**《深度学习》实验报告一**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **大三计算机科学与技术（弘深）计算机拔尖01班** | | **姓名** | | **朱云哲** | **学号** | **20224700** |
| **实验题目** | **CNN分类任务对比实验** | | | | | | | |
| **实验时间** | **2025.4.3** | | **实验地点** | | **DS1402** | | | |
| **学年学期** | **2024-2025（2）** | | **实验性质** | | **□验证性 □设计性 ■综合性** | | | |
| 一、实验目的  1．熟悉pytorch/mindspore实验环境。  2. 理解和掌握卷积神经网络的基本构成。  3. 理解不同优化器（SGD，Momentum，Adam）的原理及在卷积神经网络中的作用。  4. 应用正则化技术（Dropout、L2正则化、数据增强）缓解过拟合。  5．实践超参数调优（学习率、Dropout比率、批量大小）  6．学会分析模型性能（准确率、损失曲线、混淆矩阵） | | | | | | | | |
| 二、实验项目内容  1.构建基于CIFAR-10数据集（https://www.cs.toronto.edu/~kriz/cifar.html）的分类卷积神经网络模型  2.通过优化器和正则化技术提升验证集准确率。  3.正则化技术应用（Dropout、L2正则化、数据增强）  4．超参数调优，探索自适应学习率策略 | | | | | | | | |
| **三、实验步骤**  **1.数据预处理**  本次实验为使用CNN进行图像分类任务，使用的数据集为CIFAR-10数据集。为了使模型训练获得更好的效果，数据预处理分为了三个部分：**（1）数据转换操作；（2）加载数据集；（3）创建数据加载器。**  **1.1数据转换操作**  由于我是直接从官网下载的数据集，得到的是几个bin文件，但是该文件无法直接用于模型训练，需要先转换成模型能够接受的标准化张量。此外还需要判断是否需要使用数据增强，选择不同的数据转换操作，扩充数据集，高效利用数据。  **·数据转换**   |  | | --- | | *def* data\_preprocess(data\_dir, batch\_size, data\_augmentation=*False*):  transform = transforms.Compose([  transforms.ToTensor(),  transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  ]) |   这部分使用transforms.ToTensor将图像从NumPy数组/PIL图像转换为PyTorch张量，并使用transforms.Normalize对张量进行标准化处理，使用均值0.5和标准差0.5对RGB图像进行处理。  **·数据增强**   |  | | --- | | *if* data\_augmentation:  train\_transform = transforms.Compose([  transforms.RandomHorizontalFlip(),  transforms.RandomCrop(32, padding=4),  transforms.ToTensor(),  transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  ]) *else*:  train\_transform = transform |   该部分进行数据增强操作。主要进行两种方法：（1）水平翻转；（2）随机剪裁。其中在裁剪前，使用了四像素填充，再进行随机剪裁。数据增强后进行标准化处理整理张量。  **1.2加载数据集**  本部分主要使用自定义类CIFAR10Dataset加载CIFAR-10数据集，并从.bin文件中读取图像与标签，转换为PyTorch张量。   |  | | --- | | train\_dataset = CIFAR10Dataset(data\_dir=data\_dir, train=True, transform=train\_transform)  test\_dataset = CIFAR10Dataset(data\_dir=data\_dir, train=False, transform=transform) |   这部分进行数据的加载。由于官网下载得到的数据集和在线下载得到的数据集存在格式上的差异，因此需要自己编写自定义的数据集类完成数据集的转化操作。代码自定义了数据集CIFAR10Dataset类，该方法主要由四部分组成： （1）\_\_init\_\_()：初始化函数，设置数据目录，是否为训练集以及数据转换操作；  （2）load\_data()：数据加载函数，使用pickle读取bin文件中的数据，并将图像数据从NumPy数组转化为HWC格式。  （3）len()：返回数据集大小。  （4）\_\_getitem\_\_()：将NumPy数组转换为PIL图像，方便进行后续的图像增强操作。  **1.3创建数据加载器**  使用DataLoader函数创建训练集和测试集的数据加载器。   |  | | --- | | train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)  test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False) |   这里设置了shuffle=True表示在每个epoch开始时随机打乱数据。  **2.构建CNN模型**  实验构建的CNN模型由五个部分组成：   |  | | --- | | *class* CIFAR10CNN(nn.Module):  *def \_\_init\_\_*(*self*, dropout\_rate=0.5, l2\_reg=0.0):  *super*(CIFAR10CNN, *self*).*\_\_init\_\_*()  *self*.conv1 = nn.Conv2d(3, 32, kernel\_size=3, padding=1)  *self*.conv2 = nn.Conv2d(32, 64, kernel\_size=3, padding=1)  *self*.conv3 = nn.Conv2d(64, 128, kernel\_size=3, padding=1)  *self*.pool = nn.MaxPool2d(2, 2)  *self*.dropout = nn.Dropout(dropout\_rate)  *self*.fc1 = nn.Linear(128 \* 4 \* 4, 512)  *self*.fc2 = nn.Linear(512, 10)  *self*.l2\_reg = l2\_reg |   **（1）卷积层**：用于提取图像的局部特征。  **（2）池化层**：用于降低特征图的空间维度，减少计算量，同时保留重要特征。  **（3）全连接层**：用于将提取的特征进行分类。  **（4）Dropout操作**：用于防止过拟合。  **（5）激活函数（ReLU）**：引入非线性，使模型能够学习复杂的特征。  以下是对各个部分的详细说明：  **2.1卷积层**   |  | | --- | | *self*.conv1 = nn.Conv2d(3, 32, kernel\_size=3, padding=1) *self*.conv2 = nn.Conv2d(32, 64, kernel\_size=3, padding=1) *self*.conv3 = nn.Conv2d(64, 128, kernel\_size=3, padding=1) |   卷积层由多个卷积核组成，每个卷积核负责提取一种特定的特征，卷积层定义如下：nn.Conv2d(in\_channels, out\_channels, kernel\_size, padding)。  首先输入的图像为RGB三通道图像，因此我们第一层卷积层的输入通道数为3，我们规定卷积核大小为3×3，并填充像素数设置为1，使得卷积操作后特征图大小不变，使得模型更好利用边缘信息并减少信息丢失，输出为32通道。以此类推叠加三层卷积层。  **2.2池化层**   |  | | --- | | *self*.pool = nn.MaxPool2d(2, 2) |   这里使用了一个最大池化，降低特征图的空间维度，减少计算量，这里设置了2×2大小的池化窗口，步长为2，使得特征图长款减半。  **2.3全连接层**   |  | | --- | | *self*.fc1 = nn.Linear(128 \* 4 \* 4, 512) *self*.fc2 = nn.Linear(512, 10) |   全连接层用于特征分类，这里使用了两个全连接层，可以进行非线性变换，并逐步压缩特征信息，减少计算量。  **2.4 Dropout层**   |  | | --- | | *self*.dropout = nn.Dropout(dropout\_rate) |   这是一个正则化技术，用于防止过拟合。通过在训练中随机丢弃一部分神经元的输出减少模型获得特征信息，防止模型学习过多的特征导致过拟合。这里设置的dropout\_rate为0.5。  **2.5 ReLu层**  用于引入非线性，使得模型可以学习复杂的特征。  **2.6模型整体结构**   |  | | --- | | *def* forward(*self*, x):  x = *self*.pool(torch.relu(*self*.conv1(x)))  x = *self*.pool(torch.relu(*self*.conv2(x)))  x = *self*.pool(torch.relu(*self*.conv3(x)))  x = x.view(-1, 128 \* 4 \* 4)  x = *self*.dropout(torch.relu(*self*.fc1(x)))  x = *self*.fc2(x)  *return* x |   在前向传播函数中定义了模型的整体结构。图像通过三个卷积层与三个池化层，每个卷积层后都接入了一个池化层，提取图像的局部特征同时并降低特征图的空间维度。随后进行展平操作，将特征图展平为一维向量用于全连接层输入。  实验中使用了两层全连接层，第一个全连接层后接入ReLu激活函数，随后通过Dropout层进行正则化，防止过拟合，将得到的数据通过第二个全连接层，输出10个类别的预测结果  **3.优化器的对比**  实验主要使用三种优化器：（1）Adam；（2）SGD；（3）Momentum。可以在模型训练时自主选择：   |  | | --- | | *# 选择优化器 if* optimizer\_name == 'Adam':  optimizer = optim.Adam(model.parameters(), lr=learning\_rate, weight\_decay=l2\_reg) *elif* optimizer\_name == 'SGD':  optimizer = optim.SGD(model.parameters(), lr=learning\_rate) *elif* optimizer\_name == 'Momentum':  momentum = 0.9  optimizer = optim.SGD(model.parameters(), lr=learning\_rate, momentum=momentum) *else*:  *raise ValueError*("Unsupported optimizer. Choose from 'Adam', 'SGD', or 'Momentum'.") |   以下为三种优化器的详细介绍。  **3.1 SGD（随机梯度下降优化器）**  SGD是一种简单的优化算法，每次迭代只使用一个样本来计算梯度并更新模型参数，更新规则为：  其中，是当前参数，是学习率，是损失函数的梯度。  SGD在卷积网络中的应用主要由三种：  **·提高计算效率**：SGD每次只处理一个样本，计算量小，适合大规模数据集。  **·跳出局部最优**：由于梯度估计存在噪声，而SGD的梯度估计是基于单个样本的，因此模型可以借助这种噪声选择新的梯度更新方向，增加模型跳出局部最优的机会。  **·学习率敏感**：SGD对学习率的选择非常敏感，过大的学习率可能导致震荡，过小的学习率会导致收敛速度缓慢。基于这一点，需要对学习率这个超参数进行调优。  **3.2 Momentum（动量优化器）**  Momentum优化器在SGD的基础上引入了一个动量项，用于加速收敛并减少震荡，其更新规则如下：  其中，是动量项，是动量系数，是学习率。  Momentum在卷积网络的应用如下：  **·减少震荡**：动量项可以帮助模型在梯度变化较大的维度上保持稳定，减少参数更新的幅度，减少震荡。  **·加速收敛**：通过累计历史梯度信息在动量项中，使得模型在梯度方向一致时可以更快前进，加速收敛。  **3.3 Adam（自适应矩估计优化器）**  Adam优化器结合了Momentum和RMSProp优化器的优点，通过计算梯度的一阶矩估计（均值）和二阶矩估计（方差）来自适应调整学习率。其更新规则为：  其中，和分别是一阶矩和二阶矩的指数衰减率，是一个很小的常数，用于避免除以零。  Adam优化器具有以下作用：  **·高效收敛**：Adam优化器结合了动量和自适应学习率的优点，能够快速收敛并减少训练时间。  **·稳定训练**：在训练过程中，Adam 优化器表现出较少的震荡和抖动，使得训练过程更加稳定。  **·适应性强**：Adam 优化器能够自适应地调整每个参数的学习率，适用于不同的模型和数据集。  三种优化器的对比实验结果将在第四部分给出。  **4.正则化技术的应用**  本实验主要使用三种正则化技术用于防止模型过拟合：（1）Dropout；（2）L2正则化；（3）数据增强。可以在模型训练时自主选择：   |  | | --- | | *def* main(optimizer\_name='Adam', use\_dropout=*True*, use\_l2\_reg=*True*, use\_data\_augmentation=*True*):  *# 超参数设置* batch\_size = 64  learning\_rate = 0.001  num\_epochs = 10  dropout\_rate = 0.5 *if* use\_dropout *else* 0.0  l2\_reg = 0.001 *if* use\_l2\_reg *else* 0.0  data\_augmentation = use\_data\_augmentation |   这里use\_dropout用于确认是否需要开启dropout，开启则设置丢弃率为0.5，use\_l2\_reg用于确认是否需要使用L2正则化项，开启则设置正则化项为0.001，use\_data\_augmentation则用于确认是否需要进行数据增强。  **4.1 Dropout**  Dropout是一种通过在训练过程中随机“关闭”部分神经元来防止过拟合的技术。其核心思想是通过破坏神经元间的共适应关系，增强模型的泛化能力。具体来说，Dropout在每次训练迭代中随机丢弃一部分神经元，使得这些神经元在本次迭代中不参与前向传播和反向传播。这种随机性可以减少模型对特定训练样本的依赖，从而提高模型的泛化能力。Dropout有一个超参数rate，用于调整丢弃神经元的概率。  在CNN中，Dropout层通常被添加在全连接层之前。实验的CIFAR-10卷积神经网络中，Dropout被添加在第一个全连接层和第二个全连接层之间。这样可以减少全连接层中神经元之间的共适应性，从而防止过拟合。  **4.2 L2正则化**  L2正则化通过在损失函数中添加一个正则化项来限制模型参数的大小。这个正则化项是参数的平方和，其作用是使模型参数的值不会过大。通过这种方式，L2正则化可以减少模型的复杂度，从而降低过拟合的风险。  **4.3数据增强**  数据增强是一种通过生成更多训练样本来减少过拟合的技术。其核心思想是通过对训练数据进行随机变换（如旋转、翻转、裁剪等）来增加数据的多样性。这样可以减少模型对特定训练样本的依赖，从而提高模型的泛化能力。  在前面的数据预处理函数中，可以通过设置use\_data\_augmentation选择是否开启数据增强。  由于整个代码较长，本部分只展示核心部分代码，其他部分例如自定义类，训练函数，测试函数，绘图函数，main函数等部分不再赘述。 | | | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程、实验总结与体会  实验主要任务有三个： （1）对比不同优化器的模型训练效果；  （2）对比不同正则化技术的模型训练效果；  （3）对比不同超参数设置的模型训练效果；  为了保证不同任务的测试公平，我在进行某一组任务时将保持其他参数相同，用于比较实验结果。  **4.1 不同优化器对模型训练效果的影响**  条件设置如下：  （1）超参数设置：batch\_size=64、learning\_rate=0.001、epochs=20；  （2）正则化技术：开启数据增强、L2正则项=0.001、dropout\_rate=0.5；  **·使用SGD**  得到准确率，损失曲线，混淆矩阵如下：    图1. 使用SGD的CIFAR-10测试集准确率    图2. 使用SGD的CIFAR-10测试集损失曲线与准确率曲线    图3. 使用SGD的CIFAR-10测试集的混淆矩阵  **模型分析：**  **（1）模型性能**：从损失曲线和准确率曲线可以发现模型在训练过程中有所学习，但性能提升有限。且测试准确率较低，说明模型的泛化性较差。  **（2）欠拟合**：训练准确率远高于测试准确率，且训练准确率本身也不高，模型可能有欠拟合，应该为epoch过少导致。  **（3）混淆矩阵**：从对角线可以得知模型对于类别0，6的分类较好，但是可以发现模型在识别类别6容易与类别2，3，4，5混淆，识别类别0容易与类别8混淆，且在类别2，3，4上识别准确率较低，说明模型在这些类别上发生了欠拟合。  **·使用Momentum**  得到准确率，损失曲线，混淆矩阵如下：    图4. 使用Momentum的CIFAR-10测试集准确率    图5. 使用Momentum的CIFAR-10测试集损失曲线与准确率曲线    图6. 使用Momentum的CIFAR-10测试集的混淆矩阵  **模型分析：**  **（1）模型性能**：从损失曲线和准确率曲线可以发现，模型在训练过程中有所学习，性能提升较为明显。而且测试准确率较高，表明模型的泛化性良好。  **（2）无明显的过拟合或欠拟合**：训练准确率和测试准确率之间的差距不大，这表明模型在训练集和测试集上的表现较为一致，没有明显的过拟合或欠拟合现象。  **（3）混淆矩阵**：从对角线可得知模型对类别0-9的识别数量均良好，无明显的准确率下降，且各类别的易混淆类别识别数量较低，说明模型的性能良好。  **·使用Adam**  得到准确率，损失曲线，混淆矩阵如下：    图7. 使用Adam的CIFAR-10测试集准确率    图8. 使用Adam的CIFAR-10测试集损失曲线与准确率曲线    图9. 使用Adam的CIFAR-10测试集的混淆矩阵  **模型分析：**  **（1）模型性能**：从损失曲线和准确率曲线可以发现，模型在训练过程中的学习性能提升明显，收敛快速。而且测试准确率较高，表明模型的泛化性良好。  **（2）无明显的过拟合或欠拟合**：训练准确率和测试准确率之间的差距不大，这表明模型在训练集和测试集上的表现较为一致，没有明显的过拟合或欠拟合现象。  **（3）混淆矩阵**：从对角线可得知模型对类别0-9的识别数量均较优秀，无明显的准确率下降，且各类别的易混淆类别识别数量较低，说明模型的性能良好。  **·对比**  **（1）准确率影响**  由准确率曲线可以发现，在三类优化器中，Adam的准确率提升最高，Momentum其次，SGD最低，且基本无明显性能提升。  **（2）收敛速度**  由损失曲线可以得知，三类优化器中Adam的收敛速度最快，且loss值最小，说明模型的学习能力增加明显，而SGD的收敛速度最低，且loss值较高，说明模型的学习能力没有明显进步。  **（3）过拟合与欠拟合**  由混淆矩阵可以得知，Adam对类别的识别准确率，泛化性均优异，且没有明显的类别混淆，Momentum的识别准确率较高，但是有部分类别混淆,SGD的识别准确率较低，且存在部分类别识别不准确，部分类别易混淆情况。  **（4）总结**  Adam优化器综合了Momentum与RMSProp的优点，使用动量减少震荡，加快收敛速率，效果最为优秀；Momentum使用单一动量项，加快了收敛，但是存在训练不足混淆情况；SGD最为简单，但是简单的下降策略容易引发模型欠拟合，识别准确率较低。  **4.2 不同正则化策略对模型训练效果的影响**  条件设置如下：  （1）超参数设置：batch\_size=64、learning\_rate=0.001、epochs=20；  （2）优化器：Adam；  **·对照组（无设置）**  得到准确率，损失曲线，混淆矩阵如下：    图10. 无策略的CIFAR-10测试集的准确率    图11. 无策略的CIFAR-10测试集的损失曲线与准确率曲线    图12. 无策略的CIFAR-10测试集的混淆矩阵  **·使用Dropout（设置Dropout\_rate=0.5）**  得到准确率，损失曲线，混淆矩阵如下：    图13. 使用Dropout策略的CIFAR-10测试集的准确率    图14. 使用Dropout策略的CIFAR-10测试集的损失曲线与准确率曲线    图15. 使用Dropout策略的CIFAR-10测试集的混淆矩阵  **模型分析：**  **（1）模型性能**：从损失曲线和准确率曲线可以发现，模型在训练过程中的学习性能提升明显，收敛快速。  **（2）明显的过拟合**：训练准确率和测试准确率之间的差距较大，这表明模型在训练集和测试集上的表现不一致，有明显的过拟合现象，尽管相比无策略设置下有0.01的准确率提升。  **（3）混淆矩阵**：从对角线可得知模型对类别0-9的识别数量均良好，无明显的准确率下降，且各类别的易混淆类别识别数量均值较低，类别5，3容易相互混淆，且相比无策略设置减少了类别9相对类别0的混淆，除此之外模型的识别准确性均良好，说明Dropout策略防止了模型过拟合但是效果不明显。  **·使用L2正则项（设置l2\_reg=0.001）**  得到准确率，损失曲线，混淆矩阵如下：    图13. 使用L2正则化策略的CIFAR-10测试集的准确率    图14. 使用L2正则化策略的CIFAR-10测试集的损失曲线与准确率曲线    图15. 使用L2正则化策略的CIFAR-10测试集的混淆矩阵  **模型分析：**  **（1）模型性能**：从损失曲线和准确率曲线可以发现，模型在训练过程中的学习性能提升明显，收敛快速。  **（2）明显的过拟合**：训练准确率和测试准确率之间的差距较大，这表明模型在训练集和测试集上的表现不一致，有明显的过拟合现象，而且相比无设置的模型，准确率更低。  **（3）混淆矩阵**：从对角线可得知模型对类别0-9的识别数量均良好，无明显的准确率下降，但是各类别的易混淆类别识别数量有所上升，类别1，2，3，4，5，6均有数量较多的容易混淆类别，说明l2正则化策略没有很好的防止模型过拟合，需要搭配其他策略一起使用。  **·使用数据增强（设置剪裁与翻转方式）**  得到准确率，损失曲线，混淆矩阵如下：    图13. 使用数据增强策略的CIFAR-10测试集的准确率    图14. 使用L2正则化策略的数据增强测试集的损失曲线与准确率曲线    图15. 使用数据增强策略的CIFAR-10测试集的混淆矩阵  **模型分析：**  **（1）模型性能**：从损失曲线和准确率曲线可以发现，模型在训练过程中的学习性能提升明显，收敛快速。  **（2）明显的过拟合**：训练准确率和测试准确率之间的差距不大，尽管模型的训练集准确率相比前几种策略校低，但是在测试集上准确率明显高于其他几种策略，这表明模型在训练集和测试集上的表现基本一致，无明显的过拟合现象，  **（3）混淆矩阵**：从对角线可得知模型对类别0-9的识别数量均良好，无明显的准确率下降，除类别3，5存在部分混淆样例外，其他类别识别效果均良好。  **·对比**  **（1）准确率影响**  由准确率曲线可以发现，在三类正则化策略中，数据增强的过拟合防止效果最好，准确率最高，Dropout其次，L2正则化项最差，且低于无策略效果。  **（2）过拟合与欠拟合**  由混淆矩阵和训练/测试集准确率对比可以得知，数据增强的过拟合防止效果最好，这是由于数据增强增加了图像的类别，使得模型可以在不同位置学习同一类别图像的不同特征，Dropout的过拟合防止效果较弱不明显，L2正则化项的效果最差。  **（3）总结**  尽管三种正则化策略均有防止过拟合效果，但是单一使用某种正则化策略不能很好的提升模型整体性能，相比Adam优化器+所有正则化策略，模型的表现有明显进步，说明正则化策略还需要搭配使用才能发挥其最大的防止过拟合性能。  **4.3 不同超参数设置对模型训练效果的影响**  条件设置如下：  （1）优化器设置：Adam；  （2）正则化技术：开启数据增强、L2正则项=0.001；  （3）超参数设置：epoch=20；  **·学习率**   |  |  | | --- | --- | | Learning\_rate | Accuracy | | 0.01 | 0.3242 | | 0.0075 | 0.4608 | | 0.005 | 0.5526 | | 0.0025 | 0.7265 | | 0.001 | 0.7845 |   表1.不同学习率的模型测试集准确率（batch\_szie=64，dropout\_rate=0.5）  **·batch\_size**   |  |  | | --- | --- | | batch\_size | Accuracy | | 32 | 0.7569 | | 64 | 0.7708 | | 128 | 0.7762 | | 256 | 0.7654 | | 512 | 0.7592 |   表2.不同batch\_size的模型测试集准确率（learning\_rate=0.001，dropout\_rate=0.5）  **·Dropout\_rate**   |  |  | | --- | --- | | Dropout\_rate | Accuracy | | 0.2 | 0.7852 | | 0.4 | 0.7884 | | 0.5 | 0.7854 | | 0.6 | 0.7710 | | 0.8 | 0.7728 |   表3.不同Dropout\_rate的模型测试集准确率（learning\_rate=0.001，batch\_size=64）  **·总结**  **（1）学习率（learning\_rate）**  表1可以得知，随着学习率的减小，模型的步长减小，模型能够更好的观察梯度下降方向并进行合理的前进，学习率过大容易导致震荡严重影响模型性能，但是学习率过小会影响模型收敛速率。  **（2）批量大小（batch\_size）**  表2可以得知，batch\_size的大小对模型表现也有影响，在batch\_size≤128时，模型表现逐步上升，但是大于128后，模型准确率却在下降，这说明batch\_size对模型表现的影响并不是线性的，需要逐步调参获得最优参数。这是由于较小的批量可以增加模型的泛化能力，有助于减少模型过拟合，过大batch\_size会导致模型更关注整体的特征分布，减少泛化能力，但是大的批量可以更好利用GPU并行能力，增加训练效率，因此batch\_size的选择需要进行大量调优才能得到最适合的参数值。  **（3）Dropout\_rate**  表3可以得知，dropout\_rate的影响与batch\_size相似，对模型的表现为非线性，过低的dropout\_rate会减少神经元剪枝带来的信息删减效果，过大的dropout\_rate会导致关键信息的丢失，因此dropout\_rate的选择也和batch\_size类似，需要综合考虑。  **4.4 问题总结与实验体会**  本实验任务为使用CNN对CIFAR-10数据集进行分类，需要使用不同的优化器与正则化技术改进模型性能与表现。在实验中遇到的主要问题在于如何让一个简单的三层卷积网络有效对图像进行分类，我设计了三层卷积+三层池化+两层全连接的模型结构，这种结构能够在保证模型简单的同时兼顾效率。随后就是搭配优化器与正则化技术。实验要求选择三种优化器与三种正则化技术，并作出对比，我比较困扰如何进行不同设置间的对比。因此我选择了最简单的对比方式，即控制变量，保持其他设置相同的情况下只改变一种设置，并选择其他设置中最优的设置进行训练。这一好处是可以减少其他设置带来的负面影响。  总的来说，本次实验手动实现了一个CNN网络用于图像分类，动手学习了如何通过引入策略提升模型性能，同时也学习到了如何调参，如何搭配不同策略。 | | | | | | | | |

实验报告填写说明：

1、第一、二部分由老师提供；

2、第三部分根据实验步骤填写主要源程序，源程序要符合程序编写风格（缩进、注释等）；

3、第四部分主要填写程序结果（截图）、解决问题的方法、总结和体会等；

4、报告规范：包含报告页眉、报告的排版、内容是否填写，命名是否规范等。

5、源程序和实验报告命名：学号姓名序号.扩展名，例如学号20181234的张三同学，**他的第一次实验命名为：20181234张三1.py和20181234张三1.docx**

6、向蓝墨云提交两个文件，一个实验报告，一个程序文件（代码打包）