CIFAR10 图片分类报告

内容提要

- 背景
- 模型和方法
- 实验分析与展示
- 总结

背景 - CIFAR10

- CIFAR10 是一个带标签的数据集
- 是一个图片分类任务，一共有 10 种不同的物品
  - 'plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse', 'ship', 'truck'
- 每一张图片大小为32*32，是彩色图片，3 通道(RGB)
- 训练集一共 50000 张图片，测试集 10000 张
- 数据集及网站

  http://www.cs.toronto.edu/~kriz/cifar.html

背景 - CIFAR10

· CIFAR10 现在的成绩

Result	Method	Venue	Details
96.53%	Fractional Max-Pooling	arXiv 2015	Details
95.59%	Striving for Simplicity: The All Convolutional Net	ICLR 2015	Details
94.16%	All you need is a good init	ICLR 2016	Details
94%	Lessons learned from manually classifying CIFAR-10	unpublished 2011	Details
93.95%	Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree	AISTATS 2016	Details
93.72%	Spatially-sparse convolutional neural networks	arXiv 2014	
93.63%	Scalable Bayesian Optimization Using Deep Neural Networks	ICML 2015	
93.57%	Deep Residual Learning for Image Recognition	arXiv 2015	Details
93.45%	Fast and Accurate Deep Network Learning by Exponential Linear Units	arXiv 2015	Details
93.34%	Universum Prescription: Regularization using Unlabeled Data	arXiv 2015	
93.25%	Batch-normalized Maxout Network in Network	arXiv 2015	Details

http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130

· 背景

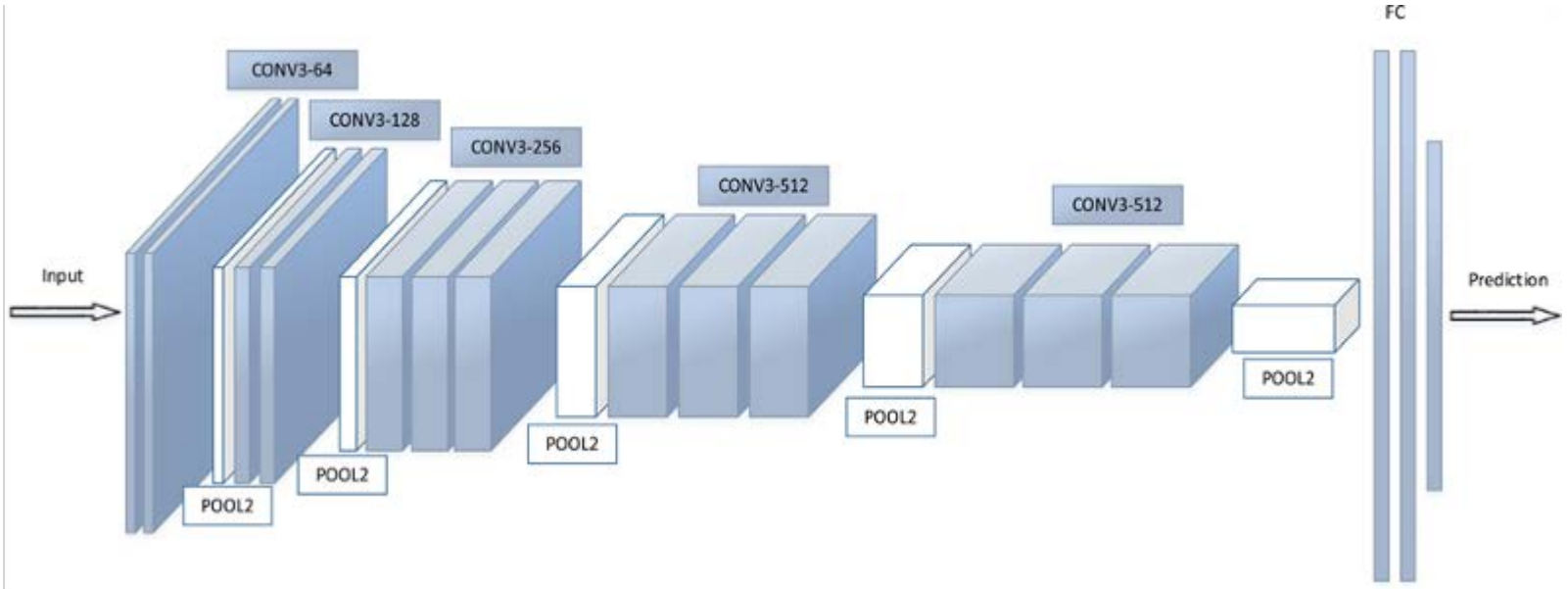
· 模型和方法

· 实验分析与展示

· 总结

模型和方法 - 模型 - VGG16

- VGG - Visual Geometry Group(University of Oxford)
- VGG16
  - 13卷积层 + 3全连接层



模型和方法 – 模型 – VGG16

- 初始模型
  - 数据初始化
    - 归一化 [-1,1]
  - 每一个神经网络节点
    - 归一化 [-1,1]
    - 激活函数 Relu
  - 损失函数：交叉熵
  - 优化器：梯度下降法
    - 学习率：0.001

模型和方法 – 模型 – VGG16

- 初始模型
  - 卷积层 C(inChannels, outChannels)
    - kernelSize = 3*3，stride=1，padding=1
  - 池化层 M
    - maxpool，size=2*2，stride=2
  - 全连接层 FC(inputSize,outputSize)
    - 由于图片较小全连接层只保留一层

模型和方法 – 模型 – VGG16

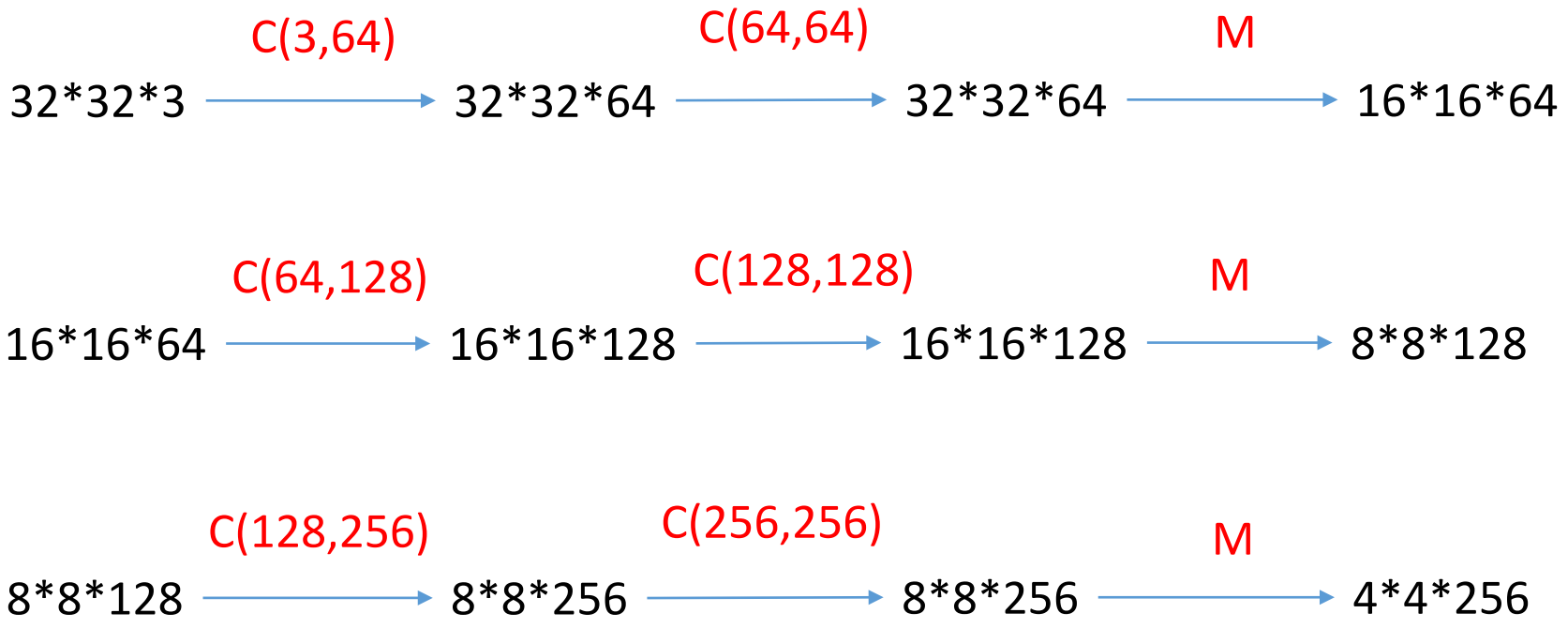
- 输入：32*32*3

1. 3*32*32 -> 卷积核个数:64
2. 64*32*32 -> 卷积核个数:64
3. 64*32*32 -> maxpool:2*2
4. 64*16*16 -> 卷积核个数:128
5. 128*16*16 -> 卷积核个数:128
6. 128*16*16 -> maxpool:2*2
7. 128*8*8 -> 卷积核个数:256
8. 256*8*8 -> 卷积核个数:256
9. 256*8*8 -> 卷积核个数:256
#10. 256*8*8 -> maxpool:2*2

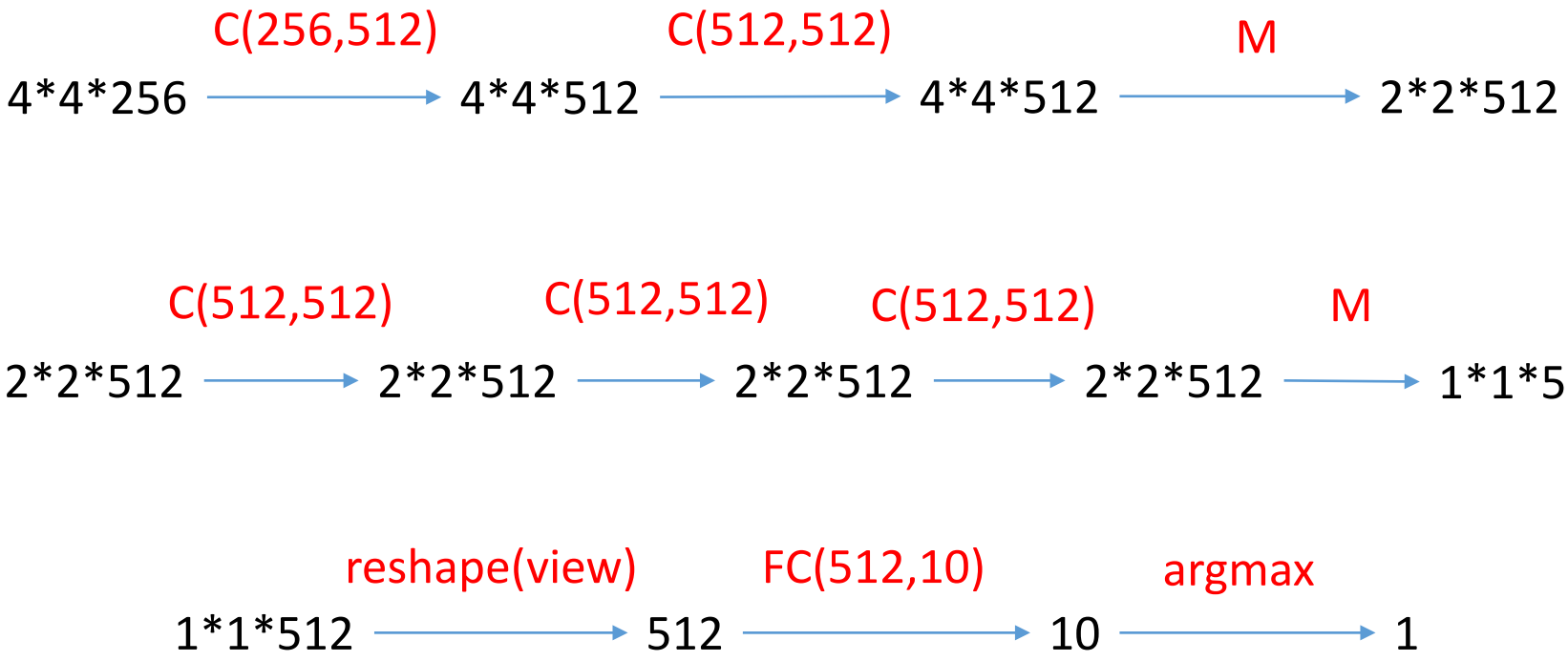
#11. 256*4*4 -> 卷积核个数:512
#12. 512*4*4 -> 卷积核个数:512
#13. 512*4*4 -> 卷积核个数:512
#14. 512*4*4 -> maxpool:2*2
#15. 512*2*2 -> 卷积核个数:512
#16. 512*2*2 -> 卷积核个数:512
#17. 512*2*2 -> 卷积核个数:512
#18. 512*2*2 -> maxpool:2*2
#19. 512*1*1 -> Linear:512->10

模型和方法 – 模型 – VGG16

- 输入：32*32*3



模型和方法 – 模型 – VGG16



模型和方法 – 方法 – pytorch

- 数据初始化：读入 + 正则化

```
# Normaize 正则化(预处理)
# torchvision.transforms.Normalize(mean, std, inplace=False)
# input[channel] = (input[channel] - mean[channel]) / std[channel]
# 如下示例,输入原本为 [0,1] -> [-1,1]
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# 训练集
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4, shuffle=True, num_workers=2)

# 测试集
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4, shuffle=False, num_workers=2)
```

- 注意每一组有 4 个数据

模型和方法 – 方法 – pytorch

- 卷积层 + 池化层

```
# 网络层
self.layers = nn.Sequential(
        # CCP
        nn.Conv2d(3, 64, 3, stride=1, padding=1),
        # 正则化
        nn.BatchNorm2d(64),
        # 激活函数
        nn.ReLU(inplace=True),
        nn.Conv2d(64, 64, 3, stride=1, padding=1),
        nn.BatchNorm2d(64),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2),
        # CCP
        nn.Conv2d(64, 128, 3, stride=1, padding=1),
        nn.BatchNorm2d(128),
        nn.ReLU(inplace=True),
        nn.Conv2d(128,128, 3, stride=1, padding=1),
        nn.BatchNorm2d(128),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2),
        # CCCP
        nn.Conv2d(128,256, 3, stride=1, padding=1),
        nn.BatchNorm2d(256),
        nn.ReLU(inplace=True),
        nn.Conv2d(256,256, 3, stride=1, padding=1),
        nn.BatchNorm2d(256),
        nn.ReLU(inplace=True),
        nn.Conv2d(256,256, 3, stride=1, padding=1),
        nn.BatchNorm2d(256),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2),
        # CCCP
        nn.Conv2d(256,512, 3, stride=1, padding=1),
        nn.BatchNorm2d(512),
        nn.ReLU(inplace=True),
        nn.Conv2d(512,512, 3, stride=1, padding=1),
        nn.BatchNorm2d(512),
        nn.ReLU(inplace=True),
        nn.Conv2d(512,512, 3, stride=1, padding=1),
        nn.BatchNorm2d(512),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2),
        # CCCP
        nn.Conv2d(512,512, 3, stride=1, padding=1),
        nn.BatchNorm2d(512),
        nn.ReLU(inplace=True),
        nn.Conv2d(512,512, 3, stride=1, padding=1),
        nn.BatchNorm2d(512),
        nn.ReLU(inplace=True),
        nn.Conv2d(512,512, 3, stride=1, padding=1),
        nn.BatchNorm2d(512),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2)
)
```

模型和方法 – 方法 – pytorch

- 全连接层

```
    # 网络层的构建
    def forward(self, x):
        output = self.layers(x)
        # 一个 batch 有4个样本
        output = output.view(4,512)
        output = self.classifier(output)
        return output
```

模型和方法 – 方法 – pytorch

- 数据初始化：argmax

```
def calcAcc(isTrain):
    correct = 0
    total = 0
    with torch.no_grad():
        for data in trainloader if isTrain else testloader:
            images, labels = data
            if useGPU:
                images = images.cuda()
                labels = labels.cuda()
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return 1.0*correct/total
```

模型和方法 – 方法 – pytorch

- 其他
  - GPU 加速

```
# GPU
useGPU = torch.cuda.is_available()
if useGPU:
    net = net.cuda()
    net = torch.nn.DataParallel(net, device_ids=range(torch.cuda.device_count()))
    torch.backends.cudnn.benchmark = True
    criterion.cuda()
    # optimizer.cuda()
    print("USING GPU(CUDA)!")
else
    print("USING CPU!")
```

模型和方法 – 方法 – pytorch

- 之后根据过拟合情况使用了其他优化方案
  - 在之后版本迭代的过程中给出
  - version1(初始版本) -> version6

· 背景

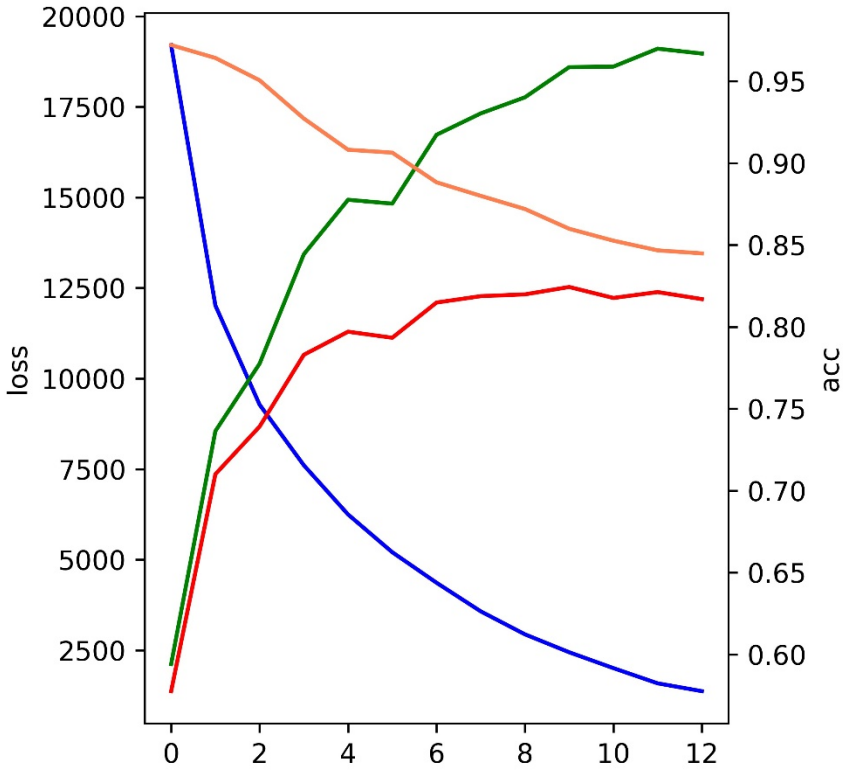
· 模型和方法

· 实验分析与展示

· 总结

实验分析与展示 – version1

· epoch [0-12]



·过拟合严重

·测试集最高准确率

·82.44%

实验分析与展示 – version2

- FractionalMaxPool

· 将 MaxPool 换成 FractionalMaxPool

· Benjamin Graham “Fractional Max-Pooling”

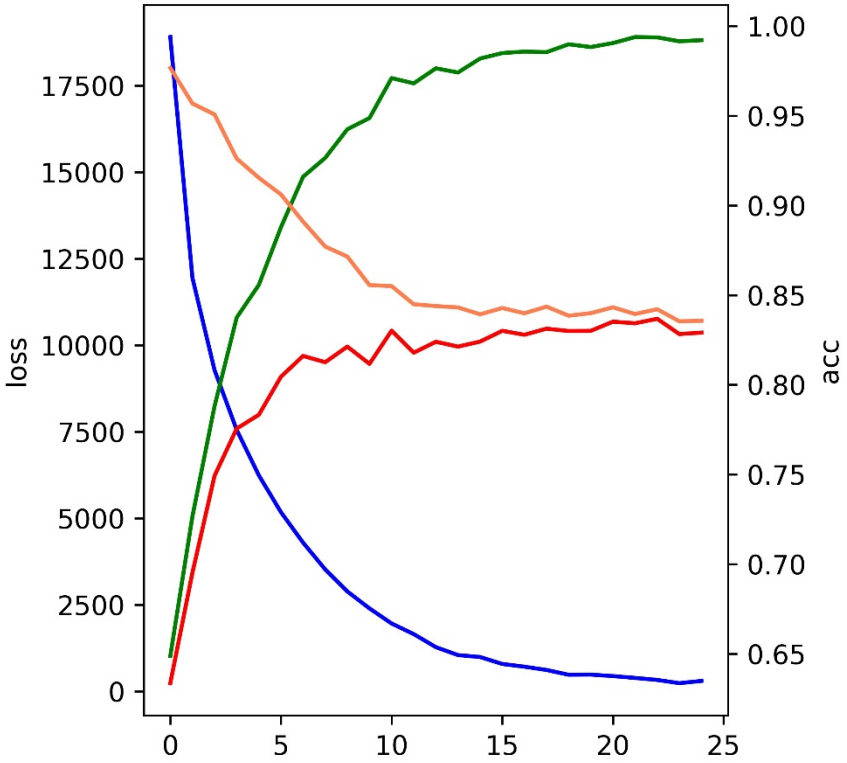
https://arxiv.org/pdf/1412.6071.pdf

· 分数池化 -> 提高泛化能力，减少过拟合

```
    nn.Conv2d(256,256, 3, stride=1, padding=1),
    nn.BatchNorm2d(256),
    nn.ReLU(inplace=True),
    nn.FractionalMaxPool2d(kernel_size=2, output_ratio=(0.5, 0.5)),
    # CCCP
    nn.Conv2d(256,512, 3, stride=1, padding=1),
    nn.BatchNorm2d(512),
    nn.ReLU(inplace=True),
    nn.Conv2d(512,512, 3, stride=1, padding=1),
    nn.BatchNorm2d(512),
    nn.ReLU(inplace=True),
    nn.Conv2d(512,512, 3, stride=1, padding=1),
    nn.BatchNorm2d(512),
    nn.ReLU(inplace=True),
    nn.FractionalMaxPool2d(kernel_size=2, output_ratio=(0.5, 0.5)),
```

实验分析与展示 – version2

· epoch [0-24]



·测试集最高准确率

·83.368%

·过拟合仍然严重

实验分析与展示 – version3(FractionalMaxPool + dropout)

- FractionalMaxPool + dropout

· 加入 dropout -> 减少过拟合

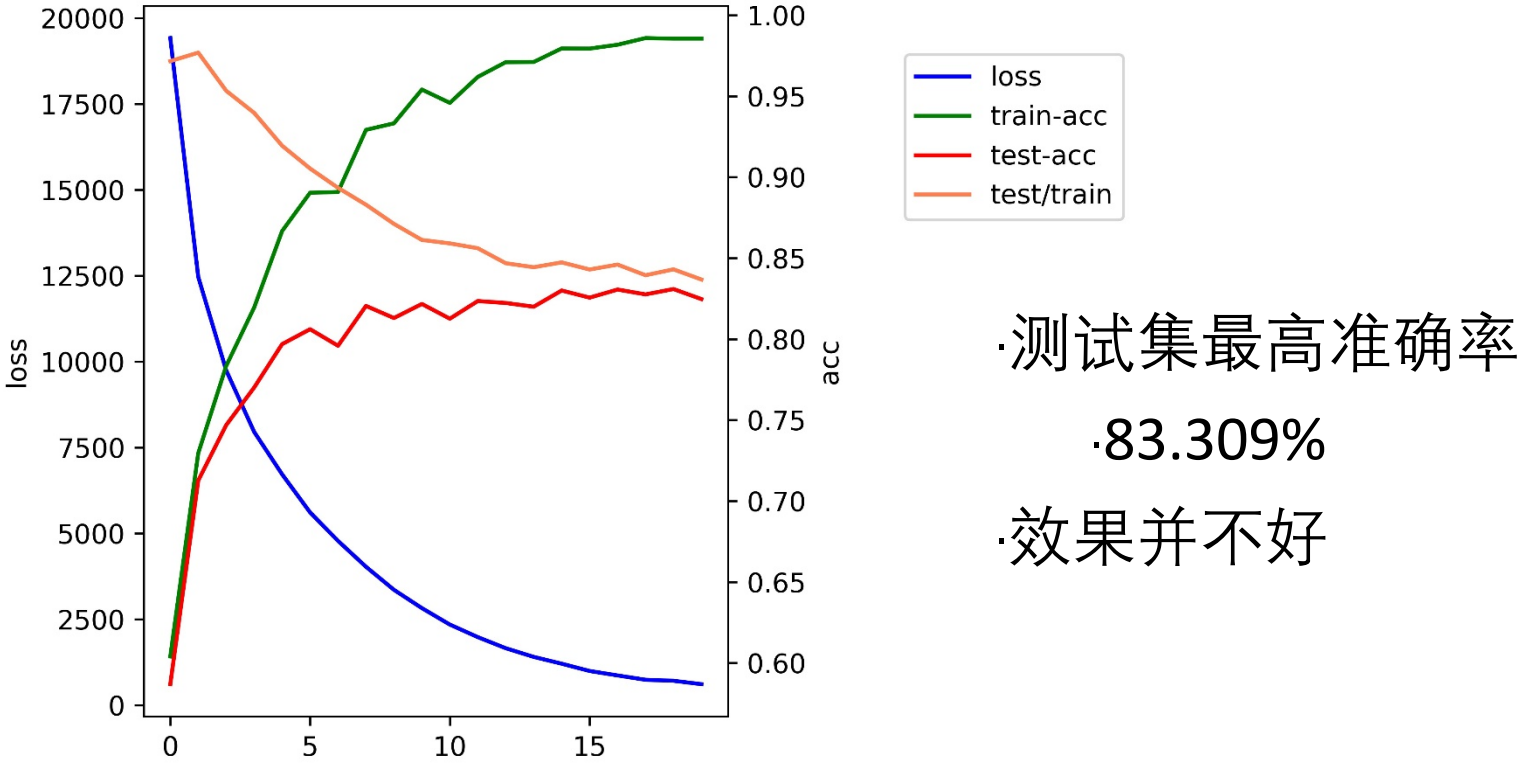
  · 在某些层上随机删除一些边

· 在最后两次最大池化后加入 20% dropout

```
        # CCCP
        # dropout-v3
        nn.Dropout(p=0.2),
        # dropout-v3
        nn.Conv2d(512,512, 3, stride=1, padding=1),
        nn.BatchNorm2d(512),
        nn.ReLU(inplace=True),
        nn.Conv2d(512,512, 3, stride=1, padding=1),
        nn.BatchNorm2d(512),
        nn.ReLU(inplace=True),
        nn.Conv2d(512,512, 3, stride=1, padding=1),
        nn.BatchNorm2d(512),
        nn.ReLU(inplace=True),
        nn.FractionalMaxPool2d(kernel_size=2, output_ratio=(0.5, 0.5)),
        # dropout-v3
        nn.Dropout(p=0.2)
        # dropout-v3
```

实验分析与展示 – version3

· epoch [0-19]



·测试集最高准确率

·83.309%

·效果并不好

实验分析与展示 – version4

- FractionalMaxPool + dropout + enhanced pre-process

· 加强对图像的预处理

· 50% 的概率将图片进行竖直翻转

· 将图片进行随机裁剪

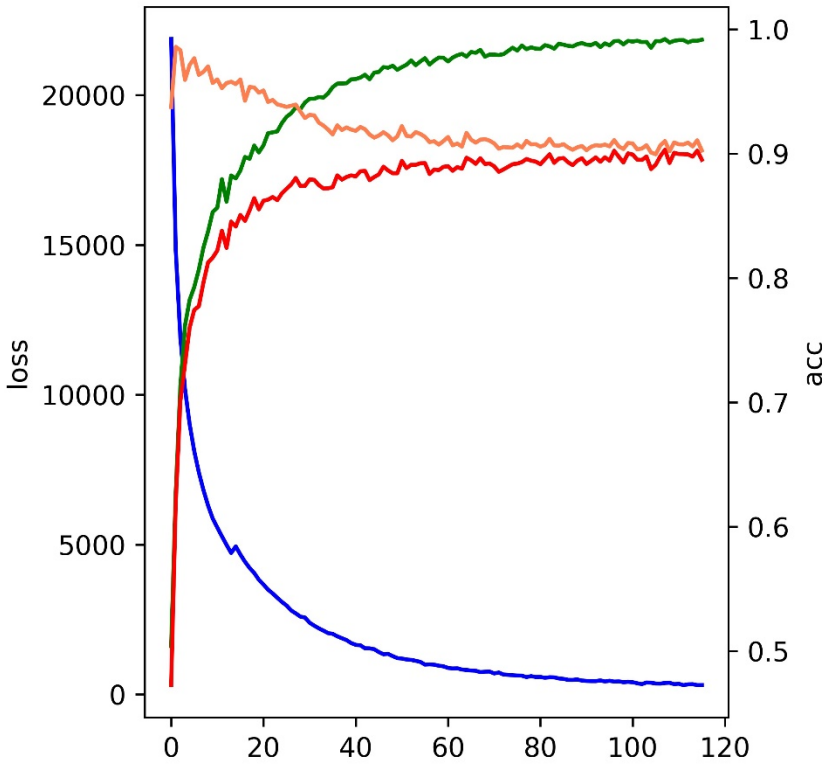
```
transform1 = transforms.Compose(
    # 随机裁剪 + padding
    transforms.RandomCrop(32, padding=4),
    # 50% 可能性竖直翻转
    transforms.RandomHorizontalFlip(),
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

transform2 = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

实验分析与展示 – version4

- 随即裁剪 + padding 之后输出变成 40*40*3
- 不影响通道数
  - 随机裁剪不不影响通道数
  - 池化不影响通道数
- 但是不影响主体结构，仍然通过 5 次池化变成 1*1
  - 由于始终是方形，只写一维
  - 40 -> 20 -> 10 -> 5 -> 2 -> 1

实验分析与展示 – version4

· epoch [0-115]



·测试集最高准确率

·90.32%

·效果拔群

·准确率稳定在 90%

实验分析与展示 – version5

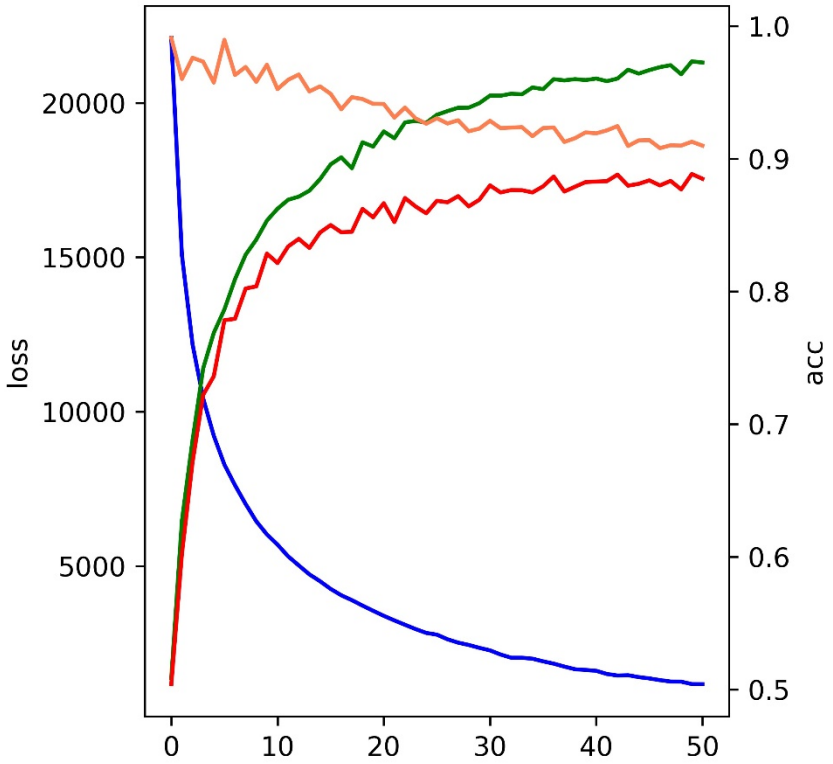
- FractionalMaxPool + dropout + enhanced pre-process + 非常规正则化

```
transform1 = transforms.Compose([
    # 随机裁剪 + padding
    transforms.RandomCrop(32, padding=4),
    # 50% 可能性竖直翻转
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))])

transform2 = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))])
```

https://github.com/kuangliu/pytorch-cifar

实验分析与展示 – version5

· epoch [0-50]



·测试集最高准确率

·88.85%

·训练集正确率还不高

97%

可以继续训练

实验分析与展示 – version6

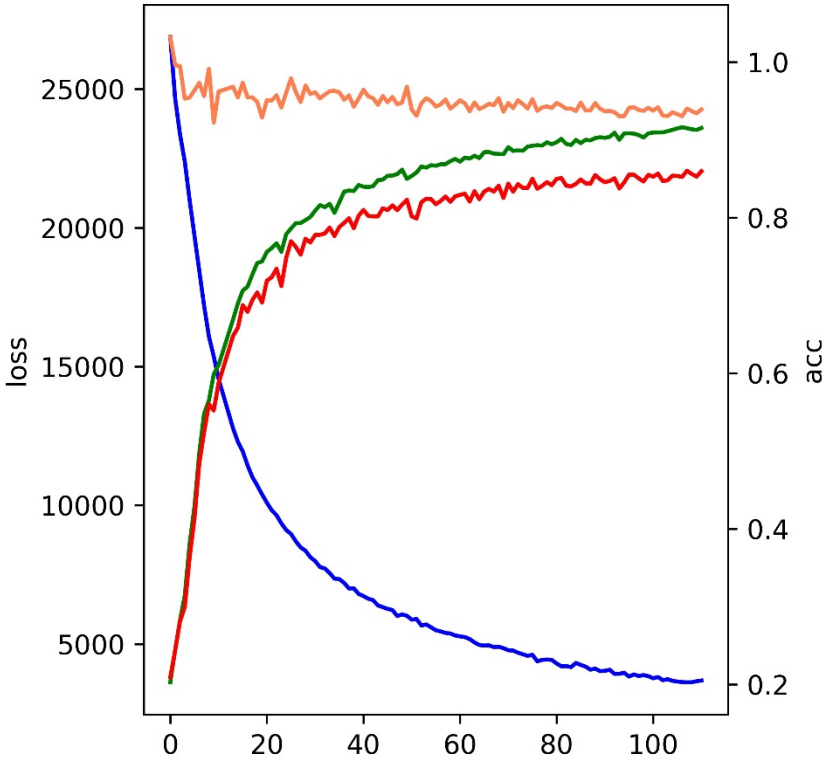
- FractionalMaxPool + enhanced pre-process
  + 非常规正则化 + 强化版dropout + 最后再加一层全连接层

http://torch.ch/blog/2015/07/30/cifar.html

```
# CCP
nn.Conv2d(3, 64, 3, stride=1, padding=1),
# 正则化
nn.BatchNorm2d(64),
# 激活函数
nn.ReLU(inplace=True),
nn.Dropout(p=0.3),# v6
nn.Conv2d(64, 64, 3, stride=1, padding=1),
nn.BatchNorm2d(64),
nn.ReLU(inplace=True),
nn.FractionalMaxPool2d(kernel_size=2, output_ratio=(0.5, 0.5)),
# CCP
nn.Conv2d(64, 128, 3, stride=1, padding=1),
nn.BatchNorm2d(128),
nn.ReLU(inplace=True),
nn.Dropout(p=0.4),# v6
nn.Conv2d(128,128, 3, stride=1, padding=1),
nn.BatchNorm2d(128),
nn.ReLU(inplace=True),
nn.FractionalMaxPool2d(kernel_size=2, output_ratio=(0.5, 0.5)),
# CCCP
nn.Conv2d(128,256, 3, stride=1, padding=1),
nn.BatchNorm2d(256),
nn.ReLU(inplace=True),
nn.Dropout(p=0.4),# v6
nn.Conv2d(256,256, 3, stride=1, padding=1),
nn.BatchNorm2d(256),
nn.ReLU(inplace=True),
nn.Dropout(p=0.4),# v6
nn.Conv2d(256,256, 3, stride=1, padding=1),
nn.BatchNorm2d(256),
nn.ReLU(inplace=True),
nn.FractionalMaxPool2d(kernel_size=2, output_ratio=(0.5, 0.5)),
# CCCP
nn.Conv2d(256,512, 3, stride=1, padding=1),
nn.BatchNorm2d(512),
nn.ReLU(inplace=True),
nn.Dropout(p=0.4),# v6
nn.Conv2d(512,512, 3, stride=1, padding=1),
nn.BatchNorm2d(512),
nn.ReLU(inplace=True),
nn.Dropout(p=0.4),# v6
nn.Conv2d(512,512, 3, stride=1, padding=1),
nn.BatchNorm2d(512),
nn.ReLU(inplace=True),
nn.FractionalMaxPool2d(kernel_size=2, output_ratio=(0.5, 0.5)),
# CCCP
nn.Conv2d(512,512, 3, stride=1, padding=1),
nn.BatchNorm2d(512),
nn.ReLU(inplace=True),
nn.Dropout(p=0.4),# v6
nn.Conv2d(512,512, 3, stride=1, padding=1),
nn.BatchNorm2d(512),
nn.ReLU(inplace=True),
nn.Dropout(p=0.4),# v6
nn.Conv2d(512,512, 3, stride=1, padding=1),
nn.BatchNorm2d(512),
nn.ReLU(inplace=True),
nn.FractionalMaxPool2d(kernel_size=2, output_ratio=(0.5, 0.5)),
nn.Dropout(p=0.4),# v6
)
# v6 FC
self.fc1 = nn.Linear(512, 512)
self.fc2 = nn.BatchNorm2d(512)
self.fc3 = nn.ReLU()
self.fc4 = nn.Dropout(p=0.5)# v6
# Linear
self.classifier = nn.Linear(512, 10)
```

实验分析与展示 – version6

· epoch [0-110]



http://torch.ch/blog/2015/07/30/cifar.html

·测试集最高准确率

·85.98%

·训练速度变慢

·参数多

·训练集正确率 91%

·还能继续训练

·GPU太贵 + 时间太长

·资料显示正确率可达

·92.45%

· 背景

· 模型和方法

· 实验分析与展示

· 总结

总结

- VGG16
- 尝试了不同的减少过拟合的方案
  - FractionalMaxPool
  - dropout
  - 增强数据预处理
  - 非标准正则化

总结

- 最终提交版本为version4
  - FractionalMaxPool + dropout + enhanced pre-process
  - 不选 version5,version6
    - GPU太贵以及训练时间过长未完成训练

- 最终提交版本为version4,epoch=107

[107]

loss: 381.36252677440643
train acc: 0.99226
test acc: 0.9032