在 kaggle 手写数字识别任务上进行

本次任务在第二次任务基础上进行

第三次任务

> 本次任务重点如下:

- **使用 kfold oof计算 local cv**
- **使用 timm库中更丰富的模型(如：efficientnet, convnext, edgenext, tiny_vit, vit, eva)**
- **使用带有 warm up 的学习率调度器**
- **使用多卡并行化训练**
- **使用标签平滑技巧**
- **TTA (选做，了解即可)**
- **一个简单的可学习融合模型 (选做，了解即可)**
- **探索 learning_rate 与 batch_size 的关系(重点)**

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)

训练 notebook 本次要求如下:

```
**==kfold中的 k设为5==**
```

1. 在 Data Progress阶段，对 train.csv 贴 kfold，可调用 sklearn中的接口(参考资料：sklearn函数：KFold（分割训练集和测试集） 以及 python sklearn中KFold与StratifiedKFold，除了其中提到的 kf、skf、还有 sgkf等按组贴kfold的方法)
    - **建议将贴好kfold的.csv文件保存下来，之后的训练都使用相同的kfold，防止数据泄露，这样得到的local cv也更有参考价值**
2. 将标签转换为独热编码格式(因为要使用标签平滑，关于标签平滑可参考：label smooth标签平滑的理解)，标签平滑可在 Dataset中加入，也可在训练迭代batch的时候加入(根据你的个人习惯选择)
    - **注意：标签平滑一般只针对训练部分，验证部分不涉及标签平滑**
3. 模型：本节大家可以多尝试 timm 中的其他模型，因为此数据集的图片较小，CNN的性能应该会优于vit，推荐大家使用CNN
    - 常用CNN模型有 efficientnet, convnext, edgenext等，因为为了节约时间，提供给大家的参考代码中使用的是 efficientnet_b0, convnext_atto等轻量化模型型号，大家也可尝试其他模型型号
    - 如果使用 vit 推荐大家使用 tiny_vit，其他 vit模型可能会限制输入图像的大小需要经过一定调整才能输入。
4. 在训练与验证函数中，使用并行化训练，只需将 **模型** 与 **数据** load到多卡上即可，torch中有直接的接口可调用，可参考：【Pytorch】一文向您详细介绍 torch.nn.DataParallel() 的作用和用法

- 另外在训练中因为标签变为了独热编码形式，所以计算acc时 应对label取argmax 得到正确标签再计算acc
- 因为涉及到 kfold的训练，所以保存 .pth文件的时候建议大家对文件命名清楚一点，让自己能区分是哪个fold训出来的模型

5. 使用 带 warm up 的学习率调度器：

   - 带 warm up 的学习率调度器代码提供如下：

```
class CosineAnnealingWithWarmupLR(_LRScheduler):
    def __init__(self, optimizer, T_max, eta_min=0, warmup_epochs=10,
last_epoch=-1):
        self.T_max = T_max
        self.eta_min = eta_min
        self.warmup_epochs = warmup_epochs
        self.cosine_epochs = T_max - warmup_epochs
        super(CosineAnnealingWithWarmupLR, self).__init__(optimizer,
last_epoch)

    def get_lr(self):
        if self.last_epoch < self.warmup_epochs:
            # Linear warmup
            return [(base_lr * (self.last_epoch + 1) /
self.warmup_epochs) for base_lr in self.base_lrs]
        else:
            # Cosine annealing
            cosine_epoch = self.last_epoch - self.warmup_epochs
            return [self.eta_min + (base_lr - self.eta_min) * (1 +
math.cos(math.pi * cosine_epoch / self.cosine_epochs)) / 2 for base_lr in
self.base_lrs]

# lr scheduler
def fetch_scheduler(optimizer, T_max, min_lr):
    if CONFIG.scheduler == 'CosineAnnealingLR': # 学习率根据 cos 函数特性下
降，可以观察最后 logs 画出来的图像中 学习率的变化
        scheduler = lr_scheduler.CosineAnnealingLR(optimizer,T_max=T_max,
                                                   eta_min=min_lr)
    elif CONFIG.scheduler == 'CosineAnnealingWithWarmupLR':
        scheduler = CosineAnnealingWithWarmupLR(optimizer, T_max=T_max,
eta_min=min_lr,

warmup_epochs=T_max//CONFIG.epochs)
    elif CONFIG.scheduler == None:
        return None

    return scheduler

# 指定优化器为 AdamW
optimizer = torch.optim.AdamW(model.parameters(),
lr=CONFIG.learning_rate, weight_decay=CONFIG.weight_decay)
scheduler = fetch_scheduler(optimizer, T_max=CONFIG.T_max,
min_lr=CONFIG.min_lr)
```

6. 开始训练，遍历 k个fold，每次训练使用其中一个fold作为验证集，其他fold作为训练集，每一折训练完成后，使用最好的模型将验证集推理并将结果保存下来 最后将k组推理结果拼接成 oof 与正确标签计算 acc 得到 local cv

7. **探索 learning_rate 与 batch_size 之间的关系(大家自己实验即可)**：

    - 控制变量法：固定batch_size如：8, 16 32 64 ...， 调节 learning_rate 如：1e-3, 5e-4, 2e-4, 1e-4, 1e-5等(min_lr也做出相应调整), 大概能得出一个较好的学习率组合即可(不用调的很精确，大概实验3，4次就差不多了，这种实验会很费时间)

推理 notebook 本次要求如下:

1. 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
2. 加载模型：一共需要实例化 k个模型(k就是训练时使用的kfold的k值)，并分别将训练得到的 k个权重load 到模型中
3. 推理：将测试集用 k个模型分别推理，并将得到的结果进行取均值融合，将融合的结果转成.csv并提交
4. TTA(选做)：在infer的notebook最后有相关代码，取消注释后可直接运行，关于 TTA 可以参考：生动理解深度学习精度提升利器——测试时增强（TTA）
    - 除 TTA 外，在推理之前，也可对每一折的模型进行 SWA，这些都是可以提分的小技巧

提供给大家的 第3个 notebook(选做)：

此notebook包含训练与推理阶段，主要使用了一个简单的可学习权重的融合模型，用于融合多个模型，大家可以学习一下
核心模型：

```
class ensemblemodel(nn.Module):
    def __init__(self, n_models) -> None:
        super(ensemblemodel, self).__init__()

        self.weights = nn.Parameter(torch.randn(n_models))

        self.softmax = nn.Softmax(dim=-1)

    def forward(self, x):
        x = x.permute(0, 2, 1) # (batch, n_model, pred_prob) ---> (batch,
pred_prob, n_model)
        Identity = x
        weights = self.softmax(self.weights) # 计算每个模型的权重
        _tmp = Identity * weights # 用每个模型的预测结果 x 其对应的权重
        output = _tmp.sum(dim=2, keepdim=True).flatten(1) # 加权求和再降维
(batch, pred_prob, 1) ---> (batch, pred_prob)
        return output
```

该模型通过对不同模型的结果进行加权融合得到结果，通过反向传播 梯度下降学习得到最优的权重选择

> 此 notebook 使用的模型是通过 <308 Digit Recognizer Train Baseline 3> 训练得到的两组不同的模型(tf_efficientnet_b0.ns_jft_in1k 与 convnext_atto.d2_in1k)，每组模型使用 k=5的kfold训练，

一共是 2×5=10个模型

对于这个 notebook 大家可以将其中模型换成你训练出来的模型，也可对这个简单的融合模型进行优化改写，学习率可调小之类的操作 得出最佳的融合权重参数，并将其与手动融合的结果进行对比，观察local cv的好坏