本科生实验报告

学生姓名：　丁晓琪

学生学号：　22336057

专业名称：　计科

一：问题背景和动机

- 背景：
  - 现实世界中从网络上获取的数据集只有少部分有标注。有监督训练需要大量标注数据，获取成本高。无监督训练的计算成本较高和训练成果一般以及无法充分利用带标注的数据。
  - 对比学习优势：对比学习是自监督学习的一种重要方法。它通过最大化正样本对的相似度和最小化负样本对的相似性，学习数据表示，可用于对数据编码。且对比学习容易理解，实现简单。
- 动机：
  - 减少对标注数据的依赖：通过 SimCLR 框架，可以在无标签数据上预训练模型，学习到有意义的特征表示，从而减少对标注数据的依赖
  - 提高分类的性能：在 SimCLR 预训练的基础上，添加分类头进行有监督微调

- 验证SimCLR的有效性：在MNIST和CIFAR10数据集上实验，验证 SimCLR 在图像分类任务中的有效性

二：当前解决问题的方法

1. **生成式方法**：
   - 通过生成模型学习数据的表示
   - 缺点：生成模型训练复杂，且生成质量对表示学习的影响较大
2. **对比学习方法**：
   - 通过对比正样本对和负样本对，学习数据的表示
   - 代表方法：SimCLR、MoCo（Momentum Contrast）、BYOL（Bootstrap Your Own Latent）
   - 优点：简单高效，适用于大规模无标签数据
3. **迁移学习**：
   - 在大规模数据集（如 ImageNet）上预训练模型，然后迁移到目标任务。
   - 缺点：需要大规模标注数据。

三：实验过程

(一）实验流程

1. MNIST 10

- 对比网络无监督训练：将mnist10训练集分成10个批次，每个批次6000张图片，使用前4个批次对对比网络进行无监督训练得到编码器
- 分类训练：使用第5个批次的数据，对对比网络在分类模型上有监督的分类训练
- 测试验证：在测试集上验证分类模型的正确性

```python
if __name__ == "__main__":
    #1. 提取数据和标签
    transform1 = transforms.Compose([
        transforms.ToTensor(),  # 将图像转换为张量
        transforms.Normalize((0.1307,), (0.3081,)),  # 归一化
    ])
    train_dataset = MNIST(root='./data', train=True, transform=transform1, download=True)
    # 只取前两个批次的样本
    num_per_batch=6000  #MNIST每个批次的样本数量
    num_batches = 4  #只取前四个批次
    indices = list(range(num_per_batch  * num_batches))
    train_subset = Subset(train_dataset, indices)

    # 定义数据加载器
    train_loader = DataLoader(train_subset, batch_size=256, shuffle=True)
    # 2.模型
    simclr_model = SimCLRModel()
    # 3.SimCLR训练
    SimCLR_train(simclr_model, train_loader, epochs=10)
    # 4.分类训练
    num_samples_per_batch_class = 6000  # MNIST 每个批次的样本数量
    start_index = 4 * num_per_batch   # 第4个批次的起始索引
    end_index = 5* num_per_batch  # 第5个批次的结束索引
    indices_class = list(range(start_index, end_index))
    train_subset_class = Subset(train_dataset, indices_class)
    train_class_loader = DataLoader(train_subset_class, batch_size=128, shuffle=True)
    # 模型定义
    classification_model=ClassificationModel(simclr_model)
    classification_train(classification_model, train_class_loader, 30)

    # 加载测试数据集
    test_dataset = MNIST(root='./data', train=False, transform=transform1, download=True)

    # 定义测试数据加载器
    test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
    # 6. 验证模型
    print("Evaluating on test set...")
    evaluate(classification_model,test_loader)
```

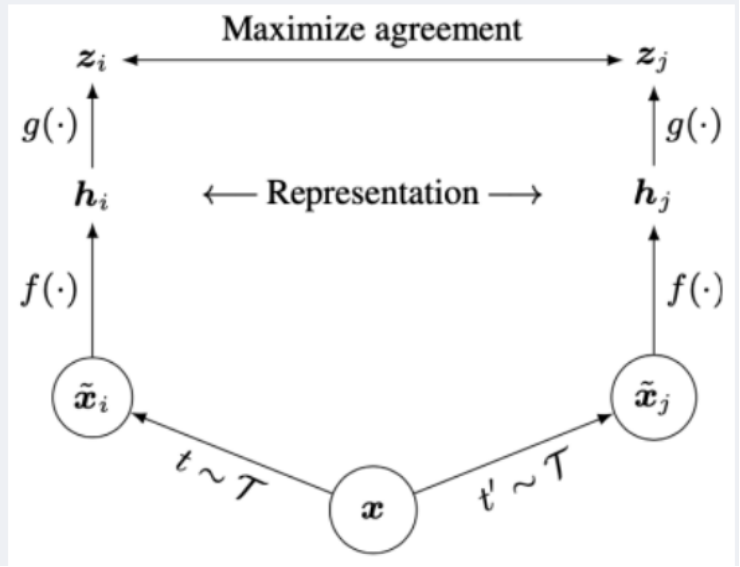
2. CIFAR 10

- 对比网络无监督训练：CIFAR10有5个批次，每个批次10000张图片，使用前4个批次对对比网络进行无监督训练得到编码器
- 分类训练：使用第1个批次和第二个批次的数据，对对比网络在分类模型上有监督的分类训练
- 测试验证：在测试集上验证分类模型的正确性

```python
if __name__=="__main__":
    #1.提取数据和标签
    transform1 = transforms.Compose([
        transforms.ToTensor(),  # 将图像转换为张量
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),  # 归一化
    ])
    train_dataset = CIFAR10(root='./data', train=True, transform=transform1,download=True)
    # 只取前四个批次的样本
    num_per_batch=10000  #CIFAR-10每个批次的样本数量
    num_batches = 4  # 只取前四个批次
    indices = list(range(num_per_batch * num_batches))
    train_subset = Subset(train_dataset, indices)

    # 定义数据加载器
    train_loader = DataLoader(train_subset, batch_size=128, shuffle=True)
    # 2.模型
    simclr_model = SimCLRModel()
    # 3.SimCLR训练
    SimCLR_train(simclr_model, train_loader, epochs=10)
    # 4. 分类训练
    num_samples_per_batch_class= 10000  # CIFAR-10 每个批次的样本数量
    start_index = 0 * num_per_batch  # 第1个批次的起始索引
    end_index = 2* num_per_batch  # 第2个批次的结束索引
    indices_class= list(range(start_index, end_index))
    train_subset_class= Subset(train_dataset, indices_class)
    train_class_loader= DataLoader(train_subset_class, batch_size=128, shuffle=True)
    # 模型定义
    classification_model=ClassificationModel(simclr_model)
    classification_train(classification_model,train_class_loader,30)

    # 加载测试数据集
    test_dataset = CIFAR10(root='./data', train=False, transform=transform1, download=True)

    # 定义测试数据加载器
    test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
    # 6. 验证模型
    print("Evaluating on test set...")
    evaluate(classification_model,test_loader)
```

(二) 对比学习网络

1. 组成



- **编码层**$h_i = f(x_i)$
  - MNIST 10 样本：由于样本简易，则自定义两层卷积层和一层全输出层为编码器

```
        #CNN,Resnet太复杂了
        self.encoder = nn.Sequential(

 nn.Conv2d(1,32,kernel_size=3,stride=1,padding=1),#1*28*28->32*28*28
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2), #32*28*28->32*14*14

 nn.Conv2d(32,64,kernel_size=3,stride=1,padding=1),#32*14*14->64*14*14
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),#64*14*14->64*7*7
            nn.Flatten(), #64*7*7->3136
            nn.Linear(3136, 128), #128
         )
```

- CIFAR 10样本：由于样本复杂，则用预训练好的RESNET 18，需要去除池化层防止维度过快下降

```
        # 编码器,使用resetnet预训练的权重
        self.encoder = resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
        # 修改第一层，适应CIFAR-10的输入尺寸 输出为3*64*32*32
        self.encoder.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
        # 去掉池化层
        self.encoder.maxpool=nn.Identity()
        # 去掉全连接层
        self.encoder.fc=nn.Identity()
```

- **投影层**$z_i = g(h_i)$
  - MNIST 10：两层线性层，一层激活层

```
        # 投影头
        self.projection_head = nn.Sequential(
            nn.Linear(128,64),
            nn.ReLU(),
            nn.Linear(64,32),#32
        )
```

  - CIFAR 10: 两层线性层，一层激活层

```
        self.projection_head=nn.Sequential(
            nn.Linear(512, 256),  # 512 是 ResNet 的输出维度
            nn.ReLU(),
            nn.Linear(256, 128),  # 投影到 128 维
        )###
```

- 前向传播：将编码层输出结果作为投影层输入结果

```
    def forward(self,x):
        # 提取特征
        features=self.encoder(x)
        #features = features.view(features.size(0), -1)  # 展平为(batch_size, 512*8*8)
        # 投影到低维空间
        projections=self.projection_head(features)
        return features, projections
```

2. 训练

- **思路:**

```
1. 每个迭代中，对每张图像生成不同的增强视图
2. 将视图输入到定义好的SimCLR模型中，获得视图投影
3. 拼接两个视图投影，计算对比损失，使同一张图像的增加视图尽量相似，不同图像不相似
4. 反向传播更新模型编码层和投影层参数，最小化对比损失。
```

```
def SimCLR_train(model, train_loader, epochs):
    model.to(device)
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    for epoch in range(epochs):
        model.train() #设置模型为训练模式
        sum_loss=0
        for batch in train_loader:
            images, _ = batch #无标签训练，不要标签
            #数据增强
            view1=images
            view2 = apply_transform(images, transform)
            view1=view1.to(device)
            view2=view2.to(device)
            #输入到模型中，不要编码结果，只要投影结果
            _, proj1=model(view1)
            _, proj2=model(view2)
            #拼接结果
            projections=torch.cat([proj1, proj2], dim=0)
            #计算loss
            loss=nt_xent_loss(projections)
            #反向传播
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            sum_loss+=loss.item()
    #计算每个批次平均损失
        avg_loss = sum_loss/len(train_loader)
        print(f"Epoch {epoch+1}, Loss: {avg_loss}")
```

- **图像增强处理：**使用`torchvision.transforms`，对图像随机裁剪，模糊，旋转...
  （`apply_transform()`对图像批量进行增强操作）

```
#定义图像增强函数，只能一张张图片用(transforms默认处理3通道)
transform = transforms.Compose([
```

```
    transforms.RandomHorizontalFlip(p=0.5), #0.5概率翻转
    transforms.RandomApply([transforms.RandomResizedCrop(size=28,scale=(0.8, 1.0))],p=0.5), #0.5概率剪裁成0.8-1.0的大小
    transforms.RandomApply([transforms.GaussianBlur(kernel_size=5, sigma=(0.1, 2.0))], p=0.3), #0.3的概率进行高斯模糊
    transforms.RandomRotation(degrees=15),  #随机旋转
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.1307], std=[0.3081]),
])
# 多张图片批量处理
def apply_transform(images, transform):
    transformed_images = []
    for image in images:
        #转换为numpy数组并调整维度顺序为HWC,适应PIL图像格式（和CCIFAR10不一样）
        image_np=image.permute(1, 2, 0).numpy()
        #转换为 PIL 图像
        image_pil=transforms.ToPILImage()(image_np)
        #transform
        transformed_image = transform(image_pil)
        transformed_images.append(transformed_image)
    return torch.stack(transformed_images)
```

- **对比损失计算（`nt_xtent_loss(projection,temperature)`）：**
  - 理论：$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)}, \quad (1)$$
  - 实现：
    1. 传入参数：
       `projections`：原图像的投影拼接增强图像的投影，大小为(2N,D), N为一个批次中图像数量，D为投影维度，前N行是原始图像，后N行是增强图像
    2. 对projections计算余弦相似矩阵`similarity_matrix`：第 i 行第 j 列的元素表示第 i 个样本和第 j个样本之间的余弦相似度, 大小为2N行2N列
    3. 对`similarity_matrix`屏蔽自身相似度：把对角线元素设置为0
    4. 提取正样本对`pos_sim`：`similarity_matrix`的前n行后n列，后n行前n列的对角线。大小为（2N，1), 为每个样本和其正样本对的相似度
    5. 计算对比公式：分母为所有样本的指数相似度，分子为所有正样本的指数相似度

```
## nt_xent_loss对比损失函数的实现
def nt_xent_loss(projections,temperature=0.5):
    # 1. 相似矩阵计算
    batch_size = projections.shape[0]  # 2N
    N = batch_size // 2  #

 similarity_matrix=F.cosine_similarity(projections.unsqueeze(1),projections.unsqueeze(0),dim=-1) #2N*2N
    # 2.屏蔽自身相似度
    mask = torch.eye(batch_size, dtype=torch.bool).to(projections.device)
    similarity_matrix.masked_fill_(mask, 0)
    # 3.提取正样本对
    pos_mask=torch.zeros_like(similarity_matrix,dtype=torch.bool)
```

```
    pos_mask[:N,N:]=torch.eye(N) #前N行后N咧
    pos_mask[N:,:N]=torch.eye(N) #后N行前N列
      # 提取正样本相似度，重塑为2N，1
    pos_sim=similarity_matrix[pos_mask].view(batch_size,1)
    # 4.将自身相似度的影响放的最小(后面要求指数)
    all_sim = similarity_matrix.masked_fill(mask, -float('inf'))

    pos_sim=pos_sim/temperature
    all_sim=all_sim/temperature
    # 计算所有的相似度和,所有列求和，形状为2N，1
    all_exp=torch.exp(all_sim)
    all_sum=torch.sum(all_exp,dim=1,keepdim=True)
    #求损失
    loss=-torch.log(torch.exp(pos_sim)/all_sum)
    # 返回的是批次的平均损失，标量张量
    loss=loss.mean()
    return loss
```

(三) 下流分类任务

1. 分类模型组成

- 预训练好的SimCLR模型的编码器
- 分类头：简单全连接神经网络，用于将编码器的输出映射到类别空间。
  - MNIST 10：

```
# 预训练的SimCLR+分类头
class ClassificationModel(nn.Module):
    def __init__(self, simclr_model):
        super(ClassificationModel, self).__init__()
        self.simclr_model = simclr_model #编码器
        self.classification_head= nn.Sequential(
            nn.Linear(128, 64),  #编码器输出是 128 维
            nn.ReLU(),
            nn.Linear(64, 10),  #10个类别
        )
        # 已经预训练好，不需要再次训练，冻结编码器的权重
        freeze_encoder(self.simclr_model)
```

  - CIFAR 10：

```
# 预训练的SimCLR+分类头
class ClassificationModel(nn.Module):
    def __init__(self,simclr_model):
        super(ClassificationModel, self).__init__()
        self.simclr_model = simclr_model #编码器
        self.classification_head= nn.Linear(512, 10)
        # 已经预训练好，不需要再次训练，冻结编码器的权重
        freeze_encoder(self.simclr_model)
```

- 注意：在初始化时调用 `freeze_encoder` 冻结对比网络的梯度，冻结SimCLR编码器参数，避免在分类任务中破坏预训练的特征表示

```python
#冻结预训练的SimCLR的参数
def freeze_encoder(model):
    for param in model.encoder.parameters():
        param.requires_grad = False  # 冻结编码器的权重
```

- 前向传播：数据输入到SimCLR预训练好的编码器中提取特征，将特征传入全连接分类层，完成分类任务

```python
    def forward(self, x):
        # 使用 SimCLR 的编码器提取特征，不需要投影器
        features, _ = self.simclr_model(x)
        # 使用分类头进行分类
        label = self.classification_head(features)
        return label
```

2. 训练

- 注意：只训练分类头

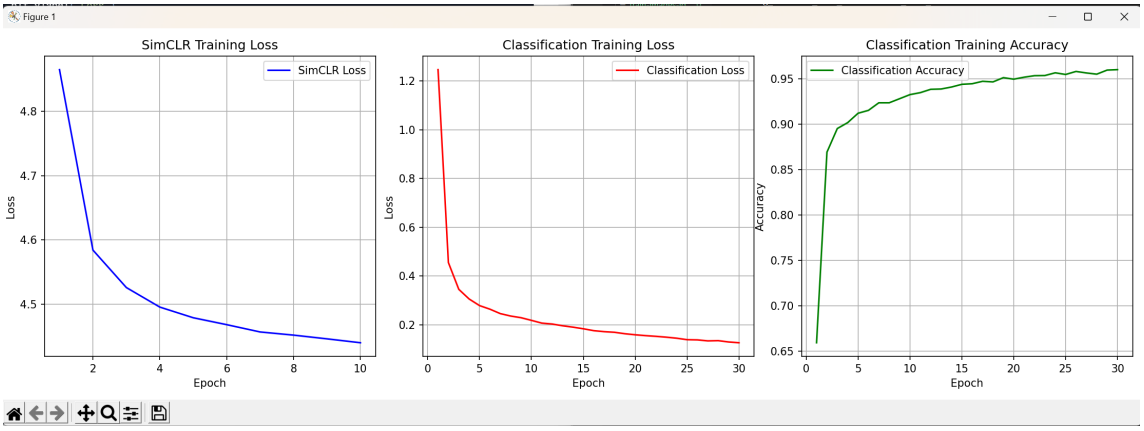
1. 将图像输入模型，获得预测结果
2. 将图像输入模型，获得每个分类的预测概率
3. 使用交叉熵损失函数计算预测结果和真实标签的差异
4. 梯度下降更新模型参数

```python
def classification_train(model,train_loader,epochs):
    model.to(device)
    optimizer=optim.Adam(model.parameters(),lr=0.001)
    criterion=nn.CrossEntropyLoss()  # 交叉熵损失函数
    for epoch in range(epochs):
        model.train()
        sum_loss=0
        total_correct=0  # 记录当前 epoch 的正确预测数量
        total_samples=0  # 记录当前 epoch 的总样本数量
        for batch in train_loader:
            images,labels=batch
            images,labels=images.to(device), labels.to(device)
            #获得分类结果
            predict_labels=model(images)
            #计算loss
            loss=criterion(predict_labels,labels)
            # 反向传播
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            sum_loss+=loss.item()
            # 计算当前批次的正确预测数量
            predictions=torch.argmax(predict_labels, dim=1)  # 获取预测的类别
            correct=(predictions == labels).sum().item()  # 计算正确预测的数量
            total_correct+=correct
            total_samples+=labels.size(0)  # 当前批次的样本数量
        # 计算每个 epoch 的准确率
        avg_loss=sum_loss/len(train_loader)
        accuracy=total_correct/total_samples
        print(f"Epoch {epoch + 1}, Loss: {avg_loss}, Accuracy: {accuracy * 100:.2f}%")
```

四：实验结果

1. mnist 10

- 取4个大小为6000的批次（训练过程中每个批次128大小）做无标签simCLR训练（迭代10次），取1个大小为6000的批次做有标签分类训练（迭代30次），结果分析如下：



  - 训练时间：simCLR训练时间占主要训练时间有179.44s，而分类训练较快有36.88s，总共训练216.31s

```
Classification Training Time: 36.88 seconds
Total Training Time: 216.31 seconds
SimCLR Training Time: 179.44 seconds
```

- 训练损失：simCLR和分类训练的损失都较快收敛，simCLR在第9次迭代的时候趋近收敛，分类训练在第28次时趋近收敛

```
Start SimCLR Traing...
Epoch 1, Loss: 4.864839021195757
Epoch 2, Loss: 4.584232020885386
Epoch 3, Loss: 4.52598744757632
Epoch 4, Loss: 4.495634119561378
Epoch 5, Loss: 4.479025039267032
Epoch 6, Loss: 4.468311822160762
Epoch 7, Loss: 4.456926690771224
Epoch 8, Loss: 4.4520227858360775
Epoch 9, Loss: 4.4462376148142715
Epoch 10, Loss: 4.4401873071142965
Starting Classification Training...
Epoch 1, Loss: 1.2471530475515, Accuracy: 65.92%
Epoch 2, Loss: 0.4549176895872076, Accuracy: 86.90%
Epoch 3, Loss: 0.34507151296798216, Accuracy: 89.50%
Epoch 4, Loss: 0.305288664204009, Accuracy: 90.15%
Epoch 5, Loss: 0.27828438009353396, Accuracy: 91.18%
Epoch 6, Loss: 0.26361099804969546, Accuracy: 91.50%
Epoch 7, Loss: 0.24534570441601125, Accuracy: 92.33%
Epoch 8, Loss: 0.2353544511059497, Accuracy: 92.33%
Epoch 9, Loss: 0.22851136707245034, Accuracy: 92.78%
Epoch 10, Loss: 0.21763438794841158, Accuracy: 93.23%
Epoch 11, Loss: 0.20635713541761358, Accuracy: 93.45%
Epoch 12, Loss: 0.20261839602855927, Accuracy: 93.82%
Epoch 13, Loss: 0.195490041628797, Accuracy: 93.85%
Epoch 14, Loss: 0.18980158960565607, Accuracy: 94.07%
Epoch 15, Loss: 0.18312137748332732, Accuracy: 94.37%
Epoch 16, Loss: 0.17533958893507084, Accuracy: 94.43%
Epoch 17, Loss: 0.1712654280535718, Accuracy: 94.70%
Epoch 18, Loss: 0.1686703827469907, Accuracy: 94.63%
Epoch 19, Loss: 0.162752247078621 65, Accuracy: 95.10%
Epoch 20, Loss: 0.15842010508826437, Accuracy: 94.93%
Epoch 21, Loss: 0.15492211662708444, Accuracy: 95.15%
Epoch 22, Loss: 0.15204207250412474, Accuracy: 95.32%
Epoch 23, Loss: 0.14859556787191552, Accuracy: 95.33%
Epoch 24, Loss: 0.14435081627774746, Accuracy: 95.63%
Epoch 25, Loss: 0.138314352073568, Accuracy: 95.45%
Epoch 26, Loss: 0.13745191027509404, Accuracy: 95.78%
Epoch 27, Loss: 0.13340390045592126, Accuracy: 95.62%
Epoch 28, Loss: 0.1343814360493041, Accuracy: 95.48%
Epoch 29, Loss: 0.12893920708844003, Accuracy: 95.93%
Epoch 30, Loss: 0.12577100716372755, Accuracy: 95.98%
Classification Training Time: 36.88 seconds
```

- 训练正确率：分类训练时正确率上升较快，且在25次迭代左右趋近收敛到95%
- 测试正确率：测试正确率高达95%

```
Evaluating on test set...
Test Accuracy: 95.27%
```

- 后经多次尝试，发现simCLR训练迭代6次/8次的效果和迭代10次差不多，在分类训练中如果对两个批次有标签数据训练，测试正确率能高达97%

- vs聚类分类和高斯混合聚类分类对比：

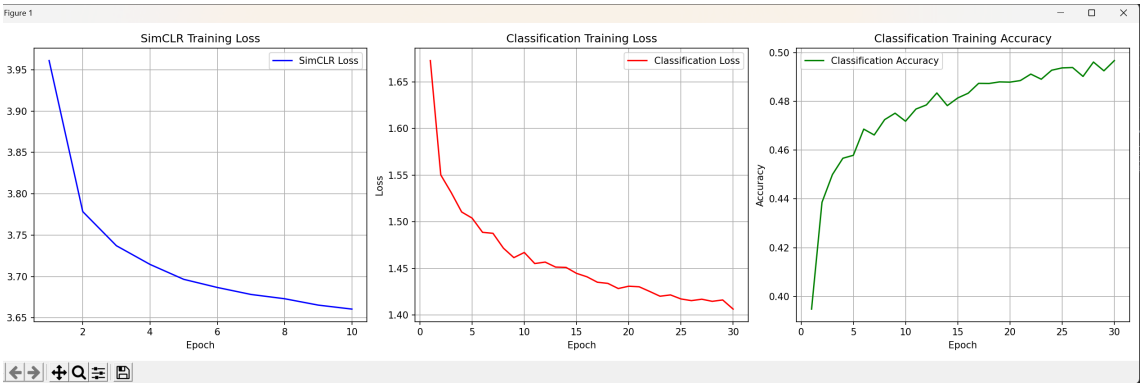
在Assignmet3时完成了对mnist10数据集的无监督聚类分类和高斯混合聚合分类。直接采用Assignment3的数据分析：

	随机初始化高斯分布的均值	根据距离初始化高斯分布的均值
以训练样本的协方差矩阵的对角矩阵为高斯分布的协方差矩阵	Iteration 0: Test Accuracy = 0.3435 Iteration 5: Test Accuracy = 0.4403 Iteration 10: Test Accuracy = 0.4687 Iteration 15: Test Accuracy = 0.4981 Iteration 20: Test Accuracy = 0.5083 Iteration 25: Test Accuracy = 0.5187 Iteration 30: Test Accuracy = 0.534 Iteration 35: Test Accuracy = 0.5388 Iteration 40: Test Accuracy = 0.5459 Iteration 45: Test Accuracy = 0.5567 Iteration 50: Test Accuracy = 0.5669 Iteration 55: Test Accuracy = 0.5775 Iteration 60: Test Accuracy = 0.5871 Iteration 65: Test Accuracy = 0.5979 Iteration 70: Test Accuracy = 0.6089 Iteration 75: Test Accuracy = 0.6138 Iteration 80: Test Accuracy = 0.6136 Iteration 85: Test Accuracy = 0.6109 Iteration 90: Test Accuracy = 0.608 Iteration 95: Test Accuracy = 0.6054 训练过程耗时 189.90 秒 final test accruacy: 0.6046 收敛速度：最慢	Iteration 0: Test Accuracy = 0.362 Iteration 5: Test Accuracy = 0.5594 Iteration 10: Test Accuracy = 0.5504 Iteration 15: Test Accuracy = 0.5577 Iteration 20: Test Accuracy = 0.5928 Iteration 25: Test Accuracy = 0.6322 Iteration 30: Test Accuracy = 0.6355 Iteration 35: Test Accuracy = 0.637 Iteration 40: Test Accuracy = 0.6382 Iteration 45: Test Accuracy = 0.6393 Iteration 50: Test Accuracy = 0.6397 Iteration 55: Test Accuracy = 0.6404 Iteration 60: Test Accuracy = 0.641 Iteration 65: Test Accuracy = 0.6412 Iteration 70: Test Accuracy = 0.6416 Iteration 75: Test Accuracy = 0.6416 Iteration 80: Test Accuracy = 0.6421 Iteration 85: Test Accuracy = 0.6421 Iteration 90: Test Accuracy = 0.6421 Iteration 95: Test Accuracy = 0.6421 训练过程耗时 194.35 秒 训练时间最长：计算训练样本呢的协方差矩阵和计算每个样本到聚类的距离花费的时间最长 正确率：最高
以训练样本某个维度的平均值为对角矩阵的元素的对角矩阵为高斯分布的协方差矩阵	Iteration 0: Test Accuracy = 0.505 Iteration 5: Test Accuracy = 0.5812 Iteration 10: Test Accuracy = 0.6031 Iteration 15: Test Accuracy = 0.6001 Iteration 20: Test Accuracy = 0.6063 Iteration 25: Test Accuracy = 0.6112 Iteration 30: Test Accuracy = 0.6109 Iteration 35: Test Accuracy = 0.6109 Iteration 40: Test Accuracy = 0.6095 Iteration 45: Test Accuracy = 0.6105 Iteration 50: Test Accuracy = 0.6112 Iteration 55: Test Accuracy = 0.6117 Iteration 60: Test Accuracy = 0.6121 Iteration 65: Test Accuracy = 0.6128 Iteration 70: Test Accuracy = 0.6134 Iteration 75: Test Accuracy = 0.6143 Iteration 80: Test Accuracy = 0.6145 Iteration 85: Test Accuracy = 0.6149 Iteration 90: Test Accuracy = 0.6156 Iteration 95: Test Accuracy = 0.6159 训练过程耗时 190.85 秒 final test accruacy: 0.6161 	Iteration 0: Test Accuracy = 0.4016 Iteration 5: Test Accuracy = 0.5712 Iteration 10: Test Accuracy = 0.5732 Iteration 15: Test Accuracy = 0.5707 Iteration 20: Test Accuracy = 0.5752 Iteration 25: Test Accuracy = 0.5787 Iteration 30: Test Accuracy = 0.5811 Iteration 35: Test Accuracy = 0.5858 Iteration 40: Test Accuracy = 0.5869 Iteration 45: Test Accuracy = 0.5867 Iteration 50: Test Accuracy = 0.5866 Iteration 55: Test Accuracy = 0.5861 Iteration 60: Test Accuracy = 0.5864 Iteration 65: Test Accuracy = 0.5928 Iteration 70: Test Accuracy = 0.593 Iteration 75: Test Accuracy = 0.5941 Iteration 80: Test Accuracy = 0.5945 Iteration 85: Test Accuracy = 0.5943 Iteration 90: Test Accuracy = 0.5947

- 训练时间：SimCLR+分类训练的时长比无监督聚类和高斯混合聚类分类都较长，在SimCLR耗时较长。
- 正确率：完全无监督聚类和高斯混合聚类的正确率不超过70%，但是SimCLR+分类训练的正确率能达到95%以上
- 现实意义：在现实中更多的是大量无标签数据和少量有标签数据混合，采用SimCLR+分类训练能够充分利用有标签数据，大大提升正确率。验证了SimCLR+分类训练的正确性。

2. CIFAR 10

- 选择10000*4大小的数据进行无标签SimCLR训练（训练过程中每个批次为128大小，迭代10次），选择20000大小的数据进行下流有标签分类训练（迭代30次）



- SimCLR训练：耗时特别长共984.69s，且10次迭代后仍未收敛，收敛较慢

```
Epoch 10, Loss: 3.660530206113578
SimCLR Training Time: 984.69 seconds
```

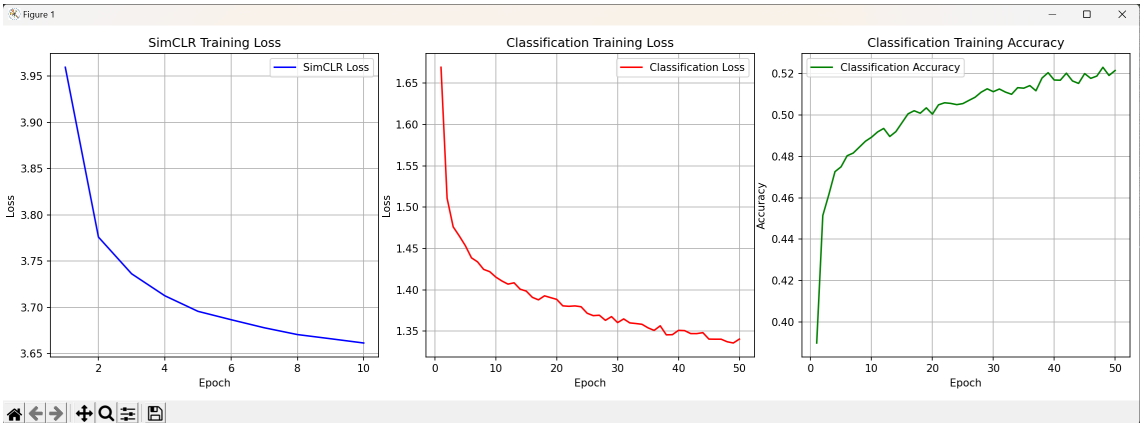
- 对比训练：耗时较短共238.02s，且30次迭代后仍未收敛，收敛较慢

```
Classification Training Time: 238.02 seconds
Total Training Time: 1222.71 seconds
```

- 正确率：训练过程中正确率为49%左右，测试正确率略低于训练正确率为47%左右

```
Epoch 25, Loss: 1.4172999099561363, Accuracy: 49.38%
Epoch 26, Loss: 1.4154930722181964, Accuracy: 49.40%
Epoch 27, Loss: 1.4170075632204675, Accuracy: 49.03%
Epoch 28, Loss: 1.4147349907334443, Accuracy: 49.62%
Epoch 29, Loss: 1.4162169398775526, Accuracy: 49.26%
Epoch 30, Loss: 1.4064204609318145, Accuracy: 49.68%
Evaluating on test set...
Test Accuracy: 47.30%
```

- 由于正确率过低，这里增大simCLR每个训练批次大小为512，以及对比训练迭代次数为50次，再次训练测试：



  - SimCLR训练：训练时间明显增长为1097.81s
  - 对比训练：增加到50次迭代后正确率和loss仍未收敛，收敛较慢
  - 正确率：训练正确率在迭代30次时已经提高到50%左右，最终测试正确率略低于训练正确率为49.97%
  - 注意：增大SimCLR训练时的批次大小有利于学习图像特征，提高正确率

- SimCLR+分类训练用于CIFAR10数据集的训练效果不佳。CIFAR10为彩色图像且样本量更大，样本更加复杂，需要编码层提取特征的能力更强，也即编码层的结构更复杂。这里采用的是resnet18作为编码层，后续尝试resnet34作为编码层，测试正确率提高到51%左右。根据查询资料得，当采用resnet50时，测试正确率有机会达到70%，但是由于个人电脑配置问题，无法继续训练resnet50作为编码层。但是由此训练过程可得，SimCLR+分类训练可用于彩色图像分类，再次验证它的正确性。