1. 介绍

1.1背景

随着互联网的发展，艺术作品乃至博物馆、艺术馆的数字化受到越来越多的关注。线上的艺术作品图片可以让收藏者、艺术品爱好者等更便捷的欣赏多种多样的艺术作品。但是大量艺术作品的数字化需要耗费非常多人力，每幅作品的作者、创作日期等的登记十分繁杂。所以荷兰的*Rijksmuseum*博物馆发布了*Rijksmuseum Challenge*和相应数据集[1]，希望挑战者根据艺术作品的图片预测艺术作品的创作者、类别、材料和创作日期。所以本项目选取其中一个挑战，希望基于CNN网络实现一个画作分类器，较为准确的判断出画作的创作者。

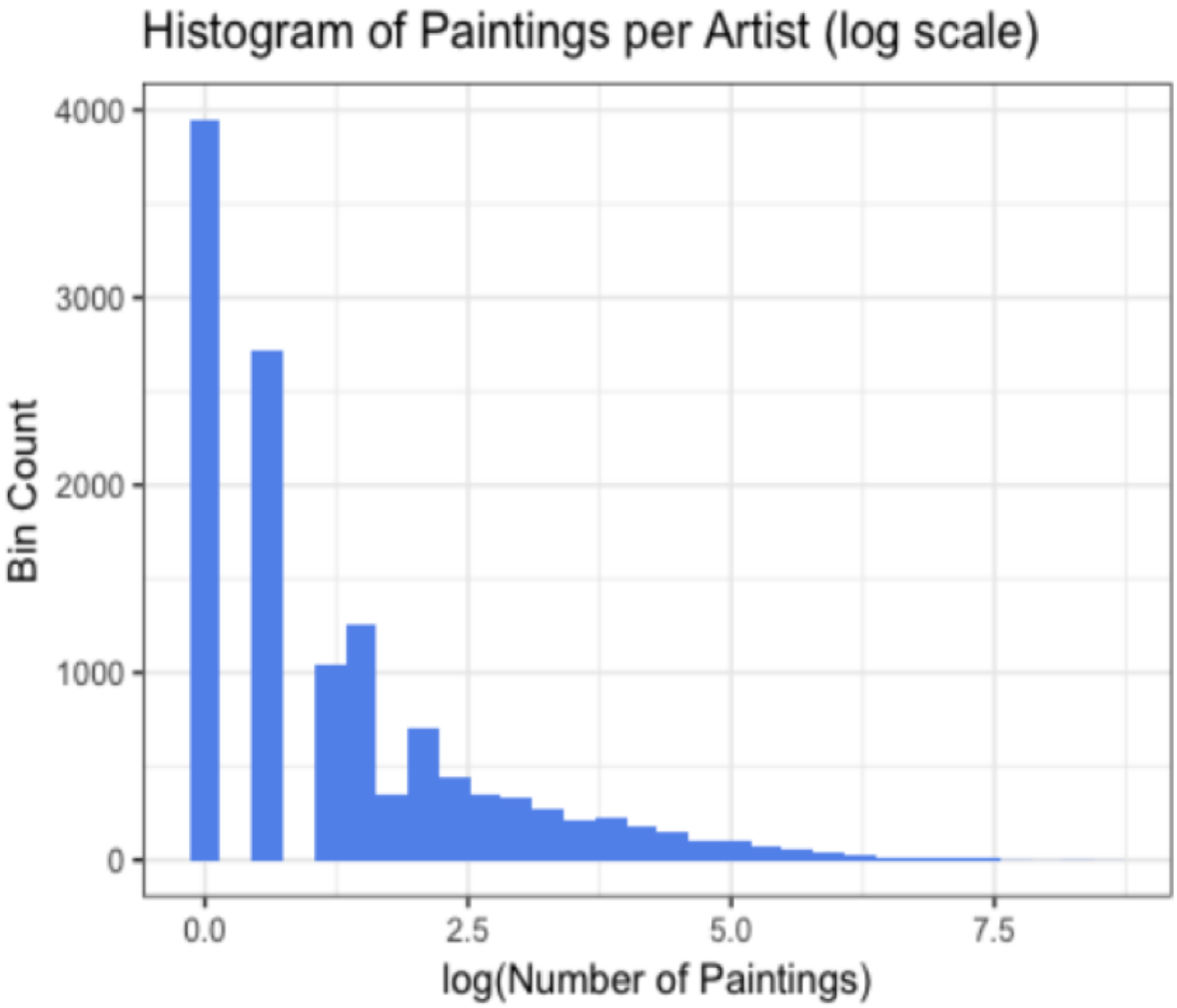
1.2 问题描述

问题分为两部分：

1. 基于CNN对画作进行识别与分类
    - 输入：一张艺术作品的图片
    - 输出：预测的创作者
2. 探究模型如何进行识别与分类

1.3 数据集

数据集来自*Rijksmuseum Challenge*中发布的数据集[2]，包含来自12641位作者的112039张艺术作品图片。但对数据分析后发现，其中21位创作者拥有超过1000张图片，而3949位创作者仅有1张图片，创作者和创作的图片数量是一个长尾分布。



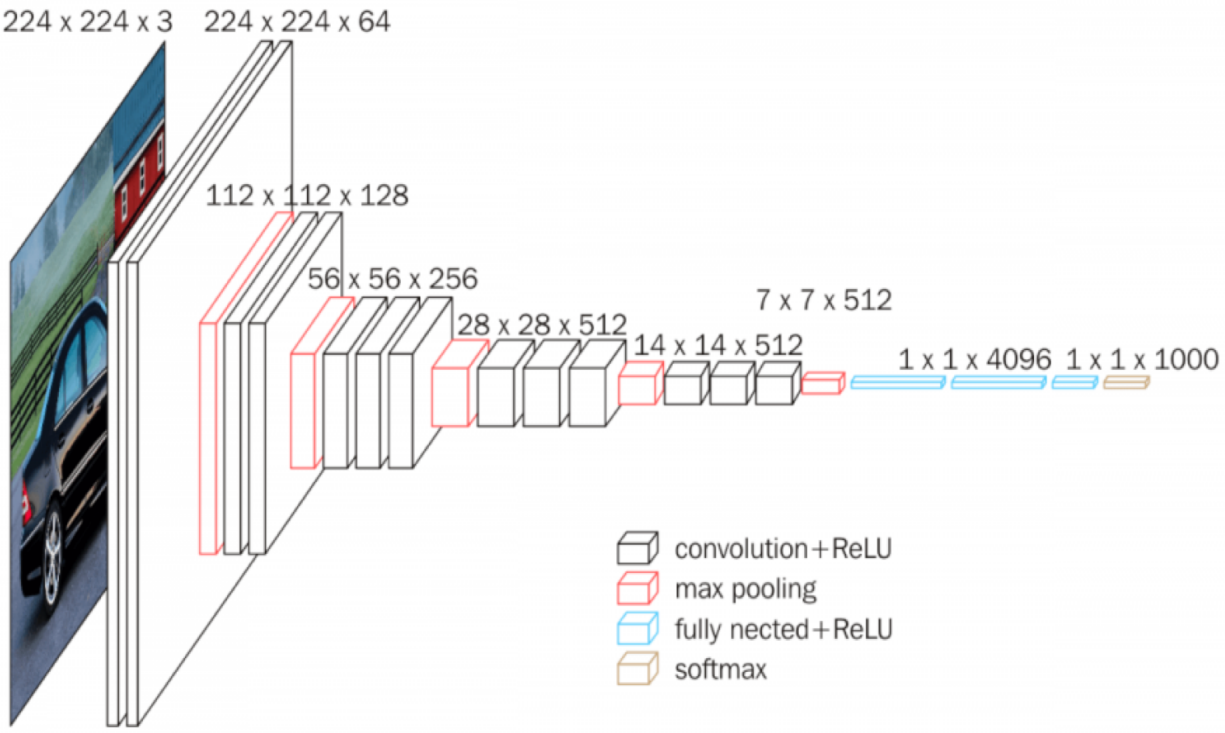
所以本项目仅选取作品数量最多的10位作者的17513张图片，George Hendrik Breitner, Jan Luyken, Reiner Vinkeles, Marius Bauer, Isaac Israels, Bernard Picart, Rembrandt Harmensz.Van Rijn, Johannes Tavenraat, Willem Witsen, Simon Fokke，对模型进行训练。

2. 模型与方法

2.1 实现画作分类

相对于用于训练大型网络的著名数据集，如ImageNet，*Rijksmuseum*数据量及类别都非常少，完全从头搭建并训练一个网络成本较高且很可能会导致过拟合，所以我采用了已搭建好的网络进行迁移学习，迁移学习分为特征提取和参数微调两步。特征提取是指保留原模型绝大部分参数，仅训练全连接层的参数。参数微调是从头训练模型中的所有参数。

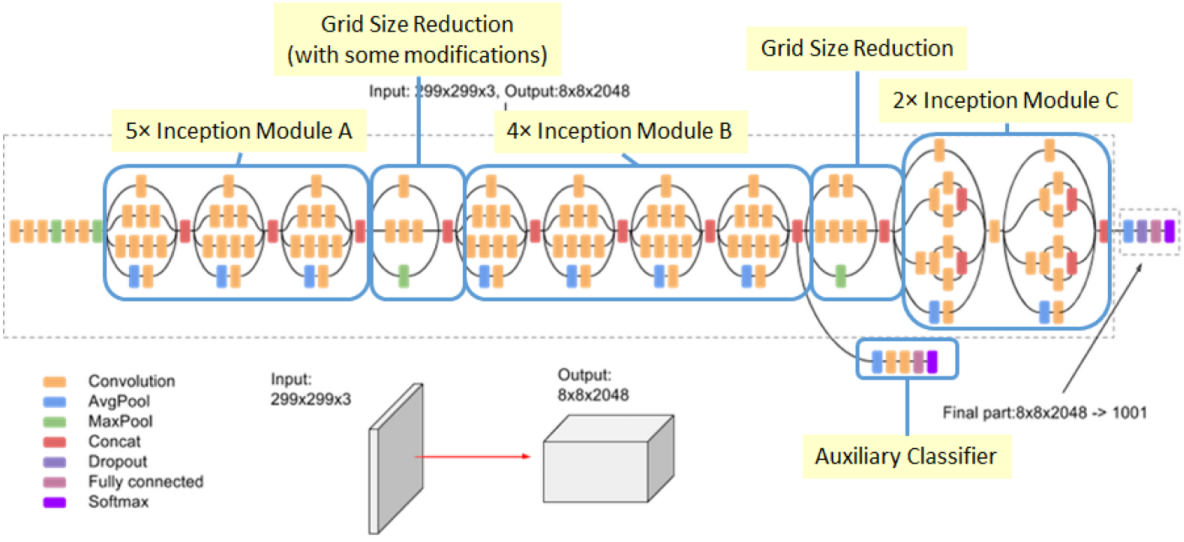
本实验采用VGG16[3], ResNet18[4]和Inception V3[5]三个著名模型的架构。VGG16的特征是小的filter，由16层3x3的卷积层，2x2的池化层和3个全连接层组成，结构如下



ResNet18的特征是batch normalization和skip connections，由于没有中间的全连接层，参数远远少于VGG，所以更易于训练。

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	$7 \times 7, 64$, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2 $\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

Inception V3的结构如下，



首先对三个模型进行特征提取训练并观察结果，即对三个模型的全连接层参数进行训练。然后选取训练速度较快，结果比较好的ResNet18模型从头训练，即利用本数据集训练ResNet18的所有参数。

2.2 探究分类结果

本项目将通过风格迁移对分类结果的探究来理解分类的结果，Style transfer works by minimizing a style loss with respect to the styling image and a content loss with respect to a content image.[9] 为了理解照片的内容和风格哪个对分类结果的影响更大将尝试将A图片的风格迁移到B图片后利用训练好的网络进行分类，查看分类结果是A的创作者或B的创作者或其他创作者。

3. 实验与结果

实验环境

所有实验都在Google Colaboratory中利用远程GPU运行，具体配置及参数如下。

```
[ ]  1 #check the version of Cuda and python
     2 !nvcc --version
     3 !python --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2018 NVIDIA Corporation
Built on Sat_Aug_25_21:08:01_CDT_2018
Cuda compilation tools, release 10.0, V10.0.130
Python 3.6.8
```

command: https://pytorch.org/

```
[ ]  1 !pip3 install https://download.pytorch.org/whl/cu100/torch-1.1.0-cp36-cp36m-linux_x86_64.whl
     2 !pip3 install https://download.pytorch.org/whl/cu100/torchvision-0.3.0-cp36-cp36m-linux_x86_64.whl
```

```
Requirement already satisfied: torch==1.1.0 from https://download.pytorch.org/whl/cu100/torch-1.1.0-cp36-cp36m-linux_x86_(
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from torch==1.1.0) (1.16.4)
Requirement already satisfied: torchvision==0.3.0 from https://download.pytorch.org/whl/cu100/torchvision-0.3.0-cp36-cp36(
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from torchvision==0.3.0) (1.16.4)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.6/dist-packages (from torchvision==0.3.0) (4.3.0)
Requirement already satisfied: torch>=1.1.0 in /usr/local/lib/python3.6/dist-packages (from torchvision==0.3.0) (1.1.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from torchvision==0.3.0) (1.12.0)
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist-packages (from pillow>=4.1.1->torchvision==0.3.0)
```

```
[ ]  1 torch.cuda.get_device_name(0)
```

```
'Tesla T4'
```

3.1 实现画作分类

实验过程

数据预处理

数据分为训练集、验证集和测试集，分别包括12454，4570、489张图片

基于VGG和ResNet对输入图片的要求，先将图片随机裁剪为224x224大小，减去均值，不进行水平翻转。

```
IMAGENET_MEAN = [0.485, 0.456, 0.406]
IMAGENET_STD = [0.229, 0.224, 0.225]

train_transform = T.Compose([
T.Scale(256),
T.RandomSizedCrop(224),
T.RandomHorizontalFlip(),
T.ToTensor(),
T.Normalize(mean=IMAGENET_MEAN, std=IMAGENET_STD),
])
```

载入数据

```
data_transforms = {
    'train': transforms.Compose([
        # Data augmentation is a good practice for the train set
        # Here, we randomly crop the image to 224x224 and
        # randomly flip it horizontally.
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
    ]),
    'val': transforms.Compose([
        transforms.Resize(224),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
    ]),
    'test': transforms.Compose([
        transforms.Resize(224),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
    ])
}

train_dir = 'gdrive/My Drive/201906AI_FinalProject/data_all/images/train'
val_dir = 'gdrive/My Drive/201906AI_FinalProject/data_all/images/val'
test_dir =  'gdrive/My Drive/201906AI_FinalProject/data_all/images/test'

train_dset =  datasets.ImageFolder(train_dir,
transform=data_transforms['train'])
train_loader = torch.utils.data.DataLoader(train_dset,
                     batch_size=batch_size,
                     num_workers=num_workers,
                     shuffle=True)

val_dset =  datasets.ImageFolder(val_dir,
transform=data_transforms['val'])
val_loader =torch.utils.data.DataLoader(val_dset,
```

```
                  batch_size=batch_size,
                  num_workers=num_workers)

test_dset =  datasets.ImageFolder(test_dir, transform=data_transforms['test'])
test_loader = torch.utils.data.DataLoader(test_dset,
                     batch_size=batch_size,
                     num_workers=num_workers,
                     shuffle=True)

image_datasets = {'train': train_dset, 'val':val_dset, 'test': test_dset}
dataloaders_dict = {'train': train_loader, 'val':val_loader, 'test':test_loader}


dataset_sizes = {x: len(image_datasets[x]) for x in ['train','val', 'test']}

for x in ['train', 'val', 'test']:
    print("Loaded {} images under {}".format(dataset_sizes[x], x))

print("Classes: ")
class_names = image_datasets['train'].classes
print(image_datasets['train'].classes)
```

初始化模型

```
def initialize_model(model_name, num_classes, feature_extract, use_pretrained=True):
    model_ft = None
    input_size = 0

    model_ft = models.resnet18(pretrained=use_pretrained)
    set_parameter_requires_grad(model_ft, feature_extract)
    num_ftrs = model_ft.fc.in_features
    model_ft.fc = nn.Linear(num_ftrs, num_classes)
    input_size = 224

    return model_ft, input_size

model_ft, input_size = initialize_model(model_name, num_classes, feature_extract, use_pretrained=True)

print(model_ft)

# Send the model to GPU
model_ft = model_ft.to(device)
```

feature_extracting=True时，只训练全连接层

feature_extracting=False时，重新训练所有参数

```
def set_parameter_requires_grad(model, feature_extracting):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False
```

训练模型

```
def train_model(model, dataloaders, criterion, optimizer, num_epochs=25, is_inception=False):
    since = time.time()

    val_acc_history = []

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    model_save_name = f"{num_epochs}.pt"
    PATH= f"gdrive/My Drive/201906AI_FinalProject/checkpoints_resnet18/{model_save_name}"
    torch.save({'epoch': 0, 'model_state_dict': model.state_dict(),'optimizer_state_dict': optimizer.state_dict(),'loss': 0}, PATH)

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    # Get model outputs and calculate loss
```

```
                    # Special case for inception because in training it has an auxiliary output. In train
                    #   mode we calculate the loss by summing the final output and the auxiliary output
                    #   but in testing we only consider the final output.
                    if is_inception and phase == 'train':
                        outputs, aux_outputs = model(inputs)
                        loss1 = criterion(outputs, labels)
                        loss2 = criterion(aux_outputs, labels)
                        loss = loss1 + 0.4*loss2
                    else:
                        outputs = model(inputs)
                        loss = criterion(outputs, labels)

                    _, preds = torch.max(outputs, 1)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / len(dataloaders[phase].dataset)
            epoch_acc = running_corrects.double() / len(dataloaders[phase].dataset)

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epoch_acc))

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
            if phase == 'val':
                val_acc_history.append(epoch_acc)

        print()

    model_save_name = f"{epoch}.pt"
    PATH= f"gdrive/My Drive/201906AI_FinalProject/checkpoints_resnet18/{model_save_name}"
    torch.save({'epoch': epoch, 'model_state_dict': model.state_dict(),'optimizer_state_dict': optimizer.state_dict(),'loss': loss}, PATH)

    time_elapsed = time.time() - since
```

```
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed //
60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model, val_acc_history
```

Optimizer

```
params_to_update = model_ft.parameters()
print("Params to learn:")
if feature_extract:
    params_to_update = []
    for name,param in model_ft.named_parameters():
        if param.requires_grad == True:
            params_to_update.append(param)
            print("\t",name)
else:
    for name,param in model_ft.named_parameters():
        if param.requires_grad == True:
            print("\t",name)

optimizer_ft = optim.SGD(params_to_update, lr=0.001, momentum=0.9)
```

训练模型

```
criterion = nn.CrossEntropyLoss()

model_ft, hist = train_model(model_ft, dataloaders_dict, criterion,
optimizer_ft, num_epochs=num_epochs, is_inception=
(model_name=="inception"))
```

测试模型

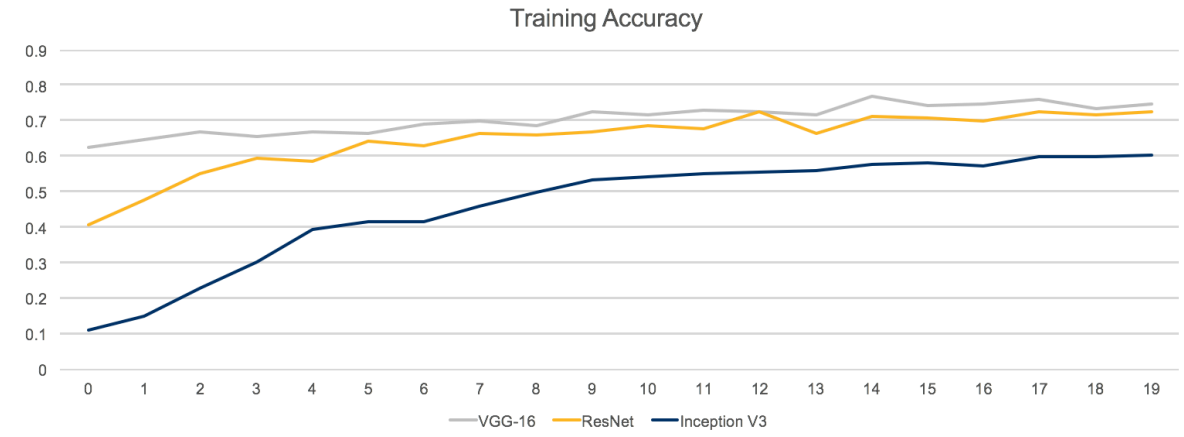
全局变量**

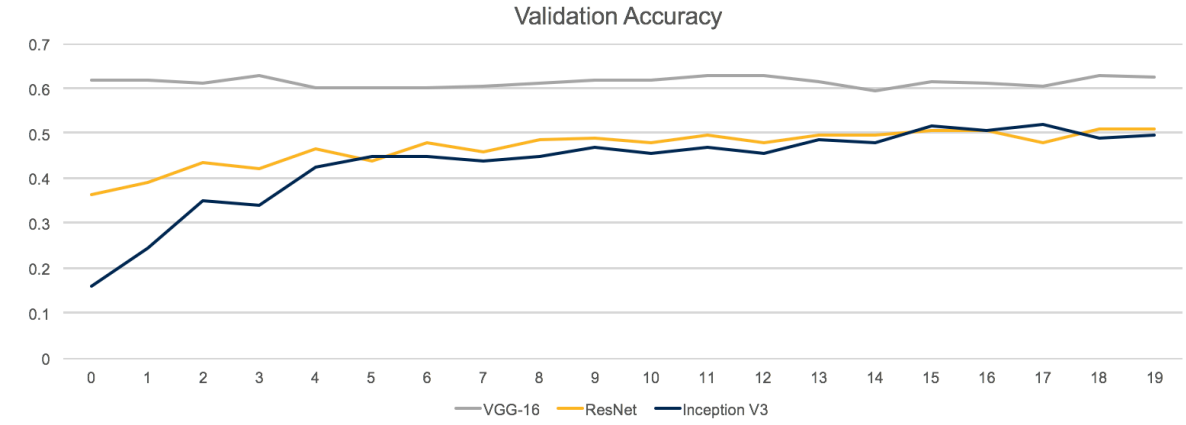
```
model_name = 'resnet'/'vgg'/'inception'

num_classes = 10
batch_size = 32

num_epochs = 20

feature_extract = False/True

input_size = 224

```

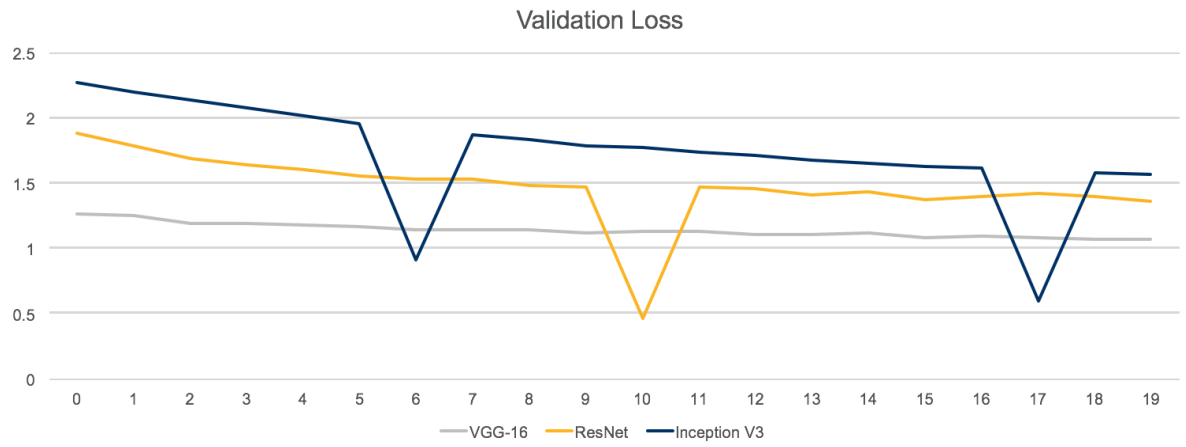
```
num_workers = 4

device = torch.device("cuda")
```

结果

首先是用艺术作品图片对三种模型进行特征提取时的结果

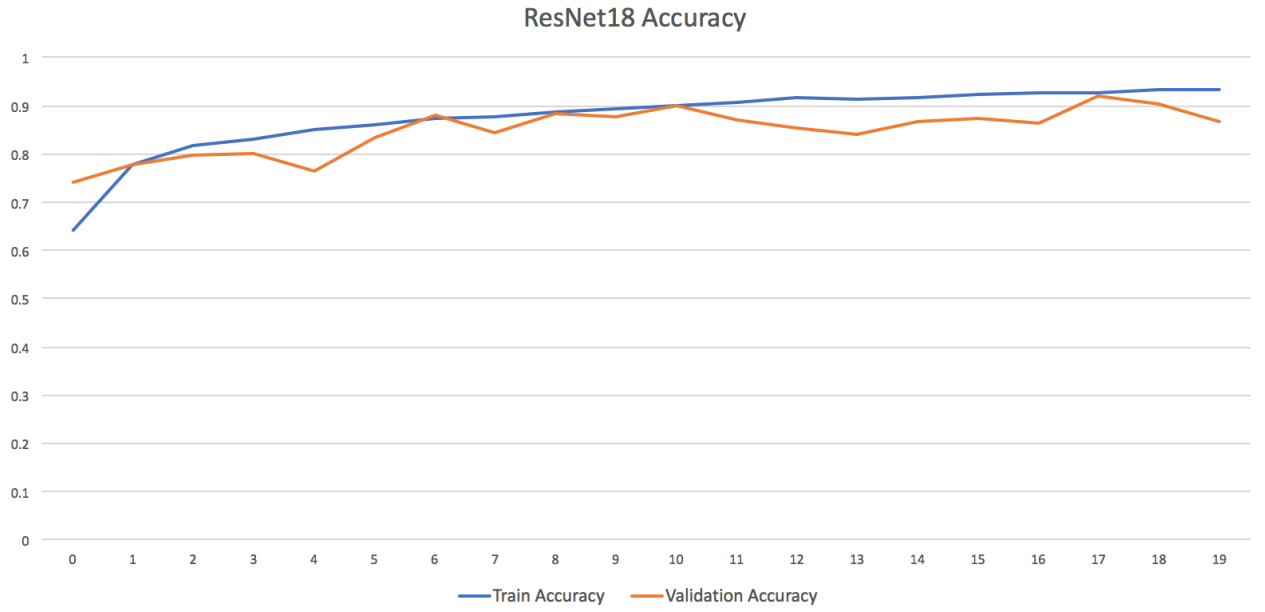


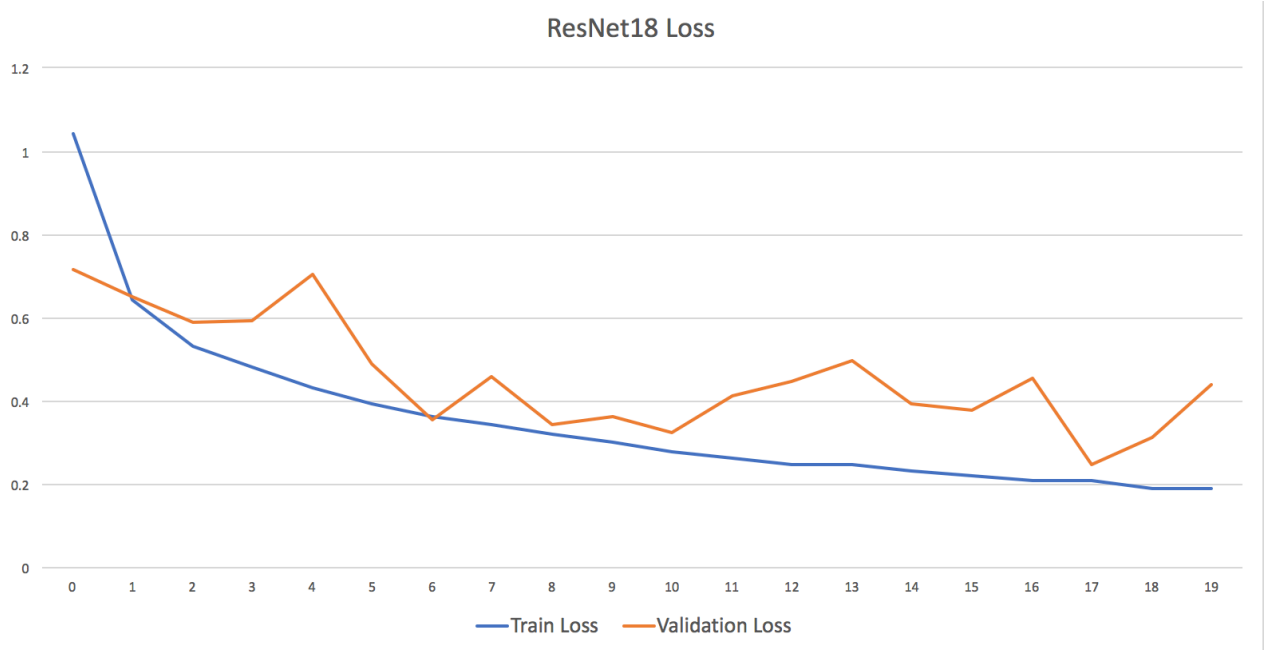




可以看到，训练集和验证集的准确度都在不断提升，损失在下降，训练效果较好。

然后是利用图片数据集从头训练ResNet18的所有参数的结果





测试集的准确率达到了93%，可以说效果很好。

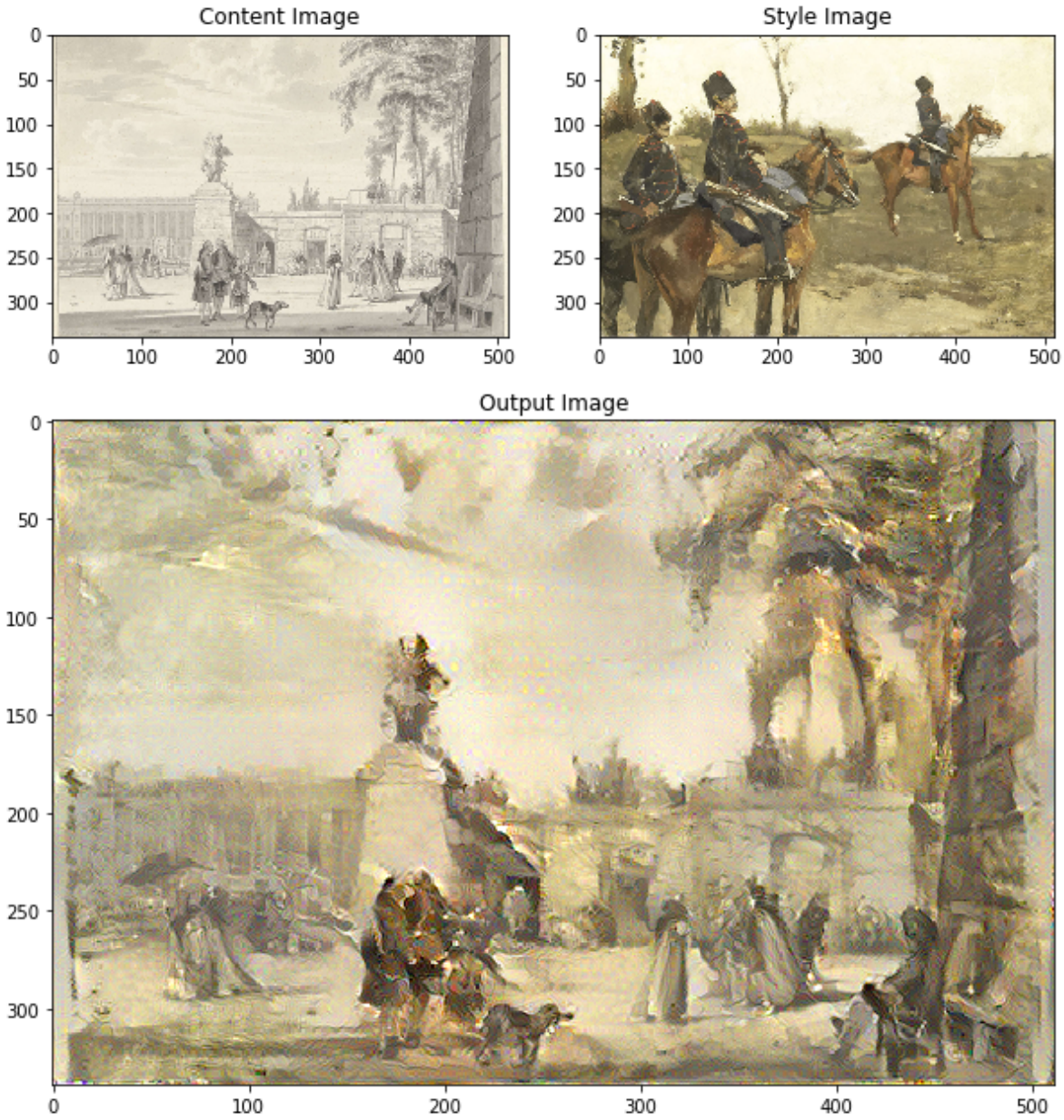
```python
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model_ft(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += torch.sum(predicted == labels)

    print('Test Accuracy of the model on the test images: {} %'.format(100 * correct / total))
```

```
Test Accuracy of the model on the test images: 93 %
```

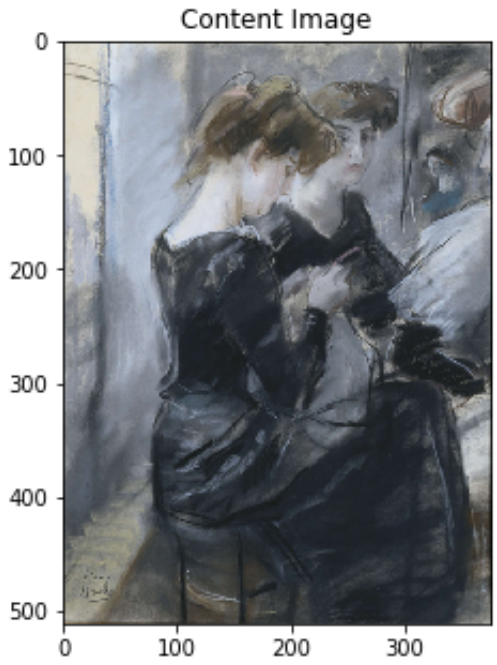
3.2 探究分类结果

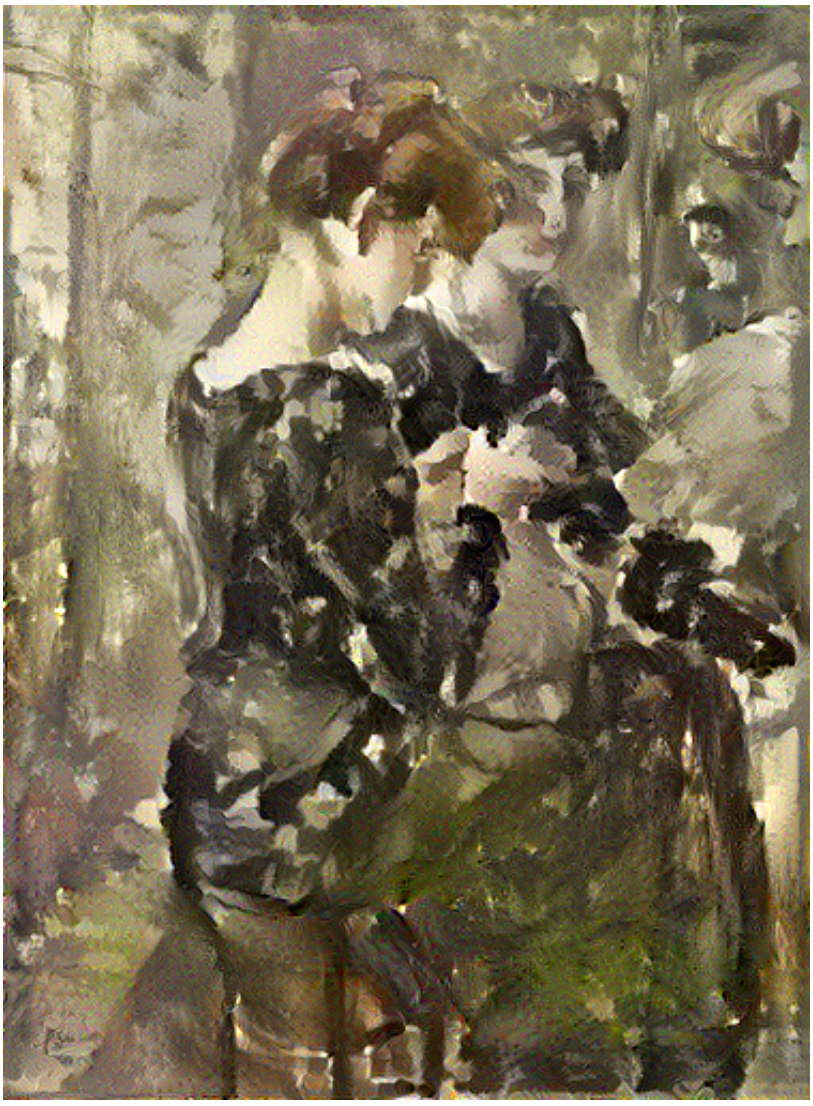
由于时间限制以及图片大小差异过大，实验选择了三组图片进行了三次风格迁移

将GeorgeHendrikBreitner的风格迁移到ReinierVinkeles

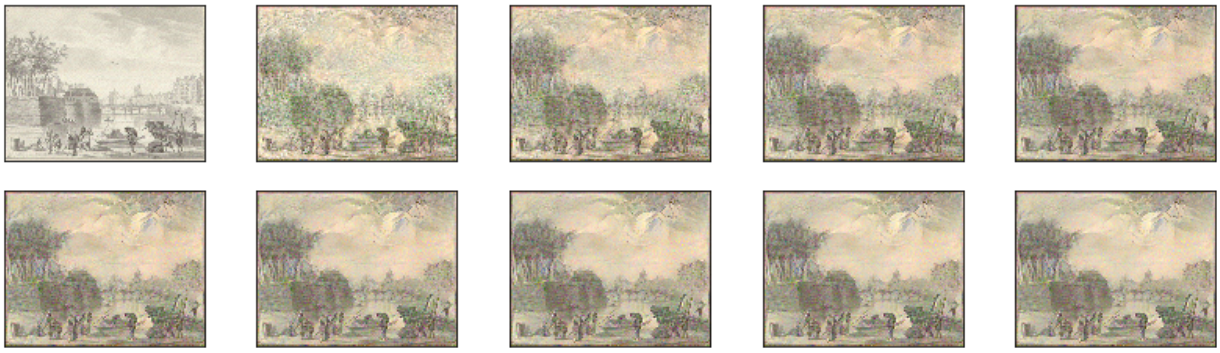


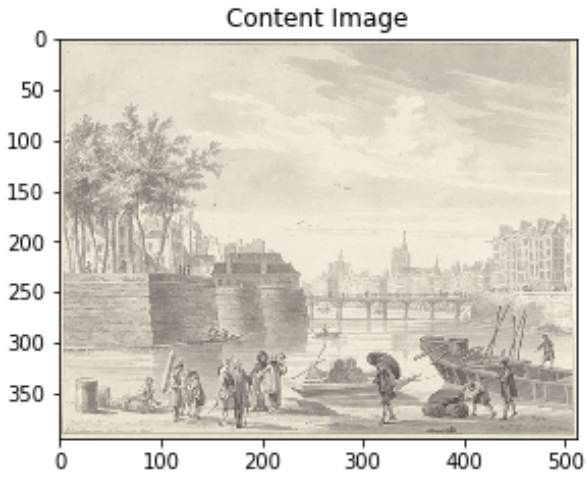
将GeorgeHendrikBreitner的风格迁移到IsaacIsraels



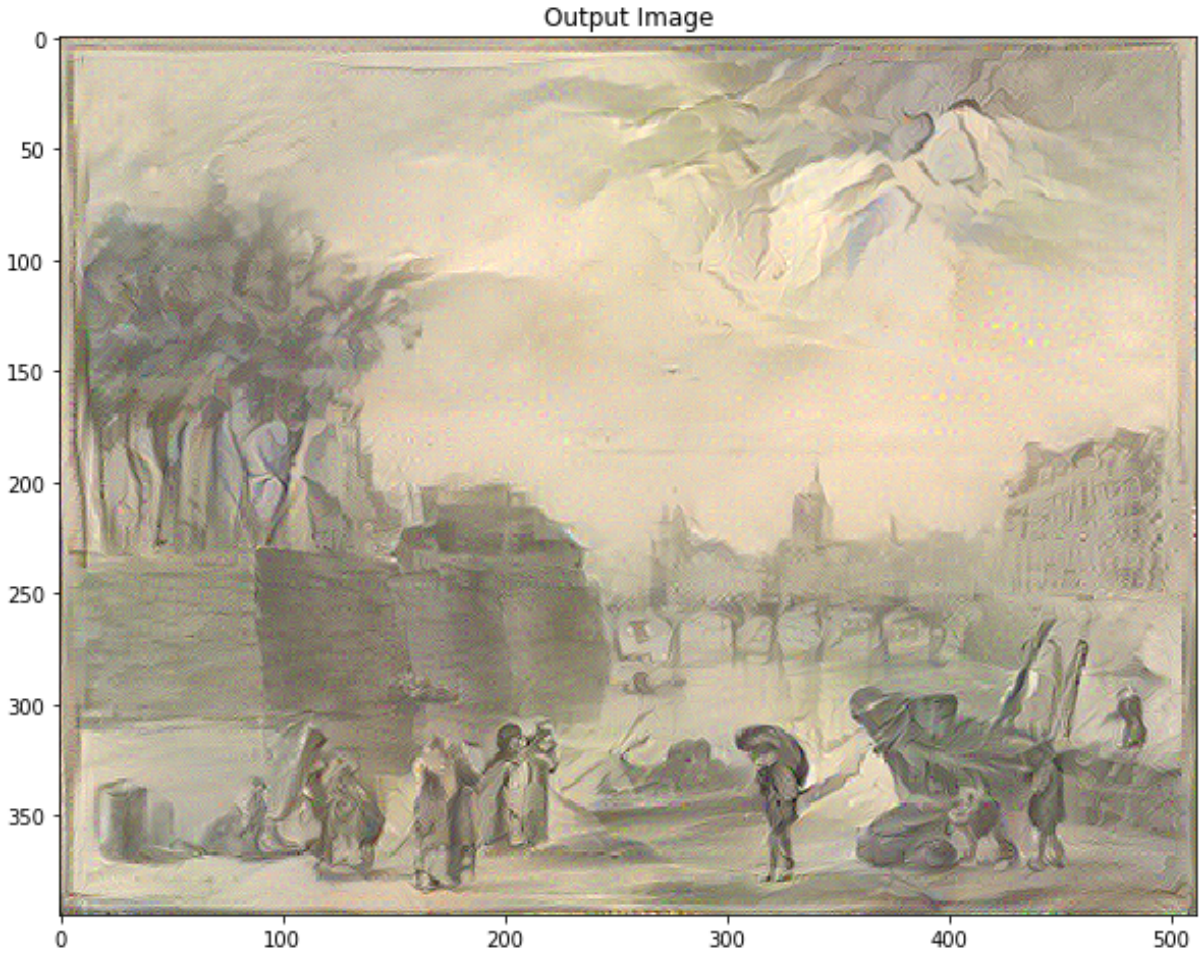


将JohannesTavenraat的风格迁移到RrinierVinkeles




Content Image
0
50
100
150
200
250
300
350
0
100
200
300
400
500


Style Image
0
50
100
150
200
250
300
350
0
100
200
300
400
500


Output Image
0
50
100
150
200
250
300
350
0
100
200
300
400
500

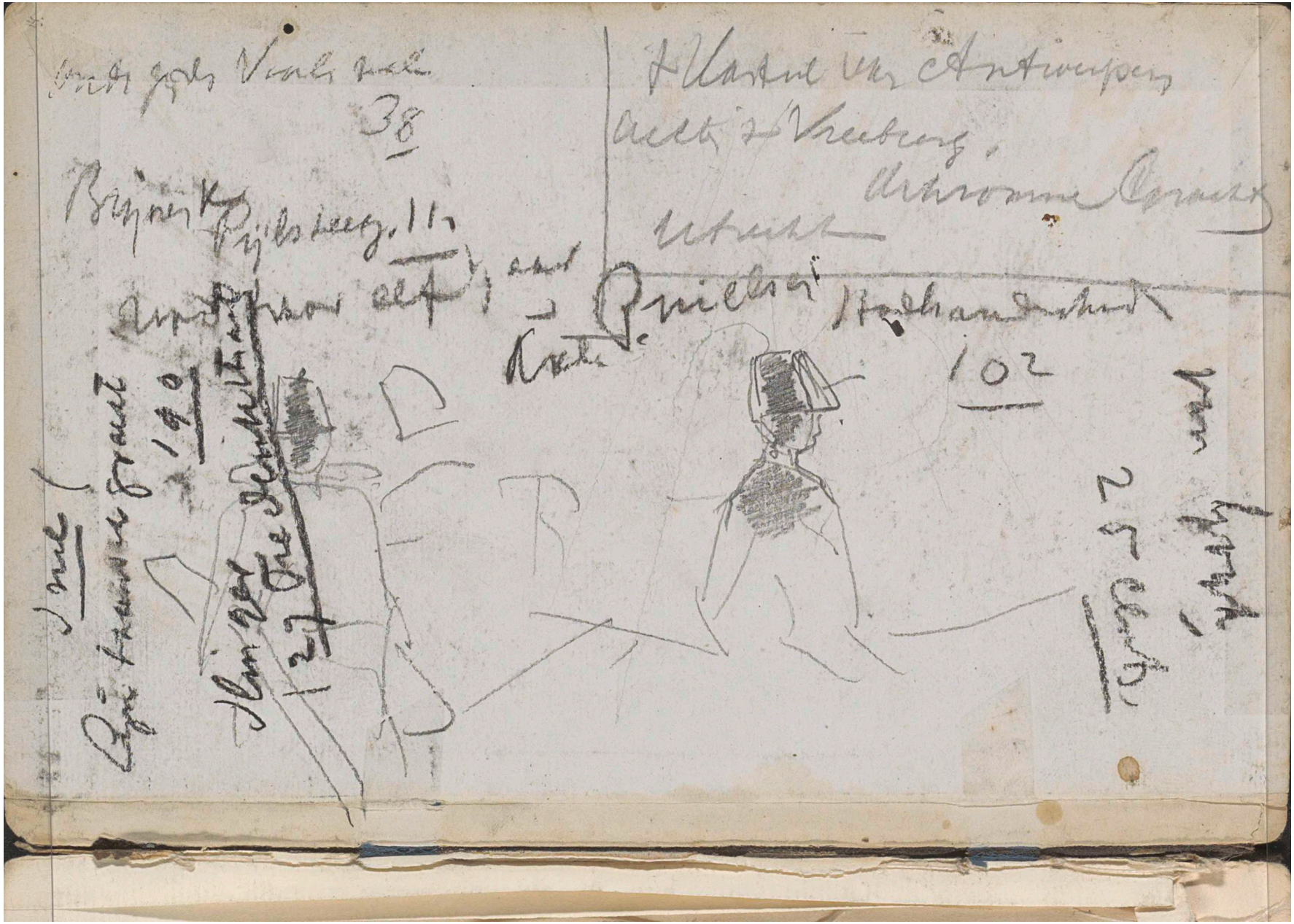
对这三张迁移的画作的分类结果如下，将A的作品风格迁移到B的作品后，给出的分类结果既不是A也不是B。当然，我们并不能根据这3张的结果给出人和结论，下一步应该对大量作品进行风格迁移后进行分类。

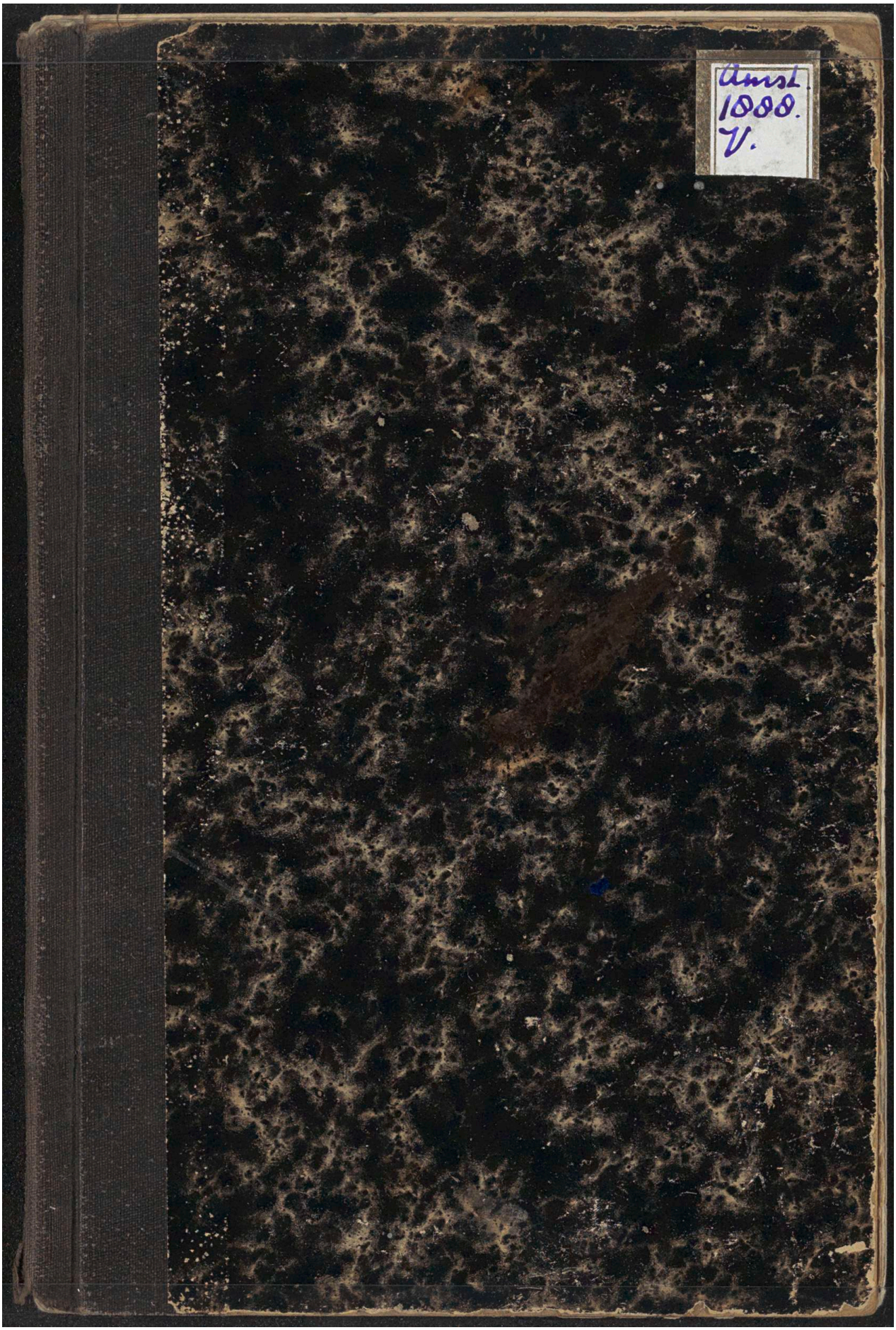
Content image	Style image	Predicted
GeorgeHendrikBreitner	ReinierVinkeles	MariusBauer
GeorgeHendrikBreitner	IsaacIsraels	MariusBauer
JohannesTavenraat	RrinierVinkeles	BernardPicart

4. 总结

本次项目希望实现一个艺术作品图片的分类器，预测输入的艺术作品的创作者病输出，并通过风格迁移对分类器有一定程度的理解。实验时利用荷兰的*Rijksmuseum*博物馆发布的数据集，利用10位创作者的17513张图片对分类器进行训练。首先利用VGG16, ResNet18和Inception V3进行特征提取，仅训练全连接层。再对效果较好、训练速度较快的ResNet18训练所有参数，分类准确率在验证集可以达到90%左右。为了更好的理解分类时的判断依据，又利用神经风格迁移探究了风格和内容哪个对分类结果影响更大。

实验还存在非常多的不足和提升空间。首先数据集中存在一些模糊不清的手稿或手稿封面（如下图）。对训练造成一定干扰。但由于图片数量较多，无效图片的种类繁多，难以全部剔除，目前还没有找到很好的方法排除。





其次，由于图像的风格迁移需要两张图片大小相近，但数据集中图片大小差异非常大，目前在有限的时间内难以对大量图片进行风格迁移，所以对实验结果的理解这一部分还需要更多的工作。

参考资料

1. T. Mensink and J. van Gemert. The rijksmuseum challenge: Museum-centered visual recognition. In *ACM International Conference on Multimedia Retrieval (ICMR)*, 2014.
2. https://uvaauas.figshare.com/articles/Rijksmuseum_Challenge_2014/5660617
3. K. Simonyan and A. Zisserman. Very deep convolu- tional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
4. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
5. L.A.Gatys,A.S.Ecker,andM.Bethge.Aneuralalgorithm of artistic style. *CoRR*, abs/1508.06576, 2015