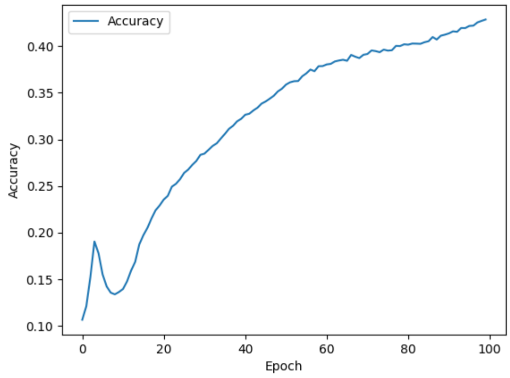
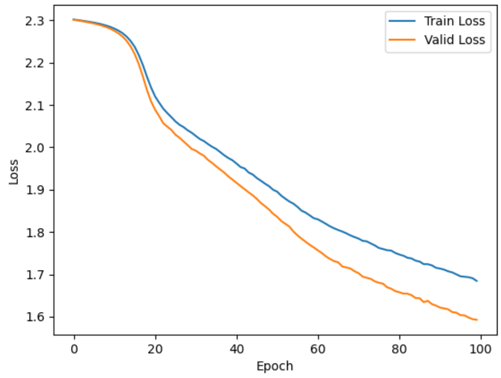
视听信息系统导论编程 1

高艺轩 毕嘉仪

第 1 题

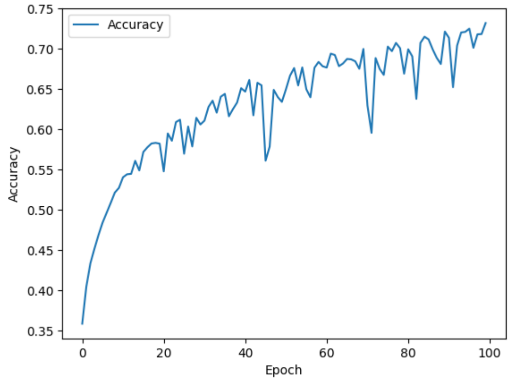
在 SGD 优化器下，输出的结果如下：

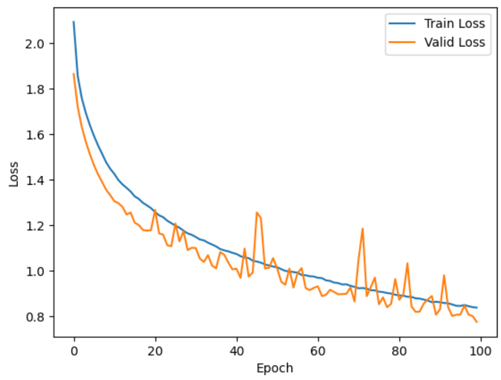




第 2 题

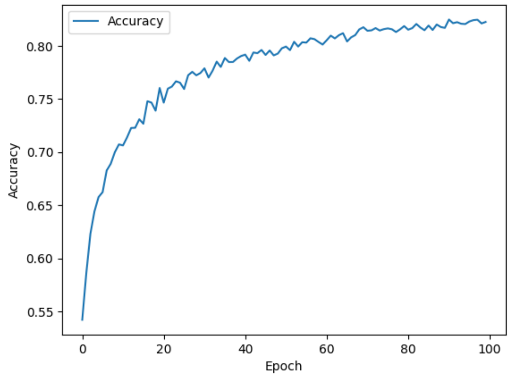
在 SGD 优化器下，使用 ReLU 激活函数与 Batch Normalization，输出的结果如下：

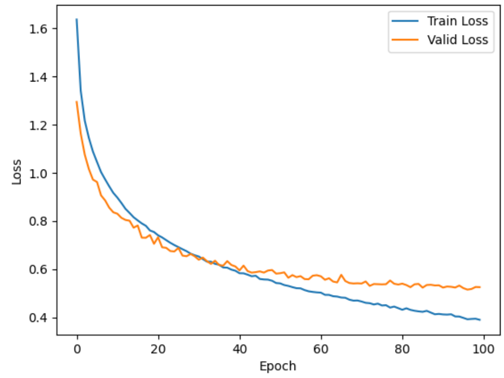




第 3 题

在 AdamW 优化器下，使用 ReLU 激活函数与 Batch Normalization，输出的结果如下：

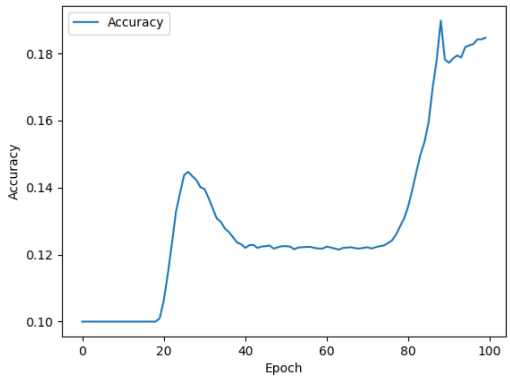


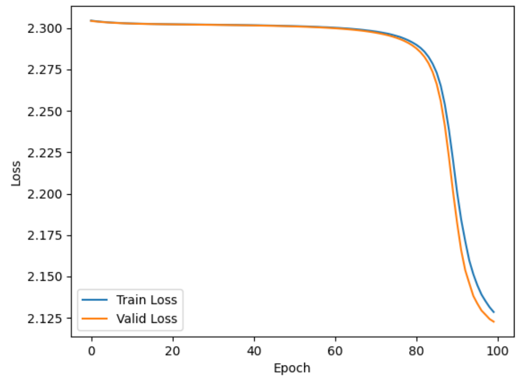


第 4 题

4.1

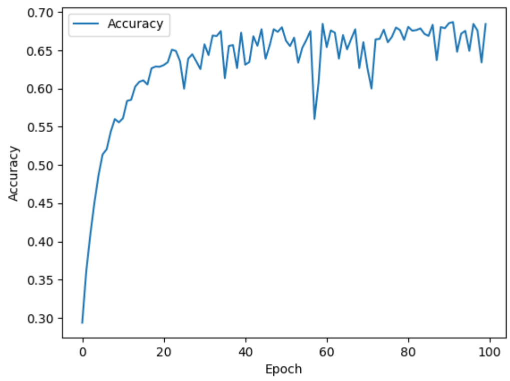
仍旧使用 tanh 作为激活函数，SGD 优化器：

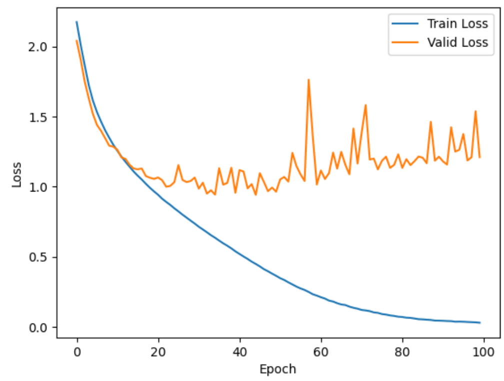




4.2

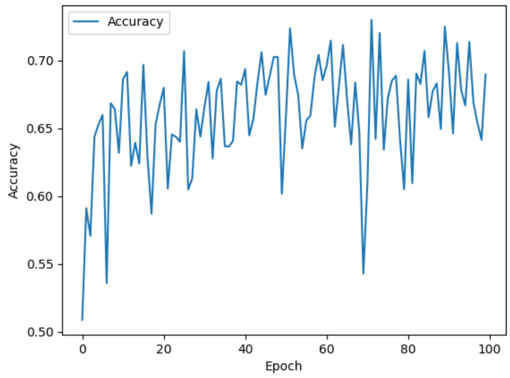
将激活函数更改为 ReLU 函数，并使用 BatchNorm，使用 SGD：

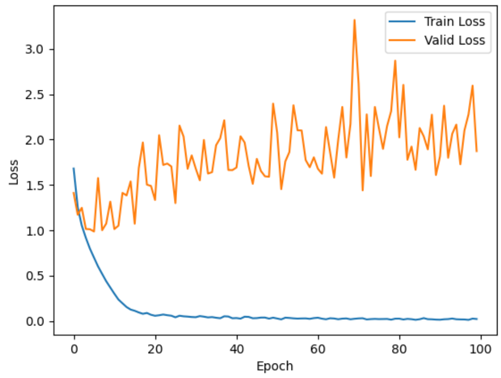




4.3

在 AdamW 优化器下，使用 ReLU 激活函数与 Batch Normalization，输出的结果如下：





第 5 题

1. 请根据结果分析 ReLU 和 tanh 激活函数的表现.

   由于实验 1 与实验 2 同时更改了激活函数与是否增加 Batch Normalization，我们无法直接从实验结果中通过对比得出 ReLU 与 tanh 到底哪个表现更好。不过从理论上进行分析，ReLU 激活函数在当前任务中的表现应当会优于 tanh 激活函数。这是由于 ReLU 激活函数不存在梯度消失问题，能够使网络快速收敛，无论网络深浅；而 tanh 函数容易陷入饱和区、导致梯度消失，使得网络收敛速度大幅降低，特别是在深层次网络中这一现象更加明显；因此，在 CNN 架构中，ReLU 激活函数表现应该会优于 tanh。

2. 请根据结果分析 BatchNorm 的作用.

   由于实验 1 与实验 2 同时更改了激活函数与是否增加 Batch Normalization，我们无法直接从实验结果中通过对比看出 BatchNorm 的作用。不过从理论上进行分析，BatchNorm 可以使网络中每一层的数据分布均值为 0、方差为 1，这样神经元输出值不会太大，加强了网络稳定性，同时防止出现梯度爆炸（或梯度消失），使网络能够较快收敛；此外，Batch Normalization 可以使相同 batch 中的所有样本相互关联，加强了泛化性，避免过拟合。

3. 请根据结果分析更换优化器的效果.

   从实验 2、3 的结果对比中可以看出，使用 SGD 优化器会导致测试准确率与训练、测试损失出现明显震荡，而更换了 AdamW 优化器以后准确率曲线与损失曲线都更加平滑。SGD 优化器容易出现振荡是由于它会导致梯度值相差较大的方向上优化步长差异也大，致使梯度值大的方向上产生振荡；而 Adam 优化器可以根据不同参数梯度值自适应地调整各个方向的优化步长大小，以使各个方向都能同时得到有效且合理的优化。

4. 请根据你的结果分析模型是否出现了过拟合，如有，请在图像中指出在哪里出现了过拟合。如无，请给出你判断的原因.

通过训练/测试损失曲线以及测试准确率曲线可知，实验 4 的第 2, 3 部分均出现了过拟合现象，分别发生在 epoch 35 附近以及 epoch 5 附近。判断依据：训练损失持续降低，而测试损失不再降低甚至开始升高且出现巨幅振荡，测试准确率也不再增加。

程序源代码

```python
import os
import torch
import torchvision
import numpy as np
from tqdm import tqdm
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision.transforms as transforms
import matplotlib.pyplot as plt

from PIL import Image
from typing import *

device = 'cuda' if torch.cuda.is_available() else 'cpu'

#################### 数据集初始化与读入 ####################
train_transform = transforms.Compose([
   transforms.RandomHorizontalFlip(),
   transforms.RandomCrop(32, padding=4),
   transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
   transforms.ToTensor()
])
class CustomCIFAR10(torchvision.datasets.CIFAR10):
   def __init__(
      self,
      root,
      train,
      transform = None,
      target_transform = None,
      download = False,
   ):
      super().__init__(root, train, transform, target_transform, download)
      new_data = []
      new_target = []
      for i in tqdm(range(len(self.data))):
```

```python
            img, target = self.data[i], self.targets[i]
            # print(self.data.shape)
            # print(img.shape)
            img = Image.fromarray(img)

            if self.transform is not None:
                img = self.transform(img)

            if self.target_transform is not None:
                target = self.target_transform(target)

            new_data.append(img)
            new_target.append(target)

        self.data = torch.Tensor(np.stack(new_data, axis=0)).to(device)
        self.targets = torch.Tensor(new_target).to(torch.int64).to(device)

    def __getitem__(self, index):
        return self.data[index], self.targets[index]

# train_dset = torchvision.datasets.CIFAR10(root='./CIFAR10',train=True,download=False,transform=train_transform)
# test_dset = torchvision.datasets.CIFAR10(root='./CIFAR10',train=False,download=False,transform=transforms.ToTensor())
train_dset = CustomCIFAR10(root='./CIFAR10',train=True,download=False,transform=train_transform)
test_dset = CustomCIFAR10(root='./CIFAR10',train=False,download=False,transform=transforms.ToTensor())
train_loader = torch.utils.data.DataLoader(train_dset, batch_size=128, shuffle=True, num_workers=0)
test_loader = torch.utils.data.DataLoader(test_dset, batch_size=128, shuffle=False, num_workers=0)
#######################################################


#################### 构建模型 ####################
class Net(nn.Module):
    def __init__(self,act):
        super(Net, self).__init__()
        # 卷积层 (32x32x3的图像)
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        # 卷积层(16x16x16)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
```

```
        # 卷积层(8x8×32)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        # 最大池化层
        self.pool = nn.MaxPool2d(2, 2)
        # linear layer (64 * 4 * 4 -> 500)
        self.fc1 = nn.Linear(64 * 4 * 4, 500)
        # linear layer (500 -> 10)
        self.fc2 = nn.Linear(500, 10)
        if act == 'relu':
            self.act = F.relu
        elif act == 'tanh':
            self.act = F.tanh
        elif act == 'sigmoid':
            self.act = F.sigmoid

    def forward(self, x):
        # add sequence of convolutional and max pooling layers
        x = self.pool(self.act(self.conv1(x)))
        x = self.pool(self.act(self.conv2(x)))
        x = self.pool(self.act(self.conv3(x)))
        # flatten image input
        x = x.view(-1, 64 * 4 * 4)

        x = self.act(self.fc1(x))

        x = self.fc2(x)
        return x
#######################################################

#################### 模型加入batchnorm ####################
class BnNet(nn.Module):
    def __init__(self, act):
        super(BnNet, self).__init__()
        # 卷积层 (32x32×3的图像)
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(16)

        # 卷积层(16x16×16)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(32)

        # 卷积层(8x8×32)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
```

```python
        self.bn3 = nn.BatchNorm2d(64)

        # 最大池化层
        self.pool = nn.MaxPool2d(2, 2)

        # linear layer (64 * 4 * 4 -> 500)
        self.fc1 = nn.Linear(64 * 4 * 4, 500)
        self.bn4 = nn.BatchNorm1d(500)

        # linear layer (500 -> 10)
        self.fc2 = nn.Linear(500, 10)

        if act == 'relu':
            self.act = F.relu
        elif act == 'tanh':
            self.act = F.tanh
        elif act == 'sigmoid':
            self.act = F.sigmoid

    def forward(self, x):
        # add sequence of convolutional and max pooling layers
        x = self.pool(self.act(self.bn1(self.conv1(x))))
        x = self.pool(self.act(self.bn2(self.conv2(x))))
        x = self.pool(self.act(self.bn3(self.conv3(x))))

        # flatten image input
        x = x.view(-1, 64 * 4 * 4)

        x = self.act(self.bn4(self.fc1(x)))
        x = self.fc2(x)
        return x
#################### 构建模型 ####################



class DeepNet(nn.Module):
    def __init__(self):
        super(DeepNet, self).__init__()
        #################### 代码填空：请在此填补模型定义代码 ####################
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv4 = nn.Conv2d(64, 128, 3, padding=1)
```

```
        self.conv5 = nn.Conv2d(128, 256, 3, padding=1)
        self.conv6 = nn.Conv2d(256, 512, 1, padding=0)
        self.fc1 = nn.Linear(512, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)
        self.act = F.tanh
        self.pool = nn.MaxPool2d(2, 2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        ####################################################################

    def forward(self, x):
        # convolutional layers
        #################### 代码填空：请在此填补前向计算代码 ####################
        x = self.act(self.conv1(x))
        x = self.pool(self.act(self.conv2(x)))
        x = self.act(self.conv3(x))
        x = self.pool(self.act(self.conv4(x)))
        x = self.act(self.conv5(x))
        x = self.avgpool(self.act(self.conv6(x)))
        x = x.view(-1, 512)

        x = self.act(self.fc1(x))
        x = self.act(self.fc2(x))
        x = self.act(self.fc3(x))
        ####################################################################
        return x

class BnDeepNet(nn.Module):
    def __init__(self,act):
        super(BnDeepNet, self).__init__()
        #################### 代码填空：请在此填补模型定义代码 ####################
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(32)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.bn3 = nn.BatchNorm2d(64)
        self.conv4 = nn.Conv2d(64, 128, 3, padding=1)
        self.bn4 = nn.BatchNorm2d(128)
        self.conv5 = nn.Conv2d(128, 256, 3, padding=1)
        self.bn5 = nn.BatchNorm2d(256)
        self.conv6 = nn.Conv2d(256, 512, 1, padding=0)
        self.bn6 = nn.BatchNorm2d(512)
```

```python
        self.fc1 = nn.Linear(512, 256)
        self.bnfc1 = nn.BatchNorm1d(256)
        self.fc2 = nn.Linear(256, 128)
        self.bnfc2 = nn.BatchNorm1d(128)
        self.fc3 = nn.Linear(128, 10)
        if act == 'relu':
            self.act = F.relu
        elif act == 'tanh':
            self.act = F.tanh
        elif act == 'sigmoid':
            self.act = F.sigmoid
        self.pool = nn.MaxPool2d(2, 2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        ###################################################################

    def forward(self, x):
        # convolutional layers
        ################### 代码填空：请在此填补前向计算代码 ###################
        x = self.act(self.bn1(self.conv1(x)))
        x = self.pool(self.act(self.bn2(self.conv2(x))))
        x = self.act(self.bn3(self.conv3(x)))
        x = self.pool(self.act(self.bn4(self.conv4(x))))
        x = self.act(self.bn5(self.conv5(x)))
        x = self.avgpool(self.act(self.bn6(self.conv6(x))))
        x = x.view(-1, 512)

        x = self.act(self.bnfc1(self.fc1(x)))
        x = self.act(self.bnfc2(self.fc2(x)))
        x = self.act(self.fc3(x))
        return x
        ##################################################################


################### 训练前准备 ###################


model = BnDeepNet('relu').to(device)
criterion = nn.CrossEntropyLoss().to(device)

optimizer_type = "AdamW" #或者换成AdamW
if optimizer_type == "SGD":
    optimizer = optim.SGD(model.parameters(), lr=0.001)
elif optimizer_type == "AdamW":
```

```python
    ########## 代码填空：请在此填补Adam优化器计算代码, lr=0.0001 ###########
    optimizer = optim.AdamW(model.parameters(), lr=0.0001)
    ##################################################################
    pass

n_epochs = 100
train_losses = []
valid_losses = []
accuracies = []
################################################


################### 训练+验证 ###################
for epoch in range(n_epochs):
    train_loss = 0.0
    valid_loss = 0.0
    model.train()
    for idx,(img,label) in tqdm(enumerate(train_loader)):
        # img, label = img.to(device), label.to(device)
        optimizer.zero_grad()
        output = model(img)
        loss = criterion(output,label)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * img.shape[0]

    model.eval()
    correct = 0
    total = 0
    for idx,(img,label) in tqdm(enumerate(test_loader)):
        # img, label = img.to(device), label.to(device)
        output = model(img)
        loss = criterion(output, label)
        valid_loss += loss.item() * img.shape[0]
        _, predicted = torch.max(output.data, 1)
        total += label.size(0)
        correct += (predicted == label).sum().item()

    train_loss = train_loss / len(train_dset)
    valid_loss = valid_loss / len(test_dset)

    train_losses.append(train_loss)
    valid_losses.append(valid_loss)
```

```
    accuracy = correct / total
    accuracies.append(accuracy)

    print(f"Epoch:{epoch}, Acc:{correct/total}, Train Loss:{train_loss}, Test Loss:{valid_loss}")
################################################

################### 曲线绘制 ###################
print("MAX ACC: ",np.max(accuracies))
plt.plot(range(n_epochs), train_losses, label='Train Loss')
plt.plot(range(n_epochs), valid_losses, label='Valid Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.savefig("Loss.png")
plt.clf()
# 绘制验证集准确率随epoch的变化曲线
plt.plot(range(n_epochs), accuracies, label='Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig("Acc.png")
################################################
```