06 卷积神经网络的应用

【20 分】自己选择一张照片，采用 3 种方法对图片增广处理。

下方选择了横向和纵向翻转，裁剪法和变色法。

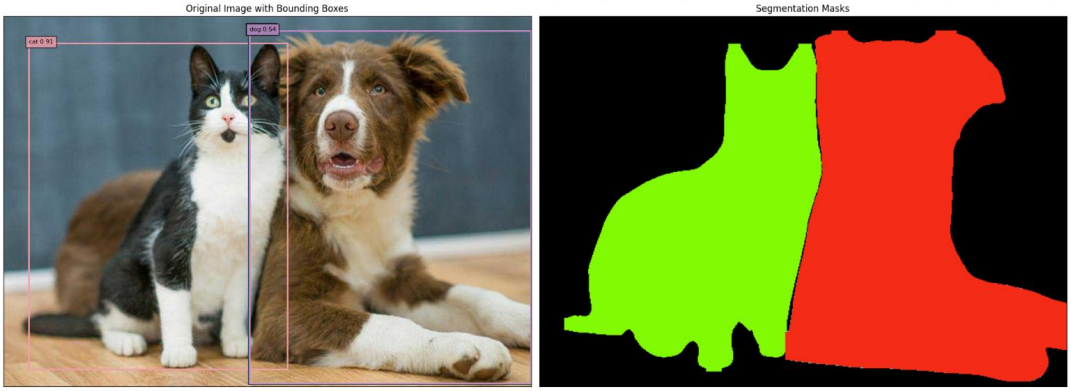
```
import torchvision
from torchvision import transforms
# 加载和预处理数据
from PIL import Image
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
# 读取图像文件
image0 = Image.open("/kaggle/input/cat-transform/cat.jpg")
num = 5 # 翻转次数
# 将图像翻转，翻转概率为50%
my_transform1 = transforms.RandomHorizontalFlip(p=0.5)
my_transform2 = transforms.RandomVerticalFlip(p=0.5)
my_transform3 = transforms.RandomResizedCrop(
    size=200, # 裁剪后的图像大小
    scale=(0.1, 1.0), # 随机裁剪面积比例范围
    ratio=(0.5, 2), # 随机长宽比例范围
    interpolation=2) # 插值方法，2表示双线性插值
my_transform4 = transforms.ColorJitter(
    brightness=0.5, # 亮度变化范围
    contrast=0.5, # 对比度变化范围
    saturation=0.5, # 饱和度变化范围
    hue=0.5) # 色相变化范围

# 初始化一个列表，用来存放翻转后的图像
image = []
```

```
# 图像翻转num次
for i in range(num):
    image.append(my_transform1(image0))
    image.append(my_transform2(image0))
    image.append(my_transform3(image0))
    image.append(my_transform4(image0))
# 创建一个图形窗口
plt.figure(figsize=(8,10)) # specifying the overall grid size

for i in range(20):
    plt.subplot(5,4,i+1)    # the number of images in the grid is 4*5 (20)
    plt.imshow(image[i])
    plt.axis('off')

plt.show()
```

【20 分】自己选择一张有多个物体的照片，分别用 YOLOn、YOLO1 模型，分别生成物体检测图片、物体分割蒙版图片。

```
Creating new Ultralytics Settings v0.0.6 file ✅
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help
see https://docs.ultralytics.com/quickstart/#ultralytics-settings.

0: 448x640 2 cats, 1 dog, 387.1ms
Speed: 48.8ms preprocess, 387.1ms inference, 43.9ms postprocess per image at shape (1, 3, 448, 640)
```


```
0: 448x640 2 cats, 1 dog, 1135.3ms
Speed: 1.5ms preprocess, 1135.3ms inference, 6.9ms postprocess per image at shape (1, 3, 448, 640)
```

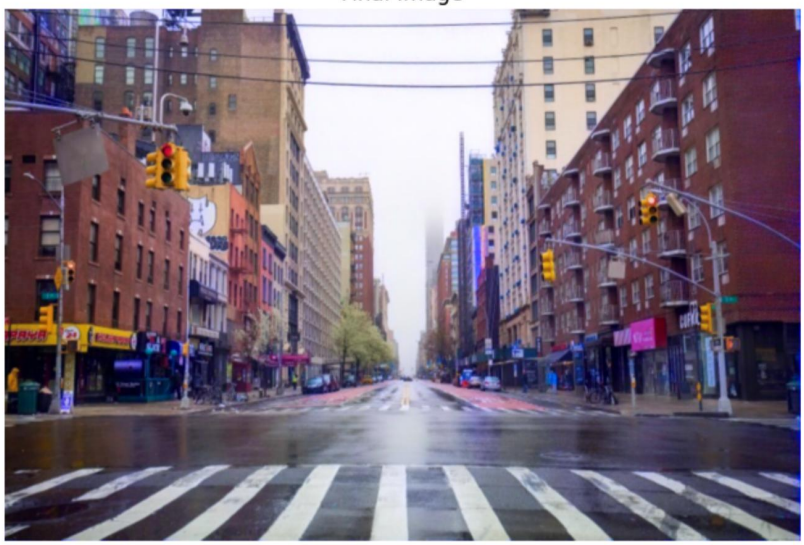


【30 分】自己选 2 张照片，一张作为内容图像，一张作为风格图像，用 VGG16 或 VGG19 模型生成一张合成图像。

原本一直无法训练（不断出现 cuda out of memory 报错），经过助教提点后才发现图片画质过高也会导致这种情况，最后选择了像素较少的图片运行成功。


```
Step 100:
Total loss:  0.06810130923986435 ,  content_loss:  0.001225449494086206 ,  style_loss:  668.7586059
570312
Step 200:
Total loss:  0.06054750829935074 ,  content_loss:  0.002013586461544037 ,  style_loss:  585.3392333
984375
Step 300:
Total loss:  0.05431867763400078 ,  content_loss:  0.002084981417283416 ,  style_loss:  522.3369750
976562
Step 400:
Total loss:  0.04945111647248268 ,  content_loss:  0.002539327135309577 ,  style_loss:  469.1179199
21875
Step 500:
Total loss:  0.044607665389776 23 ,  content_loss:  0.0024849921464920044 ,  style_loss:  421.226745
60546875
Step 600:
Total loss:  0.04149434342980385 ,  content_loss:  0.0035747564397752285 ,  style_loss:  379.195892
3339844
Step 700:
Total loss:  0.036785464733839035 ,  content_loss:  0.002896622521802783 ,  style_loss:  338.888427
734375
Step 800:
Total loss:  0.0333382673561573 ,  content_loss:  0.003004521131515503 ,  style_loss:  303.33746337
890625
Step 900:
Total loss:  0.03064826875925064 ,  content_loss:  0.003369447775185108 ,  style_loss:  272.7882080
078125
Step 1000:
Total loss:  0.028150277212262154 ,  content_loss:  0.0034104834776371717 ,  style_loss:  247.39794
921875
```

Final Image



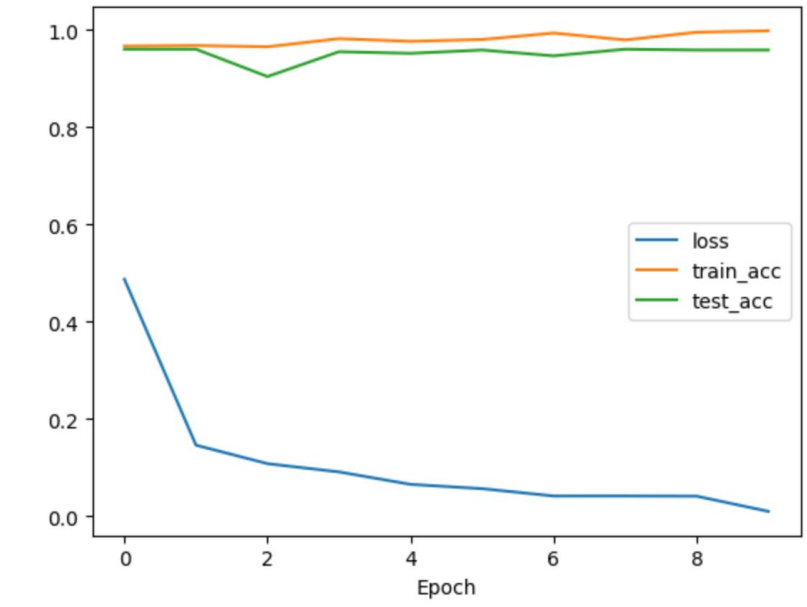
【30 分】采用 elearning 上的 sports.zip 小数据集，用 ResNet18 模型完成迁移学习练习，对比训练精度、测试精度：

4.1）仅仅载入 ResNet18 模型，但不载入已经训练好的参数，用 sports 小数据中的 train 图片进行训练，用 test 图片进行测试。

将所提供的代码中的 resnet18 载入代码部分中的 pretrained=true 改成 false 后运行：

```
# <5> 模型定义、训练、显示
import torch.nn as nn
import torchsummary
import torchvision

# <5-1> 损失函数
criterion = nn.CrossEntropyLoss()

# <5-2> 加载已训练好的Resnet18模型参数（2种方法）
# ============
# 网上下载 预训练模型
model = torchvision.models.resnet18(pretrained=False)# 网上下载
```

```
Epoch 1/10, Loss: 0.49, Train accuracy: 0.97, Test accuracy:  0.96

Epoch 2/10, Loss: 0.15, Train accuracy: 0.97, Test accuracy:  0.96

Epoch 3/10, Loss: 0.11, Train accuracy: 0.97, Test accuracy:  0.90

Epoch 4/10, Loss: 0.09, Train accuracy: 0.98, Test accuracy:  0.96

Epoch 5/10, Loss: 0.07, Train accuracy: 0.98, Test accuracy:  0.95

Epoch 6/10, Loss: 0.06, Train accuracy: 0.98, Test accuracy:  0.96

Epoch 7/10, Loss: 0.04, Train accuracy: 0.99, Test accuracy:  0.95

Epoch 8/10, Loss: 0.04, Train accuracy: 0.98, Test accuracy:  0.96

Epoch 9/10, Loss: 0.04, Train accuracy: 1.00, Test accuracy:  0.96

Epoch 10/10, Loss: 0.01, Train accuracy: 1.00, Test accuracy:  0.96
Final train accuracy: 1.00
```

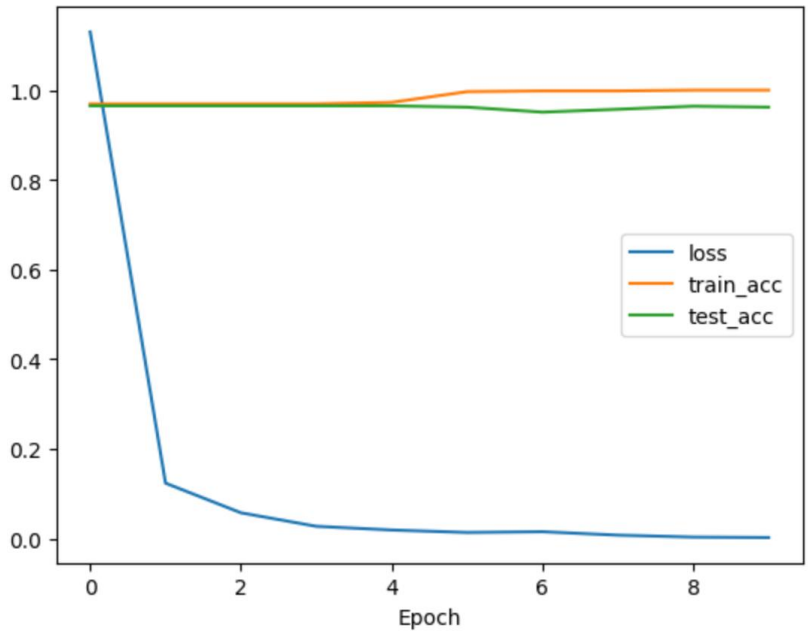


4.2）仅仅载入 ResNet18 模型，但不载入已经训练好的参数，用 sports 小数据中的 test 图片进行训练，用 train 图片进行测试。

将 train 和 test 函数被使用时的相关 loader 函数交换，如下即可：

```
num_epochs = 10  # 训练次数
for epoch in range(num_epochs):
    train_loss = train(model, test_loader)  # 训练 #originally train_loader;changed according to question
    train_acc = test(model, test_loader)  # 测试 #originally train_loader;changed according to question
    test_acc = test(model, train_loader)  # 测试 #originally test_loader;changed according to question

    # 更新绘图指标
    train_acc_history.append(train_acc)
    test_acc_history.append(test_acc)
    loss_history.append(train_loss)
```

```
Epoch 1/10, Loss: 1.13, Train accuracy: 0.97, Test accuracy:  0.97

Epoch 2/10, Loss: 0.12, Train accuracy: 0.97, Test accuracy:  0.97

Epoch 3/10, Loss: 0.06, Train accuracy: 0.97, Test accuracy:  0.97

Epoch 4/10, Loss: 0.03, Train accuracy: 0.97, Test accuracy:  0.97

Epoch 5/10, Loss: 0.02, Train accuracy: 0.97, Test accuracy:  0.97

Epoch 6/10, Loss: 0.01, Train accuracy: 1.00, Test accuracy:  0.96

Epoch 7/10, Loss: 0.02, Train accuracy: 1.00, Test accuracy:  0.95

Epoch 8/10, Loss: 0.01, Train accuracy: 1.00, Test accuracy:  0.96

Epoch 9/10, Loss: 0.00, Train accuracy: 1.00, Test accuracy:  0.96

Epoch 10/10, Loss: 0.00, Train accuracy: 1.00, Test accuracy:  0.96
Final train accuracy: 1.00
Final test accuracy: 0.96
```

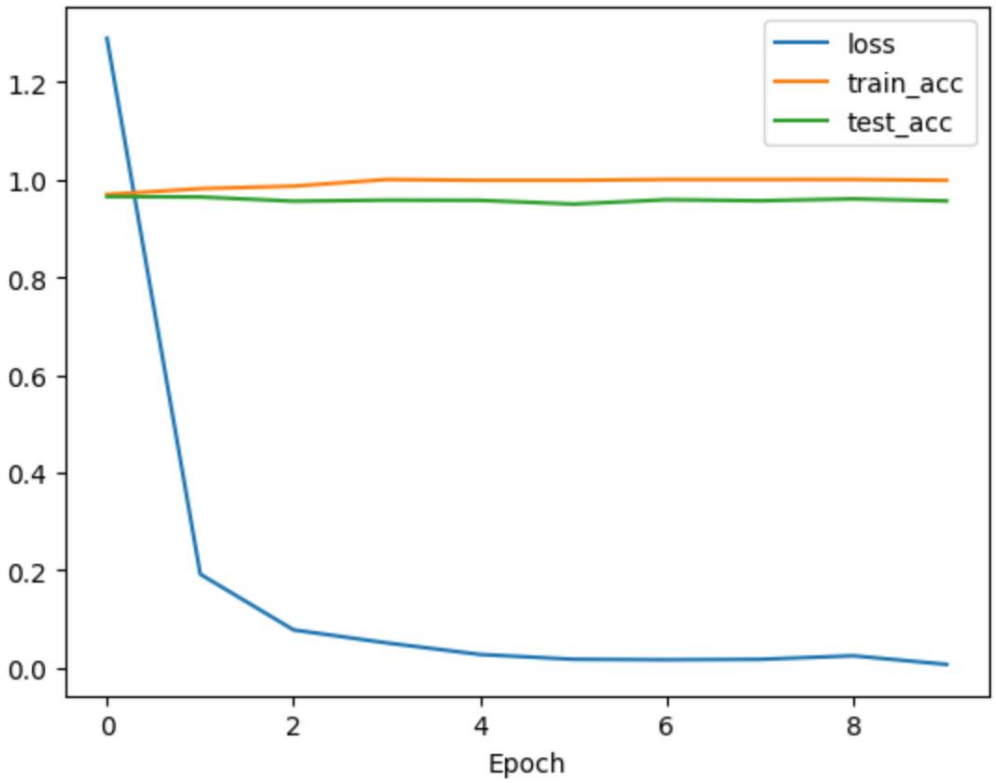


从上发现和第一小题比，虽然刚开始第一个 epoch 的 loss 很大，但之后的 epoch 结果和第一小题区别不大（做到这里是怀疑 test 数据集的数据量足以训练模型，因此即使其数据量比 train 少了 4 倍，但结果与第一小题区别不大，且还有模型本身的能力较强，不然一般来说把 train 和 test 数据集交换的准确率不会和正常情况差不多。）

4.3）不仅载入 ResNet18 模型，同时载入已经训练好的参数，用 sports 小数据中的 test 图片进行训练，用 train 图片进行测试。

将 pretrained=false 改回变 true 即可

```
Epoch 1/10, Loss: 1.29, Train accuracy: 0.97, Test accuracy:  0.97

Epoch 2/10, Loss: 0.19, Train accuracy: 0.98, Test accuracy:  0.96

Epoch 3/10, Loss: 0.08, Train accuracy: 0.99, Test accuracy:  0.96

Epoch 4/10, Loss: 0.05, Train accuracy: 1.00, Test accuracy:  0.96

Epoch 5/10, Loss: 0.03, Train accuracy: 1.00, Test accuracy:  0.96

Epoch 6/10, Loss: 0.02, Train accuracy: 1.00, Test accuracy:  0.95

Epoch 7/10, Loss: 0.02, Train accuracy: 1.00, Test accuracy:  0.96

Epoch 8/10, Loss: 0.02, Train accuracy: 1.00, Test accuracy:  0.96

Epoch 9/10, Loss: 0.03, Train accuracy: 1.00, Test accuracy:  0.96

Epoch 10/10, Loss: 0.01, Train accuracy: 1.00, Test accuracy:  0.96
Final train accuracy: 1.00
Final test accuracy: 0.96
```



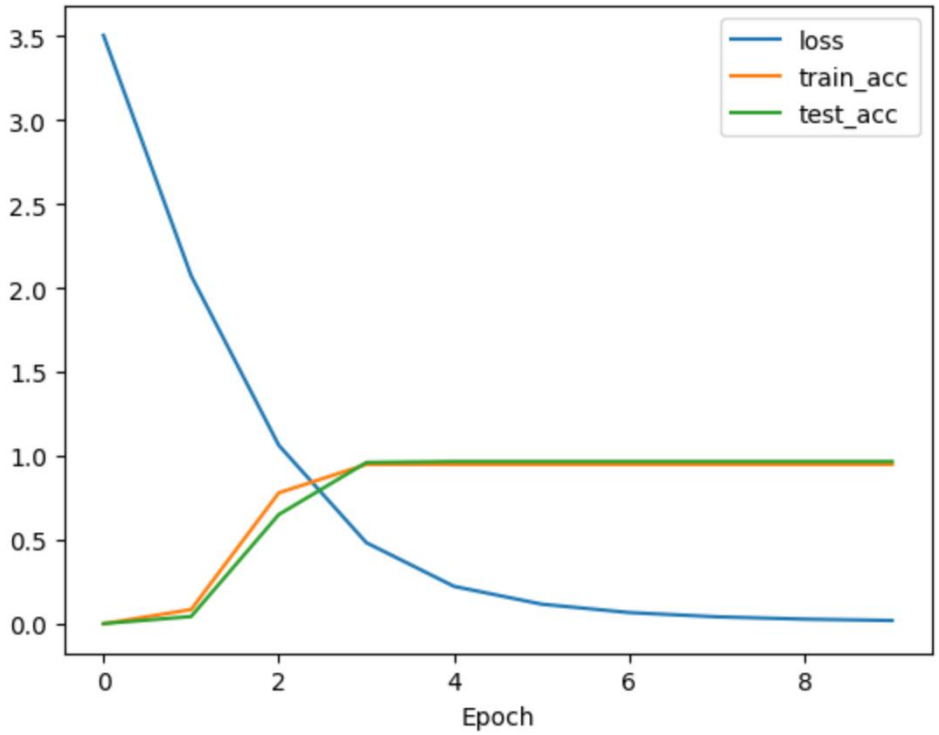
和前两个小题比，虽然一开始的损失较大且最终（在 train 数据集）进行测试的准确率和之前差不多，但训练的准确率只需要 4 个 epoch 就达到了 100%准确率，比之前两题都快达到。

额外：将 test 数据集变得更小，重做第二小题（即 test 数据集将用于训练）：

```
train_size = int(0.98 * len(dataset))
test_size  = len(dataset) - train_size
```

```
training images: 2872, testing images: 59
```

```
Epoch 1/10, Loss: 3.50, Train accuracy: 0.00, Test accuracy:  0.00

Epoch 2/10, Loss: 2.07, Train accuracy: 0.08, Test accuracy:  0.04

Epoch 3/10, Loss: 1.06, Train accuracy: 0.78, Test accuracy:  0.65

Epoch 4/10, Loss: 0.48, Train accuracy: 0.95, Test accuracy:  0.96

Epoch 5/10, Loss: 0.22, Train accuracy: 0.95, Test accuracy:  0.97

Epoch 6/10, Loss: 0.12, Train accuracy: 0.95, Test accuracy:  0.97

Epoch 7/10, Loss: 0.07, Train accuracy: 0.95, Test accuracy:  0.97

Epoch 8/10, Loss: 0.04, Train accuracy: 0.95, Test accuracy:  0.97

Epoch 9/10, Loss: 0.03, Train accuracy: 0.95, Test accuracy:  0.97

Epoch 10/10, Loss: 0.02, Train accuracy: 0.95, Test accuracy:  0.97
Final train accuracy: 0.95
Final test accuracy: 0.97
```



从上发现一开始的几个 epoch 的 loss 会变得很大，但到了第四个 epoch 就逐渐变得正常。因此推断是模型自身较强大的原因，即使在有限数据且未引入已预训练的参数情况下也能训练成功。