数算实习大作业——Cifar100 图像分类

任务描述

使用卷积神经网络实现对 Cifar-100 数据集的图像分类任务

成员分工、项目情况介绍

我们组由三个人组成——苏超、杨晨博和李潼，苏超任项目组长。在项目开始初期，我们几个人对项目进行了简单的讨论和规划，制定了初步的项目计划：由于我们对卷积神经网络不是十分了解，经验不足，于是决定每个人先独立研究学习并查阅相关资料。经过简单的自学以及互相之间的讨论交流，我们利用 Keras 框架仿照 Alexnet 和 Lenet 模型编写了一个简单的卷积神经网络，初步实验后，在测试集上准确率为 35.66%，效果不是十分理想。

通过进一步的学习讨论，我们发现主要是过拟合的问题，于是我们进行了简单的分工：一方面是进行网络参数的调整和实验，由李潼和杨晨博分别就激活函数和优化器的选择进行相关调整实验；另一方面是解决过拟合问题，由苏超负责。

完成了相关的参数实验和初步解决了过拟合问题后，我们又从网络结构出发，尝试使用更加复杂的网络结构。参考 VGG16 的网络结构，我们实现了一个较上一个版本更加复杂的卷积神经网络，网络的层数更多，需要训练的参数也更多，最终的实验结果——测试集上准确率为 69.96%，效果达到了我们的预期要求。

Cifar100 数据集

Cifar-100 是由 Alex Krizhevsky、Vinod Nair 和 Geoffrey Hinton 收集得到的一个

在机器学习领域通用的用于图像识别和分类的基础数据集；包含 60000 张 32x32 的彩色图像，根据图像内容被分为 100 个小类和 20 个大类，每个小类包含 600 张图像，其中有 500 张训练图像和 100 张测试图像。每张图像带有一个小类的“fine”标签和一个大类的“coarse”标签，本次实习作业我们采用的是小类标签。每个类别包含的图像是互斥的，比如说交通工具 1 大类中有 bicycle、bus、motorcycle、pickup truck 和 train 而不包含交通工具 2 大类中的 lawn-mower、rocket、streetcar、tank 和 tractor。

数据集有两种主要的格式：Python/Matlab 版本的格式和二进制版本的格式。Python 版本的数据文件主要包含数据和标签两种元素。数据是指一个 uint8 类型的 10000*3072 的 numpy 数组。数组的每一行存储一张 32*32 的彩色图像，前 1024 个条目代表红色通道，接下来的 1024 个条目代表绿色通道，最后的 1024 个条目代表蓝色通道。图像是按照行优先的顺序存储的，前 32 个条目代表图像第一行的像素数值。标签是指一个由 0 到 99 之间的整数组成的数组，数组中的第 i 项代表第 i 张图像所属的类别标号。

我们使用的 Keras 框架包含了自动下载数据集并进行数据格式转换的工具，不需要我们额外编写代码进行相关处理。

卷积神经网络——CNN

卷积神经网络是近些年发展起来，并引起广泛重视的一种高效识别方法。20 世纪 60 年代，Hubel 和 Wiesel 在研究猫脑皮层中用于局部敏感和方向选择的神经元时发现其独特的网络结构可以有效地降低反馈神经网络的复杂性，继而提出了卷积神经网络（Convolutional Neural Networks-简称 CNN）。现在，CNN 已经成

为众多科学领域的研究热点之一，特别是在模式分类领域，由于该网络避免了对图像的复杂前期预处理，可以直接输入原始图像，因而得到了更为广泛的应用。

卷积神经网络是一种特殊的深层的神经网络模型。相比于普通的神经网络，卷积神经网络包含了一个由卷积层和子采样层构成的特征提取器。一方面卷积层中神经元间的连接方式并不是全连接的，一个神经元只与部分邻层神经元相连；另一方面，同一层中某些神经元连接的权重和偏移是共享的（即相同的）。非全连接和权值共享的网络结构使之更类似于生物神经网络，降低了网络模型的复杂度，减少了模型的参数，降低了过拟合的风险，这对于很难学习的深层结构来说，非常重要。

局部感受野和权值共享的思想是卷积神经网络的核心。一般认为人对外界的认知是从局部到全局的，而图像的空间联系也是局部的像素联系较为紧密，而距离较远的像素相关性则较弱。因而，每个神经元其实没有必要对全局图像进行感知，只需要对局部进行感知，然后在更高层将局部的信息综合起来就得到了全局的信息。局部感受野思想就是是受生物学里面的视觉系统结构的启发产生的——视觉皮层的神经元就是局部接受信息的，这些神经元只响应某些特定区域的刺激。权值共享的思想则是基于图像的特征的提取与位置没有明显联系，也就是说图像一部分的特征提取方式适用于该图像的其他部分，因此对于图像上的所以位置，我们都可以使用相同的学习特征。

一个卷积神经网络主要由三部分构成。第一部分是输入层，第二部分是由 N 个卷积层和池化层的组合构成，第三部分则是一个全连接的多层感知分类器。

以下是卷积神经网络中经常见到的网络层类型：

1、数据输入层：对原始图像数据进行预处理，包括：

(1) 去均值：把输入数据各个维度都中心化为 0

(2) 归一化：幅度归一化到同样的范围

简单来说就是对输入数据进行一个标准化操作：

$$\text{layer0} = \frac{input - mean}{std}$$

2、卷积层：进行特征提取。

主要是进行卷积操作，使用不同的卷积核对整个图像做滑动卷积计算，得到多个不同的特征图——Feature Map。卷积核就是一个权值矩阵，相当于传统神经网络中的权值参数，将卷积核的各个权值参数与对应的像素值相乘求和（再加上偏置值），就得到了卷积操作的输出结果。

在局部扫描的过程中，有一个参数叫做步长——strides，就是指卷积核以多大的跨度上下或左右平移地扫描。对于进行卷积操作的图像，需要对边界进行相关的处理，一般有两种处理方式：same padding 和 valid padding。Same padding 就是指在处理图像边界以外的地方增添一圈多余的 0，这样使得如果卷积核的移动步长为 1，卷积后得到的新图像和原图像形状一致。

3、激励层：对卷积层的输出进行非线性映射，向网络中引入非线性。

一个只有线性关系隐含层的多层神经网络不会比一般的只包含输入层和输出层的两层神经网络更加强大，因为线性函数的函数仍然是一个线性函数。但是加入非线性以后，多层神经网络的预测能力就得到了显著提高。激活函数的作用就是给网络加入一些非线性因素，使得神经网络可以更好地解决较为复杂的问题。

4、池化层：压缩数据和参数的量，降低网络训练参数，减小过拟合程度。

池化操作也叫子采样，主要方法有两种：Max pooling 和 Average pooling。比

较常用的是最大池化的方法，就是对输入的特征图进行采样操作，取采样范围内的最大值作为采样结果。一般而言，池化操作不会进行重叠采样。

5、全连接层：根据提取出来的特征进行分类。

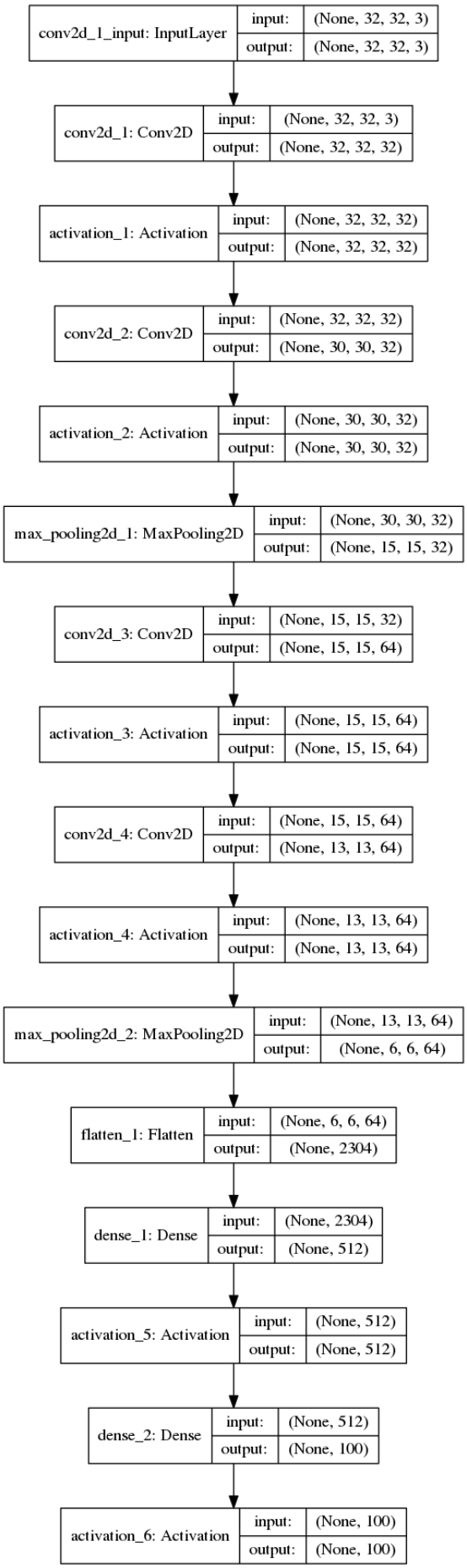
两层之间所有的神经元都有权重连接，通常全连接层在卷积神经网络的尾部，和传统的神经网络的连接方式相同。

卷积神经网络的训练方法和传统的机器学习算法类似，定义恰当的损失函数——loss function，用来衡量与正确结果之间的差距。然后使用优化器对损失函数进行优化，得到使 loss function 达到最小的训练参数，也就是我们最后得到的训练模型。

实验内容

1. 基准模型的建立

参考经典的 Lenet-5 和 Alexnet 网络结构，我们实现了一个简单的卷积神经网络，网络结构如下图所示。


conv2d_1_input: InputLayer
input: (None, 32, 32, 3)
output: (None, 32, 32, 3)
conv2d_1: Conv2D
input: (None, 32, 32, 3)
output: (None, 32, 32, 32)
activation_1: Activation
input: (None, 32, 32, 32)
output: (None, 32, 32, 32)
conv2d_2: Conv2D
input: (None, 32, 32, 32)
output: (None, 30, 30, 32)
activation_2: Activation
input: (None, 30, 30, 32)
output: (None, 30, 30, 32)
max_pooling2d_1: MaxPooling2D
input: (None, 30, 30, 32)
output: (None, 15, 15, 32)
conv2d_3: Conv2D
input: (None, 15, 15, 32)
output: (None, 15, 15, 64)
activation_3: Activation
input: (None, 15, 15, 64)
output: (None, 15, 15, 64)
conv2d_4: Conv2D
input: (None, 15, 15, 64)
output: (None, 13, 13, 64)
activation_4: Activation
input: (None, 13, 13, 64)
output: (None, 13, 13, 64)
max_pooling2d_2: MaxPooling2D
input: (None, 13, 13, 64)
output: (None, 6, 6, 64)
flatten_1: Flatten
input: (None, 6, 6, 64)
output: (None, 2304)
dense_1: Dense
input: (None, 2304)
output: (None, 512)
activation_5: Activation
input: (None, 512)
output: (None, 512)
dense_2: Dense
input: (None, 512)
output: (None, 100)
activation_6: Activation
input: (None, 100)
output: (None, 100)

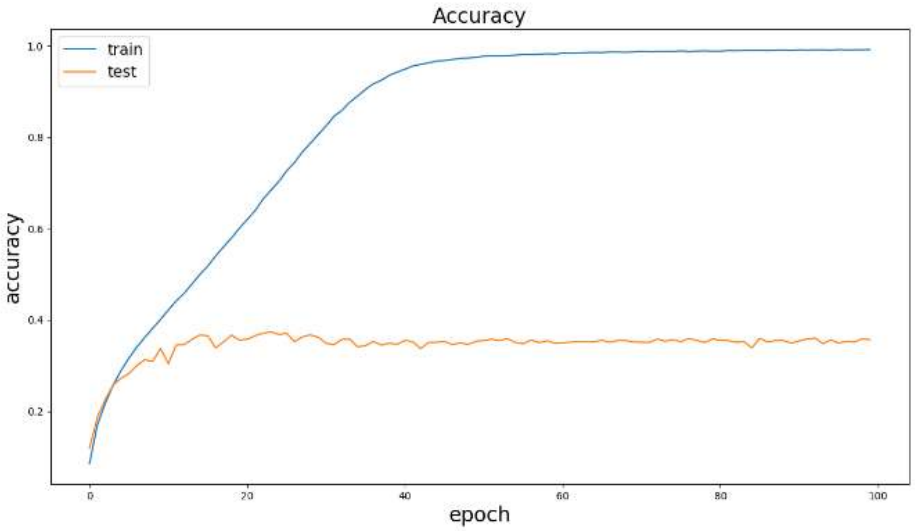
batch_size 设为 64。一般而言，适当增大 batch_size 会使梯度下降的方向更加准确，引起的训练震荡越小，但是考虑到我们所用机器的内存限制以及训练时间问题，我们将 batch_size 设为 64，平衡了训练时间和训练效果的矛盾冲突。

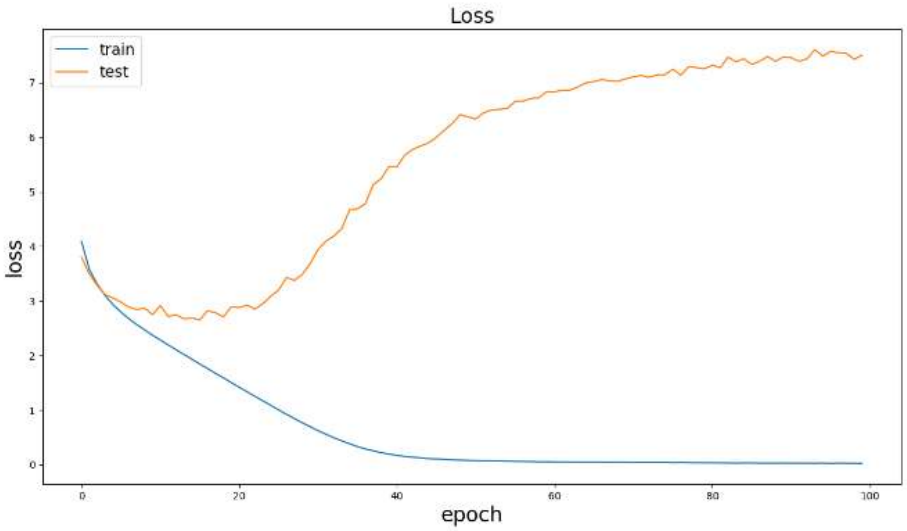
epochs 设为 100。这是我们多次实验后得到的结果，发现经过 100 轮的训练，训练集上的 loss 已基本趋于稳定收敛。

卷积核的大小设置为 3*3（深度也为 3），这是参考了卷积核的普遍大小得到的参数。pool_size 的大小设置为 2*2，也是普遍情况下采样区域的大小。激活函数使用了最近非常受欢迎的 ReLU 激活函数，优化器采用了比较实用、有效的 RMSprop 算法。

ReLU(修正线性单元)激活函数，它的特点是相比 sigmoid 和 tanh 函数，收敛快，求梯度简单，在正向传播阶段只需要设置一个阈值就可以得到激活值；反向传播过程中，减轻了梯度弥散的问题，神经网络前几层的参数也可以很快的更新。但是在训练的时候很“脆弱”，一不小心有可能导致神经元“坏死”。

实验结果如图所示：





Loss: 7.5043

Accuracy: 35.66%

从上面的实验结果，我们发现了两个问题：一是模型在训练数据上的表现明显好于测试集中的数据，这说明我们训练的模型出现了过拟合现象，我们需要解决这个问题；二是调节改变使用的激活函数以及优化器对网络性能和效果会有什么影响呢？

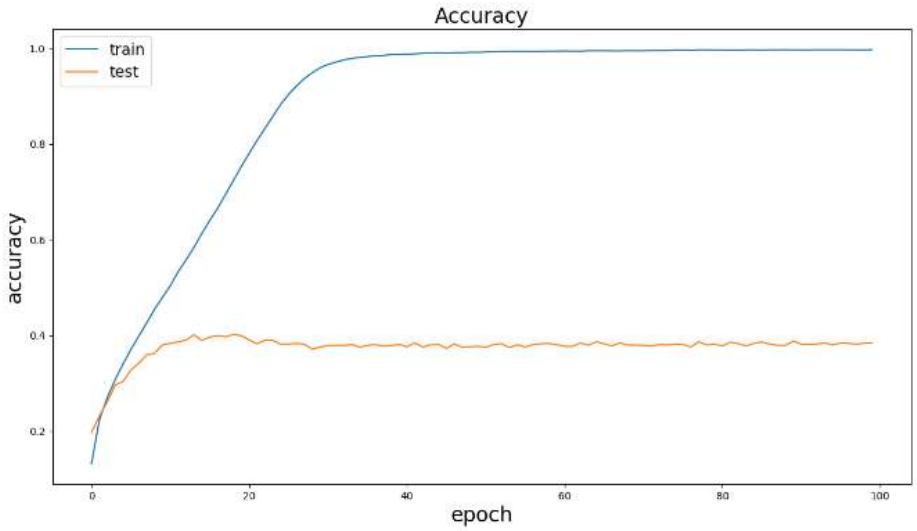
我们兵分三路，对上面的问题进行了相关的实验和研究。

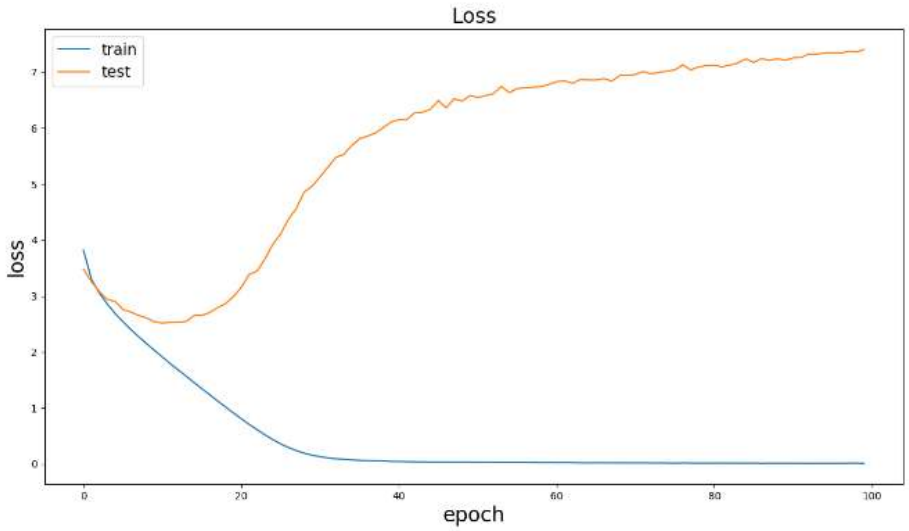
2. 网络参数调整实验

a) 激活函数

在保持网络结构不变和其他网络参数不变的情况下，我们分别采用了 ELU 激活函数和 LeakyReLU 激活函数进行了相关实验。实验结果如图所示：

ELU：

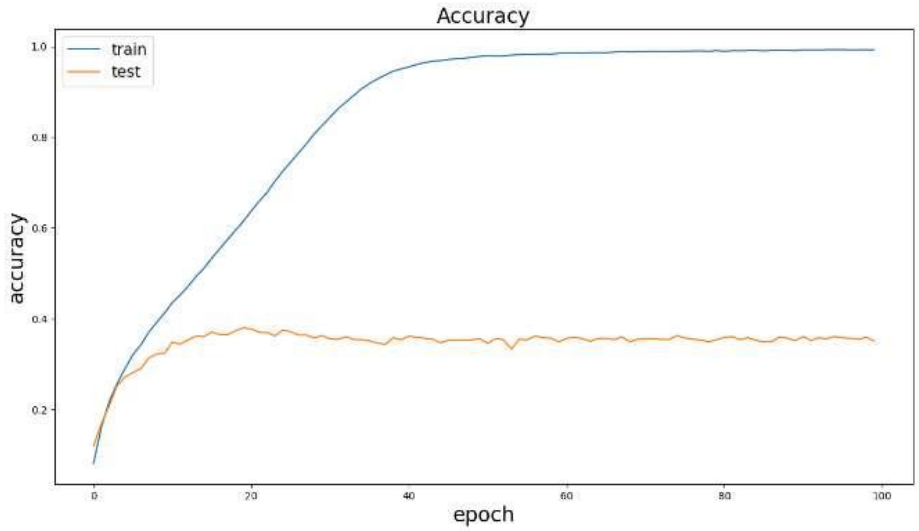


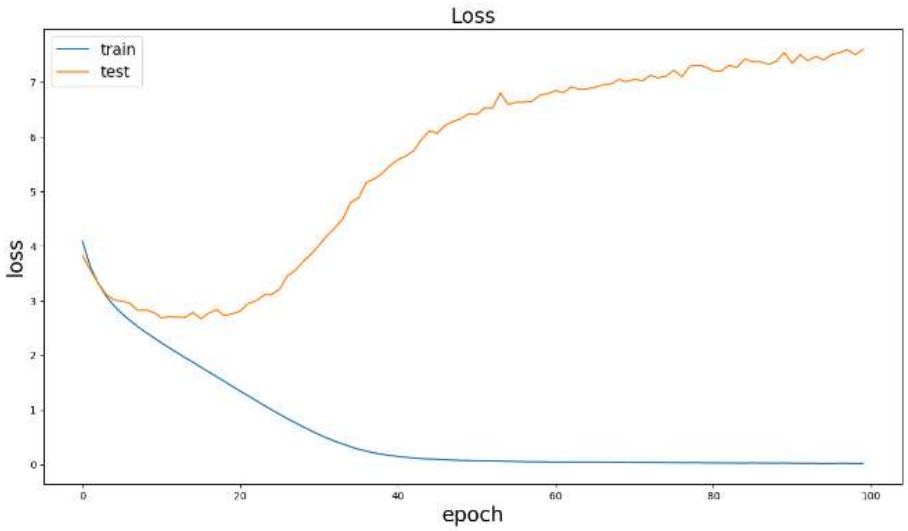


Loss: 7.4018

Accuracy: 38.43%

LeakyReLU:





Loss: 7.5961

Accuracy: 35.12%

	Loss	Accuracy
ReLU	7.5043	35.66%
ELU	7.4018	38.43%
LeakyReLU	7.5961	35.12%

通过上面的结果，我们可以发现 ELU 的效果是三者之中最好的。通过查看训练集上 loss 的收敛情况，我们可以看到 ELU 在 30 轮左右就已经稳定收敛了，而 ReLU 和 Leaky ReLU 都需要大约 40 轮才能达到收敛。而且 ELU 在测试集上的准确率也是最高的，达到了 38.43%。

Leaky ReLU 的出现就是为了解决 ReLU 会导致神经元坏死的问题的。和 ReLU 不同，当 x＜0 时，它的值不再是 0，而是一个斜率较小(如 0.01)的线性函数。理论上，这样的设置既修正了数据的分布，又保留了一些负轴的值，使得负轴信息不会全部丢失。但是实验表明，Leaky ReLU 的效果并没有达到理论的预期，当然这也有可能和我们设置的斜率 a 的大小有关系。据有关文章所说，如果斜率 a 很小，那么 Leaky ReLU 和 ReLU 差别并不大，只有当斜率大一些时，效果才会好很多。由于时间关系，我们并未对其进行过多的实验验证。我们把斜率稍微调大后，发现效果仍未发生明显变化，继续增大斜率甚至会导致模型失效，对此可以进行更深一步的研究。

ELU 被定义为：

$$\mathrm{f(x)} = \begin{cases} a(e^x - 1) & if(x < 0) \\ x & if(x \geq 0) \end{cases}, \text{ 其中 } a > 0$$

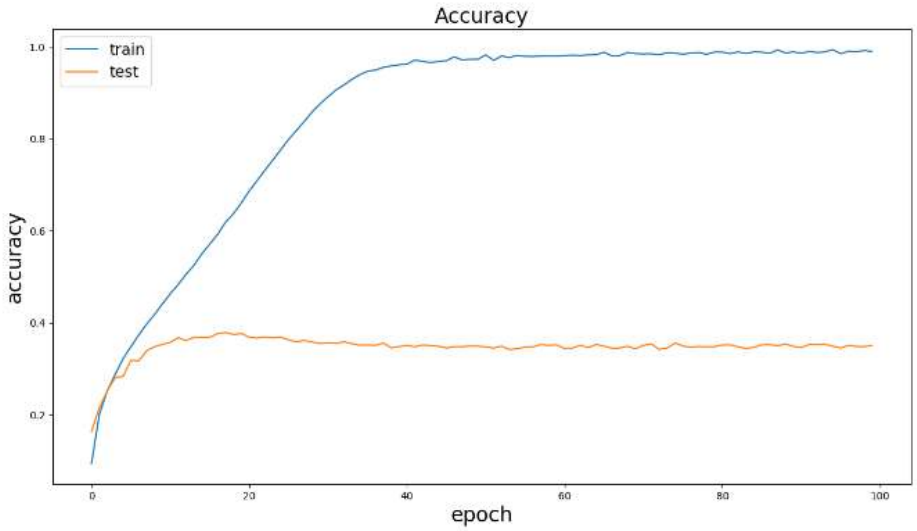
ELU 减少了正常梯度和单位自然梯度间的差距，从而加快了学习；在负

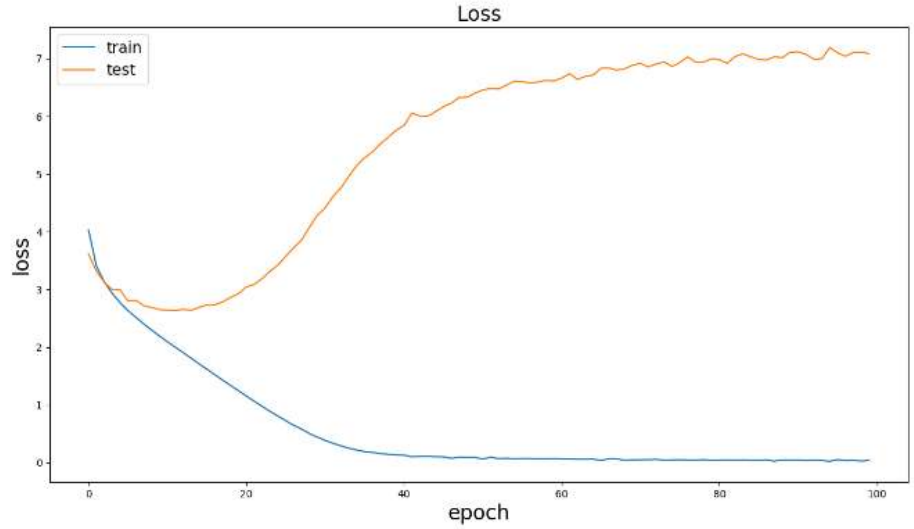
的限制条件下更有鲁棒性。

b) 优化器

在保持网络结构不变和其他网络参数不变的情况下，我们分别采用了SGD 优化器和 Adam 优化器进行了相关实验。实验结果如图所示：

Adam:

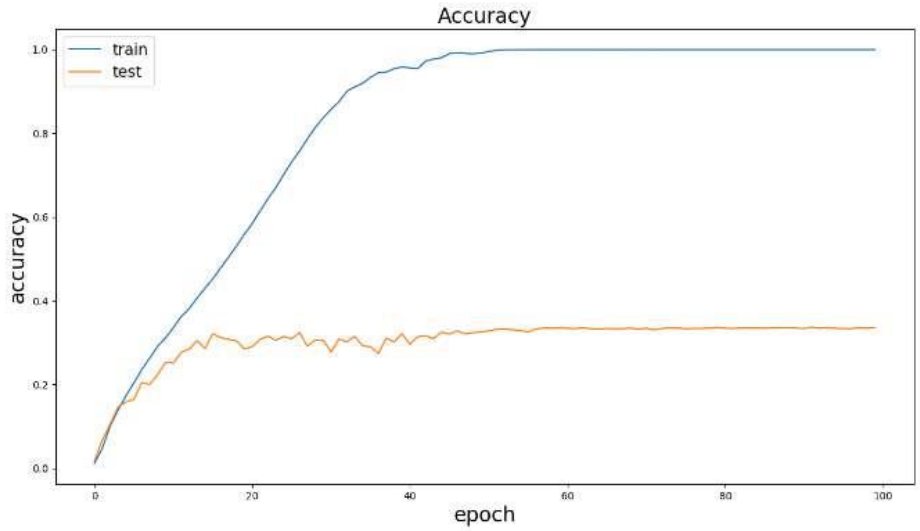


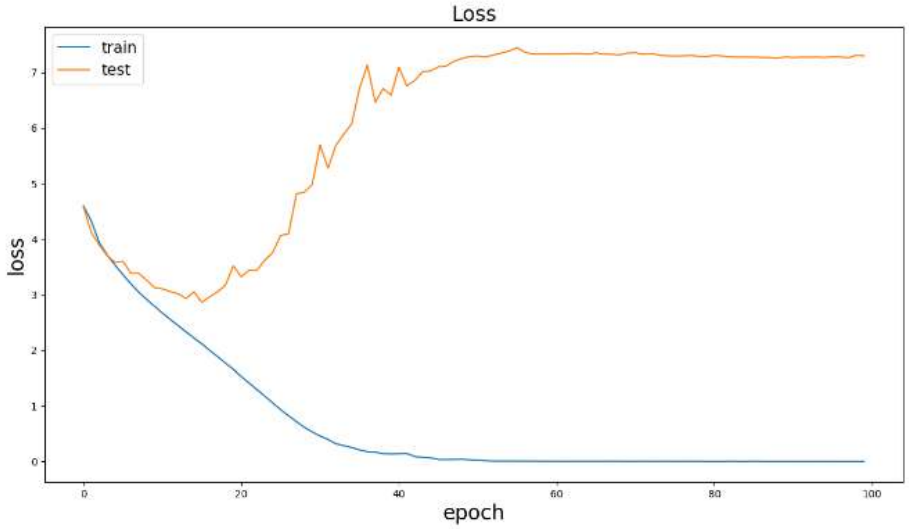


Loss: 7.0886

Accuracy: 35.03%

SGD:





Loss: 7.2962

Accuracy: 33.62%

	Loss	Accuracy
RMSprop	7.5043	35.66%
Adam	7.0886	35.03%
SGD	7.2962	33.62%

实验结果表明，Adam 的收敛速度更快，效果也十分的不错，和 RMSprop 相近，并且理论分析表明：Adam 对于超参数的选择十分鲁棒。SGD 的收敛速度比较慢，而且效果不是很好，因为更新比较频繁，loss 也有一定的震荡现象。但是对于 SGD 优化器，如果学习率调整得好的话，尽管收敛速度较慢，但是最后的效果往往会比其他自适应学习率的优化算法要好——这就是常见的随着迭代的轮数，逐渐降低学习率的方法（比如学习率减半）。

3. 解决过拟合问题

通过查阅相关资料，过拟合的问题往往是由于模型太复杂，参数过多，训

练过度导致的。还有一个重要的原因就是数据量有限，使得训练的模型无法真正了解整个数据的真实分布。

针对上面提到的方面，我们进行了相关尝试。

1. 数据增强——data augmentation

针对上面提到的训练样本数目有限，缺乏多样性导致训练模型泛化能力很差的问题，我们对已有的数据进行了提升。

进行了大量试验后，我们最终采用了对图像进行随机转动一定角度，水平、竖直偏移以及随机水平翻转的提升方法。

2. 权值衰减

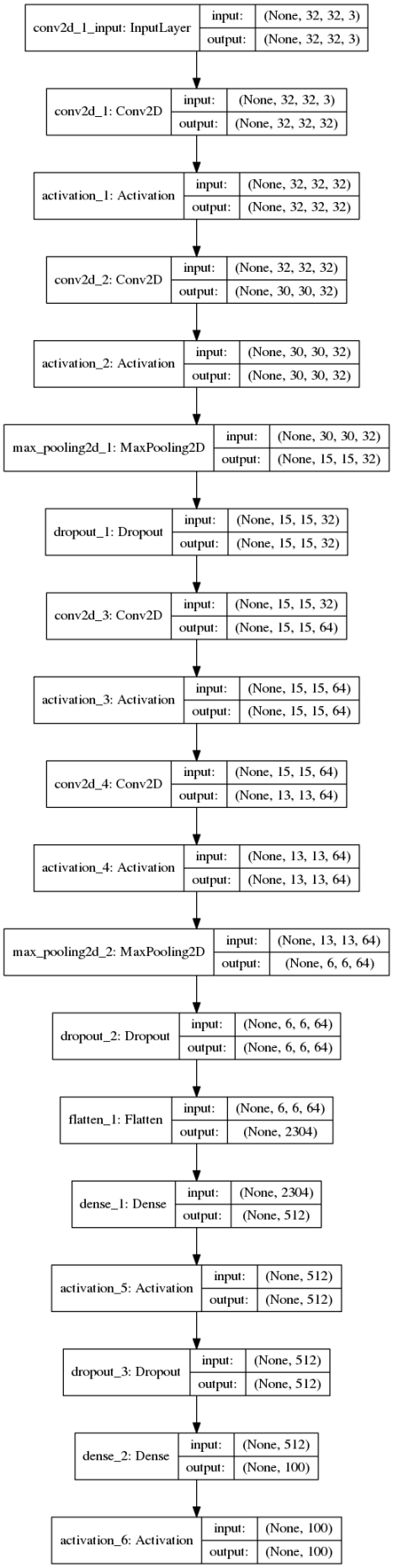
常用的权值衰减有两种：L1 和 L2 正则化。权值衰减是放在正则项前面的一个系数，而正则项一般表示模型的复杂程度，所以权值衰减的作用是利用正则项调节模型复杂度对损失的影响。L2 正则项可以起到使得参数 w 变小加剧的效果，而更小的参数值就意味着模型的复杂度更低，这样模型的泛化能力就可以得到保障。由于 L1 正则化在零点处不可导，因此使用比较广泛的是 L2 正则化，本次实验我们也是使用了 L2 正则化。

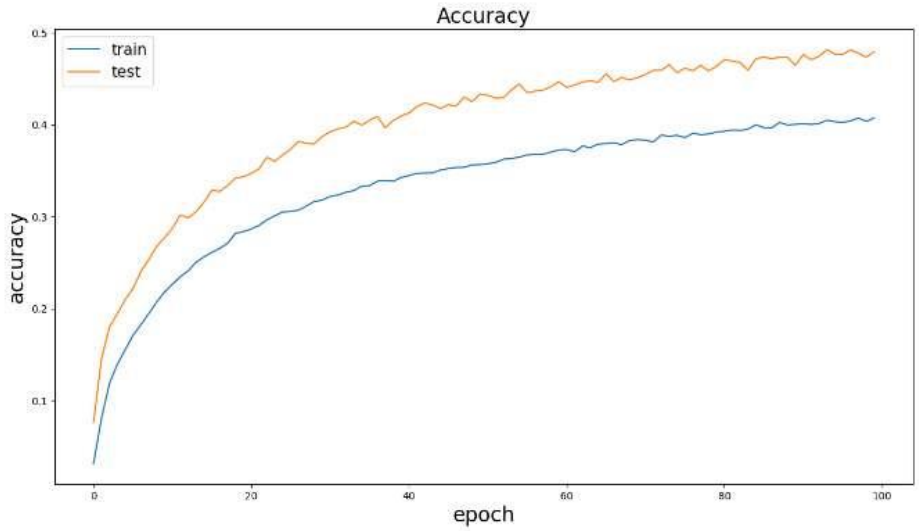
3. Dropout

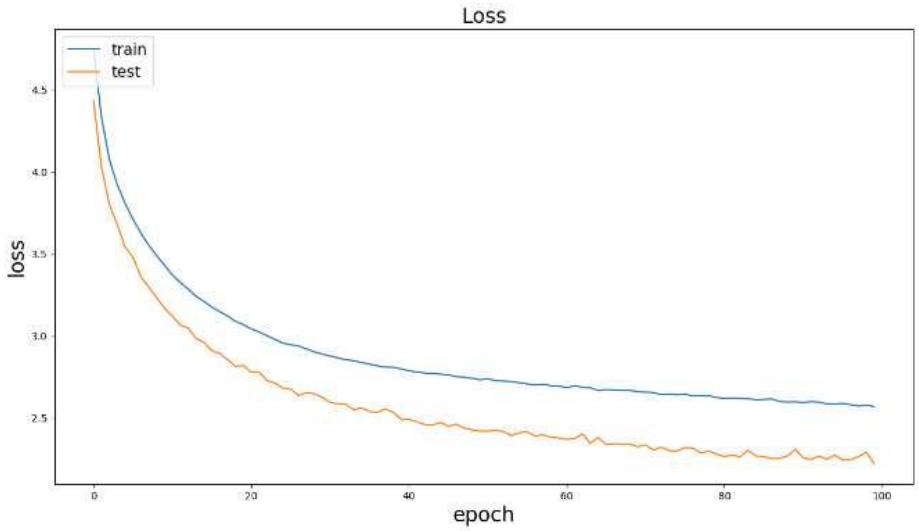
Dropout 是指在每次训练过程中随机将部分神经元的权值置为 0，也就是说让某些神经元失效。这样做的目的是缩减模型的参数量，避免过拟合的产生。关于 Dropout 的有效性，有两种主要观点：1）每次迭代随机使部分神经元失效使得模型的多样性增强，获得了类似多个模型组合的效果，避免过拟合 2）Dropout 其实也是一个数据增强的过程，它导致了稀疏性，使得局部数据差异性更加明显。

换句话说，Dropout 减少了神经元之间的依赖性，使得神经网络更能学习到与其他神经元之间的更健壮鲁棒的特征。

应用了上面三种技术后，我们的网络模型如下图所示：


conv2d_1_input: InputLayer
input: (None, 32, 32, 3)
output: (None, 32, 32, 3)
conv2d_1: Conv2D
input: (None, 32, 32, 3)
output: (None, 32, 32, 32)
activation_1: Activation
input: (None, 32, 32, 32)
output: (None, 32, 32, 32)
conv2d_2: Conv2D
input: (None, 32, 32, 32)
output: (None, 30, 30, 32)
activation_2: Activation
input: (None, 30, 30, 32)
output: (None, 30, 30, 32)
max_pooling2d_1: MaxPooling2D
input: (None, 30, 30, 32)
output: (None, 15, 15, 32)
dropout_1: Dropout
input: (None, 15, 15, 32)
output: (None, 15, 15, 32)
conv2d_3: Conv2D
input: (None, 15, 15, 32)
output: (None, 15, 15, 64)
activation_3: Activation
input: (None, 15, 15, 64)
output: (None, 15, 15, 64)
conv2d_4: Conv2D
input: (None, 15, 15, 64)
output: (None, 13, 13, 64)
activation_4: Activation
input: (None, 13, 13, 64)
output: (None, 13, 13, 64)
max_pooling2d_2: MaxPooling2D
input: (None, 13, 13, 64)
output: (None, 6, 6, 64)
dropout_2: Dropout
input: (None, 6, 6, 64)
output: (None, 6, 6, 64)
flatten_1: Flatten
input: (None, 6, 6, 64)
output: (None, 2304)
dense_1: Dense
input: (None, 2304)
output: (None, 512)
activation_5: Activation
input: (None, 512)
output: (None, 512)
dropout_3: Dropout
input: (None, 512)
output: (None, 512)
dense_2: Dense
input: (None, 512)
output: (None, 100)
activation_6: Activation
input: (None, 100)
output: (None, 100)





Loss: 2.2251

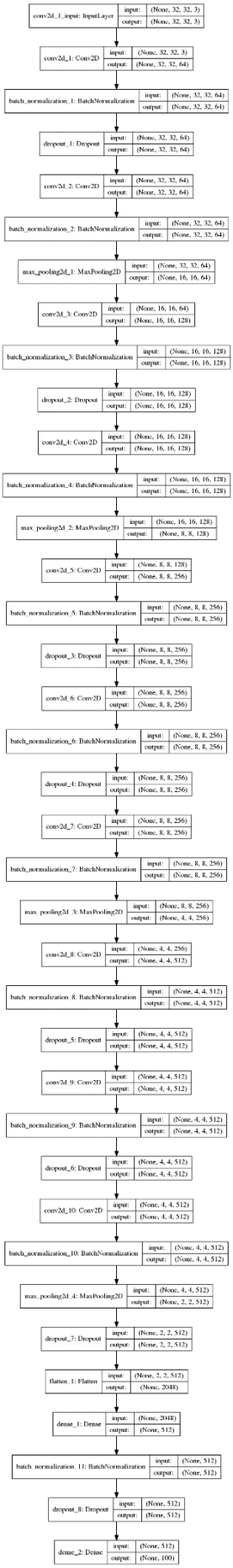
Accuracy: 47.94%

通过上面的实验结果，我们可以明显发现：过拟合现象得到了明显的解决，测试集和训练集的训练效果已没有显著的差异了。同时，我们发现 100 轮的迭代，loss 基本上已经收敛了，但下降趋势还存在，可以继续增加迭代轮数。我们将迭代轮数增加到 150 轮后，效果再次提升，过拟合现象得到了解决。

4. 网络结构的尝试

做完网络参数的调整实验以及解决了过拟合问题后，我们又尝试对网络结构进行修改。由于计算资源有限，我们只能依靠一台游戏本进行训练，所以我们没有尝试实现复杂的深度残差网络——ResNet，而是借鉴了有一定深度，但训练时间又不会过长的 VGG16 模型。

参考 VGG16 模型，我们实现了一个网络层数更深，参数更多，更复杂的网络。网络的结构如下图所示：


conv2d_1_input: InputLayer
input: (None, 32, 32, 3)
output: (None, 32, 32, 3)
conv2d_1: Conv2D
input: (None, 32, 32, 3)
output: (None, 32, 32, 64)
batch_normalization_1: BatchNormalization
input: (None, 32, 32, 64)
output: (None, 32, 32, 64)
dropout_1: Dropout
input: (None, 32, 32, 64)
output: (None, 32, 32, 64)
conv2d_2: Conv2D
input: (None, 32, 32, 64)
output: (None, 32, 32, 64)
batch_normalization_2: BatchNormalization
input: (None, 32, 32, 64)
output: (None, 32, 32, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 32, 32, 64)
output: (None, 16, 16, 64)
conv2d_3: Conv2D
input: (None, 16, 16, 64)
output: (None, 16, 16, 128)
batch_normalization_3: BatchNormalization
input: (None, 16, 16, 128)
output: (None, 16, 16, 128)
dropout_2: Dropout
input: (None, 16, 16, 128)
output: (None, 16, 16, 128)
conv2d_4: Conv2D
input: (None, 16, 16, 128)
output: (None, 16, 16, 128)
batch_normalization_4: BatchNormalization
input: (None, 16, 16, 128)
output: (None, 16, 16, 128)
max_pooling2d_2: MaxPooling2D
input: (None, 16, 16, 128)
output: (None, 8, 8, 128)
conv2d_5: Conv2D
input: (None, 8, 8, 128)
output: (None, 8, 8, 256)
batch_normalization_5: BatchNormalization
input: (None, 8, 8, 256)
output: (None, 8, 8, 256)
dropout_3: Dropout
input: (None, 8, 8, 256)
output: (None, 8, 8, 256)
conv2d_6: Conv2D
input: (None, 8, 8, 256)
output: (None, 8, 8, 256)
batch_normalization_6: BatchNormalization
input: (None, 8, 8, 256)
output: (None, 8, 8, 256)
dropout_4: Dropout
input: (None, 8, 8, 256)
output: (None, 8, 8, 256)
conv2d_7: Conv2D
input: (None, 8, 8, 256)
output: (None, 8, 8, 256)
batch_normalization_7: BatchNormalization
input: (None, 8, 8, 256)
output: (None, 8, 8, 256)
max_pooling2d_3: MaxPooling2D
input: (None, 8, 8, 256)
output: (None, 4, 4, 256)
conv2d_8: Conv2D
input: (None, 4, 4, 256)
output: (None, 4, 4, 512)
batch_normalization_8: BatchNormalization
input: (None, 4, 4, 512)
output: (None, 4, 4, 512)
dropout_5: Dropout
input: (None, 4, 4, 512)
output: (None, 4, 4, 512)
conv2d_9: Conv2D
input: (None, 4, 4, 512)
output: (None, 4, 4, 512)
batch_normalization_9: BatchNormalization
input: (None, 4, 4, 512)
output: (None, 4, 4, 512)
dropout_6: Dropout
input: (None, 4, 4, 512)
output: (None, 4, 4, 512)
conv2d_10: Conv2D
input: (None, 4, 4, 512)
output: (None, 4, 4, 512)
batch_normalization_10: BatchNormalization
input: (None, 4, 4, 512)
output: (None, 4, 4, 512)
max_pooling2d_4: MaxPooling2D
input: (None, 4, 4, 512)
output: (None, 2, 2, 512)
dropout_7: Dropout
input: (None, 2, 2, 512)
output: (None, 2, 2, 512)
flatten_1: Flatten
input: (None, 2, 2, 512)
output: (None, 2048)
dense_1: Dense
input: (None, 2048)
output: (None, 512)
batch_normalization_11: BatchNormalization
input: (None, 512)
output: (None, 512)
dropout_8: Dropout
input: (None, 512)
output: (None, 512)
dense_2: Dense
input: (None, 512)
output: (None, 100)

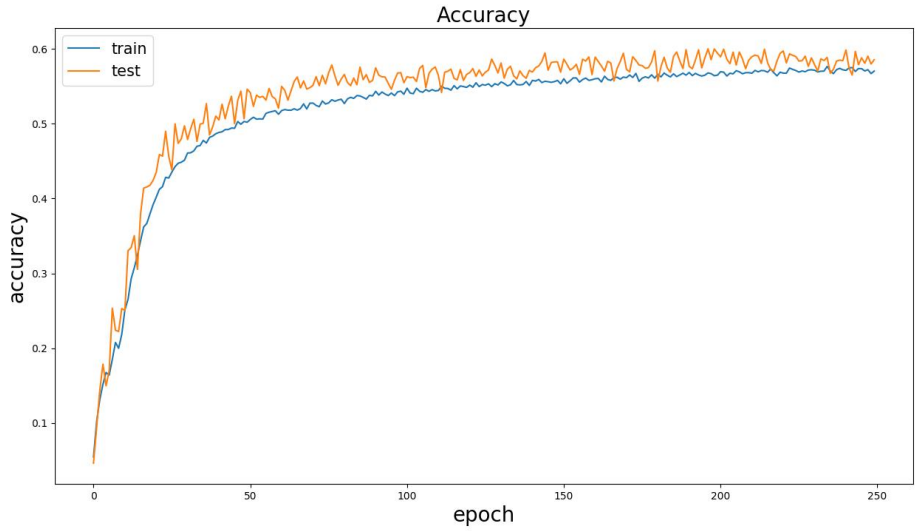
在这个网络结构中，我们使用了更多的卷积核，卷积层和池化层的组合使用的次数也更多，这导致训练模型更加复杂，需要训练的参数更多，但是由于我们使用了一些防止过拟合的办法，最后总体来看，效果还是十分不错的。

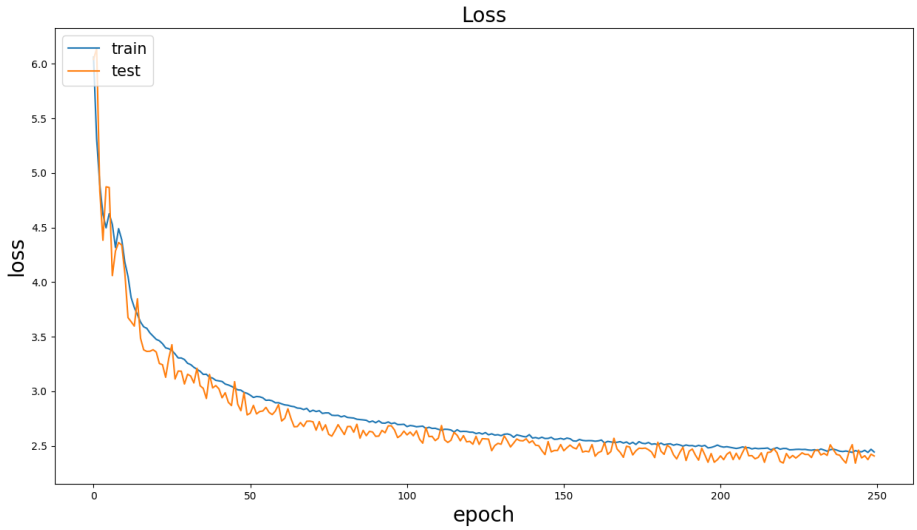
我们还使用了 Batch Normalization 技术——在网络中添加了多个归一化层，对前一层的输出进行归一化操作。通过引入可学习重构参数，使得我们的网络可以学习恢复出原始网络所要学习的特征分布。

使用 BN 层可以有效解决两个问题：一是加快网络的学习速率；二是梯度弥散和梯度爆炸问题。在深度网络中，如果网络的激活输出很大，其梯度就很小，学习速率就很慢。由于链式求导的原因，第一层的学习速率就慢，最后一层只需求一次导，梯度就大，学习速率就快。这就会造成在一个很大的深度网络中，浅层基本不学习，权值变化小，后面几层一直在学习。最终后面几层基本可以表示整个网络，这样我们的深度网络就失去了深度的意义。使用了 BN 层之后，会使得原本会减小的激活输出的规模变大，从而解决梯度弥散的问题。

另外，我们发现：尽管 SGD 优化器的收敛速度较慢，但是通过手动调节学习率（手动减半学习率）的方法，可以达到其他自适应学习率的优化算法所达不到的效果。

实验结果如下图所示：





最优的实验结果为：

Loss: 1.9641

Accuracy: 69.96%

实验总结：

通过这个项目，我们不仅对卷积神经网络的相关内容有了初步的了解，更为重要的是锻炼了工程项目能力以及团队协作、交流的能力，另外在进行实验的过程中，大家还阅读了大量相关的资料和文献，对于文献的搜集、阅读能力也有明显的提高。

总之，这次项目带给我们的收获很大，让我们充分认识到卷积神经网络作为近几年较为火热的研究方向，其内部的很多问题还等待着我们去探索发现。

附：具体代码和训练模型以及更多的实验结果见：

https://github.com/GGSuchao/Cifar100_CNN