StyTr2: Image Style Transfer with Transformers

1 Abstract and Contribution

图像风格迁移目的是为了渲染一张艺术特征风格的图像，并保持原先的内容。由于CNN的局部性，想要提取输入图像的全局信息很难。因此，传统的神经风格迁移方法会存在有偏的内容表示。为了解决这个问题，我们提出一种StyTr2的方法，在图像风格迁移中考虑输入图像的长期依赖性。与ViT的其他视觉任务不同，StyTr2包含两种不同的Transformer编码器去分别生成内容和风格特定域的序列。在编码器后面连着一个多层transformer解码器，用于通过style序列去风格化内容序列。我们也分析现有位置编码的不足，并提出了内容感知的位置编码（CAPE）,这是尺度不变的且更适合风格迁移任务。通过与sota的基于CNN的方法进行定量和定性的比较实验验证了本文提出的StyTr2方法的有效性。

本文的贡献：

1. 一种基于Transformer的风格迁移框架，StyTr2，可以生成保存良好的风格化结果以及输入内容图像的细节。

2. 一种内容感知的位置编码，具有尺度不变性并且适用于风格迁移任务。

3. 全面的实验表面StyTr2超过了baseline的方法，并且以理想的内容结构和风格模式获得出色的结果。

2 code remark

```python
# ========================================= 输入参数 =========================================
parser = argparse.ArgumentParser()
# Basic options
parser.add_argument('--content_dir', default='./datasets/train2014', type=str,
                    help='Directory path to a batch of content images')
parser.add_argument('--style_dir', default='./datasets/Images', type=str,  #wikiart dataset crawled from https://www.wikia
                    help='Directory path to a batch of style images')
parser.add_argument('--vgg', type=str, default='./experiments/vgg_normalised.pth')  #run the train.py, please download the

# training options
parser.add_argument('--save_dir', default='./experiments',
                    help='Directory to save the model')
parser.add_argument('--log_dir', default='./logs',
                    help='Directory to save the log')
parser.add_argument('--lr', type=float, default=5e-4)
parser.add_argument('--lr_decay', type=float, default=1e-5)
parser.add_argument('--max_iter', type=int, default=160000)
parser.add_argument('--batch_size', type=int, default=8)
parser.add_argument('--style_weight', type=float, default=10.0)
parser.add_argument('--content_weight', type=float, default=7.0)
parser.add_argument('--n_threads', type=int, default=16)
parser.add_argument('--save_model_interval', type=int, default=10000)
parser.add_argument('--position_embedding', default='sine', type=str, choices=('sine', 'learned'),
                    help="Type of positional embedding to use on top of the image features")
parser.add_argument('--hidden_dim', default=512, type=int,
                    help="Size of the embeddings (dimension of the transformer)")
args = parser.parse_args()


# 初始化device和日志文件
USE_CUDA = torch.cuda.is_available()
device = torch.device("cuda:0" if USE_CUDA else "cpu")

if not os.path.exists(args.save_dir):
    os.makedirs(args.save_dir)

if not os.path.exists(args.log_dir):
    os.mkdir(args.log_dir)
writer = SummaryWriter(log_dir=args.log_dir)

# 初始化网络，用的是vgg
vgg = StyTR.vgg
vgg.load_state_dict(torch.load(args.vgg))
vgg = nn.Sequential(*list(vgg.children())[:44])

# 初始化decoder
decoder = StyTR.decoder
embedding = StyTR.PatchEmbed()
```

图 1: remark 1

```python
# 初始化transformer
Trans = transformer.Transformer()
with torch.no_grad():
    network = StyTR.StyTrans(vgg,decoder,embedding, Trans,args)
network.train()

network.to(device)
network = nn.DataParallel(network, device_ids=[0])

# 构建内容网络和风格化网络
content_tf = train_transform()
style_tf = train_transform()


# 载入数据集，优化器
content_dataset = FlatFolderDataset(args.content_dir, content_tf)
style_dataset = FlatFolderDataset(args.style_dir, style_tf)

content_iter = iter(data.DataLoader(
    content_dataset, batch_size=args.batch_size,
    sampler=InfiniteSamplerWrapper(content_dataset),
    num_workers=args.n_threads))
style_iter = iter(data.DataLoader(
    style_dataset, batch_size=args.batch_size,
    sampler=InfiniteSamplerWrapper(style_dataset),
    num_workers=args.n_threads))


optimizer = torch.optim.Adam([
                              {'params': network.module.transformer.parameters()},
                              {'params': network.module.decode.parameters()},
                              {'params': network.module.embedding.parameters()},
                              ], lr=args.lr)


if not os.path.exists(args.save_dir+"/test"):
    os.makedirs(args.save_dir+"/test")



for i in tqdm(range(args.max_iter)):
    # 学习率调整策略
    if i < 1e4:
        warmup_learning_rate(optimizer, iteration_count=i)
    else:
        adjust_learning_rate(optimizer, iteration_count=i)

    # print('learning_rate: %s' % str(optimizer.param_groups[0]['lr']))
    # 获得batch数据，输入网络并计算损失。
    content_images = next(content_iter).to(device)
    style_images = next(style_iter).to(device)
    out, loss_c, loss_s,l_identity1, l_identity2 = network(content_images, style_images)
```

图 2: remark 2

```
style_images = next(style_iter).to(device)
out, loss_c, loss_s,l_identity1, l_identity2 = network(content_images, style_images)

# 输出图片保存
if i % 100 == 0:
    output_name = '{:s}/test/{:s}{:s}'.format(
                  args.save_dir, str(i),".jpg"
                )
    out = torch.cat((content_images,out),0)
    out = torch.cat((style_images,out),0)
    save_image(out, output_name)

# 计算损失
loss_c = args.content_weight * loss_c
loss_s = args.style_weight * loss_s
loss = loss_c + loss_s + (l_identity1 * 70) + (l_identity2 * 1)

print(loss.sum().cpu().detach().numpy(),"-content:",loss_c.sum().cpu().detach().numpy(),"-style:",loss_s.sum().cpu().d
          ,"-l1:",l_identity1.sum().cpu().detach().numpy(),"-l2:",l_identity2.sum().cpu().detach().numpy()
          )

# 梯度更新
optimizer.zero_grad()
loss.sum().backward()
optimizer.step()


# 打印日志
writer.add_scalar('loss_content', loss_c.sum().item(), i + 1)
writer.add_scalar('loss_style', loss_s.sum().item(), i + 1)
writer.add_scalar('loss_identity1', l_identity1.sum().item(), i + 1)
writer.add_scalar('loss_identity2', l_identity2.sum().item(), i + 1)
writer.add_scalar('total_loss', loss.sum().item(), i + 1)

# 保存模型
if (i + 1) % args.save_model_interval == 0 or (i + 1) == args.max_iter:
    state_dict = network.module.transformer.state_dict()
    for key in state_dict.keys():
        state_dict[key] = state_dict[key].to(torch.device('cpu'))
    torch.save(state_dict,
               '{:s}/transformer_iter_{:d}.pth'.format(args.save_dir,
                                                       i + 1))

    state_dict = network.module.decode.state_dict()
    for key in state_dict.keys():
        state_dict[key] = state_dict[key].to(torch.device('cpu'))
    torch.save(state_dict,
               '{:s}/decoder_iter_{:d}.pth'.format(args.save_dir,
                                                   i + 1))
    state_dict = network.module.embedding.state_dict()
    for key in state_dict.keys():
        state_dict[key] = state_dict[key].to(torch.device('cpu'))
    torch.save(state_dict,
               '{:s}/embedding_iter_{:d}.pth'.format(args.save_dir,
                                                     i + 1))
```

图 3: remark 3

```
def forward(self, samples_c: NestedTensor,samples_s: NestedTensor):
    """ The forward expects a NestedTensor, which consists of:
           - samples.tensor: batched images, of shape [batch_size x 3 x H x W]
           - samples.mask: a binary mask of shape [batch_size x H x W], containing 1 on padded pixels

    """
    content_input = samples_c
    style_input = samples_s
    if isinstance(samples_c, (list, torch.Tensor)):
        samples_c = nested_tensor_from_tensor_list(samples_c)   # support different-sized images padding is used for m
    if isinstance(samples_s, (list, torch.Tensor)):
        samples_s = nested_tensor_from_tensor_list(samples_s)

    # ### features used to calcate loss
    # vgg特征层的输出
    content_feats = self.encode_with_intermediate(samples_c.tensors)
    style_feats = self.encode_with_intermediate(samples_s.tensors)

    ### Linear projection
    # 得到风格和内容的patch编码
    style = self.embedding(samples_s.tensors)
    content = self.embedding(samples_c.tensors)

    # postional embedding is calculated in transformer.py
    pos_s = None
    pos_c = None

    mask = None

    #得到风格和图像经过transformer编码并解码的部分的部分
    hs = self.transformer(style, mask , content, pos_c, pos_s)
    # transformer解码后的部分进一步解码
    Ics = self.decode(hs)

    # vgg特征层的输出
    Ics_feats = self.encode_with_intermediate(Ics)
    # 内容mse损失
    loss_c = self.calc_content_loss(normal(Ics_feats[-1]), normal(content_feats[-1]))+self.calc_content_loss(normal(Ic
    # Style loss
    # 风格mse损失
    loss_s = self.calc_style_loss(Ics_feats[0], style_feats[0])
    for i in range(1, 5):
        loss_s += self.calc_style_loss(Ics_feats[i], style_feats[i])


    Icc = self.decode(self.transformer(content, mask , content, pos_c, pos_c))
    Iss = self.decode(self.transformer(style, mask , style, pos_s, pos_s))
```

图 4: remark 4

```
# 内容的身份损失 lambda 1
loss_lambda1 = self.calc_content_loss(Icc,content_input)+self.calc_content_loss(Iss,style_input)

# 内容的身份损失 lambda 2
Icc_feats=self.encode_with_intermediate(Icc)
Iss_feats=self.encode_with_intermediate(Iss)
loss_lambda2 = self.calc_content_loss(Icc_feats[0], content_feats[0])+self.calc_content_loss(Iss_feats[0], style_f
for i in range(1, 5):
    loss_lambda2 += self.calc_content_loss(Icc_feats[i], content_feats[i])+self.calc_content_loss(Iss_feats[i], st
# Please select and comment out one of the following two sentences
return Ics,  loss_c, loss_s, loss_lambda1, loss_lambda2   #train
# return Ics    #test
```

图 5: remark 5

图 6: content

图 7: style 1

图 8: transfer 1

图 9: style 2

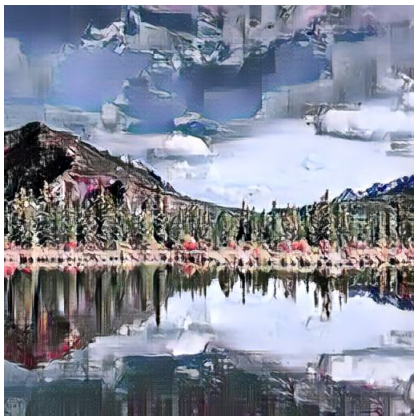

图 10: transfer 2