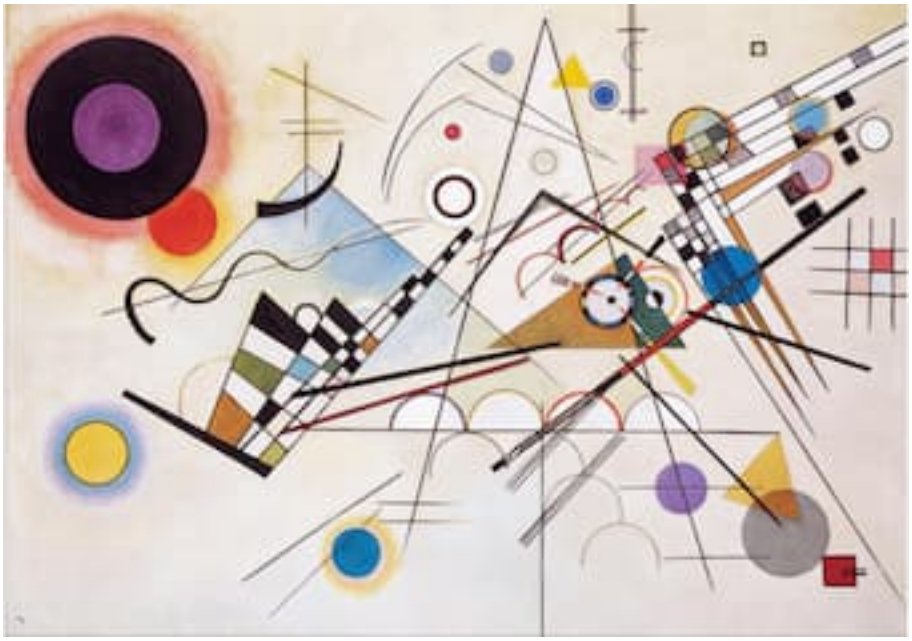
ART AND MACHINE LEARNING
CMU 2022 SPRING
FINAL PROJECT

TEXT TO PIX TO PITCH



Img :
https://interlude.hk/when-the-eye-meets-the-ear-how-are-intricate-music-concepts-represented-in-visual-arts/

PRAJWAL DESHKAR

DESCRIPTION

Text to Pix to Pitch is an experiment which aim to convert text and music into Visual and Musical art representation. It consists of two deep learning models first is GPT3 for Text to Image and CGAN for generating music which match the emotion of the art form.

Concept

I always wanted to explore relation between visual art and music, because visual art and music have numerous parallels because they share elements such as harmony, balance, rhythm, and repetition. Hence I thought of experimenting if these similarities in visual art can help machine learning algorithm explore relation between visual art and music.

Technique

For first part I have used various pertained model, mainly I used GPT3 algorithm to generate images from text. I used various online tools but mainly as well.
For second part that is PIX to PITCH, I used CGAN typically a PIX to PIX type which takes learns relation between two images. It is **a type of GAN that involves the conditional generation of images by a generator model**. A generative adversarial network (GAN) is a Machine Learning framework used to train generative models.

Process

First part that is TEXT to PIX was not challenging, because I used written model which were available online or the one which I trained in my first project so it was not a difficult part for me.
Second part that is generating Music which is in line with the kind of painting was a crucial part because professor asked me that how will your machine learning model learn that emotional relation. The emotional relation between visual art and music was where I figure it out from one paper can be done through CGANS.
I found data site from so and so library which has painting paired with emotionally similar music but it was in form of MIDI file. Really needed to pre-process my data properly and strategise thing in a way that I feed my machine learning model a data which can provide maximum information to it. Really I thought of pairing my paintings with real form of media file which consist of amplitude and fluctuations but I thought that information about frequency was missing and which is a very vital information to generate music so I skipped using a view from and decided to use a spectrogram which has more data because it shows saturation frequency as well as rhythm of the music so I decided to use spectrogram as a second image to pair with the painting.
After pre-processing my data I had pairs of image ready where is spectrogram and painting on which I trained my C GAN. I was expecting it to in that relation between colours in painting and frequency because darker the frequency of darker the colour that side and that's how that

correlation can be octane between two images and surprisingly it learnt that and I got the output which is mentioned below.

Validation

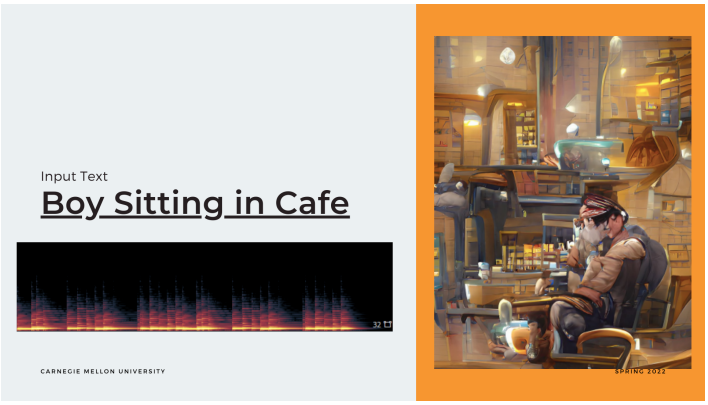
Mona Lisa
The tritone sound is of anticipation the look on her face is somber so it doesn't look positive it looks almost like she's ready to hear bad news and hoping she doesn't

Boy sitting in cafe
It appears and sounds like he's anticipating for his friends to show up and for the to go and play a game of some sort

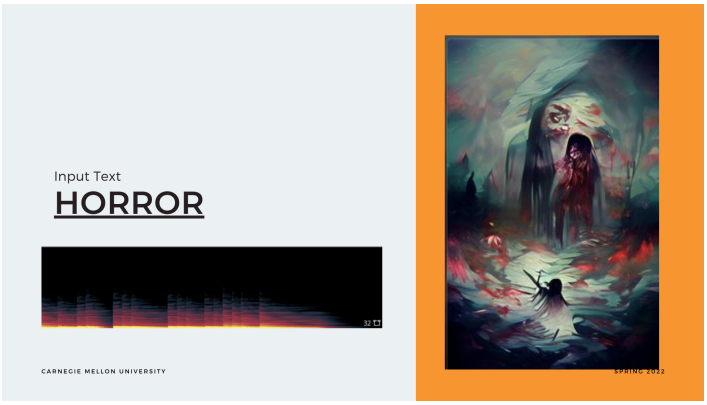
Horror
This appears and sounds like a rave invitation that is a circuit party that has a theme to gothic enthusiasts

RESULT

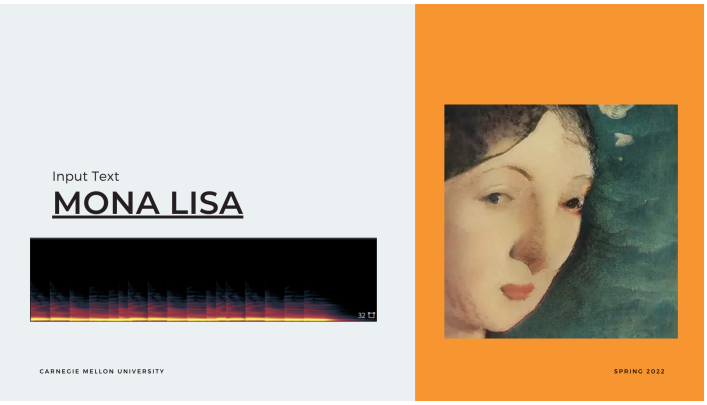
BOY SITTING IN CAFE



HORROR



MONA LISA



CODE

DRIVE FOLDER :

https://drive.google.com/drive/folders/190IPXjsblJFk-GIugtfDEfPKdCrAnsGZusp=sharing

RAW CODE:

GENERATOR MODEL :

```
import torch
import torch.nn as nn


class Block(nn.Module):
    def __init__(self, in_channels, out_channels, down=True, act="relu",
use_dropout=False):
        super(Block, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 4, 2, 1, bias=False,
padding_mode="reflect")
            if down
            else nn.ConvTranspose2d(in_channels, out_channels, 4, 2, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU() if act == "relu" else nn.LeakyReLU(0.2),
        )

        self.use_dropout = use_dropout
        self.dropout = nn.Dropout(0.5)
        self.down = down
```

```
    def forward(self, x):
        x = self.conv(x)
        return self.dropout(x) if self.use_dropout else x


class Generator(nn.Module):
    def __init__(self, in_channels=3, features=64):
        super().__init__()
        self.initial_down = nn.Sequential(
            nn.Conv2d(in_channels, features, 4, 2, 1, padding_mode="reflect"),
            nn.LeakyReLU(0.2),
        )
        self.down1 = Block(features, features * 2, down=True, act="leaky",
use_dropout=False)
        self.down2 = Block(
            features * 2, features * 4, down=True, act="leaky", use_dropout=False
        )
        self.down3 = Block(
            features * 4, features * 8, down=True, act="leaky", use_dropout=False
        )
        self.down4 = Block(
            features * 8, features * 8, down=True, act="leaky", use_dropout=False
        )
        self.down5 = Block(
            features * 8, features * 8, down=True, act="leaky", use_dropout=False
        )
        self.down6 = Block(
            features * 8, features * 8, down=True, act="leaky", use_dropout=False
        )
        self.bottleneck = nn.Sequential(
            nn.Conv2d(features * 8, features * 8, 4, 2, 1), nn.ReLU()
        )

        self.up1 = Block(features * 8, features * 8, down=False, act="relu",
use_dropout=True)
        self.up2 = Block(
            features * 8 * 2, features * 8, down=False, act="relu", use_dropout=True
        )
        self.up3 = Block(
            features * 8 * 2, features * 8, down=False, act="relu", use_dropout=True
        )
        self.up4 = Block(
            features * 8 * 2, features * 8, down=False, act="relu", use_dropout=False
        )
```

```
        self.up5 = Block(
            features * 8 * 2, features * 4, down=False, act="relu", use_dropout=False
        )
        self.up6 = Block(
            features * 4 * 2, features * 2, down=False, act="relu", use_dropout=False
        )
        self.up7 = Block(features * 2 * 2, features, down=False, act="relu",
use_dropout=False)
        self.final_up = nn.Sequential(
            nn.ConvTranspose2d(features * 2, in_channels, kernel_size=4, stride=2,
padding=1),
            nn.Tanh(),
        )

    def forward(self, x):
        d1 = self.initial_down(x)
        d2 = self.down1(d1)
        d3 = self.down2(d2)
        d4 = self.down3(d3)
        d5 = self.down4(d4)
        d6 = self.down5(d5)
        d7 = self.down6(d6)
        bottleneck = self.bottleneck(d7)
        up1 = self.up1(bottleneck)
        up2 = self.up2(torch.cat([up1, d7], 1))
        up3 = self.up3(torch.cat([up2, d6], 1))
        up4 = self.up4(torch.cat([up3, d5], 1))
        up5 = self.up5(torch.cat([up4, d4], 1))
        up6 = self.up6(torch.cat([up5, d3], 1))
        up7 = self.up7(torch.cat([up6, d2], 1))
        return self.final_up(torch.cat([up7, d1], 1))


def test():
    x = torch.randn((1, 3, 256, 256))
    model = Generator(in_channels=3, features=64)
    preds = model(x)
    print(preds.shape)


if __name__ == "__main__":
    test()
```

DISCRIMINATOR MODEL :

```
import torch
import torch.nn as nn


class CNNBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride):
        super(CNNBlock, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(
                in_channels, out_channels, 4, stride, 1, bias=False, padding_mode="reflect"
            ),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(0.2),
        )

    def forward(self, x):
        return self.conv(x)


class Discriminator(nn.Module):
    def __init__(self, in_channels=3, features=[64, 128, 256, 512]):
        super().__init__()
        self.initial = nn.Sequential(
            nn.Conv2d(
                in_channels * 2,
                features[0],
                kernel_size=4,
                stride=2,
                padding=1,
                padding_mode="reflect",
            ),
            nn.LeakyReLU(0.2),
        )

        layers = []
        in_channels = features[0]
        for feature in features[1:]:
            layers.append(
                CNNBlock(in_channels, feature, stride=1 if feature == features[-1] else 2),
            )
            in_channels = feature
```

```
        layers.append(
            nn.Conv2d(
                in_channels, 1, kernel_size=4, stride=1, padding=1, padding_mode="reflect"
            ),
        )

        self.model = nn.Sequential(*layers)

    def forward(self, x, y):
        x = torch.cat([x, y], dim=1)
        x = self.initial(x)
        x = self.model(x)
        return x


def test():
    x = torch.randn((1, 3, 256, 256))
    y = torch.randn((1, 3, 256, 256))
    model = Discriminator(in_channels=3)
    preds = model(x, y)
    print(model)
    print(preds.shape)


if __name__ == "__main__":
    test()
```

TRAIN :

```
import torch
from utils import save_checkpoint, load_checkpoint, save_some_examples
import torch.nn as nn
import torch.optim as optim
import config
from dataset import MapDataset
from generator_model import Generator
from discriminator_model import Discriminator
from torch.utils.data import DataLoader
from tqdm import tqdm
```

```python
from torchvision.utils import save_image

torch.backends.cudnn.benchmark = True


def train_fn(
    disc, gen, loader, opt_disc, opt_gen, l1_loss, bce, g_scaler, d_scaler,
):
    loop = tqdm(loader, leave=True)

    for idx, (x, y) in enumerate(loop):
        x = x.to(config.DEVICE)
        y = y.to(config.DEVICE)

        # Train Discriminator
        with torch.cuda.amp.autocast():
            y_fake = gen(x)
            D_real = disc(x, y)
            D_real_loss = bce(D_real, torch.ones_like(D_real))
            D_fake = disc(x, y_fake.detach())
            D_fake_loss = bce(D_fake, torch.zeros_like(D_fake))
            D_loss = (D_real_loss + D_fake_loss) / 2

        disc.zero_grad()
        d_scaler.scale(D_loss).backward()
        d_scaler.step(opt_disc)
        d_scaler.update()

        # Train generator
        with torch.cuda.amp.autocast():
            D_fake = disc(x, y_fake)
            G_fake_loss = bce(D_fake, torch.ones_like(D_fake))
            L1 = l1_loss(y_fake, y) * config.L1_LAMBDA
            G_loss = G_fake_loss + L1

        opt_gen.zero_grad()
        g_scaler.scale(G_loss).backward()
        g_scaler.step(opt_gen)
        g_scaler.update()

        if idx % 10 == 0:
            loop.set_postfix(
                D_real=torch.sigmoid(D_real).mean().item(),
                D_fake=torch.sigmoid(D_fake).mean().item(),
```

```
        )


def main():
    disc = Discriminator(in_channels=3).to(config.DEVICE)
    gen = Generator(in_channels=3, features=64).to(config.DEVICE)
    opt_disc = optim.Adam(disc.parameters(), lr=config.LEARNING_RATE, betas=(0.5,
0.999),)
    opt_gen = optim.Adam(gen.parameters(), lr=config.LEARNING_RATE, betas=(0.5,
0.999))
    BCE = nn.BCEWithLogitsLoss()
    L1_LOSS = nn.L1Loss()

    if config.LOAD_MODEL:
        load_checkpoint(
            config.CHECKPOINT_GEN, gen, opt_gen, config.LEARNING_RATE,
        )
        load_checkpoint(
            config.CHECKPOINT_DISC, disc, opt_disc, config.LEARNING_RATE,
        )

    train_dataset = MapDataset(root_dir=config.TRAIN_DIR)
    train_loader = DataLoader(
        train_dataset,
        batch_size=config.BATCH_SIZE,
        shuffle=True,
        num_workers=config.NUM_WORKERS,
    )
    g_scaler = torch.cuda.amp.GradScaler()
    d_scaler = torch.cuda.amp.GradScaler()
    val_dataset = MapDataset(root_dir=config.VAL_DIR)
    val_loader = DataLoader(val_dataset, batch_size=1, shuffle=False)

    for epoch in range(config.NUM_EPOCHS):
        train_fn(
            disc, gen, train_loader, opt_disc, opt_gen, L1_LOSS, BCE, g_scaler, d_scaler,
        )

        if config.SAVE_MODEL and epoch % 5 == 0:
            save_checkpoint(gen, opt_gen, filename=config.CHECKPOINT_GEN)
            save_checkpoint(disc, opt_disc, filename=config.CHECKPOINT_DISC)

        save_some_examples(gen, val_loader, epoch, folder="evaluation")
```

```
if __name__ == "__main__":
    main()
```

REFERENCES

1. PIX to PIX

2. https://www.google.com/search?q=wombo&oq=wombo&aqs=chrome..69i57j69i59i433i512j0i433i512l2j69i59j69i60l2j69i61.4807j1j9&sourceid=chrome&ie=UTF-8#:~:text=WOMBO%20Dream%20%2D%20AI,www.wombo.art

3. https://oa.upm.es/63694/1/TFM_ELENA_RIVAS_RUZAFA.pdf

4. https://arxiv.org/abs/1611.07004