

半监督图像分类

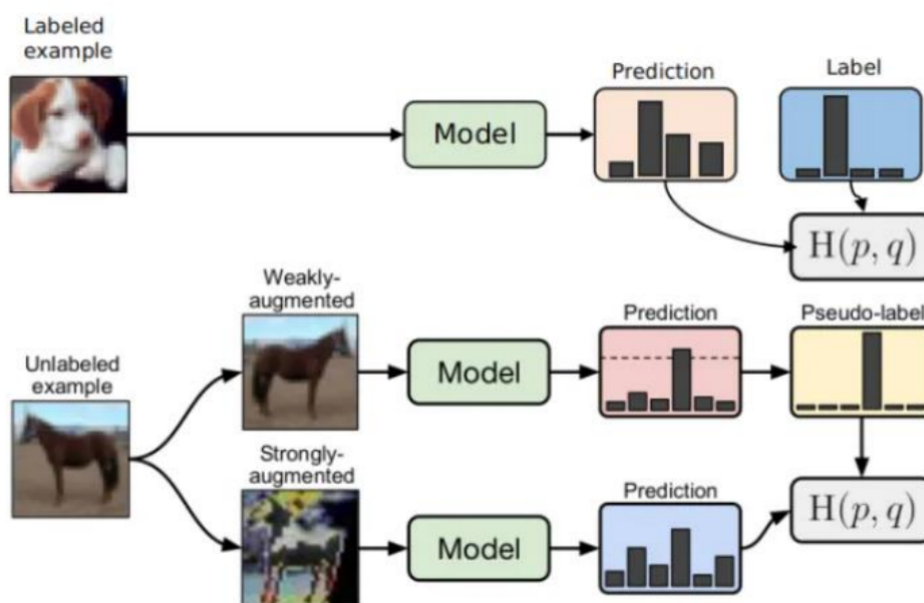
实验内容

1. 阅读原始论文和相关参考资料，基于 Pytorch 动手实现 FixMatch 半监督图像分类算法，在 CIFAR-10 进行半监督图像分类实验，报告算法在分别使用 40,250, 4000 张标注数据的情况下的图像分类结果
2. 按照原始论文的设置，FixMatch 使用 WideResNet-28-2 作为 Backbone 网络，即深度为 28，扩展因子为 2，使用 CIFAR-10 作为数据集，可以参考现有代码的实现，算法核心步骤不能直接照抄！
3. 使用 TorchSSL 中提供的 FixMatch 的实现进行半监督训练和测试，对比自己实现的算法和 TorchSSL 中的实现的效果

实验过程

一、FixMatch实现

算法步骤



1. 有标签数据训练:

- 对于每个有标签数据，计算模型预测与真实标签之间的交叉熵损失。

2. 无标签数据训练:

- 数据增强:
 - 对每个无标签数据样本，生成一个弱增强版本和一个强增强版本。
- 伪标签生成:
 - 将弱增强版本的图像输入模型，获取预测分布。
 - 从预测分布中提取伪标签（即选择概率最高的类别作为伪标签）。
- 一致性损失计算:
 - 将强增强版本的图像输入模型，获取预测分布。
 - 计算强增强版本的预测分布与伪标签之间的交叉熵损失。

3. 总损失计算和反向传播:

- 结合有标签数据的交叉熵损失和无标签数据的一致性损失，计算总损失。
- 使用反向传播算法更新模型参数。

核心代码

- 对无标签数据集使用弱增强与强增强:弱扩增与有标签的数据处理方式类似，直接使用翻转和裁剪技术；强扩增还使用了RandAugmentMC 算法进行数据增强

```
# 有标签数据
transform_labeled = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(size=32,
                           padding=int(32*0.125),
                           padding_mode='reflect'),
    transforms.ToTensor(),
    transforms.Normalize(mean=cifar10_mean, std=cifar10_std)
])

# 无标签弱增强
self.weak = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(size=32,
                           padding=int(32*0.125),
                           padding_mode='reflect')])

# 无标签强增强
self.strong = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(size=32,
                           padding=int(32*0.125),
                           padding_mode='reflect'),
    RandAugmentMC(n=2, m=10)])
```

- 训练基本框架如下：代码展示有所简化

```
# 获取数据集
labeled_dataset, unlabeled_dataset, test_dataset = DATASET_GETTERS[args.dataset]
(
    args, './data')

# 定义数据加载器
labeled_trainloader = DataLoader(
    labeled_dataset,
    sampler=RandomSampler(labeled_dataset),
    batch_size=args.batch_size,
    num_workers=4,
    drop_last=True)
unlabeled_trainloader = DataLoader(
    unlabeled_dataset,
    sampler=RandomSampler(unlabeled_dataset),
    batch_size=args.batch_size*args.mu,
    num_workers=4,
    drop_last=True)

# 定义优化器
optimizer = optim.SGD(grouped_parameters, lr=args.lr, momentum=0.9)
scheduler = get_cosine_schedule(optimizer, args.total_steps)

# 定义迭代器
labeled_iter = iter(labeled_trainloader)
```

```

unlabeled_iter = iter(unlabeled_trainloader)
# 进入循环
# 迭代得到训练数据
inputs_x, targets_x = labeled_iter.next()
(inputs_u_w, inputs_u_s), _ = unlabeled_iter.next()
# 输入数据
inputs = torch.cat((inputs_x, inputs_u_w, inputs_u_s))
# 分离输出数据
logits = model(inputs)
logits_x = logits[:batch_size] # 有标签
logits_u_w, logits_u_s = logits[batch_size:].chunk(2) # 无标签
# 损失函数cross_entropy
# 有标签损失:Lx
Lx = F.cross_entropy(logits_x, targets_x, reduction='mean')
# 无标签损失:以弱增强输入筛选出可训练数据,再用强增强输出计算损失函数Lu
pseudo_label = torch.softmax(logits_u_w.detach(), dim=-1)
max_probs, targets_u = torch.max(pseudo_label, dim=-1)
mask = max_probs.ge(args.threshold).float()
Lu = (F.cross_entropy(logits_u_s, targets_u, reduction='none') *
mask).mean()
loss = Lx + args.lambda_u * Lu
# 反向传播
loss.backward()
# 更新参数模型
optimizer.step()
scheduler.step()

```

• 训练细节

- **use-ema**: 指数移动平均模型, EMA 模型通过对模型参数进行指数加权平均来平滑参数更新过程, 从而减少参数更新的波动性, 提高模型的泛化能力。

如果不使用ema模型会出现以下loss为nan的情况

```

Train Epoch: 1/ 66. Iter: 500/ 500. LR: 0.0300. Data: 0.009s. Batch: 0.160s. Loss: 1.6778. Loss_x: 1.6752. Loss_u: 0.0025. Mask: 0.00. : 100%|█ 500/500 [01:19
Test Iter: 313/ 313. Data: 0.002s. Batch: 0.005s. Loss: 17.5113. top1: 17.27. top5: 71.08. : 100%|████████████████████ 313/313 [00:01:00:00, 206.48it/s]
Train Epoch: 2/ 66. Iter: 91/ 500. LR: 0.0300. Data: 0.026s. Batch: 0.179s. Loss: nan. Loss_x: nan. Loss_u: nan. Mask: 0.02. : 18%|█ 91/500 [00:14:01:05, 6

```

- **weight_decay**: 对所有不包含 **bias** 和 **bn** 的参数应用权重衰减, 来防止过拟合
- 关于数据如何输入模型, 考虑到如果有标签、无标签弱增强、无标签强增强 3 份数据分别输入网络, 在批量归一化中会出现 3 种不同分布, 使模型难以进一步拟合, 因此在源代码中采取将 3 种数据混合的方式, 使用 **interleave** 和 **de_interleave** 是在一个批次内混合标记数据和无标记数据, 以便共同计算批量归一化的统计量, 从而提高模型训练的稳定性和效果

```

# (a, b, c)->(a/size, size, b, c)->(size, a/size, b, c)->(size * a/size,
b, c)
def interleave(x, size):
    s = list(x.shape)
    return x.reshape([-1, size] + s[1:]).transpose(0, 1).reshape([-1] +
s[1:])
# 分离输出 interleave的逆过程
def de_interleave(x, size):
    s = list(x.shape)
    return x.reshape([size, -1] + s[1:]).transpose(0, 1).reshape([-1] +
s[1:])

```

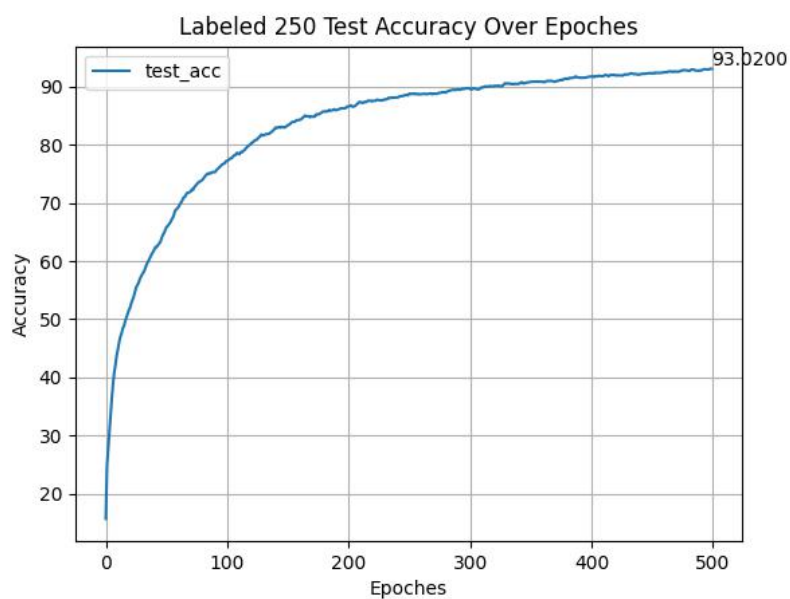
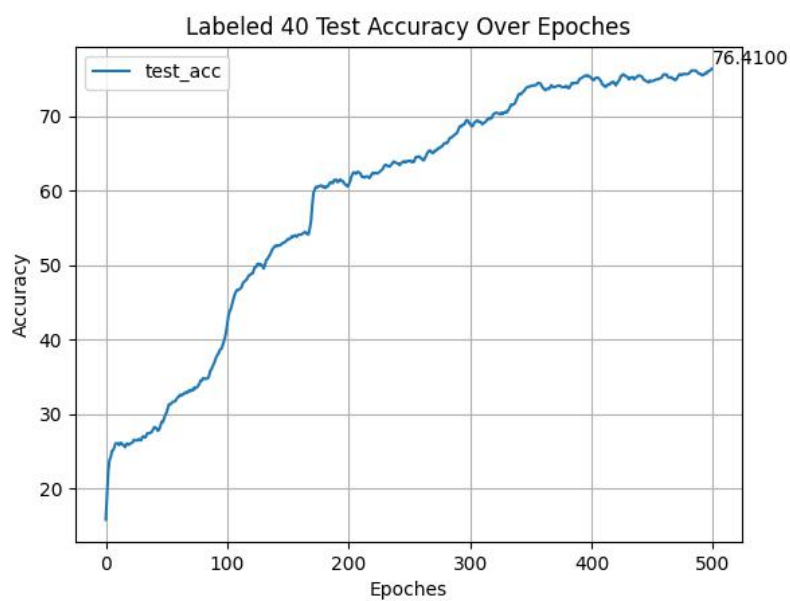
运行结果

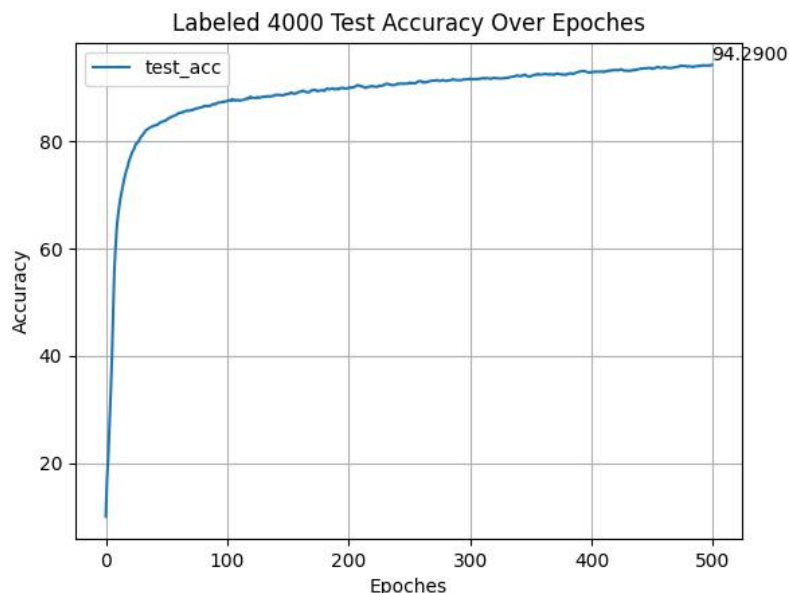
```
python train.py --num-labeled 4000 --epochs 500
```

开始运行:

```
(fix) jovyan@jupyter-21307347:~/Fix$ python train.py --num-labeled 4000 --epochs 500
Files already downloaded and verified
{'num_labeled': 4000, 'epochs': 500, 'wdecay': 0.0005, 'use_ema': True, 'ema_decay': 0.999, 'mu': 7, 'lambda_u': 1, 'lr': 0.03, 'batch_size': 32, 'threshold': 0.95, 'n_gpu': 4, 'device': device(type='cuda', index=0), 'eval_step': 224, 'total_steps': 112000}
**** Running training ****
Num Epochs = 500
Total optimization steps = 112000
labeled dataset = 4000
unlabeled dataset = 50000
test dataset = 10000
Train Epoch: 1/ 500. Iter: 224/ 224. LR: 0.0300. Data: 0.011s. Batch: 0.208s. Loss: 1.7954. Loss_x: 1.7952. Loss_u: 0.0002. Mask: 0.00. : 100%|█| 224/2
Test Iter: 313/ 313. Data: 0.002s. Batch: 0.004s. Loss: 2.3951. top1: 11.25. top5: 57.63. : 100%|██████████| 313/313 [00:01:00:00, 220.44it/s]
Train Epoch: 2/ 500. Iter: 222/ 224. LR: 0.0300. Data: 0.013s. Batch: 0.208s. Loss: 1.5337. Loss_x: 1.5310. Loss_u: 0.0028. Mask: 0.00. : 99%|█████████| 222/2
```

时间原因只在500epochs中比较不同标注数据量的分类结果对比:





其中有标签数量40在epochs=500内未能收敛，由于运行时间较长，且在之后epoch中未能及时记录运行数据，因此以上图片展示只展示到500个epochs，之后的训练效果可参考下图：标签数量40在训练到750epochs的测试准确率达到90.43%。

```
Test Iter: 313/ 313. Data: 0.002s. Batch: 0.005s. Loss: 0.5552. top1: 90.43. top5: 99.44. : 100% 313/313 [00:01<00:00, 193.
Train Epoch: 750/1000. Iter: 104/ 224. LR: 0.0154. Data: 0.331s. Batch: 0.547s. Loss: 0.1896. Loss_x: 0.0002. Loss_u: 0.1893.
```

二、TorchSSL对比

TorchSSL配置环境后运行

```
# labeled 4000
python fixmatch.py --c config/fixmatch/fixmatch_cifar10_4000_0.yaml

# labeled 250
python fixmatch.py --c config/fixmatch/fixmatch_cifar10_250_0.yaml

# labeled 40
python fixmatch.py --c config/fixmatch/fixmatch_cifar10_40_0.yaml
```

开始运行：

```
[2024-06-27 02:26:08,458 INFO] model saved: ./saved_models/fixmatch_cifar10_4000_0/latest_model.pth
/opt/conda/envs/ssl/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1245: UndefinedMetricWarning: Precision is ill-defined
and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  warn_prf(average, modifier, msg_start, len(result))
[2024-06-27 02:26:10,655 INFO] confusion matrix:
[[0.83  0.087 0.083 0.  0.  0.  0.  0.  0.  0. ]
 [0.959 0.01 0.031 0.  0.  0.  0.  0.  0.  0. ]
 [0.909 0.021 0.07  0.  0.  0.  0.  0.  0.  0. ]
 [0.95  0.019 0.031 0.  0.  0.  0.  0.  0.  0. ]
 [0.94  0.01 0.05  0.  0.  0.  0.  0.  0.  0. ]
 [0.923 0.032 0.045 0.  0.  0.  0.  0.  0.  0. ]
 [0.973 0.  0.027 0.  0.  0.  0.  0.  0.  0. ]
 [0.944 0.009 0.047 0.  0.  0.  0.  0.  0.  0. ]
 [0.867 0.065 0.068 0.  0.  0.  0.  0.  0.  0. ]
 [0.97  0.009 0.021 0.  0.  0.  0.  0.  0.  0. ]]
[2024-06-27 02:26:10,664 INFO] 0 iteration, USE_EMA: True, {'train/sup_loss': tensor(2.4592, device='cuda:1'), 'train/unsup_loss': tensor(0., device='cuda:1'), 'train/total_loss': tensor(2.4592, device='cuda:1'), 'train/mask_ratio': tensor(1., device='cuda:1'), 'lr': 0.0299999999999999974228, 'train/prefecth_time': 0.7478323364257813, 'train/run_time': 1.0183004150390624, 'eval/loss': tensor(13.4847, device='cuda:1'), 'eval/top-1-acc': 0.091, 'eval/top-5-acc': 0.5013, 'eval/precision': 0.027574393990794376, 'eval/recall': 0.091, 'eval/F1': 0.027260655222211883, 'eval/AUC': 0.4856872888888888}, BEST_EVAL_ACC: 0.091, at 0 iters
[2024-06-27 02:26:10,766 INFO] model saved: ./saved_models/fixmatch_cifar10_4000_0/model_best.pth
```

□

由于实验参数不完全相同，只对比最终测试结果：

labeled	40	250	4000
FixMatch	90.54	93.06	94.29
TorchSSL	90.58	92.58	94.05

可以看出实现的FixMatch算法与TorchSSL提供的FixMatch训练结果相差不大，TorchSSL使用默认参数没做过多调整，导致准确率略低。在40个样本时测试结果小于正常预期值，原因在于设置的epoches数偏小，训练未完全收敛。后续改进可参考FixMatch源代码中混合精度训练减少舍入误差的问题，以及调整其他参数比如使用余弦退火策略时加入预热阶段，调整batch-size大小观察对准确率的变化等。

三、对比 FixMatch 和 MixMatch

1. MixMatch 和 FixMatch 使用的半监督学习方法不同。MixMatch使用了 Mixup技术将一个样本与另一个随机样本的特征进行混合，并且通过最大化熵损失函数来优化网络，以提高模型的泛化能力；而 FixMatch则使用了自监督学习技术，选择置信度高的无标签数据加入到有标签数据中进行训练网络。
2. MixMatch 的训练过程需要迭代多次才能得到结果，每次迭代都需要对全部无标签数据进行处理；而 FixMatch 只需要迭代一次就能够输出，并且只对部分无标签数据进行处理。