2. Tesina in Inglese

INTRO

Slide 1: Emotions in AI Music Composition

During the preparation of this presentation, I began exploring the topic of music composition based on Artificial Intelligence. However, as I delved deeper into my research, I asked myself a fundamental question: if it is true that music composition partially follows rigid and mathematical rules, is it really possible for algorithms to emulate the emotional side of this creative activity as well?
To try to answer this question, I decided to move from theory to practice by developing two Python scripts that we will discuss during this presentation. Through these experiments, I wanted to investigate how AI can contribute not only to music generation but also to the reproduction of the emotions it conveys.

Slide 2: AI Overview

This slide provides an overview of the main approaches to artificial intelligence. Each type of AI offers unique advantages depending on the specific application.

On the left, we have **rule-based systems**, which operate based on an explicit set of predefined rules and logic. This approach is particularly useful in contexts where the rules are well-defined, such as medical diagnosis through expert systems or legal applications, but it becomes rigid in complex or unstructured environments.

In the center, we have **Machine Learning**, distinguished by its ability to learn from data. Depending on the problem, it can be supervised or unsupervised:

- **Supervised learning** requires labeled data and can solve problems such as image recognition, speech recognition, or, in my case, emotion analysis in music.
- **Unsupervised learning**, on the other hand, seeks hidden patterns or relationships within unlabeled data, making it useful for problems like clustering or dimensionality reduction.

On the right, we have **Symbolic AI** systems, which represent knowledge in the form of symbols and logical rules. This approach is very transparent and ideal in fields that require precise and understandable explanations, such as logical reasoning systems.

After analyzing these approaches, I chose Machine Learning, particularly supervised learning with neural networks, for my project. This choice was motivated by the flexibility of these techniques in modeling complex phenomena such as emotions and their ability to learn rich representations directly from musical data.

Slide 3: System Configuration

To implement the system, I used a series of essential libraries for data processing, audio analysis, and machine learning model creation.

Let's start with the main libraries:

- **NumPy:** It was essential for performing numerical calculations and handling multidimensional arrays, which are the basis for manipulating audio features.
- **Pandas:** It allowed me to load and manage datasets in tabular format, such as annotations for the DEAM dataset, and organize them in a structured way for analysis.
- **Scikit-Learn:** It was used for some preprocessing operations, such as encoding emotion labels and splitting the dataset into training and test sets.
- **TensorFlow:** It formed the basis for building, training, and saving the neural networks used for emotion classification and music generation (Keras).
- **Librosa:** This is a library specific to audio processing. It allowed me to extract MFCC features from music files, representing the acoustic content of the tracks.
- **Music21:** It was indispensable for working with MIDI files in music generation, allowing me to analyze and create notes and chords.

Regarding the datasets:

- **DEAM:** It was used in the first script to analyze and classify emotions. It contains annotations related to the valence and arousal of musical tracks, which are fundamental for assigning emotion labels.
- **EMOPIA:** This dataset was used in the second script for music generation. It contains MIDI files divided into specific emotions such as happiness, sadness, and fear, providing a rich and varied base for training the generative model.

Slide 4: Dataset Feature Interpretation

Here is represented how the values of **valence** and **arousal** were interpreted to classify the emotions associated with musical tracks.

Valence and arousal are two fundamental dimensions for describing emotions:

- **Valence** measures how positive or negative an emotion is.

- **Arousal** represents the intensity of the emotion, from calm to excited.

In the dataset, the values of these two variables range from 0 to 10. For simplicity, I divided each axis into two intervals:

- **From 0 to 5**, representing a “low” or “negative” value.
- **From 5 to 10**, representing a “high” or “positive” value.

This allowed me to create a Cartesian diagram with four quadrants:

1. **Happiness:** Associated with high valence and high arousal (quadrant in the upper right).
2. **Sadness:** Associated with low valence and low arousal (quadrant in the lower left).
3. **Fear:** Associated with low valence but high arousal (quadrant in the lower right).
4. The quadrant in the upper left, corresponding to emotions with high valence but low arousal, was not considered in this project.

This scheme helped me in two ways:

- **For the first dataset (DEAM):** I assigned an emotion to each musical track based on the average valence and arousal values reported in the dataset.
- **For the second dataset (EMOPIA):** I divided the MIDI files into folders, each representing one of the considered emotions. This organization was essential for training the music generation model.

CODE1

Slide 5: Song Emotion Analysis

The first code of the project focuses on **analyzing the emotions conveyed by a song**. The main objective is to create and train a model capable of recognizing the predominant type of emotion that a musical track communicates to the listener.

To do this, the system uses the acoustic features of the song as input and associates them with an emotion, chosen among the three considered: **happiness**, **sadness**, and **fear**.

The idea is to develop a system that mimics, in a very elementary way, the human ability to perceive the emotion of a song, based solely on measurable sound characteristics.

Slide 6: Script Execution Algorithm

The execution flow of this first script is as follows:

1. **Data Loading:** First, we load the DEAM dataset, which contains information about musical tracks, including the averages of valence and arousal values.
2. **Emotion Assignment:** Next, we map the valence and arousal values to a specific emotion.
3. **Feature Extraction:** We extract the main acoustic features from the audio files using the Librosa library.
4. **Data Preparation:** We transform the data into a numerical format suitable for the model. This includes operations such as normalization and one-hot encoding of emotion labels.
5. **Training:** We train a neural network model to classify emotions. During this phase, the model learns to recognize patterns in the data to correctly associate acoustic features with emotions.
6. **Evaluation:** Once trained, we evaluate the model using a separate test set to measure its accuracy in classifying emotions.
7. **Prediction:** Finally, we use the trained model to make predictions on new musical tracks, verifying whether the predicted emotion matches the perceived one.

We will now analyze in detail the implementation of each of these phases.

Slide 7: Emotion Assignment

The function you see, called assign_emotion, works as follows:

1. For each row in the dataset, it retrieves the average **valence** and **arousal** values associated with a musical track.
2. Based on these values, it decides which emotion to assign:
   - If both values are greater than 5, the assigned emotion is **happiness**.
   - If both values are less than or equal to 5, the emotion is **sadness**.
   - If valence is less than or equal to 5 and arousal is greater than 5, the emotion is **fear**.
3. Emotional states that do not fall into these three categories are excluded.

This function is applied to each row of the dataset, adding a new column called emotion that contains the assigned label. Below, we see an example of the result:

- The first track, with low values of both valence and arousal, was classified as sadness.
- The second track, with high arousal but low valence, was classified as fear.
- The third track, with both values high, was classified as happiness.

Slide 8: Features Extraction

To extract features from audio files, I defined **extract_features**: this function takes the name of an audio file as input and returns a numerical representation of the sound characteristics, specifically the **scaled MFCC**. The output of this function is a one-dimensional vector representing the **average over time** (axis=0) of the MFCC coefficients.

But what are Mel Frequency Cepstrum Coefficients?

Slide 9: Features Extraction

MFCCs are a numerical representation of the sound content based on the short-term power spectrum of a signal. They are useful because they mimic the functioning of the human ear in perceiving sounds, focusing on the most relevant frequencies. The process to calculate them is as follows:

1. **Fourier Transform (FFT):** First, the Fourier transform is calculated to obtain the frequency spectrum of the audio signal.
2. **Mapping to the Mel Scale:** The human ear does not perceive frequencies linearly. It is very sensitive to changes in low frequencies (like 100 Hz vs. 200 Hz) and less sensitive to high frequencies (like 10,000 Hz vs. 10,100 Hz). The Mel scale mimics this human sensitivity, compressing the high frequencies. For this reason, the power spectrum is projected onto the Mel scale, which is a logarithmic representation of frequencies designed to reflect human perception. To do this, a bank of **triangular filters** overlapping is used to "synthesize" each frequency band into a specific region on the Mel scale. Each filter calculates the energy present in that band.
3. **Logarithm of Powers:** The natural logarithm of the energy (power) of each triangular filter on the Mel scale is calculated. This is done because the human ear perceives sound intensities logarithmically (for example, an increase from 10 to 100 is perceived as a bigger jump compared to 1000-1100). This step simulates this perception.
4. **Discrete Cosine Transform (DCT):** Finally, a discrete cosine transform is applied to reduce the dimensionality of the data, eliminating high-frequency variations (noise or irrelevant details).
5. **Final Output:** The MFCCs are the resulting coefficients after the DCT. Each coefficient represents a characteristic of the sound (for example, overall pitch, rhythmic pattern, etc.), and together they describe the acoustic content in a compact and interpretable way for the machine learning model.

On the right, we can see in columns from top to bottom:

1. The graph of an audio signal X in the time domain (x-axis = time, y-axis = amplitude).
2. The graph of the spectrogram of the same audio signal (x-axis = time, y-axis = frequency, color = intensity of that frequency at that specific moment in time).
3. The graph of the Mel Frequency Cepstrum Coefficients (x-axis = time, y-axis = coefficient, color = contribution of that coefficient at that specific moment in time).

Slide 10: Data Preparation

In this phase, we prepare the data for training the neural network (NN). The extracted audio features (MFCC) and emotion labels are transformed into a numerical format compatible with machine learning models.

1. **Conversion to DataFrame:** The audio features and their corresponding labels are organized into a **DataFrame**.
2. **Creation of X and yy:**
   - **X**: Contains the MFCC vectors as NumPy arrays, ready to be processed by the model. With .tolist(), each row of features_df['feature'] is converted from a list to a format compatible with NumPy.
   - **yy: y** contains the emotion labels in string format (for example, "happiness"). From *y*, using **Label Encoding (**LabelEncoder**), textual labels (for example, "happiness", "sadness") are transformed into numerical values (for example, 0, 1, 2). Through** One-hot Encoding (**to_categorical**): **the numerical label values are converted into a one-hot format. For example:**
     "happiness" → [1, 0, 0]
     "sadness" → [0, 1, 0]
     *** "fear" → [0, 0, 1]**
     Thus, we obtained X **(a two-dimensional NumPy array, where each row represents an MFCC vector) and** yy* *(a two-dimensional NumPy array, where each row represents a* label - emotion*).
3. **Splitting into Training Set and Test Set:** We use the train_test_split function to divide the dataset into two parts:
   - **Training Set (80%):** Data used to train the model.
   - **Test Set (20%):** Data used to evaluate the model's performance on unseen data.

Slide 11: Neural Network

We then describe the architecture of the neural network used: it is a **feed-forward sequential** (Sequential Model) network, where data flows through the layers linearly, from input to output.

The types of layers used are:

- **Dense (256 units):** It is a fully connected layer, where each neuron receives input from all neurons in the previous layer. Here, 256 neurons compute a linear combination of the 40 input MFCC features.
- **Activation (ReLU): ReLU (Rectified Linear Unit)** activation function, defined as:

$$f(x) = \max(0, x)$$

It is used to introduce non-linearity, allowing the model to learn complex patterns.

- **Dropout:** Dropout regularization is applied, randomly disabling a percentage of neurons during training to prevent overfitting.
- **Activation (Softmax):** The **Softmax** activation function produces a probability vector, with the sum of values equal to 1:

$$\mathrm{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Each output represents the probability that the track belongs to one of the emotions.

These layouts are combined to form the Neural Network:

- **Input layer:** The network's input is a one-dimensional vector containing the **40 MFCC features** extracted from each audio file.
- **First and second level:** Both are composed of a Dense layer, an Activation layer, and a Dropout layer. The number of parameters gradually decreases to ensure that the first layer can learn complex patterns from the data, while the next layer compresses the data, emphasizing only the relevant patterns for classification.
- **Third level:** The last level consists of a Dense layer with only 3 neurons, representing the 3 emotions (*happiness, fear, sadness*) and a Softmax Activation layer producing interpretable probabilities for classification.

Slide 12: Training and Test of the Model

Training

Now we move on to training the defined model.

- **ModelCheckpoint**: This callback saves the model's weights during training. It is configured to save (*only the best*) weights of the model in the file audio_classification.keras and to be “verbose” (i.e., print detailed information during saving). It is useful to ensure that the optimal version of the model is not lost.
- **model.fit**: Trains the model with the provided data:
  - **batch_size=32**: Data is processed in blocks of 32 samples at a time, improving efficiency.
  - **epochs=20**: Training lasts for 20 epochs, meaning the model sees all training data 20 times.
  - **callbacks=[checkpointer]**: We include the callback to save the best weights during training.
  - **verbose=1**: Prints details about the training progress.

Test

At the end of training, we evaluate the model's performance on the test data:

1. We load the model.
2. We evaluate the model on the test set using the model.evaluate function.

We obtain an accuracy of 70.49%, which indicates good performance for a 3-class classification problem (*happiness, fear, sadness*), considering the complexity of the domain (music and emotions) and the fact that I had little time to improve the model.

Slide 13: Prediction

Tracks 3.mp3 and 4.mp3 were removed from the original dataset and moved to a separate test folder. This ensures that the model did not “memorize” these tracks during training. Listening to the audio files, I assigned an emotion to the two tracks based on my personal perception:

- 4.mp3 → **Happiness.**
- 3.mp3 → **Sadness.**
  The model correctly predicted both emotions. This shows that the model can generalize even to tracks not included in the original dataset, indicating that the **acoustic patterns** of emotions (for example, higher pitches for happiness, lower pitches and slow rhythms for sadness) were effectively learned.

This experiment, although simple, highlights the model's ability to **capture musical emotions** in a way consistent with human perception.

CODE2

Slide 14: Song Emotions Generation

After seeing how we can classify the emotions conveyed by music (Code1), the next step is to **create music that expresses specific emotions**. This approach explores the inverse of the previous problem: going from emotions (input) to a musical sequence coherent with that emotion (output).

Slide 15: Script Execution Algorithm

Let's overview the operational flow of the code we will analyze:

1. **Phase 1: Extract Notes**: The MIDI files of the dataset are analyzed to extract *individual notes* and *chords*. This data is organized based on the emotion associated with the MIDI file (happiness, sadness, or fear).
2. **Phase 2: Tokenize Notes and Emotions**: Notes and emotions are converted into numerical representations:
   - **Notes:** Tokenized using a dictionary that assigns a number to each note or chord.
   - **Emotions:** Encoded into numerical values (e.g., happiness = 0, sadness = 1, fear = 2).
     This step is essential to make the data understandable to the neural network.
3. **Phase 3: Create Input/Output Sequences**: Input/output sequences are created to train the network:
   - **Input:** A sequence of notes and the associated emotion.
   - **Output:** The next note in the sequence.
4. **Phase 4: Normalize Input**: Input sequences are normalized by scaling the notes between 0 and 1 to improve stability during training. Emotions are also normalized to be integrated as part of the input. This ensures that all features are on comparable scales.
5. **Phase 5: Training**: The neural network is trained using:
   - Normalized inputs (sequences of notes + emotion).
   - Outputs (next note).
6. **Phase 6: Evaluation**: After training, the model is tested on a validation set to evaluate:
   - The accuracy in generating coherent sequences.
   - The ability to associate musical features with the target emotion.
7. **Phase 7: Melody Generation**: This is the final phase, where the model is used to generate a melody:
   - A starting sequence (random) and a target emotion (e.g., happiness) are provided.
   - The model generates notes one at a time, using the previous output as input for the next step.
     The generated notes are saved as MIDI files, which can be listened to.

We will analyze in detail how these steps were implemented.

Slide 16: Notes Extraction

The function get_notes() is designed to extract notes and chords from MIDI files contained in a directory associated with a specific emotion. This data will be used to create input sequences for the neural network. The extraction process is as follows:

1. **Locating MIDI Files:** All MIDI files present in the folder associated with the specified emotion are found. Glob is used to scan the path data_midi/emotion/*.mid.
2. **Iterating over MIDI Files:** Each file is analyzed one by one.

3. **Parsing the MIDI File:** The MIDI file is converted into a readable structure using music21.
4. **Extracting Notes and Instruments:** It tries to separate notes by instrument:
    - If there are multiple instruments, only the notes from the first instrumental track are analyzed.
    - If there are no distinct instruments, all notes in the file are analyzed.
5. **Returning the List:** Finally, the function returns the complete list of notes and any chords.

The function is called separately for each emotion in the dataset, creating a set of notes for each emotional category.

Slide 17: Notes Extraction

The function get_all_notes_and_emotions() is designed to collect:

- All notes extracted from the MIDI files (regardless of the associated emotion).
- The corresponding emotions for each note.

The idea is to create two parallel lists:

- **all_notes**: Contains all notes and chords in sequence.
    - The notes extracted by get_notes(emotion) are added to the global list all_notes.
- **all_emotions**: Contains the emotional label associated with each note.
    - For each note in the notes list, the current emotional label (emotion) is replicated using multiplication by len(notes). This ensures a one-to-one correspondence between notes and emotions in the final list.

This data will be subsequently used to train the neural network.

Slide 18: Data Preparation

Tokenization of Notes and Emotions

Regarding the tokenization of notes:

1. **unique_notes**: Unique notes are created by sorting them with sorted(set(all_notes))
2. **Mapping note → integers:** A dictionary note_to_int is built that associates each note with a number (Example: { "C4": 0, "D5": 1, "E4": 2, "C4.E4.G4": 3 }).
3. **Notes to integers:** Each note in all_notes is replaced by its corresponding numerical value, creating a list notes_as_int.

Regarding the tokenization of emotions:

1. **Label Encoding**: LabelEncoder from Scikit-Learn is used to encode emotions into integer numbers (Example: ["happiness", "sadness", "fear"] → [0, 1, 2]).

Creation of Input/Output Sequences

Input (first *SEQUENCE_LENGTH* notes) and output pair *input sequence = next note* are created, and the emotion related to the output note is added to the *network_emotions* array.

Thus, 3 sequences are obtained, representing:

- **Input sequence:** data on which the neural network must make its evaluations ($\mathrm{numberOfSequences} * \mathrm{sequenceLength}$).
- **Conditioning emotion:** emotion given as input to the neural network to obtain the next note $(\mathrm{sequenceLength} * 1)$.
- **Output sequence <=> next note:** note that the neural network should predict.

Slide 19: Data Preparation

The data preparation here includes three main operations:

1. **Conversion of data into numerical arrays** usable by the model.
2. **Normalization of Notes:** The notes are divided by the number of "symbols" to normalize their values. Additionally, the notes array is reshaped as if it were a 3-dimensional array (a third dimension of width 1 is added; the array remains a matrix but is considered as a 3D array).
3. **Normalization of Emotions:** Emotions are also normalized. The resulting array is reshaped to make it three-dimensional $(\mathrm{numberOfSequences} * 1 * 1)$. At this point, emotions are replicated so that each note in a sequence corresponds to a sequence of notes of equal length $(\mathrm{numberOfSequences} * \mathrm{numberOfNotes} * 1)$.
4. **Concatenation** of notes and emotions to form a single input for the network (we have pairs $[[\mathrm{note1}, \mathrm{emotion1}], [\mathrm{note2}, \mathrm{emotion2}], \dots]$).
5. **Categorical Encoding of the Output** (one-hot format) to ensure that the model works with probability distributions for the notes.

Slide 20: Model Definition

As we saw in the previous script, a sequential (Sequential) model in Keras is a simple linear stack of layers, where each layer is connected to the next in a **strictly linear** manner.

In this case, the problem to be solved becomes more complex: it requires handling multiple inputs and capturing complex relationships between temporal data (note sequences) and static information (emotions). For this reason, a **functional** network is used (not sequential).

A functional model allows defining **more complex connections** between layers, such as multiple inputs, multiple outputs, or non-linear connections.

We can see that, in addition to the types of layers already seen in the previous code, here GRU layers are added: **Gated Recurrent Units** are a simplified version of LSTM. While maintaining many of the features of LSTM, they have a simpler architecture and are therefore more computationally efficient.
GRUs have **two main gates**:

1. **Update Gate:** Decides how much of the previous state should be retained and how much of the new input should be added.
2. **Reset Gate:** Decides how much of the past should be "forgotten" to compute the new state.

They do not have a separate memory state like LSTM; they combine **long-term memory and the current state** into a single vector.

I used these instead of LSTMs because the latter would have caused my computer to explode due to the computational load.

Regarding the choice of the number of neurons in the various layers, I chose 256 neurons in the GRUs to effectively capture the sequential dependencies of the melody and emotions, maintaining a balance between precision and computational costs. The 128 neurons in the intermediate dense layer allow us to synthesize the learned representations, while the final layer with 685 neurons exactly corresponds to the number of notes in the dataset's vocabulary.

Slide 21: Training the NN

Finally, we can train the model using the model.fit function. As we can see during training, the model has indeed learned from the note patterns, significantly reducing the loss. However, the fact that the loss reduction stops at 3.5 indicates that in this case, with this simple neural network, it is not possible to obtain neural results. However, I was unable to go further due to the hardware at my disposal.

Slide 22: Preparation for Melody Generation

We begin the preparation process for melody generation.

1. First, a dictionary is created that associates each integer (number representing a note) with its corresponding note. This dictionary is useful for transforming the numerical output of the neural network into real notes.
2. We then define a function **sample_with_temperature** that allows introducing controlled randomness in note generation to prevent the model from getting stuck generating the

"safest" notes, creating monotonous melodies. In practice, the logarithm of the probability vector *probs* output by the model is taken and divided by the chosen temperature. This results in a new probability vector, which is then normalized so that the sum of probabilities equals 1. Subsequently, a note (index) is "randomly" selected, weighting the choice according to the new probabilities.

Slide 23: Preparation for Melody Generation

We now define the note generation function **generate_notes_by_emotion**:

1. We convert the emotion chosen by the user into one-hot format using *label_encoder.transform*.
2. We randomly select one of the sequences that formed our network_input during the training phase. In this way, we extract a "slice" of size $\mathrm{sequenceLength} * 2$ from the original structure, which had dimensions $\mathrm{numberOfSequences} * \mathrm{sequenceLength} * 2$. Once done, we reshape this structure to $1 * \mathrm{sequenceLength} * 2$ to make it compatible with the model's input.
3. After that, we replace all elements of the second axis of the third dimension with the target emotion.
4. We then start a loop that will last for the number of notes we decided to generate. In each cycle:
    1. The model is used to predict the next note, that is, to obtain a probability distribution.
    2. We sample this probability distribution with the "sample_with_temperature" function to introduce the desired degree of randomness and obtain the predicted note index.
    3. We remap the index back to the "domain of notes", finally obtaining the new note.
    4. We obtain the new input to feed into the model to get the next note. Specifically:
        1. We normalize the new obtained note.
        2. We create the pair $[\mathrm{newNote}], [\mathrm{targetEmotion}]$.
        3. We update the time sequence (contained in pattern) by removing the first note and adding the new note at the end.

Slide 24: Preparation for Melody Generation

We define the last function we need for the actual melody generation: **create_midi**. This function takes as input **prediction_output** (*list of predicted notes and chords*) and **output_filename** (*name of the MIDI file to save*) to save the notes and chords in the MIDI file.

In practice, we loop over all the predicted notes and:

1. We check if it is a chord or a single note and handle them accordingly.

2. In both cases, we create the MIDI chord/note, set its instrument to "piano," and save it in the **output_notes** structure.
3. We create a **midi_stream** and save it into the already created MIDI file.

Slide 25: Melody Generation

Finally, we can choose the target emotion and use all the previously defined functions to generate the melody:

1. We generate the melody using generate_notes_by_emotion.
2. We create the MIDI file and use **create_midi** to save the generated melody in the MIDI file.

Slide 26: Resulting Melodies

Despite the results being limited by the simplicity of the model, we notice that within the generated melodies, it is possible to somewhat perceive the target emotions, indicating that it is indeed possible at least partially to emulate the emotional side of musical composition using AI algorithms.
