重构算法代码说明

1. 核心代码

1.1 超参数设定

使用argparse进行基本的超参数设定

```
    parser = argparse.ArgumentParser()
    parser.add_argument('--image_path', type=str,    # 输入的图片相对路径
                        default='./reference_images/my_face.jpg')
    parser.add_argument('--model', type=str, default='alexnet') # 使用的CNN模型
    parser.add_argument('--input_size', type=int, default=227)  # 输入的图像大小
    parser.add_argument('--layer_name', type=str,
                        default='avgpool')  # 选择反转特征的层名
    # 几个损失函数的超参数
    parser.add_argument('--alpha', type=float, default=6.0) # 图像先验正则化项超参
    parser.add_argument('--beta', type=float, default=2.0)  # 全变分正则化项超参
    parser.add_argument('--lambda_alpha', type=float, default=1e-5) # 图像先验正则化损失权重
    parser.add_argument('--lambda_tv', type=float, default=1e-5)  # 全变分正则化损失权重
    # 几个训练过程的超参数
    parser.add_argument('--epochs', type=int, default=200)  # 训练轮数，可能在100-200
    parser.add_argument('--lr', type=int, default=1e2)     # 学习率 1e2
    parser.add_argument('--momentum', type=float, default=0.9)  # SGD动量
    # 几个训练技巧
    parser.add_argument('--stdout_epoches', type=int, default=25) # 每次打印的间隔轮数
    parser.add_argument('--decay_factor', type=float, default=0.1)
    parser.add_argument('--decay_epochs', type=int, default=200)  # 每隔多少次lr进行衰减
    parser.add_argument('--device', type=str, default='cpu')
    parser.add_argument('--seed', type=int, default=2023)    # 随机种子
```

设定随机种子，包括了PyTorch、Python内置、CUDA相关的随机种子设置。

```
    # 设定种子
    random.seed(args.seed)
    torch.manual_seed(args.seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed(args.seed)
```

1.2 图形的转换和反转换

对于输入的原图，需要通过剪裁转换到Tensor类型，并进行归一化。

```
    # 进行图像预处理
    # 归一化参数
    # 大规模训练数据集的各个通道均值方差
    mu = [0.485, 0.456, 0.406]
    sigma = [0.229, 0.224, 0.225]

    # 输入原图的转换
    transform = transforms.Compose([
        transforms.Resize(size=input_size), # 修改图片大小
        transforms.CenterCrop(size=input_size), # 中心剪裁
        transforms.ToTensor(),       # 图片转换为张量
        transforms.Normalize(mu, sigma),     # 归一化
    ])

```

对于输出的图像，需要先反归一化，并剪裁像素值为0-1，转换成PIL.Image类，以便拥有更原始和完美的可视化效果。

```
    # 输出图像的转换
    # 需要进行反归一化，使得可视化效果能够更好
    def clip(tensor):
        return torch.clamp(tensor, 0, 1)

    # 反归一化
    detransform = transforms.Compose([
        transforms.Normalize(
            mean=[-m/s for m, s in zip(mu, sigma)],
            std=[1/s for s in sigma]),
        transforms.Lambda(clip),
        transforms.ToPILImage(),
    ])
```

1.3 模型的载入

给定模型的名称以及需要逆转的CNN层的名称，我们载入预训练的CNN模型。这里特别注意不能更新模型的梯度，所以使用eval()。

```
# 获取到具体的某一层layers
def get_module_layer(model: nn.Module, layer_name: str)->nn.Module:
```

```
    names = layer_name.split('_')
    if len(names) == 1:
        return model._modules.get(names[0])
    else:
        module = model
        for name in names:
            module = module._modules.get(name)
        return module

! def main()
    ...
    ...
    # 模型载入
    model = models.__dict__[model_name](pretrained=True)
    # 注意，这里不更新模型本身的梯度，我们只关心输入的白噪声图如何更新到目标图上去
    model.eval()
    model.to(device)

```

1.4 定义获取各输出层的钩子函数

在本算法中，最重要的是需要得到指定层的输出表征，PyTorch框架可以通过注册钩子函数hook，我们希望能够保存下来激活图（即该层输出的特征）。

```
    activations = []

    # TODO:考虑修改，是否不需要使用list
    def hook_activations(module, input, output):
        activations.append(output)

    def get_activations(model, input):
        del activations[:]
        _ = model(input)
        assert(len(activations) == 1)
        return activations[0]

    # 注册钩子函数，能够得到给定层的输出Tensor
    handle = get_module_layer(model, layer_name).register_forward_hook(hook_activations)

```

1.5 原始图、需要更新的白噪声图

这里尤其需要注意，白噪声图是nn.Parameter类，即可以用于反向传播更新的值。而整个训练过程有且仅有白噪声图可以被更新。

```
# 白噪声图，需要反传更新

```

```
    x = torch.nn.Parameter(1e-3 * torch.randn_like(ref_img, device=device), requires_grad=True)
    # 优化器：使用带动量的优化器，且优化目标仅为白噪声图
    optimizer = torch.optim.SGD([x], lr=lr, momentum=momentum)
    scheduler = lr_scheduler.StepLR(optimizer, step_size=decay_epochs, gamma=decay_factor)


    for i in tqdm(range(epochs), total=epochs):
        x_activations = get_activations(model, x)    # 获取需要更新的白噪声图在CNN该层的表征

        # 重建损失、图像先验正则化、TV正则化
        loss = rec_loss(x_activations, ref_actavations)
        reg_alpha = reg_img_prior(x, alpha)
        reg_tv = reg_TV(x, beta)
```

1.6 损失计算

首先定义一个图像范数值的框架。

```
# 计算图像范数
def img_norm(x, alpha=2.0):
    return torch.abs(x ** alpha).sum()
```

基于这个范数值，我们可以定义计算图像正则化的α损失

```
# 计算图像正则化
def reg_img_prior(x, alpha=6.0):
    return img_norm(x, alpha)
```

此外，还需要计算全变分正则化损失，具体而言就是从x、y两个方向进行相邻像素差分，计算二次范数，并对每一个像素位差分值求和。

```
# 计算全变分TV_norm
def reg_TV(x, beta=2.0):
    assert(x.size(0) == 1)
    image = x[0]
    dy = torch.zeros_like(image)
    dx = torch.zeros_like(image)
    dy[:, 1:, :] = image[:, :-1, :] - image[:, 1:, :]
    dx[:, :, 1:] = image[:, :, 1:] - image[:, :, :-1]
    return ((dx ** 2 + dy ** 2) ** (beta / 2.0)).sum()

```

此外，基于上述图像范数值的框架，需要计算归一化的重建损失：

```python
# 计算归一化的重建损失
def rec_loss(x, ref_img):
    return img_norm(x-ref_img, alpha=2.0) / img_norm(ref_img, alpha=2.0)
```

2. 预训练模型

本代码中所用的默认预训练模型是从torchvision库中的AlexNet。下图是AlexNet的各层模块名及其结构。

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```