Tsion Biruk Ephrem 1988189

Ezgi Coşkun 1983656

WaveSprite

Abstract

The project aimed to write a program that takes an image as an input and transcribes that input into a song, an audio file. This is achieved by analyzing the image, calculating the rig pixel colors, therefore, understanding the colors and their tones, and playing an audio file that best corresponds to that shade of the color. The program we made uses supervised training to pick the audio files from the given set, which consists of audio files of the piano notes and chords. With the research we have done, by handing out a survey to people in our environment, as well as contacting a psychology major friend who helped us analyze the findings, the program is trained to pick the appropriate audio file by deciding which feeling, therefore audio file, conveys the color given.

The primary motive for the project was its applicability in art museums and galleries, as well as its personal use for painters. As people with impaired vision are not able to enjoy the art portrayed in the museums as well as individuals with perfect vision, the program can serve as a leveling ground and a gateway to a deeper and different experience with an individual's favorite painting or picture.

Introduction

The program we made is a Python-based program equipped with a graphical user interface for aesthetic pleasure. We used several modules for the back-end code including PIL, NumPy, Wave, and Winsound. Whereas for the front-end we used customtkinter as well as again, PIL.

As we input an image into the program, it will read the image and get data from it. This data is the pixel value of the image. And PIL is an image-processing module that allows us to do just that. We will later be storing these pixel values in a list, and this is where NumPy comes in. NumPy allowed us to manipulate these pixel values and arrange them in the shape that we needed. Later, based on these arranged values we will be assigning an audio file to them. These audio files are in a wav format as most of the Python libraries for manipulating audio use this format. To be able to manipulate these audio files we needed to import the wave module as well. We used Winsound to be able to play the final audio file. We will be passing the path of the image as input to the program as PIL only takes the path of the image.

Graphical User Interface, was used to give an easier experience while using our project. We have used the CustomTkinter module which is the more flexible therefore more customizable version of the GUI module Tkinter. Tkinter is one of the most widely used user interface packages on Python, and with the customtkinter module, we were able to make the GUI we desired.

Body

In this part, we will explain the makings of the program including how we implemented it as well as what methods we used, all the problems we faced, and how we solved them. We will start by explaining the back-end code and then move along to the front end.

Our program reads images from the upper left corner to the bottom right corner. After attaining the data that we need from the image which are the pixel values by using the '.getdata()' after opening the Image using the image path through the '.open()' method from the PIL package, we faced our first problem.

We planned to get the pixels in the form of a list however an average image would have about millions of pixels and that's just too much data to work with. Additionally, when we look at images usually the pixel values for N consecutive pixels are the same. If we take a selfie picture for example, the background color, hair color, or skin tone will remain more or less the same for a certain number of consecutive values. If we leave it like this, the program will keep playing the same audio file for N consecutive plays. Therefore, for the sake of 'diversity' and to reduce the length of the final audio file, we decided to slice our output list. It's as though we have 500000 jumbled up marbles and we want to sort them. So, we start by getting containers and putting N number of marbles per container.

To this end, we use a while loop.

"As long as the list Is not empty, check if the length of the list is greater than 1000. If it is then get the first 1000 elements and put them in a second list let's call it list_2, then decrease 1000 from the length of our original list and keep doing this until the length of the original list is less than 1000. When that happens, just take whatever is left and append it to list_2 then set the length of the original list to zero. If the length is 0, break"

We now have a nested list. Where the first entry of list_2 would be a list with the first 1000-pixel values. But it wasn't long before we faced our second problem. We had successfully shrunk our length of values, however, we still needed to average them but as we know pixels are a tuple. To achieve our goal, we must be able to get all the R values separately and average them and do the same with all the G and B values. We will be looking at each nested list. Let's start with averaging down the R values.

For each nested list, we will create a third list let's call it list_3. This third list will be a nested list as well where we will add the values we have per nested list. So, within this third list, we will have a list containing three values, these values are the averaged red, averaged blue, and averaged green values for each nested list. With our marble analogy, we can think of it as getting each container and getting smaller boxes with 3 slots for each container. And we put only red, only blue, and only green marbles in each respective slot. We will be having the same number of boxes as containers. We will later be putting all the boxes into a larger container.

To get only the red values, we will be using the 'slice ()' function. For each element in list_2, create a list, List_3 (our box). At this point, our nested list still contains a list of tuples, but we need to be able to retrieve the R, B, and G values separately. For this purpose, we must turn each nested list into an array by using NumPy functions. After doing so, we will be creating a variable "r" which we will use to store all the red values from the pixel. To get only the red values, we will be using the 'slice ()' function. We will be going from the first element in the nested list until the last elements skipping every 2 elements in between so we will be skipping the blue and the green and only getting the red ones. We will then be getting the average of the red values. And appending them into list_3. We will later be repeating this process for the Blue and the Green values as well.

List_3 is our box. And we said that we will be putting these boxes into a larger container so we will append list_3 to a larger list. Let's call it list_4. List_4 is a nested list that contains the shrunk, averaged-down pixel values. At this point, our model needs to identify which pixel corresponds to which color.

As we know, different colors correlate to feelings. For example, yellow is more of a happy color than red. However, we must also take into consideration the intensity as dark red and bright red convey different emotions. Our program needs to take all of these into consideration. As we mentioned in the abstract, we used supervised training to train our model. We collected a sample of the 34 most used colors. We only used 34 colors as there are millions of colors with only a small variation in pixel values. And to make the program run faster and so it wouldn't have a lot of data to process we already averaged them down so for all the pixel values that are in a certain range we are considering them as one color. We passed these colors through the first part of the model, and we got their pixel values. Now we need to map these color values to audio files.

To map the pixel values to the audio files we made a data set as we couldn't find a dataset available online for free that applied to our project. The features of our data set were the colors, hex values, the averaged pixel values, mood, and the audio file. It contains 182 rows, and we were unable to collect more data as it took a lot of time. The data we had was enough for our program. We wanted to train our model on this dataset using gaussianNB. We defined 4 variables X_train, X_test,y_train, and y_test, and planned on using 70% of the data for the training set and 30% for testing. We wanted to train the model on the average pixel value and it to predict the audio file so to X we had indexed the dataset at average pixel value and for y we had indexed the audio. However, we were unable to proceed as X can only accept data in matrix form, so we were unable to give a single column. Converting the dataset into type integer and

passing it to the model would not have worked as well as we wanted the model to be trained on only the averaged pixel values as that is what will be returned from our program. If we had trained the model on the averaged pixel value and its hex decimal for example, it wouldn't be able to predict the audio file if we just gave it the averaged pixel value. As a result, we were unable to train the model on the dataset we made it hence we had to use if-else statements. 'If the pixel value is between this and this then play this audio file.' Another thing we needed to keep in mind is that there had to be variation among the audio files. For example, if our input is a picture of a sky, we will only be listening to the sound of blue the whole time. Hence to introduce a bit of variance, we added some random colors in between. So now the model looks more like, 'If the pixel values are between this and this value or this and this value, then play this audio'.

For the basis of the correlation between audio and color, we did research and conducted surveys by asking people what each color made them feel. And later we used those results and read papers online about the correlation between emotions and music notes. Or what kind of feelings musical notes induce in people. After we got a list of correlating emotions to the musical notes, we decided to create audio files based on that. However, during the creation of these audio files, we needed to keep in mind that these audio files are going to be assigned depending on the picture. Hence there is a chance that these assigned audios sound bad as they will not be composed. Therefore, we had to make them in a way that could sound good any way it was concatenated.

We used FL Studio Mobile to create chords and note progressions for each color in our code. We constructed 3 note melodies alongside chords that go well with each other in random order and reflect the emotion of the colors. Similar hues share similar chords, darker hues go lower, and lighter hues go higher.

If we play the sound directly from the if-else statements, we would have some gaps between the output sounds. To avoid this the if-else statements will be getting the audio files and appending them to a list. Let's call it the audio_list. We will now get the audios from this list and concatenate them.

To do this, we will be using the wave library. We created a variable that has a value of "final_audio.wav". Later we will be saving our output audio with this name. We will use a for loop to iterate over our list of audios and for each audio we will open it in a readable format and get its parameters and frames. Then we will be appending these parameters and frames into another list, let's call this list list_5, as we will need to concatenate and "write" them onto an output audio file. After opening our output audio file in the write mode, we will iteratively write the frames and parameters we got onto our output audio file and save it. Finally, we now have a complete transcription of our image in our directory. The only thing left is to open, listen, and enjoy.

When it comes to the GUI part , it is implemented by first making the window, by setting the root to customtkinter. After that, we needed to make the app and set the app to "customtkinter.CTk". In doing so, we start using the window with the customtkinter features.

These features include setting the geometry of the window and other optional attributes. We set the window to be not resizable, set the title of the window, and set the appearance mode to light (customtkinter has dark and automatic options as well) also set the default color theme, which sets the color theme used while we are interacting with the buttons, it also has 3 options: green, blue and dark blue.

There are various widgets we can use by importing customtkinter including but not limited to labels, buttons, frames, and text boxes. Although the attributes may vary from widget to widget, the general operation of a widget is that you can set a variety of attributes within the widget, like its height, width, color, etc but first, you must set the master, which is the main window we will use to attach the widget to. In our case, the master is the app itself, but in other situations, it may also be a frame or a canvas.

Using frames to create the GUI with customtkinter is the most popular method, but using frames provided us with less diversity since we wanted to add a background picture to make the window appear more visually appealing. Putting a frame on the background picture did not fit the visual we had for the project and further complicated the code. To use an image as a background we need to first import the Image module from the PIL library once again. This will allow us to import the picture. Then we'll be using the customtkinter image widget to import the picture into the ui by entering its path and then its size.

To place the image inside the window we will create a label and then use the "place" method to put the label in the window. There are different methods to put the label on the window however, if we use any other method than the place method, we can't make any other widgets like a text box or buttons on top of the image, which would defeat the purpose of the GUI.

To be able to make a button, first, we need to set the master frame as well as its attributes. For the button feature, you can set the text, text color background color, foreground color, corner radius, width, height, etc. We made the button functional by using the attribute "command", which allows us to connect the button with a function that we set for the button. For the first page, we used the "next page" function that allows us to go to the next page. While we were working on the GUI going to the next page was one of the challenging aspects that we faced. The most convenient method was to use classes to make the change however when we use class methods, customtkinter would not allow us to set a background image by rejecting the image we wanted to input. So, we had to set 2 different files for 2 different pages and when we use the next page function, to clear the first page we needed to destroy the page itself and then load the next page by importing the file itself.

We put the buttons in their places by using the place function again, however, for a different reason than why we used it while setting the background image. Place method lets us put the widget wherever we want in the app, by setting the x and y function while pack lets us put padding in x and y coordinates. Padding is the gap we allow between the widgets. And grid lets us stick the widget we have to the frame by using north, south, east, and west.

After we set the widgets, frames, and everything we want in our window, we needed to use .mainloop to run the GUI, mainloop makes sure after we press a button or use an attribute, we will be able to use the window all over again.

Now that we have an overlook at how the Customtkinter works, and the widgets we used as well as the attributes that those widgets have now, we can discuss the purpose of the first and the second page. The first page is designed to be an introductory page and to be the ‘Welcome' page just to give an idea of what the project is about. While the second page is for the actual usage of the project. Since the first page is only about the introduction, we only used a label and a button to navigate to the second page.

Most of our efforts and our time was spent on the second page, which includes an entry and 2 buttons. The entry widget is used to enter the path of the picture that we want to use, and the first button is used to submit the picture to our program. We use the command attribute and the function “button_callback”. This function uses the “get” method to take the path of the image and adds it to the list that we defined named “hey” using the append method. The list serves as a ‘transportation means from the input to the back-end part of the code. The program will take the first element from the list (which will be the path of the image) and it will use it as an input to do its job.

And with this, we can now use the image path in our main code. Also, when we press the submit button, we can see the picture we loaded, by using the open_pic function which is called in the “button_callback” function. In open_pic, like the way we made a background image, we load the image into a CTkImage widget, then make a label for it, decide on the sizing and the place we want the picture to be, therefore loading it into the window. The second Button in the window is used to start the program by sending the path of the image to the back_end part of the code, calling the make_sound function. And with this, the GUI part is concluded.