Code reference link

https://pytorch.org/tutorials/advanced/neural_style_tutorial.html

Dataset reference link

https://www.kaggle.com/datasets/almightyj/person-face-dataset-thispersondoesnotexist/data

https://www.kaggle.com/datasets/almightyj/person-face-sketches

The following is a detailed explanation of the code I added based on the reference link

These two lines of code are used to define the directory paths for content images and style images. content_dir is the directory for content images, and style_dir is the directory for style images.

```
//Set up a dataset of content and style images
content_dir = "E:/Coding3/dataset/sketches"
style_dir = "E:/Coding3/dataset/face"
```

This section of code sets the size of the output image and defines image conversion operations, including resizing the image and converting the image to Tensor format. Different image sizes are used depending on whether a GPU is available.

```
// Set the desired size of the output image, use a larger size if a GPU is available
imsize = 512 if torch.cuda.is_available() else 128

// Image conversion, including resizing and conversion to Tensor
transform = transforms.Compose([
    transforms.Resize(imsize),  // Resize an image
    transforms.ToTensor()       // Convert to Tensor
])
```

These two lines of code use the ImageFolder function to load the content and style image datasets and apply the image transformation operations defined previously.

```
// Loading the dataset
content_dataset = ImageFolder(content_dir, transform=transform)
```

```
style_dataset = ImageFolder(style_dir, transform=transform)
```

This part of the code sets the batch size of data loading to 500 and creates a data loader to batch load the content and style image datasets, using the shuffle=True parameter to randomly shuffle the data.

```
//Setting the batch size
batch_size = 500

// Creating a Data Loader
content_loader = DataLoader(content_dataset, batch_size=batch_size,
shuffle=True)
style_loader = DataLoader(style_dataset, batch_size=batch_size,
shuffle=True)
```

This code lists the files in the content and style image directories, gets the path to the first content and style image, and then uses the image_loader function to load the image. If the image is loaded successfully, it logs success information and prints the image size; if the image is not found, it logs an error message.

```
// Load image path
content_files = os.listdir(content_dir)
style_files = os.listdir(style_dir)
content_img_path = os.path.join(content_dir, content_files[0]) if
content_files else None
style_img_path = os.path.join(style_dir, style_files[0]) if
style_files else None

// Loading content images
if content_img_path:
    content_img = image_loader(content_img_path)
    loguru.logger.info(f'成功加载内容图像: {content_img_path}')
    print(f'内容图像大小: {content_img.size()}')
else:
    loguru.logger.error("在目录中找不到内容图像。")

// Loading style image
if style_img_path:
    style_img = image_loader(style_img_path)
    loguru.logger.info(f'成功加载风格图像: {style_img_path}')
    print(f'风格图像大小: {style_img.size()}')
else:
    loguru.logger.error("在目录中找不到风格图像。")
```

Two functions are defined here: unloader is used to convert images in Tensor format back to PIL image format, and imshow is used to display images. The unloader function first clones the Tensor, removes the batch dimension, and converts it to a PIL image; the imshow function calls unloader to convert the Tensor to a PIL image and displays the image using matplotlib.

```
// Defining the unloader function
def unloader(image):
    image = image.cpu().clone()  //Clone a Tensor without modifying
the original Tensor
    image = image.squeeze(0)      // Remove the fake batch dimension
    image = transforms.ToPILImage()(image)
    return image

// Define the imshow function
def imshow(tensor, title=None):
    image = unloader(tensor)
    plt.imshow(image)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)  //Pause to update the image
```

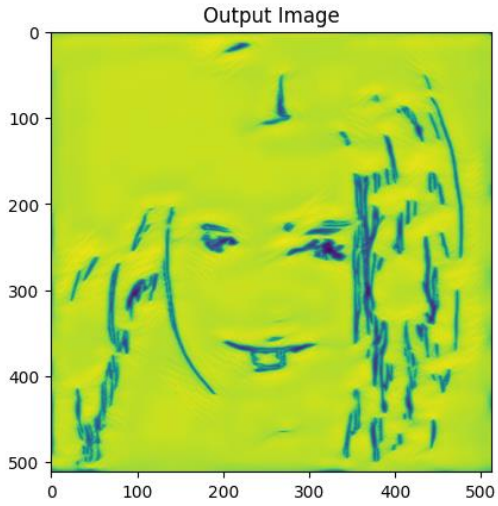
This part of the code calls the run_style_transfer function to perform style transfer and uses the imshow function to display the output image. plt.figure() is used to create a new image window, plt.ioff() is used to turn off the interactive mode, and plt.show() is used to display the image.

```
// Perform style transfer
output = run_style_transfer(cnn, cnn_normalization_mean,
cnn_normalization_std,
                            content_img, style_img, input_img)

plt.figure()
imshow(output, title='输出图像')

# sphinx_gallery_thumbnail_number = 4
plt.ioff()
plt.show()
```

Test results

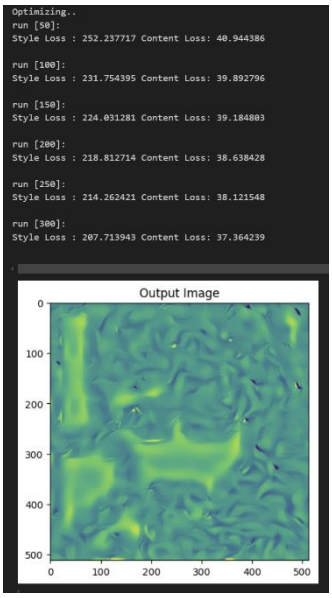


Testing process

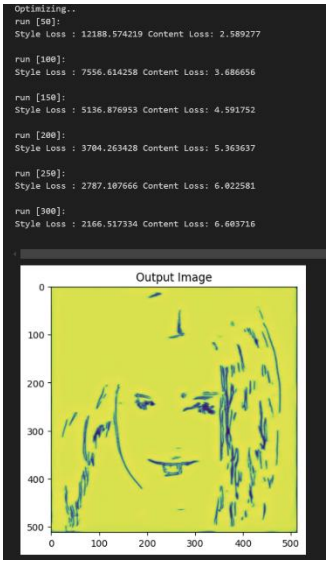
During the test, I used the following optimization methods to try different combinations and parameter settings to gradually improve the effect and performance of the model. Observe the results to find the optimal configuration.

Adjust the weights of style and content loss (adjusting the ratio of style_weight and content_weight can affect the effect of style transfer.)

```
output = run_style_transfer(cnn,
cnn_normalization_mean, cnn_normalization_std,
                            content_img,
style_img, input_img, num_steps=500,
                            style_weight=1e6,
content_weight=1)
```

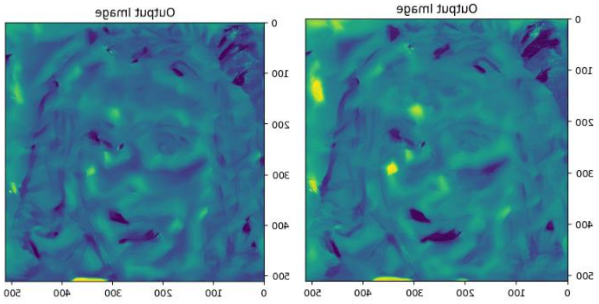


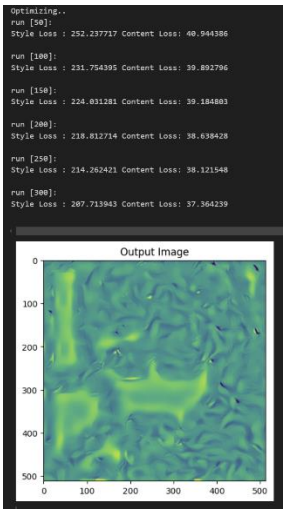
Try different optimizers and learning rates (I currently use the L-BFGS optimizer, and later tried using the Adam optimizer to adjust the learning rate, and the generated results are obviously more in line with my expectations)




```
def get_input_optimizer(input_img):
    optimizer = optim.Adam([input_img], lr=0.01)
    return optimizer
```

Other test results





Other ways to try tweaking

Use data augmentation and cropping techniques (data augmentation and cropping can improve the generalization ability and robustness of the model.)

```
transform = transforms.Compose([
    transforms.RandomResizedCrop(imsize),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

Use deeper feature maps (the default content and style levels currently used may not be enough, try extracting features at different levels.)

```
content_layers_default = ['conv_4']
style_layers_default = ['conv_1', 'conv_2', 'conv_3', 'conv_4',
'conv_5']
```