TransNet

陈乐偲

神经网络风格迁移

由特征提取网络和风格转换网络构成。

特征提取网络基于预训练的vgg模型，图像转换网络由编码器和解码器组成。

损失函数由内容损失和风格损失组成，内容损失为特征提取网络激活层的响应衡量，风格损失由特征提取网络激活层的通道Gram矩阵衡量。

定义辅助函数

```
#得到gram相似度矩阵
def gram_matrix(y):
    (b, ch, h, w) = y.size()
    features = y.view(b, ch, w * h)
    features_t = features.transpose(1, 2)
    #对特征及其转置做batch上的矩阵乘法
    gram = t.bmm(features, features_t) / (ch * h * w)
    #计算channel之间的相似度
    return gram

#输入特征提取网络之前先做batch归一化
def batch_normalize(batch):
    mean = batch.data.new(IMAGENET_MEAN).view(1, -1, 1, 1)
    std = batch.data.new(IMAGENET_STD).view(1, -1, 1, 1)
    return (batch - mean) / std
```

```
# REF: https://github.com/chenyuntc/pytorch-book/blob/master/chapter08-
neural_style/transformer_net.py

#图像改变网络
class TransformerNet(nn.Module):
    def __init__(self):
        super().__init__()

        #下采样
        self.downsample_layers = nn.Sequential(
            ConvLayer(3, 32, kernel_size=9, stride=1),
            #使用instanceNorm,对每个通道的WH做归一化
```

```
            nn.InstanceNorm2d(32, affine=True),
            nn.ReLU(True),
            ConvLayer(32, 64, kernel_size=3, stride=2),
            nn.InstanceNorm2d(64, affine=True),
            nn.ReLU(True),
            ConvLayer(64, 128, kernel_size=3, stride=2),
            nn.InstanceNorm2d(128, affine=True),
            nn.ReLU(True),
        )

        #残差层
        self.res_layers = nn.Sequential(
            ResidualBlock(128),
            ResidualBlock(128),
            ResidualBlock(128),
            ResidualBlock(128),
            ResidualBlock(128)
        )

        #上采样层
        self.upsample_layers = nn.Sequential(
            UpsampleConvLayer(128, 64, kernel_size=3, stride=1, upsample=2),
            nn.InstanceNorm2d(64, affine=True),
            nn.ReLU(True),
            UpsampleConvLayer(64, 32, kernel_size=3, stride=1, upsample=2),
            nn.InstanceNorm2d(32, affine=True),
            nn.ReLU(True),
            ConvLayer(32, 3, kernel_size=9, stride=1)
        )

    def forward(self, x):
        x = self.downsample_layers(x)
        x = self.res_layers(x)
        x = self.upsample_layers(x)
        return t.sigmoid(x)

#定义一些辅助单元

#卷积单元
class ConvLayer(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride):
        super().__init__()
        reflection_padding = int(np.floor(kernel_size / 2))
        #采用反射填充
        self.reflection_pad = nn.ReflectionPad2d(reflection_padding)
        self.conv2d = nn.Conv2d(in_channels, out_channels, kernel_size, stride)

    def forward(self, x):
        out = self.reflection_pad(x)
        out = self.conv2d(out)
        return out

#用上采样加卷积作为反卷积的代替
class UpsampleConvLayer(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride, upsample=None):
        super().__init__()
        self.upsample = upsample
        reflection_padding = int(np.floor(kernel_size / 2))
        self.reflection_pad = nn.ReflectionPad2d(reflection_padding)
        self.conv2d = nn.Conv2d(in_channels, out_channels, kernel_size, stride)
```

```
    def forward(self, x):
        x_in = x
        #x_in = t.nn.functional.interpolate(x_in, scale_factor=self.upsample)
        out = self.reflection_pad(x_in)
        out = self.conv2d(out)
        return out

#resnet中的残差单元，同样使用instanceNorm
class ResidualBlock(nn.Module):
    def __init__(self, channels):
        super(ResidualBlock, self).__init__()
        self.conv1 = ConvLayer(channels, channels, kernel_size=3, stride=1)
        self.in1 = nn.InstanceNorm2d(channels, affine=True)
        self.conv2 = ConvLayer(channels, channels, kernel_size=3, stride=1)
        self.in2 = nn.InstanceNorm2d(channels, affine=True)
        self.relu = nn.ReLU()

    def forward(self, x):
        residual = x
        out = self.relu(self.in1(self.conv1(x)))
        out = self.in2(self.conv2(out))
        out = out + residual
        return out

#用VGG为backbone作特征提取网络
class FeatureNet(nn.Module):
    def __init__(self):
        super().__init__()
        #采用VGG作为预训练的模型，取前23层作为特征提取器
        features = list(vgg16(pretrained=True).features)[:23]
        self.features = nn.ModuleList(features).eval()

    def forward(self, x):
        results = []
        for i, model in enumerate(self.features):
            x = model(x)
            #提取中间四层作为特征
            if i in {3, 8, 15, 22}:
                results.append(x)

        vgg_outputs = namedtuple("VggOutputs", ['relu1', 'relu2', 'relu3', 'relu4'])
        return vgg_outputs(*results)
```

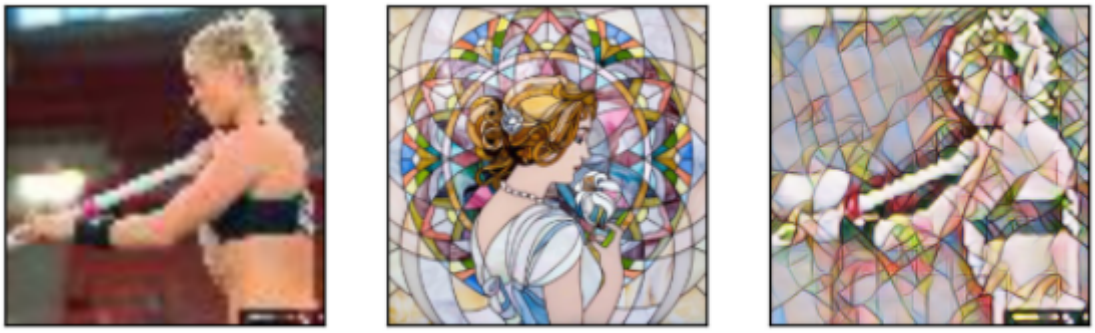
主程序

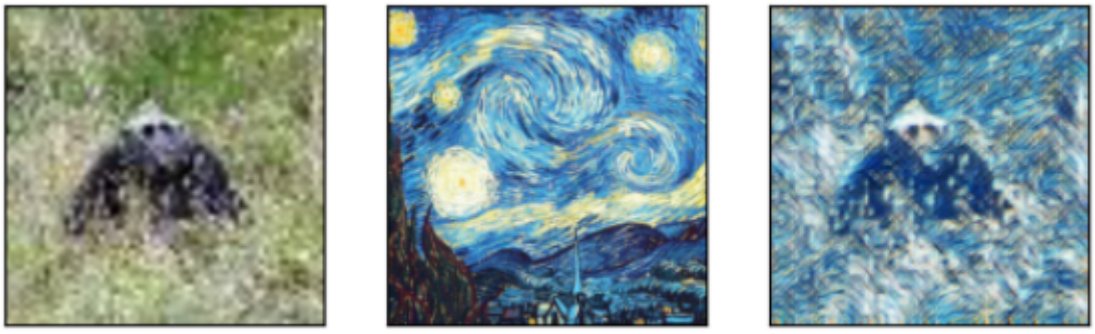
```
device=t.device('cuda') if t.cuda.is_available() else t.device('cpu')

class TransModel():
    def __init__(self):
        self.transformer = TransformerNet().to(device)
        self.extracter = FeatureNet().eval().to(device)
        self.lr = 1e-3
        self.optimizer = t.optim.Adam(self.transformer.parameters(), self.lr)
        self.content_weight = 1e5
        self.style_weight = 1e9
        self.epoches = 10
```

```
    def train(self,dataloader,style):
        with t.no_grad():
            features_style = self.extracter(style)
            gram_style = [gram_matrix(y) for y in features_style]
            #Bx64x64 的channel相似度矩阵

        for epoch in range(self.epoches):
            for i, (x, _) in tqdm.tqdm(enumerate(dataloader)):

                self.optimizer.zero_grad()

                x = x.to(device)
                y = self.transformer(x)

                if i > 100:
                    break
                #y = batch_normalize(y)
                #x = batch_normalize(x)
                features_y = self.extracter(y)
                features_x = self.extracter(x)

                #使用relu2的值计算内容的损失
                content_loss = self.content_weight * F.mse_loss(features_y.relu2,
features_x.relu2)

                gram_y = [gram_matrix(y) for y in features_y]
                style_loss = 0
                for i in range(len(gram_y)):
                    style_loss += F.mse_loss(gram_y[i], gram_style[i].expand_as(gram_y[i]))
                style_loss =  self.style_weight * style_loss
                #print(style_loss)

                loss = content_loss + style_loss
                loss.backward()
                self.optimizer.step()

            if epoch % 1 == 0:
                plt.figure()

                origin_img = x.data.cpu()[1].permute(1,2,0)
                style_img = style.cpu()[0].permute(1,2,0)
                new_img  = y.data.cpu()[1].permute(1,2,0)

                plt.subplot(131)
                plt.imshow(origin_img)
                plt.xticks([]),plt.yticks([])
                plt.subplot(132)
                plt.imshow(style_img)
                plt.xticks([]),plt.yticks([])
                plt.subplot(133)
                plt.imshow(new_img)
                plt.xticks([]),plt.yticks([])

                plt.savefig('./dump/' + str(epoch) +'.png')
                plt.close()
                #path = './dump/' + str(epoch) +'.png'
                #tv.utils.save_image(y.data.cpu()[0].clamp(min=0, max=1), path)


    #def test(self,content,save_path):
```

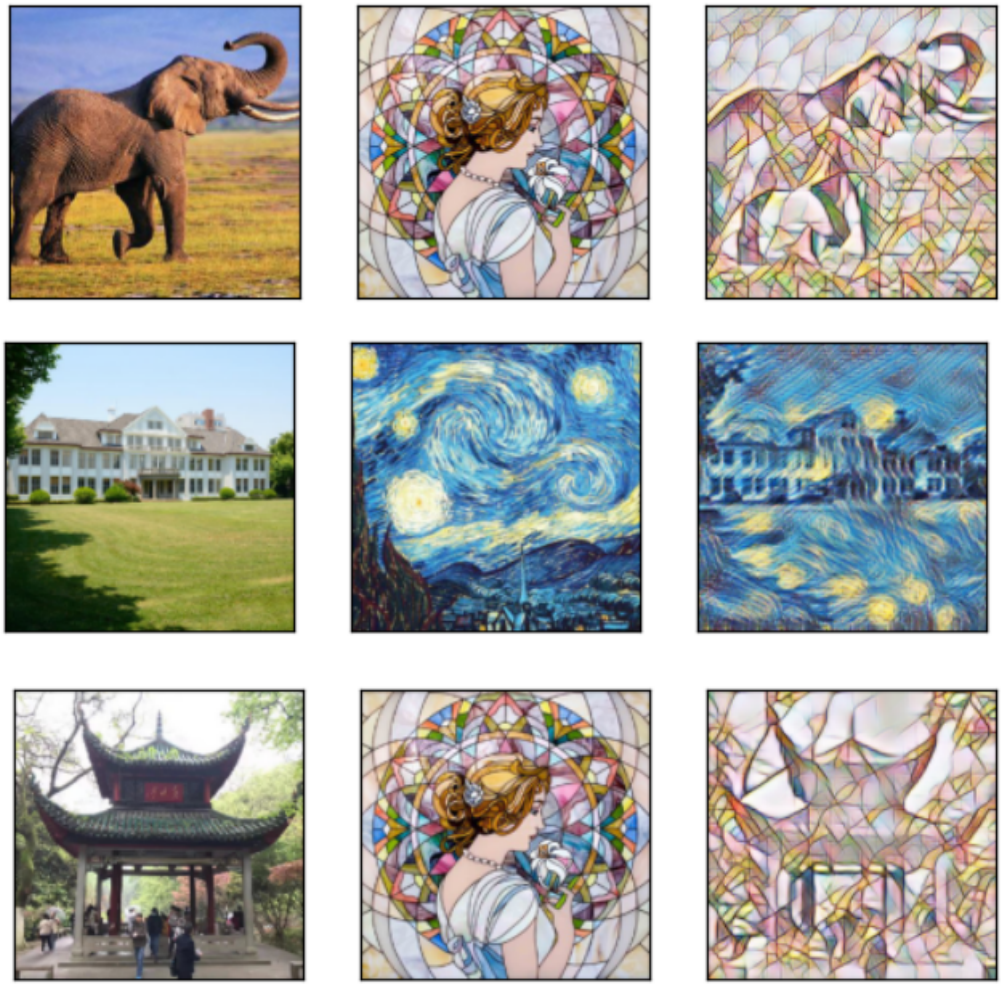
```
    #     output = self.transformer(content)


img_size = 256
img_mean = [0.485, 0.456, 0.406]
img_std  = [0.229, 0.224, 0.225]
myTransform = tv.transforms.Compose([
        tv.transforms.ToTensor(),
        tv.transforms.Resize(img_size),
        tv.transforms.CenterCrop(img_size),
        #tv.transforms.Normalize(mean=img_mean, std=img_std),
    ])

#使用imagenet的测试集作为数据集,共10000张图片
dataset = tv.datasets.ImageFolder("./tiny-imagenet-200/test", myTransform)
dataloader = DataLoader(dataset, 8)

style_image = tv.datasets.folder.default_loader('style1.jpg')
style = myTransform(style_image).unsqueeze(0).to(device)

content_image = tv.datasets.folder.default_loader('content.jpg')
content = myTransform(style_image).unsqueeze(0).to(device)

trans = TransModel()
trans.train(dataloader,style)
```

训练过程中的中间结果





结果图如下



NOTE:图二为复旦大学子彬院，图三为岳麓山爱晚亭