实验二：卷积神经网络

姓名：刘威

学号：PB18010469

实验目的

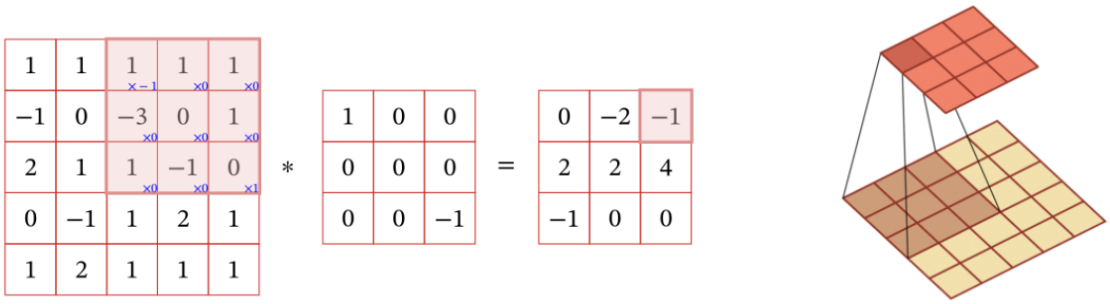
- 了解并熟悉卷积神经网络的原理及其学习算法
- 研究dropout对卷积神经网络泛化性能的影响
- 研究normalization对卷积神经网络的影响
- 研究residual connection对深层卷积神经网络性能的影响
- 研究学习率学习率衰减对卷积神经网络性能的影响
- 研究网络深度对卷积神经网络性能的影响

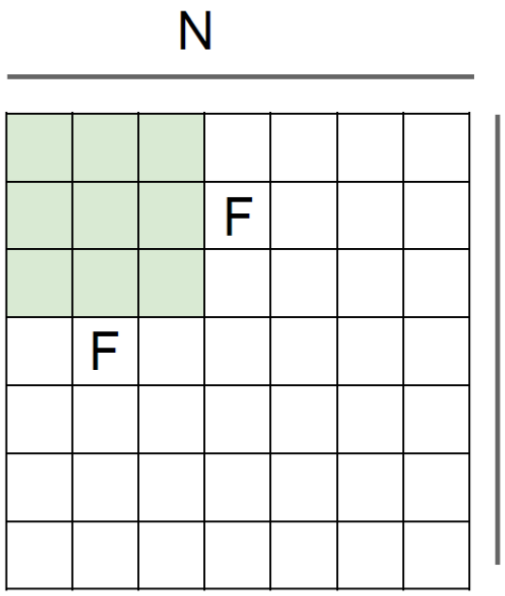
实验原理

二维卷积:

- 在图像处理中，图像是以二维矩阵的形式输入到神经网络中，因此我们需要二维卷积。
- 一个输入信息$\boldsymbol{X}$和滤波器$\boldsymbol{W}$的二维卷积定义如下

$$\boldsymbol{Y} = \boldsymbol{W} * \boldsymbol{X} \qquad y_{ij} = \sum_{u=1}^{U}\sum_{v=1}^{V} w_{uv} x_{i-u+1,j-v+1}$$

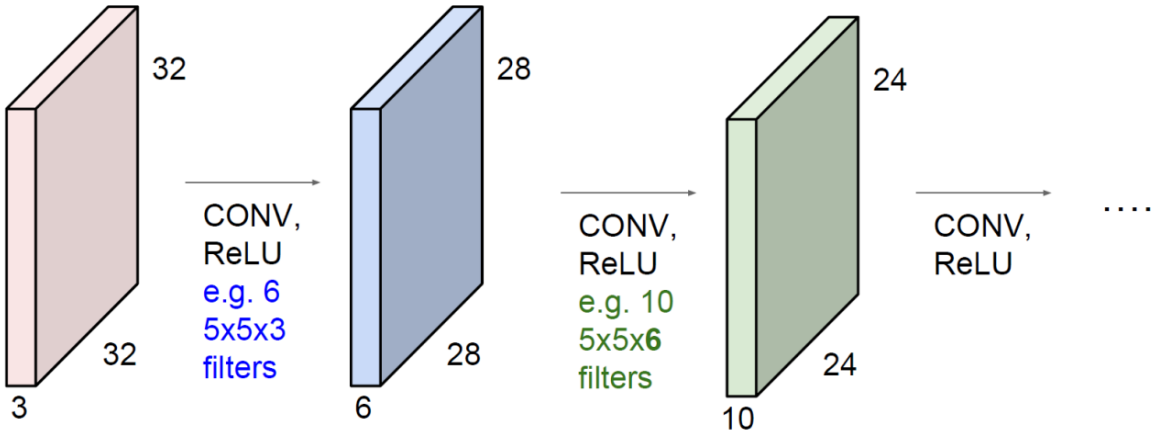




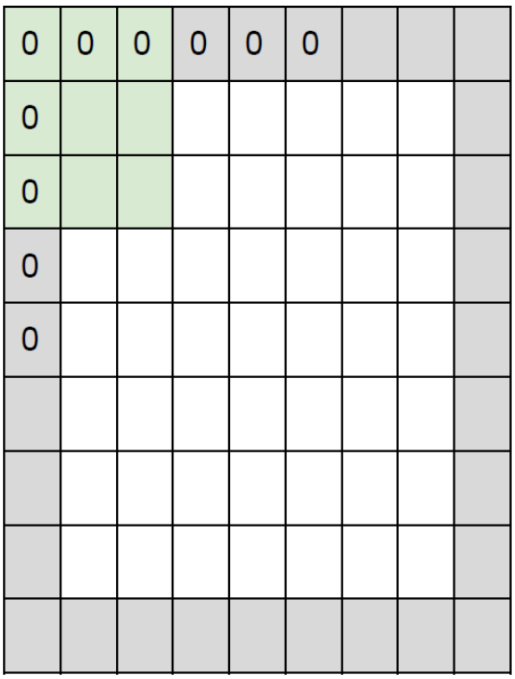
- 输出大小：(N-F)/S+1
  - S为步幅大小
- 比如N=7, F=3
  - 步幅为1，(7-3)/1+1=5
  - 步幅为2，(7-3)/2+1=3
  - 步幅为3，(7-3)/3+1=2.3

多层卷积：

卷积神经网络是用卷积层代替全连接层



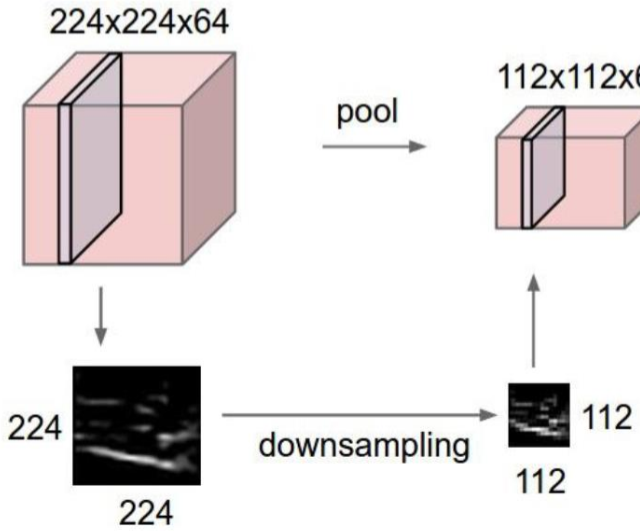
Padding:



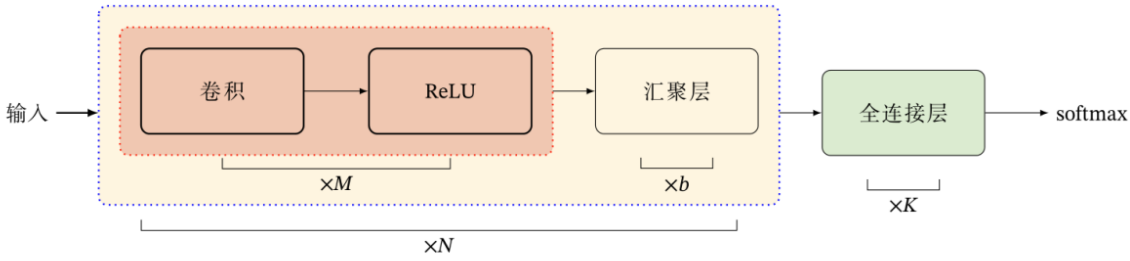
- 在输入周边填0，以解决特征图大小改变太快问题
- 例子1
  - 7x7的输入，3x3的核，步幅为1
  - 填一个像素的边界，输入实际上是9x9
  - 输出大小为：(9-3)/1+1 = 7
  - 输出7x7，与输入相同大小
- 例子2
  - 7x7的输入，3x3的核，步幅为3
  - 填一个像素边界
  - 输出大小为：(9-3)/3+1=3

池化：

- 池化函数使用某一位置的相邻输出的总体统计特征来代替网络在该位置的输出，使得特征图尺寸更小且更易于管理
  - 最大池化（max pooling）函数给出相邻矩形区域内的最大值
  - 其他常用的池化函数包括相邻矩形区域内的平均值、L_2范数以及基于据中心像素距离的加权平均函数



卷积神经网络的典型结构：



一个卷积块为连续M 个卷积层和b个汇聚层（M通常设置为2 ~ 5，b为0或1）。一个卷积网络中可以堆叠N 个连续的卷积块，然后在接着K 个全连接层（N 的取值区间比较大，比如1 ~ 100或者更大；K一般为0 ~ 2）。

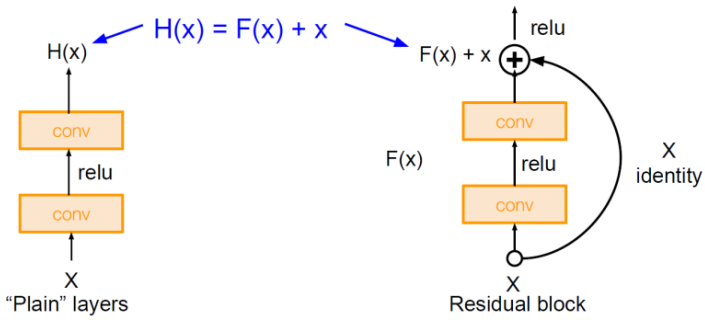
残差网络：

- 残差网络（Residual Network，ResNet）是通过给非线性的卷积层增加直连边的方式来提高信息的传播效率。
  - 假设在一个深度网络中，我们期望一个非线性单元（可以为一层或多层的卷积层）$F(x,\theta)$去逼近一个目标函数为$h(x)$。
  - 将目标函数拆分成两部分：恒等函数和残差函数

$$h(x) = x + (h(x) - x)$$

恒等函数　残差函数

$F(x,\theta)$



实验内容

使用`pytorch`或者`tensorflow`实现卷积神经网络，在`ImageNet`数据集上进行图片分类。研究dropout、 normalization、 learning rate decay、 residual connection、网络深度等超参数对分类性能的影响。

数据集：`tiny-imagenet-200`

实验结果

实验使用`pytorch`进行。

源码结构及说明

模型结构：

模型的主体结构由基础块堆叠而成。基础块由两层`3 x 3`的卷积网络构成，可以通过参数设定该基础块的输入通道数，输出通道数，是否使用批量标准化，以及是否使用残差连接。除此之外还可以选择是否在第一层卷积网络中使用长度为2的步长来缩减特征图的大小，使用多大舍弃概率的dropout层。具体实现如下：

```
class BasicBlock(nn.Module):
    def __init__(self, c_in: int, c_out: int, stride: int = 1, res: bool =
False, norm: bool = True, dropout: float = 0.2):
        super().__init__()
        self.res = res
        self.conv1 = nn.Conv2d(c_in, c_out, 3, stride, 1)
        self.bn1 = nn.BatchNorm2d(c_out) if norm else nn.Identity()
        self.relu1 = nn.ReLU()
        self.dropout1 = nn.Dropout(dropout)
        self.conv2 = nn.Conv2d(c_out, c_out, 3, 1, 1)
        self.bn2 = nn.BatchNorm2d(c_out) if norm else nn.Identity()
        if res and stride != 1:
            self.downsample = nn.Sequential(
                nn.Conv2d(c_in, c_out, 1, stride, 0),
                nn.BatchNorm2d(c_out) if norm else nn.Identity()
            )
        else:
```

```
            self.downsample = None
        self.relu2 = nn.ReLU()
        self.dropout2 = nn.Dropout(dropout)

    def forward(self, x):
        identity = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu1(out)
        out = self.dropout1(out)
        out = self.conv2(out)
        out = self.bn2(out)

        if self.downsample is not None:
            identity = self.downsample(identity)
        if self.res:
            out += identity
        out = self.relu2(out)
        out = self.dropout2(out)
        return out
```

可见如果统一参数`res, norm, dropout`，决定基础块结构的参数为`(c_in, c_out, stride)`，只需要确定这个三元组即可确定网络的结构。

固定网络的输入层和输出层，中间层使用若干基础块堆叠，可以自由调整网络的深度。传入各个基础块的三元组参数，就可以确定整个网络的结构。总的网络实现如下：

```
class CNN(nn.Module):
    def __init__(self, block_sizes, res: bool = False, norm: bool = True,
conv_dropout=0.2, fc_dropout=0.5):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, 5, 1, 2),  # 64*64*64
            nn.BatchNorm2d(64) if norm else nn.Identity(),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)  # 64*32*32
        )
        self.block_list = nn.ModuleList()
        for block_size in block_sizes:
            block = BasicBlock(*block_size, res=res, norm=norm,
dropout=conv_dropout)
            self.block_list.append(block)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.dropout = nn.Dropout(fc_dropout)
        self.fc = nn.Linear(block_sizes[-1][1], 200)

    def forward(self, x):
        x = self.conv1(x)
        for block in self.block_list:
            x = block(x)
        x = self.avgpool(x)
        x = x.view(x.shape[0], -1)
        x = self.dropout(x)
        x = self.fc(x)
        return x
```

其中`block_sizes`就是各个基础块的三元组参数组成的列表。例如，本实验使用了如下三组`block_sizes`：

```
# 结构1: 共11层卷积（算上输入层），约20M参数
[(64, 64, 1), (64, 128, 2), (128, 256, 2), (256, 512, 2), (512, 1024, 2)],
# 结构2: 共21层卷积，约45M参数
[(64, 64, 1), (64, 64, 1), (64, 128, 2), (128, 128, 1), (128, 256, 2),     (256,
256, 1), (256, 512, 2), (512, 512, 1), (512, 1024, 2), (1024, 1024, 1)]
# 结构3: 共31层卷积，约70M参数
[(64, 64, 1), (64, 64, 1), (64, 64, 1), (64, 128, 2), (128, 128, 1), (128, 128,
1), (128, 256, 2), (256, 256, 1), (256, 256, 1), (256, 512, 2), (512, 512, 1),
(512, 512, 1), (512, 1024, 2), (1024, 1024, 1), (1024, 1024, 1)]
```

结果及分析

本实验的默认参数如下：

```
batch_size = 64
default_params = {
    'block_sizes':
        [
            (64, 64, 1),
            (64, 128, 2),
            (128, 256, 2),
            (256, 512, 2),
            (512, 1024, 2),
        ],  # 堆叠的各个基础块的参数，即结构1
    'epochs': 40,  # 最大训练轮数
    'res': True,  # 是否在基础块中使用残差连接
    'norm': True,  # 是否在卷积层后使用批量标准化
    'dropout': (0.2, 0.5),  # 分别为卷积层，全连接层后的drop概率
    'lr_init': 1e-3,  # 学习率的初始值
    'lr_min': 1e-5,  # 学习率的最小值
    'lr_decay': 0.1,  # 学习率的衰减倍率
    'lr_min_delta': 0.,  # 认为验证集loss有明显降低的阈值，用于调整学习率
    'lr_patience': 1,  # 验证集loss没有明显降低的连续轮数，用于调整学习率
    'val_min_delta': 0.,  # 认为验证集loss有明显降低的阈值，由于控制早停
    'val_patience': 3,  # 验证集loss没有明显降低的连续轮数，用于控制早停
    'top': [1, 5, 10],  # top n 准确率
    'restore_best_weights': True  # 早停后，是否将模型权值恢复为验证集loss最低时的权值
}
```

将会进行调节并加以对比的参数包括：

```
'block_sizes', 'res', 'norm', 'dropout', 'lr_decay'
```

此外，本实验固定随机种子，保证结果可复现：

```
import random
import numpy as np
import torch as t
import os

def set_seed(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    t.manual_seed(seed)
    t.cuda.manual_seed(seed)
    t.backends.cudnn.deterministic = True
set_seed(17717)
```

对比一：Dropout

固定其他参数为默认参数，调整`dropout`为如下四组值：

```
'dropout': [(0., 0.), (0.1, 0.3), (0.2, 0.5), (0.3, 0.7)],
```

训练结束后，对所得模型在验证集上进行验证，得到的结果如下表：

`dropout`	loss	top 1 accuracy	top 5 accuracy	top 10 accuracy
`(0., 0.)`	2.6278	0.4119	0.6750	0.7767
`(0.1, 0.3)`	2.5348	0.4245	0.6910	0.7916
`(0.2, 0.5)`	**2.3826**	**0.4432**	**0.7116**	**0.8041**
`(0.3, 0.7)`	2.4118	0.4233	0.6981	0.7954

分析：dropout概率太小或者不使用dropout技术，过拟合风险更大；dropout概率太大，会导致欠拟合。应该选择合适的dropout概率来降低过拟合风险，提高模型性能。

对比二：Normalization

固定其他参数为默认参数，调整`norm`分别为`True`或者`False`.

训练结束后，对所得模型在验证集上进行验证，得到的结果如下表：

`norm`	loss	top 1 accuracy	top 5 accuracy	top 10 accuracy
`True`	**2.3826**	**0.4432**	**0.7116**	**0.8041**
`False`	5.2983	0.0050	0.0250	0.0500

分析：如果不加标准化层，学不到东西。因此在使用卷积神经网络时标准化层是必不可少的。

对比三：Learning rate decay

固定其他参数为默认参数，调整`lr_decay`分别为`[0.1, 0.5, 0.99]`.

训练结束后，对所得模型在验证集上进行验证，得到的结果如下表：

lr_decay	loss	top 1 accuracy	top 5 accuracy	top 10 accuracy
0.1	**2.3826**	**0.4431**	**0.7116**	**0.8041**
0.5	2.4610	0.4162	0.6913	0.7891
0.99*	2.5231	0.4066	0.6773	0.7828

> *注：设置为0.99是因为pytorch的ReduceLROnPlateau只允许小于1.0的值，可以认为是不对学习率进行衰减。

分析：在训练网络时，应该适当调节学习率。在刚开始训练时可以使用较大的学习率以加快收敛速度，在接近极小值时，如果不对学习率进行调整，会由于步长过大而越过极小值而无法收敛，这时应该减小学习率，以更好的收敛。

对比四：Residual connection

固定其他参数为默认参数，将block_sizes设置为**结构3**， 调整res分别为True 或者 False.

训练结束后，对所得模型在验证集上进行验证，得到的结果如下表：

res	loss	top 1 accuracy	top 5 accuracy	top 10 accuracy
True	**3.2994**	**0.2570**	**0.5236**	**0.6470**
False	4.1934	0.0827	0.2677	0.4042

分析：在**结构3**(31层卷积层)中，加残差连接相比不加有很大的性能提升，证实了残差连接能够极大改善深层网络难以优化的问题。

对比五：网络深度

固定其他参数为默认参数，将block_sizes分别设置为前述三种结构。

训练结束后，对所得模型在验证集上进行验证，得到的结果如下表：

block_sizes	loss	top 1 accuracy	top 5 accuracy	top 10 accuracy
结构1（11层卷积）	**2.3826**	**0.4432**	**0.7116**	**0.8041**
结构2（21层卷积）	2.5370	0.3952	0.6694	0.7711
结构3（31层卷积）	3.3368	0.2576	0.5174	0.6403

分析：性能随着网络深度的增加变差了，但理论上深层网络应该至少能和浅层网络一样好。在实践中，更深层的网络更难优化，优化地形更为复杂，更容易陷入局部极小值，即使使用的残差连接技术，如果由于网络结构不适合当前任务，训练方法不当，徒增加网络深度仍很可能会导致更差的性能。

在何凯明等人的论文中，引入残差连接后，深层网络相比浅层网络的训练loss降得更快了，最终效果也更好。而在我的实验中，深层网络的收敛速度却慢得多， 最终性能也要差一些，调了很久的参数也没有改善。其中的具体原因有待进一步做实验来弄明白，在这次实验中没有更多的时间、精力和算力来继续研究下去了(DDL要到了:-)。

实验总结

因为卷积神经网络的结构及原理比较简单，目标也很明确，本次实验实现上来说并不难。但是限于算力有限，想要跑出一次结果都要等很久，参数调节起来颇有困难，花了很多时间也没有明显提升。因此研究的并不够细致，有一些地方没有达到理论上那么好的结果。今后有机会还得多尝试尝试。