# 实验2：使用卷积神经网络进行图像识别报告

# 实验背景

本实验旨在通过使用卷积神经网络（CNN）实现一个简单的图像分类任务。实验所用的图片皆为4x4像素的图像，要根据外围黑色像素点和内圈黑色像素点的数量将其分类为0类或1类。实验使用Pytorch作为深度学习框架，并采用MNIST数据集进行模型的训练和测试。

卷积神经网络因其在处理图像和语音信号等二维数据方面的优越性能，已成为现代深度学习中不可或缺的工具之一。通过局部感知域的概念和权重共享的机制，CNN能够有效捕捉图片中的空间特征，从而在图像分类任务中取得优异的表现。本实验通过实现一个简单的CNN，并调整模型参数，实现卷积神经网络在小规模数据集上的应用。

# 实验方案设计

首先预处理实验数据，通过csv文件进行数据的加载，使用PIL库将图像转换为二值图像并进行标准化处理。随后将数据集中的样本按照标签降序排序，使得标签为1的样本排在前面，计算出数据集中标签为1的样本数量并乘以给定的比例scale，截取数据集的一部分，确保数据不那么偏斜。最后将截取后的数据集进行随机打乱，以增加训练时的随机性。

然后实验构建了一个卷积神经网络，设计了一个简单的CNN模型，包含一个卷积层和一个全连接层。卷积层使用了1x1的卷积核，输入和输出通道均为1，之后通过ReLU激活函数并连接到一个线性层进行二分类。

实验中选择的超参数：学习率0.1，批处理大小64，训练周期数为10。

训练时，采用交叉熵损失函数（CrossEntropyLoss）作为损失函数，并使用随机梯度下降法（SGD）作为优化器。在每个训练周期后，计算模型在测试集上的准确率以评估模型的性能，并进行相应的调优。

# 实验过程

进行实验时为trainAndTest函数添加了matplotlib绘图，代码如下：

def trainAndTest(net, trainIter, testIter, loss, numEpochs, batchSize, optimizer):

    epochs = []

    train\_losses = []

    train\_accuracies = []

    test\_accuracies = []

    for epoch in range(numEpochs):

        trainLossSum, trainAccSum, n = 0.0, 0.0, 0

        for X, y in trainIter:

            yHat = net(X)

            l = loss(yHat, y).sum()

            optimizer.zero\_grad()

            l.backward()

            optimizer.step()

            # 计算训练准确度和loss

            trainLossSum += l.item()

            trainAccSum += (yHat.argmax(dim=1) == y).sum().item()

            n += y.shape[0]

        # 评估测试准确度

        testAcc = evaluateAccuracy(testIter, net)

        # 记录数据

        epochs.append(epoch + 1)

        train\_losses.append(trainLossSum / n)

        train\_accuracies.append(trainAccSum / n)

        test\_accuracies.append(testAcc)

        # 打印结果

        print(

            "epoch {:d}, loss {:.4f}, train acc {:.3f}, test acc {:.3f}".format(

                epoch + 1, trainLossSum / n, trainAccSum / n, testAcc

            )

        )

    # 绘图

    fig, ax1 = plt.subplots()

    color = "tab:red"

    ax1.set\_xlabel("Epoch")

    ax1.set\_ylabel("Loss", color=color)

    ax1.plot(epochs, train\_losses, color=color, label="Loss", linestyle="--")

    ax1.tick\_params(axis="y", labelcolor=color)

    ax2 = ax1.twinx()

    color = "tab:blue"

    ax2.set\_ylabel("Accuracy", color=color)

    ax2.plot(epochs, train\_accuracies, color="tab:blue", label="Train Accuracy")

    ax2.plot(epochs, test\_accuracies, color="tab:green", label="Test Accuracy")

    ax2.tick\_params(axis="y", labelcolor=color)

    fig.tight\_layout()

    plt.title("Training Loss and Accuracy")

    ax1.legend()

    ax2.legend()

    plt.show()

进行运行测试，输出如下：

7390

10000

epoch 1, loss 0.0056, train acc 0.900, test acc 0.963

epoch 2, loss 0.0037, train acc 0.908, test acc 0.970

epoch 3, loss 0.0027, train acc 0.932, test acc 0.966

epoch 4, loss 0.0024, train acc 0.937, test acc 0.964

epoch 5, loss 0.0023, train acc 0.943, test acc 0.965

epoch 6, loss 0.0022, train acc 0.946, test acc 0.968

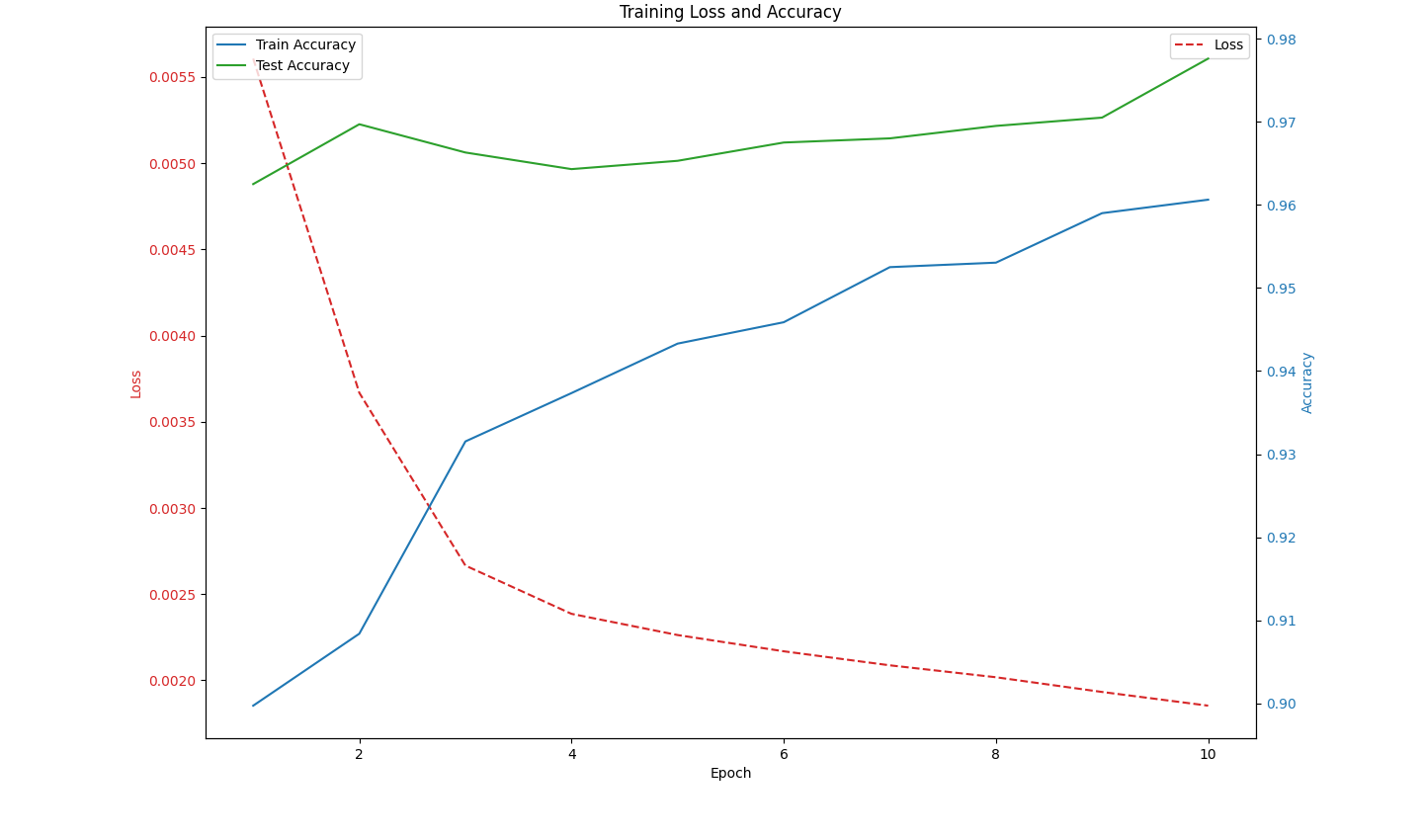
epoch 7, loss 0.0021, train acc 0.953, test acc 0.968

epoch 8, loss 0.0020, train acc 0.953, test acc 0.970

epoch 9, loss 0.0019, train acc 0.959, test acc 0.971

epoch 10, loss 0.0019, train acc 0.961, test acc 0.978

可见，loss维持相当低的水平并逐步下降，同时train acc上升，test acc维持较高水平。函数绘图输出如下，直观的体现了训练过程中的数据变化：



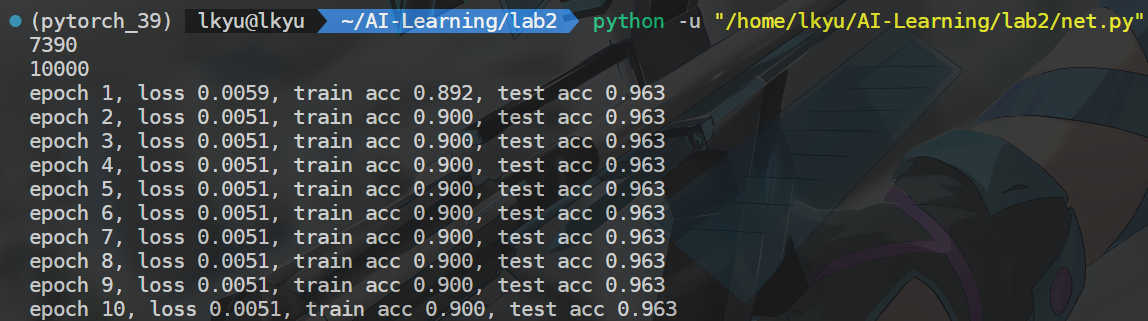
# 性能测试

由上一节中的数据可以得出，测试集上，准确率始终维持在96% 以上，证明了模型的准确性十分优良。

# 实验中遇到的问题

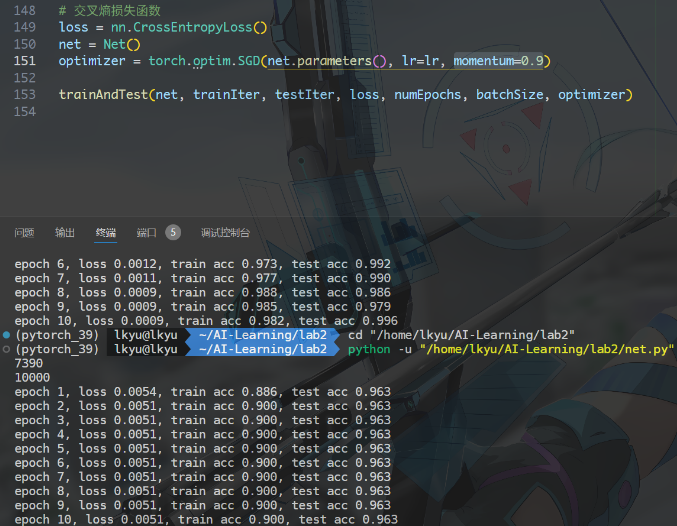
遇到了疑似局部最优的情况，具体表现：

1、模型会迅速达到固定的loss和train acc，不再更新



2、模型迅速达到1.00的完美值，不再变化。

尝试为优化器加入动量optimizer = torch.optim.SGD(net.parameters(),lr = lr, momentum=0.9)，但不起作用：



# 实验习题

1. 写出两种以上不同的卷积padding策略，并说明在实验中选取的是哪一种

Valid Padding: 不添加额外的边缘像素，输出的图像尺寸会小于输入图像。

Same Padding: 通过添加边缘像素，使得输出的图像尺寸与输入图像相同。

在本实验中，我们使用的是Valid Padding策略。这是因为实验中图像的尺寸较小（4x4），在这种情况下，使用Valid Padding可以减少模型的复杂度，避免因边缘填充而导致的冗余信息影响模型训练。