Master thesis : Drone control through a vocal interface

Auteur : Pirlet, Matthias
Promoteur(s) : Louppe, Gilles
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en science des données, à finalité spécialisée
Année académique : 2021-2022
URI/URL : http://hdl.handle.net/2268.2/15850

University of Liège - Faculty of applied Science

Drone control through a vocal interface

Author:
PIRLET Matthias
Academic Supervisor:
LOUPPE Gilles
Industrial Supervisor:
GREFFE Christophe

Jury:
LOUPPE Gilles
DEBRUYNE Christophe
WEHENKEL Louis
GREFFE Christophe

Master thesis carried out in order to obtain the degree of
Master of Science in Data Science & Engineering

Academic year 2021-2022

Abstract

Today the use of drones is widely spread for many tasks, but for some of these, such as firefighting, it is vital that the operator's hands are kept free to do their job properly. Hopefully speech recognition is an exploding discipline in deep learning. This work therefore focuses on finding a deep learning model that recognises spoken commands to control a drone from a pre-defined set.

The first part of the work was to build a training dataset of commands with the combination of complete commands generated through the use of text-to-speech APIs and hand-crafted commands. These were created thanks to the concatenation of an open source spoken words dataset and another set of spoken words acquired through a web platform created in order to complete the missing words of the vocabulary command set. The testing set was acquired by asking people to record complete commands under real conditions.

The second part of the work focuses more on the different models that could be developed and all the techniques that can be used. These are presented as an ablation study, in order to improve the results on a test set in real conditions. Several methods were applied in order to achieve the final goal: the first is the use of computer vision models where the input of these models is a simple spectrogram of the different commands. The results using these types of models were not as good as those of the new models which take directly the raw waveform as input and combines vision, attention and self-supervised learning. The best version of this model obtains a F1-Score of 0.9973 on a real conditions dataset.

Acknowledgements

First of all I would like to thank my supervisor, Prof. Gilles Louppe, who helped me to find the subject of my thesis. His wise advice, his monthly meetings, his availability and the sharing of his knowledge about data science have enabled me to finalise this thesis.

Furthermore, I would like to thank my industrial supervisor Christophe Greffe, who was present at every meeting to support me and also when I needed advice on the subject.

My gratitude also goes to François Lievens and Julien Hubar, my Datascience partners and friends for our discussions to get through all the challenges of our master theses and without whom I would not have had such motivation.

Finally, I would like to thank my parents as well as my partner Violette for supporting me and encouraging me during those days and nights of work and stress since I started university.

Contents

Chapter 1

Introduction

1 Context

Drones are increasingly present in everyday life. Civilians, companies and governments use drones in different ways, such as journalism for aerial photographs, documentaries or even news. They can also be used during disasters, such as the one that happened last summer in Belgium and Germany, to assess the situation or to search for people during rescue operations. They can be used by companies with different sensors to inspect the condition of a building or a place when it is inaccessible for a human being. Finally, they can be used for military purposes, either for reconnaissance or combat. Here is the latest Drone Industry Barometer 2021 released by Germany-based research firm Drone Industry Insights. The FIGURE 1.1 shows how people using drones for commercial reasons reported how they actually use drones:

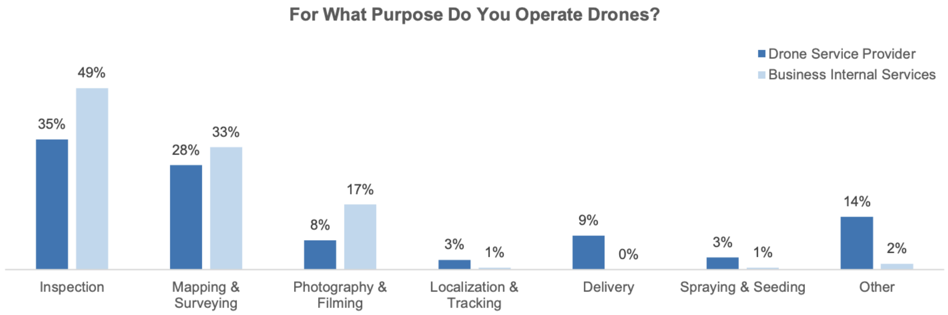


FIGURE 1.1: Purposes to operate drones

The problem is that you have to use a controller to command these drones which is sometimes annoying when the pilot's hands are already full. In the particular case of this work, the ultimate goal is the use of drones by firemen who need their hands to fully achieve their mission. Having a way to control the drone without hands is therefore very useful. Luckily, over the last few years, Deep Learning models have emerged in the field of Automatic Speech recognition (ASR) and are increasingly present in our daily lives with various personal voice assistants such as Siri, Alexa, Google home and others. Another type of task that deep learning models can achieve is command recognition. This is some sort of ASR subset because you only need to recognise a set of commands and not the entire vocabulary of a certain language.

2 Problem Statement

This Master thesis is in the line of Julien Bolland's work (2021) [4]. The final goal is to control a drone through voice, but the objective of this thesis is to focus on the speech recognition of a command set. A command is made of a **name**, an **instruction** and for some of these instructions a **number**. Each drone must be uniquely identified by its name. In this case, it was decided to limit to 4 different names [*Alpha, Beta, Gamma, Delta*]. Then, there are instructions without numbers [*emergency, go, land, stop*] and instructions with numbers [*backward, down, forward, left, mission, right, speed, up*] where numbers can be [*one, two, three, four, five, six, seven, eight, nine, free*]. Here are some examples of commands: {*Alpha land, Beta forward two, Gamma Stop, Delta up six ...*}

The most efficient way to represent this is to create a regular language. According to Louveaux (2021) [21] *"To show that a language is regular, it suffices to describe it using one of these characterizations : [Regular expressions, Deterministic finite automata, Non-deterministic finite automata, Regular grammar]"* (Slide 75 of the course). Once again according to Louveaux (2021) [21] *" a deterministic finite automaton is defined by a five-tuple* $M = (Q, \sum, \delta , s, F)$*, where :*

- Q *is a finite set of states*
- $\sum$ *is a finite alphabet*
- $\delta : Q \times \sum \to Q$ *is the transition function*
- $s \in Q$ *is the initial state*
- $F \subseteq Q$ *is the set of accepting states."*

Here the regular language is represented by a deterministic finite automaton as in Bolland's work, 2021 [4]. The initial one for this problem can be seen in Figure 1.2 and can be defined by:

- ○ $Q = \{q0, q1, q2, q3, q4\}$,

- $\sum$ ={Alpha, backward, Beta, Delta, down, eight, emergency, five, forward, four, free, Gamma, go, land, left, mission, nine, one, right, seven, six, speed, stop, three, two, up, Other, Silence }
- δ corresponds to the set $\{N, O, I\_N, I\_W, Nb\}$ where
  - N stands for "Names" and is activated with {Alpha, Beta, Gamma, Delta},
  - O stands for "Other", is activated with {silence, other},
  - I_N stands for "Instruction No number" and is activated with {emergency, go, land, stop},
  - I_W stands for "Instruction With number" and is activated with {backward, down, forward, left, mission, right, speed, up},
  - Nb, for "Number", is activated with {one, two, three, four, five, six, seven, eight, nine, free}
- $s = \{q0\}$
- $F = \{q2, q4\}$

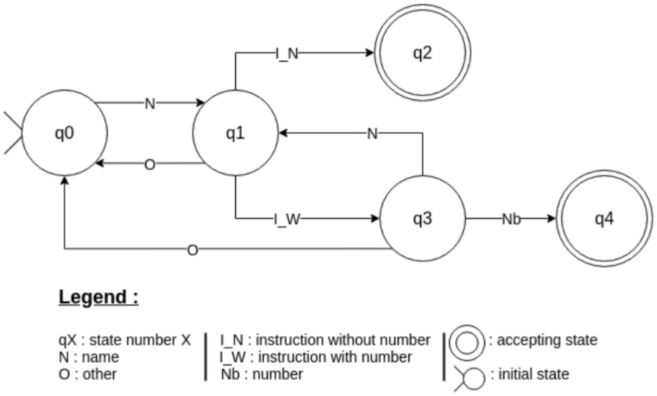


Figure 1.2: Initial representation of the deterministic finite automaton for the drone control language from Bolland [4]

With the vocabulary presented above made of 28 words and this deterministic finite automaton, there are 336 different possible ways to end up in an accepting state. This means that there are 336 different existing commands.

This work presents how state-of-the-art models in *automatic speech recognition* and *keyword spotting* were adapted in order to perform command recognition on a relatively small dataset. This is done by a concatenation of commands generated by text-to-speech APIs and commands created through the combination of an open source spoken words dataset and another set of spoken words acquired through a web platform created in order to complete the missing words of the vocabulary command set.

3 Related work

3.1 Automatic Speech Recognition (ASR)

Automatic Speech Recognition is a subfield of computer science where the goal is to transcribe spoken language into text. There are several state-of-the-art models for the different public datasets that can be found on the paperswithcode website. The best-known model for this problem nowadays is **Wav2vec 2.0** by Baevski et al. (2020) [2]. Wav2vec is explained in more detail in the Background chapter but in a few words, Wav2vec takes advantage of a fairly large amount of unlabeled data and subjects it to self-supervised learning and contrastive learning. Only after that, once the model has a good representation of the data, the model is fine-tuned on labeled data. Wav2vec has several variants such as **Wav2vec-Bert** by Chung et al. (2021) [8] which combines contrastive learning and masked language modeling (MLM). This has a big success in pre-training natural language processing models. There are plenty other variants like **W2V2-B-VP100K** by Shon et al. (2021) [32] and others that improve some of the multiple issues of the wav2vec model, but these will not be explained in great detail.

Moreover, there is **HuBERT** by Hsu et al. (2021) [15] that tackles different problems of Wav2vec 2.0, such as the presence of multiple sound units in each input utterance by using a clustering step to provide aligned target labels for a BERT-like prediction loss.

3.2 Keyword Spotting (KWS)

Keyword spotting in speech processing is a problem whereby certain keywords in spoken language are tagged. A special case of this is the so-called "wake word detection" that is mainly present in personal voice assistants like Siri with the wake word "*Hey Siri*", Alexa with simply "*Alexa*" or Google Home with "*Okay Google*" which aims to wake the assistant. There are different methods to reach the final goal: Efficient CNN-based models which are basically CNNs that either use 1D or 2D convolutions and **BC-ResNet** by Byeonggeun et al. (2021) [18] which combines both. There are also models such as **MobileNets** [30] or **ShuffleNets** [40] which utilise depthwise separable convolutions, inverted bottleneck blocks. There are other approaches than CNN-only like the one presented by Coimbra de Andrade et al. (2018) [1] which uses CNNs at the front and then a RNN (LSTM) in order to perform high-level features. ASR-based models like the one used in **Wav2KWS** by Seo et al. (2021) [31] can also be applied but the number of parameters is really much larger than in CNN-based models.

4 Thesis outline

The work is structured as follows:

- **Chapter 2: Background.** This chapter gives background notions. Section 1 gives reminders of sound, waveforms, Fast Fourier Transform (FFT) and spectrograms because these notions are used throughout this work. In section 2, it also gives a brief reminder of Transformers and attention. Then, in section 3, self-supervised learning is explained as it is used in a model. Finally, a summary of the models used in this thesis is given in section 4.

- **Chapter 3: Data.** This chapter shows what data was acquired, several preprocessing's applied to it, what datasets were created using this data and how these created datasets were separated into training, validation and testing sets.
- **Chapter 4: Experiments.** This chapter first explains how, and which metrics are used to evaluate the different models and on which dataset they are calculated. Then a large ablation study shows different experiments in order to improve those results. After that a small comparison is made with Bolland's [4] results. Finally, the best models are fine-tuned on the real conditions dataset and their inference time is computed to get an idea of how long it would take the drone to make a prediction.
- **Chapter 5: Conclusions.** In this chapter, several ideas are given for improving the work and defining the ideal next steps. Lastly, there is a short review and summary of the work as a whole.

The different codes used for the experiments of this thesis can be found on this GitHub page.

Chapter 2

Background

1 Sound signal

Sound is a pressure wave produced by a vibrating object and which can be picked up by the human ear. The average human being hears sounds in the frequency range of [15 - 20k]Hz. The frequency of a sound is calculated as follows:

$$frequency = \frac{speed}{wavelength}$$

where speed is the speed of the sound which depends on the medium (air, water, ...). The Amplitude represents the strength of the sound pressure. The range of audible sound pressure is [$2 * 10^{-5}$ - 20]Pa. Due to its large range, sound amplitude is measured on a logarithmic scale called Decibel[dB] which is not strictly speaking a physical unit but a ratio between the intensity of two sounds. However, in order to make it interpretable as a real unit, one of the pressures is fixed: $P_0 = 2 \cdot 10^{-5} Pa$ (which is the reference of 0db and corresponds to the threshold of hearing). So, decibel can be defined as:

$$20 \cdot log_{10}(\frac{P}{P_0})$$

where P is the intensity of the measured sound. Here are some examples with sound levels in dB found on the website of iacacoustics:

- ▷ Threshold of hearing: 0 dB
- ▷ Breathing: 10 dB
- ▷ Quiet living room: 25-30 dB
- ▷ Library, bird calls; lowest limit of urban ambient sound : 40-50 dB
- ▷ Conversation in restaurant, office, background music: 60-70 dB
- ▷ MP3 players (at full volume), concerts, car horns: 100 dB

- ▷ Average human pain threshold: 110 dB
- ▷ Instant perforation of eardrum: 150 dB

A waveform displays the amplitude of the signal through time (the signal is viewed in the time domain). The sounds encountered are not always so simple and periodic. They are often the sum of waveforms of different frequencies. Waveforms produced by different music instruments can be found in FIGURE 2.1. A sound spectrum displays the amplitude of each of the different frequencies present in a sound (the signal is viewed in the frequency domain) thanks to the Fast Fourier transform (FFT) algorithm. Nevertheless before using this algorithm, the sound needs to be digitised.

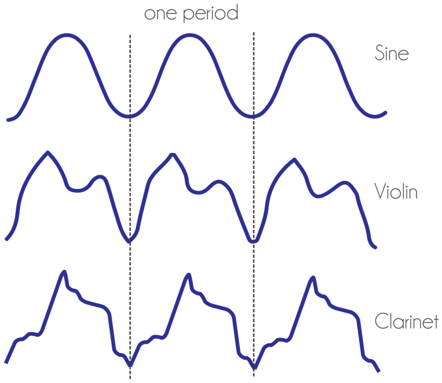


FIGURE 2.1: Waveform representations of a simple sinusoidal function, a sound produced by a violin and a sound produced by a clarinet. Credits : `https://www.pngegg.com/en/png-ejfri`

1.1 Digitising sound

In order to be processed and input into a model, the sound signal needs to be discretised and digitised. This is achieved by measuring the amplitude of this sound at a certain rate. A sample is the name of one measurement and the sample rate is the number of samples collected in one second (a sample rate of 16kHz means that 16,000 samples are collected per second). FIGURE 2.3 represents the waveform for one example of the command *"Alpha up seven"* recorded in real conditions with a samplerate equal to 16kHz and more or less 50 dB of background noise measured around.

1.2 Fast Fourier Transform (FFT)

The Discrete Fourier Transform (DFT) is an important tool in digital signal processing because it is used to find the spectrum of a finite-duration signal. It can be defined by the formula:

$$x[k] = \sum_{n=0}^{N-1} x[n]e^{-i2\pi kn/N}$$

In order to calculate this, it is necessary to compute N outputs and each of these is a sum of N terms which gives $O(N^2)$ operations. The Fast Fourier Transform reduces the number of computations needed to $O(N * log_2(N))$. It may seem insignificant, but if an operation takes 1 nanosecond, for $N = 10^9$, it will take the FFT algorithm 30 seconds to calculate the DFT. Whereas the original algorithm will take more than 32 years.

In order to compute the FFT algorithm, the Fourier Transform is separated into even and odd indexed subsequences. The advantage is that the 2 can be done in parallel and each term consists of $(N/2) * N$ computations (still N^2), but halving the calculation time.

$$x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]e^{-i2\pi k(2r)/N} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]e^{-i2\pi k(2r+1)/N} \tag{2.1}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} x[2r]e^{-i2\pi k(2r)/N} + e^{-i2\pi k/N} \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]e^{-i2\pi k(2r)/N} \tag{2.2}$$

$$= x_{even}[k] + e^{-i2\pi k/N} x_{odd}[k] \tag{2.3}$$

Then it can be further improved by applying the divide and conquer approach and halving once again the computational time. If N is a power of 2, the maximum number of times it can be divided by 2 is $log_2(N)$ and the computation becomes $O(N * log_2(N))$.

1.3 Frequency domain VS. Time domain

A figure comparing the time and frequency domain of a waveform can be found in 2.2. When FFT algorithm is applied to the sound signal, the problem is that it only gives frequency values, and the track of time information is lost. The system is not able to tell what word was spoken first if these frequencies are used as features. it is necessary to find a different way to present the features for the system in such a way that it has frequency values along with the time at which they were observed. This is where spectrograms come in.

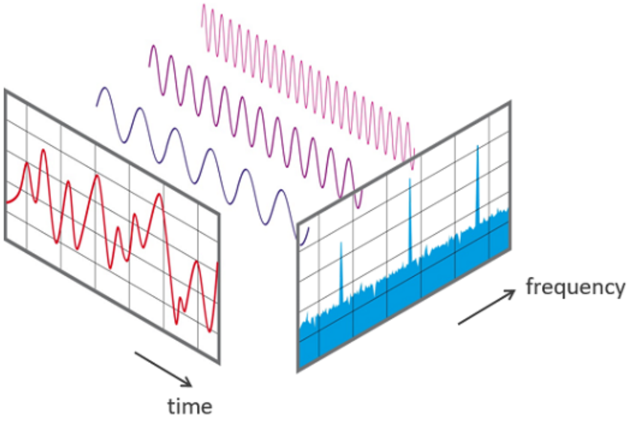


Figure 2.2: Time and Frequency domain representation.
Credits :
https://commons.wikimedia.org/wiki/File:FFT-Time-Frequency-View.png

1.4 Spectrogram

A spectrogram represents how the spectrum evolves through time. The idea to construct this spectrogram is to break the signal into smaller frames (called windows) and computes FFT for each window. This gives the frequencies for each of the windows and a window number represents time, as window 1 comes first, window 2 next and so on. It's a good practice to keep these windows overlapping, otherwise some frequencies could be lost. On the horizontal axis, time is represented. The vertical axis represents frequency. A third "dimension" is represented by the intensity of the colour of a point and indicates the amplitude of a particular frequency at a particular time. The spectrogram of the waveform represented in Figure 2.3 can be seen in Figure 2.4

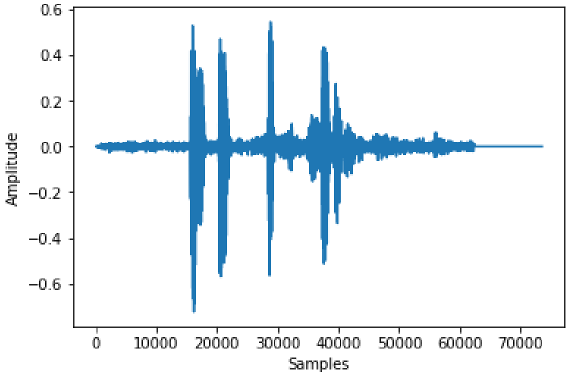


Figure 2.3: Illustration of the waveform of one of the spoken command: *"Alpha up seven"*

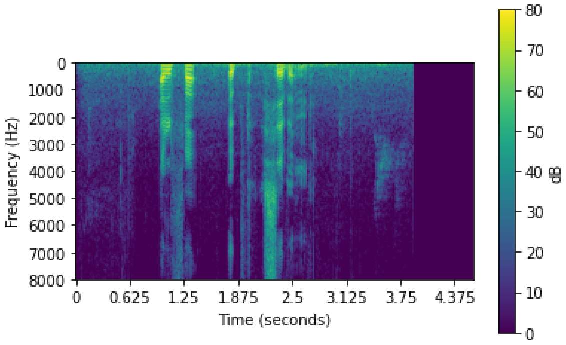


Figure 2.4: Spectrogram of one of the spoken command: *"Alpha up seven"*

2 Transformer Attention Mechanism

Before the introduction by Vaswani et al. (2017) [36] in the paper "Attention is all you need" of the Transformer architecture, attention was implemented by Recurrent Neural Networks (RNN)-based encoder-decoder architecture as in Bahdanau et al. (2014) [3] and Luong et al. (2015) [22]. Following Xu et al. (2015) [38], the context attention mechanism can be adapted to caption generation (takes as input an image and outputs a sentence describing the image) by replacing the encoder which was a Recurrent Neural Network with a Convolutional Neural Network (CNN) that extracts a feature map over the input image. Transformers have really revolutionised the implementation of attention by dispensing of recurrence and convolutions and solely relying on the use of attention. It allows to process words in parallel instead of processing them sequentially. As a reminder, the main idea of attention is that for every predicted output word, the model only uses the part of the input where there is the most relevant information, instead of using the entire input sequence.

As can be seen in Figure 2.7, there is an encoder part (on the left) made of multiple stacked modules ($N = 6$ in the original paper). A module is composed of a multi-head attention sub-module, and a one-hidden layer Feed Forward Network, with residual pass-through and layer normalisation. Each of these N modules maps the input sequence to a contextualised encoding sequence. The decoder part on the right made of multiple stacked modules (N=6 also in the original paper) is similar to the encoder but uses a Masked Multi-Head Attention. This sub-module basically masks the future tokens that are left to be generated and attention is only computed on the ones that are already present. In addition, the decoder inserts a third sub-module which performs multi-head attention over the output of the encoder stack. Each decoder module defines the conditional probability distribution of a target sequence in the case of a sequence-to-sequence problem given the contextualised encoding sequence or concatenates the different contextualised encoding sequences and passes it to a feed-forward layer for a classification problem such as sentiment analysis.

The attention-mechanism used in this architecture can be seen in Figure 2.5 and is called *Scaled Dot-Product Attention*. It is computed as follows:

First, for each word in the input sequence, a fixed size vector called *"embedding"* is computed (computers understand numbers not words; words are therefore encoded into vectors of numbers, depending on their context and more). This embedding is coupled with a positional encoding because as the words are processed in parallel here and not sequentially as in Recurrent Neural Networks, the positional encoding gives an indication of the position of the word in the sentence.

With this in mind, three matrices are defined, *"Query"* represented by matrix $\mathbf{Q} \in \mathbb{R}^{T \times D}$ where each of the T rows represents a query. *"Key"* represented by matrix

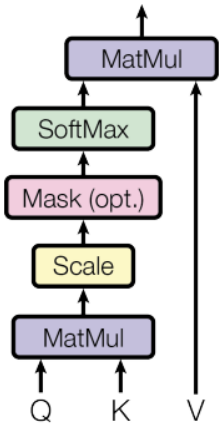


Figure 2.5: Illustration from Vaswani et al. 2017 [36] of the Scaled Dot-Product Attention. The query and key matrices are first multiplied and then normalised thanks to the scale and softmax steps. This gives a new matrix which is now multiplied with the value matrix. The final output of this multiplication gives the attention.

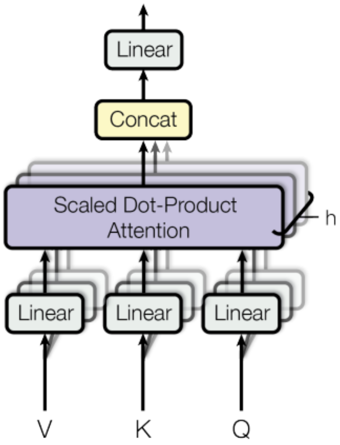


Figure 2.6: Illustration of the Multi-Head Attention from Vaswani et al. 2017 [36] which is basically a scaled dot product attention, that you can see on the left, replicated several times to have h different representations and these representations are then concatenated and projected through a weight matrix.

$\mathbf{K} \in \mathbb{R}^{T' \times D}$ where each of the T' rows represents a key. *"Value"* represented by matrix $\mathbf{V} \in \mathbb{R}^{T' \times D'}$ where each of the T' rows represents a value. These three matrices are obtained by multiplying the embedding by a weight matrix: $\mathbf{Q} = \mathbf{X}\mathbf{W}^{Q}$, $\mathbf{K} = \mathbf{X'}\mathbf{W}^{K}$ and $\mathbf{V} = \mathbf{X'}\mathbf{W}^{V}$. In the case of Self-attention : $\mathbf{X} = \mathbf{X'}$.

After that, the attention weights are computed by a dot product between $\mathbf{Q}$ and $\mathbf{K}$. This dot product is scaled by the square root of the dimension of the keys. Optional mask is applied to it and then a softmax is applied to the result.

Finally, attention weights are applied to values V to give this final formula

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\Big(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\Big)\mathbf{V}$$

The Multi-head attention that can be seen in Figure 2.6 is basically the scaled dot product attention replicated several times (like for the convolution where you have multiple filters). Thus, instead of having only one projection for $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$, there are h different projections and therefore h different representations of each matrix. Each one of these scaled dot product attention can be written as:

$$\mathbf{H}_i = attention(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$$

In the end, all the resulting $\mathbf{H}_i$ are concatenated and projected through the weight matrix $\mathbf{W}^O$. The multi-head formula can be expressed as:

$$multihead(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = concat(\mathbf{H}_i, ..., \mathbf{H}_h)\mathbf{W}^O$$

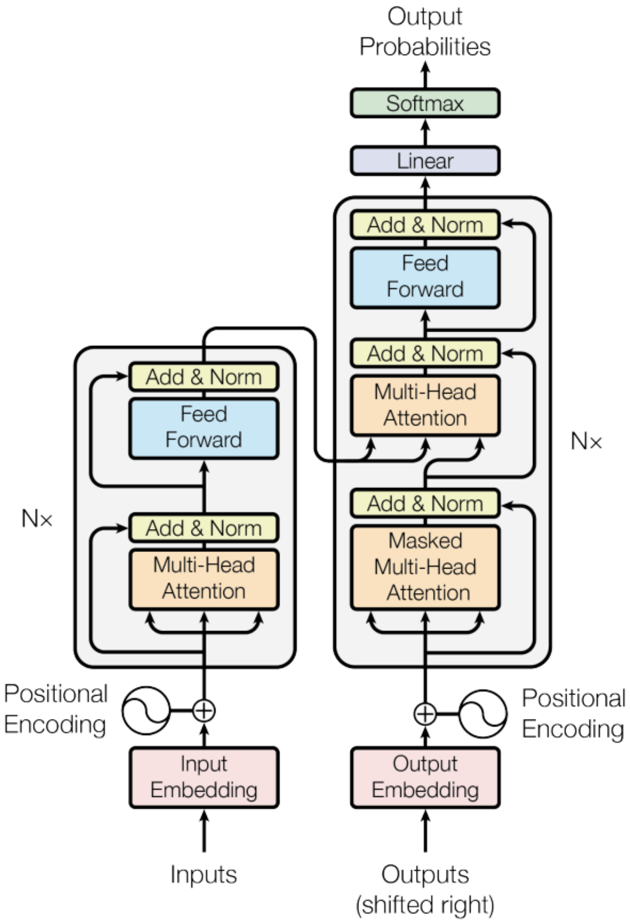


Figure 2.7: Illustration of Transformer architecture from the original paper of Vaswani et al. 2017 [36]. On the left, the encoder which takes the input embedding to which a positional embedding is added to keep track of the position. This whole is passed to the multi-head attention and a feed forward network to produce some hidden states. The decoder, on the right, takes two arguments: the outputs that are already produced with previous inputs and the output hidden states of the encoder. Once again, these 2 pass through some multi-head attention layers in order to produce some hidden states and at the end a linear layer and a softmax serve as a head classification to make a prediction of the best tokens to output.

3 Self-supervised Learning

In the last decade, Scientists focused a lot on supervised learning methods. The problem is that they require huge amounts of labeled data to train and having good quality labelled data consumes time and money. Moreover, supervised learning works very well on one specific task, but not necessarily on others. For example, if several images of cows are shown to a baby, it will eventually be able to recognise any cow he sees, however AI can fail to recognise it, if for example the cow is in an unusual situation (i.e. cow lying on a beach). According to Yann LeCun in a blog about Self-supervised learning: *"Supervised learning is a bottleneck for building more intelligent generalist models that can do multiple tasks and acquire new skills without massive amounts of labeled data"*.

Humans beings rely on their previously acquired background knowledge of how the world works thanks to observations and interactions. This is what Yann LeCun calls the "*common sense*" and this is why humans beings don't need huge amounts of labelled data to learn a new task. Self-supervised learning (SSL) is one of the most promising ways for a model to obtain such background knowledge and therefore attain some sort of common sense.

Self-supervised learning is a machine learning process whereby the model trains itself to predict a part of the input (that it pretends it doesn't have) from the rest of the input. It is also known as predictive or pretext learning. This can be done in several ways as can be seen in FIGURE 2.8 by predicting the future from the past or predicting the past from the present or predicting the top from the bottom (and vice-versa) or alternatively, predicting the occluded from the visible.

This method of self-supervised learning transforms the unsupervised problem into a supervised problem by taking the unlabelled data and creating labels. By doing this, it receives self-feedback which is its key difference compared to unsupervised learning. The goal of the pretext task is to guide the model to learn intermediate representations of data and it is expected that this representation carries good semantic or structural meanings to be beneficial to a large number of practical downstream tasks. During these downstream tasks which is the transfer of knowledge to a specific task, a subset of the features learnt during the pretext task is relevant and is rapidly adapted and used. Nonetheless, some of them are overwritten and re-learnt to get additional signal from the input that was not learnt during the predictive learning.

Predictive learning can be : *"Patch localisation"* as in the paper from Doersch et al. (2015) [10] where the goal is to extract multiple patches from an image and train the model to find the relationship between these patches as can be seen in FIGURE 2.9.

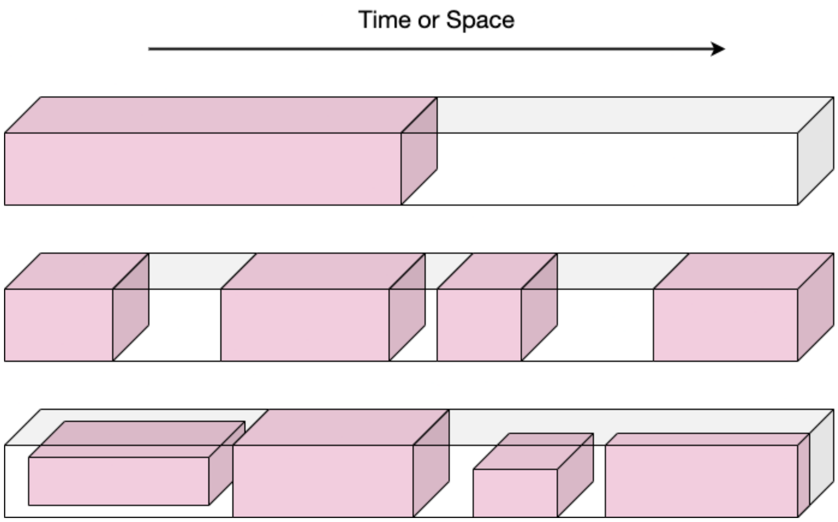


FIGURE 2.8: Predictive learning: In self-supervised learning the model hides parts of the input which can be images, videos, sentences, etc., and tries to predict these hidden parts (transparent) thanks to the parts it didn't hide (pink).

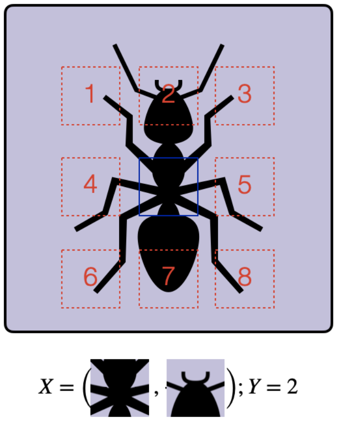


FIGURE 2.9: Illustration of self-supervised learning pre-training technique called patch localisation where the goal is to extract several patches in the image and train the model to understand the relationship between multiple patches.

The second one is *"Distortion"* introduced by Dosovitskiy et al. (2015) [12] where patches are sampled from different images with different scales, positions, etc. Then each patch is distorted by applying random transformations (translation, rotation, scaling, etc.), and all the resulting distorted patches belong to the same surrogate class (as they all come from the same image). The final goal is to train the model to find the class of a patch. The next one is called *"Colorization"* as in Zhang et al. (2016) [39]. Its aims is, as its name suggests, given a grayscale image, it outputs the same but coloured image. Precisely, the task is to map the image to a distribution over quantized colour value outputs (Quantization is the process of mapping input from a large or continuous set of values to output values in a finite "small" set. A finite number of colours are required, as in "RGB", where you have 16,777,216 different colours). Given an image (but it can be an audio or a text) with some missing information in it, *"Context Filling"* can be used. It predicts what is missing to fill the gap as in Pathak, et al., (2016) [26] and can be seen in Figure 2.10

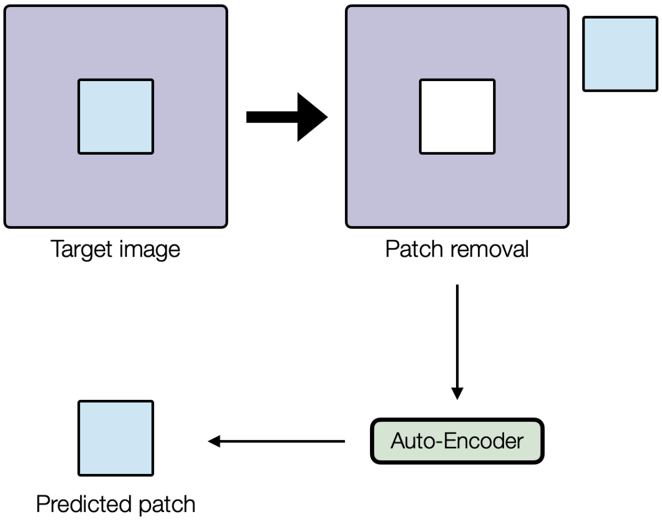


Figure 2.10: Illustration of self-supervised learning pre-training technique called context filling where the goal is to remove a part of the input image and train the model to predict the missing part.

For videos, *Video Motion Prediction* can be used and consists of predicting from a set of images all possible images that may precede or follow. For text, predictive learning is when the model is shown a short text (typically 1,000 words) in which some words have been masked or replaced. The model is then trained to predict the masked or replaced words.

Another technique which is important for later use to learn general features of an unlabeled dataset is called *"Contrastive learning"*. This technique learns by teaching the model which data points are similar or different. A simple example of this in the real world is when a baby sees two cows and one dog, he understands that the two cows look similar compared to the dog which is a little bit different. The baby recognises similarities; contrastive learning works somewhat in the same way. It tries to compare pairs of data to know if they are similar or different in order to learn high-level features about this data. There are multiple contrastive learning methods : SimCLR introduced by chen et al. (2020) [5] and SimCLRv2 again by chen et al. later that year [6], Moco introduced by He et al. (2019) [14] and Mocov2 by Chen et al. (2020) [7] and PIRL by Misra et al. (2019) [23]. Hereafter, the SimCLRv2 is explained to have a better understanding of what contrastive learning is. SimCLRv2 can be described in 3 steps:

first, for each image of the dataset, a combination of augmentations are performed among crop, resize, color distortion, grayscale. This is done twice per image to get two augmented images. The second step is to input these two augmented images in a CNN (ResNet) which creates vector representations. The objective is for the model to find similarities between these two images so it is also required to have approximately the same vector representations for the two images. The third step to achieve this is to maximise the similarity and therefore minimise the contrastive loss function which can be explained as follows:

$$\mathcal{L}_m(i,j) = -log\left(\frac{exp(s_{i,j})}{\sum\limits_{k=1}^{2N} \mathbb{1}_{[k\neq i]} exp(s_{i,k})}\right) \tag{2.4}$$

where $s_{i,j}$ is the cosine similarity : $s_{i,j} = \frac{z_i^T z_j}{(\tau \|z_i\| \|z_j\|)}$, and τ is a temperature parameter. It allows to scale the inputs and widen the range [-1, 1] of cosine similarity

These self-supervised learning algorithms are useful when you don't have a lot of labelled data, on the other hand they can suffer from several limitations: The first limitation is *"Computational Efficiency"*. This is the fact that learning models with labels can be built much faster compared to unlabeled learning models. This is due to the fact that, in SSL approaches, generating labels for a given dataset is an additional task, hence it requires more computer power. The second limitation is the amount of data, SSL approaches need a lot more data to achieve accuracies close to those of supervised learning approaches. Finally a last limitation is labeling accuracy, the model may produce inaccurate labels and those inaccuracies can lead to serious inaccurate results for your task.

4 Deep Networks for Speech Recognition

In this section the main models used are presented

4.1 ResNet

It is known that the deeper a model is, the better the results [34]. Before ResNet was introduced by He et al. (2014) [13], models such as VGG nets [33] or AlexNet [19] suffered from a serious problem, they couldn't go as deep as required. Even with a proper initialisation adding more and more layers resulted in vanishing gradients and a saturating of the network accuracy. With ResNets, the gradients can flow directly through Residual Blocks as can be seen in FIGURE 2.11 below. The idea behind this Residual Block is to add the input and the output of the 2 weight layers. Then perform the ReLU activation on this sum. Here, the skip connection helps to bring the identity function to deeper layers. This trick enables deeper networks than VGG nets but still has lower complexity.

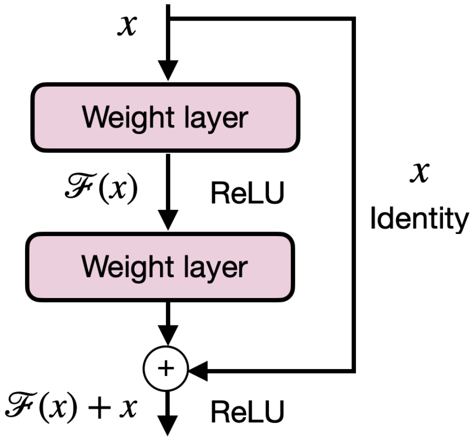


FIGURE 2.11: Description of a Residual Block: a skip connection is added from the input of the first weight layer to the output of the second weight layer in order for the gradients to flow and not vanish during backpropagation.

ResNets can have different sizes (i.e. a different number of layers and how big these layers are). Here, an "Operation" is made of a convolution, a batch normalisation and a ReLU activation. A "Block" consists of 2 or 3 operations depending on the ResNet and a "Layer" is a group of blocks. In the Appendix FIGURE 5.1, you can see a summary of the different ResNet with the different blocks in brackets and the number of these blocks. Downsampling of the volume through the network is done by increasing the stride from 1 to 2, at the first convolution of each layer and not thanks to a pooling operation. Therefore, the skip connection of the first block of each layer has to be a slightly different in order to match the output size. Two options are considered in the paper: Either a 1x1 convolution and a stride of 2 is used, or the input volume is padded with extra zero entries to increase dimensions.

At the end of all these blocks, a global average pooling layer is used. Since the introduction by Min Lin et al. (2013) [20], experts use this global average pooling (GAP) layer so as to minimise overfitting by reducing the total number of parameters in the model. Thus, it starts with a tensor of dimensions $h \times w \times d$ and this tensor is reduced through the GAP layer in a tensor of dimensions $1 \times 1 \times d$.

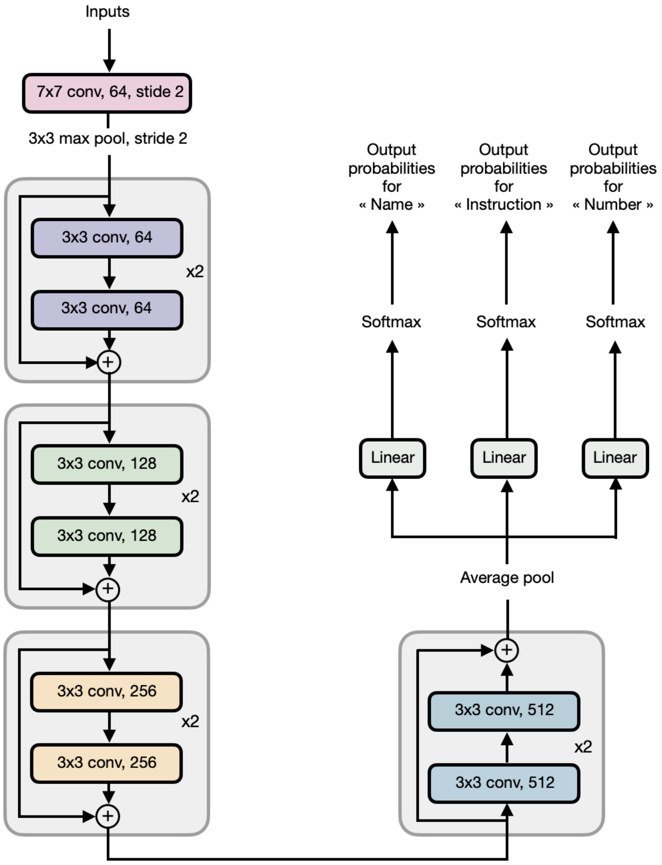


FIGURE 2.12: Description of the architecture of the ResNet 18 (He et al. (2015) [13]) in this work. The spectrogram is input and passed through a first convolution to decrease the spatial dimensions and to increase the depth of the feature map. Then the output of this convolution is passed through a stack of groups of residual blocks. As can be seen all 4 groups contain 2 residual blocks with different convolutions internal to them. At the end, the final feature map is reduced to a one-dimension representation, passed through a linear layer and is then passed through a softmax to obtain the different probabilities.

In this work, the main ResNet architecture used is the ResNet-18. The architecture can be seen in FIGURE 2.12. However, the batch normalisation and the ReLU activation applied after each operation are not shown.

4.2 EfficientNet

In the classical convolution neural networks, scaling-up is often performed to achieve better results. Nonetheless, this scaling-up is often applied to only one of the three following dimensions. The first dimension is depth: the number of layers is increased in the network to recognise more complex patterns. When depth is too high, the problem of vanishing gradients reoccurs. The second dimension is width: the number of channels is increased to recognise more variety in patterns in convolution layers. When this dimension is too high, overfitting may occur. The last dimension is image resolution: the width and height of the image are increased, leading to a higher number of pixels and therefore a better resolution. Increasing only one of these dimensions is too random and takes a lot of time, as searchers must analyse and test for the best architecture that maximises the accuracy.

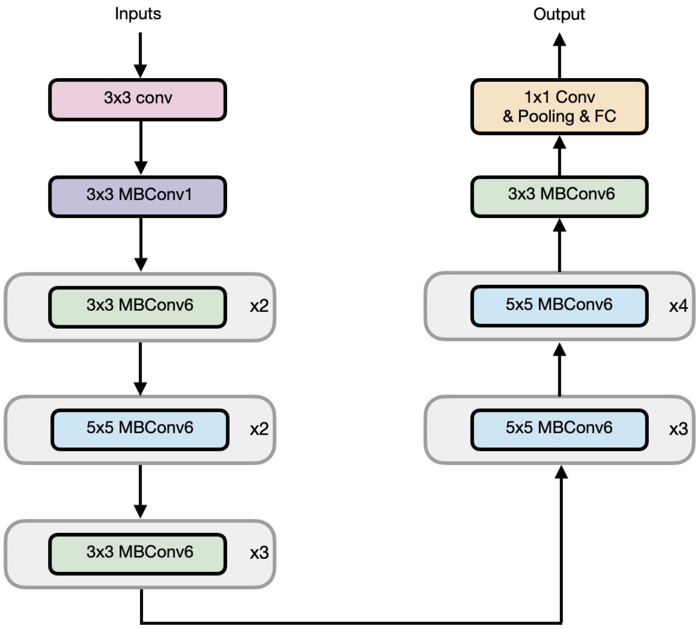


FIGURE 2.13: Description of the EfficientNet-B0 architecture (Tan and Le 2019 [35]). The input spectrogram is passed through a first convolution in order to reduce the spatial dimensions and then passed through a stack of MBConv blocks. MBConv blocks are described in FIGURE 2.14

EfficientNet introduced by Tan M. and Le Q. (2019) [35] is a new kind of architecture for CNN's but also does what is called *"Compound scaling"* and which is the scaling of all three dimensions at the same time and maintaining a sort of balance between them. This may be some sort of intuition that if the input image resolution is bigger, the network will require more layers and more channels in order to capture more fine-grained patterns on this bigger image. As said before, compound scaling is done with the aim to balance dimensions (width w, depth d, and resolution r) by scaling with a constant ratio. The method proposed in this paper is the following:

$$\begin{aligned} depth &: d = \alpha^{\phi} \\ width &: w = \beta^{\phi} \\ resolution &: r = \gamma^{\phi} \\ \text{s.t.} \;\; & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{2.5}$$

where α, β, γ are constants determined by a small grid search and ϕ is a user-specified coefficient that controls how many more resources are available for model scaling. If $\phi = 1$, it is called *EfficientNet-B0* and $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$. To obtain the scaled-up versions *EfficientNet-B1* to *EfficientNet-B7*, α, β, γ are fixed and different values of ϕ are used.

EfficientNet architecture consists of stacked *MBConv blocks* as can be seen in FIGURE 2.13 which are themselves a type of inverted residual block. These inverted residual block were introduced in the paper of MobileNetV2 by Sandler et al. (2018) [30] and the difference between an inverted residual block and the traditional residual block is that the latter has a *wide* → *narrow* → *wide* structure in terms of channels. It starts with an input that has a high number of channels, it is then compressed thanks to a 1x1 convolution and finally increased again thanks to a 1x1 convolution for the input and the output to be added. The idea with the inverted residual block is the opposite. It has a *narrow* → *wide* → *narrow* approach where it uses depthwise convolutions to reduce the number of parameters and the number of computations. Swish activation function introduced by Ramachandran et al. [27] is used inside MBConv block and is computed as follows:

$$f(x) = x \cdot sigmoid(\beta x) \tag{2.6}$$

where β is a learnable parameter.

The MBConv block which can be seen in FIGURE 2.14 also uses the *Squeeze-and-Excitation block* introduced by Hu et al. (2017) [16]. The idea behind this block is similar to what attention does, it adds a parameter to each channel of convolutional block to somehow adjust each feature map. It works as follows: the input is a convolutional block, then each channel of this block is reduced to a single value using the average pooling. These values are then passed to a dense layer followed by a ReLU to add non-linearity and afterwards it is passed to a second dense layer, followed this time by a Sigmoid function to obtain weights that can be applied on each channel of the input convolutional block. This, as well as the whole *EfficientNet-B0* can be seen in FIGURE 2.13. Finally, EfficientNet is trained to maximise accuracy but at the same time it penalises it if the network is computationally heavy. It also penalises the network if it has a slow inference time (i.e. when the network takes a long time to make a prediction).

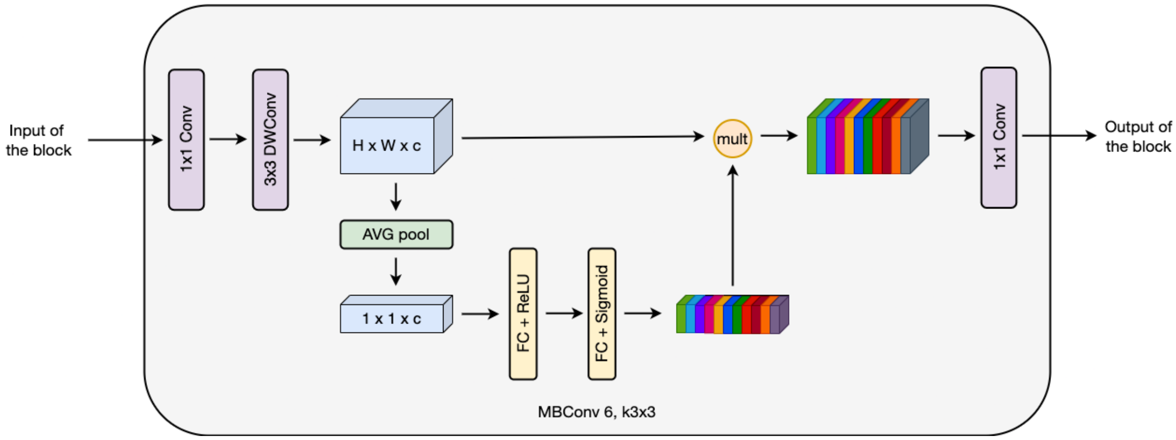


FIGURE 2.14: Description of the MBConv 6 block, k3x3 representation. The input is given to the MBconv block which starts with a 1x1 convolution to compress the number of channels then the output is given to a depthwise convolution before passing into the Squeeze-and-excitation optimisation procedure which takes the channels of the intermediate feature map and weighs them. The weights are computed by feeding an average of the feature of each channel into two fully connected layers. Then the weighted intermediate feature map is increased again thanks to a 1x1 convolution.

4.3 ViT (Vision Transformer)

The idea behind Vision Transformer is to use the self-attention mechanism of Vaswani et al. (2017) [36] on images. The problem of Transformers is that it lacks inductive biases inherent to CNNs, such as translation invariance (the fact that it can recognise an object whatever its position in the image) and locality (convolutions are linear local operators). In the original Vision Transformer paper from Dosovitskiy et al. (2020) [11] authors claim that if the model is trained on small-sized or mid-sized datasets such as ImageNet, it reaches accuracies slightly below the ResNets of comparable size because of the lack of inductive biases and it does not generalise well when not enough data is used. However, if the model is trained with large datasets (containing 14 to 300 Million images) it outweighs inductive biases and can be as good or even better than ResNets or EfficientNets.

The problem with Transformers is that they are designed to handle sequences not grid-structure data as images. Furthermore, they are known for their quadratic complexity when computing the attention matrix. If self-attention was naively applied to an simple grayscale image of size 256 × 256, it would be flattened into a sequence of 65536 pixels and the attention matrix would have a size 65536×65536 where each pixel attends to every other pixel. The authors of the paper have therefore decided to break the image into several patches.

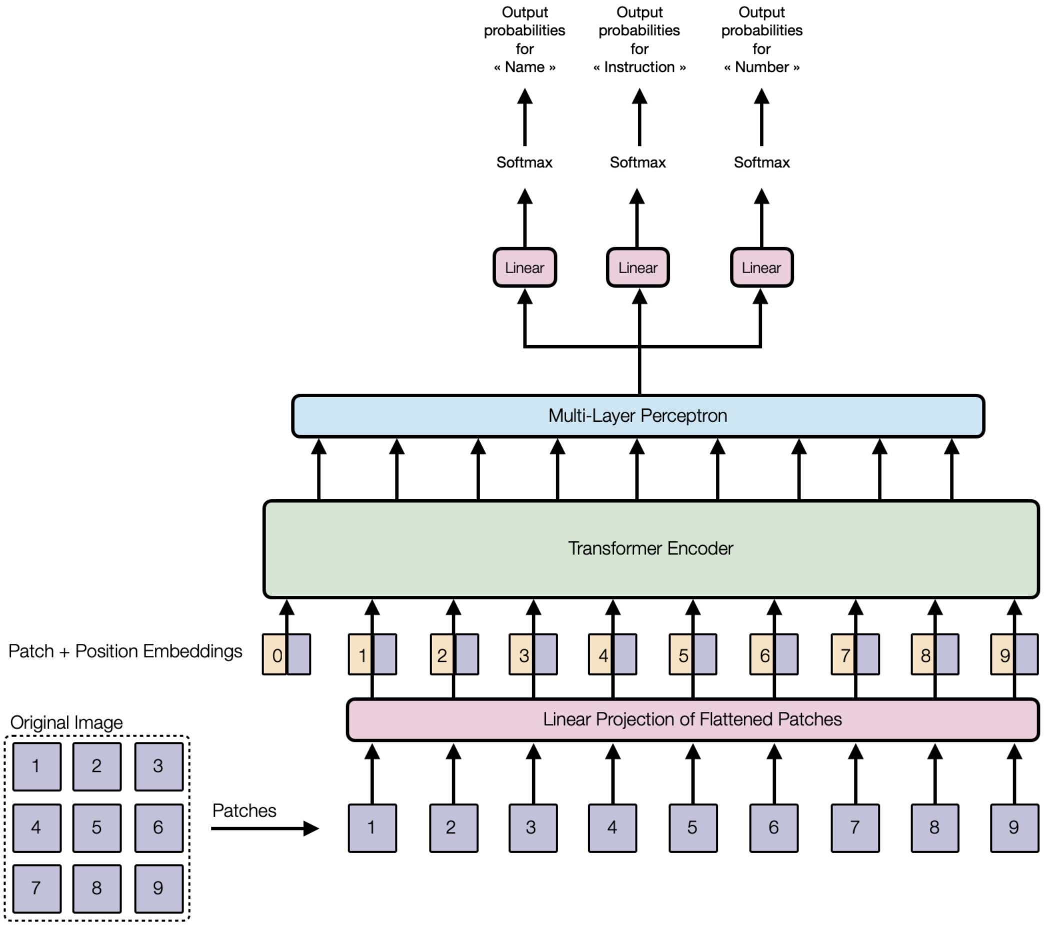

Output probabilities for « Name »
Output probabilities for « Instruction »
Output probabilities for « Number »
Softmax
Softmax
Softmax
Linear
Linear
Linear
Multi-Layer Perceptron
Transformer Encoder
Patch + Position Embeddings
0
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches
Original Image
1
2
3
4
5
6
7
8
9
Patches
1
2
3
4
5
6
7
8
9

FIGURE 2.15: Description of Vision Transformer's architecture (Dosovitskiy et al. 2020 [11]) in this work. The input image is first divided in several non-overlapping patches which are flattened and linearly projected to obtain patch embeddings to which a positional embedding is added to keep track of the position of the embedding. This whole embedding is passed through a transformer encoder then to a MLP and finally through Linear layers to get the probabilities for the three categories.

The whole architecture of the model can be seen in FIGURE 2.15 and can be described as follows: the image $x \in \mathbb{R}^{H \times W \times C}$ is reshaped into a sequence of flattened 2D patches $x_p \in \mathbb{R}^{N \times (P \cdot P \cdot C)}$, where H is the height, W the width and C the number of channels of the original image, (P, P) is the resolution of a patch, and $N = \frac{H \cdot W}{P^2}$ is the number of patches and therefore also the length of the sequence. The flattened patches are then transformed into a patch embedding of size D thanks to a linear projection. Thereafter, a positional embedding is concatenated to the patch embedding to keep track of the position of the patch in the image. The resulting embedding is input into a Transformer encoder and its output is passed to a MLP. In the case of this work, the output of the MLP is passed into three linear layers to predict the three classes.

4.4 Wav2vec 2.0

As said by Lukasz Sus on its blog about Wav2vec: Wav2vec 2.0 introduced by Baevski et al. [2] is one of the State-of-the-art (SOTA) models for Automatic Speech Recognition (ASR) thanks to the use of self-supervised learning techniques which allow to first pre-train the model on large amount of unlabeled data and fine-tune it for a particular downstream task afterwards (as discussed in section 3). The training is split into two stages, the first stage is the self-supervised mode where unlabeled data is used to learn the best data representation. The only difference with what was explained before is that, in the case of wav2vec 2.0, data is audio data, so the model tries to learn the best speech representation possible. The second stage is the supervised fine-tuning or downstream task where labeled data is used for the model to predict what it is supposed to, like words or phonemes (smallest possible unit of sound that makes one word different from another word. For example, p, b, d, and t in the English words pad, pat, bad, and bat.). The two stages can be seen in Figure 2.16.

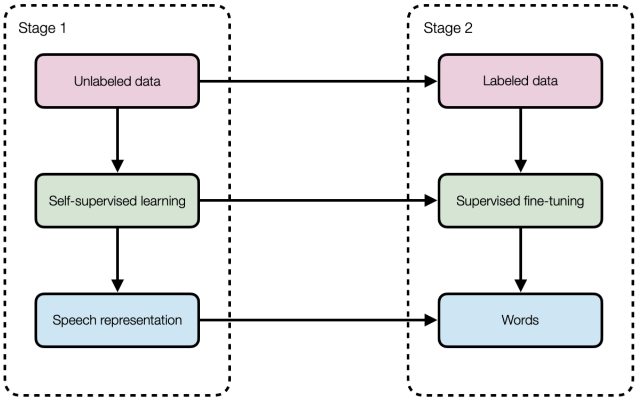


Figure 2.16: Illustration of the two stages of the Wav2vec training from Lukasz Sus in a blog on wav2vec. The first stage is the self-supervised mode where unlabeled data is used for the model to learn a good representation of the data which is speech representation here. The second stage is the supervised learning or downstream task where labeled data is used so that the model can predict what it is supposed to, which is to predict correct words here.

The most important part of this training is the first stage. If the model learns good speech representations, it can achieve state of the art results with very few labelled data. In the original paper, they pre-trained their model either on Librispeech corpus [24] without transcriptions containing 960 hours of audio or the audio data from LibriVox [17] containing more than 60k hours of audio. They then fine-tune the model on the Libri-light low-resource labeled data setups of 10 min, 1 hour, 10 hours and the clean 100h subset and using all 960 hours of LibriSpeech's labeled data. It was then tested on Librispeech dev/test sets and gave very good results even with only 10 minutes of data which is really not much data. The results can be found in Appendix Table 5.1.

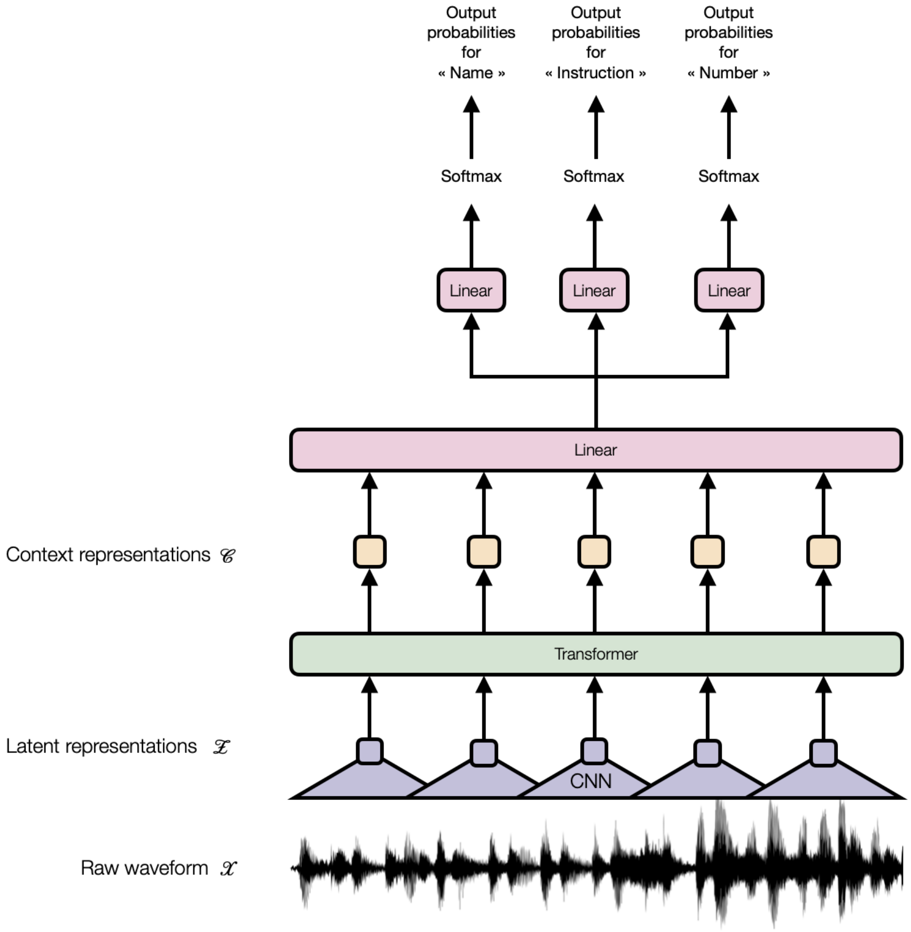

Output probabilities for « Name »
Output probabilities for « Instruction »
Output probabilities for « Number »
Softmax
Softmax
Softmax
Linear
Linear
Linear
Linear
Context representations $\mathcal{C}$
Transformer
Latent representations $\mathcal{Z}$
CNN
Raw waveform $\mathcal{X}$

Figure 2.17: Illustration of Wav2vec 2.0 architecture (Baevski et al. 2020 [2]) in this work. The input waveform $\mathcal{X}$ is passed through CNN's in order to get latent representations $\mathcal{Z}$. Then these latent representations are passed through transformers to obtain context representations $\mathcal{C}$ which are themselves linearly combined and passed through a softmax to finally get the output probabilities for each category.

The final architecture of the wav2vec 2.0 model used in this work after it has been fine-tuned can be seen in Figure 2.17 and is composed of the following elements: first the *raw waveform* $\mathcal{X}$ is given as input to the *feature encoder* which, as its name indicates, aims to reduce the dimensionality of the audio data to obtain *latent representations* $\mathcal{Z}$. This encoder is simply a 7-layer convolutional neural network. Then these latent representations are input in the *Transformer encoder* so as to obtain *contextualised representations* $\mathcal{C}$. This Transformer encoder is composed of 12 Transformer blocks for the BASE model and 24 for the LARGE model. Finally linear projections are done on these contextualised representations in order to obtain the final output probabilities for all three classes.

The key idea of the pre-training is similar to what is done in the BERT pre-training [9]. BERT focuses on Natural Language Processing (NLP) and is trained by simply masking part of the input text, then the model is asked to predict these masked words. Here, in the case of wav2vec 2.0, a part of the latent representations, which are the transformer's inputs, is masked and the goal could be to predict these masked latent feature vector representations z_t. Nevertheless, this is not the case here, as authors of the paper have improved this type of training thanks to contrastive learning.

But before digging into this contrastive learning, a few words must be said about why quantization is needed. A big challenge when using Transformers for speech data processing is that speech is continuous. Of course, speech has a high variance such as the speaker or background noise that may be present and this can be harmful to the training. In Natural Language Processing, text in any language can be easily discretised into words or sub-words and a finite vocabulary of discrete units can be created. Here, during the training, the output of the feature encoder is discretised to q_t with a quantization module $\mathcal{Z} \to \mathcal{Q}$ to represent the targets. As previously mentioned, the process of quantization is to convert values from a continuous space into a finite set of values in a discrete space. It is known that the number of phonemes in a language is finite, therefore a codebook containing all the different phonemes can be created. Then, the quantization can choose the right code word from the codebook. The only limitation is that even if it is finite, the number of all possible sounds is really large. In order to facilitate the training of Wav2vec 2.0, the authors have chosen to create G codebooks, each one composed of V codewords. The quantized representation is a linear projection of the concatenation of the best word from each codebook. This best word is chosen by using a modified softmax called *Gumbel Softmax* and comes with several upgrades, such as randomisation and temperature. Randomisation allows the model to choose different codewords and not only use a subset of them. Temperature enables to decrease the effect of this randomisation through time. In the original paper, authors used 2 codebooks with 320 codewords in each of them. This leads to a maximum of $320 \times 320 = 102,400$ speech units.

Contrastive learning has been explained before, however, hereby, for a latent representation z_t which is masked, the model should have a context representation c_t that is good enough so as to guess the correct quantized representation q_t among many others. This can be seen in FIGURE 2.18 and here is the formula in the case of Wav2vec 2.0:

$$\mathcal{L}_m = -\log \frac{\exp(sim(\mathbf{c}_t, \mathbf{q}_t)/\kappa)}{\sum_{\tilde{\mathbf{q}} \sim \mathbf{Q}_t} \exp(sim(\mathbf{c}_t, \tilde{\mathbf{q}})/\kappa)} \tag{2.7}$$

In FIGURE 2.18, it is represented as if only one latent representation was masked, however, in reality there is a certain proportion that is masked and all of them are replaced with the same trained feature vector. The input to the quantization module is of course not masked so as to obtain the correct quantized representation. To mask the latent speech representations, a proportion p of all time steps is randomly sampled without replacement. These are the starting indices, thereafter the M following time steps from every sampled index are also masked. This means that there may be overlaps. In the original paper of Wav2vec 2.0 [2] $p = 0.065$ and $M = 10$ which led to approximately 49% of all time steps to be masked.

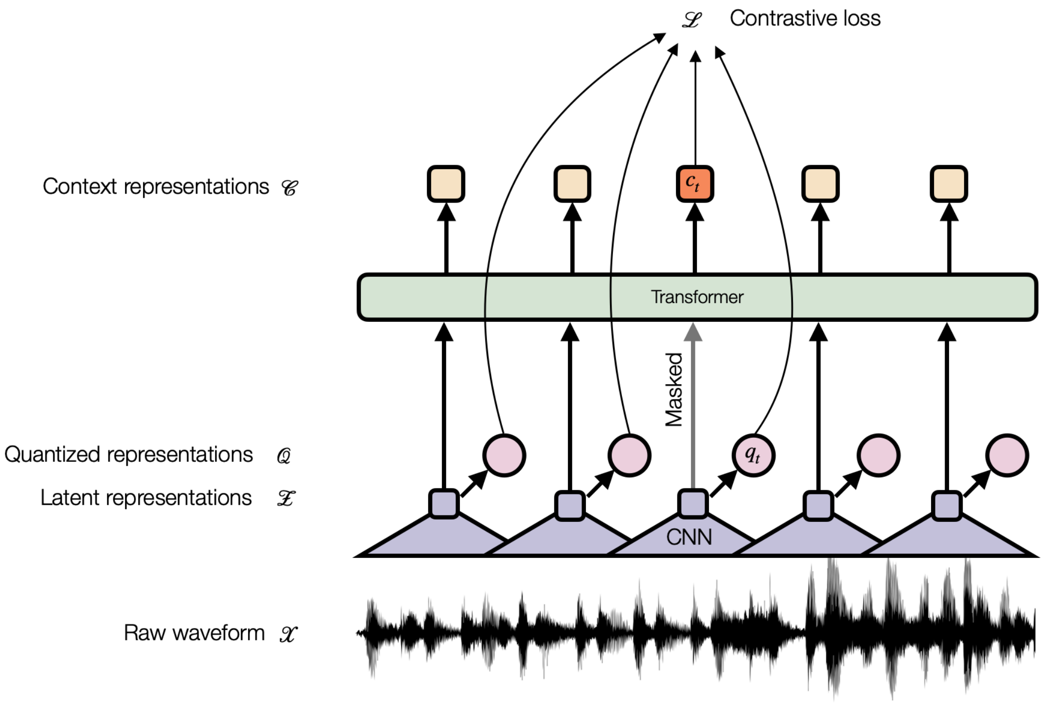

$\mathcal{L}$ Contrastive loss
Context representations $\mathcal{C}$
c_t
Transformer
Masked
Quantized representations $\mathcal{Q}$
q_t
Latent representations $\mathcal{Z}$
CNN
Raw waveform $\mathcal{X}$

FIGURE 2.18: Illustration of the wav2vec 2.0 (Baevski et al. 2020 [2]) framework which jointly learns contextualised speech representations and an inventory of discretised speech units by using contrastive learning. It masks some latent representations and tries to have the best context representation to guess the correct quantized representation among others.

During the pre-training stage, contrastive loss is not used alone. Another loss called *diversity loss* is added to the contrastive loss so that the model can use with the same probability all codewords V in each of the G codebooks and not only a subset of them. This is done by maximising the entropy of the Gumbel-Softmax distribution. This way, the model cannot always choose in a subset of all possible codewords. This loss can be express as follows:

$$\mathcal{L}_d = \frac{1}{GV}\sum_{g=1}^{G} -H(\overline{p}_g) = \frac{1}{GV}\sum_{g=1}^{G}\sum_{v=1}^{V}\overline{p}_{g,v}\log\,\overline{p}_{g,v} \tag{2.8}$$

The final objective of the pre-training is to learn representations of speech audio by using a contrastive learning augmented by the diversity loss:

$$\mathcal{L} = \mathcal{L}_m + \alpha\mathcal{L}_d \tag{2.9}$$

Chapter 3

Data

1 Speech Commands Dataset

Word	**No. of samples**	**Word**	**No. of samples**
Backward	1,664	No	-
Bed	-	Off	-
Bird	-	On	-
Cat	-	**One**	3,890
Dog	-	**Right**	3,778
Down	3,917	**Seven**	3,998
Eight	3,787	Sheila	-
Five	4,052	**Six**	3,860
Follow	-	**Stop**	3,872
Forward	1,557	**Three**	3,727
Four	3,728	Tree	-
Go	3,880	**Two**	3,880
Happy	-	**Up**	3,723
House	-	Visual	-
Learn	-	Wow	-
Left	3,801	Yes	-
Marvin	-	Zero	-
Nine	3,934	-	-

Table 3.1: List of words present in the *Speech Commands Dataset V2* by Warden, 2018 [37]. Bold words are words both present in the dataset and vocabulary presented in the problem statement.

In order to achieve the final goal, an algorithm that aims to recognise the different commands is needed and therefore a dataset is also needed. Unfortunately, there are no existing dataset containing audio files with the desired commands described in the problem statement. However, the open-source *Speech Commands Dataset* by Warden, 2018 [37] contains some of the words present in the vocabulary. The one used in this work is the second version of the dataset where more samples have been recorded and some words were added compared to the first version. The words available in this dataset are listed in Table 3.1, bold words are the ones present both in the dataset and in the vocabulary. The interesting words are present with their number of samples, the others are discarded. This dataset consists of 105,829 one-second utterances of 35 short words recorded by thousands of different people around the world on the AIY website.

2 Acquired data

This *Speech Commands Dataset* is a good starting point but some of the vocabulary words are still missing from it and had therefore to be collected through the web application created by Julien Bolland [4] once again based on Warden's work [37] which is also open source. People (mostly French speakers) were asked to record themselves in a quiet environment through this platform saying some of the vocabulary words missing from Warden's dataset. Only two words (*Backward* and *Forward*), already present in the Speech commands dataset, were requested on the platform because there were fewer of them than others in the dataset. Note that if the user wanted to stop the recording, discard a record or exit the platform, he could whenever he wanted. This unfortunately leads to unrecorded words and this is why the number of occurrences of the words that have been collected is not the same for all whether in this dataset or in the *Speech Commands Dataset*. The missing words from the vocabulary collected through the web platform and their respective number of samples can be found in the Table 3.2.

Word	**No. of samples**	**Word**	**No. of samples**
Alpha	873	**Free**	947
Backward	763	**Gamma**	918
Beta	901	**Land**	999
Delta	891	**Mission**	866
Emergency	745	**Speed**	879
Forward	811	-	-

Table 3.2: List of missing vocabulary words in the *Google Speech Dataset V2* and their respective number of samples collected through the web platform created by Bolland [4]

2.1 Data pre-processing

The number of samples presented in the two tables above is the final number of samples after a manual pre-processing. All audio files were listened to in order to eliminate mislabeled words, incomprehensible, non-complete words or even empty audios. Another interesting point is that the web platform sometimes creates for an unknown reason duplicate files (with different names) which were removed automatically during this pre-processing step simply by hashing the content of files. Then a list of hash was created, and all hash were checked to see if they were already present in the list or not.

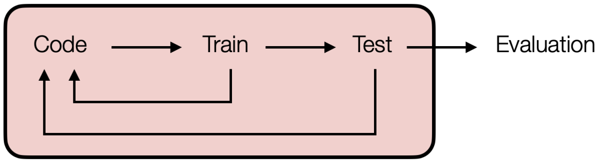


Figure 3.1: Biased training using the test set to fine-tune the model. Inspired from François Fleuret's deep learning slides.

After merging the two datasets described above, three folders were created so as not to bias the training: Training set of words, Validation set and finally Test set. On the one hand, indeed if the same word is used during training and testing, the results are biased as the model has already seen this exact same spoken word. On the other hand, three folders are created so as not to bias the final results of the evaluation of the model. The reason for this is that, without the validation set, the test set is used to fine-tune parameters, etc., as in Figure 3.1. Hence, the final evaluation is biased because the model has been changed to better ”fit” the test set. Adding the validation set no longer biases the test set since it is this validation set that is used to fine tune the model; therefore the test set has nothing to do with it; it is used once and only once to evaluate the model at the very end as can be seen in Figure. 3.2 The first folder is the training folder and contains 50% of the total number of words of each class. The second folder is the validation folder and contains 25% of the total number of words of each class and the third and last folder is the test folder that contains the last 25%.

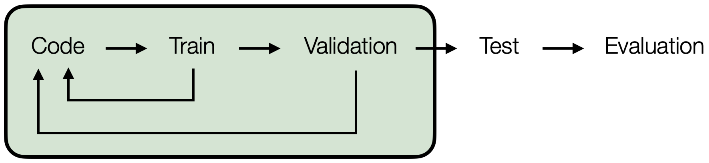


Figure 3.2: A good way to train the model without compromising the test set evaluation by using a validation set to fine-tune the model. Inspired from François Fleuret's deep learning slides.

3 Background noise dataset

The MS SNSD dataset [28] is used here. This dataset contains a large collection of clean speech files, that were discarded, because they were useless in this case, but it also contains various environmental noise files. These files are the interesting ones, However, there are background noises that are most unlikely to be heard in this project's application (sounds such as *airport announcements, munching, applause, air conditioner, ...*).
It was therefore been decided to add some real-life noise files found on the website mixkit.co such as *crowd sounds, sirens, cars sounds* and so on. All these noise files were cut into smaller files to match the length of the command files.

Chapter 4

Experiments

1 Metrics

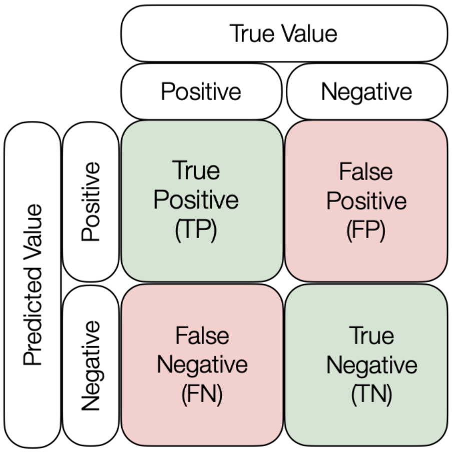


Figure 4.1: Illustration of a confusion matrix

A confusion matrix can be used to obtain information on the performances of an algorithm, such as the values for which the model is wrong and with which other values it predicts instead of the real value. It is useful as it can be used to calculate other metrics which are needed. Accuracy alone is not a good choice because: first, if there are more than two classes (for example, if your data has 10 categories), the model may have a classification accuracy of 90%. You have no idea whether all classes are predicted equally well, or whether one or two classes are wrongly predicted by the model. Second, if the dataset is unbalanced (for example, if you have two classes with 9 true values and only 1 false value), if the model always predicts "true", the model still has a classification accuracy of 90%. Though, in fact, the model is useless because it always predicts "true". Therefore, new metrics need to be defined in order to better assess the performance of the model.

- **Accuracy**: Accuracy represents the number of correct predictions out of the total number of predictions and it can be computed thanks to the confusion matrix. The range of values for the accuracy is between 0 and 1 where 1 is a perfect model that predicts everything correctly and 0 is a perfect inverse model where each prediction should be taken as the inverse of the true value (for binary classification). 0.5 is the worst accuracy score for binary classification as it means that every prediction is made at random.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \tag{4.1}$$

- **Precision**: Precision measures how many values predicted as Positives are relevant. Precision is therefore the number of "True Positives" divided by the total number of positive predictions. The value range is between 0 and 1 where 0 is a model that only makes "False Positive" predictions and 1 is the best model that does not make any "False Positive" predictions.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.2}$$

- **Recall**: Recall measures how many relevant positive values were predicted. It is the ratio between the number of "True Positives" to the total number of Positive samples which is the sum of "True Positive" and the "False Negative". The range of values is again between 0 and 1.

$$\text{Recall} : \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.3}$$

These two metrics are better suited to assess the performance of a model, than the accuracy alone. However, they are not always equally important. In certain cases, the focus is on one or the other. For example in medicine, whether for the detection of cancer or coronavirus, "False Negatives" absolutely have to be avoided absolutely because they can have really bad consequences. In this case, recall is a better measure than precision. In the case of email spam detection, if the model misses detecting a spam email it is not a big deal. Nevertheless if the model predicts that an important email is a spam email, it is slightly more serious. Here, precision is a better measure than recall.

Now, if there is a need of compromise between precision and recall, the F1-score can be used.

- **F1-score**: the F1-score is used to synthesise the precision and recall values and is defined as the harmonic average of these 2 values.

$$\text{F1-score} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = \frac{\text{TP}}{\text{TP} + \frac{\text{FN} + \text{FP}}{2}} \tag{4.4}$$

2 Evaluation Procedure

The hardware specifications of the computer that performed all the experiments can be found in the Appendix.

In order to evaluate the different models, a test set is needed. Words are available and can be grouped together in order to create commands. It is a combination of words spoken by different people. This is explained in an upcoming section, but this does not reflect reality. What was finally chosen as the final testing set is a real condition dataset. In this case, real conditions means that a command is spoken fully by one and only one person. Moreover, people were asked to record themselves in a noisy environment to represent the reality of the use case as much as possible. This means that noises heard in these audio files are noises from the real world that are effectively in the background of the people speaking. There aren't noises added during the processing of the audio. More than 50 people contributed to this dataset, recording at least 10 commands per person. The noise was measured behind them with a decibel meter application which isn't a professional instrument that accurately measures decibels but it gives an estimation. On average, there were background noises of 45 dB. Some were recorded in more extreme cases, such as the tram works in Liège, or with the song of cicadas behind them, or during the " Les Ardentes" music festival. The background noise, in some cases, reached 75-80 dB. At the end, this dataset is composed of **998** commands and the word count in each category can be found in Figures 4.2, 4.3 and 4.4.

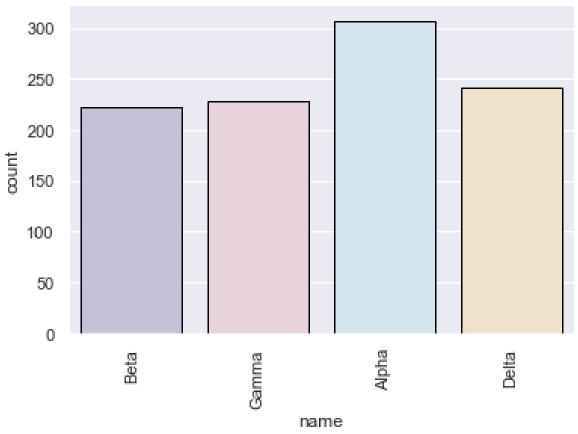


Figure 4.2: Count of the names in all the real conditions test commands set.

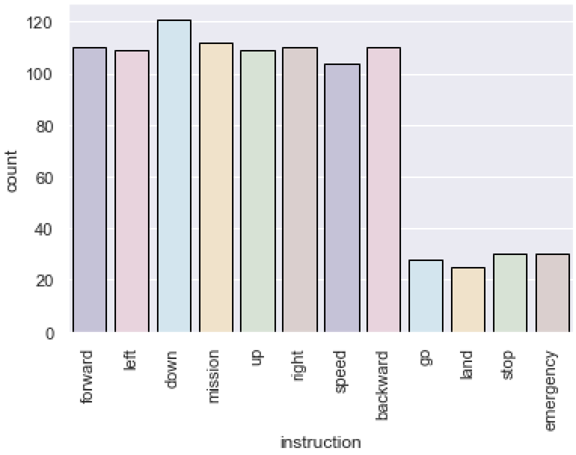


Figure 4.3: Count of the instructions in all the real conditions test commands set.

In the different models presented beforehand, all of them have 3 outputs from 3 different categories: a name, an instruction and a number. The metrics defined above are computed on the whole commands, not on each category. In the datasets, there are only true commands and no false commands, therefore there are no "True Negatives".

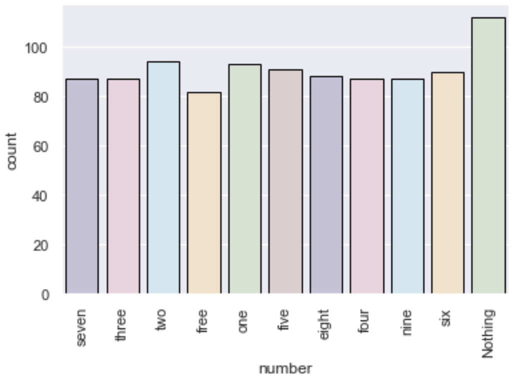


FIGURE 4.4: Count of the numbers in all the real conditions test commands set.

In order to compute metrics on complete commands, the three categories are predicted. When the three categories are well predicted the command is therefore classified as a ”True Positive”, but when one of the three categories is wrongly predicted, the command is considered as a ”False Positive”. When a certain threshold is defined, the three categories must satisfy this threshold. If one of them does not satisfy it, the command is rejected and is therefore considered as a ”True Negative”. This way, there is a binary classification on the complete commands: ”True” commands and ”False” commands. The fact that there are no ”True Negatives” commands is the reason why other metrics (such as specificity which measures the proportion of ”True Negatives” that are correctly identified by the model) are a bit useless in this case.

3 Ablation study

In this section, several experiments are performed to find the optimal models, to check the robustness of these models against noise, to check if augmentation is useful, etc.

3.1 Artificial voices commands dataset

Before using the words that were collected, it was first decided to train the model by only using a generated spoken command dataset. This dataset was generated using text-to-speech models from two different platforms. Text-to-speech models are simply Deep Learning models that take some written text as input and, as output, an audio file containing the spoken text that was input.

The first idea to create this dataset of commands was to generate these commands thanks to Microsoft text-to-speech service, part of Azure Cognitive Services. Microsoft Azure text-to-speech has an API to generate realistic speech from text in different languages with different voices, styles, emotions and tones. In the context of this work, only English voices were used and more precisely English neural voices. Neural voices are voices that are difficult to differentiate from real voices. The API contained 40 neural voices with different accents (Australia, US, India, Nigeria, UK, ...) at that time, although new accents are added on a regular basis. The 336 commands multiplied by the number of voices gives a dataset of **13,440 commands**.

The idea is exactly the same as for the Azure API but it is done with the text-to-speech API from Google Cloud. In this case, the voices that sound real are tagged as "WaveNet" and not "Neural" as in Azure and there are 23 English voices of this type once again with different accents. The size of this second dataset produced with this API is **7,728 commands**.

In both cases, it was decided to slightly slow down the speaking rate, so that the command sounds like a command to the ear and not like spoken sentences where the words flow much faster.

All commands, whether using APIs for artificial voice commands, or commands created with human voices, (explained below), have been extended to a length of 4.6 seconds so that they are uniform. This was chosen based on the longest artificial command that is the "Alpha backward seven" a command spoken by a voice mimicking a child in the Google API which is 4.52 seconds long. This extension was done during the pre-processing phase of the data. For the real conditions data, the waveforms are simply padded during the `__getitem__` function of the dataset class as if it were a transformation of the data. For all data modified during pre-processing, padding is done as follows: first, the length of the audio file is checked, as audio files are asked to be of fixed length (4.6 seconds), the length of the audio is subtracted from 4.6 to know how many seconds of empty noise have to be added. Then this new duration is divided randomly into 2 smaller durations and empty noise of a duration corresponding to the first of the 2 is added in front of the command audio file and the same is done for the second part, but it is added at the end of the command audio file.

Results of Artificial voices commands dataset

The first experiment was to train the models with this dataset only made of artificial voices and no real human voices. The results for the two models can be found in TABLE 4.1 below. For the Wav2vec model the results are already quite good, but for the ResNet model the results are really very bad. Then by looking at the different confusion matrices for the 3 categories of the Wav2vec 2.0 model, which you can see in the appendix in FIGURES 5.8, 5.9 and 5.10, it can be seen that the model struggles with some words but not particularly with specific words. The fact that the Wav2vec model is so much better than the ResNet model is probably due to self-supervised learning phase where the model trains itself to recognise the different phonemes on a very large corpus of spoken sentences with different accents, speaking rate, microphones quality and so on. So, even without much data during training on the downstream task, it can already perform well. For the small errors that remain, one intuition could be that the commands of the real conditions test set were mostly spoken by French-speaking people and in reality, people's diction is not as good as native English speakers or that of a text-to-speech model such as those of Google and Microsoft.

On the other hand, the ResNet bad results can be seen in TABLE 4.1 and in confusion matrices 5.11 5.12 and 5.13. These results may be explained by the fact that ResNet has not been pre-trained on spectrograms, but on real images from the ImageNet dataset [29]. So in fact, it is like it was trained from scratch. Several experiments were performed to highlight this like training a non-pretrained ResNet which gives approximately similar results. In addition to that, as mentioned for the Wav2vec model, artificial voices do not represent the reality as they are too perfect.

Training dataset	**Model**	**Precision**	**Recall**	**F1-Score**
artificial voices alone	Wav2vec 2.0	0.9178	1.0	0.9572
	ResNet-18	0.0862	1.0	0.1587

TABLE 4.1: Results on the real conditions Test set of the models trained on the artificial voices commands dataset.

3.2 Human voice commands dataset

Another experiment is to train the model with a dataset only made of human voices and no artificial voices to see if the results are better or not than with only artificial voices, because the real conditions test set only consists of real voices. Words are available since they were collected before, though there are only words, not commands. In order to create a command, words have to be combined together. If various commands are produced through the created language (see problem statement), the dataset is unbalanced. Indeed, there are 20 times less commands with an instruction without a number than commands with n instruction and numbers. For example there is only one command with the instruction without number *"land"* together with the name *"Alpha"*, but there are 10 commands with the instruction with numbers *"up"* together with the name *"Alpha"* because there are ten numbers : *"Alpha up one"*, *"Alpha up two"*, until *"Alpha up free"*. As there are, for a single name, only 4 instructions without a number but 80 instructions with numbers (8 instructions with a number multiplied by 10 numbers), there are 20 times less commands without a number than commands with a number. In the case of the human voice dataset, the commands are created randomly but follows this distribution. Firstly, a random name is chosen and an audio file is selected randomly in the corresponding folder. Then an instruction is chosen at random, and there is a 5% chance that the chosen instruction is one without a number.

If this is the case, there is no need to choose a number. In the other case, an instruction with a number and an additional number are chosen at random as the corresponding files. The FIGURES 4.5, 4.6 and 4.7 represent the count of the words, for each category, in all generated commands with human voices for the training dataset. It can be seen that the distribution is respected as there are about $4 * 170$ instructions without number for $8 * 1600$ instructions with a number. This is approximately the 4/80 that were expected, as explained above.

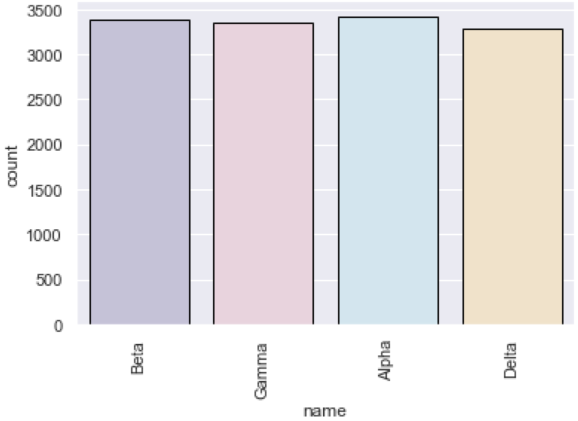


FIGURE 4.5: Count of the names in all the human voices training commands set.

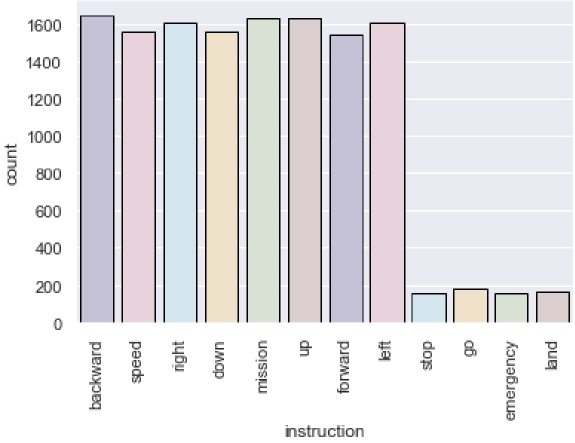


FIGURE 4.6: Count of the instructions in all the human voices training commands set.

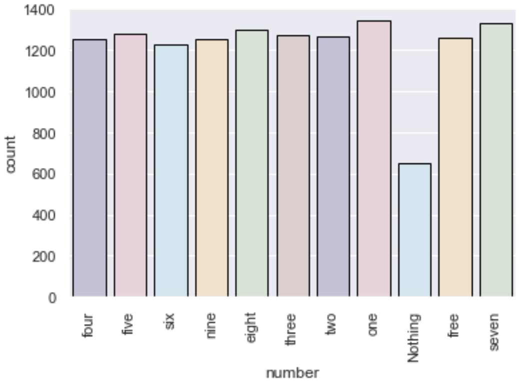
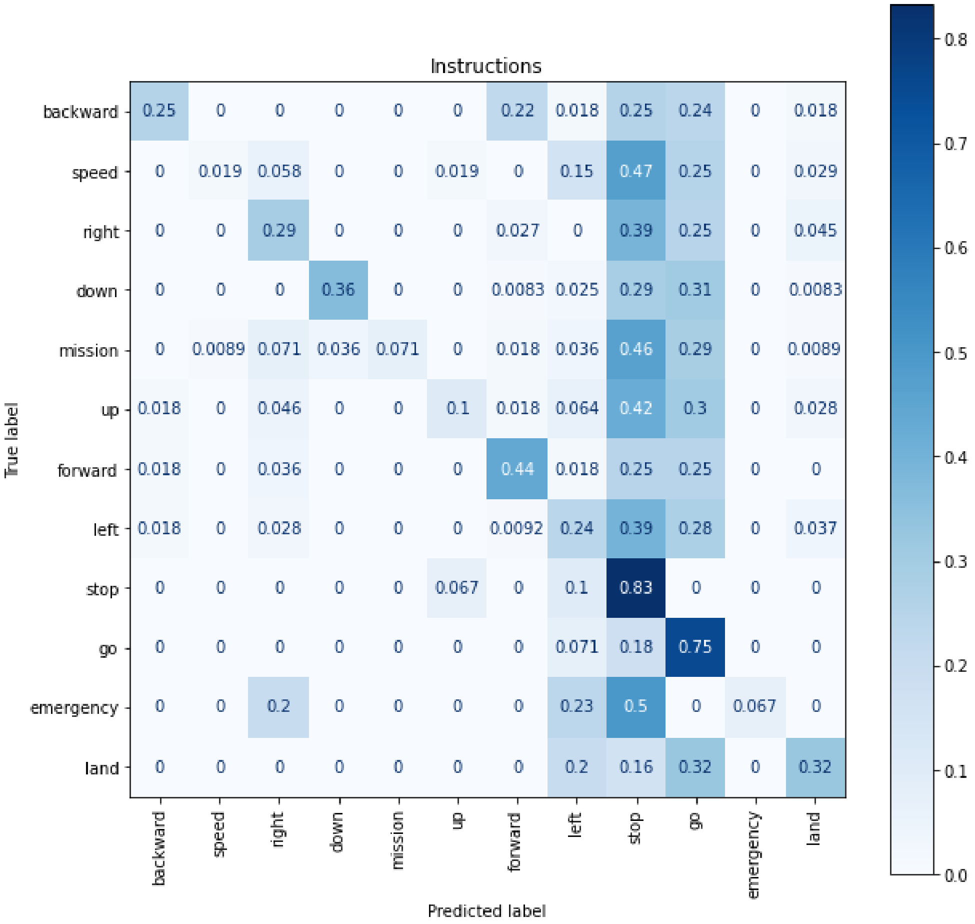


FIGURE 4.7: Count of the numbers in all the human voices training commands set.

It was decided to have the same number of artificial voices commands as human voice commands in the training and validation sets. For that, **13,440** commands were created for the training set which is equal to the 13.440 commands created, thanks to the Azure API. **7,728** commands were created for the validation set, which is equal to the 7,728 commands created hereby, thanks to the Google cloud API. The only difference is that there are no artificial voices in the test set and again 12,144 commands were created for this one.

Results of human voice commands dataset

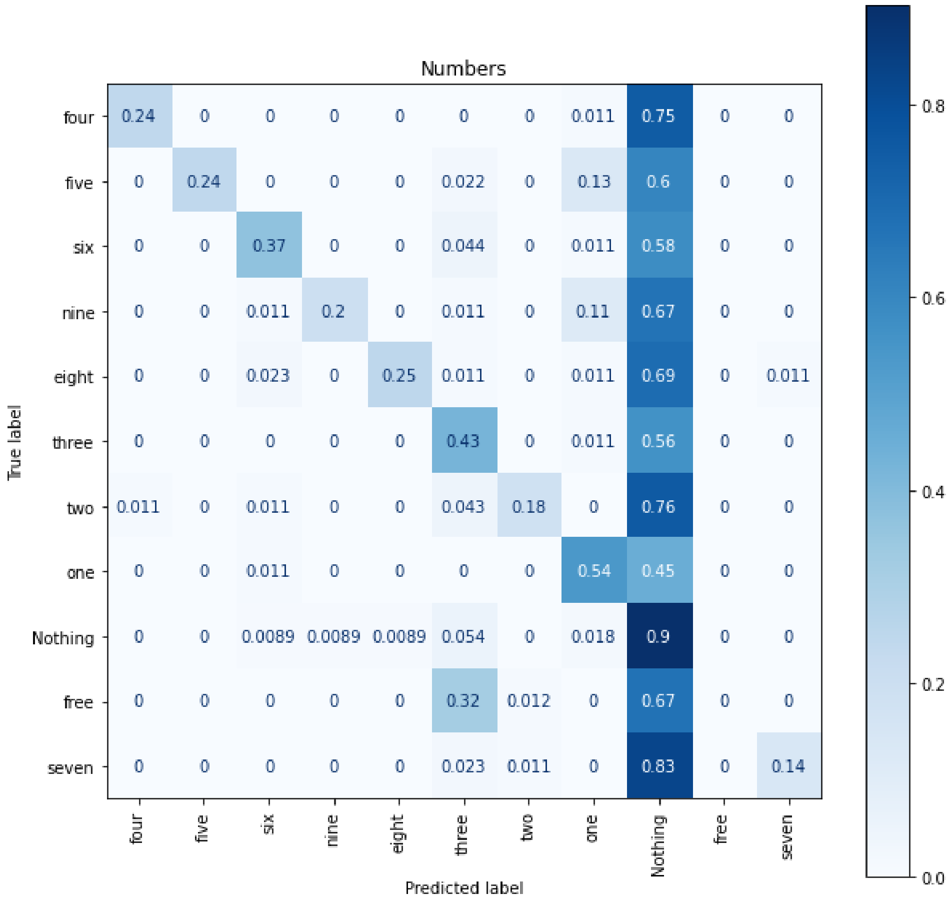
The results are in Table 4.2 and they are rather bad for both the wav2vec model and the ResNet model. The two models struggle with the commands with no number.



	backward	speed	right	down	mission	up	forward	left	stop	go	emergency	land
backward	0.25	0	0	0	0	0	0.22	0.018	0.25	0.24	0	0.018
speed	0	0.019	0.058	0	0	0.019	0	0.15	0.47	0.25	0	0.029
right	0	0	0.29	0	0	0	0.027	0	0.39	0.25	0	0.045
down	0	0	0	0.36	0	0	0.0083	0.025	0.29	0.31	0	0.0083
mission	0	0.0089	0.071	0.036	0.071	0	0.018	0.036	0.46	0.29	0	0.0089
up	0.018	0	0.046	0	0	0.1	0.018	0.064	0.42	0.3	0	0.028
forward	0.018	0	0.036	0	0	0	0.44	0.018	0.25	0.25	0	0
left	0.018	0	0.028	0	0	0	0.0092	0.24	0.39	0.28	0	0.037
stop	0	0	0	0	0	0.067	0	0.1	0.83	0	0	0
go	0	0	0	0	0	0	0	0.071	0.18	0.75	0	0
emergency	0	0	0.2	0	0	0	0	0.23	0.5	0	0.067	0
land	0	0	0	0	0	0	0	0.2	0.16	0.32	0	0.32

Figure 4.8: Confusion matrix for the "Instruction" category of the ResNet model trained only on human voices and with commands that are only part of the grammar.

By looking at the confusion matrices on the instructions for the ResNet in Figure 4.8 and for the wav2vec in appendix Figure 5.17, it can be seen that instructions without a number are not well predicted. The same can be seen for the "Nothing" word again for both models in Figure 4.9 and Figure 5.18 in the appendix. This "word", which is more the absence of a number in a command, is obviously only present in these commands. This result can be expected, since the dataset is unbalanced. As explained at the beginning of this section, there are far fewer commands with no numbered instructions, than commands containing a numbered instruction. An interesting result is that, compared to training with artificial voices only, the ResNet gains in terms of accuracy while the Wav2vec loses a lot and becomes completely unusable. This result can be due to the fact that the test set is a real conditions dataset and therefore each command is spoken by one and only one person. The fact that here, the model is trained on commands assembled with 2 or 3 different voices depending on the type of command can harm the model.



	four	five	six	nine	eight	three	two	one	Nothing	free	seven
four	0.24	0	0	0	0	0	0	0.011	0.75	0	0
five	0	0.24	0	0	0	0.022	0	0.13	0.6	0	0
six	0	0	0.37	0	0	0.044	0	0.011	0.58	0	0
nine	0	0	0.011	0.2	0	0.011	0	0.11	0.67	0	0
eight	0	0	0.023	0	0.25	0.011	0	0.011	0.69	0	0.011
three	0	0	0	0	0	0.43	0	0.011	0.56	0	0
two	0.011	0	0.011	0	0	0.043	0.18	0	0.76	0	0
one	0	0	0.011	0	0	0	0	0.54	0.45	0	0
Nothing	0	0	0.0089	0.0089	0.0089	0.054	0	0.018	0.9	0	0
free	0	0	0	0	0	0.32	0.012	0	0.67	0	0
seven	0	0	0	0	0	0.023	0.011	0	0.83	0	0.14

Figure 4.9: Confusion matrix for the ”Number” category of the ResNet model trained only on human voices and with commands that are only part of the grammar.

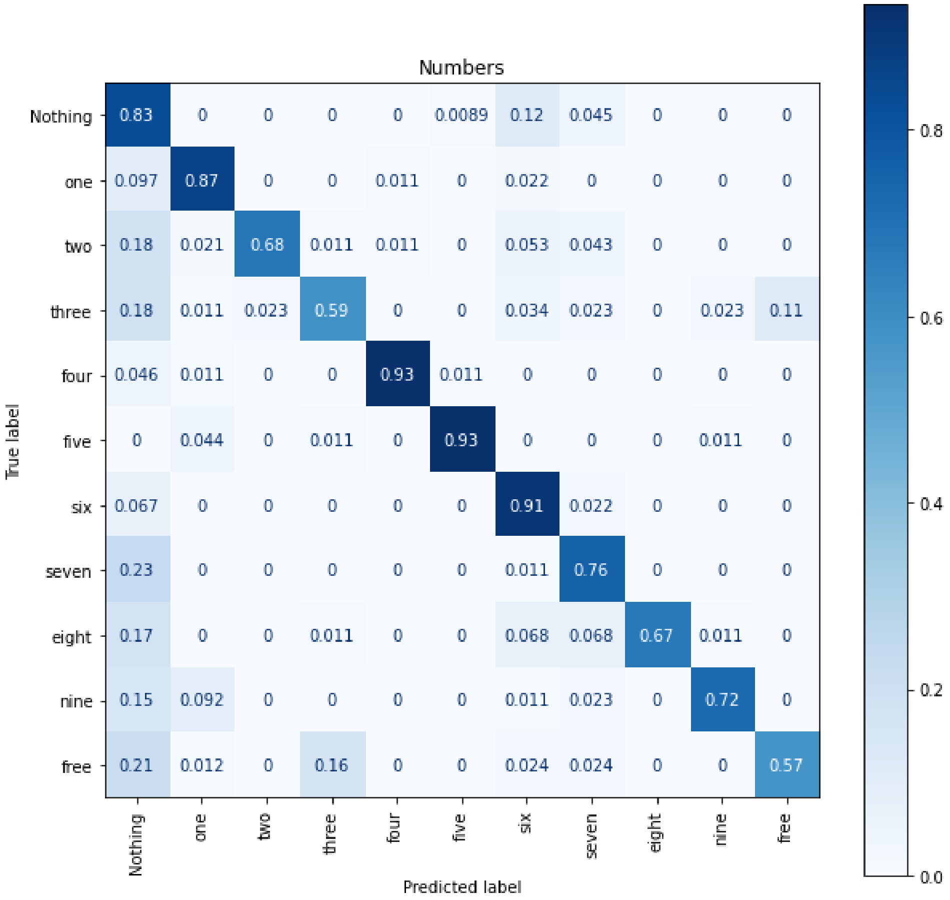
Training dataset	Model	Precision	Recall	F1-Score
human voices alone	Wav2vec 2.0	0.3818	1.0	0.5526
	ResNet-18	0.1884	1.0	0.3170

Table 4.2: Results on the real conditions Test set of the models trained on the human voices commands dataset.

3.3 Mixed commands dataset

To see if the results could be improved, a decision was taken to combine the two datasets presented above.

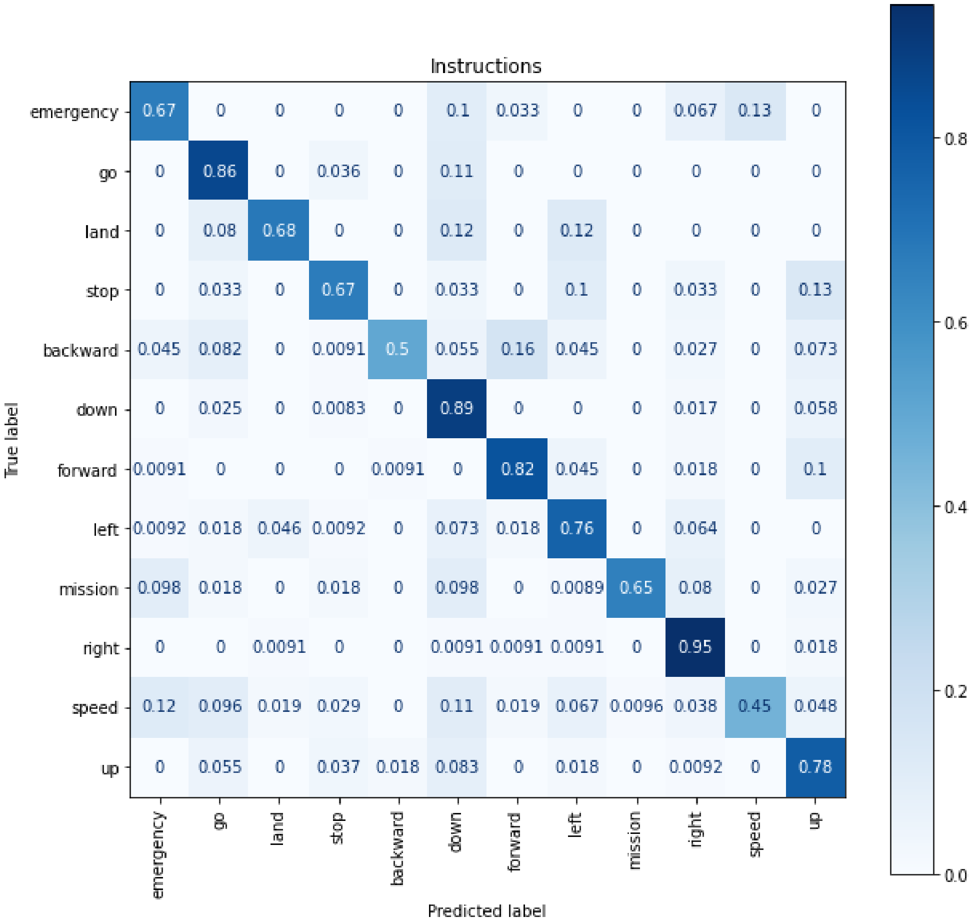
Results of the mixed commands dataset



	Nothing	one	two	three	four	five	six	seven	eight	nine	free
Nothing	0.83	0	0	0	0	0.0089	0.12	0.045	0	0	0
one	0.097	0.87	0	0	0.011	0	0.022	0	0	0	0
two	0.18	0.021	0.68	0.011	0.011	0	0.053	0.043	0	0	0
three	0.18	0.011	0.023	0.59	0	0	0.034	0.023	0	0.023	0.11
four	0.046	0.011	0	0	0.93	0.011	0	0	0	0	0
five	0	0.044	0	0.011	0	0.93	0	0	0	0.011	0
six	0.067	0	0	0	0	0	0.91	0.022	0	0	0
seven	0.23	0	0	0	0	0	0.011	0.76	0	0	0
eight	0.17	0	0	0.011	0	0	0.068	0.068	0.67	0.011	0
nine	0.15	0.092	0	0	0	0	0.011	0.023	0	0.72	0
free	0.21	0.012	0	0.16	0	0	0.024	0.024	0	0	0.57

Figure 4.10: Confusion matrix for the "Number" category of the ResNet model trained on the mixed commands dataset with commands that are only part of the grammar.

The results for the two models can be found in TABLE 4.3 and in both cases this new dataset leads to better results which means that coupling the 2 types of voices (human and artificial) is a good thing. These improved results may be explained by the fact that artificial voices are too perfect: there are only 40 different voices in the training dataset, people speak clearly even with their accent and the quality of the audio is perfect. This is not the case when you have real recorded human voices. The human recorded dataset not only brings a large diversity of voices, accents and microphones qualities, but also heterogeneity where artificial voices bring uniformity to the dataset with commands spoken clearly by only one person.



	emergency	go	land	stop	backward	down	forward	left	mission	right	speed	up
emergency	0.67	0	0	0	0	0.1	0.033	0	0	0.067	0.13	0
go	0	0.86	0	0.036	0	0.11	0	0	0	0	0	0
land	0	0.08	0.68	0	0	0.12	0	0.12	0	0	0	0
stop	0	0.033	0	0.67	0	0.033	0	0.1	0	0.033	0	0.13
backward	0.045	0.082	0	0.0091	0.5	0.055	0.16	0.045	0	0.027	0	0.073
down	0	0.025	0	0.0083	0	0.89	0	0	0	0.017	0	0.058
forward	0.0091	0	0	0	0.0091	0	0.82	0.045	0	0.018	0	0.1
left	0.0092	0.018	0.046	0.0092	0	0.073	0.018	0.76	0	0.064	0	0
mission	0.098	0.018	0	0.018	0	0.098	0	0.0089	0.65	0.08	0	0.027
right	0	0	0.0091	0	0	0.0091	0.0091	0.0091	0	0.95	0	0.018
speed	0.12	0.096	0.019	0.029	0	0.11	0.019	0.067	0.0096	0.038	0.45	0.048
up	0	0.055	0	0.037	0.018	0.083	0	0.018	0	0.0092	0	0.78

FIGURE 4.11: Confusion matrix for the "Instruction" category of the ResNet model trained only on the mixed dataset with commands that are only part of the grammar.

By looking at the confusion matrix of the instruction in the wav2vec model in FIGURE 5.19, it can be seen that the biggest errors that the model still makes are on the instructions without number. Then by looking at the model's confusion matrix on the numbers FIGURE 5.20, one can see that it is focused on the word "free" and "three". This can be expected, as these two words are really close in terms of hearing and understanding.

Training dataset	Model	Precision	Recall	F1-Score
mixed	Wav2vec 2.0	0.9739	1.0	0.9868
	ResNet-18	0.6052	1.0	0.7541

TABLE 4.3: Results on the real conditions Test set of the models trained on the dataset which is a mix of both human and artificial voices.

The ResNet model not only has the same problem as Wav2vec model, which is to differentiate between "free" and "three", but it also has a bigger problem which is to spot when there is no number at the end, as can be seen in FIGURE 4.10. For the instructions in FIGURE 4.11, one can be seen that it is a bit more disorganised.

3.4 Biased mixed commands dataset

Now, it is known that the results are better when the models are trained on the mixed commands dataset which includes both human and artificial voices. As mentioned before, it was observed that there is a problem with the 4 instructions that do not have a number after them. As explained beforehand, the training dataset is not balanced and here, the idea is to bias the dataset in order to overcome this problem. To do that, the two lists of instructions are merged into one, thus one doesn't have to be concerned whether it was a numbered instruction or not. That is to say, each instruction can include a number after it, or not. The dataset is therefore only biased during training and validation but not during testing. This is a kind of augmentation which is implemented to overcome the fact that the dataset is unbalanced. A *"Nothing"* word is therefore added in the list of numbers to indicate that there is no number in a given spoken command. Now, there is only one list of instructions [*emergency, go, land, stop, backward, down, forward, left, mission, right, speed, up*] and the list of numbers is now [*one, two, three, four, five, six, seven, eight, nine, free, Nothing*]. It is thus possible to use for instance *"Alpha land five"* or *"Alpha up"*, neither of which are in the language, but they help the model by balancing the training data. With these new lists of words, there are 4 names, 12 instructions and 11 numbers which give 528 different commands. In the case of the human voices dataset, as previously mentioned, the commands are created randomly. This means that, in the case of the biased dataset, each word in a command is chosen randomly with the same probability so at the end they should appear approximately the same number of times. The FIGURES 4.12, 4.13 and 4.14 represent the count of the words, for the different categories, in all commands of the hand crafted human voices training dataset. As expected, there are nearly the same number of words in each category. You can find the same FIGURE for the validation and test sets in the section Dataset of the Appendix.

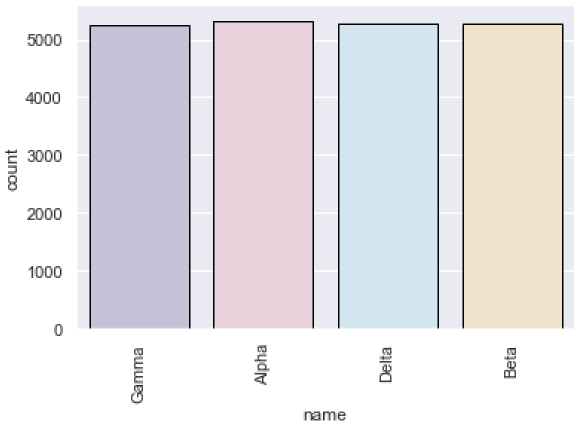


FIGURE 4.12: Count of the names in all the train hand-crafted commands set.

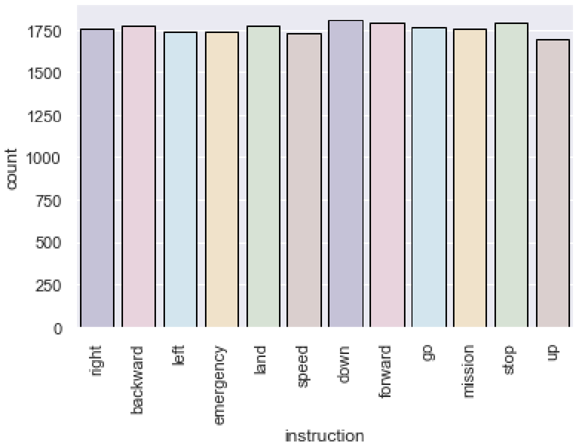


FIGURE 4.13: Count of the instructions in all the train hand-crafted commands set.

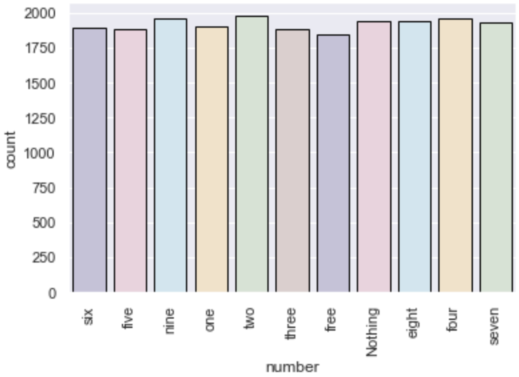


FIGURE 4.14: Count of the numbers in all the train hand-crafted commands set.

Results of the biased mixed commands dataset

The results of the biased mixed commands dataset can be found in TABLE 4.4. Before having the test set in real conditions, the test set consisted of commands created with human voices as explained in Subsection 3.2 and this biased mixed dataset helped the model to generalise better. Now, it can be seen that when the dataset is biased, the results deteriorate.

Training dataset	Model	Precision	Recall	F1-Score
biased mixed	Wav2vec 2.0	0.9319	1.0	0.9647
	ResNet-18	0.5601	1.0	0.7180

TABLE 4.4: Results on the real conditions Test set of the two models trained on the biased dataset as explained in produced datasets section

3.5 Augmentation

The results using the wav2vec model on the whole non-noised dataset are already of good quality. However, for the ResNet model, the results are not great and by looking at the "Number" category confusion matrix in Figure 4.10, it can be seen that the model predicts the number *"Nothing"* when it shouldn't (vertical "Nothing" line). This means that it thinks that there are more commands without a number than there really are.

In an attempt to correct this problem and to try to improve the model results, noise is added to the background. This is a form of augmentation and allows the model to generalise a little more and not overfit. Here, every command is augmented with a certain quantity of a random noise file as follows:

$$(1\text{ - quantity}) * \text{command\_file} + \text{quantity} * \text{noise\_file}$$

This noise is different for each epoch of the training to have a real augmentation. Firstly, this helps because it is similar to "generating" more data as the same instruction has a different background noise every new epoch. Secondly, this helps the model to be more robust to noise. If it is not trained with noise and then tested with noise, it will be bad at the task.

Another augmentation that can be used when working with spectrograms is called *"SpecAugment"* introduced by Park et al. (2019) [25]. This simple augmentation is basically applied on the input spectrogram of the CNN's or DNN's. This method is simple and computationally cheap to apply, as it directly acts on the spectrogram as if it were an image. The 2 modifications used are time and frequency masking, where either a block of consecutive time steps is masked, or some frequency channels are masked. An example of this augmentation is shown below in Figures 4.15, 4.16 and 4.17. This augmentation is applied to every spectrogram but in the `torchaudio` API used throughout the project, a maximum possible length of the mask for frequency and time (*freq_mask_param* and *time_mask_param*) must be chosen and indices are then uniformly sampled from $[0, \mathit{freq\_mask\_param})$ and $[0, \mathit{time\_mask\_param})$

The results of the noised training dataset can be found on Table 4.5. It can be seen that this really helps reduce the error rate in both models. The Wav2vec model goes from an error rate of 7% to only 2%, whereas the ResNet model goes from an error rate of 45% to 25%. Moreover, what can also be seen on the last line of the Table is that augmentation on the ResNet model further improves results. The precision is improved by 3%. In addition to that, it can be observed that the biased dataset helps the Wav2vec model improve results.

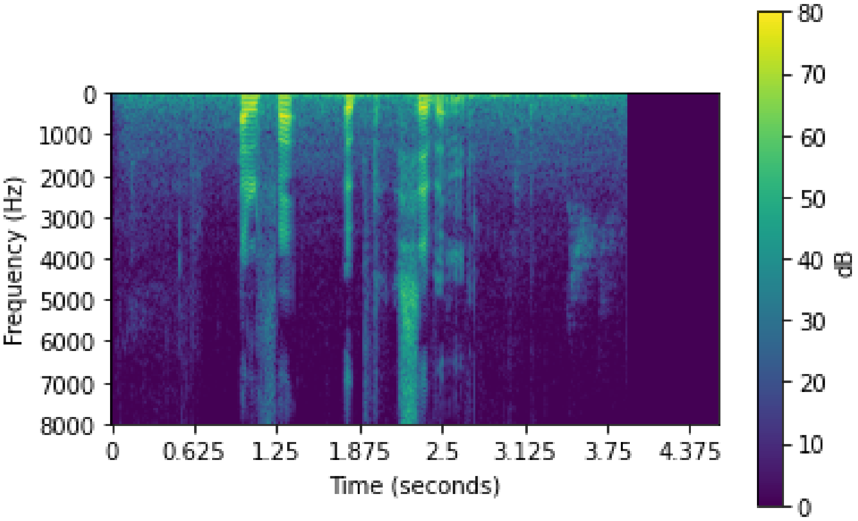


Figure 4.15: Simple spectrogram of a command within the dataset without any augmentation.

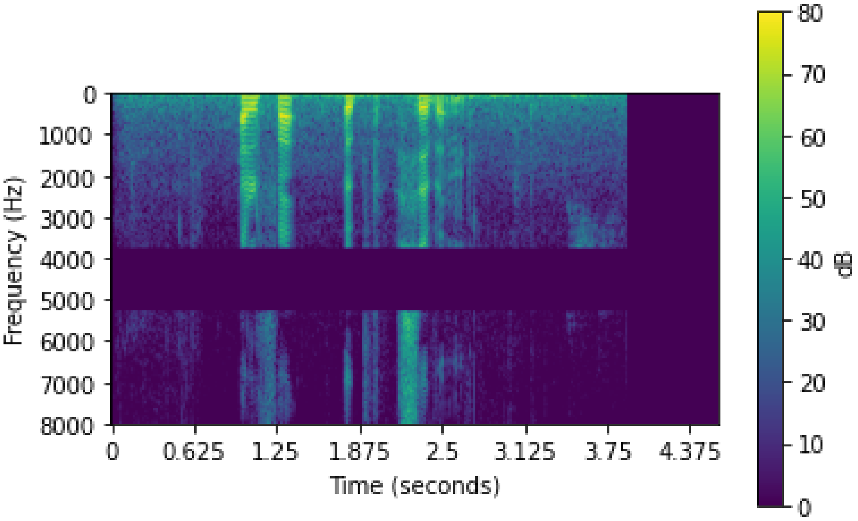


Figure 4.16: Spectrogram of the same commands as above but with frequency masking.

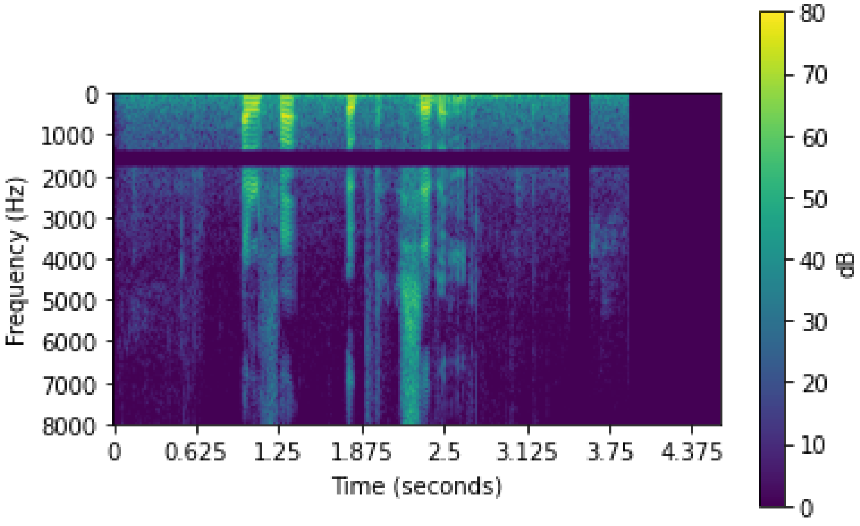


FIGURE 4.17: Spectrogram of the same command as above but with frequency and time masking.

Training dataset	Model	Precision	Recall	F1-Score
noised biased mixed	Wav2vec 2.0	0.9790	1.0	0.9894
	ResNet-18	0.7255	1.0	0.8409
noised non-biased mixed	Wav2vec 2.0	0.9760	1.0	0.9878
	ResNet-18	0.7445	1.0	0.8535
augmented noised non-biased mixed	ResNet-18	0.7735	1.0	0.8723

TABLE 4.5: Results on the real conditions Test set of the two models trained on the biased and non-biased dataset augmented with 20% of noise. The ResNet was also trained on the noised non-biased dataset augmented through SpecAugment.

3.6 Comparison between the models

Several models were trained to see which one leads the best results. The different training losses for the 3 categories can be seen in FIGURES 4.18, 4.19 and 4.20. The accuracies for the validation set can be found in the appendix to FIGURES 5.21, 5.22 and 5.23.

First, what can be noted here is that, for the ViT model, the loss decreases, but not as much as for the other models. Training was been continued, but doesn't appear on the graph. During continuation, it can be seen that the loss becomes a straight horizontal line, even if the learning rate is reduced, there is no longer any real progress. This result may be explained by the fact that, as mentioned previously, spectrograms are so different from images that the model has to be trained from scratch and, as specified in the paper, more than 14 million images are necessary to obtain SOTA results of the ResNets and other models. Here, the model may need less than 14 million images because there is less diversity in the spectrograms than in ”normal” images. Nevertheless, it can be seen that the images in this dataset are not numerous enough to reach outstanding results.

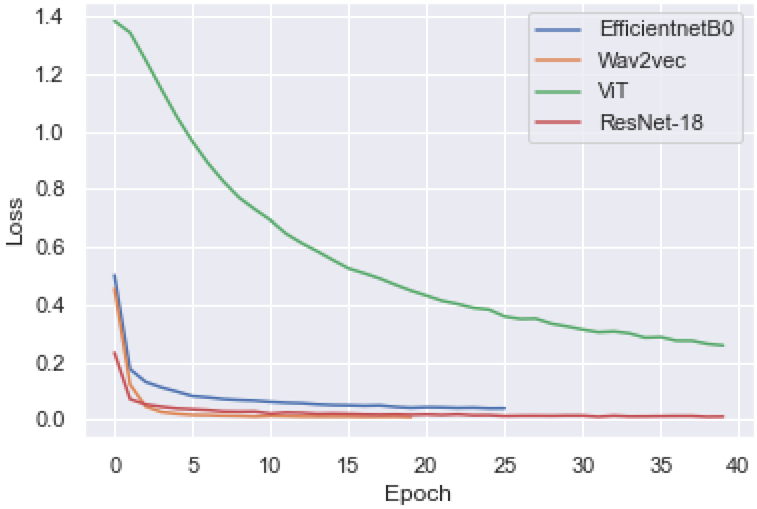


Figure 4.18: Training loss of the different best models for the ”Name category.

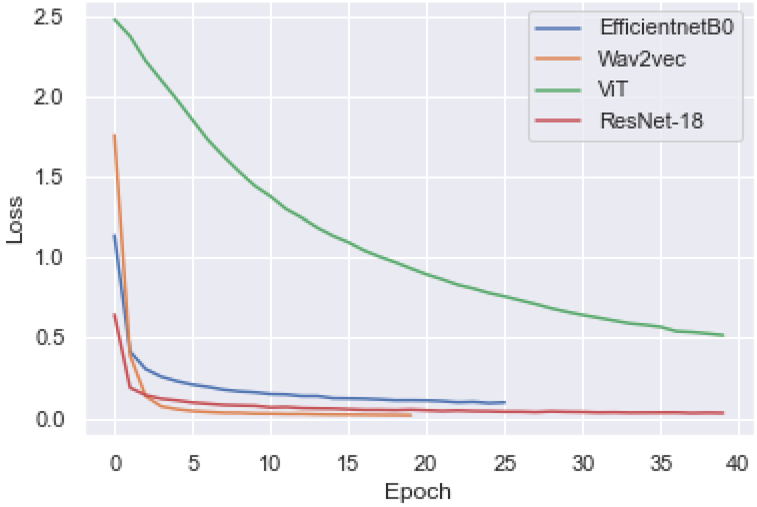


Figure 4.19: Training loss of the different best models for the ”Instruction” category.

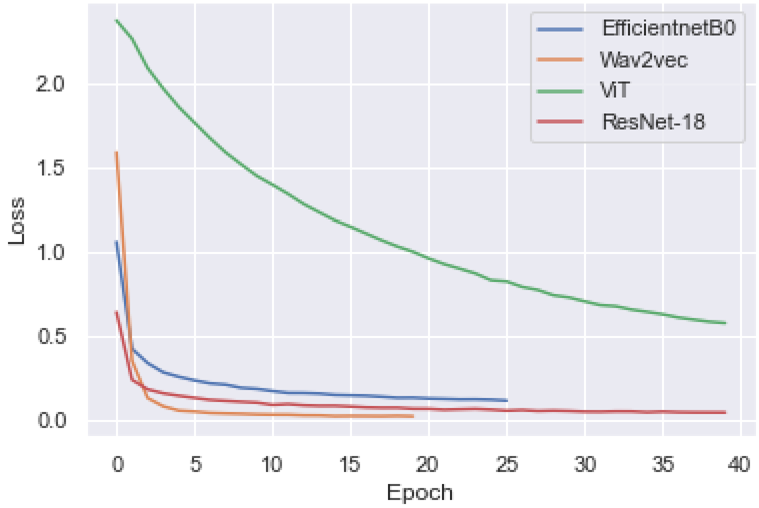


FIGURE 4.20: Training loss of the different best models for the "Number" category.

Second, when looking at the EfficientNetB0 training loss and validation accuracies, it can be seen that this model doesn't lead to better results than the ResNet model. Therefore, it was decided to continue the different experiments with the ResNet as the CNN architecture model as this is the fastest to train. The Wav2vec 2.0 was also kept as this is the one that provides the best results in all the models.

3.7 Normalisation

When listening to the files of the commands that were mis-predicted during the training, what was noticed was that the volume of the voice was very low compared to that of the noise, so all the voice files were normalised to avoid this problem.

3.8 With/without free

Since the beginning of the Experiments chapter, it can be seen in every confusion matrix on "Number" that each one of the models has big problems with the two words "free" and "three". This is normal because, as aforementioned, all the commands in the test set were recorded by French speakers, and with a French accent, the 2 words can be homonyms. Homonyms are 2 words that have the same spelling, or the same pronunciation, but have different meanings. Fortunately, most of these homonyms do not appear in the same sentence at the same place and thanks to the context of the sentence the right word can be selected.

The challenge here is that there is no context as these are commands. The two words, "free" and "three", belong at the same place and can occur with the same probability. If longer sequences, or with NLP's were used, the context of the sentence could easily be used, but here it's impossible. So, in the end, it was decided to simply delete the word "free" from the vocabulary.

The results can be found in Table 4.6 below:

Training dataset	Model	Precision	Recall	F1-Score
noised biased norm. w/o free	Wav2vec 2.0	0.9891	1.0	0.9945
noised non-biased norm. w/o free augm.	ResNet-18	0.7849	1.0	0.8795
noised biased norm. w/o free augm.	ResNet-18	0.8002	1.0	0.8890

Table 4.6: Results on the real conditions Test set of the models trained on biased or non-biased, augmented with SpecAugment in the case of the ResNet, and augmented with 20% of noise for both models, normalised datasets without the word "Free".

3.9 Quantity of Noise

Now, several quantities of noise are tested on the wav2vec model to see which one gives the best results without a classification threshold. As can be seen in Table 4.7, the best results are obtained when the quantity of noise is 30%. What can also be seen is the precision of the model even with 90% of noise and only 10% of the original command file as defined in subsection 3.5. This really shows the power of this model, it is able to concentrate on the words and not on the noise, which is very much present in this case.

Quantity of noise	Precision	Recall	F1-Score
0.	0.9727	1.0	0.9862
0.1	0.9869	1.0	0.9934
0.2	0.9891	1.0	0.9945
0.3	0.9945	1.0	**0.9973**
0.4	0.9924	1.0	0.9962
0.5	0.9825	1.0	0.9912
0.6	0.9782	1.0	0.9890
0.7	0.9836	1.0	0.9917
0.8	0.9028	1.0	0.9489
0.9	0.8188	1.0	0.9003

Table 4.7: Wav2vec model trained with different level of noise on the biased normalised dataset without the word "Free" and tested on the real conditions set with no threshold.

4 Summary of the results

Training dataset	Model	Precision	Recall	F1-Score
artificial voices alone	Wav2vec 2.0	0.9178	1.0	0.9572
	ResNet-18	0.0862	1.0	0.1587
human voices alone	Wav2vec 2.0	0.3818	1.0	0.5526
	ResNet-18	0.1884	1.0	0.3170
mixed	Wav2vec 2.0	0.9739	1.0	0.9868
	ResNet-18	0.6052	1.0	0.7541
biased mixed	Wav2vec 2.0	0.9319	1.0	0.9647
	ResNet-18	0.5601	1.0	0.7180
noised biased mixed	Wav2vec 2.0	0.9790	1.0	0.9894
	ResNet-18	0.7255	1.0	0.8409
noised non-biased mixed	Wav2vec 2.0	0.9760	1.0	0.9878
	ResNet-18	0.7445	1.0	0.8535
augmented noised non-biased mixed	ResNet-18	0.7735	1.0	0.8723
noised biased norm. w/o free	Wav2vec 2.0	0.9891	1.0	0.9945
noised non-biased norm. w/o free augm.	ResNet-18	0.7849	1.0	0.8795
noised biased norm. w/o free augm.	ResNet-18	0.8002	1.0	0.8890
30% noised biased norm. w/o free	**Wav2vec 2.0**	**0.9945**	**1.0**	**0.9973**

Table 4.8: Summary of the different results obtained during the ablation study. The models are trained on different training sets and tested on the real conditions test set. ”Noised” is by default 20% of noise applied on each command except when a percentage is specified.

4.1 Metrics according to the threshold

Various metrics are computed on the model trained with different quantities of noise, according to an increasing threshold. The F1-Score for the different quantities can be found in Figure 4.21. It can be seen that, when the model is trained to 80% or 90% of noise. The model is not as bad as was thought with no threshold. However, when this threshold increases, the F1-Score slowly decreases until a threshold of 0.8 where a real drop can be observed. Now, the focus of the Figure 4.22 is on the model trained with noise going from 0% to 50%. What can be noticed is the fact that the F1-Score increases with noise up to 30% and then decreases. For the best model, which is the one trained with 30% of noise, the big drop in the F1-Score appears at a threshold of approximately 0.95.

In the use-case of this project, precision is important because the operator controlling the drone wants the drone to obey the command he just gave it. However, recall is also important as it is not good to reject too many commands just because the model is not certain enough of its prediction. In order to better visualise the number of unpredicted commands, Figures 4.23 and 4.24 show the fractions of the dataset that are not predicted according to the classification threshold. As can be seen for the model trained to more than 70%, a large part of the dataset is not predicted. For our best model (i.e. the model trained to 30% of noise), if a threshold of 0.99 is used, ”only” 7.5% of commands are discarded.

The precision and recall for the different quantities can be seen in the appendix in Figures 5.24, 5.25, 5.26 and 5.27.

As already mentioned, both precision and recall are important and this is why the focus is on the F1-Score. By only looking at this F1-Score, the best model that can be obtained is the model trained to 30% of noise and no threshold. This may be explained by the fact that without any threshold, the precision is already 0.9945 and the maximum result that can be attained with a threshold of 0.9 is 0.9967. Therefore, there isn't a large increase in the precision where the recall falls from 1.0 with no threshold to 0.9923 with a threshold of 0.9. So, there's a bigger loss in the recall, than there is a gain in the precision. The same behavior can be seen for each of the models. A last observation is that when this model predicts a command, it does so with a great certainty, because, as can be seen in the Figure 5.25, the precision still increases until a threshold of 0.97 or 0.98 from which it is possible to see a very small slope.

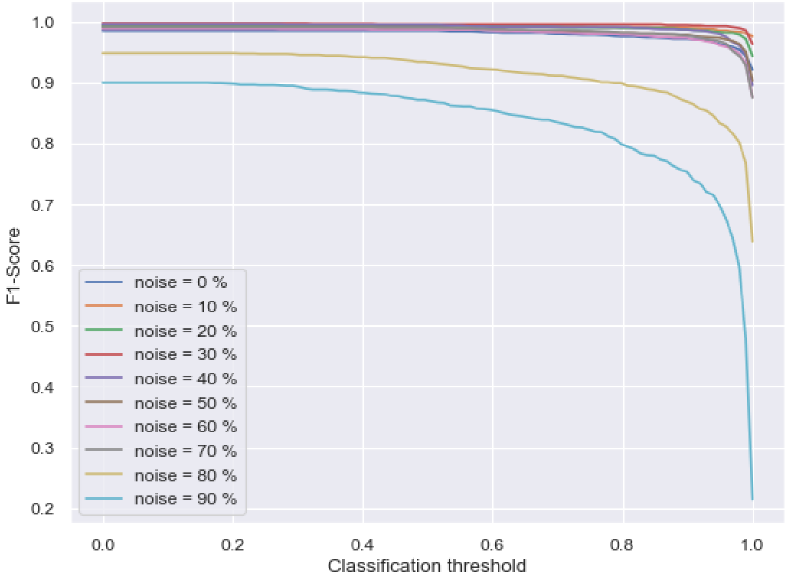


Figure 4.21: F1-Score of the best model trained with different quantities of noise going from 0 to 90% and tested on the real conditions dataset according to a classification threshold.

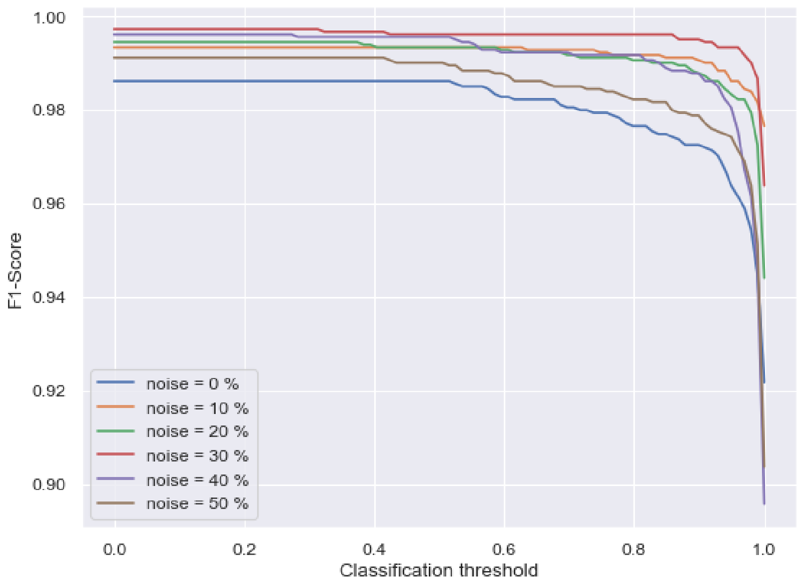


Figure 4.22: F1-Score of the best model trained with different quantities of noise going from 0 to 50% tested on the real conditions dataset according to a classification threshold.

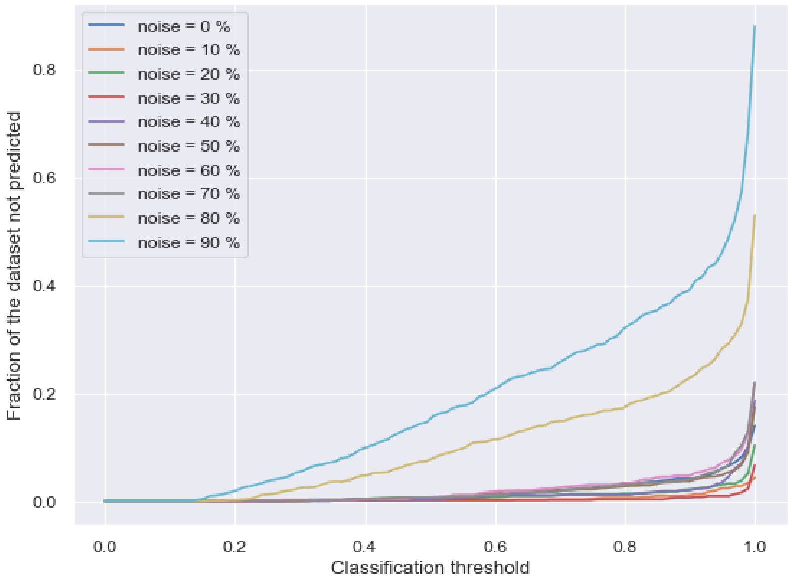


Figure 4.23: Percentage of non predicted commands of the model trained with different quantities of noise going from 0 to 90% according to a classification threshold

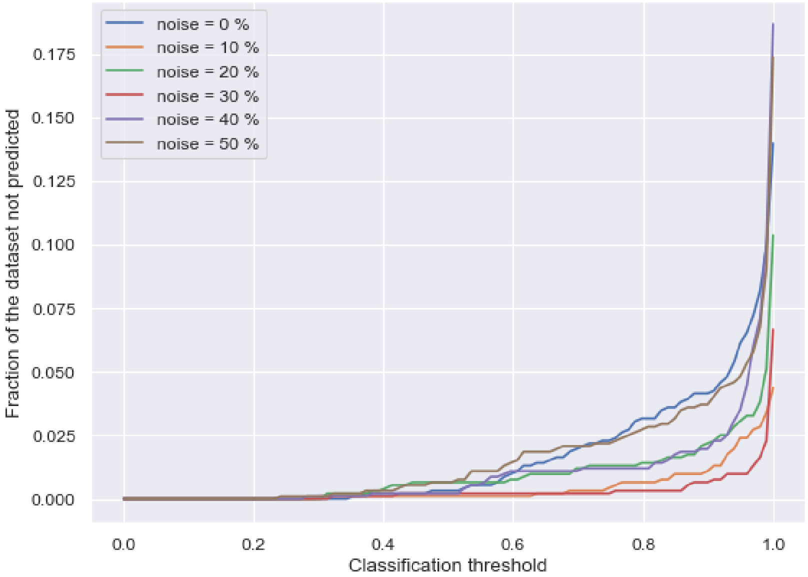


FIGURE 4.24: Percentage of non predicted commands of the model trained with different quantities of noise going from 0 to 50% according to a classification threshold

5 Comparison with Bolland's results

To make a good comparison, only the best model from Bolland's work [4] is taken. The best result he was able to obtain is through the training of a ResNet-15, which he called like this because it is a ResNet architecture consisting of 15 identical Residual blocks, as can be seen in FIGURE 4.25. Te current challenge was to try to reproduce its architecture and transpose it into this work, to re-train it with this work's dataset, and finally obtain results.

In his work, Bolland has one model for all words. As input, he only gives one word to the model, not an entire command and output probabilities for the 28 words of the vocabulary. In fact, this is quite similar to what is done in this work, because here, the same model is used for the three categories, but at the really end of the model, it's separated into 3 different heads (3 simple linear layers) that have their own learnable parameters.

Bolland's dataset only consists of the Google commands dataset combined with the collected words (note that he had fewer words collected than there is in our case). Bolland's dataset is separated into 2 folders ”train” and ”test” sets. What was observed is that Bolland didn't search for duplicate files, as explained in the Dataset section, and therefore he ended up with duplicate files between training and testing set which bias the evaluation.

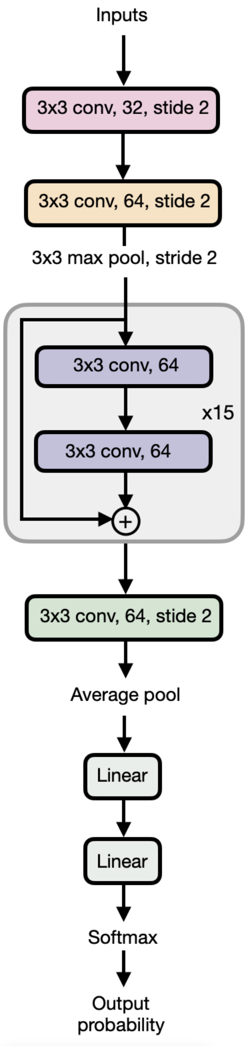


Figure 4.25: Representation of the "ResNet-15" architecture used in Bolland's work [4]. The spectrogram is input and passed through 2 convolutions and then the output of these 2 convolutions is passed through a group of 15 identical residual blocks containing 2 internal convolutions. At the end, the final feature map is reduced into a one-dimension representation and passed through a linear layer and passed through a softmax in order to obtain the probabilities for the 28 different words of the vocabulary.

To have approximately the same testing procedure as in Bolland's work, a test set was created consisting of hand-crafted commands as explained in section 3.2. By doing that, 2 or 3 different words from 2 or 3 different persons are present in a single command. He also only considered accuracy when evaluating his models. This accuracy is also used here. Moreover, he doesn't use the accuracy on the commands, but the accuracy on the words; so the exact same thing is done here and an average of the 3 categories is made.

The comparison can be found in Table 4.9. A first observation is that with the ResNet trained with noise in this work, a better accuracy can be obtained on words than the one from Bolland. This may be explained in several ways. First, in this work a lot more words were acquired compared to what Bolland had. This allows the model to better generalise. The second point is that Bolland used the noise from the Google commands dataset which consists of 5 noise files, namely the sound of a meowing cat, someone washing dishes, someone on an exercise bike, a running tap and the siren of a Belgian firefighter truck. With only 5 different noise files, the model learns these sounds by heart and doesn't try to generalise what is behind them. Furthermore, using these same sounds during the testing phase clearly biases the testing phase as they are sounds that the model has already heard during the training.

What you can also see in this Table is that word accuracy is not really an efficient measure. Very good results are obtained in terms of word accuracy with the ResNet-18, but there is a drop-off between this accuracy and the precision of the commands. A second observation regarding our results with this test set is that, for the ResNet, the results are much better than with the real conditions test set. However, the results for the Wav2vec model are not as good as with the test set in real conditions. These results are expected as there are no real condition commands in the training set and, as the ResNet has to be trained from scratch, ResNet is expected to perform better on a similar set of tests to the training. For the Wav2vec model, it is normal that it performs better on the real conditions test set since its self-supervised learning phase is done on real conditions data.

A final observation was common in both works: the fact that using the MFCC instead of the spectrogram gives worse results. This was tested, but not shown in our work, because it was thought it was not relevant.

Model	**Training set**	**Testing set**	**Accuracy of words**	**Precision of commands**	**F1-Score of commands**
ResNet-15 from Bolland	words non-noised	words non-noised	97%	/	/
		40% noised words	73%		
	40% noised words	words non-noised	91%		
		40% noised words	85%		
ResNet-18 from this work	augm. 40% noised commands norm.	non-noised test set	97.16 %	0.9167	0.9565
		40% noised test set	97.26%	0.9197	0.9598
Wav2vec 2.0 this work	30% noised commands norm.	non-noised test set	99.36%	0.9813	0.9906
		40% noised test set	99.35%	0.9813	0.9905

Table 4.9: Comparison of word accuracies between Bolland's ResNet-15 model and our models tested on a test set consisting of handcrafted commands (NOT the real conditions test set as before) to obtain a testing procedure similar to Bolland's work.

6 Fine-tune on a part of the real conditions test set

Some fine-tuning was attempted on the best model on the real condition dataset. This is a very small dataset, consisting of 916 commands, where those containing the word ”Free” are discarded.

When dealing with a small dataset, the k-fold cross validation technique can be used. This method consists of splitting the dataset into k folds of equal size. Then a fold is selected as the testing set and the remaining are combined to form the training set. This can be seen in 4.26. This process is repeated k times, so that each of the k folds is used once as the test set. The performance is measured at each of the k steps and at the end, the mean is computed in order to obtain the final performance of the model.

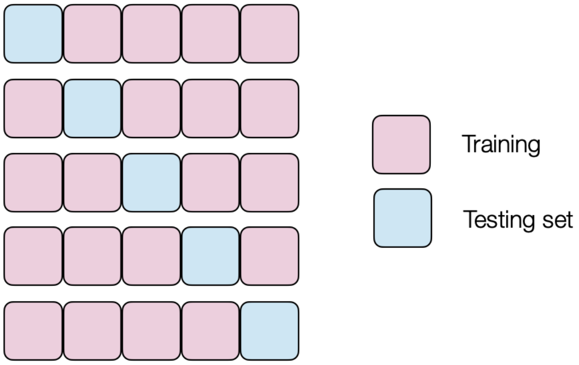


FIGURE 4.26: K-fold cross validation. The dataset is split into k folds and a fold is chosen to be the test set and the remaining are combined to form the train set. This is repeated k times for the k folds to be used once as the test set.

In this case, the dataset was split in 5 folds, which means that the first 4 folds contain 183 commands and the last one contains 184. The k-fold cross validation technique was first applied to fine-tune the best ResNet from this work which is the ResNet-18 trained on the augmented noised biased normalised dataset without the word ”Free” and the results can be found in TABLE 4.10. These results are a little bit better than the one which isn't fine-tuned on the real conditions data, but it isn't a big improvement. Then, it was also applied on a ResNet pre-trained with ImageNet [29] to see the difference and as can be seen in TABLE 4.11, there is a huge gap between the two.

The same was done, but this time on the best Wav2vec 2.0 model which is trained on 30% noised biased normalised dataset again without the word ”Free”. The results in TABLE 4.12 appear to be worse than with the original best model. This may simply be explained by the fact that there isn't enough data. With the Wav2vec 2.0 model simply coming out of its self-supervised learning phase, it can be seen in TABLE 4.13 that the results are really bad, even worse than for the pretrained ResNet.

	Model	Precision	F1-Score
Fold 1	Best ResNet18 in this work	0.7204	0.8406
Fold 2		0.7705	0.8703
Fold 3		0.8033	0.8909
Fold 4		0.9126	0.9543
Fold 5		0.9837	0.9917
Mean		**0.8381**	**0.9095**

Table 4.10: Fine-tuning of the best ResNet using a 5-folds cross validation on the real conditions dataset.

	Model	Precision	F1-Score
Fold 1	Simple ResNet18 pre-trained on ImageNet	0.5301	0.6929
Fold 2		0.1311	0.2319
Fold 3		0.2076	0.3439
Fold 4		0.5847	0.7379
Fold 5		0.6793	0.8090
Mean		**0.4266**	**0.56312**

Table 4.11: Fine-tuning of a simple ResNet pre-trained on ImageNet using a 5-folds cross validation on the real conditions dataset.

	Model	Precision	F1-Score
Fold 1	Best Wav2vec 2.0 in this work	0.9836	0.9917
Fold 2		0.9836	0.9917
Fold 3		0.9945	0.9973
Fold 4		0.9890	0.9945
Fold 5		0.9891	0.9945
Mean		**0.9880**	**0.9040**

Table 4.12: Fine-tuning of the best Wav2vec model using a 5-folds cross validation on the real conditions dataset.

	Model	Precision	F1-Score
Fold 1	Simple Wav2vec only self-supervised	0.3990	0.5703
Fold 2		0.2676	0.4224
Fold 3		0.1257	0.2233
Fold 4		0.2459	0.3947
Fold 5		0.1522	0.2642
Mean		**0.2381**	**0.3750**

Table 4.13: Fine-tuning of a Wav2vec 2.0 after the self-supervised stage using a 5-folds cross validation on the real conditions dataset.

7 Inference

In the final use case, it is important to know how much time it takes the model to understand and predict a command. This is therefore the last experiment, which is to compute the inference time on CPU for both models. The Table 4.14 below shows the time observed. What can be noted is that the prediction time of a command for Bolland is almost 3 times higher than that of our ResNet. This is quite logical since he has to make 3 predictions on the 3 different words with his model to obtain the prediction of a complete command.

Model	Audio processing time (ms)	Prediction time (ms)	Total time (ms)
ResNet-18	4	38	42
Wav2vec 2.0	3	280	283
ResNet-15 from Bolland	/	/	130

Table 4.14: Inference time on CPU for the best ResNet and the best Wav2vec 2.0 of this work but also for the best model from Bolland's work [4]

Chapter 5

Conclusions

1 Further work

1.1 Improvement of the model

The major drawback of this whole project was the data. For the human voices, only words were collected and these words were assembled into commands. Nonetheless, this doesn't reflect reality. To obtain better results on real conditions data, one should think about acquiring a large amount of data by asking people with different accents, microphones to record complete commands in noisy places and not in the tranquility of a room at home. What could also improve the model, is to add False commands. Here, no threshold was set, because there were only real commands. Therefore, the model could only go wrong in one of the 3 categories. If false commands are added, a threshold must also be defined, below which the command is not taken into account.

Moreover, advanced augmentation techniques should be used during the training of the model, such as changing the frequency by a certain amount to change the person's voice by one or more semitones. This would help the model to generalise more.

Finally, new models are launched quite regularly and fix certain problems of previous models, so training a newer model such as "HuBERT" could further improve the results.

1.2 Integration into the final project

The next step in this project is to integrate this model into a slightly more complex one. For example, *Keyword Spotting* could be carried out and more specifically *wake word detection* on each of the names. A model is trained to recognise several names and as stated before, each drone has its own name. The different drones listen continuously and when one of them detects its name, it listens for a certain number of seconds and tries to predict whether what follows its name is a valid command or not.

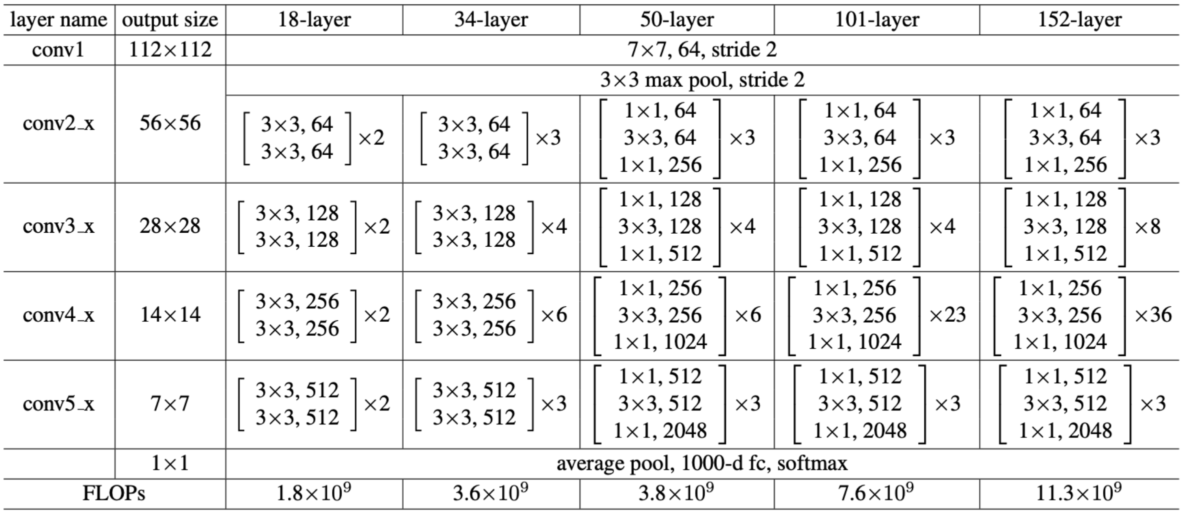
2 Discussion

In this Master's Thesis, state-of-the art *automatic speech recognition* model **Wav2vec 2.0** was used and fine-tuned on this downstream task which is *speech command recognition*. More specifically, this work shows how data was acquired and, through an ablation study, the different methods that can be used in order to improve the results, when only small amount of data is available. The results on real conditions data are quite good and the model is very confident about its predictions. This is probably thanks to the fact that the model has an initial self-supervised phase. The power of this technique can be seen when the model is only trained on artificial voices and already obtains outstanding results on data in real conditions.

The ResNet results aren't so good on this test set, but it has been seen that using the same kind of data as in the train in a test set, the ResNet results were much better and almost equivalent to those of Wav2vec 2.0. It was also established that this ResNet is 10 times faster to make a prediction. If the ResNet was trained on real conditions data, it could be as good as Wav2vec 2.0.

Appendix

Chapter 2



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

FIGURE 5.1: Summary of the different ResNet architectures with the blocks inside according to their size

Model	dev		test	
	clean	other	clean	other
10 min labeled	4.6	7.9	4.8	8.2
1h labeled	2.9	5.4	2.9	5.8
10h labeled	2.4	4.8	2.6	4.9
100h labeled	1.9	4.0	2.0	4.0
960h labeled	1.6	3.0	1.8	3.3

TABLE 5.1: Word-Error-Rate(WER) for the LARGE wav2vec 2.0 on the Librispeech dev/test sets when pre-trained on LibriVox (60k unlabeled audio) and fine-tuned on the Libri-light low-resource labeled data setups

Chapter 3

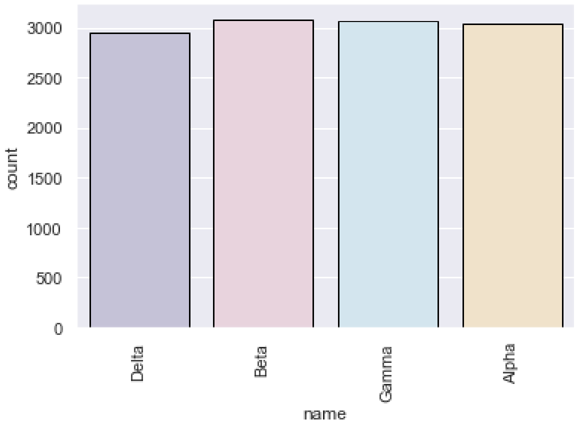


FIGURE 5.2: Count of the names in all the validation hand-crafted commands set.

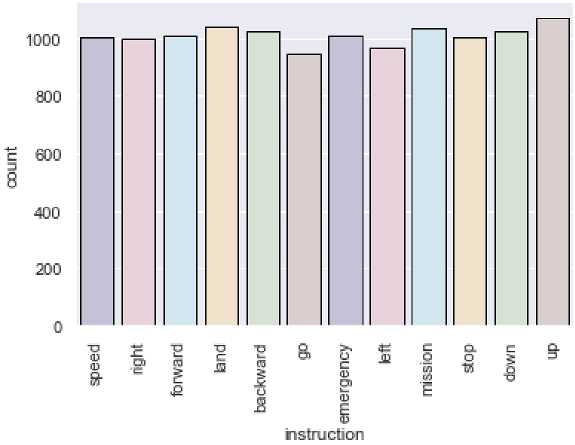


FIGURE 5.3: Count of the instructions in all the validation hand-crafted commands set.

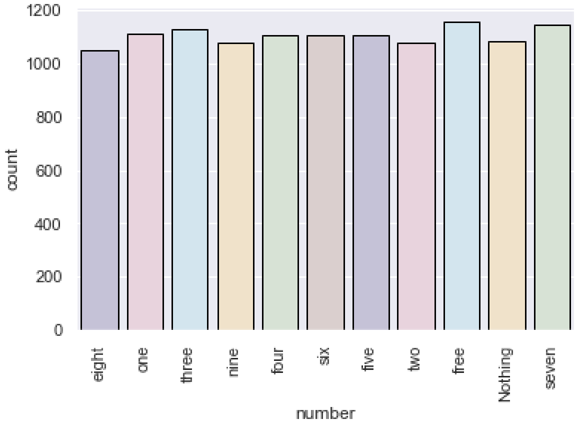


FIGURE 5.4: Count of the numbers in all the validation hand-crafted commands set.

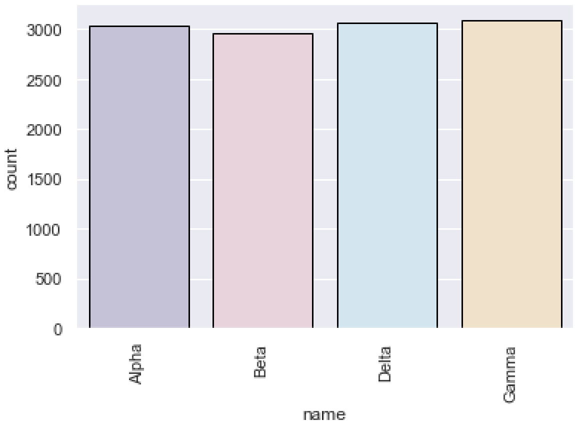


FIGURE 5.5: Count of the names in all the test hand-crafted commands set.

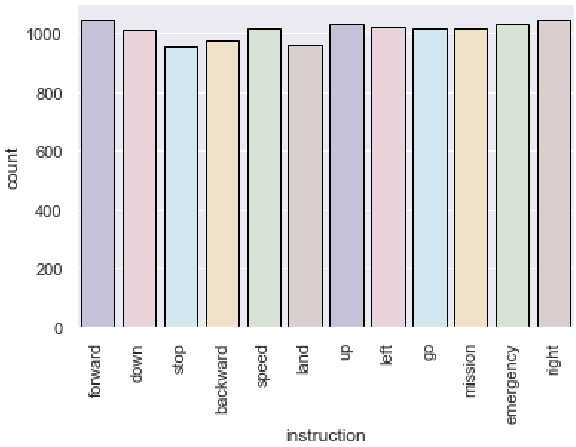


FIGURE 5.6: Count of the instructions in all the test hand-crafted commands set.

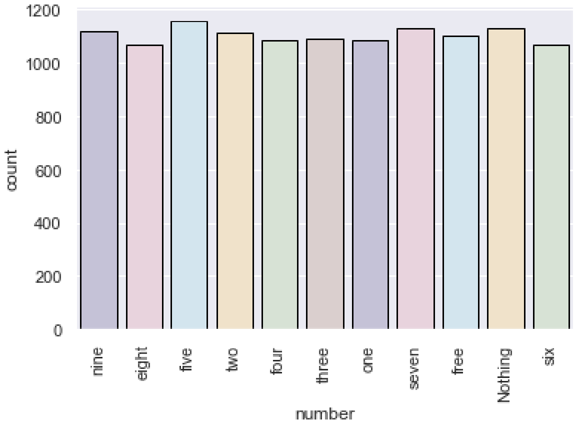


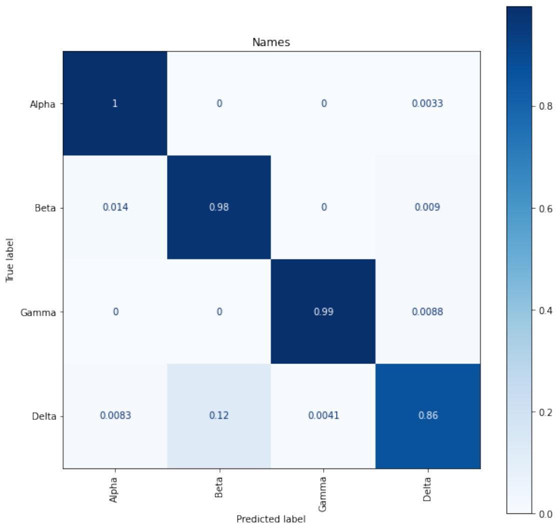
FIGURE 5.7: Count of the numbers in all the test hand-crafted commands set.

Chapter 4

The hardware specifications of the computer that performed all the experiments are the following :

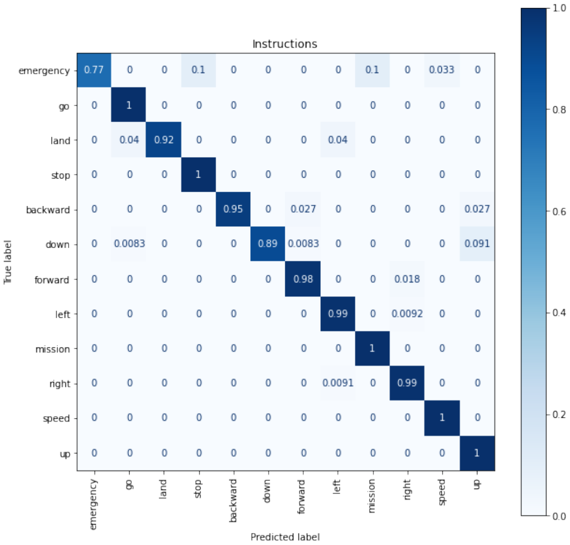
- CPU : Intel Core i5-11400F
- GPU : NVIDIA GeForce RTX 3060
- RAM : 16 GB
- VRAM : 12 GB

Concerning the ResNet18, the largest batches that can be taken with these specifications are batches of size 64. Therefore all the following experiences are performed with that batch size.



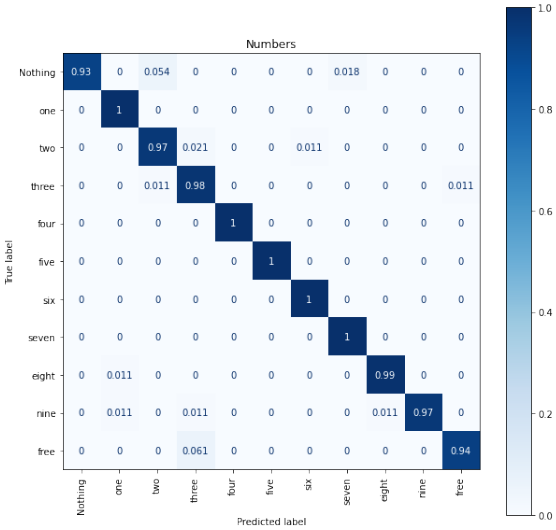
	Alpha	Beta	Gamma	Delta
Alpha	1	0	0	0.0033
Beta	0.014	0.98	0	0.009
Gamma	0	0	0.99	0.0088
Delta	0.0083	0.12	0.0041	0.86

Figure 5.8: Confusion matrix for the ”Name” category of the wav2vec model trained only on artificial voices and with commands that are only part of the grammar



	emergency	go	land	stop	backward	down	forward	left	mission	right	speed	up
emergency	0.77	0	0	0.1	0	0	0	0	0.1	0	0.033	0
go	0	1	0	0	0	0	0	0	0	0	0	0
land	0	0.04	0.92	0	0	0	0	0.04	0	0	0	0
stop	0	0	0	1	0	0	0	0	0	0	0	0
backward	0	0	0	0	0.95	0	0.027	0	0	0	0	0.027
down	0	0.0083	0	0	0	0.89	0.0083	0	0	0	0	0.091
forward	0	0	0	0	0	0	0.98	0	0	0.018	0	0
left	0	0	0	0	0	0	0	0.99	0	0.0092	0	0
mission	0	0	0	0	0	0	0	0	1	0	0	0
right	0	0	0	0	0	0	0	0.0091	0	0.99	0	0
speed	0	0	0	0	0	0	0	0	0	0	1	0
up	0	0	0	0	0	0	0	0	0	0	0	1

Figure 5.9: Confusion matrix for the ”Instruction” category of the wav2vec model trained only on artificial voices and with commands that are only part of the grammar



	Nothing	one	two	three	four	five	six	seven	eight	nine	free
Nothing	0.93	0	0.054	0	0	0	0	0.018	0	0	0
one	0	1	0	0	0	0	0	0	0	0	0
two	0	0	0.97	0.021	0	0	0.011	0	0	0	0
three	0	0	0.011	0.98	0	0	0	0	0	0	0.011
four	0	0	0	0	1	0	0	0	0	0	0
five	0	0	0	0	0	1	0	0	0	0	0
six	0	0	0	0	0	0	1	0	0	0	0
seven	0	0	0	0	0	0	0	1	0	0	0
eight	0	0.011	0	0	0	0	0	0	0.99	0	0
nine	0	0.011	0	0.011	0	0	0	0	0.011	0.97	0
free	0	0	0	0.061	0	0	0	0	0	0	0.94

Figure 5.10: Confusion matrix for the ”Number” category of the wav2vec model trained only on artificial voices and with commands that are only part of the grammar

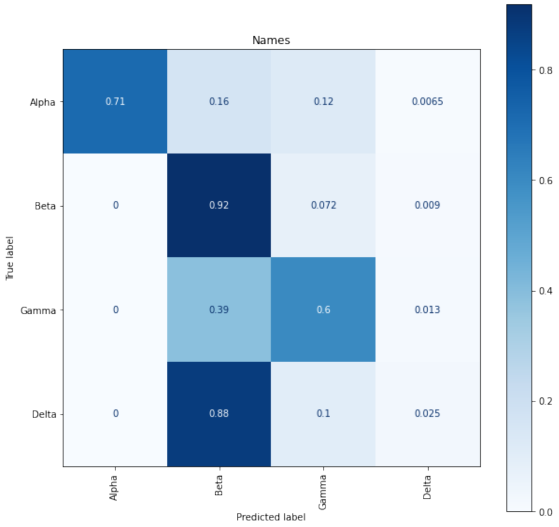


FIGURE 5.11: Confusion matrix for the "Name" category of the ResNet model trained only on artificial voices and with commands that are only part of the grammar

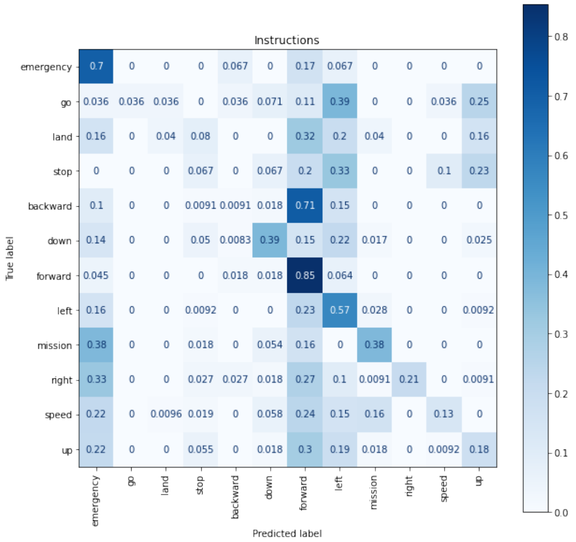


FIGURE 5.12: Confusion matrix for the "Instruction" category of the ResNet model trained only on artificial voices and with commands that are only part of the grammar

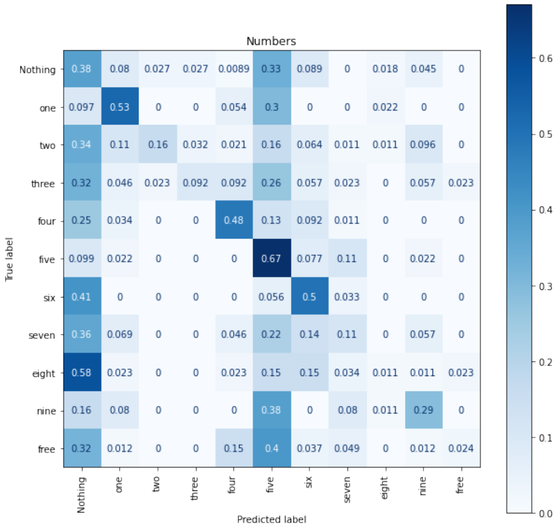
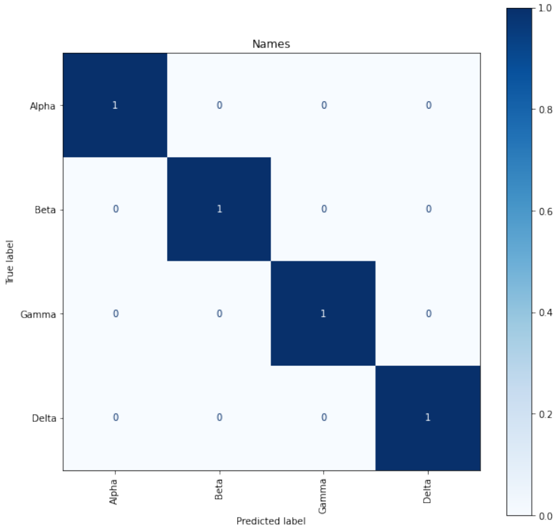
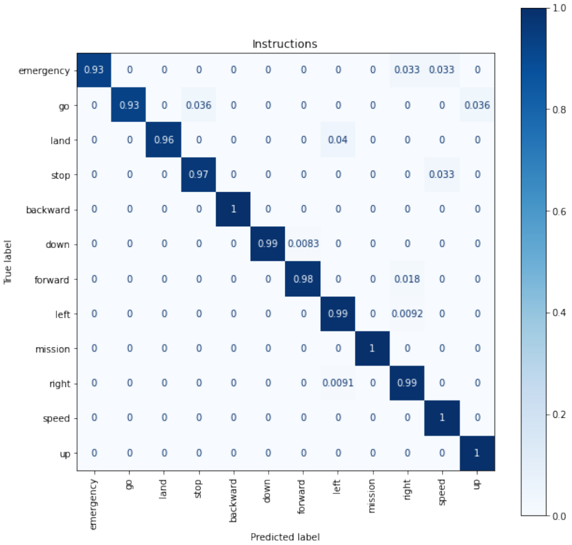


FIGURE 5.13: Confusion matrix for the "Number" category of the ResNet model trained only on artificial voices and with commands that are only part of the grammar



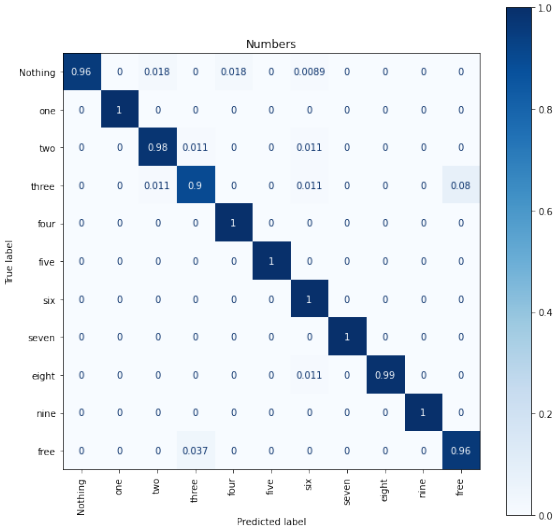
True label \ Predicted label	Alpha	Beta	Gamma	Delta
Alpha	1	0	0	0
Beta	0	1	0	0
Gamma	0	0	1	0
Delta	0	0	0	1

FIGURE 5.14: Confusion matrix for the ”Name” category of the wav2vec model trained on dataset of human and artificial voices with commands that are only part of the grammar



True label \ Predicted label	emergency	go	land	stop	backward	down	forward	left	mission	right	speed	up
emergency	0.93	0	0	0	0	0	0	0	0	0.033	0.033	0
go	0	0.93	0	0.036	0	0	0	0	0	0	0	0.036
land	0	0	0.96	0	0	0	0	0.04	0	0	0	0
stop	0	0	0	0.97	0	0	0	0	0	0	0.033	0
backward	0	0	0	0	1	0	0	0	0	0	0	0
down	0	0	0	0	0	0.99	0.0083	0	0	0	0	0
forward	0	0	0	0	0	0	0.98	0	0	0.018	0	0
left	0	0	0	0	0	0	0	0.99	0	0.0092	0	0
mission	0	0	0	0	0	0	0	0	1	0	0	0
right	0	0	0	0	0	0	0	0.0091	0	0.99	0	0
speed	0	0	0	0	0	0	0	0	0	0	1	0
up	0	0	0	0	0	0	0	0	0	0	0	1

FIGURE 5.15: Confusion matrix for the ”Instruction” category of the wav2vec model trained on the dataset of human and artificial voices with commands that are only part of the grammar



True label \ Predicted label	Nothing	one	two	three	four	five	six	seven	eight	nine	free
Nothing	0.96	0	0.018	0	0.018	0	0.0089	0	0	0	0
one	0	1	0	0	0	0	0	0	0	0	0
two	0	0	0.98	0.011	0	0	0.011	0	0	0	0
three	0	0	0.011	0.9	0	0	0.011	0	0	0	0.08
four	0	0	0	0	1	0	0	0	0	0	0
five	0	0	0	0	0	1	0	0	0	0	0
six	0	0	0	0	0	0	1	0	0	0	0
seven	0	0	0	0	0	0	0	1	0	0	0
eight	0	0	0	0	0	0	0.011	0	0.99	0	0
nine	0	0	0	0	0	0	0	0	0	1	0
free	0	0	0	0.037	0	0	0	0	0	0	0.96

FIGURE 5.16: Confusion matrix for the ”Number” category of the wav2vec model trained on the dataset of human and artificial voices and with commands that are only part of the grammar

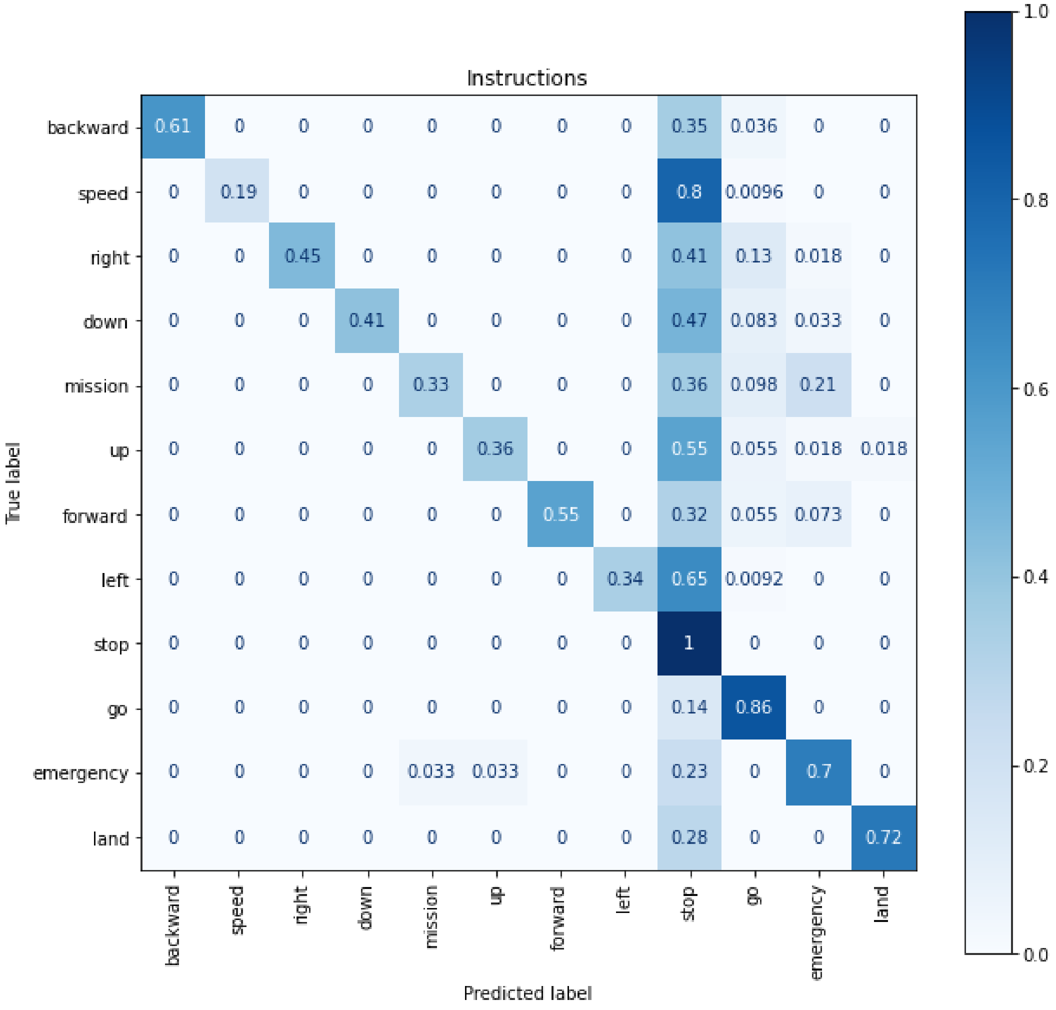


FIGURE 5.17: Confusion matrix for the ”Instruction” category of the wav2vec model trained only on human voices and with commands that are only part of the grammar.

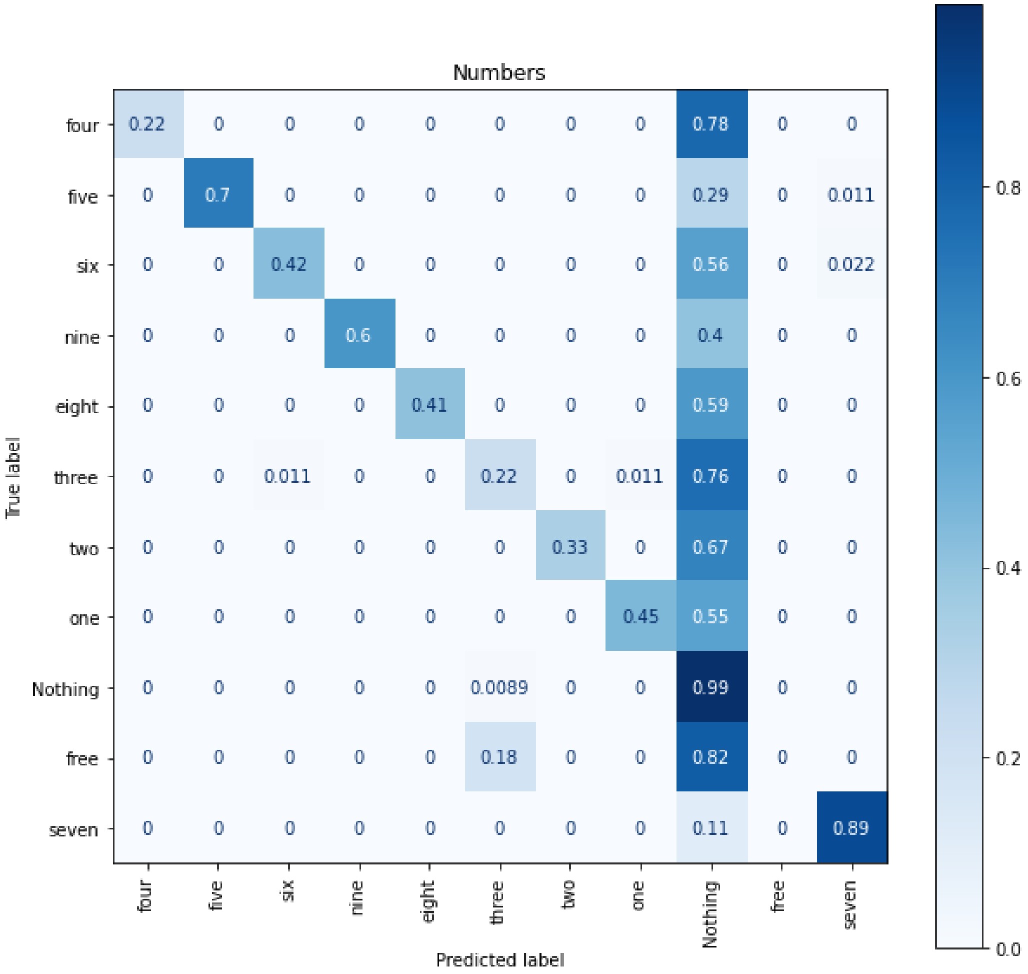
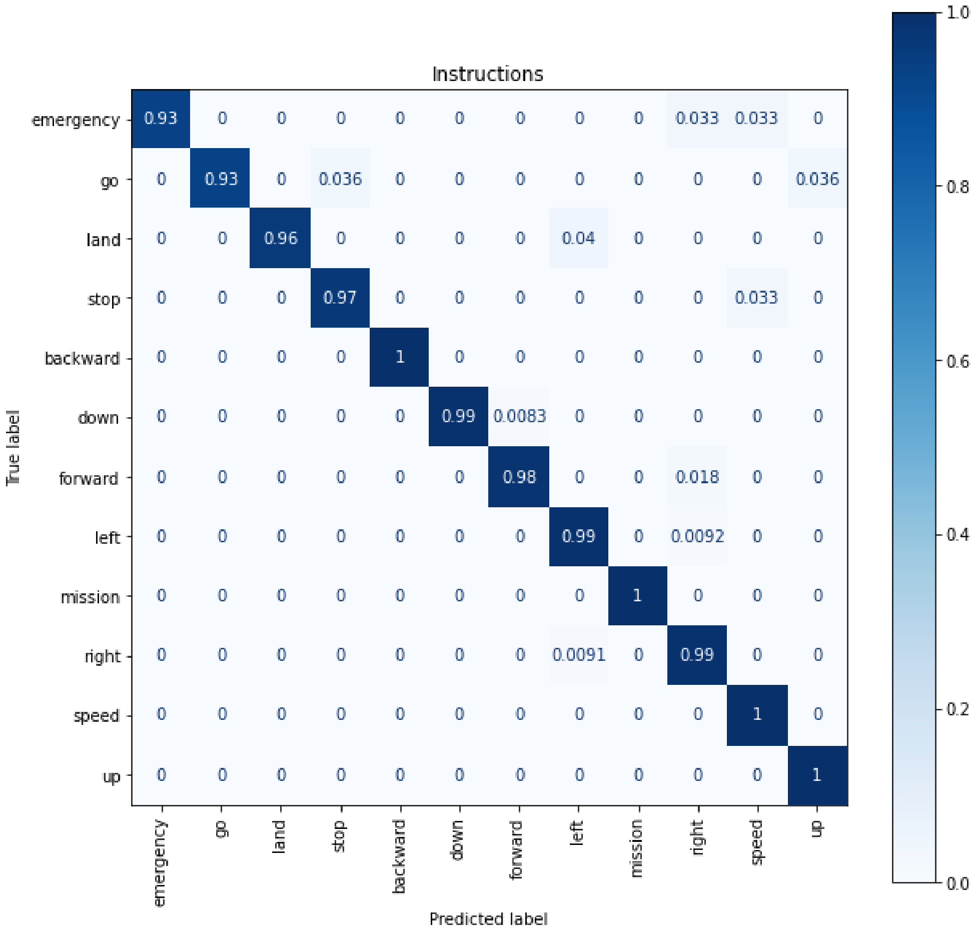


FIGURE 5.18: Confusion matrix for the "Number" category of the wav2vec model trained only on human voices and with commands that are only part of the grammar.



	emergency	go	land	stop	backward	down	forward	left	mission	right	speed	up
emergency	0.93	0	0	0	0	0	0	0	0	0.033	0.033	0
go	0	0.93	0	0.036	0	0	0	0	0	0	0	0.036
land	0	0	0.96	0	0	0	0	0.04	0	0	0	0
stop	0	0	0	0.97	0	0	0	0	0	0	0.033	0
backward	0	0	0	0	1	0	0	0	0	0	0	0
down	0	0	0	0	0	0.99	0.0083	0	0	0	0	0
forward	0	0	0	0	0	0	0.98	0	0	0.018	0	0
left	0	0	0	0	0	0	0	0.99	0	0.0092	0	0
mission	0	0	0	0	0	0	0	0	1	0	0	0
right	0	0	0	0	0	0	0	0.0091	0	0.99	0	0
speed	0	0	0	0	0	0	0	0	0	0	1	0
up	0	0	0	0	0	0	0	0	0	0	0	1

FIGURE 5.19: Confusion matrix for the ”Instruction” category of the wav2vec model trained on the mixed commands dataset with commands that are only part of the grammar.

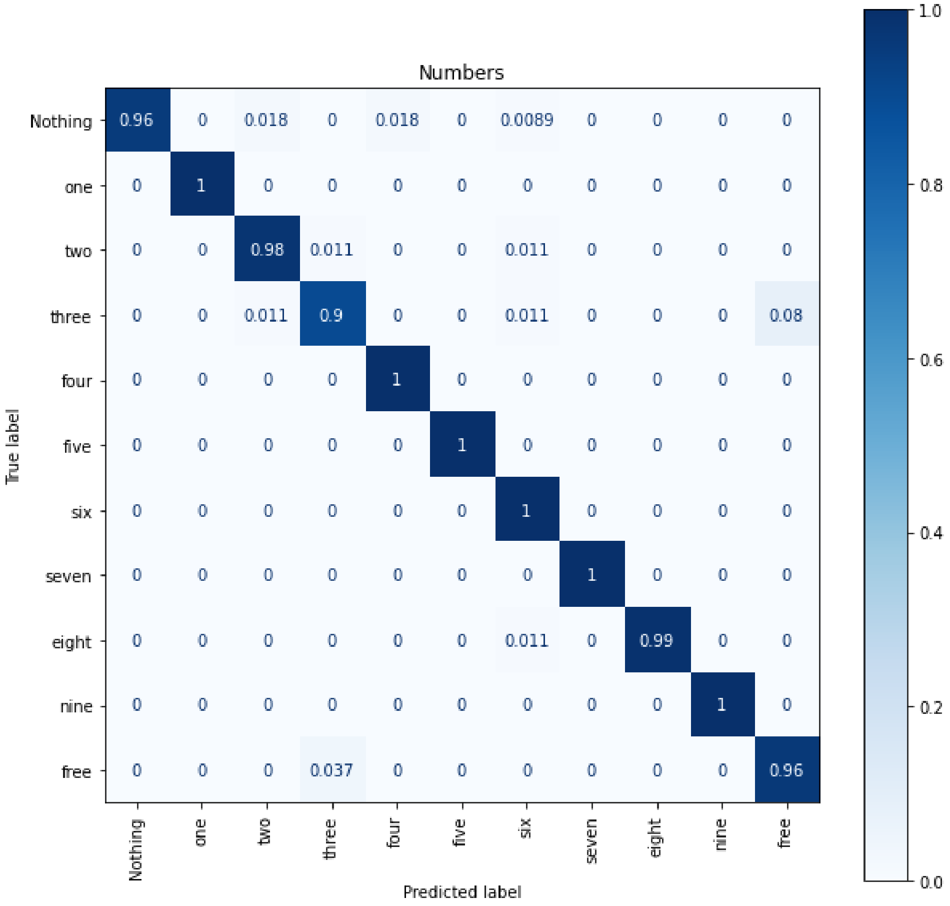


Figure 5.20: Confusion matrix for the "Number" category of the wav2vec model trained only on the mixed dataset with commands that are only part of the grammar.

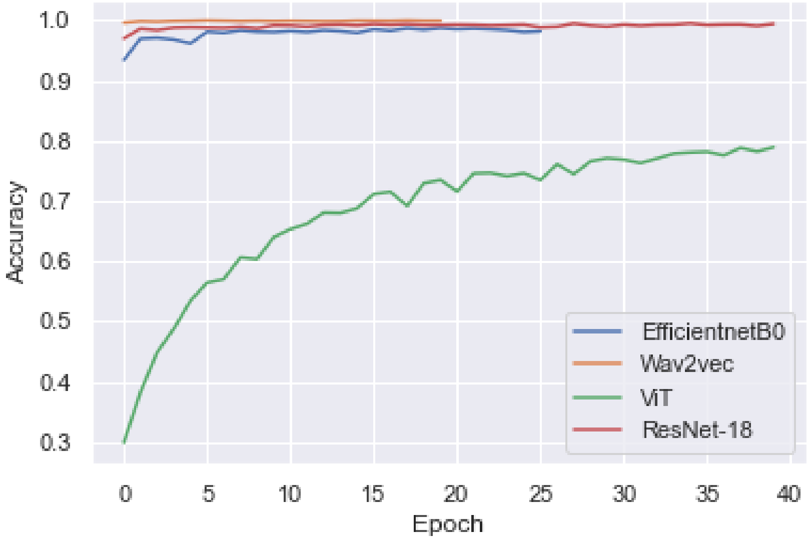


FIGURE 5.21: Validation accuracy of the different best models for the ”Name” category

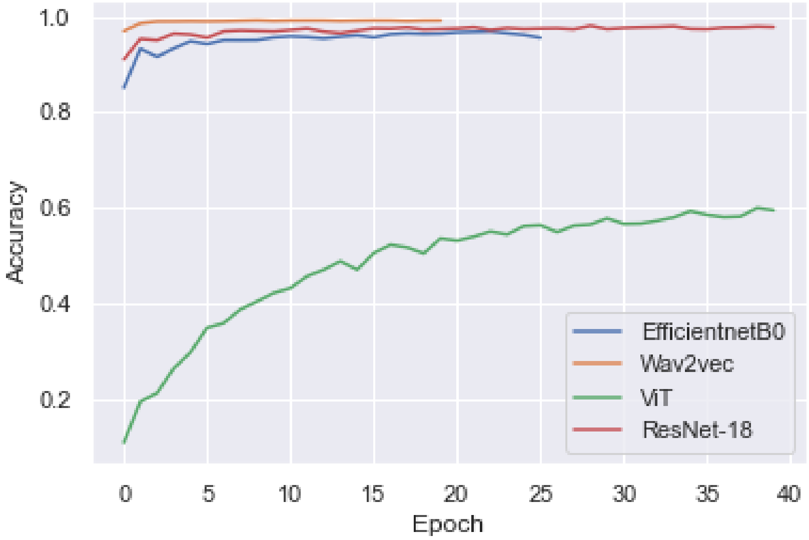


FIGURE 5.22: Validation accuracy of the different best models for the ”Instruction” category

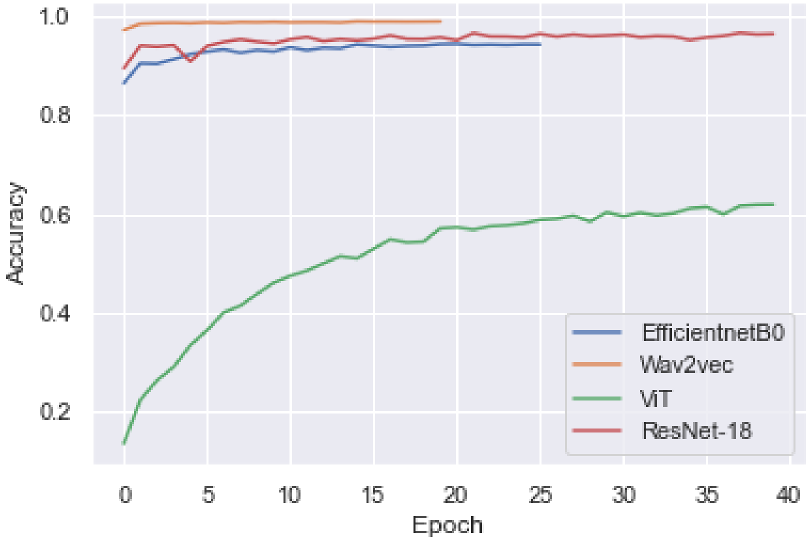


Figure 5.23: Validation accuracy of the different best models for the ”Number” category

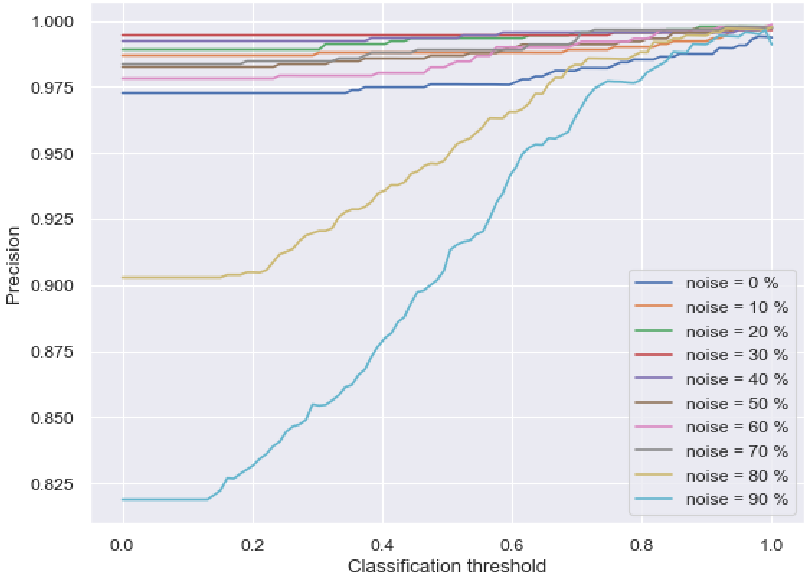


Figure 5.24: Precision of the best model trained with different quantities of noise going from 0 to 90% and tested on the real conditions dataset according to a classification threshold.

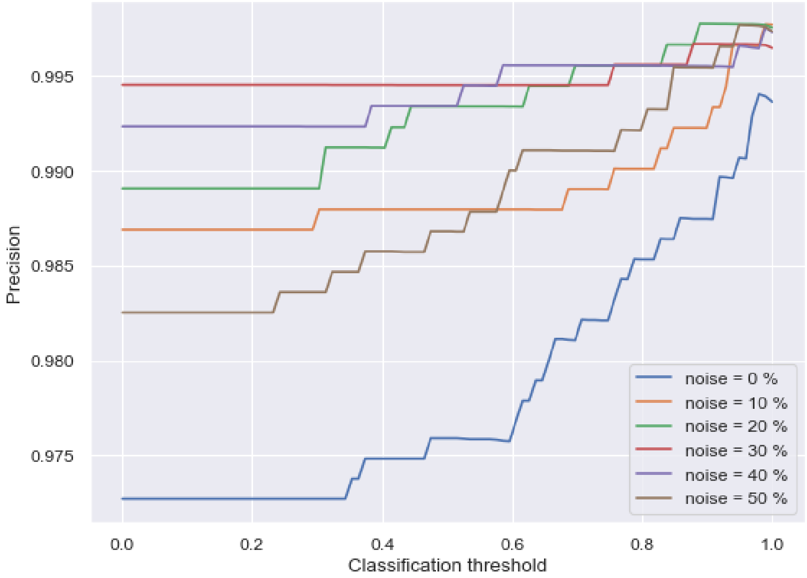


FIGURE 5.25: Precision of the best model trained with different quantities of noise going from 0 to 50% and tested on the real conditions dataset according to a classification threshold.

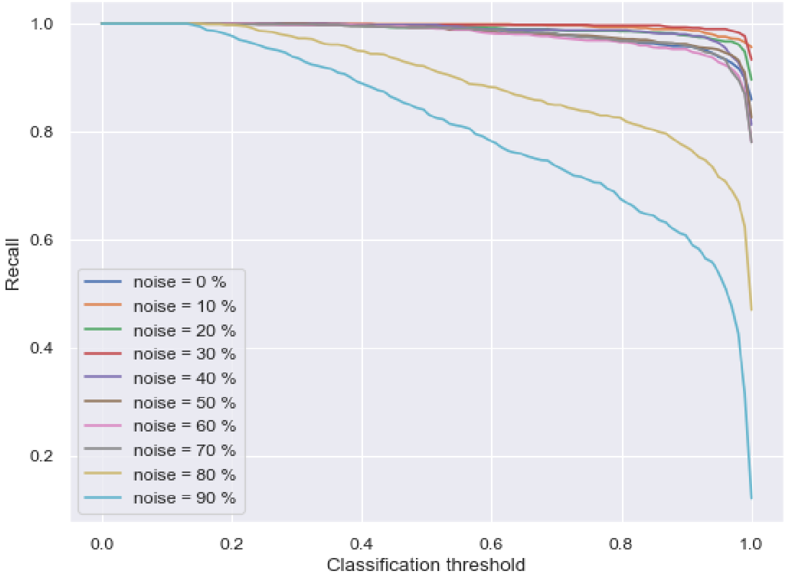


FIGURE 5.26: Recall of the best model trained with different quantities of noise going from 0 to 90% and tested on the real conditions dataset according to a classification threshold.

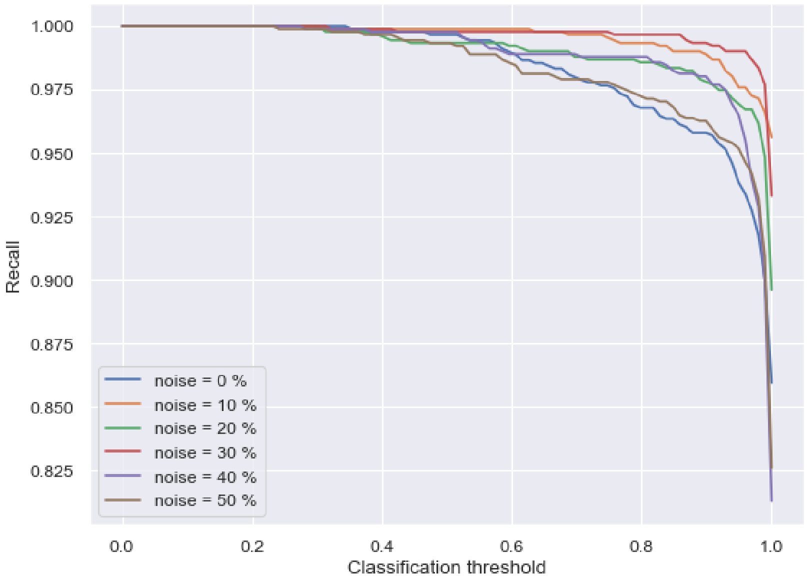


FIGURE 5.27: Recall of the best model trained with different quantities of noise going from 0 to 50% and tested on the real conditions dataset according to a classification threshold.

Bibliography

[1] Douglas Coimbra de Andrade et al. *A neural attention model for speech command recognition*. 2018. DOI: 10.48550/ARXIV.1808.08929. URL: https://arxiv.org/abs/1808.08929.

[2] Alexei Baevski et al. *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*. 2020. DOI: 10.48550/ARXIV.2006.11477. URL: https://arxiv.org/abs/2006.11477.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. DOI: 10.48550/ARXIV.1409.0473. URL: https://arxiv.org/abs/1409.0473.

[4] Julien Bolland. *Drone control through a vocal interface*. 2021. URL: http://hdl.handle.net/2268.2/11563.

[5] Ting Chen et al. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. DOI: 10.48550/ARXIV.2002.05709. URL: https://arxiv.org/abs/2002.05709.

[6] Ting Chen et al. *Big Self-Supervised Models are Strong Semi-Supervised Learners*. 2020. DOI: 10.48550/ARXIV.2006.10029. URL: https://arxiv.org/abs/2006.10029.

[7] Xinlei Chen et al. *Improved Baselines with Momentum Contrastive Learning*. 2020. DOI: 10.48550/ARXIV.2003.04297. URL: https://arxiv.org/abs/2003.04297.

[8] Yu-An Chung et al. *W2v-BERT: Combining Contrastive Learning and Masked Language Modeling for Self-Supervised Speech Pre-Training*. 2021. DOI: 10.48550/ARXIV.2108.06209. URL: https://arxiv.org/abs/2108.06209.

[9] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

[10] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. *Unsupervised Visual Representation Learning by Context Prediction*. 2015. DOI: 10.48550/ARXIV.1505.05192. URL: https://arxiv.org/abs/1505.05192.

[11] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[12] Alexey Dosovitskiy et al. *Discriminative Unsupervised Feature Learning with Exemplar Convolutional Neural Networks*. 2014. DOI: 10.48550/ARXIV.1406.6909. URL: https://arxiv.org/abs/1406.6909.

[13] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[14] Kaiming He et al. *Momentum Contrast for Unsupervised Visual Representation Learning*. 2019. DOI: 10.48550/ARXIV.1911.05722. URL: https://arxiv.org/abs/1911.05722.

[15] Wei-Ning Hsu et al. *HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units*. 2021. DOI: 10.48550/ARXIV.2106.07447. URL: https://arxiv.org/abs/2106.07447.

[16] Jie Hu et al. *Squeeze-and-Excitation Networks*. 2017. DOI: 10.48550/ARXIV.1709.01507. URL: https://arxiv.org/abs/1709.01507.

[17] J. Kahn et al. "Libri-Light: A Benchmark for ASR with Limited or No Supervision". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, May 2020. DOI: 10.1109/icassp40776.2020.9052942. URL: https://doi.org/10.1109%2Ficassp40776.2020.9052942.

[18] Byeonggeun Kim et al. *Broadcasted Residual Learning for Efficient Keyword Spotting*. 2021. DOI: 10.48550/ARXIV.2106.04140. URL: https://arxiv.org/abs/2106.04140.

[19] Alex Krizhevsky. "One weird trick for parallelizing convolutional neural networks". In: *CoRR* abs/1404.5997 (2014). arXiv: 1404.5997. URL: http://arxiv.org/abs/1404.5997.

[20] Min Lin, Qiang Chen, and Shuicheng Yan. *Network In Network*. 2013. DOI: 10.48550/ARXIV.1312.4400. URL: https://arxiv.org/abs/1312.4400.

[21] Quentin Louveaux. *Introduction to the theory of computation*. 2021. URL: https://www.programmes.uliege.be/cocoon/20212022/cours/INFO0016-1.html.

[22] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015. DOI: 10.48550/ARXIV.1508.04025. URL: https://arxiv.org/abs/1508.04025.

[23] Ishan Misra and Laurens van der Maaten. *Self-Supervised Learning of Pretext-Invariant Representations*. 2019. DOI: 10.48550/ARXIV.1912.01991. URL: https://arxiv.org/abs/1912.01991.

[24] Vassil Panayotov et al. "Librispeech: An ASR corpus based on public domain audio books". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 5206–5210. DOI: 10.1109/ICASSP.2015.7178964.

[25] Daniel S. Park et al. "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition". In: *Interspeech 2019*. ISCA, Sept. 2019. DOI: 10.21437/interspeech.2019-2680. URL: https://doi.org/10.21437%2Finterspeech.2019-2680.

[26] Deepak Pathak et al. "Context Encoders: Feature Learning by Inpainting". In: (2016). DOI: 10.48550/ARXIV.1604.07379. URL: https://arxiv.org/abs/1604.07379.

[27] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. DOI: 10.48550/ARXIV.1710.05941. URL: https://arxiv.org/abs/1710.05941.

[28] Chandan KA Reddy et al. "A Scalable Noisy Speech Dataset and Online Subjective Test Framework". In: *Proc. Interspeech 2019* (2019), pp. 1816–1820.

[29] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[30] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: (2018). DOI: 10.48550/ARXIV.1801.04381. URL: https://arxiv.org/abs/1801.04381.

[31] Deokjin Seo, Heung-Seon Oh, and Yuchul Jung. "Wav2KWS: Transfer Learning From Speech Representations for Keyword Spotting". In: *IEEE Access* 9 (2021), pp. 80682–80691. DOI: 10.1109/ACCESS.2021.3078715.

[32] Suwon Shon et al. *SLUE: New Benchmark Tasks for Spoken Language Understanding Evaluation on Natural Speech*. 2021. DOI: 10.48550/ARXIV.2111.10367. URL: https://arxiv.org/abs/2111.10367.

[33] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556.

[34] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. DOI: 10.48550/ARXIV.1409.4842. URL: https://arxiv.org/abs/1409.4842.

[35] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: (2019). DOI: 10.48550/ARXIV.1905.11946. URL: https://arxiv.org/abs/1905.11946.

[36] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[37] P. Warden. "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition". In: *ArXiv e-prints* (Apr. 2018). arXiv: 1804.03209 [cs.CL]. URL: https://arxiv.org/abs/1804.03209.

[38] Kelvin Xu et al. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. 2015. DOI: 10.48550/ARXIV.1502.03044. URL: https://arxiv.org/abs/1502.03044.

[39] Richard Zhang, Phillip Isola, and Alexei A. Efros. *Colorful Image Colorization*. 2016. DOI: 10.48550/ARXIV.1603.08511. URL: https://arxiv.org/abs/1603.08511.

[40] Xiangyu Zhang et al. *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices*. 2017. DOI: 10.48550/ARXIV.1707.01083. URL: https://arxiv.org/abs/1707.01083.