FixMatch 半监督图像分类

在深度学习领域，图像分类任务是最基础又不失重要性重要的研究方向。传统的监督学习方法依赖大量的标记数据来训练性能良好的模型。然而，标记大规模数据集是成本高昂且耗时的，尤其是在需要专业知识进行精确标注的领域。因此，半监督学习方法应运而生，它们旨在利用少量的标记数据以及大量的未标记数据来训练模型，既减少了标注的需求，又保持了模型性能。

实验目的

1. 学习 FixMatch 算法作为半监督学习框架的构造思路，代码实现方法。
2. 探究在不同标记数据比例下，FixMatch 算法的性能变化。在多个不同的标记数据子集上运行，以评估算法对标记数据量的敏感度，并找出性能与数据量的关系。
3. 通过将自己改造的实现与官方实现进行对比，发掘自己实现的 FixMatch 在处理问题时的优势和不足。

实验要求

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
4. 提交源代码，并提交实验报告，描述实现过程中的主要算法部分，可以尝试分析对比 FixMatch 和其他半监督算法的不同点，例如 MixMatch 等。

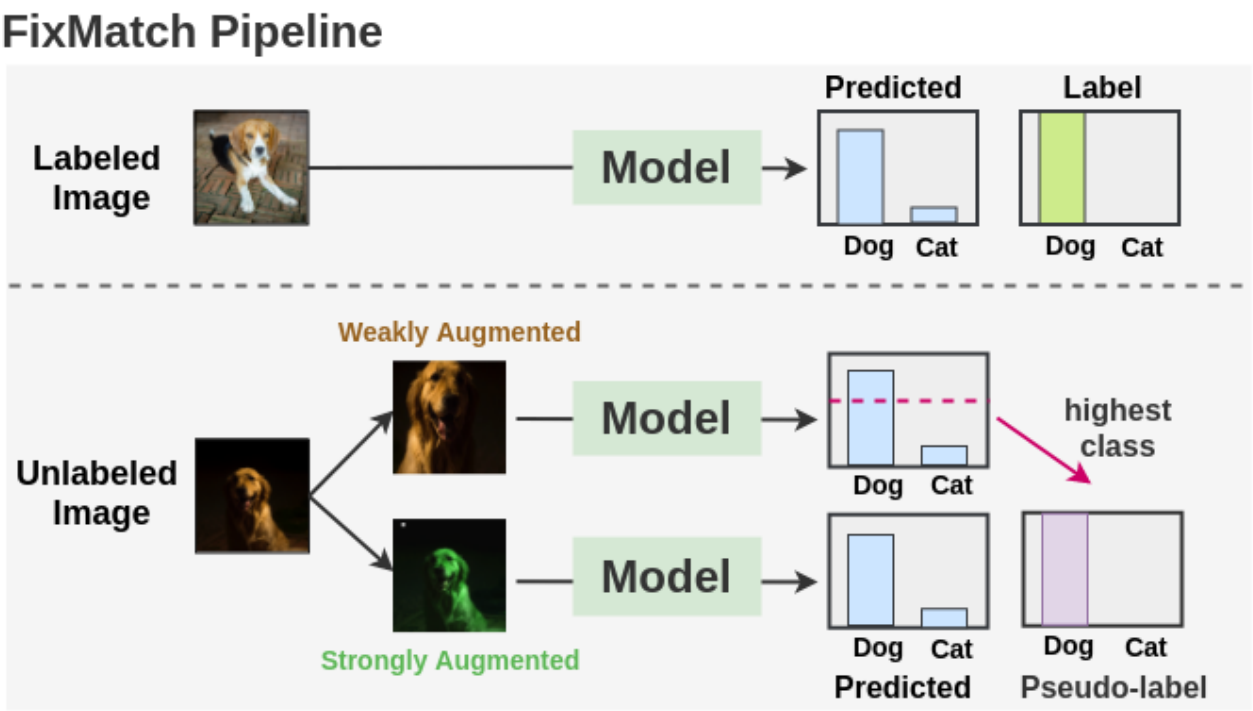
实验过程

理论：

FixMatch 结合了伪标签和一致性正则化两种技术，通过以下方式有效利用未标记数据：

1. **伪标签技术**：这是一种自训练方法，其中模型在未标记数据上进行预测，选取置信度高的预测作为标签。在 FixMatch 中，只有当模型对某个类别的预测置信度超过预设的阈值（通常很高，如0.95）时，才将该预测作为伪标签使用。这种选择性标签赋予策略减少了错误标签的引入，从而提高了学习的可靠性。
2. **一致性正则化**：一致性正则化基于这样的假设：模型对同一输入数据的轻微变化（通过数据增强实现）应产生一致的输出。在 FixMatch 中，对于每个未标记样本，生成两个版本：一个用于伪标签的弱增强版本和一个用于正则化的强增强版本。模型在训练过程中被鼓励使对强增强版本的预测与伪标签一致。

下图总结了整个FixMatch的pipeline：



如图所示，我们使用交叉熵损失在标注的图像上训练了监督模型。对于每个未标注的图像，使用弱增强和强增强获得两个图像。弱增强图像被传递到我们的模型中，我们得到了关于类的预测。将最有信心的类别的概率与阈值进行比较。如果它高于阈值，那么我们将该类作为ground truth的标签，即伪标签。然后，将经过强增强的图像传递到我们的模型中，获取类别的预测。使用交叉熵损失将此概率分布与ground truth伪标签进行比较。两种损失组合起来进行模型的更新。

详细组件：

① 训练数据与增强

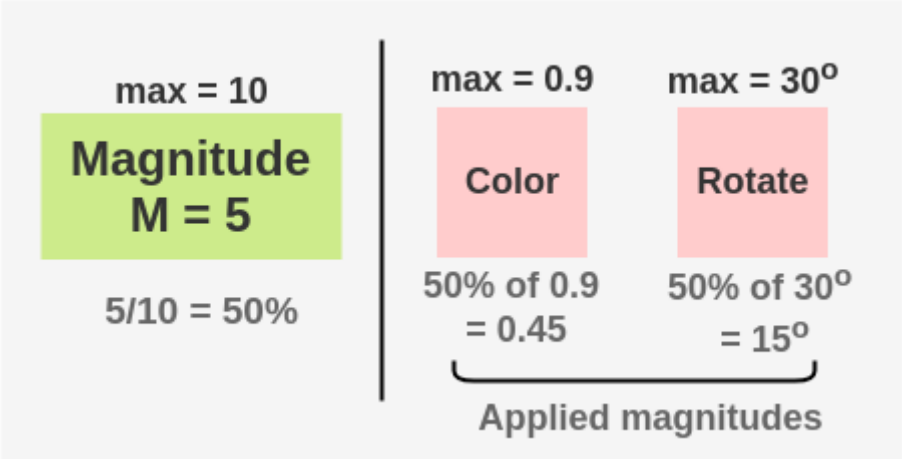
FixMatch在未标注的图像上进行弱增强以生成伪标签，同时在未标注图像上进行强增强以进行预测。

其中弱增强包括比如的标准翻转和平移。并且每种变换都自带一定的概率性。强增强包括输出严重失真的输入图像的增强版本，实际上应用RandAugment或CTAugment，然后应用CutOut增强随机删除图像的正方形部分。

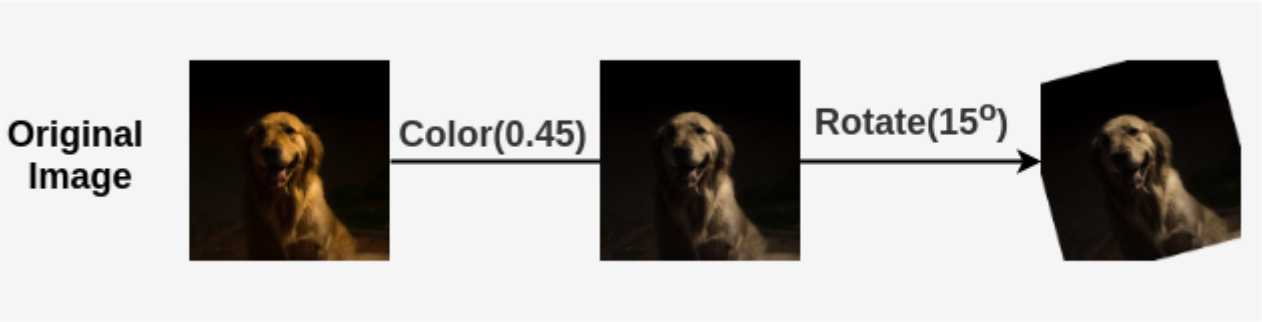
下面简要的介绍一下刚刚提到的RandAugment以及CTAugment。

RandAugment

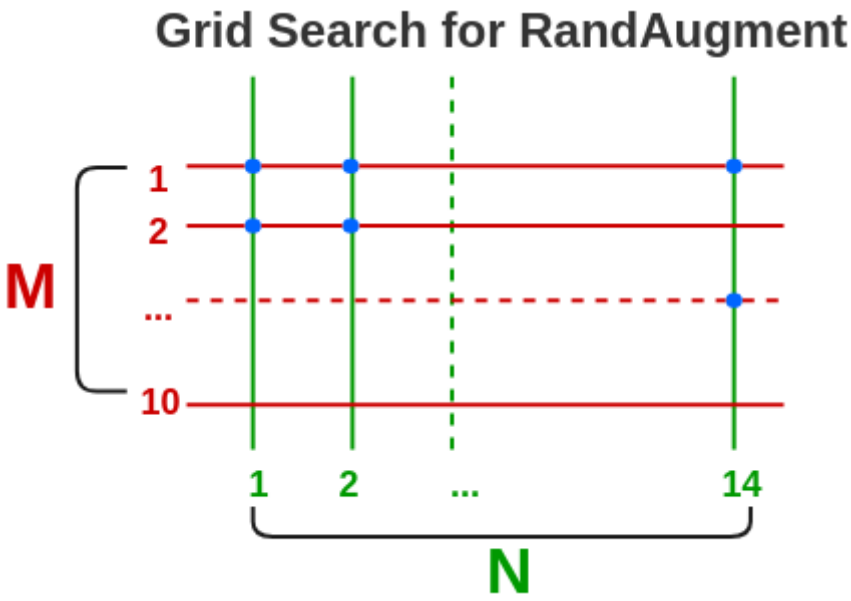
- 首先准备了一个14种可能的增强的列表，以及一系列可能的幅度。
- 从刚刚的增强列表里随机选出N个增强。
- 我们选择一个随机的幅度M，从1到10。我们可以选择一个幅度5，这意味着以百分比表示的幅度为50%，因为最大可能的M为10，所以百分比= 5/10 = 50%。



- 将所选的增强应用于序列中的图像。每种增强都有50%的可能性被应用。



- N和M的值可以通过在验证集上使用网格搜索的超参数优化来找到。在本文中，在每个训练步骤使用预定义范围内的随机幅度，而不是固定幅度。



CTAugment

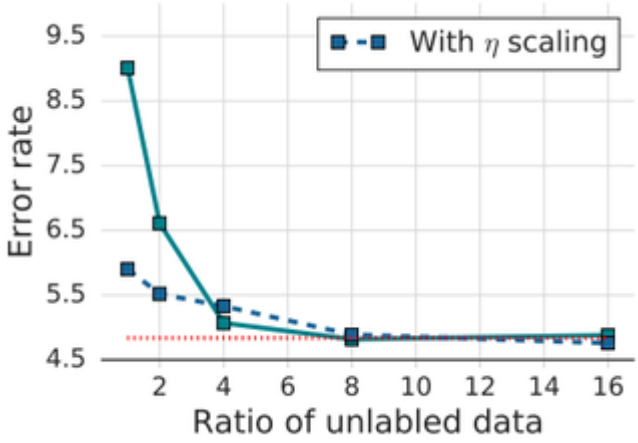
- 有一组18种可能的变换
- 变换的幅度值被划分为bin，每个bin被分配一个权重。最初，所有bin的权重均为1。
- 从该集合中以相等的概率随机选择两个变换，它们的序列形成了一条管道。
- 对于每个变换，根据归一化的bin权重随机选择一个幅值bin
- 带有标记的样本通过这两个转换得到了增强，并传递给模型以进行预测
- 根据模型预测值与实际标签的接近程度，更新这些变换的bin权重。
- 因此，它学会选择具有较高的机会来预测正确的标签的模型，从而在网络容差范围内进行增强。

② 模型选择

本文使用Wide Resnet这一具有一定深度和广度的resnet变体作为基础体系结构。具体而言使用了Wide-Resnet-28-2，深度为28，扩展因子为2。模型的宽度是ResNet的两倍。共有150万个参数。该模型与输出层堆叠在一起，输出层的节点等于所需的类数。

③ 模型训练

1. 准备批大小为B的标记图像和批大小为μB的未标记图像。μ是一个超参数，它决定批中未标注图像的相对大小。论文实验中通过增加μ的值，发现随着增加未标注图像的数量，错误率会降低，不过也正如图中所示在达到8之后降低的趋势就变得很平缓了。



2. 对于在标注图像上训练的监督部分，我们使用常规的交叉熵损失的思路。batch的总损失由下面的式子定义，并通过取batch中每个图像的交叉熵损失的平均值来计算。

$$l_s = \frac{1}{B}\sum_{b=1}^{B} H(\, p_b, p_m(y \,|\alpha(x_b)\,))$$

3. 对于未标注的图像，首先我们对未标注的图像应用弱增强，并通过argmax获得最高概率的预测类别得到伪标签，将与强增强图像上的模型输出进行比较。
4. 我们在同一张未标注的图片应用强增强，并将其输出与我们刚刚得到的的伪标签进行比较以计算交叉熵损失。这一部分的batch损失由下式进行计算。

$$l_u = \frac{1}{\mu B}\sum_{b=1}^{\mu B} 1(\max(q_b) >= \tau)\, H(\hat{q_b}, p_m(y|A(u_b)\,)$$

5. 最后，我们将两个损失加权求和得到总的损失，根据这个损失进行优化即可做到改进模型。

代码实现：

本次代码实现主要参考了github仓库kekmodel/FixMatch-pytorch中的实现，另外修改了train.py文件，尤其是train函数以及参数设置方面。

这次报告中主要也介绍train.py这一文件，尤其是train函数，因为这也是Fixmatch方法最大创新的体现之处。

在train函数中，我们首先检查是否使用混合精度训练（AMP）来提高训练速度和效率。

紧接着我们配置数据加载器，如果是分布式训练，则为有标签和无标签的数据加载器设定不同的epoch，确保数据的同步加载。

设置完毕后我们将模型也设置为训练状态并进入训练循环:

- 对于每个epoch，获取有标签的输入和标签数据（`inputs_x`, `targets_x`）和无标签的数据（`inputs_u_w`, `inputs_u_s`）。无标签数据包括两种形式的输入，一种用于生成伪标签，一种用于计算损失。

```
            try:
                inputs_l, targets_l = next(labeled_iter)
            except:
                labeled_iter = iter(labeled_trainloader)
                inputs_l, targets_l = next(labeled_iter)

            try:
                (inputs_u_1, inputs_u_2), _ = next(unlabeled_iter)
            except:
                unlabeled_iter = iter(unlabeled_trainloader)
                (inputs_u_1, inputs_u_2), _ = next(unlabeled_iter)
```

- 数据发送到GPU，并通过模型（本次实验我们采用wideresnet）进行前向传播。

- 对于有标签数据，使用交叉熵损失得到`Lx`。对于无标签数据，先计算伪标签和掩码（基于预测概率阈值），然后计算掩码的交叉熵损失`Lu`。总损失是有标签损失和加权的无标签损失之和。

```
        Ll = F.cross_entropy(logits_l, targets_l, reduction='mean')

            pseudo_label = torch.softmax(logits_u_1.detach() /
args.T, 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_2, targets_u,
reduction='none') * mask).mean()
            loss = Ll + args.lambda_u * Lu
```

- **反向传播和优化**: 根据总损失进行反向传播，更新模型参数，我们并没有像原仓库里的代码一样采用amp优化的方法，而是直接进行了梯度下降。损失计算后我们利用优化器进行参数更新，这里在我们的实验中还使用了指数移动平均（EMA）更新的方法对模型参数进行平滑处理。

```
            loss.backward()
            losses.update(loss.item())
            losses_l.update(Ll.item())
            losses_u.update(Lu.item())
            optimizer.step()
            scheduler.step()
            ema_model.update(model)
```

- 每个epoch结束时，我们使用EMA模型进行测试，评估模型的性能。具体测试代码如下所示，因为本身Fixmatch的重点也完全不在测试上，所以在测试上基本沿用原本github中的测试代码，具体而言我们还是通过交叉熵进行损失的计算，同时也用模型输出和目标输出计算top1和top5的精度，计算完成后我们输出本次测试的信息。

```
    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(test_loader):
            model.eval()

            inputs = inputs.to(args.device)
            targets = targets.to(args.device)
            outputs = model(inputs)
```

```
            loss = F.cross_entropy(outputs, targets)

            prec1, prec5 = accuracy(outputs, targets, topk=(1, 5))
            losses.update(loss.item(), inputs.shape[0])
            top1.update(prec1.item(), inputs.shape[0])
            top5.update(prec5.item(), inputs.shape[0])
            # change
            # precision = precision_score(targets, outputs,
average='macro')
            if not args.no_progress:
                test_loader.set_description(
                    "Test Iter: {batch:4}/{iter:4}. Loss:
{loss:.4f}. top1: {top1:.2f}. top5: {top5:.2f}. ".format(
                        batch=batch_idx + 1,
                        iter=len(test_loader),
                        loss=losses.avg,
                        top1=top1.avg,
                        top5=top5.avg,
                    ))
        if not args.no_progress:
            test_loader.close()
```

- 在训练的最后记录训练和测试的详细指标，保存模型的检查点，特别是当达到更高的测试准确率时。更新tensorboard日志，方便后续分析和监控训练进程。

TorchSSL：

直接从githuhb上克隆TorchSSL库，由于机器限制和时间限制，降低训练总迭代次数为131072，并且学习率上调到0.05.

如下为实验结果：

标注数据	TORCHSSL	MY FIXMATCH
40	68.90	83.12
250	94.42	92.00
4000	95.44	95.59

半监督算法对比：

对比 FixMatch 与 MixMatch 半监督流行算法，它们都试图利用少量的标记数据和大量的未标记数据来训练机器学习模型。不过，尽管它们目标相同，在实现上也有一些关键的不同点：

1. 数据增强方面：
    - MixMatch该算法将标记数据和未标记数据混合在一起，并应用数据增强技术（如翻转、裁剪等）来生成更多样化的数据样本。MixMatch 在处理未标记数据时，会生成多个增强版本，然后对这些增强版本的预测结果进行平均，以得到一个"软"标签。
    - FixMatch对于每个未标记样本，只选取一种强数据增强方式生成单个样本，并基于这个增强样本的模型预测来生成伪标签。如果模型对伪标签的置信度高于某个阈值，则将这个伪标签视为正确标签来训练模型。
2. 标签猜测方面：
    - MixMatch对未标记数据的处理中，该算法不仅平均多个增强样本的预测结果来形成软标签，而且还会对标签进行温和的调整，使其分布更平滑，这有助于增加模型的泛化能力。
    - FixMatch依赖单个强数据增强样本的预测来决定伪标签，如果预测的置信度超过预设的阈值，则接受该伪标签。这种方法侧重于高置信度的预测，从而减少标签噪声的引入。
3. 复杂性与效率：
    - MixMatch由于需要对每个未标记样本生成多个增强样本并计算平均预测，这增加了计算的复杂性和训练时间。
    - FixMatch通过使用单个增强样本来生成伪标签，大大简化了处理流程，这使得FixMatch在计算效率上通常优于MixMatch，尤其是在资源有限的情况下。
4. 性能方面：
    - MixMatch通过软标签和标签分布的平滑化，可以更好地处理标签噪声和数据不平衡的问题。
    - FixMatch虽然在一些情冗下可能对噪声更敏感，但其简洁的方法在很多基准数据集上展示了良好的性能，尤其是在标签质量较高时。

综合而言FixMatch简单高效，而MixMatch则在需要更细致处理数据和标签时表现更好。