程序报告

学号：22375080　　姓名：杨佳宇轩

一、问题重述

（简单描述对问题的理解，从问题中抓住主干，必填）

==========

实验先给出手写是数字识别的示例，用最基础最经典的图像识别示例展示图像识别的全流程，让我们理清思路。之后需要根据给出的动物图片分类的项目部分代码，完善代码并进行一定程度的改进，改进思路包括但不仅限于：采用数据增强、改变网络结构、引入半监督学习、理解和补全 ViT 模型等，通过优化以达到对于私有测试集更好的预测结果。

二、设计思想

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… 必填）

==========

1. 首先是 CNN 网络进行补全，可以达到 0.208 的正确率
2. 尝试优化网络结构，增加图像增强的内容，图像增强包括图像旋转 15 度，图像左右翻转，图像饱和度和亮度修改等，可以达到 0.508 的正确率

```python
# 在训练中进行数据增强很重要。
train_tfm = transforms.Compose([
    # 将图像大小调整为固定形状 (height = width = 128)
    transforms.Resize((128, 128)),
    # ---------- TODO ----------
    # 任务点1：数据增强
    # 在此处添加你的代码，进行数据增强
    transforms.RandomResizedCrop(128, scale=(0.8, 1.0)),  # 随机裁剪并缩放
    transforms.RandomHorizontalFlip(prob=0.5),                # 随机水平翻转
    # transforms.RandomVerticalFlip(prob=0.2),                  # 随机垂直翻转
    transforms.RandomRotation(degrees=15),                # 随机旋转
    transforms.ColorJitter(
        brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1
    ),  # 随机调整亮度、对比度、饱和度和色相
    transforms.ToTensor(),
])
```

3. 尝试补全 ViT 模型，但没有跑出比较优秀的正确率
4. 尝试使用 ResNet18、ResNet34 等模型，通过调整学习率可以达到 0.710 的正确率

```python
    model = paddle.vision.models.resnet18(pretrained=False)
    # model = paddle.vision.models.resnet34(pretrained=False)
```

5. 达到比较优秀的正确率之后尝试加入半监督学习，可以在公有测试集上达到 0.7178 的正确率

三、代码内容

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

==========

CNN：

```
import paddle
import paddle.nn as nn

class CNN(nn.Layer):
    def __init__(self, num_classes=10, dropout_prob=0.3):
        super(CNN, self).__init__()
        # 特征提取部分
        self.features = nn.Sequential(
            # 第 1 个卷积块：3->32 通道，128x128->64x64
            nn.Conv2D(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2D(32),
            nn.ReLU(),
            nn.MaxPool2D(kernel_size=2, stride=2),

            # 第 2 个卷积块：32->64 通道，64x64->32x32
            nn.Conv2D(32, 64, 3, 1, 1),
            nn.BatchNorm2D(64),
            nn.ReLU(),
            nn.MaxPool2D(2, 2),

            # 第 3 个卷积块：64->128 通道，32x32->16x16
            nn.Conv2D(64, 128, 3, 1, 1),
            nn.BatchNorm2D(128),
            nn.ReLU(),
            nn.MaxPool2D(2, 2),

            # 第 4 个卷积块：128->256 通道，16x16->8x8
            nn.Conv2D(128, 256, 3, 1, 1),
            nn.BatchNorm2D(256),
            nn.ReLU(),
            nn.MaxPool2D(2, 2),

            # 第 5 个卷积块：256->512 通道，8x8->4x4
            nn.Conv2D(256, 512, 3, 1, 1),
            nn.BatchNorm2D(512),
            nn.ReLU(),
            nn.MaxPool2D(2, 2)
        )

        # 分类器部分
        self.classifier = nn.Sequential(
            nn.Linear(512 * 4 * 4, 1024),
            nn.ReLU(),
```

```
            nn.Dropout(0.5),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = paddle.flatten(x, start_axis=1)
        x = self.classifier(x)
        return x
```

==========

ViT

```
# coding=utf-8
# 导入环境
import os
import numpy as np
import cv2
from PIL import Image
import matplotlib.pyplot as plt
import paddle
from paddle.io import Dataset
from paddle.nn import Conv2D, MaxPool2D, Linear, Dropout, BatchNorm, AdaptiveAvgPool2D, AvgPool2D
import paddle.nn.functional as F
import paddle.nn as nn

# 将输入 x 由 int 类型转为 tuple 类型
def to_2tuple(x):
    return tuple([x] * 2)

# 定义一个什么操作都不进行的网络层
class Identity(nn.Layer):
    def __init__(self):
        super(Identity, self).__init__()

    def forward(self, input):
        return input

# 图像分块、Embedding
class PatchEmbed(nn.Layer):
    def __init__(self, img_size=224, patch_size=16, in_chans=3, embed_dim=768):
        super().__init__()
```

```
        # 原始大小为 int，转为 tuple，即：img_size 原始输入 224，变换后为[224,224]
        img_size = to_2tuple(img_size)
        patch_size = to_2tuple(patch_size)
        # 图像块的个数
        num_patches = (img_size[1] // patch_size[1]) * \
            (img_size[0] // patch_size[0])
        self.img_size = img_size
        self.patch_size = patch_size
        self.num_patches = num_patches
        # kernel_size=块大小，即每个块输出一个值，类似每个块展平后使用相同的全连接层进行处理
        # 输入维度为 3，输出维度为块向量长度
        # 与原文中：分块、展平、全连接降维保持一致
        # 输出为[B, C, H, W]
        # 这里应该填入分块大小参数，步长应与分块大小一致以保证不重叠
        self.proj = nn.Conv2D(
            in_chans, embed_dim, kernel_size=patch_size, stride=patch_size)

    def forward(self, x):
        B, C, H, W = x.shape
        assert H == self.img_size[0] and W == self.img_size[1], \
                                    "Input image size ({H}*{W}) doesn't match model ({self.img_size[0]}*{self.img_size[1]})."
        # [B, C, H, W] -> [B, C, H*W] ->[B, H*W, C]
        x = self.proj(x).flatten(2).transpose((0, 2, 1))
        return x

# Multi-head Attention
class Attention(nn.Layer):
    def __init__(self,
                 dim,
                 num_heads=8,
                 qkv_bias=False,
                 qk_scale=None,
                 attn_drop=0.,
                 proj_drop=0.):
        super().__init__()
        self.num_heads = num_heads
        # ---------- TODO ----------
        # ViT 任务 1：多头注意力
        # 提示：计算每个注意力头的维度（总维度 dim 除以注意力头数）
        head_dim = dim // num_heads
        self.scale = qk_scale or head_dim**-0.5
        # 计算 q,k,v 的转移矩阵
```

```
        # ---------- TODO ----------
        # ViT 任务 2：计算 q,k,v 的转移矩阵
        # 提示：q/k/v 三个矩阵需要多少倍维度？
        self.qkv = nn.Linear(in_features=dim, out_features=dim * 3, bias_attr=qkv_bias)
        self.attn_drop = nn.Dropout(attn_drop)
        # 最终的线性层
        self.proj = nn.Linear(dim, dim)
        self.proj_drop = nn.Dropout(proj_drop)

    def forward(self, x):
        N, C = x.shape[1:]
        # 线性变换
        qkv = self.qkv(x).reshape((-1, N, 3, self.num_heads, C //
                                   self.num_heads)).transpose((2, 0, 3, 1, 4))
        # ---------- TODO ----------
        # ViT 任务 3：分割 query key value
        # 提示：按顺序取 qkv 的值
        q, k, v = qkv[0], qkv[1], qkv[2]
        # Scaled Dot-Product Attention
        # Matmul + Scale
        # ---------- TODO ----------
        # ViT 任务 4：缩放点积注意力公式
        # 提示：实现缩放点积注意力公式中的缩放步骤，考虑 self.scale
        # 题外话：可以关注一下缩放因子 self.scale 的计算方法，并思考为什么要进行缩放？
        attn = (q.matmul(k.transpose((0, 1, 3, 2)))) * self.scale
        # SoftMax
        # ---------- TODO ----------
        # ViT 任务 5：归一化
        # 提示：注意力权重的归一化操作，应沿哪个维度做 softmax？
        attn = nn.functional.softmax(attn, axis=-1)
        attn = self.attn_drop(attn)
        # Matmul
        x = (attn.matmul(v)).transpose((0, 2, 1, 3)).reshape((-1, N, C))
        # 线性变换
        x = self.proj(x)
        x = self.proj_drop(x)
        return x

class Mlp(nn.Layer):
    def __init__(self,
                 in_features,
                 hidden_features=None,
                 out_features=None,
                 act_layer=nn.GELU,
```

```
                 drop=0.):
        super().__init__()
        out_features = out_features or in_features
        hidden_features = hidden_features or in_features
        self.fc1 = nn.Linear(in_features, hidden_features)
        self.act = act_layer()
        self.fc2 = nn.Linear(hidden_features, out_features)
        self.drop = nn.Dropout(drop)

    def forward(self, x):
        # 输入层：线性变换
        x = self.fc1(x)
        # 应用激活函数
        x = self.act(x)
        # Dropout
        x = self.drop(x)
        # 输出层：线性变换
        x = self.fc2(x)
        # Dropout
        x = self.drop(x)
        return x

def drop_path(x, drop_prob=0., training=False):
    if drop_prob == 0. or not training:
        return x
    keep_prob = paddle.to_tensor(1 - drop_prob)
    shape = (paddle.shape(x)[0], ) + (1, ) * (x.ndim - 1)
    random_tensor = keep_prob + paddle.rand(shape, dtype=x.dtype)
    random_tensor = paddle.floor(random_tensor)
    output = x.divide(keep_prob) * random_tensor
    return output

class DropPath(nn.Layer):
    def __init__(self, drop_prob=None):
        super(DropPath, self).__init__()
        self.drop_prob = drop_prob

    def forward(self, x):
        return drop_path(x, self.drop_prob, self.training)

class Block(nn.Layer):
    def __init__(self,
                 dim,
                 num_heads,
```

```
                 mlp_ratio=4.,
                 qkv_bias=False,
                 qk_scale=None,
                 drop=0.,
                 attn_drop=0.,
                 drop_path=0.,
                 act_layer=nn.GELU,
                 norm_layer='nn.LayerNorm',
                 epsilon=1e-5):
        super().__init__()
        self.norm1 = eval(norm_layer)(dim, epsilon=epsilon)
        # Multi-head Self-attention
        self.attn = Attention(
            dim,
            num_heads=num_heads,
            qkv_bias=qkv_bias,
            qk_scale=qk_scale,
            attn_drop=attn_drop,
            proj_drop=drop)
        # DropPath
        self.drop_path = DropPath(drop_path) if drop_path > 0. else Identity()
        self.norm2 = eval(norm_layer)(dim, epsilon=epsilon)
        mlp_hidden_dim = int(dim * mlp_ratio)
        self.mlp = Mlp(in_features=dim,
                       hidden_features=mlp_hidden_dim,
                       act_layer=act_layer,
                       drop=drop)

    def forward(self, x):
        # Multi-head Self-attention， Add， LayerNorm
        x = x + self.drop_path(self.attn(self.norm1(x)))
        # Feed Forward， Add， LayerNorm
        x = x + self.drop_path(self.mlp(self.norm2(x)))
        return x
```

==========

图像增强

```
# 在训练中进行数据增强很重要。
train_tfm = transforms.Compose([
    # 将图像大小调整为固定形状 (height = width = 128)
    transforms.Resize((128, 128)),
    # ---------- TODO ----------
    # 任务点 1：数据增强
```

```
    # 在此处添加你的代码，进行数据增强
    transforms.RandomResizedCrop(128, scale=(0.8, 1.0)),  # 随机裁剪并缩放
    transforms.RandomHorizontalFlip(prob=0.5),                # 随机水平翻转
    # transforms.RandomVerticalFlip(prob=0.2),                 # 随机垂直翻转
    transforms.RandomRotation(degrees=15),                  # 随机旋转
    transforms.ColorJitter(
        brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1
    ),  # 随机调整亮度、对比度、饱和度和色相
    transforms.ToTensor(),
])

# 我们不需要在测试和验证中进行扩充。
# 只需要调整 PIL 图像的大小并将其转换为 Tensor。
test_tfm = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
])
```

==========

ResNet18、ResNet34

```
    model = paddle.vision.models.resnet18(pretrained=False)
    # model = paddle.vision.models.resnet34(pretrained=False)
```

==========

半监督

```
    from paddle.io import TensorDataset

    def get_pseudo_labels(dataset, model, threshold=0.65):
        # 此函数使用给定模型生成数据集的伪标签。
        # 它返回一个 DatasetFolder 实例，其中包含预测图片 logits 置信度超过给定阈值的图片。

        # 构造一个数据加载器。
        data_loader = DataLoader(dataset, batch_size=batch_size, shuffle=False, drop_last=False)

        # 确保模型处于 eval 模型）禁止训练。
        model.eval()
        # 定义 softmax 函数
        softmax = nn.Softmax(axis=-1)
        pseudo_images = []
        pseudo_labels = []
        # 遍历数据集
        for batch_id, data in enumerate(data_loader):
            batch_img, _ = data
```

```
            # 使用 paddle.no_grad() 加速前向过程。
            with paddle.no_grad():
                logits = model(batch_img)

            # 通过对 logits 应用 softmax 来获得概率分布。
            probs = softmax(logits)
            # 获取每张图片的最大概率及其对应的类别
            max_probs = paddle.max(probs, axis=1)
            predicted_labels = paddle.argmax(probs, axis=1)

            for i in range(len(max_probs)):
                if max_probs[i] >= threshold:
                    pseudo_images.append(batch_img[i].numpy())  # 保存图片
                    pseudo_labels.append(predicted_labels[i].numpy())  # 保存伪标签

        # 构造新的数据集
        pseudo_images = paddle.to_tensor(pseudo_images)
        pseudo_labels = paddle.to_tensor(pseudo_labels)
        pseudo_dataset = TensorDataset([pseudo_images, pseudo_labels])
            # ---------------------------------
        return pseudo_dataset
```

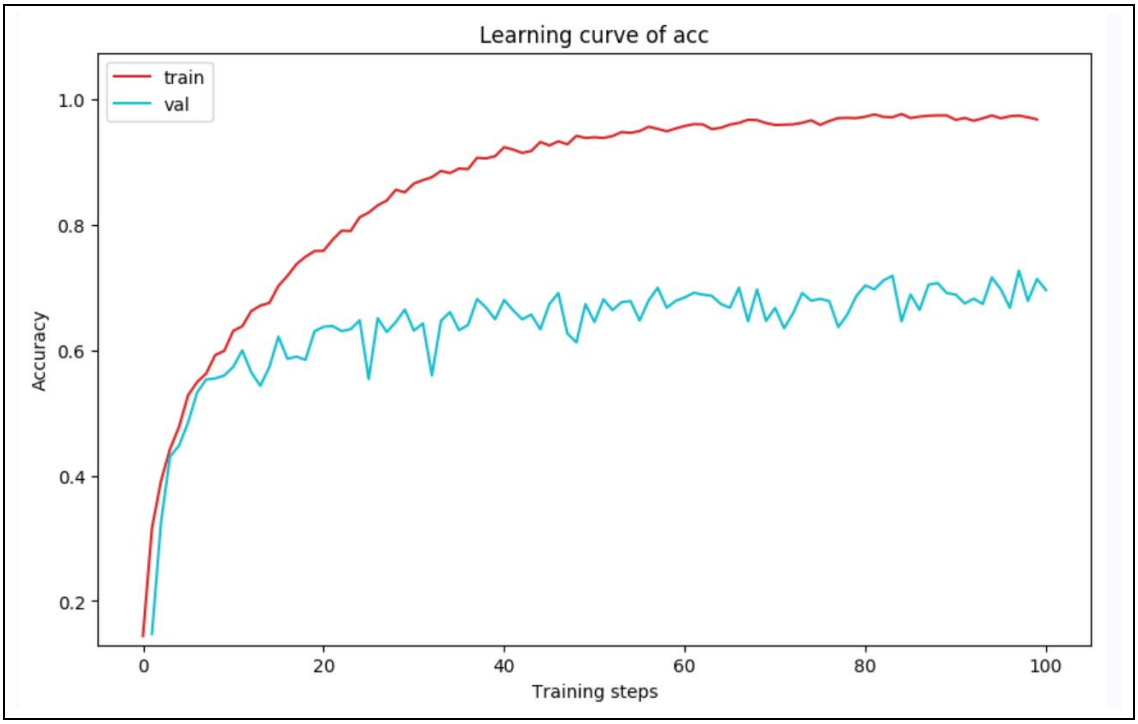
四、实验结果

（实验结果，必填）

==========

通过调整学习率并使用 ResNet18 进行训练可以得到不错的正确率

之后加入半监督学习，公有测试集上可以达到 0.7178 的正确率



五、总结

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

==========

实验结果：达到了预期目标，正确率超过了 70%，但仍然没有达到更高，更加符合实际需要的结果。

改进方向：可以进行的改进包括加入预训练模型（实测可以达到 90%的正确率）、调整模型结构以及使用更加实用的数据增强方式。

遇到的困难及解决方式：在进行 CNN 网络修改的时候，需要查阅大量资料，找到一个符合题目特点的模型，并进行一系列超参数的调整，这对于我理解模型训练有很大的帮助；另外，学习 ViT 和半监督也是实践结合理论的重要步骤；当然还有一个困难就是 paddlepaddle 平台算力，这也让我能够对于调整参数和模型更加慎重，去找寻最优的效果，而不是盲目试错。