Tiny-Caltech35 分类
实验报告

姓名：秦圣岭

班级：无 94

学号：2019011076

一、 环境配置

本次实验使用 CUDA 进行加速。代码运行环境为

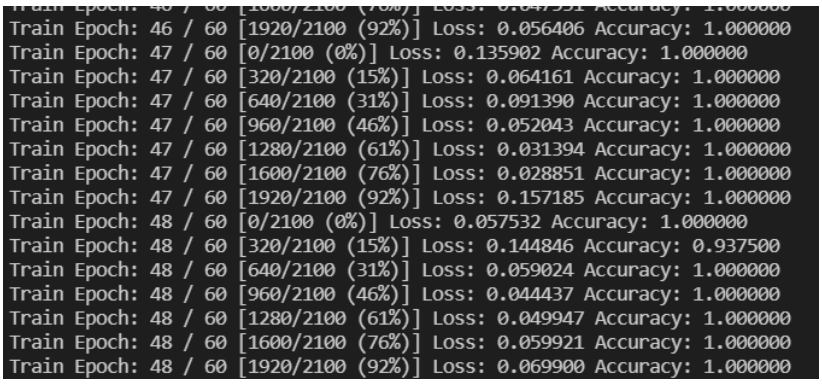
环境	版本
python	3.8.8
torch	1.8.1+cu111
torchvision	0.9.1+cu111
scikit-learn	0.24.2
numpy	1.20.2
scipy	1.6.3
matplotlib	3.3.4

编写环境测试代码，运行结果如下：

结果正确，并且显示支持 CUDA。如需使用 CPU 版本的 pytorch 运行模型，请将 main.py 中第 18 行 device = torch.device('cuda')改为 device = torch.device('cpu')。

二、 基准模型的运行

直接运行基准模型，截取部分训练阶段的结果如下：



训练结束后，在验证集和测试集上的准确度分别为 46.6%和 45.8%。

```
==========================
val accuracy:46.57142639160156%
test accuracy:45.78571319580078%
```

三、 可视化

1. loss 及准确率的可视化

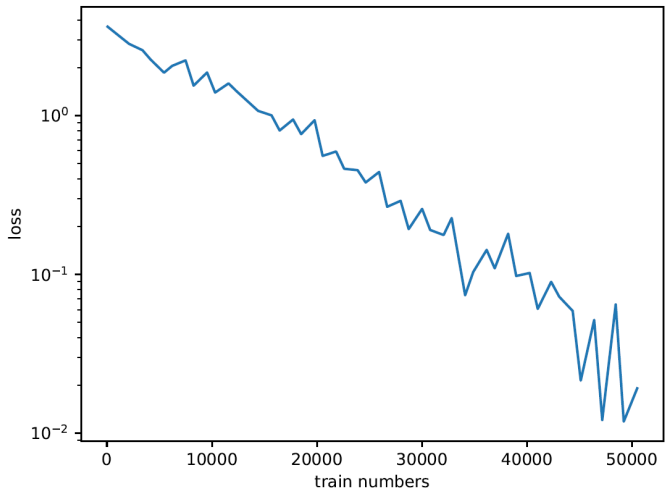
可视化代码如下：

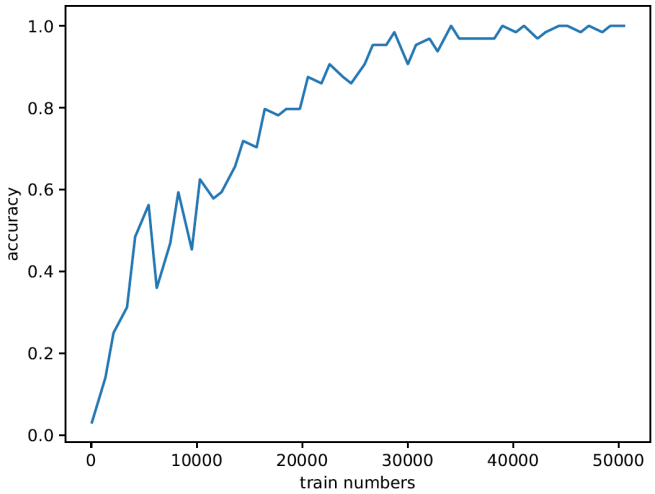
```
# you may need train_numbers and train_losses to visualize something
train_numbers, train_losses, train_accuracy = train(
    config, train_loader, model, optimizer, scheduler, creiteron)

draw(train_numbers, train_losses, 'log',
     'train numbers', 'loss', 'loss.pdf')
draw(train_numbers, train_accuracy, 'linear',
     'train numbers', 'accuracy', 'accuracy.pdf')
```

其中 draw 函数定义如下：

```
def draw(x_data, y_data, y_scale, x_label, y_label, filename):
    plt.figure()
    plt.axes(yscale=y_scale)
    plt.plot(x_data, y_data)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.savefig(filename)
    plt.close()
```

使用基准网络架构进行训练，设置学习率为 0.0035，epoch 为 25。记录训练过程中 loss 和准确率随训练样本数的变化，得到 loss 及准确率的可视化结果如下（loss 取对数坐标）：





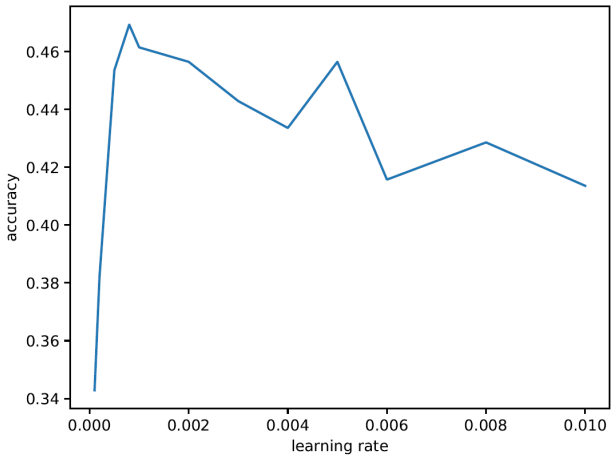
可见在训练过程中，在初始阶段，loss 下降较快，准确率上升也较快，此时网络正在学习样本的特征；训练接近结束时，loss 在一个较低值附近震荡，而准确率接近 1，说明此时网络已经训练完成。

2. 超参数对性能的影响

取不同的学习率测试性能，分别设定学习率为 0.01、0.008、0.006、0.005、0.004、0.003、0.002、0.001、0.0008、0.0005、0.0002、0.0001，观察准确率随学习率的变化情况，测试代码如下：

```
learning_rate = [0.01, 0.008, 0.006, 0.005, 0.004,
                 0.003, 0.002, 0.001, 0.0008, 0.0005, 0.0002, 0.0001]
accuracy = []
for rate in learning_rate:
    config.learning_rate = rate
    accuracy.append(main(config))
draw(learning_rate, accuracy, 'linear',
     'learning rate', 'accuracy', 'rate.pdf')
```

使用基准网络架构，改变学习率，在测试集上的准确率结果如下：



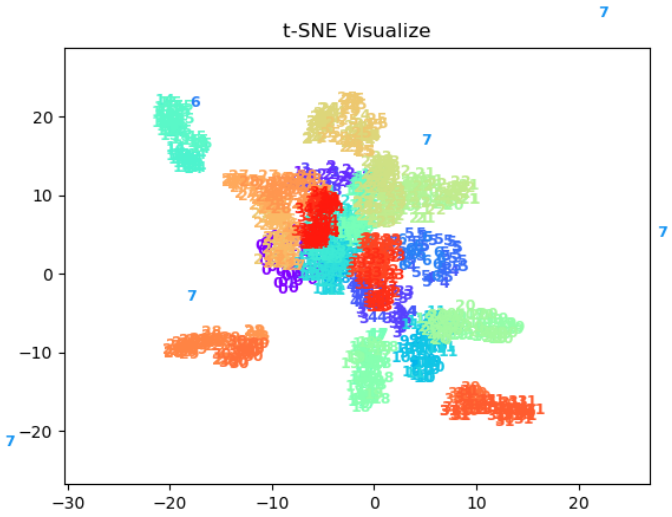
可见学习率较高时，准确率略有下降；学习率低于 0.001 时，准确率随学习率的降低而快速下降。当学习率在 0.002~0.008 之间变化时，最终的准确率在 0.45 左右小幅度变化，推测应该是网络架构限制了准确率的上升。若要获得更高的准确率，可以使用更复杂的网络架构。

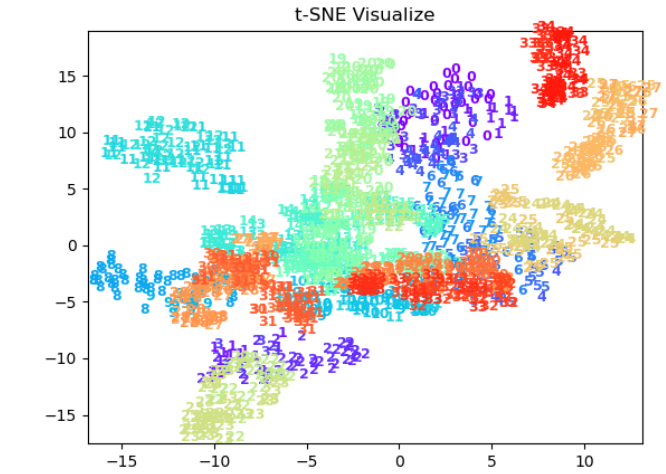
3. 样本的特征分布

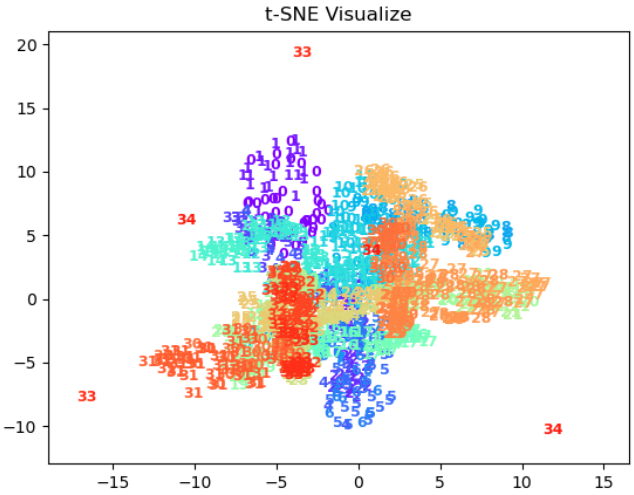
使用 t-SNE 对数据进行可视化，将降维后的数据使用不同的颜色画在 XY 平面上，并使用 matplotlib 显示数据。具体实现代码如下：

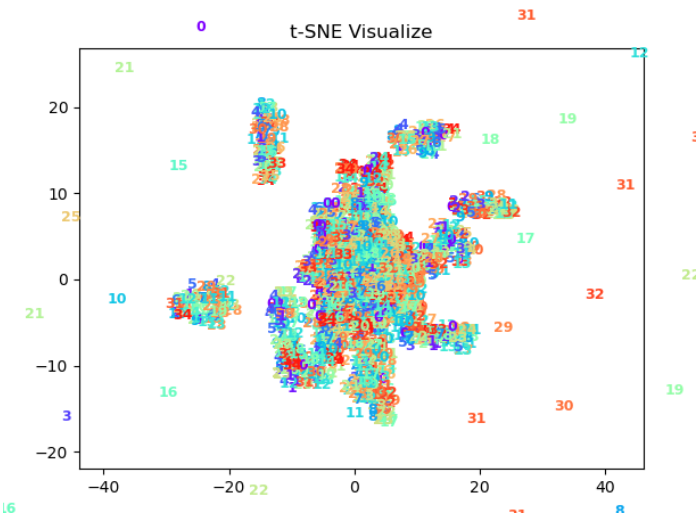
```
        # tsne visualize
        low_dim_embs = tsne.fit_transform(
            last_layer.cpu().data.numpy()[:plot_only, :])
        labels = label.cpu().numpy()[:plot_only]
        X, Y = low_dim_embs[:, 0], low_dim_embs[:, 1]
        xmin = min(xmin, X.min())
        xmax = max(xmax, X.max())
        ymin = min(ymin, Y.min())
        ymax = max(ymax, Y.max())
        for x, y, s in zip(X, Y, labels):
            c = cm.rainbow(int(255 * s / 35))
            plt.text(x, y, s, color=c, fontdict={
                'weight': 'bold', 'size': 9})
plt.xlim(xmin, xmax)
plt.ylim(ymin, ymax)
plt.title('t-SNE Visualize')
plt.show()
```

取全部 35 个类，每个点的数字表示该样本所属的类别，不同的类别使用不同的颜色。得到几个较好的可视化结果如下：





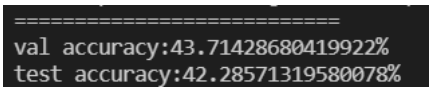




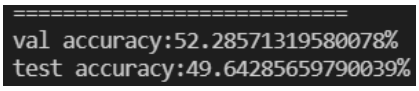
可见每一个类的点较为集中，分布在一个小区域内；不同的类别之间有较为明显的分界，且样本之间有一定的距离。这说明使用 t-SNE 进行降维的结果较为成功。

四、 调整网络结构

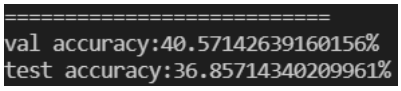
1. 在基准网络的基础上修改，添加一个 64 到 64 的卷积层、一个 256 到 512 的卷积层和一个全连接层，总共 5 层卷积层和 3 层全连接层。训练 20 轮，设置学习率为 0.0035。得到验证集上的准确率为 43.7%，测试集上的准确率为 42.3%。



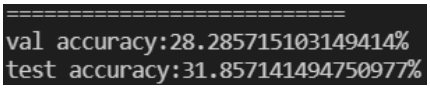
2. 使用自己搭建的 VGG 模型，具体模型见代码。此 VGG 模型采用 5 层卷积层，3 层池化层，2 层全连接层，使用 3*3 的卷积核。训练 20 轮，设置学习率为 0.001。得到验证集上的准确率为 52.3%，测试集上的准确率为 49.6%。



3. 使用 torch 中已有的 ResNet 模型，即调用 torchvision.models 中的 resnet18()，训练 20 轮，设置学习率为 0.0035。得到验证集上的准确率为 40.6%，测试集上的准确率为 36.9%。



4. 使用 torch 中已有的 AlexNet 模型，即调用 torchvision.models 中的 alexnet()，训练 50 轮，设置学习率为 0.001，得到验证集上的准确率为 28.3%，测试集上的准确率为 31.9%。

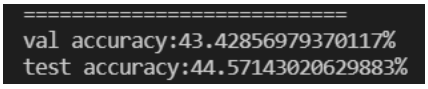


可以看到，网络架构并非越复杂性能越好。在 Tiny-Caltech35 数据集上，单纯的 CNN 的效果较好，如 base model 和 VGG 网络；而其他网络架构相对不适用，比如 ResNet 和 AlexNet，在该数据集上的表现较差，学习较慢，最终准确率也不如普通的 CNN。

五、 调整损失函数

在基准网络的基础上，使用不同的损失函数，观察准确率的变化情况。

1. 使用交叉熵损失函数 nn. CrossEntropyLoss()，设置学习率为 0.0035，epoch 为 20。得到验证集上的准确率为 43.4%，测试集上的准确率为 44.6%。



2. 使用均方损失函数 nn. MSELoss()，设置学习率为 0.01，epoch 为 30。由于均方损失函数通常在回归问题中使用，其输入的数据为两组向量，因此调整 output 和 label 的类型，做出如下修改：

```
output, idx = output.max(1)
output = output.float()
label = label.float()
loss = creiteron(output, label)
```

得到验证集上的准确率为 2.0%，测试集上的准确率为 1.7%。在学习的过程中，loss 在 10+的水平，和理想状况相差很远。可见均方损失函数完全不适用于该模型，因为该模型是分类问题，而均方损失函数通常用在回归问题。

```
==========================
val accuracy:2.0%
test accuracy:1.7142856121063232%
```

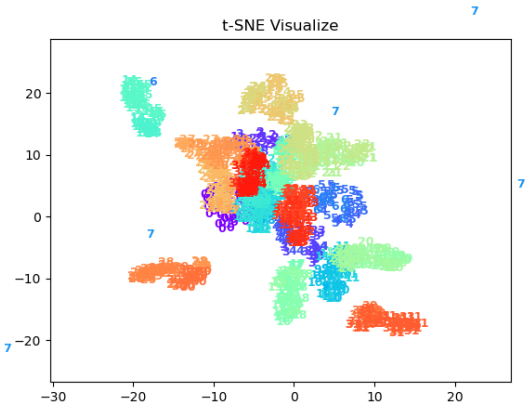
3. 使用平均绝对损失函数 nn. L1Loss()，设置学习率为 0.01，epoch 为 30。得到验证集上的准确率为 2.0%，测试集上的准确率为 2.2%。可见平均绝对损失函数也不适用于该模型。

```
==========================
val accuracy:2.0%
test accuracy:2.2142858505249023%
```

4. 使用函数 nn. SoftMarginLoss()，设置学习率为 0.01，epoch 为 30。得到验证集上的准确率为 3.7%，测试集上的准确率为 3.2%。可见该函数也不适用于该模型。

```
==========================
val accuracy:3.7142858505249023%
test accuracy:3.2142856121063232%
```

分析：在以上四种损失函数中，只有交叉熵损失函数是用于解决多分类问题的，而均方损失函数和平均绝对损失函数是用于解决回归问题的，SoftMarginLoss 是用于解决二分类问题的。因此在性能上，交叉熵损失函数要明显优于其他函数。交叉熵在计算损失时，先使用了 Softmax 将输入的数据转化为概率，再进行计算，在转化时使用了指数以放大不同输入之间的差别，因此在反向传播时会带来更大的梯度，从而使网络参数的更新较快。



在 t-SNE 可视化后，数据呈现出部分混叠的情况，在中心区域，一些数据的特征较为接

近，则计算的结果也会较为接近。此时采用其他损失函数不能有效计算出 loss，而使用交叉熵损失函数可以将数据更好地区分开，增大类之间的差异，对分类错误加以较高的惩罚，因此交叉熵函数的性能较好。

六、 添加数据，再次进行训练

将验证集和补充数据集的数据添加进样本中进行训练，即使用 train、val、addition 三个数据集中的数据作为训练样本。新的训练集的代码如下：

```
train_dataset = tiny_caltech35(transform=transform_train, used_data=[
                                'train', 'val', 'addition'])
```

使用基准模型进行训练，设置学习率为 0.0035，epoch 为 20。最终得到在验证集上的准确率为 97.7%，在测试集上的准确率为 50.5%。

```
============================
val accuracy:97.71428680419922%
test accuracy:50.5%
```

在 t-SNE 降维后得到的样本分布中，可以观察到验证集和测试集的样本分布较为接近。但是在将验证集添加进训练集后，最终在测试集上的准确率只是略有提升。在“调整网络结构”的过程中，无论采用何种网络架构，准确率的上限都是 50%左右。可以推测，限制准确率进一步提高的主要因素是训练数据过少，没有全面刻画数据特征，样本之间的差异过大，导致网络没有学习到样本的本质特征，泛化能力不足，在测试集上的表现始终较差。这是数据集的局限性。

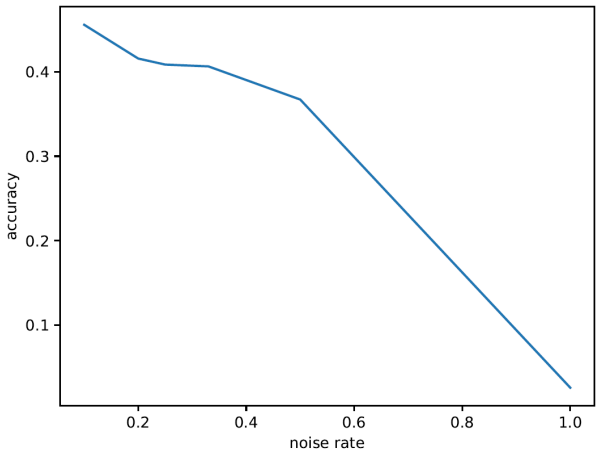
七、 抗噪声样本研究

在训练样本中添加噪声，调整噪声的比例，观察准确率的变化情况。

添加噪声的方法如下。在 label 中，按概率挑选一定比例的数据，将这些数据的标签设为一个随机值。

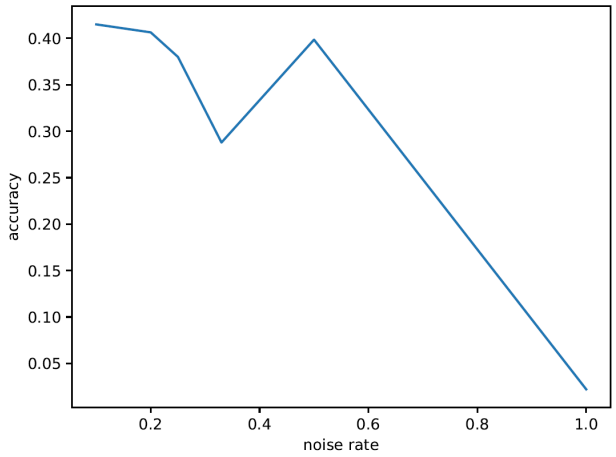
```
if config.noise != 0:
    noise_select = int(1/config.noise)
    for i in range(label.numel()):
        if random.randint(0, 10000) % noise_select == 0:
            label[i] = random.randint(0, 34)
```

从 10%的噪声开始，逐步增大噪声的比例，测试在不采取抗噪声的措施时，准确率的变化情况。噪声比例取 10%、20%、25%、33%、50%、100%，学习率取 0.0035，epoch 取 20 时，准确率的变化情况如下：



可以看到，当噪声比例提高时，准确率迅速下降。噪声比例在 0~50%时，准确率下降较为缓慢，而当噪声比例在 50%~100%时，准确率迅速下降，此时噪声已经对网络产生了严重的影响。

在抗噪声措施上，采用更加深的网络结构来对抗噪声样本。采用在“调整网络架构”中设计的 VGG 网络进行训练，学习率取 0.0035，epoch 取 35，准确率的变化情况如下:



可以看到，除了在噪声比例为 33%时表现较差外，在噪声比例为 0~50%的点表现均优于基准网络。因此可知，可以使用更深层的网络来对抗噪声样本，减小噪声对于网络参数的影响，优化网络的学习能力。