Make Neural Style Transfer (NST) model

PyTorch

https://github.com/hardiantots/nst_projectHTS.git

Hardianto Tandi Seno

hardiantotandiseno@gmail.com

Content

Neural Style Transfer Model

What is NST?

Neural Style Transfer (**NST**) is a technique for **combining the style of an image** with **content from another image** using an artificial neural network. **NST** allows us to **transfer art styles** from **one image to another** in a **very realistic** way.

Content Image Style Image Neural Style Transfer

NST **involves two images**, namely **a style image** and **a content image**. The style image is the image that contains the art style we want to apply, while the content image is the image to which we want to apply the style. NST then generates an image that has the content from the content image and the style from the style image.

NST Workflow

Preparation	Setting up image styles and content images
Feature Extraction	Use CNN to extract features from images
Loss Computation	Compute content loss and style loss, and combine them to total loss
Optimization	Adjusts the output image to minimize the calculated total loss
Output	The output result is an image that has content from the style image & content image

Implementation NST Model in Pytorch

Implementation NST Model in Pytorch

1. **Import all package that needed**

```
import torch
import numpy as np
import matplotlib.pyplot as plt

from PIL import Image
from torchvision import transforms, models
from torchvision.models import VGG19_Weights

%matplotlib inline
```

Implementation NST Model in Pytorch

2. Load VGG19 Model from PyTorch

Choosing VGG19 model because this model has faster training speed, fewer training samples per time, and higher accuracy. Additionally, **many are using VGG19 to perform NST methods**

```
device = "cuda" if torch.cuda.is_available() else "cpu"

model = models.vgg19(weights=VGG19_Weights.DEFAULT).features

for param in model.parameters():
    param.requires_grad_(False)

model.to(device)
```

Implementation NST Model in Pytorch

3. Make Function to read and processing Content & Style Image

There are **function load_image()** for **load image** using **PIL Library** and **processing image** using **torchvision.transforms**

```
def load_image(img_path, max_size=400, shape=None):
    img = Image.open(img_path)

    if max(img.size) > max_size:
        size = max_size
    else:
        size = max(img.size)

    if shape is not None:
        size = shape

    img_transform = transforms.Compose([
        transforms.Resize((size, int(1.5*size))),
        transforms.ToTensor(),
        transforms.Normalize((0.485, 0.456, 0.406),
                             (0.229, 0.224, 0.225))
    ])

    image = img_transform(img)[:3, :, :].unsqueeze(0)

    return image
```

```
content  = load_image(img_path ="contentimg/ancient_city.jpg").to(device)
style = load_image(img_path ="styleimg/bfmosaic.jpg").to(device)
```

Implementation NST Model in Pytorch

4. Displaying Content & Style Image

To show the Content & Style Image, we must make **function** (**im_convert**()) to convert the result of processing image back to numpy array for show the image

```
def im_convert(img_tensor):
    image = img_tensor.to("cpu").clone().detach()
    image = image.numpy().squeeze()
    image = image.transpose(1,2,0)
    image = image * np.array((0.229, 0.224, 0.225)) + np.array((0.485, 0.456, 0.406))
    image = image.clip(0, 1)

    return image
```

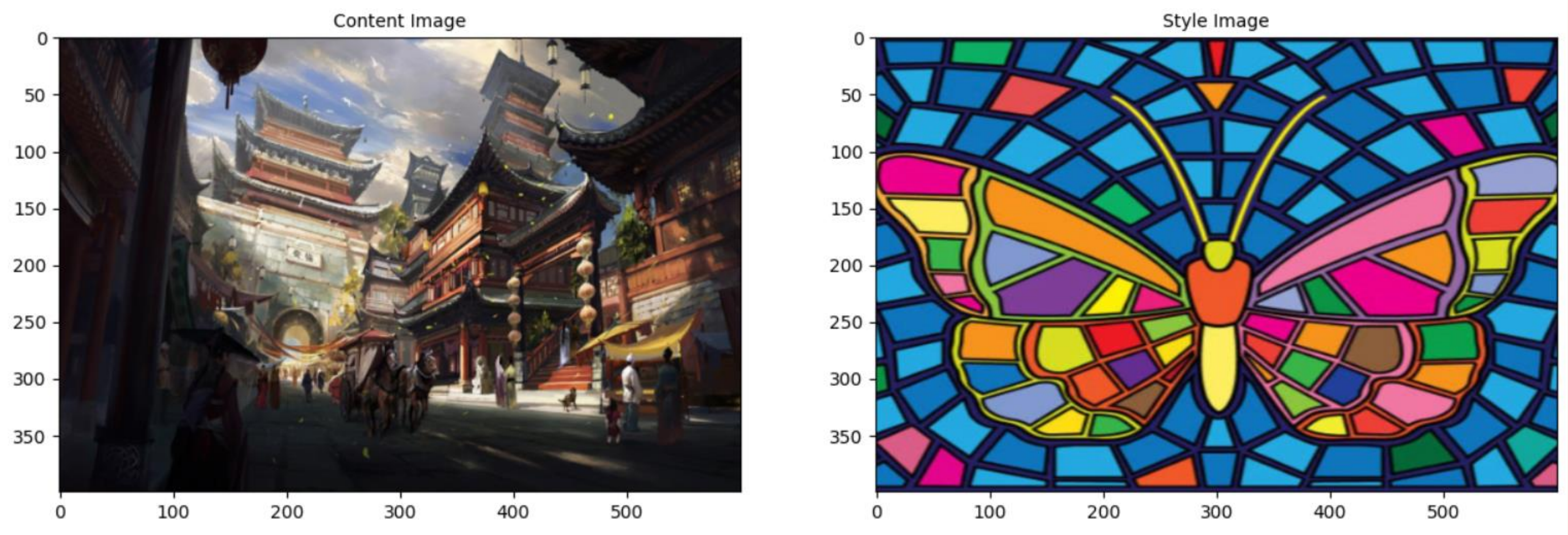
And then we can show the image with using matplotlib library

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

ax1.imshow(im_convert(content))
ax1.set_title("Content Image", fontsize=10)

ax2.imshow(im_convert(style))
ax2.set_title("Style Image", fontsize=10)

plt.show()
```



Implementation NST Model in Pytorch

5. Set the intermediate layers for Content & Style

The function **performs a forward pass** through the model, one layer at a time, and **stores the feature map responses** if the **name of the layer matches one of the keys in the predefined layer dict.** This dict serves as a mapping from the Pytorch VGG19 implementation's layer indices to the layer names defined in the paper. **If no layers are specified**, we'll **use a complete set of both** the **content layer and the layer style as a default.**

```
def get_features(image, model, layers=None):
    if layers is None:
        layers = { # 0, 5, 10, 19, 28 is Style Extraction
            '0': 'conv1_1',
            '5': 'conv2_1',
            '10': 'conv3_1',
            '19': 'conv4_1',
            '30': 'conv5_2', # Content Extraction
            '28': 'conv5_1'
        }

    features = {}
    x = image
    for name, layer, in model._modules.items():
        x = layer(x)
        if name in layers:
            features[layers[name]] = x

    return features
```

Implementation NST Model in Pytorch

6. Set the Loss Function & Assign the weights

```python
def gram_matrix(img_tensor):
    _, d, h, w = img_tensor.size()
    img_tensor = img_tensor.view(d, h * w)
    gram = torch.mm(img_tensor, img_tensor.t())

    return gram

# Set the loss function
content_features = get_features(content, model)
style_features = get_features(style, model)

style_grams = {layer: gram_matrix(style_features[layer]) for layer in style_features}

target = content.clone().requires_grad_(True).to(device)
```

Make the **function gram_matrix**() for **calculate the gram matrices** for **each layer**. Start by cloning the content image and then iteratively change its style.

Weights are assigned to each style layer. Weight the previous layer with **a higher number** to get **a bigger style artefact**. In addition, **weights are given** for the overall strength of the **two individual loss terms** (**content and style image**).

The loss function look in content loss (Mean Square Error between two feature map responses of the target image and the content image.) & **style loss** (Similar like content loss, replacing the feature map response by the Grams matrix and also dividing the mean squared error by the total number of elements in each feature map.). **Implementation of this in the next slide**

```
style_weights = {
        'conv1_1': 1,
        'conv2_1': 0.75,
        'conv3_1': 0.5,
        'conv4_1': 0.25,
        'conv5_1': 0.25
}

content_weight = 1e-2
style_weight = 1e9
```

Implementation NST Model in Pytorch

7. Train & Optimize the Model

The optimizer used in the process of training this model is **Adam Optimizer** with **Learning Rate 0.002** to **decrease loss of model**. **Total iteration** is set to **5000.**

During the loop, **call the feature from VGG** and the **content loss calculation process will be performed**. **Get a force representation** to **calculate the style loss**. Then, **calculate the total loss.** After that, do the **back-propagation step** and **update the image pixel value** iteratively until it's finished.

```python
show = 400

optimizer = torch.optim.Adam([target], lr=0.002)
steps = 5000

for i in range(1, steps+1):
    target_features = get_features(target, model)
    content_loss = torch.mean((target_features['conv5_2'] - content_features['conv5_2'])**2)

    style_loss = 0
    for layer in style_weights:
        target_feature = target_features[layer]
        target_gram = gram_matrix(target_feature)
        _, d, h, w = target_feature.shape
        style_gram = style_grams[layer]
        layer_style_loss = style_weights[layer] * torch.mean((target_gram - style_gram)**2)
        style_loss += layer_style_loss / (d * h * w)

    total_loss = content_weight * content_loss + style_weight * style_loss

    optimizer.zero_grad()
    total_loss.backward()
    optimizer.step()

    if i % show == 0:
        plt.figure(figsize=(8,4))
        print('Total loss: ', total_loss.item())
        plt.imshow(im_convert(target))
        plt.show()

final_img = im_convert(target)
```

Implementation NST Model in Pytorch

8. Show the results & Save the image

After the training process on the model is done, **we can display the results** of **combining the content & image styles.**

In addition, the result of **combining the content and style** of the image **can be saved** with **pathlib library** to make new folder and **matplotlib library** for save image

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))

ax1.imshow(im_convert(content))
ax1.set_title("Content Image", fontsize=10)

ax2.imshow(im_convert(style))
ax2.set_title("Style Image", fontsize=10)

ax3.imshow(final_img)
ax3.set_title("Result Image", fontsize=10)
plt.axis('off')

plt.show()
```

```
from pathlib import Path
RESULTS_PATH = Path("results")
RESULTS_PATH.mkdir(parents=True, exist_ok=True)

plt.imshow(final_img)
plt.axis('off')
plt.savefig('results/results_img.png')
```

Implementation NST Model in Pytorch

9. Save the Model

```
MODEL_PATH = Path("models")
MODEL_PATH.mkdir(parents=True, exist_ok=True)

MODEL_NAME = "abstract.pth"
MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME

print(f"Saving model to: {MODEL_SAVE_PATH}")
torch.save(obj=model.state_dict(),
           f=MODEL_SAVE_PATH)
```

We can also **save the results of model training** that has been done according to the image style used so that when we want to apply the style, we just have to **load state_dict into the model.**

Conclusion

From the various explanations and stages of implementing Neural Style Transfer on PyTorch, I can conclude that this **NST has quite different implementation stages** because there are stages for setting the intermediate layers, determining different loss functions, and also the process of training the model which calls the feature from VGG.

Apart from that, from implementing it on PyTorch, we can try to make style transfers with different image styles so that we can produce various styles that can be applied to the image content.

To see the completed code, **can see in ipynb file in GitHub**

https://github.com/hardiantots/nst_projectHTS.git

Sorry if there are still deficiencies in it. I will try to continue to develop my skills so that I can be even better in the future.

Thank You

Hardianto Tandi Seno

hardiantotandiseno@gmail.com