Lab 02 Report

实验内容

验要求 使用 pytorch实现卷积神经网络 CNN，在 CIFAR-10 数据集 上进行图片分类。研究 dropout、normalization、early stop、learning rate decay、卷积核大小、 网络深度等超参数对分类性能的影响。

实验步骤

1.构造CNN类

基于是否含dropout、normalization、不同的卷积核大小 构建卷积神经网络类 `CNN`。以卷积核大小为3*3，利用dropout和batch normlization方法为例构建的CNN

```
    class CNN(nn.Module):
        def __init__(self):
            super(CNN, self).__init__()
            self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
            self.bn1 = nn.BatchNorm2d(32)
            self.dropout = nn.Dropout(p=0.4)
            self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
            self.bn2 = nn.BatchNorm2d(64)
            self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
            self.bn3 = nn.BatchNorm2d(128)
            self.pool = nn.MaxPool2d(2, 2)
            self.fc1 = nn.Linear(128 * 4 * 4, 512)
            self.bn4 = nn.BatchNorm1d(512)
            self.fc2 = nn.Linear(512, 10)

        def forward(self, x):
            x = self.pool(nn.functional.relu(self.bn1(self.conv1(x))))
            x = self.pool(nn.functional.relu(self.bn2(self.conv2(x))))
            x = self.pool(nn.functional.relu(self.bn3(self.conv3(x))))
            x = x.view(-1, 128 * 4 * 4)
            x = self.dropout(x)
            x = nn.functional.relu(self.bn4(self.fc1(x)))
            x = self.dropout(x)
            x = self.fc2(x)
            return x
```

2.数据下载、划分和封装

- **数据下载**：下载 `CIFAR10`数据并保存到data文件夹中
- **数据划分**：依据 `train_ratio` 将训练集进一步划分为训练集、验证集（`torch.utils.data.random_split`函数）
- **数据封装**：利用 `torch.utils.data.DataLoader`函数将数据分成不同的batch，并指定 `batch_size`。对于训练集，每个epoch前随机打乱训练集 `shuffle=True`，以达到充分下降。

3.定义loop

- **训练过程**：遍历每一个batch，计算输入值的输出，基于目标值（labels）计算损失（交叉熵损失）和损失的梯度，最后用 `optimizer`优化参数，并记录loss值

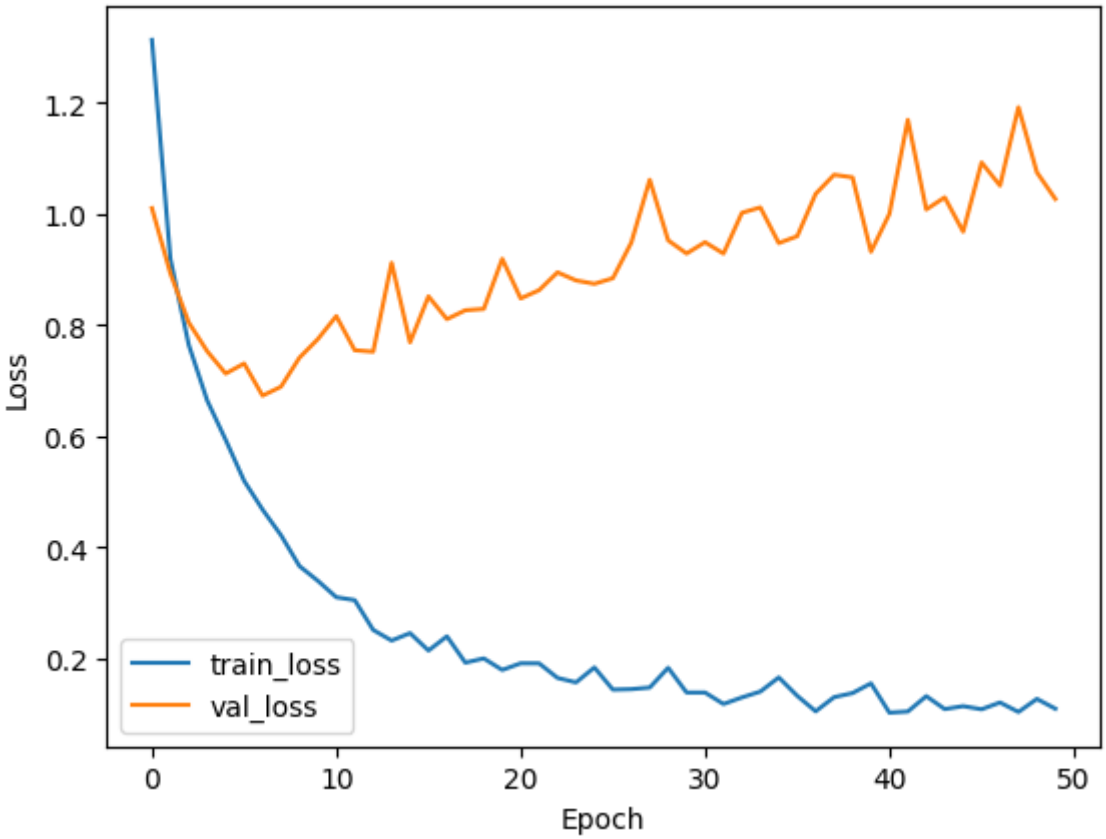
```
def train_loop(dataloader, model, optimizer, criterion = nn.CrossEntropyLoss()):
    global global_train_losses
    model.train()
    running_loss = 0
    for inputs,labels in dataloader:
        inputs = inputs.to("cuda")
        labels = labels.to("cuda")
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    global_train_losses.append(running_loss / len(dataloader)) #据此估算交叉熵损失
    return running_loss / len(dataloader)
```

- **验证/测试过程**：遍历batch，计算损失或分类正确率

4.训练模型并可视化

- **训练模型**：
    - 定义优化器(优化方法为Adam或SGD)
    - 指定 `max_num_epochs`，循环执行 `train_loop`

- 早停 因为训练集、验证集的损失函数出现了如下情况：



虽然训练集上的损失一直降低，但验证集上的损失先降低再增加，所以我们可以考虑增强泛化能力的方式，例如早停：在验证集上损失同样较低的情况下早停：当验证集上误差连续5次不大于验证集上的最低损失则停止循环。

- **模型评估**： 计算测试集上的准确率
- **可视化**:
  - 绘制训练过程中训练集和验证集损失函数

5. 参数调试

dropout、normalization、early stop、learning rate decay、卷积核大小、 网络深度进行调试。其中，dropout、normalization、卷积核大小和网络深度用于调节神经网络结构，early stop、learning rate decay调节训练方法的参数。具体结果见**实验结果**部分。

实验结果

1.调参结果

经过足够多的训练，训练集上的损失都可以降低到很小，准确率也可以达到很高（99%左右），而验证集上的误差却多在75%左右。为了增强模型的泛化性能，我们需要在验证集上调参以防止过拟合。

卷积核大小

- epoch_nums = 30时，调整神经网络相关参数，得到验证集上的loss损失和分类正确率：

dropout	loss	accuracy

dropout	loss	accuracy
True（1层）	1.00	78%
True（2层）	0.67	79%
False	1.79	74%

训练集下降不明显，但是验证集下降更充分

depth of cnn	loss	accuracy
3	1.00	78%
2	2.34	72%

size of kernal	loss	accuracy
3	1.00	78%
5	0.95	79%
7	0.89	77%

- epoch_nums = 30时，调整训练方法相关参数，得到验证集上的loss损失和分类正确率:

early stop	epochs	loss	accuracy
True	14	0.76	78%
False	30	1.00	78%

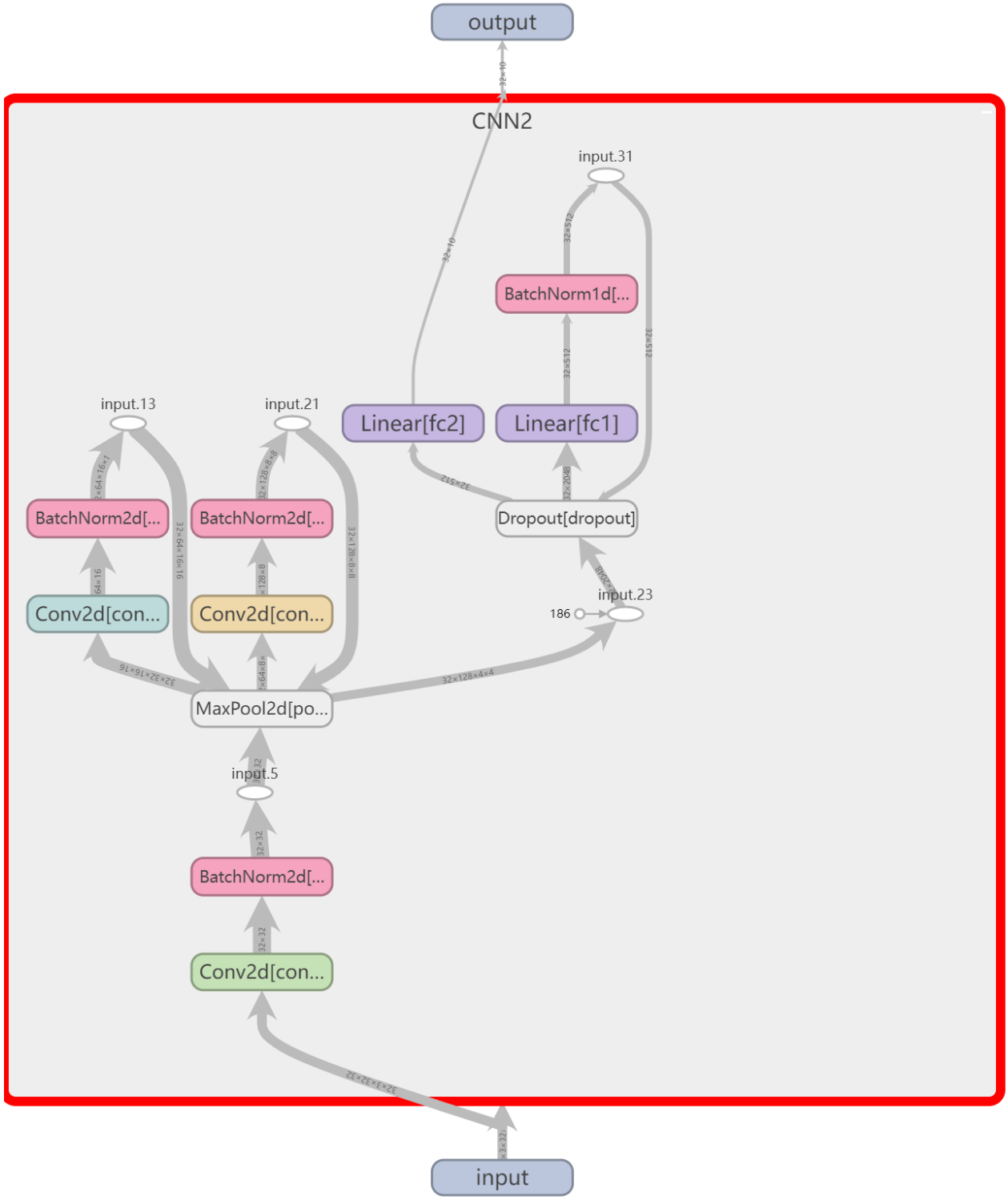
learning rate decay	loss	accuracy
True	2.30	72%
False	1.00	78%

batch-size	loss	accuracy
64	1.00	78%
32	0.57	81%

(其他调节过于繁琐，仅展示部分)

2.神经网络结构可视化

利用`tensorboard`，实现了神经网络可视化。

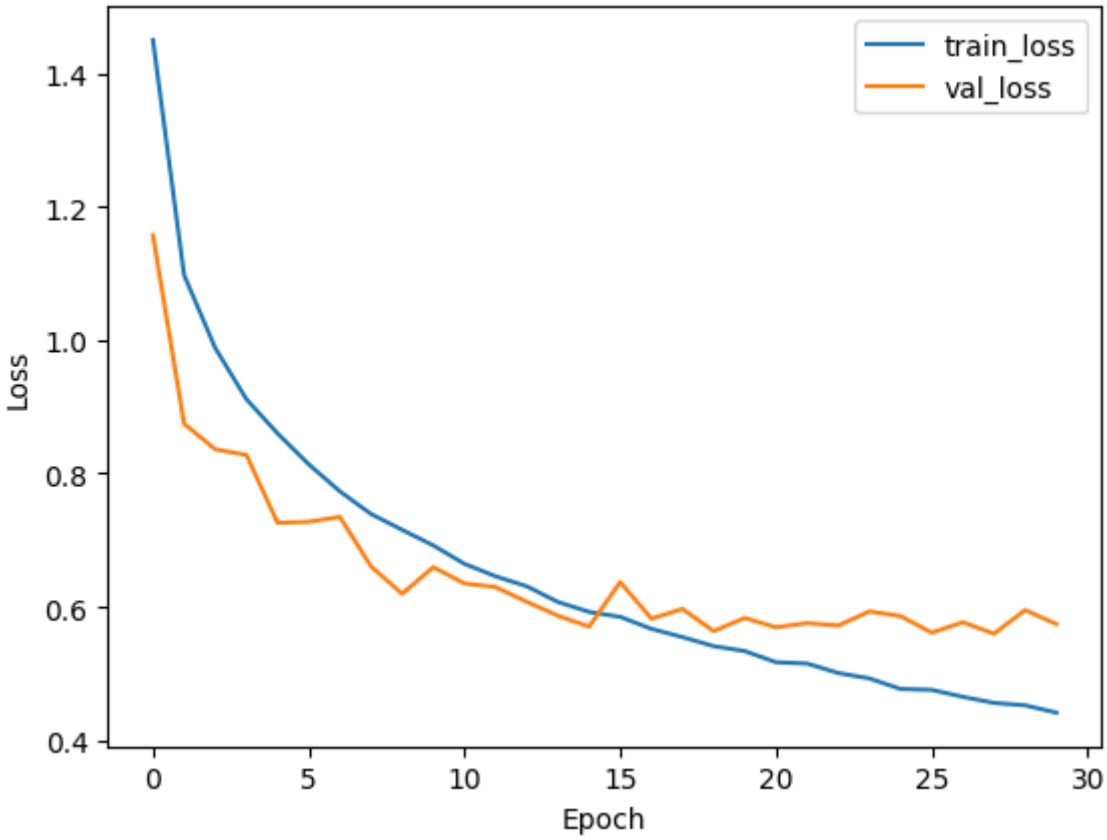


其他参数：

- batch_size=32
- max_num_epochs=30
- 优化器optimizer： Adam
- leaning rate=0.01
- activate_function = 'relu'

3.损失及准确率

- 训练集交叉熵损失：0.44
- 验证集交叉熵损失：0.57
- 验证集准确率：81%
- 测试集准确率：81%
- 损失函数图像：



结果分析

- 神经网络的参数调节
    - 3层的卷积层优于更少层的卷积层
    - 经过batch-normlization的结果更优
    - 在卷积核的大小比较小时（3-7），卷积核的大小不会明显影响准确率或loss
- 泛化性能的提高

  在训练集上，损失值低，正确率通常可以很高，但是在验证集上，损失却难以下降，正确率也几乎在75%左右波动，故我们调参的主要目标是提高泛化性能

    - dropout：通过在卷积层和全连接层加dropout，虽然训练集上误差下降不大，但是验证集下降更充分，训练的模型在验证集上表现很好。故dropout提高了模型的泛化能力。
    - early stop：我们在训练过程中 验证集上损失明显增加且训练达到一定次数时，选择早停。事实证明，这样做虽然训练集上损失会增加，但是验证集上的正确率并不比更多轮训练的结果差。这样做，还能减少训练时间

- batch size: 在训练中，选择较小的batch size,虽然会增加运行时间，但是可以大大提高泛化能力，验证集上正确率更高，损失函数更小（在一定epoch num内，训练集和验证集上的交叉熵损失变化情况几乎相同）

通过选用较小的batch-size, dropout, early stop等方法，我们成功的降低了验证集上的交叉熵损失。验证集和测试集的正确率均为81%