Deep learning techniques for audio recognition and classification

Jordi Garriga Munoz

June 16, 2024

Abstract

The world of Artificial Intelligence is advancing faster and every day we see innovations in all its facets, especially in the processing and generation of images and video. Current deep learning techniques such as Convolutional Neural Networks or transformer models have enabled great advancements never thought of before.

This work aims to introduce the reader to the use of these techniques adapted to sound and audio processing. To do this, a specific field such as sound recognition and classification has been chosen.

The work is divided into two parts: on one hand, the basic theoretical concepts of this discipline are introduced: sound characteristics and analog-to-digital conversion methods, processing, and subsequent treatment. It also explains the applications currently available on the market and the different studies and research carried out by various researchers in this field.

On the other hand, it aims to show a specific practical case of the use of deep learning for sound classification. Through the creation of a properly trained model from a dataset with a large number of sound references, it should be able to identify and classify sound fragments of the same type with the highest level of accuracy possible.

To do this, a combination of various techniques will be used, encompassed within a theoretical concept known as CLAP (Contrastive Language Audio Processing), which uses CNNs to process sound fragments from the training set along with text labels describing the sound contained in the fragment.

Index

List of figures

List of tables

1. Introduction

1.1. Context and justification of the work

Over the past two years, we have seen the explosion of the use of Artificial Intelligence (AI) beyond the academic and professional environment. New tools for generating and understanding texts are a demonstration of the enormous possibilities that deep learning offers to help computers interpret the real world.

One of the fields that has generated the most excitement is video and image generation. OpenAI recently presented Sora[1],Your video generation tool, with amazing results and other solutions like Midjourney[2]o Stable Diffusion[3]have not stopped advancing in this same field. Given the close link between image and sound, it is therefore necessary to progress in the application of these technologies in the latter field.

Especially when it comes to sound recognition, we have seen great advances in the field of speech recognition.[4],which have given rise to speech-to-text or even speech-to-speech solutions, being able to create virtual assistants that allow a certain level of conversation with the user, but the same does not happen with the other sounds that surround us: the sound of animals, combustion vehicles, industrial machinery working, and even the sounds of the human body itself, such as the heartbeat or breathing.

The proper treatment of these sounds through AI systems has given rise to countless applications. The most popular are probably the solutions to separate the different instruments present in a song or musical piece; among them, Demucs stands out.[5],The solution was created by a former engineer from the technology company Meta (formerly Facebook) and can be managed through its own Application Program Interface (API), but the range of possibilities is enormous: from medical applications to detect diseases from sounds (breathing patterns or malfunctioning of some organs) to accessibility improvements for people with disabilities. In this last field, sound recognition systems could be of great help to people with hearing or hearing limitations, as well as in transcription or simultaneous translation systems. Currently there is already software that uses AI and sound recognition to add audio descriptions in films or TV shows, for example.

From a more global perspective, there is a broad consensus that sound and audio should go hand in hand with computer vision research, as well as video generation, to create a more complete user experience and facilitate the interaction of machines with the environment. Good work in this field can allow a humanoid or industrial robotic system, through the installation of acoustic sensors, to recognize and define the space in which it is located; this can also help to increase the perception that machines have of the world around them. Some of the most recent advances can be seen in the Internet of Things (IoT) industry, such as this project from Imperial College London.[6]where a sound recognition system has been developed underground using sensors to recognise the environment and know how far away objects are by combining vision with sound.

I hope, then, to shed some light on a topic that I find fascinating and whose research goes far beyond what we will see here; however, I think that this can be a good starting point for anyone interested in the world of sound, music and audio in general. We must not forget that, after all, sound in all its forms is an essential part of human communication.

1.2. Goals

As explained above, the work seeks to achieve two main objectives: to deepen the basic concepts of digital audio processing and to demonstrate in practice how sound recognition and classification can be carried out using deep learning techniques. More specifically, the objectives are:

Theoretical, on the basic knowledge of the field

- To introduce the reader to the most basic concepts of digital audio processing in order to correctly understand what is being discussed and form their own opinion.
- Document and summarize the techniques used so far in this field.
- Research the tools, ideas and solutions that are currently being worked on in the field of research.

Practical, on the model to be built:

- Selection and justification or construction of a data set that will serve as a basis for creating the model.
- Treatment (preprocessing, cleaning and selection) of the chosen set.
- Creation of training and test sets duly justified.
- Model training.
- Evaluating the model with the test sets.
- Modify or refine the model if the metrics are not as expected.

Finally, the aim is to make the model capable of handling more advanced data inputs:

- That it can recognize sounds from sets other than the chosen one; with this we will have a good metric of the real effectiveness of the model, since this type of data set is not abundant, and using data from the same sources can corrupt the model.
- Given a sound scene in which different sounds are mixed, it is capable of distinguishing and identifying each of them separately, which is known as audio segmentation, one of the most useful applications of audio classification today.

1.3. Focus and scope of work

The main idea of the work is to create a model with the Python programming language, taking advantage mainly of the different libraries that we have available for audio processing. Most of these libraries use deep learning algorithms to train the models and methods for cleaning and processing data specific to audio.

To train the model, we will use a set of sound recordings, which are short audio fragments that can range from 2 seconds to almost half a minute in length. These sounds are accompanied by a metadata file, usually in .csv format, which contains a textual description of the corresponding sound file.

The result should be an inference model that should be able to analyze an input sound fragment and recognize which class it belongs to (what type of sound it is). The output information of the model will be conditioned by the type of training data, so depending on the structure of the training data, the inference data will be complete.

The method will be as follows:

1. *A data set appropriate to the needs of the work will be chosen*. To do this, several factors will be considered: size of the set, variety of sounds, techniques used to obtain the sounds (microphones, sampling quality, etc.) and data labeling.
2. *The dataset will be preprocessed to fit* correctly to the training needs.
3. *A model will be chosen,* based on contrastive language techniques, which will be explained in depth later in the work.
4. *The model will be trained.* Training and validation sets will be created; various validation techniques and metrics will be tested.
5. *The model evaluation will be carried out,* first with the reserved test data from the set itself and later with data from other sources.

In this context, the choice of the data set is particularly sensitive, since its metadata, which contains the textual description of the sound file, largely defines the accuracy of the results obtained. The more specific the tagging of the files, the more specific our model can be, but it is also more likely to be confused or give unexpected results.

This entire strategy is encompassed, at a higher level, in the CRISP-DM method.[7] Data mining: understanding the business, understanding the data, preparing the data, modeling, evaluating, deploying and iteratively refining the model based on the evaluation results.

Citations and bibliographical references will be in ISO690 format.

2. Theory of sound

Before creating or analyzing any type of sound classification model, it is necessary to explain some basic concepts about the nature of this model, as well as some basic mathematical concepts of digital audio processing, essential to understand how the same type of models work.

2.1. Definition

In this work, the fundamental pillar is the concept of sound. According to the *Institut d'Estudis Catalans*, sound is defined as:

> *"Impression produced on the organ of hearing by the elastic vibrations of a body which propagate in all material media in the form of waves."*

Figure 1 – Definition of sound

The body producing the vibrations can be any: the human body itself, an object, an acoustic musical instrument or an electrical amplification device, such as a speaker.
Although the definition already makes it clear that sound can propagate through any medium, it usually propagates through the air, specifically in the form of sinusoidal waves.

2.2. Basic mathematical concepts for sound manipulation

Throughout our work, we will need to manipulate sound from the computer to achieve a series of transformations or carry out a series of processes. To do so, it is essential to first present some basic concepts of mathematics applied to sound. It is not the objective of this work to go into this field in depth but simply to have a basic knowledge that helps to understand them.
To help us with the explanation, we will mainly use Python software and libraries for sound processing and visualization. In our case, we use Python version 3.11 and the numpy, scipy and matplotlib libraries. For more complex visualizations we will use the Sonic Visualizer software and the torchaudio library.

Sine wave concept

As we have explained above, sound is represented in the form of a wave that has a sinusoidal shape; we can consider that this wave is the basic unit of sound. This wave represents, as we said, a trigonometric function, which oscillates between two values and periodically and with a certain frequency. This wave can be defined, among others, with the following equation:

$$x[n] = A\cos(\omega nT + \varphi) = A\cos(2\pi f\, nT + \varphi)$$

where we have:

- A is the amplitude, the maximum deviation of the function from point 0.
- ω is the angular frequency expressed in radians/second.
- f is the frequency represented in Hertz (obtained by dividing the angular frequency by 2π).
- φ It is the initial phase of the wave, its value when t=0.

We can also usually define time as t , where: $t = nT$

- n is a variable that represents a temporal index and
- T is the sampling rate of the sound or period.

Thus, we can understand a wave as a vector $x[n]$ where n is the number of samples, that is, the values of the function at each instant of time t.

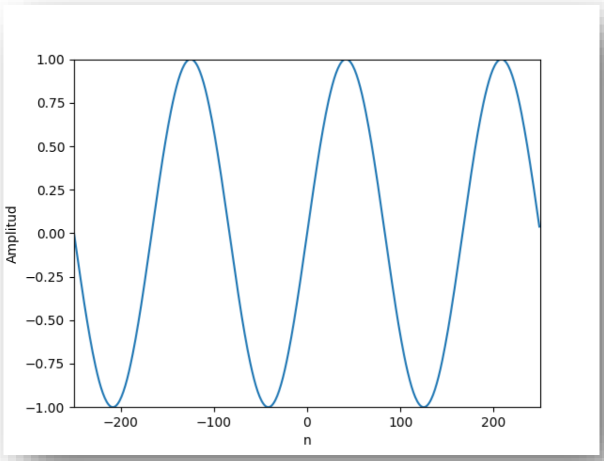


Figure 2 – Elementary sine wave [own creation with Python software].

Discrete Fourier Transform (DFT)

The wave example we saw earlier is a particular case of a sinusoid that represents a single frequency; this is what is known as a purely acoustic tone. The sounds we hear in our daily lives, such as music, noises, even our voice, are, however, composed of several layers of frequencies; this is what is known as the constituent frequencies of sound, which give the sound a timbre, richness and diversity that make it unique and distinguishable from the rest. In this case, the shape of the wave is much more complex than the previous one:

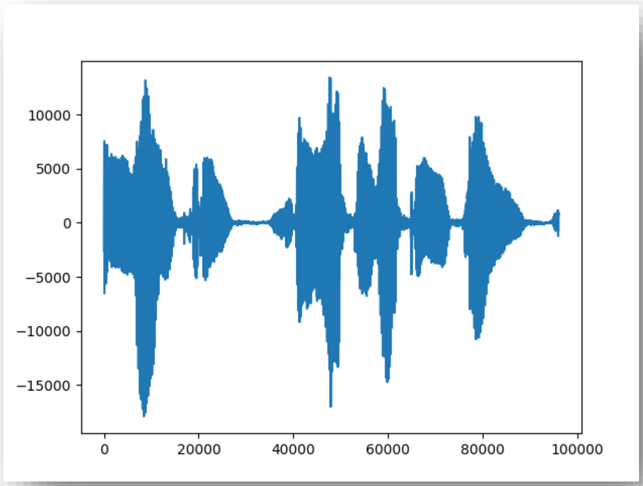


Figure 3 – Complex sine wave [sound file representation with Python software].

In order to extract and analyze these frequencies, we rely on the Discrete Fourier Transformation process, a type of Fourier analysis.[8] on the discrete values that represent the samples of the sound function. Thanks to this transformation, we obtain a new sequence of samples x[n], which we call a spectrum x[k], and which represents the harmonics present in the sound. The DFT is represented by the following equation:

$$X[k] = \sum_{N=0}^{N-1} x[n]e^{-j2\pi kn/N} \qquad per\ a\ k = 0,1,\dots,N-1$$

where we have:

- n is the discrete time index
- k is the discrete frequency index or period of the wave
- N is the number of samples
- $x[n]$ is the function that represents the input signal, with n samples
- $X[k]$ is the output function (spectrum)
- $e^{-j2\pi kn/N}$ is the so-called complex exponent or basis function of the DFT

Furthermore, it is worth remembering that to obtain the absolute frequency we have the formulas:

- $\omega_k = 2\pi\, k/N$, which gives us the frequency in radians/second
- $f_k = f_s\, k/N$, where f_s is the sampling frequency, gives us the frequency in Hertz

By multiplying the two DFT functions we obtain the complex value of each sample of the input function. We then add all the samples together to obtain the final spectrum:

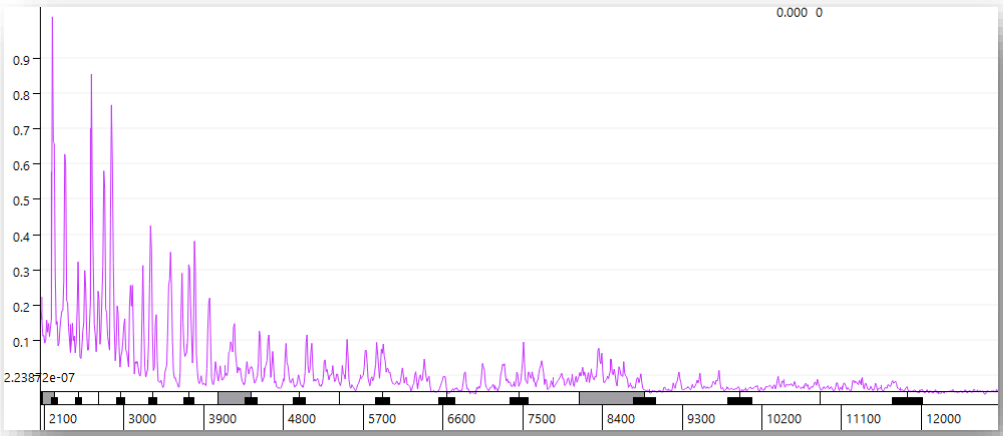


Figure 4 – Wave spectrum [own creation with Sonic Visualizer software].

We could define the spectrum as a projection of the original signal, from which we can obtain the magnitude, which indicates all the sinusoids present in the signal, and the phase, which indicates its position with respect to time t=0. It can be said that a spectrum will have two parts, corresponding to the oscillation of the original wave: the positive and the negative. Both parts are symmetrical, and therefore, for simplicity we will represent only one of the two.

On the other hand, we have seen the representation of the spectrum where the axis indicates the value of the amplitude at a given moment in time. This is obtained by computing the absolute value of the original sinusoid, which is a linear value. A more intuitive way of representing it is on a logarithmic scale, with which we can better visualize the oscillations:

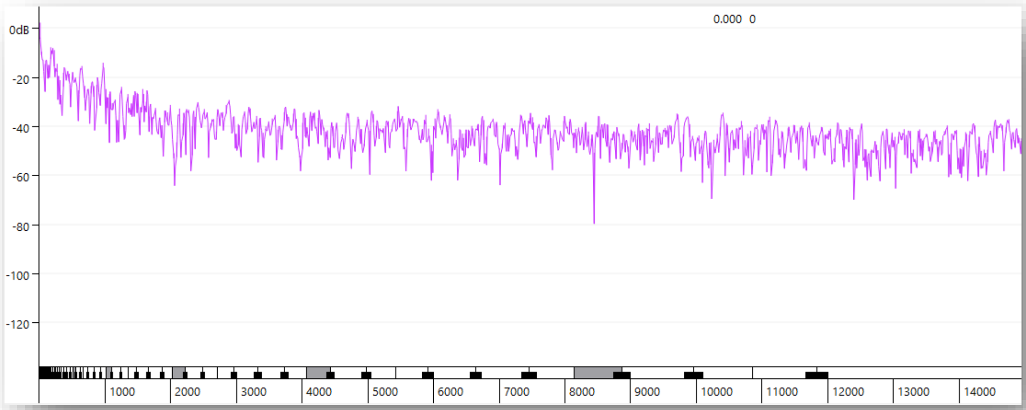


Figure 5 - Wave spectrum in decibel scale [own creation with Sonic Visualizer software].

In the field of sound processing, decibels are commonly used as a measure of amplitude. This measure is the most accepted and represents the sound pressure level of a wave, where sound pressure is understood as the difference in atmospheric pressure generated by a sound wave. To calculate the logarithmic scale of decibels, a reference sound pressure is taken as a base measure and the logarithm is calculated:

$$L_p = 20 \log_{10}\left(\frac{p}{p_0}\right)$$

where p is the sound pressure, we want to transform and $p0$ is the reference pressure, $20\ uPa$. This reference is considered the absolute threshold of hearing [9],and therefore, the spectra will represent values between 0 and 20 microPascals, which are the values that humans are considered capable of recognizing.

Fast Fourier Transform and Inverse Discrete Fourier transform

The mathematical process of the DFT is complex and although we can find a large number of algorithms to carry out an efficient implementation on a computer, the computational cost is always around $O(n2)$. To improve the efficiency and speed of calculation, a special implementation of the DFT known as Fast Fourier Transform (FFT) is commonly used in computer science. Several algorithmic implementations of this formula achieve costs close to $O(N\ log\ N)$, qualitatively improving efficiency. Thus, we can see in many published papers that, in reality, the equation used to perform wave analysis is the FFT.

In the same way that we have carried out the analysis process on a signal thanks to the DFT and the FFT, we can also do the inverse process thanks to the Inverse Discrete Fourier Transform (IDFT) in which, given a function that represents the spectrum, we multiply it by the positive base function, do the sum and multiply the result by $1/N$, where N, as before, is the number of samples.

$$x[n] = \frac{1}{N}\sum_{N=0}^{N-1} X[k]e^{j2\pi kn/N} \qquad per\ a\ k = 0,1,...,N-1$$

In this way we can carry out the wave synthesis process: from an analyzed spectrum, we obtain the exact original sinusoid from which it comes.

Short-Time Fourier Transform

The Fourier transformations we have seen so far are applied to an instant in time; we could say that they are like a photograph, a "snapshot" of the frequencies at a given instant. This is not enough to extract the information needed for our model; we need a continuous representation of the spectrum throughout the entire duration of our sound. We can obtain this representation thanks to the Reduced Time Fourier transformation (also called Windowed). In this, we divide the input signal into small segments of the same length and calculate the DFT of each one, following the following equation:

$$X_l[k] = \sum_{n=-N/2}^{\frac{N}{2}-1} w[n]\cdot x[n+lH]e^{-j2\pi kn/N} \qquad para\ l = 0,1,...$$

where:

- $W[n]$is the analysis window.
- L is the frame number, the start time of the analysis.

- H is the Hop, the jump value, which will mark the size of the analyzed fragment.

The STFT equation is practically the same as the DFT equation but adding the concept of analysis window[10]:The window is a function that defines a series of values in an interval; within this interval, the function filters out any values, and outside that interval, the values are 0. By multiplying by the window function, we get an accurate picture of the constituent frequencies, since we filter out those that do not interest us.

However, you should be careful with the window values: a high sampling value allows you to filter more frequencies and therefore have a more accurate image of the harmonics present in the sound; a lower value, on the other hand, gives you a noisier image but with much more defined time signatures. This particularity of the window size can be used to fine-tune the model according to the types of sounds you have.

2.3. The spectrogram

The sinusoidal shape of the wave does not give us enough information to create a model on its own, since it only gives us an image of the amplitude and frequency at a given time. To try to capture all the constituent frequencies of a sound, we can resort to the spectrogram that is derived from it. From the STFT that we have seen before, we can obtain a graphical representation of the constituent frequencies of our sound at each moment in time. We call this image a spectrogram:

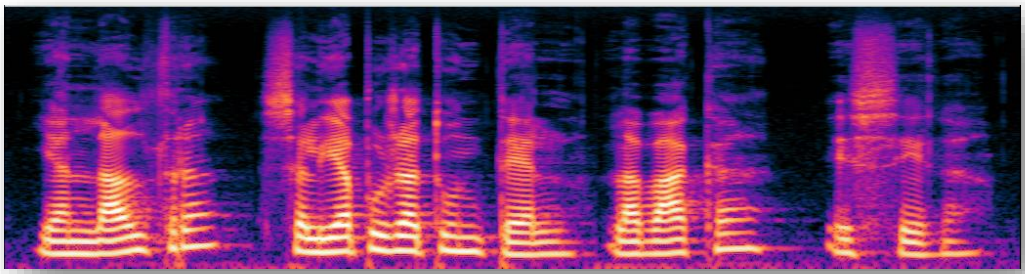


Figure 6 – Spectrogram [extracted from sound fragment with Audacity software].

The most common form of a spectrogram is a 2-dimensional image, where the vertical axis is frequency, the horizontal axis is time, and the colors represent the magnitude of each frequency over time.

This spectrogram will be the basis on which we will build our model. As we explained in the methodology section [link], the models will be built based on the similarity between two images that represent the specific sound. These images are called spectrograms. The fact that the spectrogram has this format makes it ideal for working with convolutional neural networks (CNN) and being able to extract their main characteristics, which will allow the model to search for similarities between sounds.

Mel's spectrogram

To further refine the results, our spectrograms will contain a logarithmic representation of the amplitude values of the frequencies. In a spectrogram like the one in the image, the values on the x-axis represent the absolute value of the frequencies represented. However, humans have a significant auditory tolerance to frequency changes, and, for example, the ear does not always perceive the difference between two frequencies in the same way. Specifically, for low frequencies (up to about 1000Hz) humans perceive a small difference between frequencies very clearly, but this same difference is practically imperceptible at high frequencies. To give a concrete example, between 1000Hz and 1100Hz, the difference is much more significant than between 10000Hz and 10100Hz, where it will be practically impossible for us to clearly distinguish the two frequencies.

To alleviate this problem and make the frequency analysis of the spectrogram more realistic, in 1987 the Journal of the Acoustical Society of America[11]implemented Mel's scale:

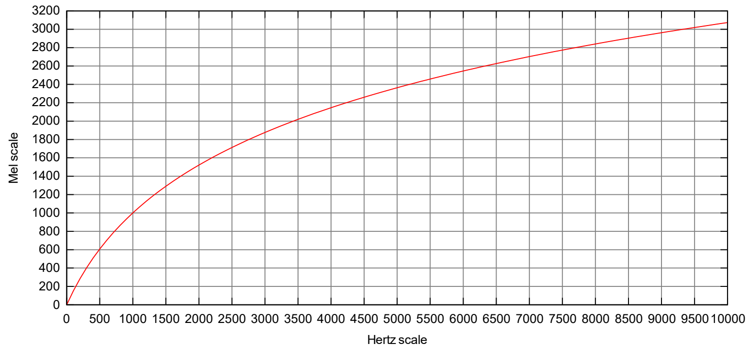


Figure 7 – Mel's Staircase [Wikimedia Commons, original by Krishna Vedala].

From the experience of similar jobs[12],We see that we can obtain better results if we apply this scale to the spectrograms, which results in what is known as a Mel spectrogram. This is the final type of image we will work with.

2.4. Digital representation of audio

Finally, we must also mention how we represent sound in a computer. In order for machines to be able to understand and manipulate audio files accurately in digital systems, we need a series of processes to convert analog sound waves into digital data.

Analog vs. Digital

Analog sound is continuous and infinite, similar to the natural fluctuations of sound waves in the environment. We usually think of analog as "natural" sound; analog actually refers to the nature of the representation, not the sound itself. In contrast, digital sound is discrete and finite, represented as a series of binary digits (bits). This representation involves sampling the analog signal (creating samples at regular intervals, as we have seen in the previous sections) and quantizing each sample to a specific amplitude level. Digital audio offers advantages in terms of precision, handling, and preservation compared to analog audio; and with current technology, digital sound can be of equal (or even superior) quality to its analog counterpart.

Analog-Digital Conversion

To convert analog sound to digital format, an analog-to-digital converter (ADC) is used. The ADC samples the analog signal at a specific rate (measured in kilohertz, kHz) and quantizes each sample into binary values. The accuracy of the digital representation depends on factors such as the sampling rate and the bit depth (the number of bits used to represent each sample).

In the reverse process, a digital-to-analog converter (DAC) is used to reproduce digital audio to convert digital data into analog sound waves. The DAC reconstructs the original analog signal by interpolating between the discrete digital samples, producing a continuous output that approximates the original waveform.

File Formats and Extensions

Digital audio files are saved in a variety of formats, each with their own compression algorithms and file extensions. Some of the most popular audio files include MP3 (MPEG Audio Layer III, widely used in online music streaming services), AAC (Advanced Audio Coding, used especially as audio tracks in videos), FLAC (Free Lossless Audio Codec, high-quality digital audio, much loved by audiophiles), and WAV, which stands for Waveform Audio File Format. WAV files typically contain uncompressed audio data, preserving the original sound quality but resulting in larger file sizes compared to compressed formats like MP3. Due to their lossless nature, WAV files are often preferred for professional audio production. This type of format is capable of generating very realistic and highly accurate spectrograms; it is therefore the format we will use for the audio tracks of our model.

3. Creating the model

After a theoretical introduction, we can start building the model based on everything we have learned. We will start by explaining the importance of the dataset for this type of work, then we will introduce the most important concepts about neural networks and deep learning adapted to audio classification, along with a review of some of the most outstanding research work in recent years, and we will finish by setting up the classification model project, which we will upload to a GitHub repository.

3.1. Choosing and justifying the dataset

As we have already explained, the objective of the work is to create an inference model using deep learning techniques that allows recognizing sounds from audio fragments.
We are generally aware of the importance of data sets for the success of the model; in our case, the scope of the project, due to the time and tools available, is limited, and in order to be able to correctly develop the model in all its phases, trying to obtain good results, the choice of the training data set for the model is especially relevant.

The sets

There are currently a number of data sets available on the Internet focused on sound processing and recognition. These sets must have certain characteristics: specifically, they must be made up of pairs of elements$\{audio, metadades\}$, where $audio$ is a sound file in .wav format, of short duration (between 1 and 15 seconds maximum) and $metadades$ is a text file (usually in .txt format) containing a textual description of the sound. For the sake of simplicity, it is expected that the audio tracks be in mono (although in a set with stereo tracks we could also convert them to mono before processing them), of a duration more or less equal to each other, and that the text files have a similar structure to each other.
The most representative sets today, and those that could best adapt to our needs are:

- **ESC-50**[13], is a set of 2000 samples of 5 seconds duration, organized in 50 classes, with 40 sounds per class; it is focused on the classification of environmental sounds, and the samples are divided into 5 large groups: animals, natural scenes, non-verbal human sounds, domestic sounds and urban sounds. The sounds are extracted from Freesound.org and selected manually.
- **UrbanSound8k**[14], UrbanSound8k contains 8,732 sounds of different lengths (but no longer than 4 seconds), classified into 10 sound types; the classes are much more specific than those of the ESC-50. The metadata comes in a .csv file and contains, apart from the textual description of the sound, various information about it. UrbanSound8k also has the particularity that all the samples are organized into 10 folders, with the data expressly arranged in this way to be able to apply a 10-subset cross-validation system (Fold Cross Validation).

- **FSD50K[15],**The FSD50K is probably the most complete available (not counting sets that include video), with around 51,000 samples labelled in around 500 different classes. In addition to the amount of data available, the FSD50K is particularly interesting because the sources from which the audio fragments are obtained are very diverse, and of varying quality: it is designed to contemplate all possible scenarios in a realistic environment: the sound does not always have the same quality, or duration, or come from the same place; the same occurs with the metadata, they do not have a clear structure, which requires a very intensive prior treatment of the set.

Obviously there are more sets available in open-source format: the DESED[16],Very focused on domestic and indoor sounds, the MS SNSD[17], which contains fragments of speech mixed with ambient sounds, etc. But we believe that these three are the most representative of the direction of the work, and specifically, we opted for UrbanSound8K, although we do not rule out the possibility of using cross-resources from the other two to check the effectiveness of the final model.

Structure

UrbanSound8K is composed of a total of 8132 audio fragments, which are classified into 10 types of urban sound, present mainly in densely populated city environments. Each class has an identifier, from 0 to 9, as follows:

0 = air conditioning (the machine that is on the outside of the buildings)
1 = car horn
2 = children playing (does not distinguish volume, speech, boy or girl)
3 = barking dog
4 = drill
5 = car engine (idling, while waiting at a traffic light, for example)
6 = gunshot (does not distinguish between types of gun or bullet)
7 = hydraulic hammer
8 = siren (does not distinguish between siren and vehicle types)
9 = music in the street (a band playing or someone with a radio)

The metadata comes in a .csv file that represents a data frame where each row corresponds to one of the audio fragments and its information.

3.2. Methodology

Before starting to work on the model, we will make a quick overview of the methodology that we will use.

Thanks to the libraries and software available in Python, we will be able to analyze the physical characteristics of the sound file of the set: frequencies, amplitude and spectrogram analysis. Using a method similar to CLIP (Contrastive Language Image

Processing), in which sets of images are labelled with a text label that defines what is in the image, a self-supervised learning method will be created, which some authors have named CLAP (Contrastive Audio Language Processing).

The model will be based on convolutional neural networks (CNN). CNNs are a deep learning architecture widely used in computer vision and image recognition. The neural network uses a filter (kernel) that runs through an input image and applies a series of mathematical operations that generate what is known as convolution, a process that can be used to extract the most important features of each part of the image and, ultimately, of the entire image.

Here we will use a convolutional neural network architecture to create the feature maps corresponding to the sound form of each file; we will encode text and audio separately and implement a linear classifier to create the inference.

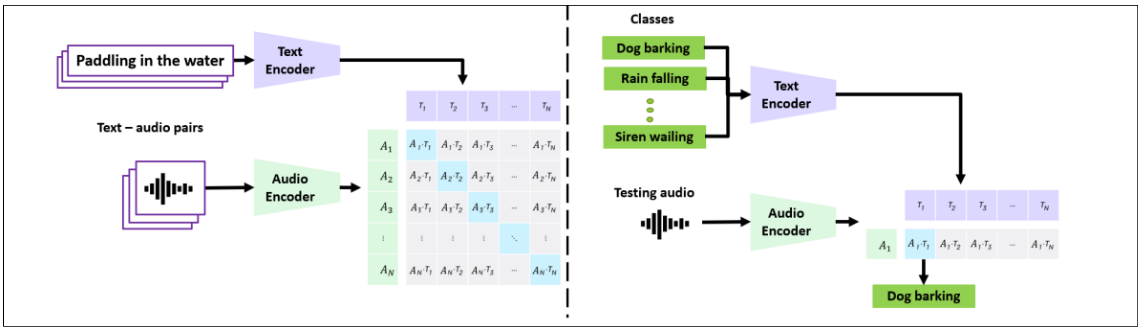


Figure 8 - Operation diagram of the CLAP system. Extracted from Learning Audio Concepts From Natural Language Supervision[18].

3.3. State of the art

Over the last 10 years, many works and studies have proliferated around sound recognition thanks to AI solutions. Each one in its own way, all provide different practical approaches with a similar background to that explained in the previous section: the transformation of an audio track into a spectrogram to be able to work with it as if it were an image, taking advantage of the great power and the widely recognized effectiveness of convolutional neural networks to process and classify images.

CNN and the images

Neural networks have proven to be one of the best architectures for 2D image recognition. As we saw in the subject of Computational Learning, CNNs are a type of neural network that consists of several layers with different functions each, and that progressively filter the images to extract their intrinsic characteristics and therefore, can help to identify and distinguish them from each other. This feature extraction is carried out hierarchically through convolutional layers, where in each convolution the image is refined, recognizing increasingly more complicated parts, textures or patterns as the network is deepened.

Apart from the convolutional layer, which is where feature extraction is mainly performed, CNNs also have:

- A **ReLU** (Rectified Linear Unit) activation function: This activation function is applied after convolution and introduces nonlinearity into the model, allowing the network to learn more complex relationships. The ReLU function is defined as:

  $$\boldsymbol{ReLU}(\boldsymbol{x}) = \boldsymbol{max}(\boldsymbol{0}, \boldsymbol{x})$$

  The function takes an input value and returns if it is positive, and if it is negative.$\boldsymbol{xxx0x}$
- A **pooling** layer (max-pooling or average-pooling), which serves to reduce dimensionality in a controlled manner, to speed up calculations and to provide invariance to the model, thus avoiding overfitting.
- A second network of **fully connected layers** located at the end of the CNN takes the result of the last convolution and performs the relevant calculations, allowing the subsequent classification or regression task.
- A **flattening** process, just before the fully connected layers, that converts the output tensors of the convolutional layers, which are multidimensional (usually $\boldsymbol{hxwxd}$ in the case of images, 3-dimensional, where $\boldsymbol{h}$ and $\boldsymbol{w}$ are the height and width of the 2D image and $\boldsymbol{d}$ is the number of channels or features) into one-dimensional vectors that can be treated as arrays allowing the computer to perform the necessary calculations.

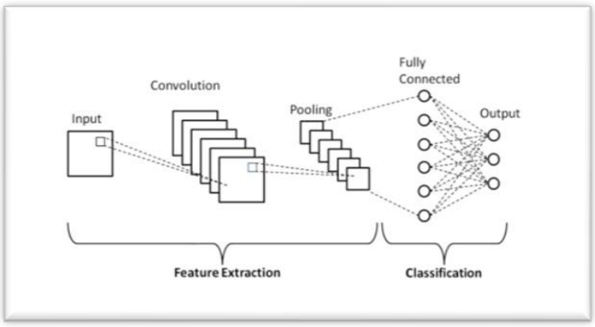


Figure 9 - Basic structure of a CNN. Extracted from A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. PHUNG Van Hiep, RHEE Eun Joo.

The process

With all the elements described in the previous point, we can start a CNN. A random initialization of the weights is performed, which are the values that will be used in each node of the network to calculate the output value based on the input.

The method known as forward propagation is used, in which the input values are passed through the network until a prediction is obtained. The model then calculates the loss function, based on the differences between the prediction and the real class of the image labeled during training. With this function, the gradient of the weights is calculated, which will be used to adjust the weights of the nodes to minimize the loss function, and consequently improve the accuracy of the model. To do this, the method known as backpropagation is used, in which the loss is searched for at each point in the network and the gradient is optimized to adjust the weights again and update it.

Each complete cycle of this process is called an epoch; in a model training process we will run the model for several epochs to improve the evaluation metrics. As we increase the epochs, gradient descent optimizes the weights better and helps increase the accuracy of the predictions. However, we must be careful with the epochs: too small a number will not give the model enough information to classify correctly; too large a number will make the weights too fine-tuned to the specific training data and will make it difficult for the model to generalize (its ability to recognize and classify data other than the training set).

CNNs in audio recognition

Based on what was explained in the previous point, numerous experiments have been carried out using CNNs for audio recognition from the sound spectrogram.

The idea is to perform the Fourier analysis process on an audio track (a sound file on the computer, usually in .wav format) and extract its constituent frequencies and how they are positioned in time, that is, to extract the spectrogram, as we have already seen in point 2 of the work. This spectrogram, for the computer, is nothing more than a 2-dimensional image, and therefore can be processed through a CNN as such. In this way, the network extracts the unique characteristics of the image, and consequently, of the sound that interests us. With this data, a trained model can subsequently recognize and classify another similar sound, comparing both images. This method has proven to be very efficient in audio recognition.[19], especially in the field of speech recognition, where they are the basis of the vast majority of work since mid-2010[20].

Apart from a large amount of academic work on the subject, there are currently several initiatives underway to promote the application of these solutions (or others) to audio recognition; perhaps the most significant is the DCASE Challenges, an event organized every year since 2013 by several internationally renowned universities (the first was held at Queen Mary University of London) in which various tasks are proposed in the form of challenges of varying difficulty. Some of the greatest advances in this field have come from DCASE.

The Contrastive Language Methodology

The second part of recognition is the processing of the text labels that accompany each sound track. The created model must be able to process each text label and compare it with the sound to return the correct class of the audio track. At this point we find a variety of solutions for this processing, although we will base ourselves on the contrastive language method.

Specifically, we are interested in CLIP (Contrastive Language Image Processing), a type of self-supervised learning method that we train with large amounts of data conveniently labeled with text files and that is subsequently capable of inferring, given an image, the text that can best define it.

During training, these models take the separate inputs of the sound fragments and the metadata with the descriptive text (which have different dimensionalities at the start) and encode them to project them into the same space, called multimodal space. Once in this space, both sets of data can be worked with to create a similarity matrix that is used to classify the fragments at the time of making the inference.

The CLAP model

In 2022, the Microsoft research team developed a method they called CLAP (Contrastive Language Audio Processing)[18],an adaptation of CLIP to generate the text that defines a spectrogram, and therefore, the sound it represents. Unlike previous audio recognition models, capable of classifying a sound based on a class identifier, which could be a word or a short phrase, CLAP processes the metadata tags of the sets with a natural language supervision tool that allows it to generate prompts or entire phrases that define the sound scene with different levels of precision. This can be an especially useful tool for tasks such as the generation of subtitles with audio description and opens the door to the integration of sound in the world of content generation with Artificial Intelligence.

Other models have also shown great advances in this field, such as the Wav2Clip.[23], one of the first studies to extensively test the effectiveness of this method on the different data sets available.

In both cases, much interest has been placed on the models' ability to make zero-shot predictions; this means that both CLAP and Wav2CLIP can be generalized to new audio classes using textual descriptions, eliminating the need for specific labeled data for each new class to be recognized. This is one of the fields where there is most work to be done, since despite the great possibilities it represents, the accuracy levels of the models are still not good enough, and much less superior to those of supervised learning. Recent advances in the field of large-scale languages (LLM, Large Language Models) may lead to experimenting with the implementation of the embeddings resulting from these models to improve this capacity, and therefore make the process more natural, more human-like.

3.4. Creating the model

Finally, it is decided to create a base model to carry out the tests, based, as we have explained in the previous section, on a CNN, which will be the core of the model's processing.
Instead of creating this model from scratch, it has been decided to take advantage of some algorithms that have been found derived from various sources (articles, research groups, specialized publications, etc.). Finally, a base has been created by taking advantage of some of these algorithms and adding other classes and features. The result

is available online in my personal repository: https://github.com/Gardenou/TFG_IA_Model. Links to the resources used are specified in the bibliography [**link**].
For the sake of clarity, I have decided to divide the project into the following classes, each of which serves a specific function:

- The **dataLoader** class loads the data from the dataset and performs preprocessing. The data is then ready to be worked directly with the model.
- The **loadAudio** class defines the data preprocessing methods, which we will see in the next section.
- The **TrainingData** class loads the available data set (in our case, the UrbanSound8K) from the directory where they are stored and generates the training and validation sets.
- The **TrainingLoop** class defines the process of training the model.
- The **Model** class defines the model itself, based on CNNs.
- The **Inference** class extracts model predictions from validation data.
- Finally, the **run** and **train** classes are used to launch all the stages of the model in order; train performs the training and saves the weights to a file and run loads these weights files to create the model predictions.

To complete the project, we later added two utility classes:

- With **calculateMetrics** we will perform a series of basic calculations of the evaluation metrics; the confusion matrix is first created with the prediction results and the values are used to find the precision, the F1-score and the recall.
- The **plotSpectrogram** class is a helper class with which we can view spectrograms on the screen, taking them directly from the data set.

Data processing

Before passing the data into the model, it will be necessary to preprocess it in order to adapt it and improve the results. This is usually one of the points of the CRISP-DM methodology, which we have already discussed before, and in this case, it is fully justified. We will divide the process into two parts: data format processing and data augmentation.
To begin with, we will have to see how the set is organized; in our case we have already explained that we have 10 folders that correspond to the typical folds of a cross-validation system (k-fold cross validation). In addition, we have one more folder that includes the files with the metadata in .csv format. We analyze this file and set up the dataLoader class to extract the corresponding information. Before starting, we can extract a first spectrogram of a channel of the original sound fragment:

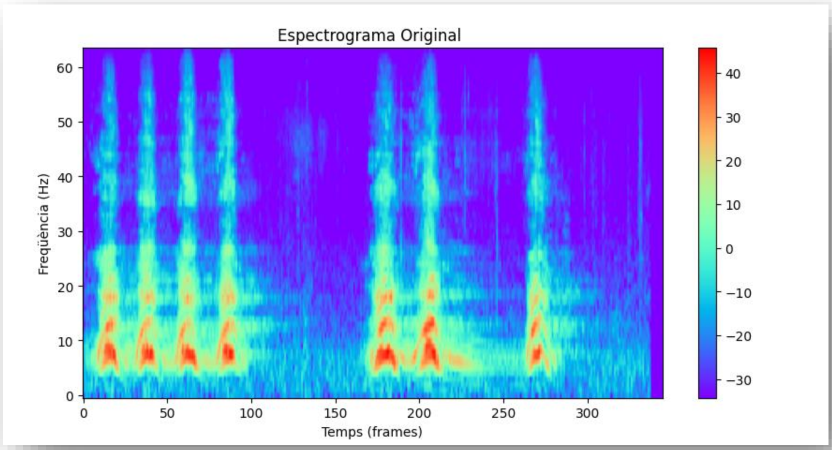


Figure 10 - Spectrogram generated from a sound file from the UrbanSound8K set (file 74723-3-0-0.wav).

In preprocessing, we will have to match the audio fragments; the experience of most consulted works tells us that better results are obtained when the sound fragments are more similar. This fact is similar to the standardization or normalization process that we could carry out on another data set prepared for mining. In our case, we will focus on the following characteristics:

- First of all, we will **equalize the number of channels** on each track. Some tracks are mono (one channel) and others are stereo (two channels). In the case of two-channel tracks, we know that both are equal (they contain the same information), but the spectrogram may be slightly different, due to the recording system; we may find, for example, that the sound was recorded with two microphones, and although the sound is the same, the position of the microphone in relation to the sound could make it vary. We will leave the stereo tracks as they are, and in the mono ones, we will duplicate the channel so that there are two.
- We will also **stabilize the sampling rate**, which determines the accuracy of the sound representation; not all tracks had the same sampling rate, which prevents the CNN calculations from being performed correctly (since the arrays representing each file do not have the same length). After performing some initial experiments guided by the available literature, we will equalize all tracks to 44100 Hz.
- The **length of the track** (the duration); the average length of all tracks is about 4 seconds. So, we'll truncate files that are longer than 4 seconds, or zero-padding files that aren't.

Once this first processing is done, we will introduce slight variations in the available data in a process known as data augmentation. This process is quite common in the training of image processing models and has proven to be very effective in improving accuracy. In the case of images, slight variations in length or amplitude are introduced, or small rotations are made, or noise is introduced into the image; in this way the model usually recognizes small variations in the data and thus generalizes better. In the case of audio, augmentations were traditionally performed on sound files, since with spectrograms a slight variation can confuse the model and give worse results; it is worth remembering that a spectrogram, despite being treated as an image, is still a space-time representation, and the position of the pixels in the image defines the unique character of the sound. Several studies have worked on this in recent years, and specifically this one by the Google audio research team.[21]of 2019 introduced the idea of increasing the spectrogram by applying three basic transformations:

1. *Longitudinal deformation* (along the x-axis, the time axis): the drawing is stretched or contracted in time, similar to how a timbral transformation is done (pitch deformation, widely used in special sound effects), so that the position and occurrence of each of the frequencies changes slightly, but not the general structure of the sound. It is also known as time shift. At this point, we can see how a final pre-processed image looks like, where we can appreciate the displacement produced by this last method:

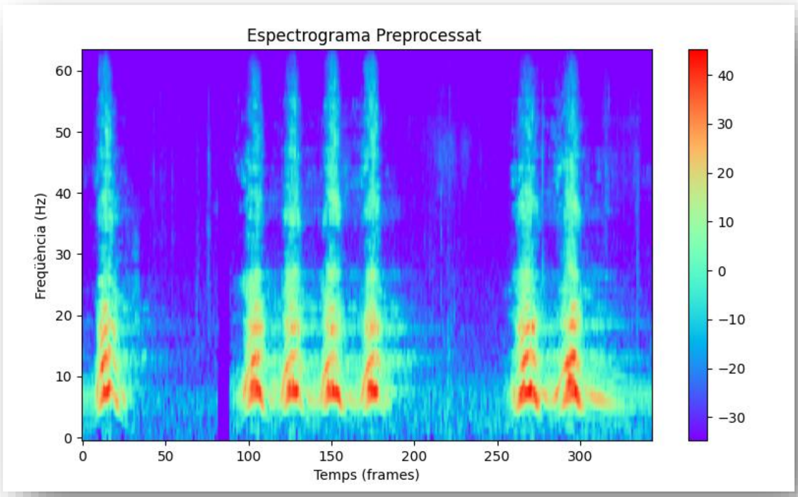


Figure 11 - Preprocessed spectrogram generated from a sound file from the UrbanSound8K set.

2. *Frequency masking*: Random horizontal lines are introduced with the aim of eliminating a series of specific frequencies, which, as the line is completely straight, will always be consecutive and will not affect the rest of the sound frequencies.

3. *Temporal masking*: Random vertical lines are introduced to eliminate temporal occurrences of frequencies; with this we also slightly modify the image, but without unbalancing the overall sound structure.

Additionally, random noise (but in a regular manner) can also be added throughout the spectrogram, or periodic silences can be added; these two techniques have proven highly effective in speech recognition models, but this is not the objective of this work and therefore, we will not use them.
All the transformations mentioned can be implemented with the transforms package, included in the Pytorch library.

Once the entire data augmentation process has been completed, we show the image of the same fragment:

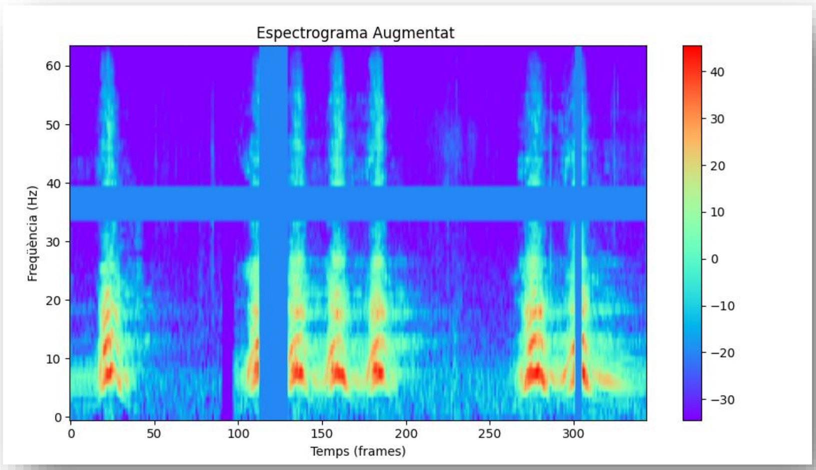


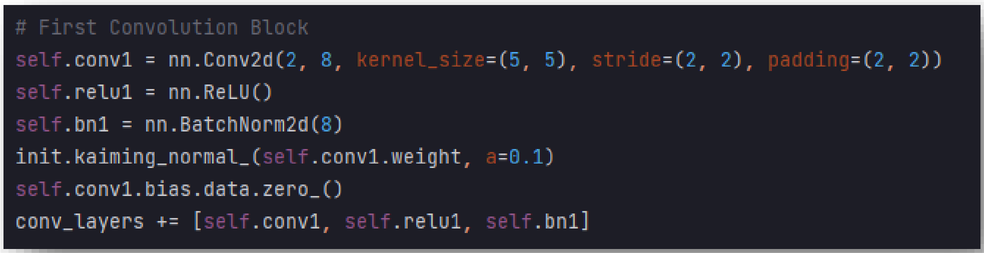
Figure 12 - Spectrogram with SpecAugment and preprocessing, generated from a sound file from the UrbanSound8K set.

We can see the vertical and horizontal lines that represent the temporal and frequency masking, introduced randomly.

The model

In the previous section, the architecture and basic operation of CNNs were defined. In this project, we implement all these elements to create our model. Specifically, we have:

- Four convolutional networks, which we will call blocks. The blocks will be responsible for processing the information. Each block is structured in five layers:



```
# First Convolution Block
self.conv1 = nn.Conv2d(2, 8, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
self.relu1 = nn.ReLU()
self.bn1 = nn.BatchNorm2d(8)
init.kaiming_normal_(self.conv1.weight, a=0.1)
self.conv1.bias.data.zero_()
conv_layers += [self.conv1, self.relu1, self.bn1]
```

Figure 13 - Python schematic of a basic convolutional block of the model.

- The first layer is the neural network itself, implemented with Conv2d; the initial parameters are, logically, the two audio channels (remember that we have transformed all the tracks to stereo), which we will convolve to 8 output channels. We will pass these 8 channels as input to the next block and return 16, and so on until we reach the 64 output channels of the fourth block.

- Afterwards, we apply the ReLU function, as we have explained before.
- Next, we will normalize the batches using the BatchNorm2d function. This type of batch normalization is widely used in CNNs for image processing; its objective is to standardize and scale the output based on the mean and standard deviation of the entire batch.
- The next step will be to initialize the weights of each neuron with a normal distribution, in this case using a Kaiming type initialization.
- Finally, we set the bias values of each neuron to zero before starting to perform the calculations.

We will end up adding the layers to the conv_layers structure to execute it later sequentially.

- A pooling layer, just below the output of the blocks; we will try several methods, but we will start with an AveragePooling.
- The flattening process to convert tensors into linear data.
- The linear classifier, which performs the processing of the descriptive text and, given the data from the previous output, returns the final prediction.

To implement each process we will mainly use the wide variety of methods included in torchaudio, specifically those defined in the nn package, which incorporates all types of processes to work with neural networks.

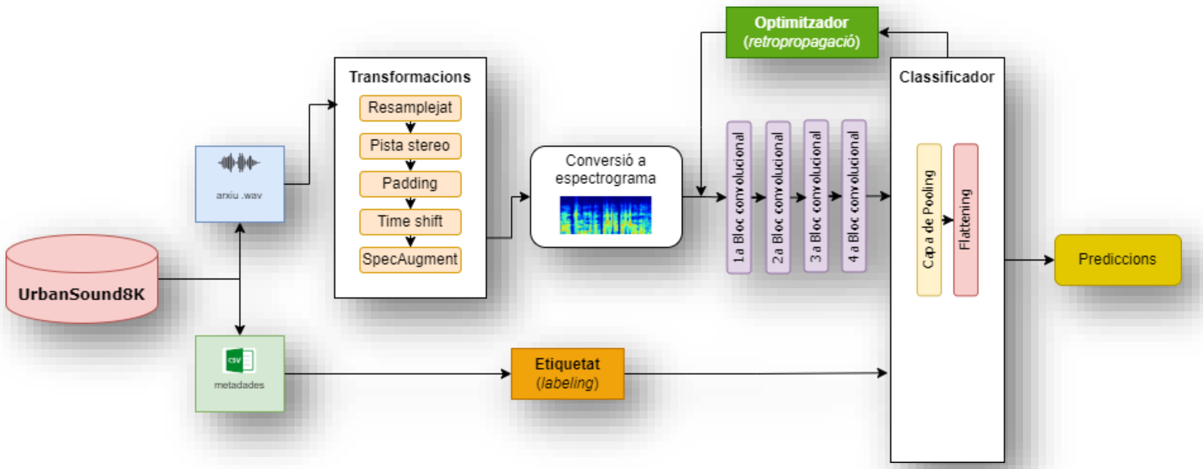


Figure 14 - Diagram of the structure of the model built for the work.

Model training

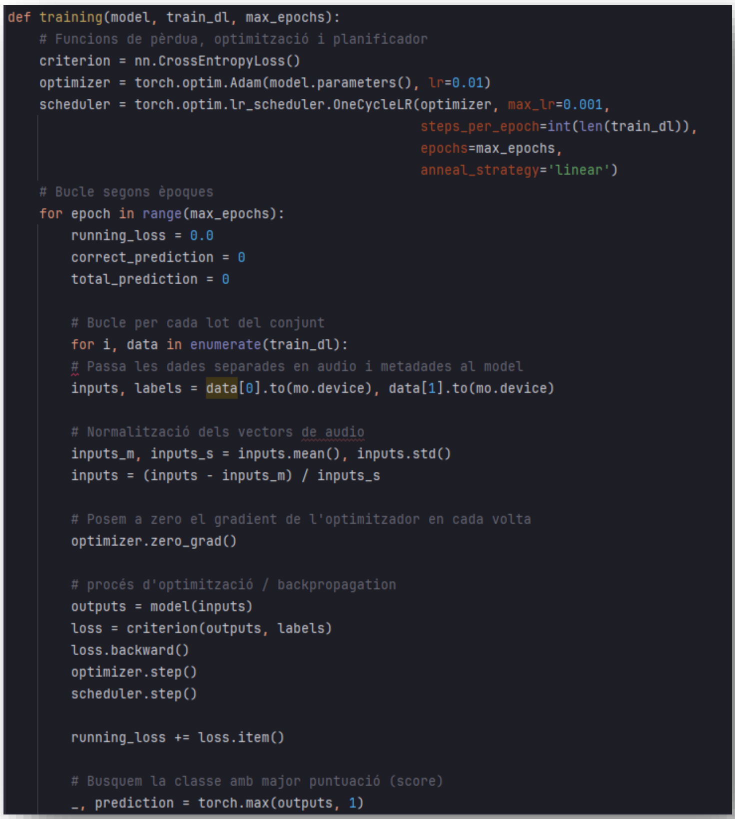
Once the data has been properly processed and the model has been implemented, the next phase of the process is to train it with a portion of the data set available in the game.

Specifically, the fairly widespread rule of 80/20 set splitting has been followed, 80% of the total set to train the model and 20% to validate it and evaluate its inference capacity. Normally, in this part of the process, a method must be found to first conveniently mix the data in a completely random way to avoid over-adjustments and biases in the resulting model. However, UrbanSound is already previously separated into folders with random samples but balanced according to the different classes.

In the TrainingData class we define the local paths where we have the available data, and we extract from the metadata file the information and specific path of each sound file, as well as its category. Then we separate the data into the two sets mentioned, *train_dl* and *val_dl*; each set will be composed of packets that we call batches, which the neural network will process in batches and randomly. Each of these batches is made up of pairs of tensors of the type $\{audio, metadades\}$ with the information that we will pass in the input networks. Each batch contains 16 samples, although we have performed some tests increasing or decreasing the size, mainly to improve the efficiency of the GPU and thus reduce the time of each training.

In the TrainingLoop class we will create the training loop, with the following structure:

- The **loss function,** in the case of image processing, and for audio classification, has been proven[22]that the function that offers better results is the categorical entropy loss function, which in torchaudio we can implement with Cross Entropy Loss.
- The **optimizer**: We will use the Adam optimizer as an input, although in the various works consulted, we have found a large number of uses of other methods, which we will compare when we refine the model.
- The **scheduler,** in which we will define the learning rate, the epochs and the linear attenuation strategy, controlling the variability and the learning rate throughout the training.
- The **loop** itself, in which we are going to separate audio and text to send it to the model, to later collect the results and propagate them back. Finally, we extract the class prediction by taking the label with the highest similarity value.



```
def training(model, train_dl, max_epochs):
    # Funcions de pèrdua, optimització i planificador
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
    scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr=0.001,
                                                    steps_per_epoch=int(len(train_dl)),
                                                    epochs=max_epochs,
                                                    anneal_strategy='linear')

    # Bucle segons èpoques
    for epoch in range(max_epochs):
        running_loss = 0.0
        correct_prediction = 0
        total_prediction = 0

        # Bucle per cada lot del conjunt
        for i, data in enumerate(train_dl):
        # Passa les dades separades en audio i metadades al model
        inputs, labels = data[0].to(mo.device), data[1].to(mo.device)

        # Normalització dels vectors de audio
        inputs_m, inputs_s = inputs.mean(), inputs.std()
        inputs = (inputs - inputs_m) / inputs_s

        # Posem a zero el gradient de l'optimitzador en cada volta
        optimizer.zero_grad()

        # procés d'optimització / backpropagation
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        scheduler.step()

        running_loss += loss.item()

        # Busquem la classe amb major puntuació (score)
        _, prediction = torch.max(outputs, 1)
```

Figure 15 - Code corresponding to the model training process.

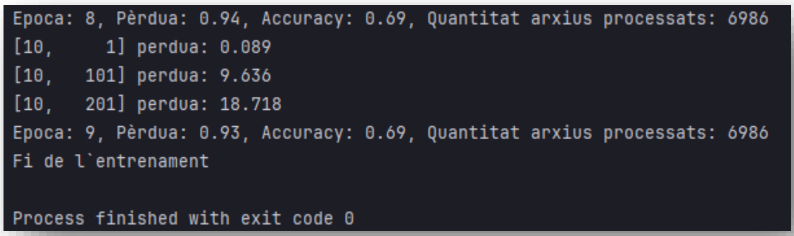
We originally defined a fixed number of epochs, but we found it impractical to have to modify their number each time in each training session, especially considering that the convergence of the model changes with each experiment. We therefore decided to run the training loop with a maximum of 100 epochs (in the different experiments we never went over 80) and a patience parameter. When the model begins to lose precision, we maintain the loop up to the number of iterations marked by this parameter. At this point, we consider that the model no longer improves the results, but rather worsens, and therefore, we recover the results of the last best epoch. Additionally, a .csv file has been created to save data from each of the experiments performed.

Inference

Once the training is finished, the resulting weights are saved in a file from the train class for later reuse. The training has been carried out on a desktop PC, with an Intel i3-10300K CPU, 32Gb of RAM and an Nvidia GTX1650Ti GPU. With the resulting weight files, the model has been used on less powerful machines to be able to carry out the various experiments in various environments. As we have already explained before, to carry out the inference we have used the validation data set created in the TrainingData class; we have created a loop in which we pass the batches of this set to the model, separated into the sound fragment and the text label. The inference class returns a tensor of 10 scored values (scores), comparing the highest value with the real class of the sound fragment (ground truth). Finally, the metrics are calculated with the function described above.

Model evaluation

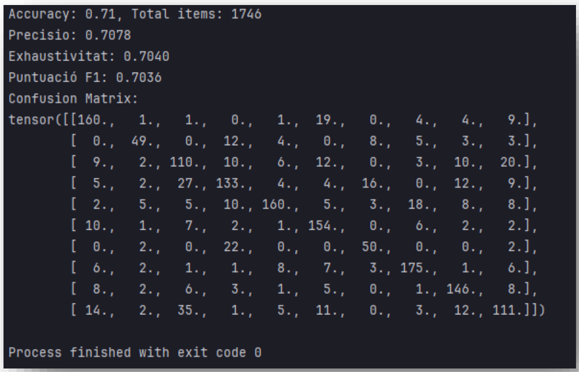
With all the above elements, we start the project with the train class to do the first training with the train_dl data set and save the corresponding weights. To start, we have launched the run class that will make the predictions with the val_dl validation set. We see a first result, with the parameters that we have set by default and running 10 training epochs:



```
Epoca: 8, Pèrdua: 0.94, Accuracy: 0.69, Quantitat arxius processats: 6986
[10,     1] perdua: 0.089
[10,   101] perdua: 9.636
[10,   201] perdua: 18.718
Epoca: 9, Pèrdua: 0.93, Accuracy: 0.69, Quantitat arxius processats: 6986
Fi de l`entrenament

Process finished with exit code 0
```

Figure 16 - Base training result.

The training gives us a training accuracy of 69%, which is obviously not enough to generate a model with a minimum of quality.
When creating the model, we defined in the CalculateMetrics() function a confusion matrix generated from the number of correct predictions of each class: we represent it as a set of 10 tensors, where each one tells us the number of predictions of each class (represented as a position of the tensor). On the diagonal we see the correct predictions. With the results of the matrix, we will extract the rest of the metrics. Next, we launch the program to make inference with the weights obtained in this first training:



```
Accuracy: 0.71, Total items: 1746
Precisio: 0.7078
Exhaustivitat: 0.7040
Puntuació F1: 0.7036
Confusion Matrix:
tensor([[160.,   1.,   1.,   0.,   1.,  19.,   0.,   4.,   4.,   9.],
        [  0.,  49.,   0.,  12.,   4.,   0.,   8.,   5.,   3.,   3.],
        [  9.,   2., 110.,  10.,   6.,  12.,   0.,   3.,  10.,  20.],
        [  5.,   2.,  27., 133.,   4.,   4.,  16.,   0.,  12.,   9.],
        [  2.,   5.,   5.,  10., 160.,   5.,   3.,  18.,   8.,   8.],
        [ 10.,   1.,   7.,   2.,   1., 154.,   0.,   6.,   2.,   2.],
        [  0.,   2.,   0.,  22.,   0.,   0.,  50.,   0.,   0.,   2.],
        [  6.,   2.,   1.,   1.,   8.,   7.,   3., 175.,   1.,   6.],
        [  8.,   2.,   6.,   3.,   1.,   5.,   0.,   1., 146.,   8.],
        [ 14.,   2.,  35.,   1.,   5.,  11.,   0.,   3.,  12., 111.]])

Process finished with exit code 0
```

Figure 17 - Base inference metrics result.

The recall and precision metrics are not excessively bad for a first test: they are both around 0.70. However, an inference accuracy of 71% is not ideal; remember that, in CNN-based models, with the UrbanSound8K dataset, the SO metrics are currently at 98% for the FACE model, which is much more complete, but with the same working base. The 1DCNN model, presented in 2019 and very similar to ours, is around 90%.

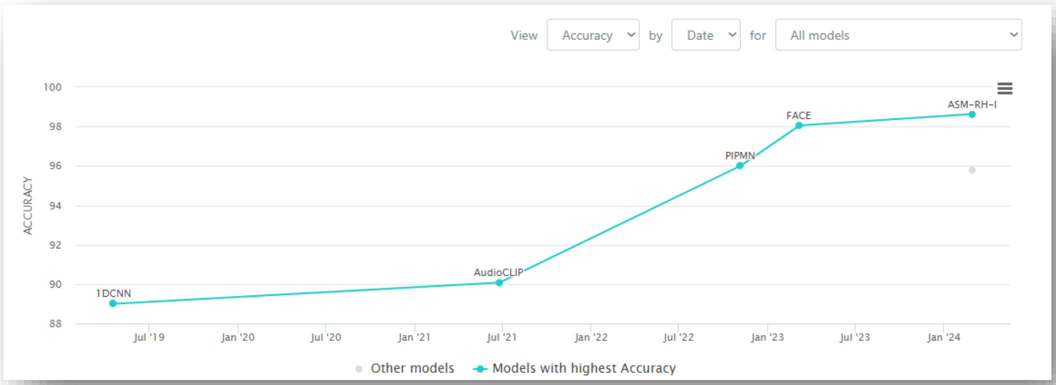
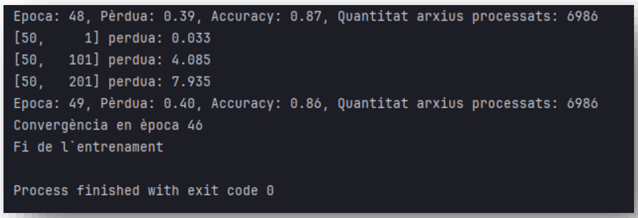


Figure 18 – SOTA results for the UrbanSound8k dataset as of May 2024.

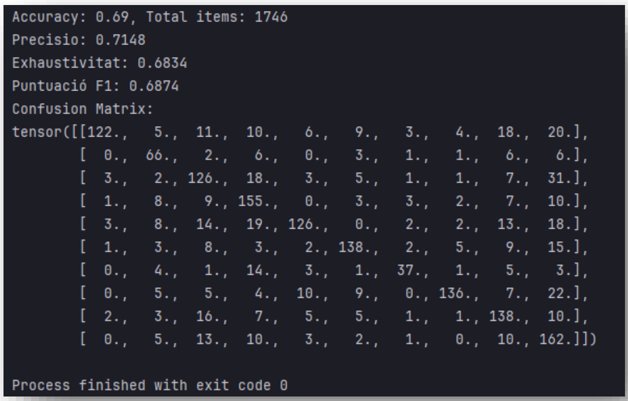
One of the first things we can do is that the number of epochs is insufficient. At this point we decided to increase the epochs, and after several tests, we ended up doing the loop explained above, with a maximum of 100 epochs (which we know we will never reach) and a patience parameter of 3. We start the training again and although it now takes us a considerable amount of time (more than two hours for UrbanSound, considering that the training set is almost 7000 audio samples), the results improve considerably:



```
Epoca: 48, Pèrdua: 0.39, Accuracy: 0.87, Quantitat arxius processats: 6986
[50,     1] perdua: 0.033
[50,   101] perdua: 4.085
[50,   201] perdua: 7.935
Epoca: 49, Pèrdua: 0.40, Accuracy: 0.86, Quantitat arxius processats: 6986
Convergència en època 46
Fi de l`entrenament

Process finished with exit code 0
```

Figure 19 - Training result with max_epochs = 100 and increasing batch size.

We launch the run class again to see the model inference, very similar to the previous one, but with the metrics slightly lower:



```
Accuracy: 0.69, Total items: 1746
Precisio: 0.7148
Exhaustivitat: 0.6834
Puntuació F1: 0.6874
Confusion Matrix:
tensor([[122.,   5.,  11.,  10.,   6.,   9.,   3.,   4.,  18.,  20.],
        [  0.,  66.,   2.,   6.,   0.,   3.,   1.,   1.,   6.,   6.],
        [  3.,   2., 126.,  18.,   3.,   5.,   1.,   1.,   7.,  31.],
        [  1.,   8.,   9., 155.,   0.,   3.,   3.,   2.,   7.,  10.],
        [  3.,   8.,  14.,  19., 126.,   0.,   2.,   2.,  13.,  18.],
        [  1.,   3.,   8.,   3.,   2., 138.,   2.,   5.,   9.,  15.],
        [  0.,   4.,   1.,  14.,   3.,   1.,  37.,   1.,   5.,   3.],
        [  0.,   5.,   5.,   4.,  10.,   9.,   0., 136.,   7.,  22.],
        [  2.,   3.,  16.,   7.,   5.,   5.,   1.,   1., 138.,  10.],
        [  0.,   5.,  13.,  10.,   3.,   2.,   1.,   0.,  10., 162.]])

Process finished with exit code 0
```

Figure 20 - Result of metrics max_epochs = 100 and increasing batch size.

The accuracy has increased slightly (so the model is more accurate when it gets the class right), but the rest of the figures have not improved. In general, these are not bad figures, although they are lower than the training accuracy, probably due to a point of overfitting of the model, caused by excess training (either due to the number of epochs, or the parameters of the optimizer or the scheduler, etc.).
Another detail that we can appreciate in these first two experiments (and that will be repeated throughout the various tests) is that the model does not have the same precision in all classes:

CLASS	ITEMS	CORRECT	ACCURACY
0 = air conditioning	208	122	***0.58***
1 = car horn	91	66	***0.72***
2 = children playing	197	126	***0.64***
3 = barking dog	198	155	***0.78***
4 = drill	205	126	***0.61***
5 = car engine	186	138	***0.74***
6 = pistol shot	69	37	***0.53***
7 = hydraulic hammer	198	136	***0.68***
8 = mermaid	188	138	***0.73***
9 = music in the street	206	162	***0.78***

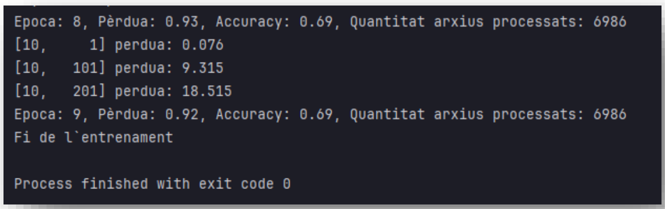
Table 1.*Results of the evaluation metrics of the first phase of experiments.*

Specifically, for this last exercise, classes 7 (hydraulic hammer) and 0 (air conditioning machine) had only 53% and 58% accuracy respectively. This also indicates the weaknesses of the model, which has a harder time recognizing more monotonous and repetitive sounds like these two. On the other hand, it shows the best performance (78% accuracy) in classes 3 (dogs barking) and 9 (music in the street), which are the classes richest in timbre and from which we have the greatest variety of fragments.

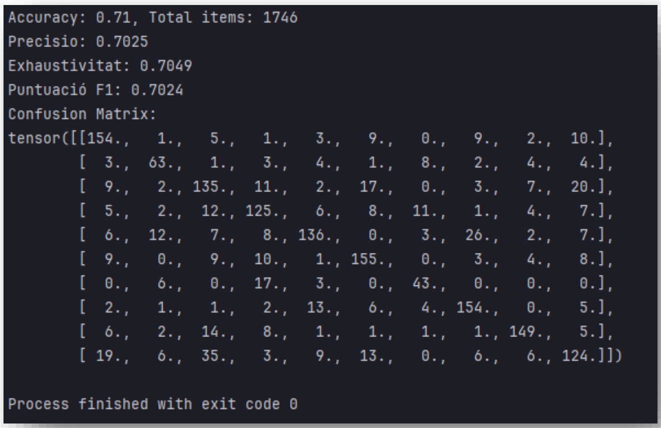
Refinement

Based on the above results and everything we have learned about sound processing theory, we decided to adjust some parameters of our model.

The first thing we can try is to modify the analysis window; we know that a larger number will give us more information about the number of frequencies present in the spectrogram, but it will blur the time stamps; on the other hand, by reducing the window, we can even lose frequencies, but we will have better defined time stamps. To clearly see the difference, we will try to modify the number of mels and the size of the window, proportionally, to 32, 64 and 1024 mels and windows of 512, 1024 and 2048:

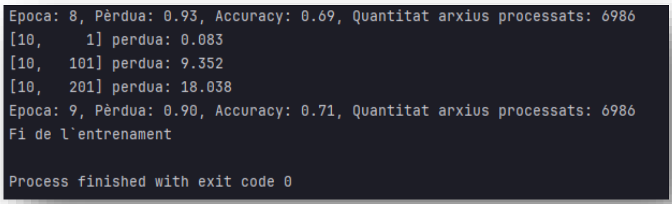


```
Epoca: 8, Pèrdua: 0.93, Accuracy: 0.69, Quantitat arxius processats: 6986
[10,     1] perdua: 0.076
[10,   101] perdua: 9.315
[10,   201] perdua: 18.515
Epoca: 9, Pèrdua: 0.92, Accuracy: 0.69, Quantitat arxius processats: 6986
Fi de l`entrenament

Process finished with exit code 0
```

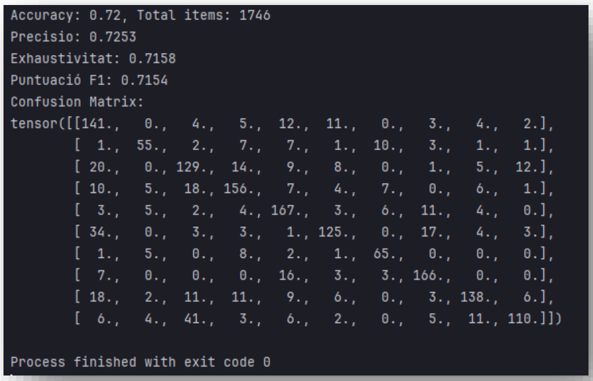


```
Accuracy: 0.71, Total items: 1746
Precisio: 0.7025
Exhaustivitat: 0.7049
Puntuació F1: 0.7024
Confusion Matrix:
tensor([[154.,   1.,   5.,   1.,   3.,   9.,   0.,   9.,   2.,  10.],
        [  3.,  63.,   1.,   3.,   4.,   1.,   8.,   2.,   4.,   4.],
        [  9.,   2., 135.,  11.,   2.,  17.,   0.,   3.,   7.,  20.],
        [  5.,   2.,  12., 125.,   6.,   8.,  11.,   1.,   4.,   7.],
        [  6.,  12.,   7.,   8., 136.,   0.,   3.,  26.,   2.,   7.],
        [  9.,   0.,   9.,  10.,   1., 155.,   0.,   3.,   4.,   8.],
        [  0.,   6.,   0.,  17.,   3.,   0.,  43.,   0.,   0.,   0.],
        [  2.,   1.,   1.,   2.,  13.,   6.,   4., 154.,   0.,   5.],
        [  6.,   2.,  14.,   8.,   1.,   1.,   1.,   1., 149.,   5.],
        [ 19.,   6.,  35.,   3.,   9.,  13.,   0.,   6.,   6., 124.]])

Process finished with exit code 0
```

Figure 21 – Training results and metrics with 128 mels and 2048 window.

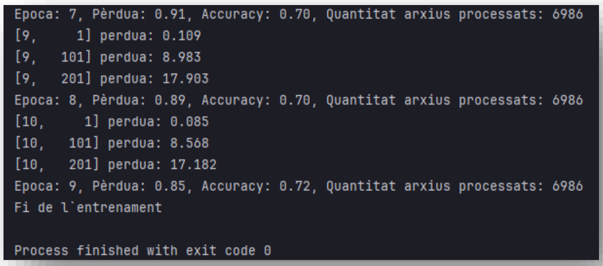


```
Epoca: 8, Pèrdua: 0.93, Accuracy: 0.69, Quantitat arxius processats: 6986
[10,     1] perdua: 0.083
[10,   101] perdua: 9.352
[10,   201] perdua: 18.038
Epoca: 9, Pèrdua: 0.90, Accuracy: 0.71, Quantitat arxius processats: 6986
Fi de l`entrenament

Process finished with exit code 0
```

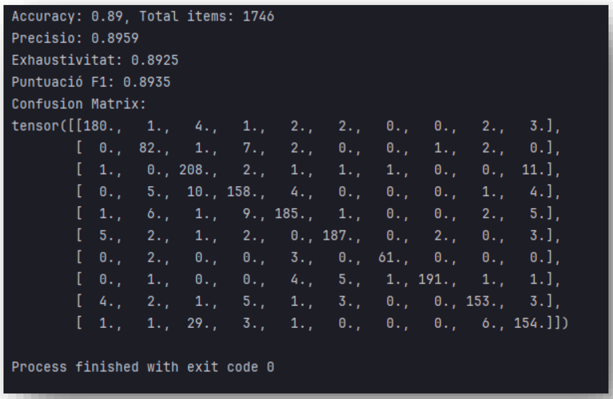


```
Accuracy: 0.72, Total items: 1746
Precisio: 0.7253
Exhaustivitat: 0.7158
Puntuació F1: 0.7154
Confusion Matrix:
tensor([[141.,   0.,   4.,   5.,  12.,  11.,   0.,   3.,   4.,   2.],
        [  1.,  55.,   2.,   7.,   7.,   1.,  10.,   3.,   1.,   1.],
        [ 20.,   0., 129.,  14.,   9.,   8.,   0.,   1.,   5.,  12.],
        [ 10.,   5.,  18., 156.,   7.,   4.,   7.,   0.,   6.,   1.],
        [  3.,   5.,   2.,   4., 167.,   3.,   6.,  11.,   4.,   0.],
        [ 34.,   0.,   3.,   3.,   1., 125.,   0.,  17.,   4.,   3.],
        [  1.,   5.,   0.,   8.,   2.,   1.,  65.,   0.,   0.,   0.],
        [  7.,   0.,   0.,   0.,  16.,   3.,   3., 166.,   0.,   0.],
        [ 18.,   2.,  11.,  11.,   9.,   6.,   0.,   3., 138.,   6.],
        [  6.,   4.,  41.,   3.,   6.,   2.,   0.,   5.,  11., 110.]])

Process finished with exit code 0
```

Figure 22 – Training results and metrics with 32 mels and 512 window.



```
Epoca: 7, Pèrdua: 0.91, Accuracy: 0.70, Quantitat arxius processats: 6986
[9,     1] perdua: 0.109
[9,   101] perdua: 8.983
[9,   201] perdua: 17.903
Epoca: 8, Pèrdua: 0.89, Accuracy: 0.70, Quantitat arxius processats: 6986
[10,     1] perdua: 0.085
[10,   101] perdua: 8.568
[10,   201] perdua: 17.182
Epoca: 9, Pèrdua: 0.85, Accuracy: 0.72, Quantitat arxius processats: 6986
Fi de l`entrenament

Process finished with exit code 0
```

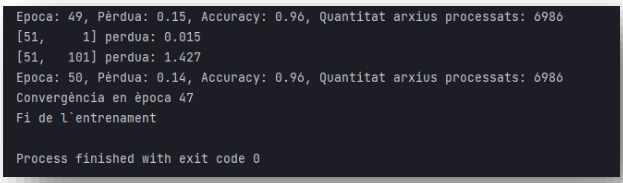


```
Accuracy: 0.89, Total items: 1746
Precisio: 0.8959
Exhaustivitat: 0.8925
Puntuació F1: 0.8935
Confusion Matrix:
tensor([[180.,   1.,   4.,   1.,   2.,   2.,   0.,   0.,   2.,   3.],
        [  0.,  82.,   1.,   7.,   2.,   0.,   0.,   1.,   2.,   0.],
        [  1.,   0., 208.,   2.,   1.,   1.,   1.,   0.,   0.,  11.],
        [  0.,   5.,  10., 158.,   4.,   0.,   0.,   0.,   1.,   4.],
        [  1.,   6.,   1.,   9., 185.,   1.,   0.,   0.,   2.,   5.],
        [  5.,   2.,   1.,   2.,   0., 187.,   0.,   2.,   0.,   3.],
        [  0.,   2.,   0.,   0.,   3.,   0.,  61.,   0.,   0.,   0.],
        [  0.,   1.,   0.,   0.,   4.,   5.,   1., 191.,   1.,   1.],
        [  4.,   2.,   1.,   5.,   1.,   3.,   0.,   0., 153.,   3.],
        [  1.,   1.,  29.,   3.,   1.,   0.,   0.,   0.,   6., 154.]])

Process finished with exit code 0
```

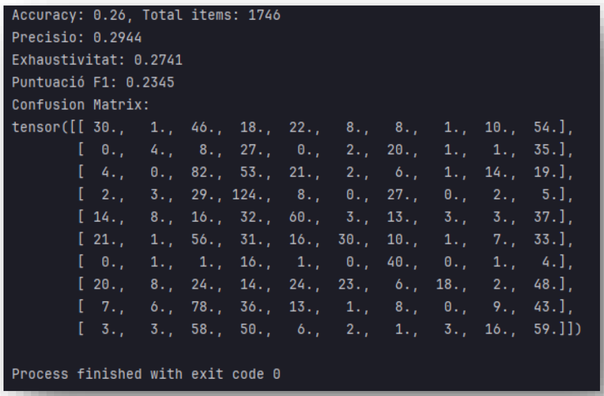
Figure 23 – Training results and metrics with 64 mels and 1024 window.

We observe that the training results are not very different from each other (since the intrinsic parameters of the model have not changed), but the inference results are: we see how increasing the mels or the window size (it has also been tested separately) worsens the model results, while reducing them improves them. The optimal results in this case are found in 64 mels and a window of 1024; this is probably related to what we have already explained in the theory section: a smaller window gives us a better image of the temporal attacks of the frequencies, while a larger one gives us greater precision in the number of frequencies. However, in this game the sounds are quite repetitive and

are not very diverse timbrally. In many cases even (dog burglar, siren, etc.) the sound fragment is a periodic repetition of the same frequencies, so what best defines the sound is the temporal position of the frequencies in the spectrogram, rather than the frequencies themselves.
Now we want to put the data augmentation process to use to improve the model's generalization capacity. We have performed a test training the model with different optimizers and windows, with and without the SpecAugment process; in many cases, without SpecAugment the training accuracy improves qualitatively, but the inference plummets:



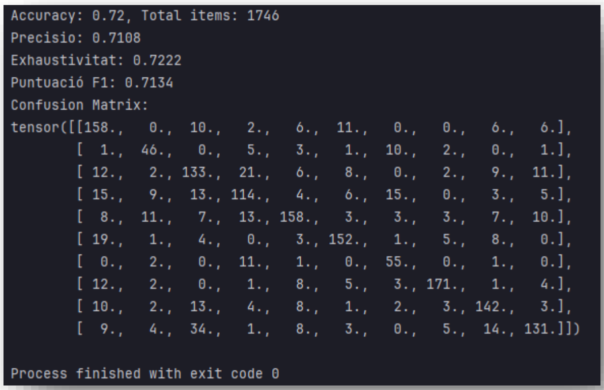
```
Epoca: 49, Pèrdua: 0.15, Accuracy: 0.96, Quantitat arxius processats: 6986
[51,     1] perdua: 0.015
[51,   101] perdua: 1.427
Epoca: 50, Pèrdua: 0.14, Accuracy: 0.96, Quantitat arxius processats: 6986
Convergència en època 47
Fi de l`entrenament

Process finished with exit code 0
```



```
Accuracy: 0.26, Total items: 1746
Precisio: 0.2944
Exhaustivitat: 0.2741
Puntuació F1: 0.2345
Confusion Matrix:
tensor([[ 30.,   1.,  46.,  18.,  22.,   8.,   8.,   1.,  10.,  54.],
        [  0.,   4.,   8.,  27.,   0.,   2.,  20.,   1.,   1.,  35.],
        [  4.,   0.,  82.,  53.,  21.,   2.,   6.,   1.,  14.,  19.],
        [  2.,   3.,  29., 124.,   8.,   0.,  27.,   0.,   2.,   5.],
        [ 14.,   8.,  16.,  32.,  60.,   3.,  13.,   3.,   3.,  37.],
        [ 21.,   1.,  56.,  31.,  16.,  30.,  10.,   1.,   7.,  33.],
        [  0.,   1.,   1.,  16.,   1.,   0.,  40.,   0.,   1.,   4.],
        [ 20.,   8.,  24.,  14.,  24.,  23.,   6.,  18.,   2.,  48.],
        [  7.,   6.,  78.,  36.,  13.,   1.,   8.,   0.,   9.,  43.],
        [  3.,   3.,  58.,  50.,   6.,   2.,   1.,   3.,  16.,  59.]])

Process finished with exit code 0
```

Figure 24 – Training results and metrics with Adam optimizer, 32 mels and 512 window without SpecAugment.

An accuracy of 96% (an unrealistic figure, by the way) indicates a clear overfitting of the model to the training data, which makes it difficult to make correct predictions later; the same parameters, with the SpecAugment method, reduce the training accuracy to 89%, but give us an accuracy of 71% in the predictions, highlighting the importance of the technique in order to improve generalization:

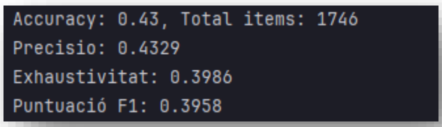


```
Accuracy: 0.72, Total items: 1746
Precisio: 0.7108
Exhaustivitat: 0.7222
Puntuació F1: 0.7134
Confusion Matrix:
tensor([[158.,   0.,  10.,   2.,   6.,  11.,   0.,   0.,   6.,   6.],
        [  1.,  46.,   0.,   5.,   3.,   1.,  10.,   2.,   0.,   1.],
        [ 12.,   2., 133.,  21.,   6.,   8.,   0.,   2.,   9.,  11.],
        [ 15.,   9.,  13., 114.,   4.,   6.,  15.,   0.,   3.,   5.],
        [  8.,  11.,   7.,  13., 158.,   3.,   3.,   3.,   7.,  10.],
        [ 19.,   1.,   4.,   0.,   3., 152.,   1.,   5.,   8.,   0.],
        [  0.,   2.,   0.,  11.,   1.,   0.,  55.,   0.,   1.,   0.],
        [ 12.,   2.,   0.,   1.,   8.,   5.,   3., 171.,   1.,   4.],
        [ 10.,   2.,  13.,   4.,   8.,   1.,   2.,   3., 142.,   3.],
        [  9.,   4.,  34.,   1.,   8.,   3.,   0.,   5.,  14., 131.]])

Process finished with exit code 0
```

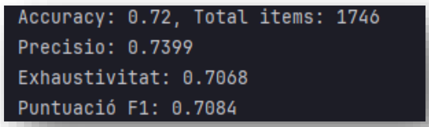
Figure 25 – Training results and metrics with Adam optimizer, 32 mels and 512 window with SpecAugment.

The optimizers

Finally, another of the parameters analyzed is the optimizer. The original training loop was defined with an Adam optimizer, quite popular in this type of audio solutions (in fact it is the one used by the CLAP model and also by Wav2Clip).[23] , one of the first models to implement the Contrastive-Language solution), but I have decided to try other solutions as well; specifically, I have tried starting the loop with the AdamW optimizer, the RMSProp, reducing the number of epochs to 20 so as not to excessively lengthen the test time and to be able to see how the results varied; finally, I have also opted for a SGD (Stochastic Gradient Descent), used in the model developed by the authors of the Transformers for Audio Classification research.[24], from which much of the ideas for processing and using the spectrograms in this work have been taken.



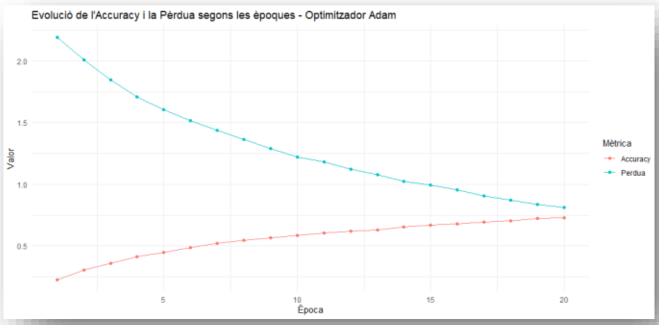
```
Accuracy: 0.43, Total items: 1746
Precisio: 0.4329
Exhaustivitat: 0.3986
Puntuació F1: 0.3958
```

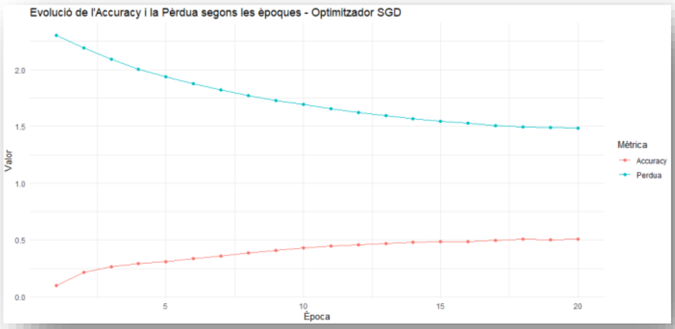


```
Accuracy: 0.72, Total items: 1746
Precisio: 0.7399
Exhaustivitat: 0.7068
Puntuació F1: 0.7084
```

Figure 26 – Result of the metrics with RMSprop optimizer on the right and SGD on the left. 64 mels and window of 1024.

Looking at the results of the metrics and the training, we realize the big difference that using one optimizer or another makes; but to see it more graphically, we have done several training tests with 20 epochs (to speed up the process) and see more clearly how the learning evolves according to each one:





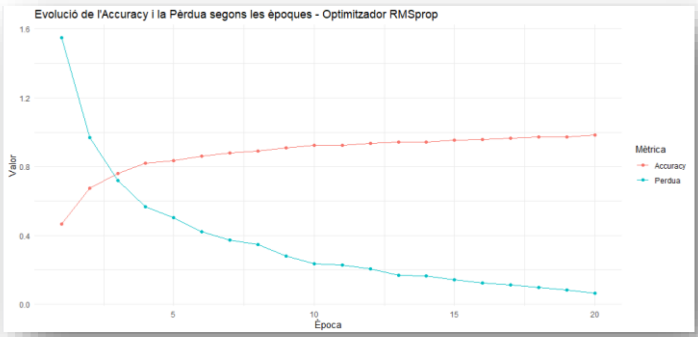


Figure 27 – Learning evolution according to Adam, SGD and RMSprop optimizers.

We see how with the Stochastic Gradient the accuracy increases more slowly and the final inference results are slightly lower than those of the RMSprop, and in both cases, lower than the best case of Adam. However, the results of the RMSprop are very surprising, as it increases the accuracy very quickly, but, above all, it decreases the results of the loss function in very few epochs. It is an interesting step, which would allow us to train the models more quickly, but we have already seen that the results are lower than those of Adam, which, despite going slower, ends up being more accurate. This is probably because the RMS adjusts very quickly to the training data since many are relatively similar. To try to compensate for this and find the perfect balance, we reduce the learning rate of the RMSprop to 0.001, so that it reduces the update steps and generalizes better. We run the training and see the resulting graph:

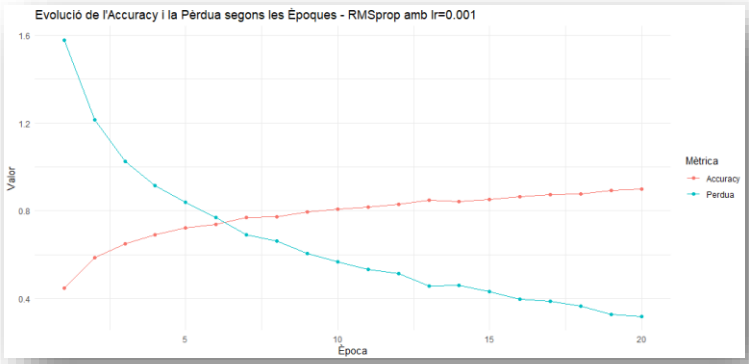
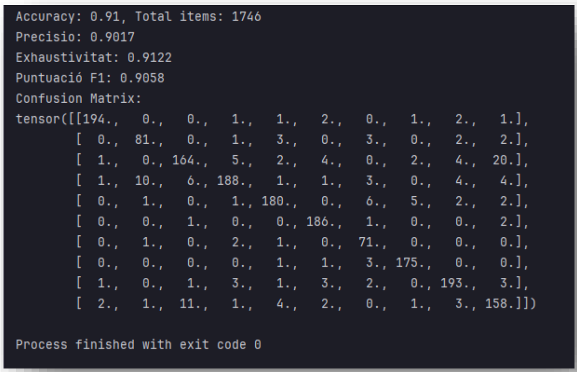


Figure 28 - Evolution of RMSprop optimizer learning with learning rate 0.001.

In this case, the curve is smoother, but learning is still much faster than in the other two optimizers.
At this point, we can deduce that the speed of RMSprop is due to its own nature: this optimizer, acronym for Root Mean Square Propagation, helps to update the weights dynamically, calculating at each step the square mean of the gradients and thus adapting the learning rate at each iteration. We can also theorize that the gradients do not change excessively (at least for this type of data, the nature of the sound can change if we change the training set, logically) and therefore this would explain the efficiency of RMSprop in this specific case. SGD, on the other hand, is penalized, since it has a fixed learning rate in all iterations. Adam uses a combination of the two techniques. It should be noted that, in each modification of the optimizers, in addition to the learning rate, the planner has also been adjusted using the OneCycle or Standard Reduce on Plateau types. OneCycleLR is the one that has given the best results in all cases.

Final evaluation

After several more tests, we see that we can combine a lower window (512) and 32 mels with this optimizer; we train the model and see the results:



```
Accuracy: 0.91, Total items: 1746
Precisio: 0.9017
Exhaustivitat: 0.9122
Puntuació F1: 0.9058
Confusion Matrix:
tensor([[194.,   0.,   0.,   1.,   1.,   2.,   0.,   1.,   2.,   1.],
        [  0.,  81.,   0.,   1.,   3.,   0.,   3.,   0.,   2.,   2.],
        [  1.,   0., 164.,   5.,   2.,   4.,   0.,   2.,   4.,  20.],
        [  1.,  10.,   6., 188.,   1.,   1.,   3.,   0.,   4.,   4.],
        [  0.,   1.,   0.,   1., 180.,   0.,   6.,   5.,   2.,   2.],
        [  0.,   0.,   1.,   0.,   0., 186.,   1.,   0.,   0.,   2.],
        [  0.,   1.,   0.,   2.,   1.,   0.,  71.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   1.,   1.,   3., 175.,   0.,   0.],
        [  1.,   0.,   1.,   3.,   1.,   3.,   2.,   0., 193.,   3.],
        [  2.,   1.,  11.,   1.,   4.,   2.,   0.,   1.,   3., 158.]])

Process finished with exit code 0
```

Figure 29 - Metrics result of the model with the best optimization.

91% accuracy, very close to the current LOW results, and with good figures also in completeness and precision; this indicates that the model is much more precise when it comes to correctly classifying classes. Likewise, we can see that the prediction differences between classes have disappeared and now we have an almost normal distribution of accuracy in all of them thanks to the adaptations made to the model.

As a summary, we can better analyze the evolution of the experiments in the following table:

EXPERIMENT (weights file)	Optimize.	LR	MELS No.	Window	Accuracy
base model (10 epochs)	Adam	0.001	64	1024	***71%***
audio_classifier_weights_64m_1024w_Adam	Adam	0.001	64	1024	***69%***
audio_classifier_weights_128m_2048w_Adam_SpecAug	Adam	0.001	128	2048	***71%***
audio_classifier_weights_32m_512w_Adam	Adam	0.001	32	512	***26%***
audio_classifier_weights_32m_512w_Adam_SpecAug	Adam	0.001	32	512	***72%***
audio_classifier_weights_64m_1024w_AdamOK	Adam	0.001	64	1024	***89%***
audio_classifier_weights_64m_1024w_SGD	SGD	0.01	64	1024	***43%***
audio_classifier_weights_64m_1024w_RMSprop	RMSprop	0.01	64	1024	***72%***
audio_classifier_weights_32m_512w_RMSprop_0001	RMSprop	0.001	128	2048	***91%***

Table 2.*Results of the evaluation metrics of the second phase of experiments (fine-tuning).*

3.5. Other models

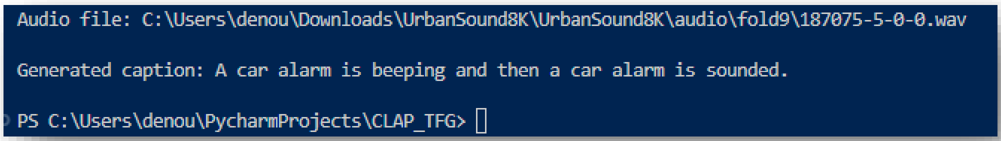
To complete the work, we have carried out a series of tests with different data sets and models. Although one of the objectives of the work was precisely to develop this second part, it has not been possible due to the time available and the complexity of the tasks. However, we do not want to miss the opportunity to mention the different possibilities that exist in this field.

First, we have analyzed the model that inspired the work in the first place, the Microsoft CLAP model. This is a much more complete project than the one presented here but it is based on the same theoretical basis. The structure of the project allows it to take several sets of data as input, some of which (such as ESC50 or AudioSet) are already predefined and trained with the corresponding weights, which can be downloaded from an online repository directly from the project itself. CLAP implements a system of six convolutional blocks, which receive only 1 input channel and end up generating 2048, and also incorporates a module that implements the concept of attention.[25], a technique that allows the model to separate the data input sequence into different parts, in order to focus on those parts that are most relevant to the task at hand.

The audio encoder, although more sophisticated, is essentially similar to the one seen here, performing much of the fragment preprocessing and feature extraction tasks;

however, it is the text encoder that really makes the difference. At the time of its publication in 2020, CLAP used a BERT-based encoder to process textual descriptions. BERT is a popular large-scale language model (LLM) trained with over 110 million parameters, which allows for the creation of a very complete embedding matrix with which the model manages to create text prompts very similar to natural language when making predictions. However, in the latest updates, CLAP already incorporates the possibility of choosing OpenAI's GPT2 version as a text encoder, offering even better results than the original BERT.

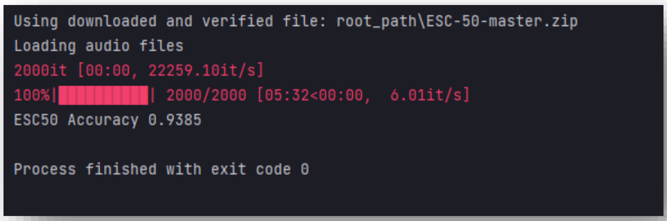
To see this, we tried to launch an UrbanSound file, which is defined in the metadata as engine idling, even though there are more sounds to be heard (it's a car with the engine running, in which an open door alarm is sounding, which suddenly stops). The CLAP response, on the other hand, is much more complete and able to distinguish and verbalize the entire scene, although it is not entirely precise:



```
Audio file: C:\Users\denou\Downloads\UrbanSound8K\UrbanSound8K\audio\fold9\187075-5-0-0.wav

Generated caption: A car alarm is beeping and then a car alarm is sounded.

PS C:\Users\denou\PycharmProjects\CLAP_TFG> 
```

Figure 30 - CLAP model output prompt, with GPT2 encoder, for one of the sounds in the UrbanSound8K set.

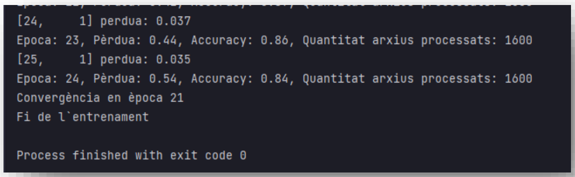
The project available in the official GitHub repository already contains some examples with some pre-assigned data sets. One of them is ESC50[13], which was considered as a possible set for this work and was ultimately discarded. To make the comparison we have launched the same set in both models, the CLAP and ours.



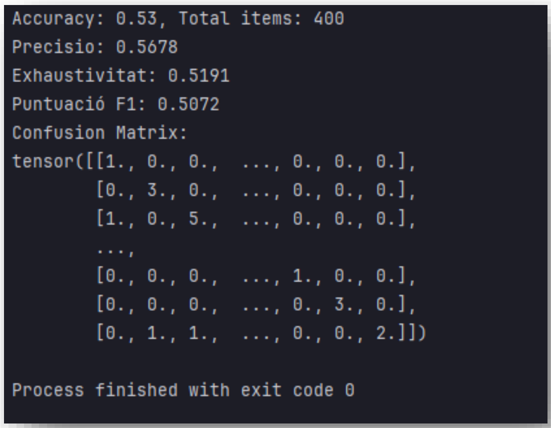
```
Using downloaded and verified file: root_path\ESC-50-master.zip
Loading audio files
2000it [00:00, 22259.10it/s]
100%|██████████| 2000/2000 [05:32<00:00,  6.01it/s]
ESC50 Accuracy 0.9385

Process finished with exit code 0
```

Figure 31 – Accuracy result of the CLAP model with the ESC-50 set and the model with pre-trained weights.

It can be seen that we have a level of almost 94% accuracy with CLAP, not bad, considering that it is an ensemble with many more classes than Urban Sound (50 classes versus 10). To perform the test we have made an adaptation of the dataLoader and TrainingData classes of our model so that they take the directory structure of ESC50 and the class identifier from the ensemble metadata file. In this way we can run the same ensemble with our model; we will use the parameters that have given us the best results with UrbanSound: 32 mels, window from 512 SamplingRate to 44100, and SpecAgment; however, we will change the original Adam optimizer for RMSprop, with a learning rate of 0.001. This is the result:



```
[24,     1] perdua: 0.037
Epoca: 23, Pèrdua: 0.44, Accuracy: 0.86, Quantitat arxius processats: 1600
[25,     1] perdua: 0.035
Epoca: 24, Pèrdua: 0.54, Accuracy: 0.84, Quantitat arxius processats: 1600
Convergència en època 21
Fi de l`entrenament

Process finished with exit code 0
```



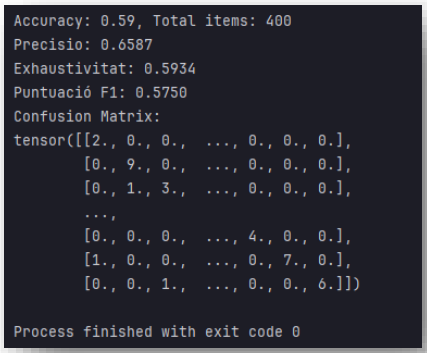
```
Accuracy: 0.53, Total items: 400
Precisio: 0.5678
Exhaustivitat: 0.5191
Puntuació F1: 0.5072
Confusion Matrix:
tensor([[1., 0., 0.,  ..., 0., 0., 0.],
        [0., 3., 0.,  ..., 0., 0., 0.],
        [1., 0., 5.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 1., 0., 0.],
        [0., 0., 0.,  ..., 0., 3., 0.],
        [0., 1., 1.,  ..., 0., 0., 2.]])

Process finished with exit code 0
```

Figure 32 – Training result and metrics with RMSprop optimizer, 32 mels and 512 window for the ESC-50 ensemble.

We see very high accuracy in training, but it drops substantially when we do inference. To try to improve these results we will make some small adjustments:

- If we analyze the audio fragments, we see that in the ESC50 all the files last exactly 5 seconds; we therefore increase this figure in the data preparation process.
- We have much less data, 2000 versus 8000, so we can make the batches smaller without excessively harming the training time. We will also take advantage of this

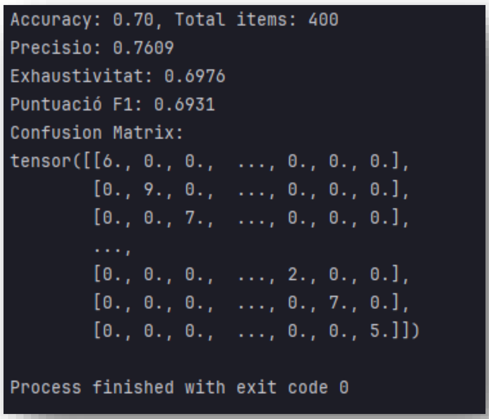
to further reduce the learning rate of the optimizer, which with smaller batches will allow us to improve the training dynamics.



```
Accuracy: 0.59, Total items: 400
Precisio: 0.6587
Exhaustivitat: 0.5934
Puntuació F1: 0.5750
Confusion Matrix:
tensor([[2., 0., 0.,  ..., 0., 0., 0.],
        [0., 9., 0.,  ..., 0., 0., 0.],
        [0., 1., 3.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 4., 0., 0.],
        [1., 0., 0.,  ..., 0., 7., 0.],
        [0., 0., 1.,  ..., 0., 0., 6.]])

Process finished with exit code 0
```

Figure 33 – Result of the metrics with RMSprop optimizer, 64 mels and 1024 window for the ESC-50 set with various modifications.

We increased the numbers slightly, but not by too much. We think that the sounds in this set are generally more timbrally rich than the UrbanSound set; there are 40 other classes, and the classes are much more varied: animal sounds, various vehicles, mixed sounds, etc. As we saw before, a small window could penalize us, so we decided to do some more tests and finally decided to increase the number of mels to 128 and the window to 2048, keeping the rest of the parameters the same when making the spectrogram. The result:



```
Accuracy: 0.70, Total items: 400
Precisio: 0.7609
Exhaustivitat: 0.6976
Puntuació F1: 0.6931
Confusion Matrix:
tensor([[6., 0., 0.,  ..., 0., 0., 0.],
        [0., 9., 0.,  ..., 0., 0., 0.],
        [0., 0., 7.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 2., 0., 0.],
        [0., 0., 0.,  ..., 0., 7., 0.],
        [0., 0., 0.,  ..., 0., 0., 5.]])

Process finished with exit code 0
```

Figure 34 – Result of the metrics with RMSprop optimizer, 128 mels and 2048 window for the ESC-50 set with various modifications.

So, it seems that this confirms our theory, even though we are still far from the CLAP metrics; here we must take into account that the CLAP is pre-trained with thousands of data of all kinds, while we only have the data of the set itself.

We summarize the results of the experiments with the ESC-50 in the following table:

EXPERIMENT (weights file)	**Optimize.**	**LR**	**MELS No.**	**Window**	**Accuracy**
audio_classifier_weights_ESC50_32m_512w_4s	RMSprop	0.001	32	512	***53%***
audio_classifier_weights_ESC50	RMSprop	0.001	64	1024	***59%***
audio_classifier_weights_ESC50_64m_1024w	Adam	0.001	64	1024	***62%***
audio_classifier_weights_ESC50_128m_2048w	Adam	0.001	128	2048	***68%***
audio_classifier_weights_ESC50_128m_2048w_SpecAugmentOK	RMSprop	0.001	128	2048	***70%***

Table 3.*Results of the evaluation metrics of the third phase of experiments (ESC-50).*

4. Conclusions

In this work we have analyzed in detail the general structure of a deep learning model designed specifically for audio recognition tasks, specifically for sounds from the urban environment. To do so, we have seen the vicissitudes of digital audio and its own mathematical nature in order to better understand how these types of models work. We have reviewed some of the main techniques used in the recognition of sounds from the environment and found that some of the most effective are those that follow the contrastive language model. We have also been able to create a complete classification model project, with the training and inference phase, and we have seen the importance of correctly parameterizing the model to obtain good prediction results.

4.1. Conclusions

Although there are a large number of solutions currently available for audio classification, architectures based on deep learning systems, such as neural networks, have proven to be the most effective in this field. There has been a lot of progress in recent years, especially in terms of voice recognition, and CNNs have played a very important role in this; their ability to process images with great precision allows them to work directly, as we have seen, with spectrograms derived from specific sound fragments, extracting their main characteristics to subsequently classify them correctly.
It seems that the effectiveness of this type of solution for audio classification is demonstrated. It should be remembered that the sets that have been seen throughout the work are some of the most complete at an international level, and as has been seen, good inference figures can be obtained with them and with the methods seen. Other approaches also discussed (Transformers, end-to-end techniques, RNN, etc.) could also yield good results, but the computational cost of carrying out experiments with them must be considered. Thus, CNNs can be great allies in the classification and recognition of audio for short sound pieces (between 2 and 10 seconds), while for longer pieces, such as musical compositions or complex human conversations, it might be better to use another approach based more on parallel computing techniques.
We have found several problems throughout the work, but perhaps the most important is the overfitting of the models. Containing sound fragments that are very similar to each other makes it difficult for the model to learn beyond what it sees in training (generalization), especially in the field of environmental sounds as we have worked. This type of model has demonstrated great effectiveness in voice recognition precisely for this reason: given phonemes or vocabulary of a given language, training by words or syllables is highly accurate.[26], because the generalization is limited to a finite set of sounds. However, in the case we have seen, the data has very good precision figures when we limit ourselves to the sounds of the set itself, but much lower when we try to pass different sounds, with different treatments, captured with other techniques (microphones, ambient sounds, background noise, etc.). As in other cases with AI, one

of the keys is the amount of training data; the more data can be used to train the model, the greater its generalization capacity will be, but it is not the only one: it is also important to define the labels well, especially in the case of contrastive language, since the weights of the texts that describe the sounds allow us to play with their variability, facilitating their interpretation by the model at the time of making the inference.

4.2. Further development

Despite the great advances in this field, I think there is still work to be done. It would probably be interesting to continue developing solutions such as CLAP or similar, which allow the incorporation of the text weight embedding system of current LLMs. Many of the works consulted are from before 2022, the year of the great explosion of ChatGPT and which was the advance of the large number of LLMs available today, many of which are also open source and allow great customization, either by parameterizing the model code itself or through Retrieval-augmented Generation (RAG). All these technologies could be incorporated into types of architectures such as those we have seen so that the final result would be an exact description of what the model is listening to at all times, thus allowing natural interaction with AI by taking advantage of all the resources offered by the language or the environment.

4.3. On a personal level

The development of this work has been a great challenge and at the same time a learning experience. I have been able to see first-hand the importance of documentation, research and deepening knowledge to fully understand all aspects of AI and deep learning techniques. I have also learned that time planning and organization are key. But I think the most critical point has been computing resources; although I have been able to carry out the experiments in a more or less solvent manner with my gear, other experiments were beyond my reach, either due to the depth of the training networks or the amount of data required.

I hope that this work will serve to spark the interest of all those who read it, and, who knows, perhaps to encourage them to embark on the path into the exciting world of sound and AI.

5. Glossary

Amplitude: Maximum size of a wave measured from its resting position to its maximum peak.

API (Application Programming Interface): Set of rules that allow different software programs to communicate with each other.

Deep Learning: Machine learning discipline that uses deep artificial neural networks to model and understand complex data.

CNN (Convolutional Neural Network): Type of neural network that is especially effective in image processing and visual pattern recognition tasks.

Open Source: Software with source code accessible to the public, allowing its modification and distribution by anyone.

Dataset: Structured collection of data used to train and evaluate machine learning models.

Decibel: Unit of measurement of sound intensity, expressed on a logarithmic scale.

DFT (Discrete Fourier Transform): Algorithm that transforms a sequence of values into a sum of sinusoidal components of different frequencies.

Spectrum: Representation of the frequency components of a signal.

Spectrogram: Visual representation of the frequency spectrum of a signal as a function of time.

Wave phase (phase): Relative position of a point within a wave cycle, measured in degrees or radians.

FFT (Fast Fourier Transform): Efficient algorithm for calculating the Discrete Fourier Transform (DFT).

Window (Analysis Window): Time segment in which a signal is analyzed during audio processing.

Frequency: Number of complete cycles of a wave that pass through a fixed point in one second, usually measured in Hertz (Hz).

AI (Artificial Intelligence): Field of computer science that focuses on the creation of systems capable of performing complex tasks requiring human intelligence.

IFT (Inverse Fourier Transform): Operation used to convert a frequency spectrum to its original form in the time domain.

Weights: Computational parameters at the nodes of a neural network that are tuned during training to minimize prediction error.

Sound pressure: Variation in air pressure caused by a sound wave, measured linearly in pascals (Pa).

Sound: Vibration that propagates as a mechanical wave through a medium such as air or water and is perceived by the human ear.

STFT (Short-Time Fourier Transform): Variation of the Fourier Transform applied to short time segments of the signal.

Sampling Rate: Number of samples per second taken from a continuous signal to create a discrete signal, measured in Hertz (Hz).

Neural Network: Computing system composed of interconnected units (neurons) that process information in a way similar to the human brain.

Internet of Things (IoT): A network of physical objects equipped with sensors, software and other technologies that allow data to be exchanged with other devices and systems over the Internet.

Overfitting: Problem in machine learning where a model learns the details and noise of the training data too well, causing it to degrade in generalization (dealing with new, unseen data).

Recovery Augmented Generation (RAG):AI technique that combines the retrieval of relevant documents from a database with the generation of text to provide answers based on them.

Large Scale Languages (LLM): AI models trained on large amounts of text capable of understanding and generating natural language as well as performing various tasks (text translation, answering questions, summaries, etc.)

6. Literature

[1] Sora. OpenAI. [online] [Accessed April 2024] Available at:
https://openai.com/index/sora

[2] Midjourney. [online] [Accessed April 2024] Available at:
https://www.midjourney.com/home

[3] CONTRIBUTORS TO WIKIMEDIA PROJECTS. Stable Diffusion - Wikipedia. Wikipedia, the free encyclopedia [online]. August 31, 2022 [accessed April 20, 2024]. Available at:
https://en.wikipedia.org/wiki/Stable_Diffusion

[4]Introducing a foundational multimodal model for speech translation. AI at Meta [online]. [undated] [accessed 18 April 2024]. Available at:
https://ai.meta.com/blog/seamless-m4t/

[5]GitHub - facebookresearch/demucs: Code for the paper Hybrid Spectrogram and Waveform Source Separation. GitHub [online]. [undated] [accessed 4 April 2024]. Available at:
https://github.com/facebookresearch/demucs

[6]MALIK, Shahid et al. In Acoustic 3D Positioning System for Robots Operating Underground. IEEE Sensors Letters [online]. 2022, 1–4 [accessed 25 April 2024]. ISSN 2475-1472. Available at:
https://ieeexplore.ieee.org/document/9893367

[7]. What is CRISP DM?. [online] [Accessed May 2024] Available at:https://www.datascience-pm.com/crisp-dm-2/

[8] CONTRIBUTORS TO WIKIMEDIA PROJECTS. Fourier analysis - Wikipedia. Wikipedia, the free encyclopedia [online]. December 20, 2001 [cited April 4, 2024]. Available in:
https://en.wikipedia.org/wiki/Fourier_analysis

[9]CONTRIBUTORS TO WIKIMEDIA PROJECTS. Absolute threshold of hearing - Wikipedia. Wikipedia, the free encyclopedia [online]. 16 August 2003 [accessed 20 April 2024]. Available at:
https://en.wikipedia.org/wiki/Absolute_threshold_of_hearing

[10]CONTRIBUTORS TO WIKIMEDIA PROJECTS. Window function - Wikipedia. Wikipedia, the free encyclopedia [online]. June 11, 2003 [accessed April 26, 2024]. Available at:
https://en.wikipedia.org/wiki/Window_function

[11] CONTRIBUTORS TO WIKIMEDIA PROJECTS. Mel scale - Wikipedia. Wikipedia, the free encyclopedia [online]. December 20, 2001 [accessed April 25, 2024]. Available at:
https://en.wikipedia.org/wiki/Mel_scale

[12]LIU Haohe, LIU Xubo, KONG Qiuqiang, WANG Wenwu, PLUMBLEY Mark D.. Learning Temporal Resolution in Spectrogram for Audio Classification [online]. October 4, 2022. [Accessed April 28, 2024]. Available at:
https://archiv.org/abs/2210.01719

[13] ESC-50 Dataset. [online] [Accessed April 2024] Available at:
https://paperswithcode.com/bajo/audio-classification-on-esc-50

[14] URBANSOUND8K DATASET. [online] [Accessed April 2024] Available at:

https://urbansounddataset.weebly.com/urbansound8k.html

[15] FSD50K Companion site. [online] [Accessed April 2024] Available at:
https://annotator.freesound.org/fsd/release/FSD50K/

[16] Domestic Environment Sound Event Detection Dataset. [online] [Accessed 4 May 2024] Available at:
https://project.inria.fr/desed/

[17]GitHub - Microsoft Scalable Noisy Speech Dataset (MS-SNSD). GitHub [online]. [2019] [accessed 28 May 2024]. Available at:
https://github.com/microsoft/MS-SNSD

[18] ELIZALDE Benjamin, DESHMUKH Soham, IN THE ISMAIL Mahmoud, WANG Huaming. Clap : learning audio concepts from natural language supervision. In: Microsoft [online]. 2022.[Accessed in June 2024]. Available at:
https://archiv.org/pdf/2206.04769.pdf

[19] HERSHEY Shawn, CHAUDHURI Sourish, PW ELLIS Daniel, GEMMEKE Jort F. CNN Architectures for Large-Scale Audio Classification. [online] 29 September 2016 [accessed 21 May 2024]. Available at:
https://archiv.org/abs/1609.09430

[20] SAINATH Tara N., MOHAMED Abdelrahman; KINGSBURY Brian; RAMABHADRAN Bhuvana. Deep convolutional neural networks for LVCSR [online] May 26, 2013 [accessed May 21, 2024]. Available at:
https://ieeexplore.ieee.org/document/6639347

[21] S.PARK Daniel, CHAN William. SpecAugment: In New Data Augmentation Method for Automatic Speech Recognition [online] April 22, 2019 [accessed May 2024]. Available at:
https://research.google/blog/specaugment-a-new-data-augmentation-method-for-automatic-speech-recognition/

[22] RADKOFF Evan. Loss Functions in Audio ML[online] 6 September 2021. [accessed June 2024]. Available at:
https://www.soundsandwords.io/audio-loss-functions/

[23] WU Ho-Hsiang, SEETHARAMAN Pulsa, KUMAR Kundan , BELLO Juan Pablo. WAV2CLIP: LEARNING ROBUST AUDIO REPRESENTATIONS FROM CLIP. [ICASSP Conference 2022] May 26, 2022. [accessed March 23, 2024]. Available at:
https://doi.org/10.48550/arXiv.2110.11499

[24] YIXIAO Zhang, LE Baihua, FANG Hui, MENG Qinggang. Spectrogram transformers for audio classification. [online] 2022. [accessed April 2024]. Available at:
https://doi.org/10.1109/IST55454.2022.9827729

[25] VASWANI Ashish, SHAZEER Noam, PARMAR Niki, USZKOREIT Jakob, JONES Llion, N. GOMEZ Aidan, KAISER Łukasz, POLOSUKHIN Illia. Attention is all you need. 2017. [accessed June 10, 2024]. Available in:
https://dl.acm.org/doi/10.5555/3295222.3295349

[26] GUPTA Rohan, KUMAR Rohan. Syllable Based Deep Learning for Multi-Dialect Speech Recognition. [online] 2019. [accessed June 12, 2024]. Available at:
https://www.ijedr.org/papers/IJEDR1904106.pdf

Annexes

Queries, software and model creation:

1. Papers With code website, with articles and models of various types:Audio Classification | Papers With Code
2. Online Hugging Face course, with a variety of pre-trained models and bases to build your own model:Audio rating (huggingface.co)
3. Coursera course on digital audio processing basics, by Xavier Serra:https://www.coursera.org/lecture/audio-signal-processing/teaser-53YTy
4. SMS-Tools, a package developed by UPF specifically for working with audio signals:https://www.upf.edu/web/mtg/sms-tools
5. Website of the Music Technology Group at UPF, specialising in digital audio processing and AI solutions for sound classification and recognition:https://www.upf.edu/web/mtg
6. Freesounds audio dataset creation and maintenance platform, creators of the ESC50 and UrbanSound8K datasets:https://freesound.org
7. GitHub repositories with various test models:
    a. https://github.com/microsoft/Semi-supervised-learning
    b. https://github.com/YuanGongND/ast
    c. https://github.com/ketanhdoshi/ml
    d. https://github.com/YuanGongND/ssast
    e. https://github.com/cwx-worst-one/EAT
    f. https://github.com/DCASE-REPO