Using AI to Create Monet-Style Portraits of Film Characters to Sell as NFT

Course Code: COM 5508

Course Title: Media and Data Analytics (2022/2023)

Course Instructor: Dr. Xiaofun LIU

Group 11

ZHOU Xingyu 57274685 ZHOU Hongxin 57452863

PENG Jinfeng 57452550 WU Yuting 57572676

Communication and New Media

City University of Hong Kong

GitHub Link

https://github.com/createpjf/AIGC-to-NFT

Abstract

AI painting is considered one of the future directions in the field of artistic creation. At the same time, the form of NFT digital collectibles has also impacted the way of buying and selling collectibles. In the rapidly changing AI era, we intend to use AI to create Monet style portraits of film characters to sell as NFT to explore and integrate the two emerging trends. This project connects image to image translation and text to image translation through GPT, and tries to tap the new market of NFT digital art creation and application.

1. Introduction

NFTs, or non-fungible tokens, have gained popularity as a fresh method of purchasing and selling digital art in recent years. These unique digital assets are stored on a blockchain and providing artists with a direct way to make money from collectors. Beeple,a digital artist, sold an NFT of his work *Everydays: the First 5000 Days* at Christie's for $69,346,250 in March, 2021. This was the third most expensive living artist's work to ever be sold at auction.(Hufnagel, S., & King, C. ,2023)This auction has made NFT art works famous. However, it is not a unique instance. In December 2021, PAK's *The Merge* sold for $91.8m, setting a new sales record for NFT artworks （Block, F.,2021）. This all reveals the great prospects of NFT commercial art.

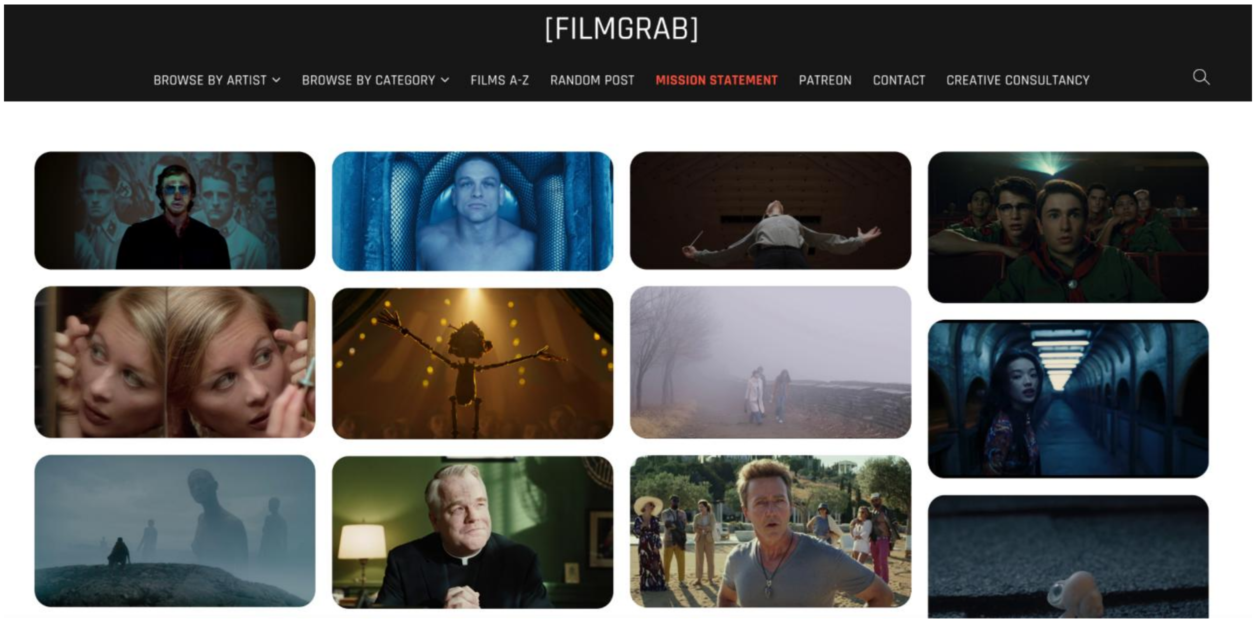
At the same time, developments in AI and machine learning have made it feasible to produce sophisticated digital art that is incredibly lifelike. In October 2018, a portrait generated by a machine learning (ML) algorithm called Generative Adversarial Network (GAN) was sold for $432500, which was more than 40 times the estimated price, shocking the art world. This painting called *Edmond De Bellamy* is hailed as the "first example of AI generated art in auctions" (Shahriar, S., & Al Roken, N. ,2022), which has attracted more attention to the combination of AI and digital art. We can produce distinctive, aesthetically arresting digital art that appeals to collectors looking for something really unique by utilizing the power of AI. AI has showed great promise in the creation of images that resemble the aesthetic of many schools of painting, such as new impressionist paintings (Gonsalves, R., 2021) and Chinese landscape painting creation (Xue, A., 2021). It is feasible to generate new photos with a comparable aesthetic to the originals by training machine learning models on extensive datasets of the work.

Classic film works have timeless characteristics, and as a painter with a strong personal style, Monet provides us with a learning direction for AI deep learning technology in our project. Therefore, our project aims to explore the intersection of these two trends (AI painting and NFT digital art) by using artificial intelligence to create Monet style portraits of movie characters that can be sold through NFT. From data mining to DLIB recognition of portraits in movie stills, to using GANs for Monet style image to image translation, we also used GPT to graft CLIP models for text to image translation technology, endowing computer-generated NFT works of movie portrait art with more personality and emotional links, rather than simply imitating artistic styles, so that these NFT works have higher market response and collection value. In addition, we explored the existing NFT trading market, and determined the online platform direction of our ai digital art works by comparing blockchain, popularity and transaction types. By exploring the application of AI art in NFT creation, this project will help provide practical significance for the practice of digital art in the AI era.

2. Method

2.1 Data source

At first, we intended to scrape the movie stills we needed from IMDb or Douban since these two platforms have already categorized the stills and have abundant resources. However, after successfully scraping the stills using Python crawlers, we encountered a serious problem: the quality of the stills on these platforms is too low, which prevents us from performing facial recognition in the subsequent steps. Therefore, we switched to a new platform, FilmGrab. This is a popular film stills and screenshot database created by Cinephile Donnacha, who collects and curates high-quality images from films spanning several decades and genres. The website offers a vast collection of film stills and screenshots for film enthusiasts and scholars to explore and use in their research and personal projects. In addition, these movie screenshots are all manually collected, which is valuable in AI color matching and imitation. Compared to official promotional photos, this provides a premium basis for subsequent portrait art creation.

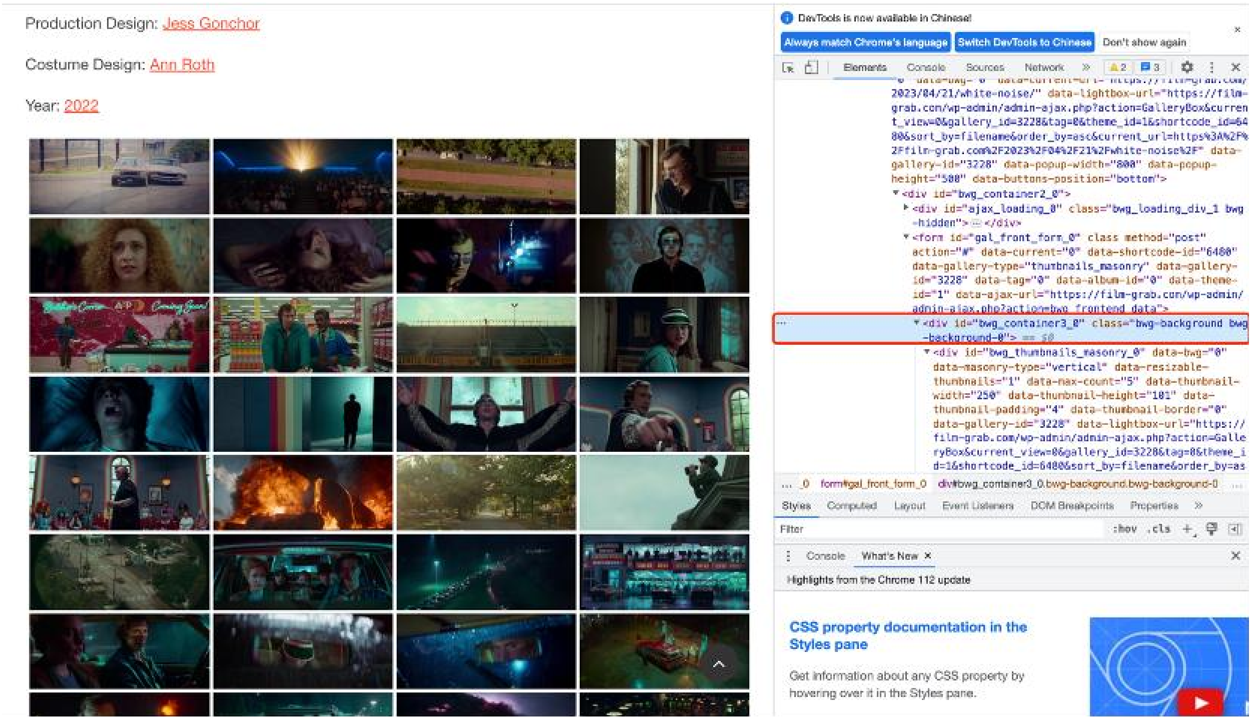


2.2 Image Data Collection

At first, we import three libraries – “requests”, “BeautifulSoup”, and “os” for making HTTP requests, parsing HTML documents, and interacting with the operating system.

Then we create a session object using the requests library, sets the user agent header to mimic a web browser and access the home page to get cookies that will be used in subsequent requests.
After that, we define the URL of *FilmGrab* and use the session object to get the HTML content of the page. The HTML content is then parsed using BeautifulSoup and stored in the 'soup' variable.

We utilized Google's developer tools to locate the tabs that contain the images on the web page and discovered that they are located within a div element with a class attribute of "bwg-background bwg-background-0".


Production Design: Jess Gonchor

Costume Design: Ann Roth

Year: 2022

DevTools is now available in Chinese!

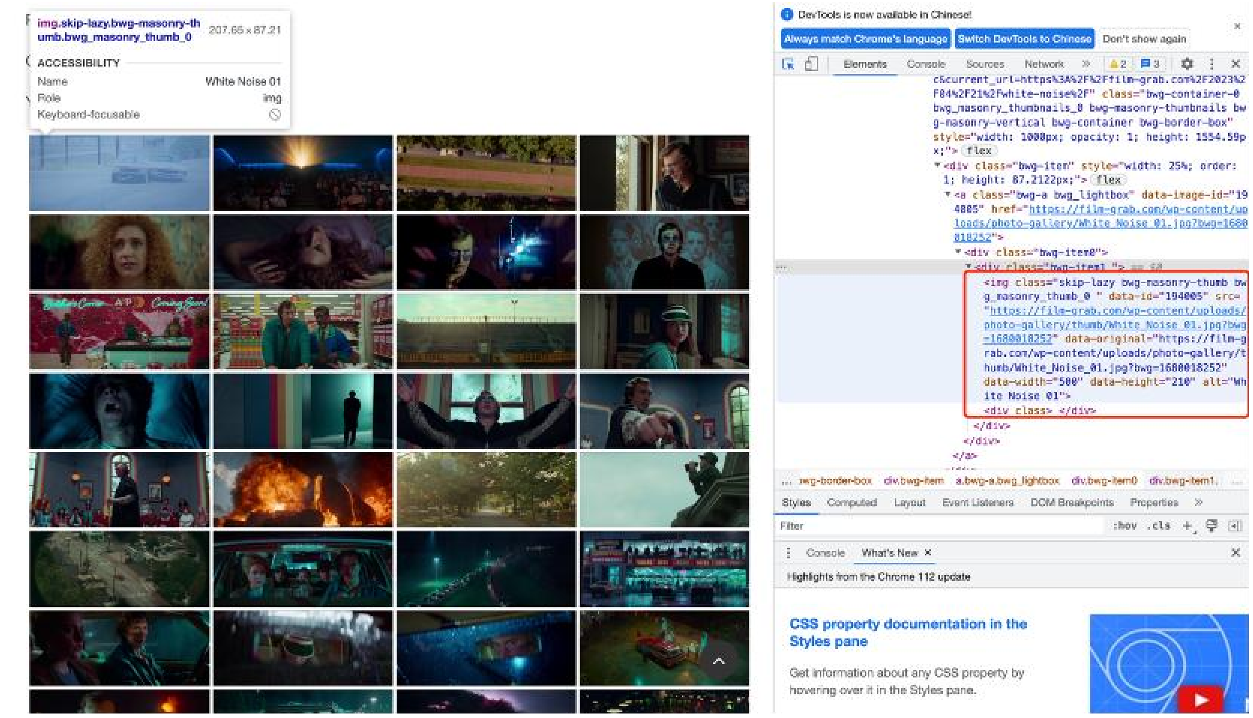
```
2023/04/21/white-noise/" data-lightbox-url="https://film-grab.com/wp-admin/admin-ajax.php?action=GalleryBox&current_view=0&gallery_id=3228&tag=0&theme_id=1&shortcode_id=6480&sort_by=filename&order_by=asc&current_url=https%3A%2F%2Ffilm-grab.com%2F2023%2F04%2F21%2Fwhite-noise%2F" data-gallery-id="3228" data-popup-width="800" data-popup-height="500" data-buttons-position="bottom">
  ▼<div id="bwg_container2_0">
    ▶<div id="ajax_loading_0" class="bwg_loading_div_1 bwg-hidden">…</div>
    ▼<form id="gal_front_form_0" class method="post" action="#" data-current="0" data-shortcode-id="6480" data-gallery-type="thumbnails_masonry" data-gallery-id="3228" data-tag="0" data-album-id="0" data-theme-id="1" data-ajax-url="https://film-grab.com/wp-admin/admin-ajax.php?action=bwg_frontend_data">
      ▼<div id="bwg_container3_0" class="bwg-background bwg-background-0"> == $0
        ▼<div id="bwg_thumbnails_masonry_0" data-bwg="0" data-masonry-type="vertical" data-resizable-thumbnails="1" data-max-count="5" data-thumbnail-width="250" data-thumbnail-height="101" data-thumbnail-padding="4" data-thumbnail-border="0" data-gallery-id="3228" data-lightbox-url="https://film-grab.com/wp-admin/admin-ajax.php?action=GalleryBox&current_view=0&gallery_id=3228&tag=0&theme_id=1&shortcode_id=6480&sort_by=filename&order_by=as
```

… _0 form#gal_front_form_0 div#bwg_container3_0.bwg-background.bwg-background-0

Highlights from the Chrome 112 update

CSS property documentation in the Styles pane

Get information about any CSS property by hovering over it in the Styles pane.


img.skip-lazy.bwg-masonry-thumb.bwg_masonry_thumb_0 207.65 × 87.21

ACCESSIBILITY

Name	White Noise 01
Role	img
Keyboard-focusable	⃠

DevTools is now available in Chinese!

```
c&current_url=https%3A%2F%2Ffilm-grab.com%2F2023%2F04%2F21%2Fwhite-noise%2F" class="bwg-container-0 bwg_masonry_thumbnails_0 bwg-masonry-thumbnails bwg-masonry-vertical bwg-container bwg-border-box" style="width: 1000px; opacity: 1; height: 1554.59px;"> flex
  ▼<div class="bwg-item" style="width: 25%; order: 1; height: 87.2122px;"> flex
    ▼<a class="bwg-a bwg_lightbox" data-image-id="194005" href="https://film-grab.com/wp-content/uploads/photo-gallery/White_Noise_01.jpg?bwg=1680018252">
      ▼<div class="bwg-item0">
        ▼<div class="bwg-item1 "> == $0
            <img class="skip-lazy bwg-masonry-thumb bwg_masonry_thumb_0 " data-id="194005" src="https://film-grab.com/wp-content/uploads/photo-gallery/thumb/White_Noise_01.jpg?bwg=1680018252" data-original="https://film-grab.com/wp-content/uploads/photo-gallery/thumb/White_Noise_01.jpg?bwg=1680018252" data-width="500" data-height="210" alt="White Noise 01">
            <div class> </div>
          </div>
        </div>
      </a>
```

… bwg-border-box div.bwg-item a.bwg-a.bwg_lightbox div.bwg-item0 div.bwg-item1.

Highlights from the Chrome 112 update

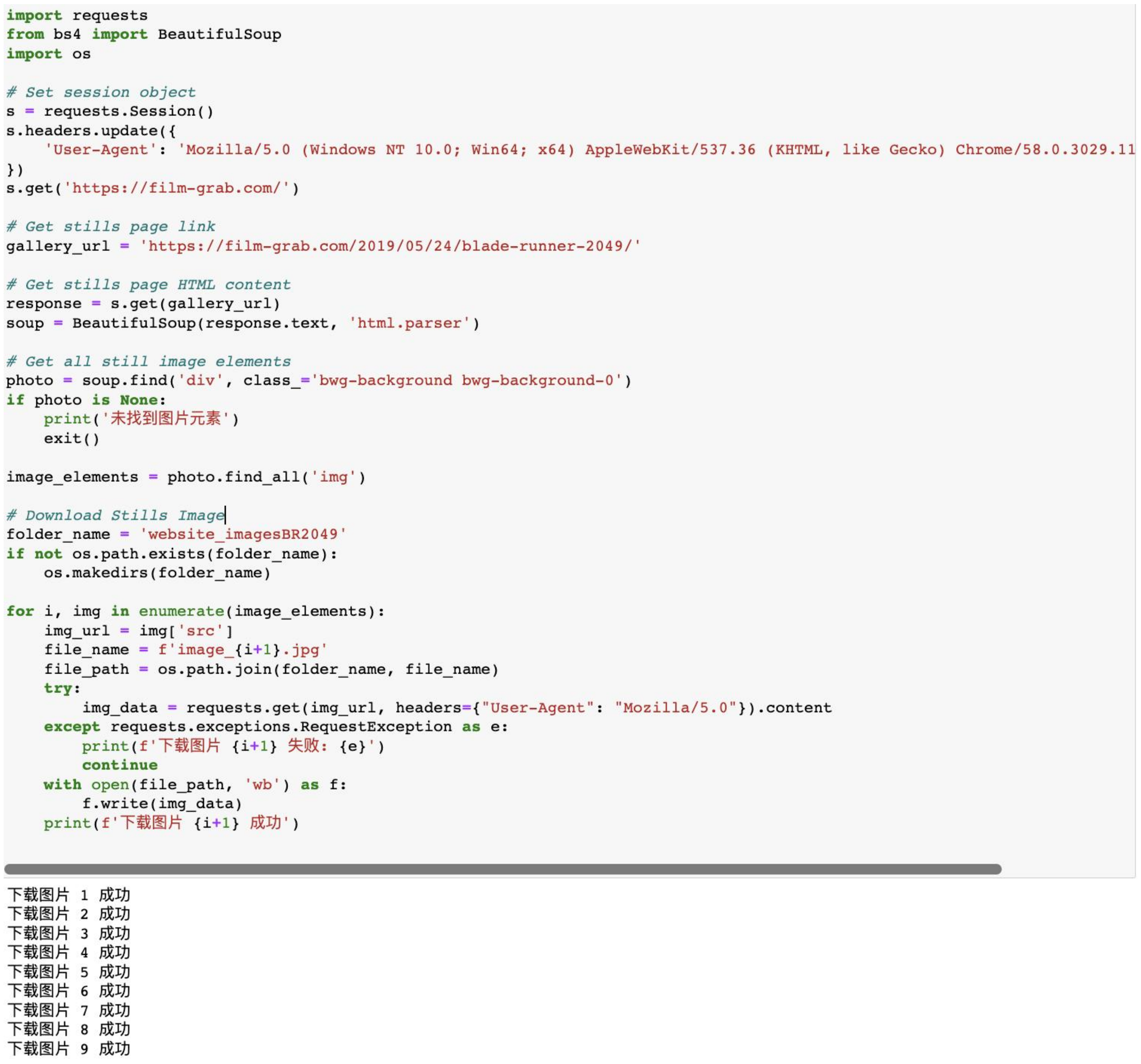
CSS property documentation in the Styles pane

Get information about any CSS property by hovering over it in the Styles pane.

Therefore, we use some lines of code to find that div element on the page with class 'bwg-background bwg-background-0' and then find all ‘img’ elements within that div. If no image elements are found, the program prints a message and exits.

Lastly, we use some code to download each image from the page and store it in a folder we named before. The code checks if the folder exists, and if it does not, creates the folder using the os library. The code then loops through each ‘img’ element and extracts the URL of the image. The URL is then used to download the image data using the requests library. If there is an error while downloading the image, the code prints an error message and continues to the next image. If the image is successfully downloaded, it is saved to the specified file path using the 'wb' flag to indicate that it should be written in binary mode.

Finally, the code prints a message indicating that the image has been downloaded. By saving the website images of the *FilmGrab* to the local computer path, our image collection work has been preliminarily completed.



```python
import requests
from bs4 import BeautifulSoup
import os

# Set session object
s = requests.Session()
s.headers.update({
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.11
})
s.get('https://film-grab.com/')

# Get stills page link
gallery_url = 'https://film-grab.com/2019/05/24/blade-runner-2049/'

# Get stills page HTML content
response = s.get(gallery_url)
soup = BeautifulSoup(response.text, 'html.parser')

# Get all still image elements
photo = soup.find('div', class_='bwg-background bwg-background-0')
if photo is None:
    print('未找到图片元素')
    exit()

image_elements = photo.find_all('img')

# Download Stills Image
folder_name = 'website_imagesBR2049'
if not os.path.exists(folder_name):
    os.makedirs(folder_name)

for i, img in enumerate(image_elements):
    img_url = img['src']
    file_name = f'image_{i+1}.jpg'
    file_path = os.path.join(folder_name, file_name)
    try:
        img_data = requests.get(img_url, headers={"User-Agent": "Mozilla/5.0"}).content
    except requests.exceptions.RequestException as e:
        print(f'下载图片 {i+1} 失败: {e}')
        continue
    with open(file_path, 'wb') as f:
        f.write(img_data)
    print(f'下载图片 {i+1} 成功')
```

```
下载图片 1 成功
下载图片 2 成功
下载图片 3 成功
下载图片 4 成功
下载图片 5 成功
下载图片 6 成功
下载图片 7 成功
下载图片 8 成功
下载图片 9 成功
```

2.3 Gathering and Pre-Processing the Training Images

We wrote two scripts to gather the source images and process them into a single file. The first gathers public domain paintings on *Kaggle* from a pre-processed Monet-style image dataset. The second collects portraits from *FilmGrab,* a popular film stills and screenshot database created by Cinephile Donnacha, who collects and curates high-quality images from films spanning several decades and genres. The website offers a vast collection of film stills and screenshots for film enthusiasts and scholars to explore and use in their research and personal projects.

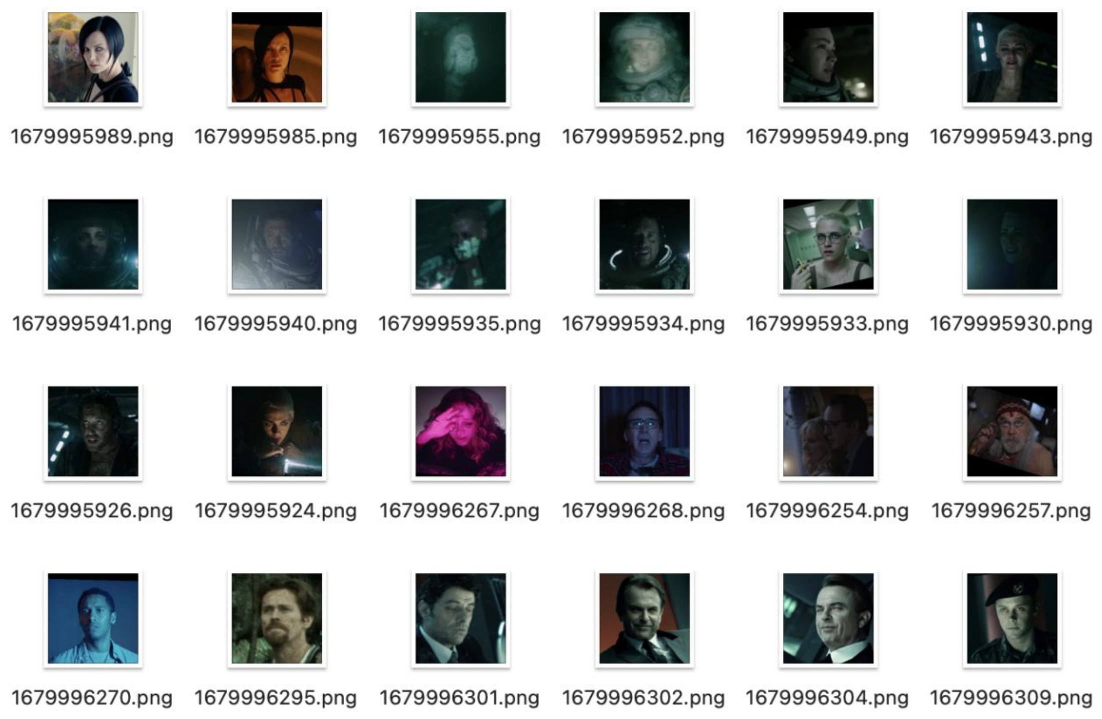
To find and orient the faces in the images, we used a face-finding algorithm from a package called DLIB. DLIB is a popular open-source software library that provides tools and algorithms for machine learning, computer vision, and image processing. One of the key features of DLIB is its face detection and recognition capabilities, which make it a powerful tool for tasks such as face identification, face tracking, and facial expression analysis.

The face-finding package in DLIB is based on a combination of two main algorithms: a Histogram of Oriented Gradients (HOG) feature descriptor and a Support Vector Machine (SVM) classifier. The HOG algorithm computes a dense gradient map of an input image, which captures the local intensity gradients in different directions. The SVM classifier then uses this feature descriptor to identify regions in the image that are likely to contain faces.

```python
# import functions
import cv2
import dlib
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import math
import os


# Load all the models we need: a detector to find the faces, a shape predictor
# to find face landmarks so we can precisely localize the face
def find_faces(face_file_path):

  detector = dlib.get_frontal_face_detector()
  sp = dlib.shape_predictor("/Users/leopeng/Desktop/face/shape_predictor_5_face_landmarks.dat")

  # Load the image using Dlib
  img = dlib.load_rgb_image(face_file_path)
  # Ask the detector to find the bounding boxes of each face. The 1 in the
  # second argument indicates that we should upsample the image 1 time. This
  # will make everything bigger and allow us to detect more faces.
  dets = detector(img, 1)
  images = []

  num_faces = len(dets)
  if num_faces > 0:
    # Find the 5 face landmarks we need to do the alignment.
    faces = dlib.full_object_detections()
    #print("len(faces)", len(faces))
    for detection in dets:
        area = detection.area()
        min_area= (1*1)
        if detection.area() > min_area:
            faces.append(sp(img, detection))

    # Get the aligned face images, image size = 256, padding size 1.25
    if len(faces) > 0:
          images = dlib.get_face_chips(img, faces, size=256, padding= 1.25)
```

Once a face is detected, DLIB can also perform facial landmark detection, which involves identifying specific points on the face such as the eyes, nose, and mouth. This is done using a pre-trained shape predictor model that is also included in the library.

To detect faces in the input image, we adjusted the resolution of the input image and cropped it to make smaller faces appear larger. In addition, since dlib requires RGB images, we will convert the images from BGR to RGB channel order.. Here are some of the results from some sci-fi films. It can be seen that the portraits have been correctly detected.



2.4 Photo to Monet — CycleGAN

For image style transfer part, we used the CycleGAN (Cycle-Consistent Adversarial Networks) architecture to build the model and trained the model by 300 Monet style photos. The main deep learning framework we used is pytorch_lightning. In particular, I will focus on the preprocessing of data and the architecture of the generator and discriminator.

Firstly, let's discuss the preprocessing of data. Before loading the datasets, we define CustomTransform for image augmentation. We define a custom image transformation pipeline that includes random cropping and resizing with RandomResizedCrop, and random flipping with RandomHorizontalFlip. These transformations introduce more variety in the images during training instead of learning from the same set of images in every epoch, which can improve the model's ability to generalize to new images. The chosen transformations were selected to avoid large changes to the distributions of the real images. Finally, the images are scaled down for better convergence. The author emphasizes that these transformations are only applied during model training, and not during inference.

```
class CustomTransform(object):
    def __init__(self, img_dim=256):
        self.transform = T.Compose([
            T.RandomResizedCrop((img_dim, img_dim),
                                scale=(0.8, 1.0), ratio=(0.9, 1.1)),
            T.RandomHorizontalFlip(p=0.5),
        ])

    def __call__(self, img, stage="fit"):
        if stage == "fit":
            img = self.transform(img)
        return img/127.5 - 1
```

To prepare the datasets for training and prediction, we load them into torch.utils.data.DataLoader and generate the dataloader for the training dataset and the prediction dataset. The dataset is defined with three main methods: init to initialize the dataset, len to retrieve the size of the dataset, and getitem to retrieve the i-th image sample after applying the transformations described earlier. Additionally, the stage argument is defined to differentiate between the training and prediction datasets when applying the transformations, and different instances of the CustomDataset class are used to retrieve the photos and Monet paintings separately.

```
SAMPLE_SIZE=5
dm_sample = CustomDataModule(loader_config={"batch_size": SAMPLE_SIZE})

dm_sample.setup("fit")
train_loader = dm_sample.train_dataloader()
monet_samples = next(iter(train_loader))["monet"]

dm_sample.setup("predict")
predict_loader = dm_sample.predict_dataloader()
photo_samples = next(iter(predict_loader))

show_img(monet_samples, nrow=SAMPLE_SIZE, title="Samples of Monet Paintings")
show_img(photo_samples, nrow=SAMPLE_SIZE, title="Samples of Photos")
```

Later, the datasets are combined while iterating through them.

Samples of Monet Paintings
Samples of Photos

Moving on to the architecture of the generator and discriminator, we use a U-Net architecture for the CycleGAN generator. U-Net is a network which consists of downsampling blocks and upsampling blocks with skip connections, giving it the U-shaped architecture.

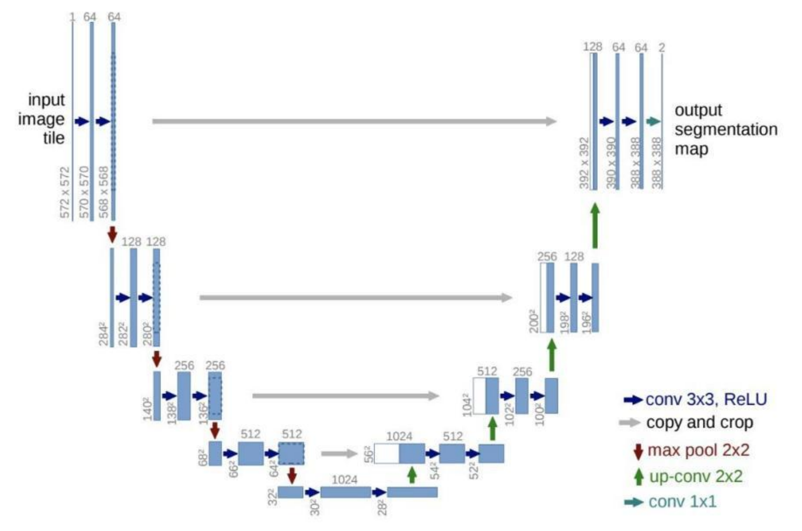


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

The downsampling blocks use convolution layers to increase the number of feature maps while reducing the dimensions of the 2D image. On the other hand, the upsampling blocks contain transposed convolution layers, which combine the learned features to output an image with the original size of 256.

With the building blocks defined, we can now build our CycleGAN generator. In the upsampling path, we concatenate the outputs of the upsampling blocks and the outputs of the downsampling blocks symmetrically. This can be seen as a kind of skip connection, facilitating information flow in deep networks and reducing the impact of vanishing gradients. The CycleGAN model consists of two generators and two discriminators, using the Adam optimizer for model training. The generators are used for photo-to-Monet and Monet-to-photo translations, while the discriminators are used for Monet paintings and photos.

Discriminator.

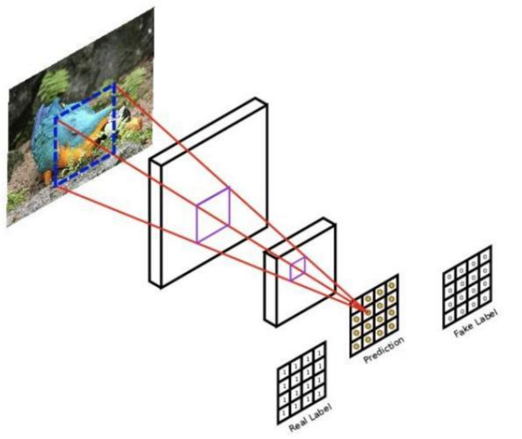
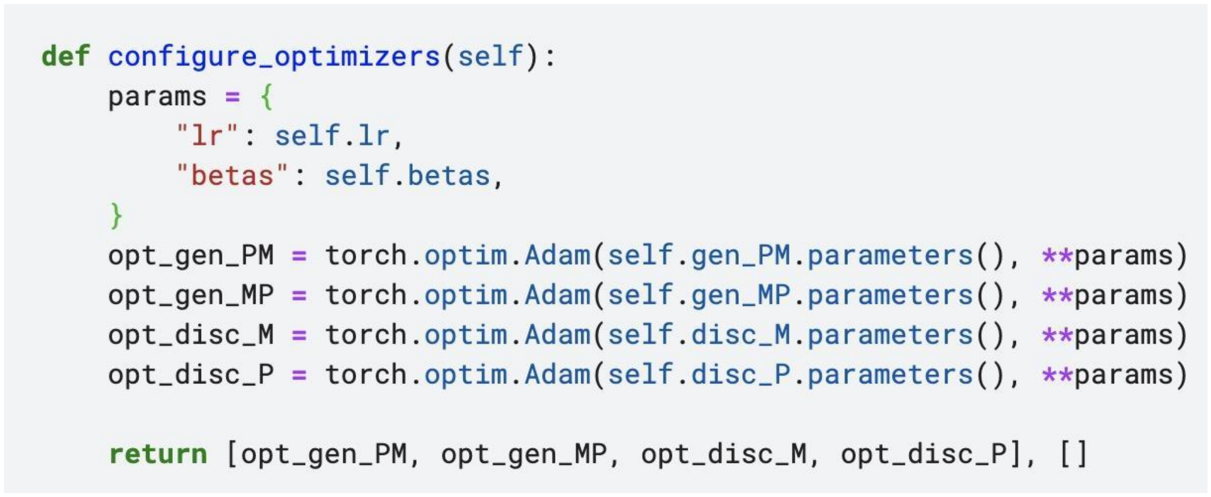


Diagram of how the PatchGAN discriminator works [source].

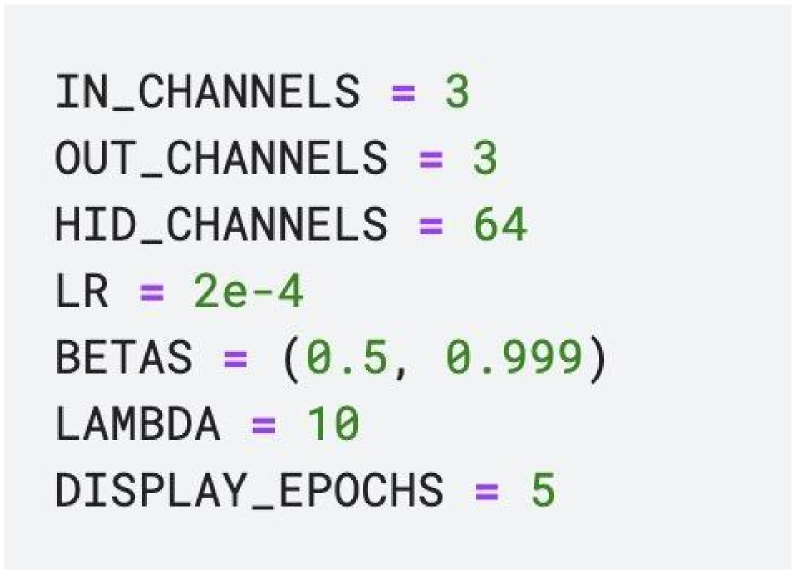
The PatchGAN discriminator consists of a sequence of convolution layers, which can be built using the downsampling blocks defined earlier. To optimize these parameters, we need to define the loss functions, including the adversarial loss, identity loss, and cycle loss.

In order to organize the modeling code for CycleGAN, the functions mentioned above are defined within the LightningModule class, along with several key methods such as init for initializing the generators, discriminators, and other parameters, forward for generating Monet-style images from input photos, configure_optimizers for defining the Adam optimizers, training_step for computing the loss functions for the generators and discriminators, and training_epoch_end for printing the average values of the loss functions over batches per epoch, and visualizing the performance of gen_PM. Additionally, predict_step is defined to run the forward method during prediction. Other useful methods can also be found.



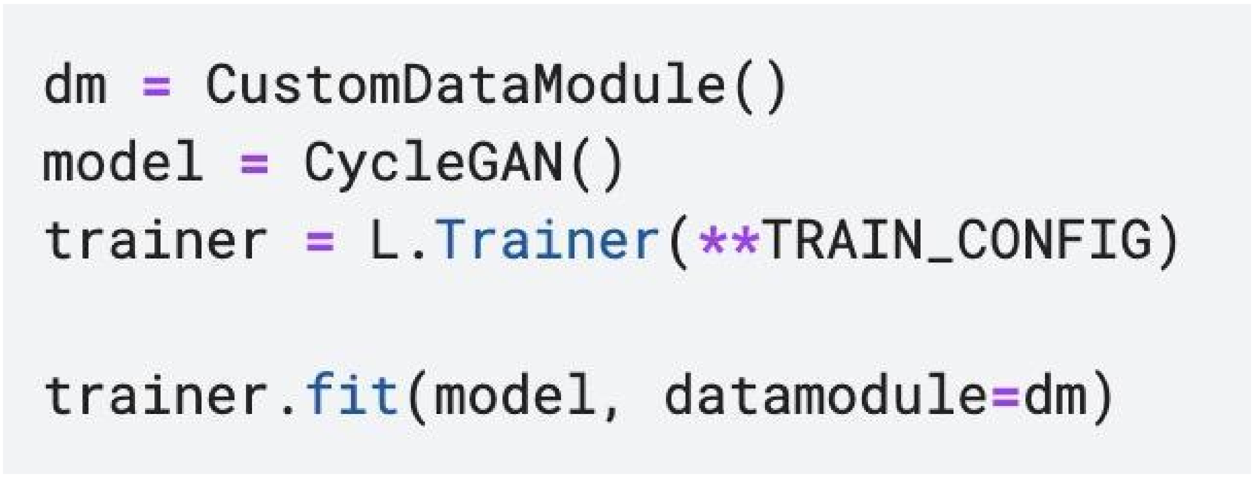
```python
def configure_optimizers(self):
    params = {
        "lr": self.lr,
        "betas": self.betas,
    }
    opt_gen_PM = torch.optim.Adam(self.gen_PM.parameters(), **params)
    opt_gen_MP = torch.optim.Adam(self.gen_MP.parameters(), **params)
    opt_disc_M = torch.optim.Adam(self.disc_M.parameters(), **params)
    opt_disc_P = torch.optim.Adam(self.disc_P.parameters(), **params)

    return [opt_gen_PM, opt_gen_MP, opt_disc_M, opt_disc_P], []
```

The model parameters include IN_CHANNELS and OUT_CHANNELS for the generator and discriminator, HID_CHANNELS for the first layer of the generator and discriminator, LR and BETAS for the Adam optimizer, LAMBDA for the weight used in the identity loss and cycle loss, and DISPLAY_EPOCHS for the frequency of visualizing the performance of gen_PM.

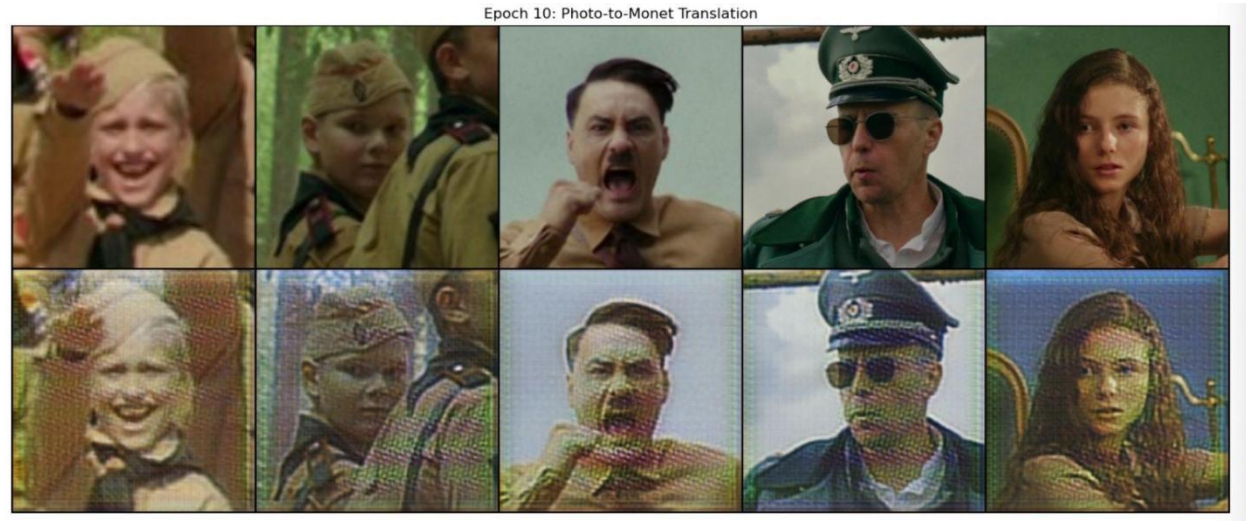


```python
IN_CHANNELS = 3
OUT_CHANNELS = 3
HID_CHANNELS = 64
LR = 2e-4
BETAS = (0.5, 0.999)
LAMBDA = 10
DISPLAY_EPOCHS = 5
```

Finally, to start training the model, we use pl.Trainer to automatically handle the training loop by running the fit method. During the training step, the model transforms a photo to a Monet painting and then back to a photo. The difference between the original photo and the twice-transformed photo is the cycle-consistency loss. We want the original photo and the twice-transformed photo to be similar to one another.

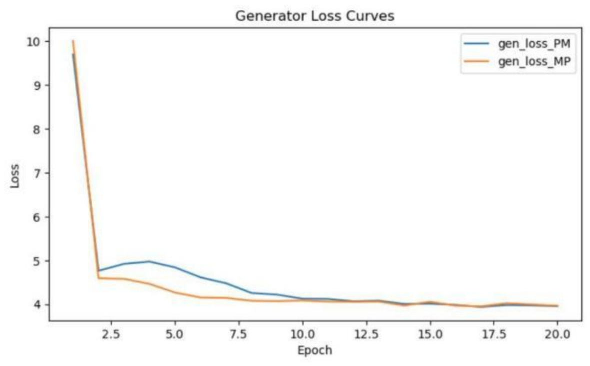


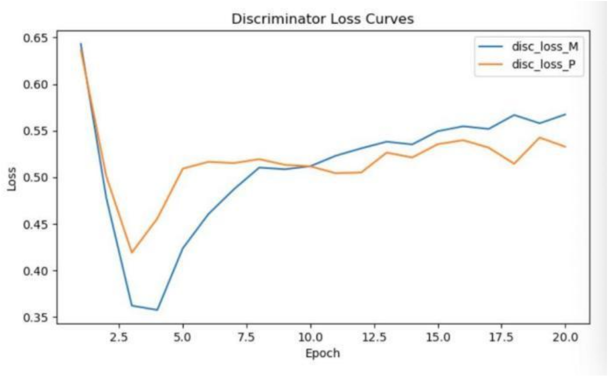
```python
dm = CustomDataModule()
model = CycleGAN()
trainer = L.Trainer(**TRAIN_CONFIG)

trainer.fit(model, datamodule=dm)
```



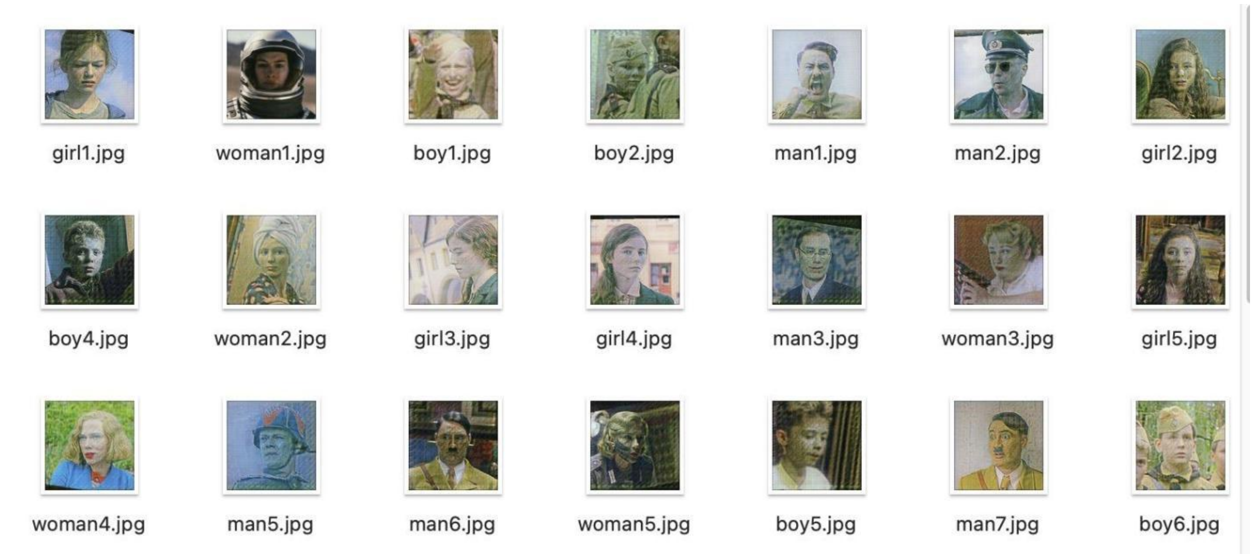
```
Epoch 11 - gen_loss_PM: 4.14807 - gen_loss_MP: 4.01358 - disc_loss_M: 0.51776 - disc_loss_P: 0.52431
Epoch 12 - gen_loss_PM: 4.13169 - gen_loss_MP: 4.07594 - disc_loss_M: 0.53893 - disc_loss_P: 0.51899
Epoch 13 - gen_loss_PM: 4.11863 - gen_loss_MP: 4.04974 - disc_loss_M: 0.54290 - disc_loss_P: 0.53587
Epoch 14 - gen_loss_PM: 4.10748 - gen_loss_MP: 4.07260 - disc_loss_M: 0.53939 - disc_loss_P: 0.53218
Epoch 15 - gen_loss_PM: 4.07853 - gen_loss_MP: 4.07730 - disc_loss_M: 0.55385 - disc_loss_P: 0.53790
```

Then we plot the loss curves. To examine the loss curves of the generators and discriminators, we record the losses in training_epoch_end and define an additional method loss_curves above to plot the curves.





With these steps, we can create a powerful model to generate Monet-style images from input photos. And these are the comparison photos of the effects before and after the style transfer.



2.5 Text to Image Translation Using GPT

In this section, we will integrate the GPT API with the previous code (Monet), aiming to automatically generate labels for locally rendered Monet-style images using OpenAI's model. We utilized the CLIP model to obtain features or labels of images, and then matched and displayed relevant images based on the input text. This model can embed both images and text into the same vector space, which means we can compare images and text by calculating the cosine similarity between them. In result, we can find the most relevant image from a set of images based on a given textual description. It starts by importing the necessary libraries and loading the pre-trained CLIP model. Then, it defines an image preprocessing pipeline, including resizing the image, center cropping, converting the image to tensor format, and normalizing.

```python
import torch
from torchvision.transforms import Compose, Resize, CenterCrop, ToTensor, Normalize
from PIL import Image
import openai
import clip

openai.api_key = "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"

# 加载 CLIP 模型
device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load("ViT-B/32", device=device)

# 设置图像预处理
image_transform = Compose([
    Resize(256, interpolation=Image.BICUBIC),
    CenterCrop(224),
    ToTensor(),
    Normalize((0.48145466, 0.4578275, 0.40821073), (0.26862954, 0.26130258, 0.27577711))
])
```

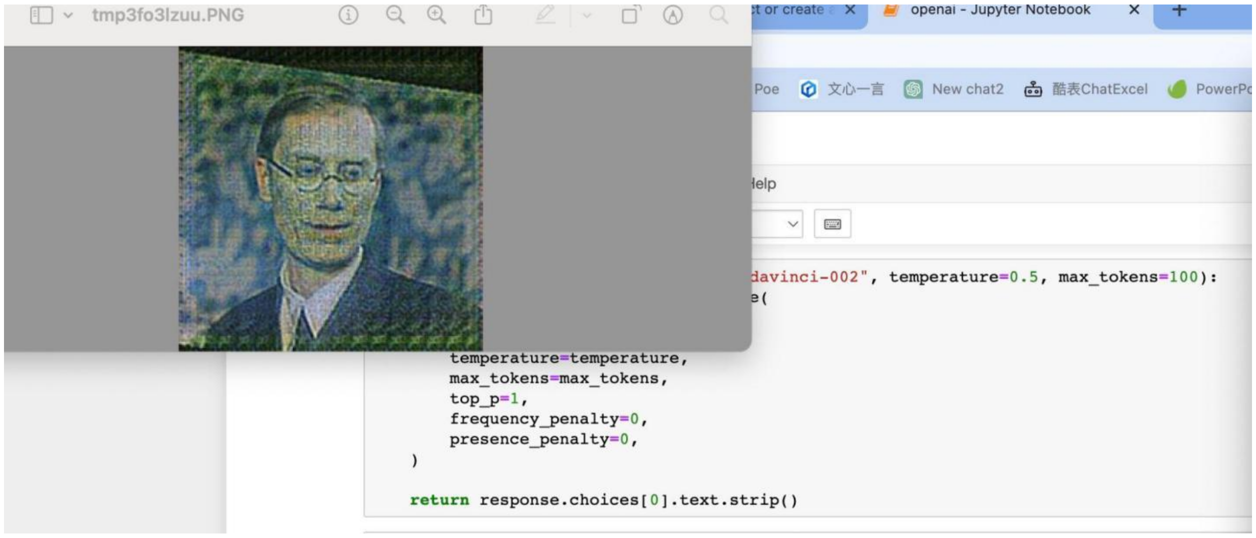
Next, it defines a function called encode_images that retrieves the embedding vectors for all images in an image folder.

```python
import os

def encode_images(image_folder):
    image_files = [f for f in os.listdir(image_folder) if f.endswith(('.jpg', '.jpeg', '.png'))]
    image_embeddings = []

    for image_file in image_files:
        image_path = os.path.join(image_folder, image_file)
        image = Image.open(image_path).convert("RGB")
        input_image = image_transform(image).unsqueeze(0).to(model.device)
        with torch.no_grad():
            embedding = model.encode_image(input_image)
        image_embeddings.append((image_path, embedding))

    return image_embeddings
```

Additionally, it defines a function called show_image_by_text that computes the cosine similarity between the text and image embedding vectors and finds the best-matching image.

```python
import random
from datasets import load_metric

def show_image_by_text(image_embeddings, text):
    text_token = model.encode_text(torch.tensor([model.tokenizer.encode(text).ids], device=model.device))

    similarities = []
    for image_path, image_embedding in image_embeddings:
        similarity = load_metric("cosine").compute(text_token.cpu().numpy(), image_embedding.cpu().numpy())
        similarities.append((image_path, similarity))

    best_image_path, best_similarity = max(similarities, key=lambda x: x[1])
    image = Image.open(best_image_path)
    image.show()
```

Finally, the code prompts the user to input a text description and passes the input text to the show_image_by_text function, which finds and displays the most relevant image based on the input text.

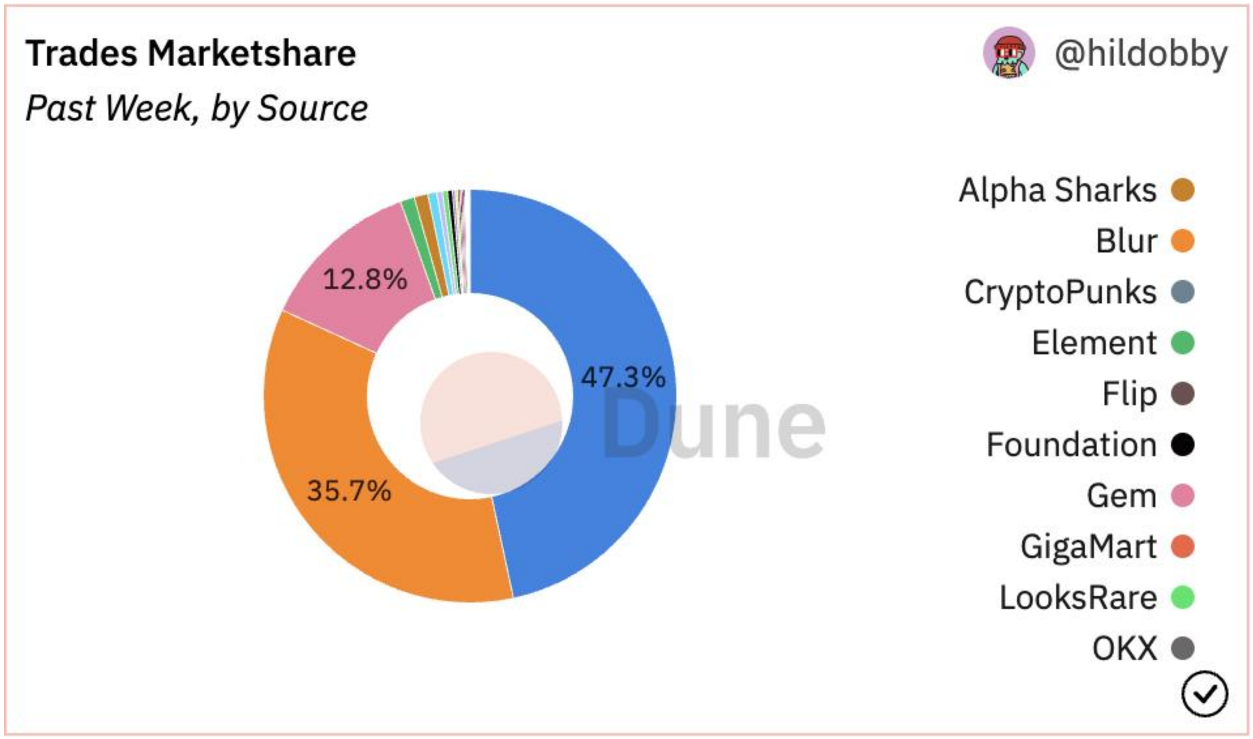


```
davinci-002", temperature=0.5, max_tokens=100):
e(
        temperature=temperature,
        max_tokens=max_tokens,
        top_p=1,
        frequency_penalty=0,
        presence_penalty=0,
    )

    return response.choices[0].text.strip()
```

Due to the limited amount of data, the number of labels we can currently input is restricted, and the accuracy of the model's automatic classification is not high. If various styles and themes of images can be added to the training in the future, the success rate of automatic matching will be greatly improved.

2.6 Minting AIGC Paintings on NFT Marketplace

Choosing the right NFT marketplace is essential for trading NFTs effectively and securely. Here are some factors to consider when selecting a marketplace:



Blockchain:

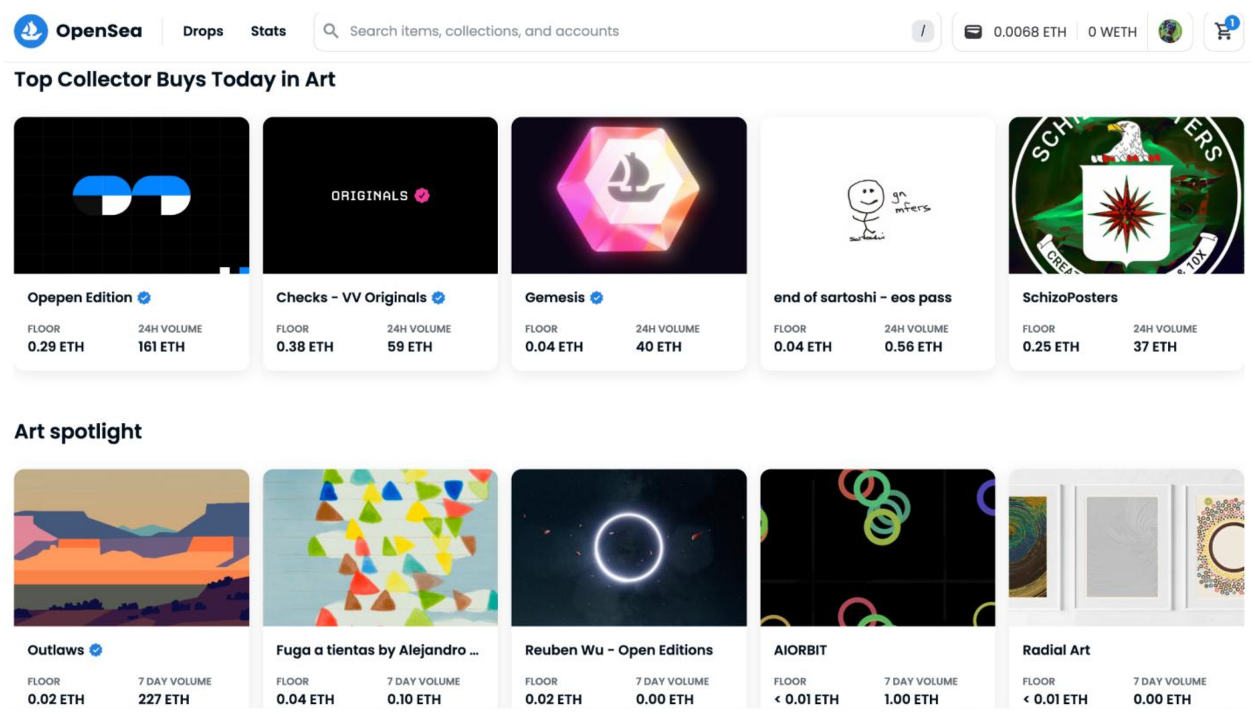
For this project, we chose to use the Polygon blockchain because it is better for the environment than Ethereal, Once we had generated 100 images, it was pretty straightforward to upload and mint them as NFTs. Firstly, the Polygon network can handle more transactions per second (TPS) than Ethereum, which reduces network congestion and, in turn, traffic. Second, transactions on Polygon generally have much lower gas fees compared to Ethereum, often just a fraction of a cent. This makes it more cost-effective for users and traders.

Popularity and user base:

Popular NFT marketplaces usually have a larger user base and more active trading, which can lead to better liquidity and a higher chance of finding a buyer or seller for your NFTs. Based on the data analytics from DUNE, there are 47.3% and 35.7% NFT traders active on Opensea and Blur since Apr 16, 2023.

Payment options:

Some marketplaces support multiple cryptocurrencies for trading NFTs, while others only accept a specific token. However, Opensea provides a way to trade NFTs with a bank card, which means users could buy a NFT with cash/ without the tedious process of exchange cryptocurrencies.



Top Collector Buys Today in Art

Collection	FLOOR	24H VOLUME
Opepen Edition	0.29 ETH	161 ETH
Checks - VV Originals	0.38 ETH	59 ETH
Gemesis	0.04 ETH	40 ETH
end of sartoshi - eos pass	0.04 ETH	0.56 ETH
SchizoPosters	0.25 ETH	37 ETH

Art spotlight

Collection	FLOOR	7 DAY VOLUME
Outlaws	0.02 ETH	227 ETH
Fuga a tientas by Alejandro ...	0.04 ETH	0.10 ETH
Reuben Wu - Open Editions	0.02 ETH	0.00 ETH
AIORBIT	< 0.01 ETH	1.00 ETH
Radial Art	< 0.01 ETH	0.00 ETH

Categories and niches:

Some marketplaces specialize in specific types of NFTs, such as art, virtual real estate, or gaming items. Opensea that cater to the ART category of the NFTs we want to trade.

Considering these factors, OpenSea can be a suitable choice for listing and trading artwork NFTs. It offers a large user base, diverse categories, and a user-friendly interface. However, it's essential to be aware of the gas fees associated with Ethereum transactions, though you can mitigate this by using Polygon-based NFTs on OpenSea.

3.Conclusions

We propose an NFT painting model for creating Monet style film character portraits from scratch. Our project is dedicated to face and StyleGAN painting. Through running multiple iterations of DLIP → GAN → CLIP, and the implantation of OpenAI API, the generation of image to image translation and text to image translation is realized, which supports the possibility of machine learning to produce digital art, and can be used for the learning and migration of other artistic portrait styles later.

However, in this project, on the one hand, due to the influence of the number of labels, the automatic classification accuracy of the model is not high enough. These problems can be solved through future large-scale machine training. On the other hand, due to our own machine learning and training techniques, we are still unable to directly produce possible NFT-GAN hybrid, which may have more value in the field of collectibles. This requires us to conceive and practice more practical solutions in the future.

Reference

Block, F. (2021). PAK's NFT Artwork 'The Merge'Sells for $91.8 Million. Retrieved from Barrons, PENTA: https://www. barrons. com/articles/paks-nft-artwork-the-merge-sells for-91-8-million-01638918205.

Gonsalves, R. (2021). GANscapes: Using AI to Create New Impressionist Paintings, Towards Data Science via Medium. https://towardsdatascience.com/ganscapes-using-ai-to-create-new-impressionist-paintings-d6af1cf94c56

Hufnagel, S., & King, C. (2023). Non-Fungible Tokens: Art and Crime in a Virtual World. Available at SSRN 4370145. http://dx.doi.org/10.2139/ssrn.4370145

Shahriar, S., & Al Roken, N. (2022). How can generative adversarial networks impact computer generated art? Insights from poetry to melody conversion. International Journal of Information Management Data Insights, 2(1), 100066. https://doi.org/10.1016/j.jjimei.2022.100066

Xue, A. (2021). End-to-end chinese landscape painting creation using generative adversarial networks. In Proceedings of the IEEE/CVF Winter conference on applications of computer vision (pp. 3863-3871).https://10.1109/WACV48630.2021.00391