# Image Style Transfer论文复现报告

学号：2112193　姓名：王可一 专业：物联网工程

论文：Image Style Transfer Using Convolutional Neural Networks (cv-foundation.org)

代码仓库：0BZ0/StyleTransfer (github.com)

---


**Image Style Transfer论文复现报告**

一、论文原理

二、复现过程

（1）准备工作

（2）误差计算

1.内容误差

2.风格误差

（3）获取模型

（4）生成图像

三、复现结果

（1）原图像

内容图像

风格图像

（2）生成图像


# 一、论文原理

本文是基于CNN的图像风格迁移，主要过程则是通过对两幅图片分别进行拟合，即拟合前一幅图像的内容，又去拟合后一幅图像的风格。同时在最后将两者结合起来，这需要令优化目标既包含和内容图像的内容误差，又包含和风格图像的风格误差。最后再通过权重的修改，即可实现图像风格迁移。

# 二、复现过程

## （1）准备工作

先导入所需基本库，并设置运算设备。

```
import torch
import torch.nn.functional as F
import torch.optim as optim
import torchvision.models as models
import torchvision.transforms as transforms
from PIL import Image
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

对照片进行形状预处理，控制照片形状相同，便于正确计算误差。

```
img_size = (256, 512)

def read_image(image_path):
    pipeline = transforms.Compose(
        [transforms.Resize((img_size)),
         transforms.ToTensor()])

    img = Image.open(image_path).convert('RGB')
    img = pipeline(img).unsqueeze(0)
    return img.to(device, torch.float)
```

保存图像时，只需调用`PIL`的API即可。

```
def save_image(tensor, image_path):
    toPIL = transforms.ToPILImage()
    img = tensor.detach().cpu().clone()
    img = img.squeeze(0)
    img = toPIL(img)
    img.save(image_path)
```

## （2）误差计算

文章中使用的误差全部是基于MSE均方误差的，故可借助`torch.nn.Module`编写误差函数。

### 1.内容误差

编写内容误差。

```
class ContentLoss(torch.nn.Module):

    def __init__(self, target: torch.Tensor):
        super().__init__()
        self.target = target.detach()

    def forward(self, input):
        self.loss = F.mse_loss(input, self.target)
        return input
```

该类虽未进行运算，但缓存了内容误差值，后续可用于直接调用。

### 2.风格误差

编写`gram`矩阵的计算方法及风格误差的计算函数。

```
def gram(x: torch.Tensor):
    # x is a [n, c, h, w] array
    n, c, h, w = x.shape

    features = x.reshape(n * c, h * w)
    features = torch.mm(features, features.T) / n / c / h / w
    return features


class StyleLoss(torch.nn.Module):
```

```
    def __init__(self, target: torch.Tensor):
        super().__init__()
        self.target = gram(target.detach()).detach()

    def forward(self, input):
        G = gram(input)
        self.loss = F.mse_loss(G, self.target)
        return input
```

原理和上诉内容误差相似。

## （3）获取模型

构建模型第一层。

```
class Normalization(torch.nn.Module):

    def __init__(self, mean, std):
        super().__init__()
        self.mean = torch.tensor(mean).to(device).reshape(-1, 1, 1)
        self.std = torch.tensor(std).to(device).reshape(-1, 1, 1)

    def forward(self, img):
        return (img - self.mean) / self.std
```

之后借助`torchvision`中的预训练VGG模型，提取出所需模块并结合误差类的引用，最后得以计算最终误差。

```
default_content_layers = ['conv_4']
default_style_layers = ['conv_1', 'conv_2', 'conv_3', 'conv_4', 'conv_5']

def get_model_and_losses(content_img, style_img, content_layers, style_layers):
    num_loss = 0
    expected_num_loss = len(content_layers) + len(style_layers)
    content_losses = []
    style_losses = []

    model = torch.nn.Sequential(
        Normalization([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]))
    cnn = models.vgg19(weights=VGG19_Weights.DEFAULT).features.to(device).eval()
    i = 0
    for layer in cnn.children():
        if isinstance(layer, torch.nn.Conv2d):
            i += 1
            name = f'conv_{i}'
        elif isinstance(layer, torch.nn.ReLU):
            name = f'relu_{i}'
            layer = torch.nn.ReLU(inplace=False)
        elif isinstance(layer, torch.nn.MaxPool2d):
            name = f'pool_{i}'
        elif isinstance(layer, torch.nn.BatchNorm2d):
            name = f'bn_{i}'
        else:
```

```
            raise RuntimeError(
                f'Unrecognized layer: {layer.__class__.__name__}')

        model.add_module(name, layer)

        if name in content_layers:
            # add content loss:
            target = model(content_img)
            content_loss = ContentLoss(target)
            model.add_module(f'content_loss_{i}', content_loss)
            content_losses.append(content_loss)
            num_loss += 1

        if name in style_layers:
            target_feature = model(style_img)
            style_loss = StyleLoss(target_feature)
            model.add_module(f'style_loss_{i}', style_loss)
            style_losses.append(style_loss)
            num_loss += 1

        if num_loss >= expected_num_loss:
            break

    return model, content_losses, style_losses
```

## （4）生成图像

准备好输入的噪声图像、模型、误差类实例的引用，并设置好需要优化的参数。

```
input_img = torch.randn(1, 3, *img_size, device=device)
model, content_losses, style_losses = get_model_and_losses(
    content_img, style_img, default_content_layers, default_style_layers)

input_img.requires_grad_(True)
model.requires_grad_(False)
```

声明图片类型，是风格还是内容。

```
style_img = read_image('style.jpg')
content_img = read_image('context.jpg')
```

利用梯度下降实现图像生成，由于得知论文内容，可进行手动约束。

```
optimizer = optim.LBFGS([input_img])
steps = 0
prev_loss = 0
while steps <= 1000 and prev_loss < 100:

    def closure():
        with torch.no_grad():
            input_img.clamp_(0, 1)
        global steps
        global prev_loss
        optimizer.zero_grad()
```

```
        model(input_img)
        content_loss = 0
        style_loss = 0
        for l in content_losses:
            content_loss += l.loss
        for l in style_losses:
            style_loss += l.loss
        loss = content_weight * content_loss + style_weight * style_loss
        loss.backward()
        steps += 1
        if steps % 50 == 0:
            print(f'Step {steps}:')
            print(f'Loss: {loss}')
            # Open next line to save intermediate result
            # save_image(input_img, f'result/output_{steps}.jpg')
        prev_loss = loss
        return loss

    optimizer.step(closure)
```

最后，保存生成图像。

```
with torch.no_grad():
    input_img.clamp_(0, 1)
save_image(input_img, 'result/output.jpg')
```

# 三、复现结果

## （1）原图像

### 内容图像

**风格图像**

## （2）生成图像

当前状态为`style_weight/content_weight=1e6`

当前状态为`style_weight/content_weight=1e4`

比重越大，风格越明显，实验基本成功。