机器学习实验报告

实验名称：CNN 图片分类任务

学生姓名：蒋雨初

学生学号：58121102

完成日期：2023/5/16

任务描述

实现卷积神经网络 CNN，并使用 CIFAR-10 数据集进行图片分类任务。

CIFAR-10 是计算机视觉领域中的一个重要的数据[1]。 原始数据集分为训练集和测试集，其中训练集包含 50000 张、测试集包含 300000 张图像。 在测试集中，10000 张图像将被用于评估，而剩下的 290000 张图像将不会被进行评估，包含它们只是为了防止手动标记测试集并提交标记结果。这些图片共涵盖 10 个类别：飞机、汽车、鸟类、猫、鹿、狗、青蛙、马、船和卡车，高度和宽度均为 32 像素并有三个颜色通道（RGB）。图 1 的左上角显示了数据集中飞机、汽车和鸟类的一些图像。本实验使用部分的 CIFAR-10 数据集，其中训练集共包含 5000 张 png 格式的图像，每个类别包含 500 张；测试集共包含 5000 张 jpg 格式的图像。

图 1 CIFAR-10 **数据示例**

实验原理

CNN 模型

卷积神经网络（Convolutional Neural Network，CNN）是一种用于图像识别、计算机视觉和深度学习任务的深度神经网络模型。它在计算机视觉领域取得了很大的成功，并广泛应用于图像分类、目标检测、人脸识别等任务。

CNN 的基本原理是模拟人类视觉系统的工作方式。它通过一系列的卷积层、池化层和全连接层构成。下面是 CNN 的基本原理阐述：

1. 卷积层（Convolutional Layer）：卷积层是 CNN 的核心组件，用于提取输入图像的特征。每个卷积层由多个卷积核组成，每个卷积核在输入图像上进行滑动操作，通过计算卷积操作来提取特征。卷积操作可以捕捉图像中的局部空间信息，并且具有参数共享的特性，使得模型可以在不同位置共享学习到的特征。

2. 激活函数（Activation Function）：在卷积层之后，通常会添加一个非线性的激活函数，如 ReLU（Rectified Linear Unit），用于引入非线性变换，增加模型的表达能力。

3. 池化层（Pooling Layer）：池化层用于减小特征图的尺寸并保留重要的特征。常用的池化操作是最大池化（Max Pooling），它从输入的特征图中选取最大值作为输出，以此减小特征图的大小。池化操作可以减少参数数量，提高模型的计算效率，并且对输入的平移、缩放和旋转具有一定的不变性。

4. 全连接层（Fully Connected Layer）：在经过一系列的卷积层和池化层之后，CNN 通常会添加一个或多个全连接层，用于将特征映射到对应的类别。全连接层的每个神经元与前一层的所有神经元相连接，通过学习权重来实现特征的组合和分类。

5. Dropout 层：为了防止过拟合，CNN 中还常常使用 Dropout 层。Dropout 层随机地将一部分神经元输出置为零，可以有效地减少神经元之间的依赖关系，提高模型的泛化能力。

[1] https://www.kaggle.com/c/cifar-10

6. Softmax 层：在分类任务中，CNN 最后一层通常使用 Softmax 层，用于将网络的输出转化为类别的概率分布。Softmax 函数可以将网络输出的原始分数归一化为概率值，使得每个类别的预测概率之和为 1。

CNN 通过反向传播算法进行训练，使用梯度下降来更新网络中的参数。在训练过程中，CNN 通过最小化损失函数（如交叉熵损失）来优化网络的权重和偏置，使得网络能够输出正确的分类结果。

CIFAR-10 数据集

内容

CIFAR-10（Canadian Institute for Advanced Research-10）是一个经典的图像分类数据集，用于计算机视觉和机器学习领域的算法验证和性能评估。该数据集由来自 10 个不同类别的彩色图像组成，每个类别包含 6000 张图像，共计 60000 张图像。具体内容如下：

1. 类别：CIFAR-10 数据集包含以下 10 个类别的图像：
   - 飞机（airplane）
   - 汽车（automobile）
   - 鸟类（bird）
   - 猫（cat）
   - 鹿（deer）
   - 狗（dog）
   - 蛙类（frog）
   - 马（horse）
   - 船（ship）
   - 卡车（truck）
2. 图像特征：每个图像的尺寸为 32x32 像素，彩色图像包含红色、绿色和蓝色三个通道（RGB）。因此，每个图像可以表示为 3 个 32x32 的矩阵，共计 3072 个特征（32x32x3）。
3. 训练集和测试集：CIFAR-10 数据集被划分为训练集和测试集。训练集包含 50000 张图像，其中每个类别有 5000 张图像。测试集包含 10000 张图像，用于评估算法在未见过的数据上的性能。
4. 数据分布：CIFAR-10 数据集中的图像相对较小且分辨率较低，使得算法需要克服一些挑战才能准确地分类图像。图像中的对象通常出现在不同的位置、姿态和背景下，而且还可能存在遮挡和噪声。

CIFAR-10 数据集作为一个相对较小且挑战性的数据集，被广泛用于训练和评估图像分类算法，尤其是卷积神经网络（CNN）。许多研究和论文都使用 CIFAR-10 数据集来展示和比较不同算法的性能。

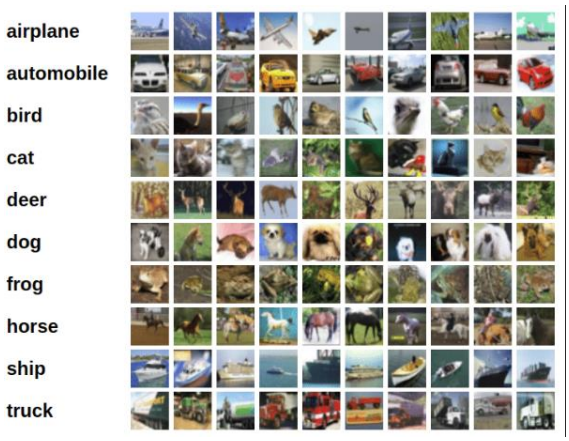


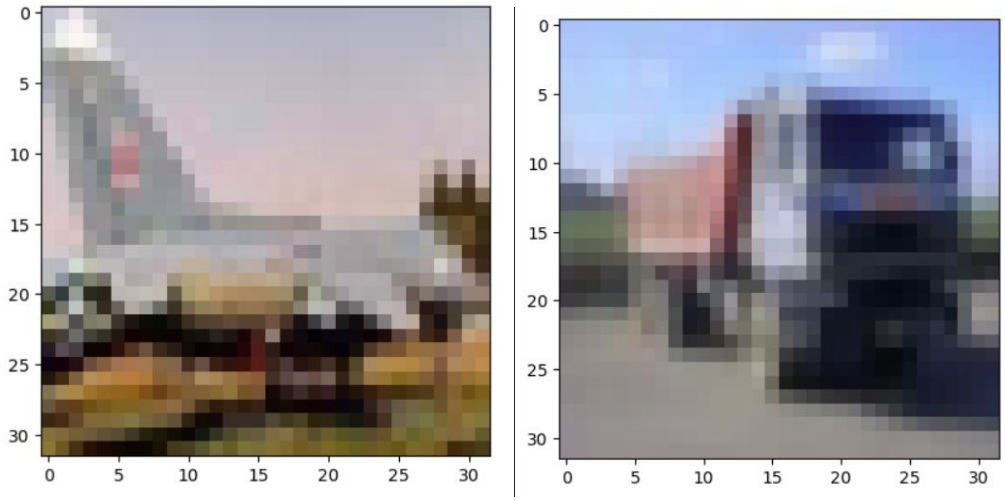
图 2 CIFAR-10 部分数据

数据预处理

数据读取

数据集位于目录 data/cifar0 中。它包含两个文件夹：训练和测试，分别包含训练集(5000 张图像)和测试集(1000 张图像)。每个文件夹包含 10 个文件夹，每类图像一个。这一目录结构（每个类一个文件夹）被许多计算机视觉数据集使用，并且大多数深度学习库都提供了用于处理此类数据集的实用程序。 我们可以使用 torchvision 中的 ImageFolder 类将数据加载为 PyTorch 张量。

之后，我们可以使用 matplotlib 查看图像，但是我们需要将张量维度更改为 (32,32,3)。在 PyTorch 中，图像的默认表示形式是（3，H，W），其中 3 表示通道数（RGB 图像为 3，灰度图像为 1），H 表示高度，W 表示宽度。然而，常用的图像显示库（如 matplotlib）期望图像的维度为（H，W，3），其中 3 表示 RGB 通道。将图像从（3，H，W）转换为（H，W，3）是为了符合常用的图像显示库对图像维度的要求，这样我们可以正确地显示和可视化图像。通过转换维度，我们可以在使用 matplotlib 等库显示图像时，正确地解释图像的通道数和像素的排列顺序。



数据增强

数据增强是在训练过程中对原始数据进行一系列变换操作，以产生更多的训练样本，从而增加数据的多样性。数据增强可以有效地扩展训练集的规模，减轻过拟合问题，并提高模型的泛化能力。

以下是一些常见的数据增强技术：

1. 翻转（Flipping）：包括水平翻转和垂直翻转，通过翻转图像可以增加数据的多样性。
2. 旋转（Rotation）：对图像进行旋转操作，以增加不同角度的训练样本。
3. 裁剪（Cropping）：随机裁剪图像的一部分，可以使模型对目标物体的位置具有更好的鲁棒性。
4. 缩放（Scaling）：将图像进行缩放操作，可以模拟不同尺度的目标物体。
5. 平移（Translation）：将图像进行平移操作，可以模拟不同位置的目标物体。
6. 亮度、对比度和饱和度调整（Brightness, Contrast, and Saturation Adjustment）：通过调整图像的亮度、对比度和饱和度，可以使模型对不同光照条件和颜色变化具有更好的适应能力。
7. 噪声添加（Noise Addition）：向图像中添加随机噪声，以增加数据的多样性和鲁棒性。
8. 变换（Transformation）：应用各种几何变换，如扭曲、拉伸等，以模拟不同视角和形变。

这些数据增强技术可以单独应用，也可以组合使用，具体选择哪些数据增强技术取决于应用场景和数据集的特性。通过数据增强，可以提高模型的泛化能力，并在有限的数据集上获得更好的性能。

此外，当使用数据增强技术时，需要小心过拟合的问题。如果应用过多的增强技术，可能会导致模型在训练集上过拟合，但在真实数据上表现不佳。因此，需要进行适当的监控和调整，确保模型在训练和测试集上都能获得好的性能。

具体来说，在本实验中我们可以仅使用来自 torchvision.transforms 中的 RandomCrop 和 RandomHorizontalFlip。

数据归一化

数据归一化是数据预处理中的一项重要步骤，它将数据按照一定规则进行缩放和标准化，以使数据具有相似的范围和统计特性。以下是几个原因说明为什么需要数据归一化：

1. 梯度下降算法：在机器学习中，很多模型的训练过程使用梯度下降算法来更新参数。梯度下降算法对输入数据的规模和范围敏感，如果不对数据进行归一化，可能会导致梯度更新过快或过慢，从而影响模型的收敛速度和稳定性。
2. 特征权重：在一些模型中，例如支持向量机（SVM）和 K 近邻（KNN），特征的尺度差异会对模型的学习和预测产生影响。如果某些特征的尺度较大，它们会在模型中起主导作用，而忽略其他尺度较小的特征。通过归一化，可以确保每个特征对模型的贡献相对平衡。
3. 模型收敛和稳定性：在训练神经网络等深度学习模型时，数据归一化可以帮助加速模型的收敛过程。对输入数据进行归一化可以使每个特征具有相似的范围，减小了梯度更新的差异，提高了模型的稳定性。
4. 避免数值计算问题：在进行数值计算时，如果输入数据具有不同的尺度，可能会导

致数值溢出或下溢等问题。通过归一化，可以将数据限制在合理的范围内，避免这些数值计算问题的发生。

5. 提高模型的泛化能力：通过数据归一化，可以减小特征之间的相关性，提高特征的独立性，从而帮助模型更好地泛化到未见过的数据。

综上所述，数据归一化可以帮助调整数据的尺度和范围，提高模型的训练效果、稳定性和泛化能力。它是数据预处理中的一个重要步骤，通常在训练模型之前应用于输入数据。

不同的数据集有不同的统计学特性，所以中值和标准差都需要单独计算。在标准的CIFAR10 数据集上，均值和方差分别是 $(0.4914, 0.4822, 0.4465)$ 和$(0.2470, 0.2435, 0.2616)$。然而本实验只使用了其中一部分数据，因此需要重新计算，可以通过如下代码

```
dataset = ImageFolder(data_dir+'/train', transform=ToTensor())
train_data = np.concatenate([np.array(image)[None, :] for image, _ in dataset], axis=0)
mean = np.mean(train_data, axis=(0,2,3))
std = np.std(train_data, axis=(0,2,3))
```

得到均值和标准差(0.4913,0.4809,0.4444)和$(0.2481, 0.2451, 0.2631)$。

在实践上，可以使用来自 torchvision.transforms 中的 Normalize。

数据集分割

在构建真实世界的机器学习模型时，将数据集分成三个部分是很常见的：

1. 训练集-用于训练模型，即用梯度下降法计算模型的损失和调整模型的权重。
2. 验证集--用于训练时评估模型，调整超参数(学习率等)，选择模型的最佳版本；
3. 测试集--用于比较不同模型或不同类型的建模方法，并报告该模式的最终准确性。

没有预定义的验证集，我们可以设置一个小的集合(500 张图像)来验证这个集合的有效性。使用来自 pytorch 的 random_split 做到这一点。为了确保我们总是创建相同的验证集，我们还将为随机数字发生器设置一个种子

```
val_size = 500
train_size = len(dataset) - val_size

train_ds, val_ds = random_split(dataset, [train_size, val_size])
len(train_ds), len(val_ds)
```

```
(4500, 500)
```

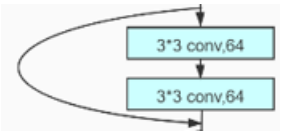
CNN 模型构建

为方便起见，以下步长（Stride）以 s 表示，填充（Padding）以 p 表示，过滤器（Filter）以 k 表示，输入通道数（in channels）以 in 表示，输出通道数（out channels）以 out 表示。

本实验中使用的模型为 resnet18，架构如下：

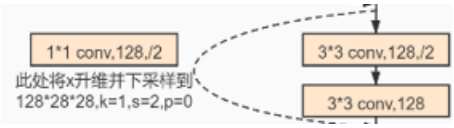
残差块：

基本残差块，参数 inchannel, outchannel, stride：



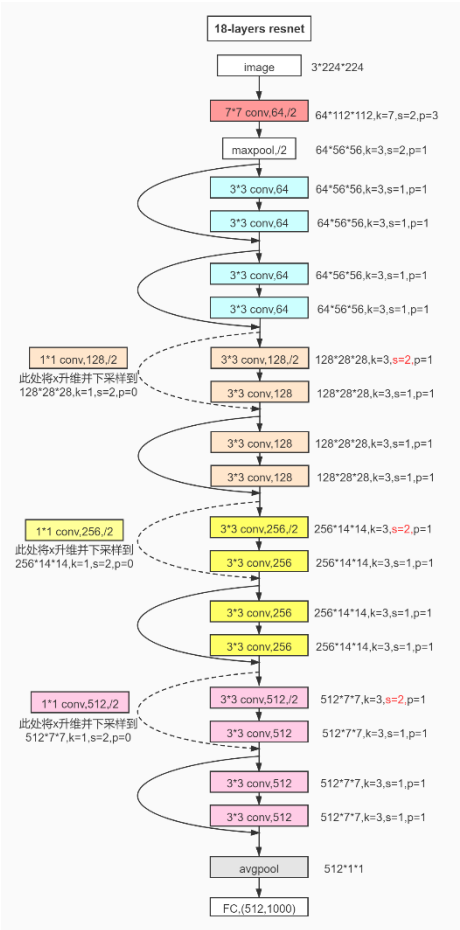
1. 卷积层 1：in=inchannel，out=outchannel, k=3x3, s=stride,p=1, BatchNorm，激活函数：Relu
2. 卷积层 2：in=outchannel, out=outchannel，k=3x3, s=1,p=1,BatchNorm
3. $f(x)+x$
4. Relu

下采样残差块，参数 inchannel, outchannel, stride：



1. 卷积层 1：in=inchannel，out=outchannel, k=3x3, s=stride,p=1, BatchNorm，激活函数：Relu
2. 卷积层 2：in=outchannel, out=outchannel，k=3x3, s=1,p=1,BatchNorm
3. $f(x)+Wx$, 此处 W 包含一个 k=1x1s=stride 的卷积层和 BatchNorm 用于调整原始输入维度
4. Relu

网络架构：



上图为 resnet18 在论文中的设置。注意到神经网络头部有一个使用了较大的卷积核的卷积层和 maxpool，这适用于较大的输入图像，而在 cifar10 数据集中图片的较小，如果原封不动照搬，会导致丢失许多信息，因此需要微调网络，把头部的两层换为一个 3x3 的卷积层。

以下是真实的网络架构：

1. 输入层：32x32 彩色图像
2. 卷积层 1：in=3,out=64,k=3,s=1,p=1,BatchNorm,激活函数：ReLU
3. 基本残差块 1：in=64,out=64,stride=1

4. 基本残差块 2：in=64,out=64,stride=1
5. 下采样残差块 1：in=64,out=128,stride=2
6. 基本残差块 3：in=128,out=128,stride=1
7. 下采样残差块 2：in=128,out=256,stride=2
8. 基本残差块 4：in=256,out=256,stride=1
9. 下采样残差块 3：in=256,out=512,stride=2
10. 基本残差块 5：in=512,out=512,stride=1
11. 平均池化
12. 展平输出
13. 全连接层 2（稠密层）：512 个神经元，激活函数：ReLU
14. 输出层（稠密层）：10 个神经元，激活函数：Softmax
15. 损失函数：分类交叉熵

实验设置

在本实验中，对比了两种情形下的训练方式：

A：one cycle policy + gradient clipping + weight decay + momentum SGD

B：adam

运行环境

Platform: google colab

GPU: RTX 3090

超参数

学习率（Learning rate）

学习率（Learning rate）是指在神经网络的训练过程中，用于调整模型权重的步长或速率。学习率对神经网络训练有以下影响：

1. 收敛速度：学习率决定了参数更新的步长。较大的学习率可以加快模型的收敛速度，因为每次参数更新的幅度更大。然而，如果学习率过大，可能会导致参数在最优解附近波动或震荡，使模型难以稳定地收敛。相反，较小的学习率会使模型收敛速度较慢，但更有可能找到更好的最优解。
2. 网络性能：学习率直接影响模型在训练数据和测试数据上的性能。过大的学习率可能会导致模型在训练集上表现良好，但在测试集上泛化能力较差，产生过拟合。过小的学习率可能会导致模型无法充分学习数据的模式和特征，表现为欠拟合。选择适当的学习率是优化模型性能的重要因素之一。
3. 跳出局部最优解：在训练过程中，模型可能会陷入局部最优解，而无法达到全局最优解。适当的学习率可以帮助模型跳出局部最优解并继续寻找更好的解决方案。较大的学习率可能有助于跳出局部最优，但也可能导致不稳定的训练过程。较小的学习率可以使模型更稳定地收敛，但可能陷入较差的局部最优。
4. 调整学习率策略：学习率还与调整策略相关。在训练过程中，学习率可能需要进行动态

调整，以便在后期阶段更加细致地调整模型参数。例如，学习率衰减、学习率衰减和动量等技术可以用来优化学习率的变化方式，以获得更好的训练结果。

在本实验当中，我们考虑实践 One Cycle Policy。

LR Range Test

2015 年，Leslie N. Smith 提出了该技术。其核心是将模型进行几次迭代，在最初的时候，将学习率设置的足够小，然后，随着迭代次数的增加，逐渐增加学习率，记录下每个学习率对应的损失，并绘图：（LR 的初始值仅为 1e-7，然后增加到 10）

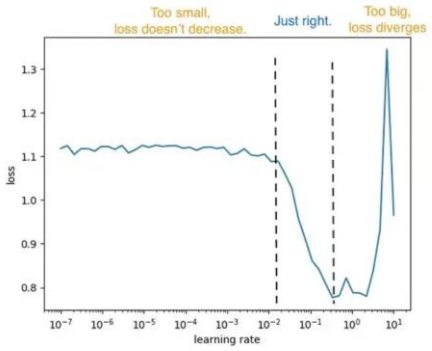


图 3 Loss 随 LR 的变化

LR Range Test 图应该包括三个区域，第一个区域中学习率太小以至于损失几乎没有减少，第二个区域里损失收敛很快，最后一个区域中学习率太大以至于损失开始发散。因此，第二个区域中的学习率范围就是我们在训练时应该采用的。我们把第二个区域的左端叫做下界学习率（base_lr），右端叫做上界学习率（max_lr）。

所以，这个方法字如其名，就是学习率范围测试，为训练寻找一个合适的学习率范围。

Cyclic Learning Rates

在一些经典方法中，学习率总是逐步下降的，从而保证模型能够稳定收敛，但 Leslie Smith 对此提出了质疑，Leslie Smith 认为让学习率在合理的范围内周期性变化（即 Cyclical LR： 在 base_lr 和 max_lr 范围内循环学习率）是更合理的方法，能够以更小的步骤提高模型准确率。

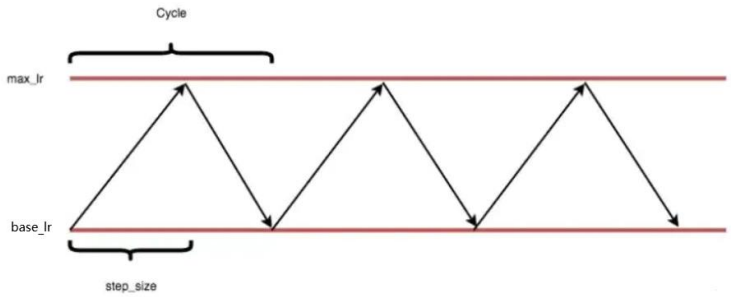


图 4 Cyclic LR 策略

One Cycle Policy

在 Cyclical LR 和 LR Range Test 的基础上，Leslie 继续改进，提出了 The one cycle policy。在一周期策略中，最大学习率被设置为 max_lr 中可以找到的最高值，最小学习率比最大学习率小几个数量级（比如设为 0.1 倍的 max_lr）。

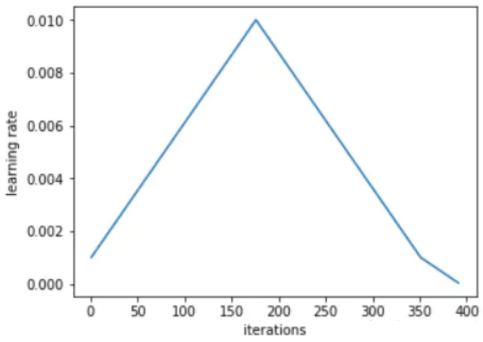


图 5 One Cycle Policy 策略

如上图，假设整个训练周期约 400 个 iter，前 175 个 iter 用来 warm-up。中间 175 个 iter 用来退火到初始学习率，由于此前有相当大的时间模型处于较高的学习率，作者认为，这将起到一定的正则化作用，防止模型在陡峭最小值驻留，从而更倾向于寻找平坦的局部最小值。最后几十个 iter 学习率进行进一步衰减至 0，这将使得模型在一个‘平坦’区域内收敛至一个较为‘陡峭’的局部最小值。

在实践上，我们可以直接使用 torch.optim.lr_scheduler.OneCycleLR。在训练时记录每一步 learning rate 并绘图，如下：

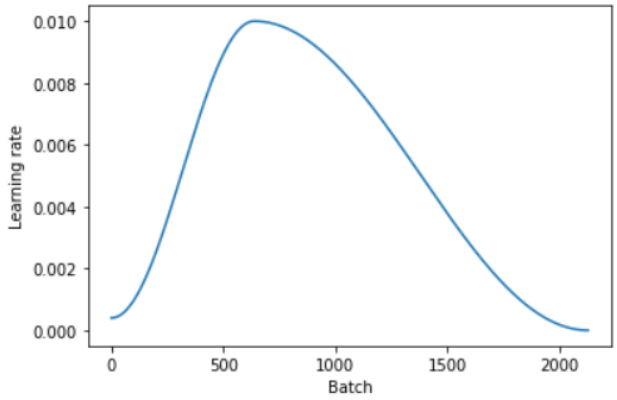


图 6 在 epochs=30,max_lr=0.01,optimizer=SGD 条件下，LR 随 Batch 变化的情况

学习周期（Epoch）

Epoch 是指将整个训练数据集在神经网络中前向传播和反向传播的一次完整迭代。Epoch 的数量直接影响模型是否能够收敛到最优解。较少的 Epoch 可能导致模型尚未完全学习训练数据中的模式和特征，从而无法达到最佳性能。较多的 Epoch 可以让模型有更多机会从训练数据中学习，但过多的 Epoch 可能导致过拟合，即模型过度适应训练数据而在未见过的数据上表现不佳。

绘制损失-学习周期图，由图可看出，在 30 轮之内还未出现明显的过拟合现象，而此时准确率的提升已经非常缓慢了，所以设置学习周期为 **30** 轮。

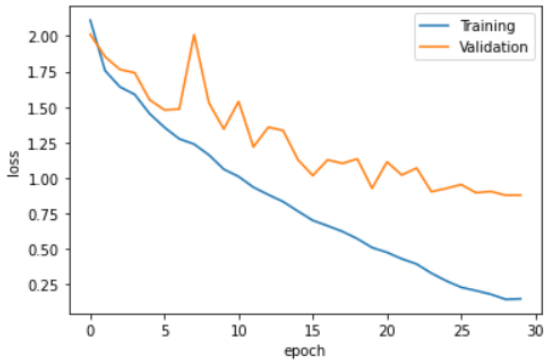


图 7 损失-学习周期图

批大小（Batch size）

批量大小（Batch Size）是指在神经网络的训练过程中，每一次迭代更新权重时所使用的样本数量。批量大小对神经网络的训练有以下几个影响：

1. 训练速度：较大的批量大小可以加快训练速度。这是因为在每个批次中同时处理多个样本，可以充分利用并行计算的能力，加快权重更新的速度。相比之下，较小的批量大小会导致更频繁的权重更新，训练过程可能会更加缓慢。
2. 内存需求：较大的批量大小需要更多的内存空间来存储中间结果和梯度。如果内存限制较低，较大的批量大小可能无法适应，需要降低批量大小或采用更高效的计算方式，如分布式训练或使用特殊的硬件加速。
3. 泛化能力：批量大小还与模型的泛化能力相关。较大的批量大小可以提供更稳定的梯度估计，有助于模型收敛。然而，较小的批量大小可能导致模型更快地收敛到局部最优解，并丧失一定的泛化能力。因此，选择合适的批量大小需要在训练数据和模型性能之间进行权衡。不过 Hoffer 等人的研究表明，大的 batchsize 性能下降是因为训练时间不够长，本质上并不少 batchsize 的问题，在同样的 epochs 下的参数更新变少了，因此需要更长的迭代次数。
4. 噪声影响：较小的批量大小引入了更多的随机性和噪声，因为每个批次的样本可能只代表了整个训练数据的一小部分。这种噪声可以被视为一种正则化机制，有助于减少过拟合。相比之下，较大的批量大小可能会降低噪声水平，导致模型更容易过拟合训练数据。

综上所述，批大小要在内存需求允许的范围内尽可能大。下面测试了当 batch_size=32,64,128,256 时的情况，经过实验，四个大小的 batch_size 运行时间相近（分别为 114, 116, 115, 114 秒），在测试集上的效果也相近，所以不妨保持原始设置，选择 **64**。

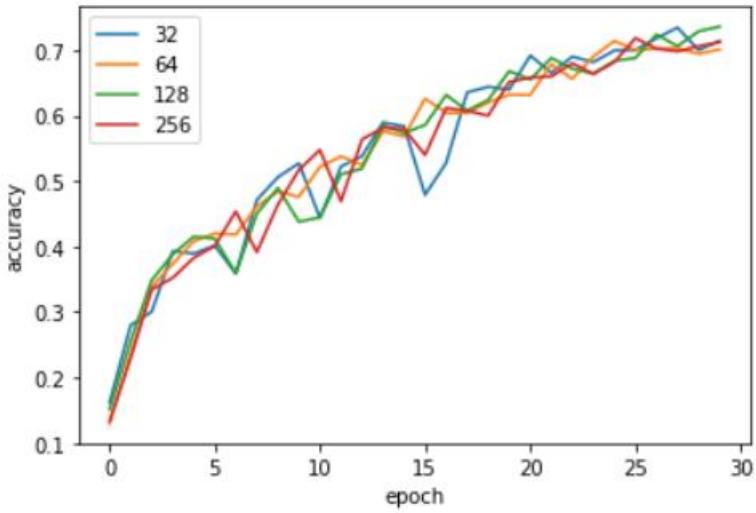


图 8 不同批大小下在测试集上准确率的变化

梯度截断（Gradient clipping）

梯度截断（gradient clipping）是一种在深度学习中用于稳定训练过程的技术。它主要应用于梯度爆炸的情况，即在网络训练过程中，梯度的数值变得非常大，导致训练不稳定或发散。具体来说，当计算出的梯度的范数（即梯度的绝对值的平方和开根号）超过了预先设定的阈值时，就将梯度进行缩放，使其范数不超过阈值。这个阈值通常称为截断值或阈值，可以手动设置。

在本实验中，我们将它设置为 **0.1**。

权重衰退（Weight decay）

权重衰减（weight decay）是一种在神经网络训练中常用的正则化技术。它的目的是通过在损失函数中引入一个附加项，来惩罚模型中较大的权重值。

在神经网络中，权重（或参数）的大小对模型的性能和泛化能力起着重要的影响。权重衰减通过在损失函数中增加一个正则化项，通常是权重的平方和（L2 范数），来限制权重的大小。这个额外的正则化项使得模型倾向于选择较小的权重值，从而降低过拟合的风险。

具体来说，权重衰减通过将权重的平方和乘以一个较小的正则化系数（也称为衰减因子）添加到损失函数中。这个正则化项会在训练过程中对较大的权重进行惩罚，使得模型倾向于选择较小的权重值。损失函数中的权重衰减项可以表示为：

$$L(w) = Loss(y, \hat{y}) + \lambda ||w||^2$$

其中，$L(w)$是加入权重衰减的损失函数，$Loss(y, \hat{y})$是原始的损失函数（例如，均方误差或交叉熵），λ是正则化系数，$||w||^2$是权重的平方和。

通过调整正则化系数λ的值，可以控制权重衰减的程度。较大的λ会对权重施加更大的惩罚，使得权重更加趋向于零。较小的λ会对权重施加较小的惩罚，允许权重值相对较大。

通过使用权重衰减，可以帮助防止模型在训练数据上过拟合，提高其泛化能力。这是因为权重衰减鼓励模型学习简单的权重分布，减少了过多依赖个别训练样本的风险。

在本实验中，我们选取$\lambda = \mathbf{10^{-4}}$。这是一个经验值，有些论文证明这是一个较优的阈值。

优化器

随机梯度下降(SGD)

SGD（随机梯度下降）是一种优化算法，常用于机器学习和深度学习中的参数优化问题。它是一种迭代算法，通过不断调整模型参数以最小化损失函数来训练模型。

SGD 的基本思想是通过每次迭代随机选择一个样本来计算梯度，并沿着梯度的负方向更新模型参数。与传统的批量梯度下降（BGD）相比，SGD 每次迭代只使用一个样本，因此更加高效。尽管每个样本的梯度估计可能不太准确，但在大规模数据集上，通过随机选择样本来估计整体梯度可以降低计算成本并且仍然能够收敛到较好的解。

具体来说，SGD 的更新规则可以表示为：

$$\theta = \theta - \alpha * \nabla J(\theta, x_i, y_i)$$

其中，θ是模型的参数，α是学习率，$J(\theta, xi, yi)$是损失函数，x_i是样本的特征向量，y_i是对应的目标值。$\nabla J(\theta, x_i, y_i)$表示损失函数对参数$\theta$的梯度。在每次迭代中，随机选择一个样本$(x_i, y_i)$，计算梯度并更新参数。

尽管 SGD 在更新模型参数时速度较快，但它也存在一些问题。首先，由于每次迭代只使用一个样本，梯度估计可能存在较大的方差，导致参数更新的不稳定性。为了解决这个问题，有一些改进的方法，如小批量随机梯度下降（Mini-batch SGD），它每次使用一小批样本来计算梯度。其次，SGD 对于设置合适的学习率很敏感，学习率过大可能导致振荡，学习率过小可能导致收敛速度过慢。

input : γ (lr), θ_0 (params), $f(\theta)$ (objective), λ (weight decay),
μ (momentum), τ (dampening), *nesterov*, *maximize*

for $t = 1$ **to** … **do**
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  if $\lambda \neq 0$
    $g_t \leftarrow g_t + \lambda\theta_{t-1}$
  if $\mu \neq 0$
    if $t > 1$
      $\mathbf{b}_t \leftarrow \mu\mathbf{b}_{t-1} + (1-\tau)g_t$
    else
      $\mathbf{b}_t \leftarrow g_t$
    if *nesterov*
      $g_t \leftarrow g_t + \mu\mathbf{b}_t$
    else
      $g_t \leftarrow \mathbf{b}_t$
  if *maximize*
    $\theta_t \leftarrow \theta_{t-1} + \gamma g_t$
  else
    $\theta_t \leftarrow \theta_{t-1} - \gamma g_t$

return θ_t

图 9 SGD 伪代码

为了解决振荡的问题，考虑使用动量法。

动量法（Momentum）是一种基于随机梯度下降（SGD）的优化算法，用于训练机器学习和深度学习模型。它通过积累之前梯度的信息来加速参数更新过程，以便更快地收敛到最优解。

在传统的随机梯度下降中，每次迭代只使用当前样本的梯度来更新参数，因此参数的更新方向完全依赖于当前样本的梯度。这可能导致在参数更新过程中出现较大的波动或者震荡，尤其在梯度方向变化较大的情况下。

动量法通过引入动量（momentum）的概念来解决这个问题。动量可以被看作是模拟物

体在梯度场中运动时的惯性。在每次迭代中，动量法不仅使用当前样本的梯度，还考虑了之前迭代中的梯度信息。具体而言，动量法引入一个参数β（取值范围为[0,1]），表示之前梯度的权重。通过累积之前梯度的方向和大小，动量法可以在梯度方向变化大的情况下继续前进，减小震荡的可能性。

动量法的参数更新规则可以表示为：

$$v = \beta v - \alpha \nabla J(\theta, x_i, y_i)$$
$$\theta = \theta + v$$

其中，v是动量向量，表示之前梯度的累积信息；α是学习率；$\nabla J(\theta, x_i, y_i)$是当前样本的梯度；$\theta$是模型的参数。

可以看到，动量法在参数更新时考虑了动量向量v，通过累积之前梯度的信息，使得参数更新的方向更加平滑，有助于快速收敛和避免震荡。

动量法的优点在于它有助于加速梯度下降过程并提高参数更新的稳定性。它特别适用于具有大规模数据集和复杂模型的训练。此外，动量法也能够帮助模型跳出局部最小值，并在搜索空间中探索更大范围的解。

需要注意的是，动量法的参数β需要根据具体问题进行调整。一般而言，较小的 β 值能够更快地适应梯度变化，但也可能导致震荡；较大的β值能够减少震荡，但可能会导致收敛速度较慢。而在本例中，使用$\beta = 0.5$.

Adam

Adam（Adaptive Moment Estimation）是一种自适应优化算法，用于训练机器学习和深度学习模型。它结合了动量法和自适应学习率的思想，旨在加快模型的收敛速度并提高优化性能。

Adam 算法维护了两个动量变量：一阶矩估计（即梯度的一阶矩）和二阶矩估计（即梯度的二阶矩）。这些动量变量类似于动量法中的动量向量，用于跟踪梯度的历史信息。

input : γ (lr), β_1, β_2 (betas), θ_0 (params), $f(\theta)$ (objective)
λ (weight decay), *amsgrad*, *maximize*
initialize : $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment), $\widehat{v_0}^{max} \leftarrow 0$

for $t = 1$ **to** … **do**
  if *maximize* :
    $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
  else
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  if $\lambda \neq 0$
    $g_t \leftarrow g_t + \lambda \theta_{t-1}$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  $\widehat{m_t} \leftarrow m_t / (1 - \beta_1^t)$
  $\widehat{v_t} \leftarrow v_t / (1 - \beta_2^t)$
  if *amsgrad*
    $\widehat{v_t}^{max} \leftarrow \max(\widehat{v_t}^{max}, \widehat{v_t})$
    $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m_t} / (\sqrt{\widehat{v_t}^{max}} + \epsilon)$
  else
    $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m_t} / (\sqrt{\widehat{v_t}} + \epsilon)$

return θ_t

图 10 Adam **伪代码**

具体而言，Adam 算法在每个参数的维度上维护以下变量：

1. 梯度一阶矩估计（Mean of gradients）：表示过去梯度的平均值。
2. 梯度二阶矩估计（Variance of gradients）：表示过去梯度平方的平均值。

Adam 算法的参数更新规则可以表示为：

$$m = \beta_1 m + (1-\beta_1)\nabla J(\theta, x_i, y_i) \quad \text{（梯度一阶矩估计）}$$

$$v = \beta_2 v + (1-\beta_2)\left(\nabla J(\theta, x_i, y_i)\right)^2 \quad \text{（梯度二阶矩估计）}$$

m和v的初始值都为 0，β_1和β_2是动量衰减率（通常设置为接近 1 的值，例如 0.9 和 0.999）。

校正偏差（Bias Correction）：

由于 m 和 v 的初始值为 0，在训练的早期阶段，它们的估计会被偏向较低的值。为了解决这个问题，Adam 引入了校正偏差修正：

$$\hat{m} = \frac{m}{1-\beta_1^t} \text{ (校正偏差修正)}$$

$$\hat{v} = \frac{v}{1-\beta_2^t}$$

其中，t 表示当前迭代的次数。

参数更新：

$$\theta = \theta - \frac{\alpha \hat{m}}{\sqrt{\hat{v}} + \varepsilon}$$

其中，α是学习率，ε 是一个小的常数（如10^{-8}），用于避免除以零的情况。

Adam 算法的主要优势在于它结合了动量法和自适应学习率的优点。它能够自适应地调整每个参数的学习率，使得参数在梯度变化较大的方向上更快地更新，并在梯度变化较小的方向上缓慢更新。这有助于加快收敛速度，并且对于不同参数具有不同的学习率。

由于 Adam 良好的自适应性质，它难以与 One Cycle Policy 结合使用，所以本实验对比了以 Adam 为优化器和以 momentum SGD+One Cycle Policy 为优化策略的两种训练方式。

其他设置

- 性能指标：准确率

实验结果

模型训练

由下图可见，使用 A 组训练方式能获得比 B 组更高的准确率与更低的训练/验证损失。

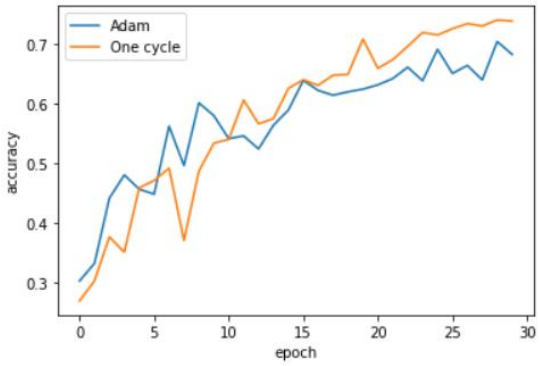


图 11 在验证集上两种训练方式的准确率与训练轮数的关系

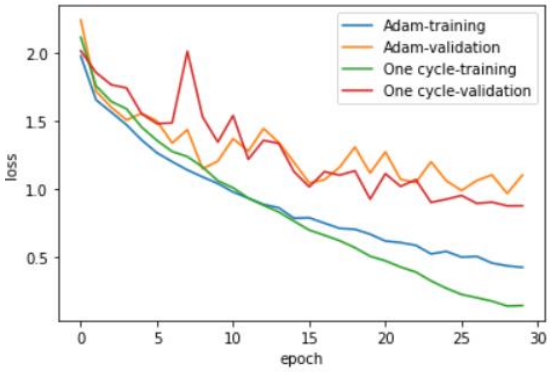


图 12 两种训练方式的损失与训练轮数的关系

模型评估

由下图可见，模型正确了给定的图像。由于之前做过数据增强所有图片有些许失真，但仍然肉眼可辨为青蛙，分类正确。

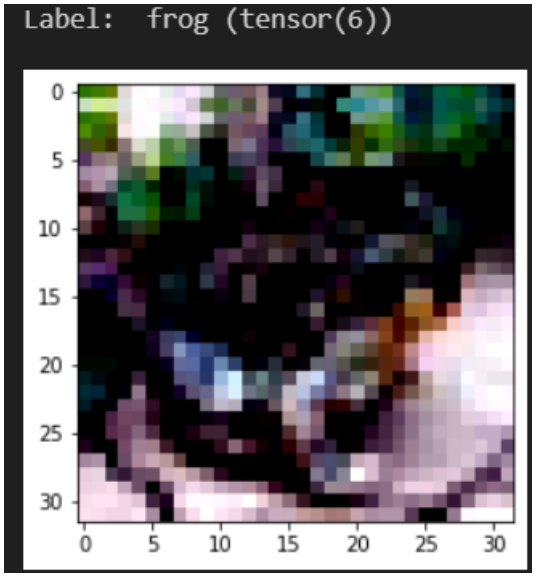


图 13 模型在测试集上给出的预测标签和对应的图片

对比

在完整测试集上，比较两种模型的准确率。A 训练方法获得了 76.2%的准确率，而 B 训练方法仅 71%。因此使用 momentum SGD+one cycle policy+weight decay+gradient clipping 更好。