CSCE 5290: Natural Language Processing

Final Project

GitHub link: https://github.com/RyanTolbert/ArtGenerator

Project Description:

This project will create an album cover art from a user-entered prompt. The user can customize this cover using basic image adjustments, like color saturation, tint, a text editor, etc.

1. Project Title and Team Members
   Title: Album Cover Generator
   Team: Ryan Tolbert

2. Goals and Objectives:
   - **Motivation:** I am a graphic designer, photographer, and music enthusiast and would like to see if I can use NLP to create an album cover from a prompt. I think this tool could be used to inspire or generate a piece of concept art.

   - **Significance:** This album cover generator can be used as a tool for inspiration and idea creation. Having AI create an image can provide a new view of the world. These images may not look particularly realistic, but they can provide the necessary proof-of-concepts for certain ideas. Using this people can brainstorm ideas for album covers and customize the cover art to their liking. There has been text to image generators before, but there are no art generators. The main idea for this is to create cover arts for music albums, but you could use it for any sort of art creation. I aim to create a useful tool for artists and designers to use to create art.

   - **Objectives:** The user will enter a prompt of what they want the cover art to look like. A prompt can be something like “A dog wearing a red hat”. Using NLP, my program will determine what the user wants and use the OpenAI model to generate an image based on the prompt. The user will also be given options on what they would like to add or change with the cover art. So, if the user wants the image to be black and white, they can do that. Perhaps the user wants to add the album title in text on the cover art, they should be able to do that. This tool should provide some basic tools to generate and customize a cover art that they like.

- **Features:** The main feature is that you can input any prompt and get an image output. There should be a feature to add text and select the font, size of the text, and position of the text on the cover art. There should also be an option to make the cover art black and white or add saturation to make the colors brighter. I plan to also add more basic image options to allow for the most customization to the user. I also may provide an option for the user to use their voice to enter a prompt. This would enable users to enter a prompt verbally or via text.

Increment 1:

- **Related Work (Background):** There currently exists methods to make an album art cover, but it requires you to upload your own photo. Alternatively, there also exists ways you can generate an image from scratch with a user inputted text prompt. However, you can not edit these generated images. I am combining both of these ideas to make for an art generator that allows the user to type what they want the photo to be of and also some basic photo editing options to adjust the image to the user's liking.

- **Dataset:** I am using the Hugging Face dataset to process the text that will be inputted from the user. With that, I am using the DALL-E model to build an image from the processed text. I am also using Vqgan-JAX to decode the images. All of these are listed and linked in the references of this document.

- **Detail Design of Features:** For this increment the feature added is the generation of an image from a user-inputted prompt.

- **Analysis:** Currently, the program is output accurate images. An interface for the user to input this prompt would be ideal. In its current state, the user must type their prompt in the command line. A level of abstraction could make this program much better for the user.

- **Implementation:** For the implementation, I did tests with hugging face first. The reason for this is to ensure that the text was being tokenized and analyzed properly. From there I separately test the generation of images with premade text prompts. Once that was working well, I combined these functions. This makes for a program that takes an prompt from the user and builds an image based on their input.

- **Preliminary Results:** First testing the processing of text:

```
prompts = ['a cat']
tokenized_prompts = processor(prompts)
tokenized_prompt = replicate(tokenized_prompts)
print(f"Prompts: {prompts}\n")
```

```
Prompts: ['a cat']
```

Testing a user inputted text:

```
userInput = input("Enter your prompt: ")

prompts = [userInput]
tokenized_prompts = processor(prompts)
tokenized_prompt = replicate(tokenized_prompts)
print(f"Prompts: {prompts}\n")
```

```
Enter your prompt: a dog
Prompts: ['a dog']
```

The first tests with a premade input:

```
decoded_images = p_decode(encoded_images, vqgan_params)
decoded_images = decoded_images.clip(0.0, 1.0).reshape((-1, 256, 256, 3))
for decoded_img in decoded_images:
    img = Image.fromarray(np.asarray(decoded_img * 255, dtype=np.uint8))
    images.append(img)
    display(img)
    print()
```

```
100% 4/4 [00:37<00:00, 6.73s/it]
/usr/local/lib/python3.7/dist-packages/jax/_src/ops/scatter.py:90: FutureWarning: sc
  FutureWarning)
```

This result is pretty good, it is very clear that the image is of a cat. Now to test with a more complex user-inputted string:

```
    for decoded_img in decoded_images:
        img = Image.fromarray(np.asarray(decoded_img * 255, dtype=np.uint8))
        images.append(img)
        display(img)
        print()
```

```
Enter your prompt: a dog with a red hat on
Prompts: ['a dog with a red hat on']
```

100% 4/4 [00:12<00:00, 3.18s/it]

I was impressed with these results. The prompt inputted was “a dog with a red hat on” and the results are quite clear given the complex input. It is easy to tell it is a dog and the hat isn’t the best, but it is red. So, for the most part, these are satisfactory results.

Finally, I tested this with a very complex prompt to see its current capabilities. I set it to output 4 different photos of the same prompt. The prompt entered was “The Eiffel Tower in the Sahara Desert” and here were the results:

This is an excellent result, especially given the tough input prompt. At this point I was satisfied with all the results and started planning towards implementing the image options.

- **Project Management:**
    - **Implementation status report**
        - **Work completed:** Generate an image from user-inputted text.
            - *Description:* The user enters a prompt and the program generates and image from scratch based on the prompt. For example, if the user enters; “a cactus”, the program will build an image of a cactus.
            - *Responsibility:* Ryan Tolbert
            - *Contributions:* Ryan Tolbert
        - **Work to be completed:** Allow the user to make adjustments to the image.
            - *Description:* With the image generated from the program, the user should be able to make basic image adjustments, like saturation, hue shift, adding text, adding a border, etc. For example, if the user wants the image to be in black and white with a white border, they should have the options to do so.
            - *Responsibility:* Ryan Tolbert
            - *Contributions:* Ryan Tolbert

3. References
   This is the model I am using to generate images from the text:
   https://github.com/openai/DALL-E

   I am using this decoder: https://github.com/patil-suraj/vqgan-jax

   I am using hugging face to process the text:
   https://huggingface.co/blog/how-to-generate

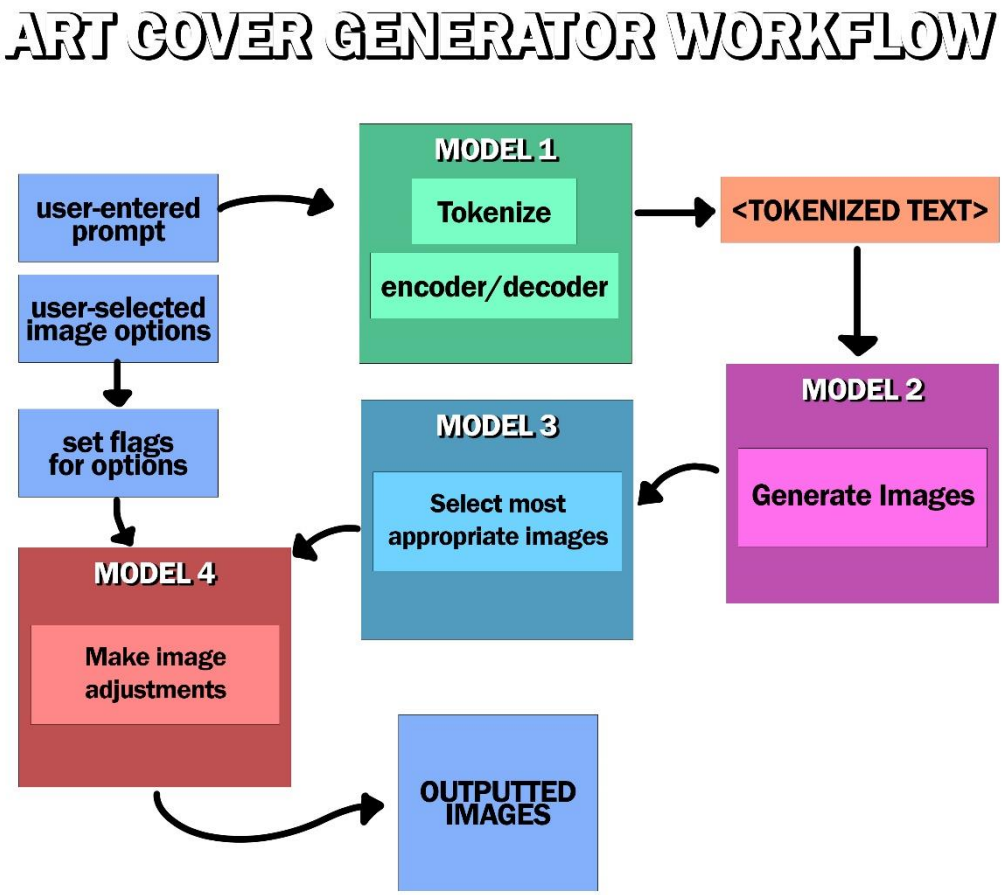
Increment 2:

- **Introduction:** For this increment I needed to add image options to give the user more control over the customization of their album cover. These adjustments are:
    1. Choosing the number of covers to generate (up to 5)
    2. Adding a black and white version of each cover generated
    3. Adding a border
    4. Selecting a border color

5. Adding text
6. Selecting text color
7. Selecting text location
8. Selecting text size
9. Adding blur to the image

- **Background:** I looked at album covers to see what image options would be needed to make these covers. In doing this, I came up with the options listed in the *Introduction* section. I used the Python Pillow Image module (https://pillow.readthedocs.io/en/stable/reference/Image.html) to process the images. This module has many capabilities and allowed me to add all the options I wanted. I referenced all the sources I used to help implement the image options in the *References* section at the end of this document.

- **My Model:**



The way this works is by first asking the user for a prompt. Then the user will also be asked a list of options that they can add to the image. For example, if the user wants to add a border, a flag will be set to let the program know that a border should be added to the image after

the image has been generated. The user-entered prompt is tokenized, encoded, and decoded using BART. VQGAN Decoder is then used to generate the images from the tokenized text. Then VQGAN CLIP is used to rank the images and select the most appropriate images. These images are then used to make the user-selected adjustments and the completed images are then outputted to the screen for the user. Model 1 uses BART, Model 2 uses VQGAN Decoder, Model 3 uses VQGAN CLIP, Model 4 uses the flags set by the user.

- **Dataset:** The dataset used to create the image adjustments is Pillow (https://pillow.readthedocs.io/en/stable/reference/Image.html). This gives me the ability to add a border, add text, add blur, or make the image black and white. Within some of these there are extra options available. So with the text, you can also change the color, size, and location of the text as well as the font. I used a default font for all these so the user does not have to download any extra fonts. I am also using W3C color names to match color choice with hex value (https://www.computerhope.com/jargon/w/w3c-color-names.htm) to allow the user to type their color choice using words.

- **Implementation:** I used flags to mark what a user wants. I did this because the program needs to know what the user wants before generating the image, otherwise the program does not know what to generate. For example, when I the user if they want a black and white version, it looks like this:

```
userInput2 = input("Do you want a black and white version? (y for yes): ")
if userInput2 == 'y':
  BW = 1
```

If the user enters: ‘y’ for yes, a flag “BW” gets set to 1. Then when it comes down to where the image is being generated and outputted, this flag is recognized like so:

```
if BW == 1:
  grayscale = img.convert('L')
  display(grayscale)
```

If the flag is true, then Pillow is used to convert the image to grayscale, or black and white. Then output this black and white version to the screen. This is method is used with each option given to the user. Also, when implementing text size and location, I wanted to make it simple for the user. I did not want the user to have to think about text size as a number, for this reason I gave the user 3 simple options; small, medium, large. Heres what this looks like on the user’s end:

```
1. Small
2. Medium
3. Large
What do you want the size of the text to be? (enter the number):
```

The user then enters “1”,”2”, or “3” depending on the size they want. This is what it looks like on the code’s side:

```
textSizeInput = input("\n1. Small\n2. Medium\n3. Large\nWhat do you want the size of the text to be? (enter the number): ")
textSizeInput = int(textSizeInput)

if textSizeInput == 1:
  textSize = 15
elif textSizeInput == 2:
  textSize = 30
elif textSizeInput == 3:
  textSize = 45
else:
  textSize = 15 #default is small
```

So the number entered gets converted from a string to an int for comparison purposes. Then the text size is set. A default size of 15 (small) is set if the user enters an invalid input. For example if the user want a large text size, they input “3”. This “3” is converted to an int type and then used to compare their choice. Then once a match is found between the inputted number and corresponding number, the text size is set to 45, which a large text size. The same method was used when determining where to place the text:

```
1. Top Left
2. Bottom Left
3. Top Right
4. Bottom Right
5. Middle
6. Top Middle
7. Bottom Middle
Where do you want the text? (enter the number):
```

The user then inputs a number 1-7 corresponding to where they want the text. For example, if they want it top right, they type “3”. I also had a similar approach when it comes to selecting text and border color. I didn’t want the user to have to enter a hex code or RGB values for the color they wanted so I allowed them to type the color they want using words. The list of acceptable colors is given here: https://www.computerhope.com/jargon/w/w3c-color-names.htm. I found giving the user less to think about provided a more enjoyable and creative experience. If I were to prompt the user to enter numeric values for precise text size and color, it would take the fun out of it. It would also make it more difficult for new users and would make the whole process of creating a cover art take much longer.

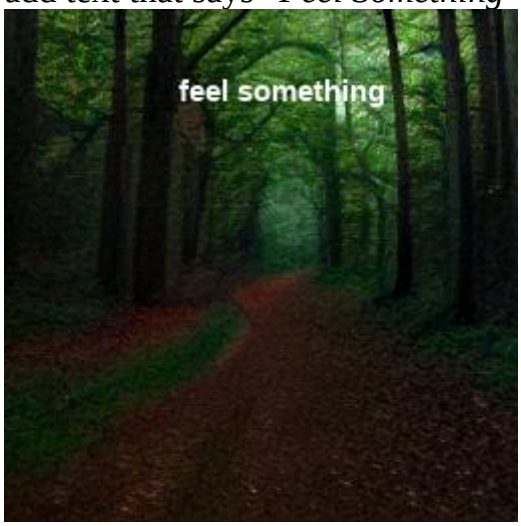
- **Results:** Here is an attempt to make an album cover with a border and a black and white version.
    - input prompt: *northern lights*
    - number of images: 1
    - black and white version

- LightGreen border
- Output Images:
- Color image:

- Black and white version:

Now to test the capabilities of, lets try to emulate already existing album covers. To do this we need to describe what the album cover is showing and use our image option to add text or borders when needed. For example, if we want to emulate the following cover art:

Our prompt will be: "*A path through a forest*" and we will need to add text that says "*Feel Something*". This is our outcome:



This isn't as good as the original as there are noticeable difference, but it shares a similar idea and showcases how useful this tool can be. Lets try another:

Prompt: "*macro shot of a cockroach with a white background*"

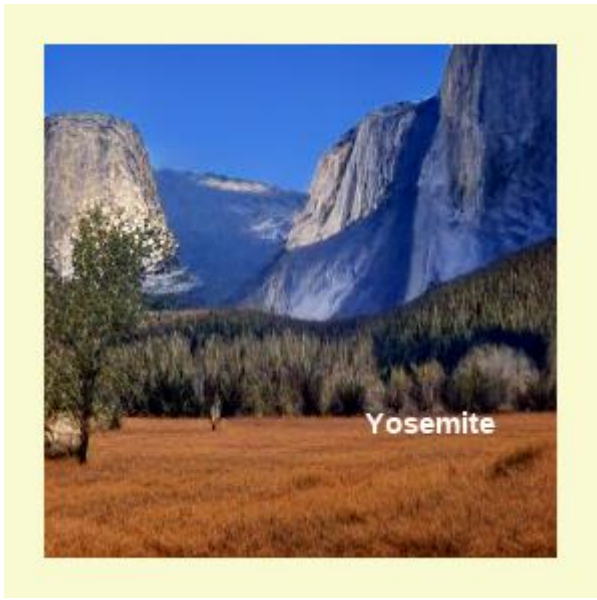
Text: PAPA ROACH
Text Location: Top Middle
Text Size: Medium
Best Output:

Again, this is not as good as the original, but it shares similar sentiment and it is good enough that it is recognizable what this is trying to emulate so long as you are familiar with the original. With that being said, the purpose of this generator is not to copy existing album cover arts. These were done as tests to show the capabilities of this program. If we are getting decent results trying to copy existing famous album covers, then it shows how powerful this program can be. At the very least, this program can function as a great tool of inspiration, brainstorming, or just getting an idea out there for people that are not graphic designers.

- **Project Management:**
  - **Work completed:** Allow the user to make adjustments to the image.
    - *Description:* With the image generated from the program, the user should be able to make basic image adjustments, like saturation, hue shift, adding text, adding a border, etc. For example, if the user wants the image to be in black and white with a white border, they should have the options to do so.
    - *Responsibility:* Ryan Tolbert
    - *Contributions:* Ryan Tolbert (100%)
    - *Issues/Concerns:* Sometimes adding text can cause issues where the text is cropped off. For example, if the text added is too long and a large size is chosen, it may be cut off.

- **Potential Future Improvements:** In its current state, this program is purely to show off the functionality of this tool. Ideally, a true user interface would be nice. A menu that allows the user to check boxes and move sliders to chose the text size and location would be much better than asking to user for each input. Also having some sort of rough preview of what the cover is going to look like as the user choses their options would cut out some confusion. I would also like to have more fonts and text options available. For example, if I want to squish the text together, or add multiple lines of text. I think for someone that wants to get an idea out quickly with not much effort, this is a very powerful tool and provides useful, but simple, options to customize the cover to the user's liking.

- **Conclusion:** To conclude, I would like to show some of my favorite covers I have made exclusively using this program:



References:

Pillow (PIL Fork): https://pillow.readthedocs.io/en/stable/reference/Image.html

Grayscale an image: https://www.entechin.com/how-to-convert-an-image-to-black-and-white-in-python/

Adding text to an image in python: https://towardsdatascience.com/adding-text-on-image-using-python-2f5bf61bf448

Adding a border to an image: https://stackoverflow.com/questions/11142851/adding-borders-to-an-image-using-python

Adding Blur to an image: https://www.tutorialspoint.com/python_pillow/python_pillow_blur_an_image.htm

Acceptable answers for the colors: https://www.computerhope.com/jargon/w/w3c-color-names.htm

BART: https://huggingface.co/docs/transformers/model_doc/bart

GitHub link: https://github.com/RyanTolbert/ArtGenerator