基于CNN神经网络的分类任务

基于Cifar-10数据集

计算机1604 王殊 1611640413

**目录**

[数据集描述 1](#_Toc531636316)

[数据格式 1](#_Toc531636317)

[数据处理 1](#_Toc531636318)

[VGG19神经网络 1](#_Toc531636319)

[超参数 1](#_Toc531636320)

[网络结构 2](#_Toc531636321)

[损失函数 3](#_Toc531636322)

[训练过程 3](#_Toc531636323)

[卷积层可视化 4](#_Toc531636324)

[卷积核可视化 6](#_Toc531636325)

[测试集准确率 8](#_Toc531636326)

[参数输出 9](#_Toc531636327)

[整体结构 9](#_Toc531636328)

[Resnet20v2 网络 11](#_Toc531636329)

[结构（局部） 13](#_Toc531636330)

[参数输出 13](#_Toc531636331)

[测试集准确率 15](#_Toc531636332)

[如何运行 15](#_Toc531636333)

[总结 15](#_Toc531636334)

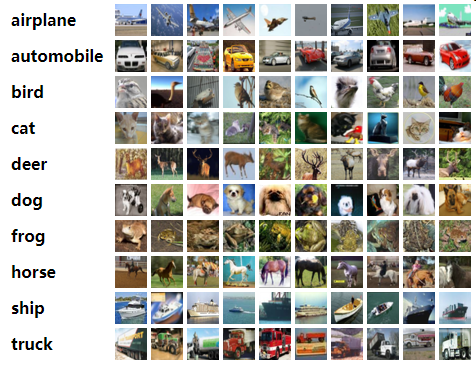
[参考 15](#_Toc531636335)

项目地址：

https://github.com/ddayzzz/machine\_learning/tree/master/homework/cnn\_cifar

# 数据集描述

Cifar-10 的数据集包括训练集和测试集，样本的大小分别是50000和10000。样本被归为10类，分别是：



数据格式

1. 导入的数据格式：Python 使用 Pickle 序列化的数据。
2. 图像数据包括是 32x32 的 uint8 的图像数据。是一个10000x3072的uint8矩阵。每一行的前1024个uint8 长度是R分量，2048开始的是G分量和3072的是B分量。
3. 使用行优先顺序存储。

数据处理

1. Tensorflow 默认的卷积层的数据输入格式是4-D张量维度信息是：

[样本数量，矩阵高度，矩阵长度，通道数量]

所以需要在原数据的基础上进行转置操作。

1. 官方的样本数低于送入到神经网络的样本总数，所以需要对原数据增广处理同时对每个样本进行长度和高度缩放。

# VGG19神经网络

参考了基于VGG19的Cifar10修改版本的网络模型。

超参数

|  |  |  |
| --- | --- | --- |
| 参数名 | 值 | 意义 |
| conv\_strides | [1,1,1,1] | 过滤器的在输入样本的各个分量的移动步数 |
| pool\_kernel\_size | [1,3,3,1] | 最大池化层的过滤器大小 |
| pool\_strides | [1,2,2,1] | 最大池化层的过滤器移动的步长 |
| keep\_prob | 0.5 | L2正则化保留的比例 |
| regularize\_ratio | 0.0002 | 对给定过滤器的L2正则化器的缩放比例(将乘以过滤器) |
| batch\_size | 128 | 送入神经网络的一批次样本数量 |
| init\_learning\_rate | 0.001 | 初始学习率 |
| learning\_rate\_decay\_per\_epoch | 8 | 每8次epoch将学习率乘以learning\_rate\_decay\_rate |
| learning\_rate\_decay\_rate | 0.5 | 学习率衰减因子 |

网络结构

针对于Cifar-10数据，不同于VGG19默认的以244x244像素的图片。需要做调整。

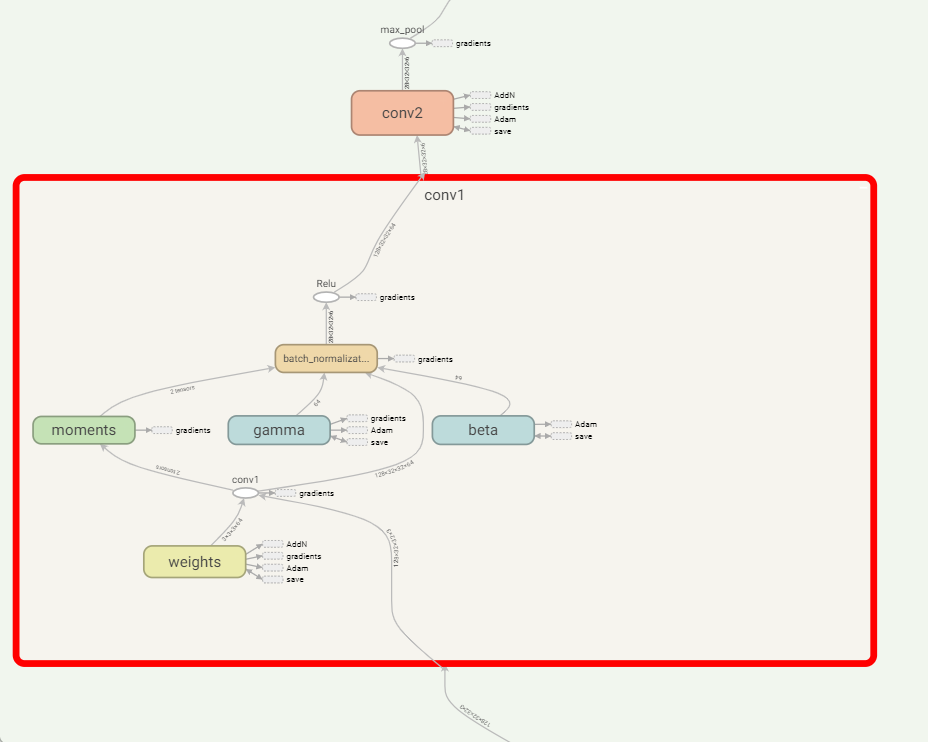
1. 输入的数据格式

|  |  |
| --- | --- |
| 名称 | 维度 |
| train\_images\_input | [128 32 32 3] |

1. 采用的结构

VGG19 网络中主要是由5个Block 结构构成：

* Block 结构：



每一个 Block 中包含有若干个卷积层以及最外层卷积层的最大池化层。具体的Block 配置：

采用的结构如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Block | 层类别 | 输出大小 | 配置（大小，输出通道） | 激活函数 |
| Block1 | Conv1 | 32x32 | 大小：3x3，输出通道：64 | ReLU |
| Conv2 | 32x32 | 大小：3x3，输出通道：64 | ReLU |
| MaxPool | 16x16 | 步长大小：2x2，填充0保持信息不变 |  |
| Block2 | Conv1 | 16x16 | 大小：3x3，输出通道：128 | ReLU |
|  | Conv2 | 16x16 | 大小：3x3，输出通道：128 | ReLU |
|  | MaxPool | 8x8 | 步长大小：2x2，填充0保持信息不变 |  |
| Block3 | Conv1 | 8x8 | 大小：3x3，输出通道：256 | ReLU |
|  | Conv2 | 8x8 | 大小：3x3，输出通道：256 | ReLU |
|  | Conv3 | 8x8 | 大小：3x3，输出通道：256 | ReLU |
|  | Conv4 | 8x8 | 大小：3x3，输出通道：256 | ReLU |
|  | MaxPool | 4x4 | 步长大小：2x2，填充0保持信息不变 |  |
| Block4 | Conv1 | 4x4 | 大小：3x3，输出通道：512 | ReLU |
|  | Conv2 | 4x4 | 大小：3x3，输出通道：512 | ReLU |
|  | Conv3 | 4x4 | 大小：3x3，输出通道：512 | ReLU |
|  | Conv4 | 4x4 | 大小：3x3，输出通道：512 | ReLU |
|  | MaxPool | 2x2 | 步长大小：2x2，填充0保持信息不变 |  |
| Block5 | Conv1 | 2x2 | 大小：3x3，输出通道：512 | ReLU |
|  | Conv2 | 2x2 | 大小：3x3，输出通道：512 | ReLU |
|  | Conv3 | 2x2 | 大小：3x3，输出通道：512 | ReLU |
|  | Conv4 | 2x2 | 大小：3x3，输出通道：512 | ReLU |
| Flatten |  | 1x1 | 大小：2x2x512 |  |
| FC1 |  | 1x1 | 大小：2048x4096 | ReLU |
|  | Dropout |  | 保留的比例：0.5 |  |
| FC2 |  | 1x1 | 大小：4096x4096 | ReLU |
|  | Dropout |  | 保留的比例：0.5 |  |
| Softmax |  | 1x1 | 大小：4096x10 | Softmax |

损失函数

Softmax 输出的是10类的可能的概率，通过将输入的样本的标签书输入可以定义Softmax交叉熵，损失函数就是Softmax交叉熵的损失。损失函数使用 Adam 优化器。

训练过程

首先是需要统计真实的图片标签与Softmax的输出 logits 的误差值加入到tf可损失集合中。这个集合包括了使用L2正则化的各层的卷积核的过滤器防止过拟合的张量。

为了获得较高的在测试集的准确率，每进行一次 epoch 就将当前的网络参数保存在外部设备。

训练的时候使用了:

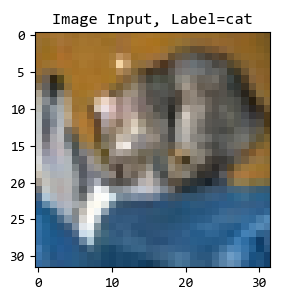
1. 学习率衰减

为了防止在最后的训练中由于过大的学习率导致损失函数的震荡问题，需要使学习率随着训练次数的进行衰减。

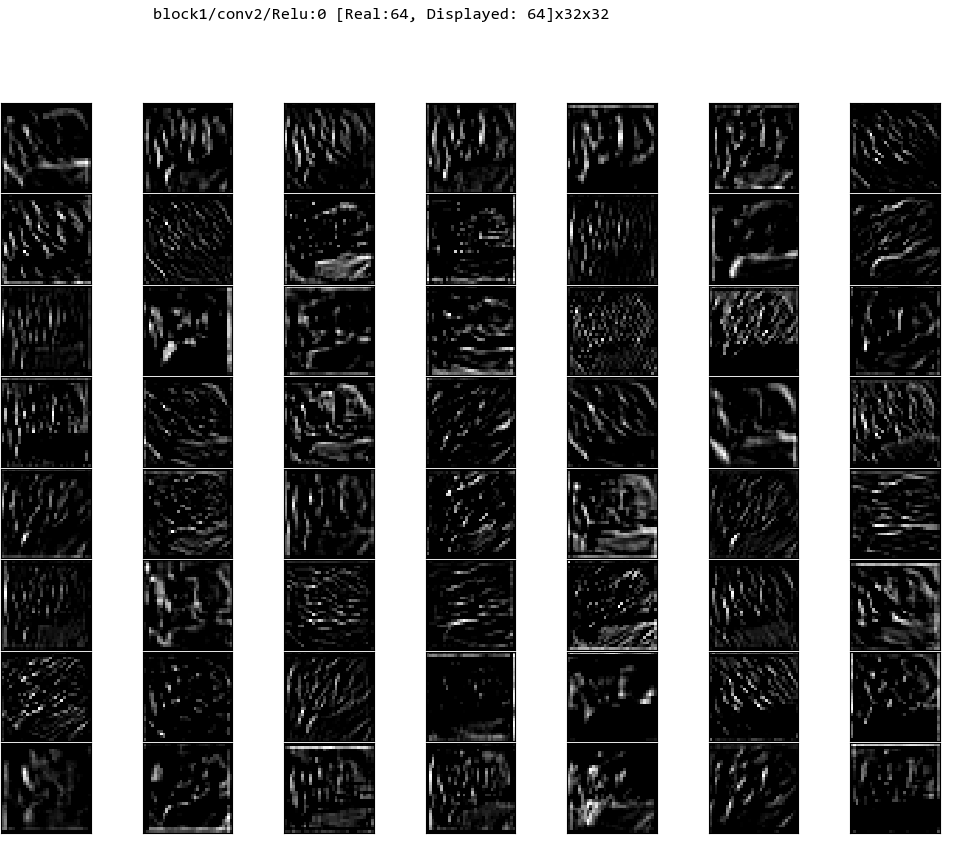
卷积层可视化

为了更好的显示图像，目前最多显示卷积核的前16个通道：

1. 原始图像：



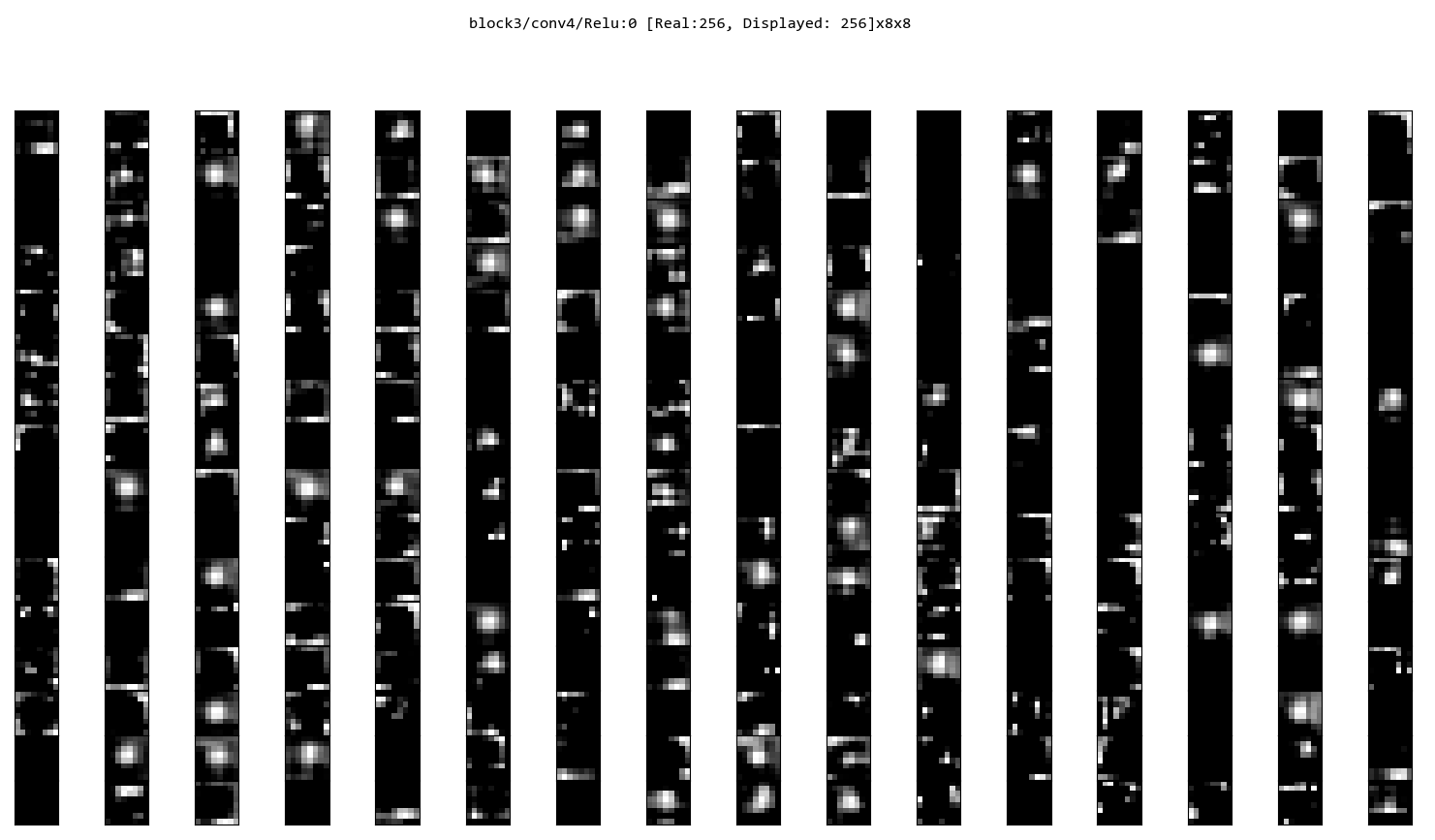
1. Block1/Conv2/Relu：



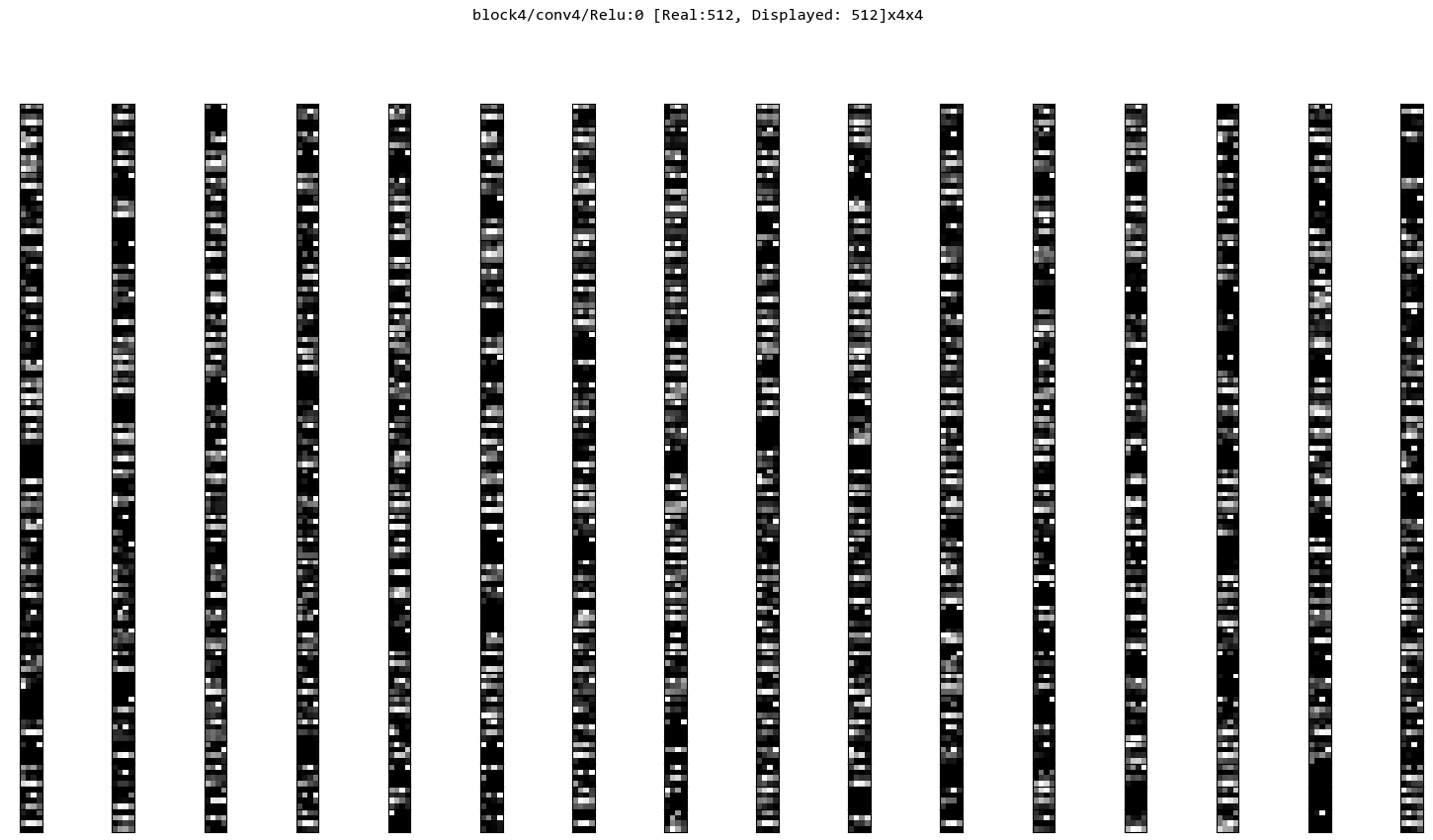
1. Block2/Conv2/Relu：



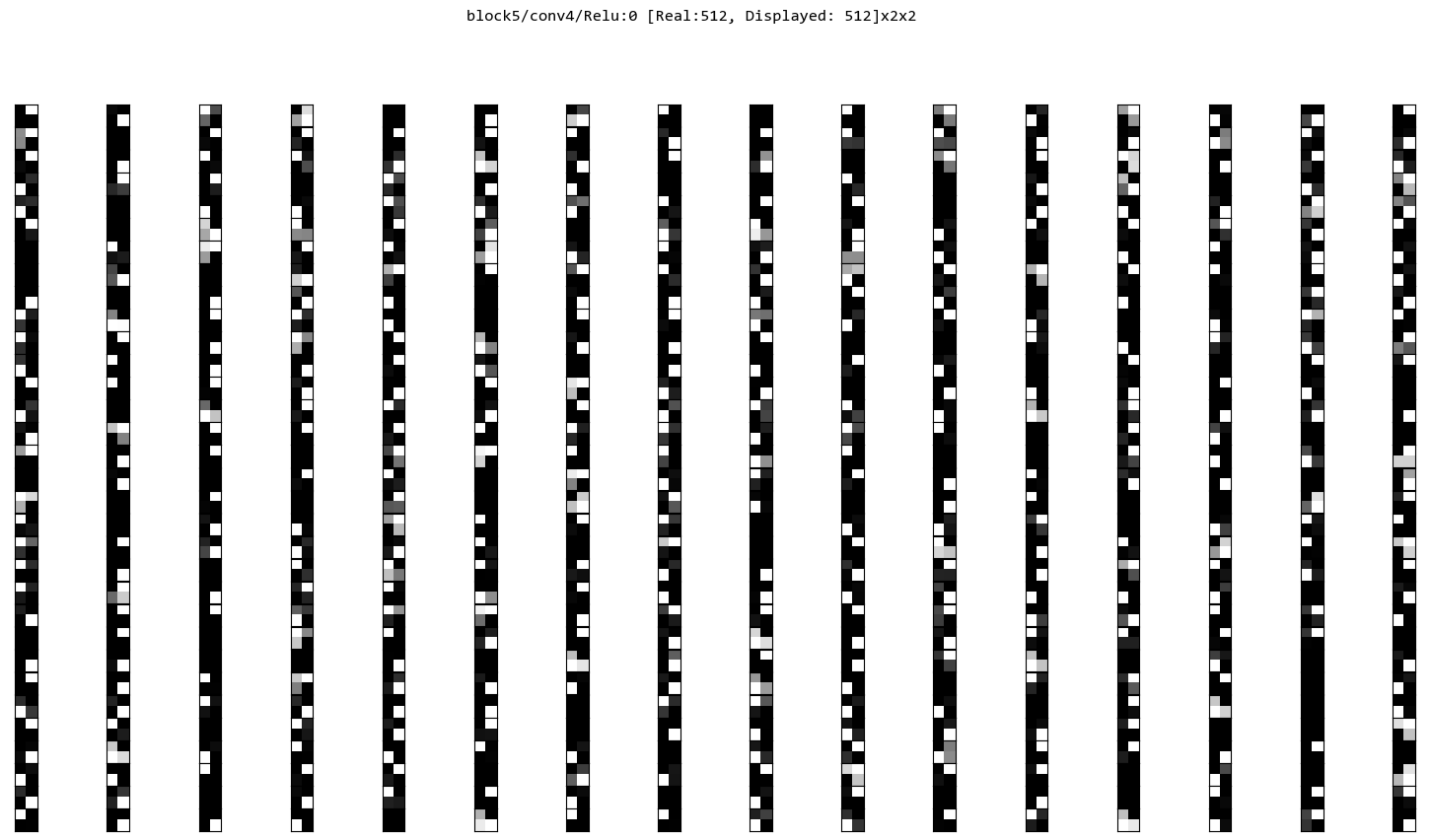
1. Block3/Conv4/Relu：



1. Block4/Conv4/Relu：



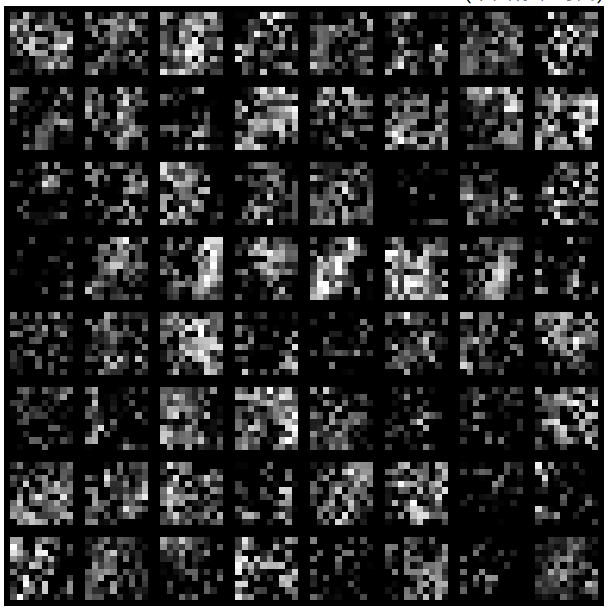
1. Block5/Conv4/Relu：



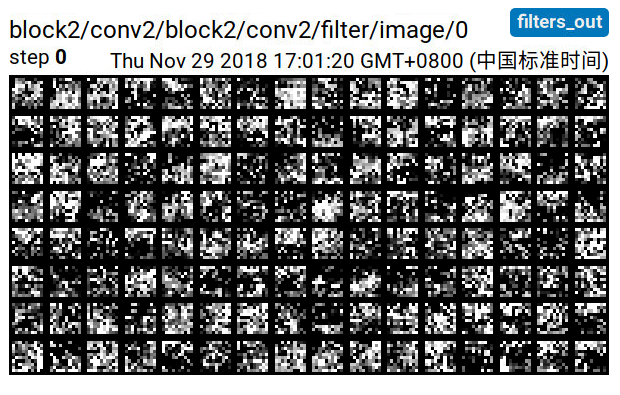
卷积核可视化

为了方便可视化，将卷积核的大小改为 7x7 以方便观察。同时调大了对比度。这里仅仅显示第一个卷积核通道。

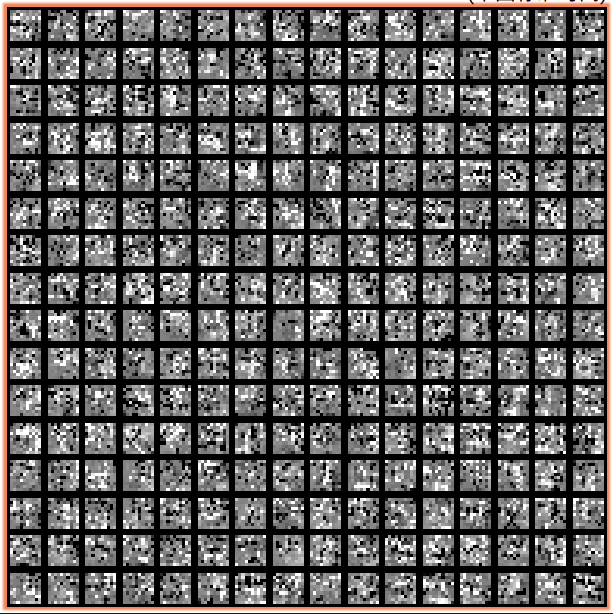
1. Block1/Conv2/filter



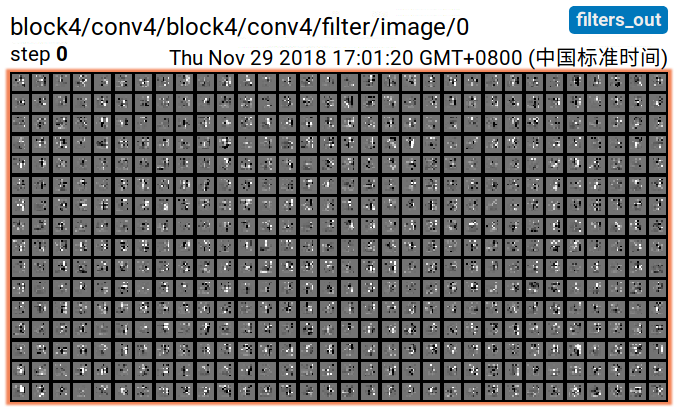
1. Block2/Conv2/filter



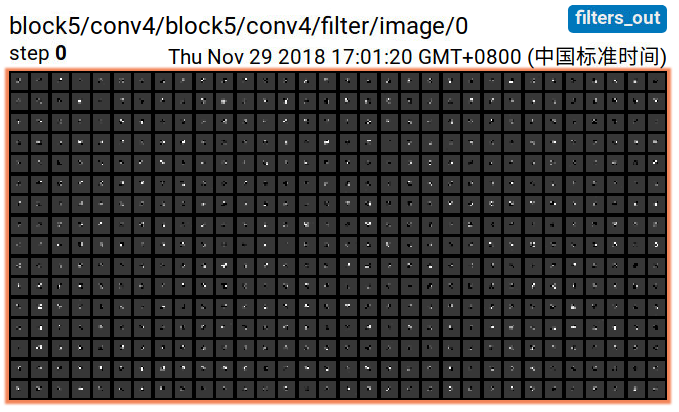
1. Block3/Conv4/filter



1. Block4/Conv4/filter

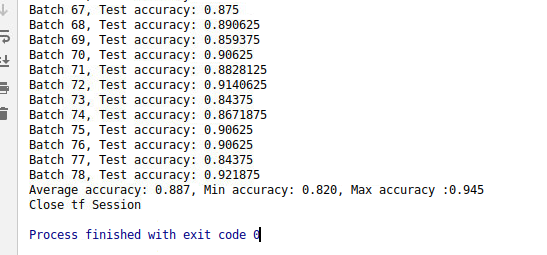


1. Block5/Conv4/filter



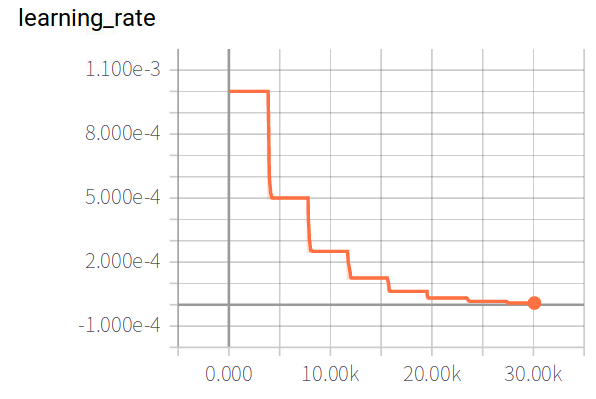
测试集准确率

测试采用的是128样本/batch 的图像数据。检查点的全局步数为65130,总共184 epoch。测试集的准确率信息：

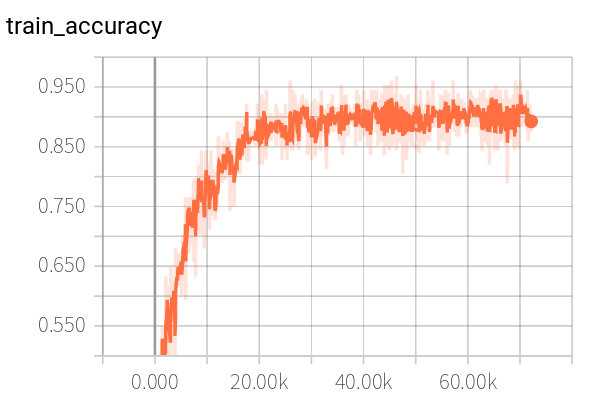


参数输出

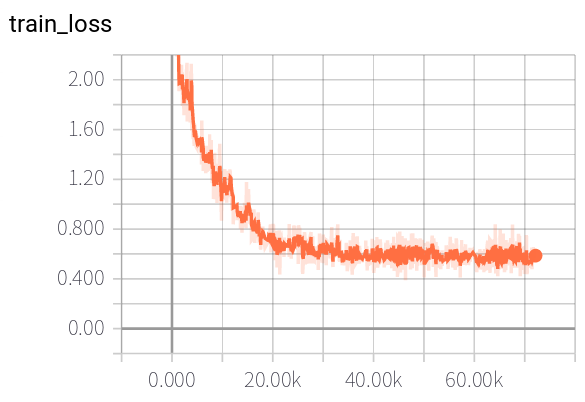
1. 学习率变化



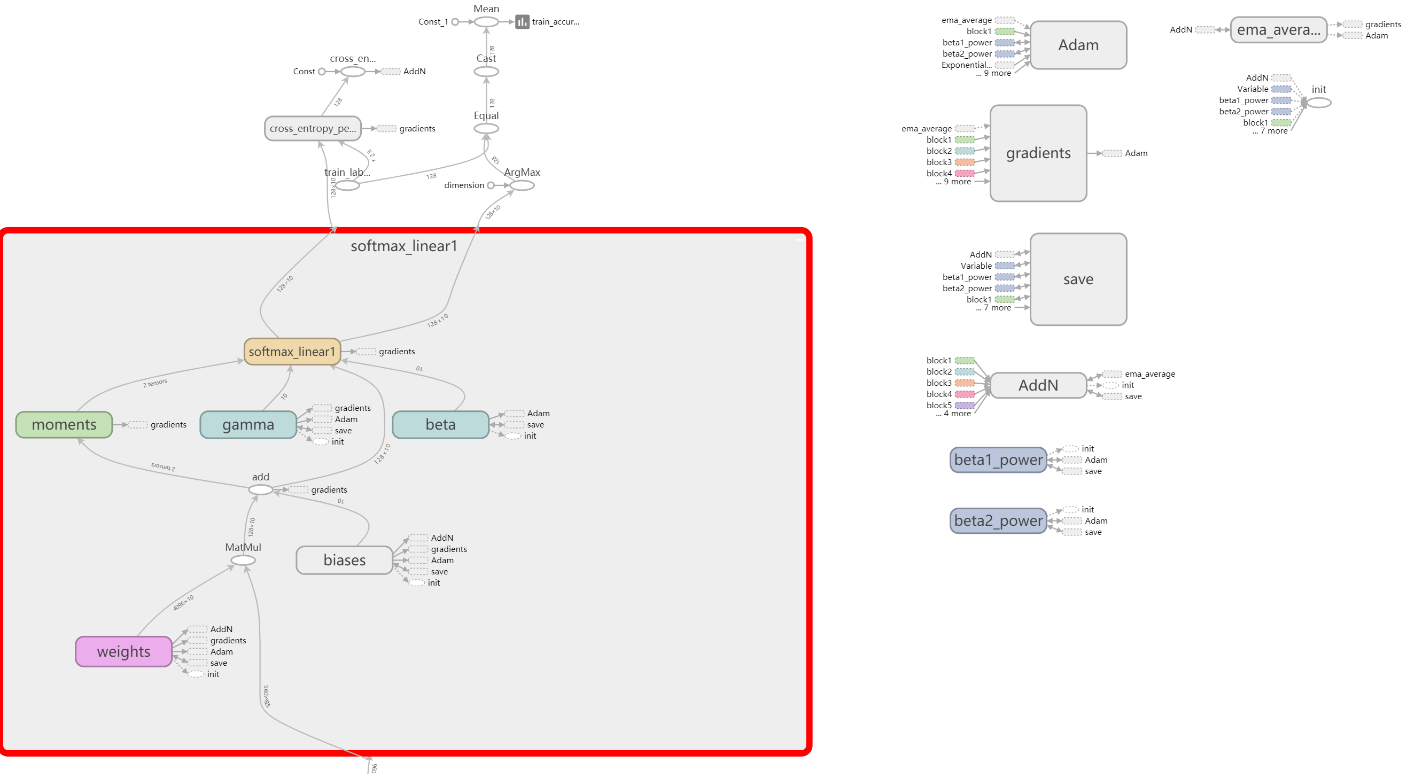
1. 模型在训练集的准确率

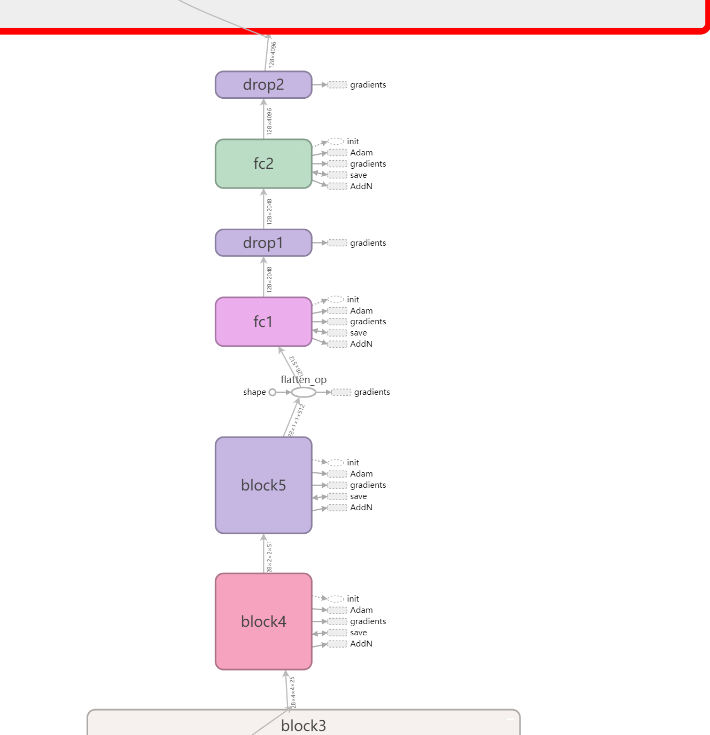


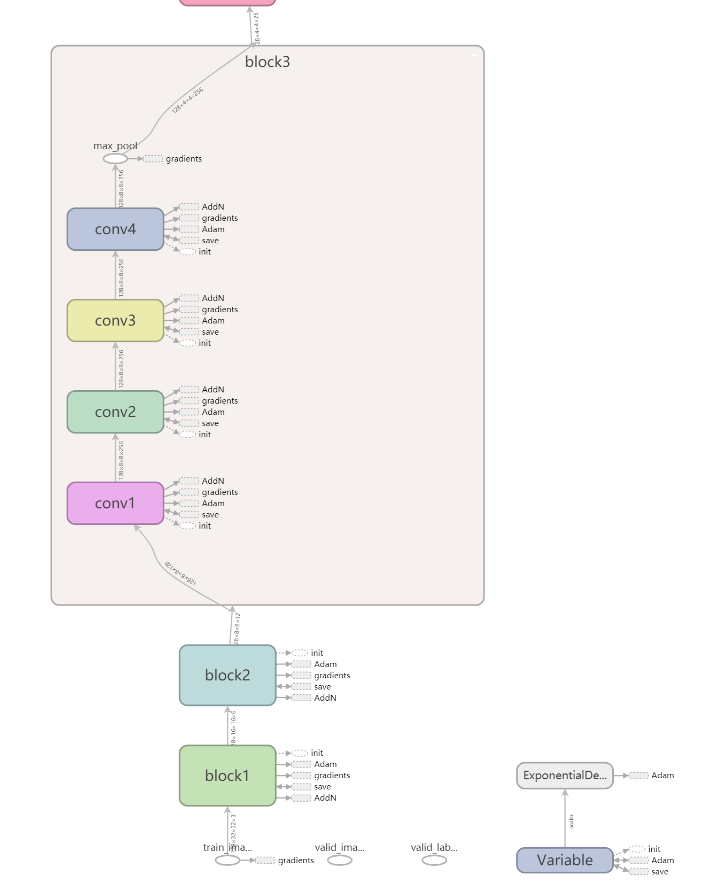
1. 训练集上的损失函数



整体结构







# Resnet20v2 网络

使用残差网络可以避免因为CNN卷积层过多而导致的训练误差上升(并非完全由过拟合而导致,过拟合会在训练集上保持很好成绩)的问题[1]。

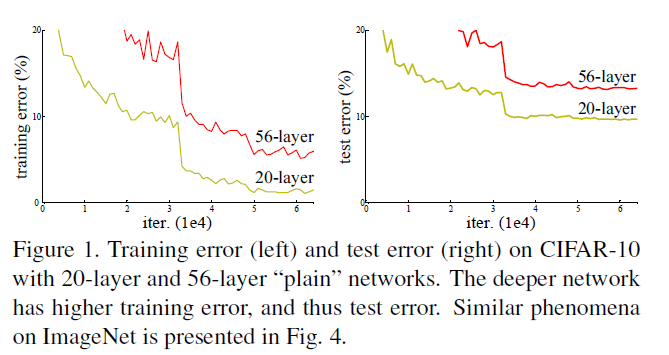


图1:左图为使用20层核56层的网络在Cifar10数据集上的训练误差

论文[1]中假设深度网络中某隐藏层为, 如果可以假设多个非线性组合可以近似于一个复杂的函数，那么同样可以假设隐含层的残差近似于某个复杂函数，也就是将隐藏层表示为。其中表示最优映射。在训练集上，深层网络不应该表现得比千层网路差，因为只需要深层网络多的那些层做恒等映射就简化为了浅层网络。所以从学习恒等映射这点出发，考虑到网络要学习一个的映射比学习的映射更难，所以可以把网络结构设计成，这样就即完成了恒等映射的学习，又降低了学习难度。其中是残差结构的输入，F是该层网络学习到的映射，H是整个残差结构的输出。

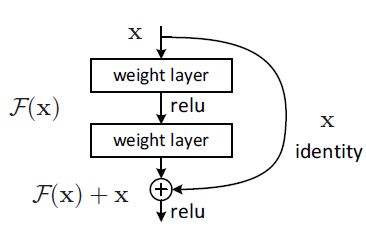


图2: 残差的学习过程

同时残差网络的特点是：

1. 没有使用Dropout， 而使用Batch normalization 从而让网络更容易训练。由于基于BP算法的网络在计算梯度的时候使用的是连乘（链式法则时使用），随着网络层数的增加就会导致在最初的几层的梯度急剧减小，所以在使用梯度下降算法的时候，最初几层的权重变化缓慢。也就是梯度弥散的问题。
2. 层数较高时，减少了3x3的卷积核的个数，并用1x1的卷积核控制了3x3卷积输如输出的特征图数量。这种结构称为“瓶颈结构”。目的时为了减少模型的复杂的度。

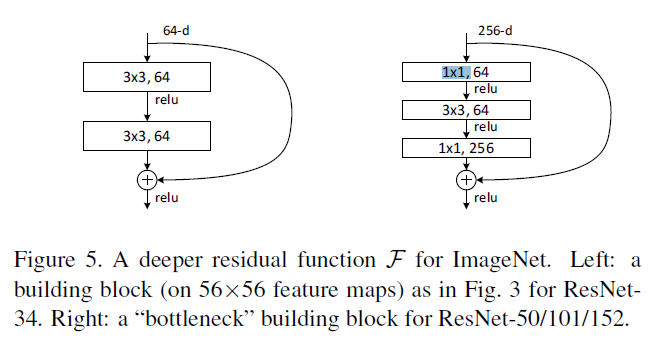


图3：右图，“瓶颈”块

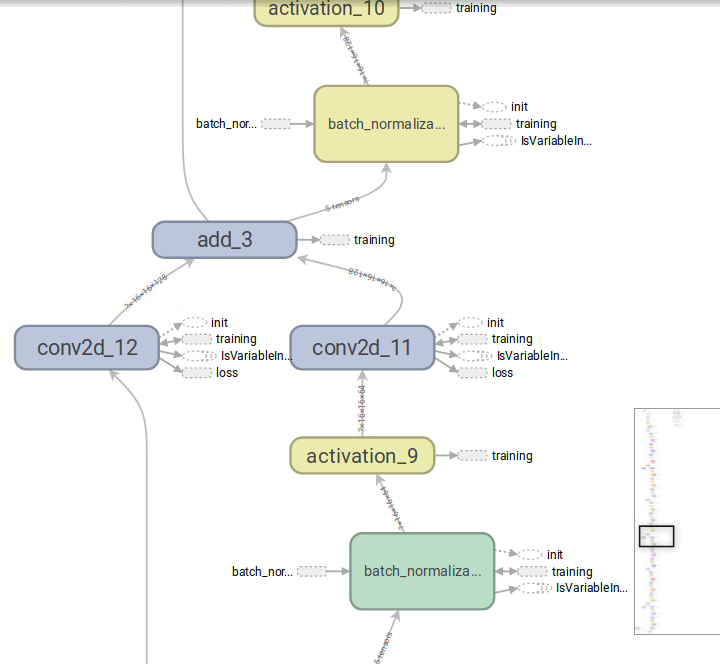
1. 快捷方式(shortcut):在图1中定义了其输出可以被定义为：

函数表示需要学习的残差映射。可以通过快捷方式来实现也就是图2的过程。例如对于一个两层的网络的输出：

表示的是一个激活函数。可以看到F与x的维度必须要一致，当不一致的时候需要在x上进行线性映射()的映射：

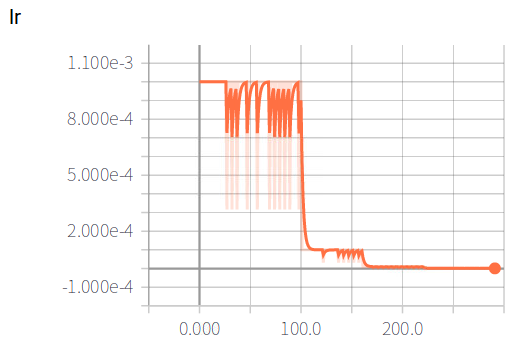
1. 网络的参数较少，使用VGG19(输入的图片大小为224x224而并非Cifar10的32x32)的浮点操作的计算量为19.6亿，而34层残差为3.6亿。

结构（局部）

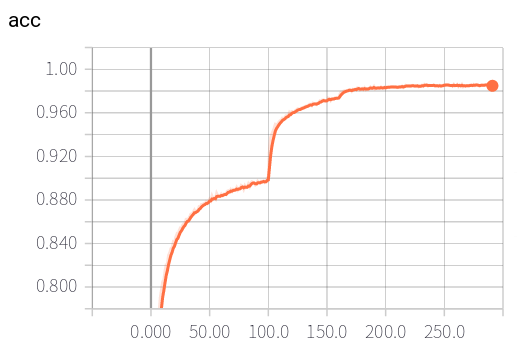


参数输出

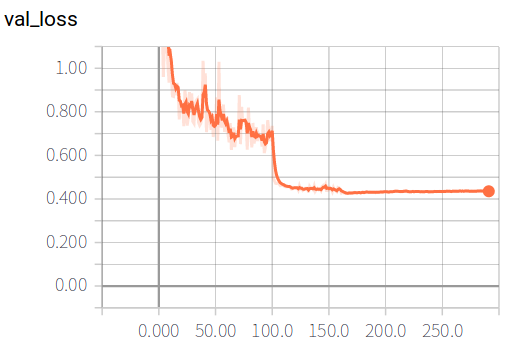
1. 学习率变化



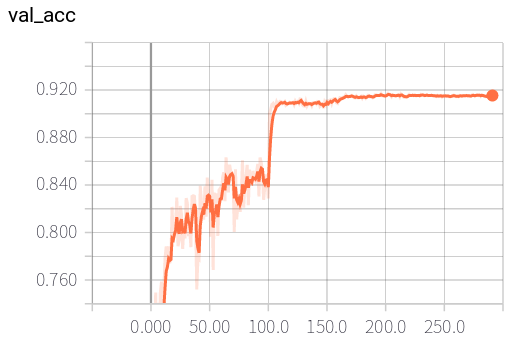
1. 模型在训练集的准确率



1. 验证集（采用测试集）上的损失函数

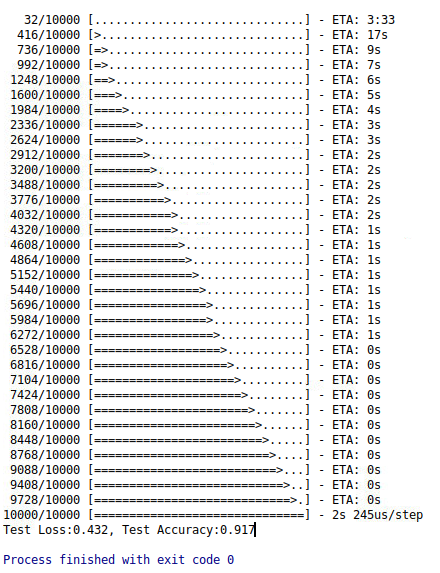


1. 验证集（采用测试集）上的准确率



## 测试集准确率

测试集准确率，在91.7%，输出结果：



# 如何运行

详情参考代码目录中的 README.md (Markdown)

# 总结

1. 深度学习的后期部分除去修改模型,其余的部分主要是调参以达到最优的测试集准确率。
2. 卷积层输出的内容的意义需要仔细的理解。理解卷积核究竟学到了什么。
3. 使用学习率衰减（随着epoch的增加）可以使模型在训练集的上的表现提高。

# 参考

1. K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, arxiv.org, vol. 7, no. 3, pp. 171–180, 2015.