ECS7022P: Computational Creativity Project

Project Title: Classify Songs (CNN) and Generate Album cover art (GAN)
Student Name: Daragh Sweeney
Student ID Number: 230732443

Colab Notebook: https://colab.research.google.com/drive/13EubTSN-MBEV90ppM06nEq3-qnDwZSH1?usp=sharing
How to use: https://youtu.be/p-8u1SthVzA

Project Overview

Album covers often embody a distinct style that corresponds to the genre of music they are representing. They act as visual indicators for the style of music the listener will look forward to, serving as the first point of contact for the audience and are a big part of the music experience. We often see similar styles in genres.

For example, heavy metal album covers often adopt a dark tone with striking images and sharp fonts.

Hip Hop frequently features the musicians in dynamic poses or scenes.

Jazz also tends to focus on the artist, but can have bold colours and can sometimes feature abstract art.

Finally, classical tends to have a more traditional and elegant feel and often highlights an instrument.

There are many papers and articles that have explored this relationship [1,2,3,4], they highlight a clear correlation between album cover art and genre, although it is noted that there are often exceptions to this rule. In this project I wish to create an album cover generator in which the user inputs a song, the song is classified into a genre using a CNN, next an album cover is generated, specific to that genre using a GAN.

Figure 1: Project Overview

I hope to explore some of the distinct features in each genre and create unique and interesting album covers that highlight some of these features. I hope not to just emulate existing art covers but to capture a style and create unique and interesting pieces that capture the essence of a genre.

Generative Models

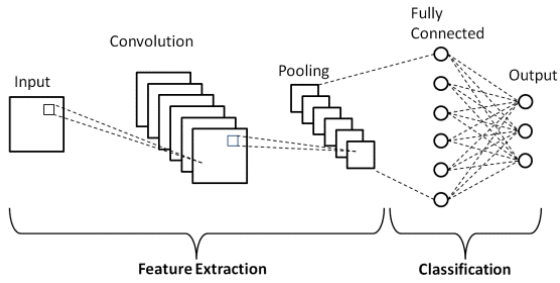
Training data

For the database the Spotify for developers API [5] was used to gather music samples and album cover art, 8 genres were selected rock, pop, classical, hip-hop, country, Latin, EDM-dance, and jazz, these were chosen as they capture a big variety in style and can be used for rough genre labels for almost any new sample. Code available here: https://github.com/Daragh-Sweeney/Download_Spotify_Datasets.git

- *The album cover database has 8 sub directories, 1 for each genre, there are between 2000-5000 images in each of the directories, the images have been reduced to 64 x 64 pixels as they require less compute and space and are easier to work with. https://www.kaggle.com/datasets/daraghsweeney/spotify-album-covers*
- *The song database is made up of 8 sub directories again, there are between 450 and 650 samples in each, the samples are 30 second previews of songs, these were captured at a 44.1 kHz sample rate. https://www.kaggle.com/datasets/daraghsweeney/spotify-tracks*

CNN Genre Classifier

CNNs are commonly used for genre classification [6,7]. My model is trained on spectrograms of the Spotify samples. The CNN extracts a 5 second sample from each piece of music and turns it into a spectrogram. These spectrograms can be thought of as images and can then be passed into a CNN network with a multi class output. I created the model with the following architecture.



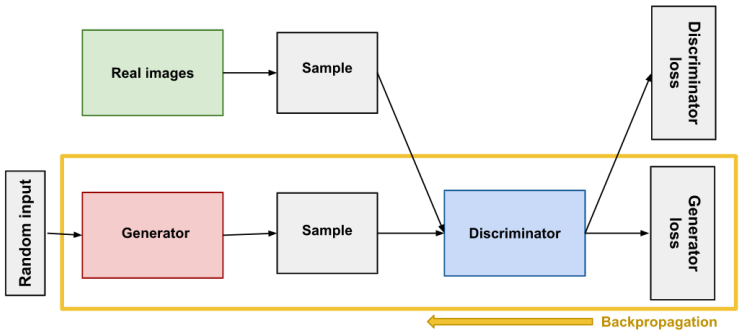
CNN Architecture

Conv2D (64) + MaxPool(2,2) + BatchNormalization
Conv2D (32) + MaxPool(2,2) + BatchNormalization
Conv2D (32) + MaxPool(2,2) + BatchNormalization
Flatten
Dense (64) + Dropout(0.50)
Dense (32) + Dropout(0.50)
Dense (16) + Dropout(0.50)
Dense(8)

Figure 2: Chosen CNN Architecture

GAN Album Cover Generator

The GAN is based on the "ARTGAN" project [8], as part of my model I used my images of album covers to train the GAN to create new Album covers. The GAN consists of a generator and discriminator trained simultaneously, the generator is trying to create features to fool the discriminator, while the discriminator tries to distinguish real from fake images. After training the generator can be passed a random latent vector to create a new image. I created my GAN with the following architecture.



Generator Architecture

ConvTranspose2d (latent_size, 512) + BatchNormalization + ReLU activation
ConvTranspose2d (512, 256) + BatchNormalization + ReLU activation
ConvTranspose2d (256, 128) + BatchNormalization + ReLU activation
ConvTranspose2d (128, 64) + BatchNormalization + ReLU activation
ConvTranspose2d (64, 3) + Tanh activation

Discriminator Architecture

Conv2d (3, 64) + BatchNormalization + LeakyReLU activation (slope=0.2)
Conv2d (64, 128) + BatchNormalization + LeakyReLU activation (slope=0.2)
Conv2d (128, 256) + BatchNormalization + LeakyReLU activation (slope=0.2)
Conv2d (256, 512) + BatchNormalization + LeakyReLU activation (slope=0.2)
Conv2d (512, 1)
Flatten
Sigmoid activation

Figure 3: Chosen GAN Architecture

Process

Example: https://youtu.be/p-8u1SthVzA

The user interacts with the system by uploading a song in mp3 format, a random 5 seconds from the song is turned into a spectrogram to be passed into the pretrained CNN for genre classification.

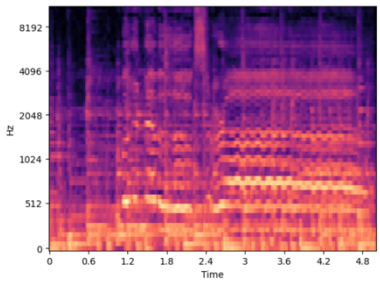


Figure 4: Sample input spectrogram

After genre is predicted, the user is asked to confirm if it is the appropriate genre, I found that genre classification is a difficult task that sometimes does not have an easy answer and so confirmation with the user is a feature to ensure they are content with the predicted genre.

Now the appropriate generator and discriminator models are retrieved. The generator is fed a latent vector 1000 times to create new images and the discriminator choses the top 5 to display to the user, the user can then download the image they prefer or rerun the script to generate new images.

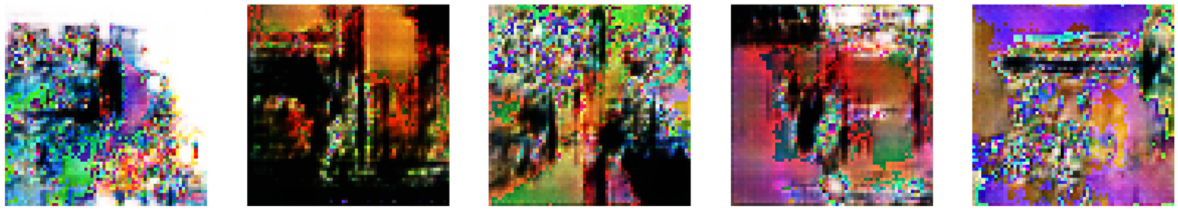


Figure 5: Sample output for Jazz

Training the CNN

While creating the CNN it was hoped to achieve a validation accuracy of around 70% [9], the model was tested with multiple architectures and hyperparameters, but the greatest accuracy achieved was 58% with the following hyperparameters (Architecture: Fig 2, Learning Rate: 0.00005, Batch Size:32, Epochs:150,Optimizer: Adam). If given more time I would like to explore further improvements however I was content with over 50% accuracy. Below is training and validation loss and accuracy for final model.

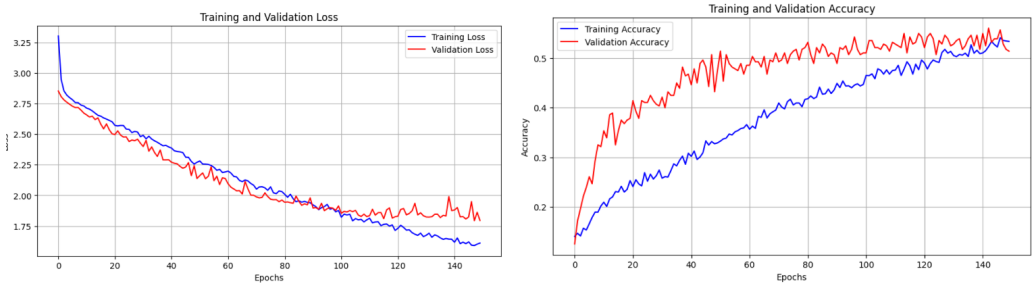


Figure 6: Final CNN loss and accuracy graphs

Training the *GAN*

To Train the GAN I used many tips to try and get more interesting shapes from my model [10], I tried with a variety of architectures and hyperparameters, however I was never able to achieve the results I wanted. Despite this I believe the artifacts are drawing some inspiration from the datasets and are gaining some features specific to their genre. I found the best results with the following hyperparameters (Architecture: Fig 3, Learning Rate: 0.00035, Batch Size:64, Latent size: 128, Epochs:150, Optimizer: Adam)

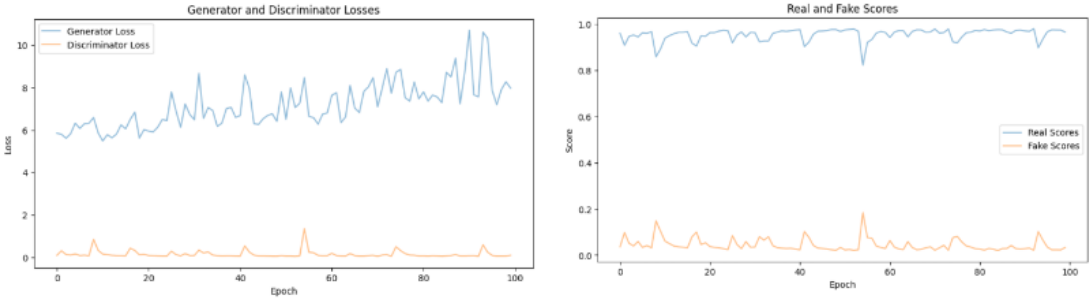
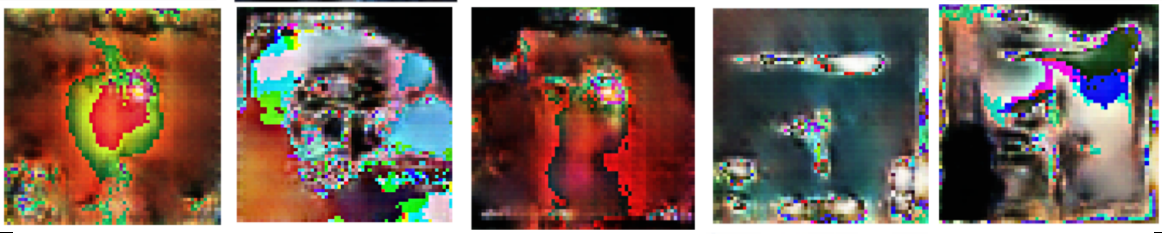
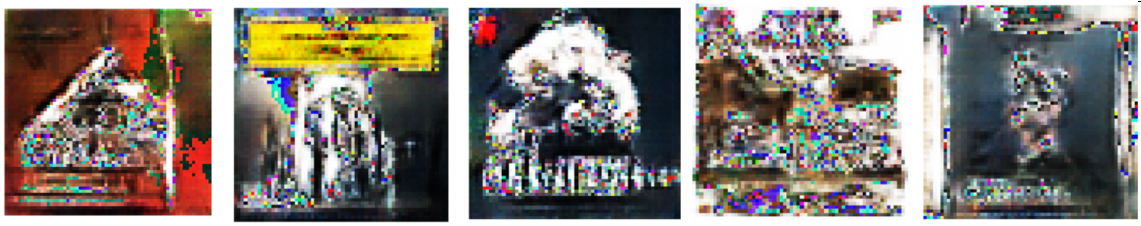
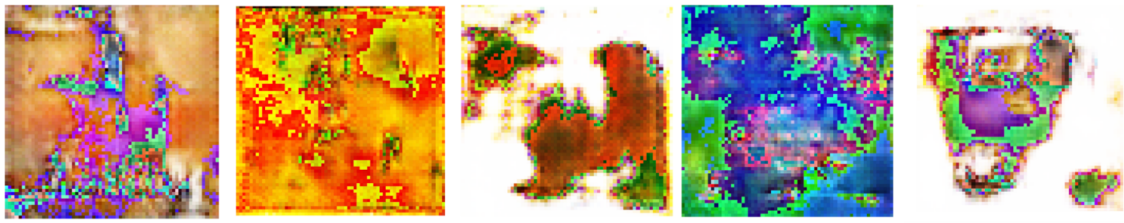
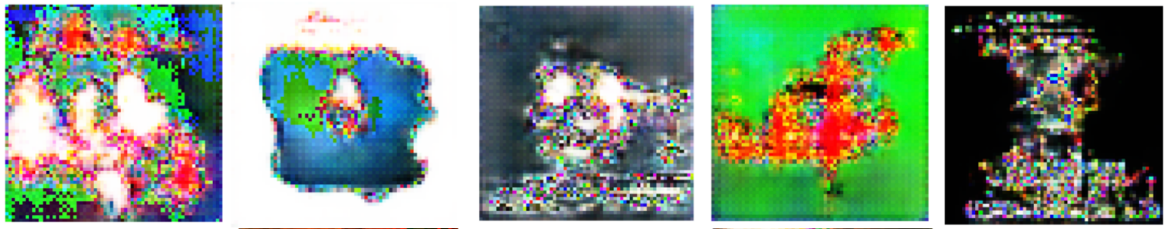
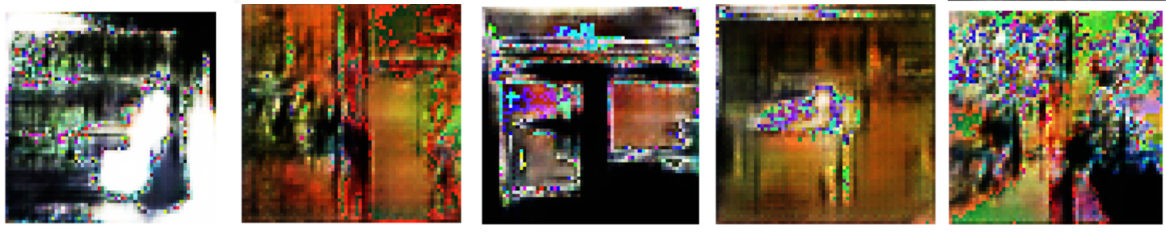


Figure 7: Generator/Discriminator losses and real/fake scores for rock GAN

Example Outputs

Please find below example outputs for each of the Music genres

Rock	
pop	
Classical	
Hip Hop	
Country	
Latin	
EDM	
Jazz	

The GAN models have been able to capture some distinct features of the genre albums, especially the colours and shapes in their respective genre, for example we see that the rock and hip-hop albums tend to be darker while country and classical tend to be lighter, also we see brighter colours in EDM and Latin. I noted that the new images produced were not as nice as the samples we see in training, for example we see below, fig 8 and fig 9, the generated images during training appear to be closer to actual album covers, however I believe that with further training these results could be improved. The following video show the training of the rock album cover generator https://www.youtube.com/watch?v=4nS-JbWD0O4

Figure 8: Comparing real rock album covers and generator covers

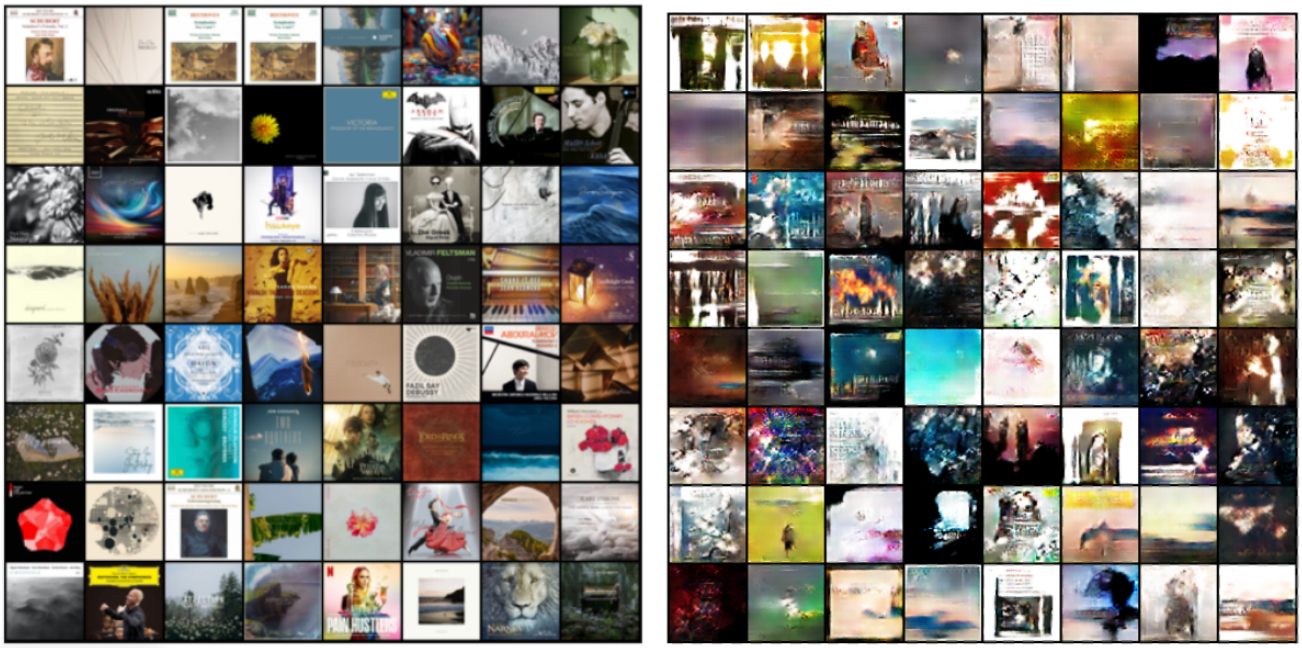


Figure 9: Comparing real classical album covers and generated covers

Evaluation

FACE evaluation

The FACE model, developed by Colton, Pease and Charnley [11] and described in lecture 9 offers a framework for describing creative systems and what they do. I liked this model as it focuses not only on the generative act but the procedural as well. The aim of my project was to create album cover art from audio, I believe I have fulfilled the 4 parts of the FACE model generative acts.

F^g**:An item of framing information:** Analysis of album covers across multiple genres
A^g: **An aesthetic measure:** This is seen as the corresponding of audio to genre to appropriate visual
C^g**: A concept:** A model that can create new album covers specific multiple genres
E^g**: An expression of a concept:** Generate new album cover specific to genre

Given the criteria I believe my system is only partially successful, I think it does not always give an appropriate aesthetic, given more training and testing I would hope to improve the results.

Results from models

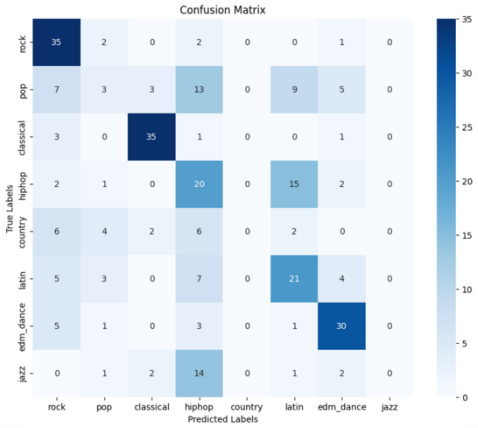


Figure 10: Confusion matrix validation results

As we can see in fig 10, the CNN genre model was good at determining rock, classical, hip hop, latin and EDM dance, however the validation accuracy is approx 58%, and so could do with some improvement for this reason I ask the user to confirm genre after classification. I believe the improvements could be made in the size of database, model architecture and the hyperparameter tunning.

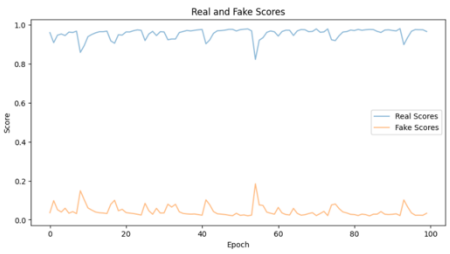
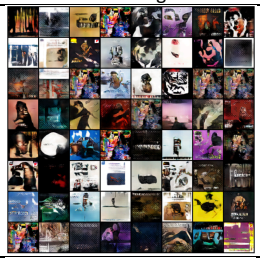
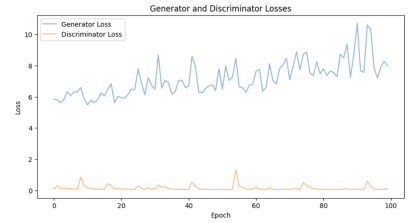
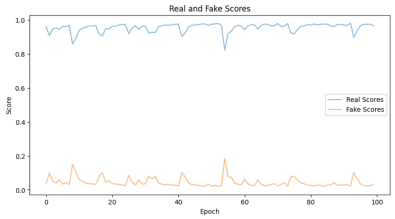
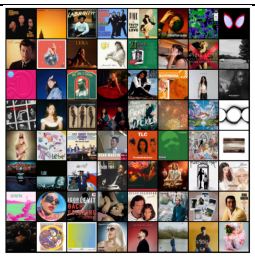
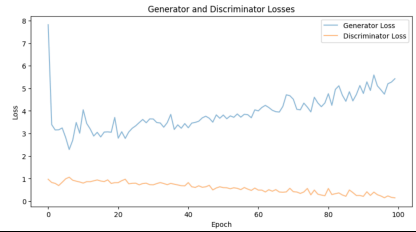
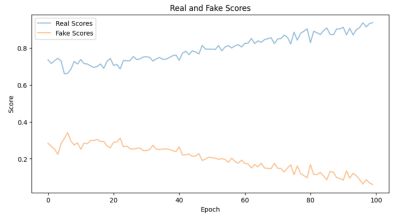
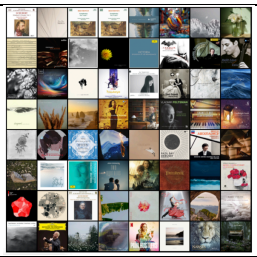
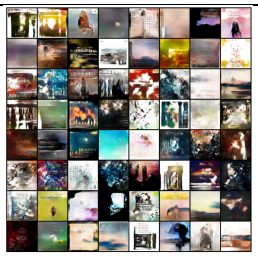
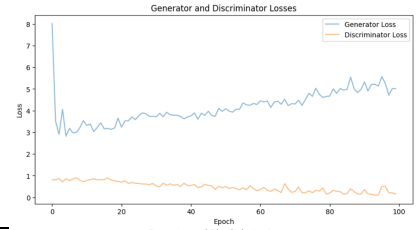
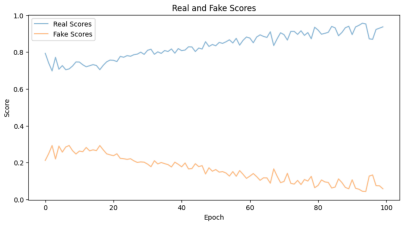
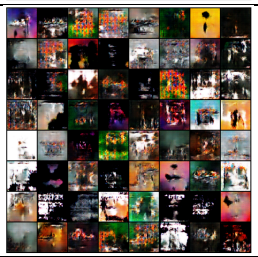
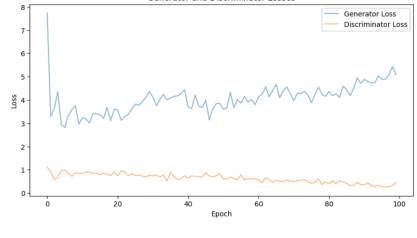
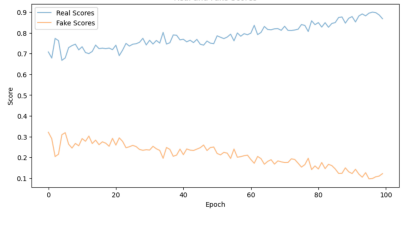
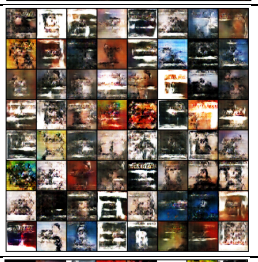
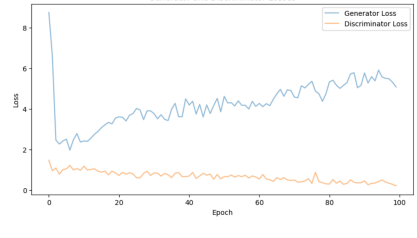
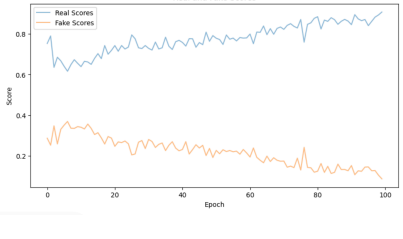
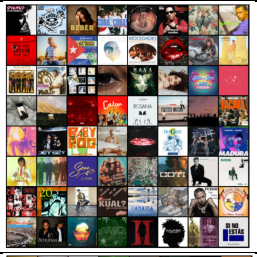
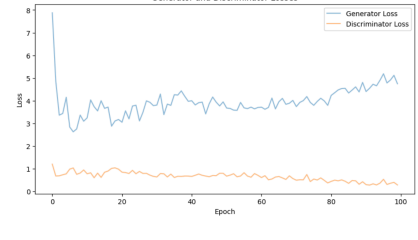
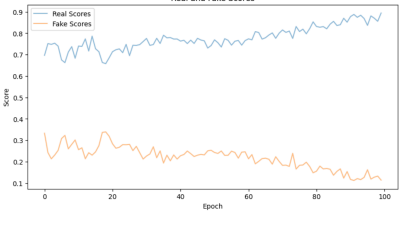
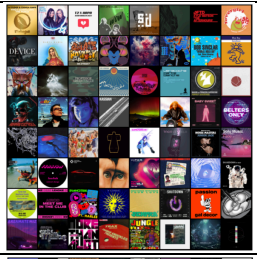
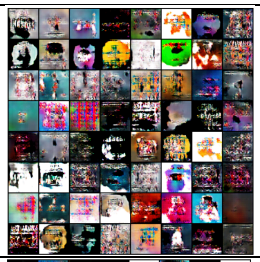
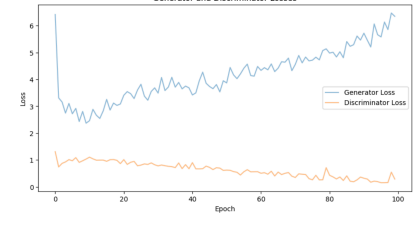
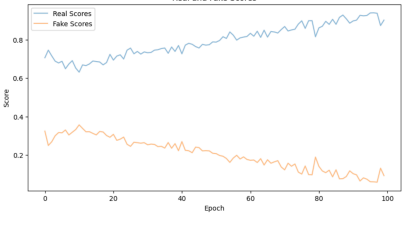
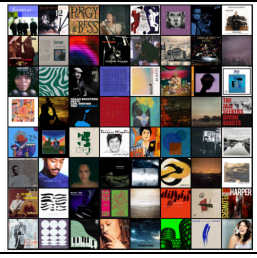
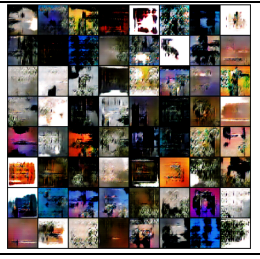
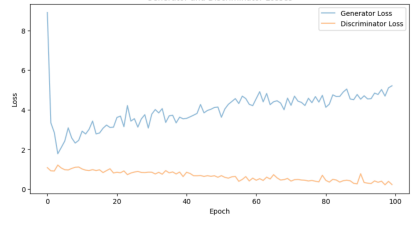
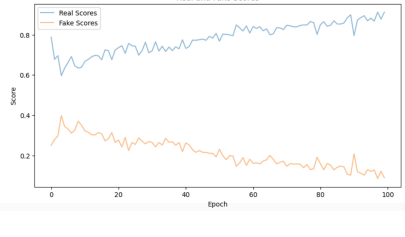


Figure 11: Real/Fake Loss score for rock

The discriminator offers an indication of the generated image's performance, when creating these models, we see that there tends to be a poor performance in general, however there are small spikes indicating the generator learning, this is typical of these models. I believe that the GAN model implemented is not complex enough to capture the varied features in the album cover dataset especially since it is diverse and non-specific. However, I am happy with some of the objects created by my model.

Genre	Album covers	Generator training	losses	Real/Fake images scores
Rock				
Pop				
Classical				
Hip Hop				
Country				
Latin				
EDM				
Jazz				

Value Added

In this project I have added value by choosing to train the models for both the CNN and the GAN from scratch rather than relying on pre trained ones, I wanted to create models specific to the task of creating album cover art from audio. I wanted to have more control over the model architecture and wanted to learn about how to create better models for myself, for these reasons I decided to do both from scratch hoping that I would have more autonomy in making changes for better results.

Similarly, I chose to create and label my own dataset, seeing the models available online I was not satisfied with their correlation to the real world and so I used the Spotify API to retrieve a more suitable and realistic dataset. I believe this has added value to the project as it means the samples are up to date and relevant, it also meant that I was able to collect as many samples as needed and didn't have to rely on available datasets.

Additionally, I have implemented multi modal capabilities in the project, the user can input a song to be classified to genre and the genre is used to create an album cover, I worked with both audio and image data to create the project which I believe adds significantly value as it creates a much more in depth experience for the user who can interact with the project with their own audio or by choosing the genre.

Finally, I have implemented and trained multiple GANs for creating album cover art, one for each of the explored genres, I did this as I was truly fascinated to see if there was any difference in the styles produced and wanted to explore if there was any difference in the outputs of the models. I believe this has both added and taken from the project, it has added to its complexity and usability as it allows the user to explore the correlation between genre and album cover, however it has drastically reduced the number of samples I have for each GAN dataset and I wonder if I had simply chosen to create any album cover regardless of genre would I have seen improved results.

Overall, I believe there are many aspects of the project that could be improved and given additional time I believe I could have fixed some of them; however, I believe the methodologies I have used has enhanced the projects value and potential, I have added a lot of depth to this task and I believe I achieved both good and bad results from my models.

Colab Notebook Code

Libraries and Functions

```
#@title Libraries and Functions
# Install and import the necessary libraries
!pip install opendatasets --upgrade --quiet
import opendatasets as od
import os
import librosa
import numpy as np
import random
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision.datasets import ImageFolder
import torchvision.transforms as tt
from torch.utils.data import DataLoader
from torchvision.utils import make_grid
from tqdm.notebook import tqdm
from tensorflow.keras.models import load_model
from tensorflow.keras import layers, models, regularizers
import tensorflow as tf
from sklearn.metrics import confusion_matrix
from IPython.display import Image
from PIL import Image, ImageFilter
from google.colab import drive
from google.colab import files

# Mount Google Drive to access saved models
drive.mount('/content/drive')
```

```
# Function to get the default device (GPU if available, else CPU)
def get_default_device():
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

# Function to move tensors to the specified device
def to_device(data, device):
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

# Class to wrap the dataloader and move data to device
class DeviceDataLoader():
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        return len(self.dl)

# Set the default device
device = get_default_device()
```

Genre Classification by CNN

```
#@title Genre Classification by CNN

#Function to sample a 5-second section of audio and create a spectrogram
def getSpectogram(filePath, duration=5, noise_factor=0.005):
    y, sr = librosa.load(filePath, duration=duration)
    y_noisy = y + noise_factor * np.random.normal(size=y.shape)
    mel_spec = librosa.feature.melspectrogram(y=y_noisy, sr=sr, n_mels=64)
    total_frames = mel_spec.shape[1]
    start_idx = random.randint(0, max(total_frames - 256, 0))
    mel_spec_slice = mel_spec[:, start_idx:start_idx + 256]
    mel_spec_db = librosa.power_to_db(mel_spec_slice, ref=np.max)
    return mel_spec_db

# Function to plot the Training vs Validation Loss and Training vs Validation Accuracy
def plot_history(history):
    plt.figure(figsize=(10, 5))
    plt.plot(history.history['loss'], label='Training Loss', color='blue')
    plt.plot(history.history['val_loss'], label='Validation Loss', color='red')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.show()

    plt.figure(figsize=(10, 5))
    plt.plot(history.history['accuracy'], label='Training Accuracy', color='blue')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy', color='red')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    plt.show()

# Function to create genre classifier model
def createGenreClassifier():

  dataset_url = 'https://www.kaggle.com/daraghsweeney/spotify-tracks' # Download the database from kaggle
  od.download(dataset_url)
  data_dir = '/content/spotify-tracks/tracks'

  # Define the percentage split for training, validation, testing and number of samples
  train_ratio = 0.8
  val_ratio = 0.1
  test_ratio = 0.1
  num_files_per_category = 400

  # Define empty lists to store file paths and labels for training, validation, and testing
  train_files,test_files,val_files = [],[],[]
  train_labels,test_labels,val_labels = [],[],[]
```

```
  # Traverse through each file in subdirectory
  for label, genre in enumerate(categories):
      genre_dir = os.path.join(data_dir, genre)

      # Get a list of all files in the genre subdirectory
      files = os.listdir(genre_dir)
      np.random.shuffle(files)

      # Take the first 100 files from each category
      files = files[:num_files_per_category]

      # Calculate the split indices
      num_files = len(files)
      train_end = int(train_ratio * num_files)
      val_end = int((train_ratio + val_ratio) * num_files)

      # Append file paths and labels to the appropriate lists based on the split indices
      for i, file in enumerate(files):

          file_path = os.path.join(genre_dir, file)
          specto = getSpectogram(file_path)

          # I found there are somtimes issues with the Spectogram sizes, I skip these
          if specto.shape != (64, 216):
            print(f"Warning: Spectrogram size for file '{file}' is not correct. Skipping...")
            continue

          if i < train_end:
              train_files.append(specto)
              train_labels.append(label)

          elif train_end <= i < val_end:
              val_files.append(specto)
              val_labels.append(label)

          else:
              test_files.append(specto)
              test_labels.append(label)

# Define the Architecture for genre model
  genre_model = models.Sequential([
      layers.Conv2D(64, (3, 3), activation='relu', input_shape=(64, 216, 1)),
      layers.MaxPooling2D((2, 2)),
      layers.BatchNormalization(),
      layers.Conv2D(32, (3, 3), activation='relu',kernel_regularizer=regularizers.l2(0.01)),
      layers.MaxPooling2D((2, 2)),
      layers.BatchNormalization(),
      layers.Conv2D(32, (3, 3), activation='relu',kernel_regularizer=regularizers.l2(0.01)),
      layers.MaxPooling2D((2, 2)),
      layers.BatchNormalization(),
      layers.Flatten(),
      layers.Dense(64, activation='relu'),
      layers.Dropout(0.5),
      layers.Dense(32, activation='relu'),
      layers.Dropout(0.5),
      layers.Dense(16, activation='relu'),
      layers.Dropout(0.5),
      layers.Dense(8, activation='softmax')
  ])

  # Define the optimizer and Compile the model
  optimizer = tf.keras.optimizers.Adam(learning_rate=0.00005)
  genre_model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

  # Train the model and plot graphs
  history = genre_model.fit(np.array(train_files), np.array(train_labels), epochs=150, batch_size=32,
                            validation_data=(np.array(val_files), np.array(val_labels)))
  plot_history(history)

  # Save the model in the users drive
  model_path = '/content/drive/My Drive/Classifier/genre_model.h5'
  genre_model.save(model_path)
```

Album Cover Generation using GAN

```
#@title Album Cover Generation using GAN

# Plot the Discriminator and Generator losses
def plot_losses(losses_g, losses_d, real_scores, fake_scores):
    plt.figure(figsize=(10, 5))
    plt.plot(losses_g, label='Generator Loss', alpha=0.5)
    plt.plot(losses_d, label='Discriminator Loss', alpha=0.5)
    plt.title('Generator and Discriminator Losses')
```

```
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

# Plot the real scores vs the fake scores
def plot_scores(real_scores, fake_scores):
    plt.figure(figsize=(10, 5))
    plt.plot(real_scores, label='Real Scores', alpha=0.5)
    plt.plot(fake_scores, label='Fake Scores', alpha=0.5)
    plt.title('Real and Fake Scores')
    plt.xlabel('Epoch')
    plt.ylabel('Score')
    plt.legend()
    plt.show()

def createAlbumCoverGAN(genre):
  # Use Kaggle to download album cover art
  dataset_url = 'https://www.kaggle.com/daraghsweeney/spotify-album-covers'
  od.download(dataset_url)

  # Import the category dataset into PyTorch
  data_dir = '/content/spotify-album-covers/album_covers'
  catagory_dir = os.path.join(data_dir, genre)
  dataset = ImageFolder(catagory_dir)

  batch_size = 64
  latent_size = 128
  stats = (0.5, 0.5, 0.5), (0.5, 0.5, 0.5)

  # This is used if the image is in a
  train_ds = ImageFolder(catagory_dir, transform=tt.Compose([tt.ToTensor(),tt.Normalize(*stats)]))
  train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=3, pin_memory=True)

  # The following functions allow us to display a batch of images
  def denorm(img_tensors):
    return img_tensors * stats[1][0] + stats[0][0]

  def show_images(images, nmax=64):
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(make_grid(denorm(images.detach()[:nmax]), nrow=8).permute(1, 2, 0))

  def show_batch(dl, nmax=64):
    for images, _ in dl:
      show_images(images, nmax)
      break

  # We show the first batch that will be used for training
  show_batch(train_dl)
  train_dl = DeviceDataLoader(train_dl, device)

  # Discriminator model
  discriminator = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(64),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(128),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(256),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(512),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Conv2d(512, 1, kernel_size=4, stride=1, padding=0, bias=False),
    nn.Flatten(),
    nn.Sigmoid())

  discriminator = to_device(discriminator, device)

  # Generator model
  generator = nn.Sequential(
    nn.ConvTranspose2d(latent_size, 512, kernel_size=4, stride=1, padding=0, bias=False),
    nn.BatchNorm2d(512),
    nn.ReLU(True),
    nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(256),
    nn.ReLU(True),
    nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(128),
    nn.ReLU(True),
    nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1, bias=False),
```

```
    nn.BatchNorm2d(64),
    nn.ReLU(True),
    nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2, padding=1, bias=False),
    nn.Tanh()
)

# Create a random latent tensor to be passed into the generator
xb = torch.randn(batch_size, latent_size, 1, 1)
fake_images = generator(xb)

def train_discriminator(real_images, opt_d):
    # Clear discriminator gradients
    opt_d.zero_grad()

    # Pass real images through discriminator
    real_preds = discriminator(real_images)
    real_targets = torch.ones(real_images.size(0), 1, device=device)
    real_loss = F.binary_cross_entropy(real_preds, real_targets)
    real_score = torch.mean(real_preds).item()

    # Generate fake images
    latent = torch.randn(batch_size, latent_size, 1, 1, device=device)
    fake_images = generator(latent)

    # Pass fake images through discriminator
    fake_targets = torch.zeros(fake_images.size(0), 1, device=device)
    fake_preds = discriminator(fake_images)
    fake_loss = F.binary_cross_entropy(fake_preds, fake_targets)
    fake_score = torch.mean(fake_preds).item()

    # Update discriminator weights
    loss = real_loss + fake_loss
    loss.backward()
    opt_d.step()
    return loss.item(), real_score, fake_score

generator = generator.to(device)

def train_generator(opt_g):
# Clear generator gradients
    opt_g.zero_grad()

    # Generate fake images
    latent = torch.randn(batch_size, latent_size, 1, 1, device=device)
    fake_images = generator(latent)

    # Try to fool the discriminator
    preds = discriminator(fake_images)
    targets = torch.ones(batch_size, 1, device=device)
    loss = F.binary_cross_entropy(preds, targets)

    # Update generator weights
    loss.backward()
    opt_g.step()
    return loss.item()

sample_dir = 'generated'
os.makedirs(sample_dir, exist_ok=True)

def save_samples(index, latent_tensors, show=True):
    fake_images = generator(latent_tensors)
    fake_fname = 'generated-images-{0:0=4d}.png'.format(index)
    save_image(denorm(fake_images), os.path.join(sample_dir, fake_fname), nrow=8)
    print('Saving', fake_fname)
    if show:
        fig, ax = plt.subplots(figsize=(8, 8))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(fake_images.cpu().detach(), nrow=8).permute(1, 2, 0))

fixed_latent = torch.randn(64, latent_size, 1, 1, device=device)
save_samples(0, fixed_latent)

def fit(epochs, lr, start_idx=1):
    torch.cuda.empty_cache()
    # Losses & scores
    losses_g_avg, losses_d_avg, real_scores_avg, fake_scores_avg = [],[],[],[]

    # Create optimizers
    opt_d = torch.optim.Adam(discriminator.parameters(), lr=lr, betas=(0.5, 0.999))
    opt_g = torch.optim.Adam(generator.parameters(), lr=lr, betas=(0.5, 0.999))

    for epoch in range(epochs):

        losses_g, losses_d, real_scores, fake_scores = [],[],[],[]
```

```
      for real_images, _ in tqdm(train_dl):
        # Train discriminator and generator
        loss_d, real_score, fake_score = train_discriminator(real_images, opt_d)
        loss_g = train_generator(opt_g)

        # Record losses & scores
        losses_g.append(loss_g)
        losses_d.append(loss_d)
        real_scores.append(real_score)
        fake_scores.append(fake_score)

      # I want to get the average values for epoch
      losses_g_avg.append(sum(losses_g) / len(losses_g))
      losses_d_avg.append(sum(losses_d) / len(losses_d))
      real_scores_avg.append(sum(real_scores) / len(real_scores))
      fake_scores_avg.append(sum(fake_scores) / len(fake_scores))

      # Log losses & scores (last batch)
      print("Epoch [{}/{}], "
      "loss_g: {:.4f}, "
      "loss_d: {:.4f}, "
      "real_score: {:.4f}, "
      "fake_score: {:.4f}".format(epoch+1, epochs, loss_g, loss_d, real_score, fake_score))

      # Save generated images
      save_samples(epoch+start_idx, fixed_latent, show=False)
    return losses_g_avg, losses_d_avg, real_scores_avg, fake_scores_avg

  # Create the model with the following hyperparameters
  lr = 0.00035
  epochs = 200
  history = fit(epochs, lr)
  losses_g, losses_d, real_scores, fake_scores = history

  # Plot losses and scores
  plot_losses(losses_g, losses_d, real_scores, fake_scores)
  plot_scores(real_scores, fake_scores)

  # Save the generated model
  torch.save(generator.state_dict(), '/content/drive/My Drive/generators/'+Category+'_G.pth')
  torch.save(discriminator.state_dict(), '/content/drive/My Drive/discriminators/'+Category+'_D.pth')
```

Input song and get predicted Genre

```
#@title Input song and get predicted Genre

categories = ["rock","pop","classical","hiphop","country","latin","edm_dance","jazz"]

# Define the file path to the Pretrained CNN
model_path = '/content/drive/My Drive/Classifier/genre_model.h5'

# This code will be used to classify song if model is available
def classifysong():
  loaded_model = load_model(model_path)

  # Prompt the user to upload a file
  print("Please upload a song file:")
  uploaded_file = files.upload()

  # Get the file name of the uploaded song
  file_name = list(uploaded_file.keys())[0]

  # Extract spectrogram from the user-input song
  new_spectrogram = getSpectogram(file_name)

  librosa.display.specshow(new_spectrogram,  x_axis='time', y_axis='mel')

  # Convert the prediction to a genre label and get the genre from list
  prediction = loaded_model.predict(np.expand_dims(new_spectrogram, axis=0))
  predicted_genre_index = np.argmax(prediction)
  predicted_genre = categories[predicted_genre_index]

  print("Predicted genre:", predicted_genre)
  return predicted_genre

# If the model is available use it for classification
if os.path.exists(model_path):
  predicted_genre = classifysong()

# If there is no model already there then we create a new one
else:
```

```
  print("Model does not exist so lets create a new one.")
  print("Please rerun this cell after the model has been created.")
  createGenreClassifier()
  predicted_genre = classifysong()
```

Confirm selected genre is correct

```
@title Confirm selected genre is correct

import ipywidgets as widgets

# Define a variable to store the selected genre
selected_genre = predicted_genre  # Initialize with the predicted genre
print('The predicted label for this song is :'+str(predicted_genre))
print('Please leave dropdown if this is correct, otherwise update')

# Dropdown widget with preset predicted genre and all available genres
genre_dropdown = widgets.Dropdown(
    options=categories,
    value=predicted_genre,
    description='Genre:',
    disabled=False,
)

# Function to handle dropdown value change
def on_dropdown_change(change):
    global selected_genre  # Use the selected_genre variable defined outside the function
    selected_genre = change.new  # Update the selected_genre variable with the new value
    print("Updated genre:", selected_genre)  # Print the updated genre value

# Assign the function to be called when dropdown value changes
genre_dropdown.observe(on_dropdown_change, names='value')

# Display the dropdown widget
display(genre_dropdown)
```

Generate Album Cover Art

```
#@title Generate Album Cover Art

Category = selected_genre
print(Category)

# Load the pretrained generator and discriminator
generator_path = '/content/drive/My Drive/generators/'+Category+'_G.pth'
discriminator_path = '/content/drive/My Drive/discriminators/'+Category+'_D.pth'

# Check if both paths exist if they do use the pretrained models
# Else create a new model with the input genre
if os.path.exists(generator_path) and os.path.exists(discriminator_path):

  generator_checkpoint = torch.load(generator_path, map_location=torch.device('cpu'))
  discriminator_checkpoint = torch.load(discriminator_path, map_location=torch.device('cpu'))

  # Instantiate the generator and discriminator models
  latent_size = 128
  generator_load = nn.Sequential(
    nn.ConvTranspose2d(latent_size, 512, kernel_size=4, stride=1, padding=0, bias=False),
    nn.BatchNorm2d(512),
    nn.ReLU(True),
    nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(256),
    nn.ReLU(True),
    nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(128),
    nn.ReLU(True),
    nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(64),
    nn.ReLU(True),
    nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2, padding=1, bias=False),
    nn.Tanh())

  discriminator_load = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(64),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(128),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1, bias=False),
    nn.BatchNorm2d(256),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1, bias=False),
```

```
    nn.BatchNorm2d(512),
    nn.LeakyReLU(0.2, inplace=True),
    nn.Conv2d(512, 1, kernel_size=4, stride=1, padding=0, bias=False),
    nn.Flatten(),
    nn.Sigmoid())

  # Load the state dictionarys into the generator and discriminator models
  generator_load.load_state_dict(generator_checkpoint)
  generator_load.eval()
  discriminator_load.load_state_dict(discriminator_checkpoint)
  discriminator_load.eval()
  generator_load = generator_load.to(device)
  discriminator_load = discriminator_load.to(device)

  # Generate 1000 images
  num_images = 1000
  top_images = [(float('-inf'), None)] * 5  # Initialize a list to store the top 5 images and their scores

  for _ in range(num_images):

      # Generate image
      latent_vector = torch.randn(1, latent_size, 1, 1).to(device)
      with torch.no_grad():
          generated_image = generator_load(latent_vector)

      # Pass the generated image through the discriminator
      generated_image = generated_image.to(device)
      discriminator_score = discriminator_load(generated_image)

      # Update the top images list
      top_images.sort(key=lambda x: x[0])  # Sort the list based on scores
      if discriminator_score > top_images[0][0]:
          top_images[0] = (discriminator_score, generated_image.clone().cpu())


  # Convert to PIL images
  pil_images = [tt.transforms.ToPILImage()(image.squeeze(0)) for _, image in top_images]

  # Display the top 5 generated images
  fig, axes = plt.subplots(1, 5, figsize=(15, 3))
  for i, image in enumerate(pil_images):
      axes[i].imshow(image)
      axes[i].axis('off')
  plt.show()

  # Save the top generated images to files
  for i, image in enumerate(pil_images):
      image.save(f'top_generated_image_{i}.png')

# If no model is available we create a new one with the genre passed in
else:
    print("One or both paths do not exist.")
    createAlbumCoverGAN(Category)
```

Usage of Automated Coding Tools

I used ChatGPT at several points when creating this project, although it was not used for the generation of any code or text it was valuable in understanding errors in code and in brainstorming ideas, this helped a lot for small problems where I knew the issue but could not pinpoint the place where the issue was happening.

References

1. Dorochowicz, A. and Kostek, B. (2019) '*Relationship between album cover design and music genres*', 2019 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA) [Preprint]. doi:10.23919/spa.2019.8936738.
2. *Genre Classification via album cover | semantic scholar*. Available at: https://www.semanticscholar.org/paper/Genre-Classification-via-Album-Cover-Li/e7c341eebec955cd77e1cc3d8a7d1ffb6aaacad0 (Accessed: 24 April 2024).
3. Milano, B. (2023) *Can you judge an album by it's cover? how artwork reflects the music:* Udiscover, uDiscover Music. Available at: https://www.udiscovermusic.com/stories/can-you-judge-an-album-by-its-cover/ (Accessed: 24 April 2024).
4. Venkatesan, T., Wang, Q.J. and Spence, C. (2022) '*Does the typeface on album cover influence expectations and perception of music?*', Psychology of Aesthetics, Creativity, and the Arts, 16(3), pp. 487–503. doi:10.1037/aca0000330.
5. Web API | Spotify for Developers. Available at: https://developer.spotify.com/documentation/web-api/ (Accessed: 20 April 2024).
6. Viroux, G. (2021) *Creating a music genre classifier using a convolutional neural network*, Medium. Available at: https://glenn-viroux.medium.com/creating-a-music-genre-classifier-using-a-convolutional-neural-network-548d06658cee (Accessed: 24 April 2024).
7. Yang, H. and Zhang, W.-Q. (2019) '*Music genre classification using duplicated convolutional layers in neural networks*', Interspeech 2019 [Preprint]. doi:10.21437/interspeech.2019-1298.
8. Nandi, S. (2021) 'ARTGAN' - a simple generative adversarial networks based on art images using Deeplearning &..., Medium. Available at: https://medium.com/analytics-vidhya/artgan-a-b77ecb1bc25a (Accessed: 25 April 2024).
9. Music genre classification. Available at: https://cs229.stanford.edu/proj2018/report/21.pdf (Accessed: 23 April 2024).
10. Brownlee, J. (2019) How to get started with generative adversarial networks (7-day mini-course), MachineLearningMastery.com. Available at: https://machinelearningmastery.com/how-to-get-started-with-generative-adversarial-networks-7-day-mini-course/ (Accessed: 14 April 2024).
11. Colton, S., Charnley, J. and Pease, A. (2011) Computational creativity theory: The face and IDEA descriptive models, Discovery. Available at: https://discovery.dundee.ac.uk/en/publications/computational-creativity-theory-the-face-and-idea-descriptive-mod-2 (Accessed: 14 April 2024).