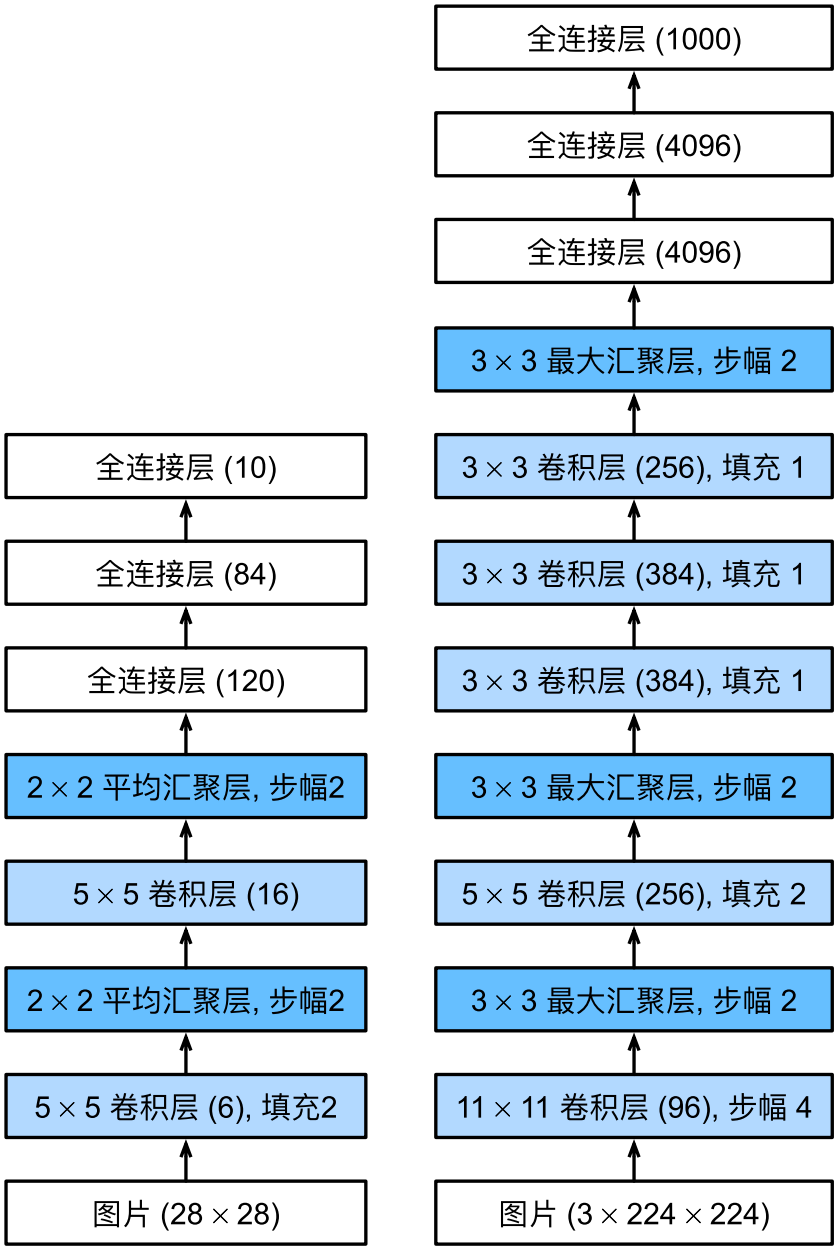
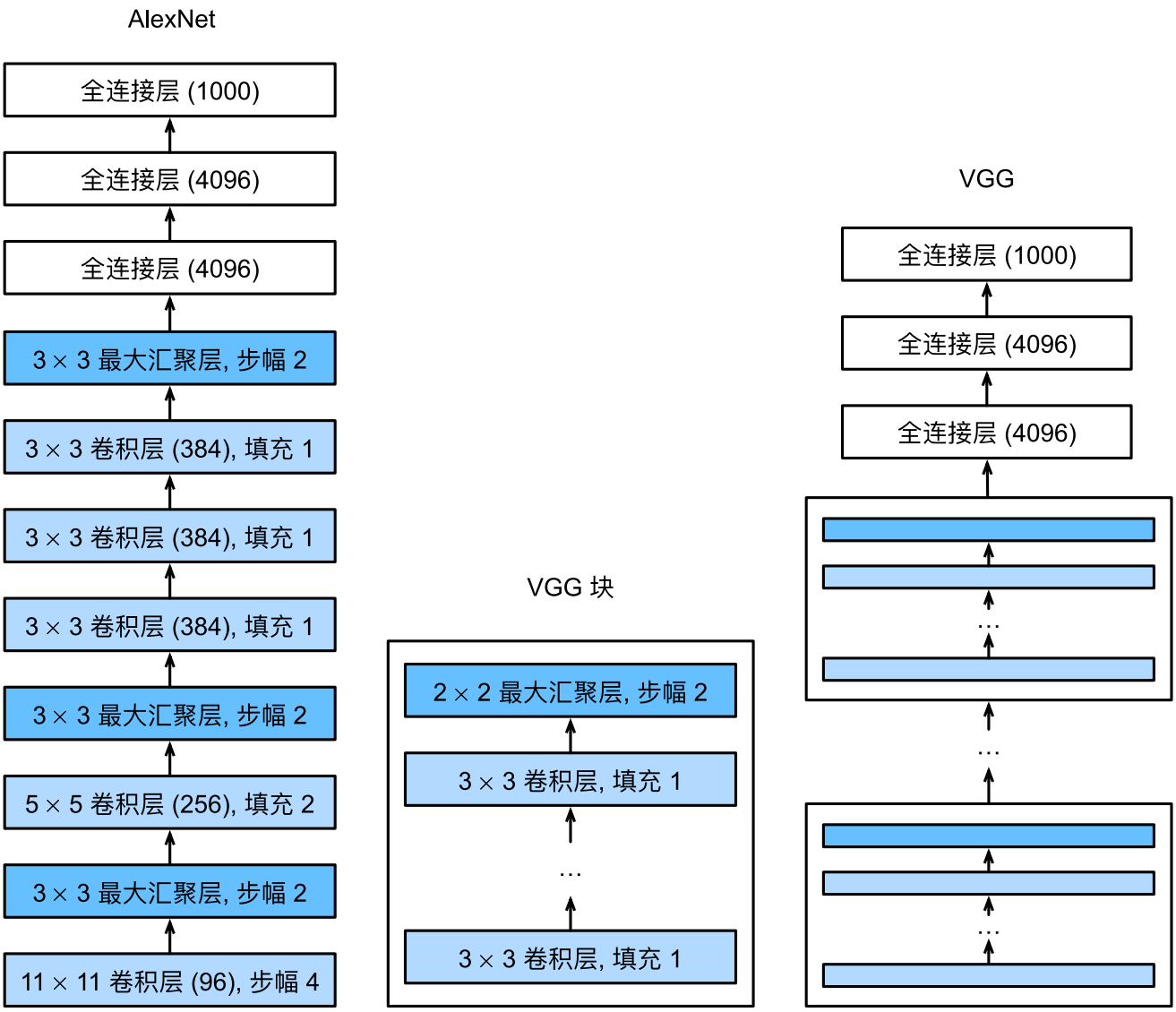
# 现代卷积网络

## Alexnet



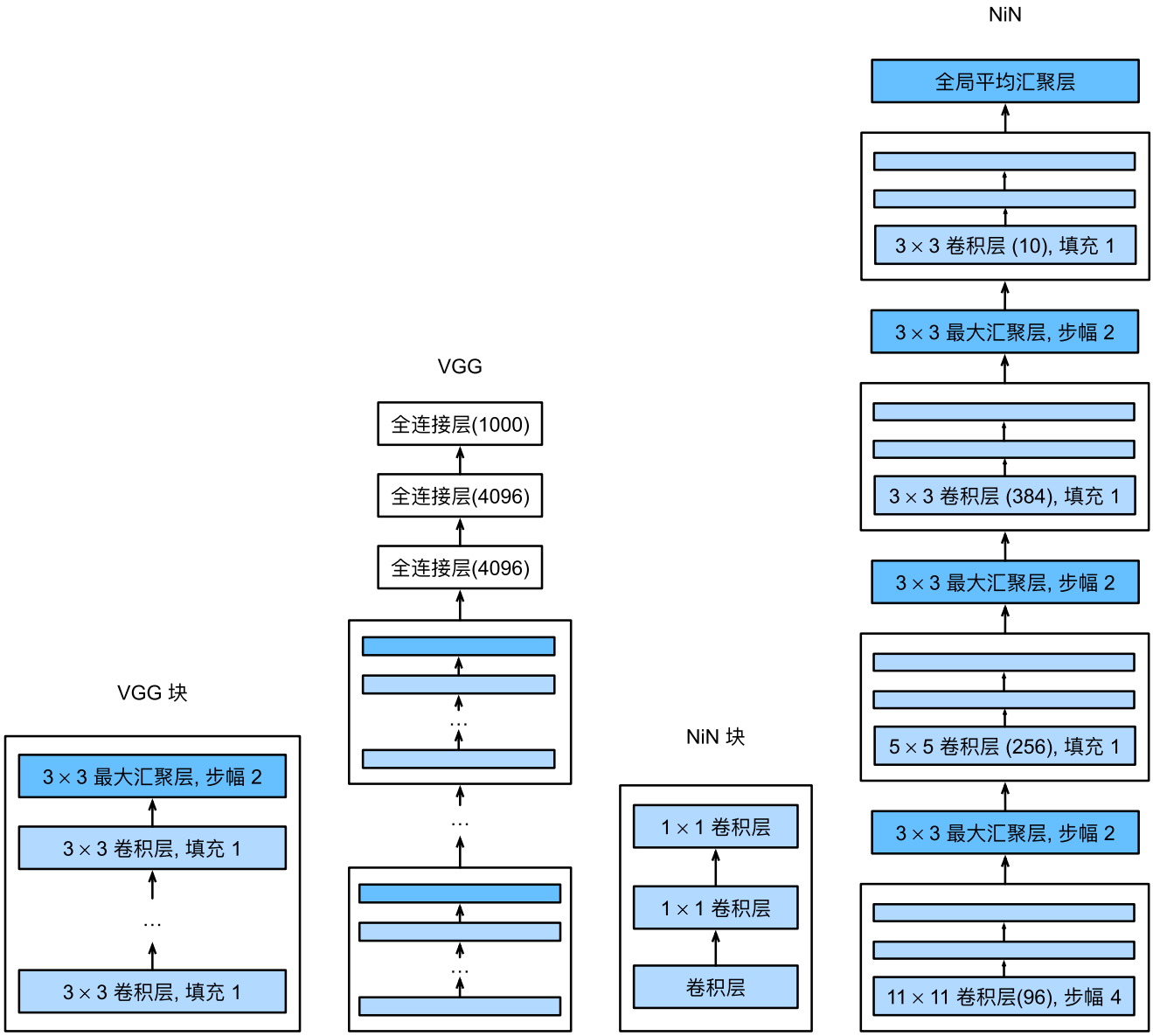

- 可以认为是Lenet plus，使用了11x11,5x5,3x3(3个)共5个大通道卷积层与3个池化层，展平后接一个多层感知机。
- 注意层与层之间通道，尺寸的匹配(或使用LazyConv2d)

## VGG



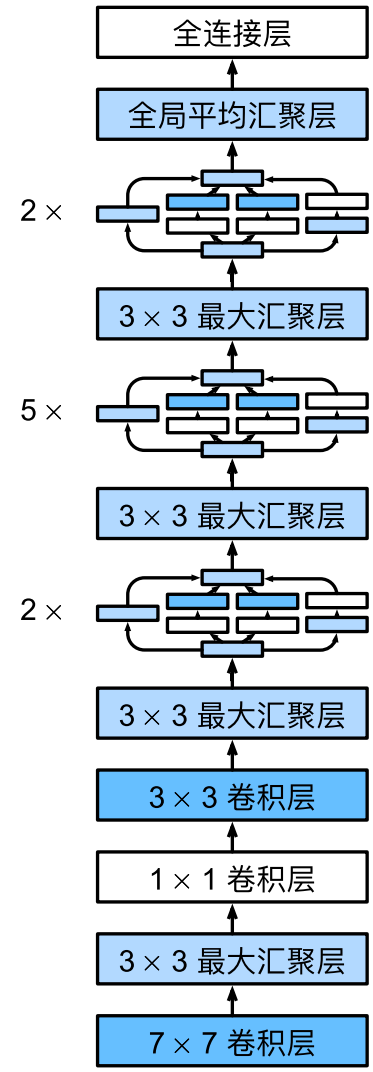

- 定义了可复用的块(n个卷积层(设置padding使图片分辨率不下降) + 1个池化层(stride >= 2))
- 利用vgg_block可以简便定义网络，只需定义每个块的卷积层数量和输出通道数即可，常用方案是让输出通道从初始值一直翻倍到目标值
- 深层且窄的卷积块比浅而宽的卷积块效果更好，可以认为深层且窄的网络利用层之间的结构实现了参数之间的约束，从而用更少的显式参数实现更强的表达力

# NiN

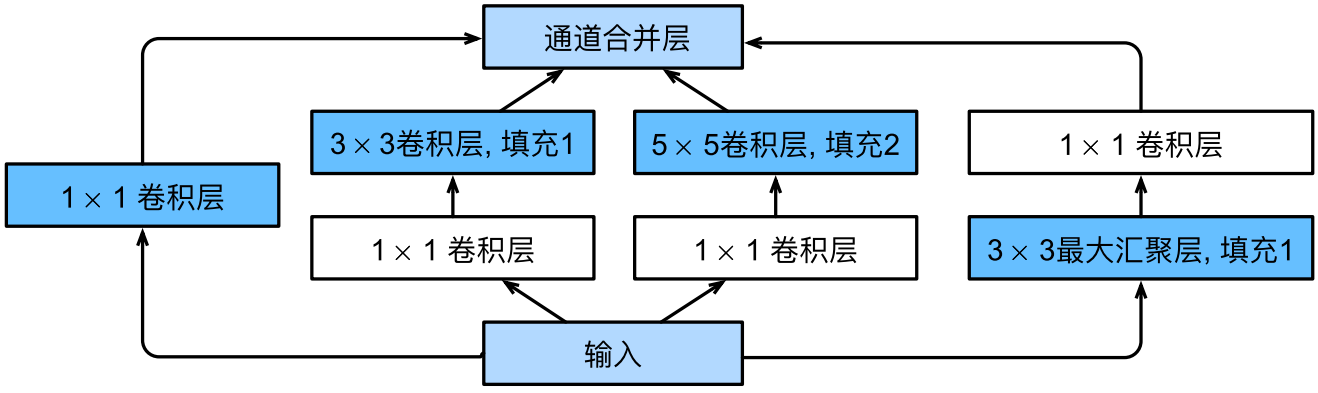



- 在前期插入全连接层能减少参数量，但会直接破坏特征的空间结构，NiN的解决方案是用1x1卷积层实现每个像素在通道上的全连接层
- NiN块实现: 1个自定义卷积层 + 2个1x1卷积层(由于可能需要做输出层，故不与池化层耦合)
- 在最后一个NiN块前加入一个Dropout层
- NiN块(输出通道设置为标签数) + 全局平均池化层(对同一通道的所有像素求均值，输出（1x输入通道数）的向量)+ Flatten -> 输出，用这一设计替代Flatten + 全连接层的好处是后者Flatten后保留了特征图内部的无用空间信息，这会增大参数量并造成过拟合

# GoogLenet




- 引入Inception块：并行完成1x1卷积, 3x3卷积, 5x5卷积, maxpool（都需要结合1x1卷积调整通道数量），保证卷积后图像分辨率不变，并将结果用torch.cat在通道方向上连结




- 通过调控Inception块中四条并行路径中的通道数量比例来实现对不同尺寸特征关注程度不同的Inception块，通道数量比一般只能由目标数据集上的实验得到
- 将Inception块串联来获得表达力更强的块，在这些大块之间插入stride = 2的最大池化层来降低图像维度
- 最后使用全局平均池化层 + Linear(通道数，标签数) 来获得分类结果

# BatchNorm

- 层的基础设计模式：

  1.把数学原理部分抽离出来，封装成一个单独的函数，执行计算和具体更新参数的任务(体现在返回值中)

  2.定义层类，用于实现把数据移动到训练设备，分配初始化必要变量，跟踪参数更新等功能。

```
def batch_norm(X, gamma, beta, moving_mean, moving_var, eps, momentum):

# 通过is_grad_enabled来判断当前模式是训练模式还是预测模式

if not torch.is_grad_enabled():

# 如果是在预测模式下，直接使用传入的移动平均所得的均值和方差

X_hat = (X - moving_mean) / torch.sqrt(moving_var + eps)

else:

assert len(X.shape) in (2, 4)

if len(X.shape) == 2:

# 使用全连接层的情况，计算特征维上的均值和方差

mean = X.mean(dim=0)

var = ((X - mean) ** 2).mean(dim=0)

else:

# 使用二维卷积层的情况，计算通道维上（axis=1）的均值和方差。

# 这里我们需要保持X的形状以便后面可以做广播运算

mean = X.mean(dim=(0, 2, 3), keepdim=True)

var = ((X - mean) ** 2).mean(dim=(0, 2, 3), keepdim=True)

# 训练模式下，用当前的均值和方差做标准化

X_hat = (X - mean) / torch.sqrt(var + eps)
```

```
# 更新移动平均的均值和方差

moving_mean = momentum * moving_mean + (1.0 - momentum) * mean

moving_var = momentum * moving_var + (1.0 - momentum) * var

Y = gamma * X_hat + beta # 缩放和移位

return Y, moving_mean.data, moving_var.data
```

```
class BatchNorm(nn.Module):

# num_features：完全连接层的输出数量或卷积层的输出通道数。

# num_dims：2表示完全连接层，4表示卷积层

def __init__(self, num_features, num_dims):

super().__init__()

if num_dims == 2:

shape = (1, num_features)

else:

shape = (1, num_features, 1, 1)

# 参与求梯度和迭代的拉伸和偏移参数，分别初始化成1和0

self.gamma = nn.Parameter(torch.ones(shape))

self.beta = nn.Parameter(torch.zeros(shape))

# 非模型参数的变量初始化为0和1

self.moving_mean = torch.zeros(shape)

self.moving_var = torch.ones(shape)
```

```
def forward(self, X):

# 如果X不在内存上，将moving_mean和moving_var

# 复制到X所在显存上

if self.moving_mean.device != X.device:

self.moving_mean = self.moving_mean.to(X.device)

self.moving_var = self.moving_var.to(X.device)

# 保存更新过的moving_mean和moving_var

Y, self.moving_mean, self.moving_var = batch_norm(

X, self.gamma, self.beta, self.moving_mean,

self.moving_var, eps=1e-5, momentum=0.9)

return Y
```

- BatchNorm层插入的位置是卷积层/线性层后，激活函数前，卷积层使用LazyBatchNorm2d,线性层使用LazyBatchNorm1d
- BatchNorm层在训练模式下与推理模式下不同，训练模式下使用的mean,var都是基于当前批量的数据(也会结合训练以来的滑动平均值)，推理模式下使用的是完整数据集的mean,var
- BatchNorm层用于减缓梯度爆炸/消失，使优化更平滑。其最初解释是"减少数据内部的协变量偏移"，但这并不是一个有效的解释

# resnet

## 核心思想：

1.只要网络能学习到恒等层，那么加深网络就不太可能使网络效果变差

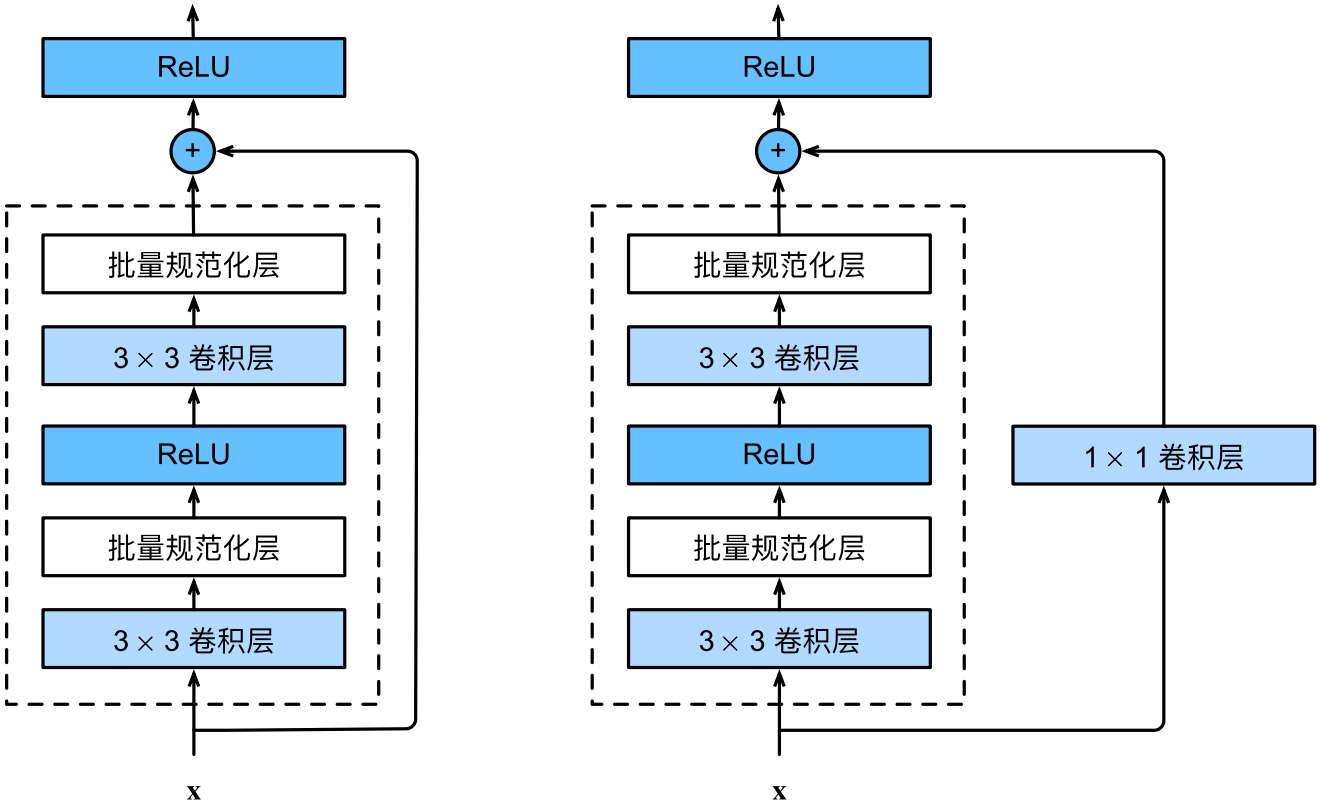
2.事实上由于激活函数等存在，网络难以学到恒等层

3.将resisual层的输入直接传到层内最后一个relu函数后，那么网络的输出变为F(X) + X，若需要学到恒等映射只需要学到F(X)横等于0,相对而言这是可行的

4.resisual层的输入与输出通道数不同时，在shortcut上用conv1x1调整

## resisual层的实现:

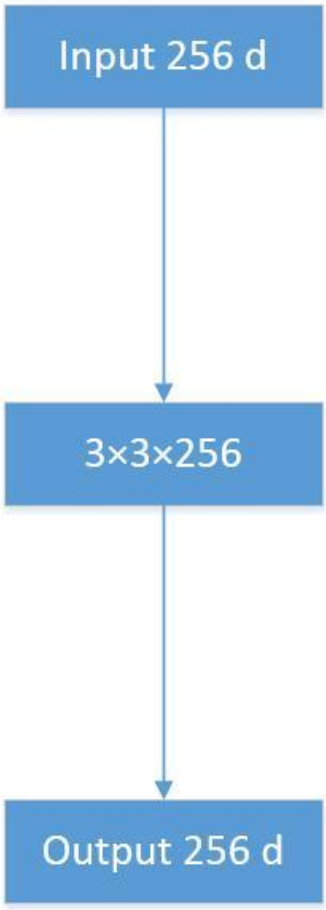
- 论文提出时使用的如下结构



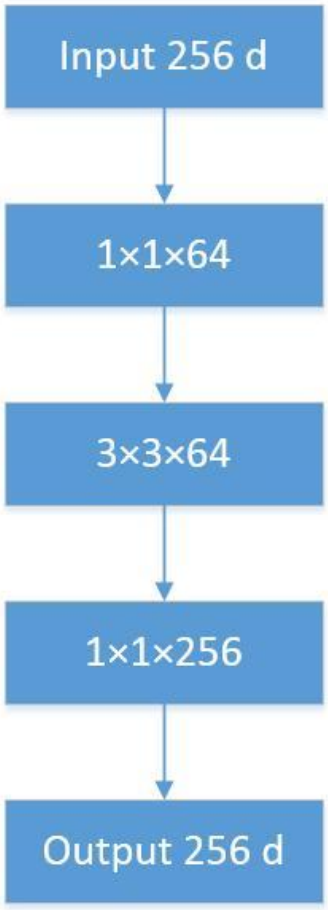

- 后来修改为BN1->relu->conv1->BN2->relu->conv2 + X or conv3(X)

## bottleneck层的实现

- 用1x1卷积降低通道数，用3x3卷积提取特征，再把特征图用1x1卷积还原到原通道数(shortcut部分不变)



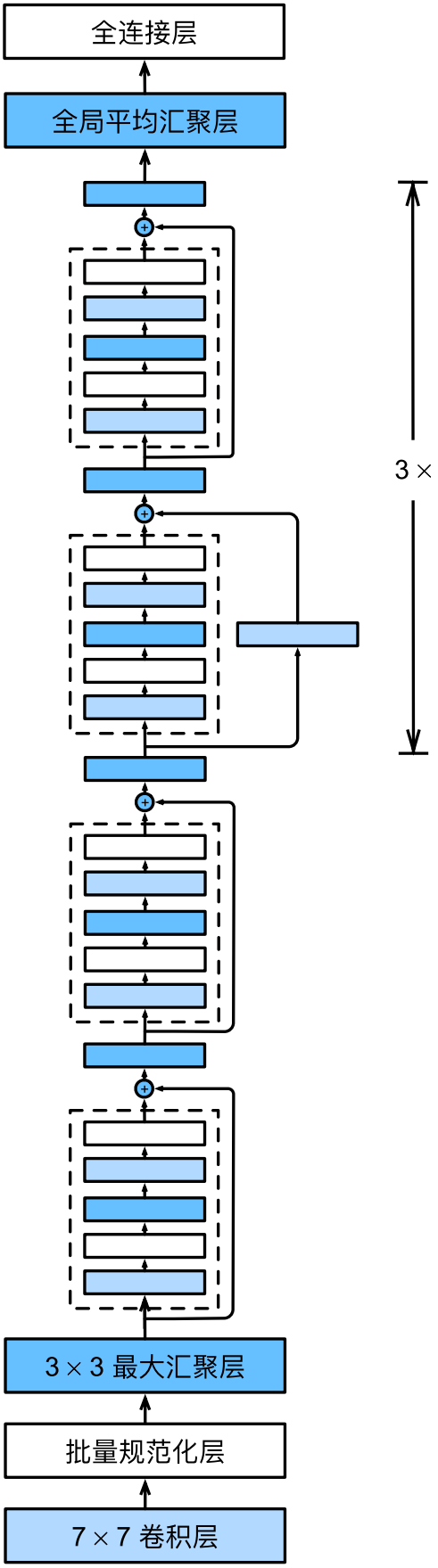

第一种




第二种

## resnet-18实现

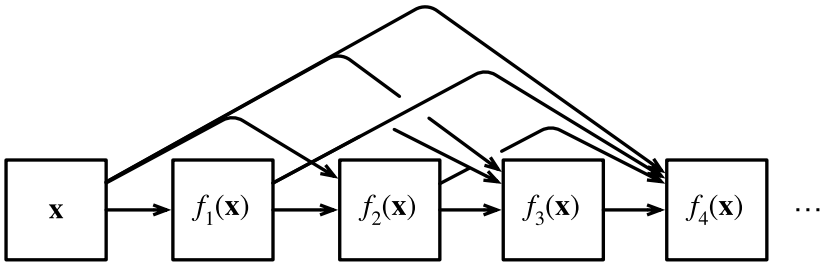
- 7x7卷积 + 3x3maxpool + 4个resisual_block(每个含两个resisual层) + 输出层
- 其中输出层实现为：全局avgpool + flatten + 全连接层



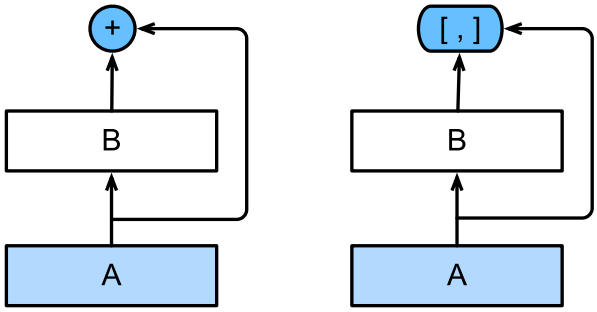

# Densenet

## 核心思想

- shortcut不仅可以发生在相邻的层之间，也可以发生在一定距离的层之间




- shortcut合并数据的方式可以是相加，也可以直接在通道维上叠加




## 实现

- DenseBlock = ((BN + ReLU + conv) + shortcut) * N + (BN + ReLU + conv)
- 可以注意到初始输入在每次进入卷积块时同时都会向后传递，在每个卷积块都参与计算
- TransitionBlock = (BN + ReLU + conv + avgpool（stride=2)）
- 过渡层用于减小通道数，同时减小图片分辨率，控制模型复杂度
- 网络前部的采样层与后部的输出层与resnet一致