天津大学

模式识别与深度学习课程

作业 3、卷积神经网络实验报告

学　　院　智能与计算学部

专　　业　计算机与科学技术

学　　号　3019244140

姓　　名　郭思齐

1．实验目标

实验 3——深度学习卷积神经网络，实验目标是让我们能够熟练掌握卷积神经网络 ResNet-18 的模型结构以及残差网络原理。通过补全数据集载入预处理、ResNet-18 模型结构设计搭建、模型的训练测试、预测结果可视化等部分的编程，实现对 CIFAR-10 数据集的分类任务。最后探究实验中各个超参数对实验结果的影响。

实验的具体要求如下：

1. 基于初始模版，补全实验代码，包含 CIRAR-10 数据预处理，ResNet-18 网络搭建，优化器、损失函数设置，模型训练、测试评估等部分，完成 CIFAR-10 数据集的分类，预测标签。
2. 调节不同的参数如 batch size、学习率、迭代次数等，统计不同参数对实验结果的影响，将不同参数对应的识别准确率通过表格记录，并分析原因，最后比较 CPU 训练和经过 CUDA 加速的 GPU 训练差异和显存占用情况。

2. 实验分析

2.1 实验环境

基于远程服务器，创建 Anaconda 虚拟环境。

配置	版本
Python	Python 3.9.12
PyTorch	1.9.0+cu111
torchvision	0.10.0+cu111
CUDA	11.1
CUDNN	8005

2.2 算法实现和模型的搭建

应用 ResNet-18 残差卷积神经网络模型对 CIFAR-10 图片进行分类训练、预测。

2.2.1 ResNet-18 的原理：

首先，ResNet 作为残差网络，由子网络堆叠构成。而所谓的残差学习，是为了解决随着网络加深而产生的梯度消失，添加了 Skip connection，在深度网络的中间层额外加入浅层的 Input，提供计算梯度的复合路径。

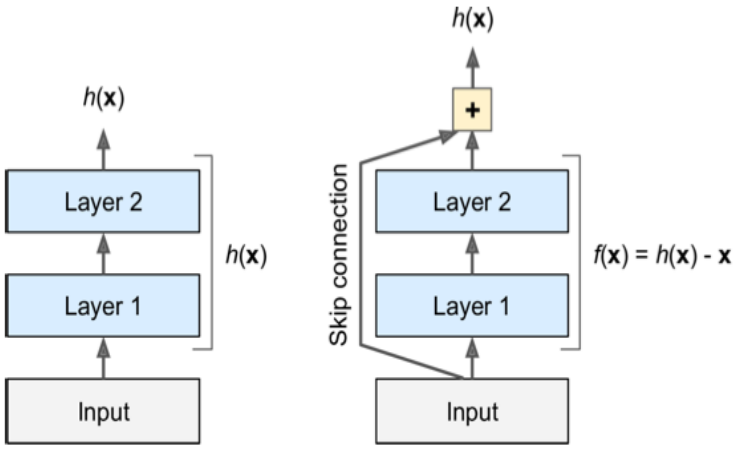


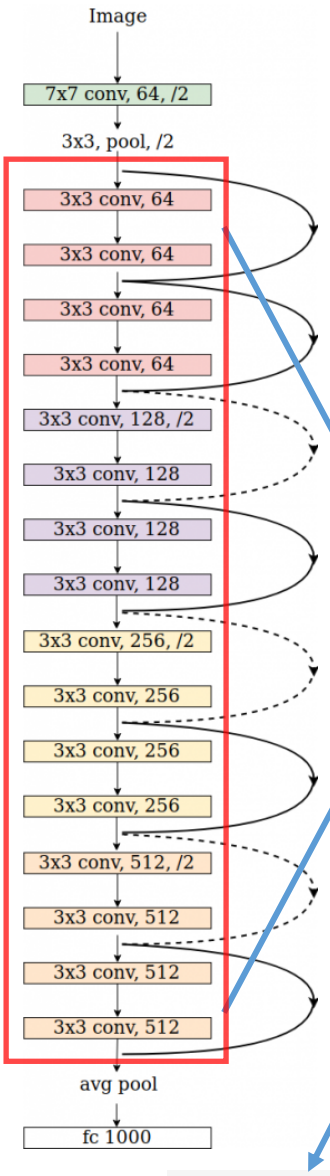
Figure 1 残差学习对比原先的神经网络（右侧示意残差网络的 Skip connection）

ResNet 设定每次 Skip connection 会越过两层卷积层，构成一个残差单元。由残差单元、池化层、全连接层等构成 ResNet 整体网络，ResNet 的层数变化比较灵活，并且能够构建出非常深的卷积神经网络，有 ResNet34、ResNet50，ResNet101，作者优化后甚至可以有 ResNet1001。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2 ResNet 结构 出自 Deep Residual Learning for Image Recognition

在实验三我们使用 ResNet-18 作为模型，补充模型搭建部分的代码。ResNet-18 中所谓的 18 包含了 17 层卷积层和最后一层全连接层。不含有其他池化层、BN 以及非线性激活函数。




```
self.conv1 = nn.Sequential(
    # 第一层卷积和bn
    nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False),
    nn.BatchNorm2d(64),
    nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
)
# 4 blocks
self.blk1 = ResBlk(64,  64,  [1, 1])
self.blk1_= ResBlk(64,  64,  [1, 1])
self.blk2 = ResBlk(64,  128, [2, 1])
self.blk2_= ResBlk(128, 128, [1, 1])
self.blk3 = ResBlk(128, 256, [2, 1])
self.blk3_= ResBlk(256, 256, [1, 1])
self.blk4 = ResBlk(256, 512, [2, 1])
self.blk4_= ResBlk(512, 512, [1, 1])
# 全连接
self.outlayer = nn.Linear(512, 10) # num_classes = 10
```

```
#### TODO ####
self.conv1 = nn.Conv2d(ch_in, ch_out, kernel_size=3, stride=stride[0], padding=1, bias=False)
self.bn1   = nn.BatchNorm2d(ch_out)
self.conv2 = nn.Conv2d(ch_out, ch_out, kernel_size=3, stride=stride[1], padding=1, bias=False)
self.bn2   = nn.BatchNorm2d(ch_out)
if ch_out == ch_in:
    self.extra = nn.Sequential()
else:
    self.extra = nn.Sequential(
        # 1*1卷积和bn
        nn.Conv2d(ch_in, ch_out, kernel_size=1, stride=2, padding=0, bias=False),
        nn.BatchNorm2d(ch_out)
    )        self.conv1 = nn.Sequential(
    # 第一层卷积和bn
    nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False),
    nn.BatchNorm2d(64),
    nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
)
```

Figure 3 搭建 ResNet-18 网络代码补全及结构说明

之后在搭建的模型中补充 forward 函数中网络向前传播和池化的部分。

```
def forward(self, x):
    x = F.relu(self.conv1(x))
    x = self.blk1(x)
    x = self.blk1_(x)
    x = self.blk2(x)
    x = self.blk2_(x)
    x = self.blk3(x)
    x = self.blk3_(x)
    x = self.blk4(x)
    x = self.blk4_(x)
    x = F.adaptive_avg_pool2d(x, [1, 1])
    x = x.view(x.size(0), -1)
    x = self.outlayer(x)
    return x
```

Figure 4　forward 神经网络向前传递补全

最后的全连接层对应 CIFAR-10 数据的十个类，多分类选取最大可能的标签作为预测标签。

2.2.2 数据集的载入和数据预处理：

数据集通过 Torch 的 dataset 来下载训练集和测试集，通过 DataLoader 封装，内部以 batch 的形式存储，保证其迭代性。DataLoader 的数据在取出时，有图像的 Tensor 格式数据和标签数据。

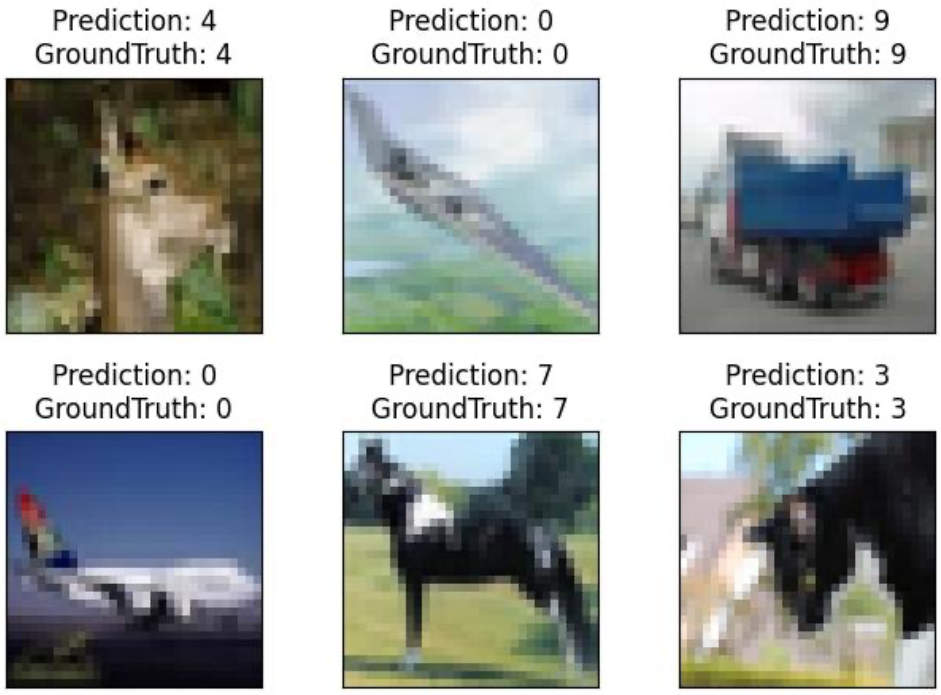
在数据预处理方面，运用到了 transforms 对图像数据进行缩放（图像本身 32*32）、标准化、Tensor 的转换、旋转翻转等操作；也进行了仅转换数据类型而不加多余操作的对比实验。

2.2.3 数据训练测试结果的可视化：

数据训练、测试及可视化部分基于实验提供的 MNIST 代码和 CIFAR-10 训练模版修改。

当 batch size 设为 32，学习率 lr 设定为 0.01，迭代次数 EPOCH 设定为 15，不添加其他数据预处理时数据增强的操作，并使用 Adam 优化器和交叉熵损失函数时，可以获得测试集上约 75%的准确率，并且可以输出一些测试结果。

根据 loss 图像和准确度图像，推测 ResNet-18 训练 32*32 的 CIFAR10 数据集存在过拟合现象。



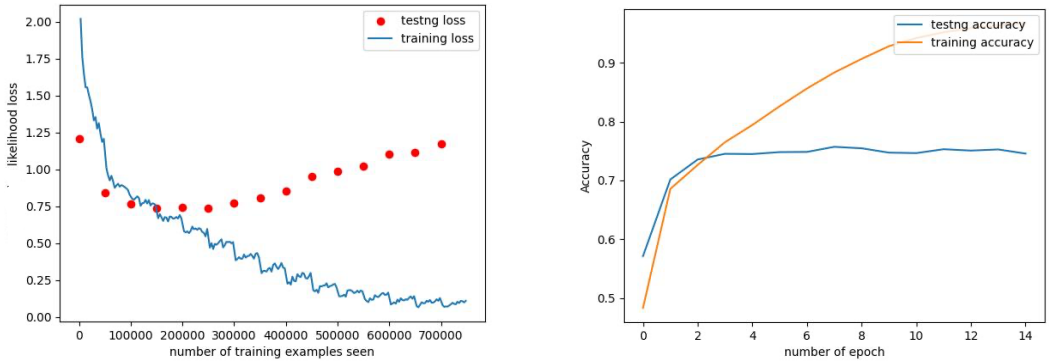


Figure 5 基本参数配置下的模型训练后测试可视化结果

过拟合的个人分析：训练的损失持续下降，准确度持续上升，但是测试评估的损失几个迭代次数就已经下降到最低值并且之后开始缓慢上升，测试的准确度也不再继续提升。在这种基础的参数配置下，测试的准确度一般，不过已经可以可视化出不错的预测结果了。

本次实验过拟合的原因可能在于，CIFAR-10 数据集像素 32x32，可能对于 ResNet-18 来说，特征不够丰富，或者说模型相对复杂，在比较小的 Epoch 内就可以达到相当不错的解，之后学习到的都是无用的细化特征。

2.3 参数调节和结果分析

2.3.1 数据增强或数据增幅对于结果的影响：

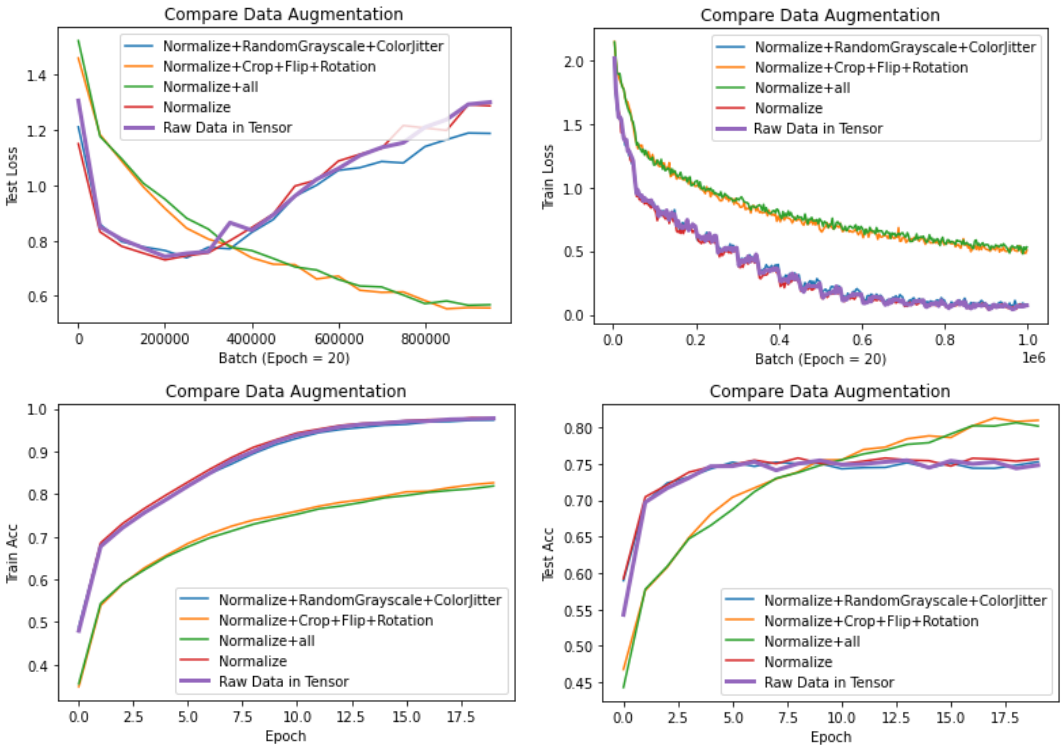
数据增强，数据增广有很多种形式，包括剪裁、缩放、翻转、旋转、灰度化、标准化、白化处理、加噪声等等方式。

根据初始实验，在短周期内模型就会过拟合，因此之后的实验大多以 20 作为 Epoch；在实验三中，我尝试了以下的几种方式组合，比较了结果：

Table 1 数据增强对 test 预测结果的影响

Normalize	Random Grayscale	ColorJitter	Random ResizedCrop	Random Rotation	Random HorizontalFlip	Best Acc
X	X	X	X	X	X	75.49%
√	X	X	X	X	X	75.81%
√	√	√	X	X	X	75.28%
√	X	X	√	√	√	**81.09%**
√	√	√	√	√	√	80.66%

可视化结果：



实验中发现在我的模型上归一化是对识别的准确度的影响不明显，ColorJitter 和灰度化的选择在我的模型上甚至对预测产生了负面的影响，但

是添加剪裁和旋转翻转这些数据增强操作之后，丰富了数据，缓解了过拟合，使得准确率结果大幅提升。

但是因为剪裁和旋转翻转这些数据增强，使得可学习特征增多，使得模型的学习时间被延长。而对于不加剪裁旋转这些操作的数据集，train 阶段过早的学习特征使得 test 无法获得更高识别准确率。

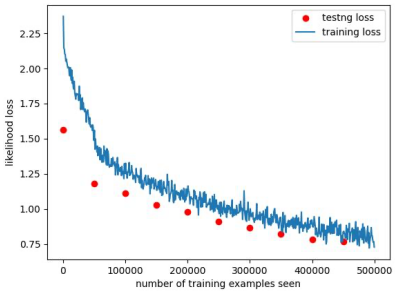
2.3.2 Batch size 对与结果的影响：

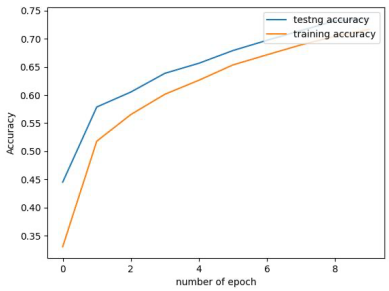
Batch size 作为超参数，就是一次要投入到模型中进行训练数据的多少，介于 1 和训练样本的总数之间。不仅影响训练速度，也影响模型精度。

从理论上讲 Batch 过小时，训练抖动大，训练收敛速度慢，而且容易欠拟合。随着 Batch 增大，训练速度会提升。但是当 Batch 增长到过大时，提速的效果就不再明显，而且虽然大 Batch 正负样本更均衡可以有效更新参数，但是训练的精度也会降低，模型的泛化能力下降，并且可能产生 Out of Memory 的问题。

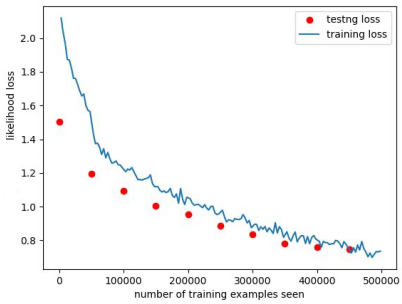
该部分对比实验 **Epoch 设为 10**（Acc 未达到最优），学习率设为 0.001，数据增强 Normalize + Crop + Flip + Rotation，Adam 作为优化器，交叉熵作为损失函数。

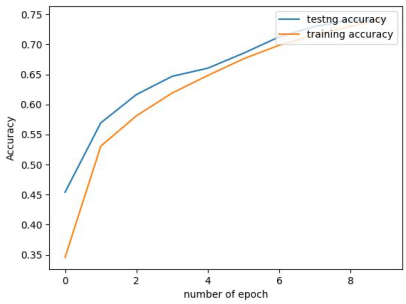
1. Batch size = 8:



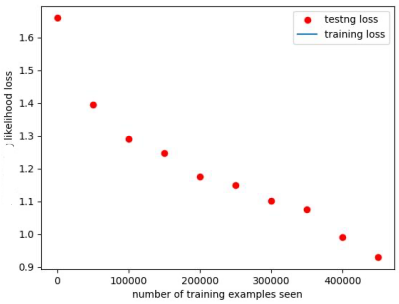


2. Batch size = 32:





3. Batch size = 512:



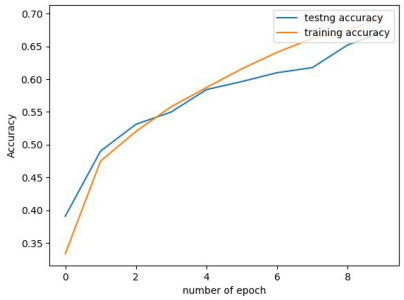
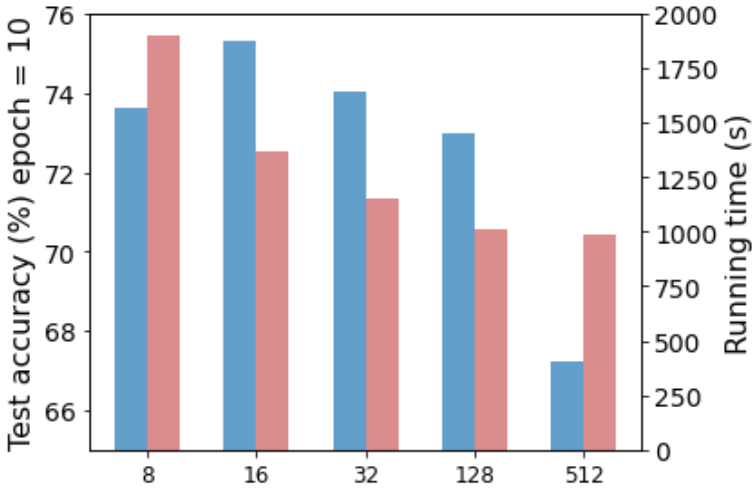


Table 2 Batch Size 下同样 epoch 下准确率和运行时间比较

Batch size	8	16	32	128	512	1024
Test accuracy **epoch=10**	73.60%	75.30%	74.03%	73.00%	67.21%	Out of Memory
Running time	1898.68	1371.9	1150.08	1014.14	983.801	Out of Memory

对比不同 batch size 下 **Epoch10** 以内的最高的准确率训练变化，和不同的程序运行用时：（Acc 仍未达到最优）



根据本次实验得到的结果可以侧面验证前面关于 Batch 的理论，因此在考虑 Batch size 的时候需要注意训练速度和训练精度的平衡，要想达到更优的结果，也需要注意调节 Batch size 的同时调节 Epoch、学习率等其他参数。

2.3.3 Epoch 对结果的影响：

Epoch 作为训练的迭代次数，随着 Epoch 的增加，如果没有发生过拟合的话，训练和测试的的 loss 都会降低到比较小的范围之后稳定下来，而测试的准确率则会上升到比较高的数据比如 90%左右，训练的准确率则是接近于 1，之后稳定。

如果发生过拟合，训练的 loss 会不断地下降，但是准确率会在更小的 Epoch 就接近 1，从而停止学习有意义的特征，而测试的 loss 会随着 Epoch 的增大先下降，之后停止下降或上升，测试的准确率会在比较下的 Epoch 就稳定下来，但是相比没有过拟合的训练准确率偏低，比如 74%左右。

2.3.4 选择不同优化器对结果的影响：

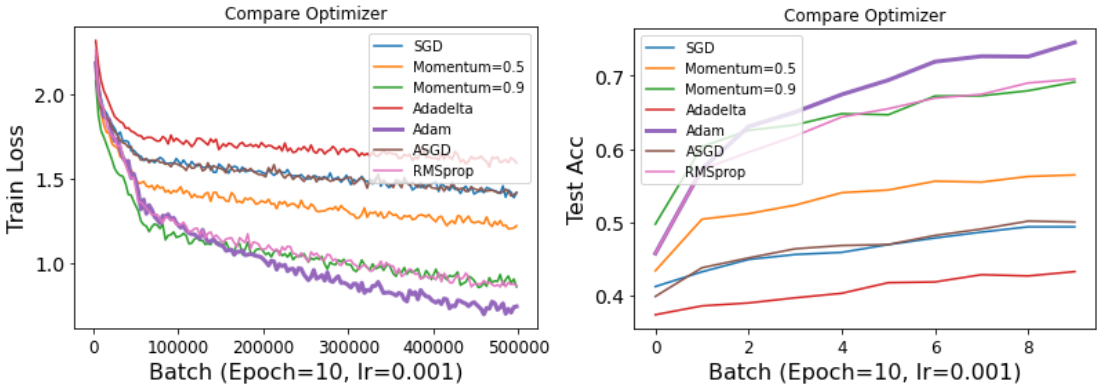
优化器和损失函数针对基于 ResNet-18 的 CIFAR-10 识别有很多选择。

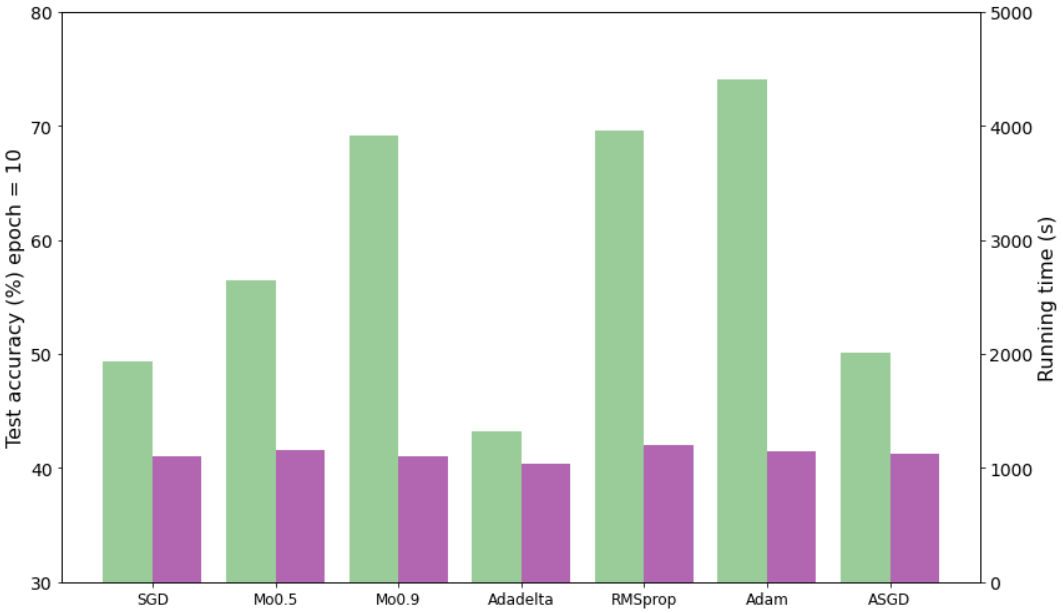
优化器的比较：（默认使用了交叉熵损失函数）

Table 3 优化器选择 在 epoch=10 之内的结果

Optimizer	SGD	Momentum 0.5	Momentum 0.9	Adadelta	RMSprop	Adam	ASGD
Best Test Acc epoch=10	49.37%	56.46%	69.15%	43.25%	69.54%	74.03%	50.15%
Running time	1104.016	1161.144239	1107.836407	1038.229	1197.636	1150.083	1127.653

选择不同的优化器对于训练的快慢有比较大的影响，一般的优化器需要更多的 Epoch 达到最优解，但是每个 epoch 运行时间的几乎不受优化器的影响。





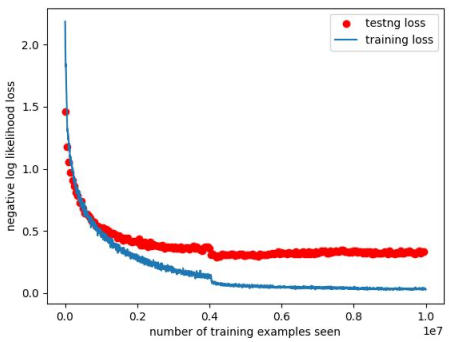
从优化效果来看，一定周期内达到更高的准确率会更高效，那么 Adam 优于 RMSprop 优于较大 Momentum，优于小一些的 Momentum，更优于 SGD，更优于 Adadelta 更优于 AGSD。

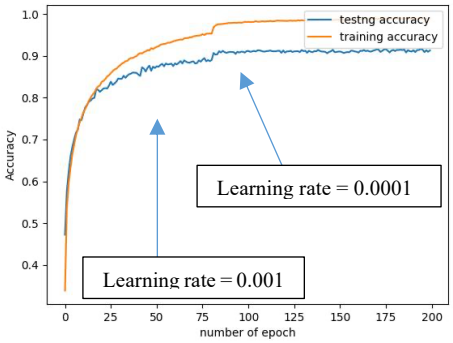
2.3.5 设置不同的学习率对结果的影响：

学习率对结果的影响我是通过在非常大的 Epoch 下，更新 learning rate 可视化了结果。

```
if epoch % 80 == 0:
    for p in optimizer.param_groups:
        p['lr'] *= 0.1
```

虽然没有具体数据，但是可以在图中看出第一次学习率的变化。





学习率大的时候，学习速率快，但是学习粗糙，精度不够高。学习率小则可以提高精度，但是会消耗更多的 Epoch。但是当训练的精度达到一定程度，

因为特征数量、Batch 等等限制，即使再减小学习率可能也不能够提高精度了。比如在第 160 Epoch 的时候 lr 降到了 0.00001 也不会对结果有明显的影响。

除此以外，学习率低的时候可能陷入局部最优解。

在这里可以看到在 batch size = 32，Adam 优化器和交叉熵损失函数的设定下，经过 200 和 Epoch 最终达到最优识别率 **90%左右**。

2.3.6 比较 CPU 和经过 CUDA 加速的 GPU 训练差异：

由于在 CPU 上运行对于时间的消耗过大，因此我只训练了一次 CPU 上的 ResNet-18, 训练的 Epoch 是 10, 对比经过 CUDA 加速的训练结果, test 的准确率相差不大，但是时间可以相差数十倍。

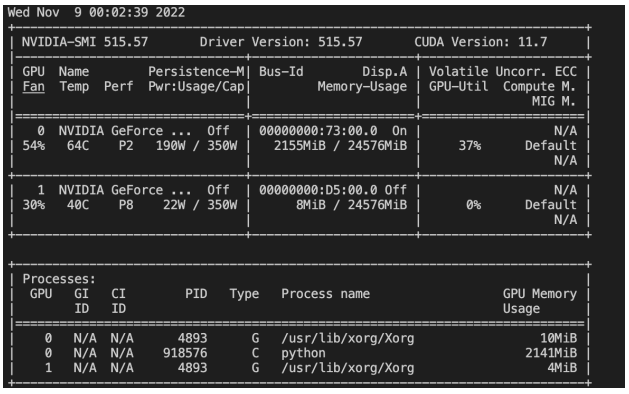
表中的计时不含有统计训练和测试的准确率、损失的部分。

Table 4 CPU GPU 比较运行时间

Device	CPU	GPU
Run Time (s)	34933.53377	791.5949
Best Test Acc	76%	75.81%

根据加速比公式 $\boldsymbol{Sp = T1 / Tp}$ 计算出整个程序的在实验三的 GPU GA102 [GeForce RTX 3090] 的加速下，达到了加速比 44.1631

最后，GPU 显存占用情况：



3. 总结

实验三作为课程的最后一个实验，为我们提供了自己搭建卷积神经网络模型的机会，并且在设计卷积核、步长、通道数之后，对传入的图像数据怎样在网络中计算、传递有了深入的理解。

除此以外，通过调用不同的优化器，调节不同的 batch size，学习率、迭代次数等参数，统计出各个情况下的实验结果，进行可视化分析，了解到在深度学习中如何训练模型才能更有效率地获得更优的结果。