# HW03

> 用pytorch实现卷积神经网络，对cifar10数据集进行分类

# 网络结构

## 初始网络结构

$$三层(Conv \to ReLU \to MaxPooling)块 + FC(512 \to 512 \to 10) \tag{1}$$

微调`lr`，多次试验，发现`1e-3`， `1e-5`最终都很优

尝试RMSProp，SGDM; 尝试平均池化

结果为`60%`左右，视为基准结果

## 引入 `batch_norm` 和改进结构

考虑VGG, 添加`batch_norm`，更改Conv；对块进行更新，为：

$$(Conv \to BatchNorm \to ReLU \to Conv \to BatchNorm \to ReLU \to MaxPooling) \tag{2}$$

提升到`70%`

## 对`classifier`进行改造

希望在给出结果时候不同特征互相参考进行适当的二次检查

1. 简单的$512 \to 10$ , train不上去
2. $512 \to 1024 \to 10$ 上到82%
3. $512 \to 2048 \to 10$ 无明显提升
4. $512 \to 4096 \to 10$ 上到90%，但是太慢

且中间层越大，达到（疑似）收敛所需`epoch`越少，无`early stop`时过拟合越严重

## 尝试不同`lr`

发现此时`1e-3` 最佳。改大(`1e-2`, `1e-1`)无明显提升，改小(`1e-4`, `1e-5`)反而训练不上去

## 正则化

添加L2正则化，`test_acc`曲线更加平滑

## 其他（非时间顺序）

- 添加`early stop`，patience为10。在10/15次test_acc无提高后停止训练

- 对Conv层的变化层次进行调整，发现扩大`channel`对于model的提升较大，此时达到`85%`
- 改最后一层为平均池化，效果不佳
- 去掉pooling, 网络过大，训练过慢，手动终止

# 数据增强

算`testData`, `trainData`的均值方差，进行归一化

- 对`test`仅仅进行归一化操作
- 对于`train`尝试如下
  - 仅仅左右翻转，上升至`83%`
  - 左右翻转 + 高斯噪音， 上升至`84%`
  - 左右翻转 + 饱和度等色彩的变化(分类结果小幅度改变色彩影响不大)，提升不大，但是训练时间大幅增加
  - 左右翻转 + 仿射变换，训练太慢
  - 左右翻转 + 色彩变化 + 高斯噪音 + 仿射变换, 训练不起来，一直上不了`80%`
  - 左右翻转 + 随机擦除， 效果提升，速度影响不大, 效果有所提升

实行数据增强后，`early stop`的时间得以延后，泛化能力更强（train与test的准确度差值显著缩小）

（这里用训练完成后test_acc与train_acc的差距 量化 泛化能力）

添加`early stop`大约差值在`4%`， 100epoch也能较好地维持到`8%`以内

```
# 自定义高斯噪声变换
class AddGaussianNoise(object):
    def __init__(self, mean=0., std=0.1, p=0.5):
        self.mean = mean
        self.std = std
        self.p = p

    def __call__(self, tensor):
        if np.random.rand() < self.p:
            return tensor + torch.randn(tensor.size()) * self.std + self.mean
        return tensor

    def __repr__(self):
        return self.__class__.__name__ + '(mean={0}, std={1}, p={2})'.format(self.mean,
self.std, self.p)


test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)) # 归一化
])
```

```
# 数据增强的数据预处理方式
train_transform = transforms.Compose([
    # transforms.RandomResizedCrop(32, scale=(0.8, 1.0)),  # 随机缩放裁剪
    transforms.RandomHorizontalFlip(p=0.5),
    # transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),  # 颜色扰动
    # transforms.RandomRotation(15),  # 小幅旋转
    transforms.ToTensor(),
    transforms.RandomErasing(p=0.5, scale=(0.02, 0.1), ratio=(0.3, 3.3)),  # 随机擦除
    # AddGaussianNoise(mean=0., std=0.1, p=0.5),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))  # 归一化
])
```

# 其余修改

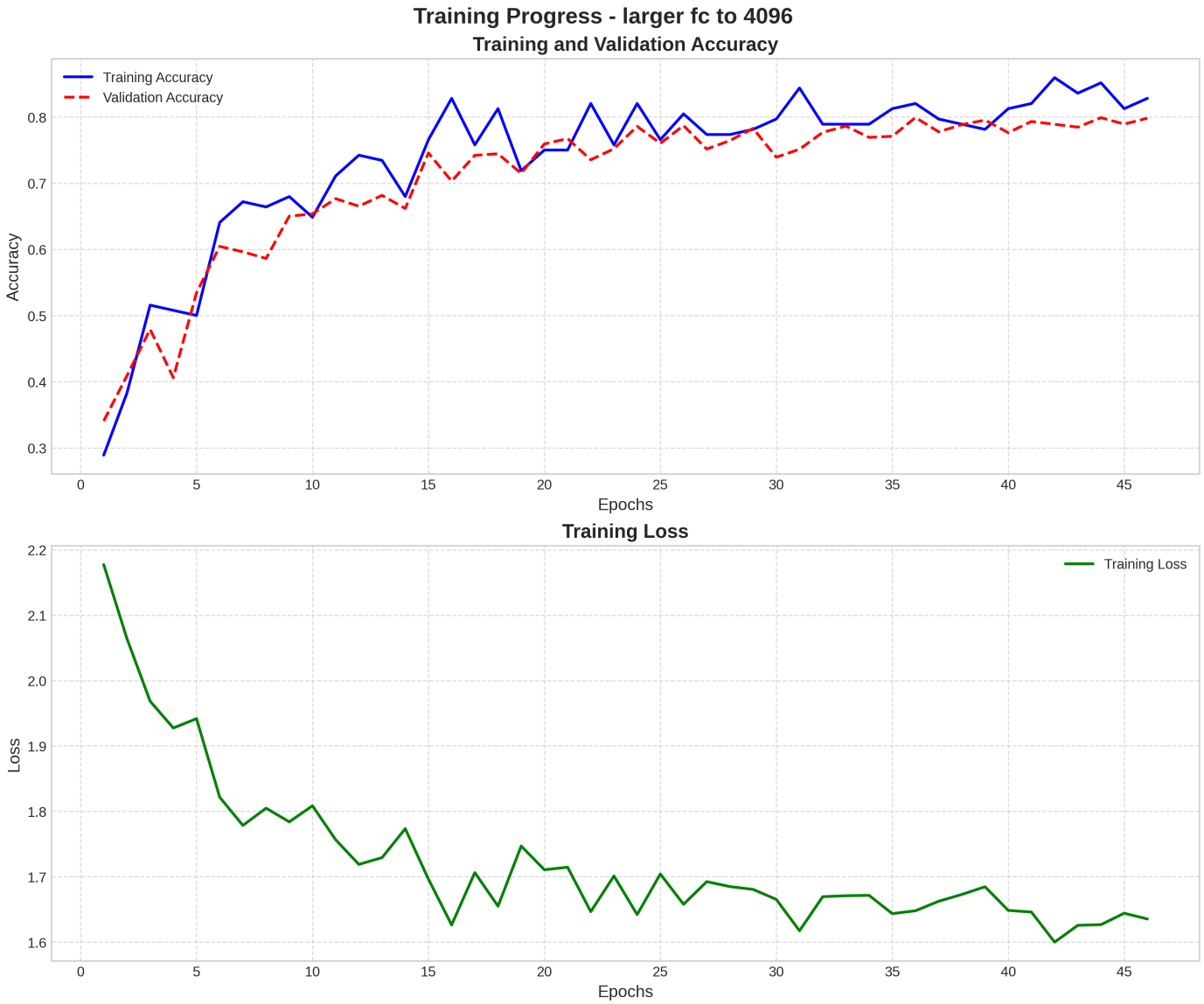
为方便模块化以及在headless模式下显示图片，在jupyter-lab中进行了全部lab。修改了训练流程的输出（tqdm进度条与图像绘制）

train_log文件夹中保存了训练结果，除部分因感觉过慢和过度overfit提前终止，没有进行自动的绘图和保存外

# 最终结果

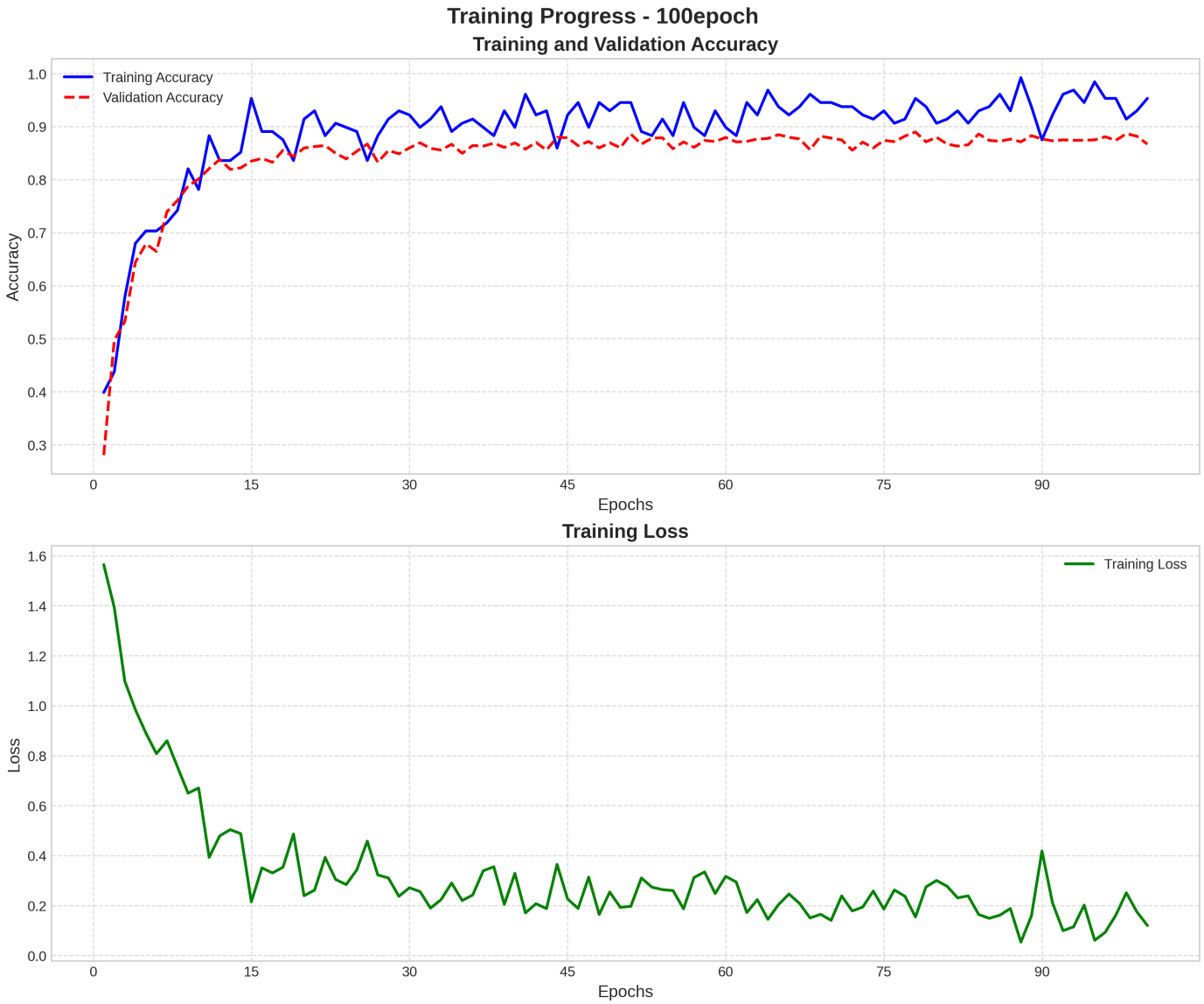
添加`early stop`后， 五层，$512 \to 4096 \to 10$

```
batch_size: 128
learning_rate: 0.001
num_epochs: 100
early_stop_patience: 10
```




关闭`early stop`后， 五层，$512 \to 1024 \to 10$

（4096的结果过拟合严重）


Training Progress - 100epoch
Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
15
30
45
60
75
90
Epochs
Training Loss
Training Loss
Loss
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
15
30
45
60
75
90
Epochs