第二次实验报告

PB21020651 武宇星

关键代码

先将超参数单独定义出来，方便在后续调试参数过程中，在这里直接更改参数

```
epochs = 25
learning_rate = 2e-3
drop = 0.3
batch_size = 1024
gamma = 0.1
step_size = 10
device = "cuda:0" if torch.cuda.is_available() else "cpu"
```

接下来从下载好的数据集读取训练部分和测试集，并将训练部分进一步按 4：1 随机分为训练集和验证集，同时也对数据进行预处理，归一化的均值标准差参考了网上搜到的 CIFAR10 的数据处理

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261))
])
dataset = torchvision.datasets.CIFAR10(root='//data/wuyux', train=True, download=False, transform=transform)
test_dataset = torchvision.datasets.CIFAR10(root='//data/wuyux', train=False, download=False, transform=transform )
dataset_size = len(dataset)
# image_size = dataset[0].size()
# print("Image size:", image_size)
train_size = int(0.8 * dataset_size)
val_size = dataset_size - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

对于神经网络结构，我参考了 VGG 的架构，考虑到 CIFAR10 每张图片为 32*32，故采用了 6 层卷积层将张量变为 256*4*4

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, 3, padding="same")
        self.conv2 = nn.Conv2d(64, 64, 3, padding="same")
        self.maxpool = nn.MaxPool2d(2, 2)
        self.avgpool = nn.AvgPool2d(2, 2)
        self.conv3 = nn.Conv2d(64, 128, 3, padding="same")
        self.conv4 = nn.Conv2d(128, 128, 3, padding="same")
        self.conv5 = nn.Conv2d(128, 256, 3, padding="same")
        self.conv6 = nn.Conv2d(256, 256, 3, padding="same")
        self.bn1 = nn.BatchNorm2d(64)
        self.bn2 = nn.BatchNorm2d(128)
        self.bn3 = nn.BatchNorm2d(256)
        self.fc1 = nn.Linear(256 * 4 * 4, 4096)
        self.fc2 = nn.Linear(4096, 1024)
        self.fc3 = nn.Linear(1024, 10)
        self.relu = nn.ReLU()
        self.flatten = nn.Flatten()
        self.dropout = nn.Dropout(drop)
```

```
def forward(self, x):
    x = (self.relu(self.conv1(x)))
    x = self.maxpool(self.bn1(self.relu(self.conv2(x))))
    x = (self.relu(self.conv3(x)))
    x = self.maxpool(self.bn2(self.relu(self.conv4(x))))
    x = (self.relu(self.conv5(x)))
    x = self.avgpool(self.bn3(self.relu(self.conv6(x))))
    x = self.flatten(x)
    x = self.relu(self.fc1(x))
    x = self.dropout(x)
    x = self.relu(self.fc2(x))
    x = self.dropout(x)
    x = self.fc3(x)
    return x
```

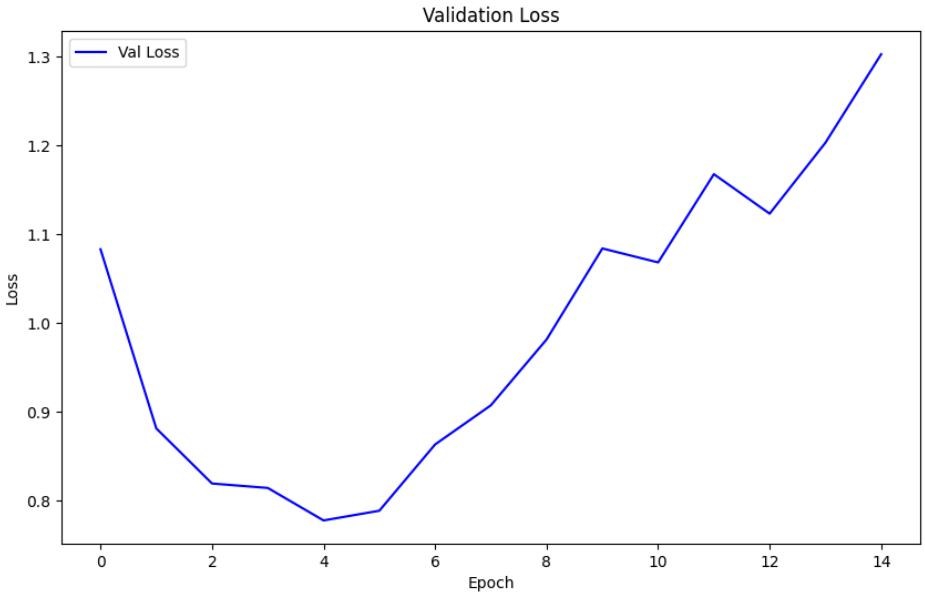
损失函数按要求选取交叉熵，优化选择 Adam，学习率衰减使用了周期性的衰减

```
model = CNN()
model.to(device)
# print(model)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=step_size, gamma=gamma)
```

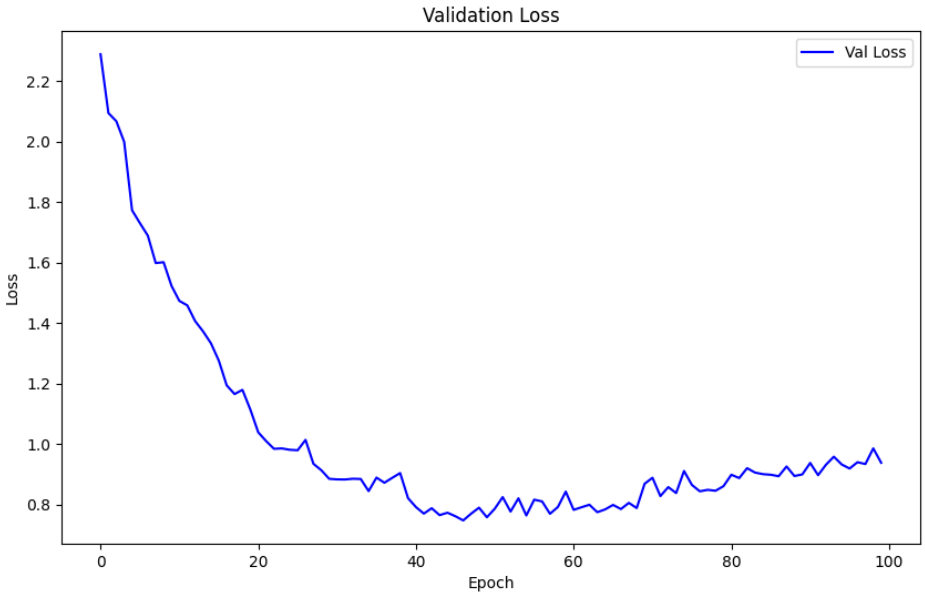
之后就是与实验一相同的训练过程

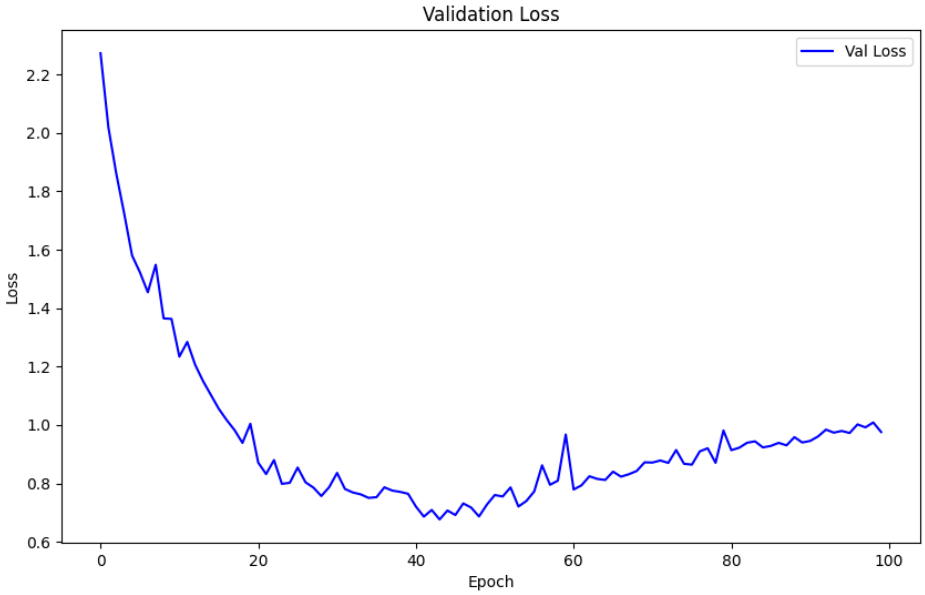
实验过程

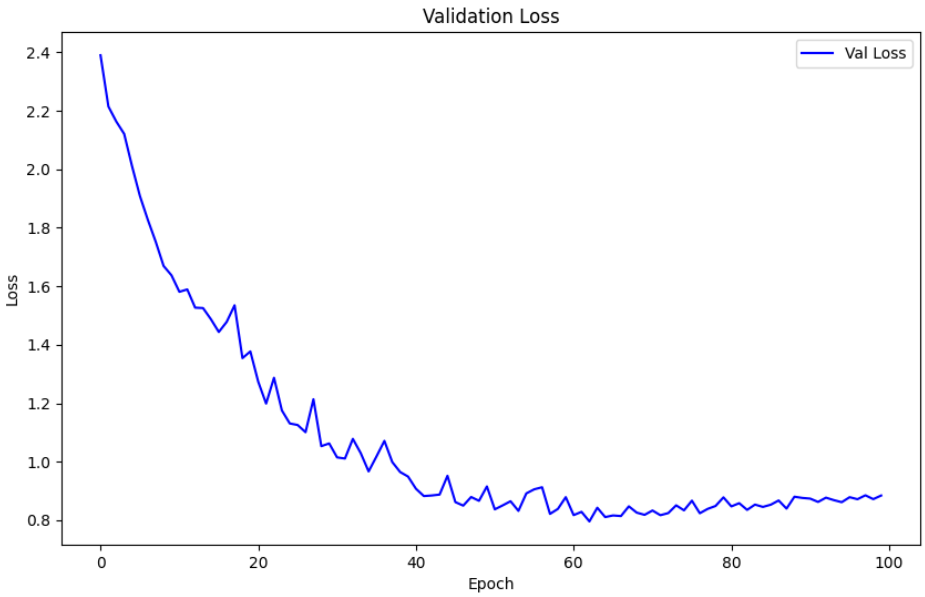
一开始自己尝试使用卷积层池化层和和全连接层搭建的网络，但验证集上正确率很低，并且 loss 函数持续走高，故参考 VGG 框架进行尝试



由于初步尝试出现在训练集上准确率很高验证集低的现象，之后的训练过程中我都加上了 dropout 并且将概率设置为 0.5 进行训练
相较于上面展示的代码，我的网络一开始是先进行 normalization，再经过 relu，同时池化层均使用最大池化，此时的验证集准确率大约在 77 左右，此时尝试对 dropout 调整，以下为 0.4-0.2 验证集上的损失函数

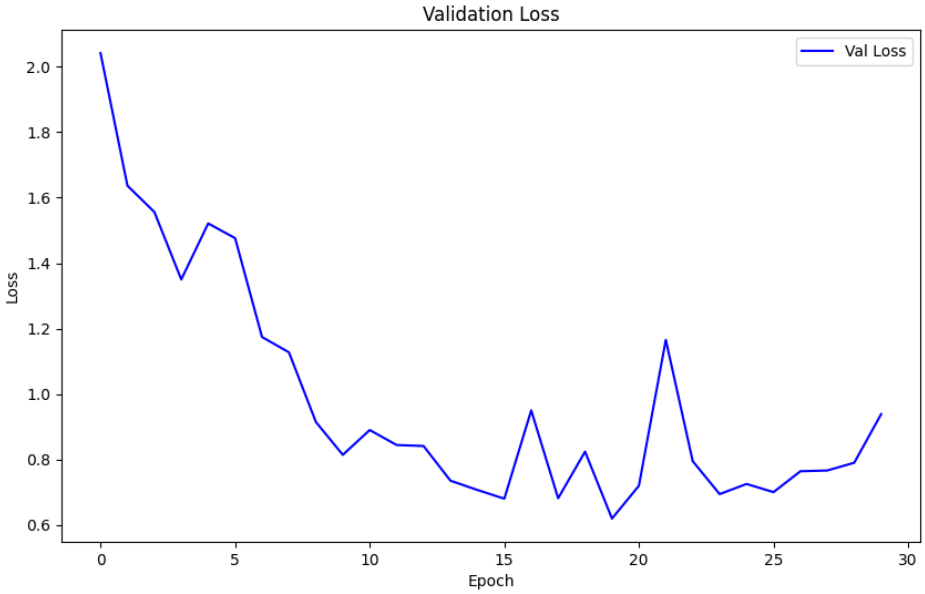






在 epoch 在 40 前时可以发现 dropout 越大越能防止过拟合问题，然而这同样也通过了学习率衰减的调试为每 20 个 epoch 乘 0.5，使曲线平滑；在这样的调整后尽管越小越平滑但 0.3 的正确率比 0.2 高，我认为这是过拟合和欠拟合之间的权衡问题

而在最终结果中，我也尝试了不用 dropout 正确率约为 80，而最高的则为 dropout0.3 约为 83，对于最终的网络我并没有对所有所有卷积层后进行 normalization，因为这样以后我发现训练集上收敛很快但验证集很糟糕



一开始以为是 batchsize 过小 32，但取 128，256，1024 后问题仍存在，删减为两层卷积仍存在该问题，引入残差也存在该问题，不确定为什么。

最后调整学习率为 0.002，25 轮左右即可得到验证集上的最好结果，最终测试集为 82 准确率

```
(py39) wuyux@Sui4:~/dl/pytorch/exp2$ python3 exp2.py
Epoch 1/25, Train Loss: 1.9607, Train Accuracy: 0.3243, Val Loss: 1.8048, Val Accuracy: 0.3878
Epoch 2/25, Train Loss: 1.3774, Train Accuracy: 0.4946, Val Loss: 1.3128, Val Accuracy: 0.5176
Epoch 3/25, Train Loss: 1.1379, Train Accuracy: 0.5911, Val Loss: 1.1767, Val Accuracy: 0.5729
Epoch 4/25, Train Loss: 0.9739, Train Accuracy: 0.6532, Val Loss: 1.0609, Val Accuracy: 0.6353
Epoch 5/25, Train Loss: 0.8177, Train Accuracy: 0.7116, Val Loss: 0.9288, Val Accuracy: 0.6863
Epoch 6/25, Train Loss: 0.6681, Train Accuracy: 0.7631, Val Loss: 0.7591, Val Accuracy: 0.7395
Epoch 7/25, Train Loss: 0.5409, Train Accuracy: 0.8125, Val Loss: 0.7822, Val Accuracy: 0.7436
Epoch 8/25, Train Loss: 0.4587, Train Accuracy: 0.8373, Val Loss: 0.7298, Val Accuracy: 0.7617
Epoch 9/25, Train Loss: 0.3528, Train Accuracy: 0.8754, Val Loss: 0.8103, Val Accuracy: 0.7661
Epoch 10/25, Train Loss: 0.2628, Train Accuracy: 0.9083, Val Loss: 0.8418, Val Accuracy: 0.7706
Epoch 11/25, Train Loss: 0.1279, Train Accuracy: 0.9579, Val Loss: 0.6590, Val Accuracy: 0.8230
Epoch 12/25, Train Loss: 0.0718, Train Accuracy: 0.9774, Val Loss: 0.6910, Val Accuracy: 0.8278
Epoch 13/25, Train Loss: 0.0531, Train Accuracy: 0.9847, Val Loss: 0.7323, Val Accuracy: 0.8273
Epoch 14/25, Train Loss: 0.0419, Train Accuracy: 0.9884, Val Loss: 0.7669, Val Accuracy: 0.8268
Epoch 15/25, Train Loss: 0.0331, Train Accuracy: 0.9912, Val Loss: 0.7921, Val Accuracy: 0.8284
Epoch 16/25, Train Loss: 0.0271, Train Accuracy: 0.9932, Val Loss: 0.8228, Val Accuracy: 0.8277
Epoch 17/25, Train Loss: 0.0203, Train Accuracy: 0.9952, Val Loss: 0.8567, Val Accuracy: 0.8288
Epoch 18/25, Train Loss: 0.0167, Train Accuracy: 0.9961, Val Loss: 0.8765, Val Accuracy: 0.8290
Epoch 19/25, Train Loss: 0.0141, Train Accuracy: 0.9972, Val Loss: 0.9043, Val Accuracy: 0.8290
Epoch 20/25, Train Loss: 0.0123, Train Accuracy: 0.9974, Val Loss: 0.9229, Val Accuracy: 0.8286
Epoch 21/25, Train Loss: 0.0097, Train Accuracy: 0.9983, Val Loss: 0.9230, Val Accuracy: 0.8291
Epoch 22/25, Train Loss: 0.0096, Train Accuracy: 0.9983, Val Loss: 0.9264, Val Accuracy: 0.8295
Epoch 23/25, Train Loss: 0.0096, Train Accuracy: 0.9988, Val Loss: 0.9319, Val Accuracy: 0.8297
Epoch 24/25, Train Loss: 0.0099, Train Accuracy: 0.9984, Val Loss: 0.9367, Val Accuracy: 0.8292
Epoch 25/25, Train Loss: 0.0094, Train Accuracy: 0.9983, Val Loss: 0.9340, Val Accuracy: 0.8297
Test Accuracy: 0.8221
```