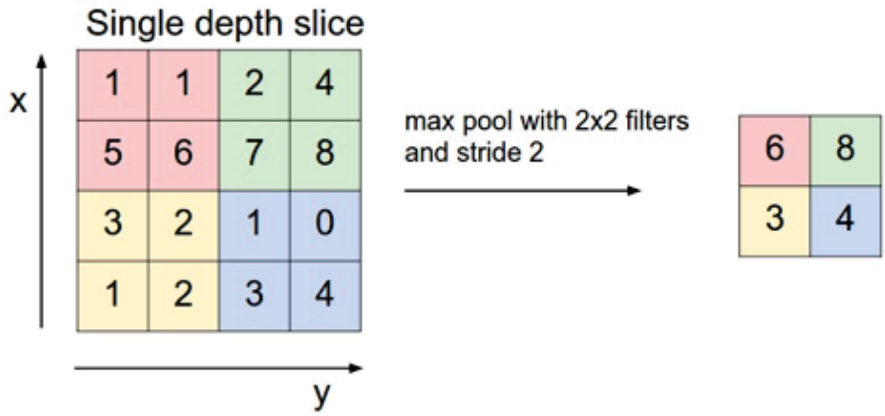
   (a) Convolutional Layer: The convolution is performed on the input data with the use of a filter to then produce a **feature map**. We execute a convolution by sliding the filter over the input. At every location, a matrix multiplication is performed and sums the result onto the feature map. Convolution is one of the main building blocks of a CNN. The term convolution refers to the mathematical combination of two functions to produce a third function. It merges two sets of information. Units map learned features and the layer put them together as the final output of the convolution layer.

Figure 20: Iterations taken between the Kernal and the input (LI; JOHNSON; YEUNG, 2018).



(b) Pooling Layer: After a convolution layer, it is common to add a pooling layer in between CNN layers. The function of pooling is to continuously reduce the dimensionality to reduce the number of parameters and computation in the network. This shortens the training time and controls overfitting. The most frequent type of pooling is max pooling, which takes the maximum value in each window. These window sizes need to be specified beforehand. This decreases the feature map size while at the same time keeping the significant information. Average Pooling or Sum Pooling could be used as well.

Figure 21: CNN Pooling Layer Illustration (LI; JOHNSON; YEUNG, 2018).



(c) Recursively Stack Layers.

2. Classification: Now, the fully connected layers will serve as a classifier on top of these extracted features. They will assign a probability for the object on the image being what the algorithm predicts it is (LI; JOHNSON; YEUNG, 2018).

   (a) Flatten: After the convolution and pooling layers, our classification part consists of a few fully connected layers. However, these fully connected layers can only accept 1 Dimensional data. To convert our 3D data to 1D, we use flatten the matrix, collapsing each vector from axes 2 and 3 into one dimension. This essentially arranges our 3D volume into a 1D vector.

   (b) Fully Connected layer: The last layers of a ConvNet are fully connected layers, so that means neurons have full connections to all the activations in the previous layer. This part is in principle the same as a regular NN seen in section 2.1.5.2. With the fully connected layers, we combined these features together to create a model.

   (c) Output layer: Finally, we have an activation function such as softmax or sigmoid to classify the outputs nouns as cat, dog, car, truck etc. For the project's application, the project would output the binary positive or negative class to the ambulance siren at this point. This will maps how probable it is that the image belongs to a class then.

We could even make use of transfer knowledge if we would like to use another classifier instead of the fully connected NN which is concatenated to the end of the feature extraction layer. They are complex due to the extensive amount of parameters available for tuning, but they are also powerful. This seems to be perfect for audio applications due to the complexity of audio features retrieval, it is more difficult to gather features compared to image and video retrieval. As the original audio data is made up of non semantic and non structured binary stream, it lacks of semantic description and structured organization (RONG, 2016). CNN are suitable for this application, as they can filter and extract semantic content from the unstructured data which would be otherwise difficult to manually work with.

2.2 Acoustic Source Localization

Acoustic Source Localization (ASL) is the estimation of the sound source position taking in account the acoustic signal the it generates. Many researches on the subject were carried out on the field of robotics as it provides a way for them to do an intelligent surrounding

environment analysis. The whole idea is to make robots and systems to benefit from hearing as we humans do.

Localization algorithms are closely related to microphone configuration. There are four main localization strategy groups (MARKOVIĆ; PETROVIĆ, 2010). The first group of algorithms are beamforming methods in which an array of microphones is steered to various locations of interest, searching for peaks in signal power. The second group are beamforming methods based on spectral analysis done via a correlation matrix which associates spatial space to power and outputs likelihood values. The third group are computational simulations algorithms of the physiologically known parts of the hearing system. The fourth and last group is comprised of localization strategies based on estimating the Time Difference of Arrival (TDOA) of audio signals matching other elements in the microphone array. Since the elements of the array are arranged with a relative distance between them, it is possible to associate the latency between detection of the signal among these elements and the source location. The choice of the author for using TDOA based algorithms is justified by the fact they that even though they are not the most accurate in comparison to other state of the art techniques (Badali et al., 2009), it is simpler and easier to implement. Since the current project has tight time constraints, and documentation on TDOA implementations is plenty this was an obvious choice.

TDOA algorithms take a whole array in consideration, but iterate between microphone pair combinations at a time. This means even a two-microphone array composition is possible. The biggest problem is that it will never be possible to triangulate the source location, and its position will always be ambiguous, since it will not be able to tell if the delay detected come orthogonality form the front or the back side of the array. That means that with two microphones the Direction of Arrival (DOA) is ambiguous for a 2-dimensional plane, but many other possibilities open up with we consider 3-dimensional spaces. For that cause, 3-microphone arrays and above are required. After the microphones receive the data, it is then necessary to process them. This processing phase looks to infer from some *a priori* knowledge, the source location.

That said, using the observed time differences between audio signals arriving at the microphone pairs, we can constrain maximum and minimum delays, as a sound source placed ortoghonally to the microphone pair will produce near to zero time difference and on an opposite case, a sound source placed on top of the line connecting the two microphones will produce a maximum delay, which is described by the distance between the pair. The first step is to calculate the time difference estimation. There are many methods that could be used,

each in varying degrees of accuracy and computational complexity. The basic method is the calculation of Cross Correlation between signals received by two different microphones, and finding which value of delay (or phase) maximizes the correlation between these two processes. The GCC method is an improved method for the Cross Correlation. To obtain the estimated time difference, we need to consider two sensor inputs (Carter, 1987):

$$r_1(t) = s(t) \tag{2.22}$$

$$r_2(t) = s(t - D) \tag{2.23}$$

where s(t) is the time variant audio signal process and D is the actual time delay. The cross correlation can be calculated as the product of the expectation:

$$R(\tau) = \mathbb{E}[r_1(t)r_2(t + \tau)] \tag{2.24}$$

Finally, the estimated correlation peak $\hat{\tau}$ will converge to D when the cross correlation is maximum:

$$D = argmax\ R(\hat{\tau}) \tag{2.25}$$

When the correlation function is more sharply peaked, performance improves. Using some kind of weighting could lead to performance improvements. A weighting technique called Phase Transform (PHAT) normalizes the signal spectral density by the spectrum magnitude. A implementation for calculating the PHAT cross correlation in discrete frequency domain using a audio samples within a window could be done as follows:

$$\hat{R}_{r1,r2} = \sum \frac{X_{r1}(k)X^*_{r2}(k)}{|X_{r1}(k)||X_{r2}(k)|} e^{j2\pi \frac{k\tau}{a}} \tag{2.26}$$

where X_{r1} and X_{r2} are respectively the microphone inputs in the frequency domain.

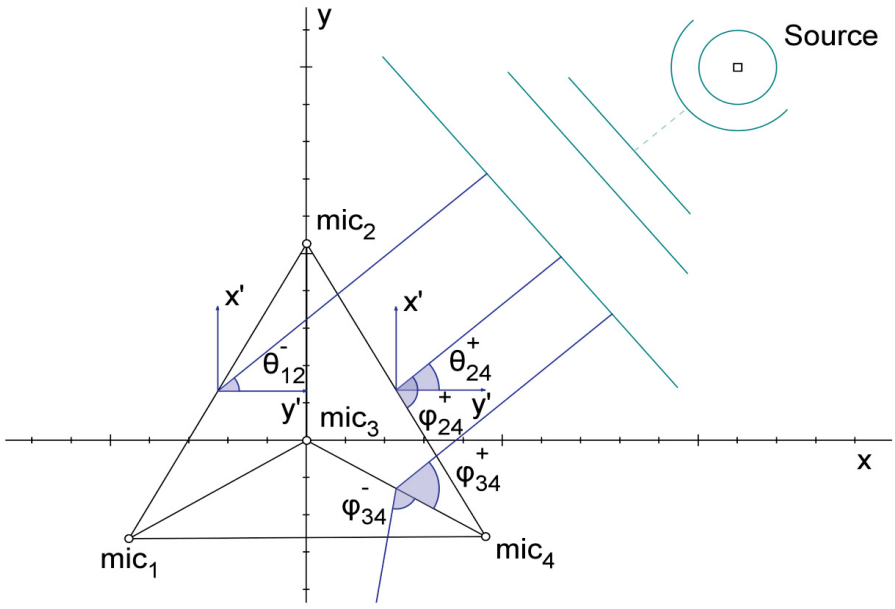
GCC-PHAT however shows some potential issues. GCC with PHAT weighting is indiscriminately weights all frequency bins regardless of signal Signal-to-Noise Ratio (SNR), and therefore makes the system less robust to noise. Also, GCC-PHAT shows multiple sharp maxima and the direct path maximum is not necessarily the strongest one causing ambiguity in via multiple possible paths. Lastly, when multiple sources are present GCC-PHAT shows multiple maxima and it is difficult to group the TDOA originating from the same source.

Since we know the distance between our microphones L, we can then transform tau into a relative number instead of an absolute delay using $\tau_{max} = L \div c$, where c is the sound speed in the medium, and thereafter $\tau_{relative} = \tau \div \tau_{max}$.

After calculating the time difference value, we can obtain the DOA by using an far-field simplification (MARKOVIĆ; PETROVIĆ, 2010). ϕ is represents the aperture from the line connecting the microphone pair, and since it could be either clockwise or counter-clockwise, we have two possibilities for each ϕ — ϕ^+ and ϕ^-, as seen in equation 2.27. Figure 22 shows how parameters ϕ and θ relate.

$$\phi_{r1,r2} = \pm arccos(\tau_{relative}(r1, r2)) \tag{2.27}$$

Figure 22: Example of interaction between the wavefront and the possible parameters θ and ϕ values (MARKOVIĆ; PETROVIĆ, 2010).



This would be enough if we would be using only two microphones, and we would fall the in the ambiguity case. However, the following transformation in order to fuse the estimated DOA:

$$\theta^{\pm}_{r1,r2} = atan2(\frac{y_{r2} - y_{r1}}{x_{r2} - x_{r1}}) \pm arccos(\tau_{relative}(r1, r2)) \tag{2.28}$$

where the angles $\pm\theta$ are the ambiguous angle between the orthogonal interception of the line connecting microphones r1 and r2, and the reference line. It makes it easier for humans to

visualize the aperture in relation to trigonometric circle. (x_{r1}, y_{r1}) and (x_{r2}, y_{r2}) pairs are the microphone (x, y) coordinates on the proving ground.

Finally, we would need to solve the front-back ambiguity. This can later on be solved using heuristics as a way to simplify the problem instead of using a statistical framework. Of course, it would offer much less robustness in exchange of implementation simplicity.

3 Methodology

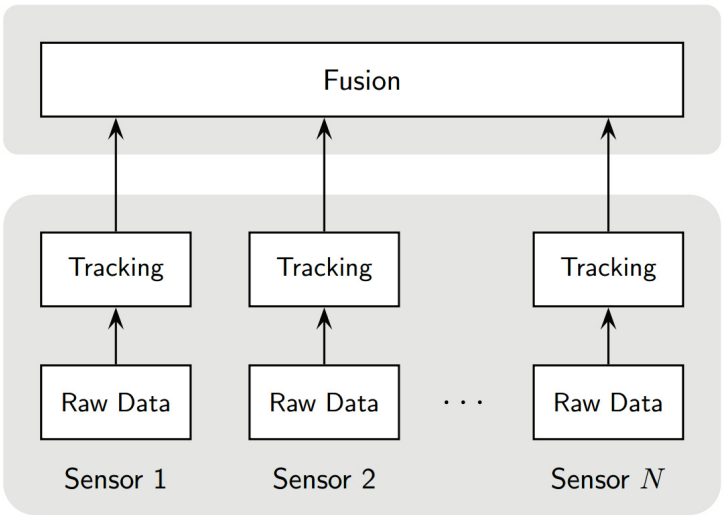
This project proposes the development of a system for classification and thereafter the sound source localization for the audio signal coming from emergency vehicles inserted in real urban auditory scenes. For accomplishing this objective, we will break the problems down and provide modular and as simple as possible systems for outputting what is necessary for each proposed specific objective.

For automated driving applications, a architecture should ideally encapsulate each sensor that provides information to be interpreted by the data fusion layer (AEBERHARD, 2017). This leads us to the point where no matter what sensors are available for a vehicle model, the same rest of the system would be able to fuse them based on data they provide. This kind of abstraction is important as it allows us not to depend on a specific implementation for a specific model of vehicle, a specific hardware, or specific firmware version. This means in this context, we should not rely on microphones being detected or integrated to a given system, but rather comply with the demands the architecture has, therefore passing over parameters which do not know or depend on which actual sensor they are being provided from as illustrated in figure 23.

Furthermore, at a sensor level, what Aeberhard (2017) proposes is an architecture as seen in figure 24. This architecture outputs an object list $\mathcal{O}$ at sensor level, i.e., each sensor shall output its own list independently of any other sensor present at this same level. This list contains information about the current state of all the objects being detected at moment t. As seen in figure 24, objects at sensor level can be described by their statistical descriptors:

- $\hat{\boldsymbol{x}}^S$: Estimated state vector
- $\boldsymbol{P}^S$: State covariance matrix
- $\hat{\boldsymbol{d}}^S$: Estimated object dimension vector
- $\hat{\boldsymbol{d}}_\sigma^S$: Dimension uncertainty vector

Figure 23: Data fusion architecture with decentralized tracking algorithms. (AEBERHARD, 2017)



- $\boldsymbol{f}^S$: Feature vector
- $\boldsymbol{c}^S$: Classification vector
- $p^S(\exists \mathrm{x})$: Probability of existence

where state vector $\hat{\boldsymbol{x}}^S$ contains the following parameters for modeling each object:

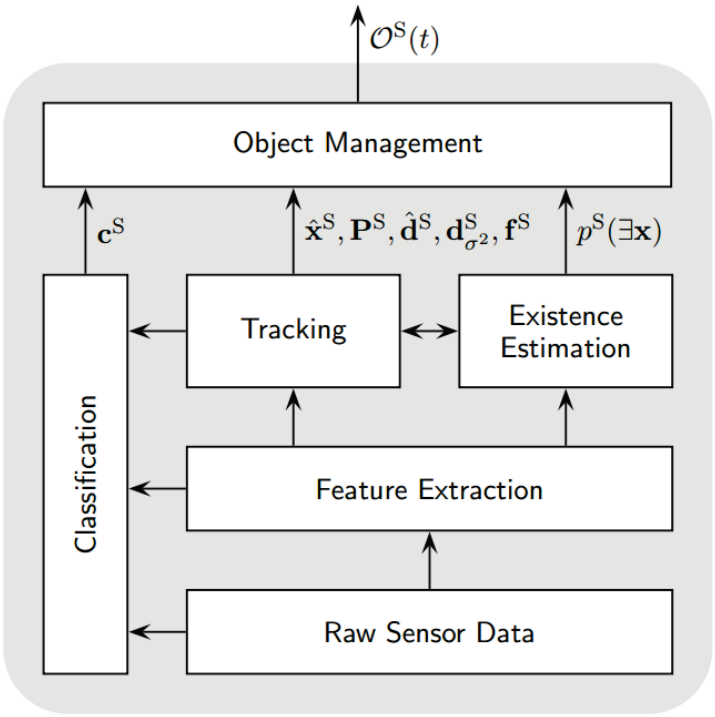
$$\hat{\boldsymbol{x}}^S = [x \quad y \quad v_x \quad v_y \quad a_x \quad a_y \quad \psi \quad \psi'] \tag{3.1}$$

where x and y are the object's geometrical coordinates of the center of mass position, v_x and v_y are the absolute velocity, a_x and a_y are the absolute acceleration, ψ_x is the orientation, and ψ' is the orientation change rate.

From this framework proposed by Aeberhard (2017), it is up to us to pick some of these parameters for our prototype. Since we are not interested in Tracking an indiscriminate object but rather classifying and localizing single emergency vehicles, the service being proposed will have to output the classification vector $\boldsymbol{c}^S$ and a simplified version of $\hat{\boldsymbol{x}}^S$ for describing object position. Therefore, due to time constraints, what is proposed by this work is to describe the estimated object position direction via the azimuth α. Also, since we are restricted to single objects we shall simplify it classification vector to c^S.

The specific objectives are described in the Figure 25. Yellow boxes indicate that these two stages should operate on compatible number of audio channels in a real application, but for this project these phases operate on different number of channels. The filtering stage

Figure 24: Sensor level object tracking architecture. (AEBERHARD, 2017)



simply aims to decouple Very Low Frequencies outside human hearing spectrum with a High Pass Filter with cut-off frequency at 200Hz. For this work, classification is proposed for mono audio streaming, herein implemented to ease data collection. On the other hand, for localizing sound sources we need multiple channels, so for that matter multi-channel data was mocked directly into its model.

From 25, the idea proposed is to isolate functionalities by self-contained projects. For this prototype, three sub-projects will be developed, as shown in figure 26.

The first sub-project — called `audioset-download-tool` – is a set of tightly coupled entities and methods designed to select, download, log, normalize parameters, and finally organize the files onto a logically structured database. The second one is called `auditory-classifier` responsible for building and testing models for classifying the presence of ambulances in an audio scene. As a co-product, it is proposed for this project to build an extensible system using services. Services can contain a single or a set of software functionalities, and for this project the architecture will be driven loosely by the dependency injection and inversion of control principles. This increases system's modularity and expansibility. For each purpose, services within the classifier project can serve different clients as needed, so they can reuse it for different purposes or even implement new ones, together with the policies that should control their usage. The third and last sub-project is called `source_localization` and is a set of scripts rather than entities. This scripts are responsible for reading the data

Figure 25: Block diagrams of the project stages

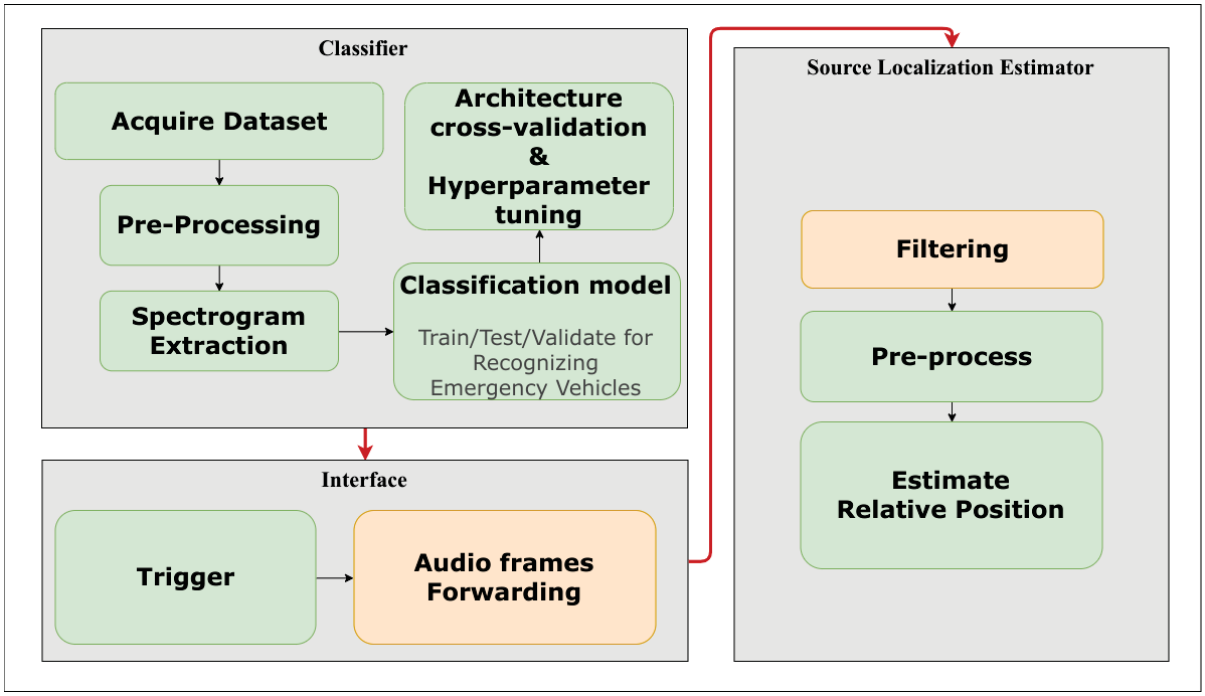


Figure 26: Sub-projects contained inside the Masters parent project and their required tasks.

AudioSet Download Tool	Auditory Classifier	Source Localization
Responsible for managing the AudioSet provided files, executing the data download, post-processing and normalizing the audios, build a directory structure, and save the files accordingly to the structure built.	Responsible for reading audio input, decode and deserialize data, normalize data, build a representation and/or extract features from data, build a neural network architecture, train the neural network, generate and save a model, generate indicators for model evaluation, apply predictions to unseen data, and export the predictions on a structured file.	Responsible for the Import of structured classification data, load audio files, filter audios, plot audio data, interface MATLAB and python, load ADMA data, massage ADMA data, generate ADMA plots, apply PHAT estimators for sound source localization, plots source localization estimation data, evaluate estimations, export results to a storage file.

written by the previous sub-projects and then perform the data source localization, providing a performance measurement against ground truth data and finally producing a `JSON` file providing a YAML object listing the classification probabilities c^s and the respective azimuths α along the time axis. The detailed tasks performed by the tools can be seen in figure 26.

No totally automated integration has been implemented between these sub-projects. They should be operated manually via its user interface, and once the user is given an output, he can manually use it as the input for the next package. Aside from this project being a prototype it is also focused for research purposes, therefore it gives freedom to the user to use different inputs at later stages other than those output by the previous stage. This further complies with the modularity principle which has been previously mentioned — a user can choose whether to only use one these sub-projects isolatedly or combine them for an end-to-end result.

3.1 Database

Two different databases are going to be developed — the first containing large amounts of data, obtained through web scrapping techniques. This is necessary as DNN are data hungry and at least one of the distributions available have to provide generic and labeled data. For collecting such a high quantity of data automation definitely plays an important role. Neural networks are robust to data outliers, however if there is a systematic problem is present between labels and its content, the algorithm performance will certainly be significantly lower. The second database comprises a set of technical tests to be performed on the outdoors proving ground of CARISSMA. This will ensure that both real and experimental data are collected, allowing for good model generalization capacity while having further unseen data by the model for validation of results at the end of the training phase. It means we can compare the model against real cases and validate them against this applied situation.

3.1.1 Google AudioSet dataset

The first package developed was an automated HTTP request robot. But before firing requests a problem had to be solved, which was a source of trustworthy labeled, structured and accessible audio fragments. There are great image open-source databases out there, such as CIFAR-10 and ImageNet. For audio, however options were more scarce and good options found were the `freesound.org` database and DCASE2017 database. DCASE2017 was proposed for the DCASE challange and therefore audio clips belonged to some restricted domain aimed for each task given by the challange rather than trying to embrace all possible sounds. `freesound.org` seemed like a great choice, having a crowd-sourced annotation system, a great ontology collection, but failing at only tagging whole videos. That means the clips would have as long as 20 minutes of playback time and if for one of those minutes there was a vehicle sound present, the other 19 minutes would misleadingly carry this label to the other audio fragments as well. Since the amount of data is beyond of careful human manual verification, it would not be possible to manually split the audio into fragments and re-annotate them. Another option considered was downloading video databases and then extract the audio from them. Two good options found were the Youtube-8M database, which consists of millions of youtube videos IDs and automated labels, and the Google AudioSet, which is composed of youtube video IDs and automated labels. Both of them were actually easier to access than static binaries, since downloading the entire database is unrealistic to this project and we do not need every single data point available. So being able to download just the audios relying on its video IDs would be a really desirable characteristic. The first had an amazing quantity

of data, but the latter had a huge advantage for us: It had an ontology instead of just entities scattered all around. This lead Google AudioSet to be chosen (GEMMEKE et al., 2017).

This sub-project was entirely written using Python 3.7, which was also used to interface the multimedia framework FFmpeg. This framework is a default tool for encode and decoding both video and audio formats. Incoming audios when downloaded directly from the internet are subject to be in almost any encoding format, the library has been used for transparently convert any incoming format into a single standardized one, not to mention other important data standardization chores.

Google AudioSet provides three types of data per class — balanced, unbalanced and evaluation data. Balanced segments are those which were used to compose 'maximally-balanced' subset, which intent to provide at least 50 segments for as many classes as possible. unbalanced segments are those which are not contemplated in the previous set (and due to the fact segments can receive multiple labels, the overall labels count is not uniform). Lastly, the evaluation files are those which were manually conferred in order to provide a performance measure for each of the labels, so in classes with high health levels, these data can be completely trusted (GEMMEKE et al., 2017).

The arguments and options exposed by the Command Line Interface (CLI) using the `audioset-download-tool python main.py -help` flag. Another way of quick start using the tool is reading its `README.md` file.

The architecture is constituted of only two main classes: the `AudioSetDownloader` and the `AudioProcessor`. The `AudioSetDownloader` is responsible for managing the input `CSVs` according to the arguments and options passed via CLI, assembling the dataframes containing the desired data, logging information, warning, and errors related to the support files management and HTTP response statuses, and finally executing the audios download consuming the dataframe which was built. The `AudioProcessor`, in turn, is responsible for fetching the audio files, recognize malformed data, trim the audios, and save the audio using the desired sampling rate, encoding and number of channels.

The dataset was prepared to accommodate two classes, negative and positive, configuring a binary classification. Auditory scenarios where ambulances are not present correspond to the negative, and scenarios where ambulances could be detected in the surrounding environment are the positive cases. The goal was to build the negative class with sounds which were sufficiently generic to provide general features that represent a traffic scene on the absence of emergency vehicles, whether on the presence of other non-emergency vehicles or even on usual urban or countryside landscapes.

Using the AudioSet ontology the following structure was assembled (for further details, refer to Gemmeke et al. (2017)):

i Positive class:

   (a) Sounds of things > Vehicle > Motor Vehicle (road) > Emergency vehicle

ii Negative class:

   (a) Sounds of things → Vehicle → Motor vehicle (road) → Car

   (b) Sounds of things → Vehicle → Motor vehicle (road) → Truck

   (c) Sounds of things → Vehicle → Motor vehicle (road) → Motorcycle

   (d) Sounds of things → Vehicle → Motor vehicle (road) → Traffic noise, roadway noise

   (e) Channel environment and background → Acoustic environment → Outside, urban or manmade

Note that classes such as "Sound of things → Alarm → Ambulance (siren)" are inherently included in the positive class, since the AudioSet ontology allows for multi-labeling, thus it can be said that every "Ambulance (siren)" is included in "Emergency Vehicle", making it a subset of the latter class.

To be on the safe side, the `AudioProcessor` is responsible for filtering the audios according to desired properties found on them. The following audio properties were selected:

1. Sampling Rate: 48 kHz
2. Bit Depth: 16 bits
3. Number of Channels: 2 (Stereo)

This provides high audio quality to the database, and future-proofs the database in case mono audios are not good enough for other applications, even though for this current application it would be of no harm.

After that, the author manually curated the classes. There were some overlaps as, for example, the class "cars" could contain siren sounds in it. This guaranteed (not taking in factor human fail-prone nature) that all audios were homogeneous property-wise and that the positive and negative classes did not leak onto each other. Minor mistakes like mislabeling do not present a major issue for CNN algorithms, but rather systematic ones could be an actual

one. Manually listening and curating them guarantees that no systematic mistake has taken place.

Finally, after downloading, filtering and curating the audio files, the negative class was composed of 3,692 files, totaling 6.9 GB GB on disk. The positive class was composed of 8.150 files, totaling 2,7 GB GB. Since the negative class would not greatly benefit from data augmentation and it would further unbalance classes, the technique was applied solely on the positive class. The modifiers applied were doubling and halving speed, and stepping 4 semitones up and 4 semitones down. It resulted in the positive class being composed now of 10,588 files totaling 7.3 GB. This slightly uneven proportion did not show negative effects on training results after iterative training sections with different configurations. The grand totals in the database resulted in 14,290 files, totaling 14,3 GB, finally amounting for 22 hours, 44 minutes and 41 seconds of audio duration.

Providing annotations between sub-types of siren in the database would be o great benefit for getting a good insight on the data. On the other hand, due to the size of the database it would not the practical to do so. Thus, a natural assumption is that the database would be naturally biased towards American emergency vehicle sirens signatures in detriment of any other, but it should not greatly hurt performance for other countries' signatures. Figures 27 and 28 show how noticeable is the difference in the signature seen between a auditory scenario in the presence of a vehicle with without any sirens and an generic ambulance with the sirens on. Also figure 29 shows the most common siren signatures found in the database, associating them with their countries via either the video title language or the name of the place mentioned in the title.

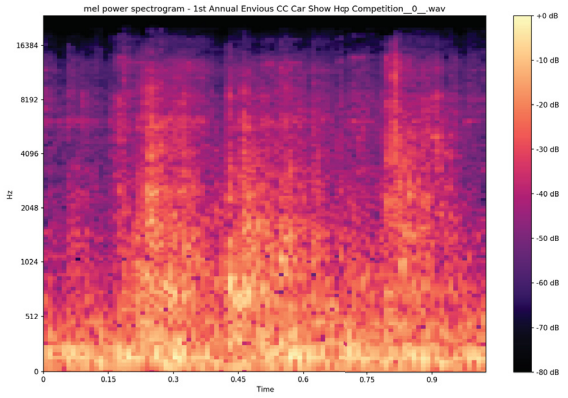


Figure 27: Negative class.

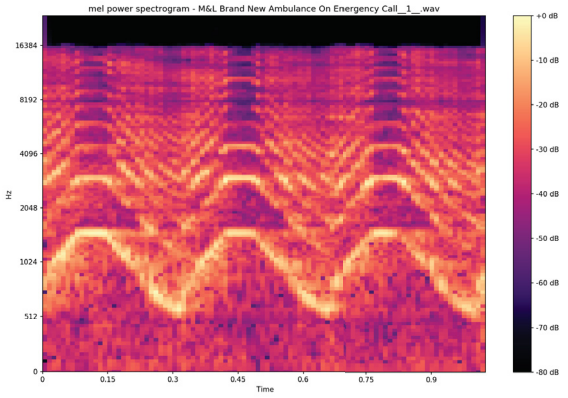


Figure 28: Positive class

As we as humans do, the model is expected be able to recognize for patterns using them as features, and thus characteristics that make a siren sound familiar for humans, in order to describe an example as an emergency vehicle.

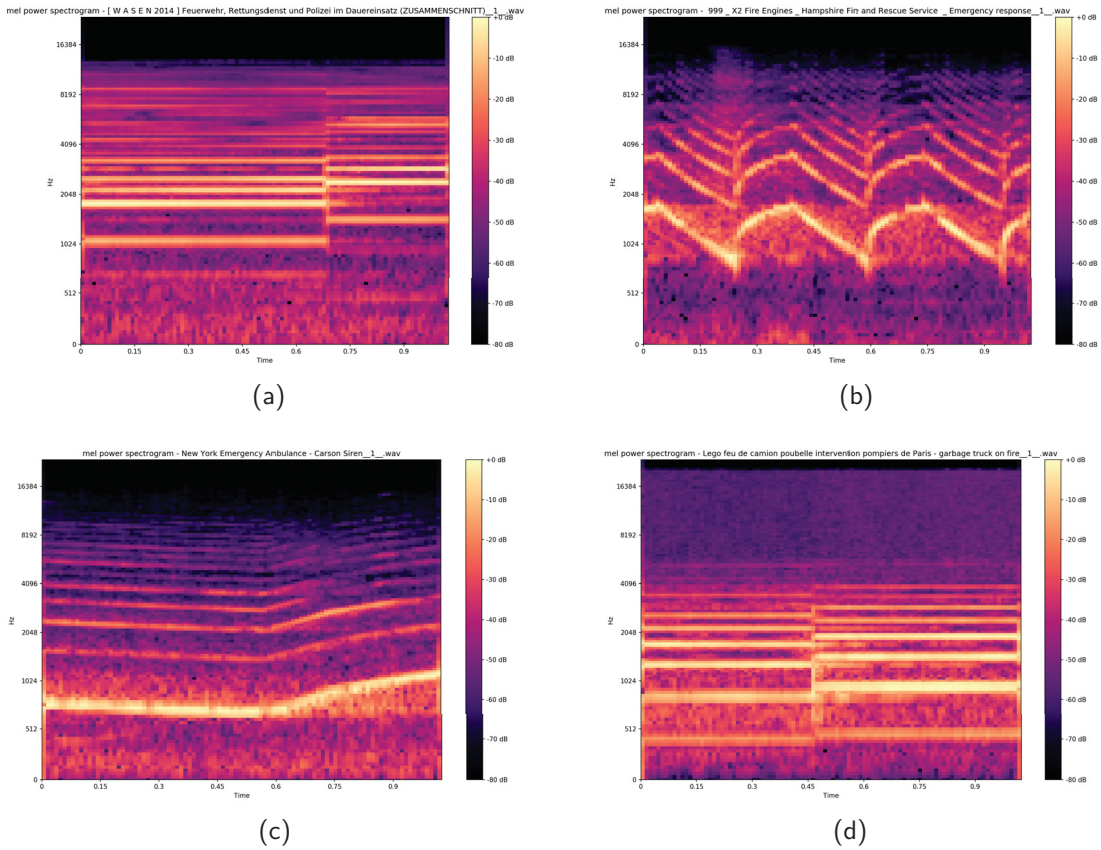


Figure 29: Mel-spectrograms noramlized to 0dB. (a) Germany (b) United Kingdom (c) USA (d) France.

3.1.2 CARISSMA dataset

CARISSMA is an institute related to the Technische Hochschule Ingolstadt responsible for performing research in the field of Automotive Safety. The assembly of dataset no. 2 was made for us to have controlled audio clip conditions. The idea was to generate the audio clips using known recording and test parameters. The following tests were realized inside its premisses, more exactly on its denominated outdoor proving ground (seen in figure 30). Track measurements can be found in figure 31. Equipment employed in tests were the following:

- Set of four omni-directional condenser microphones from the AMMOON brand
  - Frequency Response Range: 50 Hz – 16kHz
  - Sensitivity: -30dB $\pm$ 3dB
  - Interface: USB 2.0
- Siren speaker from the ZKXX brand
  - Power: 100W

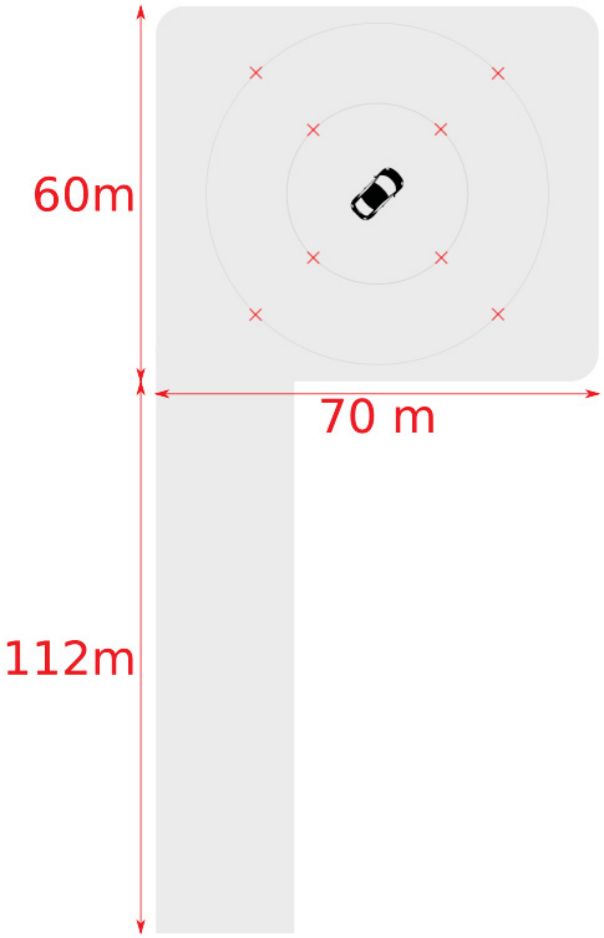
Figure 30: CARISSMA proving grounds facilities

  - Noise Level: 120 – 150 dB

- Apple MacBook notebook (MacOS v10.14.3) using GarageBand v10.3.2 as the digital audio workstation
- Audi Q7 Second Generation vehicle
- Smart Fortwo Electric Drive Fourth Generation vehicle
- GeneSys Automative Dynamic Motion Analyzer (ADMA) hardware V3.0
- Bavaria's police vehicle model BMW F31 equipped with original siren speakers used by the police force.

First test For the first test, a 4-microphone array mount was built to be placed on the roof of the Smart where distances between each microphone were measured as 101 cm, forming a square using the frame where the vertices are the points where the microphones were placed, as can be seen in figure 32. These were wired to the computer via usb interfaces for recording the data (figure 33), then the four distinct microphones were recorded each one to a respective independent channel, taking care of adjusting parameters the same for all of them. This resulted in having one four-channel recording for each run. It is worth noting that MacOS' operating system tries to keep all microphone clocks synchronized while recording, but they might drift during recordings as their crystal oscillators are never built exactly the same. As

Figure 31: CARISSMA proving ground track official measurements



a result, their frequency might differ slightly, which turn out noticeable after long periods of recording.

The polar characteristic of the microphones are omni-directional, therefore they will all be positioned with their tip pointing straight up, disregarding any specific horizontal direction. The Smart played the role of ego in this test context. The target consisted of the Q7 equipped with a multi-tonal Siren mounted on its roof utilizing a frame for fixation. It was the dynamic component of the test which was prone to move around.

The siren speaker used for the first test was the ZKXX siren. The mount directives were the following:

1. Positioning
   (a) Laterally: The siren must be placed as centered as possible in the roof of the car.
   (b) Longitudinally: The siren must be placed as offset to the front of the car as possible, respecting the limits of the frame mounted on top of its roof.

Figure 32: Smart Electronic Drive set up with the microphone array mount.

2. Roll: As centralized as possible

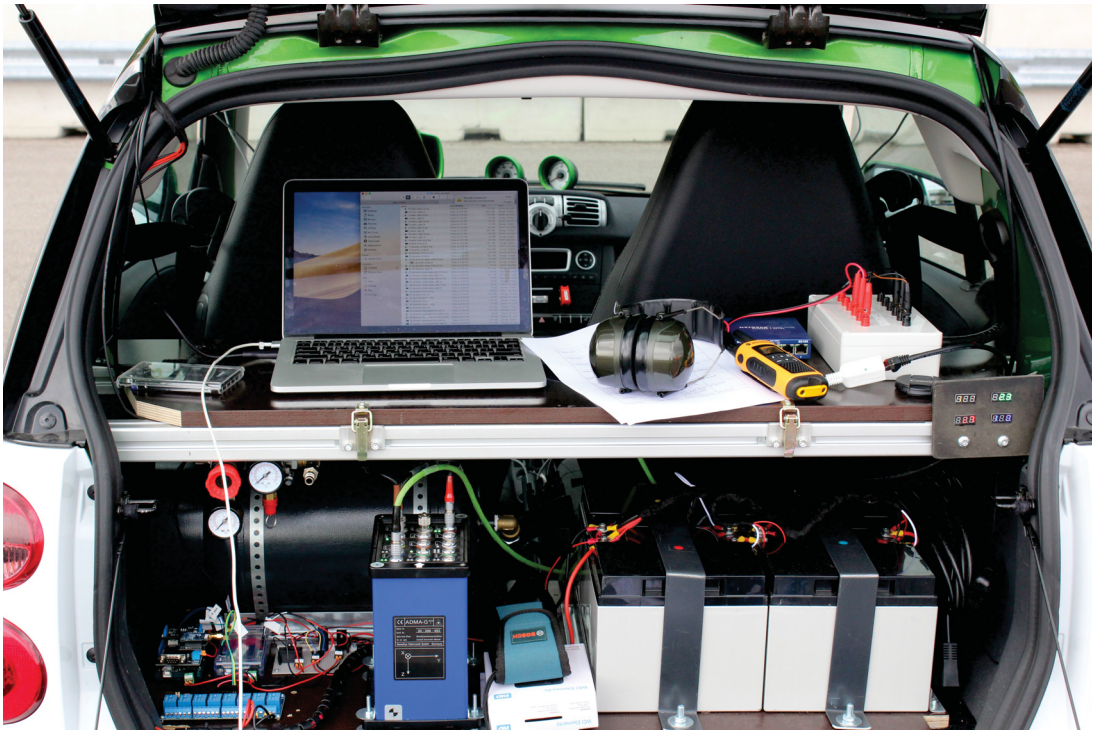
3. Pitch: Pointing straight to the horizon, as parallel as possible to the floor

4. Yaw: Pointing outwards to the center of the car, in the forward direction of the car

The test was carried out on 19th March of 2019. Before beginning the actual tests, the points of interest needed to be mapped to be offset in the ADMA's data recording software. Measurements were made in relation to a reference point and assured that both the microphone array and the siren were the points of interest being measured. The microphones and the siren were positioned 176.9 cm and 194.3 cm from the floor, respectively. Car reference position, point of interest (microphone array and siren) position, car speed, car acceleration, car turning rate, and other parameters were recorded and matched with timestamps. The sampling frequency used for ADMA measurement was 100Hz. Base noise levels averaged 74 dB.

In order to synchronize the audio recording and the ADMA data a small project containing a toolset for generating timestamps using the same 100 Hz of sampling frequency was built. The script would detect the click event from the mouse used to start the recording, and begin recording timestamps into a `CSV` file using the UNIX Epoch time protocol. That

Figure 33: Recording setup on board of the Smart Electronic Drive.



way, it was possible to know which recording sample was related to what ADMA position data. The recordings were made using a sampling rate of 44.1kHz, a bit-depth of 16 bits, and 4 channels (one for each microphone present in the array).

The test script consisted of 30 runs. Table 3 contains each routine description. Tests 1 to 16 were made in positions marked by the red crosses in figure 31. The black vehicle in the picture represents ego while the target was changing positions according to the crosses.

CARISSMA team has greatly supported the measurements, providing the vehicles, equipment, and the ADMA data recordings. Figure 34 shows the setup for test no. 1 and the measurement wheel which was used along with cones to mark the proper positions.

The result data was composed `CSV` files containing the UNIX Epoch timestamp, raw audio files for each of the four microphones, and binary MATLAB data containers containing ADMA data. All data was organized using a folder structure for placing each test's data. Helper shell scripts were built for automating bulk transport and renaming of the files which would be employed also for the second test.

Id	Routine Name	Activity Description	Recording Duration
1	Static Placement - First Siren Tone	Static Target directly in FRONT of the center point of the array - 10 m	60 seconds
2	Static Placement - First Siren Tone	Static Target directly in RIGHT of the center point of the array - 10 m	60 seconds
3	Static Placement - First Siren Tone	Static Target directly in LEFT of the center point of the array - 10 m	60 seconds
4	Static Placement - First Siren Tone	Static Target directly in BACK of the center point of the array - 10 m	60 seconds
5	Static Placement - First Siren Tone	Static Target directly in FRONT of the center point of the array - 35 m	60 seconds
6	Static Placement - First Siren Tone	Static Target directly in RIGHT of the center point of the array - 35 m	60 seconds
7	Static Placement - First Siren Tone	Static Target directly in LEFT of the center point of the array - 35 m	60 seconds
8	Static Placement - First Siren Tone	Static Target directly in BACK of the center point of the array - 35 m	60 seconds
9	Static Placement - Second Siren Tone	Static Target directly in FRONT of the center point of the array - 10 m	60 seconds
10	Static Placement - Second Siren Tone	Static Target directly in RIGHT of the center point of the array - 10 m	60 seconds
11	Static Placement - Second Siren Tone	Static Target directly in LEFT of the center point of the array - 10 m	60 seconds
12	Static Placement - Second Siren Tone	Static Target directly in BACK of the center point of the array - 10 m	60 seconds
13	Static Placement - Second Siren Tone	Static Target directly in FRONT of the center point of the array - 35 m	60 seconds
14	Static Placement - Second Siren Tone	Static Target directly in RIGHT of the center point of the array - 35 m	60 seconds
15	Static Placement - Second Siren Tone	Static Target directly in LEFT of the center point of the array - 35 m	60 seconds
16	Static Placement - Second Siren Tone	Static Target directly in BACK of the center point of the array - 35 m	60 seconds
17	Circular movement around Ego - First Siren Tone	Target performs two full circles keeping 35 m from Ego - 20 km/h	90 seconds
18	Circular movement around Ego - Second Siren Tone	Target performs two full circles keeping 35 m from Ego - 20 km/h	90 seconds
19	Moving Target at a single face - First Siren Tone	Target moves away from the FRONT of Ego and then returns - 30 km/h	110 seconds
27	Moving Target passing by Ego - First Siren Tone	Target incomes from the FRONT of Ego, offset to the RIGHT by 3.5 m, passing by and finishing to the BACK of Ego - 30 km/h	25 seconds
28	Moving Target passing by Ego - Second Siren Tone	Target incomes from the FRONT of Ego, offset to the RIGHT by 3.5 m, passing by and finishing to the BACK of Ego - 30 km/hh	25 seconds

Table 3: Description of test routines used in the first test — Part I

Id	Routine Name	Activity Description	Recording Duration
29	Moving Target passing by Ego - First Siren Tone	Target incomes from the FRONT of Ego, offset to the RIGHT by 3.5 m, passing by and finishing to the BACK of Ego - 60 km/h	20 seconds
30	Moving Target passing by Ego - Second Siren Tone	Target incomes from the FRONT of Ego, offset to the RIGHT by 3.5 m, passing by and finishing to the BACK of Ego - 60 km/h	20 seconds
33	Moving Target passing by Ego - First Siren Tone	Target incomes from the LEFT of Ego, offset to the FRONT by 3.5 m, passing by and ending the course in the RIGHT of Ego - 60 km/h	20 seconds
34	Moving Target passing by Ego - Second Siren Tone	Target incomes from the LEFT of Ego, offset to the FRONT by 3.5 m, passing by and ending the course in the RIGHT of Ego - 60 km/h	20 seconds
35	Moving Target Simulate Case - First Siren Tone	Target simulates joining an Autobahn by the RIGHT, overtaking Ego - 60 km/h	25 seconds
36	Moving Target Simulate Case - Second Siren Tone	Target simulates joining an Autobahn by the RIGHT, overtaking Ego - 60 km/h	25 seconds

Table 4: Description of test routines used in the first test — Part II.

Figure 34: Example of test setup.

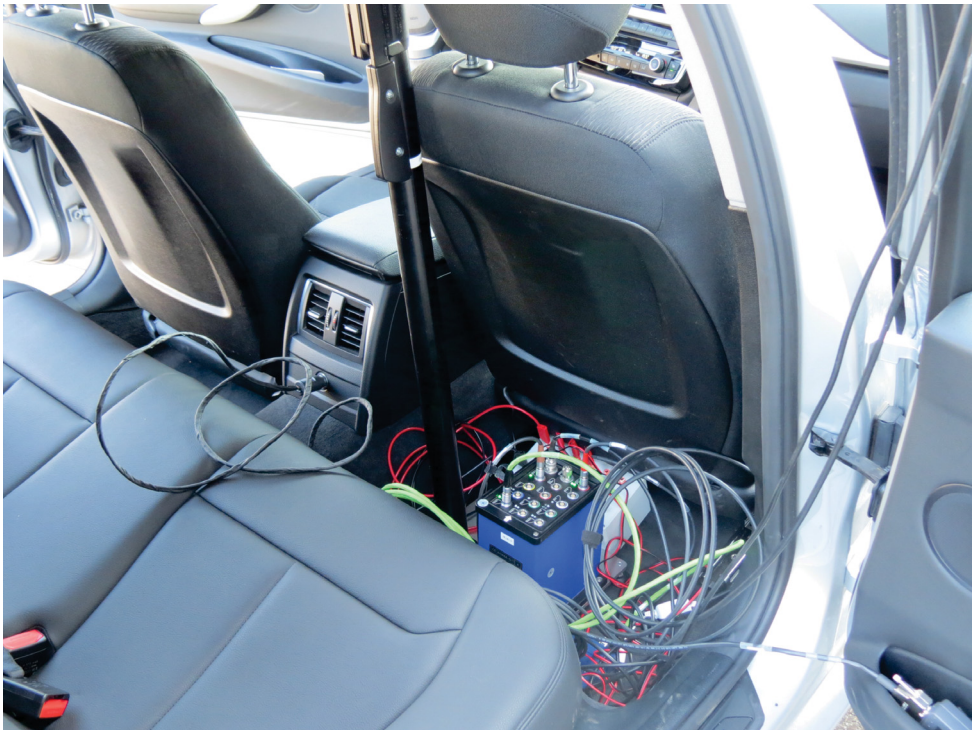
Second test The same 4-microphone array mount and setup was reused on the roof of the Smart for the second test. A big improvement was reworking the timestamp grabber script, which was relying on a single thread to capture the timestamps. That way, besides reworking the storing routine, now the script could run freely while a separate thread was responsible for getting the precise timestamp after the clock period has expired. This eliminated the variation which timestamps were experiencing during the first test. This was caused by the fact since only one thread was responsible for the whole routine, if the CPU would stutter even though slightly during the whole recording period, the measurement would be delayed. Therefore, instead of having timestamps vector being composed of `[0ms 10ms 22ms 30ms 43ms...]` — as it was the case of the first test —, now it would rather be composed of `[0ms 10ms 20ms 30ms 40ms...]`.

The test was carried out on 7th June of 2019. Like for the first test, before beginning the actual tests, the points of interest needed to be mapped to be offset in the ADMA's data recording software. However, now instead of the Q7 holding the siren, the Bavarian police vehicle model F31 would act as the target. Consequently, the ADMA equipment had to be installed, and the same measurements for offsetting the actual position of the siren sound source had to be done (35). The siren sound signature follow the German norm DIN 14610, which determines a square wave with a low tone and a high tone — the low tone fundamental must lie between 360 Hz and 630 Hz, and the high tone must be a multiple of 0.33 (tolerance from 0.293 to 0.426) higher than low tone. The duty cycle must be of 50%, with the total cycle period between 2.5 and 3.5 seconds. Differently from the first test, now the siren speaker is located in the front-end part of the vehicle at a low position, about 23cm from the ground. All ADMA configuration is the same as the first test, including the sampling rate at 100Hz. Base noise levels averaged 64 dB.

The test script now consisted of 21 runs, since the main cases have been identified now. Tests should be able to depict typical situations which could challenge the classification model not only statically having the siren on, but also toggling between siren on and off, relatively high speed to cause bigger Doppler effects, and trying sound occlusion cases. Table 5 contains each routine's description. As in the first test set, tests 1 and 3 to 9 compose the static tests described by the red cross marks in figure 31.

With all data in hands, and properly structured via the script made for the first test, the audios were trimmed in order to match exactly what is defined by the tests script. The last step was writing a script capable of plotting the data from the ADMA in a Cartesian plane, matching the timestamps from both sources — the ADMA data and the timestamps from the

Figure 35: ADMA module installed in the police vehicle.



`timestamp_grabber` script ran on the audio recording environment, calculating the distance and azimuth between Ego and Target per sample, and finally assembling a master object containing timestamp, instant position of each of the Ego microphones, instant position of the Target's siren speaker, calculated relative azimuth, and calculated relative euclidean distance. This master object is fundamental, so we may have a ground truth source to compare the estimated parameters against this data which is known to be correct.

Is worth noting that synchronizing the data was not possible for every case. The script by the author used the Unix Epoch Time timestamp which begins at 0h of 1/1/1970 and used the millisecond resolution format, generating a integer number which amounts for the total milliseconds that have passed from the clock start. This means the first timestamp recorded by the script — *1559893668905* converts to 7th of June, 09:47:48. The exact time when the first test was run. However the ADMA system, according to the Firmware version 30.5.X.X technical documentation states that its GPS time was set to match UTC in 0h 6th of January, 1980. It says also that the GPS is now ahead of UTC by 18 seconds. Therefore, a 18 seconds compensation was manually added along with other instructions for converting the ADMA data timestamp to the Unix Epoch timestamp format. However, for some tests, even after applying all information given by the technical document, timestamps would mismatch as long as up to 5 minutes. Other tests would align closely enough. Thus, the following list shows which tests had to be forcefully matched:

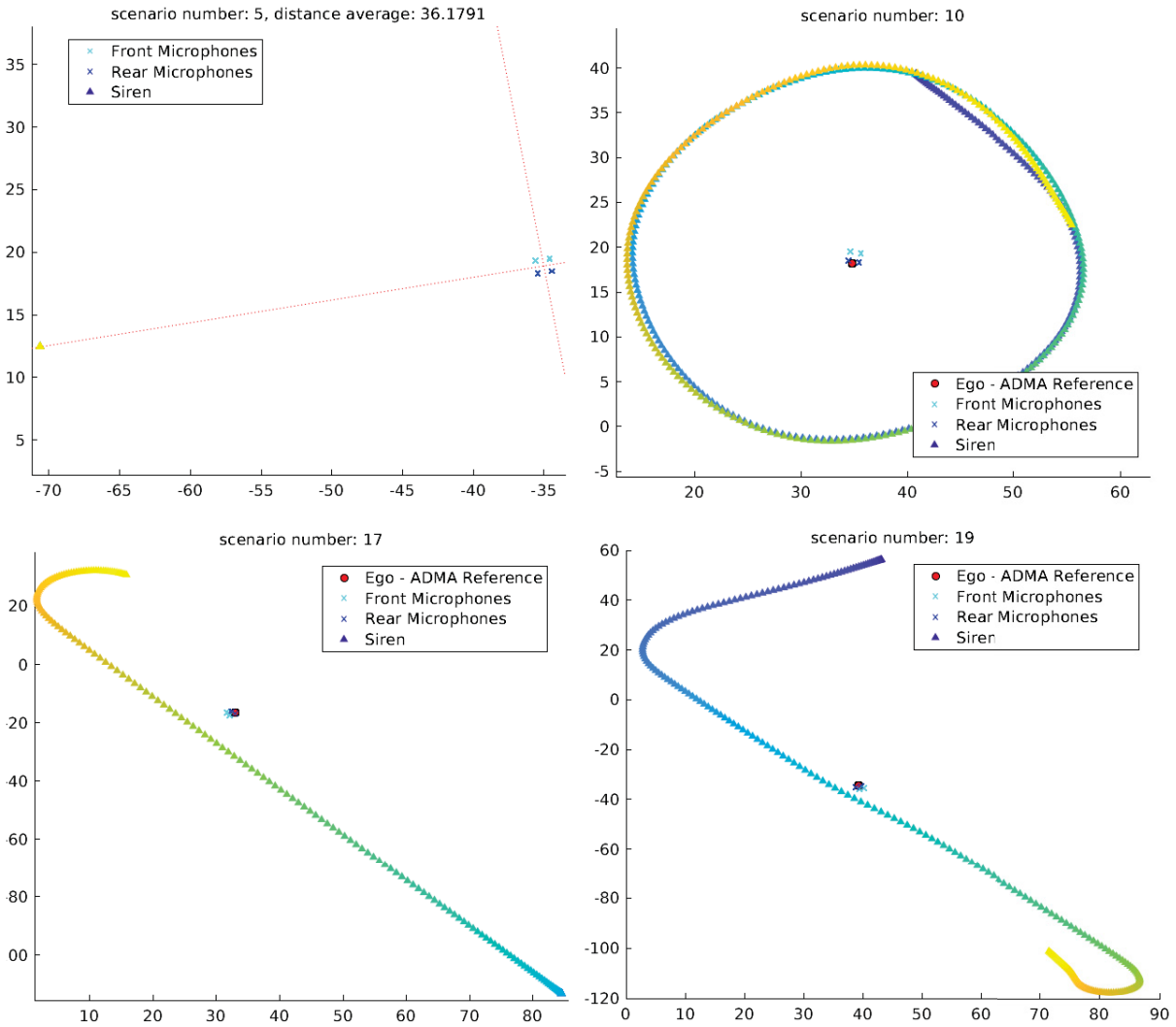
- Dynamic tests which were correctly matched: 6, 7, 11, 14, 15, 16, 18, 19, 20, 21
  (a) 6: 4 seconds difference
  (b) 7: 4 seconds difference
  (c) 11: 2.5 seconds difference
  (d) 14: 2 seconds difference
  (e) 15: 2 seconds difference
  (f) 16: 2 seconds difference
  (g) 18: 3 seconds difference
  (h) 19: 2 seconds difference
  (i) 20: 2 seconds difference
  (j) 21: 3 seconds difference
- Dynamic tests which were forcefully matched: 1, 2, 3, 4, 5, 8, 9, 10, 17

Forcing matches was done by ignoring the timestamps altogether, and rather count timestamp positions. This methodology was employed every time the first timestamp of one data source was larger than the last timestamp of the other source, bidirectionally. That is to say, the first row of data from Ego was matched to the first row of data from the Target (ADMA data) and so on. This maneuver was a last resort resource that does not guarantee that data is correctly synchronized by any means. Figure 36 shows a variety of plots examples which were made using the post-processed ADMA data. Color scale goes from darker to lighter, representing earlier to later timestamps, respectively.

3.2 Audio Source Classifier

A major difference in this sub-project is the presence of a `setup.py` file. It means that this whole project is distributable. If we take a closer look at `audio_classifier`, we can see it is a special folder — it contains a structure with all the source code inside of it. This folder represents a package, which contains other sub-packages, similar to what can be seen in Java Enterprise projects. In `setup.py`, a tool called `setuptools` includes all relevant packages of the source tree in a python egg or a python wheel, which are analogous to Java's `JAR` files. It means this code can be packed into a single file and used as a dependency for other projects as well, protected by GPL license.

Figure 36: Example plots from the ADMA data in the second test.



All functions can be accessed using the which can be accessed via CLI context menu, called via the help flag using the `audio_classifier` command `python -m audio_classifier --help`, and its modes — `fit_model`, `evaluate`, `predict`, `utilities`.

3.2.1 Pre-processing

This sub-package was responsible for all functionalities related to configuration, input preparation, application bootstrap, and finally feature or representation extraction. `Librosa` (MCFEE et al., 2019) was extensively used, which is a package focused on music and audio analysis. It provided the building blocks necessary to create the whole audio information retrieval and processing necessary for greatly speeding up the progress of this project.

Depending on the mode selected, whether model fitting, evaluation or prediction, this sub-package would have to load the proper data. For example, in cases only evaluation was

selected by the user, a model would already be available from previous mode fitting sessions. Therefore, it would not be needed to go through the phase of deserialization of the training data, but rather only load and assemble the evaluation data object. This context could be used for training sessions, evaluation and and pure prediction as well, when no data would have to be loaded at all apart from the new data. Also related to input loading, a cache function was implemented. Since more than 8 hours of audios would have to be loaded per training section, this deserialization process could take up to 4 hours to complete. Therefore, a good solution was to implement a cache system utilizing `NumPy` matrix objects along with a package called `pickle`, in order to save this objects to binary data. This made prototyping much more flexible, reducing the time spend by the CPU reading the audio files from up to 8 hours to no more than 30 seconds. If input data would change, then the cache could be cleaned up and the new data would create new cache files. Also, note that all filepaths are configurable via the `InputPreparation` constructor, so referring to specific input folders are not important, as they can be changed at will.

The first step was to remove any nonconforming audio stream, providing a consistent data structure to be injected in the NN architecture. In order to train a network, it is pivotal that all input data are of the same dimensions, so they can be passed forward through the NN layers. Thus, the pre-processor would automatically identify the standard bitstream size using the matrix dimension most present in the ensemble and discard any data which would not fit this dimensions.

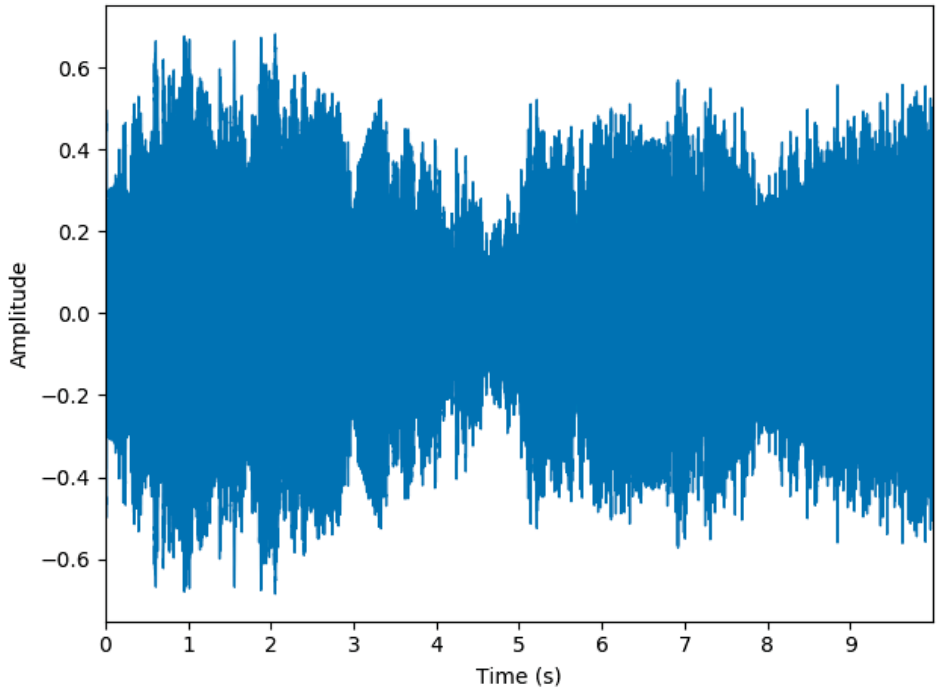
The result would then be cached for future uses and thereafter would pass through additional functions, depending on the options passed by the user. The first function is silence trimming. The silence trimming took zones of relative low power leading or trailing the data and cut it out, refilling this blank vector positions with copies of what was left intact on the matrix. That meant if silence was something undesirable for a given class, this would prevent the model to systematically learn something incoherent with what that given class fundamentally meant. This function can be customized to be applied to none, one, or more classes present in the input data. The second function was a data augmentation service — this means increasing the number of data points. Since at some point the data available through the database formed could not be enough for the whole learning process. Two supported kinds of data augmentation were developed: augmentation via time compression or widening, in which cases the same approach as in the silence trimming would be applied, and augmentation via pitch shifting, where different semitones could be passed as arguments. This provides a greater generalization power to the classifier, so if an sound signature would have a faster, slower,

higher pitched or lower pitched tone, it could still understand it as a recognized source and correctly classify it McFee, Humphrey e Bello (2015).

The last step was to extract some kind of knowledge representation from the raw audio data. The audio data at this point is a MxN matrix, where M is the number of channels and N the number of samples for that audio file. Yet, since instead of having a single audio file deserialized and vectorized, we have thousands of them, so they get stacked on top of each other forming now a MxNxO matrix where O is the number of audio files loaded. However, injecting this object directly to the neural network would greatly slow down its learning process, even hindering it from making discoveries and potentially generating new features which could enable better understanding over the goal concept. Two strategies could be adopted, implementing a representation descriptor, which could ease this processing, or implementing actual feature extractors. This is where a interesting characteristic about this package comes into play: extendability. The class `PreProcessor` is responsible for transforming and giving meaning to the raw audio input data. It can be easily extended to support new features, since a base module was designed to expose new feature extraction methods. As for the state of this project, only the mel-frequency spectrum representation is available.

A spectrogram is a visual representation of the spectrum of signal frequencies as it varies with time. In the time domain, audio signals are represented by amplitude versus time, a simple two-dimensional representation to provide us a good understanding about such a complex and rich concept which is a traffic auditory scene, thus it gives us little immediate information about the signal itself (figure 37). From this signal it is possible to get its Fourier Transform, which is a function that gets a signal in the time domain as input, and outputs its decomposition into frequencies, now describing it in the frequency domain rather than the time domain. What is actually done computationally is the use of a DFT, which gives us a really good approximation comparing to what the continuous transformation would give us. This can be achieved by the use of a common algorithm called FFT, but we do not just want to apply it to the entire signal and get its output. Rather, it we want to extract a spectrogram from this signal.

Figure 37: Example ambulance audio time domain wave envelope plot sampled at 48kHz. Amplitude is in the digital +1 to -1 limits scale.



To extract a spectrogram from an audio signal, the following steps are required (CHAKROBORTY, 2008):

1. Apply Windowing: As used as base for Finite Impulse Response (FIR) design filters, a windowing function is applied to the signal. For audio applications, commonly we use Hanning windows as the function is designed to smooth out spectral leakages that arise from the fact that real world data is not periodic as the Fourier Transform assumes.

$$\boldsymbol{s}_o = \boldsymbol{s}_i \circ \boldsymbol{w} \tag{3.2}$$

   where $\boldsymbol{s}_i$ is the audio signal matrix and $\boldsymbol{w}$ is the window function matrix applied element-wise to the audio.

2. Zero-pad: To compute the FFT, matrix $\boldsymbol{s}_o$ needs to be compatible with a factor matrix in order to obtain all DFT coefficients.

3. Apply DFT: A DFT is a discrete function which outputs coefficients. These coefficients are the discrete analog to those present in a N-periodic and N-normalized Fourier series.

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i2\pi kn/N}, \quad n \in \mathbb{Z} \tag{3.3}$$

Therefore, the vectorized implementation is as follows:

$$\boldsymbol{X} = \boldsymbol{s}_z p \boldsymbol{W} \tag{3.4}$$

where $\boldsymbol{s}_z p$ is the zero-padded, windowed, and framed input audio signal. W is the Twiddle factor used to calculate the DFT. Its elements are defined by:

$$\left(W_N\right)^k = e^{j2\pi\frac{k}{N}} = \left(\omega_N\right)^k \tag{3.5}$$

so the result is a N points discrete Fourier transforms (DFT) operation, which outputs N frequency bins. This means at this point, we have a FIR filter bank with N banks. This means we can consider DFT to be equivalent to filter bank.

4. Compute the power spectral density: The power spectrum is simply obtained by the obtaining the element-wise power of X:

$$\boldsymbol{D} = \boldsymbol{X} \circ \boldsymbol{X}^* \tag{3.6}$$

On a practical perspective to extract the spectrogram from signals, we first slice the audio into small audio frames. The idea behind this step is that frequencies in a signal change over time, so in most cases it does not make sense to do the Fourier transform across the entire signal in that we would lose the frequency contours of the signal over time. Therefore, choosing a proper frame length is important, so it is long enough to capture the phenomenon we are interested in, but also short enough that we can make this approximation that the frequencies over this period are stationary. Thus, the DFT over this short-time frame would retrieve us a good enough approximation of the frequency envelope by concatenating adjacent frames. Calling these frames reminds us from frames used in videos, and it is not by coincidence — the concept is fundamentally the same, which is finding the smallest possible block of information that provides a comprehensible insight of complete content. May it be a video or an audio. With our frames in place, we multiply our pieces of signal against a windowing function, calculating the DFT for each of them. This method is called the STFT, windowing an appropriate number of samples according to the total amount of time to be framed and the signal sample rate, and applying local DFT to the frame, which is composed of N samples. Figure 38 shows the Fast Fourier Transform (FFT) algorithm applied to a single window of the example signal, while figure 39 shows all the windows extracted using N=8196 for a physical duration of 112 milliseconds for the signal's 48kHz sampling rate. Therefore, 111 windows are showed since there is a $1/4$ window overlap per window frequency hop. The value of N can be

experimentally adjusted given specific applications, since not only it affects the window length, but also the number of times the DFT is applied, causing higher computational demand.

Figure 38: Example ambulance audio for a single DFT window. Magnitude of frequency bins are composed of complex-valued dimensionless coefficients.

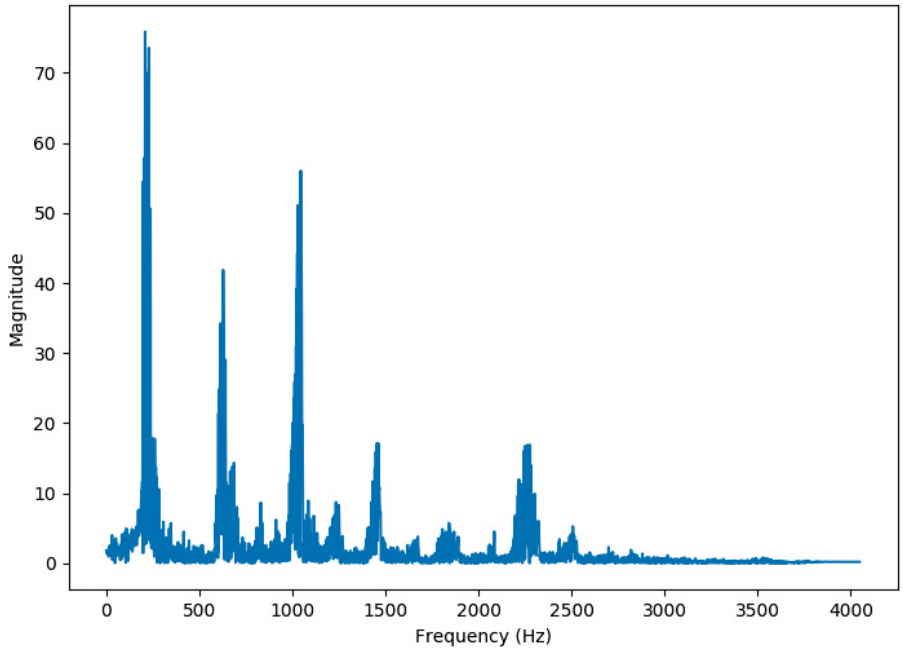
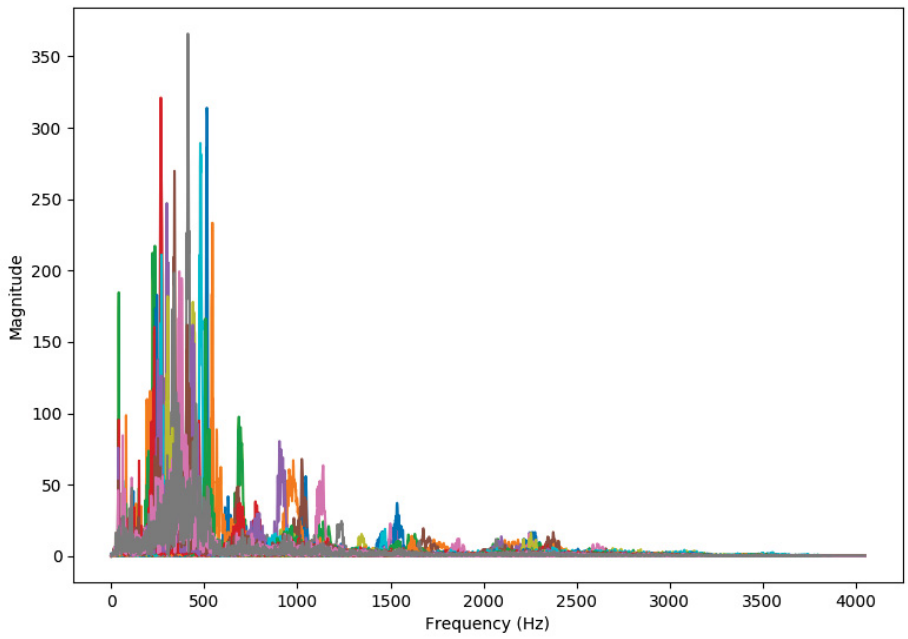


Figure 39: Example ambulance audio for multiple DFT windows. Magnitude of frequency bins are composed of complex-valued dimensionless coefficients.

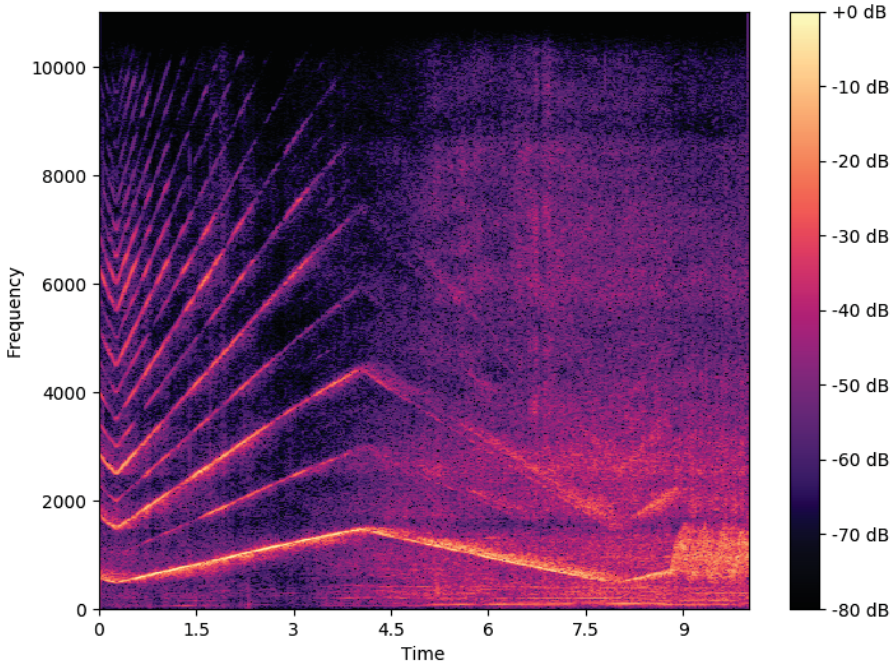


The two next steps would be to simply compute the power spectrum from the amplitude now in decibels and change its representation, accommodating each of these bins across time instead of plot them overlapped on each other. This is done by extracting the periodogram from the frequency spectrum, as shown in 3.7 (HUANG; ACERO; HON, 2001). Figure 40 shows this form known as a spectrogram on linear frequency scale. Observe how is much easier to detect data properties in this form already, which can also be used as features. This way, sound can be identified much better by the formants and their transitions.

$$P = \frac{|FT(\boldsymbol{X})|^2}{N} \tag{3.7}$$

Also note how each fundamental high magnitude peaks in figure 39 can be seen forming the fundamental formant at figure 40.

Figure 40: Spectrogram using the example ambulance audio. Time axis in seconds, frequency axis in Herz, and power normalized to 0 in dB.



For building a mel-spectrogram, we would need to modify some of the latter steps. To be more precise, what we are looking to build is a log-amplitude mel-spectrogram — this is a special kind of spectrogram which instead of using linear scales, uses a logarithmic scale on the magnitude axis and a mel scale on the frequency axis. But first we need to understand what the Mel scale is. The Mel scale relates perceived frequency of a pure tone to its actual measured frequency. As humans, we perceive pitches differently, and we also discern variations in tones

differently depending on its frequency region. The Mel scale maps the frequency spectrum to the mel-frequency spectrum using equation 3.8 (CHAKROBORTY, 2008).

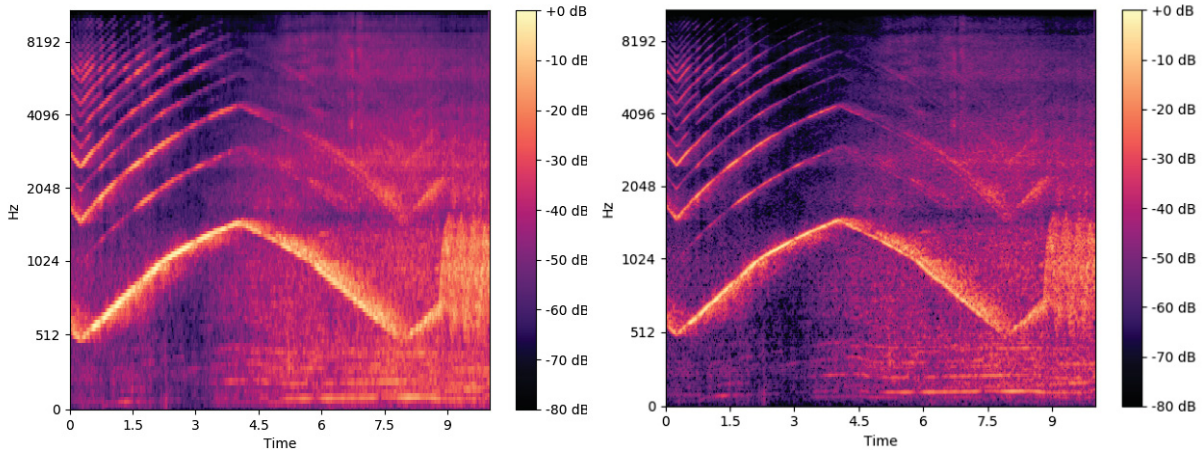
$$m(f) = 1125\, ln(1 + \frac{f}{700}) \tag{3.8}$$

With the mel-scale now applied, what we would see in the current shape of the spectrogram would mimic human sound perception which is notably nearly linear in low-frequencies and approximately logarithmic at mid- and high-frequencies (CHAKROBORTY, 2008). Re-referring to figure 38, instead of directly computing linearly spaced bins via the DFT, we now compute filter banks by multiplying the signal by triangular filters equally mel-spaced between them. Each filter in the filter bank is triangular having a response of 1 at the center frequency and decrease linearly towards 0 till it reaches the center frequencies of the two adjacent filters where the response is 0 (HUANG; ACERO; HON, 2001). This behavior can be implemented using equation 3.9 (HUANG; ACERO; HON, 2001). This will give us the filter bank comprised of an arbitrary number k of filters, combining FFT-obtained bins into Mel-frequency bins. This is called a k-filter Mel filter bank and the bins (or frequency bands) formed by this filter bank are thereafter piled up on the log-frequency axis just as before, but now matching the desired human perception scale, as seen in figure 41.

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \leqslant k < f(m) \\ 1 & k = f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) < k \leqslant f(m+1) \\ 0 & k > f(m+1) \end{cases} \tag{3.9}$$

where $f()$ represents the inter-space of the triangle for each bin, which equals to q list of $m + 2$ mel-spaced frequencies, and m is the total number of desired filters.

Figure 41: Mel-spectrograms using the example ambulance audio. Time axis in seconds, frequency axis in Herz, and power normalized to 0 in dB. Left $m = 128$ and Right $m = 512$.



From this point on, we could optionally extract the MFCC. The Discrete Cosine Transform (DCT) can be used to decorrelate filter bank coefficients, a process also referred to as whitening (HUANG; ACERO; HON, 2001). For CNN this is not necessary, as the representation alone will be fed to the initial feature extraction layers, and the work of understanding and extracting meaningful features from the data can be left to the NN. Some ML algorithms may struggle or not be capable to work with this representation alone at all, thus MFCC would be of great use in those cases. But for this project, it is not needed at all.

Before approaching the next sub-package, it is indispensable to talk about the one-hot encoding function contained in the pre-processing sub-package. This is a static method acting as a middleware, taking the entire data labels on a vector and comparing it against a list for checking for the number of unique classes present in the data and in case of a mismatch between the user intentions and what is contained in the array, a error is thrown. the one-hot encode is really simple, as it takes categorical labels or integer labels and encodes it using a fixed sized vector of bitstreams. This is a widely accepted and used form of identifying and storing labels throughout data. Now categorical data is defined as variables with a finite set of label values, can be easily be converted to integer data and vice-versa, and lastly the number of bits won't vary according to class, as all bitstreams hold a constant size which is identified before running any training operations.

3.2.2 Classification

At this point in the system the `preprocessing` sub-package has prepared the input data, applying generic data massage methods, augmentation and finally returning the pre-

processed audio mel-spectrograms matrix and a one-hot encoded labels array. Note that at this moment we are not dealing with audio data anymore. This approach is inspired by Hershey (HERSHEY et al., 2017) and the idea of extracting visual information from data to only then process it through the CNN.

Also, it is pivotal to check if data is synchronized. For this project, automatized tests are responsible for ensuring labels correspond to their origin audio streams, and when the application is run using `debug` mode, the user can visually check if the spectrogram being carried out actually matches the label being shown. This shown to be a great tool during the initial development.

This sub-package counts with a class responsible for training (`Classifier`), a module for injecting new architectures seamlessly (`_nn_architectures.py`), a debugging module (`data_verification.py`), a class responsible for evaluating model performances (`Evaluator`), and finally a module responsible for bootstrapping and executing predictions using a previously trained model (`predict.py`).

The application core class `Classifier` contains methods for taking the both examples matrix and the labels vector and perform the classifier model training when the user specifies the `model_fit` mode.

First, the class initializes taking user arguments for controlling the training lifecycle, such as using or skipping previous training checkpoints. Currently the class does not support auto hyperparameter scheduling. This method instead of receiving a single hyperparameter, would take a vector of values and train multiple models using each one of them, so we may select the best one afterwards. It does support some different hyperparameters, however.

CNN are on the opposite spectrum of Random Forests when talking about parametrization (LI; JOHNSON; YEUNG, 2018) (PATTANAYAK, 2017). CNN provide a huge amount of hyperparameters, and therefore, hyperparameter tuning efforts as well. There are some further hyperparameters which also have to be configured aside from the ones already seen in section 2.1.5. They can be correlated to the DNN architecture structure or to the training behavior itself.

Training Using a software component responsibilities segregation approach, the CNN structural design was moved and injected from another module. This guarantees new architectures can be easily built and called from within the `Classifier`. This project does not aim to train, compare, and achieve the best model for the given problem proposed to be solved, but rather

looks to accomplish a valid prototype which proves and points the direction for further developments using audio sensors for automatic driving applications. Nevertheless, three architectures were built — a simple baseline implementation inspired on (CHOI; FAZEKAS; SANDLER, 2016), an adapted AlexNet implementation (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), and lastly a adapted VGG16 implementation (SIMONYAN; ZISSERMAN, 2015). The AlexNet implementation tried to mimic to the best of the authors' abilities the original architecture, keeping the same number of activation constant and its original 8 layers provided by the original paper. As previously mentioned, what changes between these architectures are their morphology and activations. The CNN architectures had to define the Number of Layers, Number of hidden units per layer, Activation Functions, Kernel Sizes, Activation Padding, and Filter Stride. While for the training procedure, the only hyperparameters used were the Learning Rate and Number of Epochs.

For brevity purposes, only the most prominent architecture will be further approached in this thesis, which is the AlexNet architecture. In the original paper, the authors used a 224 $\times$ 224 $\times$ 3 input layer, and 4 pixels of stride. This is due to the nature of the data they used in their experiment. Therefore, this topic will be resumed in the results section, when input data size is known.

Before beginning the actual training, the `Classifier` verifies the consistency in data and then splits data between two subsets. A training and a test/development set. Those two sets are synchronously shuffled, so no bias is introduced into the training process, as until this moment all the audio excerpt spectrograms were still sequentially after each other. At this moment, it is of extreme importance that both the examples and the labels are split and shuffled synchronously. The application uses `TensorFlow` backend to accomplish model training and evaluation.

The final model (architecture plus weights) are saved using HDF5 format files. The Hierarchical Data Format version 5, is an open source file format that supports large, complex, heterogeneous data. It is largely used for high-demanding spaces such as backing up whole databases.

Model Evaluation The model evaluation class is called `Evaluator` and inherits from the `Classifier` due to the nature of sharing many of its functionalities to operate. It also performs a similar task if we consider bootstrapping, but instead of loading data from `__audio_input__` as its training counterpart, it loads audio data from `__eval_audios__`. It is important to

note that data loaded in this mode has to match the characteristics of the training data, in all aspects, even excerpts duration.

After a similar bootstrap process to training is done, the input matrix is injected into the classification model to extract its performance metrics. At this phase, all layers are frozen, meaning their weights are not open to back-propagation tuning anymore. Following recommendations found in (LI; KATANFOROOSH,), to compare and thereafter get to the best model possible using architecture and parameters tweaking some kind of measurements are needed to quantify their characteristics somehow. These concepts have already been approached in section 2.1.4.The first obvious performance measurement implemented in this module is the extraction of the confusion matrix, comparing ground truth labels against predictions. With the confusion matrix, the most important classification metrics are calculated — precision, recall (or sensitivity), and F1-score. Since we are not looking for maximizing either true positives or true negatives specifically in this application, the F1-score which combines both previous metrics will be the focus as it can summarize how well the model performs.

Another good metric which can be used also provides a visual feedback is the extraction of the Receiver Operating Characteristic (ROC) curve. This curve shows the performance at all classification thresholds, i.e., how many more False Positives are going to happen in order to obtain more True Positives. That is to say, we would set how liberal the model is, because if we would classify everything as a positive, we would maximize both True Positives and False Positives. This would be the case of having a threshold of 0. All thresholds from 0 to 1 is exactly what the ROC curve shows. However, only having the visual information does not mean much for us. Therefore, to extract a meaningful value that provides us a way to compare models quantitatively is the AUC. It measures the entire two-dimensional area underneath the entire ROC curve from graph origin to (1,1), which also means we do the integral of the curve. Therefore, the bigger the AUC, higher the rate of True positives, without giving up too many False Positives. That means the model discriminates the two classes while really understanding the underlying concept that steers this behavior.

Making Predictions This last module is responsible for taking audio files provided in the `__predict__` directory and providing them to the model to actually carry predictions out. This module also provides a bootstrap customization to only initialize what is important for making predictions using the current model available in `__output_files__/models`, created during training. It also provides some tools, such as providing verbose on the input data, and providing the the feature extraction method and frame sizes used in training, so they are compatible with the model.

This module simply outputs classification against each excerpt with the periodicity given by the user, along with the confidence level associated to each prediction. After outputting the results to the terminal, using Numeric Python package, it saves the predictions to a storage format with extension .npy, which is simply a matrix container which can be later on loaded in the next sub-project, if that would be the intent of the user.

3.2.3 Utilities

This sub-package simply provides some auxiliary tools which were developed to support common tasks such as splitting audio files into excerpts or trimming leading or trailing silence. The implementation is really similar to the one provided via `InputPreparation` but adapted for quick application without the need of any bootstrapping nor application context. Also the CLI manager can be found in this sub-package, interfacing user inputs and the application itself.

3.3 Audio Source Localization

This last sub-project connects the classification results with the sound source localization system. From a re-utilization perspective this is the sub-project which offers least offers generalization, being tightly coupled to the track tests done in this project. This sub-project has three main objectives:

1. Load and interpret classification data
2. Call the chosen MATLAB function with the correct arguments provided via the script using the MATLAB engine API
3. Fill and save the result object using `JSON` data structure

Id	Routine Category	Activity Description	Recording Duration
1	Static Placement	Static Target directly in FRONT of the center point of the array - 10 m	60 seconds
2	Static Placement - Siren Toggle	Static Target directly in FRONT of the center point of the array - 10 m - Toggles: 0 (on), 15s (off), 30s (on),	60 seconds
3	Static Placement	Static Target directly in FRONT of the center point of the array - 35 m	60 seconds
4	Static Placement	Static Target directly in LEFT of the center point of the array - 10 m	60 seconds
5	Static Placement	Static Target directly in LEFT of the center point of the array - 35 m	60 seconds
6	Static Placement	Static Target directly in BACK of the center point of the array - 10 m	60 seconds
7	Static Placement	Static Target directly in BACK of the center point of the array - 35 m	60 seconds
8	Static Placement	Static Target directly in RIGHT of the center point of the array - 10 m	60 seconds
9	Static Placement	Static Target directly in RIGHT of the center point of the array - 35 m	60 seconds
10	Circle around	Two circles around ego with a radius of 20 m and speed of 20 km/h	30 seconds
11	Circle around	Two circles around ego with a radius of 35 m and speed of 20 km/h	45 seconds
12	Static Placement - Siren Toggle	Static Target in the opposite side of the track - Toggles: 0 (on), 15s (off), 30s (on),	90 seconds
13	Static Placement - Siren Toggle	Static Target at halfway track distance from ego - Toggles: 0 (on), 15s (off), 30s (on),	90 seconds
14	Doppler Effect	Target offset to the LEFT 3.5 m from the center of the car. Incoming from the FRONT to the REAR - 60 km/h	30 seconds
15	Doppler Effect	Target offset to the LEFT 3.5 m from the center of the car. Incoming from the REAR to the FRONT - 60 km/h	30 seconds
16	Doppler Effect	Target moves away in a straight line, turns around, and returns to the original position - 60 km/h	60 seconds
17	Doppler Effect	Target offset to the FRONT 3.5 m from the center of the car. Incoming from the LEFT to the RIGHT - 60 km/h	30 seconds
18	Doppler Effect	Target offset to the FRONT 7 m from the center of the car. Incoming from the RIGHT to the LEFT - 60 km/h	25 seconds
19	Autobahn join simulation	Target simulates joining an Autobahn lane by taking over Ego from the RIGHT	25 seconds
20	Autobahn join simulation	Target simulates joining an Autobahn lane by taking over Ego from the LEFT	25 seconds
21	Use case simulation	Target approaching Ego from its REAR. Object causing occlusion between Ego and the Target	30 seconds

Table 5: Description of test routines used in the second test.

First, a file called `prediction_output.npy` placed in the root of the project gets loaded in `main.py`. This file contains all the estimated classifications for each time frame. If the class for the frame is negative, the algorithm will skip localization, as there is no ambulance to be localized. In the opposite case, the localization gets called, and based on which localization algorithm is selected python calls the MATLAB API, invoking the corresponding script developed for doing localization estimations and evaluating against the ADMA ground truth data, as seen in section 3.1.2.

In MATLAB, two folders contain different kind of scripts, the first one contain ADMA post-processing scripts. Those scripts make their best effort to match timestamps in ADMA data and recording data and plot the position of objects of interest found in ADMA data. The second folder contain scripts for loading and filtering audio data and also the actual source localization algorithms, GCC-PHAT and SRP-PHAT. It takes the 4 microphones channels audio data as input (our 4-microphone-squared array), along with some experimental parameters, such as ADMA sampling rate, sound speed in meters per second and the distance between the microphones, forming the array. SRP-PHAT implementation is based on (DO; YU; SILVERMAN, 2007) and brought very poor results while the GCC-PHAT based on section 2.2 with some tweaks brought decent ones. The main tweak needed was to zero the first τ values, both to the left and to the right. This is due to GCC-PHAT not finding any specific correlation between the to microphones in regard of the interest signal, then correlating the noise which is a stochastic process and since the power distribution is even, it maximizes at 0. This led to some improvement, but the localization algorithm had to deal with serious limitations as can be seen in section 4. Equation 2.28 results in $\pm\theta$ angles which are ambiguous, and that ambiguity has been tackled via hard coded heuristic rules that assume that the first two microphones to detect the signal would indicate a proper wavefront DOA. It seems to be a poor assumption, and is the probable cause of unsatisfying results.

`main.m` mimics the behavior of the project root `main.py` serving as a switch between the localization algorithms, but it also offers validation. Therefore, running this script is needed for getting analytical scoring for validating the estimation data against the ground truth data collected via ADMA. Validation takes a single parameter called tolerance, which is the degrees aperture the algorithm accepts deviation between estimated values and ground truth ones.

Lastly, the classification and azimuth estimation is dumped onto a `JSON` file, as the final result of this project. File `result.json` contains the fields [frameStart, frameEnd, emergencyVehicleProb, noEmergencyVehicleProb, azimuth]. Therefore, we would have the confidence levels along with the inferred class and the estimated azimuth per frame.

4 Results

This section covers the practical use of the all the fundamentals and tools built through this thesis. We will also cover needed tweaks to the tools when necessary, describing the whole process from user perspective now.

4.1 Model Training

The classifier accepts audio input in folder `__audio_input__` by default, thus all directories containing the audios were placed inside of it. Model training proved to be a really tough and problematic step. The first challenge was to select and successfully implement the best architecture possible respecting the timeframe of the project. The first step was to look for a baseline architecture, known for working accordingly with audio applications — this was fundamental for isolating problems, ease debugging and have a first working example. It was difficult to break through the 50% accuracy threshold, as it proves the algorithm is just guessing since this is what we expect having two options when just trying by chance. It denotes the algorithm has no understanding over the problem. The implementation used by the author was based on the (CHOI; FAZEKAS; SANDLER, 2016) implementation, but instead of using fixed amount of layers as for example the 4-layers MLP used in the reference, the architecture was generated generically depending on the number of layers specified by the user. The architecture presented scaling problems, being unable to learn further than presenting mediocre 78 or 79% accuracy even with architectures as big as the 12gb GDDR memory of the Nvidia Titan X. In reference to (CHOI; FAZEKAS; SANDLER, 2016), many modifications were necessary.

The input layer depends on the input data. The results achieved by (HERSHEY et al., 2017) led the author to follow similar parameters for the audio pre-processing, considering differences in application. (HERSHEY et al., 2017) had a very wide of audio applications to consider, hence he had to find a good mid-ground between small audio segments that would work for impulse-like sound responses as gunshots and large audio fragments which work better for long lived events, such as a song. For that same reason, sirens should sit at a similar place,

as short segments as 50 ms would make it difficult for a model to identify inherent ascents and descents on the frequency and thereafter to recognize periodicity along the signal.

Information in the previous paragraph led the author to build the standard input tensors sizes: The frequency is arranged in 128 bands, which as seen in previous chapters partitions the frequency scale into bins, and transforms each bin into a corresponding bin in the mel scale. Therefore those 128 bins are evenly spaced in the mel scale, so that is to say the algorithm will learn from the training data as humans would. Even though the whole Dataset 1 is loaded in the application's bootstrap, the feature extractor discards any data which is not conforming to the standard duration detected in the whole dataset. This is done using the maximum duration identified in the dataset which is 10 seconds long — which with a sampling rate of 48 kHz totals for 480,000 samples in the audio array. Any audio array which deviates from this value is not used. A windowing function was then applied to transform the excerpts into frames (the definition of audio frames comes from the number of audio samples which are played to match a single video frame, but that is a rather wide definition). Frames were windowed within 46080 samples (960 milliseconds) following (HERSHEY et al., 2017) approach, and then these frames were fed into the mel transform to extract the mel-spectrograms using a fixed hop length of 480 samples (10 milliseconds) and Fourier windows size of 1200 samples (25 milliseconds), yielding $960 \div (25 \div 2.5) = 96$ decomposed frames per original frame. Note that the overlap of 40% per hop since each window can fit 2.5 times each shift in time. And lastly, since there were 2 channels present in for each audio, the tensor shape per frame was $96 \times 128 \times 2$. Hence, the first layer (input layer) of our CNN architectures have to be able to accommodate this shape.

Also, the sigmoid function do work on output layers for multi-class classifiers, mapping the output to probabilities which fall between 0 and 1. However, since our output only has two complementary probabilities the softmax function was employed instead.

Using the same approach as seen above, there were two good candidates which presented good performance in competitions in the past — the AlexNet architecture (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) and the ResNet architecture (HE et al., 2015). ResNet showed to be problematic to fit in memory as no good results were obtained used low layer count ones. In the counterpart, AlexNet showed very promising result in pilot results, and after much tuning the end result utilized is as 6 shows.

There were three most important changes from the original architecture that needed to be changed or included in the implementation. THe first was to include sparse dropout throughout the layers. It greatly helped the architecture not to overfit the training data,

Table 6: AlexNet-based architecture employed for model training

Layer	Filter Size	Stride	Input Size
Conv1	11x11	2x2	96x128x2
MaxPool1	2x2	2x2	64x48x48
Conv2	5x5	2x2	32x24x48
MaxPool2	2x2	2x2	32x24x128
Conv3	3x3	2x2	18x14x128
Maxpool3	2x2	2x2	18x14x256
Conv4	3x3	2x2	9x7x256
Conv5	3x3	2x2	11x9x512
MaxPool5	2x2	2x2	13x11x512
FC1	-	-	15360x1x1
FC2	-	-	1536x1x1
FC3	-	-	2048x1x1
Output	-	-	2x1x1

diagnosed by high accuracy against the training data and low accuracy against the test set. This proven to be true when applying it on the learning transfer layers, as the feature learning itself don't benefit from regulation — and feature extraction happens on convolutional layers, thus the option of applying regulation more aggressively at the fully connected layers. Also Batch normalization was employed instead of Local Response Normalization. Lastly, the since AlexNet was created to handle 224×224 shape per image channel, we wanted to achieve the same number of activations in the input layers, as our input is different. Thus, since the original stride is kept at 4×4, and our data is composed by a 96×128 shape per channel, the original number of activations was calculated using $224 \cdot 224 = 5176$ (closest possible approximation) that in turn has a moving stride of $4 \cdot 4 = 16$, totaling $50176 \div 16 = 3136$ activations. In order to achieve an analogous number $128 \cdot 96 = 12288$, that turned to be $12288 \div 3136 = 3.91$. This means stride should be adapted to cover 4 tiles over the tensor per hop, which means a 2×2 stride needed to be implemented.

Also is worth noting that to fit in memory, the architecture was reduced in half regarding layer width. The original AlexNet had 62.5M parameters to be trained, whereas our architecture presented 30.7M, which is roughly half. Other multipliers were tried out as well, but showed not significant difference or benefit to the one chosen. The same ReLU activation function were used along with the same softmax activation function on the output layer.

Now, for training, the cross-entropy in all cases, and after iterating over a plethora of optimizers, the one most performed was the Adadelta optimizer. Batch normalization was applied after all convolutional layers and a very slight l2 regularization was applied using a value

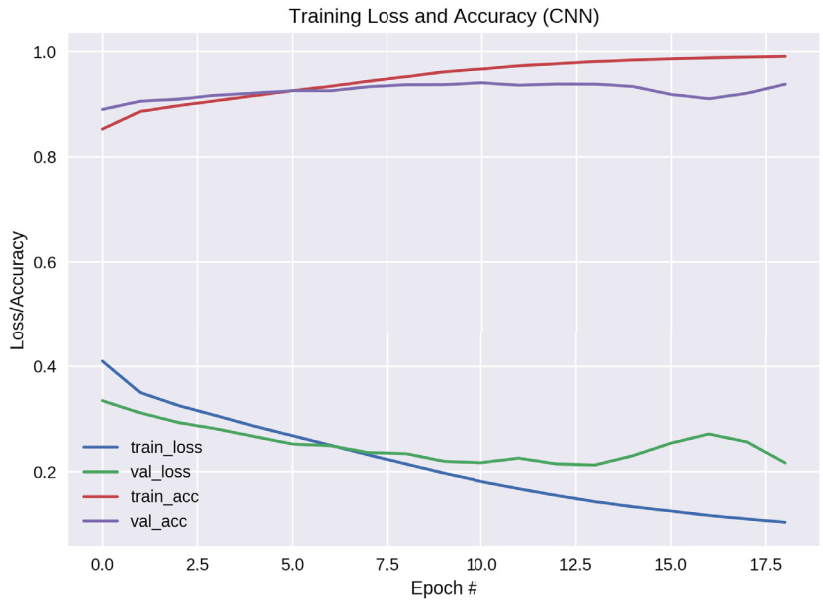
of 0.001. Training was carried out using the label association `[class 0: No EV, class 1: EV]` — EV standing for Emergency Vehicle. All trainings were employed using a Early stoppage patience of 1/20 Epochs, while the number of Epochs to be run was 100, which bring us to 5 Epochs patience. Adaptive learning rates did not show much results, therefore learning rates of 0.001 and 0.00001 were the most successful, as the best performing model was trained using a learning rate of 0.00001. The batch sizes could accommodate numbers as high as 32 using the huge chunks of memory available to us. For the chosen model, the training report can be seen in figure 42, while the training target loss and accuracy over time can be seen in figure 43.

Figure 42: Training report for chosen model.

```
TRAINING REPORT:

INFO:root:Identified 3 dimensions in the input: (128, 96, 1)
Epoch 00019: early stopping
Starting training evaluation...

              precision    recall  f1-score   support

       No EV       0.91      0.97      0.94      1756
          EV       0.97      0.90      0.93      1691

   micro avg       0.94      0.94      0.94      3447
   macro avg       0.94      0.94      0.94      3447
weighted avg       0.94      0.94      0.94      3447
```

The train report shows a sensitiveness of 0.9, which while not being ideal as the model is too liberal, classifying other sounds like silence as the positive class, shows a good performance even though. Support means the total number of labels used for each case. As training was nearly balanced, micro and macro averages turned to be similar. Macro averages calculate metrics globally, micro averages calculate metrics for each label, finding then their unweighted mean, and weighted average does the same as the micro average, but use their average taking in account the number of samples present for each label, i.e. support. The latter is most useful in unbalanced cases, which does not describe the current one.

Figure 43: Training history for chosen model — shows convergence rate and levels plateau.



Then, using the acquired model, evaluation was carried out using random unknown audio sources from youtube which were not included in neither the training nor the test dataset, along with samples from the first and second test. This small evaluation dataset was manually annotated and show very impressive results. This model has a very specific scope of classifying traffic auditory scenarios regarding presence of emergency vehicles, but table 7 shows the confusion matrices extracted from from the evaluation reports run against 4 key files: the first one is a very negative unbalanced one, the second a very positive unbalanced one, the third were CARISSMA test from the first tests (both tests describe the same routine the only difference being the tone used in the recording), and the fourth one was a tests from the second CARISSMA using a routine designed to test the algorithm with intermittent toggles on the siren.

Figure 44: ROC curve for CARISSMA first test scenarios 33 and 34.

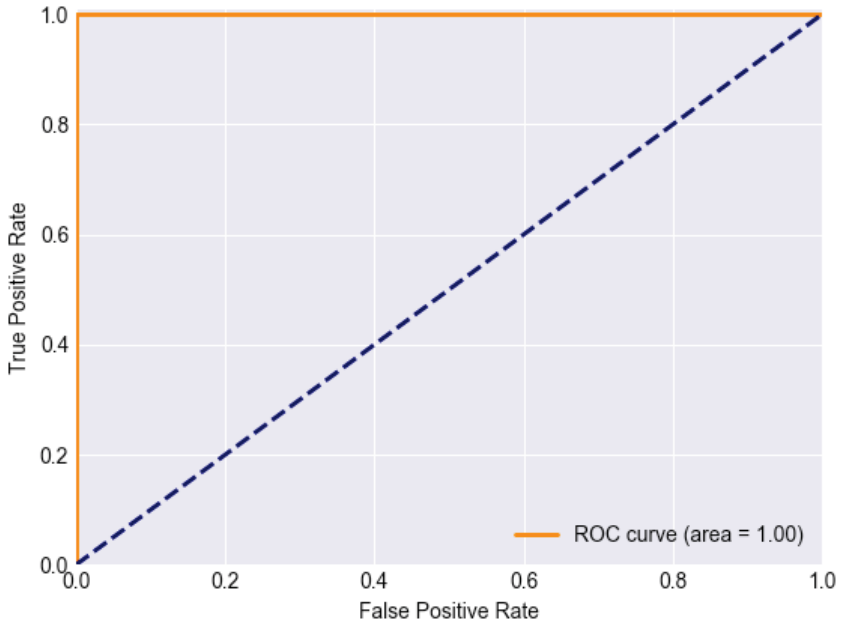
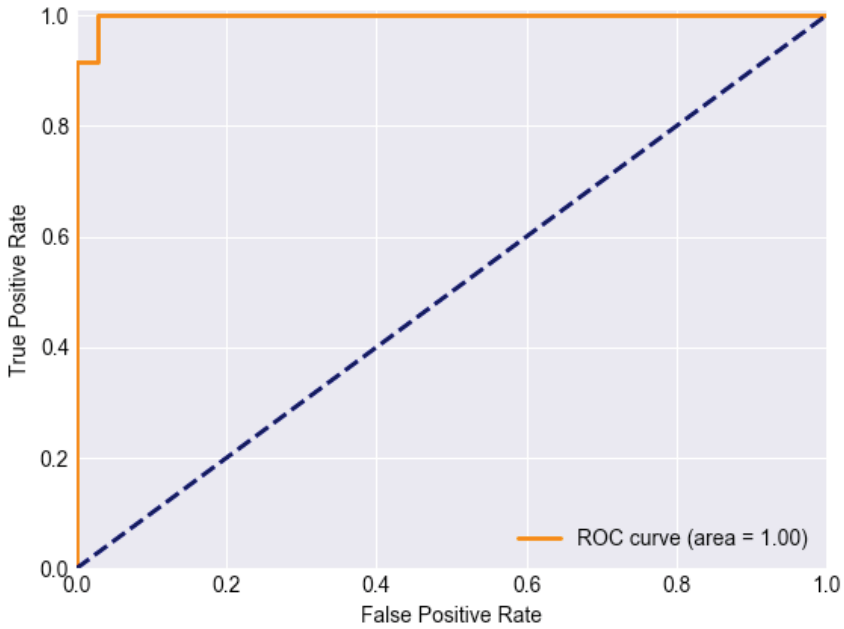


Figure 45: ROC curve for CARISSMA second test scenario 13.



The youtube videos can be found on the urls `https://www.youtube.com/watch?v=Bpu4dXxMhOs` and `https://www.youtube.com/watch?v=ac9FMGeA8FQ&t=3s`.

Confusion matrices are then synthesized in metrics. The evaluation routine calculates class precision, class recall, overall accuracy, and AUC calculated using the ROC curve (figure 44 and 45). Depending on their actual class computed, as false positives (FP) or true positives (TP), the ROC it takes a step upward on the y-axis, and when it passes a FP it takes a step

Table 7: Model evaluation using

Parameter	200 Police Cars In Las Vegas Driving Code 3 In Funeral Procession	2018 NorthamptonPA Fire Department Block Party Parade 92218	CARISSMA first test – Scenarios 33 and 34	CARISSMA second test – Scenario 13
True Positives	4	408	24	34
True Negatives	451	82	11	35
False Positives	5	3	1	1
False Negatives	3	31	0	2

Table 8: Model evaluation using evaluation mode.

	200 Police Cars In Las Vegas Driving Code 3 In Funeral Procession		2018 NorthamptonPA Fire Department Block Party Parade 92218		CARISSMA first test – Scenarios 33 and 34		CARISSMA second test – Scenario 13	
Metric	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
Negative class	0.99	0.99	0.73	0.96	1	0.92	0.95	0.97
Positive class	0.44	0.57	0.99	0.93	0.96	1	0.97	0.94
Micro Average	0.98	0.98	0.94	0.94	0.97	0.97	0.96	0.96
Macro Average	0.72	0.78	0.86	0.95	0.98	0.96	0.96	0.96
Test Accuracy	0.9827214		0.9351145		0.9722222		0.9583333	
AUC	0.9799498		0.979874		1		0.9976851	

rightward on the x-axis. The step sizes are inversely proportional to the number of actual positives (in the y-direction) or negatives (in the x-direction), so the path always ends at coordinates (1, 1). The result is a plot of true positive rate (TPR, or sensitivity) against false positive rate (FPR, or 1 - specificity), which is all an ROC curve is. Computing the area under the curve is one way to summarize it in a single value; this metric is simply the integral took from the axis base (0, 0) until the maximum (1, 1) which lies underneath the plotted ROC, and it is so commonly used that any mentions to "area under the curve" or "AUC", generally assume we mean an ROC curve unless otherwise specified.

Once again, even though reference article scope (MARCHEGIANI; POSNER, 2017) is larger than the one presented in this project, results output by the our model are more robust and perform better (0.87 against 0.98 average AUC in evaluation according to table 8).

Prediction mode outputs a raw prediction, with no knowledge regarding ground truths on the audio. This is what we would see if the algorithm would be run in production. The following excerpt exemplifies the output the algorithm returns.

```
Running Predictions on file: FL.wav
2019-06-20 10:43:27.104246: I tensorflow/stream_executor/dso_loader.cc:152] successfully opened CUDA library libcublas.
    ↪ so.10.0 locally
92/92 [==============================] - 6s 60ms/sample
X=From 00:00:00.000 to 00:00:00.950, Predicted=0 (No EV) with Confidence of 96.4%
X=From 00:00:00.950 to 00:00:01.900, Predicted=0 (No EV) with Confidence of 96.1%
X=From 00:00:01.900 to 00:00:02.850, Predicted=1 (EV) with Confidence of 59.3%
X=From 00:00:02.850 to 00:00:03.800, Predicted=1 (EV) with Confidence of 78.7%
X=From 00:00:03.800 to 00:00:04.750, Predicted=1 (EV) with Confidence of 95.7%
X=From 00:00:04.750 to 00:00:05.700, Predicted=1 (EV) with Confidence of 97.0%
X=From 00:00:05.700 to 00:00:06.650, Predicted=1 (EV) with Confidence of 94.4%
```

```
...
X=From 00:01:20.750 to 00:01:21.700, Predicted=1 (EV) with Confidence of 91.6%
X=From 00:01:21.700 to 00:01:22.650, Predicted=1 (EV) with Confidence of 77.9%
X=From 00:01:22.650 to 00:01:23.600, Predicted=0 (No EV) with Confidence of 92.8%
X=From 00:01:23.600 to 00:01:24.550, Predicted=0 (No EV) with Confidence of 97.2%
X=From 00:01:24.550 to 00:01:25.500, Predicted=0 (No EV) with Confidence of 96.6%
X=From 00:01:25.500 to 00:01:26.450, Predicted=0 (No EV) with Confidence of 96.1%
X=From 00:01:26.450 to 00:01:27.400, Predicted=0 (No EV) with Confidence of 97.5%
```

After every prediction session, a file is created on the root of the project in as `prediction_output.npy`. This is a numpy matrix carrying all the prediction results, which can then be loaded when performing further tasks. This is the mode used which will be used for end-to-end validation.

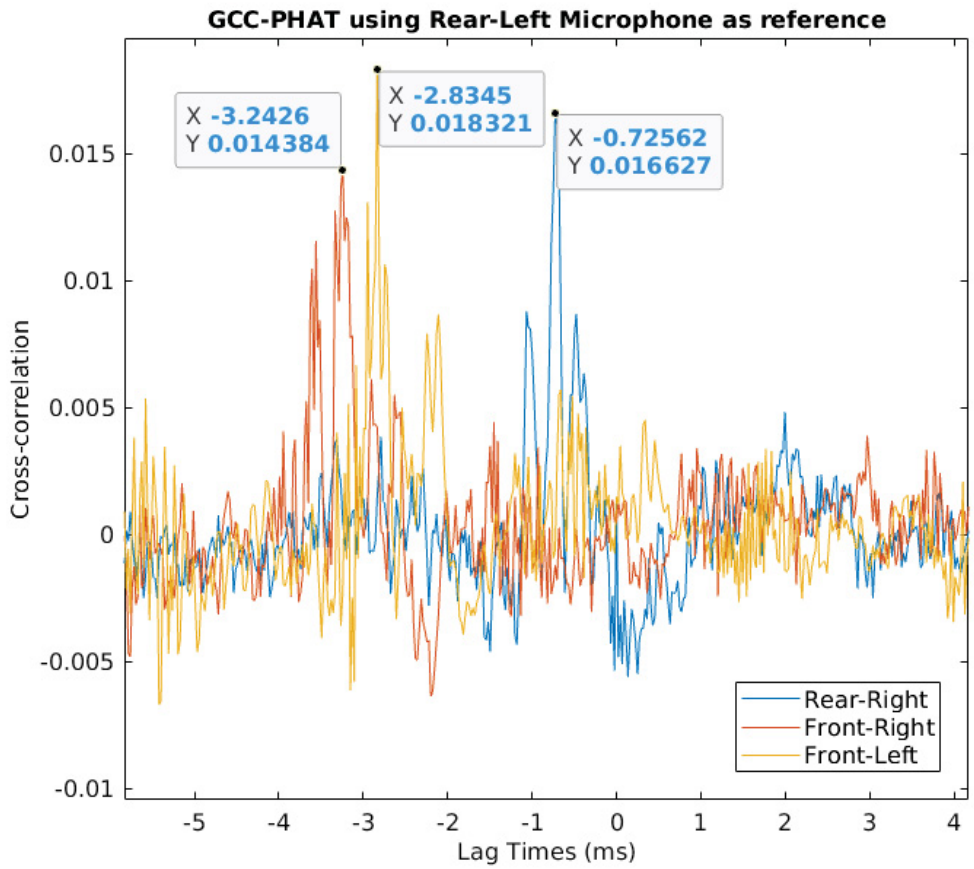
4.2 Source Localization

Last step is to load the prediction output using the `source_localization` component via a python interface — it uses the MATLAB engine API to call the by default GCC-PHAT localization algorithm. The results for static scenarios had an estimation success rate of 0.89 as we can see in table 9, but for moving targets it could not provide reliable data achieving a mere 0.03 estimation success rate. Thus, for dynamic scenarios it does not track the source position. The files are loaded on MATLAB after the API call done by the python client, then after high-pass filtering, GCC-PHAT is calculated. Figure 46 shows the rear left microphone as reference, and all how all other microphones interact with it. In scenario 1 of the second test, we know the target is in the front of ego, and this can attested here, since we can see the front microphones are maximally correlated with a delay of approximately -2.5 milliseconds, so the wavefront arrives first at them, and only then they arrive at the rear microphones. Also it is interesting noting that the rear microphone is maximally correlated at almost 0 milliseconds, which also makes sense, since the target is located roughly directly in front of ego, thus the sound wave would arrive nearly at the same time on both microphones. Afterwards, using DOA equations along the heuristic rules we get our estimations and the representation in figure 47 and figure 48.

Table 9: Estimation results against ADMA data for — Scenarios from the second CARISSMA tests.

Scenario	Correct estimations	Incorrect estimations
1	41	1
3	47	2
4	22	7
5	27	5
6	29	0
7	42	1
8	13	7
9	28	5
16	3	30
17	0	29
18	1	19

Figure 46: Cross-correlation versus time lag between the microphones



Bad performance on dynamic cases suspects are the GCC-PHAT algorithm itself — multipath issue might come in case — since at times it was detected that the audio wavefront was arriving at a different direction than the expected and the synchronization between timestamps from ADMA and the recording — when the difference in the timestamps

is bigger than the test itself, it tries to match only via sample index, which does not guarantee synchronization at all, thus that would be the explanation on why static cases could behave differently. Efforts to tackle the issues manually did not have much effect. An heuristic method was responsible for tracking the first microphone to be stimulated, then the second and so forth. The idea behind it was that the audio would come directly from the source, but it showed not to be robust against wavefront multipath. During debugging, at many times the cross correlation was correct, the time difference was correct, but at the end the wrong microphone would detect the incoming wavefront first. (MARKOVIĆ; PETROVIĆ, 2010) also mentions the problem that microphone pairs do not perform well at skewed angles, thus it is important to rotate which microphone pair to be used, striving to always use the microphone pair most orthogonal to the sound source. However, when the wrong microphone pair was activated due to multipathing, it would be positioned unfavorably in relation to the source, deeply hurting the source localization estimation, as can be seen in table 9. Apart from algorithm performance, also other options for instrumentation could be considered, as using an array of MEMS microphones, widely used in robotics and for speech recognition and localization. Note that results found in table 9 use a tolerance of 20 degrees, totaling 89.89% accuracy in the static cases, while only performing 4.88% accuracy for the presented dynamic cases.

Figure 47: Azimuth estimation for scenario 1. Distance displayed comes from ground truth data. X and Y axis are a 2D overhead view in meters.

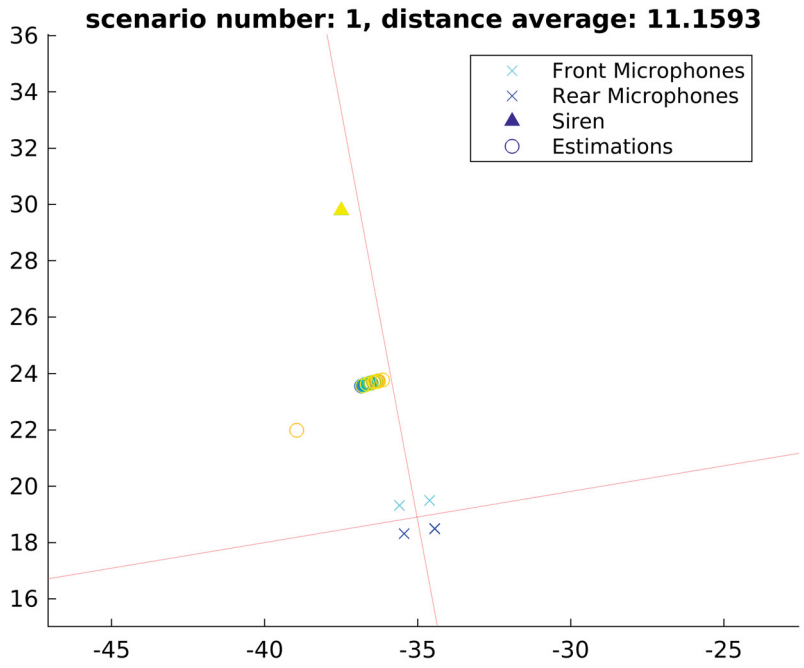
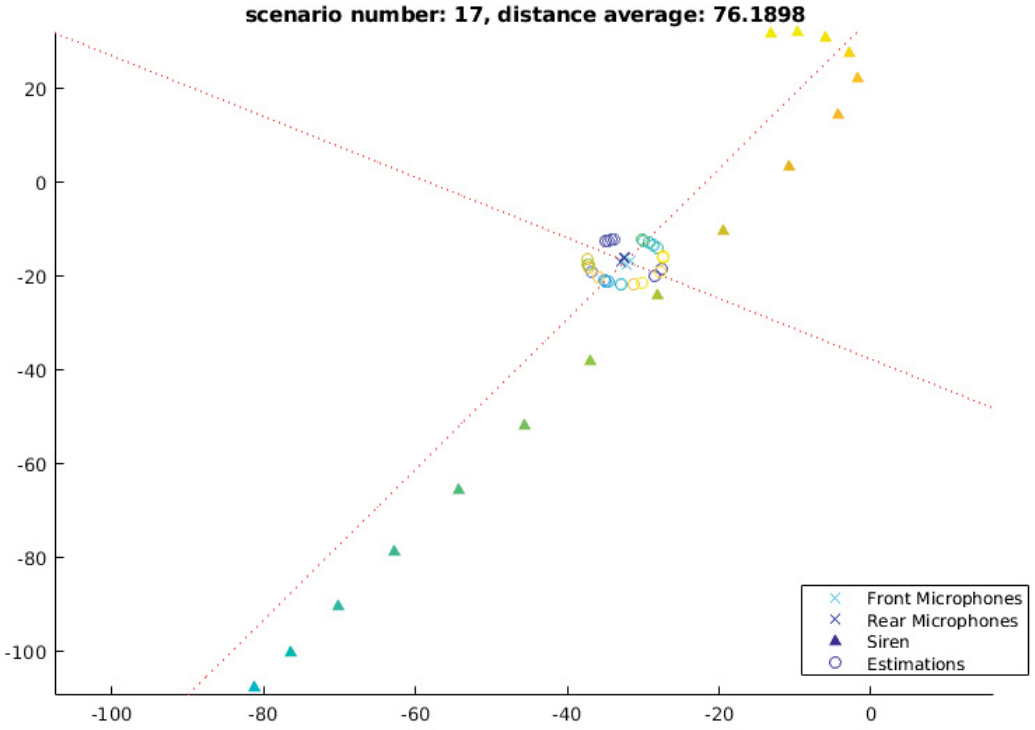


Figure 48: Azimuth estimation for scenario 11. Distance displayed comes from ground truth data. X and Y axis are a 2D overhead view in meters.



4.3 Validation

This section provides an end-to-end example using the *Rettungsgasse* situation. The CARISSMA dataset is used here, since all controlled parameters are known, providing thoroughly ground truth checks against the results. First, the classifier runs in predictions mode (figure 49 presents some key moments in scenario 21), we then copy the `prediction_output.npy` on sound localization sub-project root. After loading the file, we generate the list seen in figure 50.

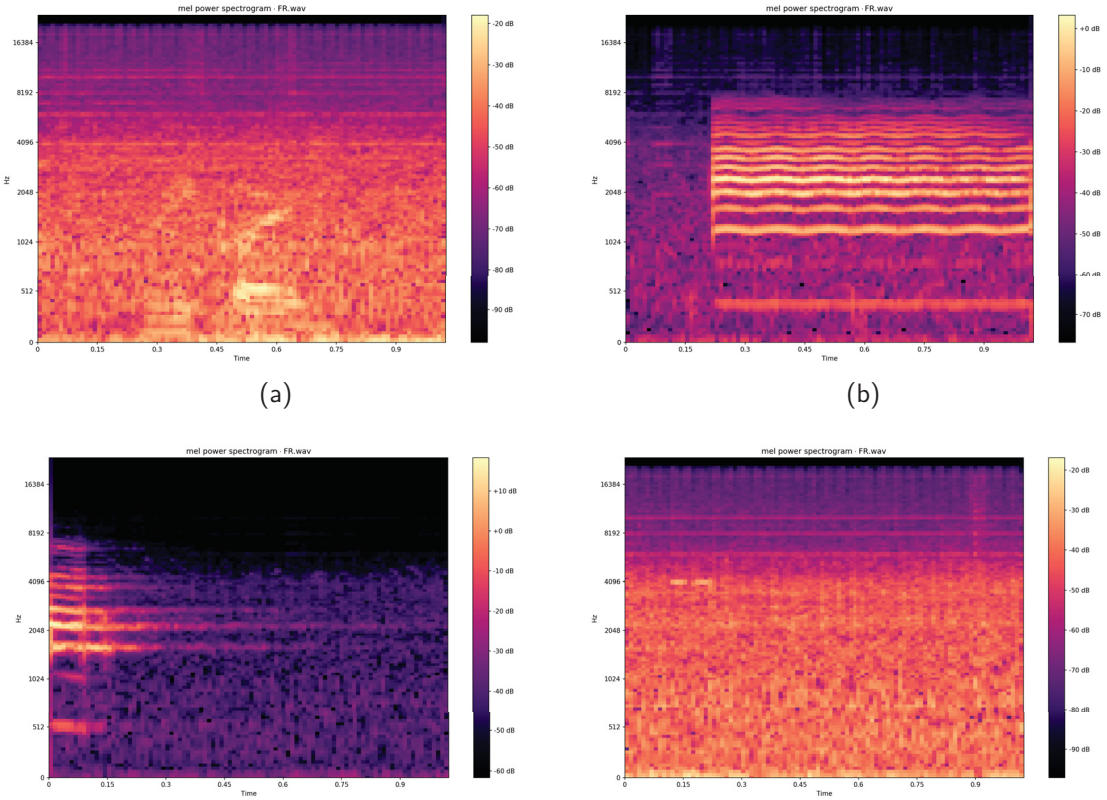
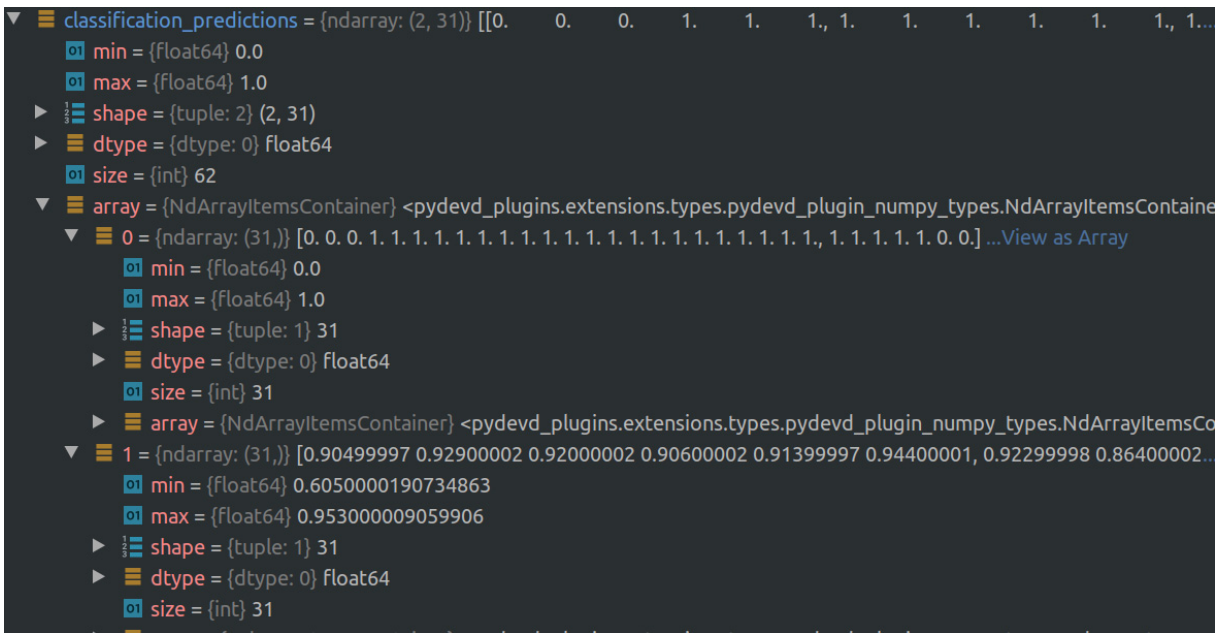


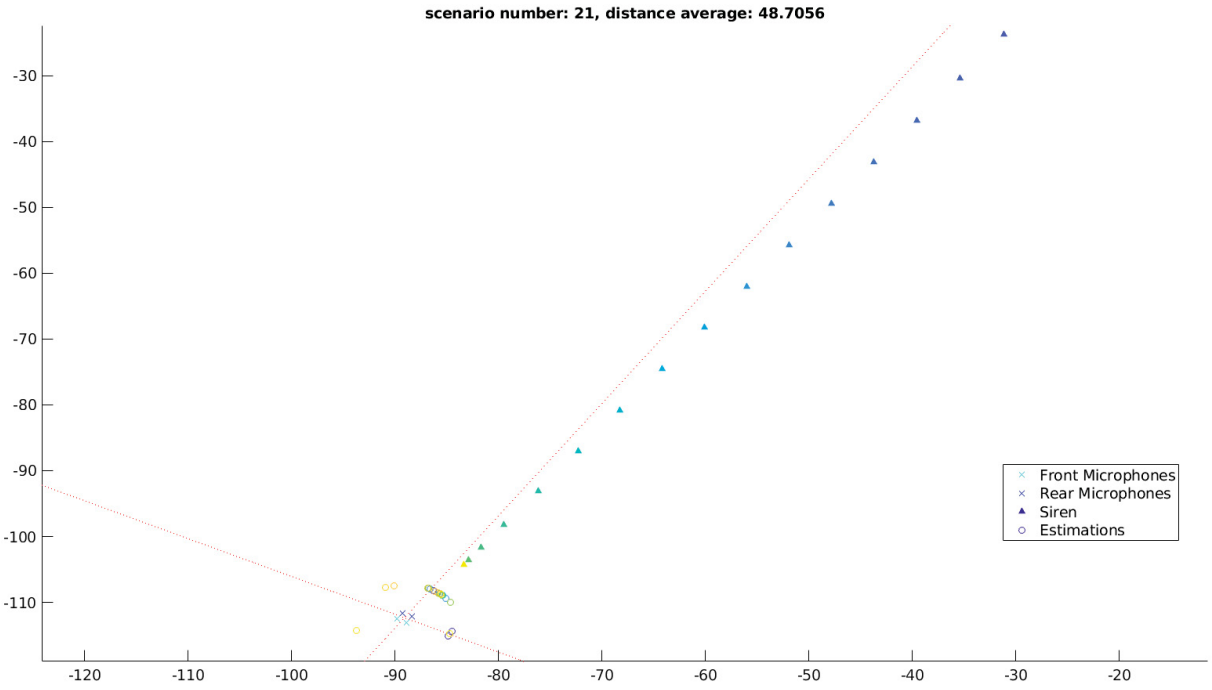
Figure 49: Use case Mel-spectrograms normalized to 0dB. (a)2.85-3.80s (b)3.80-4.75s (c)26.60-27.55s (d)27.55-28.50s.

Figure 50: Attributes of predictions list loaded by the source position estimator component.



For scenario 21, all 31 frames were correctly classified when manually listening to the audio and thereafter comparing the results. Position estimation can be seen in figure 51, totaling 1 correct position estimation and 31 incorrect ones.

Figure 51: Attributes of predictions list loaded by the source position estimator component.



finally it outputs results on a JSON file:


```
[
        {
                "frameStart": 0.0,
                "frameEnd": 0.95,
                "emergencyVehicleProb": 0.095000029,
                "noEmergencyVehicleProb": 0.9049999713897705,
                "azimuth": NaN,
        },
        {
                "frameStart": 0.95,
                "frameEnd": 1.9,
                "emergencyVehicleProb": 0.0709999999999998,
                "noEmergencyVehicleProb": 0.9290000200271606,
                "azimuth": NaN,
        },
        {
```

```
                "frameStart": 1.9,
                "frameEnd": 2.8499999999999996,
                "emergencyVehicleProb": 0,079999983,
                "noEmergencyVehicleProb": 0.9200000166893005,
                "azimuth": NaN,
        },
        {
                "frameStart": 2.8499999999999996,
                "frameEnd": 3.8,
                "emergencyVehicleProb": 0.906000018119812,
                "noEmergencyVehicleProb": 0,09399999999982,
                "azimuth": 56.842393247536,
        },

        ...

]
```

5 Conclusion

This project demanded multidisciplinary knowledge, making use of software architecture, code design, digital signal processing, digital audio and images format knowledge, machine learning techniques, automotive tests, and acoustic instrumentation. It presents the first steps towards a potential *Rettungsgasse* formation function and strives to lay good foundations that may convince the community that audio sensors can be a powerful asset for ADAS functions and autonomous driving.

Through auditory classification it is possible to provide new information to the system that can lead to better overall performance and safety for ADAS and autonomous driving. Classification models proved to be a promising audio object detection algorithm, which can provide a list of objects in the immediate environment,

CNN showed to be a very robust training method for developing classification models, which are widely employed by the scientific community for object detection and tracking. Taking advantage of information extensively available for this kind of application, transforming audio signals into mel-spectrograms, we could then approach the problem as an image detection one. This methodology has proven to be very successful, achieving the AUC of 0.979 in the worst case when validating the classification task using newly acquired data which has never been seen by the model before. This provide us enough confidence to say we have a robust model for the ambulance presence auditory classification model.

The end-to-end architecture laid in this project outputs a list containing object binary auditory classification (presence or not of an emergency vehicle in the surroundings) along with the decision confidence level and the estimated azimuth per audio frame. On the current state, sound source localization using GCC-PHAT works well for static target positions. It was implemented an automated source download robot available for the AudioSet database, a generic and extensible audio classifier powered by CNN, a sound source localization algorithm, and finally an interface to verify all components working together.

The localization algorithm when performed with moving sources did not show the same robustness against the combination of reverberation, echo, and noise. This could be due to the acoustic instrumentation setup, since the comparison between other papers using the same algorithm obtained much better success employing it. Even so, it is not possible to say for sure what caused such poor performance at this time.

Although, many immediate improvements are clear. First, we could fork the auditory classifier project to employ an audio object detector instead — this would make it possible not only to detect an auditory state (as the presence or not of an emergency vehicle), but instead also to provide an actual list of objects that could summarize what objects have been detected in the surroundings. Secondly, instead of assigning the job of localization to an external algorithm, a single model could be able to do both tasks, saving computational resources in exchange of higher project implementation complexity as training data with more than 2 audio channels is much more difficult to acquire. In case we would stick with the current components, then creating a formal interface between the classifier and the source localization algorithm is imperative for the architecture to actually be able to work in a more production-like environment. Thirdly, saving data to disk, so another component has to load it in the following moment does not make any sense for real time applications, and should rather be exchanged by an API, socket or shared application memory with no memory violation restrictions between them. Lastly, the sound source localization could use a more robust algorithm in detriment to GCC-PHAT — also providing target relative distance and elevation — and also make use of MEMS microphones instead of studio recording ones. They are still cheap, so they would still be viable for prototyping, but providing much better results when it comes to beamforming, as they are fabricated with this application in mind.

Regarding takeaways, the author of this project is not convinced whether audio lower features could be useful for the sensor fusion layer, as these are potentially less accurate and robust than other sensors with much higher resolution such as lidars. This would require an extensive study to tackle whether acoustic sensors can provide low level features data on par to the other sensors. Furthermore, to calculate objects azimuth and distance proved to be a much more difficult task than anticipated, and thus this conclusion deviates from Marchegiani e Posner (2017) when it comes to acoustic localization algorithms, treating them as out-of-the-shelves. Choosing among the many GCC family of algorithms, or even further ones depends on the recording techniques, electronics and acoustic instrumentation, and the conditions the sensor will have to work on, such as exposed environment with low SNR or high reverberation levels. This is a complex matter, that has to be addressed taking in account both training and deployment phases.

BIBLIOGRAPHY

AEBERHARD, M. *Object-Level Fusion for Surround Environment Perception in Automated Driving Applications*. PhD Thesis (PhD) — Technische Universität Dortmund, may 2017.

ARBIB, J.; SEBA, T. *Rethinking Transportation 2020-2030*. [S.l.], maio 2017.

Badali, A. et al. Evaluating real-time audio localization algorithms for artificial audition in robotics. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.: s.n.], 2009. p. 2033–2038. ISSN 2153-0866.

BOTSCH, M. *Sensor Technology and Signal Processing Lecture Notes*. out. 2018.

Carter, G. C. Coherence and time delay estimation. *Proceedings of the IEEE*, v. 75, n. 2, p. 236–255, Feb 1987. ISSN 1558-2256.

CHAKROBORTY, S. Some studies on acoustic feature extraction, feature selection and multi-level fusion strategies for robust text-independent speaker identification. *submitted to electronics and electrical communication engineering department*, II T, 2008.

An abnormal sound detection and classi- fication system for surveillance applications. [S.l.]: IEEE, 2010. 1851–1855 p.

CHOI, K.; FAZEKAS, G.; SANDLER, M. *Automatic tagging using deep convolutional neural networks*. 2016.

CHOI, K. et al. Convolutional recurrent neural networks for music classification. *CoRR*, abs/1609.04243, 2016. Available from Internet: <http://arxiv.org/abs/1609.04243>.

Corporate Partnership Board (Ed.). *International Transport Forum - Automated and Autonomous Driving Report*. 2015.

CORPORATION, M. *Global Autonomous Driving Market Outlook*. [S.l.], mar. 2018.

COWLING, M.; SITTE, R. Comparison of techniques for environmental sound recognition. *Pattern Recognition Letters*, v. 24, p. 2895–2907, 11 2003.

DEFEUG. *Rettungsgasse wird gebildet*. set. 2013. `http://www.rettungsgasse-rettet-leben.de/die-rettungsgasse`. Accessed: 2019-08-16.

A Real-Time SRP-PHAT Source Location Implementation using Stochastic Region Contraction(SRC) on a Large-Aperture Microphone Array.

DUREY, A. S.; CLEMENTS, M. A. Features for melody spotting using hidden Markov models. In: . [S.l.: s.n.], 2012. v. 2.

FANDOM. *PCM sampling, converting an analogue signal into a digital format*. abr. 2014. `https://digital-audio.fandom.com/wiki/Pulse-Code_Modulation`. Accessed: 2019-08-28.

GEMMEKE, J. F. et al. Audio Set: An ontology and human-labeled dataset for audio events. In: *Proc. IEEE ICASSP 2017*. New Orleans, LA: [s.n.], 2017.

General Motors (Ed.). *2018 Self-Driving Safety Report*. 2019.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. `http://www.deeplearningbook.org`.

GORDON, M. E.; PEARSON, J. J. Preliminary Analysis of Roadway Accident Rates for Deaf and Hard-of-Hearing Drivers — Forensic Engineering Application. *National Academy of Forensic Engineers*, National Academy of Forensic Engineers, v. 33, n. 1, jun. 2016. ISSN 2379-3252.

GUBBI, J. et al. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 2013.

HE, K. et al. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385, 2015. Available from Internet: <http://arxiv.org/abs/1512.03385>.

HERSHEY, S. et al. Cnn architectures for large-scale audio classification. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2017. p. 131–135.

HRABINA, M.; SIGMUND, M. Gunshot Recognition using Low Level Features in the Time Domain Gunshot Recognition using Low Level Features in the Time Domain. *2018 28th International Conference Radioelektronika (RADIOELEKTRONIKA)*, IEEE, p. 1–5, 2018.

HUANG, X.; ACERO, A.; HON, H.-W. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN 0130226165.

HUZAIFAH, M. Comparison of time-frequency representations for environmental sound classification using convolutional neural networks. *CoRR*, abs/1706.07156, 2017. Available from Internet: <http://arxiv.org/abs/1706.07156>.

KOHAVI, R.; PROVOST, F. Glossary of Terms. *Machine Learning*, v. 30, n. 2, p. 271–274, Feb 1998. ISSN 1573-0565. Available from Internet: <https://doi.org/10.1023/A:1017181826899>.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2012. p. 1097–1105.

LI, F.-F.; JOHNSON, J.; YEUNG, S. *CS231n: Convolutional Neural Networks for Visual Recognition*. Stanford CS class, 2018. Available from Internet: <http://cs231n.stanford.edu/>.

LI, F.-F.; KATANFOROOSH, K. *Stanford University Course on Deep Learning*. Available from Internet: <http://cs231n.stanford.edu/>.

MARCHEGIANI, L.; POSNER, I. Leveraging the urban soundscape: Auditory perception for smart vehicles. In: *Proceedings - IEEE International Conference on Robotics and Automation*. Oxford: IEEE, 2017. p. 6547–6554. ISBN 9781509046331. ISSN 10504729. Available from Internet: <https://ieeexplore.ieee.org/document/7989774/>.

MARKOVIĆ, I.; PETROVIĆ, I. Speaker localization and tracking with a microphone array on a mobile robot using von Mises distribution and particle filtering. *Robotics and Autonomous Systems*, v. 58, n. 11, p. 1185–1196, 2010. ISSN 0921-8890. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0921889010001387>.

MCFEE, B.; HUMPHREY, E.; BELLO, J. A software framework for musical data augmentation. In: *16th International Society for Music Information Retrieval Conference*. [S.l.: s.n.], 2015. (ISMIR).

MCFEE, B. et al. *librosa/librosa: 0.6.3*. Zenodo, fev. 2019. Available from Internet: <https://doi.org/10.5281/zenodo.2564164>.

MESAROS, A. et al. {DCASE} 2017 Challenge Setup: Tasks, Datasets and Baseline System. *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop (DCASE2017)*, p. 85–92, 2017.

Acoustic event detection in real life recordings. [S.l.]: IEEE, 2010. 1267–1271 p.

MESAROS, A.; HEITTOLA, T.; VIRTANEN, T. TUT database for acoustic scene classification and sound event detection. In: *European Signal Processing Conference*. Budapest, Hungary: IEEE, 2016. v. 2016-Novem, p. 1128–1132. ISBN 9780992862657. ISSN 22195491.

MEUCCI, F. et al. A real-time siren detector to improve safety of guide in traffic environment. 01 2008.

MUN, S. et al. *Generative Adversarial Network Based Acoustic Scene Training Set Augmentation and Selection Using SVM Hyper-Plane*. [S.l.], September 2017.

NG, A.; DROR, R. *Stanford University Course on Machine Learning*. 2018. Available from Internet: <http://cs231n.stanford.edu/>.

NG, A.; LI, F.-F.; KARPATHY, A. *Stanford University Course on Convolutional Neural Networks for Visual Recognition*. Available from Internet: <http://cs231n.stanford.edu/>.

NIELSEN, M. *Neural Networks and Deep Learning*. [S.l.]: Determination Press, 2015.

NILSSON, N. J. *Introduction to Machine Learning*. Nov 1998.

OPPENHEIM, A. V.; WILLSKY, A. S.; NAWAB, S. H. *Signals & Systems (2nd Ed.)*. USA: Prentice-Hall, Inc., 1996. ISBN 0138147574.

O'Shaughnessy, D. Automatic speech recognition. In: *2015 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*. [S.l.: s.n.], 2015. p. 417–424.

PATTANAYAK, S. *Pro Deep Learning with TensorFlow*. [S.l.]: Apress Media, 2017. ISBN 978-1-4842-3095-4.

PFÄFFLE, J. *Right on track to enhanced driving safety*. [S.l.]: Robert Bosch GmbH, mar. 2006.

RAMSUNDAR, B.; ZADEH, R. B. *TensorFlow For Deep Learning*. First. 1005 Gravenstein Highway North, Sebastopol, CA 95472.: O'Reilly Media Inc., 2018.

RONG, F. Audio Classification Method Based on Machine Learning. In: . [S.l.: s.n.], 2016. p. 81–84.

SAE. *Report J3016 - Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. [S.l.]: SAE International, jun 2018.

Sigtia, S. et al. Automatic Environmental Sound Recognition: Performance Versus Computational Cost. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, v. 24, n. 11, p. 2096–2107, Nov 2016. ISSN 2329-9290.

SIMONYAN, K.; ZISSERMAN, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. [s.n.], 2015. Available from Internet: <http://arxiv.org/abs/1409.1556>.

Slizovskaia, O. et al. End-to-end Sound Source Separation Conditioned on Instrument Labels. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2019. p. 306–310. ISSN 2379-190X.

STATISTISCHESBUNDESAMT. *Fachserie 8, Reihe 7, Verkehr, Verkehrsunfälle 2017.* [S.l.], 2017.

STOWELL, D. et al. Detection and Classification of Acoustic Scenes and Events. *IEEE Transactions on Multimedia*, IEEE, v. 17, n. 10, p. 1733–1746, 2015. ISSN 15209210.

TADJINE, H. et al. Acoustic/lidar sensor fusion for car tracking in city traffic scenarios. p. pp 67–72, 2015. Available from Internet: <http://publications.lib.chalmers.se/records/fulltext/222422/local_222422.pdf https://trid.trb.org/view/1412204>.

THORSLUND, B. *Effects of hearing loss on traffic safety and mobility*. PhD Thesis (PhD), 2014.

TZANETAKIS, G.; COOK, P. Musical Genre Classification of Audio Signals. *IEEE Transactions on Speech and Audio Processing*, v. 10, p. 293–302, 08 2002.

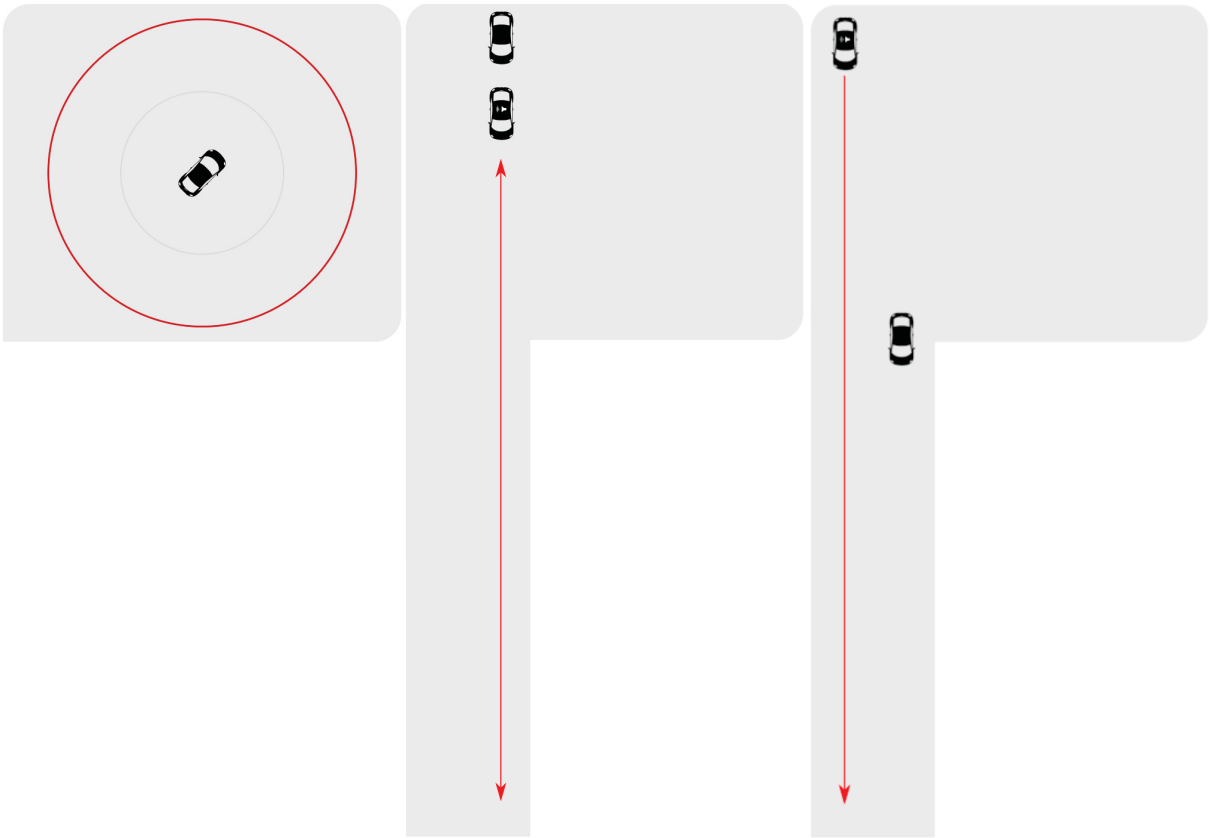
Weinstein, E.; Feder, M.; Oppenheim, A. V. Multi-channel signal separation by decorrelation. *IEEE Transactions on Speech and Audio Processing*, v. 1, n. 4, p. 405–413, 1993.

WIKIPEDIA. *Ein Rettungswagen befährt eine Rettungsgasse auf der BAB 659*. out. 2005. `https://de.wikipedia.org/wiki/Rettungsgasse`. Accessed: 2019-08-16.

ZHENG, A. *Evaluating Machine Learning Models*. [S.l.]: O'Reilly Media, Inc., 2015. ISBN 9781492048756.

ANNEX A – Visual guideline of CARISSMA's second test cases

Figure 52: Drawing representing the test carried out in scenarios: 17 and 18 (left); 19 (center); 27, 28, 29, and 30 (right).



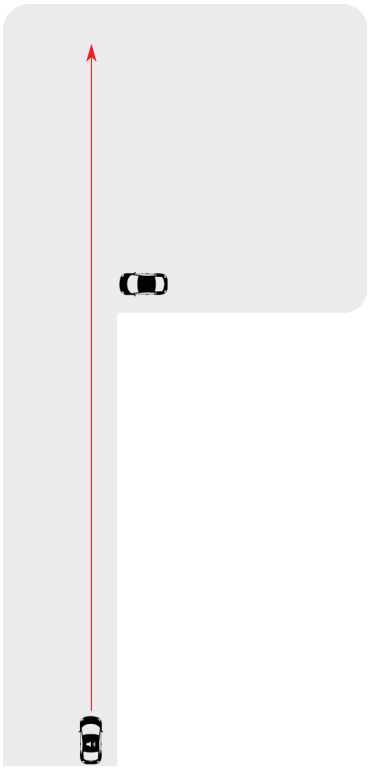


Figure 53: Drawing representing the test carried out in scenario 33 and 34.

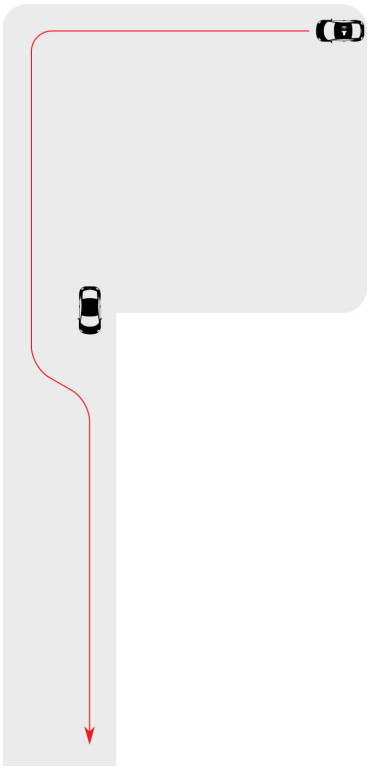


Figure 54: Drawing representing the test carried out in scenario 35 and 36.

Figure 55: Siren toggling tests — 2 (left), 12 (center), 13 (right).

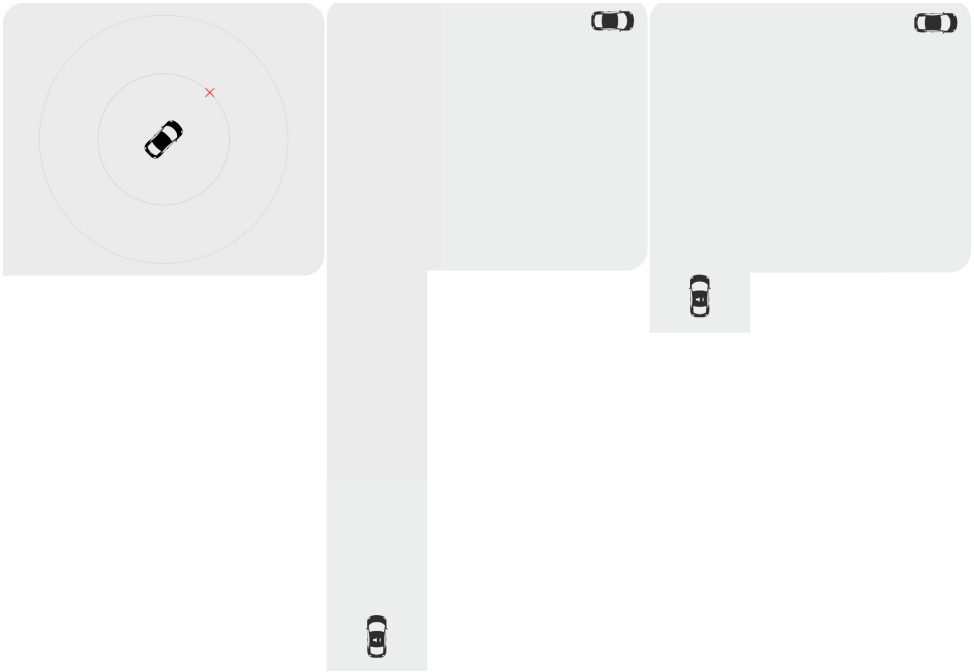


Figure 56: Target pass-by longitudinal tests — 14 (left) and 15 (right).

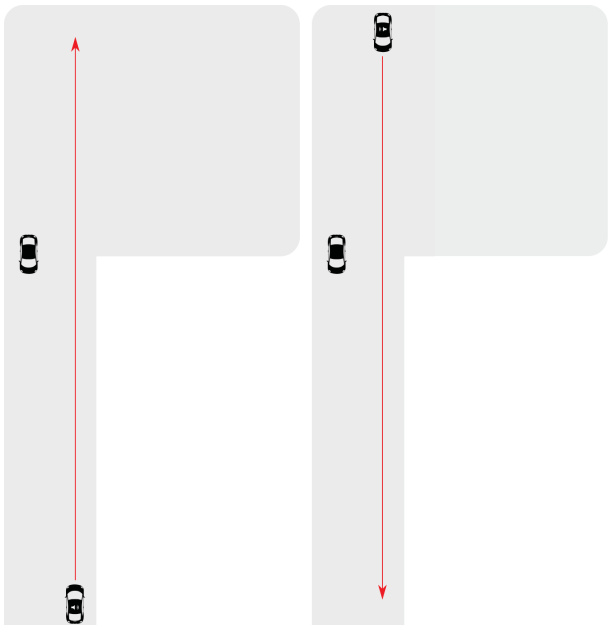


Figure 57: Target straight line drive away and back test — 16 (left) & Target pass-by transversal tests — 17 (center) and 18 (right).

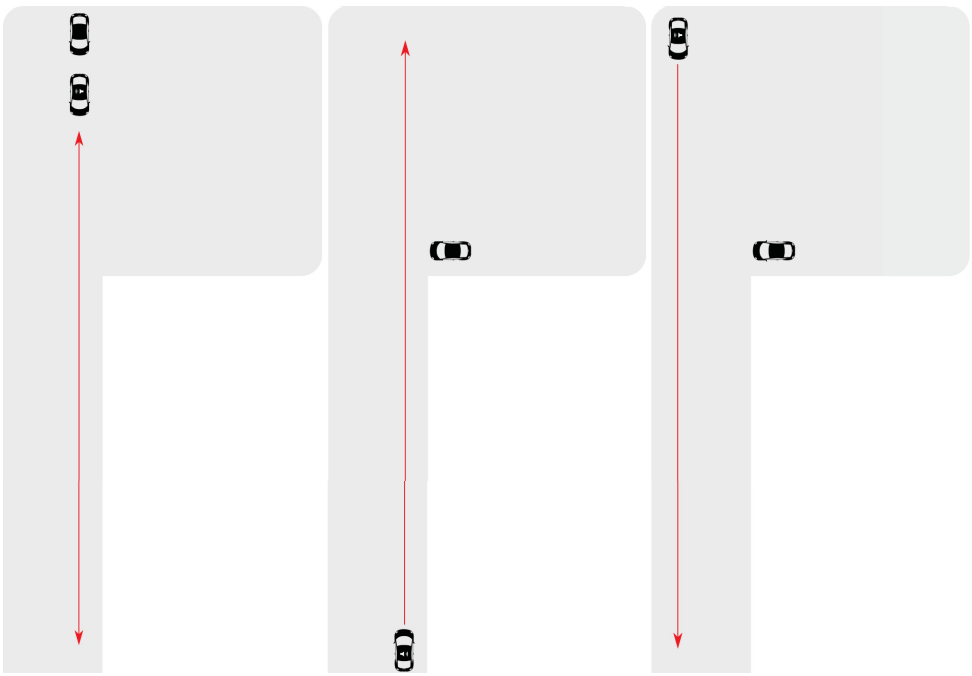


Figure 58: Target takeover tests — 19 (left) and 20 (center) & use case scenario test — 21 (right).