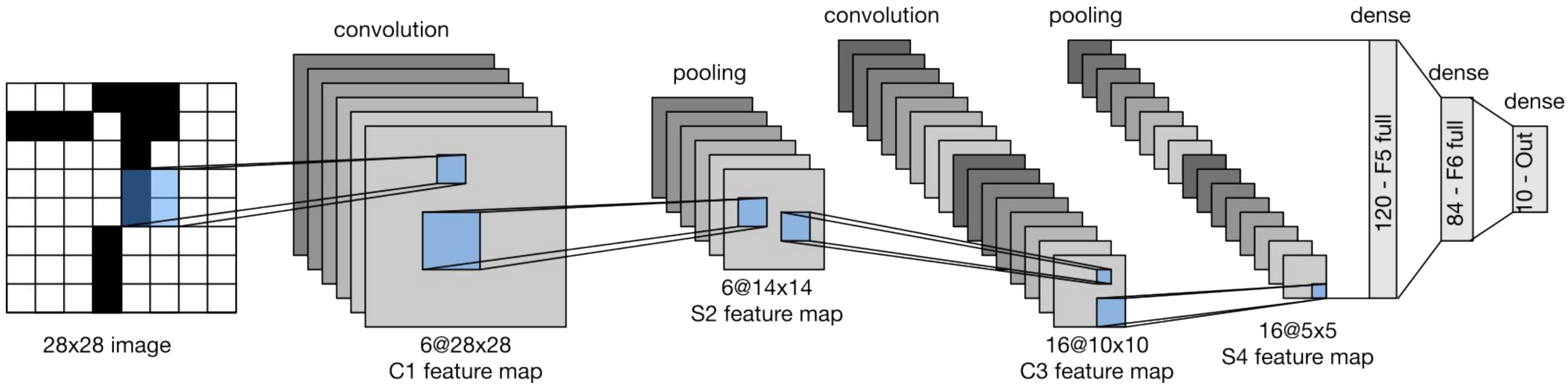
CSE 471: MACHINE LEARNING

Modern CNN architectures

LeNet



- ***Input: Hand written digits (single channel)***
- ***Output: Probability over 10 possible outcomes***
- ***At a high level, LeNet (LeNet-5) consists of two parts***
  - ***A convolutional encoder consisting of two convolutional layers***
  - ***A dense block consisting of three fully connected layers***

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., & others. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324.

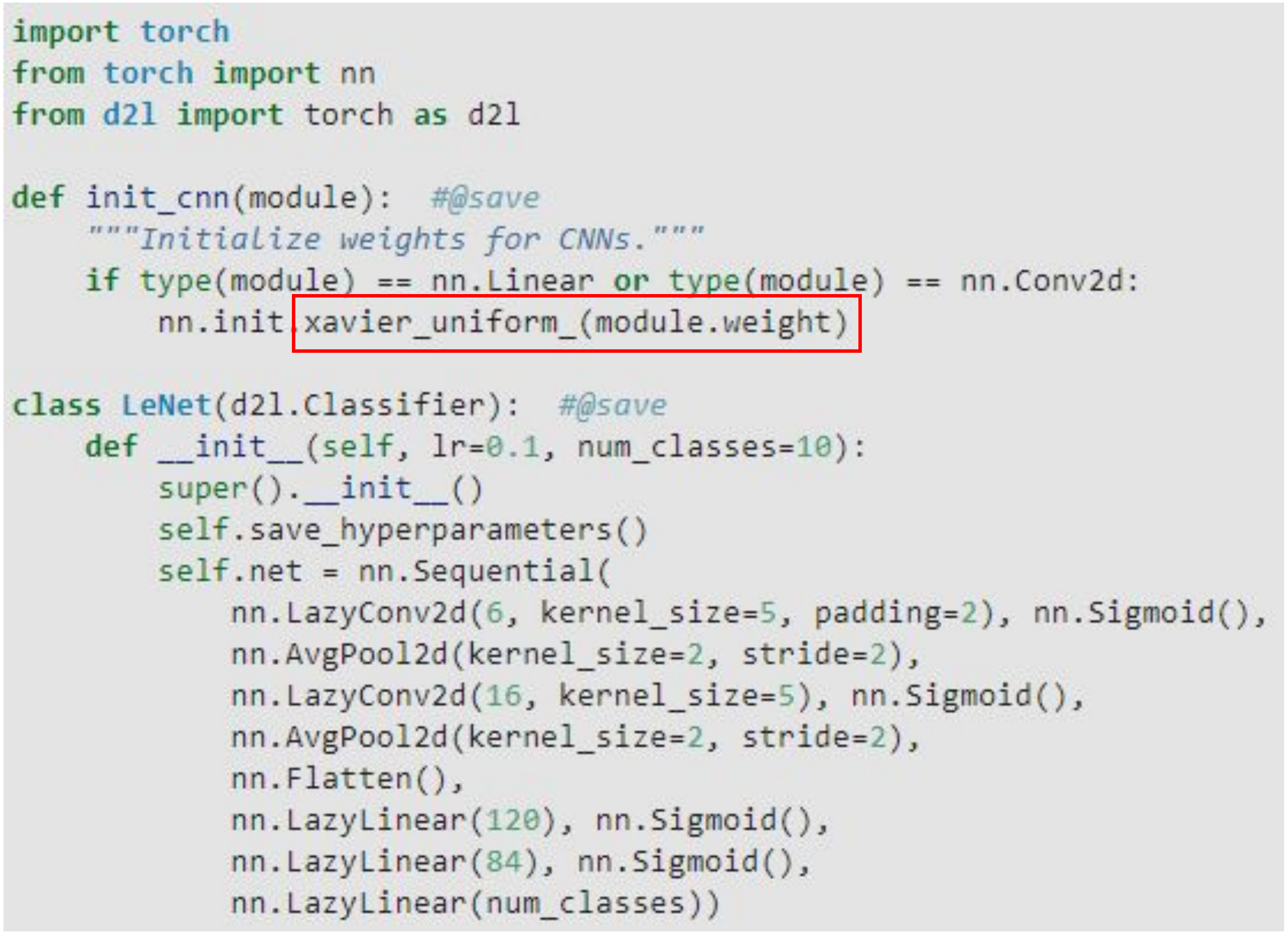
LeNet

- Convolution block
  - Convolutional layer (5 x 5 kernel)
  - Sigmoid activation
  - Average pooling
    - 2 x 2 (stride 2)
    - Spatial down sampling
  - Output channels
    - Layer 1: 6 @ 28 x 28
    - Layer 2: 16 @ 10 x 10
- Feature map is flattened before passing onto the dense layer

LeNet

- Dense block
  - 3 Fully connected layers
    - Layer 1: 120 neurons
    - Layer 2: 84 neurons
    - Layer 3: 10 neurons

LeNet (PyTorch)



```python
import torch
from torch import nn
from d2l import torch as d2l

def init_cnn(module):  #@save
    """Initialize weights for CNNs."""
    if type(module) == nn.Linear or type(module) == nn.Conv2d:
        nn.init.xavier_uniform_(module.weight)

class LeNet(d2l.Classifier):  #@save
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential(
            nn.LazyConv2d(6, kernel_size=5, padding=2), nn.Sigmoid(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.LazyConv2d(16, kernel_size=5), nn.Sigmoid(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.Flatten(),
            nn.LazyLinear(120), nn.Sigmoid(),
            nn.LazyLinear(84), nn.Sigmoid(),
            nn.LazyLinear(num_classes))
```

Xavier Initialization

- Let
  - O_i – output for some fully-connected layer (without nonlinearities)
  - There are n_{in} inputs x_i with associated weights w_{ij}
  - Weights are drawn independently from the same distribution, with 0 mean, σ^2 variance
  - x_i's also have 0 mean, γ^2 variance
    - Independent of weights
    - Independent of each other

$$\boxed{o_i = \sum_{j=1}^{n_{\text{in}}} w_{ij} x_j}$$

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249–256).

Xavier Initialization

$$
\begin{aligned}
E[o_i] &= \sum_{j=1}^{n_{\text{in}}} E[w_{ij}x_j] \\
&= \sum_{j=1}^{n_{\text{in}}} E[w_{ij}]E[x_j] \\
&= 0, \\
\text{Var}[o_i] &= E[o_i^2] - (E[o_i])^2 \\
&= \sum_{j=1}^{n_{\text{in}}} E[w_{ij}^2 x_j^2] - 0 \\
&= \sum_{j=1}^{n_{\text{in}}} E[w_{ij}^2]E[x_j^2] \\
&= n_{\text{in}}\sigma^2\gamma^2.
\end{aligned}
$$

- Variance can be kept fixed if
  - **$n_{in}\sigma^2 = 1$**
- Following same reasoning, during backprop. Gradients' variance can be kept fixed if
  - **$n_{out}\sigma^2 = 1$**
- Therefore, we try to achieve
  - **$0.5 \times (n_{in} + n_{out})\ \sigma^2 = 1$**

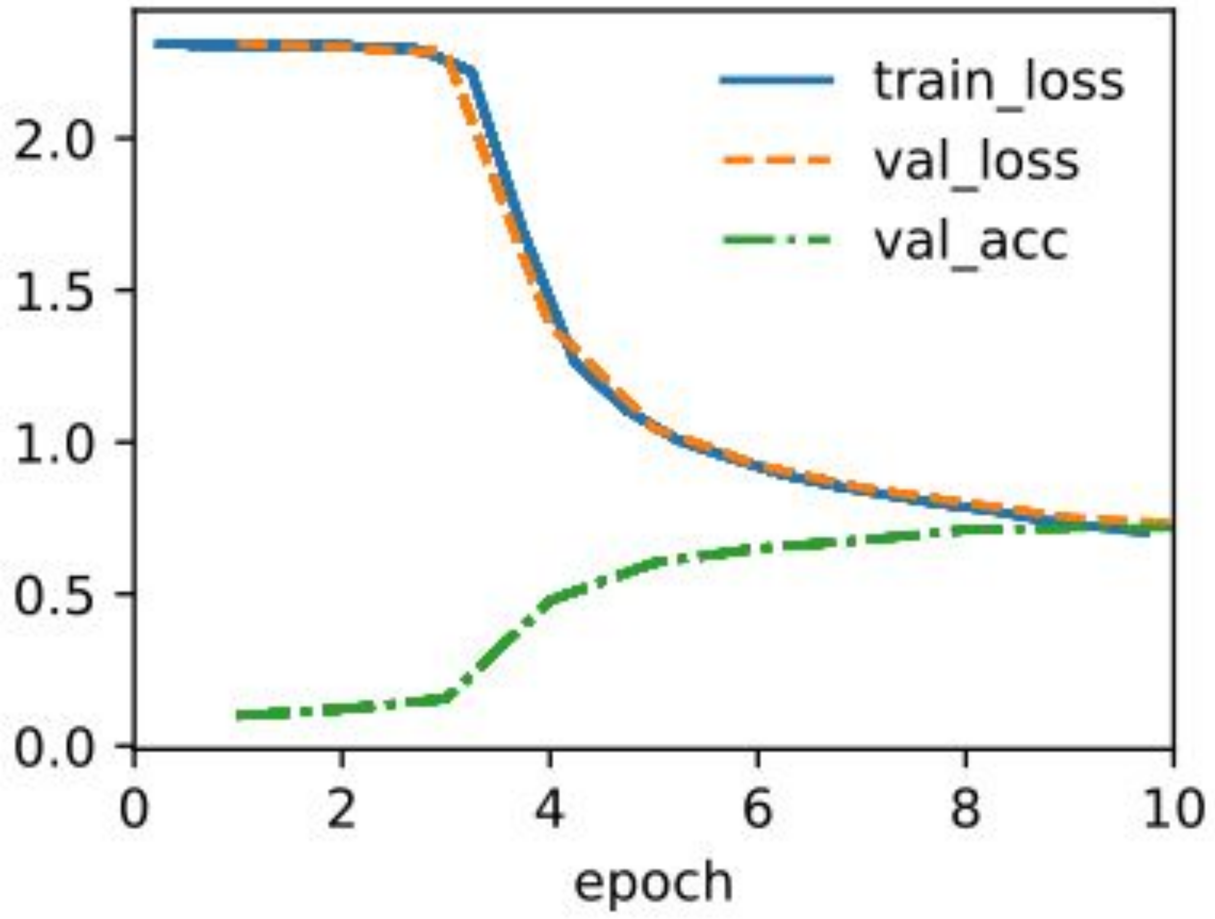
$$\boxed{\sigma = \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}}}$$

Xavier Initialization

- Sampling weights from *N(0, σ²)*
- Sampling weights from uniform distribution *U*(-*a*, *a*)

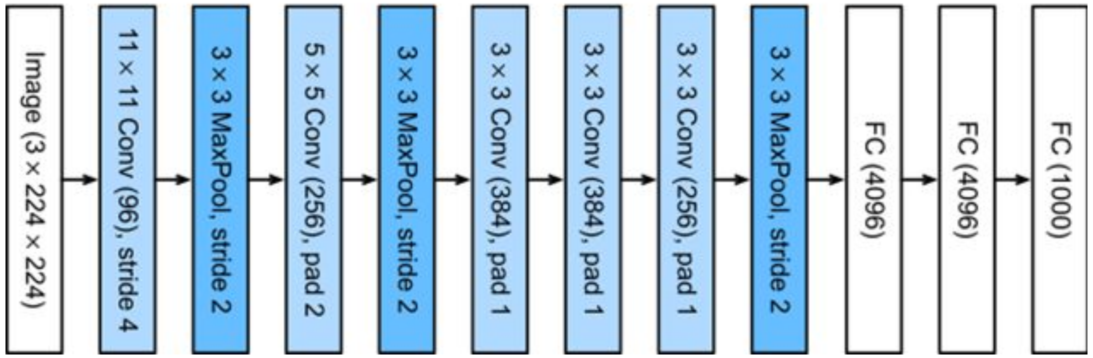
$$\boxed{U\left(-\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}\right)}$$

LeNet



AlexNet

- Runs on GPU hardware
- Won the ImageNet Large Scale Visual Recognition Challenge 2012 by a phenomenally large margin
- Architecture
  - 5 Convolutional layers
  - 3 fully connected layers
  - ReLU activation



Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems (pp. 1097–1105).

AlexNet

- Input: 224 x 224 3-channel
- 11 x 11 filter in the first layer
- 10 times more convolution channels/filters than LeNet
- Uses dropout
- Image augmentation
  - Flipping
  - Clipping
  - Color changes

Dropout

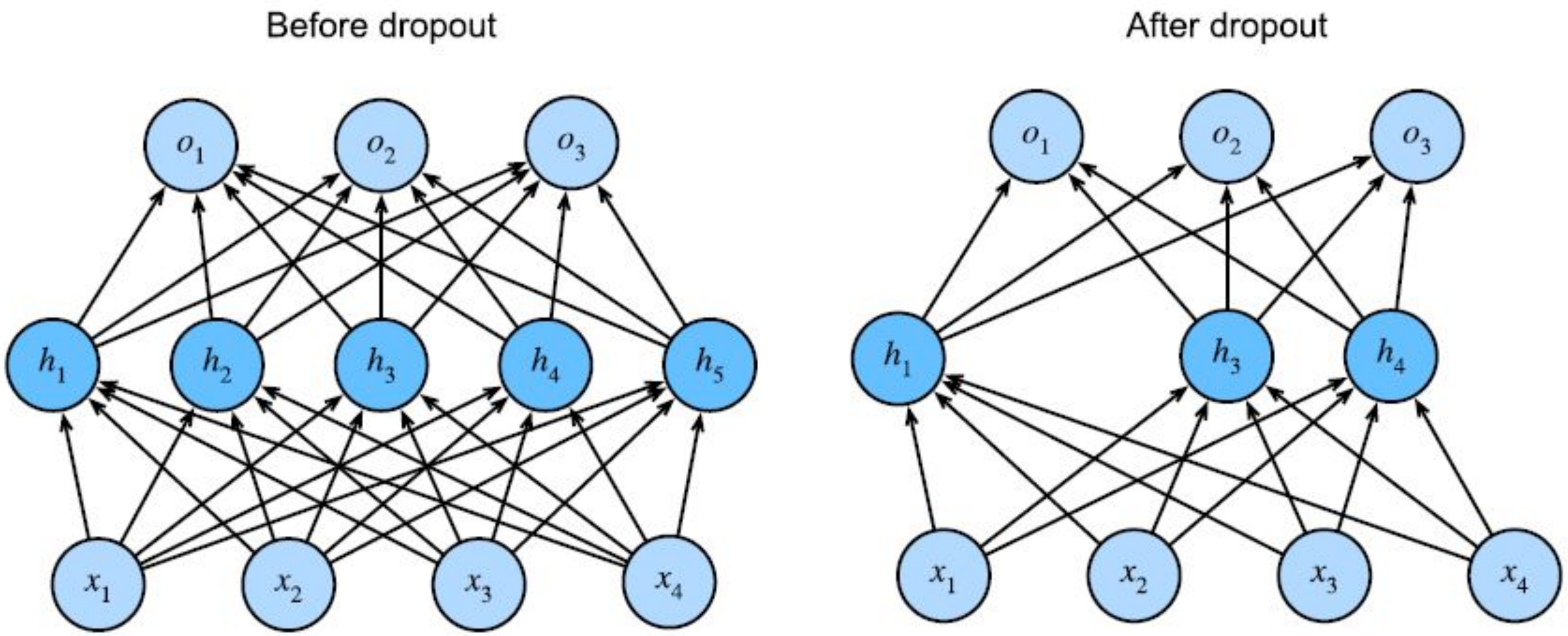


Fig. 4.6.1: MLP before and after dropout.

Dropout

- Drop out some neurons during training
  - On each iteration
  - Layer by layer
  - Different neurons will get dropped in different iterations
- Breaks up co-adaptation

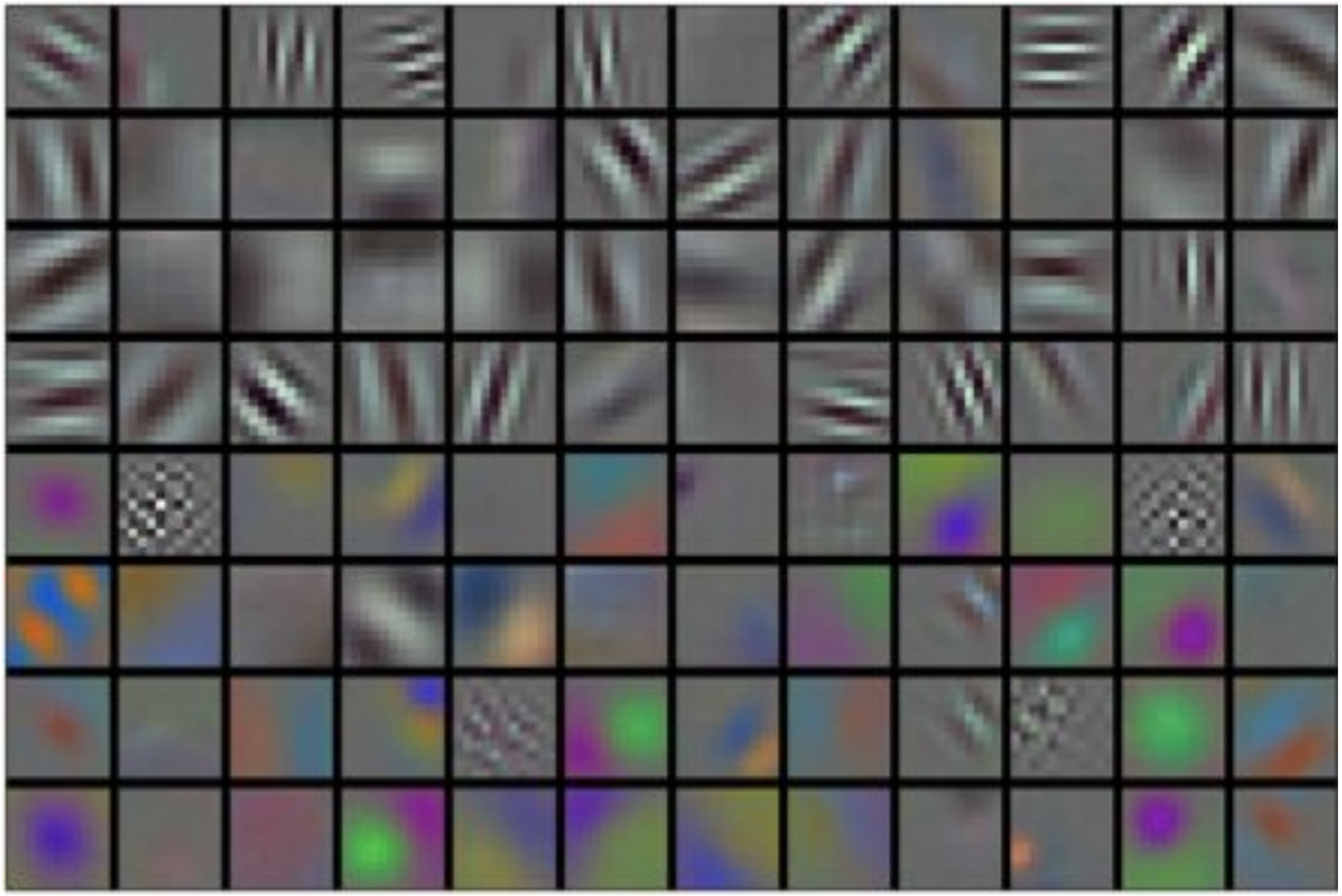
Co-adaptation. *Neural network overfitting is characterized by a state in which each layer relies on a specific pattern of activations in the previous layer.*

Dropout

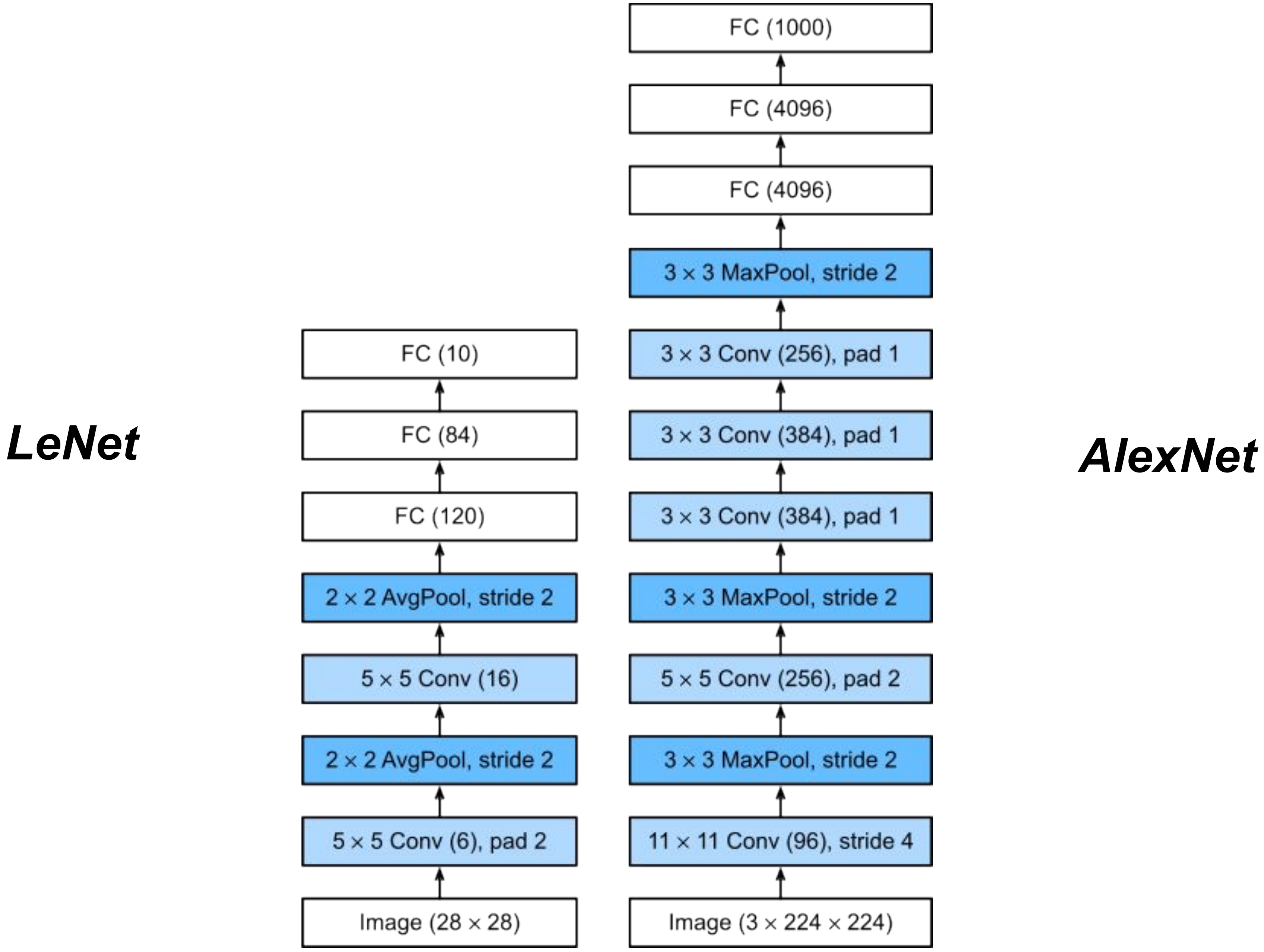
- Need to normalize the activation of the retained nodes
- Each intermediate activation *h* is replaced by a random variable *h'*
- Expectation remains unchanged, i.e., E[h'] = h.

$$\boxed{h' = \begin{cases} 0 & \text{with probability } p \\ \frac{h}{1-p} & \text{otherwise} \end{cases}}$$

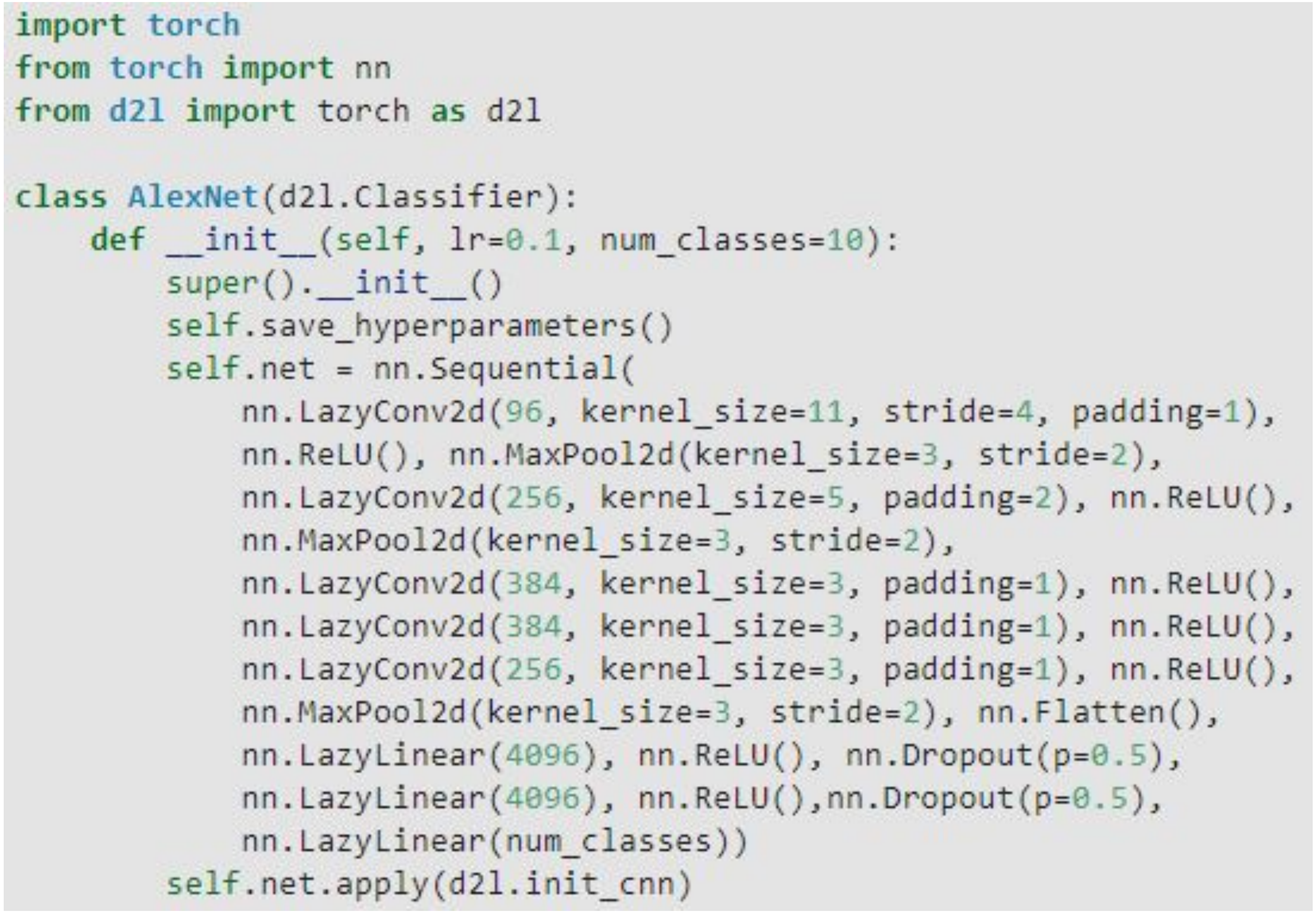
Learned filters (96)



AlexNet


LeNet
FC (10)
FC (84)
FC (120)
2 × 2 AvgPool, stride 2
5 × 5 Conv (16)
2 × 2 AvgPool, stride 2
5 × 5 Conv (6), pad 2
Image (28 × 28)
AlexNet
FC (1000)
FC (4096)
FC (4096)
3 × 3 MaxPool, stride 2
3 × 3 Conv (256), pad 1
3 × 3 Conv (384), pad 1
3 × 3 Conv (384), pad 1
3 × 3 MaxPool, stride 2
5 × 5 Conv (256), pad 2
3 × 3 MaxPool, stride 2
11 × 11 Conv (96), stride 4
Image (3 × 224 × 224)

AlexNet (PyTorch)



```python
import torch
from torch import nn
from d2l import torch as d2l

class AlexNet(d2l.Classifier):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential(
            nn.LazyConv2d(96, kernel_size=11, stride=4, padding=1),
            nn.ReLU(), nn.MaxPool2d(kernel_size=3, stride=2),
            nn.LazyConv2d(256, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.LazyConv2d(384, kernel_size=3, padding=1), nn.ReLU(),
            nn.LazyConv2d(384, kernel_size=3, padding=1), nn.ReLU(),
            nn.LazyConv2d(256, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2), nn.Flatten(),
            nn.LazyLinear(4096), nn.ReLU(), nn.Dropout(p=0.5),
            nn.LazyLinear(4096), nn.ReLU(),nn.Dropout(p=0.5),
            nn.LazyLinear(num_classes))
        self.net.apply(d2l.init_cnn)
```

VGG

- Visual Geometry Group (VGG) at Oxford University
- Neurons Layers Blocks
- Basic VGG block
  - A convolution layer with padding
  - A nonlinearity (e.g. ReLU)
  - A pooling layer (e.g. max pooling)

In the original VGG paper, the authors employed convolutions with 3x3 kernels with padding of 1 (keeping height and width) and 2x2 max pooling with stride of 2 (halving the resolution after each block)

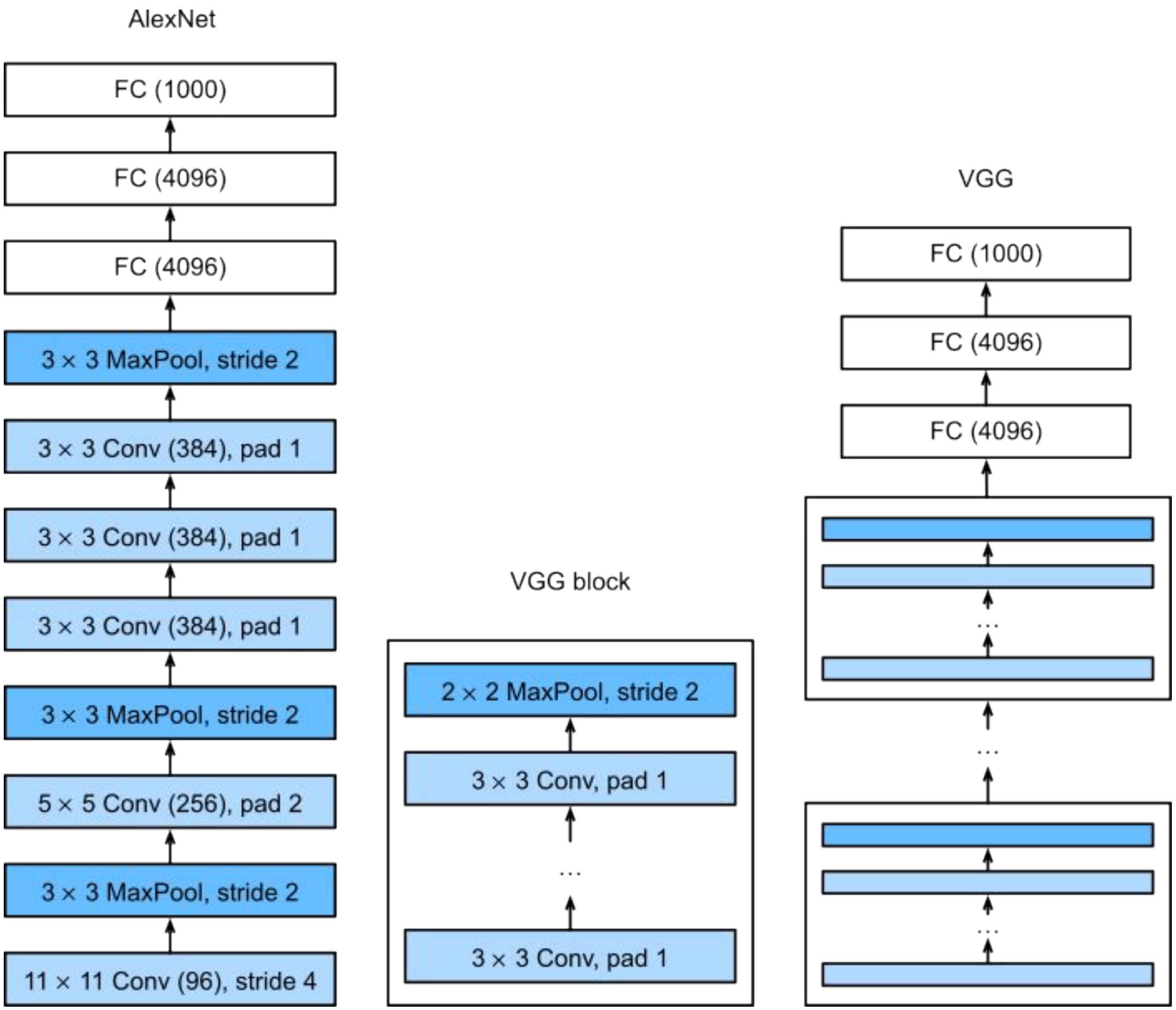
Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

VGG

```
import torch
from torch import nn
from d2l import torch as d2l


def vgg_block(num_convs, out_channels):
    layers = []
    for _ in range(num_convs):
        layers.append(nn.LazyConv2d(out_channels, kernel_size=3, padding=1))
        layers.append(nn.ReLU())
    layers.append(nn.MaxPool2d(kernel_size=2,stride=2))
    return nn.Sequential(*layers)
```

VGG

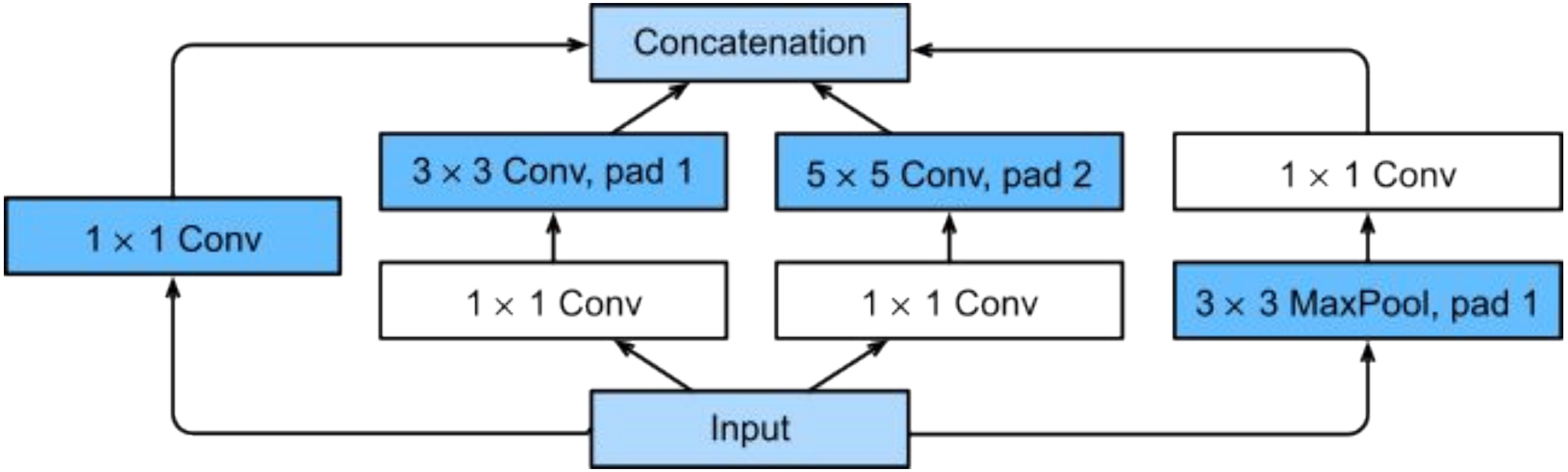

AlexNet
FC (1000)
FC (4096)
FC (4096)
3 × 3 MaxPool, stride 2
3 × 3 Conv (384), pad 1
3 × 3 Conv (384), pad 1
3 × 3 Conv (384), pad 1
3 × 3 MaxPool, stride 2
5 × 5 Conv (256), pad 2
3 × 3 MaxPool, stride 2
11 × 11 Conv (96), stride 4
VGG block
2 × 2 MaxPool, stride 2
3 × 3 Conv, pad 1
...
3 × 3 Conv, pad 1
VGG
FC (1000)
FC (4096)
FC (4096)
...

Original VGG network

- 5 convolutional blocks
  - Block# 1, 2: 1 Conv. layer each
  - Block# 3, 4, 5: 2 Conv. layer each
- Fully connected block
  - Same as AlexNet
- Called VGG-11
  - 8 Conv. Layers
  - 3 FC layers
- Uses dropout

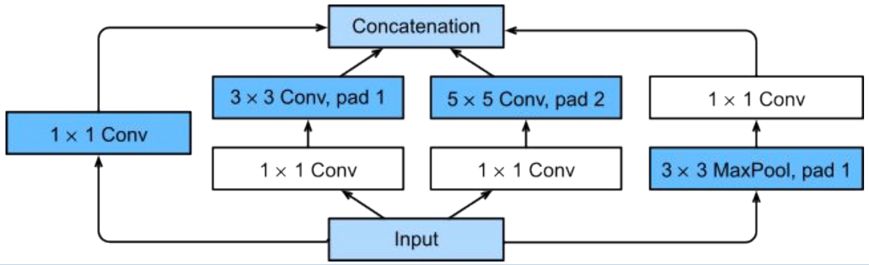
GoogLeNet

- Won ImageNet challenge in 2014.
- Investigated which sized kernels are best.
  - Employ a combination of variously-sized kernels
- The basic block is called Inception Block.



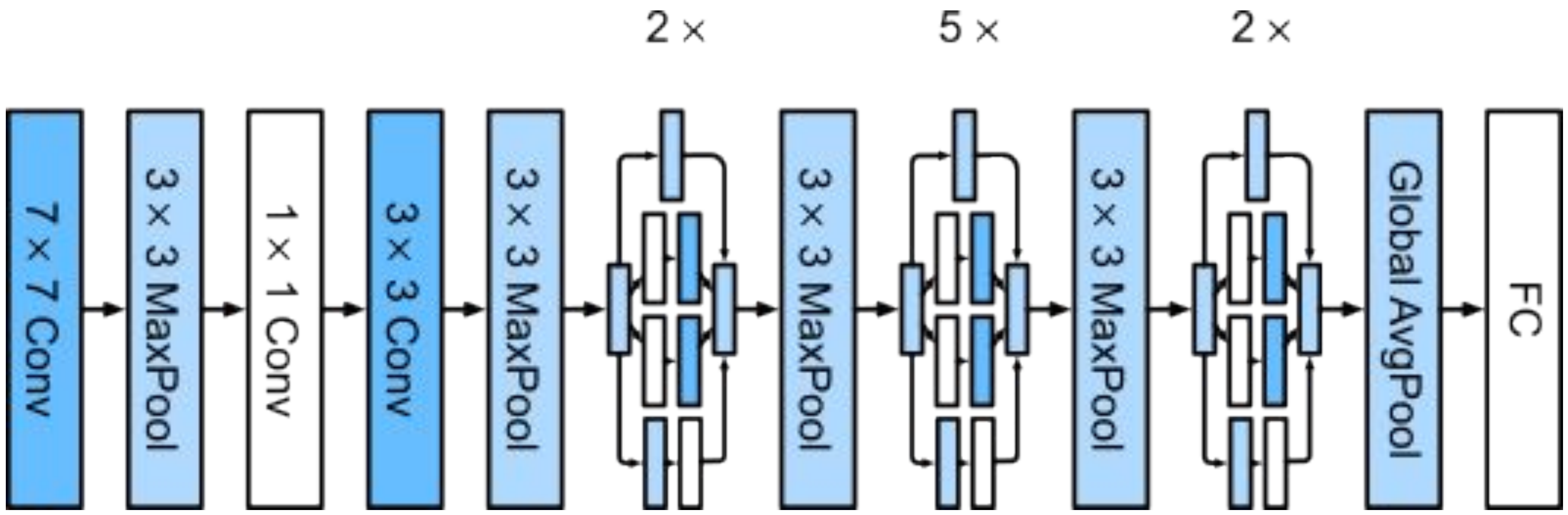
Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going deeper with convolutions. Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1–9).

Inception Block



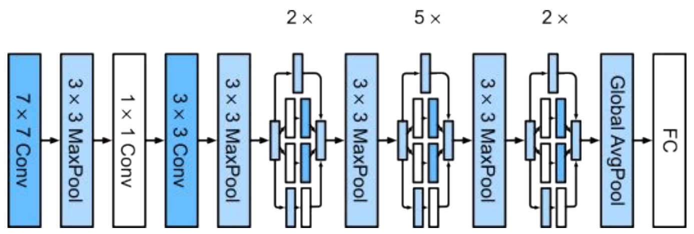
- 4 parallel paths
  - Path# 1: 1x 1 filter
  - Path# 2: 3 x 3 filter, pad = 1
    - 1 x 1 filter used beforehand to reduce channels
  - Path# 3: 5 x 5 filter, pad = 2
    - 1 x 1 filter used beforehand to reduce channels
  - Path# 4: 3 x 3 MaxPool, pad = 1
    - 1 x 1 filter used afterwards to reduce channels
- Input and output have the same height and width
- Channel count varies in the different paths and are concatenated

GoogLeNet



2 ×
5 ×
2 ×
7 × 7 Conv
3 × 3 MaxPool
1 × 1 Conv
3 × 3 Conv
3 × 3 MaxPool
3 × 3 MaxPool
3 × 3 MaxPool
Global AvgPool
FC

GoogLeNet



- 7x7 filter, stride=2, pad=3, 64 channels
- 3x3 maxpooling
- 1x1 filter – 64 channels
- 3x3 filter – 192 channels
- 2 inception blocks in series
  - Block# 1: 64 + 128 + 32 + 32 = 256 channels
  - Block #2: 128 + 192 + 96 + 64 = 480 channels
- And so on …

Residual networks (ResNet)

$$\boxed{\mathbf{z}^{(i)} = f(\mathbf{z}^{(i-1)}) = \mathbf{g}^{(i)}(\mathbf{W}^{(i)}\mathbf{z}^{(i-1)})}$$

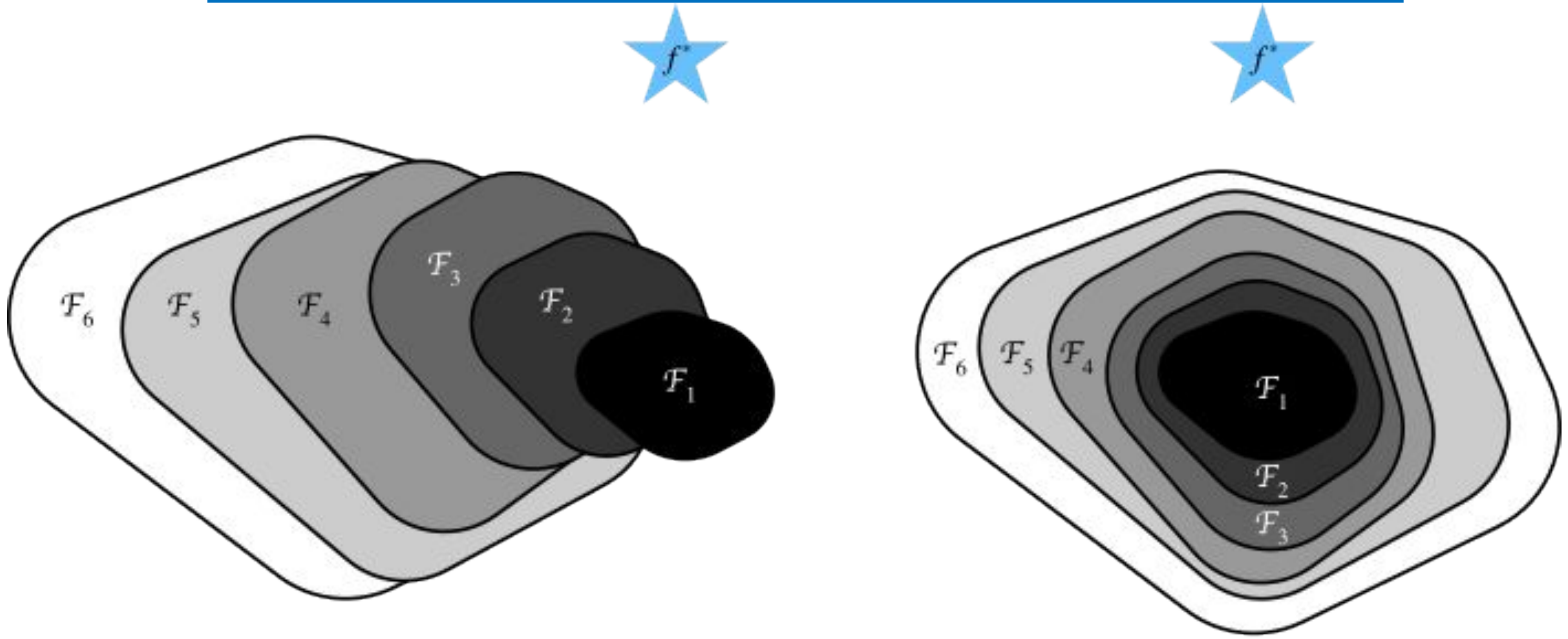
Each layer completely replaces the representation from the preceding layer,

$$\mathbf{z}^{(i)} = \mathbf{g}_r^{(i)}(\mathbf{z}^{(i-1)} + f(\mathbf{z}^{(i-1)}))$$

Whereas traditional networks must learn to propagate information and are subject to catastrophic failure of information propagation for bad choices of the parameters, residual networks propagate information by default

Functional classes

$$f^*_{\mathcal{F}} := \operatorname*{argmin}_{f} L(\mathbf{X}, \mathbf{y}, f) \text{ subject to } f \in \mathcal{F}$$



Non-nested function classes

Nested function classes

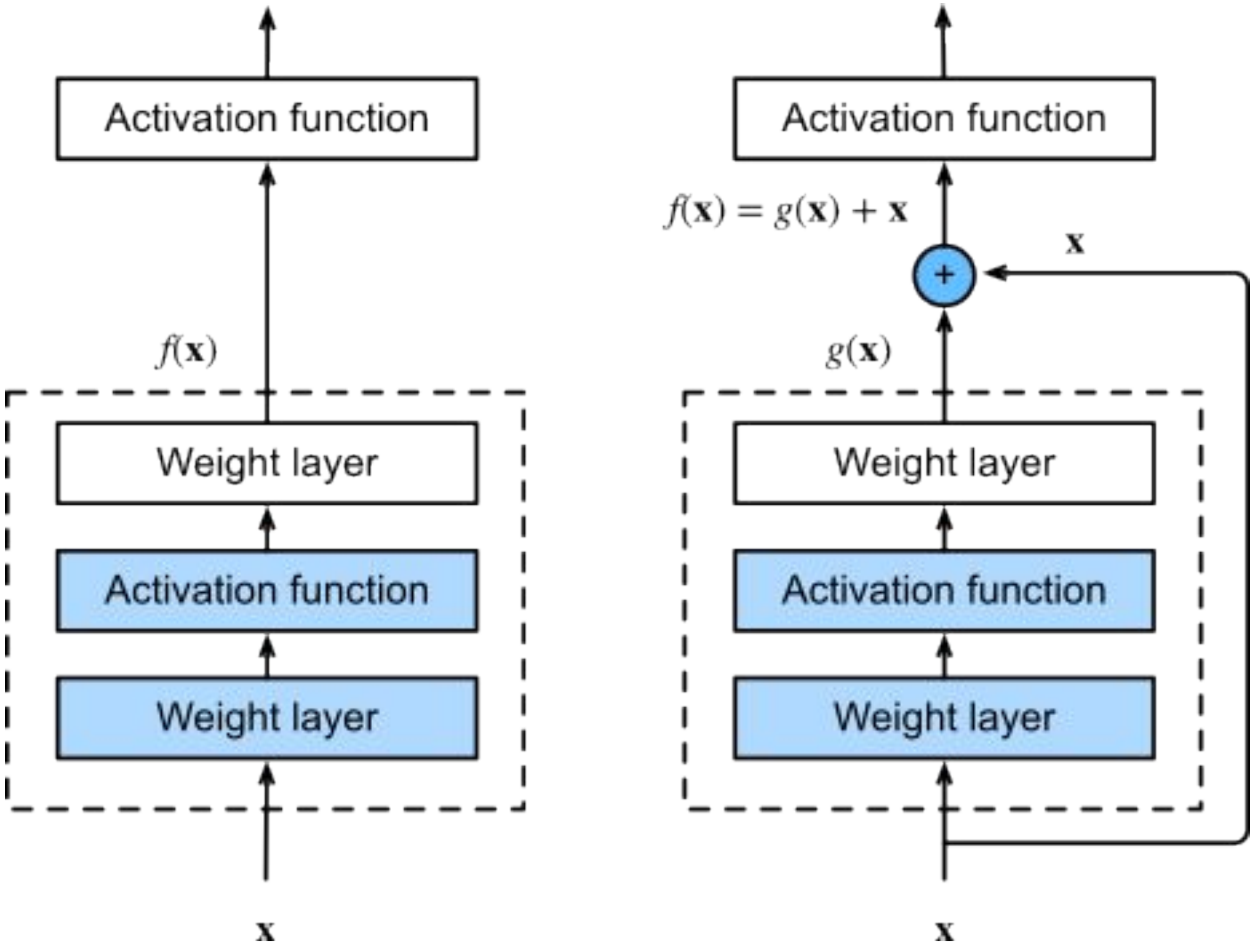
For non-nested function classes, a larger (indicated by area) function class does not guarantee to get closer to the "truth" function (f^*). This does not happen in nested function classes.

ResNet (Intuition)

- For deep neural networks, if we can train the newly-added layer into an identity function $f(x) = x$, the new model will be as effective as the original model.
- As the new model may get a better solution to fit the training dataset, the added layer might make it easier to reduce training errors.

Won the ImageNet Large Scale Visual Recognition Challenge in 2015.

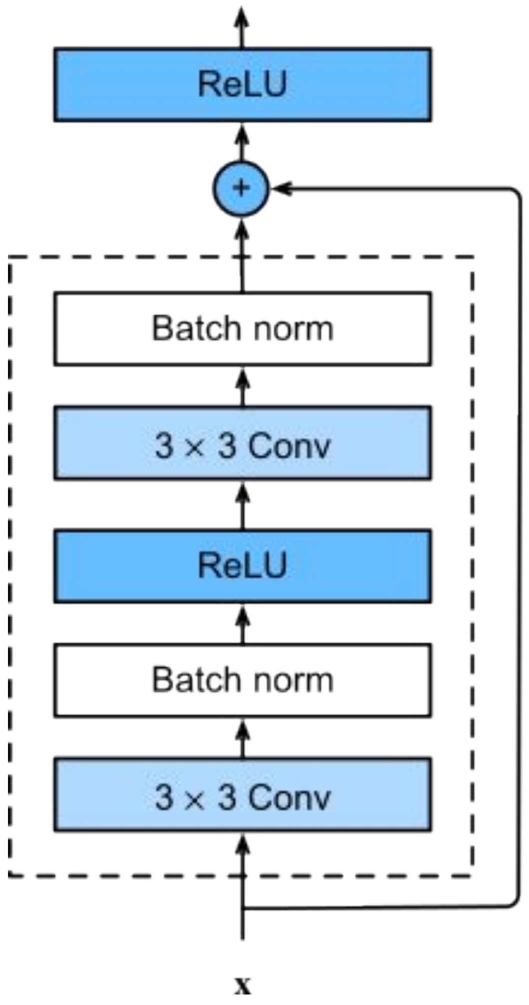
ResNet Block



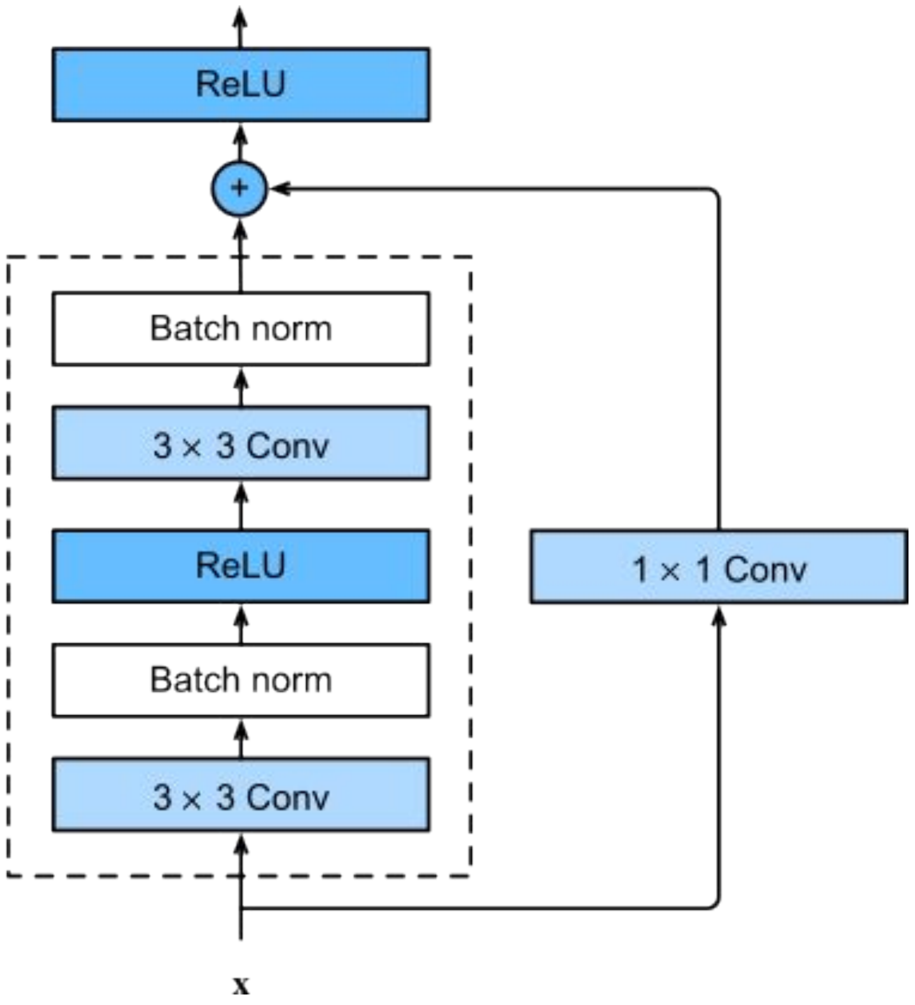
ResNet Block

- Two 3x3 convolution layers
  - Same number of output channels
- Batch normalization
- ReLU activation

ResNet Block

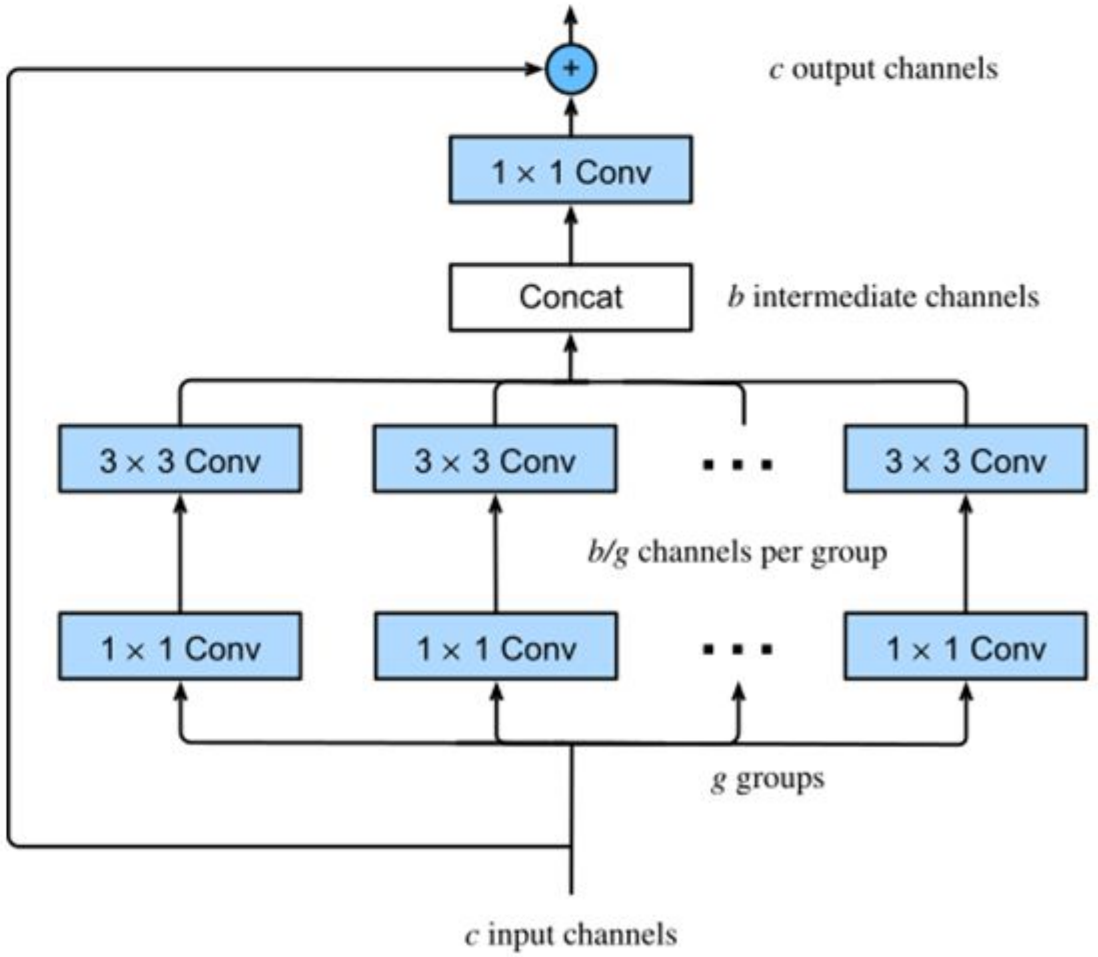


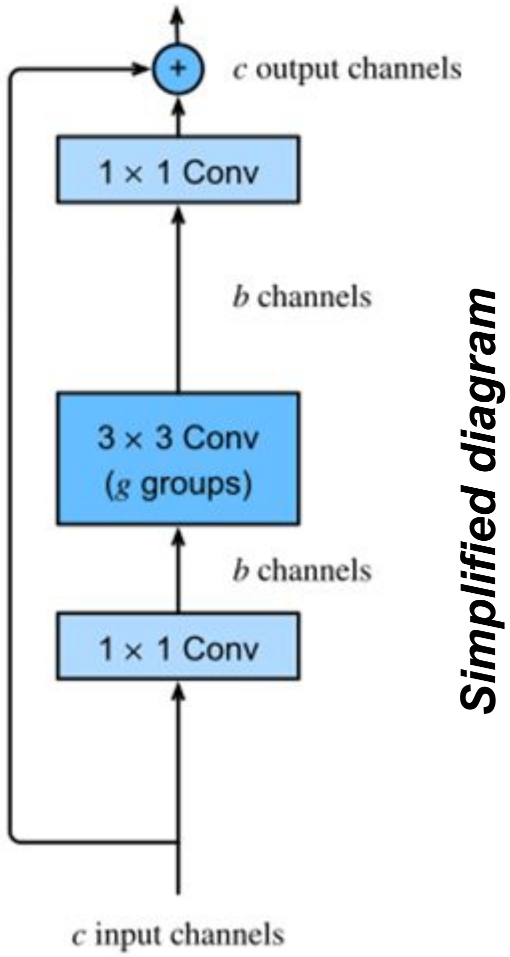
Identical input/output channels



Non-identical input/output channels

ResNext block





The use of grouped convolution with g groups is g times faster than a dense convolution. It is a bottleneck residual block when the number of intermediate channels b is less than c.