

Text to Image Synthesis using DCGAN-BERTs

Quoc-Thai Nguyen, Quang-Hien Ho và Quang-Vinh Dinh

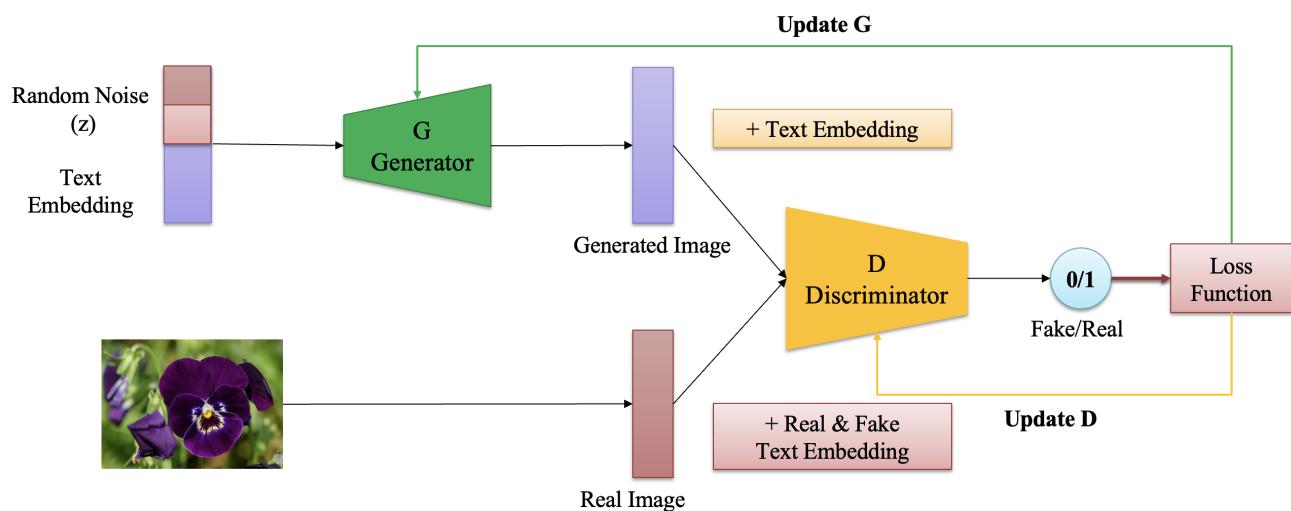
Ngày 8 tháng 3 năm 2025

Phần 1. Giới thiệu

The flower has petals that
are bright pinkish purple
with white stigma



Hình 1: Ví dụ sinh văn bản thành hình ảnh sử dụng mô hình DCGAN.



Hình 2: Mô hình DCGAN sinh văn bản thành hình ảnh.

Sinh hình ảnh từ văn bản hay tổng hợp văn bản thành hình ảnh (Text to Image) là bài toán ngày càng được ứng dụng mạnh mẽ trong lĩnh vực trí tuệ nhân tạo. Các mô hình được huấn

luyện với đầu vào là đoạn văn bản và đầu ra là hình ảnh mô tả hoặc chứa các đối tượng được mô tả trong đoạn văn bản. Ví dụ về tổng hợp hình ảnh từ văn bản được mô tả trong Hình 1.

Hiện nay, có nhiều mô hình có thể xây dựng và giải quyết bài toán này có thể kể đến như GANs, Diffusion Models,... Ở trong phần này chúng ta sẽ sử dụng mô hình DCGAN (Deep Convolution Generative Adversarial Networks) để huấn luyện mô hình giải quyết bài toán này.

Mô hình DCGAN gồm 2 mạng là Generator và Discriminator. Trong đó:

1. Generator: Bao gồm các lớp mạng CNN nhận đầu vào là Noise (z) và Text Embedding (e - Biểu diễn của đoạn văn bản đầu vào). Đầu ra của khối Generator là ma trận ảnh được sinh ra $G(z, e)$.
2. Discriminator: Bao gồm các lớp mạng CNN nhận đầu vào là biểu diễn của ảnh (ảnh thật hoặc ảnh fake được sinh ra từ khối Generator) sau đó kết hợp với Text Embedding để dự đoán ảnh thật hay ảnh fake.

Phần 2. Text To Image Synthesis using DCGAN-BERTs

Trong phần này chúng ta sẽ xây dựng và huấn luyện mô hình DCGAN trên bộ dữ liệu flowers. Bộ dữ liệu flowers có thể được tải về [tại đây](#).

Các bước để huấn luyện mô hình bao gồm:

- (a) Dataset: Tải bộ dữ liệu Flower.
- (b) Caption Tokenizer: Sử dụng mô hình BERT sử dụng sentence-transformers để biểu diễn văn bản thành vector. Với mỗi văn bản sẽ được biểu diễn thành một vector tương ứng.
- (c) Preprocessing: Xây dựng dữ liệu I/O cho mô hình.
- (d) Model: Xây dựng mô hình Generator và Discriminator.
- (e) Training: Huấn luyện mô hình.
- (f) Deployment: Triển khai ứng dụng sử dụng Streamlit.

1. Dataset

Thực thi đoạn code sau để tải về bộ dữ liệu, sau đó giải nén thu được 2 thư mục "flowers" - chứa các file .txt chứa các đoạn văn bản mô tả hình ảnh, trong đó mỗi hình ảnh sẽ lấy một mô tả đầu tiên để huấn luyện mô hình và "images" - chứa các hình ảnh.

```

1 # Download dataset
2 !gdown 1JJjMiNieTz7xYs6UeVqd02M3DW4fnEfU
3 !unzip cvpr2016_flowers.zip
4
5 # Load captions
6 import os
7
8 def loadCaptions(captions_folder, image_folder):
9     captions = {}
10    image_files = os.listdir(image_folder)
11
12    for image_file in image_files:
13
14        image_name = image_file.split(".")[0]
15        caption_file = os.path.join(captions_folder, image_name + ".txt")
16        with open(caption_file, "r") as f:
17            caption = f.readlines()[0].strip()
18            if image_name not in captions:
19                captions[image_name] = caption
20
21    return captions
22
23 captions_folder = "./cvpr2016_flowers/captions"
24 image_folder = "./cvpr2016_flowers/images"
25
26 captions = loadCaptions(captions_folder, image_folder)
27 captions

```

2. Caption Encoder

Trong phần này, chúng ta sử dụng mô hình BERTs để biểu diễn mỗi câu mô tả thành một vector có kích thước 768 chiều.

```

1 import torch
2 import numpy as np
3 from sentence_transformers import SentenceTransformer
4
5 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
6 bert_model = SentenceTransformer("all-mpnet-base-v2").to(device)
7
8 def encode_captions(captions):
9     encoded_captions = {}
10    for image_name in captions.keys():
11        caption = captions[image_name]
12        encoded_captions[image_name] = {
13            "embed": torch.tensor(bert_model.encode(caption)),
14            "text": caption
15        }
16    return encoded_captions
17 encoded_captions = encode_captions(captions)

```

3. Preprocessing

Thực thi đoạn code sau đây để chuẩn bị I/O cho mô hình:

```

1 from PIL import Image
2 from torch.utils.data import Dataset
3
4 class FlowerDataset(Dataset):
5     def __init__(self, img_dir, captions, transform=None):
6         self.img_dir = img_dir
7         self.transform = transform
8         # Load captions
9         self.captions = captions
10        self.img_names = list(self.captions.keys())
11
12    def __len__(self):
13        return len(self.img_names)
14
15    def __getitem__(self, idx):
16        img_name = self.img_names[idx]
17        img_path = os.path.join(self.img_dir, img_name+".jpg")
18        image = Image.open(img_path).convert("RGB")
19        if self.transform:
20            image = self.transform(image)
21        encoded_caption = self.captions[img_name]["embed"]
22        caption = self.captions[img_name]["text"]
23        return {
24            "image": image,
25            "embed_caption": encoded_caption,
26            "text": caption
27        }

```

```

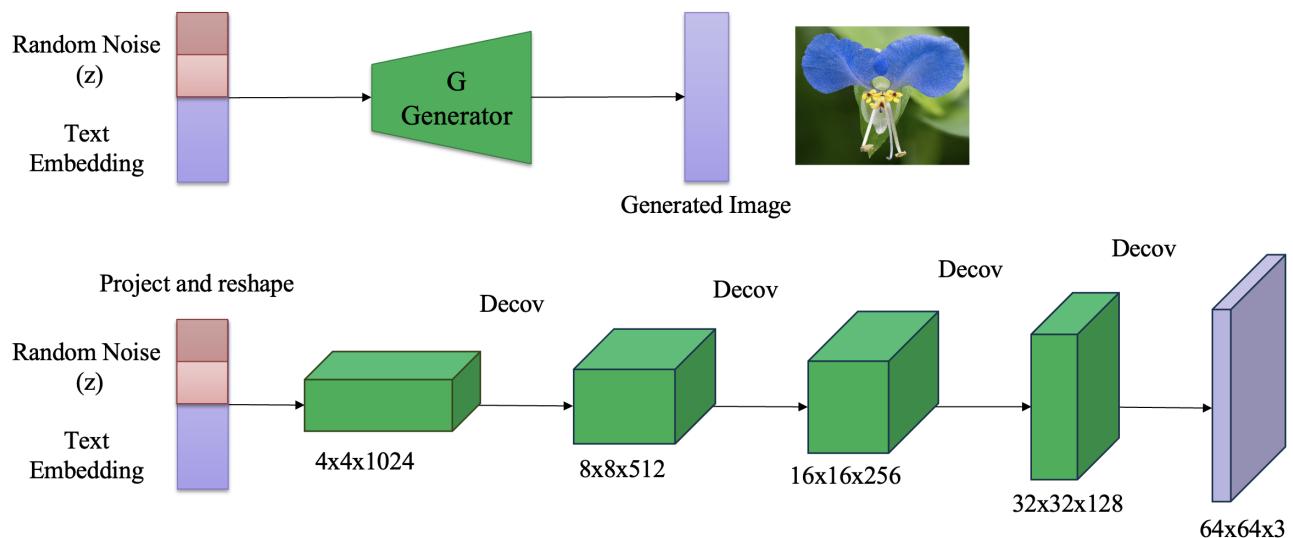
1 import torchvision.transforms as transforms
2
3 from torch.utils.data import DataLoader
4
5 IMG_SIZE = 128
6
7 transform = transforms.Compose([
8     transforms.Resize((IMG_SIZE, IMG_SIZE)),
9     transforms.ToTensor(),
10    transforms.Normalize([0.5], [0.5])
11 ])
12
13 ds = FlowerDataset(
14     img_dir="/content/cvpr2016_flowers/images",
15     captions=encoded_captions,
16     transform=transform
17 )
18
19 BATCH_SIZE = 1024
20 dataloader = DataLoader(ds, batch_size=BATCH_SIZE, shuffle=True)

```

3. Model

Trong phần này chúng ta xây dựng mô hình DCGAN để sinh văn bản thành hình ảnh bao gồm 2 mạng Generator và Discriminator:

3.1. Generator



Hình 3: Khối Generator trong DCGAN.

Khối Generator sinh ra ảnh từ văn bản đầu vào:

- Input: Nhận đầu vào là vector: Random Noise (z) có kích thước R, được nối với Vector Embedding (Vector e - biểu diễn cho cả đoạn văn bản đầu vào) có kích thước là D. Vì vậy, vector đầu vào mạng Generator là R + D.
- Output: Sau khi học mối quan hệ để sinh ảnh, giá trị đầu ra của Generator sẽ là biểu diễn các

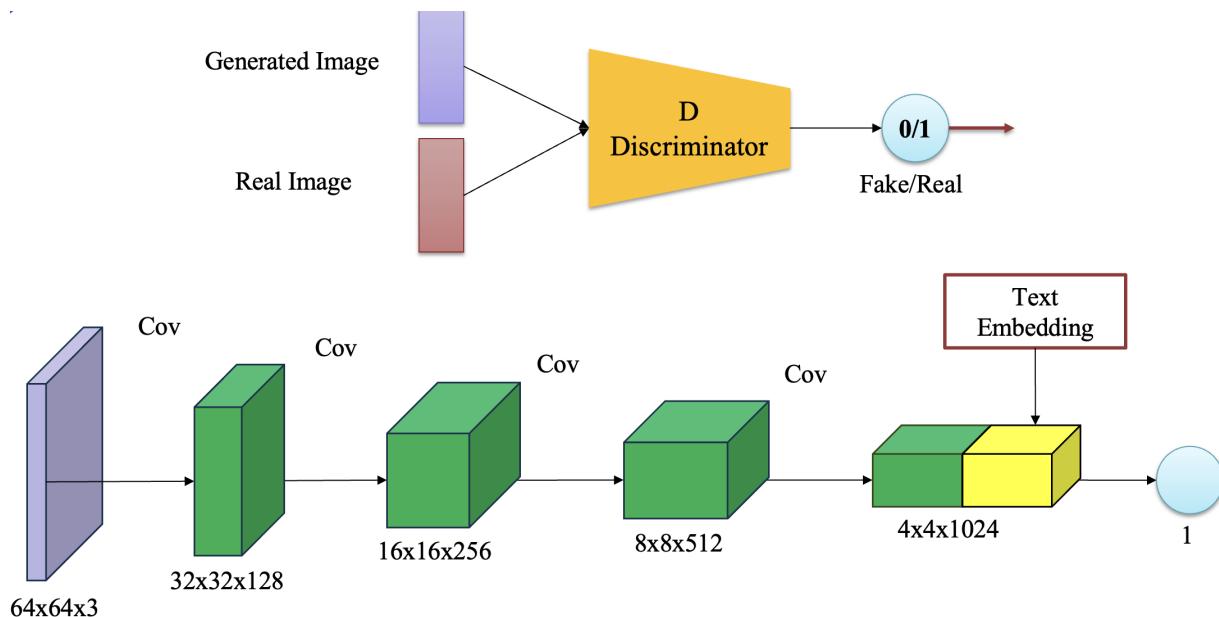
điểm ảnh dự đoán trong không gian 3 chiều có kích thước là Channel x Width x Height (Ví dụ, CxWxH - 3x64x64)

```
1 import torch.nn as nn
2
3 class Generator(nn.Module):
4
5     def __init__(self, noise_size, feature_size, num_channels, embedding_size,
6      reduced_dim_size):
7         super(Generator, self).__init__()
8         self.reduced_dim_size = reduced_dim_size
9         #768-->256
10        self.textEncoder = nn.Sequential(
11            nn.Linear(in_features = embedding_size, out_features =
12            reduced_dim_size),
13            nn.BatchNorm1d(num_features = reduced_dim_size),
14            nn.LeakyReLU(negative_slope = 0.2, inplace = True)
15        )
16
17        self.upsamplingBlock = nn.Sequential(
18            #256+100 --> 1024
19            nn.ConvTranspose2d(noise_size + reduced_dim_size, feature_size * 8,
20            4, 1, 0, bias = False),
21            nn.BatchNorm2d(feature_size * 8),
22            nn.LeakyReLU(negative_slope = 0.2, inplace = True),
23            # 1024 --> 512
24            nn.ConvTranspose2d(feature_size * 8, feature_size * 4, 4, 2, 1, bias
25 = False),
26            nn.BatchNorm2d(feature_size * 4),
27            nn.ReLU(True),
28            # 512 --> 256
29            nn.ConvTranspose2d(feature_size * 4, feature_size * 2, 4, 2, 1, bias=
30 False),
31            nn.BatchNorm2d(feature_size * 2),
32            nn.ReLU(True),
33            # 256 --> 128
34            nn.ConvTranspose2d(feature_size * 2, feature_size, 4, 2, 1, bias=
35 False),
36            nn.BatchNorm2d(feature_size),
37            nn.ReLU(True),
38            # 128 --> 3
39            nn.ConvTranspose2d(feature_size, num_channels, 4, 2, 1, bias=False),
40            nn.Tanh()
41
42    def forward(self, noise, text_embeddings):
43        encoded_text = self.textEncoder(text_embeddings)
44        concat_input = torch.cat([noise, encoded_text], dim = 1).unsqueeze(2).
45        unsqueeze(2)
46        output = self.upsamplingBlock(concat_input)
47        return output
```

3.2. Discriminator

Khối Discriminator dự đoán hình ảnh Fake/Real, bao gồm các lớp CNN với:

- (a) Input: Nhận đầu vào là ma trận điểm ảnh và Text Embedding. Ma trận điểm ảnh ($C \times H \times W$) sau khi qua các lớp CNN để học các đặc trưng của ảnh đầu vào sẽ được nối với vector embedding.
- (b) Output: Vector sau khi được nối sẽ được sử dụng để dự đoán 0(Fake) hoặc 1(Real).



Hình 4: Khối Discriminator trong DCGAN.

```
1 class Discriminator(nn.Module):
2
3     def __init__(self, num_channels, feature_size, embedding_size,
4                  reduced_dim_size):
5         super(Discriminator, self).__init__()
6         self.reduced_dim_size = reduced_dim_size
7         self.imageEncoder = nn.Sequential(
8             # 3 -> 128
9             nn.Conv2d(num_channels, feature_size, 4, 2, 1, bias = False),
10            nn.LeakyReLU(0.2, inplace = True),
11            # 128 -> 128
12            nn.Conv2d(feature_size, feature_size, 4, 2, 1, bias = False),
13            nn.LeakyReLU(0.2, inplace = True),
14            # 128 -> 256
15            nn.Conv2d(feature_size, feature_size * 2, 4, 2, 1, bias = False),
16            nn.BatchNorm2d(feature_size * 2),
17            nn.LeakyReLU(0.2, inplace = True),
18            # 256 -> 512
19            nn.Conv2d(feature_size * 2, feature_size * 4, 4, 2, 1, bias=False),
20            nn.BatchNorm2d(feature_size * 4),
21            nn.LeakyReLU(0.2, inplace=True),
22            # 512 -> 1024
23            nn.Conv2d(feature_size * 4, feature_size * 8, 4, 2, 1, bias=False),
24            nn.BatchNorm2d(feature_size * 8),
25            nn.LeakyReLU(0.2, inplace=True),
26        )
27        self.textEncoder = nn.Sequential(
28            nn.Linear(in_features=embedding_size, out_features=reduced_dim_size),
29            nn.BatchNorm1d(num_features=reduced_dim_size),
30            nn.LeakyReLU(negative_slope=0.2, inplace=True)
31        )
32        self.finalBlock = nn.Sequential(
33            nn.Conv2d(feature_size * 8 + reduced_dim_size, 1, 4, 1, 0, bias=False
34        ),
35            nn.Sigmoid()
36        )
37    def forward(self, input_img, text_embeddings):
38        image_encoded = self.imageEncoder(input_img)
39        text_encoded = self.textEncoder(text_embeddings)
40        replicated_text = text_encoded.repeat(4, 4, 1, 1).permute(2, 3, 0, 1)
41        concat_layer = torch.cat([image_encoded, replicated_text], 1)
42        x = self.finalBlock(concat_layer)
43        return x.view(-1, 1), image_encoded
```

4. Training

Trước khi training mô hình, chúng ta định nghĩa mô số hàm để hiển thị hình ảnh được sinh ra sau mỗi 10 epochs của training.

```
1 import matplotlib.pyplot as plt
2 import torchvision
3
4 def show_grid(img):
5     npimg = img.numpy()
6     plt.imshow(np.transpose(npimg, (1, 2, 0)))
7     plt.show()
8
9 show_grid(torchvision.utils.make_grid(ds[0]["image"], normalize=True))
10
11 def plot_output(generator):
12     plt.clf()
13     with torch.no_grad():
14
15         generator.eval()
16         test_images = generator(fixed_noise.to(device), plt_o_text_embeddings.to(
17             device))
18         generator.train()
19
20         grid = torchvision.utils.make_grid(test_images.cpu(), normalize=True)
21         show_grid(grid)
```

Định nghĩa model, optimizer để huấn luyện model:

```
1 import torch.optim as optim
2
3 generator = Generator(100, 128, 3, 768, 256).to(device)
4 discriminator = Discriminator(3, 128, 768, 256).to(device)
5
6 optimizer_G = optim.Adam(generator.parameters(), lr=0.0002, betas=(0.5, 0.999))
7 optimizer_D = optim.Adam(discriminator.parameters(), lr=0.0002, betas=(0.5,
8             0.999))
9 bce_loss = nn.BCELoss()
10 l2_loss = nn.MSELoss()
11 l1_loss = nn.L1Loss()
```

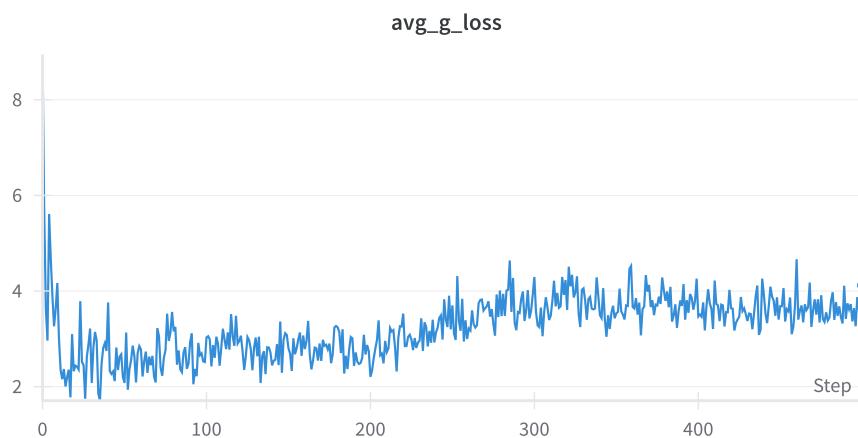
```
1 import time
2
3 epochs = 500
4
5 for epoch in range(epochs):
6     d_losses, g_losses = [], []
7     epoch_time = time.time()
8
9     for batch in dataloader:
10         images = batch["image"].to(device)
11         embed_captions = batch["embed_caption"].to(device)
12         wrong_images = batch["wrong_image"].to(device)
13
14         # labels
15         real_labels = torch.ones(images.size(0), 1, device=device)
16         fake_labels = torch.zeros(images.size(0), 1, device=device)
17
18         # Training the discriminator
19         optimizer_D.zero_grad()
20
21         # Gen fake image
22         noise = torch.randn(size=(images.size(0), 100), device=device)
23         fake_images = generator(noise, embed_captions)
24
25         # Compute real loss
26         outputs, _ = discriminator(images, embed_captions)
27         real_loss = bce_loss(outputs, real_labels)
28
29         # Compute contrastive loss for wrong image
30         outputs, _ = discriminator(wrong_images, embed_captions)
31         wrong_loss = bce_loss(outputs, fake_labels)
32
33         # Compute fake loss
34         outputs, _ = discriminator(fake_images.detach(), embed_captions)
35         fake_loss = bce_loss(outputs, fake_labels)
36
37         d_loss = real_loss + fake_loss + wrong_loss
38
39         # Update weight
40         d_loss.backward()
41
42         optimizer_D.step()
43         d_losses.append(d_loss.item())
```

```

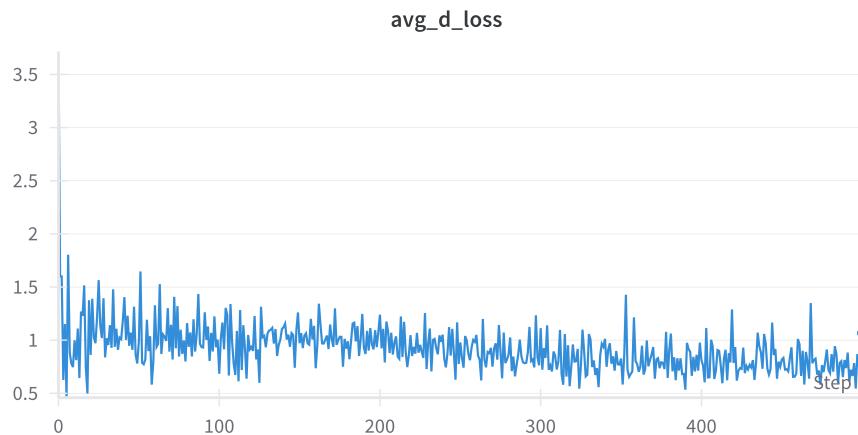
1      # Training generator
2      optimizer_G.zero_grad()
3      # create noise
4      noise = torch.randn(size=(images.size(0), 100), device=device)
5      fake_images = generator(noise, embed_captions)
6
7      outputs, fake_features = discriminator(fake_images, embed_captions)
8      _, real_features = discriminator(images, embed_captions)
9      activation_fake = torch.mean(fake_features, 0)
10     activation_real = torch.mean(real_features, 0)
11
12     # Compute loss
13     real_loss = bce_loss(outputs, real_labels)
14     g_loss = real_loss + 100 * 12_loss(activation_fake, activation_real.
15         detach()) + 50 * 11_loss(fake_images, images)
16     g_loss.backward()
17     optimizer_G.step()
18     g_losses.append(real_loss.item())
19
20     avg_d_loss = sum(d_losses)/len(d_losses)
21     avg_g_loss = sum(g_losses)/len(g_losses)
22     if (epoch+1) % 10 == 0:
23         plot_output(generator)
24
25     print("Epoch [{}/{}] loss_D: {:.4f} loss_G: {:.4f} time: {:.2f}{}".format(epoch
26         +1, epochs, avg_d_loss, avg_g_loss, time.time() - epoch_time))
27
28     # save model
29     model_save_path = "./save_model"
30     torch.save(generator.state_dict(), os.path.join(model_save_path, "generator.pth"
31         ))
32     torch.save(discriminator.state_dict(), os.path.join(model_save_path, "discriminator.pth"))

```

Kết quả training của model:



Hình 5: Generator loss.



Hình 6: Discriminator loss.

5. Deployment

Thử nghiệm mô hình sau khi huấn luyện:

```
1 generator.eval()
2 caption = "this pale pink flower has a large yellow and green pistil."
3 embed_caption = torch.tensor(bert_model.encode(caption))
4 noise = torch.randn(size=(1, 100))
5 text_embedding = embed_caption.unsqueeze(0)
6 with torch.no_grad():
7     test_images = generator(noise.to(device), text_embedding.to(device))
8 grid = torchvision.utils.make_grid(test_images.cpu(), normalize=True)
9 show_grid(grid)
```



Hình 7: Kết quả thử nghiệm.

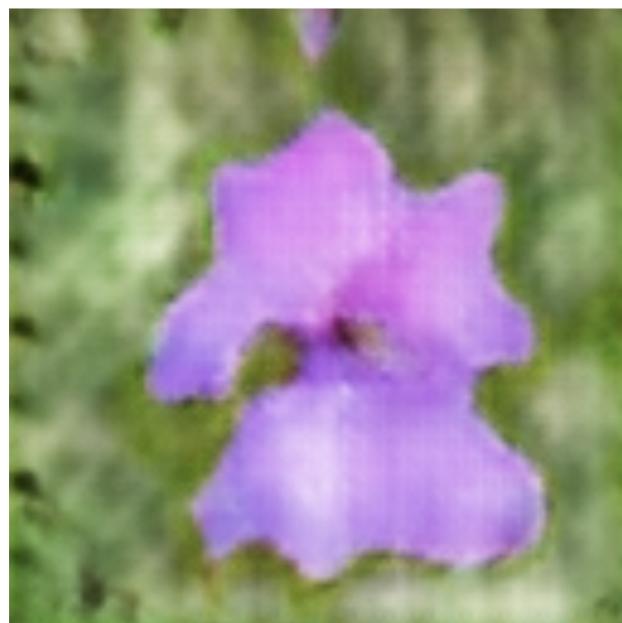
Triển khai mô hình trên streamlit. Tham khảo về [code](#) và [demo](#).

Text To Image Using DCGAN-BERTs

Model: DCGAN. Dataset: Flower. Text Encoder: BERTs

Sentence:

this pale pink flower has a large yellow pistil.



Generated Image

Hình 8: Triển khai ứng dụng trên Streamlit.

Phần 3. Câu hỏi trắc nghiệm

Câu hỏi 1 Mục đích của bài toán Text to Image là gì?

- a) Tạo hình ảnh tương ứng với thông tin mô tả từ văn bản
- b) Phân loại hình ảnh
- c) Phân loại văn bản
- d) Phát hiện cạnh của các đối tượng trong hình ảnh

Câu hỏi 2 Mô hình nào được xây dựng để giải quyết bài toán Text-to-Image?

- a) GAN-CLS
- b) BERT
- c) ResNet
- d) VGG

Câu hỏi 3 Khối Generator được sử dụng để làm gì?

- a) Sinh hình ảnh mới từ Noise và Text Embedding
- b) Dự đoán hình ảnh Real/Fake
- c) Sinh hình ảnh mới từ hình ảnh cũ
- d) Sinh văn bản mới từ Noise và Text Embedding

Câu hỏi 4 Khối Discriminator được sử dụng để làm gì?

- a) Sử dụng kết hợp hình ảnh và Text Embedding dự đoán hình ảnh Fake/Real
- b) Sử dụng Text Embedding dự đoán hình ảnh Fake/Real
- c) Cả 2 đáp án trên đều đúng
- d) Cả 2 đáp án trên đều sai

Câu hỏi 5 Kiến trúc của Generator và Discriminator không bao gồm layer nào sau đây?

- a) CNN
- b) Batch Normalization
- c) LeakyReLU Activation
- d) RNN

Câu hỏi 6 Số chiều biểu diễn văn bản được sử dụng trong phần thực nghiệm 2 là bao nhiêu?

- a) 512
- b) 1024
- c) 768
- d) 2048

Câu hỏi 7 Dựa vào code thực nghiệm trong phần 2, lớp Convolution cuối cùng trong khối Discriminator có tham số ‘in channels’ là bao nhiêu?

- a) 512

- b) 64
- c) 576
- d) 1024

Câu hỏi 8 Hàm loss nào sau đây không được sử dụng trong phần thực nghiệm?

- a) BCELoss
- b) MSELoss
- c) L1Loss
- d) CTCLoss

Câu hỏi 9 Mô hình pre-trained language model nào được sử dụng trong thực nghiệm?

- a) BERT
- b) DistilBERT
- c) RoBERTa
- d) T5

Câu hỏi 10 Dựa vào phần code thực nghiệm trong phần 3, kích thước chiều embedding của text của mô hình pre-trained language model được sử dụng là?

- a) 512
- b) 768
- c) 576
- d) 1024

Phần 4. Phụ lục

1. **Hint:** Dựa vào file tải về **Text-to-Image Using DCGAN-BERTs** để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. **Rubric:**

Phần	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none">- Hiểu rõ bài toán Text To Image và các mô hình có thể giải quyết bài toán này- Hiểu rõ mô hình DCGAN	<ul style="list-style-type: none">- Huấn luyện mô hình DCGAN cho bài toán Text to Image trên bộ dữ liệu Flowers
2.	<ul style="list-style-type: none">- Hiểu rõ mô hình ngôn ngữ ứng dụng biểu diễn văn bản thành vector- Sử dụng pretrained BERTs cho trích xuất đặc trưng văn bản	<ul style="list-style-type: none">- Sử dụng DistilBERT từ thư viện transformers- Cải tiến mô hình DCGAN với DistilBERT

- *Hết* -