Development of Turbocharger Speed Recognition Application via Sound Classification using Neural Networks

Loukas Arvanitis

Diploma Thesis

School of Naval Architecture and Marine Engineering
National Technical University of Athens

Supervisor: Assistant Prof. G. Papalambrou

Committee Members:

Prof. L. Kaiktsis
Associate Prof. C. Papadopoulos

November 2021

This page is intentionally left blank.

Acknowledgements

This thesis concludes my years of studies at the School of Naval Architecture and Marine Engineering of the National Technical University of Athens. It has been, without a doubt, a great academic journey, where I gained great knowledge in multiple engineering fields and broadened my scientific and academic horizons.

A great opportunity was presented to me through this thesis project. Machine Learning and Computer Programming were two fields that I haven't had the chance to work with. It is of great importance for each and everyone of us, to challenge ourselves daily, overcome obstacles and strive for progress. Throughout this year, I had the chance to familiarize myself with a new programming language and develop several machine learning skills, which I am sure will a valuable asset for my feature engineering career.

I owe my sincere gratitude to my thesis advisor, As. Professor George Papalambrou. A friend, a mentor, providing me in every phase of this project with tireless advice and guidance. I would like to thank him for giving me the opportunity to work on such an interesting project and I wish him all the best on his academic career.

Finally, I am also sincerely grateful to my parents Kostas and Popi, my sister Sophia and my close people, family and friends for their unceasing support, patience and encouragement throughout my studies. Last but not at least, I would like to thank Polyxeni, as I feel extremely grateful for having her beside me, for believing in me and supporting me through this process. This thesis is dedicated to all of you. Thank you!

Abstract

This thesis investigates the implementation of an audio classification application with the use of neural networks to monitor the changes in the ICE's turbocharger speed in the HIPPO-2 testbed in LME. Our study's goal is to provide a monitoring system cheaper, but as robust as the traditional solutions applied in such cases without jeopardizing the structural integrity of the turbocharger.

The web application created would be responsible for the detection of changes in the T/C speed, by analysing the sound produced by it. Two plots are presented, one depicting a visual implementation of the Fast Fourier Transformation and one presenting a Spectrogram, depicting the sound that the algorithm is trying to classify. Machine learning techniques were implemented in order to create a fast and efficient model, able to perform this tasks simply in a browser tab.

For the creation of the model, Google's Teachable Machine module was used. After taking recordings from different locations around the experimental facility, for five different torque demands, we created five different labels in the module, each representing the speeds that the T/C achieved. An estimation of the speeds, was provided by a thermodynamic model of the HIPPO-2 testbed in order to assist us in the classification process.

Furthermore, we examined 4 different combinations of number of epochs and audio samples inputted at each category, as it was the only available option provided by the Teachable Machine module, in order to achieve the best performance possible. After the completion of training for each of the above mentioned cases, the model was inputted in our web application created for our research. Snapshots from the produced results are presented in order to help the reader get a better understanding on how the application works. A conclusion on which is the best performing model is presented, along with a few suggestions on how we could enhance the application features and ways to improve the dataset.

Περίληψη

Στην παρούσα διπλωματική εργασία διερευνήθηκε η ανάπτυξη μιας εφαρμογής Αναγνώρισης Ήχων, μέσω της χρήσης Νευρονικών Δικτύων, ώστε να ελέγχονται οι αλλαγές στις ταχύτητες που αναπτύσσει ο στροβιλοϋπερπληρωτής στην κλίνη δοκιμών HIPPO-2 του εργαστηρίου Ναυτικής Μηχανολογίας. Βασικός στόχος της έρευνας αποτελεί η εύρεση ενός εναλλακτικού συστήματος που είναι ταυτόχρονα οικονομικό αλλά και εξίσου αξιόπιστο με τα παραδοσιακά συστήματα παρακολούθησης, χωρίς να διακινδυνεύει η δομική ακεραιότητα του στροβιλοϋπερπληρωτή.

Η διαδικτυακή εφαρμογή που δημιουργήθηκε, έχει ως στόχο την αναγνώριση των διαφορετικών στροφών που αναπτύσσονται στον στροβιλοϋπερπληρωτή, μέσω ανάλυσης των διαφορετικών παραγόμενων ήχων. Στην εφαρμογή συναντώνται δύο γραφήματα, ένα που παρουσιάζει τον Ταχύ Μετασχηματισμό Φουριέ και ένα Φασματογράφημα, του προς ανάλυση ηχητικού αποσπάσματος που προσπαθεί να αναγνωρίσει ο αλγόριθμος. Με χρήση τεχνικών Μηχανικής Μάθησης, πραγματοποείται μια προσπάθεια δημιουργίας ενός γρήγορου και αποτελεσματικού μοντέλου, ικανού να εκτελέσει όλες τις παραπάνω λειτουργίες μόλις σε ένα παράθυρο φυλομετρητή.

Για την δημιουργία του μοντέλου χρησιμοποιήθηκε η εφαρμογή Teachable Machine της Google. Αφού πραγματοποιήθηκαν ηχογραφήσεις σε διάφορες θέσεις στην κλίνη δοκιμών του εργαστηρίου, για τις πέντε διαφορετικές απαιτήσεις ροπής που τέθηκαν, δημιουργήθηκαν πέντε κατηγορίες εντός της εφαρμογής, μια για κάθε διαφορετική ταχύτητα στροφων στροβιλοϋπερπληρωτή που επιτεύχθηκε. Μια αρχική εκτίμηση των στροφών δόθηκε μέσω θερμοδυναμικού μοντέλου που εφαρμόστηκε στην πειραματική κλίνη, για να μας βοηθήσει να κατηγοριοποίησουμε τα διάφορα ηχητικά αποσπάσματα που προέκυπταν.

Επιπρόσθετα, ελέχθηκαν τέσσερις διαφορετικοί συνδιασμοί αριθμού κύκλων εκπαίδευσης (εποχές ή epochs) και αριθμού ηχητικών αποσπασμάτων που τροφοδοτούνται σε κάθε κατηγορία, καθώς η πλατφόρμα της Google δεν παρείχε άλλες δυνατότητες παραμετροποίησης του παραγόμενου μοντέλου. Με την ολοκλήρωση της εκπαίδευσης της εκάστοτε κατηγορίας, τα εξαγόμενα μοντέλα εισήχθησαν στην διαδικτυακή εφαρμογή που δημιουργήθηκε στα πλαίσια της έρευνας. Εξήχθησαν στιγμιότυπα με τα παραγόμενα αποτελέσματα από τις διάφορες φάσεις εκτέλεσης για την καλύτερη κατανόηση της λειτουργίας της εφαρμογής από πλευράς αναγνώστη. Τέλος παρατέθηκαν συμπεράσματα ως προς την ακρίβεια των παραγόμενων μοντέλων, μαζί με μερικές προτάσεις για περαιτέρω βελτίωση της εφαρμογής και την δημιουργία καλύτερης βάσης δεδομένων.

Abbreviations

AI Artificial Intelligence.

CNN Convolutional Neural Network.

CONV Convolutional Layer.

CV Computer Vision.

DFT Discrete Fourier Transform.

DL Deep Learning.

EB Electric Brake.

EM Electric Motor.

FC Fully-Connected Layer.

FFT Fast Fourier Transformation.

ICE Inter Combustion Engine.

LME Laboratory of Marine Engineering.

ML Machine Learning.

NN Neural Network.

OS Operating System.

POOL Pooling Layer.

ReLU Rectified Linear Unit.

RGB Red Green Blue.

SGD Stochastic Gradient Descent.

T/C Turbocharger.

TM Teachable Machine.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter describes briefly the scope of the investigation. It also provides useful information about the HIPPO II testbed, in Laboratory of Marine Engineering at the School of Naval Architecture and Marine Engineering, of the National Technical University of Athens (NTUA), stating the absence of a monitoring system, that would provide the user with the necessary information regarding the changes in the Internal Combustion Engine's turbocharger speed. It was our belief that the right fit for the experimental facility, would be the creation an audio classification model by implementing machine learning techniques, providing an accurate, reliable and maintenance free tool, that would classify the different sounds produced from the engine's T/C while at various speeds. The advantages of our selected method are described briefly in the thesis outline.

1.1 Problem Description and Motivation

The main objective of this thesis is development of an easy to use, cheap but reliable monitoring system able to detect the changes in the ICE's T/C speed in the experimental facility at LME.

The conduction of periodic operational checks of the technical condition of an ICE's turbocharger in operation, is of great necessity. As a common practice, rotational speed sensors are used to monitor the T/C speed. The working principal of the rotational sensors, is the transformation of mechanical rotary positions into electric signals that are then inputted in the engine's control unit.

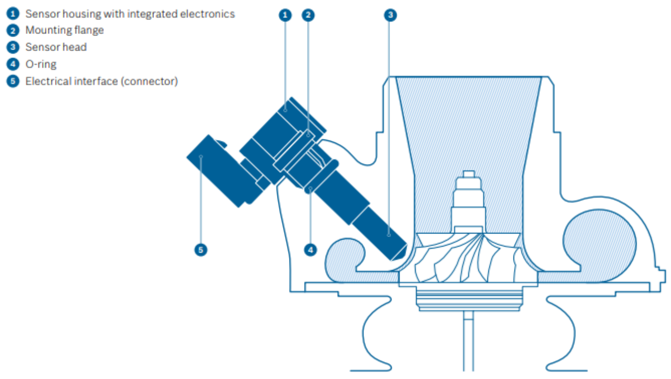

1 Sensor housing with integrated electronics
2 Mounting flange
3 Sensor head
4 O-ring
5 Electrical interface (connector)
1
2
3
4
5

Figure 1.1: Cross section from a rotational speed sensor.

Unfortunately, during the installation of the HIPPO-2 powertrain, the ICE had no such equipment preinstalled. As we can see in Figure 1.1, in order to fit the T/C with an aftermarket solution, the structural integrity of the T/C could be severely affected, as we would need to penetrate the housing in order to fit the sensor.

A 2021 research named *Acoustic Method for Estimation of Marine Low-Speed Engine Turbocharger Parameters* describes a diagnostic method on the determination of the blade harmonics in the turbocharger, while also calculating the main rotor speed by analyzing the produced acoustic signals.[17] It was very interesting to see that machine learning techniques were used to classify environmental sounds, in order to track and monitor the behaviour of various species. A research named *Bat detective—Deep learning tools for bat acoustic signal detection*, proposed that state-of-the-art deep learning techniques can provide really accurate data for automatic detection and monitoring of bat population, by analysing the various sounds that bats produced while travelling, mating etc. All the above lead us in the conclusion that it would be a perfect fit for our problem, to devise a method able to estimate T/C speeds, by analysing the sounds produced by it at various speeds by implementing deep learning techniques.

1.2 Hybrid Integrated Propulsion Powertrain- HIPPO II

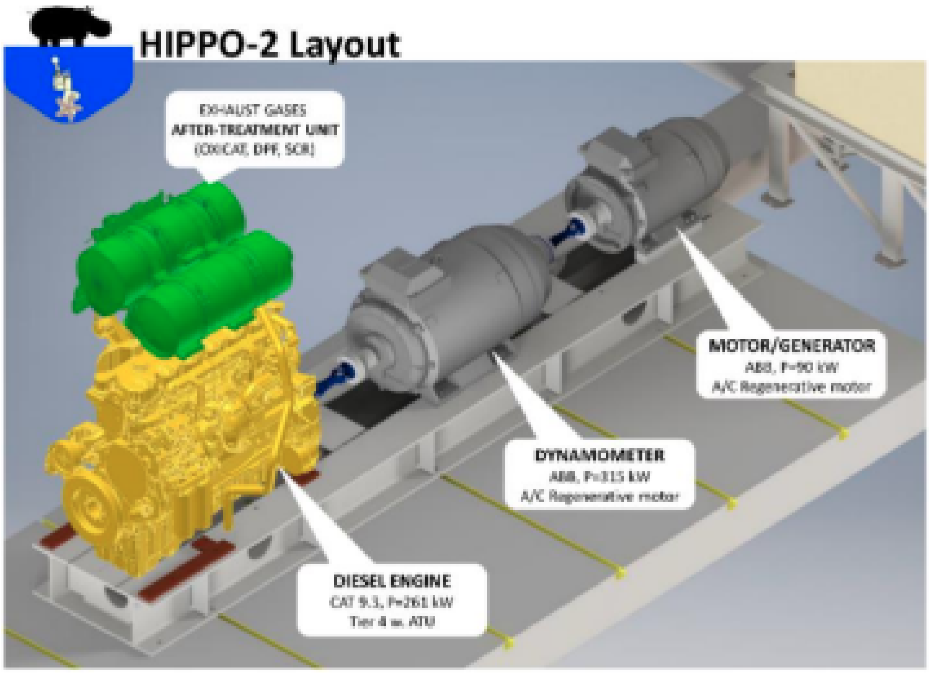

HIPPO-2 Layout
EXHAUST GASES AFTER-TREATMENT UNIT (OXICAT, DPF, SCR)
MOTOR/GENERATOR ABB, P=90 kW A/C Regenerative motor
DYNAMOMETER ABB, P=315 kW A/C Regenerative motor
DIESEL ENGINE CAT 9.3, P=261 kW Tier 4 w. ATU

Figure 1.2: 3-D representation of the units of the hybrid testbed placed on the bedplate.

The HIPPO-2 hybrid diesel-electric power plan, comprises of an internal combustion engine (ICE) in a serial connection to an electric motor (EM). In such configuration as presented in Figure 1.2, the rotational speed of the ICE and EM are identical and the supplied torques add together in order to maintain the total torque demand applied by an electric motor brake (EB).[18] Based on the large experience gained from it's predecessor the HIPPO-1 testbed, this next generation testbed comprises of some state-of-art components aiming for increased flexibility in operation and more accurate control and measurement capability. A list of it's main components is presented below,

- ICE: CATERPILLAR® 6-cylinder 9.3 liter 4-stroke industrial diesel engine, model C9.3, producing 261 kW at 1800-2000 rpm, with max torque 1596 Nm at 1400 rpm.
- EM: ABB®, AC induction 3-phase motor with rated power of 90 kW at 1483 rpm.
- EB: ABB®, AC induction 3-phase motor with 315 kW load capacity at 1488 rpm .

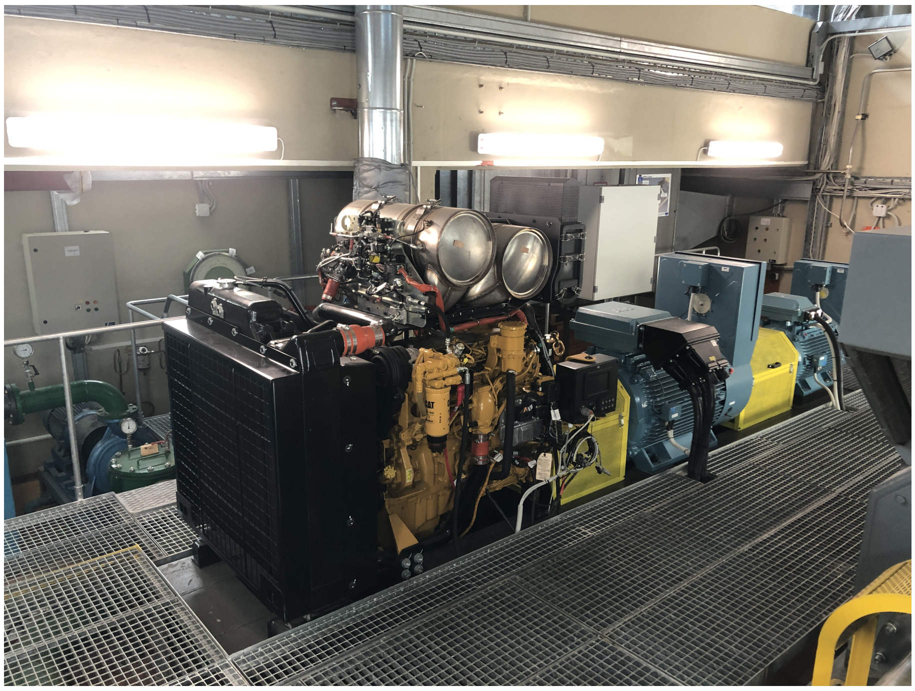


Figure 1.3: Placement of all units on the common bedplate.

The monitoring and control room is overviewing the installation as shown in Figure 1.4. The monitoring and high-level control of thermal engine and electric motors of the HIPPO-2 testbed are implemented based on the dSPACE MicroAutobox II platform, in real time operation with MatLab/Simulink, using industrial Ethernet and CAN-bus networks.

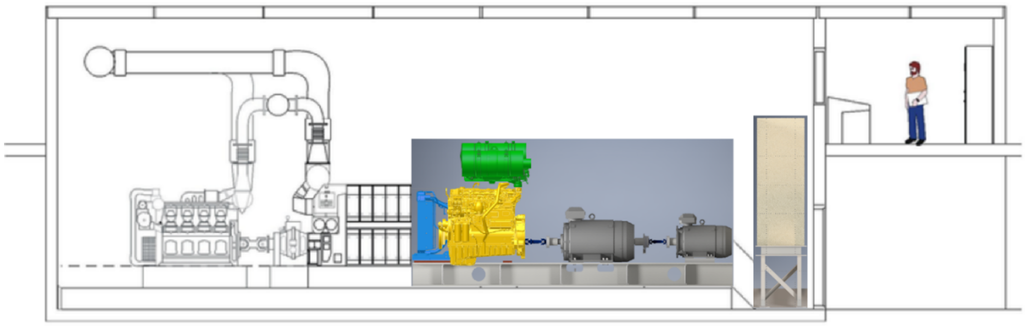


Figure 1.4: Location of the HIPPO-2 testbed in the existing experimental facility at the LME.

1.3 Contributions and Engineering Challenges

According to the predefined project goal, the contributions and Engineering Challenges of this thesis are presented in the following list:

- Choose the most convenient tools for developing the application.
- Perform a comparison experiment and create an efficient and fast audio classification model.
- Develop a user friendly application that performs audio detection on the audio samples captured from the ICE's T/C inside the experimental facility.
- Test the limits of the selected model.
- Train the new model.

1.4 Thesis Outline

- **Chapter 2** introduces the basic principles of artificial intelligence, machine learning, deep learning and computer vision, analyzes the NN and CNN architectures and explains the model training phase.
- **Chapter 3** lists the tools used for the development of this thesis and presents the basic module used to develop our neural network. A brief analysis of the tools depicted in our web application is provided to the reader for a better understanding
- **Chapter 4** lists the several training cases performed, while providing snapshots taken from the web application user interface were prediction scores achieved are depicted. It also presents plots recording the loss of each model leading ultimately in important conclusion.
- **Chapter 5** presents the concluding thoughts about this thesis's targets based on the extracted results and suggests future work for the next steps of the project.

Chapter 2

Theoretical Background

This chapter contains the theoretical background this thesis is based upon. It's main purpose would be to familiarize the reader with the tools and methods implemented throughout this project. The listed citations in the last chapter, can provide a better insight to the concepts presented in this study. In detail this chapter introduces the basic principles of artificial intelligence, machine learning, deep learning, speech recognition and sound classification, analyzes the NN and CNN architectures and explains the model training procedure.

2.1 The AI Roadmap

Oftentimes, the terms machine learning, artificial intelligence and neural networks (NN) are used interchangeably. In fact, machine learning is actually a branch of AI whereas neural networks consist of an assortment of algorithms used in machine learning for data modelling using graphs of neurons.In this section we will explain briefly the above terms, as well as, analyse the Neural Network and Convolutional Neural Network structure.

2.1.1 Artificial Intelligence

Artificial Intelligence (AI) is often perceived as an extremely complicated term that is only related to computers and robots, but nothing could be further from the truth. As an umbrella term it describes a concept in machines that are able to be intelligent and complete "smart" tasks, those that were originally thought to require human intelligence. As the field matured from its beginning in the 1950s thanks to our own understanding of how the brain works and the growth of technology, computers began to mimic human decision-making processes. Perhaps the easiest way to understand about artificial intelligence, machine learning, neural networks, and deep learning is to think of them like Russian nesting dolls as shown in Fig. 2.1. Each is essentially a component of the prior term[1].

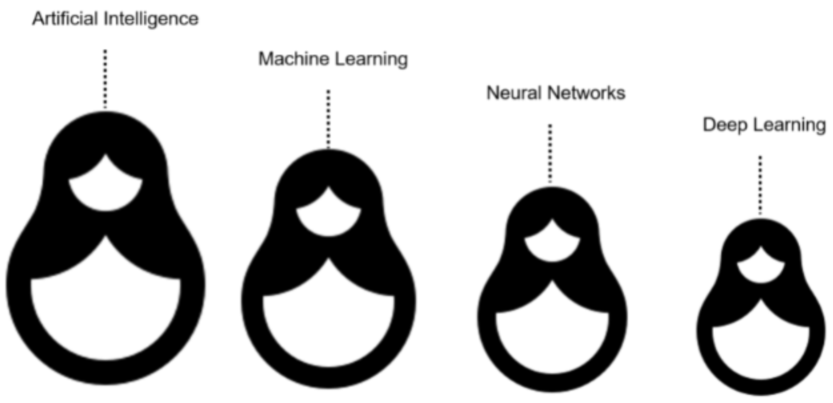


Figure 2.1: Relations between AI, machine learning, neural networks, and deep learning.[1]

Regarding the last two, it is the number of hidden layers, or depth of neural networks that distinguishes a single neural network from a deep learning algorithm, where the former usually comprises of no more than three layers, while the later can have from five up to 150.

2.1.2 Machine Learning

Machine Learning (ML) is a sub-field of AI. It is based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention. It enables us to tackle tasks that are too difficult to solve with fixed programs written and designed by human beings helping us overcome the difficulties faced by systems created with the classical programming philosophy.[19] The main differences between classical programming and machine learning are schematically depicted in Fig. 2.2

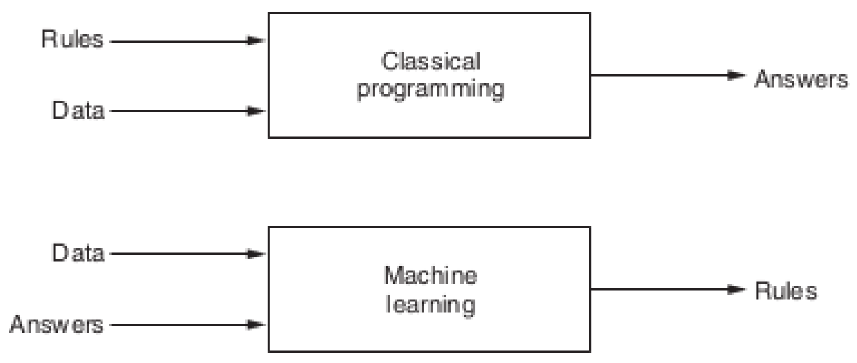


Figure 2.2: Comparison between the classical programming and machine learning paradigm.[2]

In order to acquire a better understanding of the machine learning concept, a brief analysis of the steps involved is presented. We can distinguish two main phases: the *training* and *inference phase* as shown in Fig. 2.3. The first one, processes data and answers together reffered to as *training data*. Each pair of data with the expected answers is called an example. Through the use of the examples, the training process produces the discovered rules. An appropriate blueprint provided by human engineers at the beggining of the training, is actually assisting and directing the machine learning algorithms in their rule generating task. This blueprint is wrapped in the concept of a *model* which forms a *hypothesis space* for the rules the machine may possibly learn.[2] The results of the training process are called *labels* and they are used as reference point for the machine learning algorithm to calculate and gradually reduce the error in a model's output. The above mentioned method describes a style of Machine Learning called *supervised learning* which is the one used in this thesis. Besides that, there are other types of ML not implemented in this thesis such as, *unsupervised learning*, *reinforcement learning*, *self-supervised learning* etc.

Once the trained model is ready, the learned rules are applied on new data, never before presented during the training process. In more detail when we are referring to this new set of learnt rules, we are actually implying that the above methods contributed in finding a way to effectively transform data into another form, assisting us on solving the task at hand. This would be the second phase, known as *inference phase*.

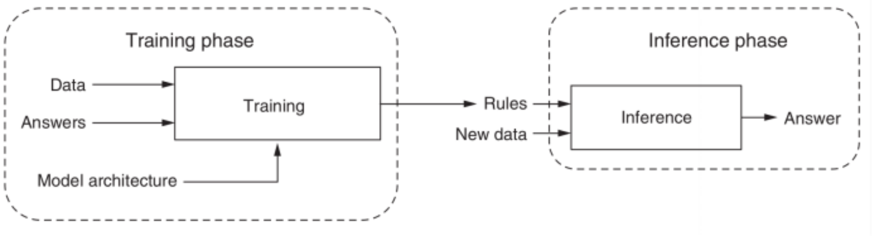


Figure 2.3: Detailed view of the machine learning paradigm. [2]

2.1.3 Neural Networks

Neural Networks (NN) are the heart of deep learing algorithms. They are a subset of machine learning. Their name and architecture derives from the human brain, mimicking the way that biological neurons signal to one another. Succesful applications of neural networks are classifications of handwritten digits, speech recognision, prediction of stock prices, medical applications etc. Our focus is on neural networks as efficient models for statistical pattern recognition. Our main concern will be directed to a specific class of neural networks, namely *multilayer perceptron (MLP)* in which the input units and output layer are interconnected with an intermediate hidden layer.[20]

2.1.4 Anatomy of Neural Networks

The basic building block of a neural network would be the *neuron* reffered also as *node* where all mathematical computations take place. One can compare the *layers* that where mentinoned above as a mathematical function, mappping an input value to an output value. However a distinguishing feature between a pure mathematical function and the NN layers, would be the fact that they are *stateful*, meaning that internal memory is held. A layer's memory is encapsulated by its weights. Simply, *weights* are a set of numerical learnable parameters helping with the transformation of input data within the network's hidden layers. Figure 2.4 depicts a simple neural network with three hidden layers. The input, hidden and output variables are presented by *nodes* and *weights* by links between the nodes. Arrows denote the direction of information flow through the network.[21]

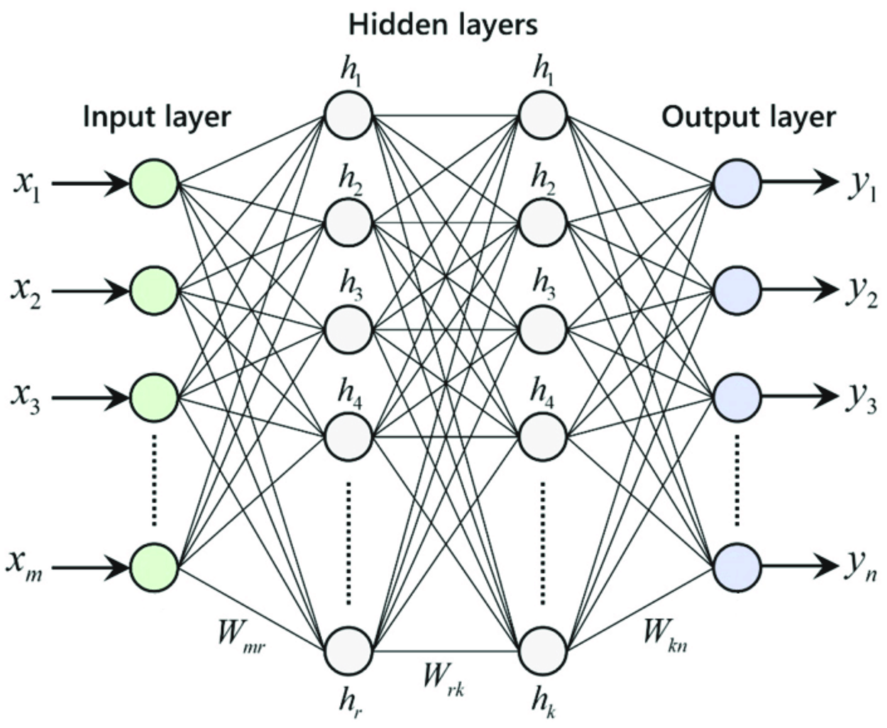


Figure 2.4: Portrayal of a neural networks nodes and weight parameters. [3]

Activation Function

Activation Functions are specially used in artificial neural networks to transform an input signal into an output signal which in turn is fed as input to the next layer in the stack.The accuracy of the prediction produced by our network is contingent upon various factors with some of the most important being the number of layers used and the type of the activation function used.[22] As shown in Fig. 2.5, the weighted sum of inputs is nothing more than a linear transformation. The linear value z is then passed through a non-linear function $f(z)$ which produces the input for the next layer. There is wide variety of activation functions, some of them will be briefly analysed in the following chapters.

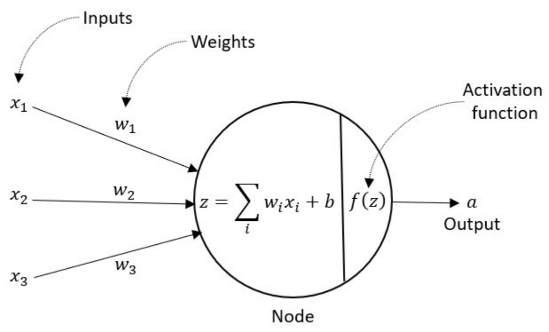


Figure 2.5: Representation of a neural network node.[4]

One of the most widely used activation functions in NN's is the *sigmoid function*. This function takes as input any real value and outputs values in the range between 0 and 1 as it's mathematical equation 2.1 suggests. As seen in Fig. 2.6, negative numbers become 0 and possitive numbers transform into 1. A major drawback of the sigmoid function would be the fact that sigmoids saturate and significantly minimize gradients. Moreover sigmoid outputs are not zero centered.[5]

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

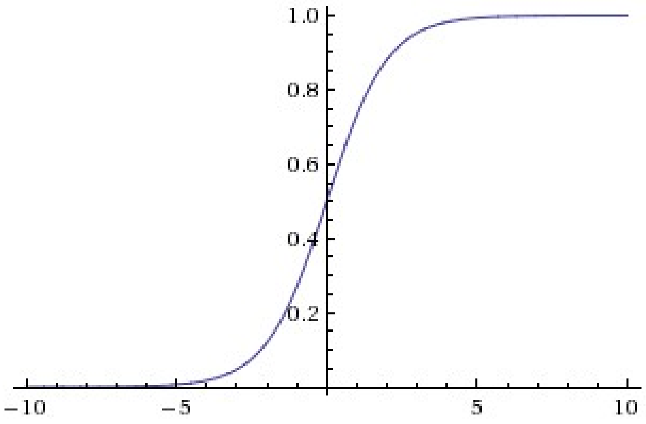


Figure 2.6: Sigmoid Activation Function.[5]

The rectified linear activation function or *ReLU* for short, solves the problem of saturation unfortunately only in the positive region meaning that all positive numbers stay unconverted and negative numbers are transformed to zeros. It is one of the most preferable activation functions due to its computational simpicity and the fact that it functions equivalently to a linear function. Knowing that a neural network is easier to optimize when its behaviour is linear or close to linear, this property of the activation function assists in almost completely avoiding the problem of vanishing gradients. The above implementation can be seen in the Figure 2.7.[23]

$$f(x) = max(0, x) \tag{2.2}$$

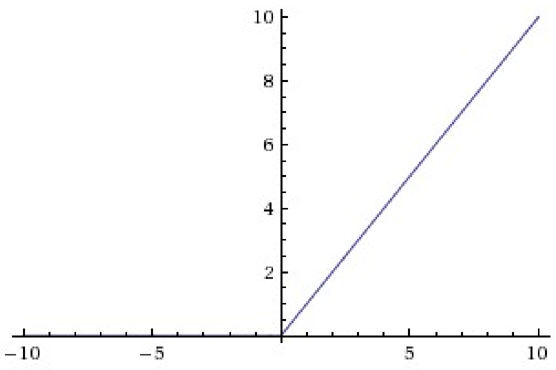


Figure 2.7: ReLU Activation Function.[5]

The ReLU does have some limitations. In case there are large weight updates, the summed input of the activation function would always be negative, disregarding the input resulting that the node will forever output an activation value of zero. This special case of ReLU function is reffered to as a *dying ReLU*.[23] Some popular extensions to the ReLU relax the non-linear output of the function to allow small negative values in some way. The *Leaky ReLU*(LReLU or LReL) modifies the function to allow small negative values when the input is less than zero meaning that this particular function computes:

$$f(x) = \mathbb{1}(x < 0)(\alpha x) + \mathbb{1}(x \geqslant 0)(x) \tag{2.3}$$

where α is a small constant.

In general this form of activation function is reported as rather successive, however the results are not always consistent.

The above mentioned functions presented in this thesis are obviously not the only ones. There are several others used by the machine learning community, such as *Tanh*, *Softmax*, *Swish* etc.

Loss Function

The nodes, assisted by the aforementioned activation functions aim in transforming the input throughout the layers to a desired output. It is of great importance to evaluate how close the produced output is to the expected. For this purpose we are introducing a rather crucial tool for the functionality of a neural network. The *loss function*, essentially computes how poorly our model is performing by comparing what the model is predicting with the actual value supposed to output. Usually the preferred loss function is defined by the objective of the model.

It is of great importance for the model to function correctly, the selection of an appropriate loss function. In every other case, if the calculated results do not express the properties of true values, the model will be trained on doing the wrong things. Fortunately, there are simple guidelines that can be followed.[24]

For *regression problems* where the model has to predict certain values, the loss function commonly used is *Mean Squared Error (MSE)*.[4]

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.4}$$

For *classification problems* where the model has to classify between two classes, the loss function commonly used is *Binary Cross-entropy*.[4]

$$BCE = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y})) \tag{2.5}$$

For *multi-class classification problems* where the model has to classify between more than two classes, the loss function commonly used is *Cross-entropy*.[4]

$$CE = -\sum_{i=1}^{k}\log(\hat{y}) \tag{2.6}$$

Optimizer

As mentioned in the previous paragraph, the loss function quantifies how well or poor the model is performing. The objective would be to minimize the loss, implying that the model is performing as needed. Generally optimization means, minimizing (or maximizing) any mathematical expression. That being said, *optimizers* are algorithms used to update the parameters of the network such as weghts and biases resulting on minimization of loss. In order to define the way that weight parameters are being updated, optimizer shall calculate the *gradient* (∇) of the loss function. It is clear that many iterations will be needed in order to evaluate the gradient and the performance of each and everyone of the updated parameters, not to mention that it would require a lot of time. There is a better mechanism, frequently used for machine learing namely the *gradient descent.*[25]. The main idea would be, talking repeated steps in the opposite direction of the gradient, in search for the direction of the steepest descent and ultimately to the global minimum of the loss function as shown in Fig. 2.8. Choosing the step size (usually refered to as *learning rate*) is one of the most important hyperparameter settings in the training process of a neural network.[5] Some of the most commonly preferred optimization algorithms are shortly presented in the following paragraphs.

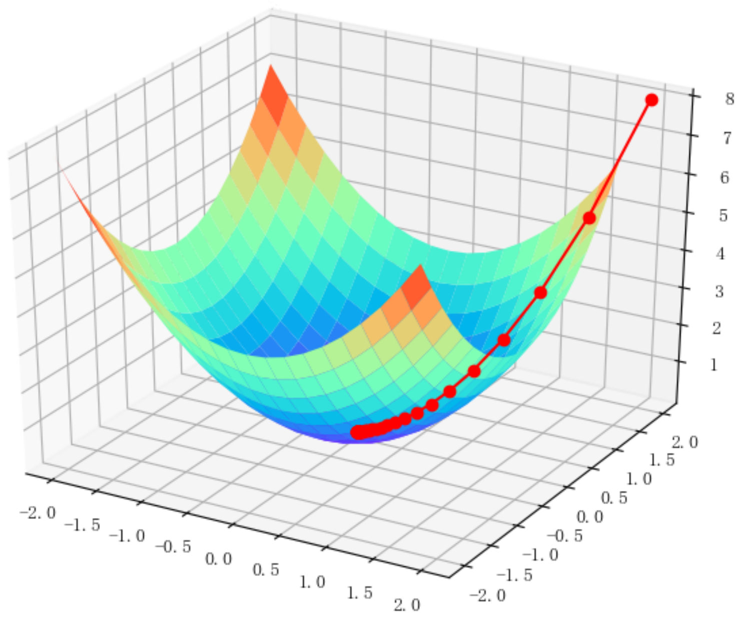


Figure 2.8: Convergence of the *gradient descent method.*[6]

To sum up, the goal of any machine learning algorithm would be to minimize loss function with optimum learning rate to lower error between actual and the predicted values.[25] Depending on the amount of data used for the gradient descent computation, we differentiate three variants: *Batch gradient descent*, *Stochastic gradient descent* and *Mini-batch gradient descent.*

Batch gradient descent computes the gradient of the loss function for the entire training dataset. In order to update one parameter, it would have to calculate the whole dataset as we can see in equation 2.7, resulting into really slow performance.[26]

$$\theta = \theta - \eta \nabla_{\theta} * J(\theta) \tag{2.7}$$

On the other hand, *Stochastic Gradient Descent (SGD)* performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$ overcoming the obstacle that batch gradient descent faces, resulting in really fast performed calculation. It should be stated that due to the frequent updates of the parameters, SGD can create cause the function to fluctuate heavily and with high variance.

$$w = w - \eta \nabla_w L(w; x^{(i)}; y^{(i)}) \tag{2.8}$$

On one hand, SGD's fluctuation enables it to find potentially a better local minima. On the other hand this results in a much more difficult convergence to the exact minimum as SGD will keep overshooting. However, it has been shown that when the learning rate is slowly decreased, SGD presents similar behaviour to the batch gradient descent.
Having addressed all the above problems, *Mini-batch gradient descent* takes the best of both worlds and as it name states, performs an update for every mini-batch of n training examples. This way, the variance of the parameters update is reduced leading to more stable convergence. The following equation 2.9 describes the mathematical form of this type of gradient descent

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{2.9}$$

There are a lot of gradient descent optimization algorithms that are widely used by the machine learning community to deal with the afformentioned challenges such as *ADAM, Momentum, Nesterov accelerated gradient, Adagrad, Adadelta, RMSprop, AdaMax, Nadam, AMSGrad* etc.[26]

Lastly, the schematic representation of the way a Neural Network operates is presented in Figure 2.9.

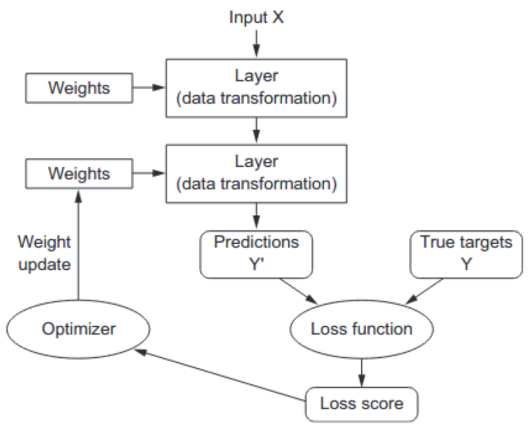


Figure 2.9: Neural Network model.[7]

2.1.5 Deep Learning

As mentioned before in this thesis, *Deep Learning* is subfield of machine learning concerned with algorithms processing vast amount of data. Deep learning algorithms are based on learning and improving on their own, mimicking the way human brain operates and gains certain type of knowledge. While machine learning uses simpler concepts, deep learning algorithms are stacked in a hierarchy of increasing complexity and abstraction.[19] To be more accurate, *deep neural networks* are neural networks with a notably number of layers each responsible for the completion of a certain task, as shown in Fig. 2.10. The actual number of layers in a deep neural network is referred to as the *model's depth*. While a neural network with a only a few layers can still make approximate predictions, the additional hidden layers found in the deep neural network models contribute in the models ability on learning, only be exposure to training data.

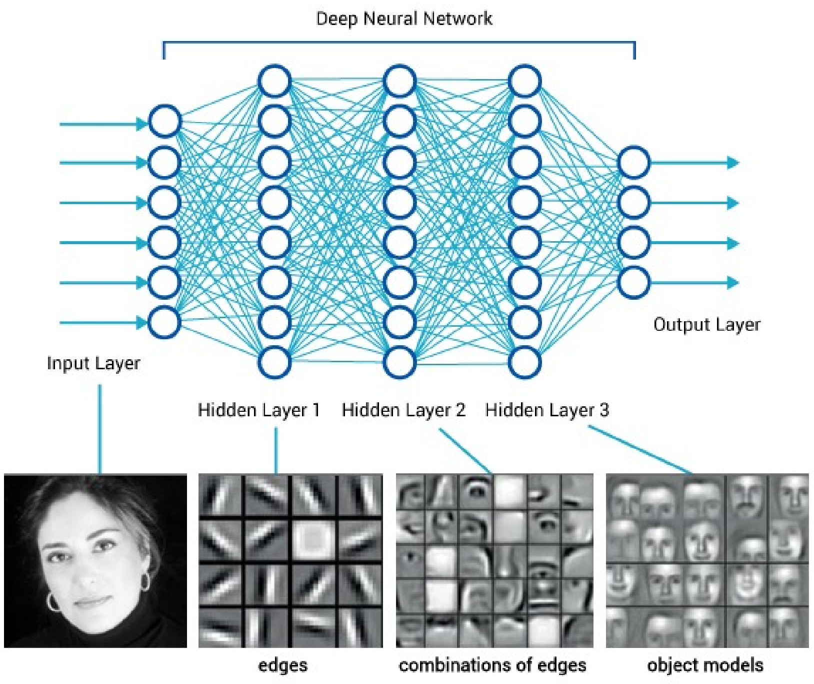


Figure 2.10: Deep Learning Model illustration.[8]

2.1.6 Sound Classification and Convolutional Neural Networks

Convolutional Neural Networks (CNN)

The main principles which *Convolutional Neural Networks* operate, are similar to the above mentioned neural networks. Actually a CNN is a multilayered neural network with a special architecture to detect complex features in data (Image recognition, audio classification etc.).[27] They comprise of nodes with learnable weights and biases. Every node receives inputs and performs a dot product and optionally follows it with non-linearity.[5] A loss function is being used to evaluate the performance of the network, triggering an optimizer in order for the weight parameters to be updated.

Having mentioned the above, it can be easilly stated that these two machine learning algorithms operate and behave the same way. The main difference between them, is the way the nodes are arranged throughout the layers. *Hidden layers* are layers between the input and output layer. In a typical NN, each hidden layer is connected to all nodes in the previous layer. Nodes in a single layer function completely independently and do not share any connections resulting in the creation of large numbers of weight parameters as the size of input data increases. As stated in section 2.1.6, sound can be interpreted as an image, which proves to be really helpful when working with CNN's. Contrary to a regular NN, the layers of a CNN have nodes arranged in three dimensions: *widht, height, depth.*[5] Figure 2.11 visualizes the above mentioned differences between the two types of architecture.

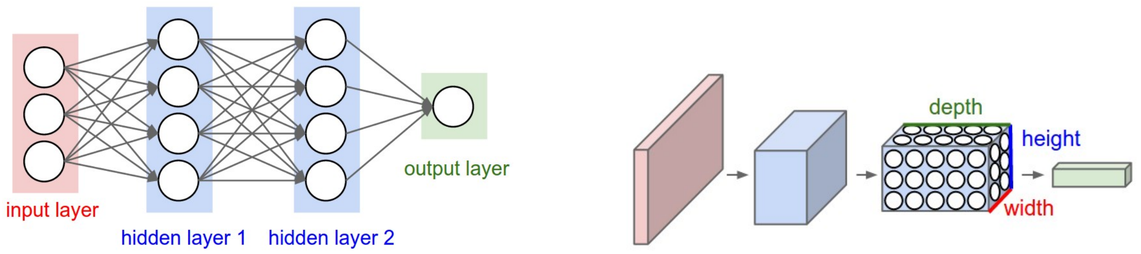


Figure 2.11: Main differences between regular NN (left) and CNN (right).[5]

Anatomy of a Convolutional Neural Network

By taking a closer look at the anatomy of CNN's, we can distinguish three main types of layers: *Convolutional Layers (CONV), Pooling Layers (POOL)* and *Fully-Connected Layers (FC).* All together stacked, form the *ConvNet* architecture.[5] A typical CNN performing an Audio Classification task is structured by:

- An *INPUT* layer that holds the raw pixel values of the spectrogram image (for more details see subsection 2.1.6), with shape [imageWidth, imageHeight, 3]. The number 3 represents the RGB (red, green, blue) color values of a pixel at given coordinates.
- A repeated pattern of *CONV, RELU, FLATTEN, DENSE* layers. *ReLU* layers (not depicted in Figure 2.12) apply the element-wise activation ReLU function described in the subsection 2.1.4.
- An *FC* layer. Nodes in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

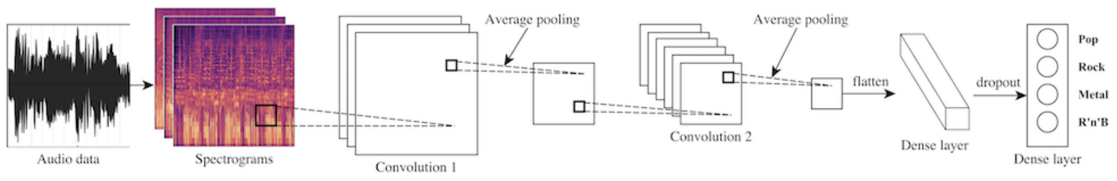


Figure 2.12: ConvNet architecture in an Audio Classification problem.[9]

As depicted in Figure 2.12, the *Convolutional Layer* is the first layer we come across in a CNN and is liable for the most demanding computational tasks. It comprises a set of learnable filters called *convolutional kernels*, or simply *kernels*. Every kernel is responsible for filtering through the full depth of the input layer as shown in Fig. 2.13. Simply to be put, kernel is a matrix that moves over the input data, performs the dot product with the sub-region of input data and gets the output as the matrix of dot products.

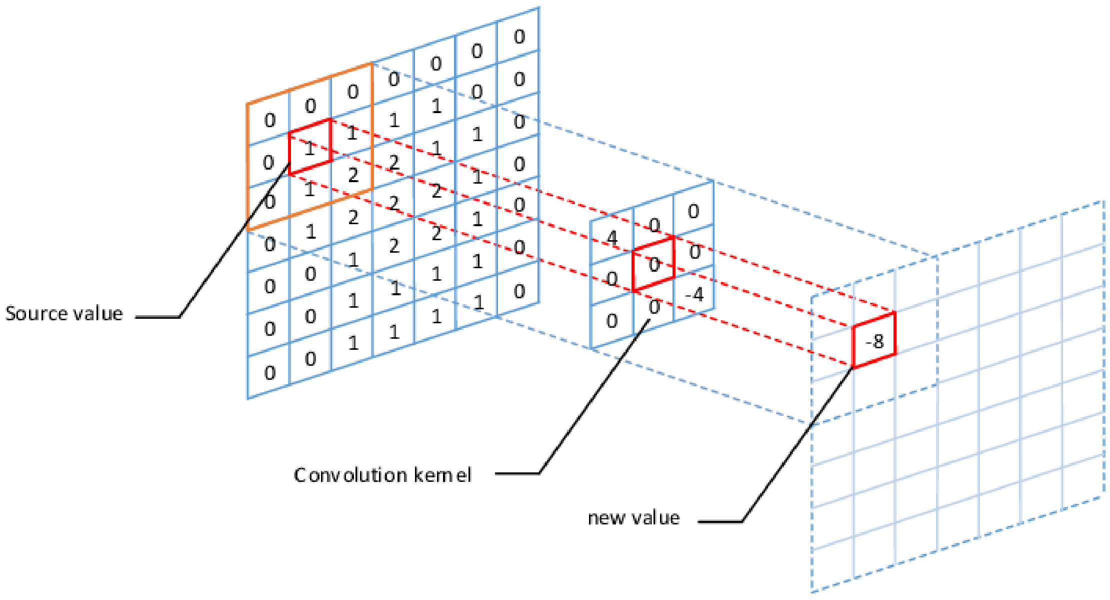


Figure 2.13: Convolution process of a one-dimensional kernel.[10]

For example a typical kernel size on a first layer of a ConvNet would be 5x5x3. The first digit stands for the width, the second for the height and the third presents the colour channel RGB short for Red, Green and Blue. As the kernel slides over the width and and height of the input volume, a 2-dimensional activation map is produced giving the response of that kernel at every position.[24] After going throughout the whole dataset, an entire set of activation maps will be created, enabling the network to intuitively learn which kernels activate resulting in the recognition of a specific pattern of a visual feature. Bundling all activation maps along the depth dimension, the output volume is produced.[5] In more detail, a convolution layer consists of:[2]

- *Kernel size*: e.g. 3x3, 5x5
- *Depth*: number of filters used, each learning to look for something different in the input. (Unlike the input image, the depth in the output tensor doesn't actually have to do with colors.)
- *Stride*: the size of the kernel's step.
- *Zero-padding*: the amount of zeros padded around the output volume.

As mentioned before, in audio classification problems the audio is depicted as a spectrogram (see paragraph 3.2.2 for more), a visual interpretation of the signal strength, helping CNN's to classify input data in a way similar to image classification. For a better understanding of a ConvNet's architecture Fig. 2.14 is presented. The input image, for the sake of simplicity, is of shape [width: 5, height: 3, color channels: 2]. The convolutional layer is defined by kernel size: 3x3, depth: 3, stride: [1, 1] and zero-padding: 0.

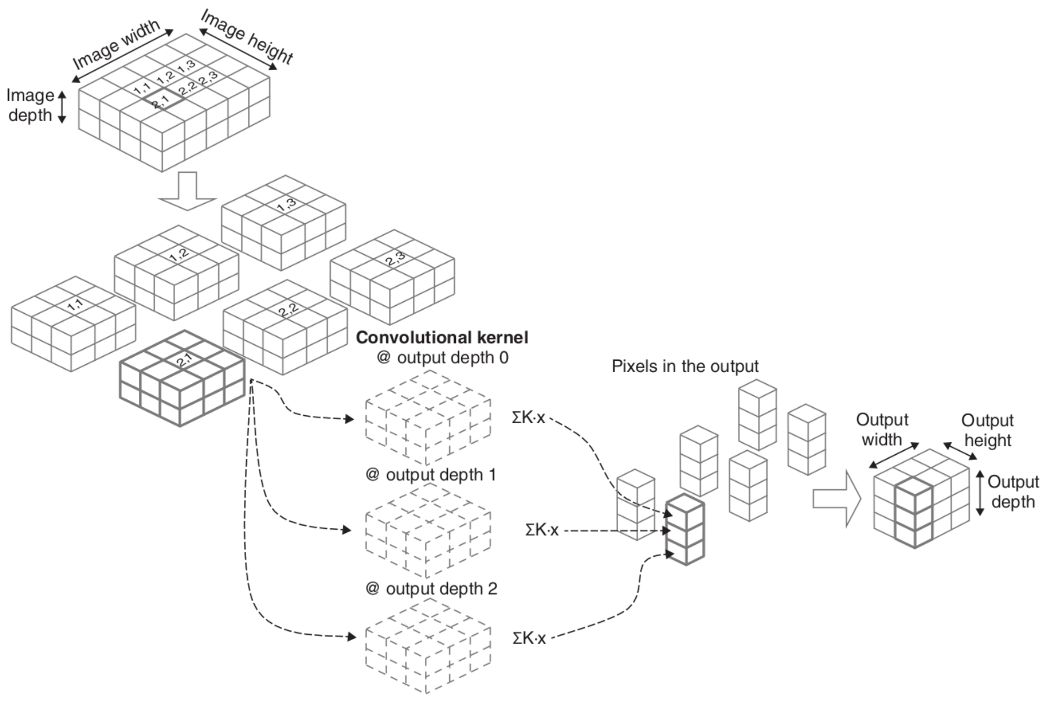


Figure 2.14: In depth view of a ConvNet.[2]

Another building block of a CNN is the *pooling layer*. Its function is to progressively reduce the amount of parameters (meaning the filters that where created in the convolution layer) and computation in the network. Pooling layer operates in each activation map separately. The most frequent approach used in pooling is *max pooling*. A pooling layer is also defined by:

- *Pool Size*
- *Stride*

In Fig. 2.15 we can see a Pooling layer performing max pooling operation with poolSize: 2x2 and stride: [2,2] [11]

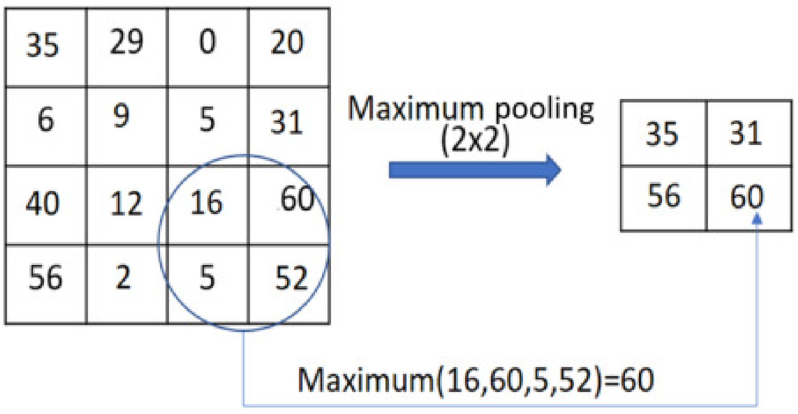


Figure 2.15: Pooling layer operation.[11]

Sound Classification tasks performed by ConvNets

Sound Classification is probably one of the most broadly used applications in Audio Deep Learning. The scope of the problem, would be to classify accordingly all the sounds presented and successfully predict the category of that sound. This type of problem can be applied to many different type of scenarios such as:

- *Music Clip Classification to identify the genre*
- *Speech Recognition*
- *Signal Processing*
- *Environmental Sound Classification*

Both image and audio classification were challenging tasks for a machine to do until Artificial Intelligence and neural networks came to the scene. At the beginning, *Image Classification Models* such as *EfficientNet, MobileNet, VGG* etc. where producing sufficient results and became more popular in the machine learning community. All the aforementioned ConvNets are able to perform computer vision tasks. As mentioned in this thesis, neural networks and machine learning tend to mimic and interpret in computer language, human perception. Surely, it there is not just vision but also audio. The question that rose was, if it were possible to used the so far established knowledge on ConvNets and Image processing, in audio classification.

As in any deep learning application, in order to understand how the model works we need to understand the data. In this particular case, a sound signal is produced by variations in air pressure. A microphone pick's up the changes and converts them into electrical signals which then are digitalized by a compute's sound card.[2]

Key layers of a convnet, exploit spatial relations in 2D spaces resulting in a compatibility problem, regarding the fact that sound data are presented in 1D arrays. It turns out, that sounds can be represented as special types of images called spectrograms. Spectrograms not only make it possible to apply convnets on sounds but also have theoretical justifications beyond deep learning. A spectrogram of a signal can be characterized as a "photograph" of it. They are produced using Fourier Transform (see subsection 3.2.2) to decompose any signal into its constituent frequencies.[2] They possess some rather interesting properties. Firstly, they save space, meaning the the number of float numbers in a spectrogram is usually a few times less than the number of float values in the raw waveform. They are corresponding to how hearing works in biology. A structure inside our ear called *cochlea* essentially performs the biological version of the Fourier transform, by decomposing different frequencies which are then picked up by different set of auditory neurons. Lastly, as shown in Fig. 2.16 the spectrogram representation makes speech sounds much different. In this thesis, spectrograms played a crucial role not only in helping the ConvNet perform all the above mentioned tasks, but also helped us understand the way the T/C performed throughout the sampling process.

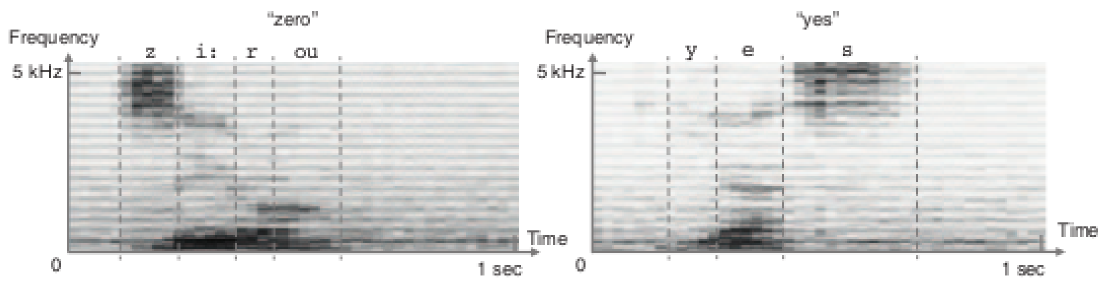


Figure 2.16: Spectrograms of the isolated spoken words ”zero” and ”yes”.[2]

2.2 Training Neural Networks

The accuracy of a neural network is increased when many examples are fed to it during the training phase. As mentioned before, the input data goes through the layers, producing an output. Then, the loss function is calculated and compared with the actual values. The results are then fed to the optimizer, so that the weights are parameterized in a way to minimize the loss function until the desired accuracy of the model is achieved. Some worth mentioning parameters of the models *training phase* are the following:

- *Input Data*: data fed into the model. In this thesis, input data is raw audio, depicted as an image-like state (spectrogram) compatible with the CNN architecture.
- *Labels*: desired output of the model for each input data, created by a time-consuming procedure called *labeling* [24]
- *Batch Size*: number of input processed data by the model. A bigger batch size is prefered to a smaller one in order to avoid gradient variation, resulting in increased weight updates of the model. It should also be taken into consideration that the larger the batch size the more memory would be required while training.[2]
- An *Epoch* describes the number of iterations performed on every batch
- *Validation Split* depicts the percentage of data used while evaluating the model, about 15% of the initial data. In order to avoid overfitting problems (see subsection 2.2.2), validation loss and accuracy should be closely monitored, while training.

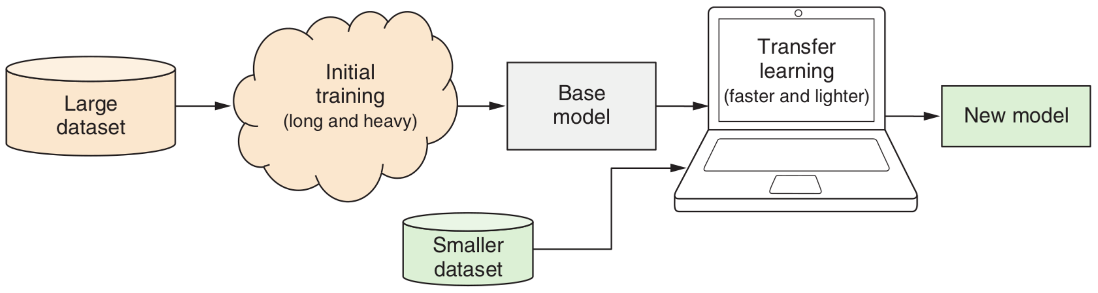

Large dataset
Initial training (long and heavy)
Base model
Transfer learning (faster and lighter)
New model
Smaller dataset

Figure 2.17: Schematic presentation Transfer learning.[2]

2.2.1 Transfer Learning

In Fig. 2.17, the workflow of a rather useful and time-saving technique used in machine learning called *Transfer Learning* is described. The better the training data, the better the performance of the model. Oftentimes, there are cases in which, the available data proposed for the model's training are inadequate and the task to be accomplished is rather complicating i.e object detection, sound classification etc. There are a few proposed ways to tackle this problem. Firstly, *data augmentation* is a technique where sound for example, is manipulated in such way that the new dataset created would result in better training results. Applying low-pass or high-pass filters would allow frequencies bellow (or above) a certain threshold to pass in our signal, resulting in the creation of spectrogram with less interference and finally a new dataset which would benefit the model's training phase. *De-noising* methods usually prove to be helpful be filtering out background noise resulting in spectrograms with less data thus creating a new dataset way more friendly during the training phase. An example of the above mentioned method is presented in Fig. 2.18.

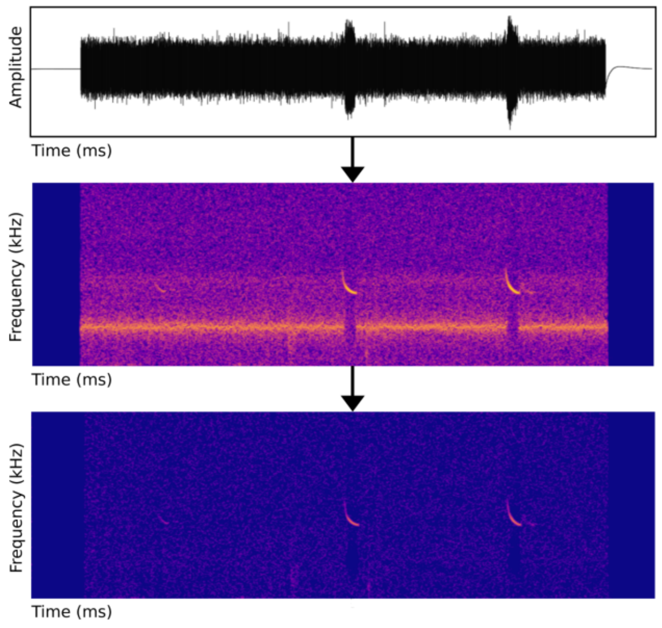


Figure 2.18: De-noising process depicted on spectrograms.[12]

Transfer Learning, could be defined as the reapplication of a pre-trained model used to speed up a new learning task by reusing the results of previous learning. This technique is proven to work along with data augmentation. The general idea would be to use the knowledge a model has gained from a previews training process to a different but at the same time related problem.[28] This design methology was also implemented in this particular thesis project by using Google's Techable Machine to perform a sound classification task. Mostly transfer learning is applied in computer vision and natural language processing tasks due to the huge amount of computational power required. The already trained model is reffered to as the *base model.* Transfer learning sometimes involves retraining the base model and sometimes involves creating a new model on top of the base model. We refer to the new model as the transfer model.[2] Some of the most common approaches for transfer learning are briefly analysed in the following paragraphs.

Freezing Layers

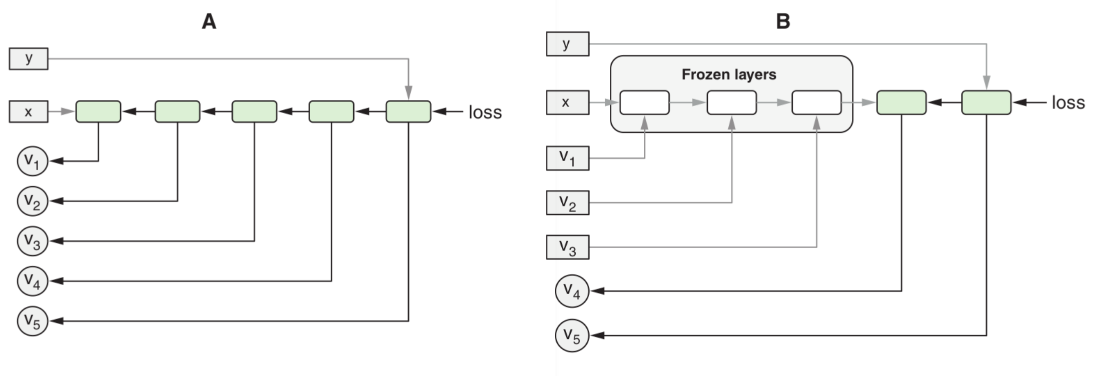


Figure 2.19: The Freezing Layers technique.[2]

There are plenty of ways to achive greaters speed in the training phase of the model. One technique assisting us accomplish the above is the *Freezing Layers* technique. Freezing a layer in context of neural networks is about controlling the update of the weights.[29] One way to implement this technique, would be to restrict the update of the weight parameters during training phase, resulting in a significant reduction of the time need for the model to train. It should be stated that before the new dataset is fed to the network, all layers except the last ones (known as *head* of the model), should "freeze". This technique is frequently used when the desired shape of the output is congruent with the base model.[2] All the above, are explained in Figure 2.19.

Two Models

Contrary to the freezing layers technique, *Two Models* technique apply in cases that the output shape differs from the input. The head and base model are removed resulting in the creation of a new modified model, namely *truncated model.* The output of this model is then used as an input to a much smaller model which operates as a new independent head model producing the desired output shape.[24] A major advantage of this technique would be the fact that the outputs of the truncated model (known as embedding tensors) are directly accessible, making the ability to perform classification tasks way easier. One the other hand complexity increases, having to deal with two models.[2] In Figure 2.20 a schematic presentation of the *two model* technique, implemented on a webcam controlled model, using *MobileNet* as the base model.

Single Model

In order for this technique to be implemented, a different shape in the output and input layer is required. In this case, a new model is created, containing the feature extracting layers of the original model and those of the new head. To accomplish that, an extra step between the truncated model and the new head is needed, so to combine them into a single one, defined by the input and output of the two models respectively.[24]

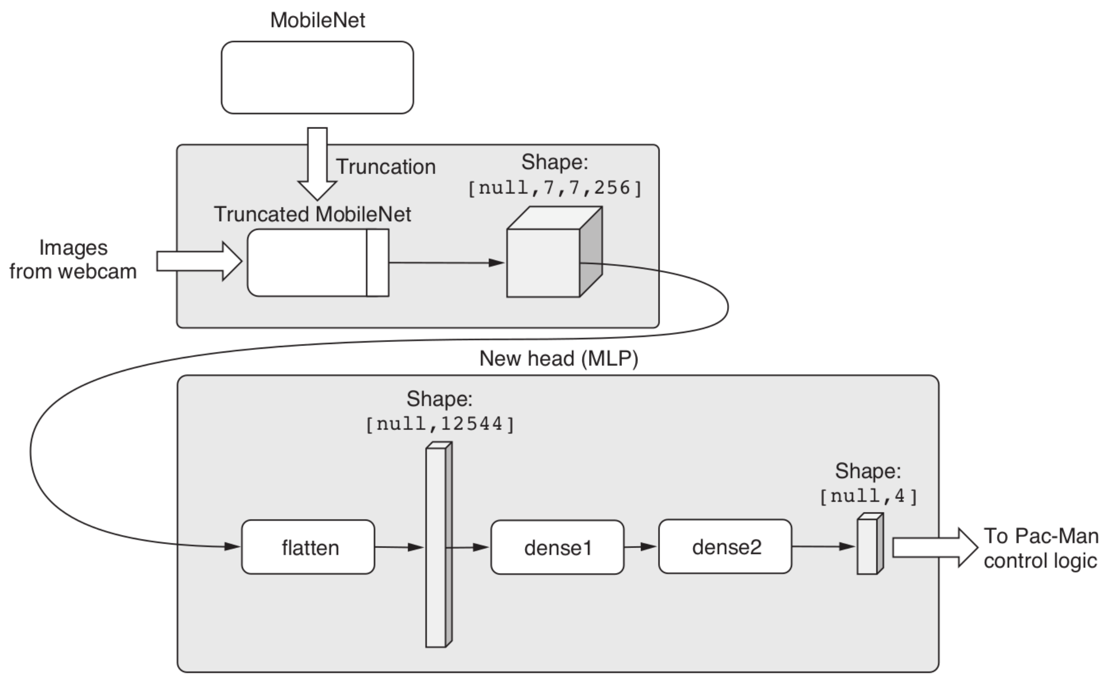


Figure 2.20: The Two Models technique.[2]

Fine Tuning

As last, optional step is *Fine-Tuning*, which consists of unfreezing the entire model obtained (or even a part of it) and re-training it on the new data with the lowest learning rate possible. This can potentially achieve meaningful improvements, by incrementally adapting the pretrained features to the new data.[30] Fine-tuning achieves a robust connection between the truncated model and the new head, resulting in an increase in accuracy. The layer's unfreezing process is depicted in the Fig. 2.21.

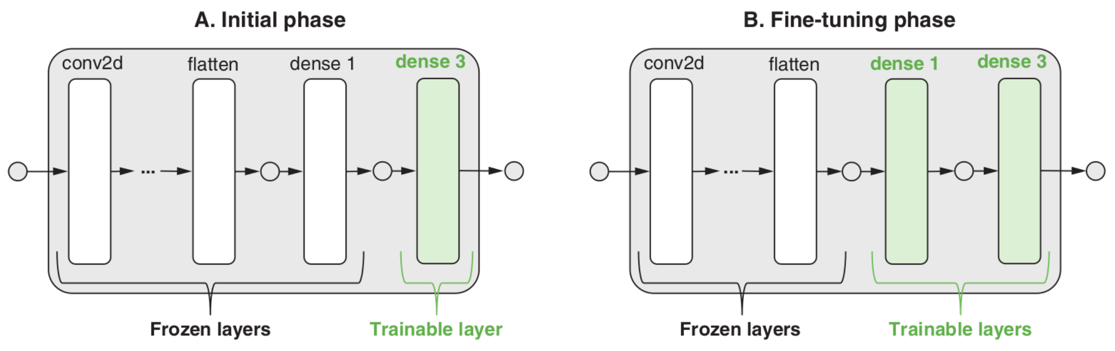


Figure 2.21: Illustration of the Fine-tuning phase.[2]

2.2.2 Overfitting and Underfitting

The main cause of poor performance in machine learning is either caused due to overfitting or underfitting the data. A model that doesn't classify well the sounds presented is said to be *underfit*. whereas the model that classifies paterns to well, to the extent that what it learns generalizes poorly to new datais said to be *overfit*.[2]

Underfitting

Usually an underfitted model is the one that can neither model the training data nor generalize to new ones. To overcome underfitting, we usually tend to create a more powerful model by making it bigger, meaning that some more layers are added with an increased size.[2]

Overfitting

Overfitting in general refers to a model that is training data to well. This occurs when noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.[31] A few good ways to limit overfitting are the following:

- *L2 regularizer*: Also called regularization for simplicity.A positive loss is assigned to the weight by calculating the summed squares of parameter values of the weight. L2 regularization forces weights toward zero without giving them exactly that value.
- *L1 regularizer*: Like L2 regularization, L1 regularization gives output in binary weights from 0 to 1 for the model's features and is adopted for decreasing the number of features in a huge dimensional dataset. L2 regularization disperse the error terms in all the weights that leads to more accurate customized final models.
- *Combined L1-L2 regularizer*: A weighted sum of L1 and L2 regularization losses.
- *Dropout*: The key idea is to randomly drop out units (along with their connections) from the neural network during training, preventing them from co-adapting to much as shown in Fig. 2.22. [13]
- *Batch normalization*: Learns the mean and standard deviation of its input values during training and uses the learned statistics to normalize the inputs to zero mean and unit standard deviation as its output.
- *Early stopping of training based on validation-set loss*: Stops model training as soon as the epoch-end loss value on the validation set stops decreasing.

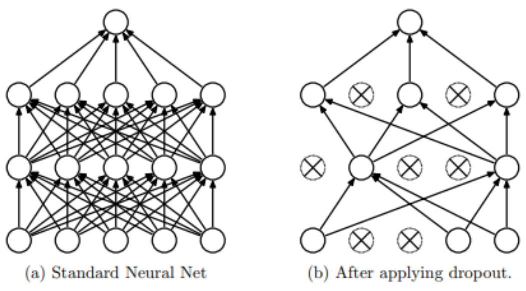


Figure 2.22: Neural Network structure after applying the *Dropout* technique.[13]

Chapter 3

Tools and Model Selection

Sound classification with the use of neural networks can be without a doubt a really demanding task. In order to achieve the goals of this thesis the most appropriate tools were chosen. This chapter presents the hardware and software setup that covered the needs of this project. A short introduction in Signal Processing and Fast Fourier Transform is presented. In addition, Google's **Teachable Machine**, the basic tool of this thesis is introduced to the reader.

3.1 Software and Hardware

Few years ago, deep learning tasks such as image and audio classification were inconceivable for individual researchers and developers, requiring extreme amounts of computational power and in detail knowledge of neural networks architecure and principles. Leading corporations in this field, such as Google, would use *TPUs* short for *Tensor Processing Units* a custom developed application-specific intergrated circuits used to accelerate machine learning workloads, allowing them to perform deep learning computations with huge dataset.[32] *GPUs (Graphics Processing Units)* and *CPUs (Central Processing Units)* found in personal computers couldn't handle the vast amount of calculations performed during the training process. However, nowadays with the advance in material technologies, high performance personal computers are available in more affordable prices resulting in an increase in the number of individual researchers and developers getting involved with deep learning tasks. Cloud computing, the enormous amount of available data and the development of more advanced algorithms has led to the rise of deep learning.

Hardware

All data collected and augmented for this thesis, front-end and back-end development, training and testing of the neural network were performed in a **laptop**. A quick review of its specifications is presented in Table 3.1 bellow:

Table 3.1: Hardware Specifications

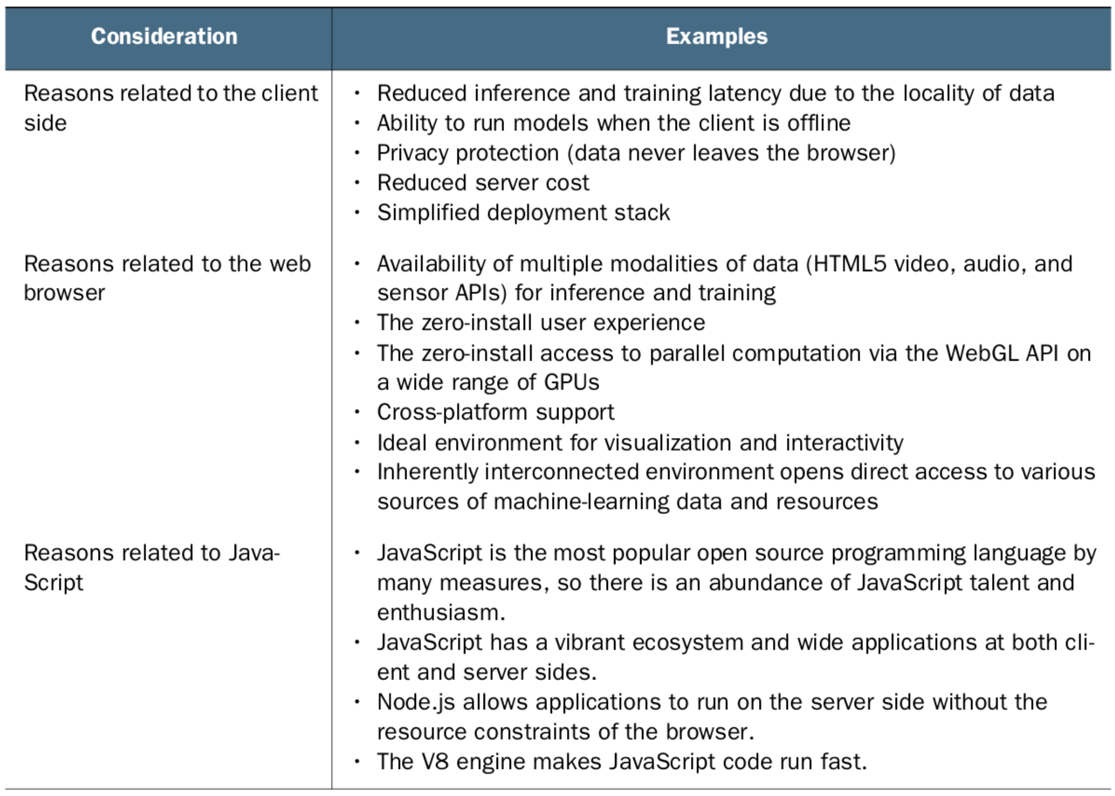
Specification	**Value**
Laptop Model	Lenovo ThinkPad E450
CPU	Intel® Core™ i7-5500U CPU @ 2.40GHz × 4 threads
GPU	AMD® Radeon R7 M260 / Intel® HD Graphics 5500 (GT2)
RAM Memory	8 GB
Disk Capacity	500 GB SSD

Operating System

The choice of a suitable operating system (OS), would benefit our project in many ways. An OS easy to program on, with simple configuration and high processing speeds would make our task, much easier. As the main operating system was the pre-installed Windows 10, we decided to convert the computer to a dual boot system. We choose to work with **Ubuntu 18.04.05 LTS** and not with the latest version of Ubuntu distributors available, due to some errors presented during the installation phase. In any case, Ubuntu is the most preferred Linux based OS when one wants to perform Machine Learning tasks. Being the most popular distributor, results in having a great online community with users presenting their problems and suggesting solutions which proved to be really helpful throughout the whole project. The vast community in sites such as GitHub, AskUbuntu, Stack Overflow etc. with questions and solution in almost every problem that came across our project, endorsed our decision to run with the above mentioned OS.

Programming Language

While most machine learning tasks, are performed with more ”traditional” backend-focused languages like Python and R, our goal was to create a web based user friendly environment, were everyone would be able to understand and interact with. We choose **Javascript** as the project's main development language, obviously along with **CSS** and **HTML** for the frontend part of the application. As stated before in this thesis, the training of deep neural networks requires high computational power, not always available at the browser tab. Mostly due to the extensive amount of data processed, back-end based programming language would help significally reduce the time required for the training process.[2] However, with the release of TensorFlow.js library for machine learning (before, it was only available as TensorFlow, working with Python and C++) and Node.js for backend-development and the latest updates in Google's Teachable Machine in Audio models, training an audio model in the browser seem to be a feasible task. A few advantages of deploying deep learning in JavaScript are presented in Figure 3.1 below.



Consideration	Examples
Reasons related to the client side	• Reduced inference and training latency due to the locality of data • Ability to run models when the client is offline • Privacy protection (data never leaves the browser) • Reduced server cost • Simplified deployment stack
Reasons related to the web browser	• Availability of multiple modalities of data (HTML5 video, audio, and sensor APIs) for inference and training • The zero-install user experience • The zero-install access to parallel computation via the WebGL API on a wide range of GPUs • Cross-platform support • Ideal environment for visualization and interactivity • Inherently interconnected environment opens direct access to various sources of machine-learning data and resources
Reasons related to Java-Script	• JavaScript is the most popular open source programming language by many measures, so there is an abundance of JavaScript talent and enthusiasm. • JavaScript has a vibrant ecosystem and wide applications at both client and server sides. • Node.js allows applications to run on the server side without the resource constraints of the browser. • The V8 engine makes JavaScript code run fast.

Figure 3.1: The benefits of doing deep learning in JavaScript.[2]

Deep Learning Libraries

The most appropriate library for our project, was **TensorFlow.js**. Being totally compatible with its predecessor *TensorFlow*, the Python framework for deep learning, one can imagine that this library would assist us, in the best way possible. After researching online, we found a wide variety of machine learning libraries available to fit each developers needs such as brain.js, ConvNetJS, Deeplearn.js etc. Firstly, due to the fact that Google's Teachable Machine, uses TensorFlow.js as its basic library for performing audio classification tasks was one of the key parameters, assisting us in choosing the most appropriate library for our prpject. Another great aspect of this library is its comprehensiveness. Some key parts are presented below: [2]

- Supports both inference and training.
- Supports web browsers and Node.js.
- Leverages GPU acceleration (WebGL in browsers and CUDA kernels in Node.js).
- Supports definition of neural network model architectures in JavaScript.
- Supports serialization and de-serialization of models.
- Supports conversions to and from Python deep-learning frameworks.
- Compatible in API with Python deep-learning frameworks.
- Equipped with built-in support for data ingestion and with an API for visualization.

The second reason is the ecosystem. Most JavaScript deep-learning libraries define their own unique API, whereas TensorFlow.js is tightly integrated with TensorFlow and Keras.[2] Furthermore, the fact that most applications nowadays are programmed in JavaScript, a great online community has formed, with users from all over the world posting their problems and possible solutions, helping in the improvement of the language and the enhancement of its libraries (i.e TensorFlow.js)

3.2 Model Selection

The inspiration for this thesis, derives from the need of monitoring the Turbocharger's speed in the *Hybrid Integrated Propulsion Powertrain (HIPPO-2)* of NTUA's Laboratory of Marine Engineering. Until now monitoring and controlling of the testbed was accomplished through *DSpace Microautobox II* with the use of MATLAB/Simulink via Ethernet and CAN-bus. A general mapping with the use of the above mentioned software has been performed, providing us with the appropriate data needed to perform the classification task.

Many studies have been conducted, focusing on environmental sound classification as it is proven to be a very accurate way to monitor various ecosystems health. Birds sound classification[33], bat detection[34] are some really interesting projects, inspiring us to tackle our task in a similar way. It is of great interest in the machine learning community, how would image classification models perform sound classification tasks. It turns out that spectrograms can be treated as images, meaning that they can be used in standard neural networks architectures such as AlexNet or ImageNet producing state of the art results. To be precise, AlexNet model achived 78% on the GTZAN music genre classification dataset.[35] Our major concern was the fact that, all the above mentioned researches, have a few problems presented bellow:

- The tests were performed neither in JavaScript or Node.js
- The machine learning library used was not TensorFlow.js
- The tests were not executed in browser, but instead in powerful computer systems.

The question that rose was, if it would be possible to perform audio classification tasks with relevant accuracy in a browser tab. Surely, in order to conclude if the above mentioned is feasible we had to conduct an experiment, by creating a small audio dataset, thoroughly selected in order to cover some cases regarding the distance from the source of the sound, background noise etc.

3.2.1 Teachable Machine

Google's *Teachable Machine* is a web-based tool that makes creating machine learning models fast, easy, and accessible to everyone. It's main goal would be to help individuals with low or even no expertise in the field, to perform machine learning tasks in a browser tab. It performs three main operations:

- Image Recognition
- Audio Classification
- Video/Pose recognition

This web-based application, combines all stages of creating a machine learning model meaning that it allows the user to create the desired dataset, train the model and finally test it. With the use of Transfer Learning technique (see subsection 2.2.1) allows the user to create and even export a machine learning model without having to code anything. The Audio Classification tasks performed by TM assisted us for the completion of this project. A brief analysis of the basic working principals for the Audio Classifications operations, are presented in the following paragraphs.

Teachable Machine User Interface

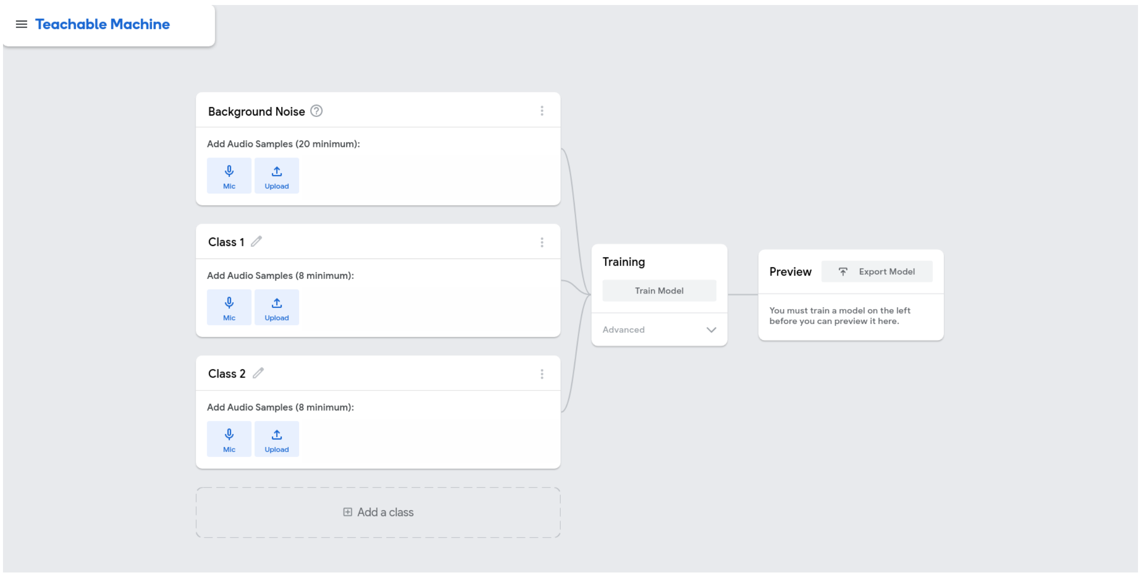

Teachable Machine
Background Noise
Add Audio Samples (20 minimum):
Mic
Upload
Class 1
Add Audio Samples (8 minimum):
Mic
Upload
Class 2
Add Audio Samples (8 minimum):
Mic
Upload
Add a class
Training
Train Model
Advanced
Preview
Export Model
You must train a model on the left before you can preview it here.

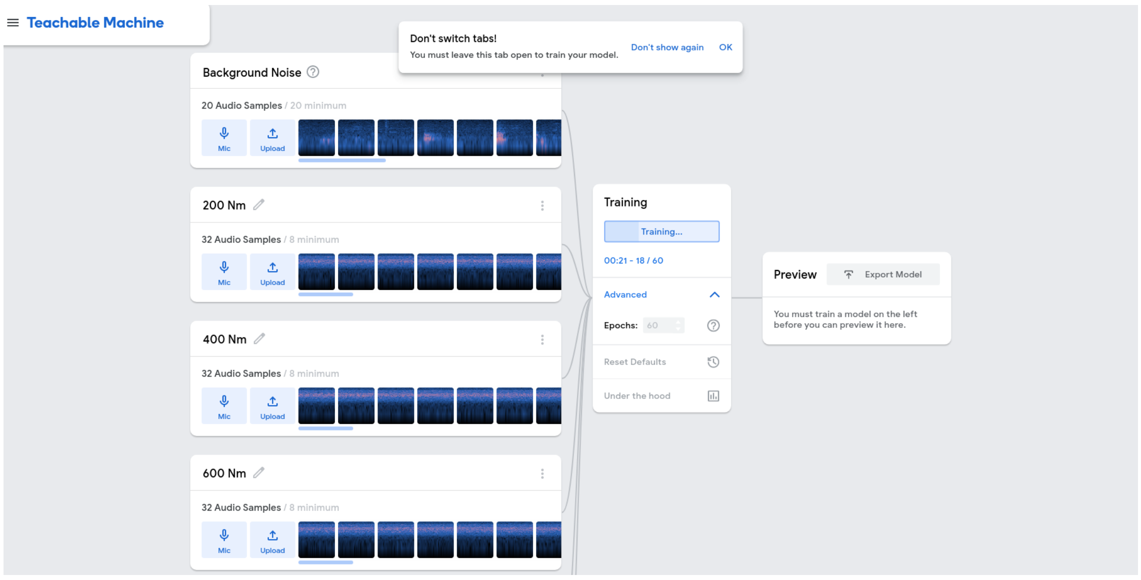
Figure 3.2: Creating a new Audio Classification project in *Teachable Machine*[14]

Google's *Teachable Machine* main focus as a project, would be to get more people involved in Machine Learning tasks, without any specific coding knowledge, just by using their webcamera, built-in microphone, or sounds. The complexity of this tasks, is hidden from users, who simply benefit by needing less data and training time to create useful and accurate ML models. Since launch, people have trained over 125,000 classification models. It provides an approachable yet well-featured interface for children and adults to create their own ML classification models through its website. It enables users to train classifiers for an arbitrary number of classes, provides data collection and classification, model training, and model evaluation in the same interface, and trains on-device (which results in faster performance and enables training to be free).[36]

Creating and Exporting a Model

The same procedure followed while creating a ML classification model is also followed when the user initiates the creation of a new project in the TM environment. All common steps i.e model-data collection and classification, training and evaluation are performed by Teachable Machine and are presenter to the user only if decided so. This technique assists novice users experiment with this platform without getting into more confusing details, allowing at the same time researchers and in general more advanced users to make the most of this powerful tool.

To begin with, user first selects whether the model should classify images, sounds or poses as input (see Fig 3.2). For the purposes of this thesis project only sound classification operations were performed. Secondly, according to users preferences, classes are created, for the model to learn to classify. As Figure 3.3 shows, for variable engine outputs measured in the dynamometer, different sound snippets were recorded and classified in order to help the model in training and classification process. After having all required sound data gathered, by clicking the *Train Model* button, TM training is initiated. In the meantime, some messages pop up in the browser tab, notifying the user to stay in the tab until the training process is completed. For a more experienced user who requires more control during training, TM provides an "advanced" training section for hyperparameter tweaking (i.e epochs, batch size, learning rate) and an "under the hood" panel for model evaluation visualizations. Unfortunately, for audio classification projects, only *epochs* and *overlapping* can be parameterized by the user. More will be discussed in the following subsection.



Don't switch tabs!
You must leave this tab open to train your model.

20 Audio Samples / 20 minimum

32 Audio Samples / 8 minimum

32 Audio Samples / 8 minimum

32 Audio Samples / 8 minimum

00:21 - 18 / 60

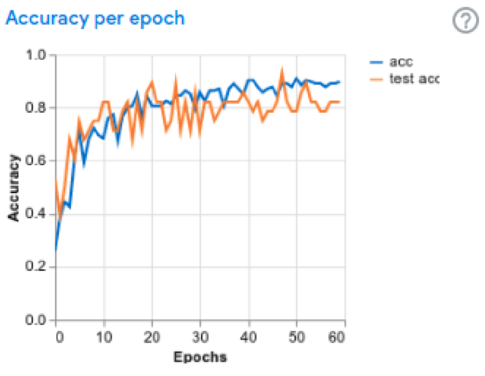
Epochs: 60

You must train a model on the left before you can preview it here.

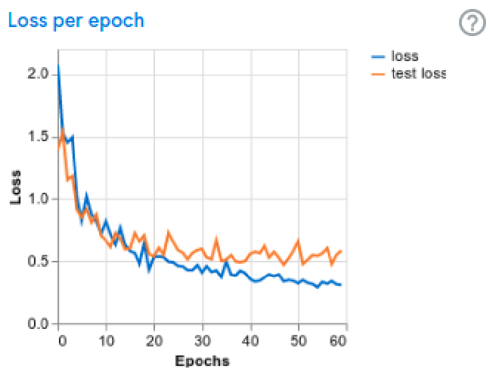
Figure 3.3: Training of an audio project in *Teachable Machine*[14]

Advanced Options and Model Evaluation

Despite being an easy to operate web tool enabling novice users to get involved with ML tasks, Teachable Machine has offers a few advanced options for those who want to have a closer look during the training process. As pictured in Figure 3.3 while TM is training the model, by clicking the *Advanced* button, the user is able to select the preferred number of epochs. Furthermore another option namely *"Under the Hood"* is presented to the user. By enabling it, a new section is displayed were a few model evaluation visualizations are shown. In detail, the user comes across two plots depicting the accuracy of the model per epoch and the loss per epoch. As epochs, we encounter the number of iterations which the model has gone through the entire dataset. If for example the model has been set for 60 epochs, this means that the model trained, will work through the entire dataset 60 times.[14] It should be stated that in both diagrams two different curves are presented. The one with the blue colour monitors the test set, created by the user while entering data in the classification process and orange which monitors the trained models correspondence to the new data given through the microphone. The *Accuracy per epoch* plot provides some useful information regarding the model's performance. The higher the curves rise, as we go through the epochs, the better it performs. The *Loss per epoch* provides the user with some really significant intel, regarding the model's overfitting or underfitting behaviour. The training loss(blue colour) indicates how well the model is fitting the training data, while the validation loss(orange colour) indicates how well the model fits new data. The above trends observed in Figure 3.4 are also known as *Learning Curves*.



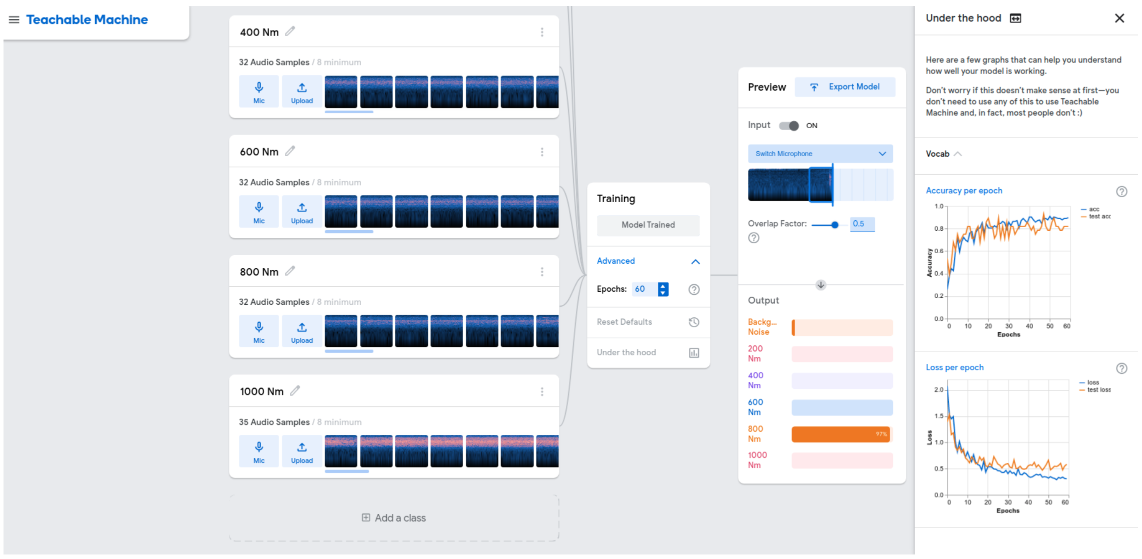
(a)



(b)

Figure 3.4: *Learning Curves* provided by Teachable Machine. The *Accuracy per epoch* trend presented on the left and *Loss per epoch* trend on the right

Teachable Machine allows the user to interfere with the *Overlap Factor*. Simply to be put, the overlap factor determines how frequently the last second of audio is tested against the created model. For example in an model with overlap rate of 0, audio will be classified every second, whereas with 0.5 overlap audio will be classified every half a second and so on. All the above mentioned, are shown in Figure 3.5 as depicted in Teachable Machine's UI.



32 Audio Samples / 8 minimum

32 Audio Samples / 8 minimum

32 Audio Samples / 8 minimum

35 Audio Samples / 8 minimum

Epochs: 60

Overlap Factor: 0.5

Here are a few graphs that can help you understand how well your model is working.

Don't worry if this doesn't make sense at first—you don't need to use any of this to use Teachable Machine and, in fact, most people don't :)

Figure 3.5: Trained TM audio project with "under the hood" parameters shown [14]

Speech Commands Recognizer

Teachable Machine, performs all this various tasks, by implementing the technique of Transfer Learning. In detail, a base model called *Speech Commands Recogniser* is already pretained with the *Speech Commands 18w* dataset.[37] Speech Command Recognizer is a JavaScript module that enables recognition of spoken commands comprised of simple isolated English words from a small vocabulary. The default vocabulary trained with Speech Commands 18w dataset includes the following words: the ten digits from ”zero” to ”nine”, ”up”, ”down”, ”left”, ”right”, ”go”, ”stop”, ”yes”, ”no”, as well as the additional categories of ”unknown word” and ”background noise”. It uses the web browser's WebAudio API. It is built on top of TensorFlow.js and can perform inference and transfer learning entirely in the browser, using WebGL GPU acceleration.[38] In detail, the end user in Teachable Machine web interface interacts only with the classification layer, which is actually the only layer trained with our new dataset. While creating a new project, the above mentioned layer is depicted in the webpage as shown in Figure 3.2.

3.2.2 Web Environment

Since, the preferred language for this project is JavaScript, it was decided to take advantage of it by creating a custom web page to present our audio classification model. However, it was necessary to provide the reader with a deep analysis of how sound is processed in a way that allows the model to understand it. Taking into consideration that the tools presented in the webpage, assisted us in preprocessing and labelling the audio samples, a brief analysis in Fast Fourier Transformation, spectrograms and the bar graphs is offered in the following paragraphs.

Fast Fourier Transformation - FFT

The Fast Fourier transform (FFT) is an efficient algorithm used for converting a time-domain signal into an equivalent frequency-domain signal, based on the discrete Fourier transform (DFT).[39] The Fourier transform can be powerful in understanding and also troubleshooting errors in everyday signals. Although the Fourier transform is a complicated mathematical function, it isn't a perplexing concept to understand and relate to your measured signals. Essentially, it takes a signal and breaks it down into sine waves of different amplitudes and frequencies.[40] The FFT is one of the most commonly used operations in digital signal processing to provide a frequency spectrum analysis.

In general, Fourier transformation allows us to describe a complicated function as an infinite summation of sines and cosines. With the use of DFT, we try to approximate the Fourier Series on a finite interval were the function is periodic. The Fast Fourier Transform algorith, could be described as computational efficient way for calculating the DFT. For the sake of completeness, in the following paragraphs short mathematical analysis of the above mentioned concepts is presented.[15] Firstly, we come across the Fourier Series. A fundamental result of Fourier Analysis in general, is that if a $f(x)$ is periodic and piecewise smooth, it could be written in terms of a Fourier Series as described in equation 3.1 below:

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k cos(kx) + b_k sin(kx)) \tag{3.1}$$

The Fourier series for an L-periodic function on $[0,L]$ is given by the following equation

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k cos(\frac{2\pi kx}{L}) + b_k sin(\frac{2\pi kx}{L})) \tag{3.2}$$

Due to the fact, that we are expanding functions in terms of cosine and sine, Euler's Formula is implemented in order to describe the Fourier Series in a complex form as shown below,

$$e^{ikx} = \cos(kx) + i\sin(kx) \tag{3.3}$$

and finally by combining all the above, **Fourier Series** is produced:

$$f(x) = \sum_{k=-\infty}^{\infty} C_k e^{ik\pi x/L} \tag{3.4}$$

with C_k the complex coefficient being equal to:

$$C_k = \frac{1}{2L}\int_{-L}^{L} f(x)e^{-ik\pi x/L}dx \tag{3.5}$$

Restating all the above, as shown in equation 3.2, $f(x)$ is now presented by a sum of sines and cosines with a discrete set of frequencies given by $\omega_k = k\pi/L$. Taking the limit as $L \to \infty$, these discrete frequencies become a continuous range of frequencies. By defining $\omega = k\pi/L$, $\Delta\omega = \pi/L$ and by taking the limits accordingly meaning that $L \to \infty$, $\Delta \to \infty$, the **Fourier Transform** is presented:

$$f(x) = \lim_{\Delta\omega\to 0} \sum_{k=-\infty}^{\infty} \frac{\Delta\omega}{2\pi} \int_{-\frac{\pi}{\Delta\omega}}^{\frac{\pi}{\Delta\omega}} f(\xi)\; e^{-ik\Delta\omega} d\xi\; e^{ik\Delta\omega x} \tag{3.6}$$

By further analysing equation 3.6 one can observe that, while $\Delta \to \infty$ the sum presented in the equation becomes a *Riemann integral* as listed below,

$$f(x) = \int_{-\infty}^{\infty} \frac{1}{2\pi} \int_{-\infty}^{\infty} f(\xi)e^{-i\omega\xi}\; d\xi\; e^{i\omega x}\; dx \tag{3.7}$$

with $\hat{f}(\omega)$ representing the *Fourier Coefficients*:

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} e^{-i\omega\xi} d\xi \tag{3.8}$$

For the sake of completeness, the following integrals known as the *Fourier Transform Pair* are presented. These two integrals, assist us in calculating the *Fourier Transform* given the $f(x)$ function and the so called *Inverse Fourier Transform* given the *Fourier Coefficients* $\hat{f}(x)$,

$$\begin{aligned} f(x) = \mathcal{F}^{-1}(\hat{f}(\omega)) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\hat{f}(\omega)e^{i\omega x}d\omega \\ \hat{f}(x) = \mathcal{F}(f(x)) = \frac{1}{2\pi}\int_{-\infty}^{\infty}f(x)e^{-i\omega x}dx, \end{aligned} \tag{3.9}$$

When computing and working with real-data (i.e sound), it is necessary to approximate the Fourier transform on discrete vectors of data. Simply to be put, lets think of the way that sound is transformed from analogue to digital form, so that it can be understood from computers. Like all physical phenomena sound is considered analogue or continuous. By going through the process of sampling we can convert analogue sound to digital with the use of an ADC short for Analogue to Digital Converter device, by reading and storing the instantaneous amplitude of the analogue sound wave at regular intervals of time. Lets think of each individual point presented in Figure 3.7 as these readings mentioned before.

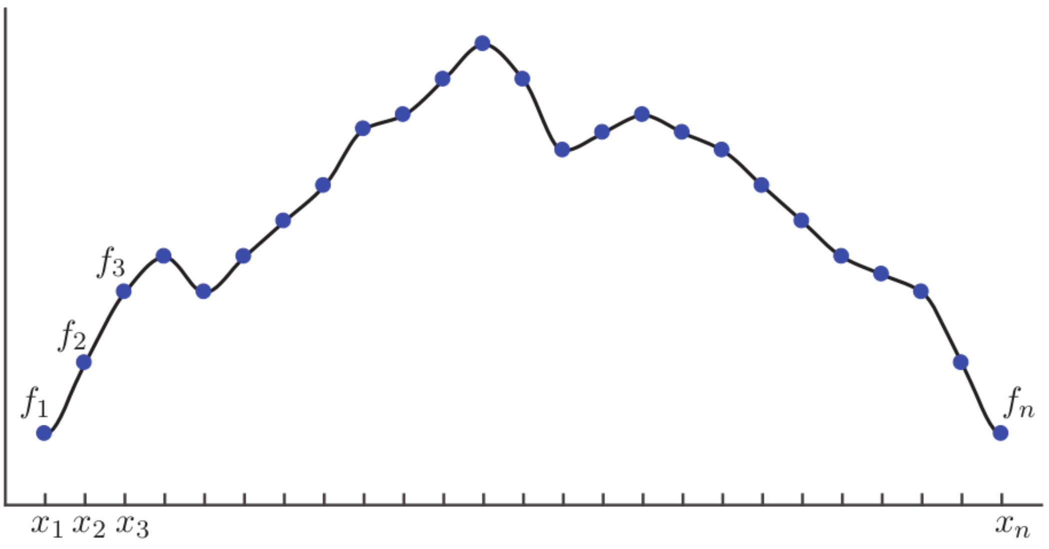


Figure 3.6: Discrete data sampled for the discrete Fourier Transform [15]

As its name suggests, a vector of data f is created by discretizing the function $f(x)$, were discretizing in our case is the process of sampling. For each of this individual data points in the above mentioned vector, its *Fourier Coefficient* $\hat{f}_k$ shall be calculated. The simplest formulation of the DFT is as follows:

$$\hat{f}_k = \sum_{j=0}^{n-1} f_j e^{-i2\pi jk/n} \tag{3.10}$$

Essentially, the DFT is a matrix that maps data points $\mathbf{f}$ to the frequency domain $\hat{f}$:

$$\{f_1, f_2, \cdots, f_n\} \Rightarrow \left\{\hat{f}_1, \hat{f}_2, \cdots, \hat{f}_n\right\} \tag{3.11}$$

For a given number of points n, the DFT represents the data using sine and cosine functions with integer multiples of a fundamental frequency, $\omega_n = e^{-2\pi i/n}$. DFT is finally calculated by matrix multiplication as follows: [15]

$$\begin{bmatrix} \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \vdots \\ \hat{f}_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)^2} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{bmatrix}$$

Figure 3.7: DFT calculation via matrix multiplication [15]

Surely a major drawback of DFT, is the fact that a tremendous amount of computation is needed, as it involves a dense n x n matrix, requiring N^2 operations. The **Fast Fourier Transform** is closely connected to the DFT, by assisting us to calculate it faster requiring only $N \log(N)$. As N becomes very large, the $\log(N)$ component grows slowly and the algorithm aproaches a linear scaling[15]. This algorithm was developed by James W. Cooley and John W. Turkey in 1965 and its since used in a vast amount of applications. The main concept behind this algorithm is the fact that, when number of data points is a power of 2, the DFT can be implemented way faster and more efficiently. For example lets take $n = 1024 = 2^{10}$ points. The same steps followed while implementing the DFT algorithm will be made.

$$\hat{f} = \mathcal{F}_{1024} f = \begin{bmatrix} \mathcal{I}_{512} & -\mathcal{D}_{512} \\ \mathcal{I}_{512} & -\mathcal{D}_{512} \end{bmatrix} \begin{bmatrix} \mathcal{F}_{512} & 0 \\ 0 & \mathcal{F}_{512} \end{bmatrix} \begin{bmatrix} \mathbf{f}_{\mathbf{even}} \\ \mathbf{f}_{\mathbf{odd}} \end{bmatrix} \tag{3.12}$$

As shown in equation 3.12, while implementing the DFT, the matrix $\mathcal{F}_{1024}$ was written as a product of two other matrices and the entries of f were a bit reorganised depending if they were odd, or even. Now we have created diagonal matrices resulting in much lower computational costs. To conclude, the FFT is based in the observation that the *Fourier Coefficient* matrix of the DFT algorithm possesses susch symetry, that with only a minor rearrangements the computational cost, could be significantly reduced. Even in cases, that the number of entry points is not a power of 2, just by padding a few zeros in our produced matrices, a much smaller computational cost would be achived compared to the DFT algorithm.[15]

Finally as presented in the web page created for the need of this thesis, we implemented *p5.js* library for the creation of the graph visualizing the FFT. The above library is an open-source Javascript library for creative coding, providing the user with the essential tools to fulfill a variety of different tasks such as 2D and 3D rendering content on the HTML canvas element.[41] It was first introduced to the community as *Processing.js* a discontinued JavaScript port of Processing, a framework designed to produce visualizations, images and interactive content. The development of Processing.js was started by John Resig (RIT), known for the creation and development of the jQuery JavaScript library. In the following years, Lauren Lee McCarthy (MIT) created the p5.js library, an open-source and web based version of the above mentioned software.

For the sake of good order, it was deemed necessary to provide the reader with the basic parameters of the Fast Fourier Transform algorithm, implemented in our code.

- *Sampling Rate (SR)*: is the number of times a signal is read in a second. In our case the SR would be 44100 times. In some cases it is also presented as *sample frequency*, or 44.1 kHz
- *Bins (n)*: The number of "horizontal" strips that the window is divided. The number of bins determines how accurate the analysis would be in terms of frequency detection. The number of bins must be a power of 2 as mentioned before, so in our case we choose 1024 bins
- f_{max} : The highest possible frequency that can be possibly analysed conventionally called the *Nyquist limit frequency*, exactly equal to one-half the sampling rate, meaning that the algorithm created can represent frequencies up to 22050 Hz.
- *FFT Size*: The number of samples taken for the FFT algorithm to be performed.It defines the number of bins used for dividing the window into equal strips, namely the bins. The FFT Size is always twice of the number of bins, in our case 2048.
- *Frequency Resolution (FR)*: The frequency band of a bin, the "energy" or amplitude collected around its individual bin. It can be calculated either by dividing the Nyquist limit frequency by the number of bins, or by dividing the sampling rate by the FFT Size

All the exact values used in our algorithm are presented in Table 3.2 and the output presented in our web page is depicted in Figure 3.8.

Table 3.2: FFT algorithm Parameters

Parameter	**Value**
Sampling Rate (SR)	44100 Hz
f_{max}	22050 Hz
FFT Size	2048
Bins	1024
Frequency Resolution	43.066 Hz

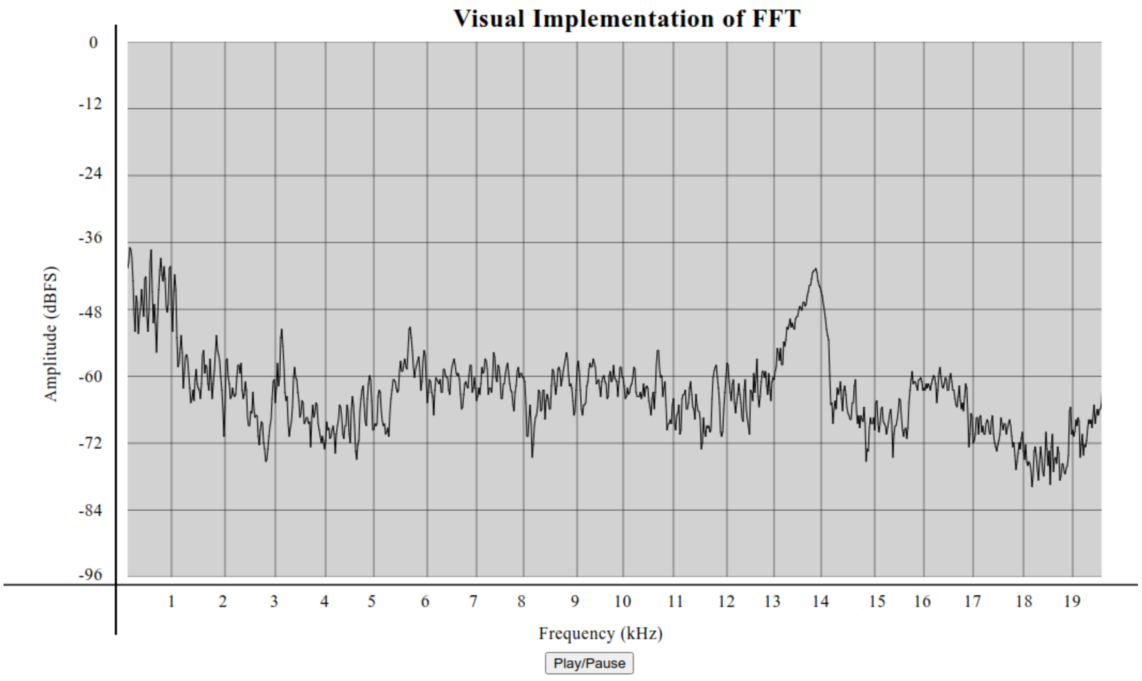


Figure 3.8: Visual Implementation of the FFT Transformation in the Frequency domain

Spectrograms

A spectrogram is a visual depiction of a signal's frequency composition over time.[42] In the previews subsection, the visual implementation of the Fast Fourier Transform was presented, a very useful tool that allowed us to describe an audio signal in the frequency domain. The only drawback was that we weren't able to store information regarding the time domain, meaning when the specific frequencies depicted in the graph occured.[15] Teachable machine implemented in this thesis and in general most audio classification machine learning algorithms, use spectrograms, to represent raw audio in an image like form. By deciding to work with 2D ConvNets, we should come up with a way to preserve both time and frequency domain of the sound recordings. The most efficient way to do that is by using spectrograms. A few basic properties are presented in the following paragraphs.

Firstly, a brief analysis on how a spectrogram is created. It appears that another form of the Fourier transform is used, namely the *Short Fourier transform*. A windowed FFT is computed as shown in Figure 3.9 helping to localize the frequency content in time.

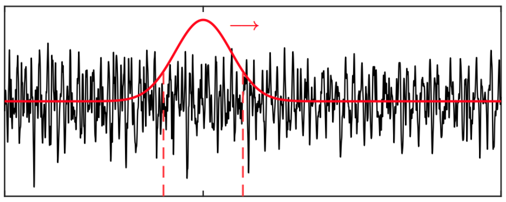


Figure 3.9: Windowed FFT helping to localize the frequency content [43]

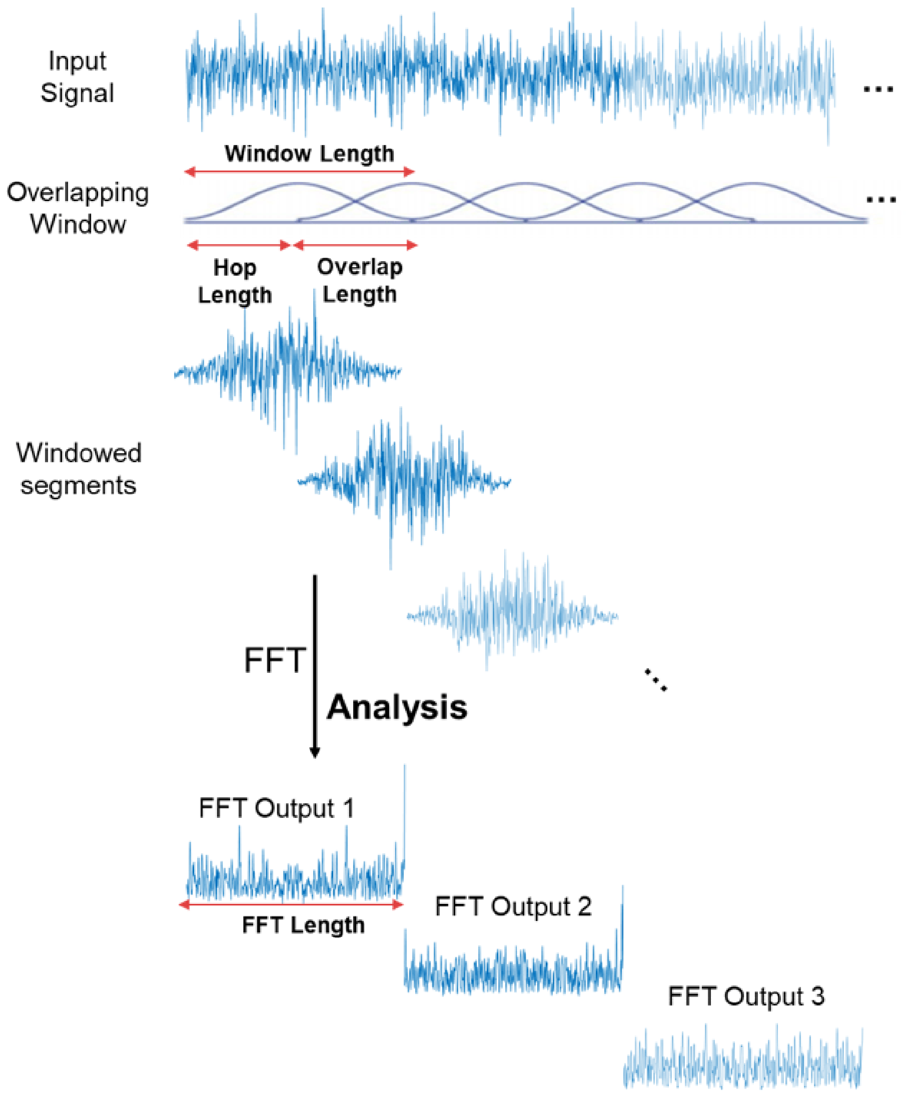


Figure 3.10: Visual Representation of the Short-Fourier Transform [43]

Simply to be put, we can think of a spectrogram as a representation of FFT's stacked on top of each other. There are a few other details worth mentioning. Taking into concideration that humans can only percive a really small and concentrated range of frequencies and amplitudes, an appropriate scaling in the frequency axis (y-axis) should be applied. Studies have shown that humans do not perceive frequencies on a linear scale. For example, it is thought to be easier to tell the difference between 500 and 1000 Hz rather than 10,000 and 10,500 Hz.[44] To solve this issue a mathematical operation were a a unit of pitch such that equal distances in pitch sounded equally distant to the listener was proposed, namely the *mel scale*.[45]

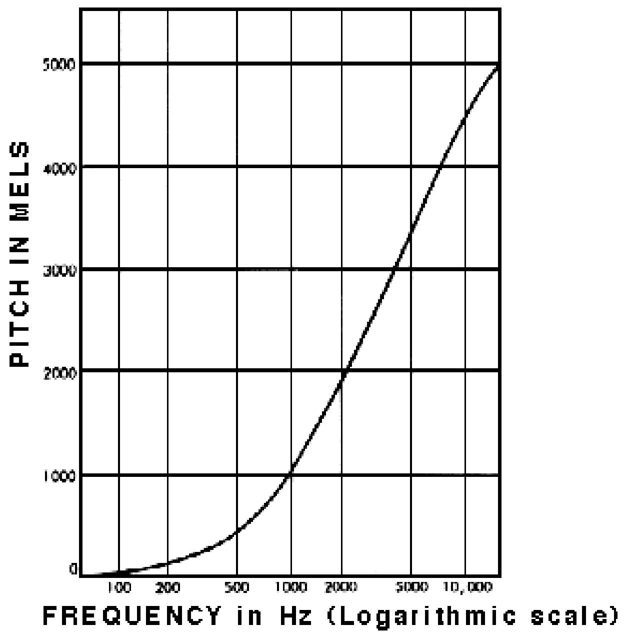


Figure 3.11: Mel Scale Explained
[45]

All the above operations, result finally in the creation of the so-called *Mel Spectrograms.* As the name suggests, these are spectrogram representations where the frequencies are converted to mel frequencies. Another really important mathematical process applied in spectrograms is *windowing.* The mathematical process of analyzing a frequency presumes that every sound ”snapshot” taken will immediately repeat. It is often the case, that at the end of the first snapshot taken, doesn't smoothly mesh with the start of the next one. This phenomena occurs due to the fact that signal are not repetitve. If we were to assume that they actually were, a small discontinuity should appear at the edge of the recording period. The only way to produce a repeating signal with a discontinuity is to add a high frequency component to the signal to resemble the ”jump” created by the discontinuity. This process/mathematical operation is called windowing and is actually a way to reduce those small errors mentioned before when the frequency content of a signal is computed. [46]

At this point, it should be stated that Teachable Machine, the platform responsible for producing the audio classification model implemented in this thesis, doesn't allow the user to control features like windowing, thus a further analysis was deemed unnecessary. For the sake of good order we state same of the most common windowing operatos used while creating spectrograms. The spectrogram created for our webpage as depicted in Fig. 3.12, uses a Hanning Window.[46]

- Hanning Window
- Hamming Window
- Blackman Window
- Kaiser Window
- Kaiser-Bessel Window

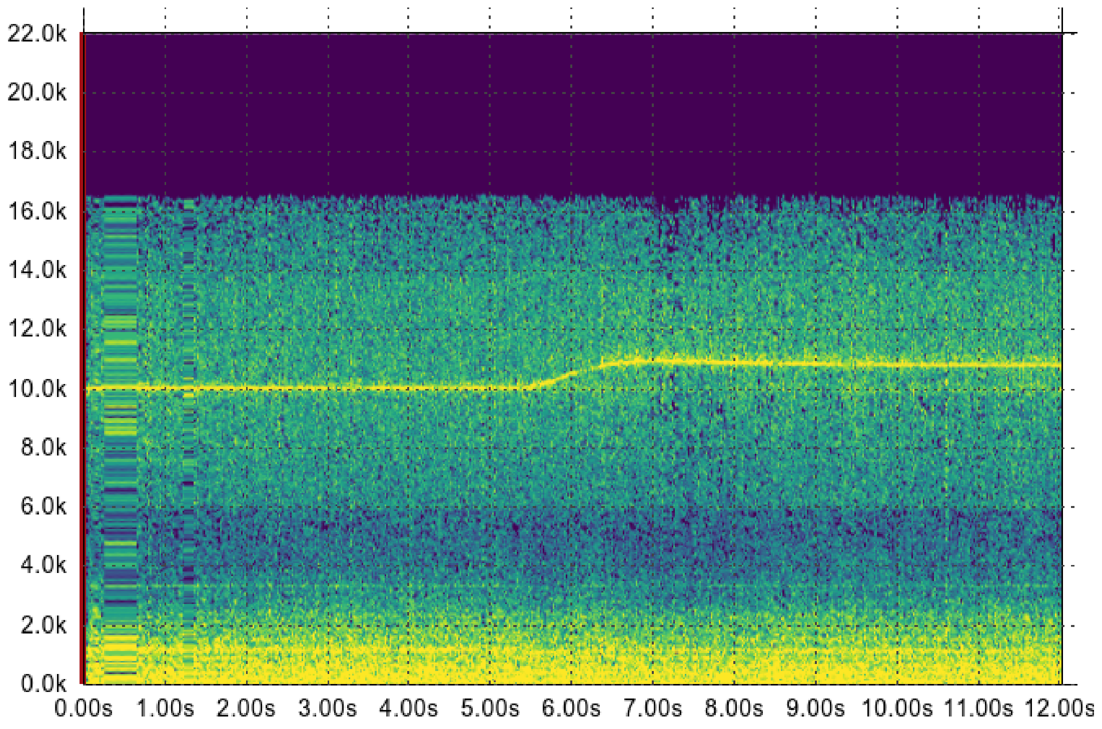


Figure 3.12: Custom Spectrogram with Hanning window created for our website.

3.3 Model Architecture

As mentioned in the first chapter, the main goal of this thesis would be the use of an audio classification model to monitor the changes of the Main Engine's turbocharger speed of the HIPPO II testbed. The main purpose of this project, is not the development and training of a sound classification model from the very beginning. Instead, we decided to implement a pre-developed state-of-art audio classification model that would allow us a few modifications according to the project's needs. We wanted to adopt all the above stated tools and in the mean time perform all machine learning tasks in JavaScript and engage them in a web environment. Google's Teachable Machine assisted us in creating a whole new audio classification model trained on our very own dataset. As mentioned in subsection 3.2.1 the model powering our application is based on Tensorflow.js *Speech Commands* module.[37] We decided to work with Tensorflow.js framework due to the fact that it supports the use in web browsers. The above stated library provides a Javascript implementation of the Fast Fourier Transform (see more in subsection 3.2.2) allowing us straight forward preprocessing of the recorder audio files. Combined with the WebAudio API supported by all major browsers, FFT produces a spectrogram which serves as a 2D image representation.[47]

As shown in Figure 3.13, the model consists of 13 layers with *four* pairs of *Conv2D* to *Max Pooling* layers, a *Flatten* and a *Dropout* layer and finally *two Dense* layers.

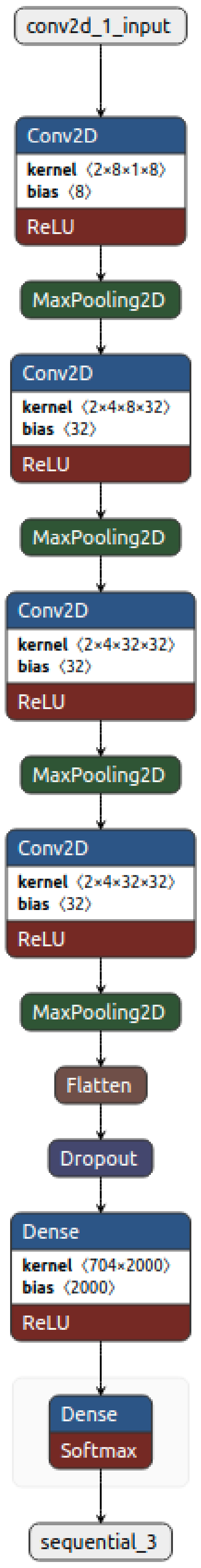


Figure 3.13: Model Architecture

Chapter 4

Model Simulation and Results

This chapter explains in depth the simulation process of our model, on detecting the various changes of Internal Combustion Engine's (ICE) T/C speed and the various obstacles faced during development. Extracted images from our web application interface are presented, in order to assist the reader in understanding the performance of our neural network, the techniques implemented to maximize it, as well as the conclusions that we reached by evaluating the results.

4.1 Transfer Learning on Teachable Machine: Dataset Creation

One really important step to train an audio classification model with success is to posses accurate input data. This section presents all steps followed in order to create a small yet efficient sound dataset.

4.1.1 Data Selection

The first step one has to follow in order to create an audio classification model is to determine in which form will the sound data be captured. Most projects published suggest that sound data should be captured in mono, *.wav* format with a sample rate of 44,1 kHz so, we decided to do so also. As it turned out, it wouldn't matter a lot for Teachable Machine module, due to the fact that while creating our dataset, Teachable Machine performs its own recording. Unfortunately,the properties of the captured sound are not available. Our best guess would be, that in order to be compatible with WebAudio API and the Fast Fourier Transformation performed by TensorFlow.js it has to follow the above mentioned values.

We decided that, in order to have a really accurate model, the sound recordings would have to be performed at different locations. As shown in Figures 4.2 and 4.3, the recordings took place in two different locations. One would be in an approximate distance of 0,5 m between the laptop and the T/C and the second one was performed about 6,0 m away from the ICE. The main idea was to perform recordings also in various distances around the experimental facilty, but after inspecting the spectrograms of each recording, we discovered that there were no major differencies when comparing them in mid range distances. The ICE is electronically controlled either in speed control mode or by demanding the desired indicated torque output which leads to injection of a certain amount of fuel by the engine ECU using an internal mapping.[16] We decided that the ICE would operate in continuous range of approximately 1800 rpm and with proper control and the dynamometer (EB) would demand a final indicated torque output of 1000 Nm. Starting from 200 Nm which was indicated as IDLE in the labelling process, we took a total of four(4) steps of 200 Nm each as seen in Fig. 4.1.

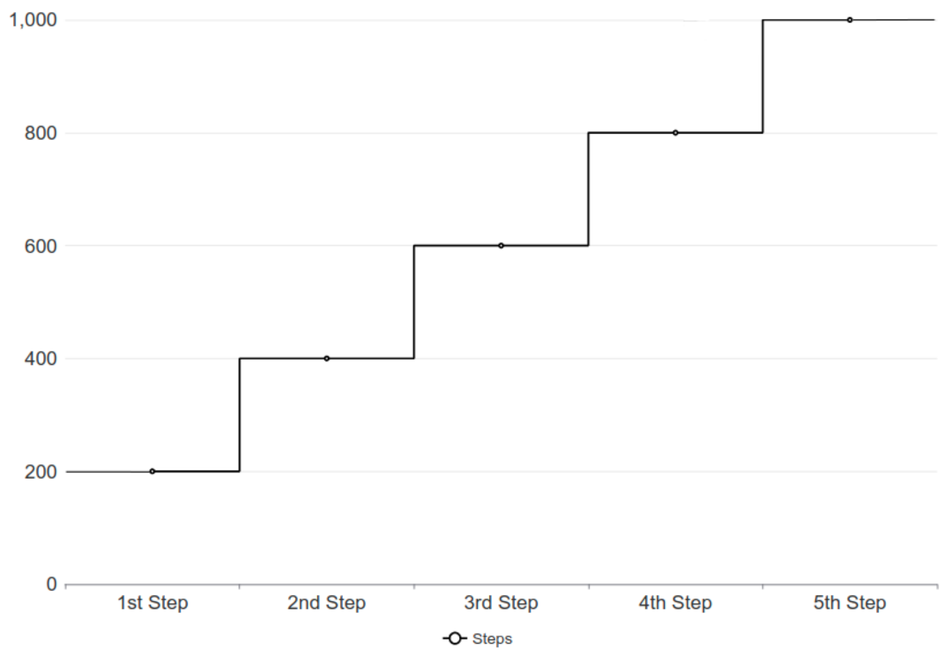


Figure 4.1: Indicated Torque Output Steps

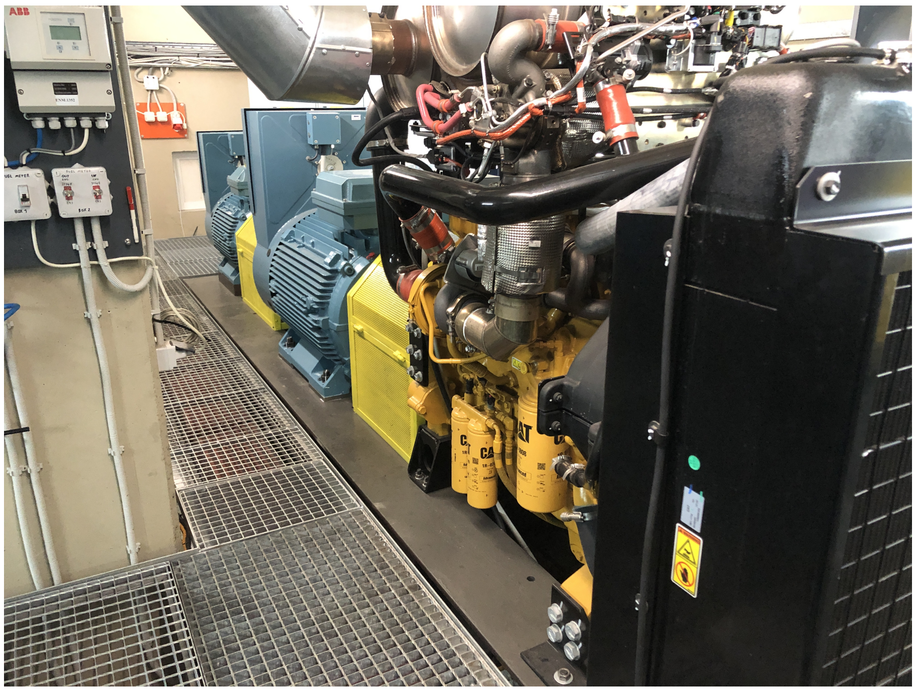


Figure 4.2: Experimental facility in LME showing the first sampling position (near the T/C)

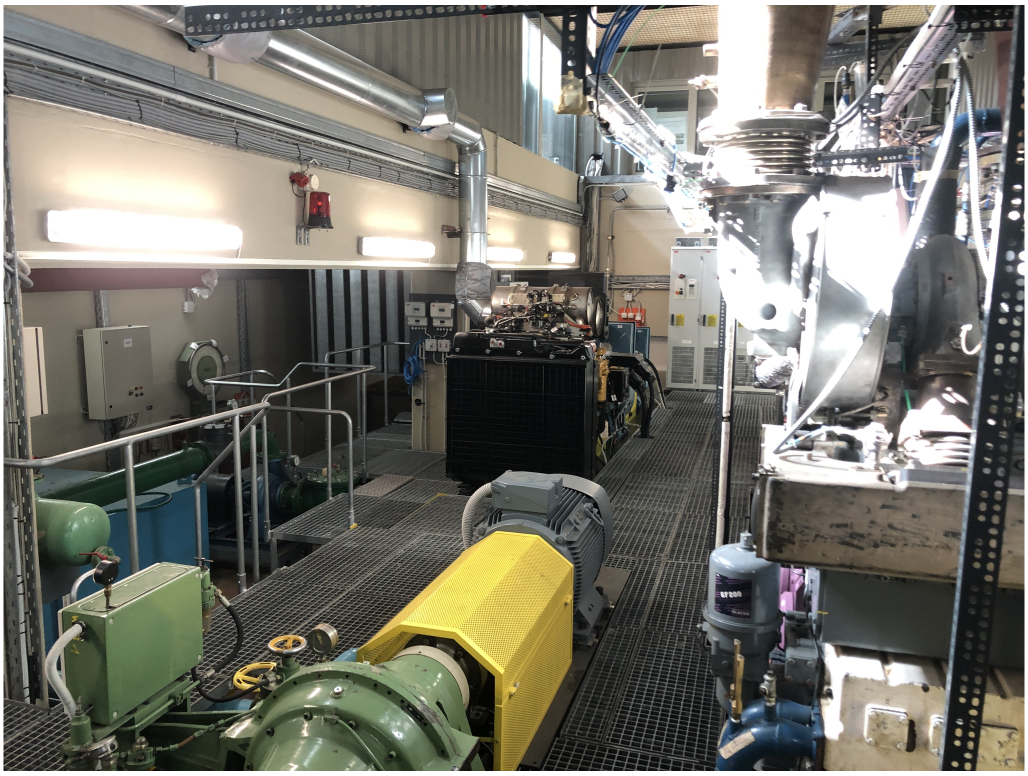


Figure 4.3: Experimental facility in LME showing the second sampling position.

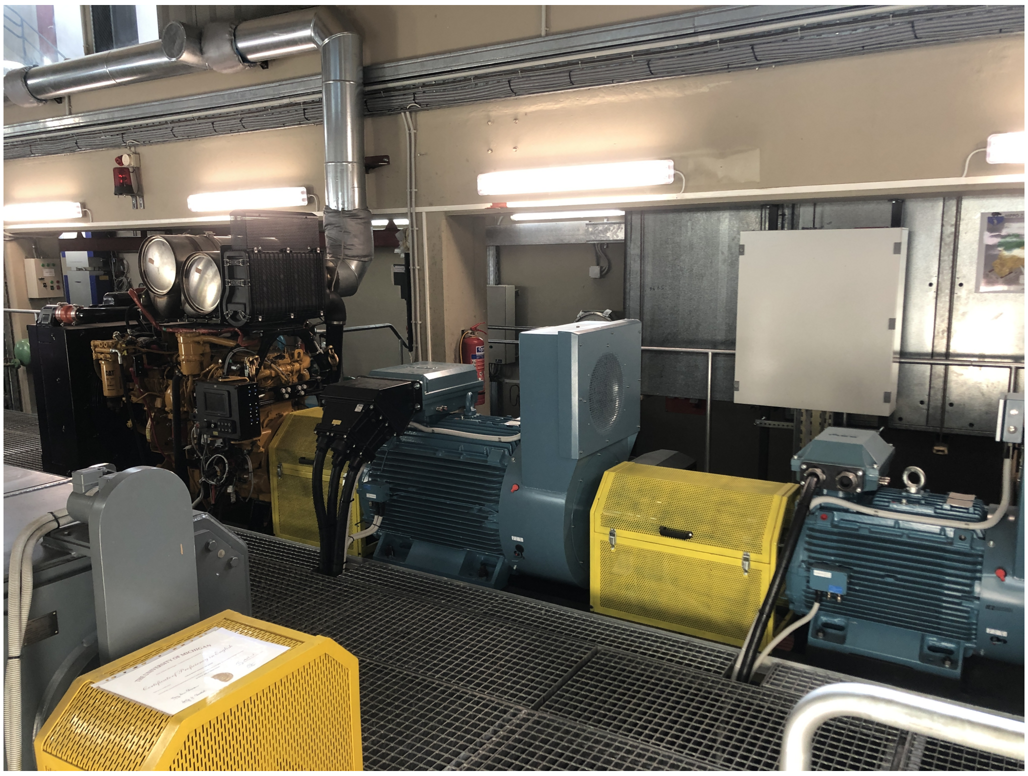
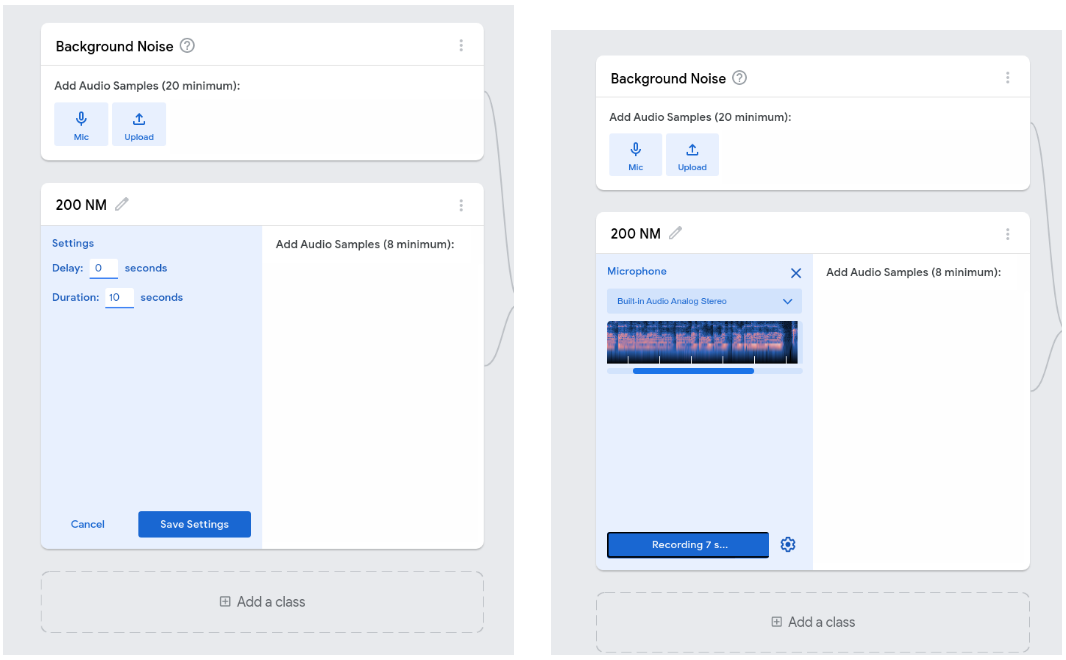


Figure 4.4: HIPPO-2 testbed at the experimental facility of LME

4.1.2 Data Labelling

Having performed all the above mentioned sound recordings, we started the creation of our dataset. Out of all the different recordings performed, we decided to work with two recordings with a total time of 180 seconds each, one for every position listed above. Teachable Machine allows the user to choose the duration of the sound recordings performed in the web browser as shown in Figure 4.5.


Background Noise

Add Audio Samples (20 minimum):

200 NM

Settings

Delay: 0 seconds

Duration: 10 seconds

Add Audio Samples (8 minimum):

Background Noise

Add Audio Samples (20 minimum):

200 NM

Microphone

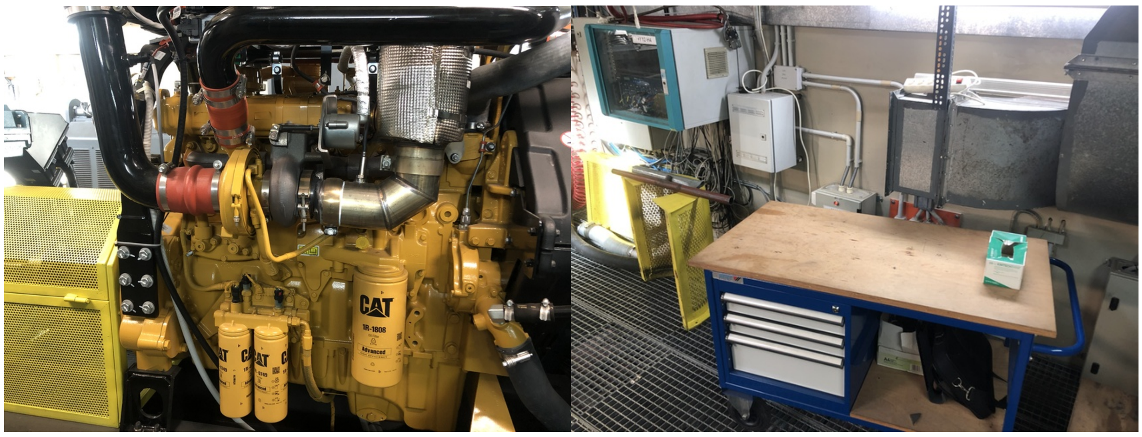
Built-in Audio Analog Stereo

Add Audio Samples (8 minimum):

Recording 7 s...

(a) (b)

Figure 4.5: Selection of *duration* and *delay*(left) and Recording procedure followed in TM web environment (right)



(a) (b)

Figure 4.6: Actual Sampling Positions

Regardless the users preference on recording duration, Teachable Machine "breaks" the sound in small audio snippets in a total extent of one(1) second. In order for TM to perform the above mentioned tasks, an internet connection is required. The recordings were performed as stated in the experimental facility area, were no internet connection could be established. Thus, having the sound recorded as shown in Figure 4.6, we played the recordings through the computers speakers, outside the facility grounds, were a stable internet connection was available.

The control and data acquisition system of the testbed, combined with the desired torque outputs as presented in Figure 4.1 played a key role in the labelling process. In fact, for every step performed, with the use of a thermodynamic model of the HIPPO-2 that was supplied to us by the LME staff, we were able to estimate the turbocharger speed for every recording that was taken, allowing us to classify the sound recordings accordingly. All different T/C speeds for every step performed with a +/- 10% variation are presented in the Table 4.1.

Table 4.1: Estimated T/C speed for various Torque Outputs

Step	**Torque Output (Nm)**	**Estimated Turbocharger Speed(rpm)**
1st	200	104.500
2nd	400	117.000
3rd	600	124.500
4th	800	129.500
5th	1000	133.000

Oftentimes, while creating a dataset it is very common to prepare the data before inputting them in the neural network. The creation of an audio dataset requires actions like filtering, denoising etc. In our case, Teachable Machine module, does not offer such operations. A way to tackle this problem was to apply a High Pass filter module, provided by p5.js library to our recorded sound before the classification procedure in Teachable Machine. However this action was finally removed from our project, due to the fact that it required the user to record the audio and then play each individual sound clip with the applied filter, making the final application a bit more difficult to handle. Simply to be put, we aimed to develop an application working in a web browser tab, that would only "listen" to whatever audio is presented by the user and immediately classify it accordingly.

4.2 Transfer Learning on Teachable Machine: Model Training

Having created our dataset, the next step would be to train our model. Considering the small size of our dataset, we decided to tweak Teachable Machine's parameters in order to get the best results. The basic differences between the cases we decided to test, were the number of epochs (see subsection 3.2.1) and the number of samples provided in each category. In the following paragraphs we present the training results, our web application final output and we conclude on the best fit for our project.

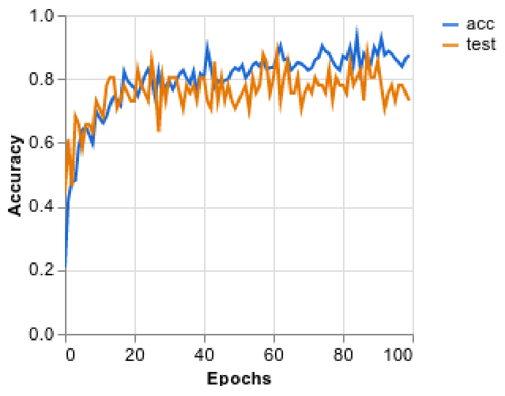
As presented in Table 4.2, we decided to present four(4) different training cases, with an alternating number of audio samples inputted during classification and number of epochs performed from every individual case. In the following sections, the results produced in every case are commented and presented.

Table 4.2: Training Cases

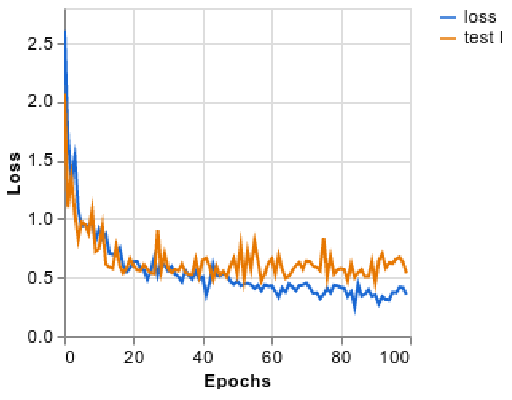
n/n	**Audio Samples**	**Epochs**
Case 1	50	100
Case 2	46	120
Case 3	20	100
Case 4	16	50

4.2.1 Training Case N^o1

For the first training case we decided to set the model for 100 epochs, which generally can be described as a big number for such tasks. However we wanted to try to understand the models performance after having processed the dataset for so many times.



(a)



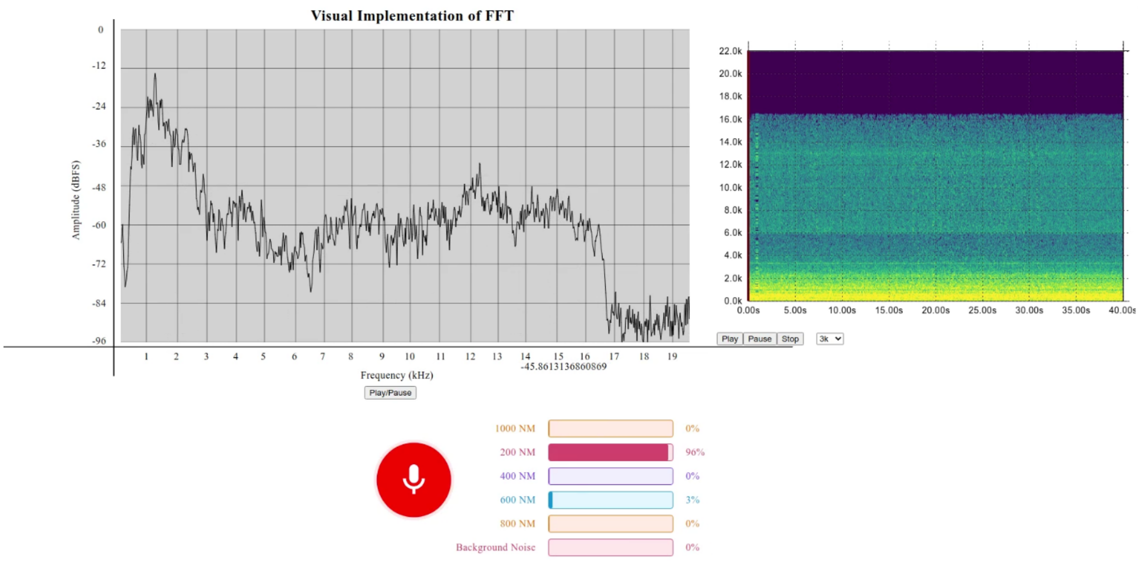
(b)

Figure 4.7: *Accuracy per epoch* (left) and *Loss per epoch* (right) diagrams as produced from Teachable Machine web environment

Table 4.3: Training with 100 epochs

Epoch	loss	acc	val_loss	val_acc
1/100	2.6134	0.2088	2.0730	0.4390
2/100	1.7161	0.4217	1.0997	0.6098
3/100	1.3355	0.4779	1.3626	0.4634
...	...	...	...	...
97/100	0.3717	0.8514	0.6597	0.7805
98/100	0.4191	0.8394	0.6765	0.7805
99/100	0.4151	0.8635	0.6283	0.7561
100/100	0.3554	0.8755	0.5391	0.7317

In Figure 4.7, *Accuracy per epochs* and *Loss per epochs* plots are presented. While observing the second diagram, combined with the results provided by Table 4.3, the model's loss appears to be decreasing after each epoch, suggesting that the model is "learning". By taking a closer look we can see that the validation function moves rather noisily around the training curve. One interpretation would be that validation data is scarce and not very representative of the training data, so the model struggles in evaluating these examples. In order to reach a conclusion, we would have to assess the performance and the behaviour of the curves produced in the other training cases. After the model has reached approximately 80 epochs, we can see that there is a relative rise of the validation function whereas the training function descends. This observation is really insightful, because it points us that after 80 epochs the model starts to overfit a little bit meaning that the algorithm captures well training data, but it performs poorly on new, so it's not able to generalize. In our case the extent of this phenomena is not concerning regarding the model's performance. In fact it only points us into the right direction for choosing the optimal number of iterations the model has to go through so to achieve high predicting scores.

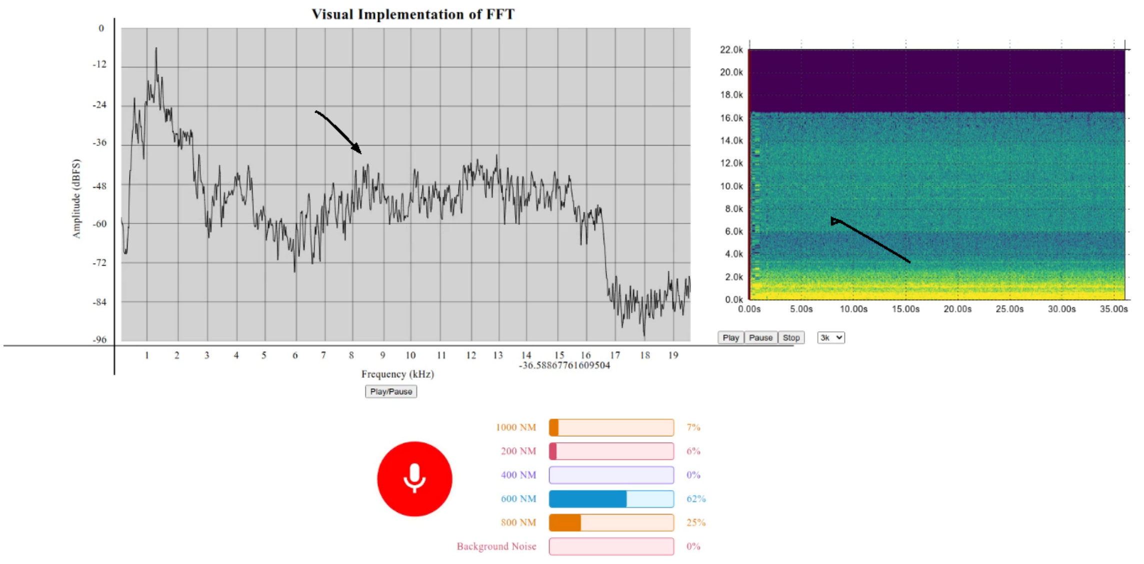


1000 NM	0%
200 NM	96%
400 NM	0%
600 NM	3%
800 NM	0%
Background Noise	0%

Figure 4.8: Highest prediction result achieved in 200 Nm required torque output. Screenshot taken from the created application web interface

Despite the promising results presented in the above mentioned diagrams, we had to provide a few snapshots from the actual web application. In Figure 4.8 we can see that the model can predict accurate results up to 96% while the demanded torque was **200 Nm**. Actually when the required torque output is low, it is rather difficult for the model to predict correctly. By observing the spectrogram created in our website we can see that there is a concentration of frequencies of high energy below 2kHz, making the model classification task quite difficult. Also in such low torque demands, the ICE's T/C is in low speeds, thus the sound produced by it requires much more accurate tools to capture. As mentioned before, the FFT Diagram and the Spectrogram were used in our web application as evaluation tools helping us comment on the integrity of our model's predicted results.

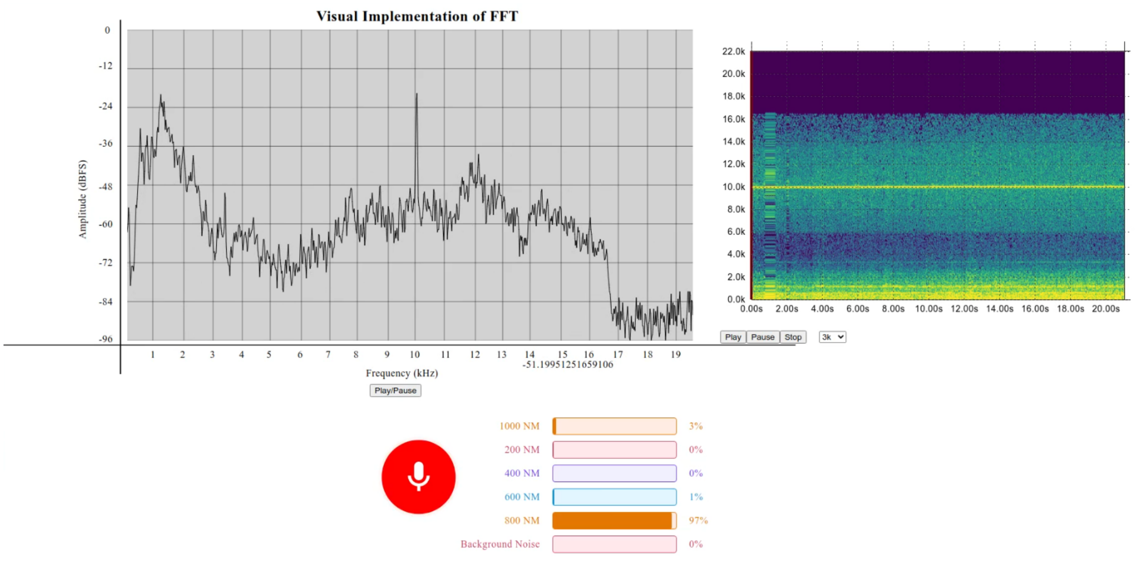
Moving on to higher torque demands, our model faced difficulties in some steps. To be more precise, in the process of classifying T/C speeds from **116.905 rpm** to **124.309 rpm** (2nd and 3rd step as per Table 4.1) our model had a really unstable behaviour providing poor results with a prediction accuracy ranging from 50% to 65% while sometimes mistakenly classifying the provided sound as a lower T/C speed. Especially when speeds around **124.309 rpm** are achieved, the only way to understand these changes is by closely observing the FFT diagram and the Spectrum as shown in Figure 4.9, were we can see a small peak in the FFT and light green line at around 9 kHz in both plots.



1000 NM	7%
200 NM	6%
400 NM	0%
600 NM	62%
800 NM	25%
Background Noise	0%

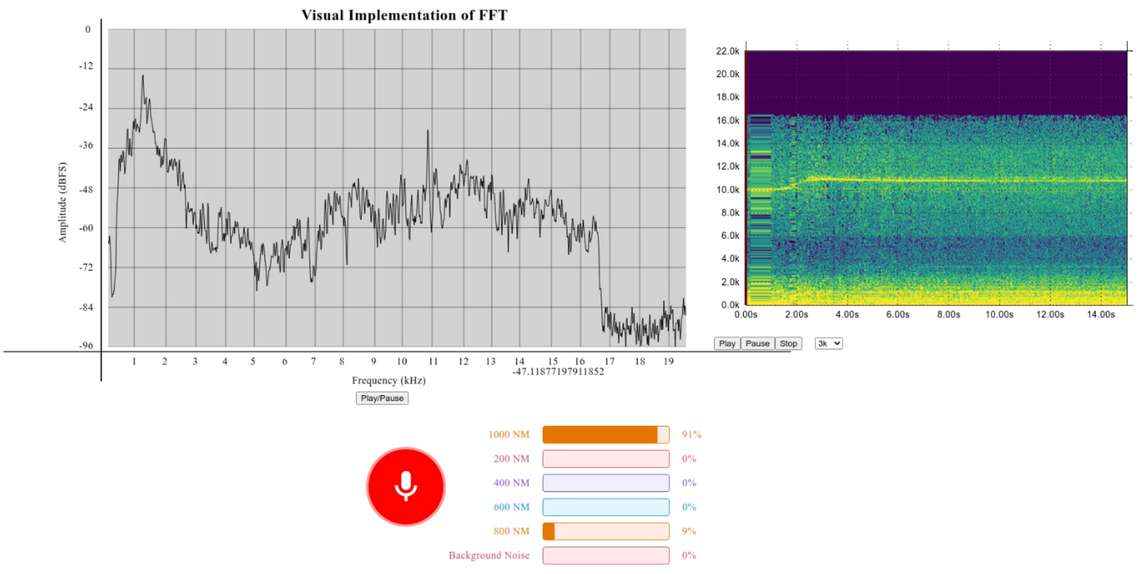
Figure 4.9: Low predicting results achieved at approximately 124.309 rpm (3rd Step). Screenshot taken from the created application web interface.

The predicting results of our model became more promising while the T/C's speed started to rise. The best model performance was achieved for speeds around **129.439 rpm** (4th Step). While observing the model behaviour in our web developed app, 97% prediction accuracy was achieved. When operating in such high speeds, as one can clearly observe in both plots, a distinctive high pitched sound is produced. To be precise, as depicted both in the FFT diagram and the spectrogram in Figure 4.10 this sound has a frequency of approximately 10 kHz. Finally, when the T/C reaches speeds up to **133.261 rpm**, our model performing very well, achieving predicting scores of around 91%. One main problem that we faced was the fact that the predictions appeared to have a few variations. In Figure 4.11, we present the best prediction achieved, while our model was listening to the sounds produced while executing the last step of the experiment. While observing the model's performance from our web application, we observed that sometimes the model falsely predicted that the T/C speed was at **129.439 rpm** instead of the actual **133.261 rpm**. Unfortunately we couldn't understand why this phenomenon occurred, presenting us with a challenge as discussed in the paragraph 5.2.



1000 NM	3%
200 NM	0%
400 NM	0%
600 NM	1%
800 NM	97%
Background Noise	0%

Figure 4.10: Model's predicting accuracy rose up to 97% for high T/C speeds (4th Step).



1000 NM	91%
200 NM	0%
400 NM	0%
600 NM	0%
800 NM	9%
Background Noise	0%

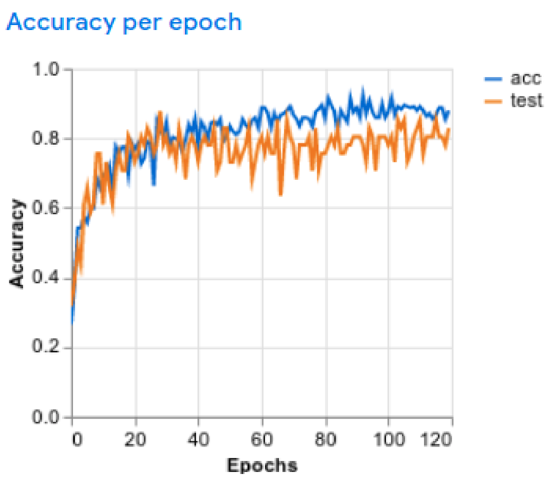
Figure 4.11: Similar behaviour for demanded torque output of 1000 Nm. Accurate prediction of around 91%.

4.2.2 Training Case N^o2

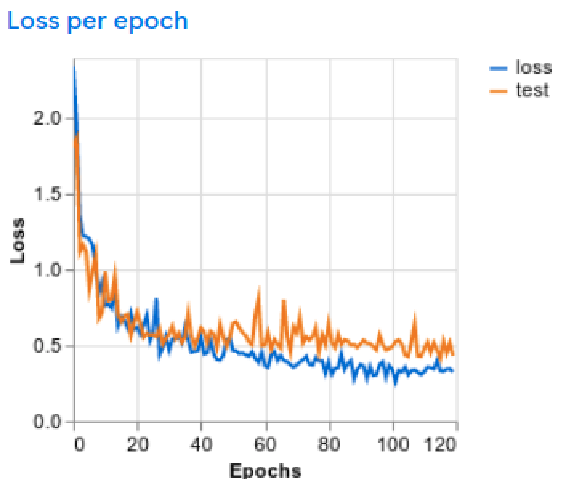
As for the second training case, given the good results produced from *Training Case* N^o1 we decided to slightly increase the number of epochs while decreasing the number of samples used for each individual class. So we decided to set the model for 120 epochs, with 46 samples for each one of the five(5) labels.

Table 4.4: Training with 120 epochs

Epoch	loss	acc	val_loss	val_acc
1/120	2.3456	0.2651	1.8131	0.3171
2/120	1.9442	0.3855	1.8641	0.3902
3/120	1.3606	0.5422	1.1164	0.4878
...	...	...	...	...
117/120	0.3302	0.8876	0.5303	0.8049
118/120	0.3438	0.8876	0.4492	0.8049
119/120	0.3458	0.8554	0.5176	0.7805
120/120	0.3260	0.8795	0.4319	0.8293



(a)



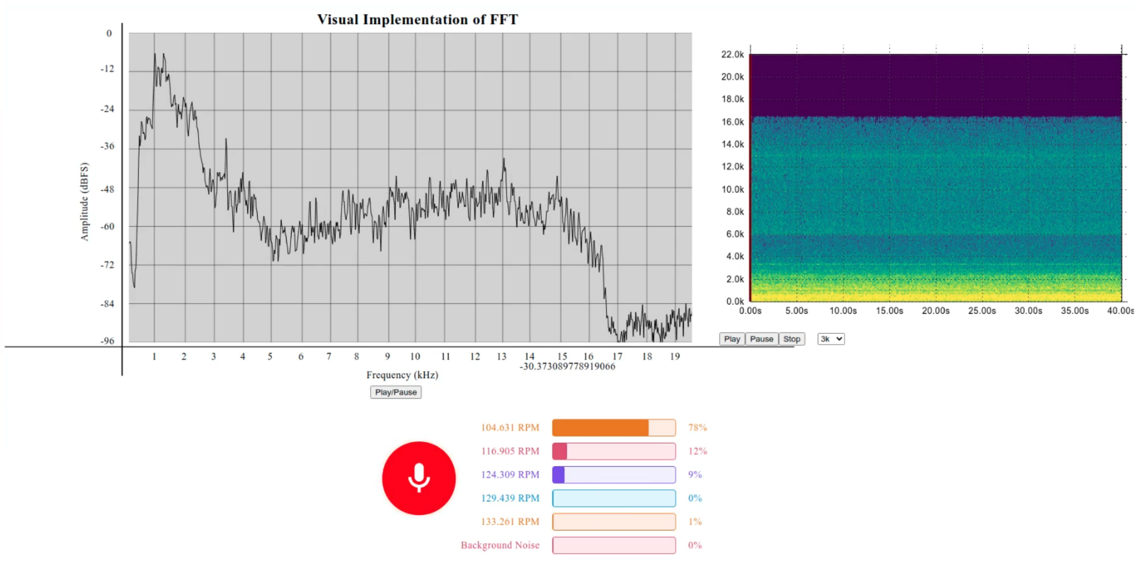
(b)

Figure 4.12: *Accuracy per epoch* (left) and *Loss per epoch* (right) diagrams as produced from Teachable Machine web environment for the 2^{nd} training case

One major concern, while increasing the number of epochs was if the model would start to overfit. In the first training case, we mentioned that around 80 epochs the model started to overfit a little bit. However in this case, not only better performance is achieved, but also the model seems to have minimized its overfitting behaviour as one can clearly observe in the *Loss per Epoch* plot presented in Figure 4.12. Also the validation function appears to move less noisily around our training curve meaning that the model produces more accurate predictions and it is able to generalize all examples given. Regarding the model's performance, as seen clearly in Table 4.4 we have reached an even smaller loss value compared to the first case resulting in an increased performance. Due to the fact that we don't have a better insight on how Google's Teachable Machine module operates, the only way to try and understand how it is functioning, is by trying many different training cases. In the context of this thesis, only a few cases out of all the different epoch and number of samples combinations performed are presented.

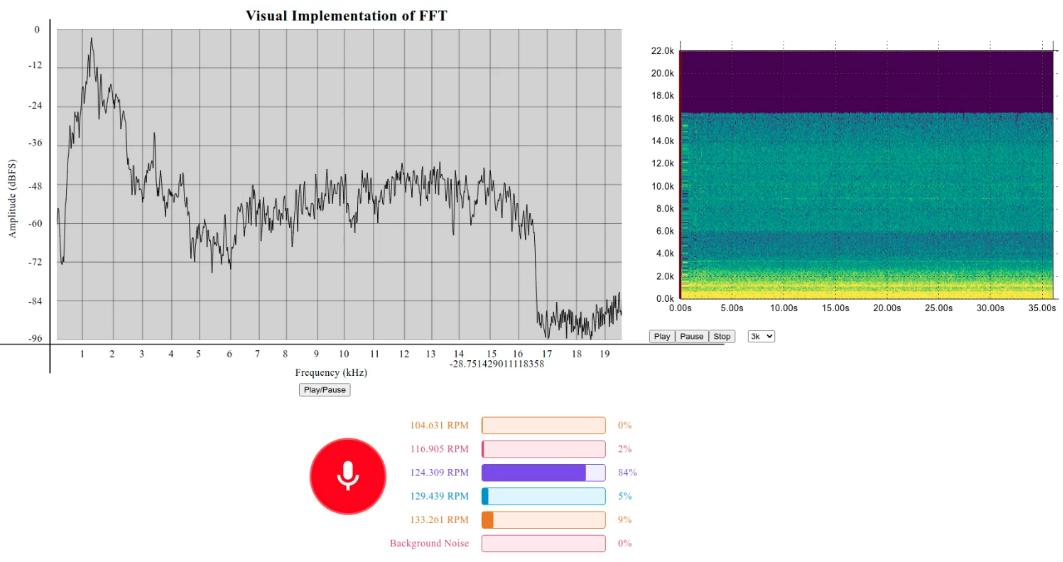
While the T/C was at low speeds around **104.631 rpm** (1st step), the maximum prediction achieved by the model was around 78% which overall can be described as an adequate prediction score for such low speeds considering that in the previous training case the equivalent score achieved was 98%. As mentioned in the previews paragraphs, in such operational circumstances, it is rather difficult for the model to produce not only accurate but also consistent results, due to the fact that there is significant concentration of frequencies in this specific area, as shown in Figure 4.13.

It was rather interesting to see, that while the T/C speed increased in order to achieve higher torque demands (2^{nd} and 3^{rd} step as per Figure 4.1) the model produced even better results, compared to the previews training case in the above mentioned steps. To be accurate, while the trying to classify T/C speeds from **116.905 rpm** and **124.309 rpm** the model produced some very promising results, reaching prediction scores up to 84% for the 3^{rd} step and 98% for the 2^{nd} step. It should be stated that in equivalent steps performed in *Training Case N^{o}1* we had some really poor results, with a lot fluctuations in the models behaviour and scores varying from 50% to 65% for the 2^{nd} step. To our surprise, while observing Figure 4.14, both the FFT and Spectrogram plots still produce some unclear for the naked eye results while classifying T/C speeds of **124.309 rpm** compared to the first training case, were we could at least see a small peak in the FFT and a light green line at around 9 kHz in both plots (see Figure 4.9). Our best guess would be that, due to the lowering of the number of samples provided at the classification stage and with the increase in the number of epochs, our model could fit the data better, resulting in more precise results.



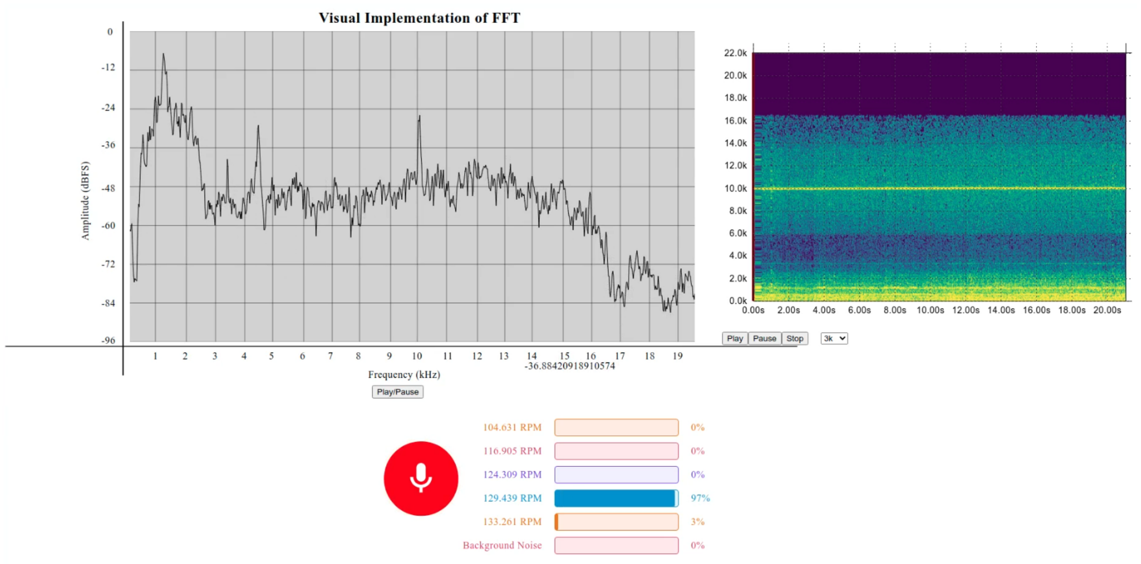
104.631 RPM 78%
116.905 RPM 12%
124.309 RPM 9%
129.439 RPM 0%
133.261 RPM 1%
Background Noise 0%

Figure 4.13: Model's predicting accuracy rose up to 78% for low T/C speeds (1^{st} Step).



104.631 RPM 0%
116.905 RPM 2%
124.309 RPM 84%
129.439 RPM 5%
133.261 RPM 9%
Background Noise 0%

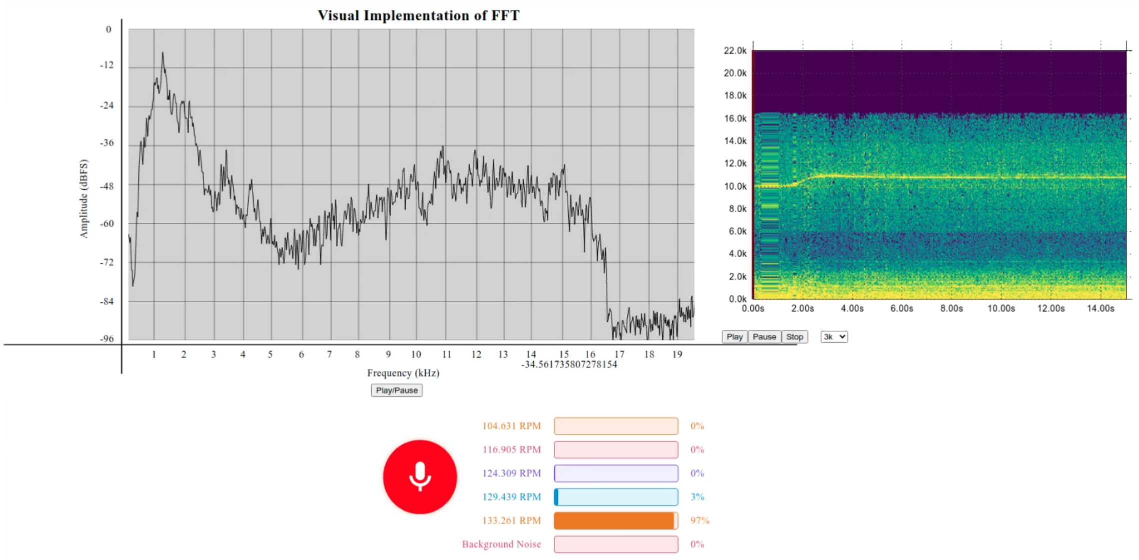
Figure 4.14: High predicting scores of around 84% at **124.309 rpm** (3^{rd} step) were achieved.


Visual Implementation of FFT
Amplitude (dBFS)
0
-12
-24
-36
-48
-60
-72
-84
-96
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
-36.88420918910574
Frequency (kHz)
Play/Pause
22.0k 20.0k 18.0k 16.0k 14.0k 12.0k 10.0k 8.0k 6.0k 4.0k 2.0k 0.0k
0.00s 2.00s 4.00s 6.00s 8.00s 10.00s 12.00s 14.00s 16.00s 18.00s 20.00s
Play Pause Stop 3k

104.631 RPM 0%
116.905 RPM 0%
124.309 RPM 0%
129.439 RPM 97%
133.261 RPM 3%
Background Noise 0%

Figure 4.15: Snapshot from the created web application, classifying **129.439 rpm** with scores up to 97%.

As we can clearly see in Figure 4.15 above, for higher torque demands around **800 Nm** with an achieved T/C speed of **129.439 rpm** the model classifies the sound presented with ease, with predicting scores reaching up to 97%. Having performed two different training cases so far, we can say that generally the model produced from Teachable Machine performs better in such operational circumstances, due to the fact that the ICE's turbocharger produces unique high pitched sounds, easily recognised not only from the model but also from the human ear. Finally, when the T/C reaches speeds up to **133.261 rpm**, as we can clearly see in Figure 4.16 presented above, the model provides us with some really accurate predicting scores. Both FFT and Spectrogram plots converge on the fact, that the T/C produces a continuous high pitched sound of 11 kHz. As mentioned before such high frequency sounds seem to be a very simple classification task for our model.



104.631 RPM	0%
116.905 RPM	0%
124.309 RPM	0%
129.439 RPM	3%
133.261 RPM	97%
Background Noise	0%

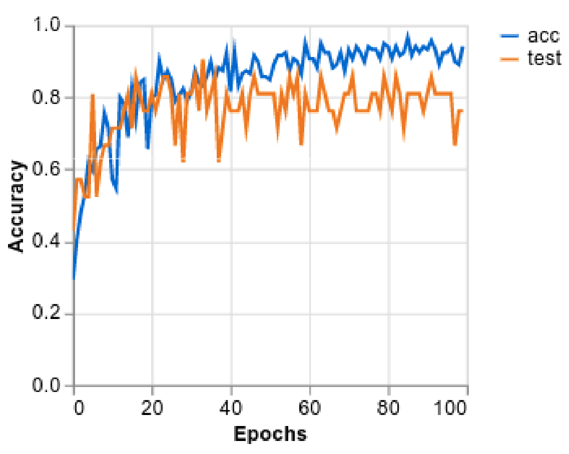
Figure 4.16: Snapshot from the created web application, classifying **133.261 rpm** with scores up to 97%.

4.2.3 Training Case $N^o 3$

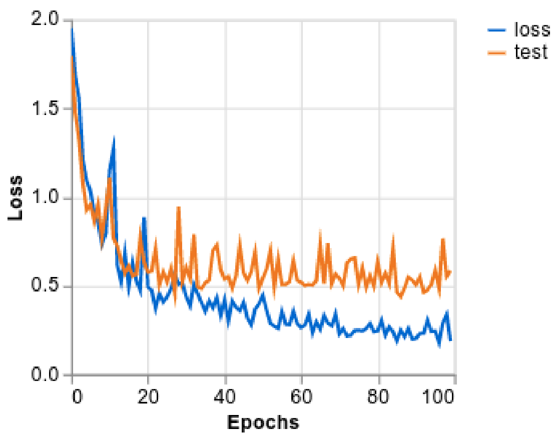
Having performed the above mentioned training cases, we though it would be of great interest to see how would the model respond with the same number of epochs, but with a significantly lower number of audio samples during the classification. To our surprise, by providing each label with just twenty(20) audio samples, high prediction scores were achieved for almost every different class of our model.

Table 4.5: Training with 100 epochs

Epoch	loss	acc	val_loss	val_acc
1/100	1.9525	0.2941	1.7935	0.4286
2/100	1.6885	0.4034	1.4772	0.5714
3/100	1.5589	0.4790	1.3201	0.5714
...	...	...	...	...
97/100	0.1795	0.9412	0.4890	0.8095
98/100	0.2910	0.8992	0.7664	0.6667
99/100	0.3358	0.8908	0.5520	0.7619
100/100	0.1864	0.9412	0.5840	0.7619



(a)

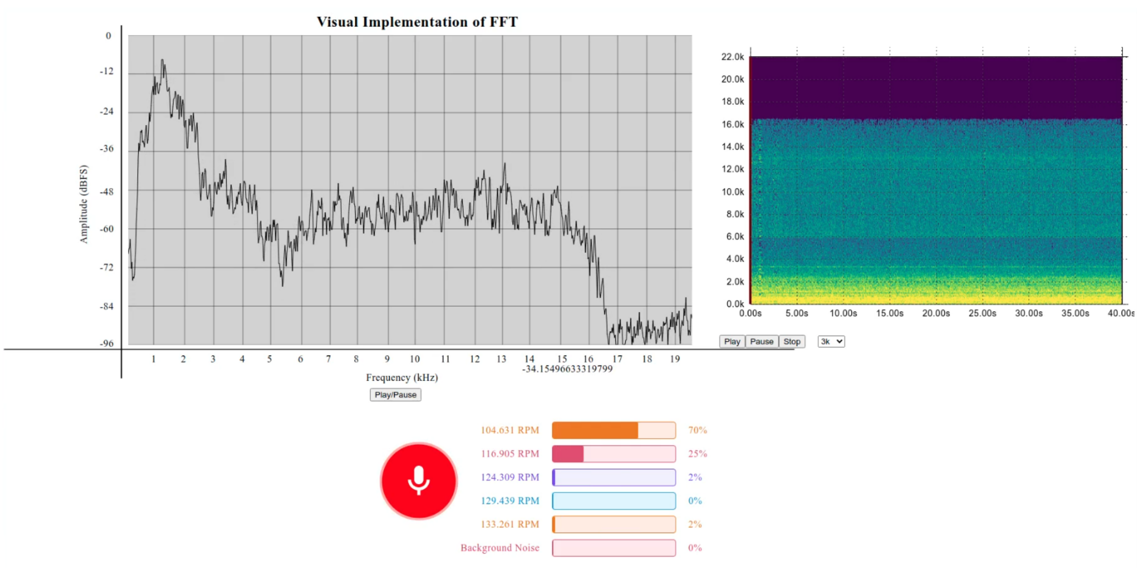


(b)

Figure 4.17: *Accuracy per epoch* (left) and *Loss per epoch* (right) diagrams for the 3^{rd} training case

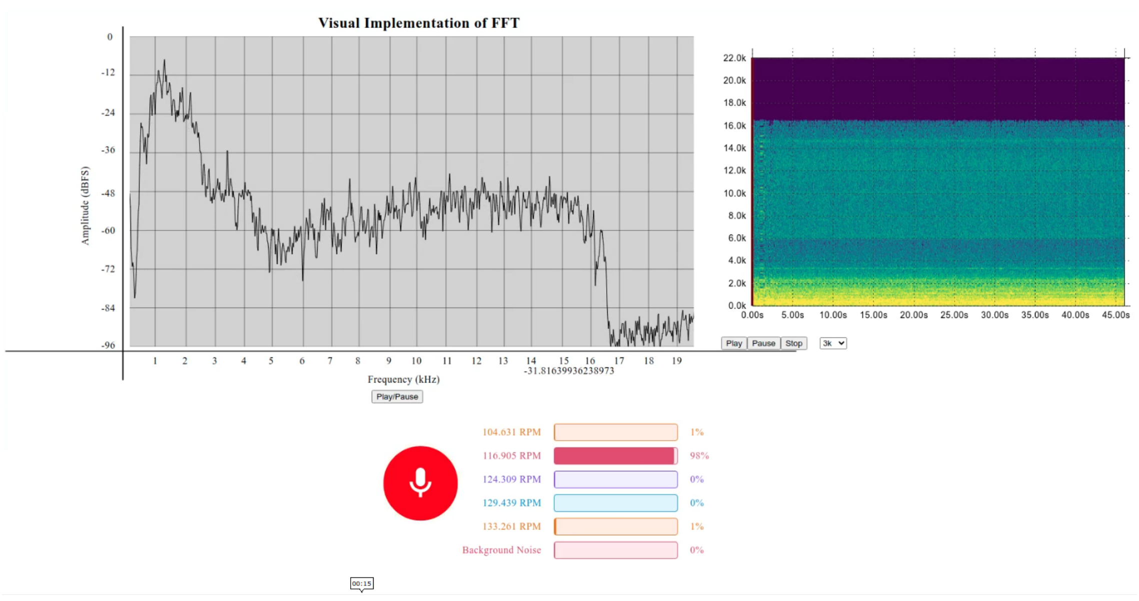
Compared to the other two training cases performed, here we achieved the lowest so far value for our loss function as shown in Table 4.5. In this case, we expect to get some really accurate prediction results from our model. Despite the really promising values produced for our loss function, while observing Figure 4.17 and especially the *Loss per Epoch* plot on the right, we can see that the model tends to overfit a little bit, as there seems to be a significant gap between the trends of the loss and the validation function. Actually, it is the biggest yet presented in the context of this thesis. In all three different training cases performed the validation function tends to move quite noisily. Our best guess would be that the dataset provided during the training in TM environment, needs to be a bit more representative for each and every different torque demand (see Figure 4.1) in order to help the model generalise its results more efficiently. However, to reach a conclusion for the overall performance of the model produced for this case, we should see how well the model predicts the changes in T/C speed, for each of the 5 different classes provided in the TM web interface, by providing to reader with a few snapshots from the created web application.

As per Figure 4.1, while the demanded torque is around **200 Nm**, the model's prediction scores were varying from 45% to 70%. At this stage, were the T/C has reached speeds up to **104.631 rpm**, our model predicts the lowest scores so far compared to the other training case, were the prediction scores reached up to 98%. However in this case, the results produced were very consistent, meaning that while the model was classifying the sound that the T/C made at such speeds, as we can see in Figure 4.18 below, the bar graph responsible for depicting the first step, was the only one to provide results. Given the fact that, at such low speeds it is rather difficult for our model to predict efficiently due to the high concentration of low frequency sounds (as depicted from the FFT and Spectrogram plot presented), we could say that this predicting results, are the best yet presented in this thesis, for such T/C speed/demanded torque.



104.631 RPM 70%
116.905 RPM 25%
124.309 RPM 2%
129.439 RPM 0%
133.261 RPM 2%
Background Noise 0%

Figure 4.18: At around **104.631 rpm** the model produced consistent and efficient results.

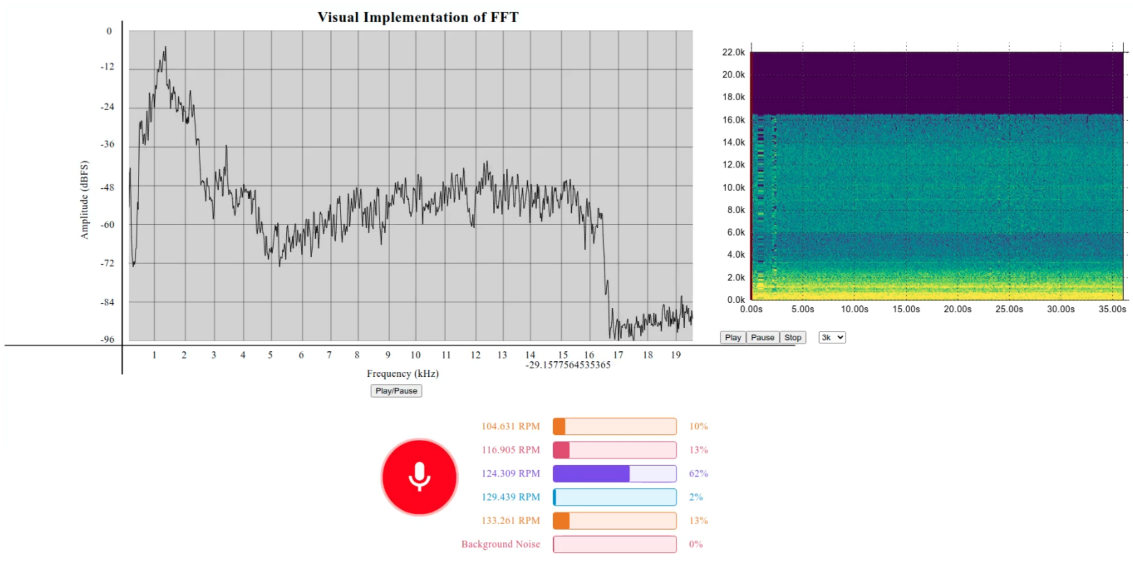


104.631 RPM 1%
116.905 RPM 98%
124.309 RPM 0%
129.439 RPM 0%
133.261 RPM 1%
Background Noise 0%

Figure 4.19: For T/C speeds up to **116.905 rpm**, the model prediction scores reached up to 98%.

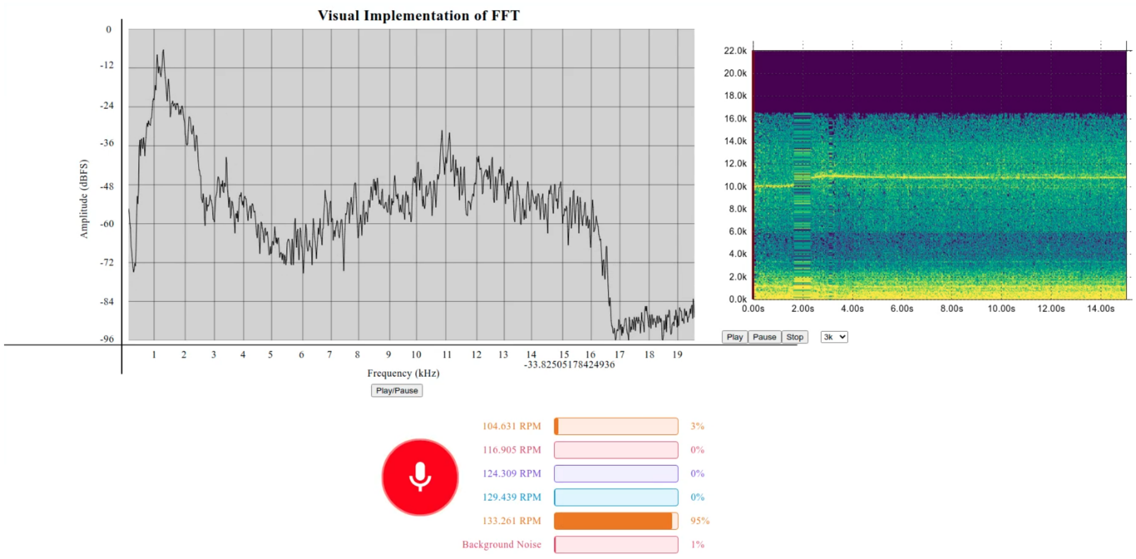
During the 2nd step as Figure 4.1 suggest, having reached T/C speeds up to **116.905 rpm** the model predicts scores with an accuracy over 64%. The highest predicted score was around 98%. Following the same strategy as in the second training case, with reduced number of samples and a relatively higher number of epochs, our model seems to perform quite well in these low T/C speed's. It appears that when there are less samples the model fits training data way more efficiently providing very good and consistent results.

Unfortunately moving on to higher torque demands, for T/C speed's of approximately **124.309 rpm** our model struggles to predict correctly the sounds produced by the ICE's turbocharger. The provided results were really poor and unstable, with the highest predicting score being around only 62% while sometimes providing us with mistaken classifications of higher T/C speeds. As mentioned before, during this step the depicted spectrogram as seen in Figure 4.20 does not have enough "information" to provide the model, making the process of classifying quite difficult.


Visual Implementation of FFT
Amplitude (dBFS)
0
-12
-24
-36
-48
-60
-72
-84
-96
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
-29.1577564535365
Frequency (kHz)
Play/Pause
22.0k 20.0k 18.0k 16.0k 14.0k 12.0k 10.0k 8.0k 6.0k 4.0k 2.0k 0.0k
0.00s 5.00s 10.00s 15.00s 20.00s 25.00s 30.00s 35.00s
Play Pause Stop
3k

104.631 RPM	10%
116.905 RPM	13%
124.309 RPM	62%
129.439 RPM	2%
133.261 RPM	13%
Background Noise	0%

Figure 4.20: For T/C speeds up to **124.309 rpm**, the model prediction scores reached only up to 62%.



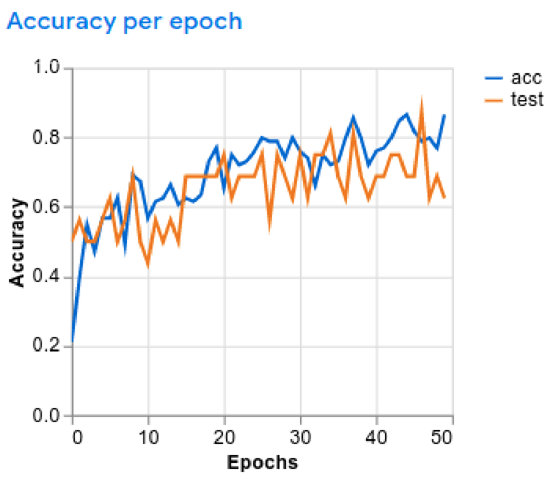
104.631 RPM	3%
116.905 RPM	0%
124.309 RPM	0%
129.439 RPM	0%
133.261 RPM	95%
Background Noise	1%

Figure 4.21: The model produced really accurate results for high T/C speeds. Snapshot from classification of **133.261 rpm**.

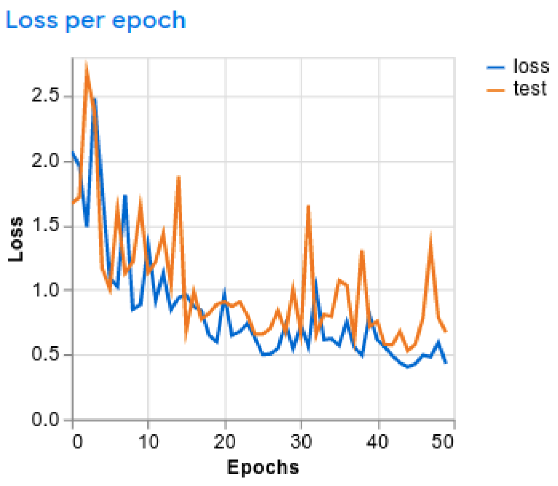
The model produced some very accurate results for the last two steps performed (4^{th} and 5^{th} **step**. To be precise, while the T/C speed was about **129.439 rpm** prediction scores up to 95% were achieved. As we observed in the previews training cases, while the demanded torque output is either **800 Nm** or **1000 Nm** the results produced were really accurate. As we can see from the produced spectrogram in Figure 4.21, this distinct high pitched noises allow the model to classify the sound easily producing efficient and consistent results.

4.2.4 Training Case N^o4

The results produced from performing the above mentioned training cases, gave us some insight on how one can create an efficient audio classification model with Google's Teachable Machine module. We tried different combinations of chosen number of epochs and audio samples provided in each category, with the ultimate goal being the creation of a model producing accurate and high predicting scores. We decided that our last attempt should follow the minimum requirements proposed from TM module so to create an audio model, meaning that we lowered the number of epochs to 50 and provided only sixteen(16) audio samples, only eight(8) above the minimum required to train the model. The results are presented and discussed in this section.



(a)



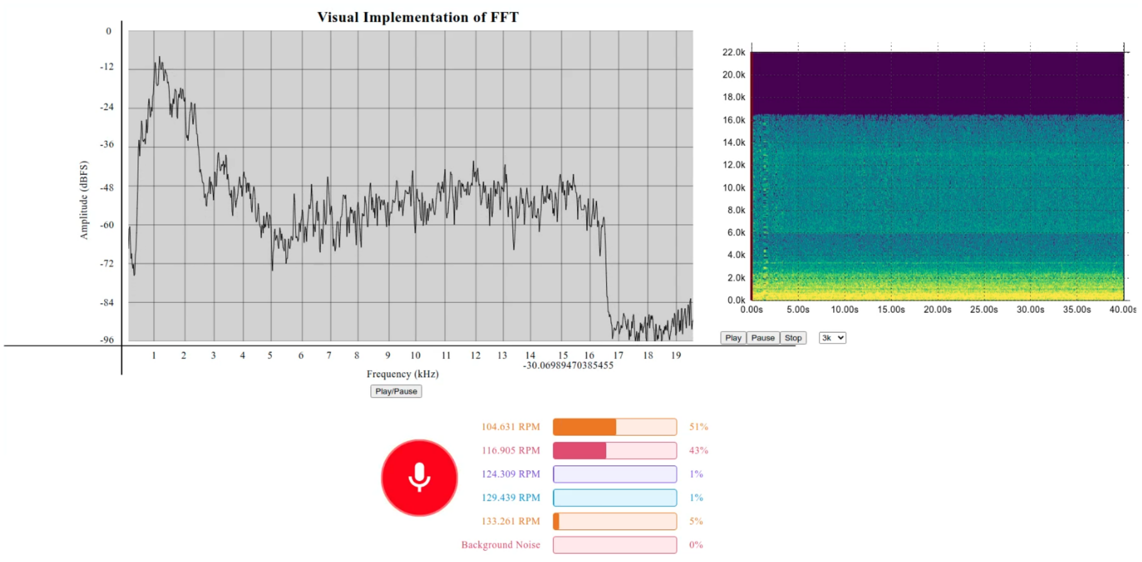
(b)

Figure 4.22: *Accuracy per epoch* (left) and *Loss per epoch* (right) diagrams for the 4^{th} training case

By taking a closer look in the plots provided by Teachable Machine in Figure 4.22, the validation trends both in *Accuracy per epoch* and *Loss per epoch* plot move noisily around the training trend. We believe that this phenomenon occurs due to the fact that, we provided our model with a small number of samples deeming the dataset unrepresentative for the task performed. Moreover we can see that the model performance is growing over time, which means the model despite the poor dataset provided is improving with experience (it's learning). We also see it grows at the beginning, but over time around 35^{th} epoch it reaches a plateau, meaning it's not able to learn any more. Compared to the preview training case, the one examined here has produced the highest loss function values. We would expect the model produced would perform poorly while trying to classify new audio samples presented.

Table 4.6: Training with 50 epochs

Epoch	loss	acc	val_loss	val_acc
1/50	2.0698	0.2115	1.6668	0.5000
2/50	1.9605	0.3942	1.7160	0.5625
3/50	1.4859	0.5481	2.6770	0.5000
...	...	...	...	...
47/50	0.4960	0.7885	0.7865	0.8750
48/50	0.4819	0.7981	1.3392	0.6250
49/50	0.5945	0.7692	0.7861	0.6875
50/50	0.4292	0.8654	0.6730	0.6250



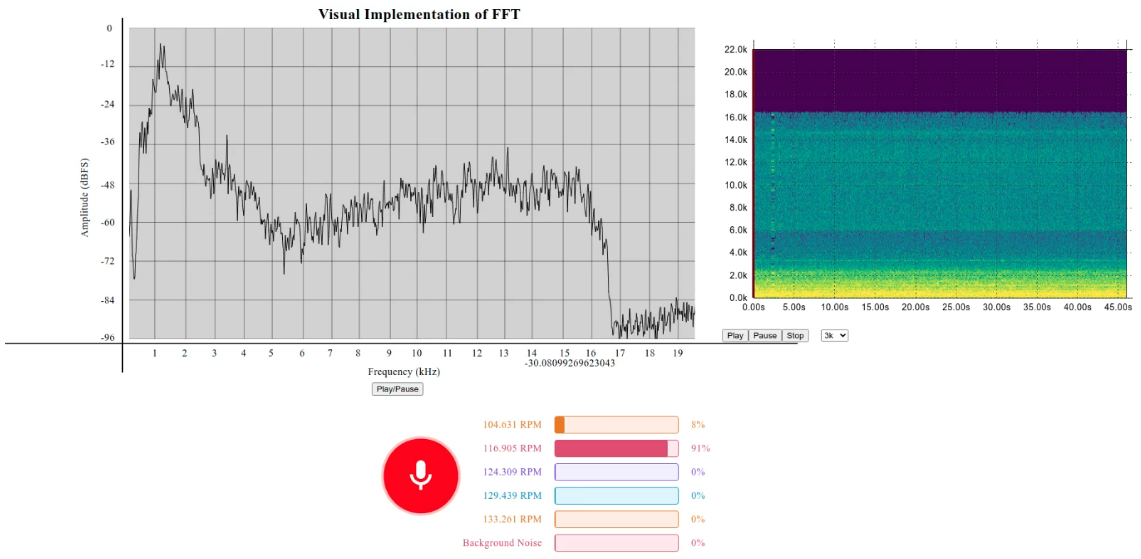
```
104.631 RPM        51%
116.905 RPM        43%
124.309 RPM        1%
129.439 RPM        1%
133.261 RPM        5%
Background Noise   0%
```

Figure 4.23: As expected, for low T/C speed, approx. **104.631 rpm**, the model is unable to predict correctly.

As expected from the above listed readings from Table 4.6 and the above presented figures, the model is struggling to classify correctly sound produced when the ICE's T/C is at low speeds (approx. **104.631 rpm**). In all training cases we observed that the model produced poor results in these stages, so we can expect that in a such poorly trained model the results would be uncertain. As shown in Figure 4.23, it is rather difficult for the model to produce correct predictions and even sometimes it classifies mistakenly wrong labels. For the record the best achieved performance was about 78% but since the model's behaviour during this step is unstable, therefore we decided it should not be included in our analysis.

Moving on to torque demands up to **400 Nm** (2^{nd} Step as per Figure 4.1) the model surprisingly presented a really stable behaviour, producing very accurate and consistent results. The predicting scores achieved were ranging from 54% up to 85% with the best recorded predicting score reaching 91% as seen in the snapshot provided from our web application in Figure 4.24.

Unfortunately, due to the lack of information on how Teachable Machine operates, we can't be sure why the model appears to have a more stable behaviour for such small changes of T/C speeds (In the 1st Step the speed is at approx **104.631 rpm** whereas in the 2nd Step is at about **116.905 rpm**).

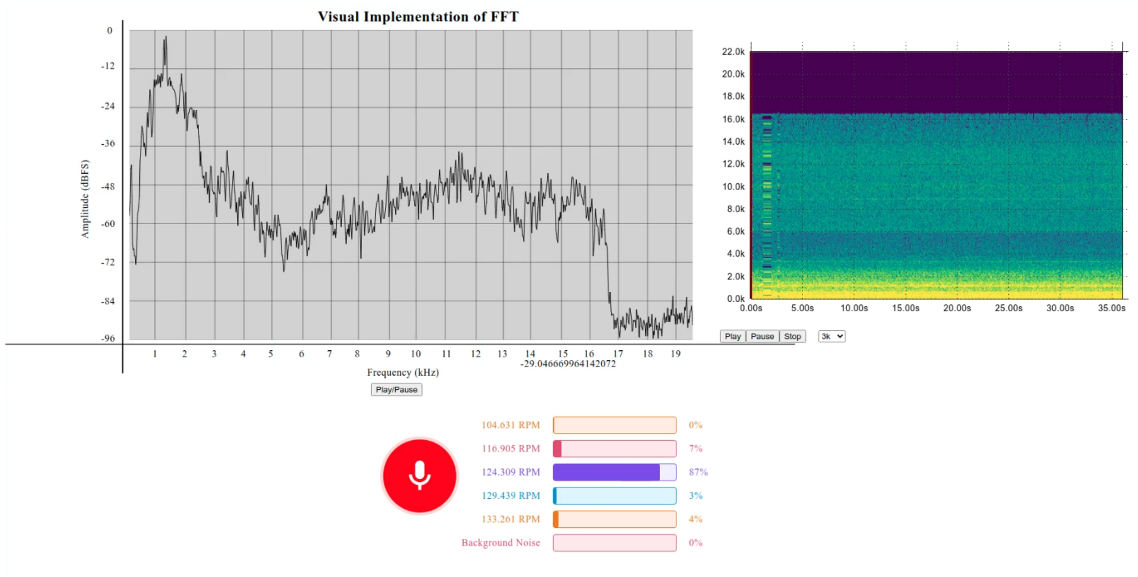


104.631 RPM	8%
116.905 RPM	91%
124.309 RPM	0%
129.439 RPM	0%
133.261 RPM	0%
Background Noise	0%

Figure 4.24: A much more stable behaviour was recorded while the model was classifying T/C of approx. **116.905 rpm**.

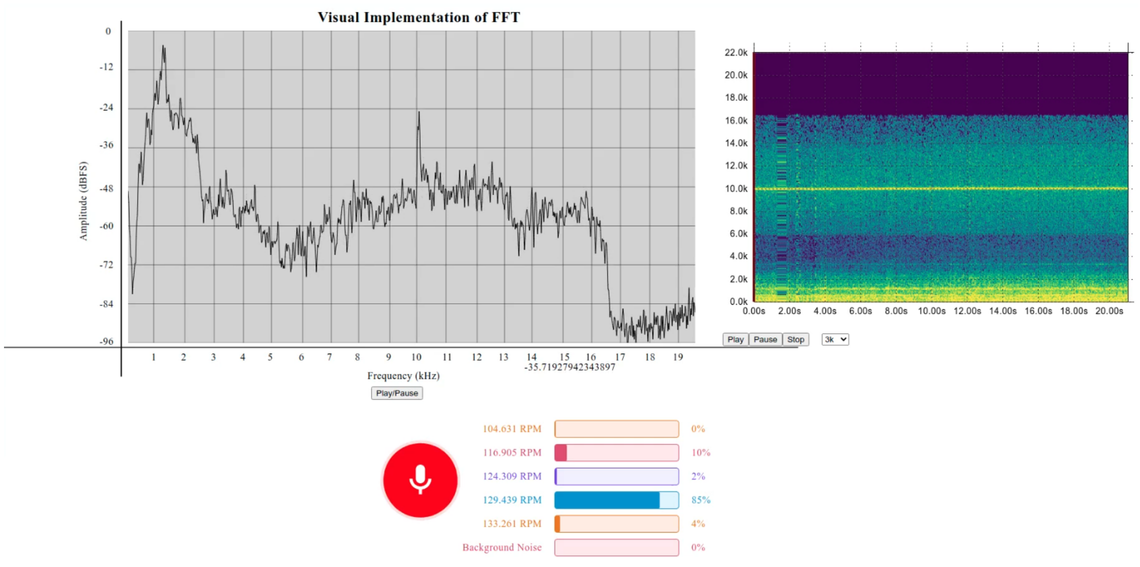
The model predicted some really accurate results while trying to classify T/C speeds up to **124.309 rpm**. Interestingly, the scores varied from 54% reaching up to 85% with the highest recorded value being equal to 87%. In order to understand the reason why the model predicts such results, we should at least take a closer look at the tools presented in the created web interface in Figure 4.25. Having performed all the above mentioned training cases, we can conclude that generally the model created in the Teachable Machine web environment, seems to classify really well sounds produced while the demanded torque is about **600 Nm**, despite the fact the two plots presented in our webpage seem to struggle in depicting the frequency changes. The only thing we can see is a barely distinguishable line at around 9 kHz in the spectrogram plot, whereas in the FFT diagram the results depicted are uncertain.

The same observation made while commenting on the results produced from the previews training case, applies also in this final training case studied. While the T/C operates under heavy loads, reaching speeds from **129.439 rpm** up to **133.261 rpm** the model has a very stable behaviour, producing really consistent and high predicting scores up to 98%, as depicted in Figure 4.26 below, were we can also see that all tools presented in our webpage seem to converge on the results produced by the model.



104.631 RPM	0%
116.905 RPM	7%
124.309 RPM	87%
129.439 RPM	3%
133.261 RPM	4%
Background Noise	0%

Figure 4.25: High predicting scores up to 87% were produced while the T/C speed's were around **124.309 rpm**.



104.631 RPM	0%
116.905 RPM	10%
124.309 RPM	2%
129.439 RPM	85%
133.261 RPM	4%
Background Noise	0%

Figure 4.26: High predicting scores up to 85% were produced while the T/C speed's were around **129.439 rpm**.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, the implementation of an audio classification application using neural networks to monitor the changes in T/C speed of the HIPPO II testbed was investigated. Having performed a detailed research regarding similar projects, a very convenient and user friendly module was selected for the development state and by creating several different training scenarios, a comparing experiment was conducted to find out, the most suitable combination of parameters in order to perform our task in the most efficient way possible. We choose as the most appropriate technique for our experiment to be conducted, the *Transfer Learning* method, in which we used Google's *Teachable Machine* to create an audio recognition model with our own custom dataset.

JavaScript and Teachable Machine

Research on similar projects like the one presented in our thesis, shows that JavaScript is the less preferred programming language to implement machine learning tasks, as also mentioned in the previews chapters. We had the privilege to use Google's *Teachable Machine* module, which allowed us to perform some computationally intensive model training simply on our browser tab. Surely, if we were to perform such tasks on a personal laptop, we would have to use some backend programming language like Python and C++. Unfortunately, TM didn't allow the user to interfere with the model architecture, the activation functions used and all other hyper parameters that we could probably tweak in order to produce even better results. However, we can confirm that it is plausible to recognise with significant accuracy the changes in the T/C speed of a ship's Main Engine, with the assistance of an audio classification model.

Results regarding the model's performance

To sum up, for the needs of this project, by simply controlling the demanded torque with the help of the Electric Motor of the HIPPO II testbed, we compartmentalized five(5) different steps in order to investigate how accurately would our audio classification model recognise the changes in the ICE's T/C speed. The results are provided in Figure 5.1 below.

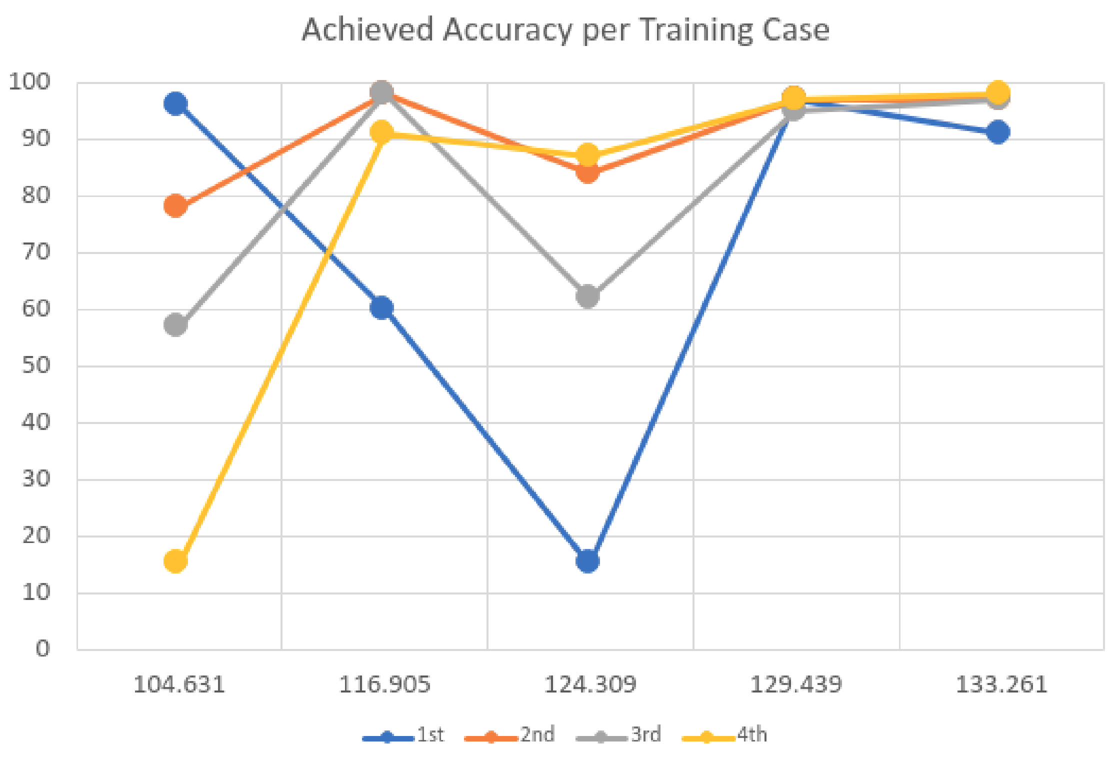


Figure 5.1: Best prediction scores achieved in all training cases performed, for different T/C Speeds.

From the above plot provided, we can conclude that the 2^{nd} training case, has provided us with the highest predicting scores out of all different cases performed during this thesis. Thus, the model provided from this case will be used for our final web application.

Performing transfer learning in Teachable Machine, proved to be a simple task. The fact that this module is directly connected with TensorFlow.js, allows the user to perform machine learning tasks efficiently and with almost no programming expertise required. At this point is should be stated that we encountered a few difficulties due to the nature of our task. Teachable Machine does not allow the user, modify the produced model's architecture or the hyperparameters of it, making it really difficult for us to interfere with it and possibly provide some solutions on how to get more efficient results. Certainly a more efficient method would be to create our own custom model, even in another programming language i.e Python, based on TM produced audio model architecture.

5.2 Future Work

The inspiration for this thesis project, was the lack of tool to provide us with live and accurate data on the current speed of the ICE's turbocharger in the HIPPO II testbed. We believe that, the results presented in this thesis prove that it is plausible to create such a tool by implementing machine learning techniques. Moreover, several improvements can be made to further enhance the next steps and ultimately to achieve this project's goals. Machine Learning and especially deep neural networks, have proven to be really promising fields with limitless capabilities. In the next paragraphs, the suggested future work is presented.

Data Augmentation

The dataset used to train our model in Teachable Machine UI, consisted of only two sound recording of 180 seconds each, at two different location in the experimental facility. It should be stated that some of the most commonly used datasets for audio classification tasks like the Urban8k dataset, contain more than 8700 labeled sound snippets with a duration around 4 seconds each.[48] In order to solve problems such as overfitting and improve generalization, would be to create an efficient audio dataset, comprising of several sound snippets gathered from several T/C at various speeds, allowing our model to recognise the changes in T/C speeds in all types of engines. A key concept of data augmentation is that the deformations applied to the labelled data do not change the semantic meaning of the labels. By training the network on the additional deformed data, the hope is that the network becomes invariant to these deformations and generalizes better to unseen data.[49] Some of the most common data augmentation techniques applied in raw audio are time stretching, pitch sifting, background noise addition etc.[50]

Enhanced equipment and User Interface

For the creation of our dataset as mentioned in subsection 4.1.2, all audio was captured by the computer's built in microphone, meaning that there is a lot of room for improvement in ways of recording the sound. With the use of an LDC short for Large Diaphragm Condenser microphone, which are very sensitive making them ideal for extremely dynamic sources. Probably by using also a windscreen, we can even reduce the background noise, resulting in a much more accurate recording.

A few enhancements can also be made in the UI of our web application. A more meticulous presentation of our tools could be provided, giving the user the opportunity to zoom in the plots and take a closer looks to the results provided. It would be really useful to provide the user with the option, to record himself an audio sample, that would be then processed from us, helping us to create an even better dataset.

Possible Applications

The main concept of this thesis would be to monitor the changes in the T/C speed of the HIPPO II testbed. Granted that the model is able to produce accurate results, the application could be used for monitoring the main engine's health and detect problems like T/C surging in low loads. Moving on into a more complete application even a mobile app could be provided, were the user only with the use the device's microphone, would record a sound snippet and the app would classify accordingly providing results on the current T/C speed, abnormal operating behaviour like surging, worn bearings causing irregular sounds etc.

Appendices

Appendix A

Model Creation Pipeline

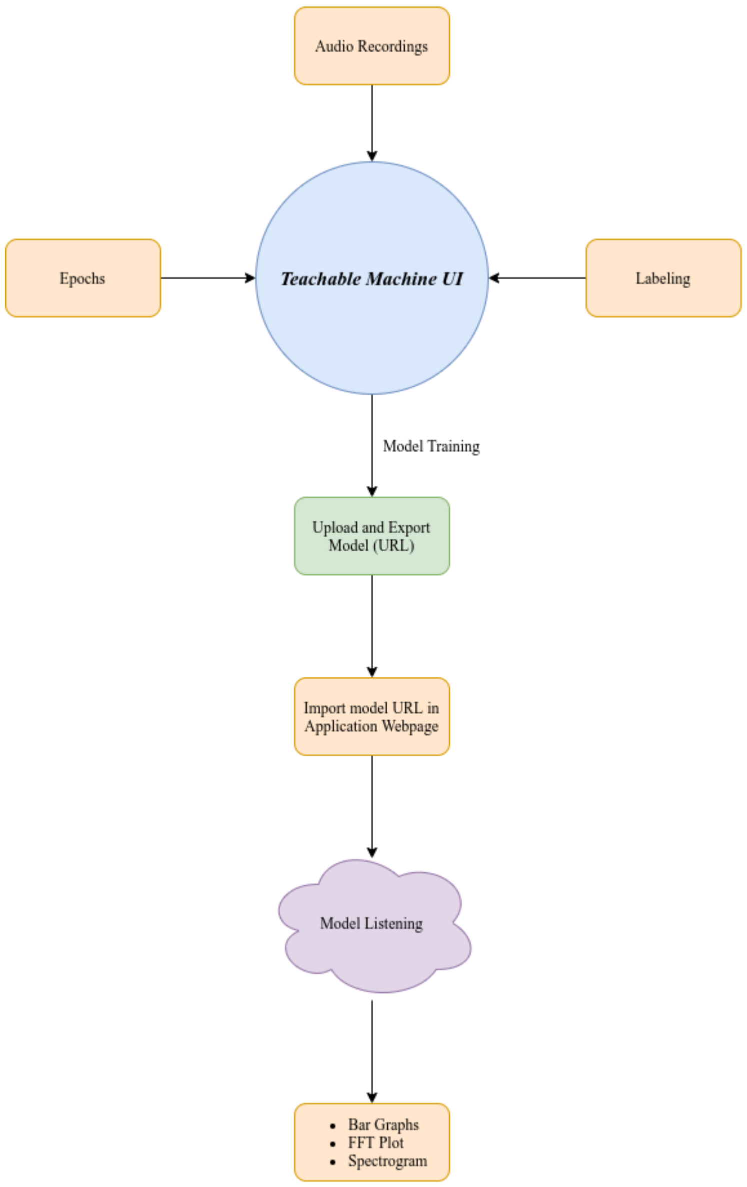


Figure A.1: Code pipeline presenting the creation of the model up until inputing it in our developed web application.

Bibliography

[1] IBM Cloud Education. AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference? `https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks`, 2020.

[2] Shanqing Cai, Stanley Bileschi, Eric D. Nielsen, and François Chollet. *Deep Learning with JavaScript: Neural Networks in Tensorflow.js*. Manning Publications Co., 2020.

[3] Pedro Fernández-Cabán, Forrest Masters, and Brian Phillips. Predicting roof pressures on a low-rise structure from freestream turbulence using artificial neural networks. *Frontiers in Built Environment*, 2018.

[4] Devraj Agarwal. A Review of the Math Used in Training a Neural Network. `https://levelup.gitconnected.com/a-review-of-the-math-used-in-training-a-neural-network-9b9d5838f272`, 2020.

[5] Fei-Fei Li, Ranjay Krishna, and Danfei Xu. CS231n: Convolutional Neural Networks for Visual Recognition. `http://cs231n.stanford.edu/`, 2021.

[6] Olive Zhao. Machine learning training method: Gradient descent method. `https://forum.huawei.com/enterprise/en/machine-learning-training-method-gradient-descent-method/thread/708303-895?page=1&authorid=2976461`, 2021.

[7] Stefanos Georgis. Position estimation of multiple sea vessels using a stereo-based camera system and neural networks. Master's thesis, NTUA, 2020.

[8] Sachdeva Aashay. Deep Learning for Computer Vision for the average person. `https://medium.com/diaryofawannapreneur/deep-learning-for-computer-vision-for-the-average-person-861661d8aa61`, 2017.

[9] Phillip Scholze. Complete Guide to Gradient-Based Optimizers in Deep Learning. `https://cyanite.ai/2020/09/30/the-4-essential-steps-for-analyzing-music-with-neural-networks/`, 2020.

[10] Zhao Guyu, Guoyan Huang, Hongdou He, Haitao He, and Jiadong Ren. Regional spatiotemporal collaborative prediction model for air quality. *IEEE Access*, 2018.

[11] Sinam Ajitkumar Singh and Swanirbhar Majumder. Chapter one - short and noisy electrocardiogram classification based on deep learning. In Himansu Das, Chittaranjan Pradhan, and Nilanjan Dey, editors, *Deep Learning for Data Analytics*, pages 1–19. Academic Press, 2020.

[12] Oisin Mac Aodha, Rory Gibb, Kate E. Barlow, Ella Browning, Michael Firman, Robin Freeman, Briana Harder, Libby Kinsey, Gary R. Mead, Stuart E. Newson, Ivan Pandourski, Stuart Parsons, Jon Russ, Abigel Szodoray-Paradi, Farkas Szodoray-Paradi, Elena Tilova, Mark Girolami, Gabriel Brostow, and Kate E. Jones. Bat detective—deep learning tools for bat acoustic signal detection. *PLOS Computational Biology*, 2018.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

[14] Teachable Machine Audio Project. `https://teachablemachine.withgoogle.com/train/audio`.

[15] Steven L. Brunton and J. Nathan Kutz. *Fourier and Wavelet Transforms*. Cambridge University Press, 2019.

[16] Nikolaos Planakis, George Papalambrou, and Nikolaos Kyrtatos. Predictive power-split system of hybrid ship propulsion for energy management and emissions reduction. *Control Engineering Practice*, 111:104795, 06 2021.

[17] Roman Varbanets, Oleksij Fomin, Václav Píštěk, Valentyn Klymenko, Dmytro Minchev, Alexander Khrulev, Vitalii Zalozh, and Pavel Kučera. Acoustic method for estimation of marine low-speed engine turbocharger parameters. *Journal of Marine Science and Engineering*, 9(3), 2021.

[18] Nikolaos Planakis, Vasileios Karystinos, George Papalambrou, and Nikolaos Kyrtatos. Nonlinear model predictive control for the transient load share management of a hybrid diesel-electric marine propulsion plant. In *2020 American Control Conference (ACC)*, pages 1955–1960, 2020.

[19] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[20] Matthieu Sainlez and Georges Heyen. Recurrent neural network prediction of steam production in a kraft recovery boiler. `https://www.sciencedirect.com/science/article/pii/B9780444542984501355`, 2011.

[21] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, 2006.

[22] Anidhya Athaiya Siddharth Sharma, Simone Sharma. *Activation Functions in Neural Networks*. International Journal of Engineering Applied Sciences and Technology, Vol.4, 2020.

[23] J. Brownlee. *Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions*. Machine Learning Mastery, 2018.

[24] Nikolaos Chatzistamatiou. Development of an object detection application that monitors vessel traffic, using neural networks. Master's thesis, NTUA, 2021.

[25] Chirag Goyal. The 4 essential steps for analyzing music with neural networks. `https://www.analyticsvidhya.com/blog/2021/06/complete-guide-to-gradient-based-optimizers/`, 2021.

[26] Sebastian Ruder. An overview of gradient descent optimization algorithms. `https://ruder.io/optimizing-gradient-descent/`, 2016.

[27] Derrick Mwiti. Convolutional neural networks: An intro tutorial. `https://heartbeat.comet.ml/a-beginners-guide-to-convolutional-neural-networks-cnn-cf26c5ee17ed`, 2018.

[28] Donges Niklas. What Is Transfer Learning? Exploring the Popular Deep Learning Approach. `https://builtin.com/data-science/transfer-learning`, 2021.

[29] Sagar Ram. What Does Freezing A Layer Mean And How Does It Help In Fine Tuning Neural Networks. `https://analyticsindiamag.com/what-does-freezing-a-layer-mean-and-how-does-it-help-in-fine-tuning-neural-networks` 2019.

[30] Chollet Francois. Complete guide to transfer learning fine-tuning in Keras. `https://keras.io/guides/transfer_learning/`, 2020.

[31] Jason Brownlee. Overfitting and Underfitting With Machine Learning Algorithms. `https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/`, 2016.

[32] Google Cloud Team. Cloud Tensor Processing Units (TPUs). `https://cloud.google.com/tpu/docs/tpus`, 2021.

[33] Ágnes Incze, Henrietta-Bernadett Jancsó, Zoltán Szilágyi, Attila Farkas, and Csaba Sulyok. Bird sound recognition using a convolutional neural network. In *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*, pages 000295–000300, 2018.

[34] Oisin Mac Aodha, Rory Gibb, Kate E. Barlow, Ella Browning, Michael Firman, Robin Freeman, Briana Harder, Libby Kinsey, Gary R. Mead, Stuart E. Newson, Ivan Pandourski, Stuart Parsons, Jon Russ, Abigel Szodoray-Paradi, Farkas Szodoray-Paradi, Elena Tilova, Mark Girolami, Gabriel Brostow, and Kate E. Jones. Bat detective—deep learning tools for bat acoustic signal detection. *PLOS Computational Biology*, 14:1–19, 03 2018.

[35] Kamalesh Palanisamy, Dipika Singhania, and Angela Yao. Rethinking CNN models for audio classification. *CoRR*, abs/2007.11154, 2020.

[36] Michelle Carney, Barron Webster, Irene Alvarado, Kyle Phillips, Noura Howell, Jordan Griffith, Jonas Jongejan, Amit Pitaru, and Alexander Chen. Teachable machine: Approachable web-based tool for exploring machine learning classification. CHI EA '20, page 1–8, New York, NY, USA, 2020. Association for Computing Machinery.

[37] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *CoRR*, 2018.

[38] Speech Commands Recognizer. `https://github.com/tensorflow/tfjs-models/tree/master/speech-commands`.

[39] Rulph Chassaing and Donald Reay. *Digital Signal Processing and Applications with the TMS320C6713 and TMS320C6416 DSK (Topics in Digital Signal Processing)*. Wiley-Interscience, 2008.

[40] National Instruments. Understanding ffts and windowing. 2019.

[41] p5 Javascript Library. `https://p5js.org/`.

[42] Sam Thornton Boyang Zhang, Jared Leitner. Audio recognition using mel spectrograms and convolution neural networks. 2018.

[43] J.B. Allen and L.R. Rabiner. A unified approach to short-time fourier analysis and synthesis. *Proceedings of the IEEE*, 65(11):1558–1564, 1977.

[44] Leland Roberts. Understanding the mel spectrogram. `https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53`, 2020.

[45] Donald Funes. The development and practice of electronic music: Edited by jon appleton and ronald perera. englewood cliffs, new jersey: Prentice-hall, inc., 1975. 400 pp. photographs, illustrations, music examples. hard cover. *Music Educators Journal*, 62(5):90–93, 1976.

[46] William Rose University of Delaware. Hesc686:mathematics and signal processing for biomechanics. https://www1.udel.edu/biology/rosewc/kaap686/notes/windowing.html, 2016.

[47] Solomon Kim, Vivian Chen, Daniel Tan, and Amil Khanzanda. Virufy on-Device Detection for COVID-19. `https://stanford-cs329s.github.io/reports/virufy-on-device-detection-for-covid-19/`, 2021.

[48] J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research. In *22nd ACM International Conference on Multimedia (ACM-MM'14)*, pages 1041–1044, Orlando, FL, USA, Nov. 2014.

[49] Justin Salamon and Juan Pablo Bello. Deep convolutional neural networks and data augmentation for environmental sound classification. *CoRR*, abs/1608.04363, 2016.

[50] Augustin Arnault, Baptiste Hanssens, and Nicolas Riche. Urban sound classification : striving towards a fair comparison. *CoRR*, abs/2010.11805, 2020.

[51] David Flanagan. *JavaScript The Definitive Guide*. O'Reilly, 2011.

[52] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly, 2008.

[53] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.

[54] Daniel Smilkov and Nikhil Thorat. Tensorflow.js: Machine learning for the web and beyond. *arXiv*, 2019.

[55] Hugo Zanini. Custom object detection in the browser using TensorFlow.js. `https://blog.tensorflow.org/2021/01/custom-object-detection-in-browser.html`.

[56] TensorFlow Core v2.5.0. `https://www.tensorflow.org/api_docs/python/tf/all_symbols`.

[57] Shanqing Cai. TensorFlow.js: Custom Loss. `https://codepen.io/caisq/pen/QZYZEZ?editors=1111g`.

[58] Vincent Mühler. 18 Tips for Training your own Tensorflow.js Models in the Browser. `https://itnext.io/18-tips-for-training-your-own-tensorflow-js-models-in-the-browser-3e40141c9091`, 2018.