在 kaggle 手写数字识别任务上进行

!!! 本次任务在第一次任务基础上进行，可对第一次 notebook 进行相应修改即可 !!!

第二次任务

对该竞赛的代码，训练 与 推理 分开编写 (训练在一个notebook中，推理在另一个notebook中)

导出图片(使用 kaggle 创建新的 notebook 进行编写)

1. 在该 notebook 中完成以下要求：将train.csv中的数据 reshape 成 28*28 的图像数据，图像名称可用对应在.csv中的index代替如：1.jpg, 2.jpg，然后将图像打包成压缩包下载到本地，将压缩包上传至训练的 notebook 中作为图像数据(test.csv中的数据不用进行转换，这里的打包只针对训练的notebook)，在训练过程中调用图像数据从上传的数据集中调用。(对于在第一次任务中按顺序取 前80% 数据作为训练集，后20% 数据作为验证的同学，可以直接从train.csv的index对应到图像中的数据；对于使用 train_test_split 操作的同学，这里的对应关系自己想想办法将图像的标签与图像对应起来即可)。对于图像数据的打开推荐使用 PIL 库中的 Image.open()打开，再转换成 numpy 形式；也可使用 cv2 来读取。(这里希望大家能够掌握从线上下载数据集，以及 Image 和 cv2 的读取图像等方法)

训练 notebook 本次要求如下:

1. 对 Dataset DataLoader 进行相应的修改，使其返回图像格式的数据
2. 数据增强：使用 albumentations库 进行数据增强操作，对于 albumentations库 的使用可以参考这里 --> Albumentations数据增强方法
3. 转为tensor格式：使用 albumentations库 中的 ToTensorV2 操作将图像转换为 tensor 格式
4. 模型：使用 timm库 创建神经网络模型，这次任务要求使用型号为 tf_efficientnetv2_s.in21k_ft_in1k 的模型
    - 要求使用预训练权重
    - 预训练权重一般在 3通道的 RGB 图像上训练获得，所以对于本次任务原图像只有 1channel 的灰度图来说无法直接传入，可以将原灰度图的通道复制3份按通道拼接成为 3通道图像 再传入模型。
    - 更多关于 timm库 中的所有模型可以参考这里 --> timm (需要科学上网进入)
5. 将 timm 创建的 tf_efficientnetv2_s.in21k_ft_in1k 模型中最后的全局池化层换成 GeMPooling 层
    - GeMPool 的代码提供如下：

```
class GeMPool(nn.Module):
    def __init__(self, p=3, eps=1e-6):
        super(GeMPool, self).__init__()
        self.p = nn.Parameter(torch.ones(1) * p)
        self.eps = eps

    def forward(self, x):
        return self.gem(x, p=self.p, eps=self.eps)

    def gem(self, x, p=3, eps=1e-6):
```

```
        return torch.mean(x.clamp(min=eps).pow(p), dim=(-2,
-1)).pow(1./p)

    def __repr__(self):
        return self.__class__.__name__ + f'(p={self.p.data.tolist()
[0]:.4f}, eps={self.eps})'
```

6. 修改模型最后的分类层，使其能够输出本次任务需求的 10 输出
7. 学习率调度器(scheduler):
    - 本次任务选用 CosineAnnealingLR 调度器，可在 torch.optim 中引用，如下：

```
from torch.optim import lr_scheduler

scheduler = lr_scheduler.CosineAnnealingLR(optimizer, T_max=T_max,
eta_min=min_lr)
```

    - 针对 scheduler 的出现，第一次任务中的训练函数需要做出相应修改，以使得学习率也可以在训练中更新
8. 优化器：本次任务将第一次任务中的 Adam优化器 替换为 AdamW
9. 自模型融合(SWA):
    - 将训练的到的权重参数进行 SWA融合
    - SWA 自模型融合 必须是相同的模型在相同的训练条件下训练出来的模型权重融合
    - 例如 本次训练中保存下来的 不同 epoch 的模型权重(满足相同训练条件 --> 相同的训练集，相同的模型)
    - SWA 是对模型参数的融合，所以必须满足以上条件
    - SWA 代码以及示例用法给出如下：

```
from collections import OrderedDict

def swa(ckpt_list:list, save_path:str):
    n_ckpt = len(ckpt_list)
    ckpts = []
    for path in ckpt_list:
        ckpts.append(torch.load(path))
    swa_ckpt = {}
    # 初始化 swa_ckpt
    for name in ckpts[0].keys():
        swa_ckpt[name] = torch.zeros_like(ckpts[0][name],
dtype=torch.float32)

    for name in swa_ckpt.keys():
        for i in range(n_ckpt):
            swa_ckpt[name] += ckpts[i][name]
        swa_ckpt[name] /= n_ckpt

    swa_ckpt = OrderedDict(swa_ckpt)
    torch.save(swa_ckpt, save_path)

    return swa_ckpt
```

```
if __name__ == "__main__":
    ckpt_list = ["/kaggle/working/output/cv1_epoch5.pth",
                 "/kaggle/working/output/cv2_epoch6.pth",
                 "/kaggle/working/output/cv3_epoch9.pth"]
    save_path = "/kaggle/working/swa.pth"

    # 执行 SWA 操作
    swa_ckpt = swa(ckpt_list, save_path)
    print(f"\nSWA 执行成功：权重保存至 {save_path}")
```

10. 将保存得到的 swa.pth 下载到本地再传入 infer 的 notebook 作为推理使用的权重参数

推理 notebook 本次要求如下:

1. 在第一次任务的推理的 notebook 上进行对应的修改
2. 对于推理：可直接将 test.csv 中的表格数据转换成 28*28 的图像数据，不需要进行打包下载操作
3. 模型：模型采用与训练相同的模型，此时 pretrained 参数应为 False，因为 kaggle 中进行提交的 notebook 不允许连接网络，设置 pretrained=True 会报错
4. 推理：推理时使用通过训练 notebook 得到的 swa.pth 作为权重