《人工智能导论》大作业

任务名称：带 OOD 检测的 Mnist 分类器

小组人员：陈子钏 林丹琪 王一寒 江晓雨

完成时间：2023/06/17

1. 任务目标

基于 Mnist 数据集和非数字图像数据集，构建一个分类模型，模型可以对 mnist 数据集中的图像正确识别所代表的数字，同时对于非数字的图像，识别为 OOD 类。

2. 具体内容

2.1. 数据集

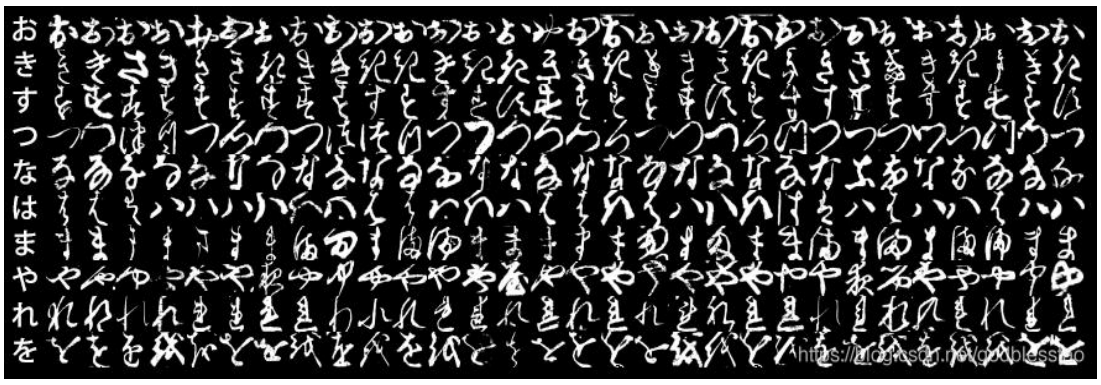
2.1.1. 数据集准备

使用 pytorch 的 datasets 中的数据集，其中的每个数据集都分有训练集和测试集。

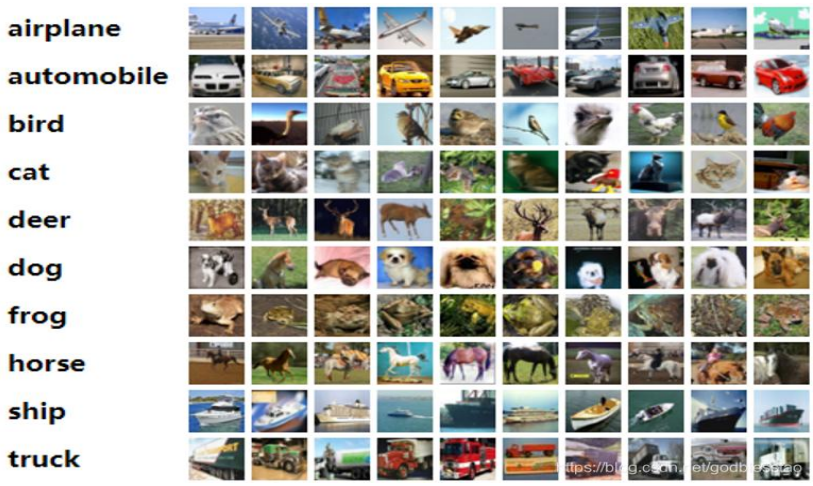
各数据集简介如下：

1. MNIST：手写数字的数据集，共计 70000 张 28x28 灰度图像。其中，训练集包含 60000 张图像，测试集包含 10000 张图像

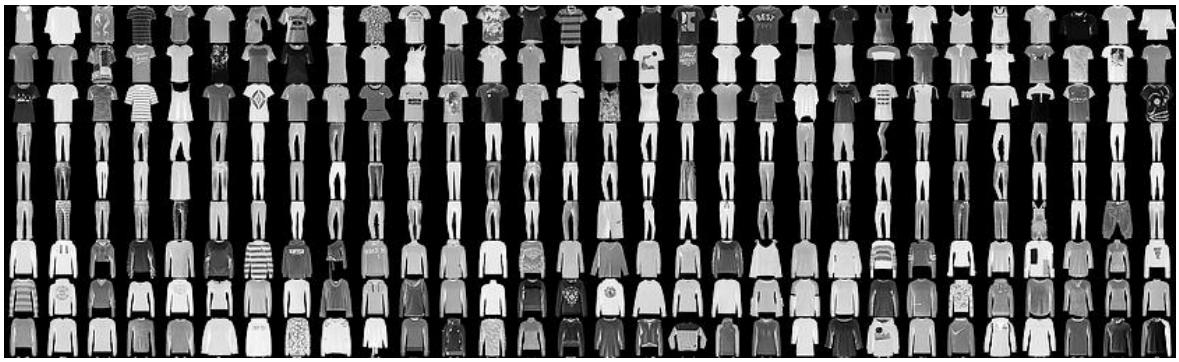
2. Kmnist：手写日语数据集，共计 70000 张 28x28 灰度图像。其中，训练集 60000 张，测试集 10000 张



3. CIFAR-10：图像分类数据集，包含 50000 张 32x32 的 RGB 彩色图像，训练集 50000 张，测试集 10000 张



4. fashionMNIST：图像分类数据集，包含 60000 张 28x28 的灰度图像，其中，训练集 50000 张，测试集 10000 张



2.1.2. 数据处理代码

1. OOD 数据集预处理

(1) 转换为 28*28 灰度图像：transforms.Resize 函数 + grayscale 函数

(2) 增强对比度：transforms.RandomAutocontrast 函数

(3) 直方图均衡化：transforms.RandomEqualize 函数

(4) 标签更改为 10（OOD 类）

以 CIFAR 数据集为例，代码如下图所示

```
# 加载 CIFAR-10 数据集
# 加载过程中，调整 CIFAR-10 图像大小并转换为灰度图像
transform_cifar10 = transforms.Compose([
    transforms.Resize(28),
    transforms.Grayscale(num_output_channels=1),
    transforms.transforms.RandomEqualize(p=1),       #------直方图均衡化
    transforms.RandomAutocontrast(p=1),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])


train_cifar10 = datasets.CIFAR10(root='./data/', train=True, download=True, transform=transform_cifar10)
test_cifar10 = datasets.CIFAR10(root='./data/', train=False, transform=transform_cifar10)

# 修改 CIFAR-10 的 label , test集和train集都要修改
label_list1 = train_cifar10.targets
label_list1 = [10 for label in label_list1]

label_list2 = test_cifar10.targets
label_list2 = [10 for label in label_list2]

train_cifar10.targets = torch.tensor(label_list1, dtype=torch.long)
test_cifar10.targets = torch.tensor(label_list2, dtype=torch.long)

train_cifar10.targets = train_cifar10.targets.tolist()
test_cifar10.targets = test_cifar10.targets.tolist()
```

2. 数据集整合：ConcatDataset 函数

```
# 合并数据集

train_data = ConcatDataset([train_cifar10, train_mnist, train_kmnist, train_fmnist])
test_data = ConcatDataset([test_cifar10, test_mnist, test_kmnist, test_fmnist])
```

需要指出，在机器学习的任务中，数据集太大并不会导致过拟合。相反，数据量越大，模型一般越容易得到更好的泛化性能。故尽管 OOD 远大于 mnist 数据集，我们选择不对数据集进行缩减。

2.2. 卷积神经网络 CNN

2.2.1. 方案选择：

1. 正则化：有/无 dropout(弃权技术)

正则化可以有效防止模型的过拟合。Dropout 正则化对于某层的每个神经元，在训练阶段均以概率 p 随机将该神经元权重置零。测试阶段所有神经元都呈激活态，权重乘以 (1−p)，相当于平均集成(average ensemble)。

设计两个方案，比较加入 dropout 正则化的效果：

方案 1：两层卷积层+池化层+正则化+全连接层+正则化

方案 2：两层卷积层+池化层+全连接层

比较结果如下，可以看出加入 dropout 正则化后，精度明显提高，故选择加入正则化的方案

	Epoch1		Epoch2		Epoch3		Epoch4		Epoch5	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
有 dropout 正则化	**97.4082**	**99.21**	**99.0318**	**99.365**	**99.2827**	**99.325**	**99.4173**	**99.45**	**99.4936**	**99.44**
无 dropout 正则化	90.4618	95.925	96.3845	97.5	97.4445	98.22	97.9618	98.515	98.1882	98.545

2. 卷积层：2/3/4 个卷积层

通常情况下，卷积层数越多，模型的复杂度越高，学习程度更深。然而，层数过高可能会导致过拟合的问题。为了选择最佳的卷积层数，设计三个方案，比较 2/3/4 层卷积的训练效果。增加层数，效果并没有显著提高，2 层卷积的效果相对最佳，选择 2 层卷积。

	Epoch1		Epoch2		Epoch3		Epoch4		Epoch5	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
2 层卷积	**97.4082**	**99.2100**	**99.0318**	99.365	**99.2827**	99.325	**99.4173**	**99.45**	**99.4936**	99.44
3 层卷积	96.6182	99.0950	99.0145	**99.46**	99.2336	**99.525**	99.3727	99.38	99.4473	**99.46**
4 层卷积	96.1182	99.1300	99.0073	99.325	99.2273	99.48	99.3836	99.42	99.4564	99.42

3. 卷积核大小：3*3 / 5*5

考虑 MNIST 数据集中是 28x28 的灰度图像，在 MNIST 数据集上进行卷积操作时，常见的做法是使用 3x3 或 5x5 的卷积核。为了选择合适的卷积核大小，设计两个方案，比较两种卷积核效果，可以看出 5*5 卷积核略优于 3*3，选择 5*5 卷积

	Epoch1		Epoch2		Epoch3		Epoch4		Epoch5	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
3*3	96.6182	99.0950	99.0145	99.46	99.2336	**99.525**	99.3727	**99.38**	99.4473	99.46
5*5	**99.5382**	**99.5200**	**99.5318**	**99.51**	**99.6027**	99.435	**99.6**	99.17	**99.6564**	**99.56**

2.2.2. 代码解析：

1. 卷积神经网络模型

（1）卷积层 + 池化层

① 第一层卷积+池化：conv1 卷积+relu 激活函数+2*2 池化

② 第二层卷积+池化：conv2 卷积+relu 激活函数+ dropout 正则化(弃权技术)+2*2 池化

③ 第三层卷积+池化：conv3 卷积+relu 激活函数+2*2 池化

（2）全连接层

① 第一层全连接层：fc1 + relu 激活函数 + dropout 正则化(弃权技术)

② 第二层全连接层：fc2

（3）优化器：采用 Adam 优化器可以更快地收敛，提高模型的收敛速度和稳定性。设置 weight_decay 参数为 0.01，用于控制正则化强度的值；设置学习率 lr 参数为 0.01，控制在一个较低的值以保证训练效果

（4）训练周期：进行 5 轮训练

```
class Model(torch.nn.Module):

    def __init__(self):
        super(Model, self).__init__()

        # 添加卷积层，使用L2正则化
        self.conv1 = torch.nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1)
        self.conv2 = torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.conv3 = torch.nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)

        # 添加全连接层，使用L2正则化
        self.fc1 = torch.nn.Linear(64*3*3, 1024)
        self.fc2 = torch.nn.Linear(1024, 11)

        # 定义正则化项
        self.regularizer = torch.nn.MSELoss()

    def forward(self, x):
        # 卷积层
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2)

        # 卷积层，使用Dropout正则化
        x = F.relu(self.conv2(x))
        x = F.dropout(x, p=0.2)
        x = F.max_pool2d(x, 2)

        # 卷积层，使用L2正则化
        x = F.relu(self.conv3(x))
        x = F.max_pool2d(x, 2)

        # 全连接层，使用L2正则化
        x = x.view(-1, 64*3*3)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, p=0.5)
        x = self.fc2(x)
        return x
```

```
optimizer = optim.SGD(model.parameters(), lr=0.01, weight_decay=0.01)
```

2. 模型训练过程

相关变量：

（1） 用 data_loader_train 和 data_loader_test 加载训练数据集，根据 batch_size(64)将该数据集分为若干个 batch

（2） n_epochs 记录需要训练的次数(5 次)

（3） running_loss 记录当前周期损失，running_correct 记录正确分类的样本数量

训练过程：

（1） 从每个 batch 中取出输入数据 X_train 和标签数据 y_train，并将它们封装为 PyTorch 中的 Variable 类型，然后将输入数据 X_train 传入模型中计算得到输出 outputs

（2） 使用 torch.max 函数获取 outputs 中每个样本预测结果得分最高的类别，并将其作为预测结果 pred

（3） 通过 optimizer.zero_grad()清空上一次计算得到的梯度信息，准备计算本次的梯度信息

（4） 使用损失函数 cost 计算预测结果与标签数据之间的损失值 loss

（5） 使用 loss.backward()函数计算损失值对模型参数的梯度信息，并使用 optimizer.step()函数更新模型参数的值，以使损失值得到最小化

（6） 统计本次训练过程中的损失值 running_loss 和预测正确的样本数 running_correct

验证过程：

（1） 从每个 batch 中取出输入数据 X_test 和标签数据 y_test，并将它们封装为 PyTorch 中的 Variable 类型，然后将输入数据 X_train 传入模型中计算得到输出 outputs。

（2） 使用 torch.max 函数获取 outputs 中每个样本预测结果得分最高的类别，并将其作为预测结果 pred

（3） 统计本次测试过程中预测正确的样本数 testing_correct。对于一个 batch 的数据，使用 torch.sum(pred == y_test.data)函数计算预测正确的样本数，并将其累积到 testing_correct 中

（4） 每一轮训练，计算输出训练集上的损失、准确率和测试集上的准确率。

在训练结束后，保存模型参数。

具体代码如下：

```python
for epoch in range(n_epochs):
    running_loss = 0.0
    running_correct = 0
    print("Epoch {}/{}".format(epoch, n_epochs))
    print("-"*10)
    for data in data_loader_train:
        X_train, y_train = data
        X_train, y_train = Variable(X_train), Variable(y_train)
        outputs = model(X_train)
        _,pred = torch.max(outputs.data, 1)
        optimizer.zero_grad()
        loss = cost(outputs, y_train)

        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        running_correct += torch.sum(pred == y_train.data)
    testing_correct = 0


    for data in data_loader_test:
        X_test, y_test = data
        X_test, y_test = Variable(X_test), Variable(y_test)
        outputs = model(X_test)
        _, pred = torch.max(outputs.data, 1)
        testing_correct += torch.sum(pred == y_test.data)
    print("Loss is:{:.4f}, Train Accuracy is:{:.4f}%, Test Accuracy is:{:.4f}".format(running_loss/len(train_data),
                                                                                      100*running_correct/len(train_data),
                                                                                      100*testing_correct/len(test_data)))

# 保存训练参数
torch.save(model.state_dict(), "model_parameter.pkl")
```

2.3. 接口函数

1. **初始化函数__init__(self)**

    self.M=Model()，调用 OOD type.py 里的 Class Model 并将这个对象实例化。

    self.M.load_state_dict(torch.load('model_parameter.pkl'))，加载 CNN 分类器的参数。

2. **其他处理函数 misc(self, x)**

    针对输入的 1*28*28 维度的图片进行处理，使数字更容易被识别，OOD 更容易被区分

    包括：转为灰度图像

3. **接口函数 classify(self,imgs :torch.Tensor) -> torch.Tensor**

    classify()中先对给出的 tensor 进行装载，然后调用 self.M()产生结果。

    函数的返回值：preds 为整数型 n 维的 tensor，整数代表分类值。

具体代码如下：

```python
class OodCls:
    def __init__(self):
        self.M = Model()
        self.M.load_state_dict(torch.load('model_parameter_ood.pkl'))

    def misc(self, x):
        # 转为灰度图像
        x_gray = transforms.Grayscale(x)

    def classify(self, imgs : torch.Tensor) -> torch.Tensor:

        # 获取输入张量的维度
        num_dims = imgs.ndim

        data_loader = torch.utils.data.DataLoader(dataset = imgs,
                                                  batch_size = num_dims,
                                                  shuffle = True)
        x = next(iter(data_loader))
        x_modify = self.misc(x)
        x = Variable(x)
        preds = self.M(x)
        _, preds = torch.max(preds, 1)

        return preds
```

3．工作总结

3.1. 收获、心得

1. 通过接口文件的编写，对接口、以及面向对象的思想有了更深入的了解，想通了接口函数 classify 的具体思路以及模型具体如何初始化。

2. 学习了 CNN 的相关的内容，了解了 CNN 图像分类优化的一些知识，包括多卷积层和卷积核大小等因素对于 CNN 提高泛化能力的结果。

3. 在完成本次作业的过程中，我的感触最大还是要做好实践前的前期准备，才能更顺利的进行。

3.2. 问题及解决思路

1. CIFAR 数据集的处理：一方面 CIFAR 在转黑白图像后灰度较高，另一方面在数据集合并时，遇到了标签数据类型变化的问题。解决思路：另外定义一个 CIFAR 集的 transformer 初始化函数，增强对比度和直方图均衡化；查找资料后使用相关函数进行 tensor 变量和 list 变量相互转化

2. 接口函数：最开始做文件编写的时候感到棘手毫无头绪，不知道支持文档中描述的功能具体如何实现，在寻找了网络资源、并仔细阅读理解了组长发来的代码和参考文章之后略有了思路，最后实现了接口类 OodCls。