

Style Transfer – Exercise

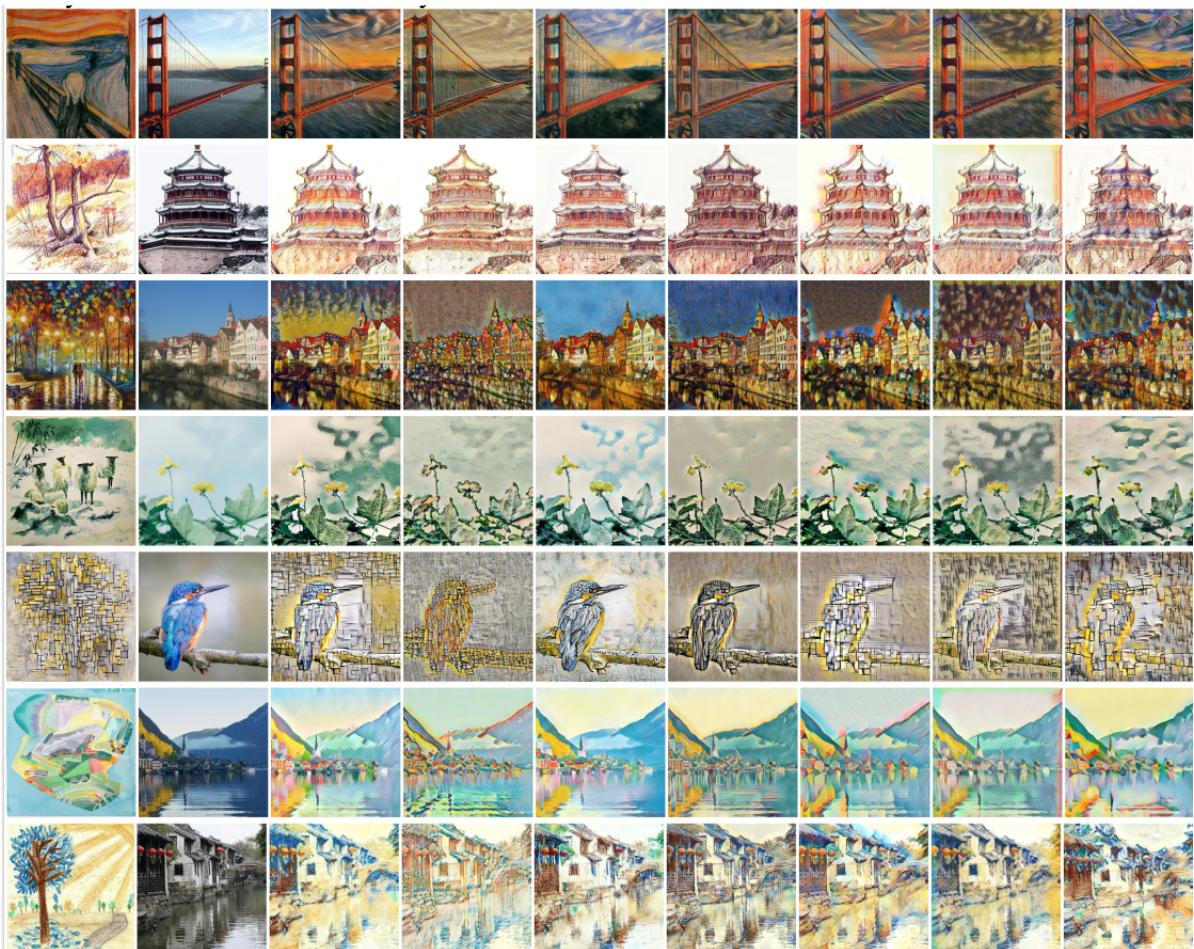
Khoa Tho Anh Nguyen

Ngày 1 tháng 3 năm 2025

Phần I: Style Transfer

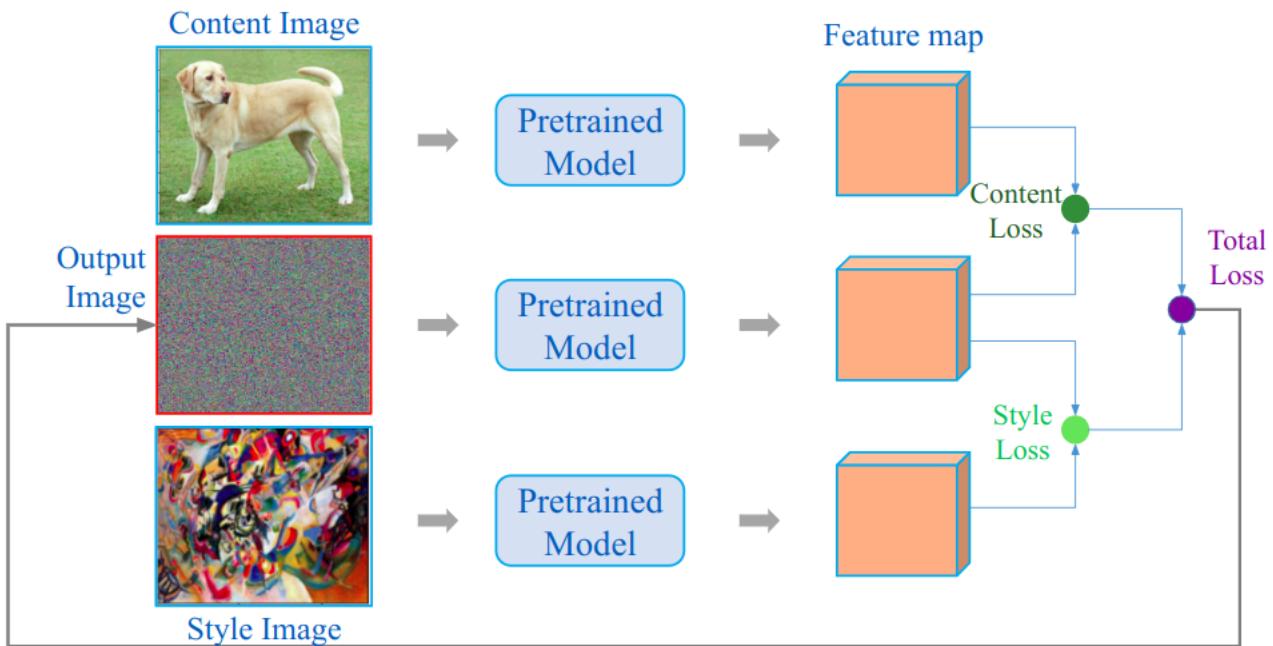
1. Style Transfer:

Style transfer là sự kết hợp giữa trí tuệ nhân tạo và nghệ thuật (art), đây là lĩnh vực thú vị nơi mà thẩm mỹ của nghệ thuật hình ảnh được kết hợp với nội dung số hóa. Quá trình này bao gồm việc áp dụng style (phong cách) nghệ thuật từ một hình ảnh và áp dụng nó lên nội dung (content) của một hình ảnh khác, từ đó tạo ra một hình ảnh mới thể hiện bản chất của cả hai.



Hình 1: Style Transfer

2. Neural Representations of Content and Style:



Hình 2: Style Transfer Pipeline

Bài toán này dựa trên việc sử dụng Mạng Nơ-ron Tích chập (CNNs), đặc biệt là các mạng được huấn luyện trước (pretrain) như VGGNet, có khả năng nắm bắt các đặc trưng của hình ảnh theo từng cấp bậc - từ các cạnh và kết cấu cơ bản đến các biểu diễn nội dung phức tạp hơn.

Content Representation: Nội dung (content) của một hình ảnh được học bởi feature map của một số layer trong mạng, thường là các lớp sâu hơn nơi mạng đã trừu tượng hóa khỏi dữ liệu pixel thô và thay vào đó biểu diễn các đặc điểm cấp cao hơn như hình dạng và đối tượng.

Style Representation: Phong cách (style) của một hình ảnh được học bởi các tương quan giữa các đặc điểm trong các layer khác nhau của mạng. Điều này được biểu diễn học bởi ma trận Gram, nắm bắt được phân phối của các đặc điểm

3. Loss Functions:

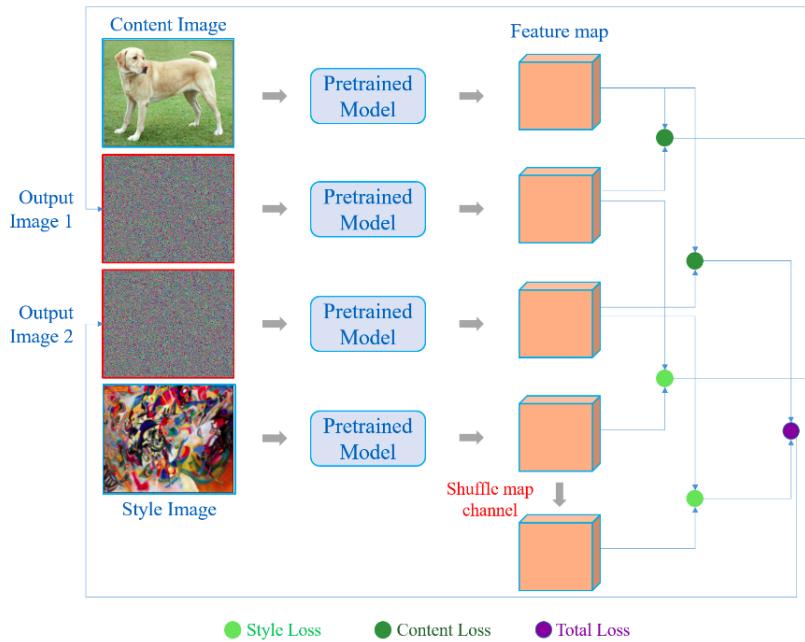
Content Loss: Đo lường mức độ nội dung của hình ảnh được tạo ra từ model sai lệch so với nội dung của hình ảnh nội dung gốc. Nó thường được tính toán bằng MSE của hình ảnh nội dung và hình ảnh được tạo ra trong một hoặc nhiều layer của CNN.

Style Loss: Đo lường sự khác biệt về phong cách giữa hình ảnh được tạo ra và hình ảnh tham khảo phong cách. MSE được sử dụng giữa các ma trận Gram của các biểu diễn phong cách của hình ảnh được tạo ra và hình ảnh phong cách qua nhiều layer của CNN.

Total Loss: Là hàm kết hợp content và style loss, thường là tổng hoặc kết hợp với các tham số để điều chỉnh lượng loss phù hợp

4. Multi-modal Style Transfer:

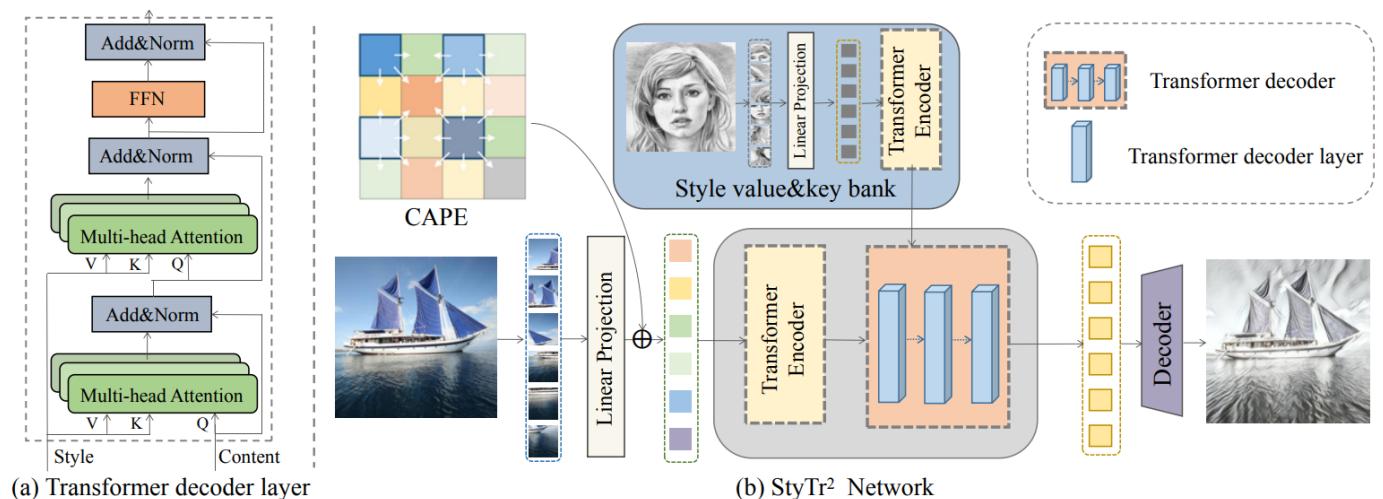
Bằng cách biến đổi feature map của ảnh phong cách (style), ta có thể tạo ra nhiều phong cách khác nhau chỉ với một ảnh phong cách đầu vào



Hình 3: Multi-modal Style Transfer

5. Image Style Transfer With Transformers:

Đây là phương pháp sử dụng transformer model cho chuyển đổi phong cách hình ảnh. Phương pháp này giải quyết những hạn chế của CNN trong việc nắm bắt thông tin toàn cục. Nó sử dụng hai transformer encoder cho nội dung và phong cách tương ứng, và một multi-layer transformer decoder để tạo phong cách cho nội dung. Một Content-Aware Positional Encoding (CAPE) được đưa vào để cải thiện positional encoding cho style transfer.



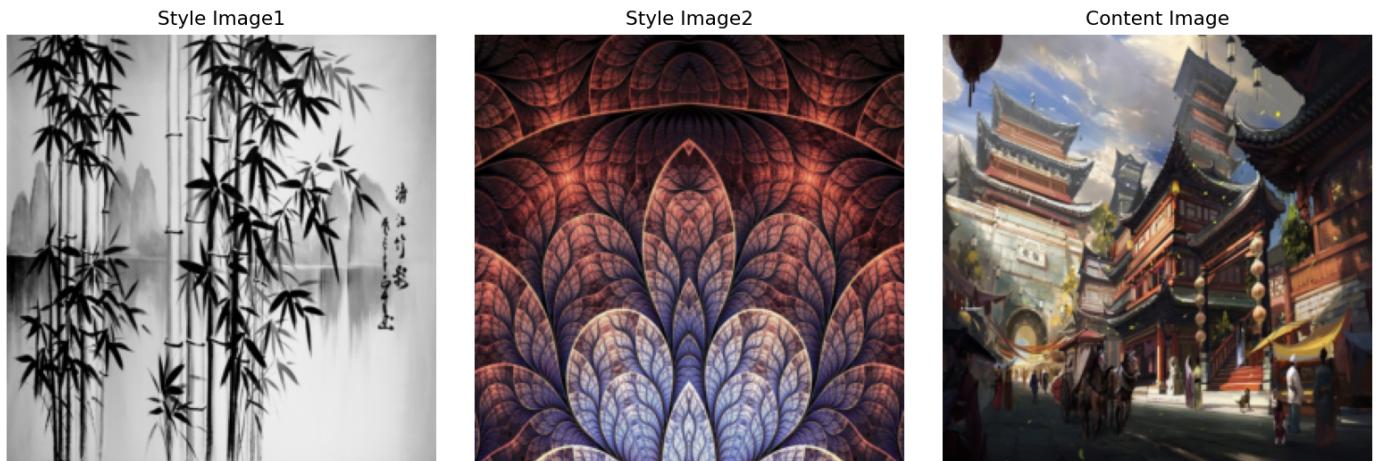
Hình 4: Style Transfer with Transformer

Phần II: Bài Tập và Gợi Ý

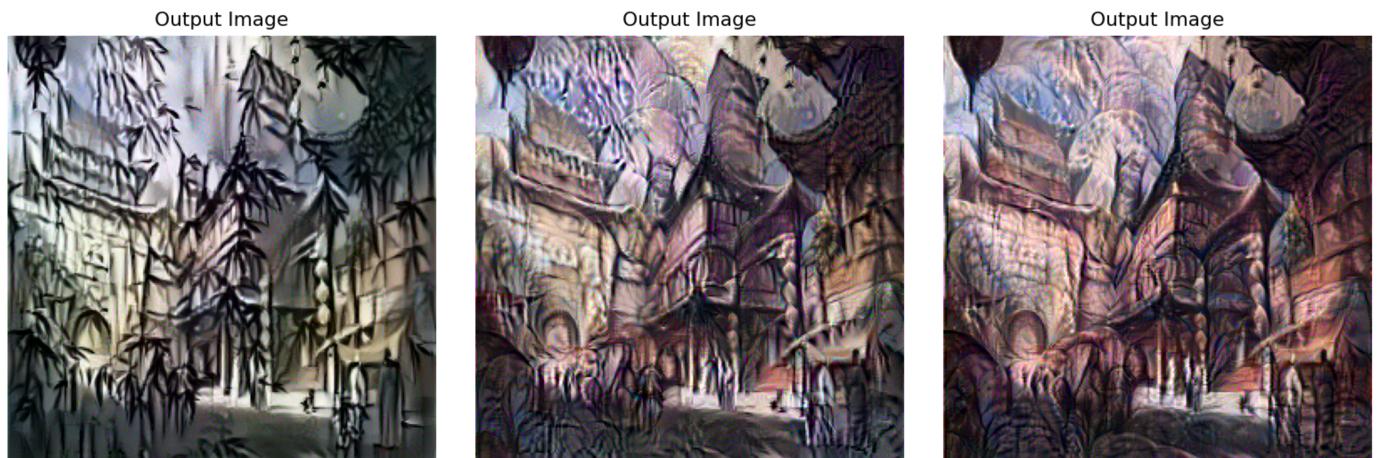
Input đầu vào cho bài toán style transfer thông thường là ảnh nội dung (content) và ảnh (style) và sinh ra ảnh output. Ma trận Gram được dùng ở hàm loss để xác định mức độ ảnh output chứa các đặc trưng có trong ảnh phong cách (style).

Trong bài tập này các bạn sẽ xây dựng các biến thể khác của bài toán style transfer. Chúng ta sẽ tập trung vào các feature của ảnh style để sáng tạo ra một biến thể phong cách (style) mới bằng cách kết hợp feature của 2 phong cách (style) hoặc tác động trực tiếp biến đổi vào feature của phong cách (style) để tạo ra một phong cách mới.

Bài 1: Dual-style Transfer, input nhận 1 ảnh content và 2 ảnh style. Chúng ta cần thực hiện trích xuất feature của cả 2 ảnh style, sau đó combine theo tỉ lệ 1/4 và 3/4 để tạo ra style mới. Cuối cùng transfer style mới này lên ảnh content để tạo ra output cuối cùng



(a) Ảnh đầu vào của bài tập 1, ảnh trái là ảnh style 1, ảnh giữa là ảnh style 2, và ảnh phải là ảnh content

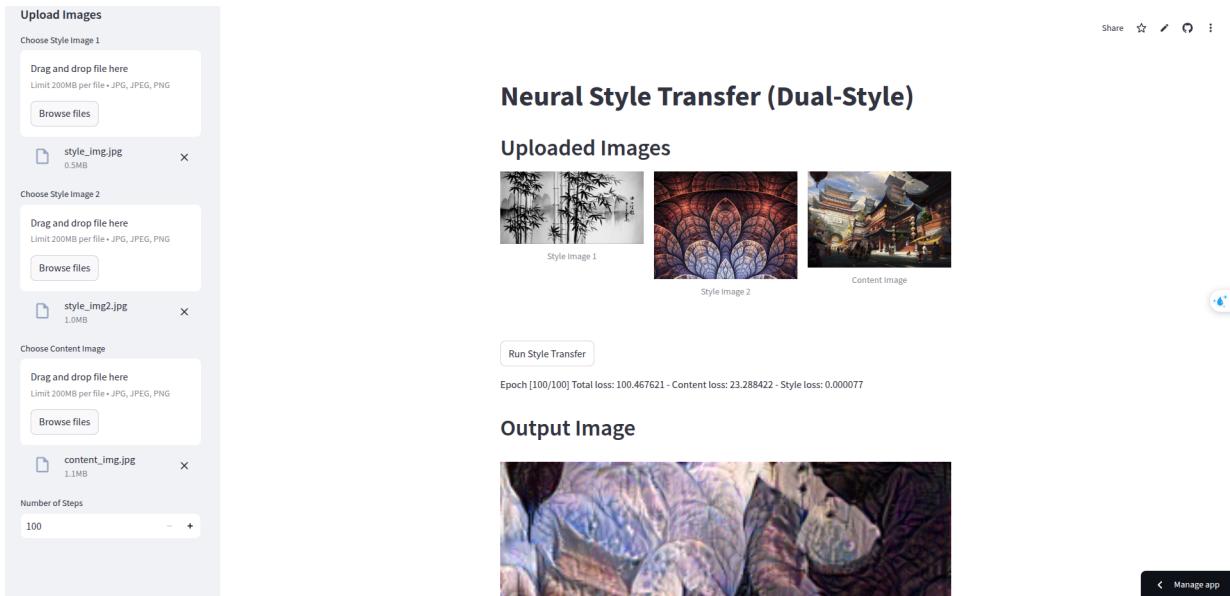


(b) Ảnh trái là ảnh khi transfer theo style 1, ảnh giữa là ảnh khi transfer theo 1/2 style 1 và 1/2 style 2, và ảnh phải là ảnh output của bài tập 1 khi transfer theo 1/4 style1 + 3/4 style 2

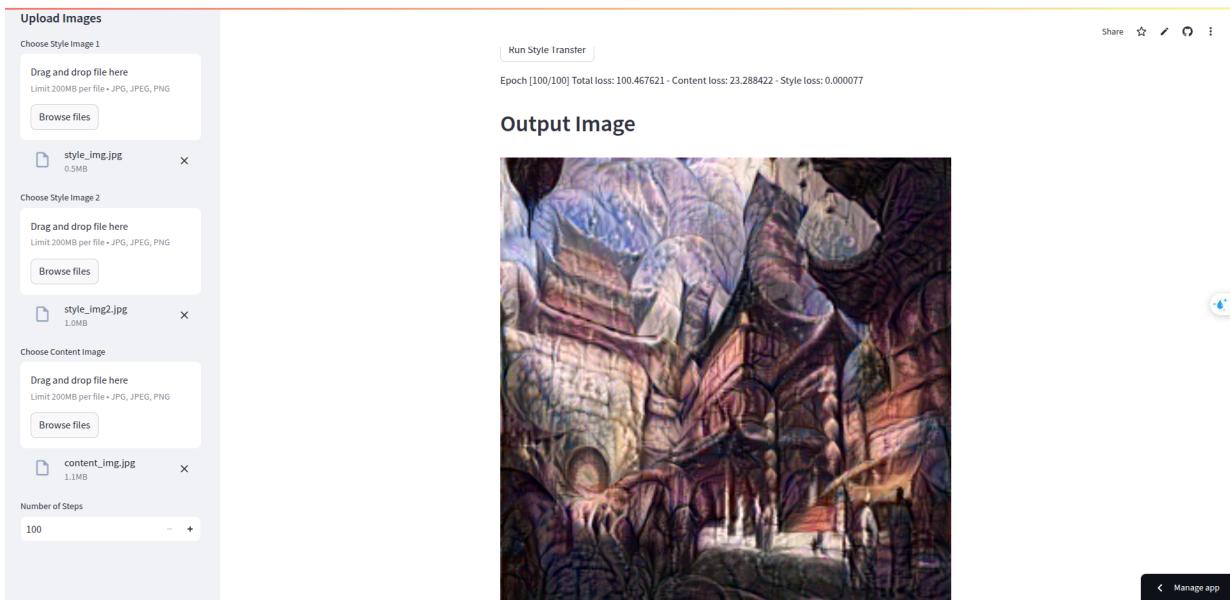
Hình 5: Bài 1 (a) input và (b) output (chỉ ảnh bên phải)

Deploy: Các bạn có thể tham khảo kết quả từ chương trình được deploy trên streamlit:

- [Github](#)
- Ảnh minh họa khi deploy bài tập 1 trên streamlit

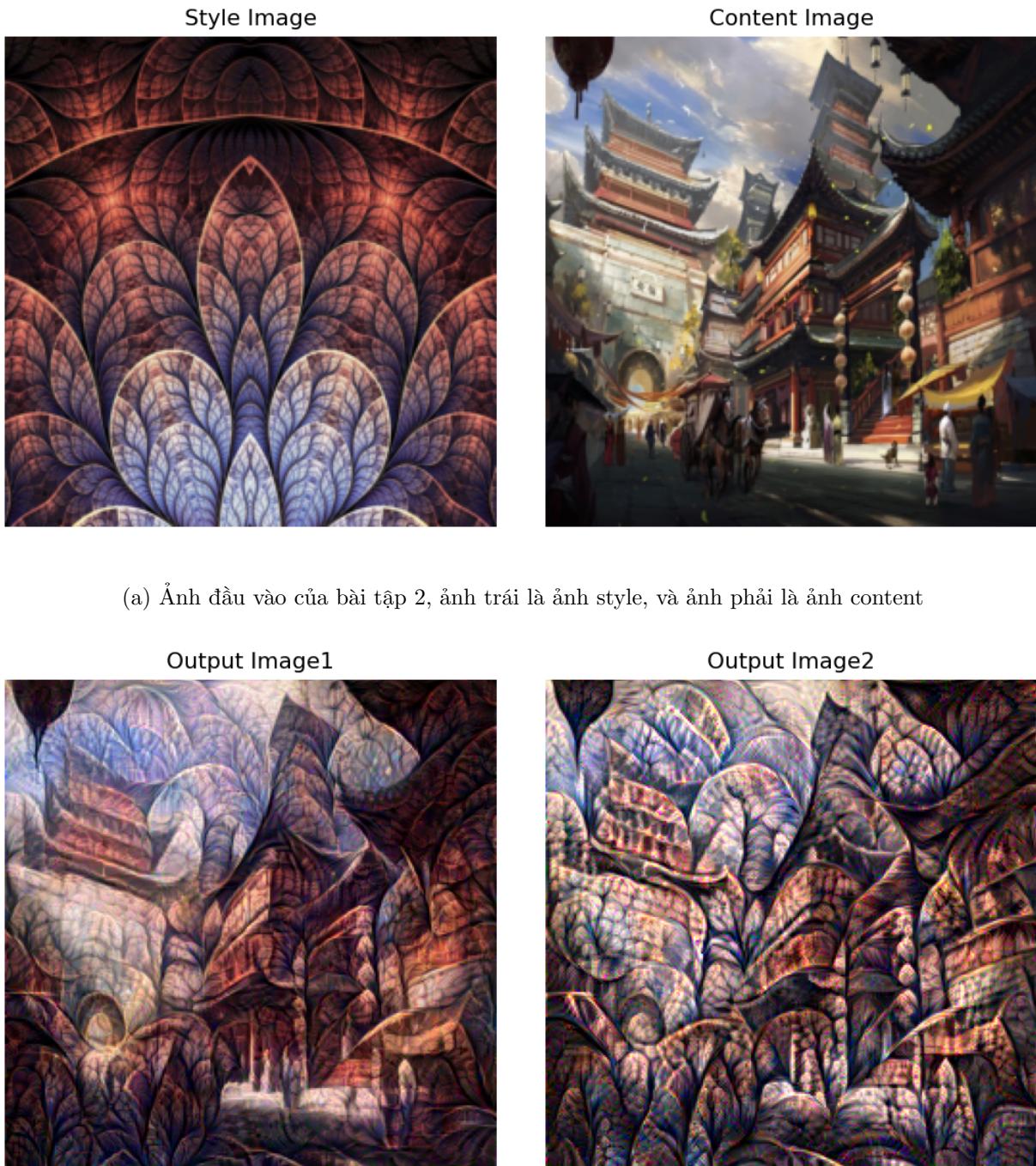


Hình 6: Bài 1 upload style1, style2 và content image



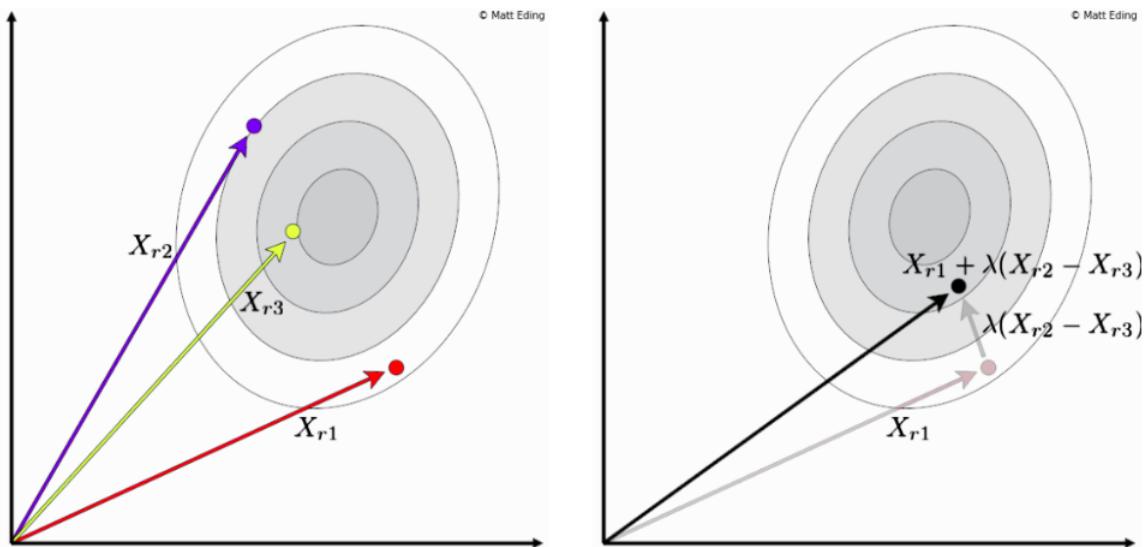
Hình 7: Bài 1 kết quả

Bài 2: Input nhận 1 ảnh content và 1 ảnh style. Chúng ta cần thực hiện trích xuất feature của ảnh style như thông thường, sau đó tạo ra 2 biến thể mới bằng cách xoay 90, và 180 độ. Tiếp theo tạo ra style mới theo công thức được lấy ý tưởng từ thuật toán differential evolution $x_{new} = x + (x_{90} - x_{180})$. Cuối cùng tạo ra 2 output được transfer theo style cũ và mới này lên ảnh content.



(b) Ảnh output của bài tập 2, ảnh trái là ảnh khi transfer theo style và ảnh phải là ảnh khi transfer theo style đã thực hiện các biến đổi rotate 90, 180 và Differential Evolution

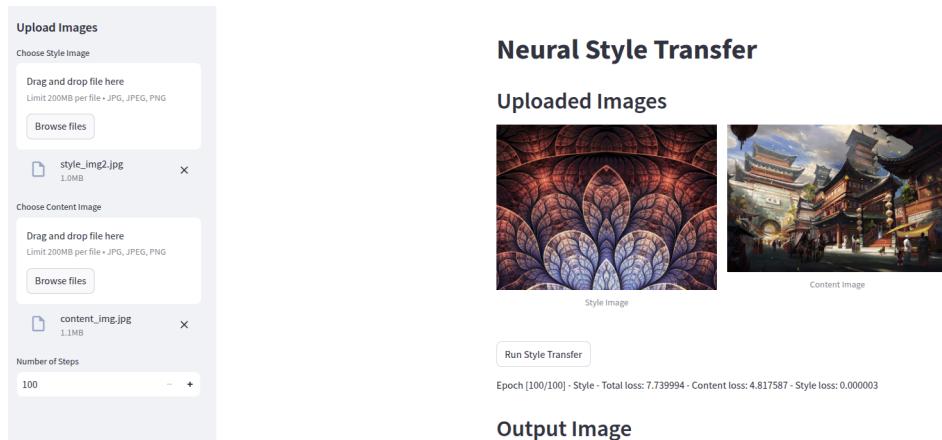
Hình 8: Bài 2 (a) input và (b) output



Hình 9: Differential Evolution

Deploy: Các bạn có thể tham khảo kết quả từ chương trình được deploy trên streamlit:

- [Github](#)
- [Ảnh minh họa khi deploy bài tập 2 trên streamlit](#)



Hình 10: Bài 2 upload style và content image

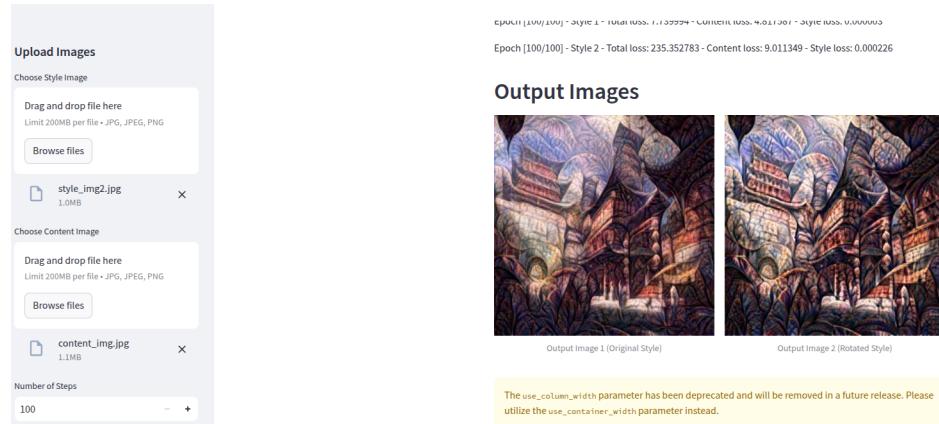
Các bạn tham khảo code hướng dẫn bên dưới để hoàn thành bài tập

1. **Load data:** Đầu tiên chúng ta cần load ảnh content và style bằng thư viện PIL, rồi resize về cùng kích thước 256x256 và convert thành tensor để sử dụng với Pytorch. Đối với bài 1 chúng ta cần load 3 ảnh 2 style và 1 content, còn đối với bài 2 chỉ cần 1 style và 1 content

```

1 from PIL import Image
2 import torchvision.transforms as transforms
3
4 imsize = 256
5
6 img_transforms = transforms.Compose([
7     transforms.Resize((imsize, imsize)),

```



Hình 11: Bài 2 kết quả (100 steps). Khuyến khích chạy từ 500 steps và chạy trên local machine do cloud của streamlit giới hạn và chạy chậm

```

8     transforms.ToTensor(),
9 )
10
11 def image_loader(image_name):
12     image = Image.open(image_name)
13     image = img_transforms(image).unsqueeze(0)
14     return image.to(device, torch.float)
15
16 style_img1 = image_loader("style_img.jpg")
17 style_img2 = image_loader("style_img2.jpg")
18 content_img = image_loader("content_img.jpg")

```

2. Loss Function: Đoạn code bên dưới thiết lập các hàm loss được sử dụng trong style transfer để cân bằng giữa việc giữ content của hình ảnh trong khi áp dụng style của một hình ảnh khác. Content loss đảm bảo rằng các đặc điểm chính của hình ảnh content được giữ nguyên, trong khi style loss đảm bảo rằng phong cách của hình ảnh style được chuyển một cách hiệu quả sang hình ảnh được tạo ra.

```

1 # Content Loss
2 content_weight = 1
3 ContentLoss = nn.MSELoss()
4
5 # Style Loss
6 def gram_matrix(tensor):
7     a, b, c, d = tensor.size()
8     tensor = tensor.view(a * b, c * d)
9     G = torch.mm(tensor, tensor.t())
10    return G.div(a * b * c * d)
11
12 style_weight = 1e6
13 StyleLoss = nn.MSELoss()

```

Content Loss:

- **content_weight:** trọng số cho content loss trong hàm loss tổng. Trọng số này điều chỉnh tầm quan trọng của content được giữ lại so với các thành phần khác của loss tổng (chẳng hạn như mất đi kiểu dáng).

- **ContentLoss:** Sử dụng Mean Squared Error (MSE) Loss. Trong bối cảnh của style transfer, MSE được sử dụng để đo lường sự khác biệt giữa các đặc điểm nội dung của hình ảnh content và các đặc điểm của hình ảnh được tạo, khuyến khích hình ảnh được tạo giữ nguyên nội dung chính của hình ảnh content.

Style Loss:

- **style_weight:** trọng số cho style loss trong hàm loss tổng. Trọng số này điều chỉnh cường độ của phong cách được transfer vào ảnh output.
- **StyleLoss:** Sử dụng Mean Squared Error (MSE) Loss
- **gram_matrix(tensor):** hàm tính toán ma trận Gram của một tensor. Ma trận Gram được sử dụng trong style loss để giữ lại các thành phần của style (kết cấu, hoa văn, màu sắc, v.v.) của hình ảnh.

3. **Model:** Chúng ta sẽ không build model từ đầu như các bài toán supervised learning khác mà ở đây ta sẽ tận dụng pre-trained VGG19 model để trích xuất feature của cả ảnh content và ảnh style. Ta sẽ cho ảnh đi qua pre-trained VGG19 model và trích xuất các feature map bên trong model. Thiết lập này thường được sử dụng trong style transfer, trong đó các feature layer của VGG19 được sử dụng để trích xuất các đặc trưng của ảnh style và ảnh content. Sau đó, đặc trưng này được sử dụng để hướng dẫn chuyển đổi hình ảnh content theo phong cách của hình ảnh style trong khi vẫn giữ nguyên nội dung của nó.

```

1 from torchvision.models import vgg19, VGG19_Weights
2
3 VGG19_pretrained = vgg19(weights=VGG19_Weights.DEFAULT).features.eval()
4 VGG19_pretrained.to(device)

```

Tiếp theo dựa vào các thành layer trong VGG19 chúng ta sẽ lấy một số layer dùng cho trích xuất style và ảnh. Đoạn code bên dưới thể hiện cách trích xuất các feature từ các layer cụ thể của VGG19 cho bài toán style transfer. Xác định các layer nào được xem xét để biểu diễn nội dung (conv_4) và phong cách (conv_1, conv_2, conv_3, conv_4, conv_5)

```

1 content_layers = ["conv_4"]
2 style_layers = ["conv_1", "conv_2", "conv_3", "conv_4", "conv_5"]
3
4 def get_features(pretrained_model, image):
5     layers = {
6         "0": "conv_1",
7         "5": "conv_2",
8         "10": "conv_3",
9         "19": "conv_4",
10        "28": "conv_5"
11    }
12    features = {}
13    x = image
14    x = normalization(x)
15    for name, pretrained_layer in pretrained_model._modules.items():
16        x = pretrained_layer(x)
17        if name in layers:
18            features[layers[name]] = x
19    return features

```

get_features(pretrained_model, image): Hàm này được thiết kế để lấy model được đào tạo trước (pretrain_model) và tensor hình ảnh đầu vào (image) và trả về một từ điển chứa các featture từ các layer được chỉ định

```

Sequential(
(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU(inplace=True)
(2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(3): ReLU(inplace=True)
(4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(6): ReLU(inplace=True)
(7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(8): ReLU(inplace=True)
(9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(11): ReLU(inplace=True)
(12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(13): ReLU(inplace=True)
(14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(15): ReLU(inplace=True)
(16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(17): ReLU(inplace=True)
(18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(20): ReLU(inplace=True)
(21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(22): ReLU(inplace=True)
(23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(24): ReLU(inplace=True)
(25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(26): ReLU(inplace=True)
(27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(29): ReLU(inplace=True)
(30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(31): ReLU(inplace=True)
(32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(33): ReLU(inplace=True)
(34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(35): ReLU(inplace=True)
(36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)

```

Hinh 12: Các layer trong VGG19

Đối với bài 1 chúng ta cần thực hiện trích xuất feature của cả 2 ảnh style, sau đó combine theo tỉ lệ 1/2 và 1/4 để tạo ra style mới. Do đó ta cần hàm get_dual_style như code bên dưới:

```

1 def get_dual_style(style_features1, style_features2, style_layers):
2     final_style_features = {}
3     for layer in style_layers:
4         sf1 = style_features1[layer]
5         sf2 = style_features2[layer]
6         ##### YOUR CODE HERE #####
7         # 1. Calculate the size of the first portion of sf1 to be used.
8         #      This is a quarter of the number of channels (dimension 1).
9         # 2. Concatenate the first portion of sf1 with the second portion of sf2
10        # along the channel dimension.
11        #      - The first portion of sf1 should contain the first "sf1_size"
12        # channels.
13        #      - The second portion of sf2 should contain the channels starting
14        #          from "sf1_size" to the end.
15        #####
16     return final_style_features

```

get_dual_style(style_features1, style_features2, style_layers):

- Hàm này nhận ba arguments: style_features1 và style_features2, là các từ điển chứa các feature được trích xuất từ hai hình ảnh có style khác nhau bằng cách sử dụng hàm get_features. style_layers, là danh sách tên layer mà các feature này được trích xuất.
- **step1** tính một 1/4 số lượng của tổng channel trong sf1 (feature của style 1)
- **step2** lấy 1/4 số lượng channel của style 1 và 3/4 số lượng channel của style 2 sau đó concatenate lại tạo thành feature của style mới.

Đối với bài 2 chúng ta cần thực hiện trích xuất feature của ảnh style như thông thường, sau đó tạo ra 2 biến thể mới bằng cách xoay 90, và 180 độ. Tiếp theo tạo ra style mới theo công thức được lấy ý tưởng từ thuật toán differential evolution $x_{new} = x + (x_{90} - x_{180})$. Do đó ta cần hàm rot_style_features như code bên dưới:

```

1 def rot_style_features(style_features, style_layers):
2     final_rot_style_features = {}
3     for layer in style_layers:
4         sf = style_features[layer].clone()
5         ##### YOUR CODE HERE #####
6         # 2. Rotate the cloned tensor 90 degrees in the spatial dimensions (2, 3)
7         .
8         # 3. Rotate the 90-degree rotated tensor another 90 degrees (180 degrees
9         # total).
10        # 4. Calculate the final rotated feature by adding the original feature
11        # to the difference between the 90-degree and 180-degree rotations.
12        #####
13        final_rot_style_features[layer] = final_rot
14    return final_rot_style_features

```

rot_style_features(style_features, style_layers):

- Hàm này nhận hai arguments: style_features, là một từ điển chứa các feature của style được trích xuất cho từng layer được chỉ định của VGG19 và style_layers, là danh sách các tên layer mà từ đó các feature của style được trích xuất
- **step1** rotate feature 90 độ
- **step2** rotate feature 180 độ
- **step3** Tạo ra feature mới theo công thức $x_{new} = x + (x_{90} - x_{180})$

4. **Train:** Cách train của style transfer sẽ khác với các bài toán supervised learning thông thường ở cái mà ta xem là biến và cần update. Đối với bài toán thông thường ta sẽ xem model là các biến và update các biến này để minimize hàm loss và input là cố định. Tuy nhiên đối với style transfer thì model là cố định và ảnh input là biến và sẽ được update qua các vòng lặp để mỗi bước có thể transfer style của ảnh style và giữ các đặc điểm nội dung chính của ảnh content.

Đầu tiên ta cần khai báo ảnh input và optimizer, ở đây ta sẽ khai báo ảnh input chính là ảnh content, từ đó ta sẽ transfer style vào ảnh input này.

```

1 import torch.optim as optim
2 target_img = content_img.clone().requires_grad_(True).to(device)
3 optimizer = optim.Adam([target_img], lr=0.02)

```

- **line 2:** Copy ảnh content thành ảnh input đồng thời cho phép tính gradient trên ảnh input để update ảnh qua các train step
- **line 3:** Sử dụng Adam optimizer và nhận ảnh input để update theo thuật toán của Adam với learning rate = 0.02

Chúng ta cần thực hiện hàm transfer cho ảnh input theo 1 step. Chúng ta sẽ trích xuất content feature và style của ảnh input. Tiếp theo ta tính toán content loss và style loss (qua ma trận gram). Sau đó ta tính hàm loss tổng bằng tổng trọng số của content loss và style loss. Kế đến ta tính gradient và update gradient này lên ảnh input.

```

1 def style_tranfer_(model, optimizer, target_img,
2                     content_features, style_features,
3                     style_layers, content_weight, style_weight):
4
5     optimizer.zero_grad()
6     with torch.no_grad():
7         target_img.clamp_(0, 1)
8         target_features = get_features(model, target_img)
9
10    content_loss = ContentLoss(content_features["conv_4"], target_features["conv_4"])
11
12    style_loss = 0
13    for layer in style_layers:
14        target_gram = gram_matrix(target_features[layer])
15        style_gram = gram_matrix(style_features[layer])
16        style_loss += StyleLoss(style_gram, target_gram)
17
18    total_loss = content_loss*content_weight + style_loss*style_weight
19    total_loss.backward(retain_graph=True)
20    optimizer.step()
21    return total_loss, content_loss, style_loss

```

style_tranfer_: Hàm được thiết kế để thực hiện một step lặp lại quy trình style transfer, trong đó mục tiêu là cập nhật target_img (ảnh input) sao cho kết hợp nội dung của hình ảnh content và phong cách của (các) hình ảnh style. Hàm này bao gồm việc tính toán content loss và style loss, tính toán tổng loss và cập nhật target_img bằng cách sử dụng gradient

- **line 5:** Đặt lại tất cả gradient được tính toán trong lần lặp trước. Trong PyTorch, gradient tích lũy theo mặc định cho mỗi lệnh gọi backward(), vì vậy chúng cần được đặt về 0 trước khi tính toán gradient cho lần lặp tiếp theo.
- **line 8, 10:** trích xuất feature của ảnh input gồm style và content. Sau đó tính content loss của ảnh input và ảnh content
- **line 12-16:** Tính style loss của các feature map ta đã trích xuất trong VGG19. mỗi feature map sẽ được lấy ra của cả ảnh input và ảnh style. Sau đó tính toán ma trận gram của cả 2 và dùng MSE để đo lường style loss. Tích lũy các style loss qua các feature map ta đã trích xuất
- **line 18:** Tính tổng có trọng số của content loss và style loss
- **line 19:** Tính gradient cho ảnh input từ loss tổng
- **line 20:** update gradient vừa tính được vào ảnh input
- Hàm này được gọi lặp đi lặp lại, với mỗi lần gọi sẽ cập nhật target_img (ảnh input) để phù hợp hơn với nội dung và style mong muốn. Qua nhiều lần lặp lại, target_img dự kiến sẽ trở thành phiên bản có content_img kết hợp các thành phần của (các) hình ảnh style.

Đối với bài 1 khi thực hiện 1 step transfer ta chỉ cần gọi hàm style_tranfer_ 1 lần. Với đoạn code bên dưới ta thực hiện quy trình tối ưu hóa lặp đi lặp lại của việc transfer phong cách với 500 step, cập nhật hình ảnh input ở mỗi bước để dần dần phù hợp hơn với nội dung của hình ảnh content và phong cách của hình ảnh style.

```

1 STEPS = 500
2
3 for step in range(STEPS):
4     optimizer.zero_grad()
5     with torch.no_grad():
6         target_img.clamp_(0, 1)
7
8     total_loss, content_loss, style_loss = style_tranfer_(
9                                     VGG19_pretrained, optimizer,
10                                    target_img, content_features,
11                                    final_style_features,
12                                    style_layers, content_weight,
13                                    style_weight)
14
15     if step % 100 == 99:
16         print(f"Epoch [{step+1}/{STEPS}] Total loss: {total_loss.item():.6f} - \
17               Content loss: {content_loss.item():.6f} - Style loss: {style_loss \
18 .item():.6f}")
19
20     with torch.no_grad():
21         target_img.clamp_(0, 1)

```

- **line 18-19:** Ta cần chặn dưới và trên ảnh output trong range từ 0 đến 1. Vì ảnh thường được biểu diễn trong range 0-255 hoặc đã normalize trong range 0-1

Đối với bài 2 khi thực hiện 1 step transfer ta cần gọi hàm style_tranfer_ 2 lần. Lần thứ nhất với ảnh style gốc và lần thứ 2 với style mới đã được biến đổi ở trên. Output sẽ là 2 ảnh đã được chuyển đổi theo 2 phong cách khác nhau. Với đoạn code bên dưới ta thực hiện quy trình tối ưu hóa lặp đi lặp lại của việc transfer phong cách với 500 step, cập nhật hình ảnh input ở mỗi bước để dần dần phù hợp hơn với nội dung của hình ảnh content và phong cách của hình ảnh style.

```

1 STEPS = 500
2
3 for step in range(STEPS):
4
5     total_loss1, content_loss1, style_loss1 = style_tranfer_(
6                                     VGG19_pretrained, optimizer1,
7                                     target_img1, content_features,
8                                     style_features1, style_layers,
9                                     content_weight, style_weight)
10
11     total_loss2, content_loss2, style_loss2 = style_tranfer_(
12                                     VGG19_pretrained, optimizer2,
13                                     target_img2, content_features,
14                                     final_rot_style_features,
15                                     style_layers, content_weight,
16                                     style_weight)
17
18     if step % 100 == 99:
19         print(f"Epoch [{step+1}/{STEPS}] Total loss1: {total_loss1.item():.6f} - \
20 \
21               Content loss1: {content_loss1.item():.6f} - Style loss1: { \
22 style_loss1.item():.6f}")
23         print(f"Epoch [{step+1}/{STEPS}] Total loss2: {total_loss2.item():.6f} - \
24 \
25               Content loss2: {content_loss2.item():.6f} - Style loss2: { \
26 style_loss2.item():.6f}")
27
28     with torch.no_grad():
29         target_img1.clamp_(0, 1)

```

26

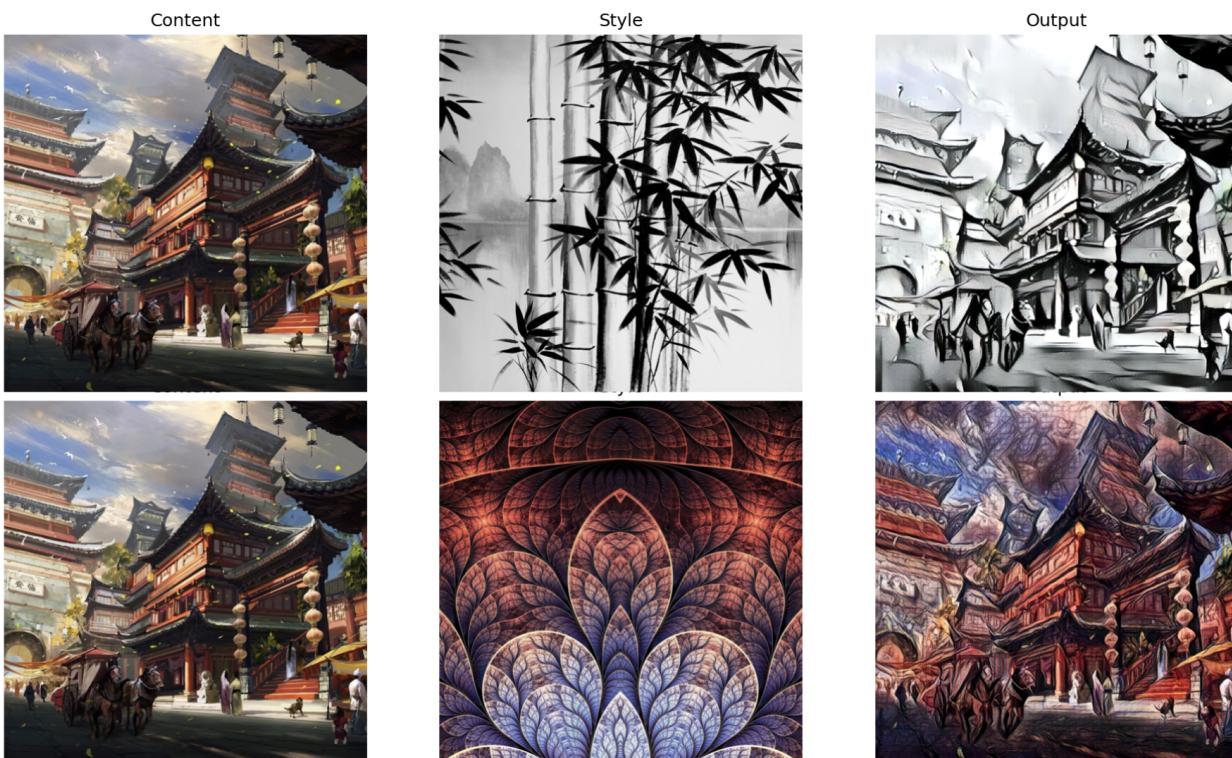
target_img2.clamp_(0, 1)

Bài 3: Mục tiêu của bài này chỉ sử dụng code thư viện đã đă có sẵn để chạy style transfer với transformer model và so sánh kết quả với CNN (VGG19). Các bạn có thể tham khảo cách sử dụng ở file hint chỉ với vài dòng code đơn giản.

```

1 import torch
2 import matplotlib.pyplot as plt
3 from StyTR.util.utils import process_images
4 from StyTR.util.utils import network as StyTR_model
5 import numpy as np
6
7 content_path = "/content/content_img.jpg"
8 style_path = "/content/style_img2.jpg"
9
10 content, style = process_images(content_path, style_path)
11
12 with torch.no_grad():
13     output= StyTR_model(content,style)
14 output = output[0].cpu()
15
16 fig, ax = plt.subplots(1, 3, figsize=(15, 5))
17 ax[0].imshow(content.cpu()[0].permute(1, 2, 0))
18 ax[0].set_title("Content")
19 ax[1].imshow(style.cpu()[0].permute(1, 2, 0))
20 ax[1].set_title("Style")
21 ax[2].imshow(output[0].permute(1, 2, 0))
22 ax[2].set_title("Output")
23 plt.show()

```



Hình 13: Style Transfer with Transformer

Phần III: Trắc Nghiệm

1. Style transfer là sự kết hợp giữa?
(A). Trí tuệ nhân tạo và ngôn ngữ tự nhiên
(C). Xử lý ảnh và lập trình máy tính
(B). Trí tuệ nhân tạo và nghệ thuật
(D). Trí tuệ nhân tạo và âm thanh
2. Phương pháp nào thường được sử dụng để biểu diễn phong cách (style) của hình ảnh trong style transfer?
(A). Gram matrix
(C). Sobel operator
(B). Fourier transform
(D). Histogram
3. Style transfer thường dựa trên việc sử dụng mạng nơ-ron nào?
(A). RNN (Recurrent Neural Network)
(C). LSTM (Long Short-Term Memory)
(B). CNN (Convolutional Neural Network)
(D). MLP (Multilayer Perceptron)
4. Các loại model nào thường được sử dụng trong Style transfer?
(A) VGG (Very Deep Convolutional Networks)
(B) ResNet (Residual Neural Network)
(C) Tất cả đều sai
(D) Các pretrain CNN model
5. Đối với Style transfer, content loss thường được tính toán như thế nào?
(A). Histogram của hình ảnh
(B). Mean Squared Error (MSE) giữa biểu diễn nội dung của hình ảnh gốc và hình ảnh được tạo
(C). Gradient của hình ảnh
(D). Mean Squared Error (MSE) giữa biểu diễn Gram matrix của hình ảnh phong cách và hình ảnh được tạo
6. Đối với Style transfer, style loss thường được tính toán như thế nào??
(A). Histogram của hình ảnh
(B). Mean Squared Error (MSE) giữa biểu diễn nội dung của hình ảnh gốc và hình ảnh được tạo
(C). Gradient của hình ảnh
(D). Mean Squared Error (MSE) giữa biểu diễn Gram matrix của hình ảnh phong cách và hình ảnh được tạo
7. Trong style transfer, loss cuối cùng dùng để cập nhật ảnh được tạo ra là ?
(A). Style loss
(C). Tổng có trọng số của style và content loss
(B). Content Loss
(D). Tất cả đều đúng
8. Style transfer thường được áp dụng trong lĩnh vực nào sau đây?
(A). Phân loại hình ảnh y tế
(C). Phát hiện gian lận tài khoản ngân hàng
(B). Nghệ thuật số và thiết kế đồ họa
(D). Tổ chức sự kiện

9. Cách nào sau đây có thể thực hiện được multi-modal style transfer?

- (A). Bằng cách biến đổi feature map của ảnh nội dung (content), ta có thể tạo ra nhiều phong cách khác nhau chỉ với một ảnh phong cách đầu vào.
- (B). Bằng cách biến đổi feature map của ảnh output được tạo ra, ta có thể tạo ra nhiều phong cách khác nhau chỉ với một ảnh phong cách đầu vào.
- (C).** Bằng cách biến đổi feature map của ảnh phong cách (style), ta có thể tạo ra nhiều phong cách khác nhau chỉ với một ảnh phong cách đầu vào.
- (D). Tất cả đều đúng.

10. Mục tiêu của style transfer là gì?

- (A). Chuyển đổi một hình ảnh thành một hình ảnh khác với cùng một phong cách nhưng nội dung khác nhau.
- (B). Giảm kích thước của hình ảnh để tiết kiệm không gian lưu trữ.
- (C). Tạo ra một hình ảnh có chất lượng cao hơn so với hình ảnh gốc.
- (D).** Chuyển đổi một hình ảnh thành một hình ảnh khác với cùng một nội dung nhưng phong cách khác nhau.

11. Đoạn code nào sau đây là phù hợp với yêu cầu bài tập 1:

```

1 def get_dual_style(style_features1, style_features2, style_layers):
2     final_style_features = {}
3     for layer in style_layers:
4         sf1 = style_features1[layer]
5         sf2 = style_features2[layer]
6         ##### YOUR CODE HERE #####
7         # 1. Calculate the size of the first portion of sf1 to be used.
8         # This is a quarter of the number of channels (dimension 1).
9         # 2. Concatenate the first portion of sf1 with the second portion of sf2
10        # along the channel dimension.
11        # - The first portion of sf1 should contain the first "sf1_size"
12        # - The second portion of sf2 should contain the channels starting
13        #   from "sf1_size" to the end.
14        #####
15     return final_style_features

```

(A).

```

1 sf1_size = int(sf1.size()[1] / 4)
2     final_style_features[layer] = torch.concatenate([
3             sf1[:, :sf1_size, :, :], 
4             sf2[:, sf1_size:, :, :]
5         ], dim=1)

```

(B).

```

1 sf1_size = int(sf1.size()[1] * 4)
2     final_style_features[layer] = torch.concatenate([
3             sf1[:, :sf1_size, :, :], 
4             sf2[:, sf1_size:, :, :]
5         ], dim=1)

```

(C).

```

1 sf1_size = int(sf1.size()[1] / 4)
2     final_style_features[layer] = torch.concatenate([
3         sf1[:, :sf1_size, :, :], 
4         sf1[:, sf1_size:, :, :]
5     ], dim=1)

```

(D).

```

1 sf1_size = int(sf1.size()[1] * 4)
2     final_style_features[layer] = torch.concatenate([
3         sf2[:, :sf1_size, :, :], 
4         sf2[:, sf1_size:, :, :]
5     ], dim=1)

```

12. Đoạn code nào sau đây là phù hợp với yêu cầu bài tập 2:

```

1 def rot_style_features(style_features, style_layers):
2     final_rot_style_features = {}
3     for layer in style_layers:
4         sf = style_features[layer].clone()
5         ##### YOUR CODE HERE #####
6         # 2. Rotate the cloned tensor 90 degrees in the spatial dimensions (2, 3)
7
7         # 3. Rotate the 90-degree rotated tensor another 90 degrees (180 degrees total).
8         # 4. Calculate the final rotated feature by adding the original feature
9         # to the difference between the 90-degree and 180-degree rotations.
10        #####
11        final_rot_style_features[layer] = final_rot
12    return final_rot_style_features

```

(A).

```

1 rot90 = torch.rot90(sf.clone(), 1, (2, 3))
2 rot180 = torch.rot90(rot90.clone(), 1, (2, 3))
3 final_rot = sf + (rot90 - rot180)

```

(B).

```

1 rot90 = torch.rot90(sf.clone(), 1, (2, 3))
2 rot180 = torch.rot180(rot180.clone(), 1, (2, 3))
3 final_rot = sf + (rot90 - rot180)

```

(C).

```

1 rot90 = torch.rot90(sf.clone(), 1, (2, 3))
2 rot180 = torch.rot90(rot180.clone(), 1, (2, 3))
3 final_rot = sf + (rot90 - rot180)

```

(D). Tất cả đều sai

Phần III: Phụ Lục:

- (a) **Datasets:** Các file data có thể tải tại [Link](#)
- (b) **Hint:** Các file code gợi ý có thể tải tại [Link](#)
- (c) **Solution:** Các file code cài đặt hoàn chỉnh và phâ trả lời nội dung trắc nghiệm có thể tả tại [Link](#) (**Lưu ý** Sáng thứ 3 khi hết deadline phần bài tập ad mới copy các nội dung bài giải nêu trên vào đường dẫn)
- (d) **Rubric:**

Style Transfer - Rubric		
Câu	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Khái niệm về Neural Style Transfer - Khái niệm về Content Loss trong Style Transfer - Khái niệm về Style Loss trong Style Transfer khi kết hợp 2 style feautre để tạo ra syle mới - Hiểu biết về cách trích xuất đặc trưng từ hai ảnh style - Kỹ năng kết hợp các đặc trưng để tạo ra style mới. 	<ul style="list-style-type: none"> - Biết cách code kết hợp đặc trưng một cách sáng tạo để tạo ra style mới - Biết cách code trích xuất feature của các layer trong pre-trained model - Biết cách code full luồng cho bài toán style transfer
2	<ul style="list-style-type: none"> - Khái niệm về Neural Style Transfer - Khái niệm về Content Loss trong Style Transfer - Khái niệm về Style Loss trong Style Transfer khi áp dụng các thuật toán làm biến đổi feature map để tạo ra style mới - Hiểu biết về cách biến đổi feature map của style để tạo ra style mới 	<ul style="list-style-type: none"> - Biết cách code kết biến đổi đặc trưng (áp dụng các thuật toán khác nhau) một cách sáng tạo để tạo ra style mới - Biết cách code trích xuất feature của các layer trong pre-trained model - Biết cách code full luồng cho bài toán style transfer