POLITECNICO DI TORINO

Master degree course in Data Science

Master Degree Thesis

creativeAI
visualizing music emotions

Supervisors

Prof. Tatiana TOMMASI

Ing. Giuseppe RIZZO

Candidate

Marco TESTA

matr.: 265861

Accademic Year 2020-2021

Summary

Nowadays Artificial Intelligence (AI) is a widespread tool that has consistently came into our lives recently due to the enhancement of the Deep Learning techniques made possible by means of the growth of the hardware computational power and data availability. Despite its support on *hard* tasks such as object classification, geo-localization, or text completion, AI can contribute to some other called *soft* tasks related to art and creativity. However, as of now, there are few studies that aim to deal with these aspects using a multimodal approach, yet as humans we know that these subjects don't have a single communication channel, moreover, there are not many datasets supporting these goals. This thesis work aims to show how deep learning can be used for artistic purposes than practical ones, which only recently have attracted the attention of the research community.
This work stems from the music emotion recognition task, which is exploited to connect sound information to visual one through a conditioned generative process. As a first step, a detailed analysis of the sound-emotion and visual-emotion existing datasets took place together with their alignment to have a shared set of emotion categories to be used to connect audio and visual information. Consequently, different sound representations have been used and compared to perform the emotion recognition task to build a classifier whose output is wanted to be used as the input of generative model, obtaining new creative images reflecting the music sentiment.

Acknowledgements

At first, I would like to manifest my gratification for the support to my academic supervisor Prof. Tatiana Tommasi and my thesis industrial supervisor Ing. Giuseppe Rizzo, from LINKS Foundation, who gave me constructive suggestions during my work.
Then I would like to thank my colleagues for every dialog moment spent together during these academic years and my friends for a lot of things to be listed.

I want to say that I am grateful to my girlfriend Emma, for her understanding and kind help.

Finally, I am delighted to thank Maria Grazia and Antonino, my dear parents, for their love, their trust and for the endless encouragement.

Marco Testa

Table of Contents

List of Tables

List of Figures

Acronyms

AI
artificial intelligence

MIR
Music Information Retrieval

MER
Music Emotion Recognition

GWMER
Group-wise Music Emotion Recognition

PMER
Personalized Music Emotion Recognition

EQ
Emotional Intelligence

MFCC
Mel-Frequency Cepstral Coefficient

GAN
Generative Adversarial Network

ICMC
International Computer Music Conference

ICMA
International Computer Music Association

IIA

Independence from Irrelevant Alternatives

FFT

Fast Fourier Transform

DFT

Discrete Fourier Transform

STFT

Short-Time Fourier Transform

CNN

Convolutional Neural Network

RNN

Recurrent Neural Network

LSTM

Long-Short Term Memory

FID

Fréchet Inception Distance

IS

Inception Score

m-IS

modified Inception Score

Part I

First Chapter

Chapter 1

Introduction

1.1 Problem Statement

'Would it be possible to make AI creative and empathic?'

This is the starting point of this work, whose purpose is to create an AI capable of *'listening'* to music to get inspiration for visual artwork.
The work is composed of two natures: audio and visual. The first difference between them comes directly from the human sense required to perceive an audio and an image. Obviously, there is not an objective *direct-map* between what we can hear and its visual representation thought as an image or a color.
Could this mapping be found?

This question guides towards the search of a solution aiming to create a direct link between these two domains: they are two human's art forms and since they are art forms they are created by the human instinct, at least in a very primordial form. Going a bit deeper and talking about human evolution we can assert that the factor of human growth is given by the neocortex, the largest part of the cerebral cortex. Among all its functions it's worth mentioning that it is responsible for cognitive processes, social and emotional processing. As neocortex becomes more evolved, humans become more conscious about themselves, understanding that art is the direct expression of feelings that can be controlled to give this expression a wanted structure, a thought-form. In other words art is, in a more conscious meaning, the output given by human's intellectual choices made in order to give a form to emotional inputs.

Leo Tolstoy: *"Art is a human activity consisting in that one human consciously hands on to others feelings they have lived through, and that other peple are infected by these feelings, and also experience them"*

We can find a better, scientific definition reading Daniel Goleman, a scientific journalist, author and psychologist, who gave a structure to this concept calling it *'Emotional Intelligence'* (EQ), defined as the humans' ability to understand and manage their own emotions and feelings such that EQ can be applied to reach **self-awarness**, objectivity and equality. He is not the founder of the emotional intelligence theory but he developed a framework of five key components that compose the emotional intelligence, plus a set of skills that can be developed and improved by anyone.

Music art and Visual art share a common space identified by the emotions they bring in themselves, so at this point we can think a ***music-emotion map*** and a ***image-emotion map*** thus use the emotion space as the direct link between audio and image.

Taking head of this, this thesis work focuses on two emotion-labelled dataset: a first dataset containing audio files used to train a music-to-emotion classifier and a second one containing paintings used to train an image-to-emotion classifier which is exploited in the context of the *conditional*GAN framework, in order to generate visual artworks inspired by music emotions.

1.2 High Level Pipeline

The high-level AI system is depitcted in 1.1.

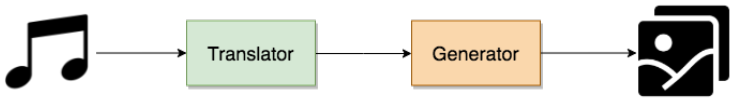


Figure 1.1: High Level Pipeline

Translator is a Deep Learning classification model which is in charge of taking an audio file as input and extract the emotion carried by the music.
More details are in Chapter 4, where a comparison between different modal-classifier is explained. Since audio is information, we can have different representations (**.mp3*, **.wav*, Spectrogram, MFCC) of the same information thus different music-to-emotion classifiers based on the different starting representations of sound.

Generator is a conditional GAN capable of generating visual art inspired by the emotions given by the **Translator**.

Chapter 2

Related works

2.1 Music Classification

The first time music field was mentioned to be studied under the AI domain was in 1974 at ICMC (International Computer Music Conference), Michigan State University, East Lansing, USA. Starting from that moment the ICMC became an annual event organized by the ICMA (International Computer Music Association). Although it's been almost 60 years since the AI experiments on music composition began, the field is still considered at its beginning by many music experts and researchers.
There is no a primary context in which music is placed because it could be present in most of the routine activities performed by humans. It represents a connection point for people, a topic of discussion or even a background for reflection.
Music has always played an important role in humans' lives, and nowadays with the enhancement of the digital and the internet network everyone can have an easy access to a huge library of music such that music industry has shifted away from physical media formats towards selling online contents and services. This events have changed how music content have to be stored due to the increasing challenges related to its searching, thus retrieving. To cope with this arising problems several interdisciplinary experts, ranging from music perception, cognition, musicology to engineering and computer science, have been involved in activities that have been resulted in many proposed algorithmic and methodological solutions to music search using content-based methods. [1]

Music Information Retrieval (MIR) is the name given to the pool of strategies aiming to get other information from sound analysis for enabling a personalized access to music collections to a different type of users such as recording or aggregation Industries, end users, music performers, teachers, musicologists and so on.[1]
Before the development of the first MIR-systems the most common method used to

access music was through textual metadata, whose expressiveness was sufficient in many scenarios. The actual problem is related to the impracticability of scaling this approach, because as catalogues become larger, maintaining consistent expressive metadata is an hard challenge. Furthermore these descriptions represent opinions, so the editorial supervision of the metadata is paramount. [2]
Going further the *metadata-based* systems have been by *content-based* systems whose strongest point is to hold music descriptors able to identify what the user is seeking for even when he does not know exactly what he is looking for.
An example of content-based MIR is *Shazam*, capable of retrieving a soundtrack only with a few seconds of recording given as input for the query. [3]
MIR-systems are commonly divided into *content-based* search systems of general *audio-data* and *music-based* search systems which works on the notes, moreover there are also *hybrid* systems which convert any type of audio data into a symbolic version of the notes.
Because of the many different aspects and uses about MIR-systems and their relative tasks, addressing all of them is pointless but it's worth mentioning that the majority of most interesting tasks is related to music genre classification [4], instrument activity detection [5], chords recognition [6], audio segmentation [7], automatic speech recognition [8] and music emotion recognition [9].

As told in the Introduction section 1.1, this work aims to analyze audio with an emotion-centric point of view, thus the task of interest is the latter mentioned.

Music Emotion Recognition (**MER**) is the task which nowadays involves Machine Learning and even Deep Learning techniques to extract emotion as a feature from music. The detection of emotions in music has raised a particular interest since music is a pervasive subject and it is the finest language of emotions [10], moreover the recent rise of social music applications has generated the demand on the emotional ability of search engines [11]. The research community has developed many music emotion recognition solution such as MoodTrack [12], MusicSense [13], MoodCloud [14], just to name a few.
All of them accomplish that MER task is one of the most challenging due to the intrinsic subjective nature of the relationship between music and emotions, and to the difficulty to give a worldwide way to express emotions, because adjectives used to their description might vary from person to person.
Starting from saying that a single listener could be differently affected from the same song [15], it's quite intuitive to think that every single person has a different perception of the same song, thus people can recognize different emotions from the same song. This topic was deeply studied and it's still being studied from different point of view as it is possible to read in paper [16]. It appeared that the role of individuality has the major contribute in a MER-system in terms of its success.

Build a objective MER-system, working in all scenarios, seems really unfeasible unless other considerations are taking in account when designing it, but still it would require a lot of effort especially at the beginning.
The first thing to be taken in account is undoubtedly the individuality: authors of [16] point that none of other's previous works considered this aspect.
This thesis work goes beyond this approach, although it should, as well.
The major contribute of [16] was to focus the attention in this preliminary concept rather than approach MER as an aseptic classification problem, proposing a *group-wise* MER approach (**GWMER**) and a *personalized* MER approach (**PMER**). The first evaluates the importance of each individual factor such as sex, personality and music experience, while the latter evaluates whether the prediction accuracy for a user is significantly improved if the MER system is tailored for the user.

Usually, there are two common classification approaches: a dimensional and a categorical one. The dimensional problem has been generally handled with a shallow classifier such as Support Vector Regressor (SVR) or Support Vector Machine (SVM) trained on real-values labels, such as Valence-Arousal points, or on handcrafted features called Mel-Frequency Cepstral Coefficients (MFCCs).
The categorical one focuses on characteristics that differentiate emotions from one another taking in account music features such as timbre, harmony or rhythm. Once Deep Learning has gained effectiveness and popularity, different novel approaches have been tried to process sound, regardless the task. The most common approach exploits 2D-Convolutional Neural Networks and consists on fitting an audio waveform into a pre-processing pipeline which outputs another representation of the sound: the most famous and used is the Spectrogram representation seen as an image, perfect to be fed into a 2D-CNN.
Most recent approaches to audio-classification discard handcrafted features extraction pipeline and focus only on the direct waveform analysis by feeding it into a 1D-CNN. Starting from this findings, the research community moved towards the application of CNN [17], [11], LSTM-RNN [18] and combination of those architectures [19], [18]. Despite the subjectivity nature of MER together with its impact on MER-systems performances are well recognized, this thesis wants to be an attempt to be the basis for the development of the deep learning project described, concerning the artistic field. Although there are improvements that could be made, the works want to give a new idea about a possible, different, application of an emotion classifier, aimed at giving AI the possibility to create visual art in a new way, being inspired by music, as many human artists do.

2.1.1 Audio representations

'What is Sound?'
From a physical point of view sound is a vibration that propagates as an acoustic wave through a transmission medium.

Acoustic waveform

The first, raw representation of sound is a **pressure wave**, whose propagation, in one dimension, is governed by the second-order linear partial differential equation 2.1.

$$\frac{\partial^2 p}{\partial x} - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = 0 \tag{2.1}$$

where:

- x is the dimension of propagation
- c is the speed of sound through the medium
- p is acoustic pressure

and equation 2.2 as solution:

$$\Delta P = \Delta P_{max} sin(kx \mp wt + \varphi) \tag{2.2}$$

When acoustic waves have frequencies $f \in [20Hz,\ 20kHz]$ they can be perceived by humans ears.
Figure 2.1 represents the plot of the amplitude-envelope of a waveform and since this represents the *raw* signal it holds the highest level of detail.

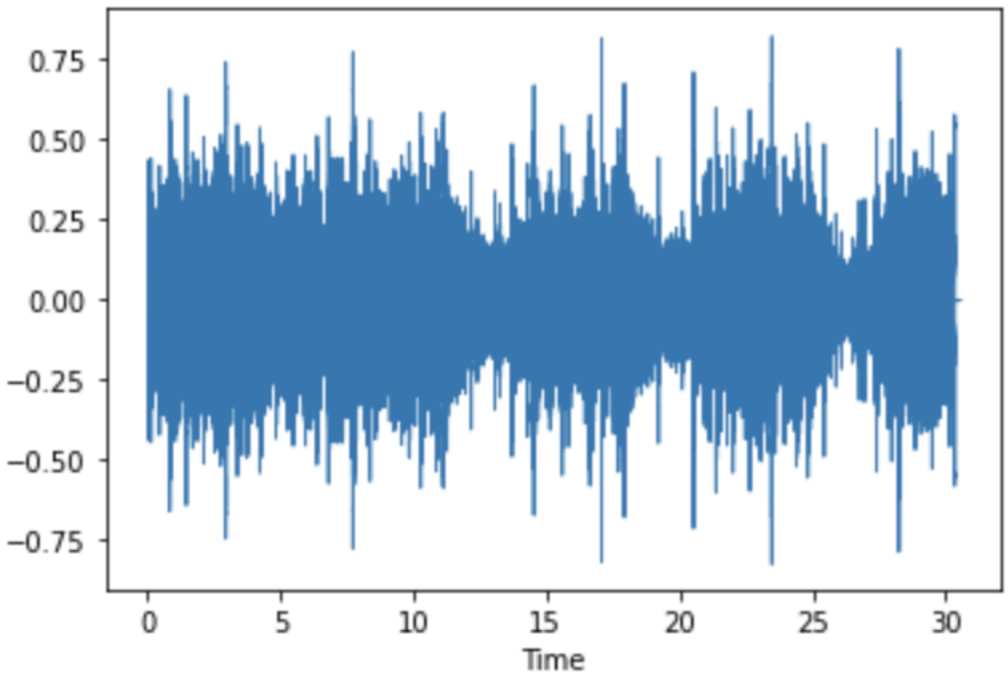


Figure 2.1: Audio Waveform

This representation is used in section 3.4 and explored as input in Deep Learning models in section 4.3.3.

Spectrogram

A single waveform is an ensemble of several single-frequency sound waves and the single sample which is taken over time represents only the resulting amplitude. The mathematical transformation which allows to decompose a function depending on space or time is the **Fourier Transform** (***FT***), defined by equation 2.3.

$$\mathcal{F}\{g(t)\} = G(f) = \int_{-\infty}^{\infty} g(t)e^{-i2\pi ft}dt \tag{2.3}$$

It is a tool used to break down a waveform into an alternative representation showing that a single waveform can be written as the sum of several sinusoids of different frequencies. The resulting function depends on frequency f and gives the amount of power $g(t)$ contained at frequency f, generally called **spectrum** of g. In other words this tool converts the signal from the time domain into the frequency domain (Figure 2.2).

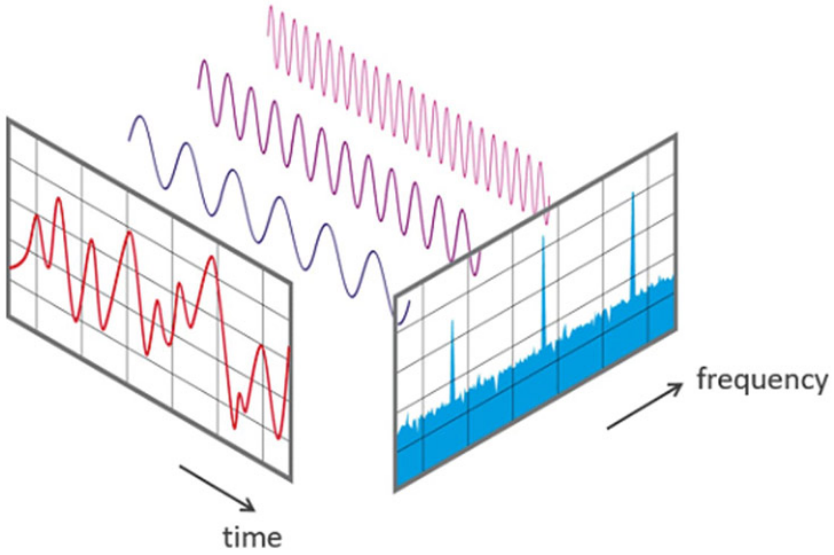


Figure 2.2: FT Illustration

Since we are dealing with the digital signal processing we are referring to discrete functions, thus we will use discrete version of the **FT**, the **Discrete Fourier Transform** (**DFT**) from now on (eq. 2.4). The **DFT** transforms a sequence of N complex numbers $x_n := x_0, x_1, ..., x_{N-1}$ into another sequence of complex numbers $N_k := X_0, X_1, ..., X_{N-1}$ defined by equation 2.4:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} = \sum_{n=0}^{N-1} x_n \cdot \left[cos\left(\frac{2\pi}{N}kn\right) - i \cdot sin\left(\frac{2\pi}{N}kn\right)\right] \tag{2.4}$$

Fast Fourier Transform (**FFT**) is the name given to the algorithm which efficiently compute the **DFT**, used in signal processing in order to analyze the frequency content of a signal.
When the frequency content varies over time, such as in Music, speech or more in general in **non-periodic** signals, we can use a method which involves the computation of several FFT spectrums performed on several windowed segments of the signal.
Short-time Fourier Transform (**STFT**) consists in taking a *sliding window* along the signal (or *time series object*) and computing the **DFT** on the time dependent segment. Figure 2.3 and 2.4 show the outputs of **STFT** using one fixed window of 2048 samples on 4 different audio lengths ($[k * 2048]\ for\ k = 1, 2, 3, 4$) of the same file, sampled at 44100Hz.

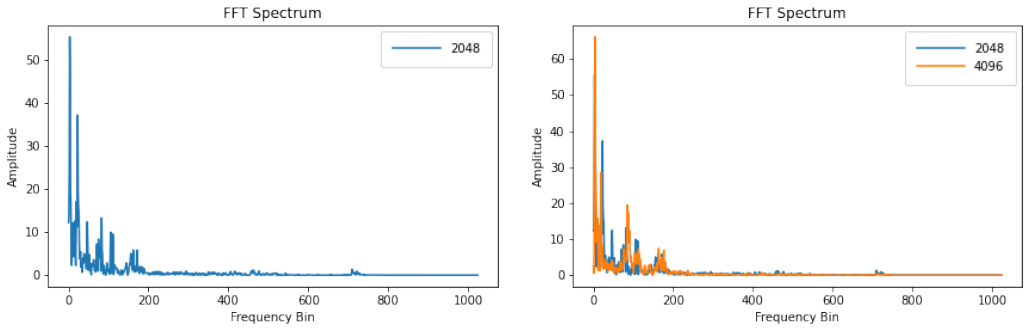


Figure 2.3: STFT outputs: spectrum of '2.wav'

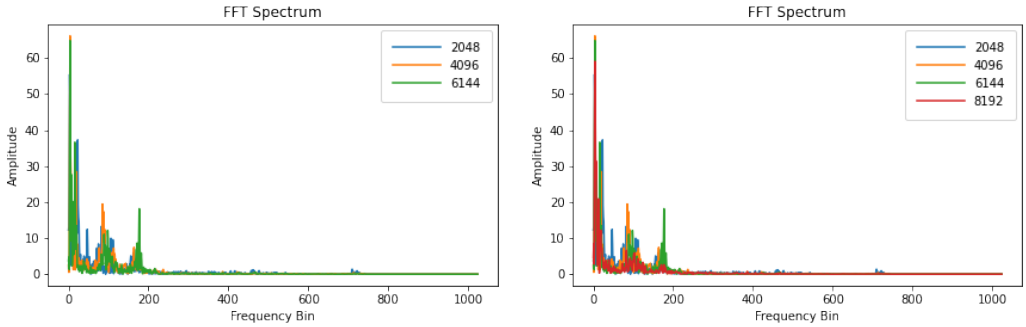


Figure 2.4: STFT outputs: spectrum of '2.wav'

Converting Amplitudes to dB and stacking several FFTs on top of each other it is possible to obtain the **Spectrogram** where the color dimension represents dB. (Figure 2.5).

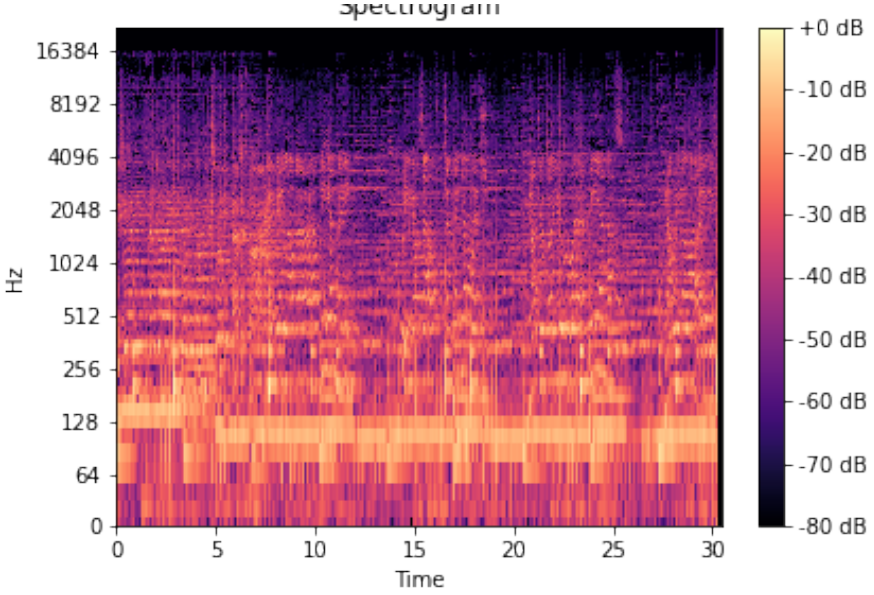


Figure 2.5: Spectrogram of '2.wav'

As explained in [20] humans do not have a linear perception of sound, such that the difference between two sounds at 400 and 800Hz is more noticeable than the difference between a 7000 and 7400Hz sounds. To fill this gap, a scale where sounds are spaced in pitch as humans perceive has been constructed: the **Mel-Scale** (Figure 2.6).

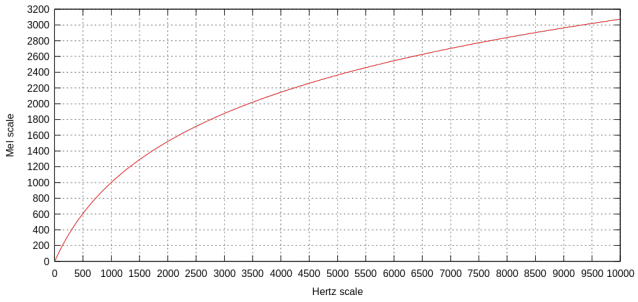


Figure 2.6: Hz-Mel Scale Conversion Curve

To construct it, i.e. apply non-linear transformations to the Hz scale, we can use the python library `librosa` and specifically the command:

```
librosa.filters.mel(sr=sr, n_fft=n_fft, n_mels=n_mels)
```

which partitions the Hz scale into bins and transforms each bin into a Mel-Scale

bin, using overlapping triangular filters depicted in Figure 2.7, while filterbanks used for the Hz to Mel conversion are shown in Figure 2.8.

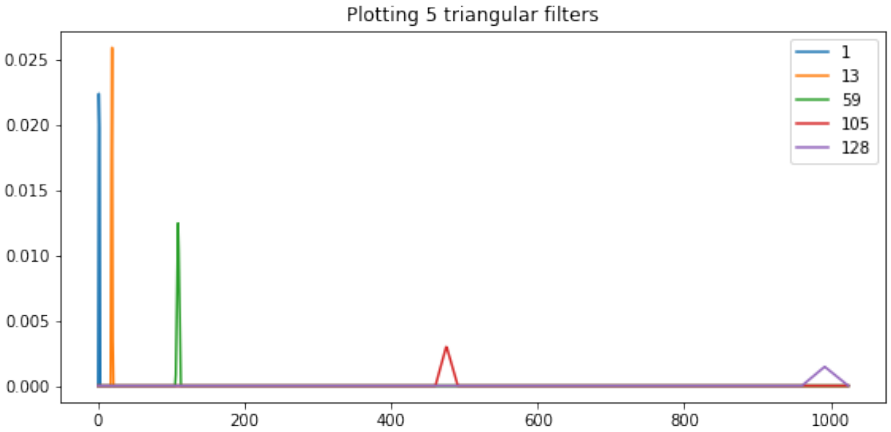


Figure 2.7: Triangular filters

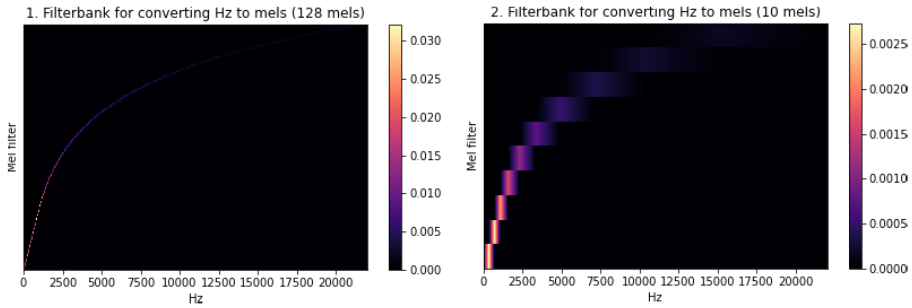


Figure 2.8: Hz to Mel Filterbanks

At this point, performing a *dot product* between the amplitude of one time window and mel bins we obtain the spectrum in orange color depicted in Figure 2.9. The blue spectrum is the same as in Figure 2.3, left plot.

After the Hz to Mel-scale conversion, the **Mel-Spectrogram** has been obtained (Figure 2.10) and it represents the input of the Deep Learning Model discussed in section 4.3.2.

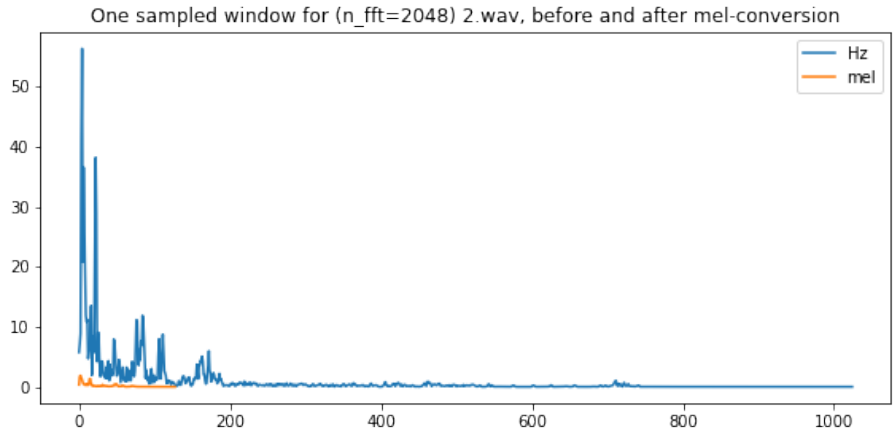


Figure 2.9: Converted spectrum **Hz to Mel**

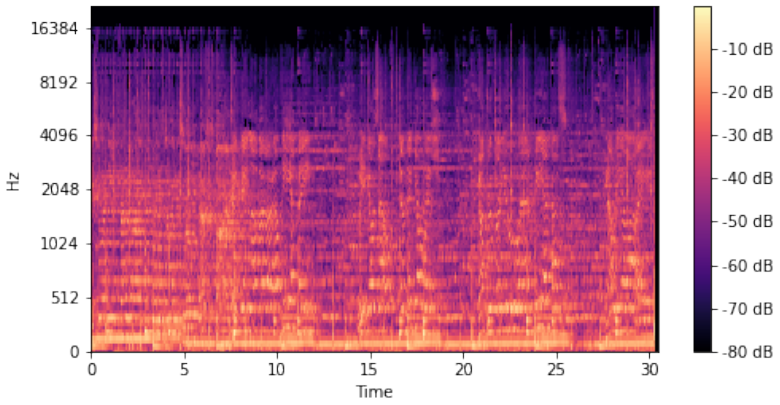


Figure 2.10: Mel-Spectrogram of '2.wav'

This 'image representation' of audio signals is exploited by Deep Learning techniques to perform audio analysis.

2.2 Image Generation

Generative Adversarial Networks have been introduced by Ian Goodfellow in 2014 [21] and they are mostly used to perform image and video generation tasks([22], [23], [24]).
They are used in the unsupervised representation learning, meaning that they do

not require any label associated with the sample during the training phase: they are able to learn *P(X)* where X is a set of features.

Vanilla GAN

'How does the GAN framework work?'

Generative Adversarial Networks are a particular type of generative models [21], able to learn the distribution of a dataset (p_{data}) and then represent an estimation of this distribution (p_{model})[25]. A standard GAN is composed by two main entities (Figure 2.11):

1. **Generator (G)**: learns to generate non-existing images (*fake*) which have to look as real as possible.
2. **Discriminator (D)**: learns to distinguish between real and fake images. **G** and **D** play an adversarial min-max game given by the loss function in the equation 2.5.

$$\begin{gathered}\mathbf{J}^{(D)} = \mathbb{E}_{x \in p_{data}} \left[log D(x) \right] \; - \; \mathbb{E}_z \left[\frac{1}{2} log(1 - D(G(z))) \right] \\ \mathbf{J}^{(G)} = -\mathbf{J}^{(D)}\end{gathered} \tag{2.5}$$

Where:

- x is a real data sampled from p_{data} distribution.
- z is a random vector sampled from the latent space distribution p_z.
- **G** and **D** are the Generator and Discriminator network, respectively.

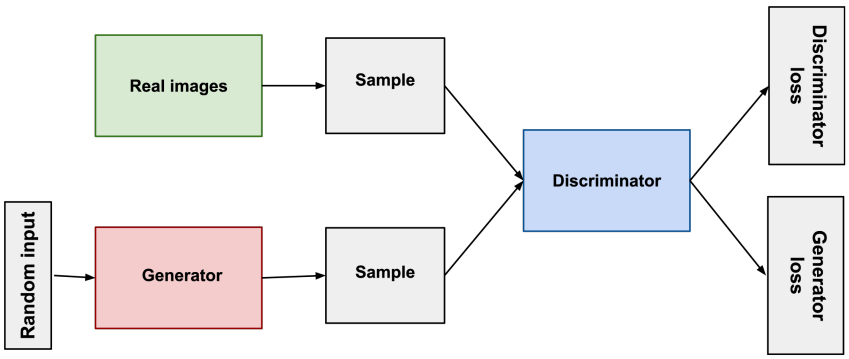


Figure 2.11: Structure of GAN, adopted from [26]

The Discriminator is a classifier trained to output two different values: when it outputs 1, it means that the image generated by the Generator is classified as *real*, while when it outputs 0 the generated sample is classified as *fake*. Its loss is evaluated taking in account how well real samples $D(x)$ and fake samples $D(G(z))$ are classified. Moreover $D(x)$ should be as close as possible to 1, $D(G(z))$, instead, to 0.
The Generator has a cost function opposite to the discriminator's one because it is trained to fool the discriminator.
To learn a generator distribution p_g over data x, **G** builds a mapping function from a prior noise distribution $p_z(z)$ to data space as $G(z;\theta_g)$ and **D**, $D(x;\theta_d)$, outputs the single scalar representing that x came from from training data rather than p_g. **G** and **D** are trained simultaneously adjusting parameters for **G** to minimize $log(1 - D(G(z))$ and adjusting parameters for **D** to minimize $logD(X)$, as if they are following the two-player min-max game with value function $V(G, D)$[27]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[logD(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[log(1 - D(G(\mathbf{z})))] \tag{2.6}$$

GAN training consists in finding Nash equilibrium of this two-players non-cooperative game [28], achieved when each player has the minimum possible cost, and this equilibrium research is one of the most challenging problem for GANs expecially when cost functions are non-convex and the space of parameters has high dimensionality.

DCGAN

*DeepConvolutional*GAN is the first major improvement in GAN introduced in [29] which demonstrates how the use of deep convolutional layers inside the architectures of the generator and discriminator leads to better perfomances together with a fast convergence. The authors of the paper [29] give the following general guidelines:

- Replace any *Pooling Layer* with *Fractional-strided Convolutions* in the generator and with *Strided Convolutions* in the discriminator.
- Use *Batch Normalization* [30] in both generator and discriminator.
- Remove *Fully Connected hidden layers* for deeper architectures.
- Use *ReLU* activation [31] for any layer in the generator but for the last, which has to use Tanh.
- Use *LeakyReLU* activation [32] for all layers in the discriminator.

Figure 2.12 shows the general architecture of the generator receiving a 100-dimensional random vector z, sampled from a uniform distribution, and outputting a 64x64 image thanks to the concatenation of fractional-strided convolutions.
The architecture of the discriminator mirrors the generator's one, receiving a 64x64 image and outputting a scalar real-number $n \in [0, 1]$.

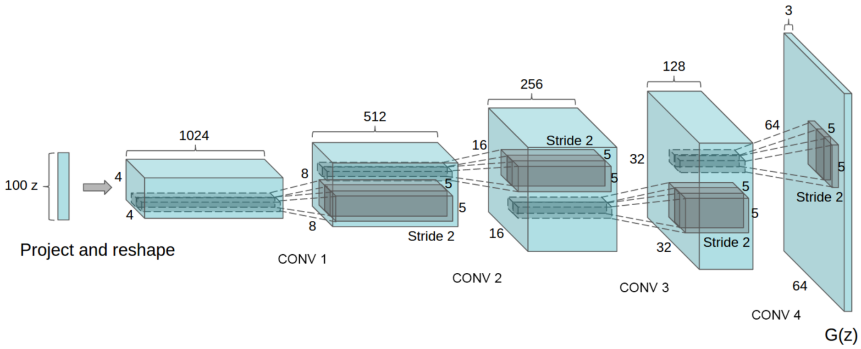


Figure 2.12: DCGAN generator adopted from [29]

Condition the generation of GANs

The paper [27] introduces the conditional version of GANs, which can be constructed by simply feeding the data y we wish to condition on to both the generator and discriminator. The data y is an additional information which lets the models learn $P(X|Y)$, leading to the generation of specific class samples resulting in a higher control on what is generated. This conditional architecture has been applied to the Vanilla GAN case, but it can be applied to other variants as well.
In the generator the input vector noise $p_z(z)$ and y are combined in joint hidden representation which can be composed in a lot of different ways thanks to the adversarial training framework.
The objective function of a two-player minimax game 2.6 becomes the following:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[logD(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[log(1 - D(G(\mathbf{z}|\mathbf{y})))] \quad (2.7)$$

The proposed structure of a *conditional*GAN is depicted in Figure 2.13, where x and y are both inputs for the discriminative function while y is combinated with z for the generator.

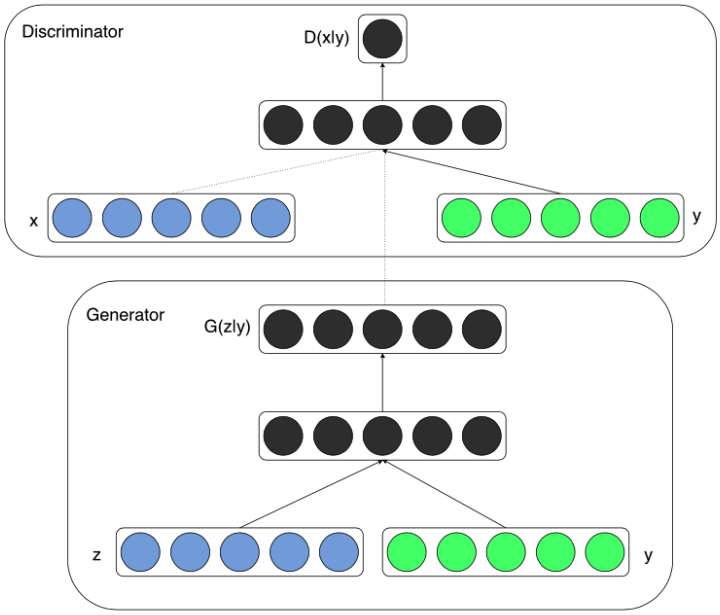


Figure 2.13: conditionalGAN architecture adopted from [27]

Class labels can be encoded and incorporated into discriminator and generator in several different ways. One of the best practices concerns in using an embedding layer followed by a Fully-Connected layer with a linear activation that scales the embedding to the size of the image before concatenating it in the model as an additional channel or feature map [33].

Chapter 3

Data and evaluation

3.1 Image perspective

The major interest lies in generating artworks by conditioning the generation with emotions, hence a dataset containing artworks annotated with emotions it is needed. There are a lot of image-emotion datasets in literature [34], [35] but none of them fits for the thesis intent.
The dataset which holds artworks is the well-known **WikiArt** dataset [36], whose information are readable from Table 3.1, but it does not contain any emotion information.

paintings	*artists*	*art styles*	*maxWidth*	*maxHeight*	*minW*	*minH*	*max Resolution*
81,447	**1,119**	**27**	***15,530****px*	***9,541****px*	***63****px*	***50****px*	***107,327,830****px*

Table 3.1: Wikiart Dataset stats

Further researches put lights on the Panos Achlioptas et al. paper [37] which brings a new vision of Wikiart dataset by asking annotators to annotate the dominant emotion they feel for a given image and to also provide a grounded verbal explanation for their emotion choice. This dataset contains a set of signals for both the objective content and the affective impact of an image, creating associations with abstract concepts or references that go beyond what is directly visible.
The now called **ArtEmis** dataset contains 439k emotion attributions and explanations from humans on 81k artworks from wikiart.
A content-based dataset containing images descriptions in human language already exists in literature and it is called COCO [38], but with the huge difference lieing into the semantic of the utterances. COCO-captions refer to objects and actions directly

visible, while inside ArtEmis the annotators exposed a wide range of abstract semantics and emotional states associated with the concept of an object or an animal. In Figure 3.1 is possible to notice the semantic difference between ArtEmis and COCO.

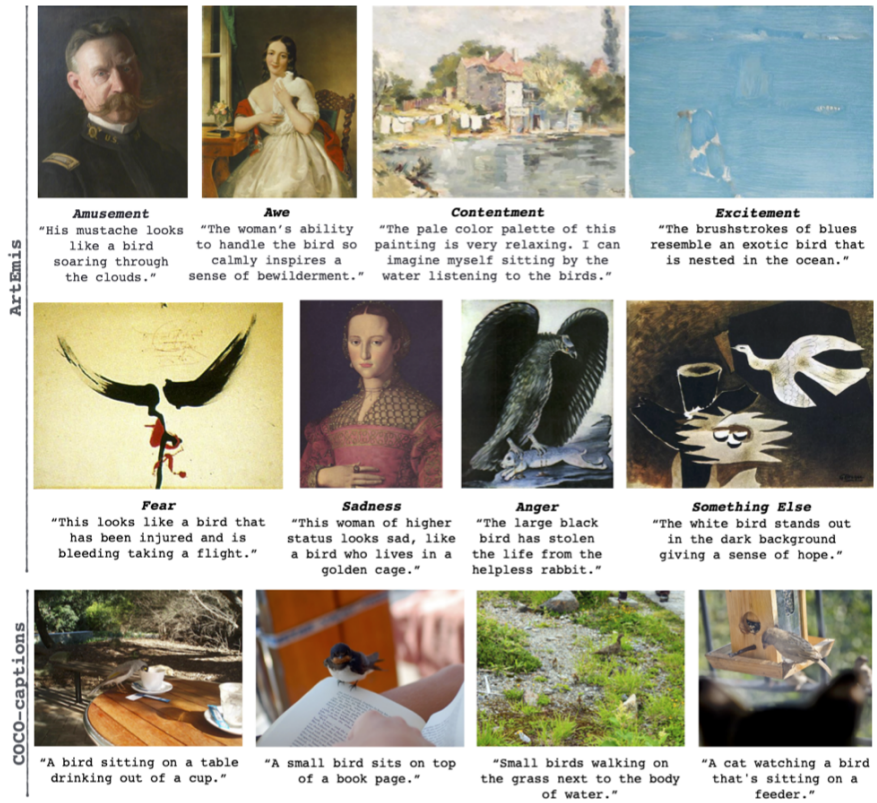


Figure 3.1: Affective explanations vs. content-based captions mentioning the word 'bird'

Data Collection

Each artwork was annotated by at least five annotators who were asked to express their dominant emotion through Amazon's Mechanical Turk GUI together with the reason related to the choice. If none of the given emotions was fine to describe what the annotator felt, it was possible to choose the *something else* option, providing an explanation as well.
In total the authors of [37] collected 439,121 explanatory utterances and emotional responses resulting in 36,347 distinct words written by 6,377 annotators.

label_id	*emotion*	*color*
0	amusement	*#EE82EE*
1	awe	*#FFFF00*
2	contentment	*#DC143C*
3	excitement	*#000080*
4	anger	*#F0E68C*
5	disgust	*#C0C0C0*
6	fear	*#696969*
7	sadness	*#228B22*

Table 3.2: ArtEmis Dataset Labels

The most important side of this dataset, for this thesis task, is related to the emotion categorical label (Table 3.2). Since each artwork has at least 5 annotators it was possible to build an **emotion-histogram** for each painting, given a basis for a possible multi-label classification task: the classifier could attribute more than one emotion class to the painting, each with its weight.
In section 3.2 a single 30s excerpt of music is mapped with one class emotion, hence for the generative experiment the ArtEmis labels were simplified such that from each emotion-histogram only the most important emotion has been selected: images with ambiguous dominant emotion (i.e. two or more emotions with same value) have been discarded. The resulting dataset for the generative phase, has the following characteristics (Table 3.3):

label_id	***emotion***	***number of paintings***
0	*amusement*	**5893**
1	*awe*	**9332**
2	*contentment*	**24938**
3	*excitement*	**3283**
4	*anger*	**290**
5	*disgust*	**1884**
6	*fear*	**5955**
7	*sadness*	**6665**

Table 3.3: Wikiart dataset usable for cDCGAN training

3.2 Music perspective

Music shares emotions through time passing, images are the representation of an instant of time: 'How do we have to classify the emotions from music?'
This first question was answered after some researches in literature on existing music datasets after having understood how music annotations are generally collected. The most interesting dataset been found is the DEAM dataset [39], containing *744* .mp3 files. Each file has Valence-Arousal values annotated every 500ms and lasts approximately 45 seconds. The major interest about this dataset is related to the annotations, both for their nature and for the modality of their collection, because they could allow to perform a time-continuous approach to classify emotions from music.

'What Valence and Arousal are?'

From a mathematical point of view they are values belonging to $\mathbb{R}$, from a psychological one they are ways to describe an information:

Valence refers to positive or negative affectivity;
Arousal measures how calming or exciting the information is.

In other words, *Valence* indicates the positivity or negativity of an emotion, (i.e. *happiness* has a positive valence whereas *fear* has a negative one) while *Arousal*, ranging from excitement to relaxation, is high being related to *anger* and low referring to *sadness*.

Data Collection

The 775 songs were annotated by crowd workers on Amazon Mechanical Turk through a GUI accessible via Browser while listening to the music. Each song was annotated once for Valence and once for Arousal separately, producing two distinct `csv` files containing V-A values of the song excluding the first 15 seconds: at the beginning of the clip the annotations were not stable. Annotators were asked to write down any personal emotion but the one music intended to induce. The continuous annotations were collected at a different sampling rate, depending on browsers and computer performances, thus after having collected 10 annotations

for each song, they were re-sampled at 2Hz and their values were squashed into the interval $[-1, +1]$.

Since the annotations were taken at a sampling-rate of 2Hz (Figure 3.2 shows the first ten lines of `arousal.csv`), it could be possible to predict both (A, V) $\in [-1, +1]$ in the continuous dimension, approaching it as a regression problem, as said in the Related Works section 2.1 and as done by Orjesek et al. [11] who proposed a novel way to handle to the regression-approach MER task by stacking a Convolutional and a Recurrent layer for feature mining and *V-A* values prediction.

song_id	sample_15000ms	sample_15500ms	sample_16000ms	sample_16500ms	sample_17000ms	sample_17500ms
2	-0.109386	-0.1149416	-0.1164126	-0.1186133	-0.1264569	-0.13319875
3	-0.1108463	-0.1239729	-0.1311026	-0.1359562	-0.1407755	-0.1446635
4	0.2223271	0.1794461	0.1783881	0.1840561	0.1760421	0.1787201
5	-0.2556127	-0.2515787	-0.2519577	-0.2511237	-0.2507627	-0.2519567
7	0.4642337	0.4607887	0.4609907	0.4610457	0.4572397	0.4657017
8	-0.1381319	-0.1369336	-0.1404691	-0.1389876	-0.1381456	-0.1407516
10	-0.129007	-0.1329306	-0.1386527	-0.139502	-0.142657	-0.151037

Figure 3.2: head of arousal.csv

Keeping in mind that the ArtEmis dataset [37] has categorical emotion labels, before trying to implement this solution, (i.e. handling the problem as a regression one) the music dataset was further explored to understand if changing the regression-approach to the categorical one would lead to a loss of annotations' details. For each song, all its V-A values have been plotted in a 3D Scatter Plot in Figure 3.3 using *time (t)* as the third dimension, ending up with the evidence that *the emotions* ***do not change*** *inside an interval of 30 seconds of music.*
This assertion could be false for the *classical music* case, but it can be applied on this dataset because it does not contain any classical music content but rock, R&B, pop one.

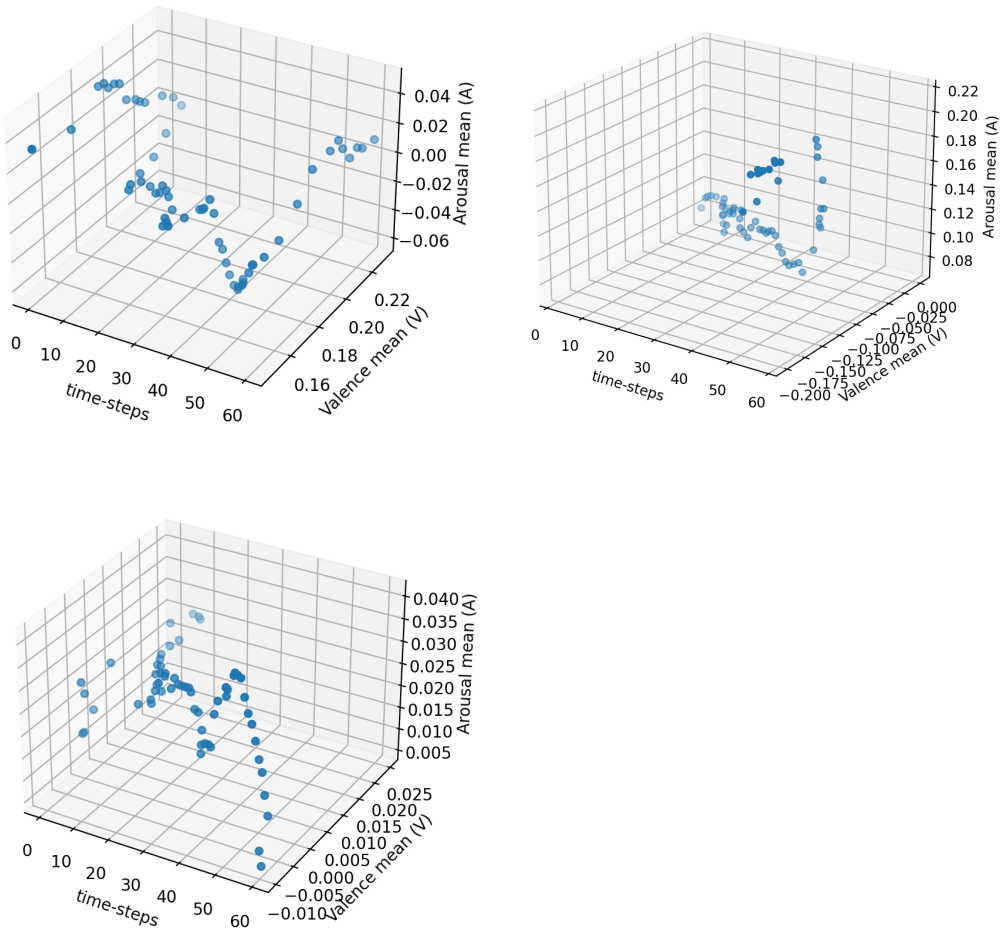


Figure 3.3: 3D Scatter of A-V values for 3 different songs

3.3 Music-Image labels alignment

Taking head of these insights an amount of time was spent to investigate how to change the type of information descriptors such that the continuous prediction could be moved to a categorical one, taking in account emotions instead of their representation through V-A values. This shift was necessary to have the two image-emotion and the music-emotion datasets, aligned on the same number and type of categorical emotion-labels.
Going further with researches it has been found that in 1980 the psychologist James Russell described the Circumplex Model of Affect (Figure 3.4) which represents the affective state, arising from the behavior of two independent neurophysiological

systems (the arousal and the valence systems), into a goniometric circumference[40].

Letting **valence** = **v** be a generic point on the *x-axis* and **arousal** = **a** be a generic point on the *y-axis* the equation 3.1 mathematically describes the Russell's Circumplex model of Affect circumference.

$$\mathbf{v}^2 + \mathbf{a}^2 \leqslant \mathbf{1} \tag{3.1}$$

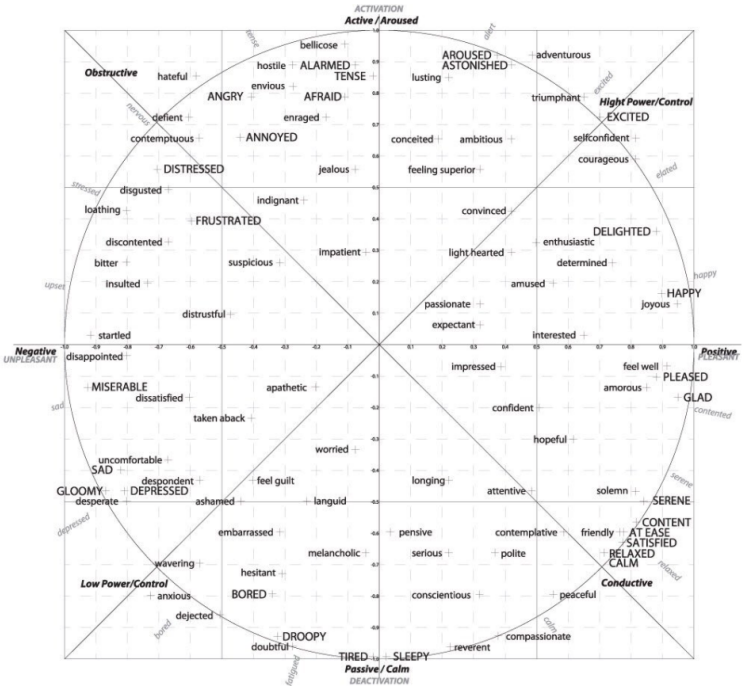


Figure 3.4: Russell Model of Affect adopted from [40]

According to the Russell's framework [40] we can consider Valence and Arousal as the pair (x, y) identifying a point in the (V-A) space: the point is a generic emotion.
Referring to the ArtEmis emotions collected in Table 3.2 and looking at the Russell Model in Figure 3.4, each ArtEmis emotion was manually provided with a pair of coordinates, in order to be identified inside the V-A space. (Table 3.4)

label_id	*emotion*	*v*	*a*
0	amusement	**0.55**	**0.2**
1	contentment	**0.82**	**-0.56**
2	awe	**0.42**	**0.89**
3	excitement	**0.7**	**0.72**
4	anger	**-0.42**	**0.78**
5	disgust	**-0.67**	**0.49**
6	fear	**-0.11**	**0.78**
7	sadness	**-0.82**	**-0.4**

Table 3.4: ArtEmis emotion Coordinates in V-A Russell Space

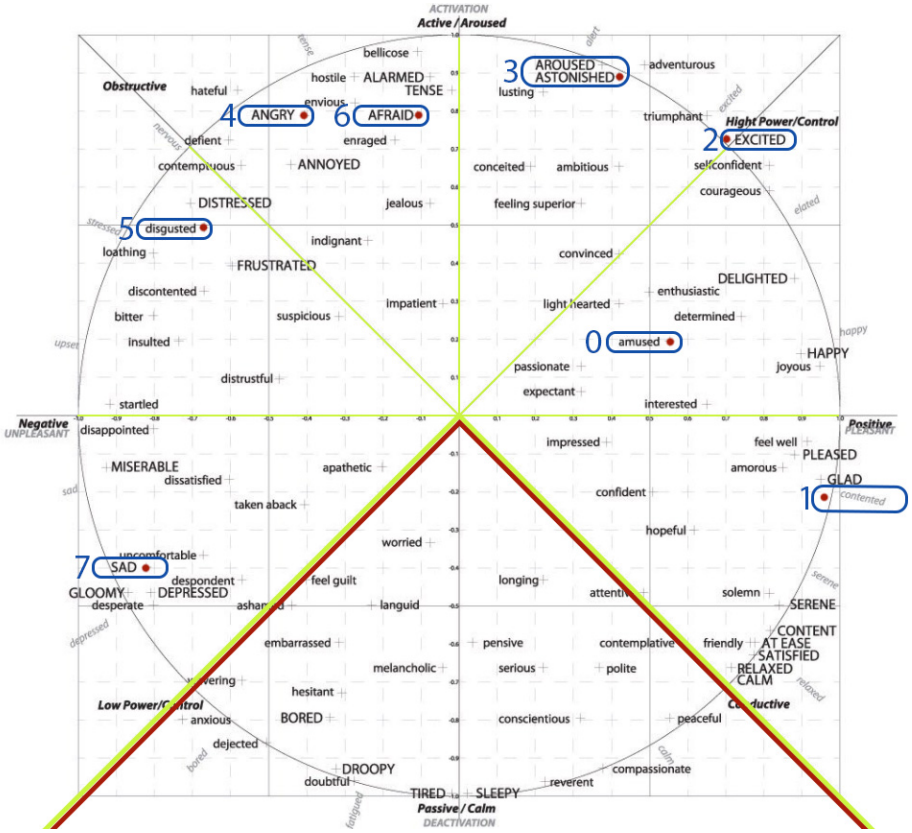


Figure 3.5: Map-Function abstract view on Russell Model of Affect [40]

After having manually pointed the eight ArtEmis emotions inside the Russell's model it was possible to write a map function whose aim was to rewrite the `csv` files containing V-A values in order to categorize emotions and shift the point of view: considering the music classification problem as a multi-class task and

together aligning the music dataset labels with the image dataset ones.

`va2emotions.py`
The plane was divided into circular sectors, one every 45º (Figure 3.5) and the following algorithm was applied:

Algorithm 1 VA values to Categorical Emotions conversion.

procedure VA2EMOTION_ATAN2(v, a)
  ▷ v and a are the values of the j^{th} 500ms interval of song i
  ▷ Check sign
  $sign = v \cdot a$
  if $sign > 0$ **then**
    $\theta \leftarrow atan2(a, v)$
    if $a > 0 \wedge v > 0$ **then**
      if $\theta < 45$ **then** ▷ Q1.1
        $emotion \leftarrow' amusement'$
      else ▷ Q1.2
        $emotion \leftarrow get\_nearest(v, a)$ ▷ Awe or Excitement
      end if
    else $a < 0 \wedge v < 0$ ▷ Q3
      if $\theta < -135$ **then** ▷ Q3.1
        $emotion \leftarrow' sadness'$
      else ▷ Q3.2
        $emotion \leftarrow' sadness'$ ▷ NOT USED
      end if
    end if
  else
    $\theta \leftarrow atan2(a, v)$
    if $v < 0$ **then** ▷ Q2
      if $\theta < 135$ **then** ▷ Q2.1
        $emotion \leftarrow get\_nearest(v, a)$ ▷ Afraid or Angry
      else ▷ Q2.2
        $emotion \leftarrow' disgust'$
      end if
    else $a < 0$ ▷ Q4
      if $\theta < -45$ **then** ▷ Q4.1
        $emotion \leftarrow' contentment'$ ▷ NOT USED

```
            else                                                    ▷ Q4.2
                emotion ←′ contentment′
            end if
        end if
    end if
end procedure
```

High level algorithm 1 shows the implementation of the map function used to remap all the *Valence-Arousal* values into categorical *emotions* ones, moving the point of view considering this as a multi-class classification task, hence having aligned music labels with images ones.

Music Dataset description

At this point some statistics have been calculated for the music dataset [39] from the two `csv`*s* files outputted by algorithm 1:

1. `music_emotions_labels.csv`
2. `music_single_emotion_labels.csv`

NOTE: Artemis label 'something_else' was not considered in this process.

For each song a frequencies array was built, then summing up all arrays the global frequencies array was created, to give a description of the dataset from a categorical emotion point of view.
Since an emotion-frequencies array for each song is available, it was possible to easily notice how emotions change inside a single song. Despite some songs present at most two distinct, non-conflicting emotions, the overall trend is that a 30s excerpt of music corresponds to one distinct emotion, as said at the end of section 3.2.
The *.mp3* files were not all sampled at the same frequency and did not have the exact same length, which approximately is around ~45 seconds. During the pre-processing step some information were collected, and they are presented through the followings charts.

- **Pie** chart (*Figure 1.1*) shows the percentage associated with each emotion w.r.t the entire dataset. The legend shows the map between emotion and color.
- **Bar** chart (*Figure 1.2*) shows the same information, but giving us the number of bins associated with a particular emotion, always w.r.t the entire dataset, composed by *45,384* bins (*i.e.* slices).

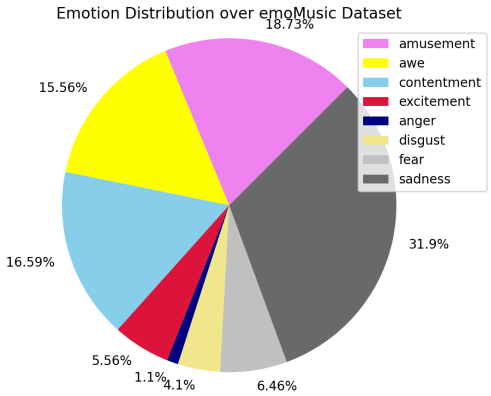


Figure 3.5.0 emotion percentages Pie

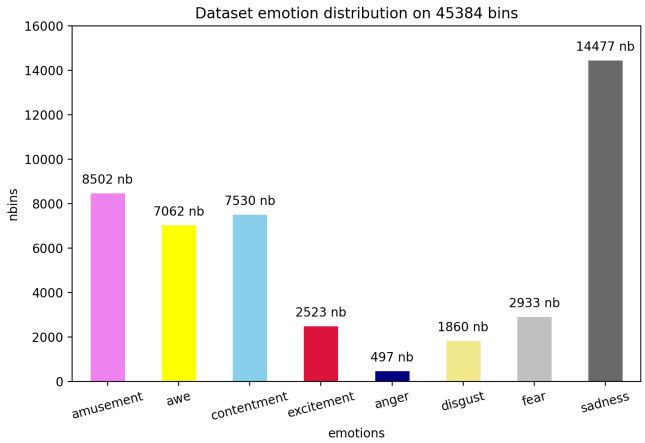


Figure 3.5.1 emotion distribution bar

Figure 3.6: The pie chart refers to percentage of emotions characterizing the dataset. The bar chart shows the number of slices in which an emotion is present.

3.4 Audio Preprocessing Pipeline

This section illustrates the four-stages-pipeline data were fitted in.

- **Resampling**
- **Padding**
- **Clipping**

- **Trimming**

All the work aims to have raw data arranged in such a way we can match exactly the `.csv` informations.
We want to realize two reading modes of the dataset: the first utilized to fit the network with an input shape that is made up of a number of samples which matches a 500 milliseconds time interval, and the second to fit the network with the whole 30s excerpt.
Original annotations .csv have a number of columns equal to 61, starting from *sample_15000ms* ending to *sample_45000ms*. Each column name wraps the value of Valence (Arousal in the other case) calculated as the average Valence in the time interval centered in the column name timestamp: it means that the first column refers to the 500ms raw input going from *14750ms* to *15250ms* and so on.
Pre-processing pipeline starts with **Resampling** in which each song.mp3 is converted to a *.wav* file with a samplerate of *44100Hz*.
Then the audio length is checked by dividing its duration in seconds by 500ms and converting time into samples, obtaining a length multiple of one slice: if the length is less than a multiple of 500ms = 22050 samples, it is **padded**.
At this point the song is clipped considering the interval *[14,750 - 45,250] ms*.
The pipeline is made on-the-fly and if Dataset is in `slice_mode` the **Trim** stage takes place and each song is divided in slices, all of the same 500ms length.
Each song:

- **total samples**: 1,345,050 each is of type: `np.int32`
- **number of slices**: 61
- **samples per slice**: 22,050

Whole dataset:

- **total songs**: 744
- **total slices**: 45,385
- **total samples**: 1,000,717,200 ~25.21h

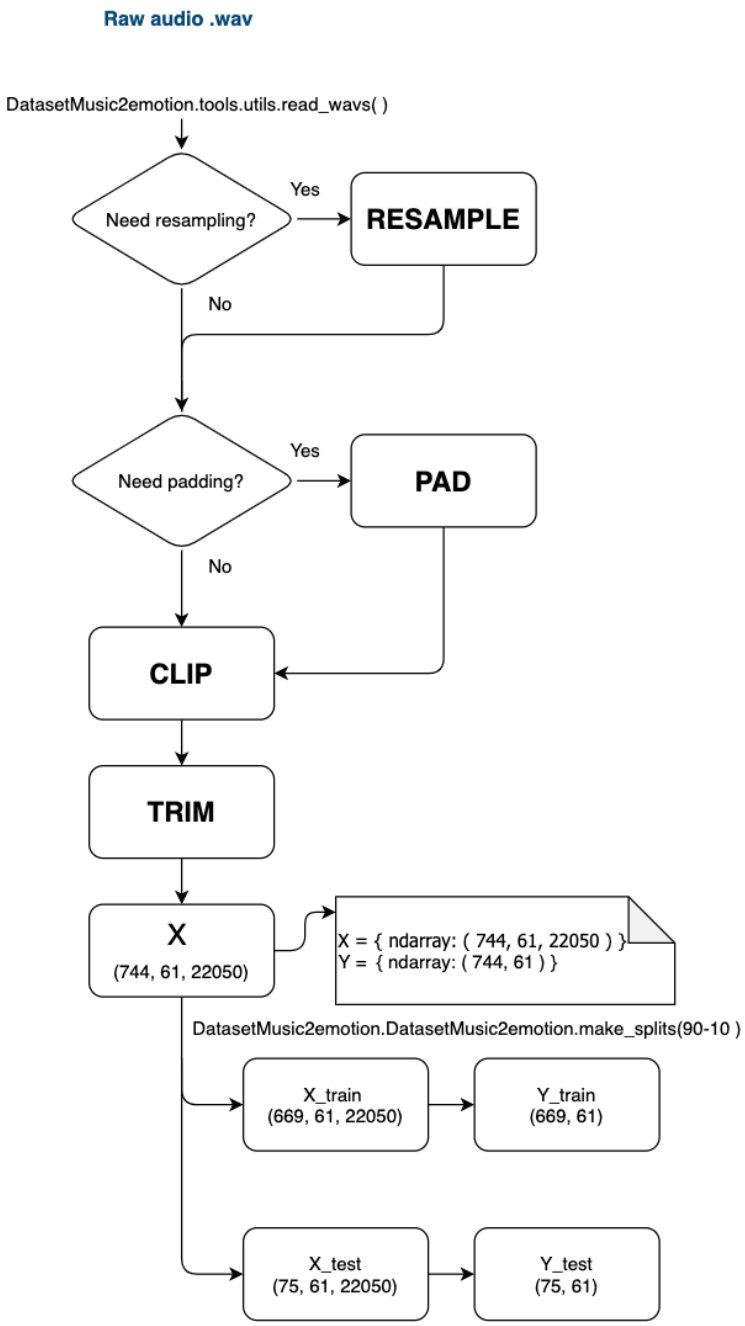


Figure 3.7: Preprocessing pipeline: X and Y are part of the emoMusicPTDataset object

3.5 Evaluation methods

3.5.1 Music perspective

Each classifier is a multi-class problem solver. To evaluate the performances, despite the overall *accuracy* has been the main metric to perform the *model selection*, other three metrics, separately for each class, has been calculated.

- *precision*
- *recall*
- *f1-score*

Before explaining what these metrics are, it is worth to give a brief introduction. For a binary classification task it is possible to build a table like this: where the

		Predicted	
		Negative	**Positive**
Actual	**Negative**	*True Negative (TN)*	*False Positive (FP)*
	Positive	*False Negative (FN)*	*True Positive (TP)*

Table 3.5: Binary classification

rows refer to the ground truth label while the columns refer to the predicted labels, moreover we are able to specify how an instance is classified by the model and thus calculate some different metrics to evaluate it.
When we have to deal with a *multi-class* classification problem we cannot refer to this table while build a confusion matrix is the right solution.
Following table is a concrete output from this thesis work. It is the TorchM18 model's output used for the raw-audio classification task processing the dataset without slice mode turned on and using a kernel size of 880 audio samples, kernel shift of 440 and a number of kernel features maps equals to 512.

	0	1	2	3	4	5	6	7
0	4	2	4	3	0	0	1	3
1	1	0	0	0	0	0	0	0
2	1	0	3	1	1	0	1	2
3	0	0	0	0	0	0	0	0
4	0	1	1	0	0	0	0	0
5	1	1	3	0	0	0	1	0
6	2	1	1	0	0	1	0	1
7	5	6	1	0	0	2	2	18

Table 3.6: TorchM18 confusion matrix example

Since we don't have Positive/Negative classes we can find TP, TN, FP and FN for each individual class. An example is given for **class 0**, *'amusement'*:

- $TP = 4$
- TN = sum of all elements except class' row and column = 48
- FP = row except TP = 13
- FN = column except TP = 10

$\forall$ *class* $c \in C = \{0, 1, 2, 3, 4, 5, 6, 7\}$ we can define:

Precision

Answers the question *'How many selected items are relevant?'*

- $Precision_c = \frac{TP_c}{TP_c + FP_c}$

Recall

Answers the question *'How many relevant items are selected?'*

- $Recall_c = \frac{TP_c}{TP_c + FN_c}$

f1-score

It is the harmonic mean between Precision and Recall.

- $f1\text{-}score_c = \frac{2 \cdot TP_c}{2 \cdot TP_c + FP_c + FN_c}$

This metric is preferable when the dataset has an **imbalanced** *class distribution.*

3.5.2 Image perspective

The evaluation of a GAN model is not straightforward as the evaluation of a generic classifier model. Classifiers have an objective loss to minimize so they can objectively be assessed also from just the Loss value, whereas the GAN models are composed by two entities, Generator and Discriminator, which are trained together and their objective is to maintain an equilibrium.
The most intuitive way to evaluate a GAN model is to manually visualize and examine generated samples, human can certainly give the best type of evaluation, but with this approach there will be a lot of limitations thus although this could seem the best and simplest way to assess a model, it is discarded. Evaluating the quality of generated images with human vision is expensive and cumbersome, biased, difficult to reproduce and does not fully reflect the capacity of models[41].

Qualitative Metrics

These measures are not numerical and they leverage both human subjective evaluation and evaluation via comparison.
The following list wants to point what are the most common qualitative metrics used for assess a GAN model[41]:

- Nearest Neighbors: to detect overfitting, generated samples are shown next to their nearest neighbors in the training set
- Rapid Scene Categorization: in these experiments, participants are asked to distinguish generated samples from real images in a short presentation time (i.e. 100ms) to perform the *real vs fake* scenario
- Preference Judgment: Participants are asked to rank models in terms of the fidelity of their generated images
- Mode Drop and Collapse: Over datasets with known modes, modes are computed as by measuring the distances of generated data to mode centers
- Network Internals: Regards exploring and illustrating the internal representation and dynamics of models (i.e. space continuity) as well as visualizing learned features

Quantitative Metrics

Authors of [41] give twenty-four quantitative techniques for evaluating GAN models, and they refers to the calculation of specific numerical scores used as quality indicators of generated images.
Listing all metrics is pointless, but it's worth mentioning the most used:

- Inception Score (IS)
- Modified Inception Score (m-IS)
- Fréchet Inception Distance (FID)

Fréchet Inception Distance (FID) was introduced by authors of [42] and it evaluates the distance between the real and fake distributions for each feature expressing it with a scalar number (equation 3.2). The FID metric is the squared Wasserstein metric between two multidimensional Gaussian distributions:
$\mathcal{N}(\mu, \Sigma)$, the distribution of some neural network features of the images generated by the GAN, and $\mathcal{N}(\mu_w, \Sigma_w)$, the distribution of the same neural network features from the "world" or real images used to train the GAN.
It can be computed from the mean the the covariance of the activations when the synthetized and real images are fed into the Inception network as:

$$d^2((\mu, \Sigma), (\mu_w, \Sigma_w)) = \|\mu - \mu_w\|_2^2 + tr(\Sigma + \Sigma_w - 2(\Sigma\Sigma_w)^{1/2}) \tag{3.2}$$

Where:

- (μ, Σ) are the feature-wise mean and covariance matrix for real samples
- (μ_w, Σ_w) are the feature-wise mean and covariance matrix for the generated samples

Chapter 4

Experimental Setup and Results

4.1 Introduction

This chapter contains details about the conducted experiments, in particular:

- *section* **4.3** refers to the two different modalities used to handle MER task:
  - *section* **4.3.2** refers to the solution using **Mel-spectrogram** as input of a Convolutional Neural Network
  - *section* **4.3.3** refers to the solution using **raw-audio** as input of three different Convolutional Neural Networks
- *section* **4.4** refers to the **DCGAN** and **cDCGAN** baselines implemented to conduct the generation process

4.2 GitHub Repo

The code repository is available at *creativeAI: visualize music emotions*.
The code has been thought to be modular, easier to use and maintain.
The repository has two *'sides'* one for image and one for music and, after being cloned, with these commands it could be installed as a python module with the commands[1]:

[1] Working example for Google Colab

```
%cd /content
!git clone https://github.com/mHead/creativeAI.git
%cd /content/creativeAI
%pip install -e .
os.environ['PYTHONPATH'] += ":/content/creativeAI"
```

At this point it possible to `import creativeAI` and by means of the `.` operator it is possible to import all the modules inside the Package just installed.

Due to Google Colab time and resources limitations all experiments have been made on HPC@polito Legion Cluster[2] thus a `Command Line Intefrace` approach was developed in order to create and launch a `job` for the `SLURM` scheduler.
After some `#SBATCH` directives for the scheduler the command called by the `bash` script `run_legion.sh` is:

```
python3 ${code}$ -v -r ${repo_root_legion}$ -task
```

Where:

- `-v` stands for "verbose"
- `-r` stands for "repository" and it must be followed by the root of the cloned repository
- `-task` has to be replaced by one of the element of the list: `[-raw || -mel || -dcgan || -cdcgan]`

High Level Pipeline
For convenience, the High Level Pipeline picture is again shown below:

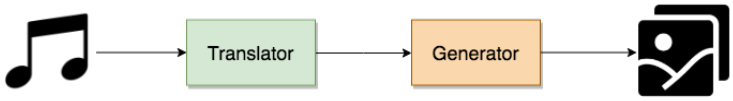


Figure 4.1: High Level Pipeline

[2]Risorse di calcolo fornite hpc@polito, progetto di Academic Computing del Dipartimento di Automatica e Informatica presso il Politecnico di Torino (http://www.hpc.polito.it)

4.3 creativeAI.musicSide

This package focuses on the definition and training of the **Translator** block depicted in Figure 4.1.
Translator is an audio classifier, in charge of extract emotions as features from the music, regardless its representation.
For its realization four different models have been defined, trained and compared, one for the **MEL-spec** MER approach, and three for the **RAW-audio** MER approach.
To instantiate the dataset, the models and perform their training, the following classes are provided:

- **Shared Dataset Object**
  - `DatasetMusic2emotion.emoMusicPTDataset`
  - `DatasetMusic2emotion.emoMusicPTSubset`
  - `DatasetMusic2emotion.emoMusicPTDataLoader`
- **MEL-spec task:**
  - `Model.MEL_baseline`
  - `Model.MEL_Runner`
- **RAW-audio task:**
  - `Model.TorchM5`
  - `Model.TorchM11`
  - `Model.TorchM18`
  - `Model.Runner`

4.3.1 Shared Dataset Object

The dataset which has been taken into account is the one described in section 3.4 whose labels have been previously remapped using algorithm 1 thus is could be thought, thus instantiated, in two different modes by means of the boolean `slice_mode` class parameter:

- `emoMusicPTDataset(slice_mode=False, env=bundle)` for **song-level** samples
- `emoMusicPTDataset(slice_mode=True, env=bundle)` for **slice-level** samples

The dataset class extends the *map-style* `torch.utils.data.Dataset` and has the following peculiarities:

When `slice_mode=True` the `emoMusicPTDataLoader.__getitem__` returns a 500ms slice of the song of type `torch.Tensor([1, 22050])` with its associated categorical emotion label and its `__len__` is equal to `744 * 61 = 45,385` samples (61 slices per song), whereas when `slice_mode=False` the dataloader returns the whole 30s excerpt of the song of type `torch.Tensor([1, 1345050])` with the global categorical label.
The default mode is the **raw-audio** with `slice_mode=False`. Different experiments have been conducted, resulting in better performances when the dataset is not in `slice_mode`. A visual representation of two modalities is depicted in Figure 4.2.

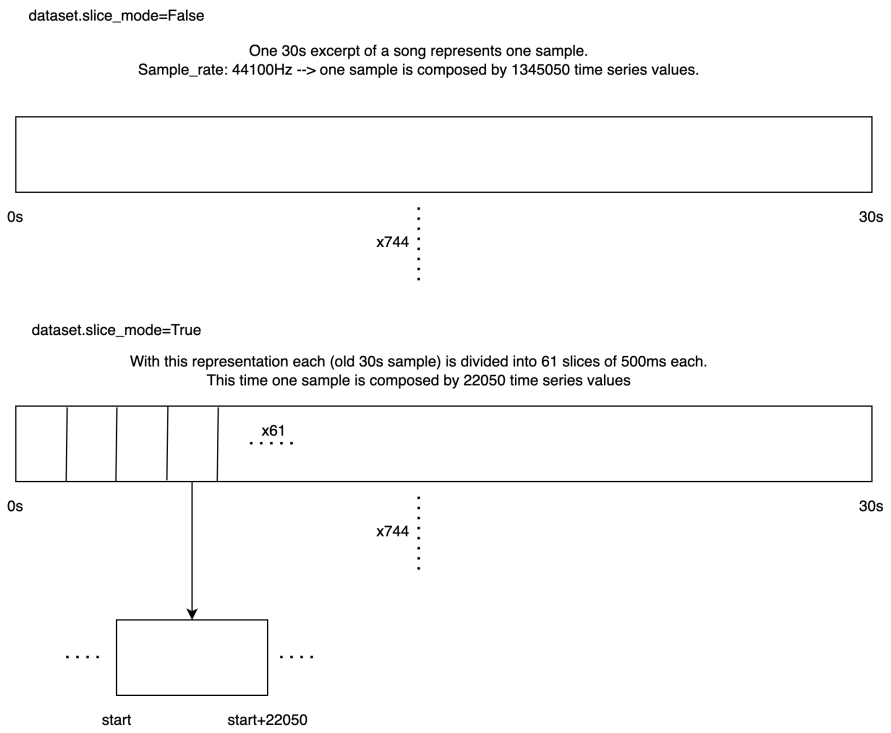


Figure 4.2: emoMusicPTDataset raw-audio **slice modes**

To preserve the dataset classes distribution both in training and test set the Subsets indexes (90%-10%) were calculated using:

`sklearn.model_selection.StratifiedShuffleSplit`

which ensures the two classes distributions to be equals. Figure 4.3 shows the classes distribution, evidencing that the dataset is strongly **unbalanced**.
NOTE: the total number of 30s excerpts is 744, leading to have 669 samples for TrainingSet and 75 for TestSet

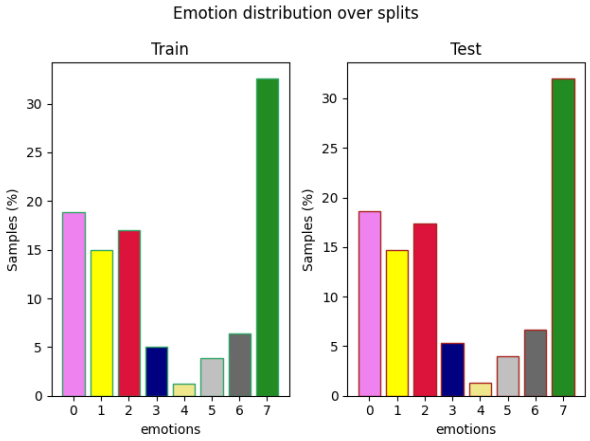


Figure 4.3: Train-Test classes distribution

To cope with the **unbalanced** dataset problem the CrossEntropyLoss criterion method is provided with a `np.ndarray` of size `n_classes` which contains values to perform a classes probability re-weighting which aims to solve the samples imbalance.

Re-Weighting modes

As we can read in Table 4.1, 4.2 column *cwm* we have three values: **None**, **freq**, **max**. The first value points that no re-weigthing is performed, while the other two are different in the implementation.

I. Freq: Frequency re-weight Mode

Given the frequency class distribution vector:
[140, 11, 127, 38, 9, 29, 48, 242] $\rightarrow sum = 744$

The **freq-weights-vector** is build as: $w_i = 1/f_i$ obtaining the following vector of weights:
$[0.00714286, 0.00900901, 0.00787402, 0.02631579, 0.11111111, 0.03448276, 0.02083333, 0.00413223]$

II. Max: Max Frequency re-weight Mode

Given the frequency class distribution vector:
[140, 11, 127, 38, 9, 29, 48, 242] $\rightarrow sum = 744, max = 242$

The **max-freq-weights-vector** is build as: $w_i = max/f_i$ obtaining the following vector of weights:
$[1.72857143, 2.18018018, 1.90551181, 6.36842105, 26.88888889, 8.34482759, 5.04166667, 1]$

4.3.2 MEL-spec task

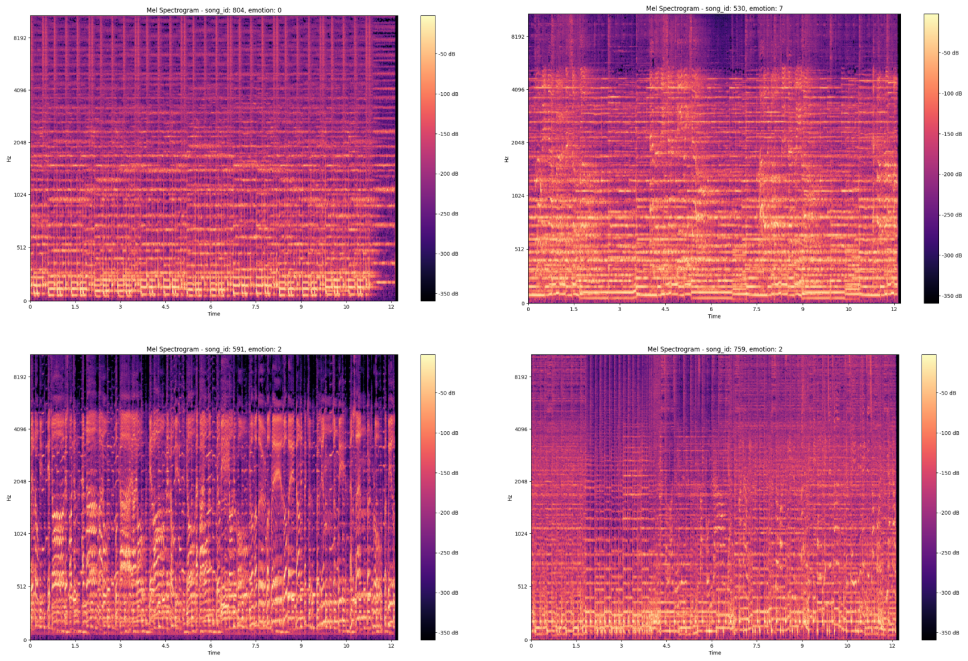
When performing the Mel-Spectrogram task the Dataset constructor is provided with boolean `melspec=True` variable and the `bundle` object brings information on how the Mel-Spectrograms have to be constructed. In particular, for this task, `bundle` object will contain the following additional informations:

- `n_fft: 5120`
- `hop_length: 2560`
- `n_mel: 526`
- `sample_rate: 44100`

The `emoMusicPTDataset.__init__()` function will define a transformation used by the `__getitem__` every time a new sample is picked from the dataset. This transformation is composed by the following operations:

- `S = librosa.features.melspectrogram(waveform, sr, n_mel, n_fft, hop_length, fmax=8000)`
- `S = librosa.power_to_db(S, ref=np.max)`
- `S = torch.from_numpy(S)`
- `S = S.expand(3, -1, -1)`
- `S = torchvision.transform.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])(S)`

The input of the transformation is a `np.ndarray([1, 1345050]`, i.e. (n_channel, time), which is then passed to the `power_to_db()` function and its output is converted in a `torch.Tensor([1, 526, 526])` expanded to have the three RGB channels as the first dimension, resulting in a `torch.Tensor([3, 526, 526])`, i.e. (..., n_mels, time) : (n_channels, H, W), which after being Normalized is fed into the architecture.
It's worth noticing that no mel-spectrogram image is stored on the disk, because every time the DataLoader pick an .mp3 sample it generates this representation on-the-fly and use it as input for the Network.
Figure 4.3.2 shows four different on-the-fly generated inputs from four different 30s excerpts.



The chosen architecture for this task is the *pretrained* version of the **ResNet** (Residual Network), an artificial neural network (ANN) that mimics the pyramidal cells in the cerebral cortex by means of the use of skip connections, or *shortcuts*, to jump over some layers in order to avoid the problem of **vanishing gradient** which basically originates from activation functions together with the Network depth growth: gradients calculated during the **backpropagation** would be so small that the weights and biases of the initial layers will not be updated effectively during training.
There are small ResNet architectures such as ResNet18 and ResNet34 and big ones like ResNet50, ResNet101, ResNet152.
Figure 4.4 shows the difference between a standard block and a residual one.

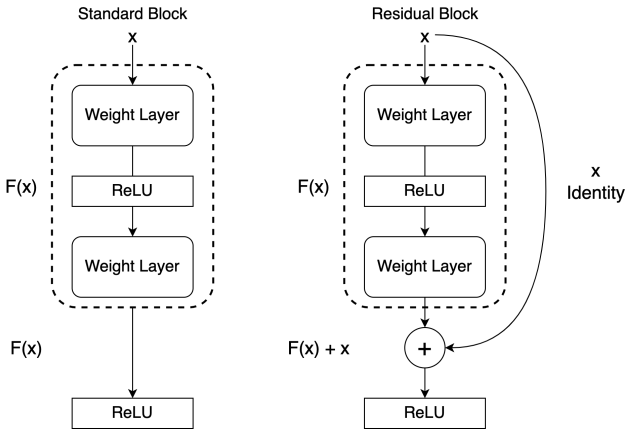


Figure 4.4: Stardard vs Residual Block

The main advantages this architecture brings are related to an overall speedup of the training phase, a better handling of the vanishing gradient problem and authors of [43] have empirically demonstrated that this kind of network is easier to be optimized thus it can better exploit depth to gain accuracy. Figure 4.5 represents the ResNet18 architecture with *skip-connections* on top.

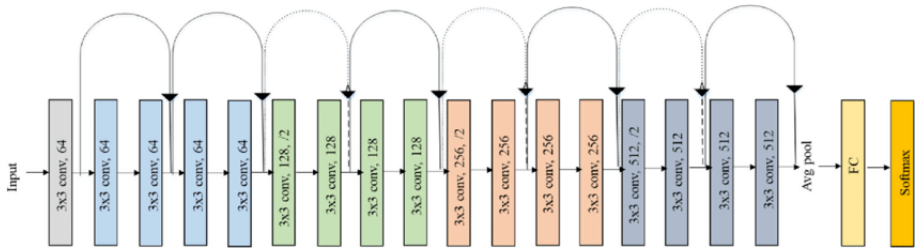


Figure 4.5: ResNet18 architecture

Training Settings

Training phase was conducted performing a grid search over the values depicted in Table 4.1 and 4.2 to select the best model basing on the overall accuracy on test set.
To train a general Deep Neural Network the basic elements that are needed are the following:

Loss function: used to measure how wrong are the network's predictions.

Backpropagation algorithm: it is in charge of computing the gradient ofthe loss function w.r.t the weights of the network for a given *input-output* pair by means of the *chain-rule*. Gradients tell how much each weight is contributing to the loss, thus to reduce the related error the weight can be changed.

Optimizer: uses the gradients to find the network's parameters, i.e. weights and biases, which minimize the loss function.

The **optimizer** chosen for the ResNet training is the *Adam Optimizer*, while the **Loss** function is the *Cross Entropy Loss*.

Cross Entropy

For each class, a **loss value** is calculated with equation 4.1 where X is the $class_i$, $p(X)$ is the probability of X in *Ground-truth* vector and $q(X)$ is the probability of X in *Prediction* vector:

$$L(X) = -p(X) \cdot log\,[q(X)] \tag{4.1}$$

Then all class losses are summed up together, giving the CrossEntropyLoss (eq. 4.2):

$$CrossEntropy = -\sum_{x} p(x) \cdot log\,[q(x)] \tag{4.2}$$

An example of *Ground-truth* (left) and Predicted (right) vectors is provided below:

$$\text{GT} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{P} = \begin{bmatrix} 0.8 \\ 0.1 \\ 0.1 \end{bmatrix}$$

Ground truth and Predicted vectors for 3 class example

TrainingSettings	*bs*	*ep*	*lr*	*wd*	*m*	*cwm*	*d*
I.0	32	120	0.0001	1e-5	0.8	***None***	0.2
I.1	32	120	0.0001	1e-5	0.8	*None*	0.25
I.2	32	120	0.0001	1e-5	0.8	*None*	0.5
II.0	32	120	0.0001	1e-5	0.8	***Freq***	0.2
II.1	32	120	0.0001	1e-5	0.8	*Freq*	0.25
II.2	32	120	0.0001	1e-5	0.8	*Freq*	0.5
III.0	32	120	0.0001	1e-5	0.8	***Max***	0.20
III.1	32	120	0.0001	1e-5	0.8	*Max*	0.25
III.2	32	120	0.0001	1e-5	0.8	*Max*	0.5

Table 4.1: Training Settings for ResNet18

TrainingSettings	*bs*	*ep*	*lr*	*wd*	*m*	*cwm*	*d*
IV.0	16	120	0.0001	1e-5	0.8	***None***	0.2
IV.1	16	120	0.0001	1e-5	0.8	*None*	0.25
IV.2	16	120	0.0001	1e-5	0.8	*None*	0.5
V.0	16	120	0.0001	1e-5	0.8	***Freq***	0.2
V.1	16	120	0.0001	1e-5	0.8	*Freq*	0.25
V.2	16	120	0.0001	1e-5	0.8	*Freq*	0.5
VI.0	16	120	0.0001	1e-5	0.8	***Max***	0.20
VI.1	16	120	0.0001	1e-5	0.8	*Max*	0.25
VI.2	16	120	0.0001	1e-5	0.8	*Max*	0.5

Table 4.2: Training Settings for ResNet18

Here acronyms written in tables 4.1, 4.2 columns are evidenced:

- *bs*: batch size
- *ep*: number of epochs
- *lr*: learning rate
- *wd*: weight decay
- *m*: momentum
- *cwm*: criterion weight mode[3]

[3]Explained in section 4.3.1

- d: dropout

The Training lifecycle is handled by `MEL_Runner Object` which is in charge of training the model, storing relevant data, computing statistics and plots, thus evaluating the model itself.
Figure 4.6 represents the Training lifecycle implemented in creativeAI repository.

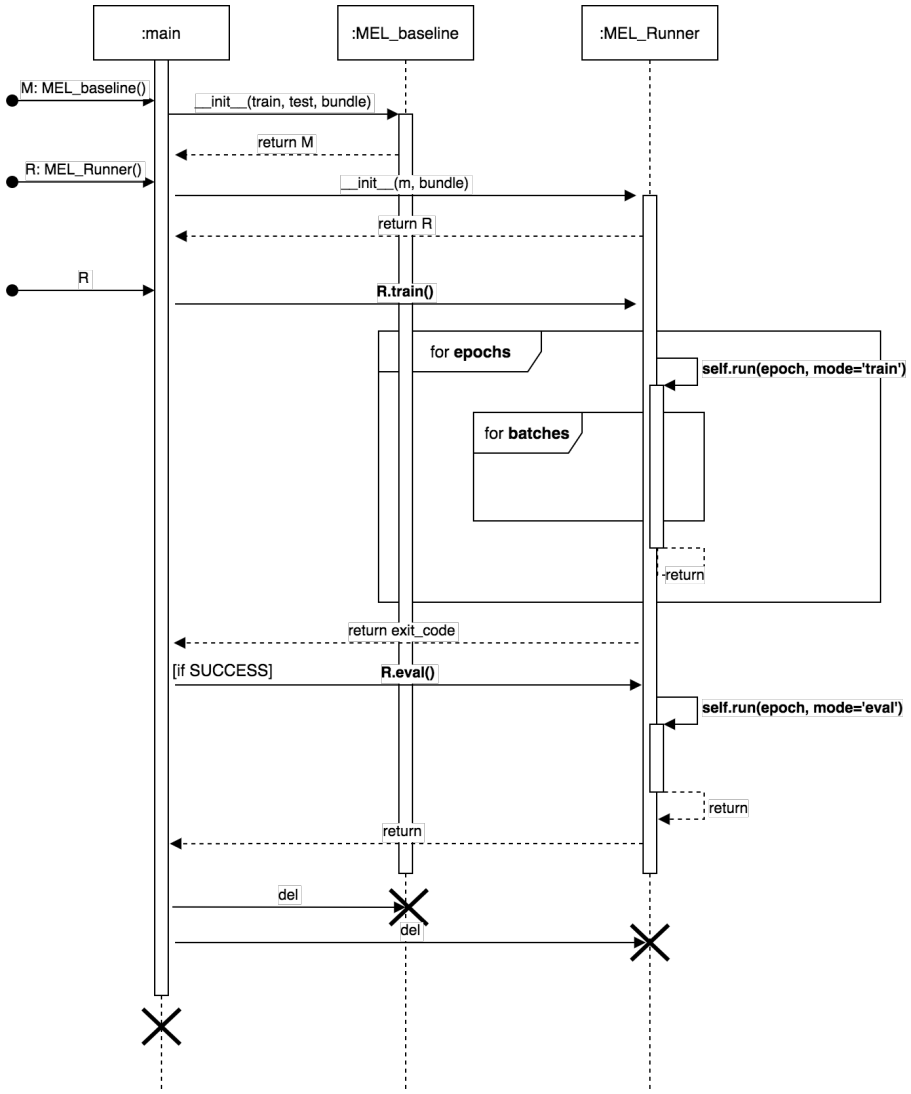


Figure 4.6: Sequence Diagram of ResNet18 Training

4.3.3 RAW-audio task

As said in chapter 2, section 2.1 new approaches tend to use directly the waveform as input of an 1D-CNN.
There are a few models made for different tasks, and the ones used for MER are usually built to implement the dimensional-approach [18], but since the music dataset has been transformed into a categorical one to have aligned labels with ArtEmis dataset, in this section are presented the architectures used to perform the categorical MER task.
Moreover, the use of raw-audio avoid the "feature-engineering" process which is often challenging, time-expensive and could lead to find some heuristic designed features which might not be optimal for the predictive task.
Architectures are adopted from paper [17] and are used to perform the MER task although they were built for Speech Audio Recognition (SAR) task, thus the aim of the work is to see if they are capable of working in another domain knowledge w.r.t. the one that originated their construction.
The names of the **end-to-end** models implemented are: **M5**, **M11**, **M18**. Their constructors are respectively `Model.TorchM5`, `Model.TorchM11`, `Model.TorchM18` and the object handling the Training Phase is `Model.Runner`.

Models use a large receptive field in the first convolutional layer to mimic the bandpass filters, and very small (`ks=3`) receptive fields subsequently, to control the model capacity. Figure 4.7 is taken from paper [17] and represents the **M3** model with an input length of 32000 and the first receptive field length of 80. The implemented models take this idea but the way first convolutional layer is

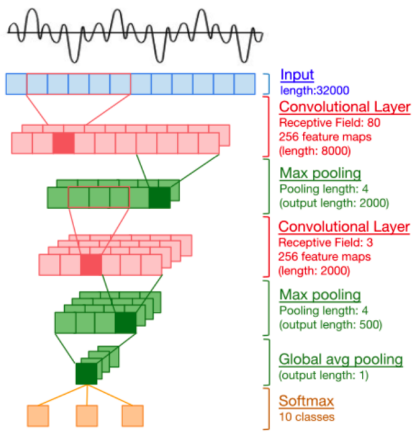


Figure 4.7: Input audio is represented by a channel or (kfm) adopted from [17]

dimensioned is different.
The model constructor is provided with a dictionary called `hyperparameters` containing:

- `n_input`: it is the real audio channel (mono = 1)
- `n_output`: the number of classes
- `kernel_size`: (`ksz`) it is the first kernel receptive field
- `kernel_stride`: (`ksf = ks // 2`) the kernel shift to compute the 1D convolutions along the signal
- `kernel_features_maps`: (`kfm`) corresponds to n_channel in the `nn.conv1d` constructor
- `slice_mode`: (`sm`) whether the `emoMusicPTDataset` is in slice-mode or not
  - `slice_mode=True`: the .mp3 file enter in the pre-processing pipeline 3.7 executed on-the-fly, outputting a `torch.Tensor([1,1, 22050])`
  - `slice_mode=False`: input is 30s excerpt `torch.Tensor([1, 1, 1345050])`

The input of the `nn.Conv1d` is expected to have shape $(N,\ C_{in},\ L)$ thus the output size will be $(N,\ C_{in},\ L_{out})$, where:

$$L_{out} = \left[\frac{L_{in} + 2 \cdot padding\ -\ dilation \cdot (ksz - 1) - 1}{ksf} + 1 \right] \tag{4.3}$$

Here follows the dumps of models **TorchM5** (Fig. 4.8) and **TorchM11** (Fig. 4.9) with `ks=1760`, `ks=880` and `kfm=64`, the deepest model representation is avoided because it is equal to the **TorchM11** with the addition of 7 convolutional layers all with the small receptive field represented by `ks=3`.

```
TorchM5(
  (conv1): Conv1d(1, 64, kernel_size=(1760,), stride=(880,))
  (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool1): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv1d(64, 64, kernel_size=(3,), stride=(1,))
  (bn2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool2): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv1d(64, 128, kernel_size=(3,), stride=(1,))
  (bn3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool3): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv4): Conv1d(128, 128, kernel_size=(3,), stride=(1,))
  (bn4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool4): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=128, out_features=8, bias=True)
)
  - The model has 200840 trainable parameters
```

Figure 4.8: TorchM5

```
TorchM11(
  (conv1): Conv1d(1, 64, kernel_size=(1760,), stride=(880,))
  (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool1): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(1,))
  (bn2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(1,))
  (bn3): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool2): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv4): Conv1d(64, 128, kernel_size=(3,), stride=(1,), padding=(1,))
  (bn4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv5): Conv1d(128, 128, kernel_size=(3,), stride=(1,), padding=(1,))
  (bn5): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool3): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv6): Conv1d(128, 256, kernel_size=(3,), stride=(1,), padding=(1,))
  (bn6): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv7): Conv1d(256, 256, kernel_size=(3,), stride=(1,), padding=(1,))
  (bn7): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv8): Conv1d(256, 256, kernel_size=(3,), stride=(1,), padding=(1,))
  (bn8): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool4): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv9): Conv1d(256, 512, kernel_size=(3,), stride=(1,), padding=(1,))
  (bn9): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv10): Conv1d(512, 512, kernel_size=(3,), stride=(1,), padding=(1,))
  (bn10): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=512, out_features=8, bias=True)
)
  - The model has 1892936 trainable parameters
```

Figure 4.9: TorchM11

Training Settings

Loss function used is *CrossEntropyLoss*, **Optimizer** is *Adam*.
Table 4.3 contains the shared TrainingSettings among the three models. Hyperparameters are depicted in Table 4.4 and for each raw of tab 4.3 all rows of 4.4 have been tried for the Model Selection step.

Total configurations

9 *TrainingSettings* * [10 * 4*Hyperparameters*{I-IV} + 5 * 4*Hyperparameters*{V-VIII}] = 540.

Has been empirically seen how processing very small batches increased the network's performances, thus the only batch sizes analyzed are [2, 4, 8]: the smallest the batch is, the better performances are. *NOTE: As M18 is the deepest network, to achieve similar result compared to the other models, it has been trained for* ***350 epochs***.

TrainingSettings	*bs*	*ep*	*lr*	*wd*	*m*	*cwm*
I.0	2	250	0.001	0.00001	0.9	**None**
I.1	4	250	0.001	0.00001	0.9	None
I.2	8	250	0.001	0.00001	0.9	None
II.0	2	250	0.001	0.00001	0.9	**Freq**
II.1	4	250	0.001	0.00001	0.9	Freq
II.2	8	250	0.001	0.00001	0.9	Freq
III.0	2	250	0.001	0.00001	0.9	**Max**
III.0	4	250	0.001	0.00001	0.9	Max
III.0	8	250	0.001	0.00001	0.9	Max

Table 4.3: Shared Training Settings

Hyperparameters	*ksz*	*ksf*	*kfm*	*sm*
I[0-9]	**110**	55	[8, 64, 128, 256, 512]	[False, True]
II[0-9]	**220**	110	[8, 64, 128, 256, 512]	[False, True]
III[0-9]	**440**	220	[8, 64, 128, 256, 512]	[False, True]
IV[0-9]	**880**	440	[8, 64, 128, 256, 512]	[False, True]
V[0-4]	**1100**	550	[8, 64, 128, 256, 512]	False
VI[0-4]	**1320**	660	[8, 64, 128, 256, 512]	False
VII[0-4]	**1540**	770	[8, 64, 128, 256, 512]	False
VIII[0-4]	**1760**	880	[8, 64, 128, 256, 512]	False

Table 4.4: Hyperparameters summary

All training lifecycle is handled by `Runner Object` which is in charge of training the model, storing relevant data, computing statistics and plots, thus evaluating the model itself.
Figure 4.10 represents the Training lifecycle implemented in creativeAI repository.

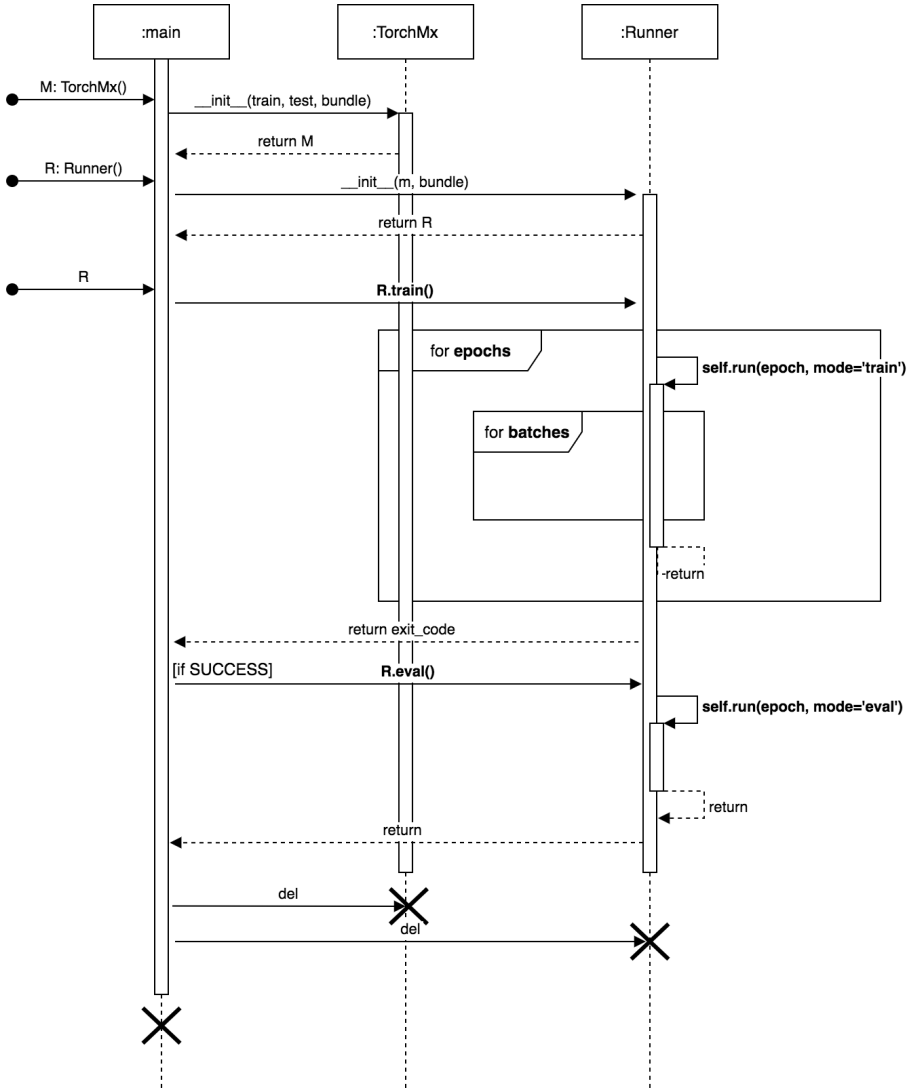


Figure 4.10: Sequence Diagram of TorchM[5-11-18] Training

4.4 creativeAI.imageSide

This package contains software elements which compose a baseline for the generation process.
The **Generator** in Figure 4.1 is a Generative Adversarial Network, in charge of generating artworks coming from the ArtEmis dataset. Most of the work related to this section was spent in the identification of a suitable dataset and on the label alignment discussed in chapter 3, section 3.3. Furthermore to train a *conditional*GAN model the dataset was re-organized in 8 subfolders, in order to allow the use of the `ImageFolder` pyTorch class to have the object of the dataset. Inside the `creativeAI` package following classes are provided:

- `GenerativeModel.DCGAN` which contains the Deep Convolutional GAN model of paper [29]
- `GenerativeModel.CDCGAN` which contains the conditioned version of the model above
- a fine-tuned version of ArtEmis Classifier which might be used to realize an **Auxiliary Classifier GAN**

NOTE: this is a starting point, thus for the models a small version of the dataset was used composed by 2782 abstract images of size (64x64). Some training samples are visible in Figure 4.11.

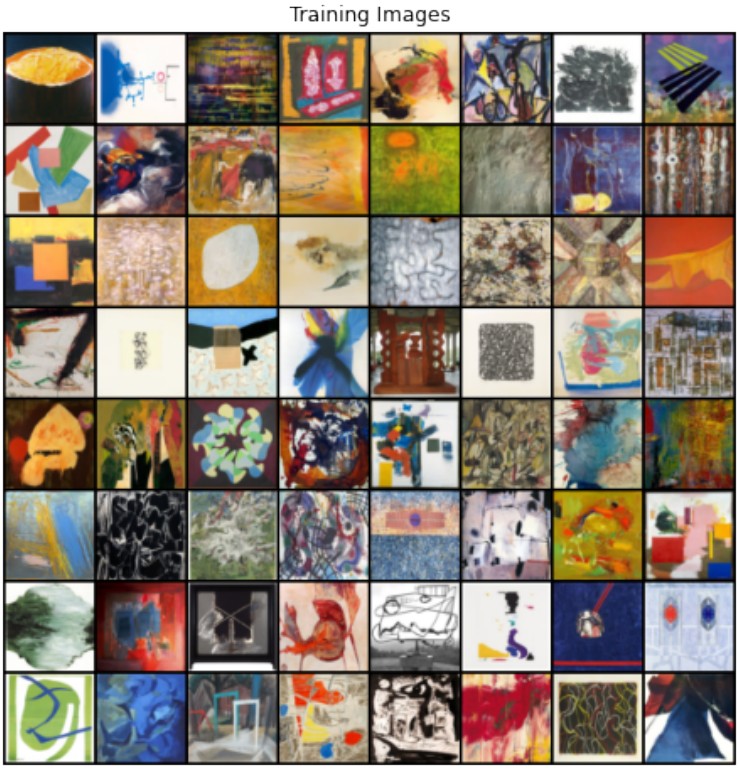


Figure 4.11: Abstract Gallery for DCGAN tests

4.4.1 DCGAN

The model is the one pointed in paper [29], in Figure 4.12 is possible to see the architecture of the Generator, whereas in Figure 4.13 the Discriminator's one.

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(150, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

Figure 4.12: Generator architecture

```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.2, inplace=True)
    (6): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): LeakyReLU(negative_slope=0.2, inplace=True)
    (12): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (13): Flatten(start_dim=1, end_dim=-1)
    (14): Sigmoid()
  )
)
```

Figure 4.13: Discriminator architecture

Training Settings

The training settings were designed following the DCGAN paper mentioned above together with the architectures' design.

TrainingSettings	*batch size*	*img size*	*n channels*	*epochs*
	64	**(64, 64)**	**3**	**100**

Table 4.5: Training Settings for DCGAN

The Generator G maps the latent space vector of size `nz = 100` to data-space. It creates images of size (3x64x64) through a series of strided 2D-Convolutional Transpose layers followed by a 2D Batch Normalization layer and a ReLU activation function. Hyperparameters of G are:

- `nz`: the length of latent space vector
- `ngf`: the length of the features maps propagated inside G
- `nc`: the number of channels in the output image

All the weights are initialized from a Normal distribution with `mean = 0` and `std=0.02`.

The Discriminator D is a binary classifier which takes an input image of size 3x64x64 and tells if the image is real rather than fake through a Sigmoid function, as commonly done in classification networks.

Loss function is the *BinaryCrossEntropyLoss* (i.e. `nn.BCELoss()`) defined in equation 4.4:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -\left[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)\right] \quad (4.4)$$

Optimizers are *Adam* optimizer with, as specified in the paper, `lr = 0.002` and β_1 `= 0.5`.
Fixed Noise is a fixed batch of latent vectors drawn from a Gaussian distribution that are periodically fed into G during training to watch if going on with the iterations the images generated become as desired.

Training Phase

The training of a GAN is almost an 'art form' thus an invalid set of hyperparameters could lead the model to collapse, having a few elements to understand what went wrong. The training loop was build following Algorithm 1 from Goodfellow's paper while implementing some tips from ganhacks GitHub repository [44].

- different mini-batches for real and fake have been constructed
- G's objective function was changed in order to maximize $logD(G(z))$

The Discriminator is trained taking in account that it has to maximize the probability of correctly classifying a given input as real or fake by means of maximizing $log(D(x)) + log(1 - D(G(z)))$.
Due for the separate mini-batches this is calculated in two steps:

- a batch of real samples is constructed, forwarded through D, the loss $log(D(x))$ is calculated and then the gradients are calculated in a backward pass.
- a batch of fakes samples with current G is created, forwarded through D, the loss $log(1 - D(G(z)))$ is calculated and gradients are accumulated in the backward pass.
- Discriminator optimizer's step is called with the gradients accumulated both from real and fake batches

The Generator wants to minimize $log(1 - D(G(z)))$ aiming to generate better fakes, but since Goodfellow has shown that this does not provide sufficient gradients the goal of G is changed to maximize $log(D(G(z)))$.
This is achieved with the following steps:

- Generator's output is classified by Discriminator in the first part of the training
- G's loss is computed using real labels as Ground Truth
- G's gradients are computed in a backward pass
- G's parameters are updated with an optimizer step

NOTE: the Ground Truth labels are used to allow the $log(1-x)$ part of BCELoss() to be replaced with $log(x)$, accomplishing Goodfellow's hint.

4.4.2 conditional DCGAN

The presented DCGAN architecture has been modified to be conditioned by the emotion labels. To achieve this the following hyperparameters have been added:

- `ne`: the length of the embedded label to be concatenated to z
- `n_classees`: the number of classes, 8 in our case

The Generator G was modified as follows:

- the first `convTranspose2d()` takes the z length plus the `embed_size = 64`
- `forward()` pass takes class labels of the current batch, creates an embedding (64, 64, 1, 1) and concatenates it with the images Tensor obtaining a `torch.Tensor([64, 164, 1, 1])`

The Discriminator D:

- the first `Conv2d()` takes one additional channel, used for label handling
- `forward()` pass receives class labels of the current batch, creates an embedding (64, 1, 64, 64) and concatenates it with the images Tensor obtaining a `torch.Tensor([64, 4, 64, 64])`

4.5 Results

This section reports results of the experiments.
For each model the classification confusion matrix together with evaluation measures explained in chapter 3, section 3.5 are reported. Moreover, are reported the following quantitative metrics:

- *General* **accuracy**
- *General* **macro avg**
- *General* **weighted avg**
- *Per-class* **precision**
- *Per-class* **recall**
- *Per-class* **f1-score**

Discussion on results is written in section 4.5.2.

4.5.1 MEL-spec task

TrainingSettings and Hyperparameters in figures and tables captions, refers to tables 4.2 and 4.1.

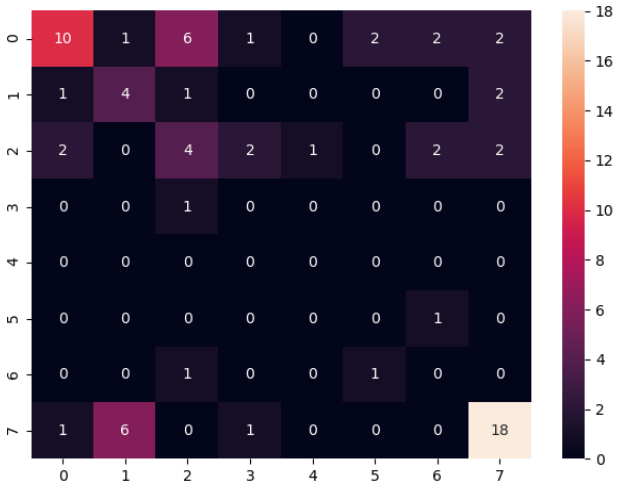


Figure 4.14: Loss and Accuracy on Test Set for TrainingSettings[II.0]

emotion-class	***precision***	***recall***	***f1-score***	*support*
amusement: 0	**0.42**	**0.71**	**0.53**	14
contentment: 1	**0.50**	**0.36**	**0.42**	11
awe: 2	**0.31**	**0.31**	**0.31**	13
excitement: 3	0.00	0.00	0.00	4
anger: 4	0.00	0.00	0.00	1
disgust: 5	0.00	0.00	0.00	3
fear: 6	0.00	0.00	0.00	5
sadness: 7	**0.69**	**0.75**	**0.72**	24

Table 4.6: Classification Report for Set II.1

metric	***precision***	***recall***	***f1-score***	*support*
accuracy			**0.48**	75
marco avg	**0.24**	**0.27**	**0.25**	75
weighted avg	**0.43**	**0.48**	**0.44**	75

Table 4.7: Classification Report cont for Set II1

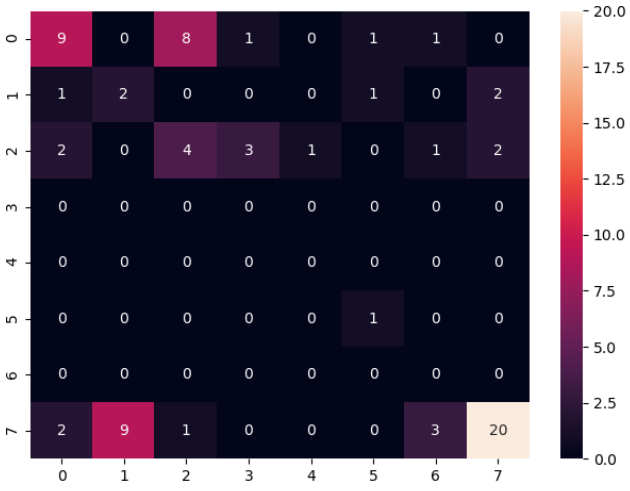


Figure 4.15: Loss and Accuracy on Test Set for TrainingSettings[V.1]

emotion-class	***precision***	***recall***	***f1-score***	*support*
amusement: 0	**0.45**	**0.64**	**0.53**	14
contentment: 1	**0.33**	**0.18**	**0.24**	11
awe: 2	**0.31**	**0.31**	**0.31**	13
excitement: 3	0.00	0.00	0.00	4
anger: 4	0.00	0.00	0.00	1
disgust: 5	**1.00**	**0.33**	**0.50**	3
fear: 6	0.00	0.00	0.00	5
sadness: 7	**0.57**	**0.83**	**0.68**	24

Table 4.8: Classification Report for TrainingSettings[V.1]

metric	***precision***	***recall***	***f1-score***	*support*
accuracy			**0.48**	75
marco avg	**0.33**	**0.29**	**0.28**	75
weighted avg	**0.41**	**0.48**	**0.42**	75

Table 4.9: Classification Report cont for TrainingSettings[V.1]

4.5.2 RAW-audio task

TrainingSettings and Hyperparameters in figures and tables captions, refers to tables 4.3 and 4.4.

TorchM5

Although this model is not deep as the M11 and M18 version its perfomances are near to the other model's.
Loss and Accuracy plots for the best configuration are depicted in Figure 4.16.
By comparing confusion matrices in figures 4.17 and 4.18 the best pair (*TrainingSettings, Hyperparameters*) is (I.0-IV.2).

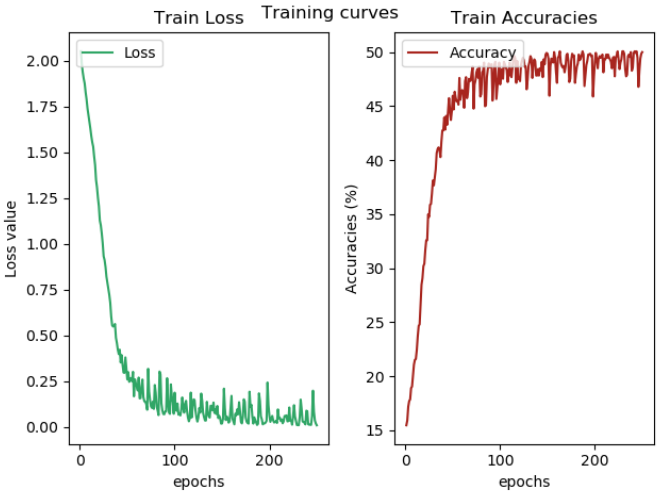


Figure 4.16: Loss and Accuracy on Test Set for [I.0-IV.1] pair

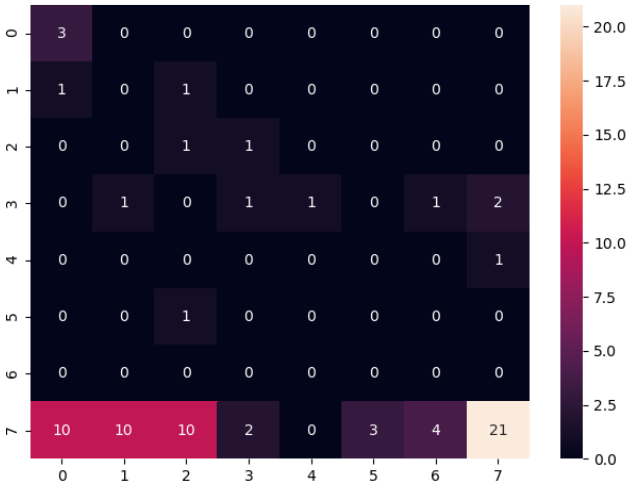


Figure 4.17: Loss and Accuracy on Test Set for [I.0-IV.1] pair

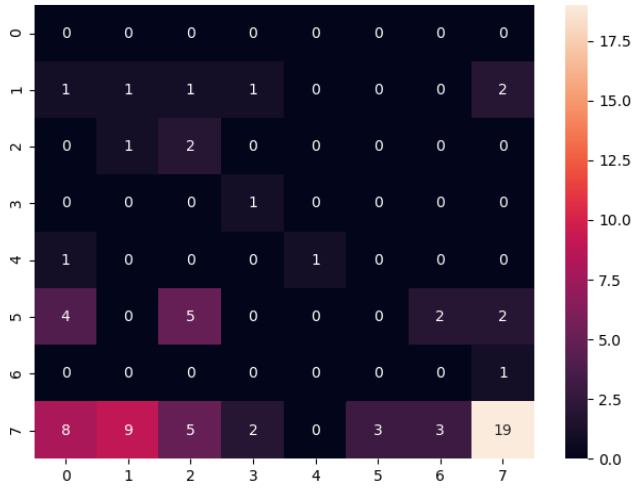


Figure 4.18: Loss and Accuracy on Test Set for [I.0-IV.2] pair

emotion-class	***precision***	***recall***	***f1-score***	*support*
amusement: 0	**0.50**	**0.07**	**0.12**	14
contentment: 1	0.00	0.00	0.00	11
awe: 2	0.00	0.00	0.00	13
excitement: 3	**0.50**	**0.25**	**0.33**	4
anger: 4	0.00	0.00	0.00	1
disgust: 5	**0.33**	**0.33**	**0.33**	3
fear: 6	0.00	0.00	0.00	5
sadness: 7	**0.35**	**0.83**	**0.49**	24

Table 4.10: Classification Report for [I.0-IV.2] pair

metric	***precision***	***recall***	***f1-score***	*support*
accuracy			**0.31**	75
marco avg	**0.21**	**0.29**	**0.16**	75
weighted avg	**0.25**	**0.31**	**0.21**	75

Table 4.11: Classification Report cont [I.0-IV.2] pair

TorchM11

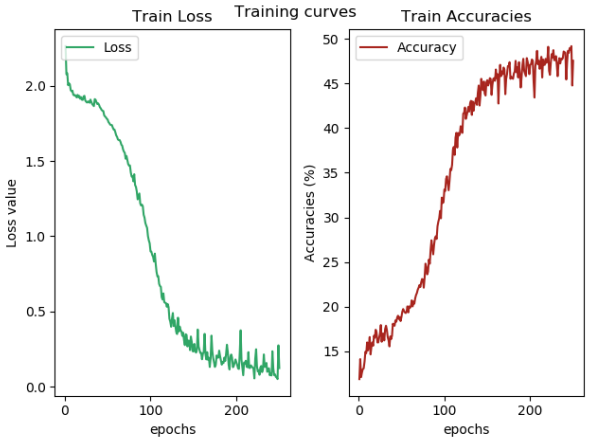


Figure 4.19: Loss and Accuracy on Test Set for [II.0-III.2] pair

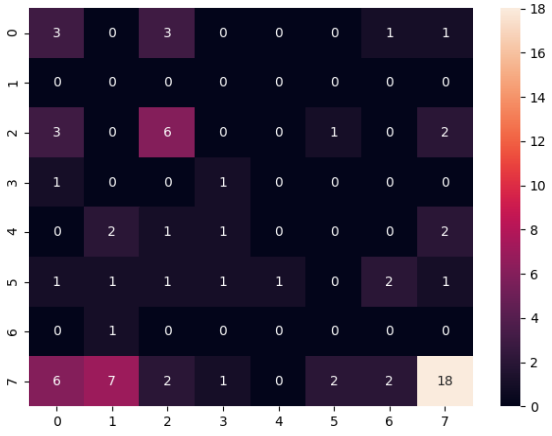


Figure 4.20: Loss and Accuracy on Test Set for [II.0-III.2] pair

emotion-class	***precision***	***recall***	***f1-score***	*support*
amusement: 0	**0.38**	**0.21**	**0.27**	14
contentment: 1	0.00	0.00	0.00	11
awe: 2	**0.50**	**0.46**	**0.48**	13
excitement: 3	**0.50**	**0.25**	**0.33**	4
anger: 4	0.00	0.00	0.00	1
disgust: 5	0.00	0.00	0.00	3
fear: 6	0.00	0.00	0.00	5
sadness: 7	**0.47**	**0.75**	**0.58**	24

Table 4.12: Classification Report for [II.0-III.2] pair

metric	***precision***	***recall***	***f1-score***	*support*
accuracy			**0.37**	75
marco avg	**0.23**	**0.21**	**0.21**	75
weighted avg	**0.33**	**0.37**	**0.34**	75

Table 4.13: Classification Report cont [II.0-III.2] pair

TorchM18

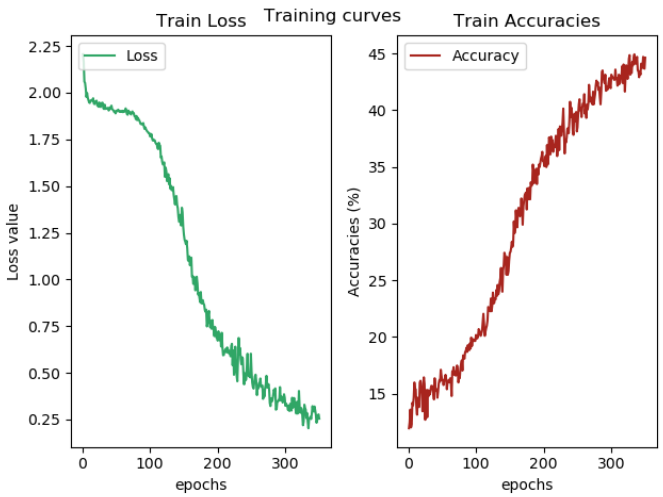


Figure 4.21: Loss and Accuracy on Test Set for [II.0-V.2] pair

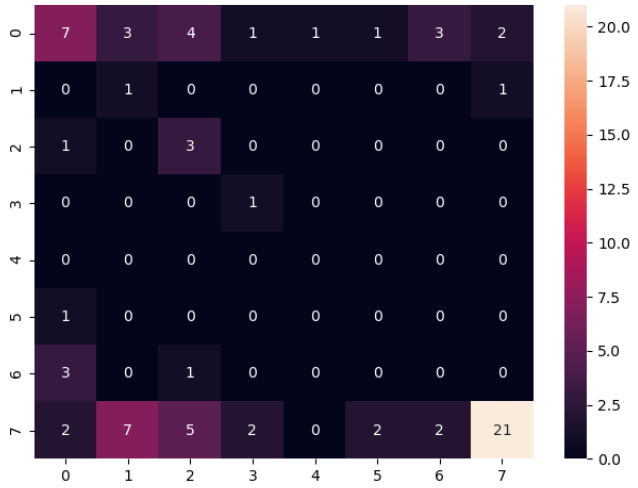


Figure 4.22: Loss and Accuracy on Test Set for [II.0-V.2] pair

emotion-class	***precision***	***recall***	***f1-score***	*support*
amusement: 0	**0.32**	**0.50**	**0.39**	14
contentment: 1	**0.50**	**0.09**	**0.15**	11
awe: 2	**0.75**	**0.23**	**0.35**	13
excitement: 3	**1.00**	**0.25**	**0.40**	4
anger: 4	0.00	0.00	0.00	1
disgust: 5	0.00	0.00	0.00	3
fear: 6	0.00	0.00	0.00	5
sadness: 7	**0.51**	**0.88**	**0.65**	24

Table 4.14: Classification Report for [II.0-V.2] pair

metric	***precision***	***recall***	***f1-score***	*support*
accuracy			**0.44**	75
marco avg	**0.39**	**0.24**	**0.24**	75
weighted avg	**0.48**	**0.44**	**0.38**	75

Table 4.15: Classification Report cont [II.0-V.2] pair

Considerations

Model	*Modality*	*best f1-score : class*	best accuracy
ResNet18	*Mel-spec*	**0.72 : sadness**	**48%**
TorchM18	*Raw-audio*	**0.65 : sadness**	**44%**
TorchM11	*Raw-audio*	**0.58 : sadness**	**37%**
TorchM5	*Raw-audio*	**0.49 : sadness**	**31%**

Table 4.16: MER task model performances summary

Although class re-weighting techniques were utilized, the predicted class more correctly still is the most present in the dataset.
Classifiers' results are poor to be used in a real-context application, underlining the challenges of this task might be emphasized even more by a weak test samples support to perform experiments.
However as seen in literature, the new end-to-end 1D-CNN approach is comparable with the standard Mel-spectrogram 2D-CNN one, even in the MER task. Although the TorchM18 model does not have residual connections, has performances similar to the ResNet18 ones, thus it should be examined if a deeper 1D-CNN model with residual connections could overcome it.

Exploring the model's **internal features maps** it has been found that the sizes of the features maps are of the form `(X, Y, 3)` thus, interpreting them as images where the first dimension is the RGB channel. Figure 4.23 shows the sizes of features maps for the **TorchM5** model.
As said by authors of paper [45] we can interpret this finding pointing that **end-to-end 1D-CNN** are able to learn Frequency Decomposition and Phase Invariance without any pre-processing step, thus it is possible to see that the CNN has learned the filterbanks which are used when a Spectrogram is created.

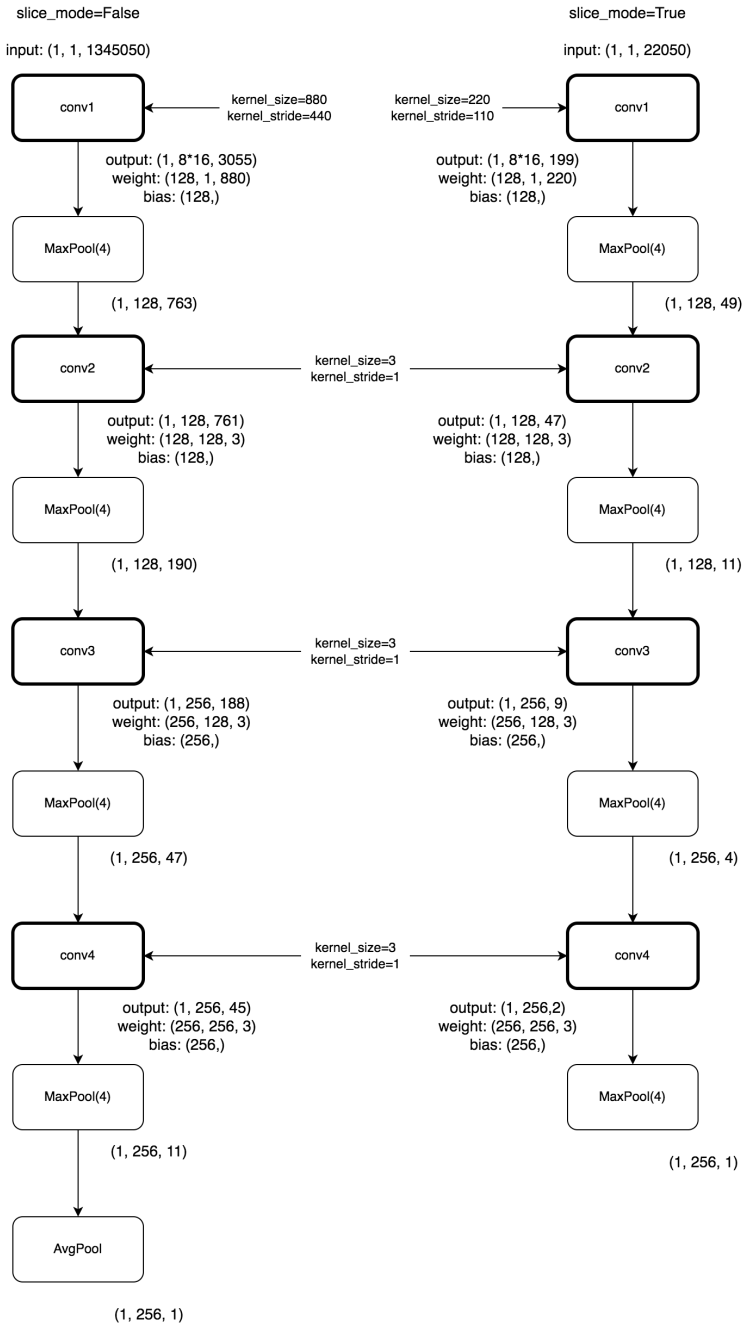


Figure 4.23: Internal kernel features map dimensions

4.5.3 DCGAN

The DCGAN was trained for 100 epochs, without any generation target but only to see the convergence of the model, thus no metric for a quantitative evaluation has been implemented. Moreover each epoch consists in 44 iteration.
Figure 4.24 shows both G and D losses during training loop. Images in Table 4.17

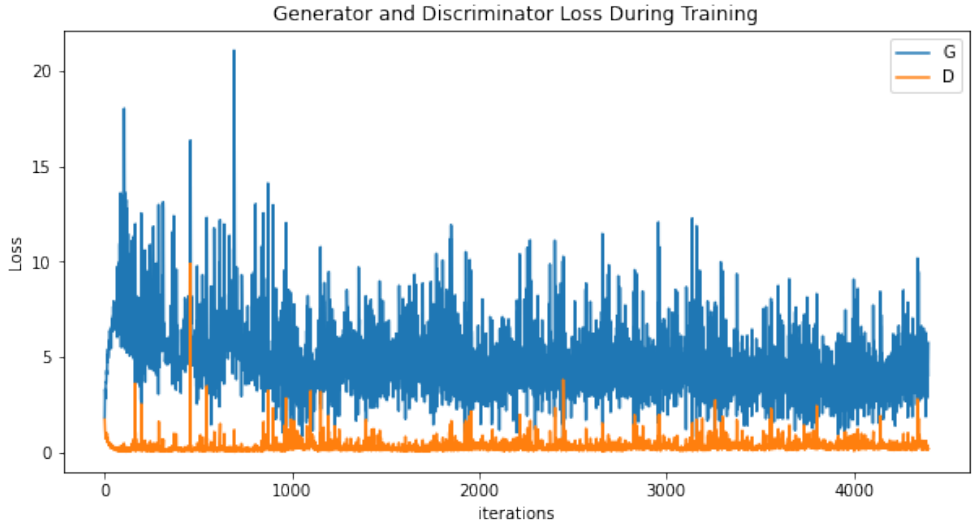


Figure 4.24: Generator and Discriminator Losses over 100 epochs (4400 iterations)

show batches of images generated every 1000 iterations during the training process. In Figure 4.25 represents real samples against the fakes generated at the end of the training loop.

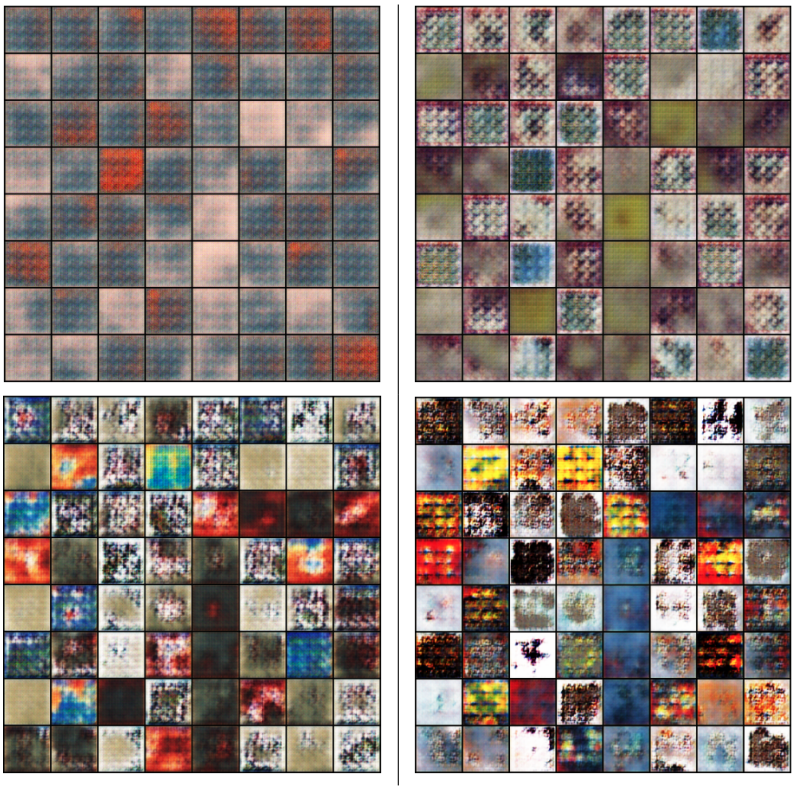


Table 4.17: Left to Right: iteration 1000, 2000, 3000, 4000

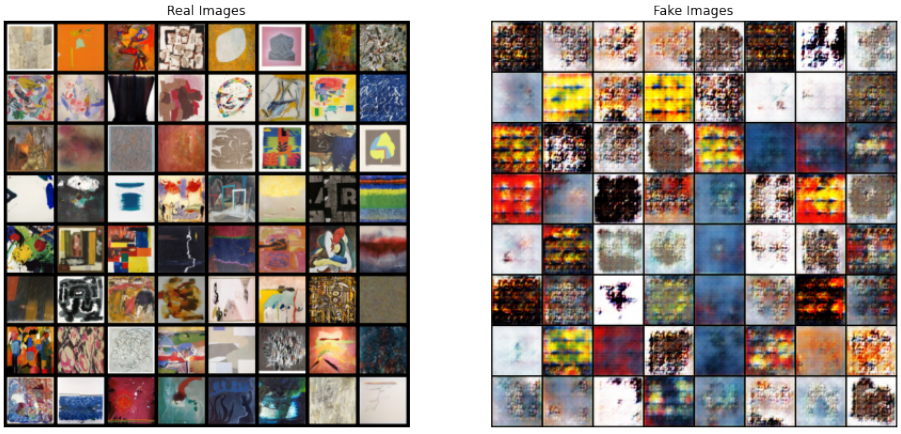


Figure 4.25: Real vs Fakes

Chapter 5

Conclusions

This work represents an attempt to be the basis for the development of the deep learning project described, concerning the artistic field. Many issues raised during the work's conducting, proving that the thesis idea was not composed by easy tasks.
On the first hand, it has been proved that Music Emotion Recognition is an hard challenge for several reasons, first of all for its intrinsic subjectivity. On the other, time and effort were spent on the research thus the pre-processing of suitable datasets useful for the realization of the project idea, taking time away from generative analysis but at the same time giving it a solid starting point from a data point of view.
Besides the *'structural'* problems related to MER, the work aimed to give a comparison between the two different input treatment modalities.
The evidence is that the new 1D-CNN approach achieves similar results as the 2D-CNN one. Raw waveforms avoid hand-crafted features, which theoretically should exploit modeling capability of deep learning models, at the expense of an high computational cost together with the need of data availability. What representation is preferable is not still clear, thus there remain many open research questions related to scenarios in which one representation is better than the other. The models used to perform the raw-audio modality were taken by the Speech domain so it is possible to assert that even if in their domain of application they perform well, the representations from the raw audio they learn are not enough to allow the generalization across domains.
Besides what audio representation is preferable, there are doubts on which model is better to be fed with raw waveform. From existing literature, there is not a single answer because several research groups have achieved state-of-art with different models on the same tasks.
Moreover, chosen an architecture, it is hard to understand how to change its design to meet the goal of emotion prediction, since emotion is one of the most difficult

feature to be modeled hence captured.
The generative process is, instead, at its starting point although a baseline is provided. Since GANs models need a lot of data to be successfully trained, the discovery of the ArtEmis dataset is certainly a good finding for the future of this work, because it contains 81k artworks related to the affective dimension. The baseline provided is the DCGAN of the paper [29] modified to be conditioned on emotion classes.
Although this part has still to be explored the generative process wants to represent a possible, different, application of an emotion classifier, aimed at enabling AI the possibility to create visual art in a new way, inspired by music, as many human artists do.

Bibliography

[1] Michael Casey, Remco Veltkamp, Masataka Goto, Marc Leman, Christophe Rhodes, and Malcolm Slaney. «Content-Based Music Information Retrieval: Current Directions and Future Challenges». In: *Proceedings of the IEEE* 96 (May 2008), pp. 668–696. DOI: 10.1109/JPROC.2008.916370 (cit. on p. 19).

[2] adrian freed adrian. «music metadata quality: a multiyear case study using the music of skip james». In: *journal of the audio engineering society* (Oct. 2006) (cit. on p. 20).

[3] Avery Wang. «The Shazam Music Recognition Service». In: *Commun. ACM* 49.8 (Aug. 2006), pp. 44–48. ISSN: 0001-0782. DOI: 10.1145/1145287.1145312. URL: https://doi.org/10.1145/1145287.1145312 (cit. on p. 20).

[4] Yu-Huei Cheng, Pang-Ching Chang, Duc-Man Nguyen, and Che-Nan Kuo. «Automatic Music Genre Classification Based on CRNN.» In: *Engineering Letters* 29.1 (2020) (cit. on p. 20).

[5] Siddharth Gururani, Cameron Summers, and Alexander Lerch. «INSTRUMENT ACTIVITY DETECTION IN POLYPHONIC MUSIC USING DEEP NEURAL NETWORKS». In: Apr. 2019 (cit. on p. 20).

[6] Linlin Gu Wen Tang. *Harmonic Classification with Enhancing Music Using Deep Learning Techniques*. 2021. DOI: \url{https://doi.org/10.1155/2021/5590996} (cit. on p. 20).

[7] Ashutosh Kulkarni and Deepak Iyer. «Audio Segmentation». In: (cit. on p. 20).

[8] Saliha Benkerzaz, Youssef Elmir, and Abdeslem Dennai. «A Study on Automatic Speech Recognition». In: 10 (Aug. 2019), pp. 77–85. DOI: 10.6025/jitr/2019/10/3/77-85 (cit. on p. 20).

[9] yi-hsuan Yang and Homer Chen. «Machine Recognition of Music Emotion: A Review». In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 3 (May 2012). DOI: 10.1145/2168752.2168754 (cit. on p. 20).

[10] R. W. Picard. *Affective Computing*. 1997 (cit. on p. 20).

[11] Richard Orjesek, Roman Jarina, Michal Chmulik, and Michal Kuba. «DNN Based Music Emotion Recognition from Raw Audio Signal». In: *2019 29th International Conference Radioelektronika (RADIOELEKTRONIKA)*. 2019, pp. 1–4. DOI: 10.1109/RADIOELEK.2019.8733572 (cit. on pp. 20, 21, 37).

[12] Jialie Shen, Bin Cui, John Shepherd, and Kian-Lee Tan. «Towards Efficient Automated Singer Identification in Large Music Databases». In: *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '06. Seattle, Washington, USA: Association for Computing Machinery, 2006, pp. 59–66. ISBN: 1595933697. DOI: 10.1145/1148170.1148184. URL: https://doi.org/10.1145/1148170.1148184 (cit. on p. 20).

[13] Rui Cai, Chao Zhang, Chong Wang, Lei Zhang, and Wei-Ying Ma. «MusicSense: Contextual Music Recommendation Using Emotional Allocation Modeling». In: *Proceedings of the 15th ACM International Conference on Multimedia*. MM '07. Augsburg, Germany: Association for Computing Machinery, 2007, pp. 553–556. ISBN: 9781595937025. DOI: 10.1145/1291233.1291369. URL: https://doi.org/10.1145/1291233.1291369 (cit. on p. 20).

[14] C. Laurier, M. Sordo, J. Serrà, and Perfecto Herrera. «Music Mood Representations from Social Tags». In: *International Society for Music Information Retrieval (ISMIR) Conference*. Kobe, Japan, 26/10/2009 2009, pp. 381–386. URL: http://mtg.upf.edu/files/publications/music_moods_representation_from_social_tags-claurier_msordo_jserra_pherrera.pdf (cit. on p. 20).

[15] Konstantinos Drossos, Andreas Floros, Andreas Giannakoulopoulos, and Nikolaos Kanellopoulos. «Investigating the Impact of Sound Angular Position on the Listener Affective State». In: *Affective Computing, IEEE Transactions on* 6 (Jan. 2015), pp. 27–42. DOI: 10.1109/TAFFC.2015.2392768 (cit. on p. 20).

[16] Yi-Hsuan Yang, Ya-Fan Su, Yu-Ching Lin, and Homer H. Chen. «Music Emotion Recognition: The Role of Individuality». In: *Proceedings of the International Workshop on Human-Centered Multimedia*. HCM '07. Augsburg, Bavaria, Germany: Association for Computing Machinery, 2007, pp. 13–22. ISBN: 9781595937810. DOI: 10.1145/1290128.1290132. URL: https://doi.org/10.1145/1290128.1290132 (cit. on pp. 20, 21).

[17] Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. «Very deep convolutional neural networks for raw waveforms». In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017, pp. 421–425. DOI: 10.1109/ICASSP.2017.7952190 (cit. on pp. 21, 63).

[18] Miroslav Malík, Sharath Adavanne, Konstantinos Drossos, Tuomas Virtanen, Dasa Ticha, and Roman Jarina. «Stacked Convolutional and Recurrent Neural Networks for Music Emotion Recognition». In: July 2017 (cit. on pp. 21, 63).

[19] Zhen Wang, Suresh Muknahallipatna, Maohong Fan, Austin Okray, and Chao Lan. «Music Classification using an Improved CRNN with Multi-Directional Spatial Dependencies in Both Time and Frequency Dimensions». In: *2019 International Joint Conference on Neural Networks (IJCNN)*. 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852128 (cit. on p. 21).

[20] S. S. Stevens, J. Volkmann, and E. B. Newman. «A Scale for the Measurement of the Psychological Magnitude Pitch». In: *The Journal of the Acoustical Society of America* 8.3 (1937), pp. 185–190. DOI: 10.1121/1.1915893. eprint: https://doi.org/10.1121/1.1915893. URL: https://doi.org/10.1121/1.1915893 (cit. on p. 26).

[21] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML] (cit. on pp. 28, 29).

[22] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. *Analyzing and Improving the Image Quality of StyleGAN*. 2020. arXiv: 1912.04958 [cs.CV] (cit. on p. 28).

[23] Aidan Clark, Jeff Donahue, and Karen Simonyan. *Adversarial Video Generation on Complex Datasets*. 2019. arXiv: 1907.06571 [cs.CV] (cit. on p. 28).

[24] Andrew Brock, Jeff Donahue, and Karen Simonyan. *Large Scale GAN Training for High Fidelity Natural Image Synthesis*. 2019. arXiv: 1809.11096 [cs.LG] (cit. on p. 28).

[25] Ian Goodfellow. *NIPS 2016 Tutorial: Generative Adversarial Networks*. 2017. arXiv: 1701.00160 [cs.LG] (cit. on p. 29).

[26] *Google - Generative Adversarial Networks*. https://developers.google.com/machine-learning/gan/gan_structure (cit. on p. 29).

[27] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG] (cit. on pp. 30–32).

[28] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG] (cit. on p. 30).

[29] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG] (cit. on pp. 30, 31, 69, 70, 88).

[30] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG] (cit. on p. 30).

[31] Vinod Nair and Geoffrey Hinton. «Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair». In: vol. 27. June 2010, pp. 807–814 (cit. on p. 30).

[32] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. «Rectifier nonlinearities improve neural network acoustic models». In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013 (cit. on p. 30).

[33] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. *Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks*. 2015. arXiv: 1506.05751 [cs.CV] (cit. on p. 32).

[34] Ronak Kosti, Jose Alvarez, Adria Recasens, and Agata Lapedriza. «Context based emotion recognition using emotic dataset». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019) (cit. on p. 33).

[35] *ImageEmotion dataset*. https://www.imageemotion.org/ (cit. on p. 33).

[36] *WikiArt dataset*. https://wikiart.org (cit. on p. 33).

[37] Panos Achlioptas, Maks Ovsjanikov, Kilichbek Haydarov, Mohamed Elhoseiny, and Leonidas Guibas. «ArtEmis: Affective Language for Visual Art». In: *CoRR* abs/2101.07396 (2021) (cit. on pp. 33, 34, 37).

[38] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollar, and C. Lawrence Zitnick. *Microsoft COCO Captions: Data Collection and Evaluation Server*. 2015. arXiv: 1504.00325 [cs.CV] (cit. on p. 33).

[39] Mohammad Soleymani, Micheal N. Caro, Erik M. Schmidt, Cheng-Ya Sha, and Yi-Hsuan Yang. «1000 Songs for Emotional Analysis of Music». In: *Proceedings of the 2Nd ACM International Workshop on Crowdsourcing for Multimedia*. CrowdMM '13. Barcelona, Spain: ACM, 2013, pp. 1–6. ISBN: 978-1-4503-2396-3. DOI: 10.1145/2506364.2506365. URL: http://doi.acm.org/10.1145/2506364.2506365 (cit. on pp. 36, 42).

[40] James Russell. «A Circumplex Model of Affect». In: *Journal of Personality and Social Psychology* 39 (Dec. 1980), pp. 1161–1178. DOI: 10.1037/h0077714 (cit. on pp. 39, 40).

[41] Ali Borji. *Pros and Cons of GAN Evaluation Measures*. 2018. arXiv: 1802.03446 [cs.CV] (cit. on pp. 48, 49).

[42] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2018. arXiv: 1706.08500 [cs.LG] (cit. on p. 49).

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV] (cit. on p. 59).

[44] *Gan Hacks*. URL: %5Curl%7Bhttps://github.com/soumith/ganhacks%7D (cit. on p. 72).

[45] Jongpil Lee, Jiyoung Park, Keunhyoung Luke Kim, and Juhan Nam. «SampleCNN: End-to-End Deep Convolutional Neural Networks Using Very Small Filters for Music Classification». In: *Applied Sciences* 8.1 (2018). ISSN: 2076-3417. DOI: 10.3390/app8010150. URL: https://www.mdpi.com/2076-3417/8/1/150 (cit. on p. 82).