Generating Event Posters with GANs

BY - ANSH SHARMA

By accessing the random poster website to acquire the training data - event_posters

```
import requests
from bs4 import BeautifulSoup
import os

# Create a folder for storing the posters
os.makedirs("event_posters", exist_ok=True)

url = 'https://venngage.com/templates/posters'  # Example URL
response = requests.get(url)
soup = BeautifulSoup(response.content, 'html.parser')

# Find poster images and store them
posters = soup.find_all('img')  # Find all images
for idx, poster in enumerate(posters):
    img_url = poster['src']
    img_data = requests.get(img_url).content
    with open(f"event_posters/poster_{idx}.jpg", 'wb') as f:
        f.write(img_data)

```

Preprocess the data , and removing **outliers**

```
import os
from PIL import Image
import numpy as np

# Resize and normalize images
def preprocess_images(folder_path):
    image_size = (256, 256)  # Resize to 256x256
    processed_images = []
    for file_name in os.listdir(folder_path):
        if file_name.endswith(".jpg"):
            file_path = os.path.join(folder_path, file_name)
            try:
                img = Image.open(file_path).convert("RGB")
                img = img.resize(image_size)
                img_array = np.array(img) / 255.0  # Normalize to [0, 1]
                processed_images.append(img_array)
            except Exception as e:
                print(f"Error processing {file_name}: {e}")
    return np.array(processed_images)

data = preprocess_images("event_posters")  # Preprocess your event posters

```

```
Error processing poster_0.jpg: cannot identify image file '/content/event_posters/post
Error processing poster_56.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_35.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_41.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_63.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_62.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_59.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_54.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_64.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_65.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_44.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_27.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_18.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_60.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_66.jpg: cannot identify image file '/content/event_posters/pos
Error processing poster_61.jpg: cannot identify image file '/content/event_posters/pos
```

Apply CNN and discriminator to distinguish between fake and real **images**

```
import torch
import torch.nn as nn
import torch.optim as optim

# Generator model (cGAN)
class Generator(nn.Module):
    def __init__(self, latent_dim, label_dim):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(latent_dim + label_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, 3 * 256 * 256),  # 3 channels, 256x256 image
            nn.Tanh()  # Normalize to [-1, 1]
        )

    def forward(self, z, label):
        # Concatenate the noise vector with the label (event type)
        inputs = torch.cat([z, label], dim=1)
        return self.model(inputs).view(-1, 3, 256, 256)  # Reshape to image format

```

```
# Discriminator model (cGAN)
class Discriminator(nn.Module):
    def __init__(self, label_dim):
        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(3 * 256 * 256 + label_dim, 1024),
            nn.LeakyReLU(0.2),
            nn.Linear(1024, 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 1),
            nn.Sigmoid()
        )

    def forward(self, x, label):
        # Concatenate image with the label
        inputs = torch.cat([x.view(x.size(0), -1), label], dim=1)  # Flatten image
        return self.model(inputs)

```

McLaren Pub, 598 Main St. W
TRIVIA NIGHT
Every Thursday, game begins at 8pm
Grand prizes for the top 3 winning teams!
$10 per person (Teams of 4 members)
Trivia themes include pop culture, sports, history & more!
www.McLarenPub.com/TriviaNights

Training Generatial Adversial Network with 100 **epochs**

```
from torch.utils.data import DataLoader, Dataset

# Example metadata (replace with actual data)
metadata = ["music", "art", "sports", "conference", "party"]
label_map = {label: idx for idx, label in enumerate(metadata)}  # Map labels to integers

# Label encoding (dummy example)
labels = [label_map["music"]] * 50 + [label_map["art"]] * 50  # Repeat for example purpo

# Convert data and labels into tensors
import torch
data_tensor = torch.tensor(data, dtype=torch.float32)
labels_tensor = torch.tensor(labels, dtype=torch.long)

class PosterDataset(Dataset):
    def __init__(self, images, labels):
        self.images = images
        self.labels = labels

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

    def __getitem__(self, idx):
        # Convert label to one-hot
        label = torch.zeros(len(metadata))
        label[self.labels[idx]] = 1
        return self.images[idx], label

# Create data loader
dataset = PosterDataset(data_tensor, labels_tensor)
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)

# Hyperparameters
latent_dim = 100  # Latent dimension for noise vector
label_dim = len(metadata)  # Number of categories
epochs = 100
lr = 0.0002  # Learning rate

# Initialize models
generator = Generator(latent_dim, label_dim).cpu()
discriminator = Discriminator(label_dim).cpu()

# Loss function and optimizers
criterion = nn.BCELoss()
optim_G = optim.Adam(generator.parameters(), lr=lr, betas=(0.5, 0.999))
optim_D = optim.Adam(discriminator.parameters(), lr=lr, betas=(0.5, 0.999))

# Training loop
for epoch in range(epochs):
    for real_imgs, labels in dataloader:
        # Move to CPU
        real_imgs, labels = real_imgs.cpu(), labels.cpu()

        # Create labels for real and fake images
        real_labels = torch.ones(real_imgs.size(0), 1).cpu()  # Real labels
        fake_labels = torch.zeros(real_imgs.size(0), 1).cpu()  # Fake labels

        # Train the discriminator
        optim_D.zero_grad()
        output_real = discriminator(real_imgs, labels)
        d_loss_real = criterion(output_real, real_labels)

        z = torch.randn(real_imgs.size(0), latent_dim).cpu()  # Latent vector
        fake_imgs = generator(z, labels)
        output_fake = discriminator(fake_imgs.detach(), labels)
        d_loss_fake = criterion(output_fake, fake_labels)

        d_loss = d_loss_real + d_loss_fake
        d_loss.backward()
        optim_D.step()

        # Train the generator
        optim_G.zero_grad()
        output_fake = discriminator(fake_imgs, labels)
        g_loss = criterion(output_fake, real_labels)
        g_loss.backward()
        optim_G.step()

    print(f"Epoch [{epoch+1}/{epochs}] | D Loss: {d_loss.item():.4f} | G Loss: {g_loss.i

```

```
Epoch [1/100] | D Loss: 1.2022 | G Loss: 0.6607
Epoch [2/100] | D Loss: 0.7160 | G Loss: 2.1262
Epoch [3/100] | D Loss: 16.6243 | G Loss: 0.4245
Epoch [4/100] | D Loss: 3.5644 | G Loss: 0.0242
Epoch [5/100] | D Loss: 4.5099 | G Loss: 0.0236
Epoch [6/100] | D Loss: 3.9674 | G Loss: 0.0625
Epoch [7/100] | D Loss: 2.6373 | G Loss: 0.5105
Epoch [8/100] | D Loss: 12.7443 | G Loss: 0.0015
Epoch [9/100] | D Loss: 10.3043 | G Loss: 0.0001
Epoch [10/100] | D Loss: 8.3684 | G Loss: 0.0015
Epoch [11/100] | D Loss: 5.2373 | G Loss: 0.0423
Epoch [12/100] | D Loss: 1.5311 | G Loss: 2.0688
Epoch [13/100] | D Loss: 5.5989 | G Loss: 0.0007
Epoch [14/100] | D Loss: 7.5570 | G Loss: 0.0010
Epoch [15/100] | D Loss: 6.7236 | G Loss: 0.0031
Epoch [16/100] | D Loss: 5.1933 | G Loss: 0.0133
Epoch [17/100] | D Loss: 3.8888 | G Loss: 0.0660
Epoch [18/100] | D Loss: 1.8202 | G Loss: 0.8415
Epoch [19/100] | D Loss: 17.2876 | G Loss: 0.0090
Epoch [20/100] | D Loss: 6.3104 | G Loss: 0.0014
Epoch [21/100] | D Loss: 6.5118 | G Loss: 0.0027
Epoch [22/100] | D Loss: 5.1867 | G Loss: 0.0112
Epoch [23/100] | D Loss: 4.4505 | G Loss: 0.0272
Epoch [24/100] | D Loss: 3.1968 | G Loss: 0.1252
Epoch [25/100] | D Loss: 1.3525 | G Loss: 1.3034
Epoch [26/100] | D Loss: 7.1935 | G Loss: 0.0001
Epoch [27/100] | D Loss: 9.6685 | G Loss: 0.0002
Epoch [28/100] | D Loss: 8.7002 | G Loss: 0.0007
Epoch [29/100] | D Loss: 6.2303 | G Loss: 0.0079
Epoch [30/100] | D Loss: 3.6935 | G Loss: 0.2256
Epoch [31/100] | D Loss: 15.6058 | G Loss: 0.0021
Epoch [32/100] | D Loss: 7.2894 | G Loss: 0.0005
Epoch [33/100] | D Loss: 7.7352 | G Loss: 0.0010
Epoch [34/100] | D Loss: 6.3979 | G Loss: 0.0044
Epoch [35/100] | D Loss: 5.1364 | G Loss: 0.0151
Epoch [36/100] | D Loss: 3.6581 | G Loss: 0.0720
Epoch [37/100] | D Loss: 1.9869 | G Loss: 0.7718
Epoch [38/100] | D Loss: 11.6797 | G Loss: 0.0023
Epoch [39/100] | D Loss: 8.7417 | G Loss: 0.0001
Epoch [40/100] | D Loss: 8.9033 | G Loss: 0.0004
Epoch [41/100] | D Loss: 7.3275 | G Loss: 0.0017
Epoch [42/100] | D Loss: 5.8970 | G Loss: 0.0068
Epoch [43/100] | D Loss: 4.4535 | G Loss: 0.0291
Epoch [44/100] | D Loss: 3.0995 | G Loss: 0.1416
Epoch [45/100] | D Loss: 1.2562 | G Loss: 1.1804
Epoch [46/100] | D Loss: 6.5732 | G Loss: 0.0002
Epoch [47/100] | D Loss: 9.6771 | G Loss: 0.0001
Epoch [48/100] | D Loss: 8.1548 | G Loss: 0.0010
Epoch [49/100] | D Loss: 6.0668 | G Loss: 0.0087
Epoch [50/100] | D Loss: 3.8812 | G Loss: 0.0857
Epoch [51/100] | D Loss: 1.3710 | G Loss: 1.2912
Epoch [52/100] | D Loss: 5.7531 | G Loss: 0.0004
Epoch [53/100] | D Loss: 7.9875 | G Loss: 0.0006
Epoch [54/100] | D Loss: 7.0214 | G Loss: 0.0025
Epoch [55/100] | D Loss: 5.6822 | G Loss: 0.0103
Epoch [56/100] | D Loss: 3.7879 | G Loss: 0.0630
Epoch [57/100] | D Loss: 2.1705 | G Loss: 0.4051
Epoch [58/100] | D Loss: 5.1336 | G Loss: 0.0042
```

Generating 20 Posters of Various **Events**

```
import os
import torch
from torchvision.utils import save_image

# Directories for saving generated images
generated_dir = './generated_posters'
os.makedirs(generated_dir, exist_ok=True)

# Load the trained Generator
latent_dim = 100
label_dim = 10
num_images = 20
generator = Generator(latent_dim, label_dim).cpu()
torch.save(generator.state_dict(), 'generator_checkpoint.pth')
generator.load_state_dict(torch.load('generator_checkpoint.pth', map_location='cpu'))
generator.eval()

# Generate and save synthetic images
for i in range(num_images):
    with torch.no_grad():
        z = torch.randn(1, latent_dim)
        label = torch.zeros(1, label_dim)
        label[0, 0] = 1  # Example: Setting label for category "Music"
        fake_img = generator(z, label)
        fake_img = fake_img.squeeze(0)
        save_image(fake_img, os.path.join(generated_dir, f'poster_{i+1}.png'), normaliz

print(f"Generated {num_images} images in the directory: {generated_dir}")

```

```
<ipython-input-27-673ec93b5f45>:15: FutureWarning: You are using `torch.load` with `we
  generator.load_state_dict(torch.load('generator_checkpoint.pth', map_location='cpu')
Generated 20 images in the directory: ./generated_posters
```

CHILDREN AND FAMILIES IN UKRAINE NEED YOUR SUPPORT
DONATE TODAY
PRESENTED BY:
EVERGREEN ASSOCIATION
GLOBAL WARMING AWARENESS
September 5, 2023
10AM TO 4 PM
Energy Centre
4248 Benson Park Drive,
Star City , Minnesota, 55394
www.evergreenassociation.org
TICKET
$35
GOLF
Tournament
Employer Reopening Guide:
Safety Standards For Workplaces
SOCIAL DISTANCING
HYGIENE PROTOCOLS
STAFF & OPERATIONS
VISUALS & GUIDES
Join us in celebrating
International Women's Day
March 8th
11am-4pm
S.T.E.A.M
Activities
Science Labs
Museum
SCIENCELABS
TECH
ROUND
2023
THE LATEST IN BUSINESS GROWTH TECHNOLOGY
3-Day Registration:
$500.00 per attendee
April 6 to April 9, 2023
9:00am to 7:00pm daily
The Elliot House Hotel
216 Gladwell Boulevard
Crested Butte, Colorado
Register online at:
eventlite.com

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

# Create a folder for storing the posters
os.makedirs("event_posters", exist_ok=True)

url = 'https://venngage.com/templates/posters'  # Example URL
response = requests.get(url)
soup = BeautifulSoup(response.content, 'html.parser')

# Find poster images and store them, ensuring they are valid image formats
posters = soup.find_all('img')  # Find all images
```

```python
for idx, poster in enumerate(posters):
    img_url = poster['src']
    try:
        # Check if the image URL is valid and has a recognized image extension
        if img_url and any(img_url.lower().endswith(ext) for ext in ['.jpg', '.jpeg', '
            img_data = requests.get(img_url).content
            with open(f"event_posters/poster_{idx}.jpg", 'wb') as f:
                f.write(img_data)
        else:
            print(f"Skipping invalid image URL: {img_url}")
    except Exception as e:
        print(f"Error saving image {idx}: {e}")
```

Show hidden output

Calculation of FID

```python
import os
from PIL import Image
import numpy as np
from pytorch_fid import fid_score

# Resize real event poster images to 256x256
real_posters_dir = "event_posters"
generated_dir = './generated_posters'  # Directory containing generated posters

def resize_posters(folder_path, target_size=(256, 256)):
    for filename in os.listdir(folder_path):
        if filename.endswith((".jpg", ".jpeg", ".png")):
            filepath = os.path.join(folder_path, filename)
            try:
                img = Image.open(filepath).convert("RGB")
                img = img.resize(target_size)
                img.save(filepath)  # Overwrite the original image
            except Exception as e:
                print(f"Error resizing {filename}: {e}")

# Resize posters in both directories
resize_posters(real_posters_dir)  # Resize real posters

# Calculate FID score after ensuring all images have the same size
fid_value = fid_score.calculate_fid_given_paths([real_posters_dir, generated_dir], batc
print(f"FID Score: {fid_value}")
```

```
100%|██████████| 2/2 [00:19<00:00,  9.84s/it]
Warning: batch size is bigger than the data size. Setting batch size to data size
100%|██████████| 1/1 [00:08<00:00,  8.72s/it]
FID Score: 444.65861934867
```

DEPLOYEMENT USING FAST **API**

```
from flask import Flask, request, jsonify
import torch

app = Flask(__name__)

# Load the trained model (ensure the generator is loaded)
generator = Generator(latent_dim, label_dim).cpu()
generator.load_state_dict(torch.load('generator.pth'))  # Load the pre-trained generatc

@app.route('/generate', methods=['POST'])
def generate_poster():
    data = request.json
    event_type = data.get('event_type', 'music')
    theme = data.get('theme', 'festival')

    # Convert event metadata to one-hot encoded tensor
    label = torch.zeros(len(metadata))
    label[label_map[event_type]] = 1
    label[label_map[theme]] = 1

    # Generate the poster
    z = torch.randn(1, latent_dim).cpu()  # Latent vector
    generated_image = generator(z, label)

    # Convert tensor to image and return
    generated_image = generated_image.squeeze().detach().cpu().numpy()
    return jsonify({'message': 'Poster generated!', 'image': generated_image.tolist()})

if __name__ == '__main__':
    app.run(debug=True)

```