学生实验报告

学号	1120192933	学院	计算机学院
姓名	李桐	专业	人工智能

商品识别模型优化

1 实验目的

（1）理解各个深度模型的优劣。
（2）理解优化器、损失函数、学习率策略对模型的影响。
（3）理解数据预处理对模型的影响。

2 实验原理

（1）深度学习中的优化器。
（2）深度学习中的损失函数。
（3）深度学习中的学习率策略。
（4）深度学习中的数据增强

3 实验条件与环境

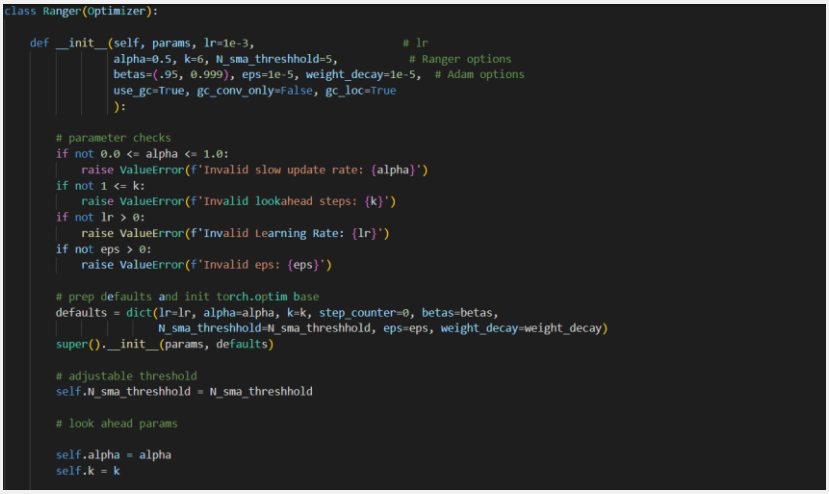
要求	名称	版本要求	备注
编程语言	python	3.6 以上	
开发环境	dsw	无要求	
第三方工具包/库/插件	opencv-python	4.5 以上	
第三方工具包/库/插件	tqdm	4.32	

第三方工具包/库/插件	pytorch	1.0 以上	
第三方工具包/库/插件	mmdetection	1.2	
其他工具	无	无要求	
硬件环境	台式机、笔记本均可	无要求	

4 实验步骤及操作

步骤序号	1
步骤名称	测试不同模型对于检测模型的训练效果
步骤描述	测试不同模型对于检测模型的 MAP。
代码及讲解	就是运行代码呗，这个是训练的 python mmdetection/tools/train.py 3-1.py，训练完了就自己在测试集上测试了。如果是都训练好再测试，可以用这样的一段：python mmdetection/tools/test.py 3-1.py ./work_dirs/3/i.pth --format-only --options='jsonfile_prefix=./result/i_fs_cascade'，修改模型路径即可。

步骤序号	2
步骤名称	测试不同模型对于识别模型的训练效果
步骤描述	测试不同模型对于识别模型的准确率。
代码及讲解	这个是用 pytoch 写的，所以直接运行文件即可，可以使用 python 6.py。在文件里面修改模型或者调用函数。

步骤序号	3
步骤名称	测试不同优化器对识别模型的训练效果
步骤描述	分别使用 adam、RAdam+lookahead 和 Ranger 测试训练效果。
代码及讲解	 RAdam 可以说是优化者在培训开始时建立的最佳基础。RAdam 利用动态整流器根据方差调整 Adam 的自适应动量，并有效地提供

```
class Ranger(Optimizer):

    def __init__(self, params, lr=1e-3,                    # lr
                 alpha=0.5, k=6, N_sma_threshhold=5,           # Ranger options
                 betas=(.95, 0.999), eps=1e-5, weight_decay=1e-5,  # Adam options
                 use_gc=True, gc_conv_only=False, gc_loc=True
                 ):

        # parameter checks
        if not 0.0 <= alpha <= 1.0:
            raise ValueError(f'Invalid slow update rate: {alpha}')
        if not 1 <= k:
            raise ValueError(f'Invalid lookahead steps: {k}')
        if not lr > 0:
            raise ValueError(f'Invalid Learning Rate: {lr}')
        if not eps > 0:
            raise ValueError(f'Invalid eps: {eps}')

        # prep defaults and init torch.optim base
        defaults = dict(lr=lr, alpha=alpha, k=k, step_counter=0, betas=betas,
                        N_sma_threshhold=N_sma_threshhold, eps=eps, weight_decay=weight_decay)
        super().__init__(params, defaults)

        # adjustable threshold
        self.N_sma_threshhold = N_sma_threshhold

        # look ahead params

        self.alpha = alpha
        self.k = k
```

	自动预热，根据当前数据集定制，以确保扎实的训练开始。
	LookAhead 受到深度神经网络损失表面理解的最新进展的启发，并在整个训练期间提供了稳健和稳定探索的突破。“减少了对广泛超参数调整的需求”，同时实现“以最小的计算开销实现不同深度学习任务的更快收敛”。
	因此，两者都在深度学习优化的不同方面提供了突破，并且这种组合具有高度协同性，可能为您的深度学习结果提供最佳的两种改进。因此，对更加稳定和强大的优化方法的追求仍在继续，通过结合两项最新突破（RAdam + LookAhead），Ranger 的整合有望为深度学习提供又一步。
	这里根据 Ranger 原论文构建了优化器，实现了 RAdam+lookahead 的功能。与 torch 提供的优化器的实例化、使用方法相同。
	`def __setstate__(self, state):`
	`print("set state called")`
	`super(Ranger, self).__setstate__(state)`
	`def step(self, closure=None):`
	`loss = None`
	`for group in self.param_groups:`
	`for p in group['params']:`
	`if p.grad is None:`
	`continue`
	`grad = p.grad.data.float()`
	`if grad.is_sparse:`
	`raise RuntimeError(`
	`'Ranger optimizer does not support sparse gradients')`
	这里是用了参数的方法，按照输入的参数选择不同的优化器。
	`if optimizerkind=='Adam':`
	`optimizer=torch.optim.Adam(params=filter(lambda p: p.requires_grad, model.parameters()), lr=0.01)`
	`elif optimizerkind=='Ranger':`
	`optimizer =Ranger(params=model.parameters())`

步骤序号	4
步骤名称	测试不同损失函数对识别模型的训练效果
步骤描述	分别比较 MultiSimilarityLoss 和 TripletLoss， CrossEntropyLabelSmooth 和 cross_entropy 的训练效果，并尝试修改加权的权重比较训练效果。
代码及讲解	下面将按照倒叙介绍这几种损失函数，即顺序为 CrossEntropyLabelSmooth、TripletLoss、MultiSimilarityLoss。（因为这样排版好看一些，cross_entropy 太简单了，而且可以看作是平滑系数为 0 的 CrossEntropyLabelSmooth，就不过多说明。） CrossEntropyLabelSmooth：Label Smoothing 也称之为标签平滑，其实是一种防止过拟合的正则化方法。传统的分类 loss 采用 softmax loss，先对全连接层的输出计算 softmax，视为各类别的置信度概率，再利用交叉熵计算损失。在这个过程中尽可能使得各样本在正确类别上的输出概率为 1，这要使得对应的 z 值为+∞，这拉大了其与其他类别间的距离。 现在假设一个多分类任务标签是[1,0,0]，如果它本身的 label 的出现了问题，这对模型的伤害是非常大的，因为在训练的过程中强行学习一个非本类的样本，并且让其概率非常高，这会影响对后验概率的估计。并且有时候类与类之间的并不是毫无关联，如果鼓励输出的

概率间相差过大，这会导致一定程度上的过拟合。

因此 Label Smoothing 的想法是让目标不再是 one-hot 标签。

使得 softmax 损失中的概率优目标不再为 1 和 0，同时 z 值的最优解也不再是正无穷大，而是一个具体的数值。这在一定程度上避免了过拟合，也缓解了错误标签带来的影响。

```python
class CrossEntropyLabelSmooth(nn.Module):
    """
    用于指定带标签平滑的交叉熵公式 ,用于指定带标签平滑的交叉熵公式
    """

    def __init__(self, num_classes, epsilon=0.2, use_gpu=True):
        """
        __init__()方法参数有num_classes与epsilon
        第一个参数指定分类数量
        第二参数即标签平滑公式中的epsilon  这里是对应的标签平滑的过程的。
        """
        super(CrossEntropyLabelSmooth, self).__init__()
        self.num_classes = num_classes  # num_classes = 8
        self.epsilon = epsilon # epsilon = 0.1
        self.use_gpu = use_gpu # 是否是要来使用gpu的过程的。
        self.logsoftmax = nn.LogSoftmax(dim=1) # 把相应的Softmax在来通过log的形式的。

    def forward(self, inputs, targets):
        """
        Args:
            inputs: prediction matrix (before softmax) with shape (batch_size, num_classes)
            targets: ground truth labels with shape (num_classes) 是来对应的真实的标签的。
        """
        log_probs = self.logsoftmax(inputs)  # torch.Size([4, 8])这里是得到批次为4，每一个属于这8个类别中那一个概率是最大的
        # scatter_是来沿着1，列方向上这个维度来进行索引的，zeros[4,8]的列数要与scatter这边的列数是相同的。
        # ongTensor中的index最大值应与zeros(4, 8)行数相一致
        #targets = torch.zeros(log_probs.size()).scatter_(1, targets.unsqueeze(1).data, 1) # 这里是来输出相应的标签信息的。
        #torch.topk(pre_labels, 1)[1].squeeze(1)
        #targets = torch.zeros(log_probs.size()).scatter_(1, torch.topk(log_probs, 1)[1].squeeze(1).data, 1) # 这里是来输出相应的标签
        #print(targets)
        #if self.use_gpu:
        #targets = targets.cuda()  #如果是存在gpu话，就是放在gpu上面来进行计算的过程的。
        # y = (1 - epsilon) * y + epsilon / K.
        targets = (1 - self.epsilon) * targets + self.epsilon / self.num_classes  # 这里是来对应的标签平滑化的过程的。
        #print(targets)
        loss = (- targets * log_probs).mean(0).sum()
```

TripletLoss：Triplet loss 最初是在 FaceNet: A Unified Embedding for Face Recognition and Clustering 论文中提出的，可以学到较好的人脸的 embedding。

softmax 最终的类别数是确定的，而 Triplet loss 学到的是一个好的 embedding，相似的图像在 embedding 空间里是相近的，可以判断是否是同一个人脸。

```python
    n = inputs.size(0)#取出输入的batch
    # Compute pairwise distance, replace by the official when merged
    #计算距离矩阵，其实就是计算两个2048维之间的距离平方(a-b)**2=a^2+b^2-2ab
    #[1,2,3]*[1,2,3]=[1,4,9].sum()=14  点乘

    dist = torch.pow(inputs, 2).sum(dim=1, keepdim=True).expand(n, n)
    dist = dist + dist.t()
    dist.addmm_(1, -2, inputs, inputs.t())#生成距离矩阵32x32，.t()表示转置
    dist = dist.clamp(min=1e-12).sqrt()  # for numerical stability#clamp(min=1e-12)加这个防止矩阵中有0，对梯度下降不好
    # For each anchor, find the hardest positive and negative
    mask = targets.expand(n, n).eq(targets.expand(n, n).t())#利用target标签的expand，并eq，获得mask的范围，由0，1组成，，红色1表示是
    dist_ap, dist_an = [], []#用来存放ap，an
    for i in range(n):#i表示行
        # dist[i][mask[i]],,i=0时，取mask的第一行，取距离矩阵的第一行，然后得到tensor([1.0000e-06, 1.0000e-06, 1.0000e-06, 1.0000e-0
        dist_ap.append(dist[i][mask[i]].max().unsqueeze(0))#取某一行中，红色区域的最大值，mask前4个是1，与dist相乘
        dist_an.append(dist[i][mask[i] == 0].min().unsqueeze(0))#取某一行，绿色区域的最小值,加一个.unsqueeze(0)将其变成带有维度的tens
    dist_ap = torch.cat(dist_ap)
    dist_an = torch.cat(dist_an)
    # Compute ranking hinge loss
    y = torch.ones_like(dist_an)#y是个权重，长度像dist-an
    loss = self.ranking_loss(dist_an, dist_ap, y) #ID损失：交叉商输入的是32xf f.shape=分类数,然后loss用于计算损失
                                                  #度量三元组，输入的是dist_an（从距离矩阵中，挑出一行（即一个ID）的最大距离），dist_a
                                                  #ranking_loss输入 an ap margin y:倍率  loss： Relu(ap - anxy + margin)这个relu就起
    # from IPython import embed
    # embed()
    return loss
```

MultiSimilarityLoss：是对比损失函数的一种变体，不再是使用绝对距离，还要考虑 batch 中其他样本对的整体距离分布来对损失进行加权。

```python
def forward(self, feats, labels):
    assert feats.size(0) == labels.size(0), \
        f"feats.size(0): {feats.size(0)} is not equal to labels.size(0): {labels.size(0)}"
    batch_size = feats.size(0)
    feats = nn.functional.normalize(feats, p=2, dim=1)

    # Shape: batchsize * batch size
    sim_mat = torch.matmul(feats, torch.t(feats))

    epsilon = 1e-5
    loss = list()

    mask = labels.expand(batch_size, batch_size).eq(
        labels.expand(batch_size, batch_size).t())
    for i in range(batch_size):
        pos_pair_ = sim_mat[i][mask[i]]
        pos_pair_ = pos_pair_[pos_pair_ < 1 - epsilon]
        neg_pair_ = sim_mat[i][mask[i] == 0]

        neg_pair = neg_pair_[neg_pair_ + self.margin > min(pos_pair_)]
        pos_pair = pos_pair_[pos_pair_ - self.margin < max(neg_pair_)]

        if len(neg_pair) < 1 or len(pos_pair) < 1:
            continue

        # weighting step
        pos_loss = 1.0 / self.scale_pos * torch.log(
            1 + torch.sum(torch.exp(-self.scale_pos * (pos_pair - self.thresh))))
        neg_loss = 1.0 / self.scale_neg * torch.log(
            1 + torch.sum(torch.exp(self.scale_neg * (neg_pair - self.thresh))))
        loss.append(pos_loss + neg_loss)
        # pos_loss =
```

同样根据输入的参数，按照输入的参数选择不同的损失函数。

```python
if criterionkind=='cross_entropy':
    #criterion=nn.CrossEntropyLoss().to(device)
    criterion=CrossEntropyLabelSmooth(num_classes=23,epsilon=0)
elif criterionkind=='CrossEntropyLabelSmooth':
    criterion=CrossEntropyLabelSmooth(num_classes=23)
elif criterionkind=='MultiSimilarityLoss':
    criterion=MultiSimilarityLoss()
elif criterionkind=='TripletLoss':
    criterion=TripletLoss()
```

步骤序号	5
步骤名称	测试不同的学习率策略对识别模型的训练效果
步骤描述	比较 WarmupMultiStepLR 和 WarmupCosineLR 两种学习策略的训练效果。
代码及讲解	

```python
if warm_up_epochs and epoch < warm_up_epochs:
    warmup_percent_done = epoch / warm_up_epochs
    warmup_learning_rate = init_lr * warmup_percent_done  #gradual warmup_lr
    learning_rate = warmup_learning_rate
else:
    if warmkind=='cos':
        learning_rate = np.sin(learning_rate)  #预热学习率结束后,学习率呈sin衰减
    elif warmkind=='multistp':
        learning_rate = learning_rate**1.0001 #预热学习率结束后,学习率呈指数衰减(近似模拟指数衰减)
for p in optimizer.param_groups:
        p['lr'] = learning_rate
```

哈哈哈，我就是做到这一个实验的时候，才发现了第六个实验里面效果很差的原因的。

这两个是差不多的方法，不同的就是 warm 之后怎么衰减。一个是类似于指数的形式，一个是 cos 的形式。

步骤序号	6
步骤名称	测试数据增强对识别模型的训练效果
步骤描述	比较是否添加 Autoaugment 的模型训练效果

代码及讲解	

```
            self.transform = transforms.Compose([
                transforms.Resize((224, 224)),
                transforms.RandomHorizontalFlip(p=0.5),
                transforms.ToTensor(),
                transforms.Normalize(mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225])])  # 归一化
```

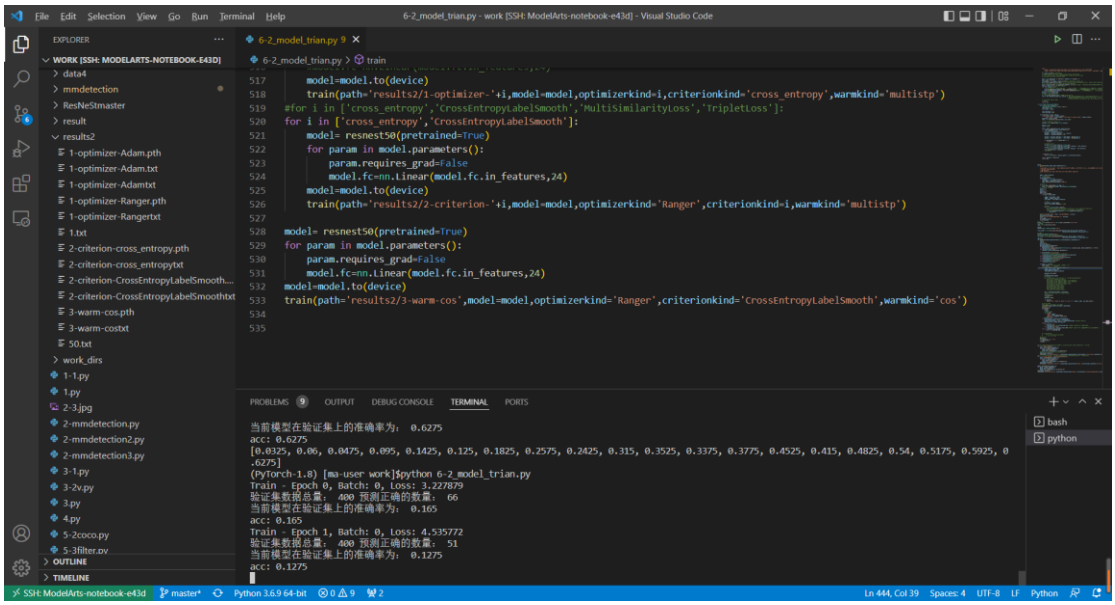
进行数据增强，很简单。

5 实验结果及分析讨论

（1）最终结果的具体结果（文字说明）

进行了不同轮次的训练、获得了数个结果，并且进行了对比。adam 和 Ranger 的区别最大，Ranger 的训练步伐会比较缓一些，准确率提升会比较慢，但是相对来说会比较稳定。学习率调整策略上，WarmupMultiStepLR 的效果稳定强于 WarmupCosineLR。进行数据增强的上上限强于不进行数据增强。

（2）最终结果界面截图（界面截图）

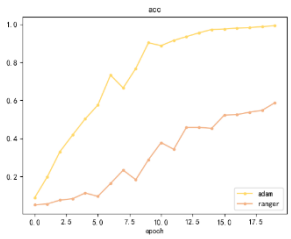


（3）最终结果的说明（注意事项或提醒）

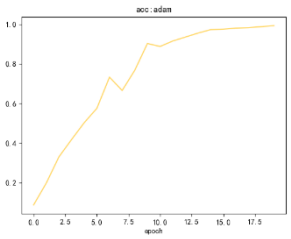
这个只截了一个图，因为其实都是类似的，也看不出来啥东西，主要看一下下面的解读吧。

（4）最终结果的解读与讨论

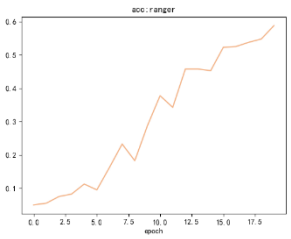
优化器：



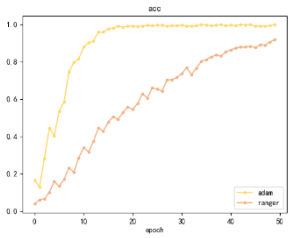
(1)汇总



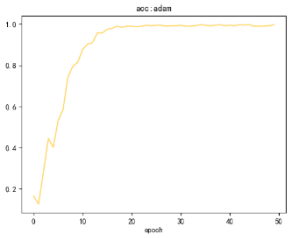
(2)adam



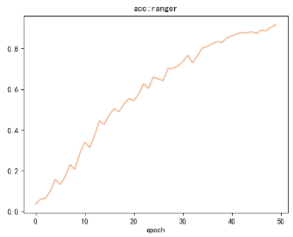
(3)ranger



(4)汇总



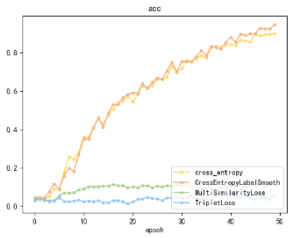
(5)adam



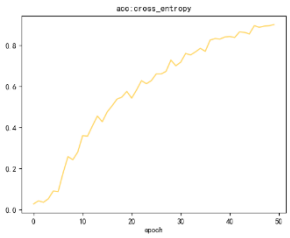
(6)ranger

优化器为 adam 以及 ranger，其他配置为损失函数为交叉熵(criterionkind='cross_entropy'), 学习率策略为'multistp'(warmkind='multistp') ，进行数据增强，batchsize=200。一开始只做了 20epoch 的，发现 ranger 效果比较差，但是按理不应该呀。发现准确率总体趋势是一直增加的，于是加做了 50epoch 的对比试验。Ranger 的训练步伐会比较缓一些，准确率提升会比较慢，但是相对来说会比较稳定。但是我其实更喜欢 adam，训练的比较快。

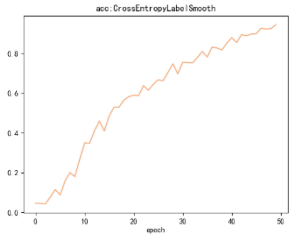
损失函数:



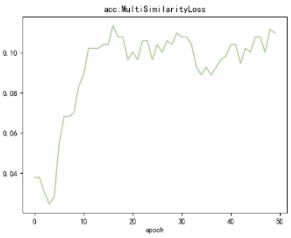
(1)汇总



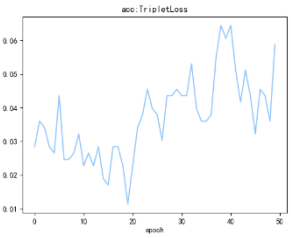
(2) cross_entropy



(3) CrossEntropyLabelSmooth



(4) MultiSimilarityLoss

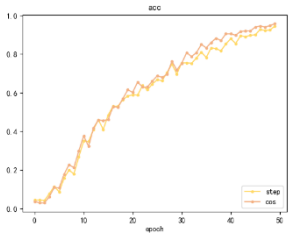


(5) TripletLoss

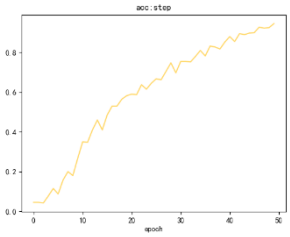
优化器为 ranger，学习率策略为'multistp'(warmkind='multistp') ，进行数据增强，损失函数分别为 cross_entropy、CrossEntropyLabelSmooth、MultiSimilarityLoss、TripletLoss，cross_entropy、CrossEntropyLabelSmooth 时 batchsize=200，MultiSimilarityLoss、TripletLoss 时 batchsize=24，训练 50epoch。

MultiSimilarityLoss、TripletLoss 需要和同一个 batch 里面的对比，所以 batchsize 与标签大小相同。但是我这里获得的结果很差，我认为一方面可能是它本来就差，还有一种可能是我的修改使得结果很差。我的修改方法可以详见第 6 节收获与体会。

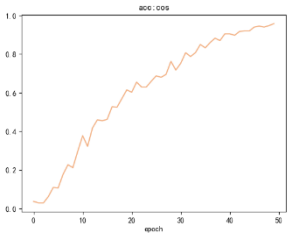
学习率策略:



(1)汇总



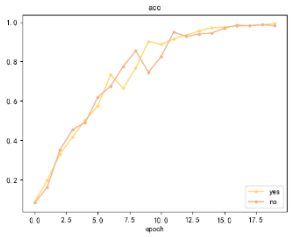
(2) WarmupMultiStepLR



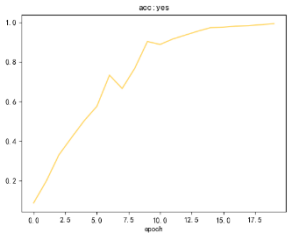
(3) WarmupCosineLR

优化器为 adam，损失函数为交叉熵标签平滑(criterionkind=' CrossEntropyLabelSmooth '),学习率策略分别为 WarmupMultiStepLR 和 WarmupCosineLR，进行数据增强，batchsize=200，进行 50epoch 训练。全期基本都是 WarmupCosineLR 的效果更好，这个没啥疑问了。

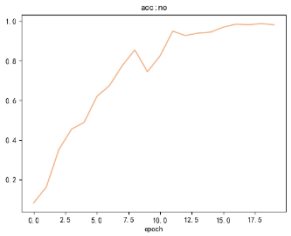
数据增强:



(1)汇总



(2)数据增强



(3)数据未增强

优化器为adam，损失函数为交叉熵(criterionkind='cross_entropy'),学习率策略为'multistp'，warmkind='multistp'，batchsize=200，分别在数据增强与不增强的情况下，进行 20epoch 的训练。数据增强在前期的效果稍差一些，可能是因为变化比较多，但是后期因为进行了数据增强，所以上限比不增强的时候要高。

6 收获与体会

进行对比试验，了解了深度学习中的优化器、损失函数、学习率策略、数据增强的效果。

这里遇到的问题其实就是一开始效果很差，正如在实验六中说的，一开始最高准确率只有 22%，后来发现是学习率调整策略写错了，所以出现了问题。解决了这个问题之后也就又进行了一遍本次实验，我哭死。实验都做完了，然后开始写报告了，去截图的时候才发现代码里面的学习率调整策略写错了。然后又是重新实验!!! 真的!!! 写代码一定要仔细!!!

另外还遇到了一些问题，这里说一下，使用 TripletLoss 出现 RuntimeError: operation does not have an identity:

```
Traceback (most recent call last):
  File "6-2_model_trian.py", line 596, in <module>
    train(path='results2/2-criterion-'+i,model=model,optimizerkind='Ranger',criterionkind=i,warmkind='multistp')
  File "6-2_model_trian.py", line 515, in train
    loss = criterion(pre_labels, true_labels)
  File "/home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages/torch/nn/modules/module.py", line 889, in _call_impl
    result = self.forward(*input, **kwargs)
  File "6-2_model_trian.py", line 334, in forward
    dist_an.append(dist[i][mask[i] == 0].min().unsqueeze(0))#取某一行，绿色区域的最小值,加一个.unsqueeze(0)将其变成带有维度的tensor
RuntimeError: operation does not have an identity.
(PyTorch-1.8) [ma-user work]$
```

Triplet loss 的优势在于细节区分，即当两个输入相似时，Triplet loss 能够更好地对细节进行建模，相当于加入了两个输入差异性差异的度量，学习到输入的更好表示。常用在人脸识别任务中。目的是做到非同类极相似样本的区分，比如说对兄弟二人的区分。存在 operation does not have an identity 是因为没有一个正样本一个负样本，可能的原因有没有进行数据打乱、batchsize 设置太小。

我这里调整了好久，还是没有办法，就改成了，如果出错，那就直接增加之前的均值:

```
for i in range(n):#i表示行
    # dist[i][mask[i]],,i=0时，取mask的第一行，取距离矩阵的第一行，然后得到tensor([1.0000e-06, 1.0000e-06, 1.0000e-06, 1.0000e-06
    dist_ap.append(dist[i][mask[i]].max().unsqueeze(0))#取某一行中，红色区域的最大值，mask前4个是1，与dist相乘
    #print(dist[i],mask[i]  )
    #print(dist[i][mask[i] == 0].min().unsqueeze(0))
    try:
        dist_an.append(dist[i][mask[i] == 0].min().unsqueeze(0))#取某一行，绿色区域的最小值,加一个.unsqueeze(0)将其变成带有维度的t
    except:
        #print(dist_an)#tensor([1.9137], device='cuda:0', grad_fn=<UnsqueezeBackward0>)
        #print(torch.mean(torch.tensor(dist_an)).view(-1))
        dist_an.append(torch.mean(torch.tensor(dist_an)).view(-1).cuda())
```

MultiSimilarityLoss 也有类似的问题，max 里面有可能是空列表，所以我直接设定为如果没有那就是和 tensor([0])作比较。这也可能是我的效果差的原因吧。

```
        neg_pair = neg_pair_[neg_pair_ + self.margin > min(pos_pair_)]
        #print(neg_pair_)
        try:
            pos_pair = pos_pair_[pos_pair_ - self.margin < max(neg_pair_)]
        except:
            pos_pair = pos_pair_[pos_pair_ - self.margin < max(torch.tensor([0]))]
```

7 备注及其他

无。