Unibertsitate Masterra

Konputazio **I**ngeniaritza eta **S**istema **A**dimentsuak

Konputazio Zientziak eta Adimen Artifiziala Saila –
Departamento de Ciencias de la Computación e Inteligencia Artificial

Master Thesis

Development of a deep learning system for hummed melody identification for BertsoBot

Asier Alkorta Zabaleta

Master eta Doktorego Eskola
Escuela de Máster y Doctorado
Master and Doctoral School

Tutors

Ignacio Arganda-Carreras

Konputazio Zientziak eta Adimen Artifiziala saila
Informatika Fakultatea

Elena Lazkano Ortega
Konputazio Zientziak eta Adimen Artifiziala saila
Informatika Fakultatea

October 2020

Master eta Doktorego Eskola
Escuela de Máster y Doctorado
Master and Doctoral School

Table of Contents

Table of Figures

Table of Equations

Abstract

The system introduced in this work tries to solve the problem of melody classification. The proposed approach is based on extracting the spectrogram of the audio of each melody and then using deep supervised learning approaches to classify them into categories.

As found out experimentally, the Transfer Learning technique is required alongside Data Augmentation in order to improve the accuracy of the system.

The results shown in this thesis, focus further work on this field by providing insight on the performance of different tested Learning Models.

Overall, DenseNets have proved themselves the best architectures o use in this context reaching a significant prediction accuracy.

1. Introduction

Basque Folk Culture has many ways of expressing itself, but one of the most complex and spectacular ones are Bertsos. Bertsos are verses or poems composed on the spot and sung using some pre-accorded melodies.

The melody library is vast and usually each melody establishes a set of rules to follow for the composition of the Bertso, such as metrics and the fixed places where the rhymes go in it. So, the melody itself, is the frame where the Bertsolari (the person who composes and sings the Bertso in the moment) has to enclose his or her content in.

Bertsos are commonly sung in public appearances and celebrations, even though they are commonplace, they require a skilled Bertsolari to really engage the listener in what the Bertsolari wants to transmit in his or her effort.

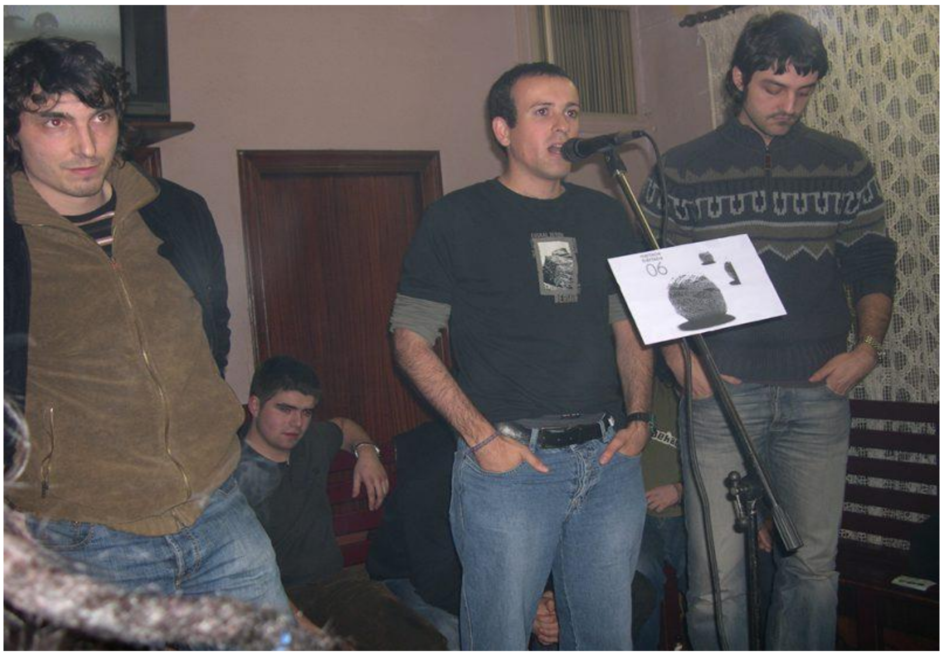


Figure 1*: Martxoan Bertsoa Urretxu.*
A local Bertso event held in a very informal way, resembling the official challenge. The Bertsolari is improvising the lyric in the moment.
Source: http://www.bertsolari.eus/erreportajeak/txapelketa-alternatiboak-vi-martxoan-bertsoa-urretxu-zumarraga-gipuzkoa/

Despite Bertsos are often sung in very informal situations, most likely at dinner parties and in a comical way as can be seen Figure 1, every four years a solemn championship is held where all the Bertsolaris compete with each other for the Txapela, or trophy in the form of a beret.

Set in the art of lyric improvisation, as proposed by Astigarraga A. et al. at[1], which can be seen in Figure 2, is a Bertsolari robot which improvises lyrics for given melodies, and it needs a way to input a melody by singing it. This poses the need to identify melodies given in an audio format, for the robot to know which melody to follow, setting the lyrical metric constraints, for the improvisation to be carried out.

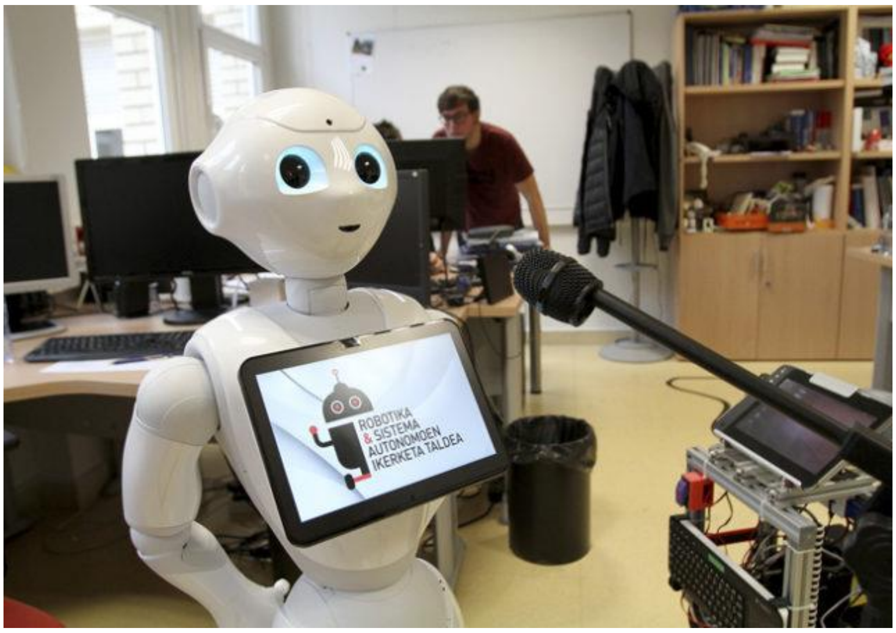


***Figure 2**: Bertsobot.*
Bertsobot is a robot replica of a Bertsolari, and tries to accomplish the lyric improvisation. The project is led by the Euskal Herriko Unibertsitatea[2].
Source: https://culturacientifica.com/2018/02/16/bertsobot-robot-versolari/

Such classification problems can be tackled using Machine Learning (ML) techniques, but the nature of those techniques have yet to be proven useful in the context of the problem at hand. This classification is vital to achieve the goal of identifying sung melodies and implement a Query-by-humming[3] system.

Latest advances in ML suggest that the use of state-of-the-art models in the situation, using Transfer Learning, can lead to obtaining meaningful results in the task.

The objective of this study is to see if the proposed solution is any good, quantify the results and set the path for future refinements in the process.

The advances made in the ML and Computer Vision fields help solve classification problems. If those paradigms are used correctly, many previously unsolved issues can be addressed and given a solution. The knowledge obtained developing those ML models can be harnessed in a very direct way by using Transferred Models, which lets us build a solution for our problem using the solutions given to other similar problems.

Knowing all this, a system may be possible to be built, that using the knowledge of current ML solutions via Transferred Models, can give a solution to
the melody classification problem at hand, to be used with the BertsoBot.

The system should be able to express audio data in an image format, and then using previously built ML solutions, it should be enough to map the input audio files to a known melody name or label.

Computer Vision models can be harnessed via Transfer Learning, to see if they are any good to solve the classification problem as used in the Urbansounds[4] classification dataset, where we are challenged to classify sounds recorded in an urban environment.

2. Basic concepts

2.1. Physics

Before delving into the actual work done, some time has to be taken to clear the air on some basic acoustic concepts.

Generally speaking, sound is a vibration that propagates as an acoustic wave, through a transmission medium, such as a gas, liquid or solid, as sound on Wikipedia states[5]. This sound is produced in a source, gets transmitted by the medium, and then is perceived by the listener.

If sound is a vibration that is propagated as a wave, it has the properties of a wave, such as frequency and amplitude. An example of a wave can be the dropping of a stone in a puddle, a wave is created from the point where the stone has entered the water in all directions, and it is spread by the medium, water in this case, as shown in Figure 3.

***Figure 3**: Sound's propagation in air.*
Sound's propagation in air, resembles the of a wave in a puddle after throwing a stone in it.
Source:https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Map%3A_Physical_Chemistry_for_the_Biosciences_(Chang)/11%3A_Quantum_Mechanics_and_Atomic_Structure/11.01%3A_The_Wave_Theory_of_Light

The waves have properties that have to be explained, albeit in a generic way, and not dwelling on the physics too much.

In Figure 4, we see the common representation of a wave used in physics. There we appreciate how it is defined around an axis and fluctuates up and down that axis regularly. This fluctuation repeats in time, therefore, if we measure how many times this fluctuation occurs in a time interval, we would be measuring the **frequency** of the wave, as it is defined. Another property is the **amplitude,** which is defined as the distance from the axis to the peak of the wave. Also, the **wavelength** is defined as the distance between two peaks of the wave.

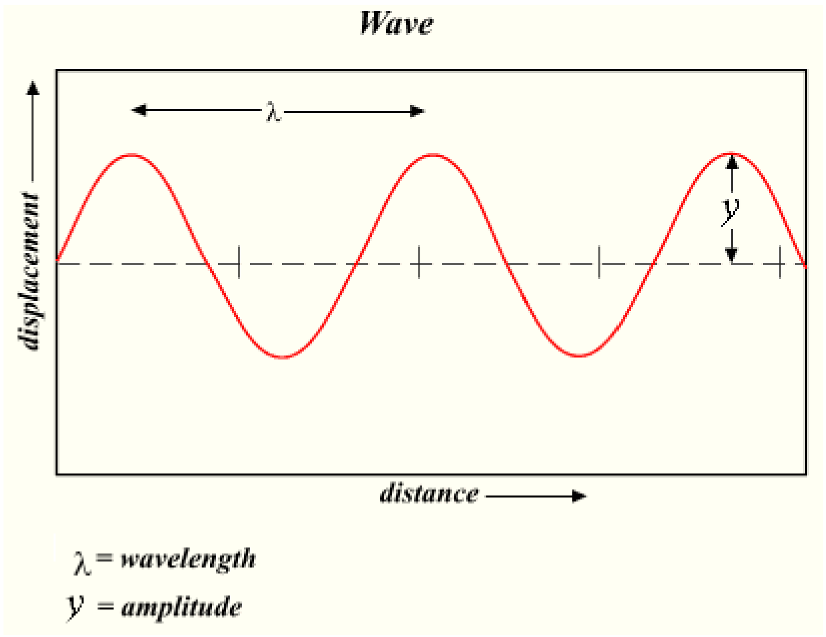


Figure 4*: Wave properties*
Physics define the properties of a wave like wavelength, amplitude and frequency
Source: https://www.toppr.com/guides/physics-formulas/wave-formula/

Those basic concepts will help us understand some of the physics involved in the process followed in this document.

A source produces a sound (voice, loudspeaker…), it gets transmitted usually by the air as a wave, then a listener's ears perceive this wave, and his or her brain interprets this perceived sound. The field that analyses the perception of sound is called psychoacoustics, and it is very important to note that how the sound is perceived can help us focus the process we are about to describe, at some particular stages.

Sounds produced in different sources produce different situations depending on the amplitude and frequency of the sounds that reach the ear. This again falls into the field of psychoacoustics, as it is the brain that plays a major part in the interpretation of the information that the ears generate.

Synthetic sounds can be generated resulting in a pure electric sine wave, using an oscilloscope for instance. In the example shown in Figure 5, a pure wave can be created of a certain amplitude and frequency.

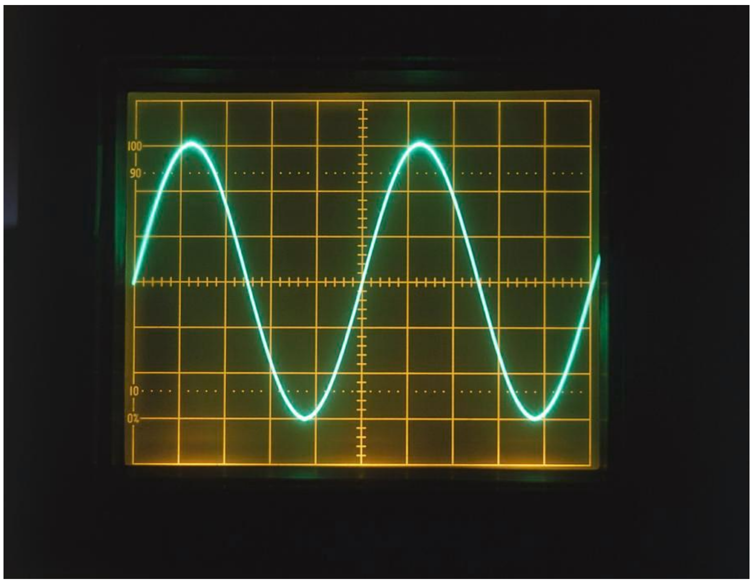


***Figure 5**: A pure sine wave can be created in an oscilloscope.*
The sound of a pure sine wave's sound is not natural, as its composition is too simple
Source: https://fineartamerica.com/featured/sine-wave-display-on-oscilloscope-screen-dorling-kindersleyuig.html

Using a synthesizer, the same can be done to generate a pure sine wave, that can be reproduced by a loudspeaker later on and transformed into an audible sound.

If this process is followed, the resulting sound is very clean and artificial. Nature never produces this kind of clean sounds that are composed by a single wave. In Figure 6, we can see what happens when two waves get summed, and how the combined wave is more complex than the clean synthetic waves we have seen until now.

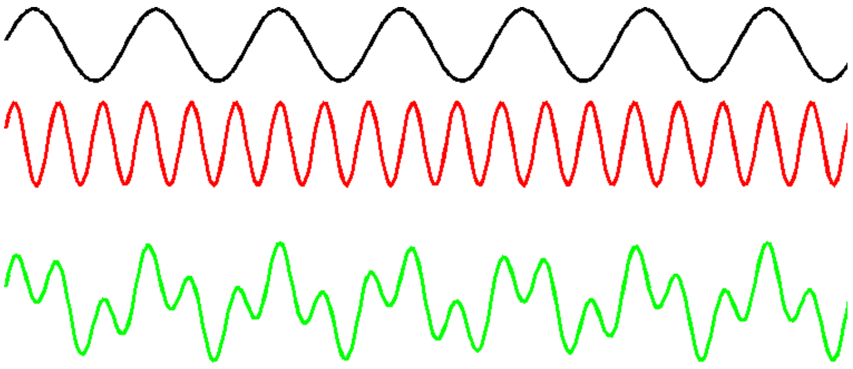


***Figure 6**: Sine wave addition example.*
By adding multiple waves, like the first two simple but different waves, a more complex third wave can be achieved, as nature is prone to produce.
Source: https://www.fiberoptics4sale.com/blogs/wave-optics/100149702-phase-velocity-and-group-velocity

In Figure 7, we observe how nature is not a friend of regularities, and that the sounds that are naturally created, hold little resemblance to the simple synthetic wave, and how they, in fact, are far more complex.

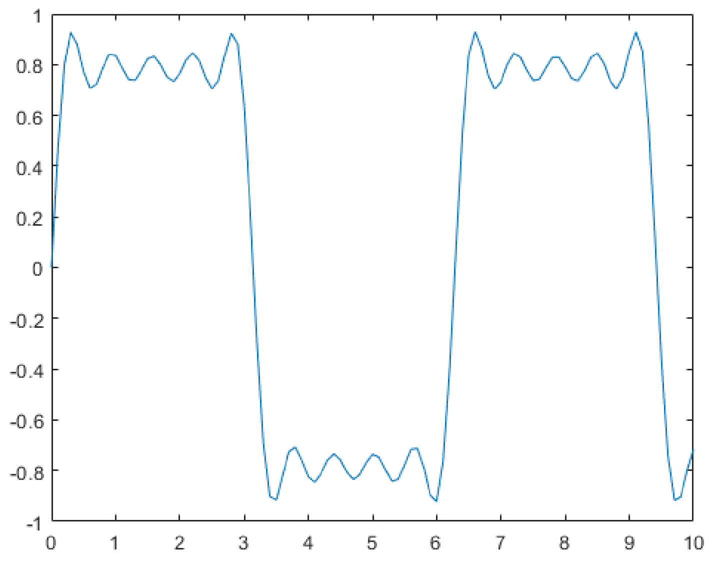


***Figure 7**: Complex waveform.*
After adding numerous base waves, a more complex waveform is achieved
Source: http://digitalsoundandmusic.com/chapters/ch2/

If we take an image of a graphical equalizer, as we can see in Figure 8, where a spectral representation of the audio is displayed, showing the relation between the amplitude and the frequency of the audio in a moment in time, we can easily see that a natural sound resonates in many frequencies at the same time, thus, its composition is far more complex.

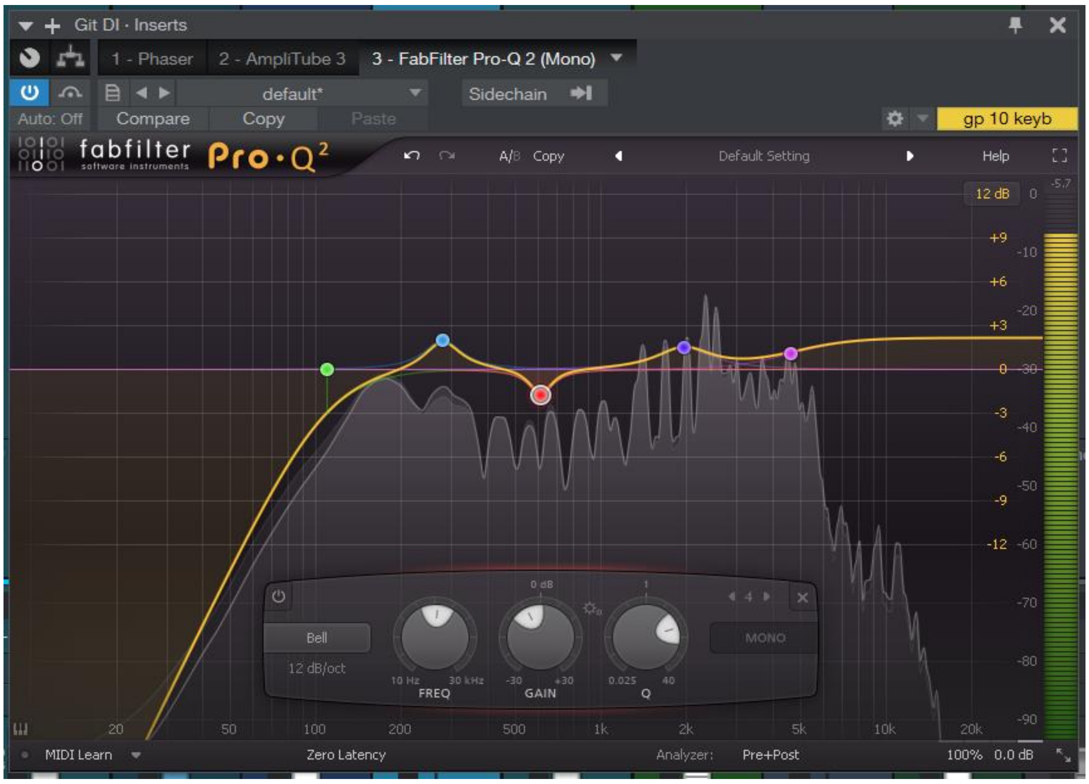


***Figure 8**: Equalizer showing natural voice wave.*
In its complexity, many frequencies resonate, and the equalizer tries to shape and improve the audio by manipulating those harmonics

A sound is composed of the fundamental frequency and its harmonics. The fundamental frequency is the main frequency in which the sound resonates the most, the lowest frequency in this case. A harmonic is described as a resulting wave after the positive multiplication of an integer with the fundamental wave. After summing both waves, we get the composite sound. Thus, a natural sound is composed of the fundamental frequency, and its harmonics sounding together. Usually not all the harmonics resound with the same energy and the difference of those results in a separate timber for each sound source. This is how a violin sounds different from a guitar or a human voice for instance.

Being sound a waveform, there are many ways it can be visualized, one such way being via the use of spectrograms. This kind of representation happens to be very fitting to the task we are planning to undertake.

In a spectrogram, data is pictured in an x-y two-dimensional representation using, in addition, some colouring to incorporate the third-dimension data.

In our case, in the X axis the time parameter is shown, using the Y axis to represent the frequency parameter. Finally, the sound pressure level accumulated in a certain frequency, measured as dBs in a given time, is displayed by a colouring representation using cold colours to show low values and hot colours for the high values.

In the example in Figure 9, we can see how, in the whole time range displayed, energy is accumulated in the frequencies below 2,000 Hz, and we can also see how above that limit, much less sound pressure is accumulated. This paints a picture where in the below 1 KHz area, more purple or yellowish colours are present whereas in the above 2KHz area more blue, black or darker colours are used.

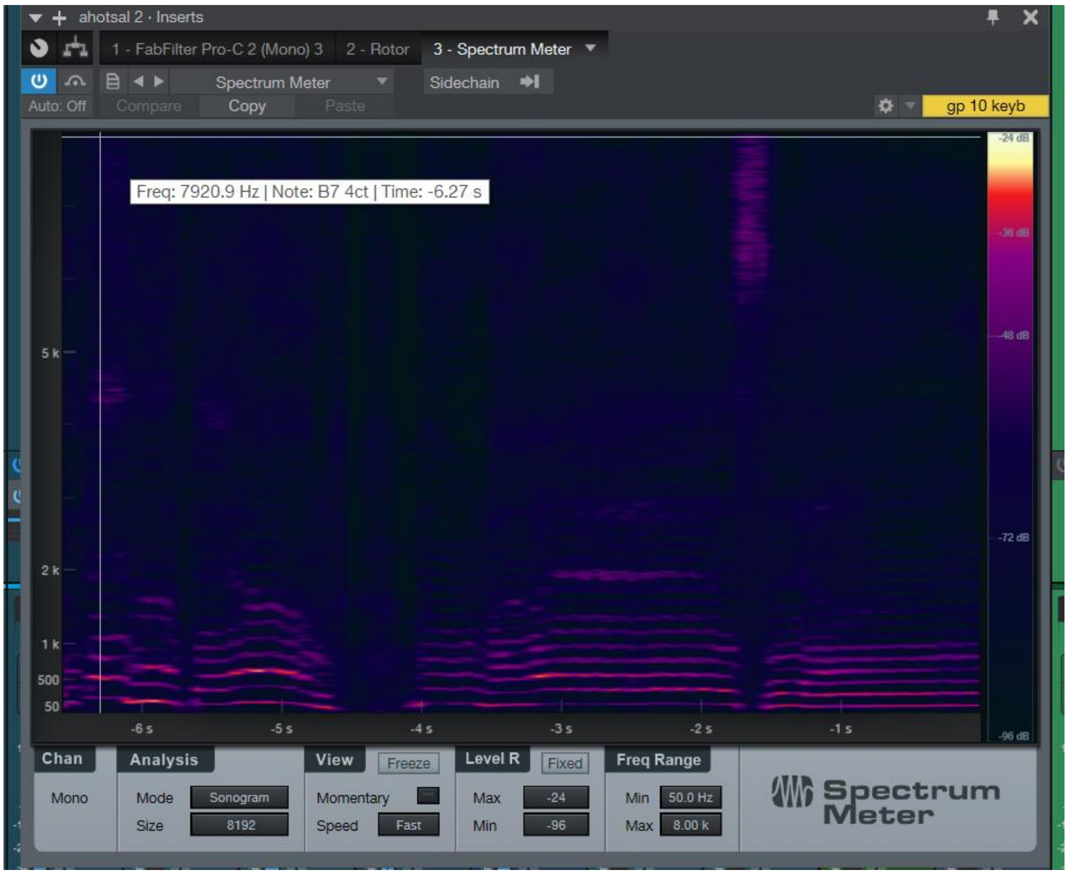


***Figure 9**: A example of a spectrogram.*
A spectrogram shows where in the audible frequency spectrum energy is accumulated. It displays such energy accumulations using a colour code, using warmer colours for where more energy is present and colder colours for when there is absence of energy.

2.2. Artificial Neural Networks

In Artificial Intelligence (AI), Artificial Neural Networks (ANNs) or Neural Networks (NNs) are computer models or set of algorithms, that are vaguely based on actual biological neurons, and try to loosely resemble their way of working. The base of an ANN is mathematics, and so each one of its components is a mathematical simulation.

For instance, a neuron is connected to other neurons by its synapse or connection, and signals are passed from one to another. Artificial Neurons (ANs) are also connected by edges and they simulate the behaviour of their biological counterparts by forwarding signals depending on the algorithm of activation function they are executing, the inputs they receive, and the weights that are adjusted in the learning process, while the signals are some mathematical real number values.

ANNs are commonly used in Supervised Learning, and this learning process maps input to a given output.

As we can see a learning phase is necessary for the NN to learn how to behave when it’s presented with the input. The goal would be that after the learning process took place, the Neural Network will be able to produce a correct answer based on the inputs it will receive.

But before we delve into the nuts and bolts of the process, and see how all this is implemented, a little bit of history should be told.

2.2.1. History of ANNs

History of ANNs began in 1943 when neurophysiologist a study was conducted on how biological neurons might work, and an actual electrical NN was modelled as proposed by McCulloch WS et al at[6]. In the process, they created the computational model using threshold logic and algorithms. Two approaches were born from this paper, one that studies the biological process that takes place, and another that focuses on the applications of NNs, AI.

In 1949, the fact that the more a neural pathway is used, the more it is strengthened was seen as proposed by Hebb DO et al at[7].

In the 50s, the computational power became enough so that those NNs could be simulated, even though the first attempts were not satisfactory. Despite this, several advances were made, and the perceptron was created as proposed by Rosenblatt F. at[8].

In 1959, it was discovered that two types of cells in the primary visual cortex, simple and complex cells as proposed by Hubel Dh et al at[9].

In the first many-layered functional networks were introduced as proposed by A. G. Ivakhnenko et al at[10].

Four years later, in 1969, two problems were discovered with the machines that processed the NNs as proposed by Minsky M et al at[11], the first one being the inability of the perceptrons to process the eXclusive-or circuit, and the other one related to the amount of computational power required by a large NN. The discovery of those issues, led to a loss of interest in the field.

In those uncertain times, AI focused on Expert Systems. Those systems were centred around certain explicit algorithms, which modelled some rules.

In 1975, the training of multi-layer networks was made possible using backpropagation as proposed by Werbos P et al at[12].

In 1980s, the parallel processing power became available via connectionism, as proposed by Rumelhart et al at[13], [14], and Support Vector Machines and simpler gradual linear classifiers grabbed the focus of NNs.

In 1992, max-Pooling was proposed to obtain a better 3D object recognition, and in 2010 training using backpropagation with the help of GPU acceleration, the combination of those components showed a better performance than others in the publications proposed by Weng J et al at[15], [16] and [17].

The vanishing gradient problem appears in Feedforward Multilayered Networks and Recurrent NNs. This error consists of when the error gets propagated from layer to layer, when performing the backpropagation shrink exponentially which prevents the tuning of the weights of the neurons.

In 1992, the solution to this problem as proposed by Schmidhuber J et al[18] was introduced who used a multi-layered hierarchy of networks, and pre-trained one level at a time and then fine-tuned it with backpropagation.

In 2006, the idea was posed of using a restricted Boltzmann machine as proposed by Hinton G et al at[19] and Hinton G at[20] , learning a high-level representation using successive layers of binary or real-valued latent variables. Once enough layers have been learned the architecture could be used as a generative model to reproduce the data and when the sampling down occurs from the top-level feature activations.

In 2011, Convolutional Neural Networks, using supervised Deep Learning methods obtained human-cognitive performance on a number of practical applications.

In 2012, a network was created which could learn high-level concepts such as cats using unlabelled Youtube images as proposed by Le QV at[21].

Those presented methods in training Deep NNs, like unsupervised learning, were used on previously proposed challenges successfully, and as the use of distributed computing and GPUs became widely available the deployment of NNs became widespread.

2.2.2. How does an AN work?

The simplest element in a NN is the neuron. It uses inputs (i) and weights (w) as parameters, and produces an output (y), based on the activation or threshold function (T).

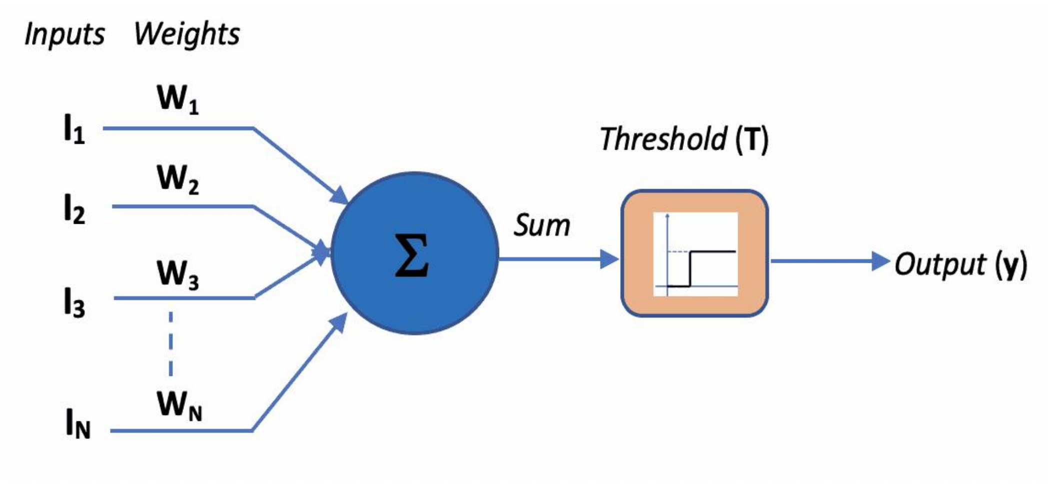


***Figure 10**: AN example, showing basic structure.*
*The **N** number of **I** inputs and **W** weights, how they are **sum**med, the **T** threshold or activation function and the **y** outputs*
Source:
https://aibusiness.com/document.asp?doc_id=761027&site=aibusiness

The representation in Figure 10 responds to a certain mathematical definition displayed in Equation 1.

$$y_k = \varphi \sum_{j=0}^{n} w_{jk}\, i_k$$

***Equation 1**: AN formula.*
The AN formula is the mathematical representation of the AN that will be used for its implementation.

Here we see that if we have, using k as the neuron index, for the k neuron, we have to multiply the input of the neuron i_k with the corresponding weight of the k neuron for that input w_{jk}, then calculate the sum of all those from 0 to n, and pass the result through the activation function (φ). If depending on the input to the activation function passes the threshold or it doesn't, the neuron produces its output y_k.

2.2.3. Activation functions

An activation function is what defines the output of a function given its input. It is what says what the result will be depending on the input data.

To illustrate this, we will explain the Rectified Linear Unit (ReLU) activation function.

In Equation 2, we can see its mathematical representation:

$$\phi(v_i) = max(0, v_i)$$

***Equation 2**: ReLU formula.*
The ReLU activation function has its formula that its showed here

If we define ϕ, the ReLU function that gets the value with index i, v_i as a parameter, then the maximum between 0 and that v_i value is performed.

And in Figure 11 we show the illustration of how the function works:

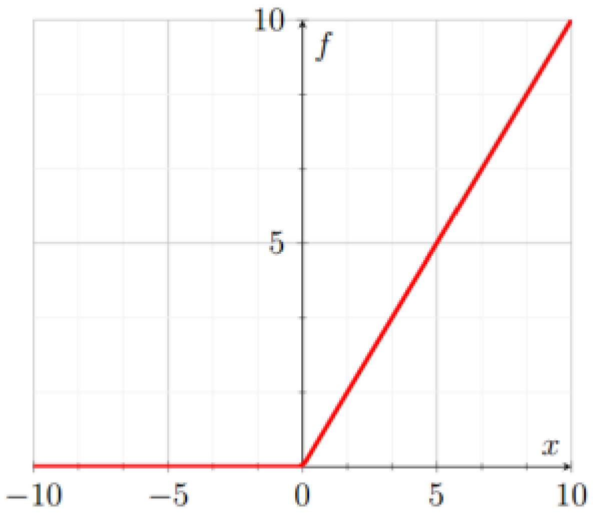


***Figure 11**: ReLU activation function.*
The illustration of how the ReLU activation function works
Source: https://yashuseth.blog/2018/02/11/which-activation-function-to-use-in-neural-networks/

As we see, it is trivial. If the input value is smaller than 0, then the output is 0, otherwise, if the input is bigger or equal to 0, then the output is the same as the input.

Although we explained this simple ReLU example, there are numerous activation functions available, and using the correct one can help to obtain better results, such as linear, Gaussian, softmax etc.

2.2.4. NNs

As we have said a NN, based upon biological neurons, builds a computer system that tries to learn how to behave depending on the input, and some weights that will be set in the learning process.

But first let's see a schematic of how a NN is and its components, in Figure 12.

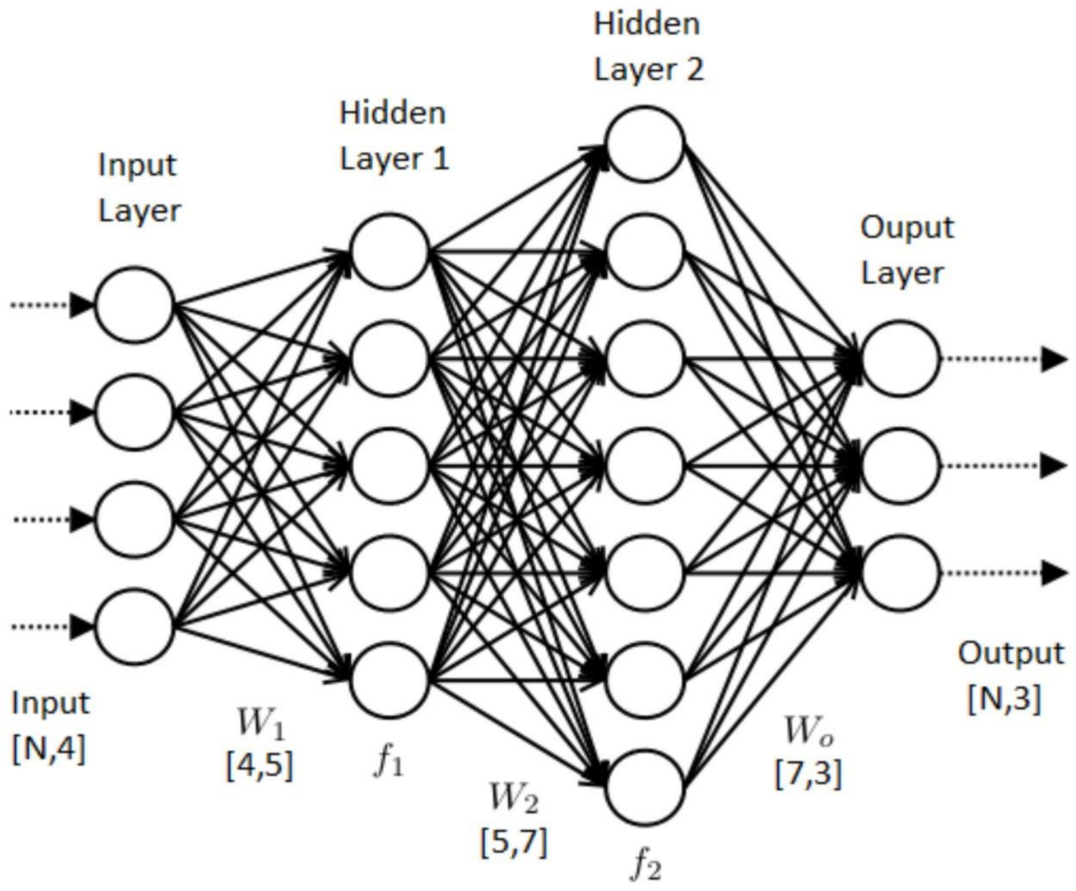


***Figure 12**: Inner structure of a NN.*
The NN showing its multiple layers, for data input and output, and the intermediate hidden layers. All the edges that communicate the neurons are shown. The layered nature of the internals of the network should be noted, where each layer, present neuron, and edge weight has its corresponding index.
*Source:https://miro.medium.com/max/700/1*ZB6H4HuF58VcMOWbdpcRxQ.png*

Here we see how a NN's first component would be, well, the **neuron** itself, which in the image we see represented as a circle. Then we would have to connect each neuron with another one, those connections are pictured with the lines that represent the **edges**. Then we have the input that the NN will have to process in order to obtain a result. This **input** is where the data will be fed to the NN. If we have some inputs where the data will be read by the NN, there will also be some outputs where the result will be expressed and presented. To adjust the behaviour of the NN, each neuron will have a **weight** associated that will be adjusted in the learning process.

Next, we can see how the network is divided into **layers**. Each layer will be composed by N neurons, each one with m number of inputs and p outputs, so in order to easily name the rapidly rising number of neurons, usually, each layer is given its layer index, and then to identify each neuron within the layer an additional index is set. Then, if we talk about L 3,4, we know that we are talking about neuron number 4 in the 3rd layer. Each Layer can be represented by an array of neurons. The outputs of each layer will be connected to the inputs of the neurons in the next layer.

There is an **input layer**, where all the neurons that are responsible to get the input data are placed, and of course, the **output layer** where the neurons responsible to produce the output are.

In addition to this input and output layers, there are some extra intermediate layers that are called **hidden layers**. In Figure 12, the hidden layer structure is a very simple one consisting only in 2 hidden layers.

Talking about the connections between the neurons in different layers, we see how the input layer having 4 neurons, propagate the output of each input layer neuron to all the neurons in the next layer, which is composed of 5 neurons, then, knowing the structure of each layer, we can determine the number of edges necessary to make the connections. In this case, each input neuron will propagate its output to each of the 5 neurons in the first hidden layer, and subsequently, each neuron in the first hidden layer will read the input from the output of each of the neurons in the input layer. This can be applied to the next layer until the process finishes. With this knowledge, we can determine which the edges between layers will be and making the connections will be straight forward.

2.2.5. The learning process and Backpropagation

We know that when we use a NN to perform supervised learning, the network is able to map the provided input to the given output, but how does this happen?

When we train a NN, the network the input and the output data is given, but we said that there are weights in each neuron that have to be calculated. These weights give more or less importance to the value they are weighting taking the result into account, or better said knowing which the result will be. This way the network can adjust itself to try to produce the correct answer. This phase is called **forward propagation** because it adjusts the weight parameters when going forward in the process.

2.2.6. Loss Function and Backpropagation

But sometimes, after the learning process has occurred, the NN will not give the correct result, and in this case, we will get an error. We can easily give an error percentage after the forward propagation phase has taken place, because the correct output result data is known to the system, and the correct/error percentage result can be calculated. So, this information can be used, and a **Loss Function** (also cost function or objective function) can be proposed. This function will measure the error or how good or bad a result is in comparison with the correct result we already have. If this function value is 0 then we will know that the answer we got is correct.

Generally, this Loss Function works calculating an average of each of the losses on all the training examples.

For example, if Mean Squared Error is used, it would be calculated as stated in Equation 3. In a vector of values where the difference between the observed value Y and the predicted value Y' is squared, and then the average of all those differences is calculated, performing a sum of the squared differences, and then dividing them by the total number of values n.

$$MSE = 1/n \sum_{i=1}^{n} (Y_i - Y'_i)^2$$

***Equation 3**: Mean Squared Error formula.*
The Mean Squared Error can be used as a Loss Function to calculate the error in each result obtained

This method would give us a way to determine how close is the result we are getting, to the correct one. Then, the NNs propagate this information backwards, starting from the final layers, through the hidden layers, each neuron in each layer receiving a fraction of the loss associated with the whole layer. This step is repeated layer by layer in a backward direction; thus, the whole process is named as **backpropagation**.

A general overview of the whole learning process is shown in Figure 13.

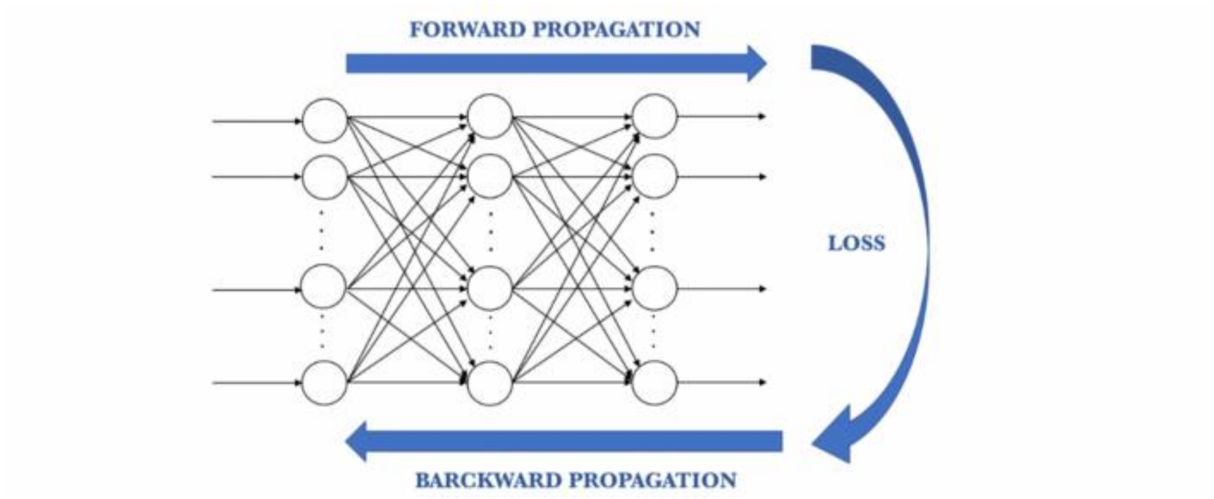


***Figure 13**: Showing the direction in which forward and backpropagation occur.*
As the loss can be calculated when the results are obtained, it can be propagated backwards to adjust the inner weights of the NN in accordance.
Source: https://torres.ai/deep-learning-inteligencia-artificial-keras/

Now each neuron knows which is its loss or cost, in each of its predictions or mappings that it has made, and now, it can adjust its weight minimizing this loss. This algorithm is called gradient descent and tries to find a local minimum, on each step it is executed, hoping to find the best solution, as seen in Figure 14**.**

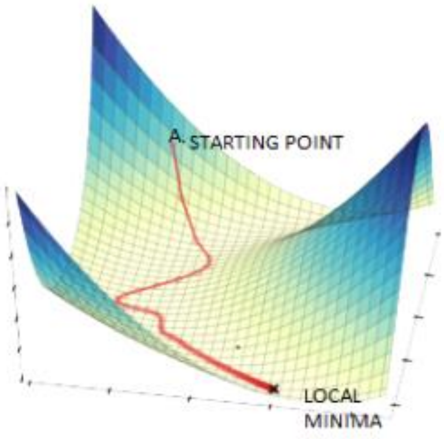


***Figure 14**: The gradient descent algorithm.*
The gradient descent algorithm tries to find the best solution or minimum by descending in the direction where the steepest downward slope is
Source: https://www.datasciencecentral.com/profiles/blogs/alternatives-to-the-gradient-descent-algorithm

A common way to explain the gradient descent algorithm is the example of the hiker in the mountain with a dense fog.

Imagine we go hiking in a nearby mountain and that when we are on our way some heavy fog comes in, we get absolutely lost, and do not know where our way back is.

This is a difficult situation because we don't really see any reference of which the way back should be, but what we do see is the slopes surrounding us and the steepness of each of them. Having that information, we determine that a safe behaviour to find our way back is to always take the steepest path going down. This will take us closer to the bottom of the mountain each time, and hoping that eventually, we will end up there. The method has its problems, like getting to a local minimum instead of a global one, but this example is a great way to understand the gradient descent method.

There are certain methods to aid the gradient descent algorithm to come out of local minima and try to find a better solution to the problem. In this case, the search algorithm will manage to get out of a good solution and try to get a better one. In Figure 15, we can see how although a good solution is found, the algorithm tries other solutions, that maybe are not as good as the optimum solution found up to that moment, but that eventually may lead to other even better solutions.

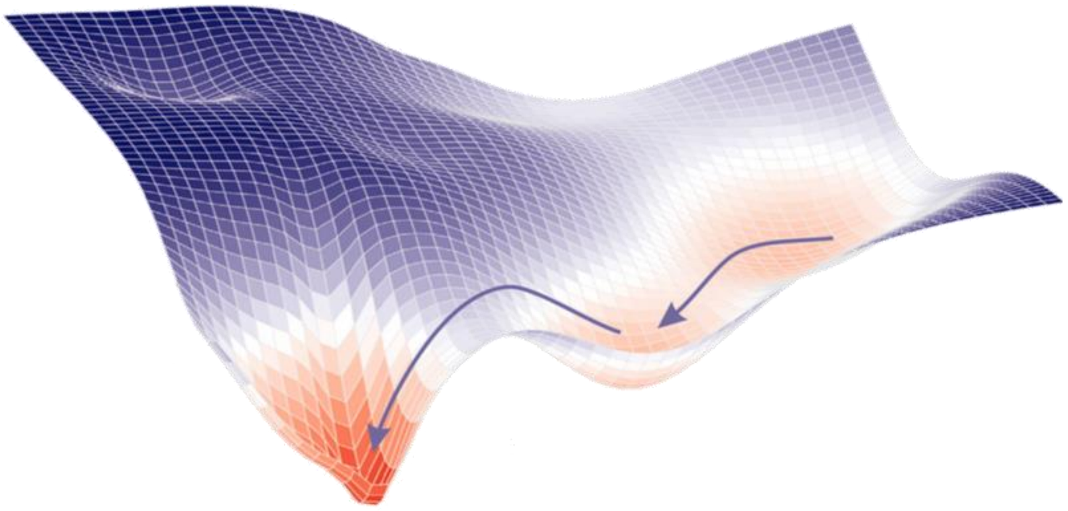


***Figure 15**: Gradient descent searching for a better solution.*
As gradient descent can get stuck in a local minimum, some improvements can be implemented in order to get past that point and search for better solutions.
Source:http://primo.ai/index.php?title=Gradient_Descent_Optimization_%26_Challenges

2.2.7. Overfitting

There is a concept that has to be taken into account that affects ML models and that has to be explained. This concept comes from the statistic nature of the field.

When an analysis corresponds too closely to the dataset that it analyses, then when the model is presented with new data it is not able to make a reliable prediction.

This happens when the model has more parameters than the data needs, and in the learning process, after one point, the model starts learning data noise as if it were part of the structure of the data.

In the same way that overfitting occurs, there is also the effect of underfitting, which happens when the statistical model is not capable of learning the data close enough and fails to capture the structure of the data in a proper way.

For example, this could happen when a linear model is used to represent nonlinear data. In that case, the model will not be able to predict the values that are away from the linear representation, rendering the linear model inappropriate due to underfitting.

In the learning process, as we are using the gradient descent, the iterative nature of the method lets us evaluate the situation on every iteration, and lets us implement an early stopping technique.

The evaluation uses the Loss and Accuracy metrics already explained, evaluate the model against the validation data, to see what kind of values are gotten.

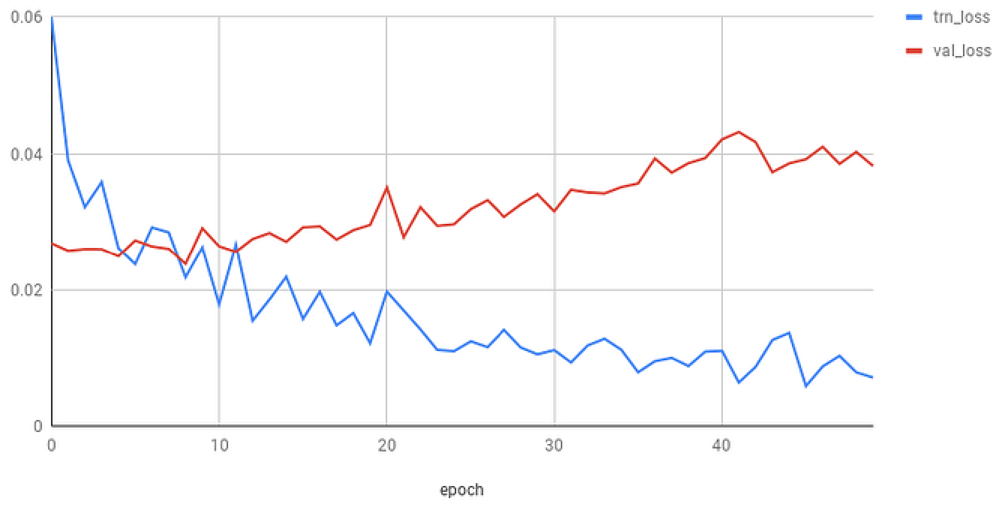


Figure 16*: Overfitting detection using loss metric.*
Overfitting can happen when training a model when the validation loss obtains bigger numbers than the train loss. If that occurs, there is no point in keeping training because the model is learning noise instead of the structure of the input data, as it should be doing.
Source: https://forums.fast.ai/t/determining-when-you-are-overfitting-underfitting-or-just-right/7732/9

When the validation data loss is higher than the training loss, then we can say that overfitting is taking place, so there is no point in keeping training the model because we already know that it is overfitting as shown in Figure 16.

Seen that knowing when to stop is key in the learning process, this early stopping technique lets us control the overfitting effect, and gives us a way to prevent it from happening.

2.2.8. Convolutional Neural Networks

Used in Deep Learning, Convolutional Neural Networks (CNNs) are a kind of NN which are usually used for image processing and pattern recognition. The idea is to be able to recognize patterns in images using this kind of networks, as the algorithm assigns importance (weights) to certain image characteristics that the network will recognize. Based upon the organization of the visual cortex, in this kind of networks, the neurons will capture the stimuli of a region of the visual field, this way, recognizing a certain pattern when the data matches the criteria close enough.

That kind of networks were designed to map image input data to an output. As they are so effective and get such good results, they have become the preferred solution for problems of classification involving image input data.

Convolutional Neural Networks internally consist of something called kernels to detect features in the input image, and then some pooling stage to control the amount of intermediate data generated. Afterwards, those features can be learned in subsequent layers of neurons. Those kernels work as templates that are able to detect shapes in the input data.

2.2.8.1. Kernels

In the digital domain, images are represented as a matrix of pixel values, and this information is what will be fed to the CNN. Then we need something to detect the features in the image, where something called a kernel filter or feature detector is used. This kernel is going to contain a smaller set of pixels that will represent a shape that will be used to traverse the original image to try to detect the image features that will match the criteria set by the kernel. This phase is carried out by traversing the kernel through the image and calculating the level of matching of the image with the actual kernel. Then, the level of matching will be forwarded to the output for the next layers to process.

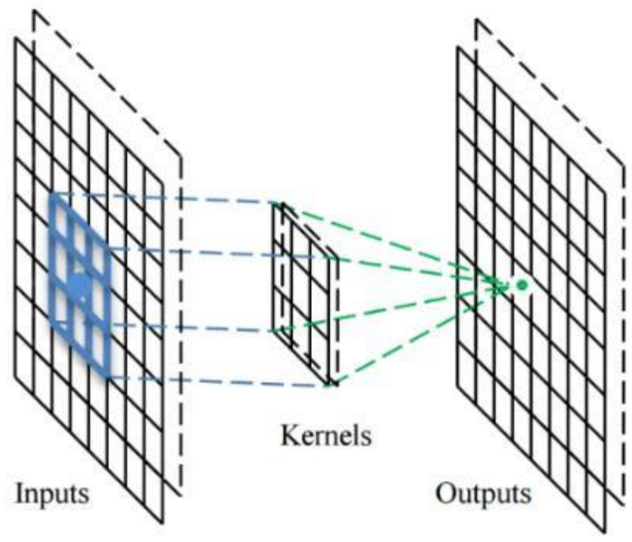


***Figure 17**: Kernels usage example.*
Different kernels are used to detect features in the input data by trying to match the shapes defined by those kernels.
Source:https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7?gi=cebb20623953

In Figure 17, we see how a kernel searches in a certain input image area and checks whether the input data meets the criteria set by the kernel (set of kernels in reality with each feature to be detected), and finally, the corresponding result is created and outputted.

2.2.8.2. Pooling

After those results have been produced, the usual step is pooling. In this step, the objective is to downsample feature maps to summarize the presence of features in the original data map.

Different pooling algorithms can be used to perform this step. For example, seeing the following image, if a max-pooling behaviour is used, then the 4x4 map can be downsampled to a 2x2 map, just by substituting 2x2 sample maps by the maximum value in the original 2x2 value map.

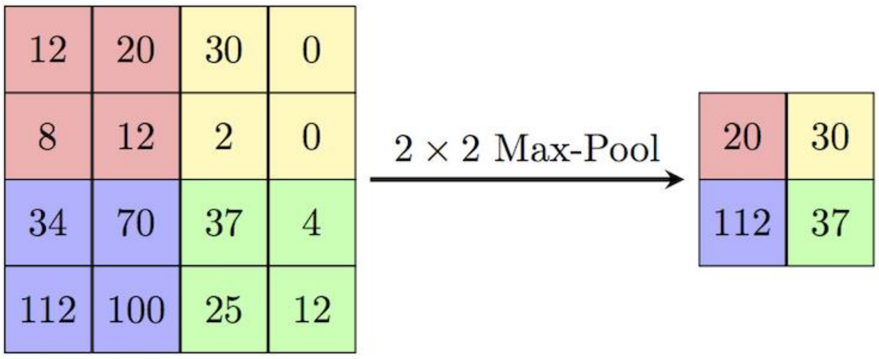


***Figure 18**: Pooling usage illustration.*
Pooling helps to keep the data in a manageable size by downsampling it and reducing the number of values to pass to the next layer.
Source: https://cs231n.github.io/convolutional-networks/

In Figure 18, we see how the first 2x2 map is substituted by 20, because 20 is the maximum value in the 12,20,8,12 value set.

Pooling is used to control the size of data to be processed, due to the fact that the number of data outputted by the feature extraction conducted by the convolutional layers can be very high.

2.2.9. Transfer Learning

ML techniques achieve notorious success in solving the problems they are intended to solve, and usually, their design is aimed with that objective in mind, but the truth is that in the real world there are problems that even if they are not equal, they certainly are similar. The idea behind Transfer Learning is that if we already have a solution for a given problem, maybe the same solution could be used to solve another similar problem.

This way of trying to search for a solution, using solutions proven useful in other situations, has shown itself as a successful and a very interesting one. Because of the ample calculation power and investigation needed to come up with those solutions, the use of solutions that have been able to solve a certain situation can ease the process of searching for a suitable solution.

In many situations, the search for the appropriate ML model requires an extensive search to be conducted on the possible fitting models to find the optimal one and train it with lots of data. Sometimes, the data available for training a ML model is not enough and the use of those pre-trained models makes the harnessing of those models invaluable, saving the user of having to collect a much more ample dataset, and obviously the necessary time and process power of making the actual training process to happen.

For instance, if we train a model to recognize bicycles, maybe this same model could help recognize motorcycles. Although bicycles and motorbikes are completely different elements, it is undeniable that they hold some resemblance between them, thus, if we already have the solution for recognizing bicycles, we could use the same solutions with motorcycles, and we can be very successful in it, saving many hours of search and calculations in the process.

Sometimes the application of those Transfer Learning techniques is not straight forward, and call for some extra work for example if an image processing model is used for something else, the input data of the NN, will have to be expressed as an image, for instance, otherwise, the results obtained by the whole process will not be as good, because the aim of the transferred model is to process images. This extra effort that has to be undertaken by the whole proposed solution pays off big time, if the solution is given in accordance, due to the reasons previously posed.

We already know that some NN models have outstanding performance in solving computer vision problems using the imagenet[22] dataset, and bibliographic revision taught us that this method has been used in the past to identify emotion in spoken language. This leads us to think that those same models can be used and their power harnessed to reach the goal of this research. The only thing we need to do is to express audio data in a visual format, which we will address in future sections.

2.2.10. Popular NN architectures

Although there are many different models available, the study will focus on those available in the used software package Keras[23], which will be presented later in this document.

The use of other models involves extra installation effort and as the list of provided models is extensive enough, we will stick with that original list.

In that pool of models, there are different types of them at our disposal, and it would be interesting to write a word about each of them and comment on their differences.

VGG16 and VGG 19[24] are two very Deep CNNs, setting the objectives in large-scale image recognition. They use 3x3 convolution filters in

combination with 16 and 19 weight layers (this is where the names are taken after), trying to increase the depth of the network architecture.

ResNets[25] or Residual Networks try to ease the training of substantially deeper networks than the ones used for those tasks before, using 50, 101 and 152 layers (again, here is where the names come from), They reformulate residual functions, and they end up being 8 times deeper than the VGG nets, but retaining lower complexity.

A posterior effort in the development of those Residual Nets, lead to propose their second version or V2. These versions of the models improve generalization and make training easier by directly propagating forward and backward signals from one block to another, when identity mappings are used to skip connections and after-addition activations. This proposal led to reformulate the available ResNets using the new behaviour, for each previous configuration.

Mobile and embedded vision applications have different requirements than regular ones, so MobileNets[26] were presented to meet such requirements. Those nets use depth-wise separable convolutions to build lightweight networks.

A second version of the architecture was defined improving the performance of those MobileNets. This architecture inverts the input and output of the residual block layers, converting them in bottlenecks as traditional residual models do not do. Those traditional residual models use expanded representations in the input. MobileNetV2[27] uses depth-wise convolutions to filter out and get hold of the features in the intermediate expansion layer.

To maintain representational power, it is important to remove non-linearities in the narrow layers.

NasNets[28], are defined with transferability in mind, designing a new search space (NASNet search space), to search for an architectural building block on a small dataset, and transferring the actual block to a larger dataset.

As improvements model size and the associated computational cost result in better results, the InceptionV3[29] network tries to get a bigger scale network trying to use the added computation as efficiently as possible. This architecture uses accordingly factorized convolutions and an aggressive regularization to achieve its objectives.

After the presentation of the Residual networks, a new reformulation of the Inception networks has been introduced by Szegedy et al at[30], trying to combine those two architectures.

Xception[31] networks are presented as the solution between regular convolution and depth-wise separable convolution operation.

Building on the work done before Densely Connected Convolutional Networks[32] are presented. DenseNets rely on the idea that as CNN can be substantially deeper and efficient to train if the connections between the layers close to the input and the ones close to the output are shorter.

Instead of using L layers with L connections, DenseNets use L(L+1)/2 direct connections, the feature maps of all previous layers are used as inputs, and its own feature maps are used as inputs to all the next layers.

This enables the architecture to alleviate the vanishing gradient problem and improve strengthening the propagation, encouraging feature reuse and reducing the number of parameters in the process.

3. State of the art and bibliographic revision

Melody recognition is not a new challenge, and there already are some commercial applications that perform this task. Allegedly, those applications use very complex and advanced algorithms to get the job done, but taking into account the latest advances in AI, we hypothesize a simpler approach can be taken to reach the same result harnessing its power.

Music can be represented in many ways in the digital domain, as scores, audio… and musical databases are indeed compiled from there, but a common problem is how to consult those databases.

Sometimes, even the person that needs to perform a search in the database does not remember or simply does not know the name of the musical piece that he or she wants to search for.

As a remedy for such situations, Query By Humming (QBH) can be used for searching. Even though the name of the tune eludes the searcher, as the melody is known, the possibility to conduct a search using the information given by humming the melody is exploited, creating a natural way to express the query.

QBH is the search method tailored to the nature of the data that has to be queried.

The system has to process the audio information of the hummed melody and then perform the search. Usually, such processing relies on audio processing which makes use of the wave nature of the signal and applies several techniques to extract information from the query audio.

Song representations like the Hidden Markov Models were used as proposed by Qin J et al at[33] to represent each musical event or note as a point, and a song as a sequence of them. Also, some representations use the Modified Discrete Cosine Transform as proposed by Shifrin J et al at[34] to represent the audio information. Other techniques perform onset detection in the music by using human hearing models with local minimum function as proposed by Jang J et al at[35].

In other works, a time series approach is used to query the music in a time series, which help overcome the note error issue as proposed by Fu L et al at[36].

Another possibility is to use the spatial arrangement of instrument and voices in the stereo mix to create the query as proposed by Fu L et al at[37]. Also, music can be segmented, then indexed so that after that, a query can be made to the database using pitch vectors as proposed by Muda L et al at[38].

Others suggest that a correct way to encode the information is by using the relative pitch changes in the melodic information of the song. This can be done by extracting the notes form a monophonic humming, and then segmenting and quantizing the results to some discrete notes as proposed by Casey MA[39].

The combination of a number of techniques has been also used. After building a humongous music database, then the skeleton of the melodies is extracted so that Genetic Algorithms can perform similarity matching. The arrangement of the point sequences provides robustness against pitch errors and tempo variations, and if the data is likely to generate the query, they can be judged similar as proposed by Antoniol G et al at[40].

When retrieving the information, the sung query can be translated into notes, and then the pitch vector can be searched as done with the index construction. A list of candidate melodies can be maintained and then an error-tolerant similarity search could be carried out as proposed by Casey MA et al at[39]. The author also proposes, two algorithms for musical extraction can be used, to improve the results.

Another proposal is the use of time series to represent the melodies and to use time warping distance metric similarity for comparisons as proposed by Fu L et al at [37].

Some methods loose retrieval precision as they use the melodic contour of hum and rely on error-prone segmentation procedures. Others may have better precision, but while matching the actual music they are slower, as they rely heavily on Dynamic Time Warping techniques as proposed by Nagavi T et al at[41].

Some previous work has been done to implement QBH and technology like Parsons code[42] has been developed and put to work. This encoding system encodes a melody using its note information as a melodic motion using three simple codes ‘u’ for up, ‘d’ for down and ‘r’ for repeat. Using this system, a melody can be represented in a way that the actual singer has no real need to stay in tune, or even start singing the melody in the right note because he or she has no reference of the actual note due to the lack of use of a tuning fork for instance.

This way of representing can be a way to overcome the challenges to represent a melody in the digital domain, but it requires the information to be already represented in a digital manner, for example, encoded in a Midi[43] file. A Midi file contains the information of the actual notes. For example, if a note is played, the stored information is not the audio of the note, but the standard numeric code of the note in Midi format.

This information can later be fed to a synthesizer who will be in charge of producing the actual sound of the note. The aim of this format is to provide a way to encode music and for the musical instruments and hardware to interface with each other. Such a goal leads to an encoding that represents music in an absolute fashion. Although this format can be useful for the task at hand, there still is the step of turning audio into Midi data to be done.

Having those elements in place, a potentially valid process shown in Figure 19 can be proposed that will perform the melody recognition.

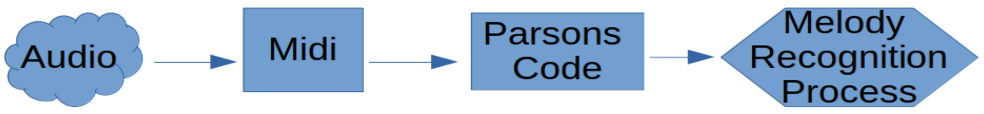


Figure 19*: One proposed melody recognition process.*
A proposed melody recognition process involves the transformation of the audio data into Midi, then encode the information using Parsons Code as a pre-processing, and afterwards using the output as input in a melody recognition process

There are libraries like Librosa[44] for Python that perform this task of converting audio to Midi, but right off the bat, the resulting Midi file produced by the processing of the audio is not accurate enough to be used as a starting point for the whole process of melody recognition to begin. Usually, in a recording, external sounds are recorded along with the recorded performance, and those noises are encoded as false positive notes, resulting in a representation that contains wrong notes, and will surely be misleading for the recognition process that will come afterwards.

This problem renders this approach impractical, as the source audio files are always likely to contain some kind of background noise, despite the effort made by the recorder to keep a quiet environment in the moment of the recording. Usually, the recordings are made in a comfortable place for the performer, like a bedroom, and some noise is inevitable.

In recent years, some interesting work has been done on different recognition efforts.

The hype we are living nowadays in AI is highly regarded to the advances in the field and fairly attributed to the positive results obtained by the application of ML techniques to various problems and the achievement of very good results.

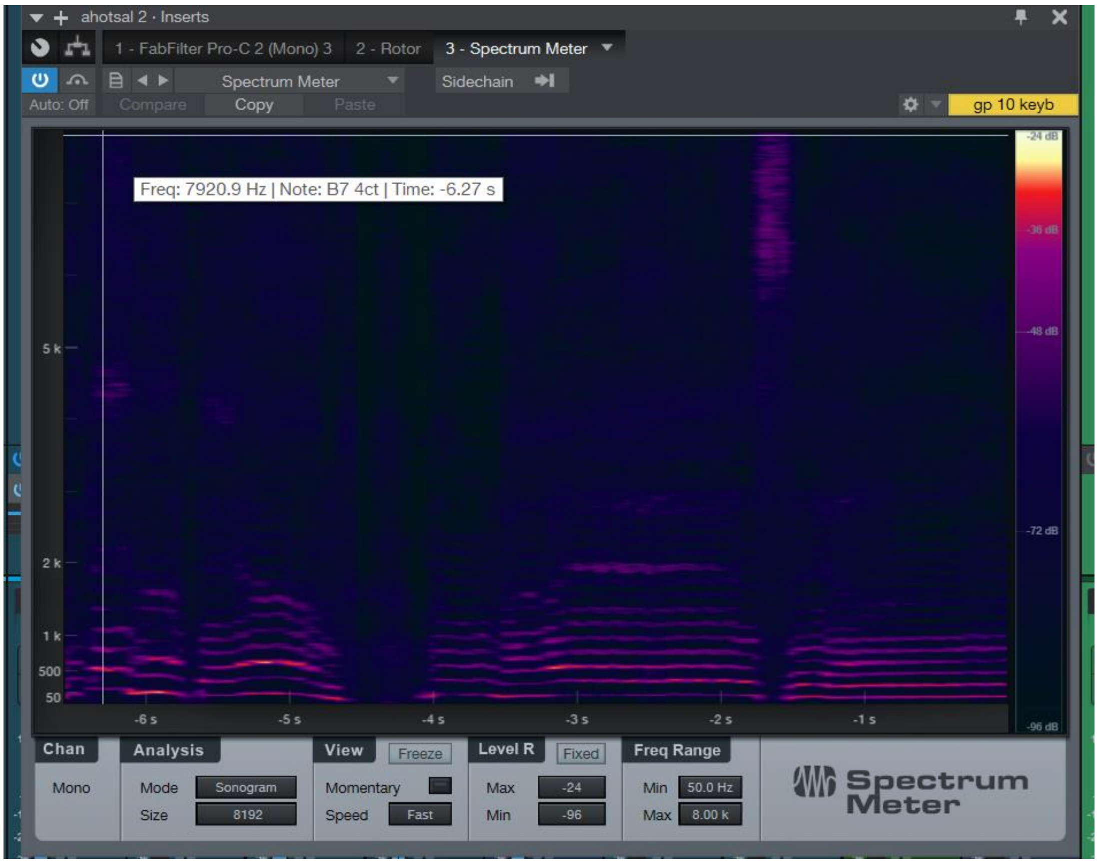
For instance, some previous work has been done on emotion recognition using spectrograms as input and Deep CNNs for processing the data with satisfactory results. Explicitly, this idea is very well put and explained as proposed by Satt A et al at[45] and as proposed by A. M. Badshah et al at[46].

Given the positive results published in those researches, it seems safe to assume beforehand that this approach should present some good results too in the problem we are trying to solve.

One common problem that pops up in cases where one tries to apply ML in a certain situation, is how the data is going to be encoded and then fed to the ML model. Many of those models are built to solve Computer Vision problems, which do not necessarily meet the format in which the data is presented. In the case at hand, audio data is not directly feedable to Computer Vision models, then some conversion and processing work has to be done before, for the ML model to understand and use it properly.

Audio can be represented in multiple ways, but the spectrogram is used in those papers so that the NN can process the resulting image. If a spectrogram can be used to recognize emotion in speech, everything leads to believe that the recognition of a melody should be a less complex task, due to the more structured nature of the information.

Let’s bring back the image of the spectrogram shown before, in Figure 20.



Freq: 7920.9 Hz | Note: B7 4ct | Time: -6.27 s

Mono
Mode Sonogram
Size 8192
Momentary
Speed Fast
Max -24
Min -96
Min 50.0 Hz
Max 8.00 k

***Figure 20**: Spectrogram taken from a human voice.*
Looking at the image of a spectrogram of a human voice, some structure is recognisable in pink colour. Those patterns are what the NNs should detect and then learn to accomplish the melody recognition objective.

Just by watching the image, some discernible patterns emerge to the naked eye, due to the duration in time of the sounds, and the harmonic content of the sounds. As the objective of Computer Vision models is to detect and recognize such patterns, everything points us again in the direction to believe that the proposed approach should be valid.

4. Preliminary attempts

The audio classification problem has been posed before in other situations, and so, some datasets are publicly available to provide researchers, some data on which their work can be based on and compared with that of others. Taking advantage of such datasets, some experiment can be performed using them.

4.1. UrbanSounds 8k dataset

This dataset is a publicly available dataset in the UrbanSounds[4] dataset website. It provides data of different sound excerpts recorded in the urban environment which are sorted in different categories and labelled accordingly. The data is categorized into 10 classes that include *air_conditioner, car_horn, children_playing, dog_bark, drilling, engine_idling, gun_shot, jackhammer, siren*, and *street_music.*

Two different datasets are available for download based on the same base data, each of them providing a different number of samples. One basic set is fashioned from the excerpts taken from freesound.org[47] on which this particular dataset is based upon, and is downloadable with 1,302 samples which can be enough for some uses. Another group of data is far bigger and manages to gather 8,732 samples. This second bigger dataset is useful where more data is needed, which can be critical in certain situations where such an amount of data is needed.

In the current research, the amount of data needed is unknown beforehand, but as the more data at our disposal, the more useful it can be, the UrbanSounds 8k is preferred.

The dataset is made available in a single compressed file for each of the options, containing as well as the necessary data, one metadata Comma Separated Value (CSV, from now on) file with the label information of each sample, alongside some general documentation on the set. The CSV files are plain text files with values separated by commas, which can be easily imported by applications as well as specific libraries, allowing this way the easy interchange of information.

The data is presented in multiple wav files in standard 16bit 44,1Khz stereo format, otherwise known as CD-quality format. This data can be easily converted to any desired format using free and open source tools, such as ffmpeg[48]. Once the data is converted to the appropriate format, the audio processing libraries available are enough to treat the samples and produce the correctly formatted data needed by the NNs to process.

No alterations were made to the dataset, other than selecting the appropriate number of samples, taking the computational resources available into account.

The licensing of the package is Creative commons 3.0 attribution non-commercial[49], which perfectly fits the needs of the conducted research.

4.2. Proof of concept on the UrbanSounds 8k dataset

Before beginning the work on the actual dataset subject of the research, some previous work was deemed to be necessary. In those first steps of the research, the idea is to really know if the method proposed to be used, is any good. As proposed by Satt A et al at[45], those methods have been proven useful in similar circumstances, but it is important to try and reproduce those results reported by researchers in similar situations, just to know that everything falls into place and that we are able to reproduce the process.

Those initial steps are very important because any mistakes in these early stages can lead to major deviations in the results in the end, so it is vital to build a good foundation for the rest of the work to rest on. The idea would be that before starting searching for a good model that gives a good performance, first check that the rest of the process is viable and that favourable results are being obtained, before taking further steps in the long and arduous path. This extra initial step is considered a way to make all the tests and checks necessary in a fast way and confirm that the approach to be followed is going in the right direction.

The public availability of the UrbanSounds 8k dataset presented before is invaluable at this moment. This data has been used to try and get favourable results on the predictions made. Such tests, let us test the steps of data processing and formatting for later processing, and to go through the necessary stages until the data is finally consumed by the ML models.

Albeit the data presented in the UrbanSounds 8k dataset is similar in nature, the problem posed by the dataset of recognizing each sound type, is slightly different to the problem that is going to be addressed with the EHU Bertso dataset, which will be presented shortly. In the UrbanSounds 8k dataset, the nature of the audio source is different each time, but in the EHU Bertso dataset in the other hand, the audio source is always of the same nature, the human voice, knowing such information, some constraints can be set that with the UrbanSounds 8k dataset do not apply. Knowing such conditions, the pre-processing of the data is much simpler with the UrbanSounds 8k dataset, and it consists only in creating the necessary spectrograms for the ML models to consume them as image data, not as sound.

By this moment, all the steps that have to be done to prepare the data, have to be crystal clear. All the audio conversion steps to format the audio and convert it to spectrograms have to be ready for testing, and the Transfer models need to be available.

After conducting the required processing, a previously presented model has been selected such as Inception V3 to try and keep a good calculation performance and has been used to test those initial assumptions. The model is used to classify the samples of the dataset into the sound categories or classes, for the efficiency of the classification to be tested and deemed as reproducible. Although the whole UrbnaSounds 8k[4] dataset is available, the actual truth when performing those tests is that the resources are limited, and so, only a fraction of 1000 samples is used in the calculations. This can affect the final results, if the model needs more data to achieve optimal results, but should be enough to show signs of learning and for the process of the pre-processing steps to be validated.

4.3. Results of the attempt

The results obtained after performing the experiments are shown in Table 1, where alongside the model, the step number where overfitting is detected, and the train and validation accuracy scored in the step before this overfitting is detected.

Model	Overfitting Step	Train accuracy in previous step correct %	Validation accuracy in previous step correct %
InceptionV3	5	70,3	73,91

***Table 1**: Test results obtained in the UrbanSounds 8K dataset with an InceptionV3.*
Those results show clear signs that the model is capable of learning the structure of the data.

After seeing those results, one thing is clear, even though we have not spent any time whatsoever fine-tuning the model, or processing the data, getting a 70,3% in the training set and a 73'91% in the validation set, is a clear indicator that the solution proposed is sound and then valid, and that we can go further down this road knowing that we are going in the right direction.

Again, these calculations can be fine-tuned and additional work can be done here, but at this stage, the objective is not to provide precise data but to test the process, by trying to get similar results as reported by other researchers. Having done that, we can say that we are safe to continue with the process.

5. Methods for data processing, approach and testing phases

5.1. The EHU Bertso dataset

The dataset on which this research is based is provided by the Euskal Herriko Unibertsitatea[2] itself. The sample data was previously gathered in a previous effort to pick up and consolidate the samples to be used in researches like this.

The dataset is formed of 311 elements, where each of them is supposed to be a person or group of people singing a single melody.

Examining the data, we see that 17 melodies are represented, but not all of them containing the same number of samples. This imbalance is a situation that has to be addressed at some point in the progress of this research.

The initial exploration of the dataset, opening and listening to each one of the files revealed that not all of the data was provided in a desirable way. Format wise, all the recordings were in an appropriate format or in one that could be easily exported into a uniform 8 bit, 8 Khz sampling rate mono wav for further processing.

This exploration showed that some of the samples were not correctly recorded. The idea of the dataset is to have one single audio file per singer and melody, and data is found, where within a single audio file, the singer recorded multiple melodies, thus rendering the audio file useless for the purpose of the research.

To overcome the issue of multiple melodies sung in one audio file, the easiest of the solutions is used. After opening the audio file with an audio editing software package such as audacity[51], only the first melody is kept, chopping the audio and thrashing the rest. More data could be obtained by carefully examining and labelling this remaining data, but the goal of this research is not the creation of this dataset, and the number of lost melodies would not suppose a significant improvement over the initial number, so this solution is deemed to be the one that best fits the needs of the moment. After such data cleaning process, the dataset ends up having 311 valid samples, all containing one single melody in each audio file, by one or multiple performers.

Another aspect that has to be noted, is that some of the recordings were provided by a group of children singing in a choir fashion. Although this is not reason enough to discard the data, it is a fact that has to be noted, based on the idea that the sum of the multiple voices may produce a different sounding audio than just a single person singing it, with a different spectral signature, due to the combination of the different voices.

Taking the number of samples into account, we can see that 311 is a very small sample count. Usually, NNs perform better the more data they are fed with. Provided this, here there is another aspect of the dataset to be improved on.

Watching the files provided, the metadata is given encoded in the names of the files. Although this is an easy and valid way to be used, some processing is made to generate a CSV file with this metadata for every single sample. This way, the post-processing of the set, to generate the necessary data splits to be fed into the NNs, can be carried out very easily using the functions provided by the libraries available for such use.

Particularly, the metadata consists of the *classId*, *filename*, *id* of the performing person, *length* of the file *in seconds*, and the *length* of the file *in minutes*. This file length is not necessary for the actual use of the data but has shown invaluable for the spotting of the inappropriately recorded data. This is the actual listing of columns:

id: id of the sample
classID: id class of the sample
class: same as the previous column, kept for compatibility reasons
fitxategia: filename of the sample
bolondresa: id of the performer
luzeraSec: length of the sample in seconds
luzeraMin: length of the sample in minutes

After this analysis of the dataset, it is clear that it poses the following concrete challenges and that need to be addressed:

1. Incorrect audio file recordings (already addressed).
2. Lack of metadata file with the corresponding data embedded in file names rather than in a CSV metadata file.
3. Number of the samples.
4. Uneven number of samples for each class.

The pending issues will be addressed in sections to come until the data is ready to be used.

5.1.1. Data conversion

As we want data to be fed to ML models it has to be properly formatted, in a way that such models will be able to comprehend.

The audio information is first converted into the uniform mono 8000Hz 8bit format. Then, some cleaning is made in the actual audio file for the EHU Bertso dataset, which will be clarified in later sections. Next, a conversion is made from the audio file into image format via spectrograms, and finally, the result is fed to the ML model. The whole process can be seen in Figure 21.

Those steps could be taken out of band processing and put into a batch like process where the actual formatting and filtering will happen. This is done that way to lower the processing power needed, and thus the time needed, at the moment of handling all the data. As many models will be tested, the total running time of the calculations will be much lower using this simple technique.

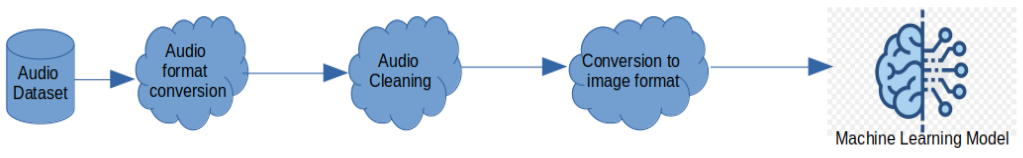


***Figure 21**: The data conversion process used is displayed.*
The audio data is converted into a uniform audio format. After that, an audio cleaning step is taken to try to get rid of information that is not useful before the conversion to image format happens. After that, the data is ready to be forwarded to the ML model.

5.1.2. Audio filtering

Talking about the perception of sound, the human ear has a frequency range limit in which it perceives it. Perception of sound doesn't work in the whole spectrum of frequencies but in a specific region of it. It is commonly assumed that this range is from 20Hz to 20KHz. This range is generously set as the older we get, some frequencies get lost in a natural way, as an effect of the ageing process. We could say that a young child can hear more frequencies or better than an old person.

Those hearing limits are taken into account by audio encoding algorithms, that do not encode anything outside those frequency limits. This technique allows them to save space and to avoid useless work in the process.

To all of this, we can add the composition of sound explained so that further refining can be made. The human voice creates sound within the limits of the human hearing, so it can be heard by another individual, and interpreted. It happens that the natural sound produced by human voice, as all the natural sounds, is composed of the fundamental and its harmonics. We said that the fundamental is the lowest frequency of a waveform, thus it contains the tonal information needed that characterizes the sound, and we could argue that the harmonic information is actually redundant, given its composition. As those harmonics are built multiplying a positive number by the fundamental, we could say that no extra information is given and that the necessary information could be derived from the fundamental.

This phenomenon is illustrated in Figure 22, which shows the frequency information over time. The amplitude of the sound is represented with lighter colours as opposed to darker green for the lowest amplitude regions.

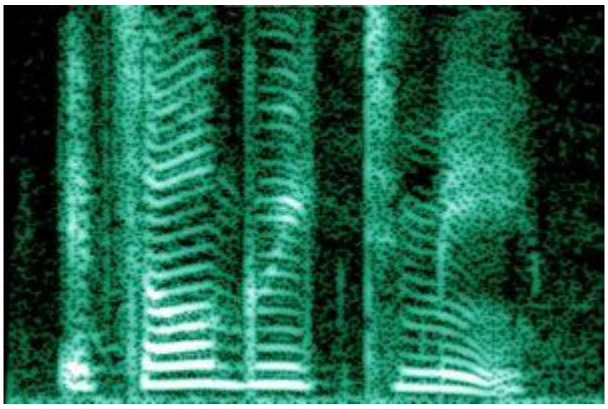


***Figure 22**: Spectrogram of a human voice.*
Looking at the spectrogram of an audio of a human voice, knowing the structure of the composition of the wave some safe assumptions can be made to help and focus the data in the area of interest.
Source:
https://es.m.wikipedia.org/wiki/Archivo:Human_voice_spectrogram.jpg

In Figure 22, we can see how the fundamental frequency of the sound is shown in the bottom of the image (lowest frequency), and repetitions of it appear higher in the y axis of the image (higher frequencies).

Knowing that, setting the limits of the fundamental frequency of the human voice could help reduce the space where the useful information of a sound is held. This limit is set from 85 to 180 Hz for adult males, and of 165 to 255 Hz for females as read in Human voice in Wikipedia[52]. Thus, a safe assumption can be adopted by saying that human voice fundamentals lay somewhere between 85 to 255 Hz.

Audio filters are the tools that audio processing has for filtering the audio signal, leaving out certain parts of the signal and selecting others. Usually, those tools are implemented in the various libraries that are available for audio processing in different languages, as this kind of work can be carried out in an easy and scripted way, to batch process large amounts of information leveraging this way the requirements in runtime.

Combining those audio filters with the spectrogram representation of the audio, a focused image of the audio can be generated, leaving out the frequency spectrum parts where we know that the interesting information needed is not present, and setting our attention in the area where the interesting information dwells. Then this smaller and more focused part of the spectrogram can be used as input for the image processing NNs to process.

Examples of how audio filtering occurs are widely available on the internet, and in Figure 23, two commonly used filters are displayed, showing how each audio filtering affects the signal cleaning it. Those filters, are High Pass and Low Pass Filters.

A High Pass Filter filters out the information below a set threshold, and lets the information above it to pass, as can be inferred from the name.

On the other hand, a Low Pass Filter performs the contrary operation. Filters out the data above a given threshold letting pass whatever comes below that limit.

Note that combining both filters, a filter can be created that lets pass information within any given low and high limits, keeping anything in between. This tool is very easy to build and extremely useful for audio processing in all of its variants, as it lets the user filter out any information that is deemed redundant or just plain noise that does not provide with any additional value.

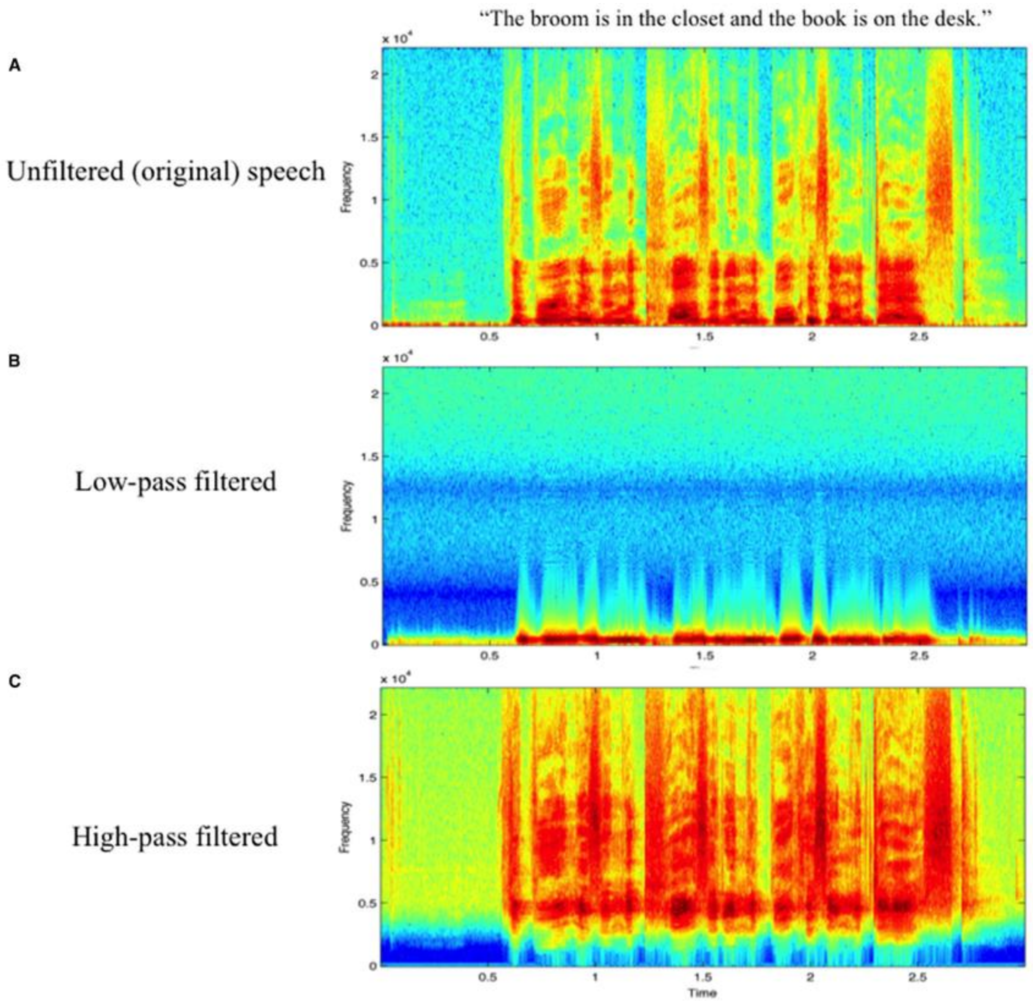


***Figure 23**: Result of high and low pass filters.*
The result of low pass and high pass filters a spectrogram reveal themselves as invaluable audio processing tools, to implement the narrowing of the frequency spectrum to the field of interest in the frequency spectrum.
Spectrogram A shows the signal unfiltered, B is the result after applying a low pass filter, and C is the result of a high pass filter applied to the original signal.
Source: https://www.frontiersin.org/articles/10.3389/fpsyg.2015.01340/full

By doing a basic analysis of human voice, sound and its composition, we are able to focus and reduce the space where the information we are interested in will be held, narrowing the field from 20 Hz – 20 Khz, to 85 Hz - 255 Hz, focusing the field in 0,85 % of the complete audible frequency spectrum, setting aside the remaining 99,15 %, which may contain redundant or no data.

5.1.3. Dataset preparation

The analysis of the dataset has provided us with the insight of knowing which aspects work has to be done to meet the requirements for nowadays standards and proper handling of the files by the contemporary audio and ML libraries.

5.1.3.1. File recordings

First of all, we already addressed the issue of incorrect audio file recordings, by leaving out the parts that were not correctly recorded and labelled, which has left us with 311 correctly named samples.

5.1.3.2. Metadata file

The next issue we would have to give a solution to, is the “Lack of metadata file”. Each file is correctly labelled, using the following naming convention. Each file is named after the melody (class) it represents, and the id of the volunteer who performs the melody in the recording, for tracking purposes, but maintaining the privacy of the person, as no information that can lead to the identification of the individual has been given away.

The file name is composed using the following pattern n-uxxx.wav.

n: melody or class
xxx: code for the anonymous volunteer.

This can be seen in Figure 24.

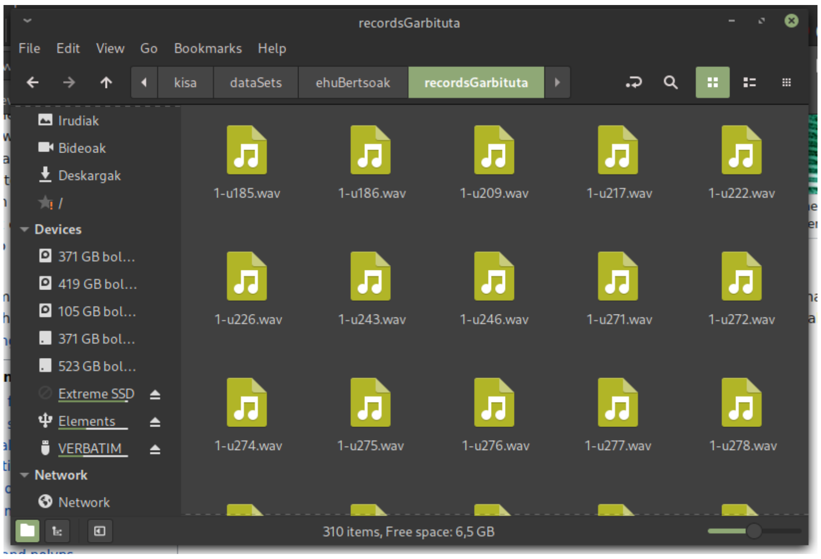


***Figure 24**: Metadata encoded in file names.*
In the original dataset, metadata is embedded in the file names. Although it is a valid encoding system, the generation of a metadata file with all this information is recommended further processing afterwards.

Using this encoding system to read the meta-information, a metadata file can be trivially created, which will be redistributed alongside the rest of the data. CSV handling libraries make it easy to get past this step. In order to detect needlessly long files, some extra data is captured from the files themselves, like the duration of the audio file in seconds and minutes. The class and volunteer information are basic, and actually, the only information is needed for the learning process to occur. The name of the file is also saved in the metadata file.

This is the relation of fields used.

· classID and class fields: The melody or class id.
· fitxategia: Thefile name to look for in the dataset to which corresponds the data to.
· bolondresa: The id of the volunteer performing the singing.
· luzeraSec: Length of the audio file in seconds
· luzeraMin: Length of the audio file in minutes

All this information is compiled in a CSV file, which is one of the most used formats for loading this kind of data.

Another issue with the original dataset is that the metadata does not provide any information about the melodies. They are just labelled with a classId, this is enough for the research, but it is very interesting to have those melodies identified and their real name accounted for and stored, for reference purposes.

To add this information in the dataset, another CSV file is used named "ehuBertso-doinuak.csv". The information stored in the file will only hold the relation between the classId with the melody name, as that's the only thing needed for linking the classId already present in the first CSV file with the melody name, in this last file. The relation of melodies and their classIDs is shown in Table 2.

classID	doinua
1	Aita izena kanta beharrak
2	Antton eta Maria
3	Behin batian Loiolan
4	Betroiarena
5	Haizeak bidali du
6	Ikusi nuenean
7	Insumisoarena
8	Iparragirre habila dela
9	Gitarra zahartxo bat det
10	Triste bizi naiz eta
11	Langile baten seme
12	Loreak udan ihintza bezala
13	Aizak hi mutil mainontzi
14	Mendian gora haritza
15	Mutil koxkor bat itsuaurreko
16	Norteko ferrokarrila
17	Xarmangarria zera

***Table 2**: Class melody relation.*
The relation between the classId and the melody name is handled by the file ehuBertso-doinuak.csv . Having this relation stored in such file eases the identification of each melody using the name instead of the less intuitive id.

In addition, aiming to provide a uniform processing format, 8000 Hz sample rate, 8bit depth Mono format is selected, which is enough for the needs of the research conducted.

5.1.4. Number of samples

As the preliminary exploration of the exposed dataset, the number of data is quite low, therefore this is a known problem that has to be addressed before going any further.

5.1.4.1. Data augmentation

As we already have stated before, NNs tend to perform better the more data they are provided with, but there’s also the constraint of the resources needed for the learning process to occur. As the used memory and CPU resources are limited, the challenge is to find the sweet spot where the learning can occur, and the resources needed do not exceed the ones available.

Talking about the “Number of samples” issue, when there are not enough samples for whichever reason, and there is a need to get hold of more, instead of just trying to collect more, which sometimes is not even possible. There is a number of techniques that can be applied to come up with some new samples, always based on the instances that already are there.

Usually, this data generation implies that for each sample available, we can apply some kind of transformation that will generate a new sample. That way the initial dataset can be expanded by a significant number of new samples, and this new expanded dataset can be then used by the ML model to learn better due to the more suitable figures that go with the learning process.

There are different techniques for new sample generation, those techniques usually are specific to the nature of the data, meaning if we have audio data, those techniques will be applied in a different way than when we have image data at our disposal.

The first thing we can do with audio is some noise generation. We can generate some noise in form of a random number, and then add this noise to each sample audio file. By performing this step, the new sample will incorporate some background noise and the original sample will be transformed into a new file.

In the images below we can see side by side, how noise addition affects the audio file.

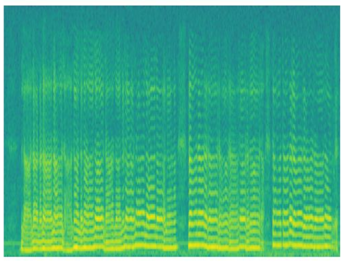


***Figure 25**: Original unaltered spectrogram.*

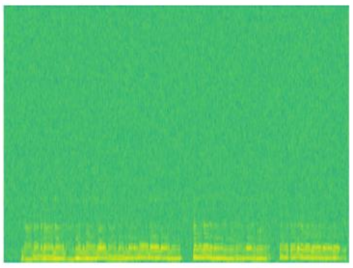


***Figure 26**: Spectrogram after the noise addition process for comparison. The green colour and lighter colours are the result of added noise energy in all the frequencies*

In Figure 25, the original audio is displayed, and the noise added altered audio can be seen in Figure 26. The greener colour of the image is because of the noise. The audio gets more noise information data that makes the overall audio with more energy on all the frequencies, and so, greener.

Next, we can shift time. This means that we can add some silence at the beginning of the file that will make the actual audio start later in time, even if the audio content will be the same after that silence is finished. We can even create a set of different time displacements, that will be applied each separately, and then generate a new file for each one of them. The optimal option is to use positive displacements, just to be sure that no audio data is lost because of the processing.

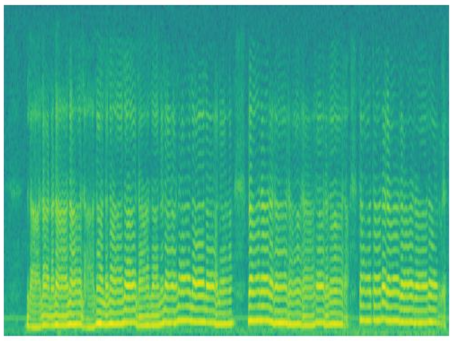


Figure 27*: Original unaltered spectrogram.*

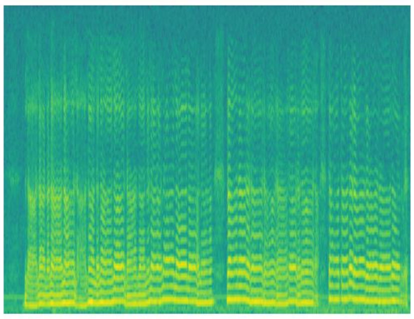


Figure 28*: Spectrogram after the time-shifting process.*
Even though the time-shifting has occurred it is hardly noticeable due to the small amount of time used for the shifting process of some milliseconds

Again, we can compare side by side the effect, but this time around, we would have to zoom in very close to the image for the effect to be noticeable because the shift in time is very small, just some milliseconds, but certainly, it’s there. We have the unaltered spectrogram in Figure 27 and the time-shifted version in Figure 28.

Another transformation can be achieved by manipulating the pitch. Audio processing libraries can shift the actual pitch of a melody altering it by a number of half steps. Although this transformation is valid, usually the alteration of the pitch by more than 5 half steps, or two and a half steps distort the original sound too much, and so it is not recommended, because the human ear can really tell the difference. Although this can be even an advantage in the case at hand, we will play it safe and keep it within the two and a half step limits.

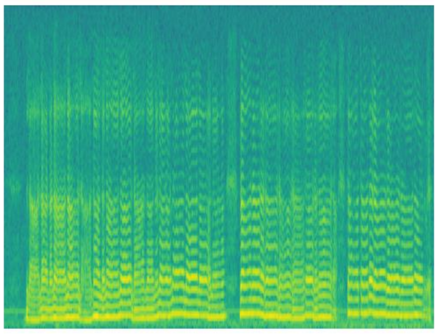


***Figure 29**: Original unaltered spectrogram.*

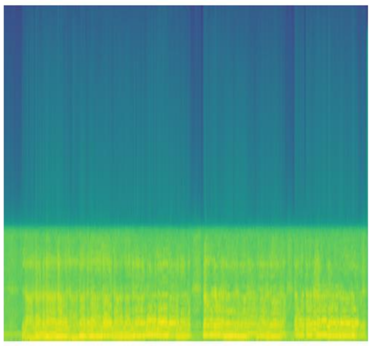


***Figure 30**: Filtered and pitch-shifted spectrogram.*
The more uniform and sleeker shape of the information is the result of the filtering. On top of that the pitch-shifting can be noticed if we pay attention to the lowest fundamental in the unaltered spectrogram and see how it has moved to the slightly lower frequencies

For this final comparison, we again have the original signal in Figure 29 and the pitch-shifted transformed spectrograms in Figure 30 to compare. This time around, the final audio file is a more refined one yet, not only the pitch-shifting transformation has been applied, but the filtering has been done too. The audio filtering that has been done to clean the signal is responsible for the blue tone of the upper part of the audio. This is caused by the lack of information and data in that segment because it has been filtered out. Returning to the pitch-shifting transformation, if we take a closer look at the lowest energy accumulation in the original audio (that would be the fundamental of the note sung by the performer), and compare it with the one in the processed file, we can see how this fundamental is lower. After seeing that change, we would argue that a pitch-shifting has been performed and that that shifting has lowered the signal.

One important thing to notice is that all those ways to come up with new data can be easily combined, just as we have just shown, therefore we can, for example, create a new audio file that adds some noise, delays the beginning of the song by 30 milliseconds and then shifts the pitch down by half a step.

This whole Data Augmentation process can result in a considerable expansion of the dataset, but again we have to remember that resources are limited and that we will not be able to load all the data we would want in memory

5.1.5. Uneven number of samples per class

The last problem to be addressed is the “Uneven number of samples for each class”.

For the learning process to occur appropriately, the number of samples per class has to be balanced. If we have a substantially bigger number of samples in some classes than others, the ML model will learn better some classes than the others, as it responds better to this higher number of samples.

This is a common problem in real-world datasets, as usually they hold some kind of unbalance in them. Libraries exist to sort the issue out and make things right. The algorithms use the larger number of samples and for example, can even things out by simply dropping samples in the classes which have the most number of data, and keeping the same number of samples in all of the classes, thus setting the minimum number of data among all the classes as the minimum number of data that all the classes have.

5.2. Approach

Applying ML techniques involves a reformulation of the problem in a way that the data can be processed by the model, especially when the model to use has been thought and proven useful in another situation, which is different from the actual one.

If a Computer Vision model has to be used to process audio, the audio data needs to be expressed visually. In the audio domain, this can be achieved using spectrograms which provide a visual representation of the energy in the frequency spectrum over time. This image representation is enough to bridge the gap between the input audio data and the image processing nature of the models intended to be used. This way the proven efficiency of the Computer Vision models can be tapped into, to solve a problem whose initial statement differed from the Computer Vision field.

In Figure 31, we can see a spectrogram where we can appreciate what we just said.

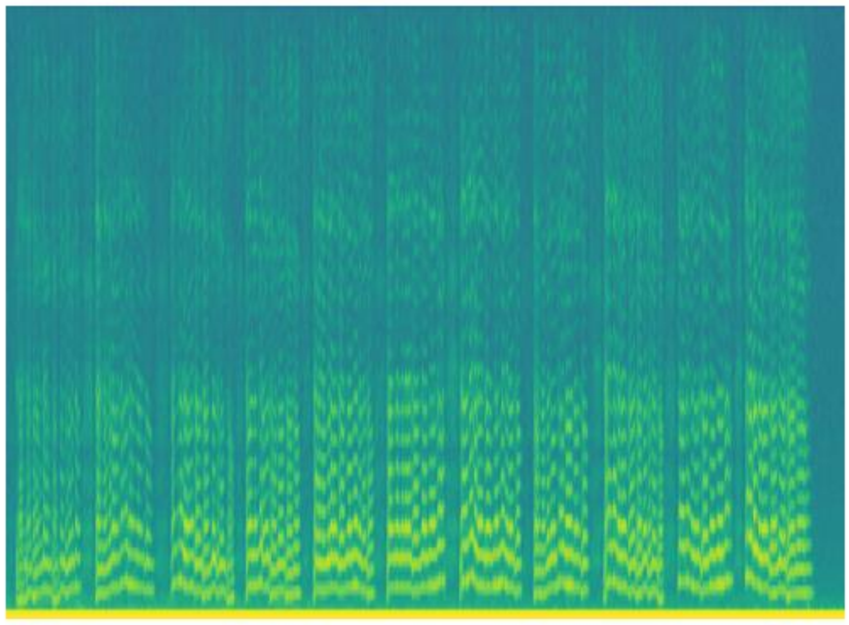


Figure 31: *Example spectrogram of melody.*
If an audio file can be transformed into image data, then Computer Vision approaches can be used to extract and learn the structure of the audio data. Spectrograms are used to bridge the gap between the original data format and the visual data processing nature of the Computer Vision models intended to be used.

Knowing the data, can provide a way to refine and focus the input, and thus increase the accuracy of the learning process, avoiding to learn noise data patterns which give no additional information to help solve the problem. Given the nature of the input data, which is a human voice singing a melody, knowing the limits of the fundamental frequency of the spoken voice (from 85Hz to 255Hz), gives the chance to apply an audio filter and leave out data that only gives redundant information concerning the data filtered in, and reducing the field of data to learn in a significant way.

The melodies to be processed and over which the lyrics are improvised are not usually long, therefore the need of splitting the audio into several chunks is not needed, and each audio file can be fed directly to the model.

The original set consisted of raw data that was gathered from people singing some melodies. But the rawness of the data included the misuse of the technologies for whatever reasons, that even though the users might be masters in the craft of Bertsos, they were not able to provide the data properly. Provided we have the need to have data correctly represented in the set, this led to the manual processing of the original dataset consisting of 310 samples, to be properly formatted. This processing mostly included the task of cutting the audio including multiple melodies in a single file, into a single file with a single melody in each one, and labelling the file accordingly.

The limited size of the dataset has led to limited accuracy in the results. Although some promising results have been achieved, showing the process to be valid, the problem at hand did not get 90 or above accuracy. Those results shown by the model statistics suggested that more data is needed, and as those ML models tend to perform better with the more data they are fed with, a Data Augmentation process has been executed on the original dataset. This process consisted in several modifications made to each of the files explained before, that in the end resulted in 16 extra files obtained from each one of them, and raising the dataset sample count to 4960, and providing richer input for the models to consume.

Those modifications performed included three kinds of different processing, involving noise addition, time-shifting and pitch-shifting. Trying to keep the distortion of the original audio low, 3 time-shifting and noise factors were used, alongside 10 different pitch-shifts that do not vary more than 5 steps from the root of the original note. The different combinations of each of the transformations suggest that a much bigger dataset can be achieved of almost 14.000 samples, or even bigger, but such size is not applicable in the current situation as the computational requirements to use the data would have to be increased, both CPU and memory wise, and the resources are limited.

5.3. Some words on the challenges of the problem

Regarding this melody or sound classification issue, some work has been done using the UrbanSounds 8k dataset before. This classification problem, although challenging enough, is less demanding than the one at hand. Classes in the UrbanSounds 8k dataset are different in harmonic structure, children playing and a jackhammer sound different because their harmonic structures are particular. If we were to perform a spectral analysis of each of those sounds, we would see that tonally they are distinct, one is composed by human voices, and the other is created by a sounding motor and the collision of the metal with the ground, then the structure of the sound itself is different in nature, and the myriad of harmonics present in each sound vary completely from one to the other.

The problem of classification of melodies is different because all the input provided comes from various people singing, but human voices after all, with a more common and uniform tonal structure, as the variance in the input data is more subtle. These more uniform input data calls for more refined classification techniques such as the use of Transfer Models, in order to get a meaningful result.

The problem found in this situation is a common classification problem, which in this case has to classify melodies to a given base class. This poses an interesting challenge for these tunes or melodies are old and they often come from different sources, varying in many ways from one singer to another.

One such way is that a single melody can vary in the way of singing, provided it comes from a different geographical location, for geographical distance leads to significant micro changes that are now embedded in the way different people sing those tunes.

Given the main objective of a Bertso is the improvisation of the actual lyrics, not the melody itself, each Bertsolari tends to use rhythm to his or her own advantage, although not changing its rhythmic structure, adding pauses and changing the speed at will, using the time to come up with the next lines in the Bertso. This adds another layer of complexity to the classification problem.

The tuning problem is another hard one, due to the lack of accuracy provided by many Bertsolaris. As stated before, the main objective of the Bertsolari is to improvise the lyrics sung, providing them with meaning, and often a comical touch, but the melody and good intonation itself falls out of this arena. This has resulted in many Bertsolaris although being top-notch improvisers, not having enough musical background to sing properly and accurately.

A common way to overcome these differences and challenges is using ML type classification. Using this method, the system should be able to learn the variations in the tuning, timing or even structure, mapping the input data to a correct melody class.

5.4. Technological framework

In the development of Machine Learning, a programming language that has developed hand in hand with it in the last years is Python. It is an interpreted language that stresses legibility, is high level and very easy to learn and use. Along with those features, a vast library ecosystem is available for almost any work to be done.

No stranger to audio processing with libraries like Librosa, Python has excellent tools for ML tasks, and also different ML frameworks like keras[23], have their Python implementation.

Python also has support for notebooks through Jupyter notebooks[53]. This technology enables text and code to be in one single file, combining in an easy self-documented file, everything that is needed to develop a ML project.

Also, this kind of notebook format, lets users share their work in a very easy way because everything needed is embedded inside the notebook.

Keras[23] is a ML Api designed with simplicity of usage in mind, that let the user develop fast and powerful works without the hassle of other frameworks, which makes it ideal for fast prototyping and high level developing. It provides access to all the power of the library Tensorflow[54] with a high-level approach and ease of use.

Another development in this area is what Google colab[55] proposes. Jupyter notebooks[53] have all the code and documentation a ML project can need, but there is always the catch of the machine and storage needed to run it.

Google colab[55] offers just that, a way that using virtual machines running in Google's cloud infrastructure where all the required software is installed and ready to work. Storage wise, Google colab[55] can connect to Google drive[56], where all the data can be saved and accessed.

The virtual machine provided by Google Colab[55] in its free version changes depending on the resources needed by Google at the moment, but it has kept consistent during the development of this work.

Those are the characteristics:

12 GB Mem
CPU 2 Intel Xeon Proc 2,3Ghz
GPU Acceleration: Nvidia K80s, T4s, P4s and P100s
  (Automatically assigned T4, and not selectable in the free version)
100 GB HardDrive
15GB of external storage using the access to Google Drive

All these features, render the combination of technologies ideal for the job at hand. They provide a simple infrastructure where nothing has to be maintained but the actual data, code and documentation of the project. Everything is accessible remotely and can be easily shared.

Also, the programming language and libraries are widely available and everything is ready to work.

5.5. The testing method proposed

As a grid search can easily snowball and result in a humongous amount of computational resource requirements, the process is designed to keep the computational needs at bay.

The proposed procedure consists of three different testing phases. The first one to determine which of the base models gets the best results, a second phase to determine if training the base model improves the performance of the models selected, and a third one that tries to fine-tune these models in order to get better results.

With the goal to get a feel of which models are more useful to solve the current problem, the initial round of tests is carried out, involving different Transfer Models. The idea behind this first round of tests is to set the focus in the best performing models setting aside the ones that perform poorly, using a grid search to obtain the data.

The rankings of the models are generated using 3 executions of each model, to get a more accurate average accuracy than just a single one.

The first 3 models of the ranking are selected as the best candidates for the next rounds.

Talking about the number of samples used in the learning process and how the data is split into different sets, the optimal overall sample count has turned to be 1445 using 80% for the training and the rest for validation, 289. 20% of which (57) are used for testing purposes. This number of samples has turned out to be suitable one that could be used without having memory issues with the virtual machine assigned. This data split has been consistent throughout all the process.

6. Experiments and results

In this chapter, we will present the actual process followed to make the calculations and the results obtained. As we stated before, there will be three different rounds and depending on the results the decisions taken will be different.

6.1. Round results

6.1.1. 1st round

For the first round of tests the models involved are the following:

· VGG16
· VGG19
· ResNet50
· ResNet101
· ResNet152
· ResNet50V2
· ResNet101V2
· ResNet152V2
· MobileNet
· MobileNetV2
· NasNetMobile
· InceptionV3
· InceptionResnetV2
· Xception
· DenseNet121
· DenseNet169
· Densenet201

After following the steps for each of the models using the augmented dataset, the presented results were gathered. Using a set of 1445 samples, the input data is split in the following fashion.

80% of the data is used for training. 1156 samples
20% for validation. 289 samples
20% out of the validation, set is used for testing. 57 samples

The figures shown are taken from the test accuracy obtained after performing 3 iterations of the calculations and averaging out the results. The results are sorted out by their accuracy.

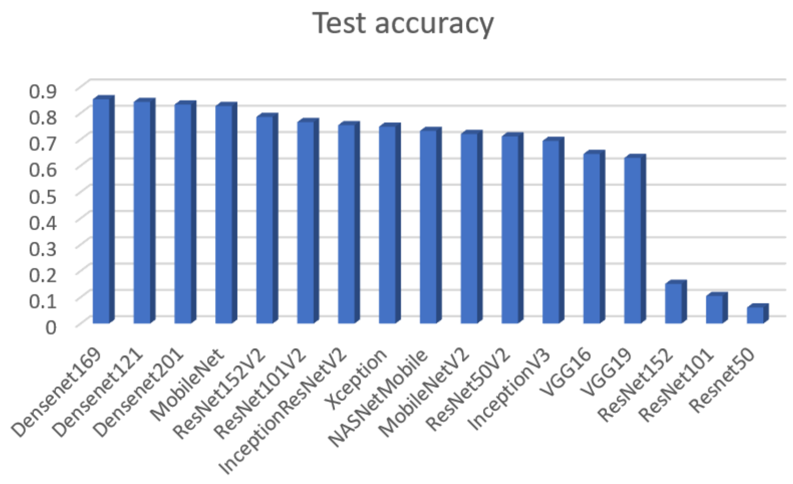


***Figure 32**: Test accuracy graph.*

DenseNets get the best test accuracy score and they are declared the winning models. We can see this in Figure 32.

Model	Train Accuracy	Validation accuracy	Test accuracy
Densenet169	**0,6576479077**	**0,5833333333**	**0,8529411765**
Densenet121	**0,628427128**	**0,561781625**	**0,8425605536**
Densenet201	**0,7265512347**	**0,5459770163**	**0,8327566321**
MobileNet	0,6655844053	0,6106321812	0,8269896194
ResNet152V2	0,7572150032	0,5847701033	0,785467128
ResNet101V2	0,662698408	0,5675287247	0,7658592849
InceptionResNetV2	0,5306637883	0,4181034466	0,7543252595
Xception	0,5840548277	0,4899425407	0,7485582468
NASNetMobile	0,5692640642	0,4597701132	0,7324106113
MobileNetV2	0,6471861402	0,5402298768	0,7208765859
ResNet50V2	0,6056998571	0,5143678188	0,7116493656
InceptionV3	0,455988457	0,366379311	0,694348328
VGG16	0,2651515106	0,2543103447	0,6447520185
VGG19	0,2792207897	0,2543103496	0,6297577855
ResNet152	0,07287157327	0,04597701132	0,1510957324
ResNet101	0,07864358028	0,05172413836	0,1049596309
Resnet50	0,06240981321	0,06465517109	0,06113033449

***Table 3**: Full ranking of the 1st round of calculations.*
The variants of DenseNets are in the lead of the classification outscoring the rest of the models in the round.

As the classification is presented in Table 3, and Figure 32, the best three models have shown to be Densenet169, Densenet121 and Densenet201. Table 4 is the result table showing only those three model results for easier reading.

Model	Test accuracy
Densenet169	0,8529411765
Densenet121	0,8425605536
Densenet201	0,8327566321

***Table 4**: Best 3 results of the 1st round.*
For better reading, the best 3 results of the 1st round are selected and displayed

Getting quite similar results, the Densenet169 model outperforms the other two but not in a clear way. All of the three models perform well, although there still is room to improve.

6.1.2. 2nd round

The second round of tests uses the best 3 models selected in the first phase to try to train the whole base model instead of using it with the weights untouched, just the way they are after the training using the Imagenet dataset[22]. This gives us a taste of what might get better results.

After training the whole models for all three options remaining from the 1st round, in Table 5 are the results obtained.

Model	Train accuracy	Validation accuracy	Test accuracy
DenseNet 121	0,1024531027	0,05028735598	0,6297577855
Densenet 201	0,06746031716	0,05747126415	0,2848904268
DenseNet 169	0,07070707281	0,06321839243	0,2514417532

***Table 5**: The results of the 2nd round are displayed.*
The expected results were to improve over the results of the 1st round, but worse numbers were obtained instead.

Looking at the numbers we got in those calculations, we can see that contrary to what we expected, the results obtained are worse than the ones of the 1st round.

Knowing this, instead of using the results from this 2nd round to go to the next one, the way to go is to actually use the models resulting from the 1st round and try to enhance them in the 3rd round.

6.1.3. 3rd round

Finally, some fine-tuning has been tested to try to enhance performance. This fine-tuning phase involved the search for the best values for parameters like the number of dense layers and dropouts values applied. The calculations for this round will be made using the best 3 models from the first round.

Getting the best 5 averaged results for each model Table 6 is arranged.

Base model	No of dense layers	Dropout	Train accuracy	Validation accuracy	Test accuracy
DenseNet 169	1	0,2	0,771284262339	0,665229876836	0,860438292964
DenseNet 169	2	0,4	0,697330454985	0,616379320621	0,859284890427
DenseNet 169	3	0	0,678571403027	0,604064027468	0,859284890427
DenseNet 169	2	0	0,568542569876	0,510057479143	0,856978085352
DenseNet 169	1	0,6	0,761904756228	0,660919527213	0,853517877739
DenseNet 121	3	0	0,484487722317	0,481321841478	0,848904267589
DenseNet 121	1	0,2	0,572150091330	0,564655184746	0,845444059977
DenseNet 121	1	0,6	0,651154398918	0,599137922128	0,845444059977
DenseNet 121	2	0,4	0,583333343267	0,524425278107	0,845444059977
DenseNet 121	2	0	0,653679639101	0,550287355979	0,844290657439
Densenet201	2	0,4	0,645021637281	0,568965514501	0,844290657439
Densenet201	1	0,6	0,796897570292	0,668103456497	0,843137254902
Densenet201	2	0,2	0,629509389400	0,573275874058	0,841983852364
Densenet201	1	0,2	0,731601715088	0,639367818832	0,841983852364
Densenet201	2	0	0,607142845790	0,564655164878	0,836216839677

***Table 6**: Best 5 results for each model in the 3rd round.*
In the 3rd round, the best 5 results for each model are gathered along with their configurations to see where those results lead us.

Looking at those figures, we can set a ranking even between those three models, as seen in Table 7:

Rank	Model	Test Accuracy
1	DenseNet169	0,860438292964
2	DenseNet121	0,84890426758
3	DenseNet201	0,8442906574

Table 7*: The ranking of the 3rd round taking the models into account. Results for each model are aggregated and ranked. DenseNet169 gets the best result overall before DenseNet121 closely followed by DenseNet201*

Although Table 7 can be arranged from the results, the best results obtained overall are the ones that have come out of the DenseNet169 model with different configurations.

The best results obtained in the whole process are shown in Table 8.

Rank	Model	No of dense layers	Dropout	Test Accuracy
1	DenseNet169	1	0,2	0,860438292964
2	DenseNet169	2	0,4	0,859284890427
3	DenseNet169	3	0	0,859284890427

Table 8*: Best 3 results of the 3rd round overall.*
If we take the best 3 results of the 3rd round overall, DenseNet in its 169 variant with different configurations on its dense layer and dropouts get the best results. In particular, DenseNet169 with a single dense layer and using a 0,2 dropout value is the best of all the configurations tested

Those results leave us knowing that the best performing base model to solve this problem of melody recognition is the DenseNet169 and that the best results are obtained using one dense layer with a dropout value of 0,2.

6.2. Result analysis

At this point, it might be interesting to look at the results obtained by the best model. As we stated earlier, trying to get a more stable figure, each score is calculated averaging three results for each configuration.

The best accuracy is gotten using DenseNet169 with 1 dense layer and using a dropout value of 0,2. In Table 9 we display the results obtained in the 3 calculations run out of which the average value has been calculated for the full ranking we already displayed.

Base model	No of dense layers	Drop out	Epo chs	Train accuracy	Train loss	Validation accuracy	Validation loss	Test accuracy
Densenet169	1	0,2	2	0,7564935088	0,9250562191	0,6379310489	1,921296716	0,8512110727
	1	0,2	2	0,7521644831	0,9618831873	0,663793087	1,33889699	0,8650519031
	1	0,2	2	0,8051947951	0,6666011214	0,6939654946	1,518410683	0,8650519031
	Avera ges:			0,771284262339		0,665229876836		0,860438292964

***Table 9**: Break down of the best calculation.*
This break down shows the values and the configurations that got the best results in the 3 rounds of calculations conducted.

With the intention to show the specific values, we will examine one of those results to see which exact values appear and analyse which classes are the most difficult to classify correctly. Even though we will focus on a single result at this moment, the full results are available in the section A)b) of the Appendix A chapter. The whole classification result obtained is also available in the Appendix A chapter.

Watching the scores that we have for those three calculations, we would pick the middle value of the three available, but as two of them are equal we will choose one of those, the first one of them for instance.

The accuracy percentage says that 86’5 % of the classes are correctly classified. 250 out of 289 samples have the correct label assigned, and 39 are not properly labelled.

Having the results, we can draw the confusion matrix shown in Figure 33 for this to see which of those classes are easier or harder to classify.

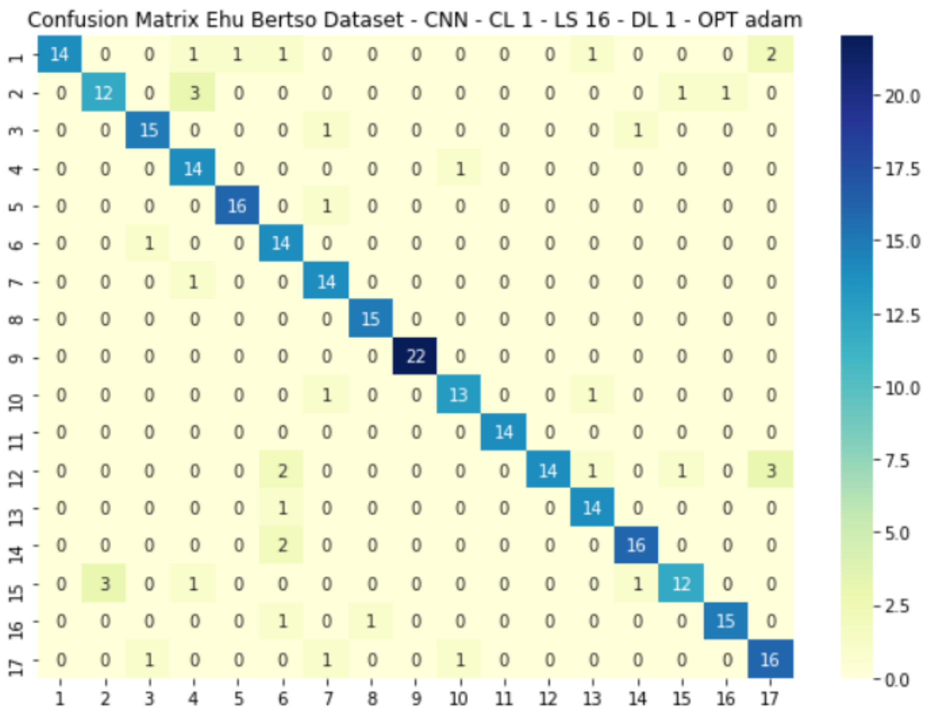


***Figure 33**: Confusion matrix of the calculation that got 86’5%.*

Based on the confusion matrix we can arrange Table 10 with the number of errors that are accounted for, for each class.

Melody	Class	Successes	Errors	% Success	% Error
Aita izena kanta beharrak	1	14	6	70	30
Antton eta Maria	2	12	5	70.58823529	29.41176471
Behin batian Loiolan	3	15	2	88.23529412	11.76470588
Betroiarena	4	14	1	93.33333333	6.666666667
Haizeak bidali du	5	16	1	94.11764706	5.882352941
Ikusi nuenean	6	14	1	93.33333333	6.666666667
Insumisoarena	7	14	1	93.33333333	6.666666667
Iparragirre habila dela	8	15	0	100	0
Gitarra zahartxo bat det	9	22	0	100	0
Triste bizi naiz eta	10	13	2	86.66666667	13.33333333
Langile baten seme	11	14	0	100	0
Loreak udan ihintza bezala	12	14	7	66.66666667	33.33333333
Aizak hi mutil mainontzi	13	14	1	93.33333333	6.666666667
Mendian gora haritza	14	16	2	88.88888889	11.11111111
Mutil koxkor bat itsuaurreko	15	12	5	70.58823529	29.41176471
Norteko ferrokarrila	16	15	2	88.23529412	11.76470588
Xarmangarria zera	17	16	3	84.21052632	15.78947368

***Table 10**: Success and error calculation from confusion matrix for the calculation that got 86%.*

Examining the numbers we can say that melodies “Iparragirre habila dela”, ” Gitarra zahartxo bat det” and “Langile baten seme” are the easiest to classify because a perfect score is obtained without misclassifying any of the samples.

On the other hand, melody “Aita izana kanta beharrak”, “Antton eta Maria”, “Loreak udan ihintza bezala” and “Mutil koxkor bat itsuaurreko” are the most difficult ones to label, scoring 70, 70.58, 66.6 and 70.58 respectively.

The “Loreak udan ihintza bezala” is often confused with the “Xarmangarria zera” and “Ikusi nuenean”, and sometimes with the “Mutil koxkor bat itsuaurreko” and “Aizak hi mutil mainontzi”.

“Aita izena kanta beharrak” is mostly confused with the “Xarmangarria zera”, and other times with the “Betroiarena”, “Haizeak bidali du”,” Ikusi nuenean” and “Aizak hi mutil mainontzi”.

The “Antton eta Maria” is often confused with the “Betroiarena” and less frequently with “Mutil koxkor bat itsuaurreko” and “Norteko ferrokarrila”.

“Mutil koxkor bat itsuaurreko” often confused with “Antton eta Maria” and less frequently with “Betroiarena” and “Mendian gora haritza”.

Seeing this, ” Loreak udan ihintza bezala” seems to be the most difficult to label, as it gets easily confused with “Ikusi nuenean” and “Xarmangarria zera”, and some other occasionally.

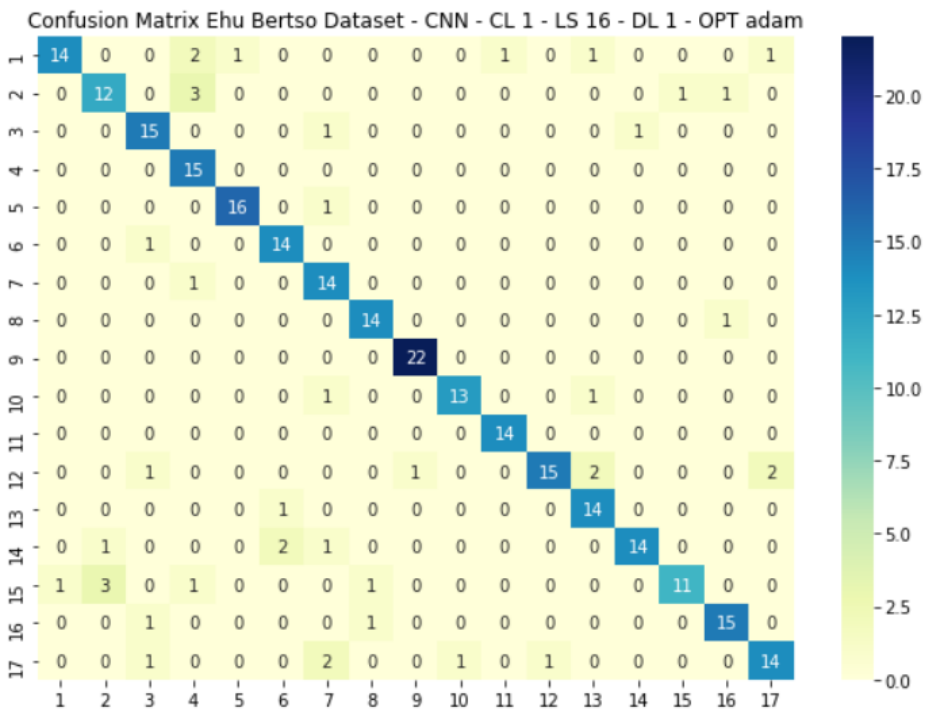
“Aita izena kanta beharrak” is not clearly identifiable and get confused a lot even though it is not confused with a particular class. The errors we get seem to be are more spread.

We could also argue that for example “Antton eta Maria” and “Mutil koxkor bat itsuaurreko” are hard to distinguish between them, and hold structural similarities, as misclassification occurs quite frequently between both classes.

The rest of the remaining classes get quite a good score, above 84%, then we could say that the classification works quite well on them, but for some sparse errors. Because those errors are not focused on a specific class, no affirmation can be made about the specific confusion with another class.

Examining the other two results, from the other one obtaining 86% we obtain pretty much the same conclusions, as the differences are not meaningful enough, but if we look at the one getting 85%, some more insight can be extracted.

In Figure 34 is the confusion matrix for that result.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	14	0	0	2	1	0	0	0	0	0	1	0	1	0	0	0	1
2	0	12	0	3	0	0	0	0	0	0	0	0	0	0	1	1	0
3	0	0	15	0	0	0	1	0	0	0	0	0	0	1	0	0	0
4	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	16	0	1	0	0	0	0	0	0	0	0	0	0
6	0	0	1	0	0	14	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	1	0	0	14	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	14	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	1	0	0	13	0	0	1	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0
12	0	0	1	0	0	0	0	0	1	0	0	15	2	0	0	0	2
13	0	0	0	0	0	1	0	0	0	0	0	0	14	0	0	0	0
14	0	1	0	0	0	2	1	0	0	0	0	0	0	14	0	0	0
15	1	3	0	1	0	0	0	1	0	0	0	0	0	0	11	0	0
16	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	15	0
17	0	0	1	0	0	0	2	0	0	1	0	1	0	0	0	0	14

***Figure 34**: Confusion matrix for the calculation that got 85%.*

From which we can again assemble Table 11.

Melody	Class	Successes	Errors	% Success	% Error
Aita izena kanta beharrak	1	14	6	70	30
Antton eta Maria	2	12	5	70.58823529	29.4117647
Behin batian Loiolan	3	15	2	88.23529412	11.7647059
Betroiarena	4	15	0	100	0
Haizeak bidali du	5	16	1	94.11764706	5.88235294
Ikusi nuenean	6	14	1	93.33333333	6.66666667
Insumisoarena	7	14	1	93.33333333	6.66666667
Iparragirre habila dela	8	14	1	93.33333333	6.66666667
Gitarra zahartxo bat det	9	22		100	0
Triste bizi naiz eta	10	13	2	86.66666667	13.3333333
Langile baten seme	11	14		100	0
Loreak udan ihintza bezala	12	16	5	76.19047619	23.8095238
Aizak hi mutil mainontzi	13	13	2	86.66666667	13.3333333
Mendian gora haritza	14	17	1	94.44444444	5.55555556
Mutil koxkor bat itsuaurreko	15	11	6	64.70588235	35.2941176
Norteko ferrokarrila	16	16	1	94.11764706	5.88235294
Xarmangarria zera	17	14	5	73.68421053	26.3157895

***Table 11**: Success-error table of the calculation that got 85%.*

Here melody “Betroiarena” gets a perfect score so we could say that it is easily discernible from the others.

The error seen in the melodies “Aita izena kanta beharrak”, “Antton eta Maria” and “Loreak udan ihintza bezala” show pretty much what we have seen until now.

Watching melody ”Mutil koxkor bat itsuaurreko” again, we see how it is often confused with “Antton eta Maria” along with other sparse errors.

Finally talking about melody “Xarmangarria zera” we can see how it can be frequently confused with melody “Insumisoarena”.

In a nutshell, after executing the whole best model search process, we can say that the most difficult classes to classify are “Aita izena kanta beharrak”,” Antton eta Maria”, “Loreak udan ihintza bezala” and “Mutil koxkor bat itsuaurreko”, and the easiest are “Betroiarena”, “Gitarra zahartxo bat det” and “Triste bizi naiz eta”.

7. Conclusions and further work

Reflecting on the work done, the following main conclusions have been extracted.

1. The method proposed is valid.
2. Transferred models perform very well.
3. The data set is too small.
4. Data augmentation helps the models learn better.
5. Training the whole model does give satisfactory results.

Even though the problem is not completely solved, as the results obtained do not get the accuracy to make that statement, the method used has been proved to be useful. The results show how the models are able to learn the data to some extent, keeping in mind the limitations of the small dataset.

A more extensive search also can be performed to find most suitable final layer configurations for the model and even more base models can be tested, to see if the results can be improved. The results shown in this work are of the calculations that can be performed, given the resources available at the moment. There is no theoretical limit to the extension of the search that could be conducted if the computational resources are in place.

Given the nature of the NNs, and the tendency to perform better the more data they are fed with, the small size of the initial dataset has been deemed limiting. Even though it has been invaluable and enough to prove that the work is on the right track, the truth is that the results obtained do not lead to a categorical affirmation. Seeing this, the next logical step would be to first strengthen the obvious weak point in the chain, and gather more data keeping in mind that the more data and the more variability it has, the better the system will learn and the more consistency the results will get.

One thing to note on the results obtained is that test accuracy consistently has shown to get better results than the validation or training accuracy. This could happen because the data is not properly divided into training/validation/test, therefore the system learns very well some features that are well represented in the test dataset. To check this, some tests were made, which are not documented here, splitting the dataset in different ways, but the effect kept arising, then data split can be ruled out as a cause. The cause for this to happen then is the dropout since behaviour in training and testing is completely different. In the training phase, the dropout proportion of features is set to zero, but when testing, all the features are used. When this happens, the model is more robust in testing than in the training phase, and higher testing accuracies are obtained.

Even though the data augmentation process has been performed on the dataset, it is crucial to keep in mind, that the extended dataset obtained does not have any extra features that the original dataset does not have. This happens to be that way because all the data generated is based on the original data, thus despite getting to learn the features already in the dataset better, the truth is that no new information is learned, that the original dataset doesn't have.

The use of transferred models has shone in particular. It has provided right off the bat, with quite satisfactory figures, preventing the search for a good model from becoming an arduous and overloading issue. The use of weight and calculations used in imagenet[22] has proven completely necessary in order to obtain good results.

The initial path of three rounds of calculations set in this document has been laid thinking that the results would get better in each step, but it has not been that way. Against all odds, the retraining of the whole transferred model has not proved itself valuable, and what we have seen in the results is that in that situation, overfitting is a major challenge to overcome, that sternly shows itself. This makes the results not to get better at this stage, and thus makes us overlook the models obtained in that particular stage.

Further development of the system can be carried out, taking advantage of Midi libraries and Parsons Code. Even though this path has not been thoroughly examined, if the audio to Midi conversion can be properly addressed, then using a synthesizer and recoding the midi file in audio, and using this very process to learn the resulting data, could help the system focusing on the real issues at hand that would be the variation in things like melody and not the environmental noise.

In general terms, the initial idea of building a system that can recognize a melody from a given audio recording can be labelled as a success, because after the work done, all the steps needed are in place and the challenges met have been overcome to the extent that has been possible, given the computational limitations at hand. The results obtained if not conclusive, at least can be deemed as promising because even though the results obtained do not go above 90% of accuracy, the system is clearly capable of learning the necessary features and begins to show its potential.

Coming back full circle to Bertsobot, and how all this research can be used in that context, the logical step would be to integrate the system proposed in this work in the robot itself. In order to do that, some requirements have to be met by the robot, on the one hand, some kind of microphone to record the user humming the melody must be available, and on the other hand, the necessary means to execute the pre-trained NN and do the previous processing has to be in place. After the microphone records the user and passes the input audio data to the system, first the data preparation phase has to be executed before anything else is done. This process consists on the format conversion, audio filtering and spectrogram generation steps that have been explained before in sections 5.1.1 5.1.2. the process is displayed in Figure 35.

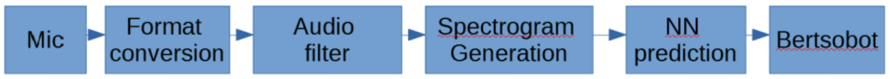


Figure 35*: Process of integration with Bertsobot.*
After the microphone records the audio, data undergoes the process of format conversion, audio filtering and spectrogram generation, before the NN can make its prediction. Then this predicted melody can be forwarded to Bertsobot for the Bertso creation process to carry on.

When those steps are concluded, the data is ready to be consumed by the NN, and when that phase ends, a prediction is outputted with the class of the melody recorded by the user. This melody class, is the data that the BertsoBot will use to compose the Bertso using the abilities it already has.

Those are the software requirements.

Ffmpeg
Python 3.6
Librosa 0.6.3
Keras, 2.4.3 backend: tensorflow

Another aspect worth noting is that when Bertsobot records an input to classify it if some way to confirm the predicted melody is available, this new recorded input can be stored and added to the original dataset, taking advantage this way of the data that the use of the system will provide. Even though this feature falls out of the scope of this research, it can be very interesting because it paves the way to follow for the dataset to be expanded, and thus, improve on the matter of the gathering of more samples.

After all, those steps have been carried out, and everything is installed and running, Bertsobot will have acquired the new ability to identify a melody just by hearing someone humming it. This is a feature taken for granted by the ones attending a Bertso event and adding this possibility to the robot, might be both impressive and useful, as Bertsobot will be able to respond to a Bertso sung by another Bertsolari using the same melody just by listening to it.

Appendix A

a) Full result tables

i) 1st round

The results of the 1st round are compiled in Table 12.

		Train accuracy	Train loss	Validation accuracy	Validation loss	Test accuracy	Overfit epoch
Densenet169							
		0,6406926513	1,626300573	0,5431034565	2,145730257	0,8373702422	1
		0,5757575631	2,001916885	0,5172413588	2,491200924	0,8546712803	1
		0,7564935088	0,9095561504	0,6896551847	1,806303501	0,8512110727	2
	Average:	0,6576479077		0,5833333333		0,8477508651	
Densenet121							
		0,5670995712	1,780809402	0,5	2,469398737	0,8477508651	1
		0,5270562768	2,179206133	0,5344827771	2,281628847	0,8442906574	1
		0,791125536	0,856875062	0,6508620977	1,411714315	0,8408304498	2
	Average:	0,628427128		0,561781625		0,8442906574	
Densenet201							
		0,791125536	0,771800518	0,6422413588	1,359444976	0,830449827	2
		0,82359308	0,6504089832	0,4698275924	2,851834059	0,8373702422	2
		0,5649350882	2,217761755	0,5258620977	2,382128477	0,830449827	1
	Average:	0,7265512347		0,5459770163		0,8327566321	
MobileNet							
		0,6742424369	1,764331341	0,6206896305	1,883572578	0,8269896194	1
		0,6709956527	1,725733876	0,6336206794	2,159816504	0,8408304498	1
		0,6515151262	1,848223805	0,5775862336	2,304608583	0,8131487889	1
	Average:	0,6655844053		0,6106321812		0,8269896194	
ResNet152V2							
		0,7651515007	0,9391310811	0,6034482718	1,973695517	0,7993079585	1

		0,7207792401	1,211236954	0,5818965435	2,441248894	0,76816609	1
		0,7857142687	0,8799658418	0,5689654946	2,398824215	0,7889273356	1
	Average:	0,7572150032		0,5847701033		0,785467128	
ResNet101V2							
		0,6450216174	1,823051572	0,5517241359	2,5232265	0,76816609	1
		0,6634199023	2,085978031	0,5301724076	2,550966501	0,7577854671	1
		0,6796537042	1,479425192	0,6206896305	2,013459444	0,7716262976	1
	Average:	0,662698408		0,5675287247		0,7658592849	
InceptionResNetV2							
		0,479437232	2,468545914	0,4224137962	3,308064222	0,7612456747	1
		0,4329004288	2,564582586	0,3620689511	2,885351419	0,76816609	1
		0,6796537042	1,108098865	0,4698275924	2,100576401	0,7335640138	2
	Average:	0,5306637883		0,4181034466		0,7543252595	
Xception							
		0,5909090638	1,440718293	0,4525862038	2,4988029	0,7439446367	1
		0,5995671153	1,570790887	0,4913793206	2,086574793	0,7612456747	1
		0,5616883039	1,572791576	0,5258620977	2,102257729	0,7404844291	1
	Average:	0,5840548277		0,4899425407		0,7485582468	
NASNetMobile							
		0,6547619104	1,410544157	0,4827586114	2,866377354	0,76816609	2
		0,4285714328	3,544701815	0,3663793206	4,287319183	0,6470588235	1
		0,6244588494	1,597381473	0,5301724076	2,602808237	0,7820069204	2
	Average:	0,5692640642		0,4597701132		0,7324106113	
MobileNetV2							
		0,658008635	2,950284958	0,5172413588	4,632489204	0,7301038062	1
		0,6190476418	3,34551096	0,5732758641	3,389594793	0,7266435986	1
		0,6645021439	2,562045336	0,5301724076	4,313624859	0,7058823529	1
	Average:	0,6471861402		0,5402298768		0,7208765859	

ResNet50V2							
		0,6017315984	2,293468952	0,5	3,49044776	0,6435986159	1
		0,6136363745	2,205449581	0,5215517282	3,138751507	0,7439446367	1
		0,6017315984	2,213868856	0,5215517282	3,853109121	0,7474048443	1
	Average:	0,6056998571		0,5143678188		0,7116493656	
InceptionV3							
		0,456709951	3,560863256	0,375	4,016449928	0,716262976	1
		0,476190478	2,8422575	0,37068966	4,972299099	0,6816609	1
		0,435064942	4,231535435	0,353448272	5,279798508	0,685121107	1
	Average:	0,455988457		0,366379311		0,694348328	
VGG16							
		0,2673160136	2,448743105	0,2241379321	2,492310524	0,660899654	1
		0,3019480407	2,284521341	0,3232758641	2,451333046	0,6366782007	2
		0,2261904776	2,547956467	0,2155172378	2,647443533	0,6366782007	1
	Average:	0,2651515106		0,2543103447		0,6447520185	
VGG19							
		0,2900432944	2,315188169	0,2456896603	2,385848761	0,615916955	2
		0,2586580217	2,476728439	0,2198275924	2,54472971	0,6539792388	1
		0,2889610529	2,316070795	0,2974137962	2,42603898	0,6193771626	2
	Average:	0,2792207897		0,2543103496		0,6297577855	
ResNet152							
		0,07359307259	3,428166151	0,04310344905	3,881465197	0,1349480969	2
		0,05844155699	3,871030331	0,06034482643	4,273066998	0,1453287197	2
		0,08658009022	3,553316832	0,03448275849	3,875352621	0,1730103806	4
	Average:	0,07287157327		0,04597701132		0,1510957324	
ResNet101							
		0,07900433242	3,717307091	0,0517241396	3,738756418	0,107266436	2
		0,07251082361	3,61204195	0,04310344905	4,187629223	0,1211072664	5
		0,0844155848	3,247865677	0,06034482643	3,535975218	0,08650519031	3

	Average:	0,07864358028		0,05172413836		0,1049596309	
Resnet50							
		0,05086579919	9,05861187	0,04741379246	9,693528175	0,05882352941	2
		0,06493506581	6,523808956	0,09051723778	7,95814991	0,05882352941	3
		0,07142857462	8,193541527	0,05603448302	13,62116623	0,06574394464	3
	Average:	0,06240981321		0,06465517109		0,06113033449	

***Table 12**: Full results of the first round of calculations.*

And Table 13 shows the results for the best 3 models.

		Train accuracy	Train loss	Validation accuracy	Validation loss	Test accuracy	Overfit epoch
Densenet169							
		0,6406926513	1,626300573	0,5431034565	2,145730257	0,8373702422	1
		0,5757575631	2,001916885	0,5172413588	2,491200924	0,8546712803	1
		0,7564935088	0,9095561504	0,6896551847	1,806303501	0,8512110727	2
	Average:	0,6576479077		0,5833333333		0,8477508651	
Densenet121							
		0,5670995712	1,780809402	0,5	2,469398737	0,8477508651	1
		0,5270562768	2,179206133	0,5344827771	2,281628847	0,8442906574	1
		0,791125536	0,856875062	0,6508620977	1,411714315	0,8408304498	2
	Average:	0,628427128		0,561781625		0,8442906574	
Densenet201							
		0,791125536	0,771800518	0,6422413588	1,359444976	0,830449827	2
		0,82359308	0,6504089832	0,4698275924	2,851834059	0,8373702422	2
		0,5649350882	2,217761755	0,5258620977	2,382128477	0,830449827	1
	Average:	0,7265512347		0,5459770163		0,8327566321	

***Table 13**: Best results of the 1st round.*

ii) 2nd round

The results obtained in the 2nd round of calculations are shown in Table 14, Table 15 and Table 16.

	Train accuracy	Train loss	Validation accuracy	Validation loss	Test accuracy	Overfit epoch
	0,113636367	14,11982346	0,03448275849	19015,51367	0,892733564	0
	0,1233766228	14,41026306	0,0517241396	50,61285782	0,678200692	0
	0,07034631819	14,82822704	0,06465516984	1126,05188	0,3183391003	0
Average:	0,1024531027		0,05028735598		0,6297577855	

Table 14*: 2nd round DenseNet121.*

	Train accuracy	Train loss	Validation accuracy	Validation loss	Test accuracy	Overfit epoch
	0,05303030461	16,50066948	0,07327586412	1774615,75	0,09342560554	0
	0,08766233921	15,87051773	0,07327586412	41910532	0,5467128028	0
	0,07142857462	12,26207066	0,04310344905	328353216	0,1141868512	0
Average:	0,07070707281		0,06321839243		0,2514417532	

Table 15*: 2nd round DenseNet169.*

	Train accuracy	Train loss	Validation accuracy	Validation loss	Test accuracy	Overfit epoch
	0,05627705529	16,33445168	0,06034482643	81155808	0,05190311419	0
	0,06709956378	18,04464912	0,05603448302	1636550875	0,05190311419	0
	0,07900433242	14,73151779	0,05603448302	635617728	0,7508650519	0
Average:	0,06746031716		0,05747126415		0,2848904268	

Table 16*: 2nd round DenseNet1201.*

Table 17 shows the ranking in end of the round.

Model	Train accuracy	Validation accuracy	Test accuracy
DenseNet 121	0,1024531027	0,05028735598	0,6297577855
DenseNet 169	0,07070707281	0,06321839243	0,2514417532
Densenet 201	0,06746031716	0,05747126415	0,2848904268

***Table 17**: Ranking in the end of the round ranking.*

iii) 3rd round

In Table 18, Table 19 and Table 20 the final results on the 3rd round of calculations are displayed.

Base model	No of dense layers	Dropout	Epochs	Train accuracy	Train loss	Validation accuracy	Validation loss	Test accuracy
Densenet121	1	0	1	0,5725108385	1,632450819	0,5646551847	1,79624784	0,8477508651
	1	0	2	0,7922077775	0,6947026849	0,6034482718	1,563604832	0,8339100346
	1	0	1	0,5703462958	1,64335835	0,5258620977	1,923976421	0,8477508651
	Averages:			0,645021637281		0,564655184746		0,843137254902
	1	0,2	1	0,585497856140	1,640271067619	0,556034505367	1,913555503	0,8269896194
	1	0,2	1	0,5389610529	1,800844312	0,5732758641	1,806759834	0,8546712803
	1	0,2	1	0,591991365	1,579875469	0,5646551847	1,742066383	0,8546712803
	Averages:			0,572150091330		0,564655184746		0,845444059977
	1	0,4	1	0,604978382587	1,611921548843	0,543103456497	1,981299043	0,8477508651
	1	0,4	2	0,8095238209	0,7053077221	0,6508620977	1,187196136	0,8442906574
	1	0,4	1	0,5822510719	1,645889997	0,4870689511	2,189252853	0,8339100346
	Averages:			0,665584425131		0,560344835122		0,841983852364
	1	0,6	2	0,780303001404	0,745592653751	0,685344815254	1,112517238	0,8477508651
	1	0,6	1	0,5660173297	1,783664346	0,5474137664	2,374410152	0,8442906574
	1	0,6	1	0,6071428657	1,489379883	0,5646551847	1,77577889	0,8442906574
	Averages:			0,651154398918		0,599137922128		0,845444059977
	2	0	2	0,7391774654	0,9961191416	0,5603448153	1,534731388	0,8408304498
	2	0	2	0,7316017151	0,9859474897	0,6163793206	1,334600687	0,8408304498
	2	0	1	0,4902597368	1,770056844	0,4741379321	1,865986347	0,8512110727
	Averages:			0,653679639101		0,550287355979		0,844290657439
	2	0,2	2	0,6645021439	1,172387362	0,5043103695	1,752875328	0,8408304498

	2	0,2	2	0,7467532754	0,9539435506	0,6336206794	1,223316908	0,8373702422
	2	0,2	1	0,4339826703	1,929306388	0,4094827473	1,940587521	0,8442906574
	Averages:			0,615079363187		0,515804598729		0,840830449827
	2	0,4	2	0,5790043473	1,474817157	0,5301724076	1,601625204	0,8408304498
	2	0,4	2	0,6937229633	1,061415911	0,5689654946	1,437756777	0,8615916955
	2	0,4	1	0,4772727191	1,729095936	0,4741379321	1,733545184	0,8339100346
	Averages:			0,583333343267		0,524425278107		0,845444059977
	2	0,6	2	0,6190476418	1,356812	0,4956896603	1,625120401	0,830449827
	2	0,6	1	0,5216450095	1,661210537	0,4568965435	1,863216162	0,8166089965
	2	0,6	1	0,4415584505	1,889153004	0,4224137962	1,925581932	0,7993079585
	Averages:			0,527417033911		0,458333333333		0,815455594002
	3	0	2	0,5703462958	1,416152596	0,5431034565	1,614483595	0,8477508651
	3	0	1	0,3896103799	1,968187332	0,4137931168	2,062932253	0,8581314879
	3	0	1	0,4935064912	1,770658731	0,4870689511	1,788105369	0,8408304498
	Averages:			0,484487722317		0,481321841478		0,848904267589
	3	0,2	3	0,5010822415	1,527376771	0,4827586114	1,660612345	0,7958477509
	3	0,2	2	0,3755411208	1,952921987	0,375	1,988816261	0,8096885813
	3	0,2	4	0,5909090638	1,283418775	0,5948275924	1,359694481	0,7958477509
	Averages:			0,489177475373		0,484195401271		0,800461361015
	3	0,4	16	0,6038960814	1,128800154	0,6508620977	1,151860476	0,7024221453
	3	0,4	23	0,67640692	0,9209855199	0,7284482718	0,9725818038	0,7197231834
	3	0,4	6	0,4350649416	1,681078792	0,4396551847	1,735738635	0,7474048443
	Averages:			0,571789314349		0,606321851412		0,723183391003
	3	0,6	None	0,6038960814	1,128800154	0,6508620977	1,151860476	0,4152249135
	3	0,6	None	0,67640692	0,9209855199	0,7284482718	0,9725818038	0,4498269896
	3	0,6	None	0,4350649416	1,681078792	0,4396551847	1,735738635	0,5363321799

	Averag es:			0,571789314349		0,606321851412		0,467128027682

***Table 18**: 3rd round full final results DenseNet121.*

Base model	No of dense layers	Dro pout	Ep oc hs	Train accuracy	Train loss	Validation accuracy	Validation loss	Test accuracy
Densenet169	1	0	2	0,8030303121	0,7482200265	0,5689654946	1,807530403	0,8512110727
	1	0	2	0,8257575631	0,6214348078	0,6810345054	1,439844251	0,8408304498
	1	0	2	0,7846320271	0,8290356398	0,6379310489	1,461913705	0,8512110727
	Avera ges:			0,804473300775		0,629310349623		0,847750865052
	1	0,2	2	0,7564935088	0,9250562191	0,6379310489	1,921296716	0,8512110727
	1	0,2	2	0,7521644831	0,9618831873	0,663793087	1,33889699	0,8650519031
	1	0,2	2	0,8051947951	0,6666011214	0,6939654946	1,518410683	0,8650519031
	Avera ges:			0,771284262339		0,665229876836		0,860438292964
	1	0,4	1	0,5573592782	2,287307739	0,5	2,666698694	0,8546712803
	1	0,4	1	0,5649350882	2,362616777	0,5129310489	2,658260584	0,8408304498
	1	0,4	1	0,5422077775	2,270934582	0,4655172527	2,544887304	0,8512110727
	Avera ges:			0,554834047953		0,492816100518		0,848904267589
	1	0,6	2	0,7997835279	0,6810685992	0,7068965435	1,4049685	0,8512110727
	1	0,6	2	0,708874464	1,073990226	0,6336206794	1,773485541	0,8615916955
	1	0,6	2	0,7770562768	0,850259006	0,6422413588	2,117208242	0,8477508651
	Avera ges:			0,761904756228		0,660919527213		0,853517877739
	2	0	1	0,5454545617	1,471175432	0,4913793206	1,693032265	0,8581314879
	2	0	1	0,4902597368	1,725324512	0,4224137962	1,930057883	0,8512110727
	2	0	2	0,6699134111	1,175844193	0,6163793206	1,409353733	0,8615916955
	Avera ges:			0,568542569876		0,510057479143		0,856978085352
	2	0,2	1	0,4956710041	1,720434666	0,4741379321	1,805100322	0,8442906574
	2	0,2	2	0,7413420081	0,9497066736	0,7284482718	1,162490726	0,875432526

	2	0,2	1	0,4859307408	1,722361207	0,4525862038	1,893262148	0,8442906574
	Averages:			0,574314584335		0,551724135876		0,854671280277
	2	0,4	2	0,7207792401	1,099277377	0,5603448153	1,563274503	0,8615916955
	2	0,4	2	0,6980519295	1,061227202	0,6508620977	1,291505337	0,8581314879
	2	0,4	2	0,6731601954	1,081410885	0,6379310489	1,302940607	0,8581314879
	Averages:			0,697330454985		0,616379320621		0,859284890427
	2	0,6	2	0,6493506432	1,25937295	0,5818965435	1,471002817	0,8442906574
	2	0,6	1	0,4913419783	1,722838283	0,4612068832	1,814691424	0,8719723183
	2	0,6	1	0,405844152	2,026377439	0,3836206794	2,027172565	0,8235294118
	Averages:			0,515512257814		0,475574702024		0,846597462514
	3	0	2	0,6450216174	1,279285073	0,5215517282	1,547018886	0,8546712803
	3	0	2	0,678571403	1,120224357	0,678571403	1,380342245	0,8581314879
	3	0	2	0,7121211886	1,028459311	0,6120689511	1,379039764	0,8650519031
	Averages:			0,678571403027		0,604064027468		0,859284890427
	3	0,2	2	0,364718616	1,923084497	0,4181034565	2,017138958	0,7370242215
	3	0,2	3	0,5487012863	1,409915686	0,5487012863	1,69201839	0,8027681661
	3	0,2	5	0,6244588494	1,184107304	0,6422413588	1,274829507	0,7439446367
	Averages:			0,512626250585		0,536348700523		0,761245674740
	3	0,4	None	0,364718616	1,923084497	0,4181034565	2,017138958	0,7543252595
	3	0,4	11	0,5021644831	1,408201814	0,5021644831	1,420678616	0,7474048443
	3	0,4	24	0,6363636255	1,046108603	0,7155172229	1,084875345	0,6920415225
	Averages:			0,501082241535		0,545261720816		0,731257208766
	3	0,6	None	0,364718616	1,923084497	0,4181034565	2,017138958	0,5051903114
	3	0,6	None	0,5021644831	1,408201814	0,5021644831	1,420678616	0,5051903114
	3	0,6	None	0,6363636255	1,046108603	0,7155172229	1,084875345	0,4429065744

	Averages:			0,501082241535		0,545261720816		0,484429065744

***Table 19**: 3rd round full final results DenseNet169.*

Base model	No of dense layers	Dropout	Epochs	Train accuracy	Train loss	Validation accuracy	Validation loss	Test accuracy
Densenet201								
	1	0	2	0,8106060624	0,7026959062	0,6896551847	1,221144438	0,830449827
	1	0	1	0,6136363745	1,71508944	0,5732758641	1,799008369	0,8269896194
	1	0	1	0,5974025726	1,950765014	0,5215517282	0,5215517282	0,8477508651
	Averages:			0,673881669839		0,594827592373		0,835063437140
	1	0,2	2	0,7792207599	0,8347085118	0,6767241359	1,719776392	0,8442906574
	1	0,2	1	0,5681818128	2,052151918	0,5086206794	2,361958504	0,8373702422
	1	0,2	2	0,8474025726	0,5104438066	0,7327586412	0,7327586412	0,8442906574
	Averages:			0,731601715088		0,639367818832		0,841983852364
	1	0,4	1	0,5725108385	2,202069044	0,5560345054	2,463086605	0,8373702422
	1	0,4	1	0,635281384	1,580265164	0,5775862336	2,154562712	0,8339100346
	1	0,4	1	0,5681818128	2,163578033	0,5517241359	0,5517241359	0,8442906574
	Averages:			0,591991345088		0,561781624953		0,838523644752
	1	0,6	2	0,7878788114	0,7208494544	0,6379310489	1,290333033	0,830449827
	1	0,6	2	0,8019480705	0,751943469	0,6465517282	1,371255636	0,8477508651
	1	0,6	2	0,800865829	0,6387084126	0,7198275924	0,7198275924	0,8512110727
	Averages:			0,796897570292		0,668103456497		0,843137254902
	2	0	1	0,5606060624	1,474141479	0,5818965435	1,517821789	0,8373702422
	2	0	1	0,5422077775	1,555957556	0,5431034565	1,619737983	0,8200692042
	2	0	2	0,7186146975	1,010622859	0,5689654946	1,406909227	0,8512110727
	Averages:			0,607142845790		0,564655164878		0,836216839677

	2	0,2	2	0,7878788114	0,8766360879	0,6681034565	1,232735753	0,830449827
	2	0,2	1	0,5054112673	1,765280724	0,4956896603	1,787030339	0,8512110727
	2	0,2	1	0,5952380896	1,335375071	0,5560345054	1,483379602	0,8442906574
	Averag es:			0,629509389400		0,573275874058		0,841983852364
	2	0,4	1	0,5032467246	1,712877274	0,4568965435	1,7898736	0,8408304498
	2	0,4	2	0,7283549905	1,097471595	0,625	1,391799212	0,8546712803
	2	0,4	2	0,7034631968	1,118208528	0,625	1,374688387	0,8373702422
	Averag es:			0,645021637281		0,568965514501		0,844290657439
	2	0,6	2	0,6677489281	1,161429644	0,625	1,356667042	0,8235294118
	2	0,6	2	0,6915584207	1,110368252	0,5732758641	1,443659425	0,8408304498
	2	0,6	1	0,4880952239	1,692321539	0,4439655244	1,741569996	0,8166089965
	Averag es:			0,615800857544		0,547413796186		0,826989619377
	3	0	1	0,4307359159	1,740642309	0,4310344756	1,788967133	0,8235294118
	3	0	1	0,4707792103	1,673663616	0,4439655244	1,791682959	0,8581314879
	3	0	2	0,6796537042	1,048750877	0,5431034565	1,485444427	0,8235294118
	Averag es:			0,527056276798		0,472701152166		0,835063437140
	3	0,2	4	0,57359308	1,280831337	0,6206896305	1,350141048	0,8200692042
	3	0,2	2	0,4350649416	1,745856166	0,4224137962	1,856045246	0,7958477509
	3	0,2	3	0,5779221058	1,285151362	0,5172413588	1,511307955	0,7404844291
	Averag es:			0,528860042493		0,520114928484		0,785467128028
	3	0,4	21	0,7012987137	0,8494861722	0,7327586412	0,8513110876	0,7024221453
	3	0,4	20	0,6601731777	1,003918052	0,6896551847	1,008953333	0,7128027682
	3	0,4	15	0,6396104097	1,02798903	0,7068965435	1,053260684	0,6989619377
	Averag es:			0,667027433713		0,709770123164		0,704728950404
	3	0,6	No ne	0,7012987137	0,8494861722	0,7327586412	0,8513110876	0,491349481

	3	0,6	None	0,6601731777	1,003918052	0,6896551847	1,008953333	0,4948096886
	3	0,6	None	0,6396104097	1,02798903	0,7068965435	1,053260684	0,5467128028
	Averages:			0,667027433713		0,709770123164		0,510957324106

Table 20*: 3rd round full final results DenseNet201.*

Summarizing the average results, we can obtain Table 21 , Table 22 and Table 23.

No of Dense layers	Dropout	Train accuracy	Validation accuracy	Test accuracy
1	0	0,645021637281	0,564655184746	0,843137254902
1	0,2	0,572150091330	0,564655184746	0,845444059977
1	0,4	0,665584425131	0,560344835122	0,841983852364
1	0,6	0,651154398918	0,599137922128	0,845444059977
2	0	0,653679639101	0,550287355979	0,844290657439
2	0,2	0,615079363187	0,515804598729	0,840830449827
2	0,4	0,583333343267	0,524425278107	0,845444059977
2	0,6	0,527417033911	0,458333333333	0,815455594002
3	0	0,484487722317	0,481321841478	0,848904267589
3	0,2	0,489177475373	0,484195401271	0,800461361015
3	0,4	0,571789314349	0,606321851412	0,723183391003
3	0,6	0,571789314349	0,606321851412	0,467128027682

Table 21*: 3rd round result summary for Densenet121.*

No of dense layers	**Dropout**	**Train accuracy**	**Validation accuracy**	**Test accuracy**
1	0	0,804473300775	0,629310349623	0,847750865052
1	0,2	0,771284262339	0,665229876836	0,860438292964
1	0,4	0,554834047950	0,492816100518	0,848904267589
1	0,6	0,761904756228	0,660919527213	0,853517877739
2	0	0,568542569876	0,510057479143	0,856978085352
2	0,2	0,574314584335	0,551724135876	0,854671280277
2	0,4	0,697330454985	0,616379320621	0,859284890427
2	0,6	0,515512257814	0,475574702024	0,846597462514
3	0	0,678571403027	0,604064027468	0,859284890427
3	0,2	0,512626250585	0,536348700523	0,761245674740
3	0,4	0,501082241535	0,545261720816	0,731257208766
3	0,6	0,501082241535	0,545261720816	0,484429065744

Table 22*: 3rd round full final results DenseNet169.*

No of dense layers	Dropout	Train accuracy	Validation accuracy	Test accuracy
1	0	0,673881669839	0,594827592373	0,835063437140
1	0,2	0,731601715088	0,639367818832	0,841983852364
1	0,4	0,591991345088	0,561781624953	0,838523644752
1	0,6	0,796897570292	0,668103456497	0,843137254902
2	0	0,607142845790	0,564655164878	0,836216839677
2	0,2	0,629509389400	0,573275874058	0,841983852364
2	0,4	0,645021637281	0,568965514501	0,844290657439
2	0,6	0,615800857544	0,547413796186	0,826989619377
3	0	0,527056276798	0,472701152166	0,835063437140
3	0,2	0,528860042493	0,520114928484	0,785467128028
3	0,4	0,667027433713	0,709770123164	0,704728950404
3	0,6	0,667027433713	0,709770123164	0,510957324106

Table 23*: 3rd round full final results DenseNet201.*

Picking the best 5 results of each model we can assemble Table 24.

Base model	No of dense layers	Dropout	Train accuracy	Validation accuracy	Test accuracy
DenseNet 169	1	0,2	0,771284262339	0,665229876836	0,860438292964
DenseNet 169	2	0,4	0,697330454985	0,616379320621	0,859284890427
DenseNet 169	3	0	0,678571403027	0,604064027468	0,859284890427
DenseNet 169	2	0	0,568542569876	0,510057479143	0,856978085352
DenseNet 169	1	0,6	0,761904756228	0,660919527213	0,853517877739
DenseNet 121	3	0	0,484487722317	0,481321841478	0,848904267589
DenseNet 121	1	0,2	0,572150091330	0,564655184746	0,845444059977
DenseNet 121	1	0,6	0,651154398918	0,599137922128	0,845444059977
DenseNet 121	2	0,4	0,583333343267	0,524425278107	0,845444059977
DenseNet 121	2	0	0,653679639101	0,550287355979	0,844290657439
Densenet201	2	0,4	0,645021637281	0,568965514501	0,844290657439
Densenet201	1	0,6	0,796897570292	0,668103456497	0,843137254902
Densenet201	2	0,2	0,629509389400	0,573275874058	0,841983852364
Densenet201	1	0,2	0,731601715088	0,639367818832	0,841983852364
Densenet201	2	0	0,607142845790	0,564655164878	0,836216839677

Table 24*: 3rd round best 5 results per model.*

b) Best result data breakdown

Here we will present the calculations for each of the individual calculations run.

Table 25 shows the figures for the first calculation.

Base model	No of dense layers	Drop out	Epo chs	Train accuracy	Train loss	Validation accuracy	Validation loss	Test accuracy
Densenet169								
	1	0,2	2	0,7564935088	0,9250562191	0,6379310489	1,921296716	0,8512110727

***Table 25**: Result of first calculation.*

Figure 36 shows its confusion matrix.

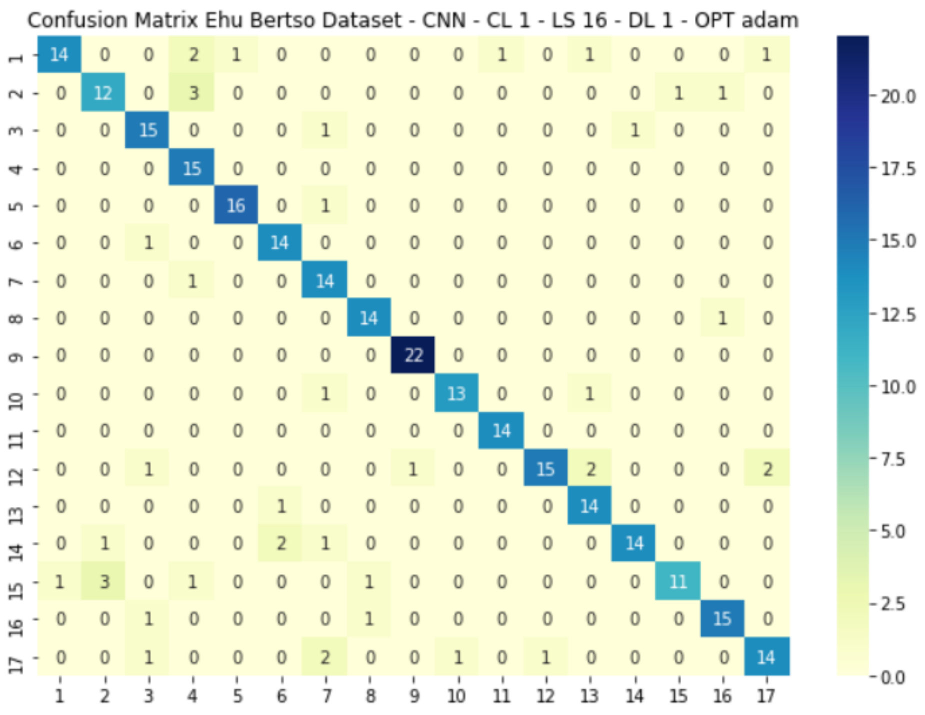


***Figure 36**: Confusion matrix for the first calculation.*

And Table 26 shows the success/error percentages.

Melody	Class	Successes	Errors	% Success	% Error
Aita izena kanta beharrak	1	14	6	70	30
Antton eta Maria	2	12	5	70.58823529	29.41176471
Behin batian Loiolan	3	15	1	93.75	6.25
Betroiarena	4	15	0	100	0
Haizeak bidali du	5	16	1	94.11764706	5.882352941
Ikusi nuenean	6	14	1	93.33333333	6.666666667
Insumisoarena	7	14	1	93.33333333	6.666666667
Iparragirre habila dela	8	14	1	93.33333333	6.666666667
Gitarra zahartxo bat det	9	22	0	100	0
Triste bizi naiz eta	10	13	2	86.66666667	13.33333333
Langile baten seme	11	14	0	100	0
Loreak udan ihintza bezala	12	15	6	71.42857143	28.57142857
Aizak hi mutil mainontzi	13	14	1	93.33333333	6.666666667
Mendian gora haritza	14	14	4	77.77777778	22.22222222
Mutil koxkor bat itsuaurreko	15	11	6	64.70588235	35.29411765
Norteko ferrokarrila	16	15	2	88.23529412	11.76470588
Xarmangarria zera	17	14	5	73.68421053	26.31578947

Table 26*: Success - error results for the first calculation.*

Then we show the same data for the second calculation run, as for the figures we have Table 27.

Base model	No of dense layers	Drop out	Epochs	Train accuracy	Train loss	Validation accuracy	Validation loss	Test accuracy
Densenet169								
	1	0,2	2	0,7521644831	0,9618831873	0,663793087	1,33889699	0,8650519031

Table 27: *Results for the second calculation.*

Figure 37 displays the confusion matrix.

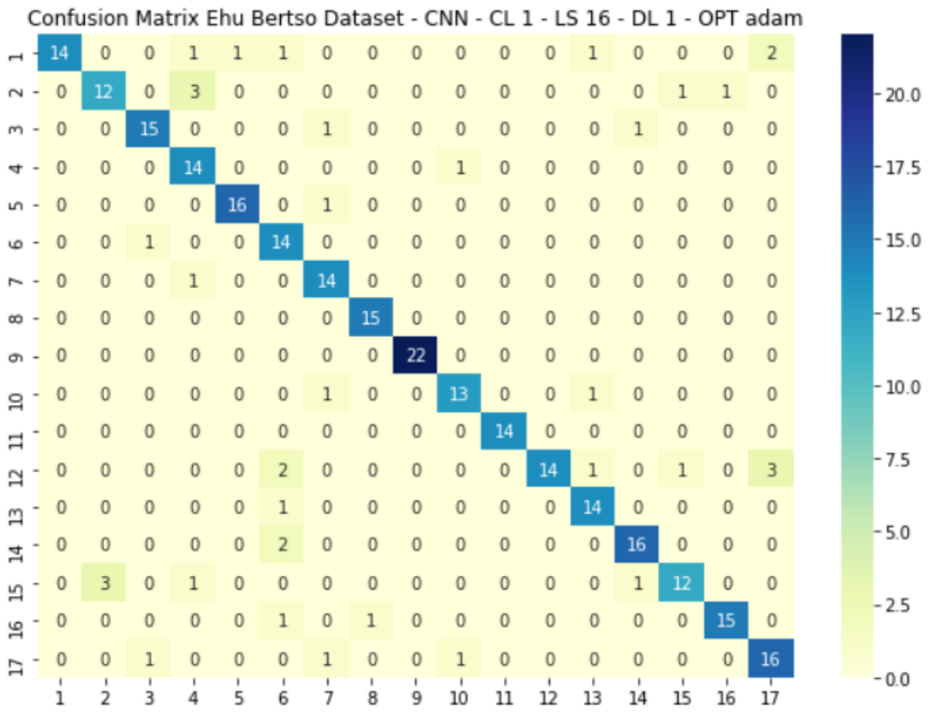


Figure 37: *Confusion matrix for the second calculation.*

And in Table 28 we have the success/error percentages.

Melody	Class	Successes	Errors	% Success	% Error
Aita izena kanta beharrak	1	14	6	70	30
Antton eta Maria	2	12	5	70.58823529	29.41176471
Behin batian Loiolan	3	15	2	88.23529412	11.76470588
Betroiarena	4	14	1	93.33333333	6.666666667
Haizeak bidali du	5	16	1	94.11764706	5.882352941
Ikusi nuenean	6	14	1	93.33333333	6.666666667
Insumisoarena	7	14	1	93.33333333	6.666666667
Iparragirre habila dela	8	15	0	100	0
Gitarra zahartxo bat det	9	22	0	100	0
Triste bizi naiz eta	10	13	2	86.66666667	13.33333333
Langile baten seme	11	14	0	100	0
Loreak udan ihintza bezala	12	14	7	66.66666667	33.33333333
Aizak hi mutil mainontzi	13	14	1	93.33333333	6.666666667
Mendian gora haritza	14	16	2	88.88888889	11.11111111
Mutil koxkor bat itsuaurreko	15	12	5	70.58823529	29.41176471
Norteko ferrokarrila	16	15	2	88.23529412	11.76470588
Xarmangarria zera	17	16	3	84.21052632	15.78947368

***Table 28**: Success - error results for the second calculation.*

Doing the same one last time for the third calculation, we have Table 29 to show the results obtained.

Base model	No of dense layers	Drop out	Epochs	Train accuracy	Train loss	Validation accuracy	Validation loss	Test accuracy
Densenet169								
	1	0,2	2	0,8051947951	0,6666011214	0,6939654946	1,518410683	0,8650519031

Table 29: *Results for the third calculation.*

Figure 38 displays the confusion matrix for this second calculation.

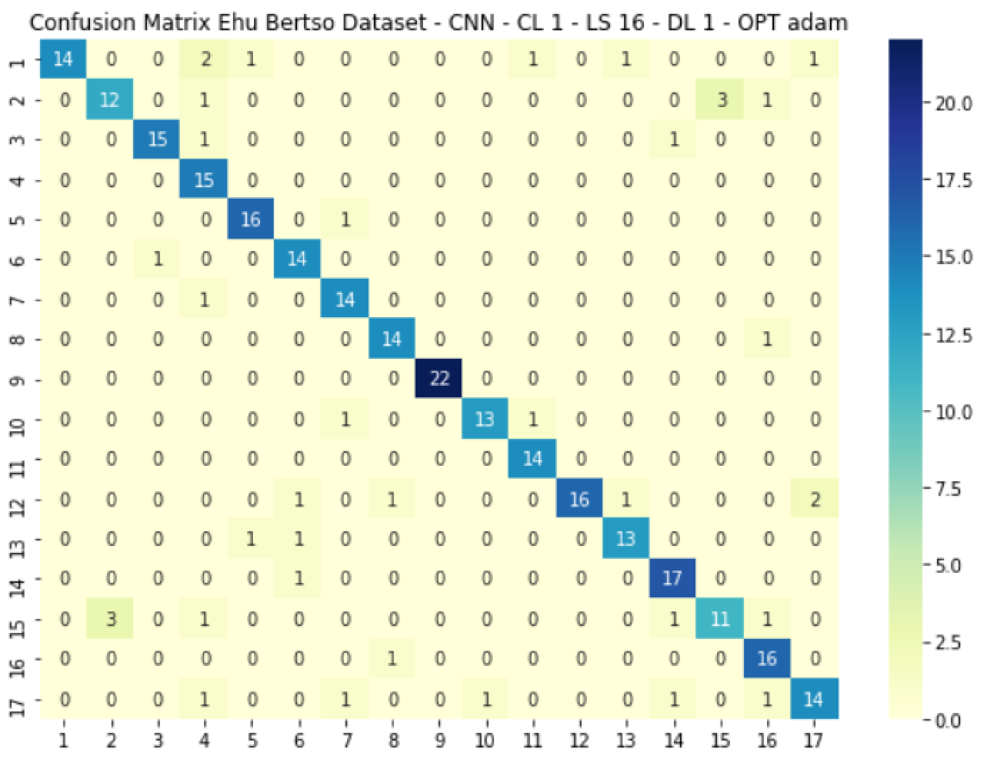


Figure 38: *Confusion matrix for the third calculation.*

And finally, the success/error percentage is shown in Table 30.

Melody	Class	Successes	Errors	% Success	% Error
Aita izena kanta beharrak	1	14	6	70	30
Antton eta Maria	2	12	5	70.58823529	29.4117647
Behin batian Loiolan	3	15	2	88.23529412	11.7647059
Betroiarena	4	15	0	100	0
Haizeak bidali du	5	16	1	94.11764706	5.88235294
Ikusi nuenean	6	14	1	93.33333333	6.66666667
Insumisoarena	7	14	1	93.33333333	6.66666667
Iparragirre habila dela	8	14	1	93.33333333	6.66666667
Gitarra zahartxo bat det	9	22		100	0
Triste bizi naiz eta	10	13	2	86.66666667	13.3333333
Langile baten seme	11	14		100	0
Loreak udan ihintza bezala	12	16	5	76.19047619	23.8095238
Aizak hi mutil mainontzi	13	13	2	86.66666667	13.3333333
Mendian gora haritza	14	17	1	94.44444444	5.55555556
Mutil koxkor bat itsuaurreko	15	11	6	64.70588235	35.2941176
Norteko ferrokarrila	16	16	1	94.11764706	5.88235294
Xarmangarria zera	17	14	5	73.68421053	26.3157895

***Table 30**: Success - error results for the third calculation.*

Bibliography

1. Astigarraga A, Agirrezabal M, Lazkano E, Jauregi E, Sierra B. Bertsobot: The first minstrel robot. *2013 6th International Conference on Human System Interactions (HSI)*. 2013:129-136. doi: 10.1109/HSI.2013.6577813.
2. Euskal herriko unibertsitatea. https://www.ehu.eus/eu/.
3. Query by humming. https://en.wikipedia.org/wiki/Query_by_humming.
4. UrbanSounds 8k. https://urbansounddataset.weebly.com/urbansound8k.html.
5. Wikipedia community. Sound on wikipedia. https://en.wikipedia.org/wiki/Sound.
6. McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bull Math Biophys*. 1943;5(4):115-133. https://doi.org/10.1007/BF02478259 https://doi.org/10.1007/BF02478259. doi: 10.1007/BF02478259.
7. Hebb DO. *The organization of behavior: A neuropsychological theory.* Taylor & Francis; 2005. https://books.google.es/books?id=ddB4AgAAQBAJ https://books.google.es/books?id=ddB4AgAAQBAJ.
8. Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol Rev*. 1958;65(6):386-408. doi: 10.1037/h0042519.
9. HUBEL DH, WIESEL TN. Receptive fields of single neurones in the cat's striate cortex. *J Physiol (Lond)*. 1959;148(3):574-591. https://pubmed.ncbi.nlm.nih.gov/14403679 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1363130/ https://pubmed.ncbi.nlm.nih.gov/14403679 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1363130/. doi: 10.1113/jphysiol.1959.sp006308.
10. A. G. Ivakhnenko and V. G. Lapa. Cybernetic predicting devices. . 1966.
11. Minsky M, Papert S. *Perceptrons.* Oxford, England: M.I.T. Press; 1969.
12. Werbos P, John P. Beyond regression : New tools for prediction and analysis in the behavioral sciences /. . 1974.
13. Rumelhart, Mcclelland J, L. J. *Parallel distributed processing: Explorations in the microstructure of cognition. volume 1. foundations.* ; 1986.
14. Rumelhart DE, McClelland JL. Information processing in dynamical systems: Foundations of harmony theory. In: *Parallel distributed processing: Explorations in the microstructure of cognition: Foundations.* MITP; 1987:194-281. http://ieeexplore.ieee.org/document/6302931 http://ieeexplore.ieee.org/document/6302931.
15. Weng J, Ahuja N, Huang T. Learning recognition and segmentation using the cresceptron. *International Journal of Computer Vision*. 1997;25:109-143. doi: 10.1023/a:1007967800668.
16. Weng J, Ahuja N, Huang TS. Cresceptron: A self-organizing neural network which grows adaptively. *Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. 1992;1:576-581 vol.1. doi: 10.1109/IJCNN.1992.287150.
17. Weng JJ, Ahuja N, Huang TS. Learning recognition and segmentation of 3-D objects from 2-D images. *1993 (4th) International Conference on Computer Vision*. 1993:121-128. doi: 10.1109/ICCV.1993.378228.
18. Schmidhuber J. Learning complex, extended sequences using the principle of history compression. *Neural Comput*. 1992;4(2):234-242.
19. Hinton G, Osindero S, Teh Y. A fast learning algorithm for deep belief nets. *Neural Comput*. 2006;18:1527-54. doi: 10.1162/neco.2006.18.7.1527.
20. Hinton G. Deep belief networks. *Scholarpedia*. 2009;4:5947. doi: 10.4249/scholarpedia.5947.
21. Le QV. Building high-level features using large scale unsupervised learning. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 2013:8595-8598. doi: 10.1109/ICASSP.2013.6639343.
22. Imagenet. http://www.image-net.org/.
23. Keras. . . https://keras.io/.

24. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. *arXiv 1409 1556*. 2014.
25. He K, Zhang X, Ren S, Sun J. *Deep residual learning for image recognition.* ; 2016:770-778. 10.1109/CVPR.2016.90.
26. Howard A, Zhu M, Chen B, et al. MobileNets: Efficient convolutional neural networks for mobile vision applications. . 2017.
27. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L. *MobileNetV2: Inverted residuals and linear bottlenecks.* ; 2018:4510-4520. 10.1109/CVPR.2018.00474.
28. Zoph B, Vasudevan V, Shlens J, Le Q. *Learning transferable architectures for scalable image recognition.* ; 2018:8697-8710. 10.1109/CVPR.2018.00907.
29. Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna ZB. *Rethinking the inception architecture for computer vision.* ; 2016. 10.1109/CVPR.2016.308.
30. Szegedy C, Ioffe S, Vanhoucke V, Alemi A. Inception-v4, inception-ResNet and the impact of residual connections on learning. *AAAI Conference on Artificial Intelligence*. 2016.
31. Chollet F. *Xception: Deep learning with depthwise separable convolutions.* ; 2017:1800-1807. 10.1109/CVPR.2017.195.
32. Huang G, Liu Z, van der Maaten L, Weinberger K. *Densely connected convolutional networks.* ; 2017. 10.1109/CVPR.2017.243.
33. Qin J, Lin H, Liu X. Query by humming systems using melody matching model based on the genetic algorithm. *JSW*. 2011;6:2416-2420. doi: 10.4304/jsw.6.12.2416-2420.
34. Shifrin J, Pardo B, Meek C, Birmingham W. HMM-based musical query retrieval. *Proceedings of the Second ACM/IEEE-CS Joint Conference on Digital Libraries*. 2002:295-300. http://www.scopus.com/inward/record.url?scp=0036989244&partnerID=8YFLogxK http://www.scopus.com/inward/citedby.url?scp=0036989244&partnerID=8YFLogxK.
35. Jang J, Gao M. A query-by-singing system based on dynamic programming. . 2000.
36. Fu L, Xues X. A new efficient approach to query by humming. *ICMC*. 2004. Accessed Sep 11, 2020.
37. Fu L, Xues X. *A new spectral-based approach to query-by-humming for MP3 songs database.* ; 2005:117-120.
38. Muda L, Begam M, Elamvazuthi I. Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques. *J Comput*. 2010;2.
39. Casey MA, Veltkamp R, Goto M, Leman M, Rhodes C, Slaney M. Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*. 2008;96(4):668-696. doi: 10.1109/JPROC.2008.916370.
40. Antoniol G, Rollo V, Venturi G. *Linear predictive coding and cepstrum coefficients for mining time variant information from software repositories.* Vol 30. ; 2005. 10.1145/1083142.1083156.
41. Nagavi T, Bhajantri N. An extensive analysis of query by singing/humming system through query proportion. *International Journal of Multimedia & Its Applications*. 2013;4. doi: 10.5121/ijma.2012.4606.
42. Parsons code. https://en.wikipedia.org/wiki/Parsons_code.
43. Midi. https://en.wikipedia.org/wiki/MIDI.
44. Librosa. https://github.com/librosa/librosa.
45. Satt A, Rozenberg S, Hoory R. *Efficient emotion recognition from speech using deep learning on spectrograms.* ; 2017:1089-1093. 10.21437/Interspeech.2017-200.
46. A. M. Badshah, J. Ahmad, N. Rahim, S. W. Baik. Speech emotion recognition from spectrograms with deep convolutional neural network. *2017 International Conference on Platform Technology and Service (PlatCon)*. 2017:1-5. doi: 10.1109/PlatCon.2017.7883728.
47. Freesound.org. https://freesound.org/.
48. Ffmpeg. https://ffmpeg.org/.
49. Creative commons 3.0 attribution non-comercial. https://creativecommons.org/licenses/by-nc/3.0/.
50. Inception V3. https://en.wikipedia.org/wiki/Inceptionv3.

51. Audcaity. https://www.audacityteam.org/.
52. Human voice in wikipedia. https://en.wikipedia.org/wiki/Human_voice.
53. Jupyter notebooks. https://jupyter.org/.
54. Tensorflow. https://www.tensorflow.org.
55. Google colab. https://colab.research.google.com/.
56. Google drive. https://drive.google.com/drive/u/0/my-drive.