# Introduction to Deep Learning

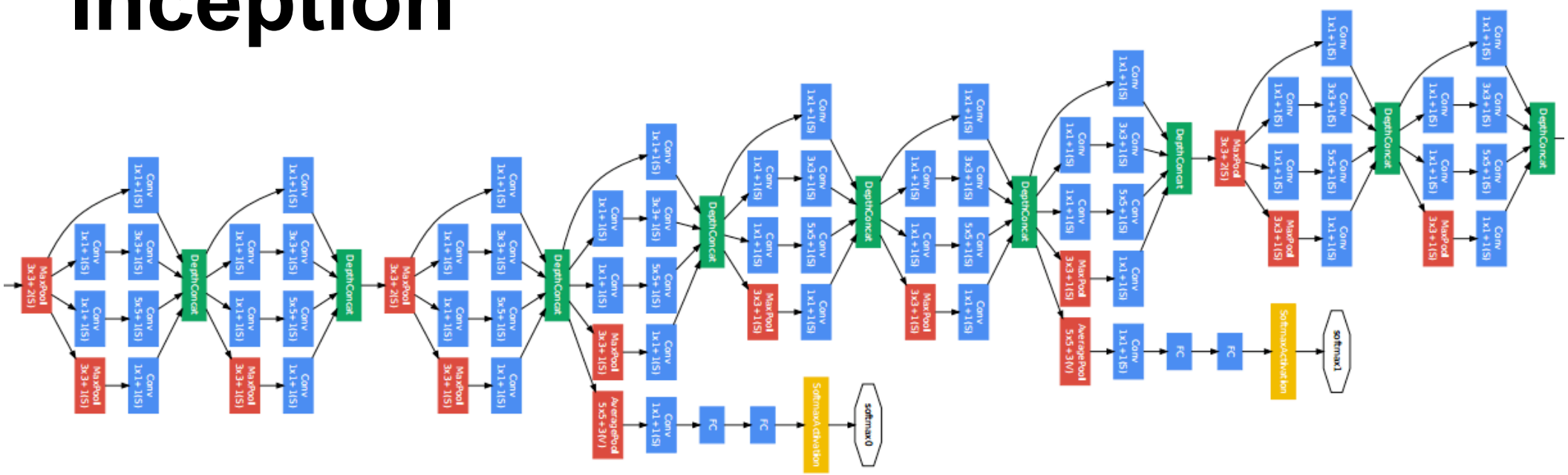
## Advanced Convolutional Neural Networks

## MGMT 735 DEEP LEARNING IN FINANCE

**Alex Smola and Mu Li**

courses.d2l.ai/berkeley-stat-157

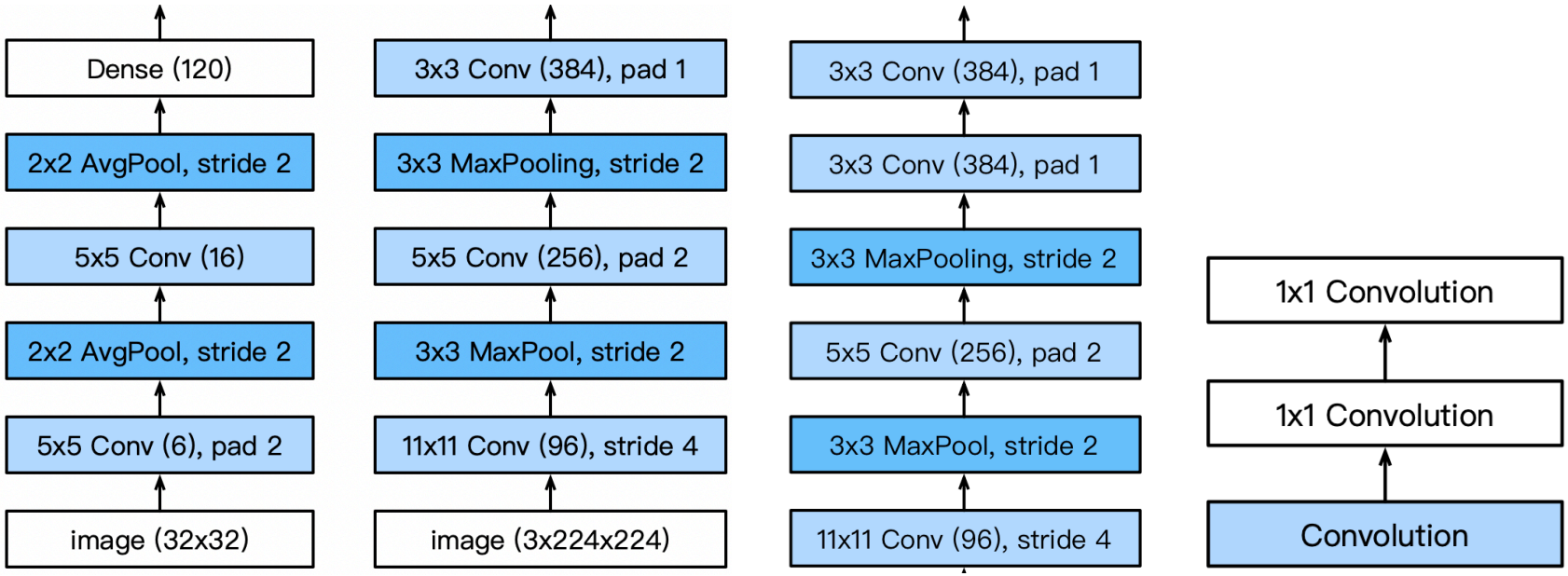
# Inception


MaxPool 3x3+ 2(S)
Conv 1x1+ 1(S)
Conv 3x3+ 1(S)
Conv 5x5+ 1(S)
MaxPool 3x3+ 1(S)
DepthConcat
AveragePool 5x5+ 3(V)
FC
SoftmaxActivation
softmax0
softmax1

# Picking the best convolution …

1x1 | 3x3 | 5x5 | Max pooling | Multiple 1x1

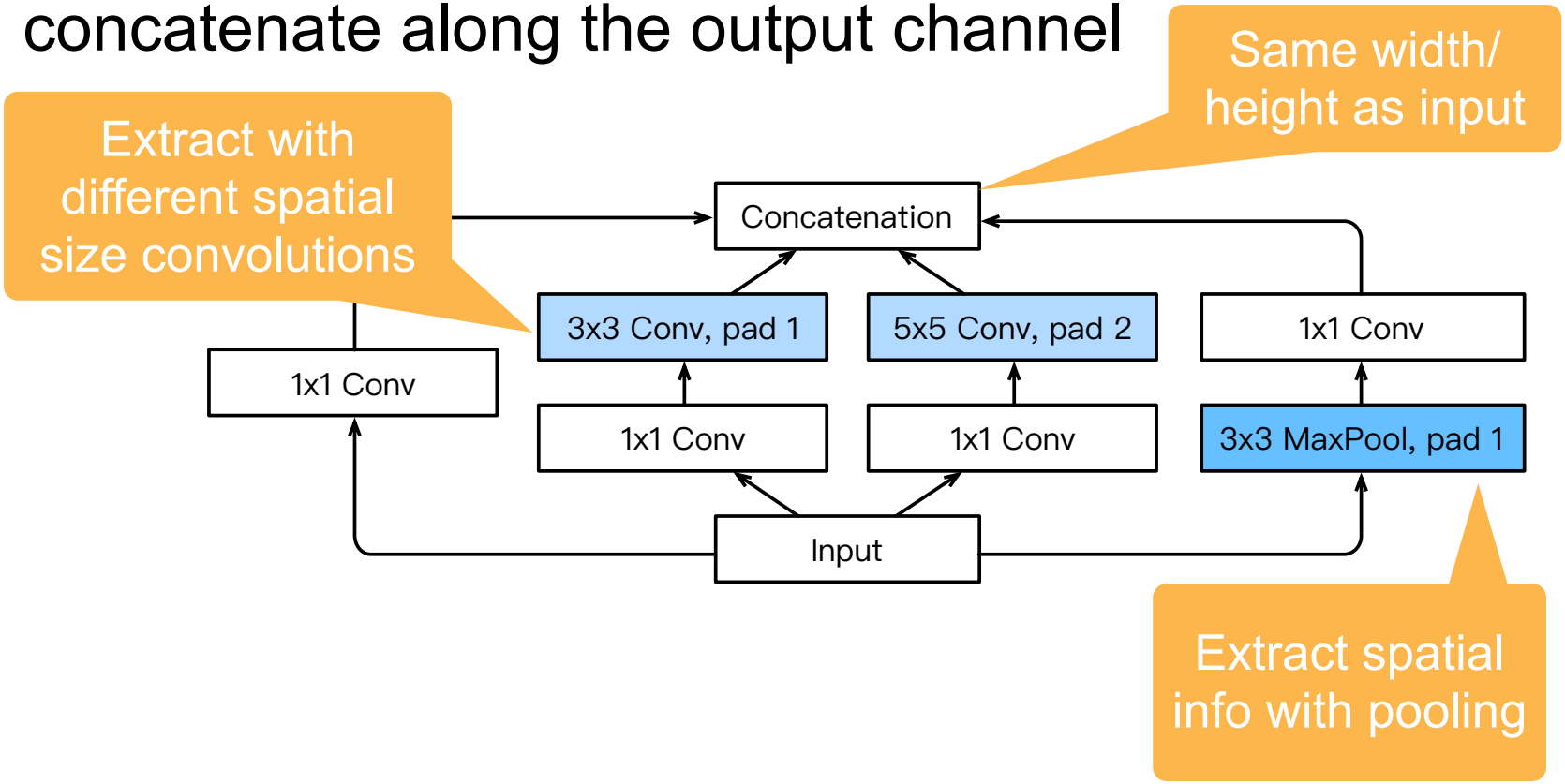
LeNet | AlexNet | VGG | NiN



# Why choose? Just pick them all.

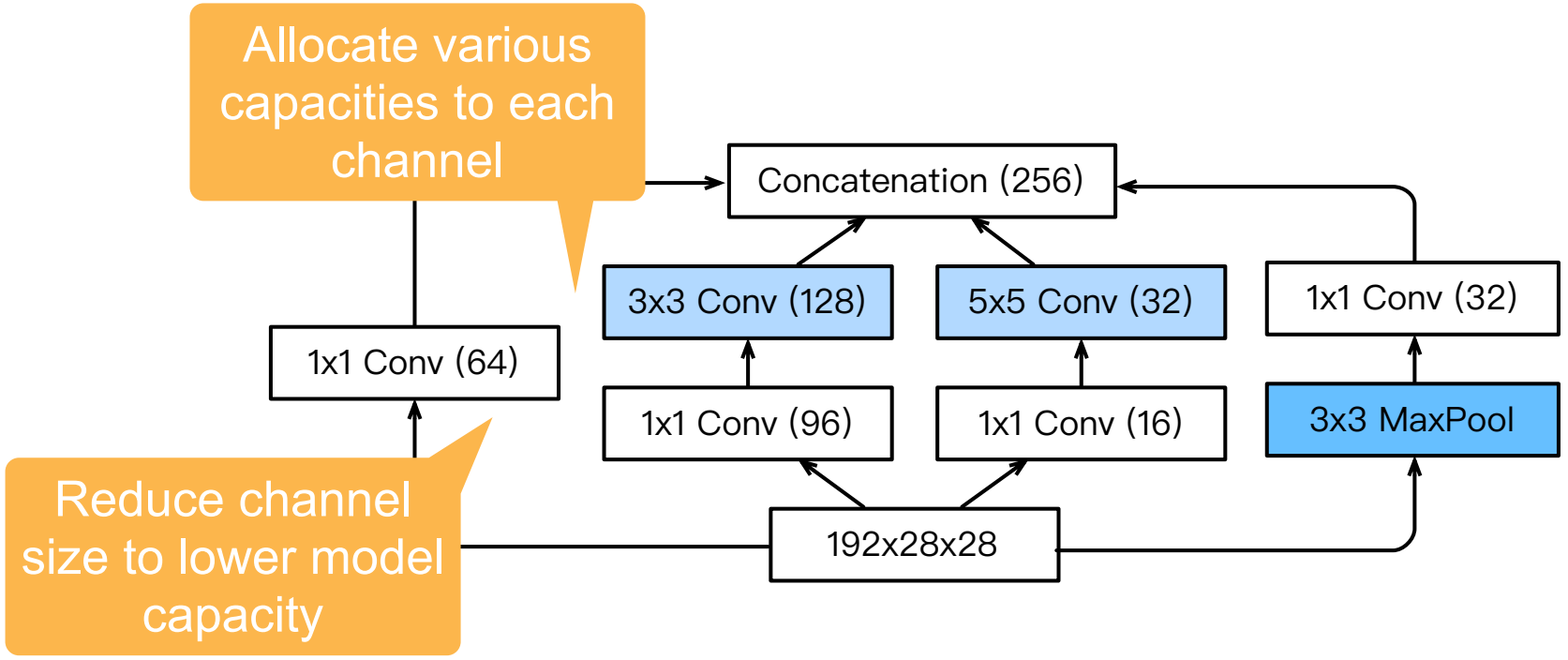
# Inception Blocks

4 paths extract information from different aspects, then concatenate along the output channel



# Inception Blocks

The first inception block with channel sizes specified

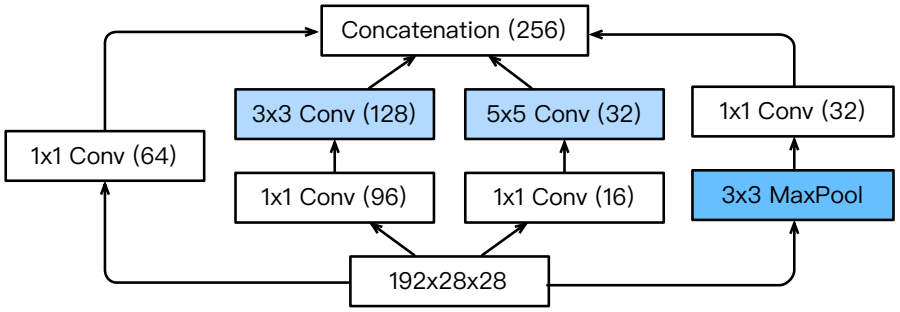


# Inception Blocks

Inception blocks have fewer parameters and less computation complexity than a single 3x3 or 5x5 convolutional layer (smaller)

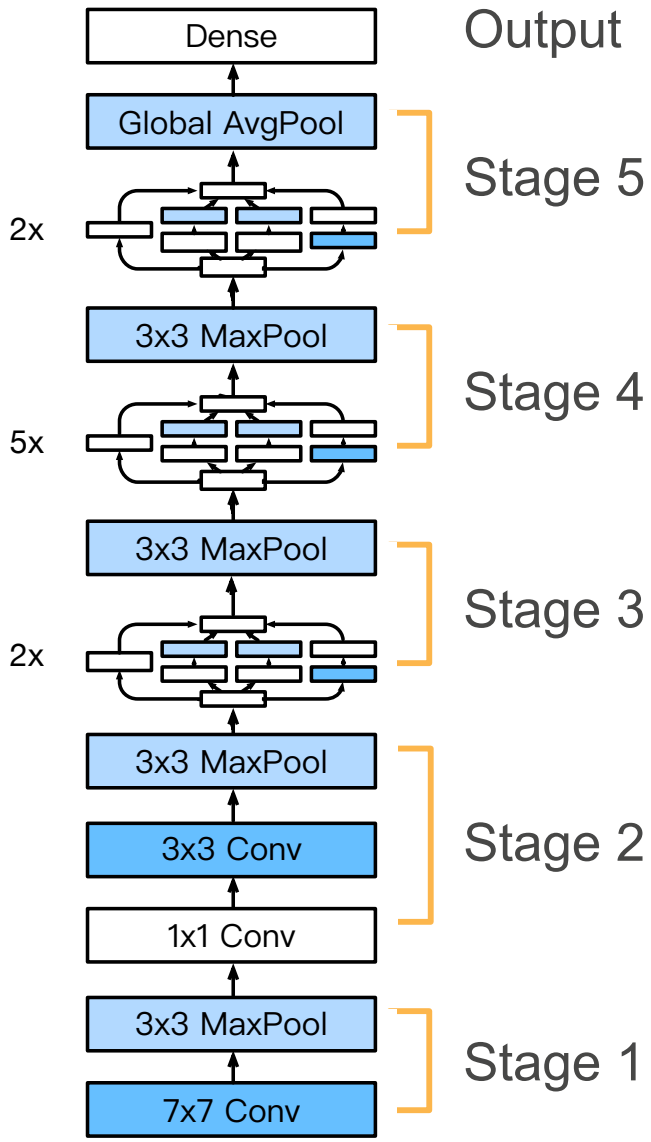
- Mix of different functions (powerful function class)
- Memory and compute efficiency (good generalization)

| | #parameters | FLOPS |
|---|---|---|
| **Inception** | 0.16 M | 128 M |
| **3x3 Conv** | 0.44 M | 346 M |
| **5x5 Conv** | 1.22 M | 963 M |



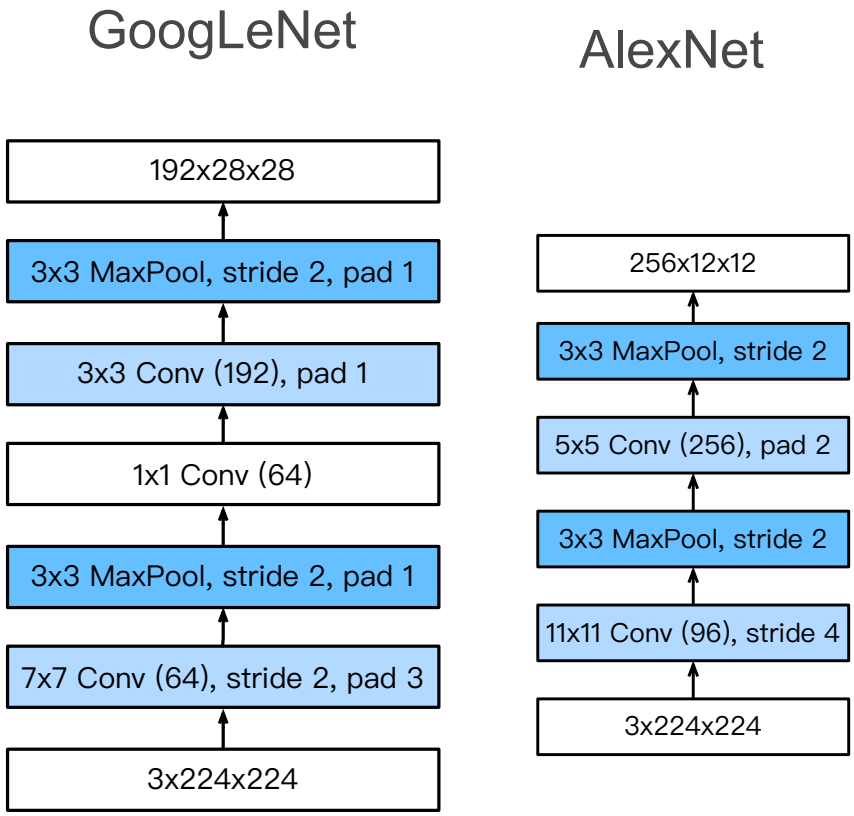
# GoogLeNet

- 5 stages with 9 inceptions blocks

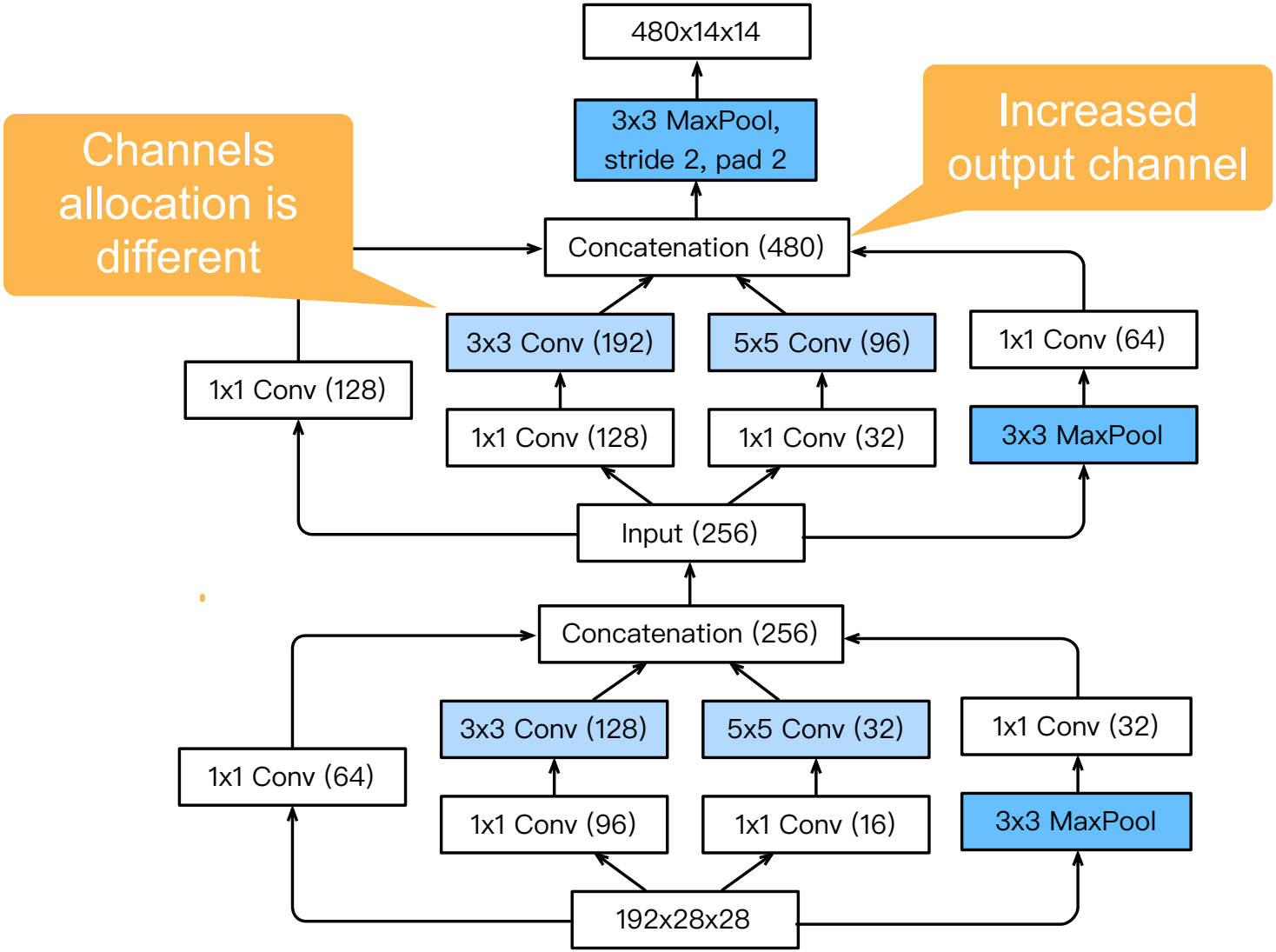


# Stage 1 & 2

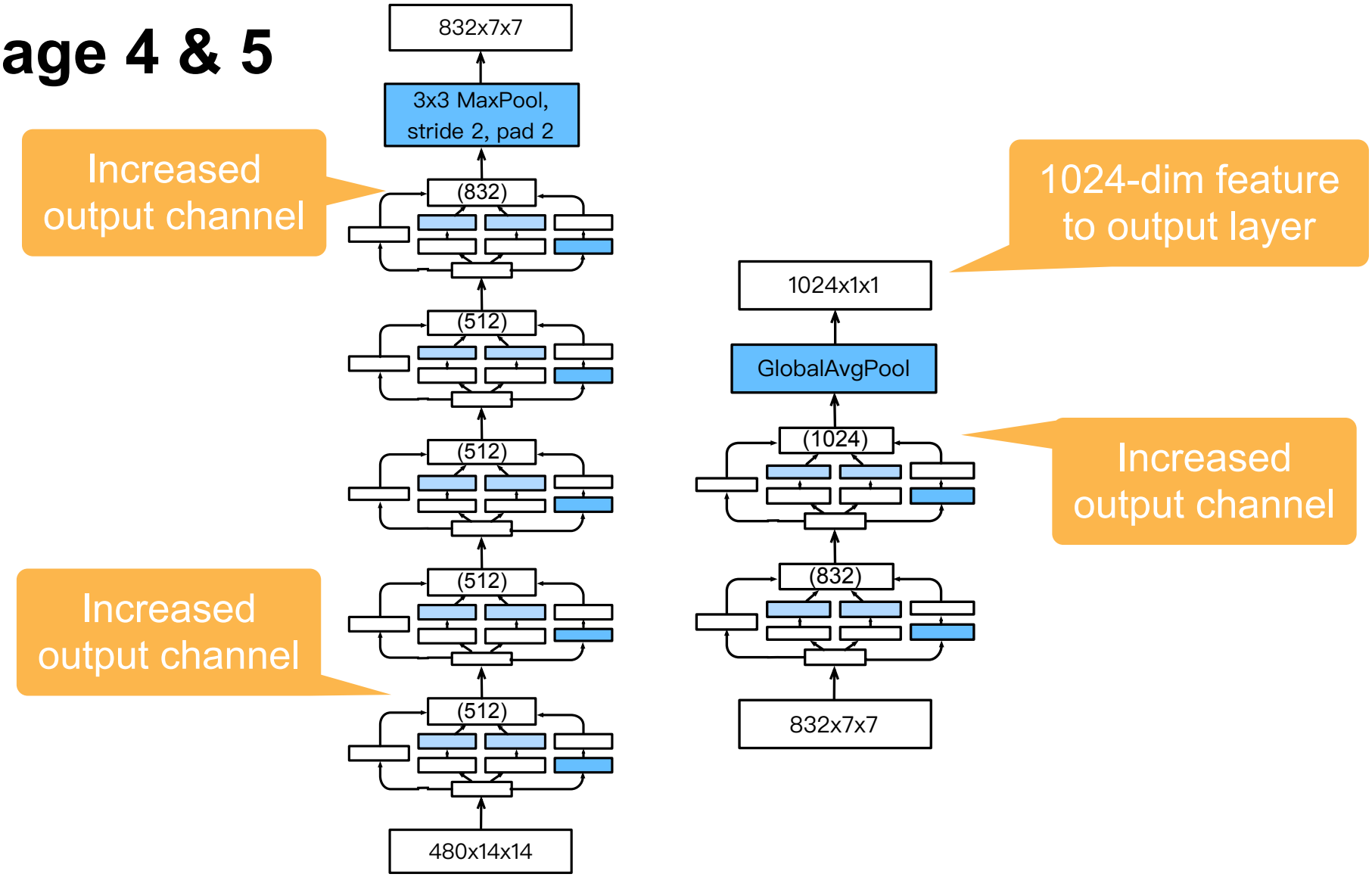
- Smaller kernel size and output channels due to more layers



7


480x14x14
3x3 MaxPool, stride 2, pad 2
Increased output channel
Channels allocation is different
Concatenation (480)
3x3 Conv (192)
5x5 Conv (96)
1x1 Conv (64)
1x1 Conv (128)
1x1 Conv (128)
1x1 Conv (32)
3x3 MaxPool
Input (256)
Concatenation (256)
3x3 Conv (128)
5x5 Conv (32)
1x1 Conv (32)
1x1 Conv (64)
1x1 Conv (96)
1x1 Conv (16)
3x3 MaxPool
192x28x28

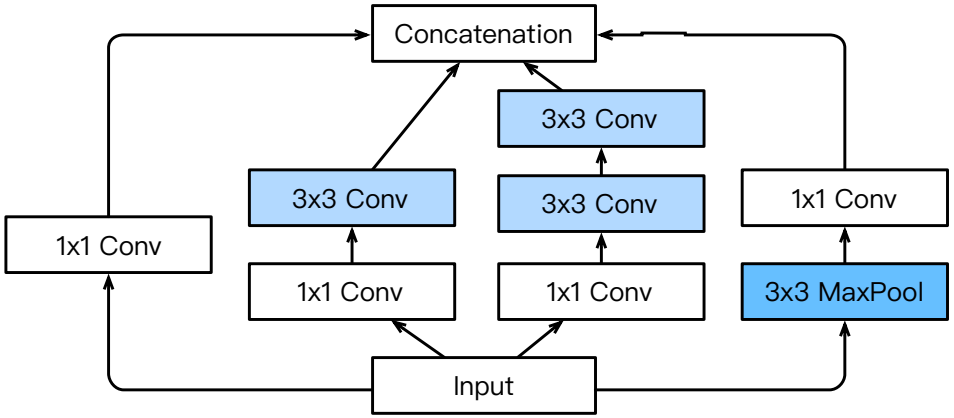
# Stage 4 & 5

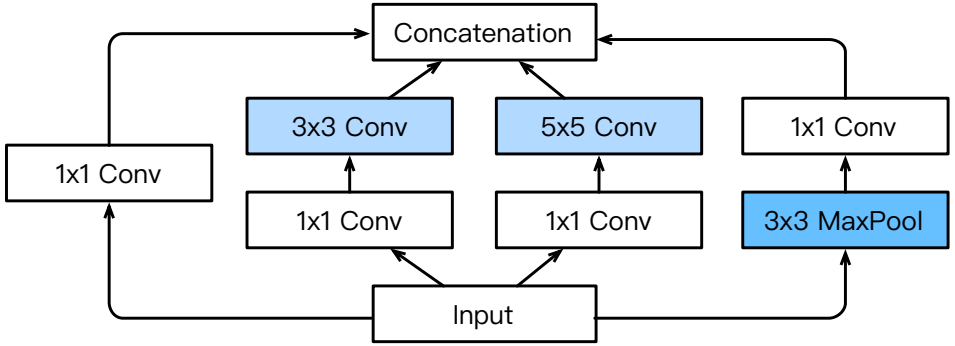

832x7x7
3x3 MaxPool, stride 2, pad 2
Increased output channel
(832)
(512)
(512)
Increased output channel
(512)
480x14x14
1024-dim feature to output layer
1024x1x1
GlobalAvgPool
(1024)
Increased output channel
(832)
832x7x7

## The many flavors of Inception Networks

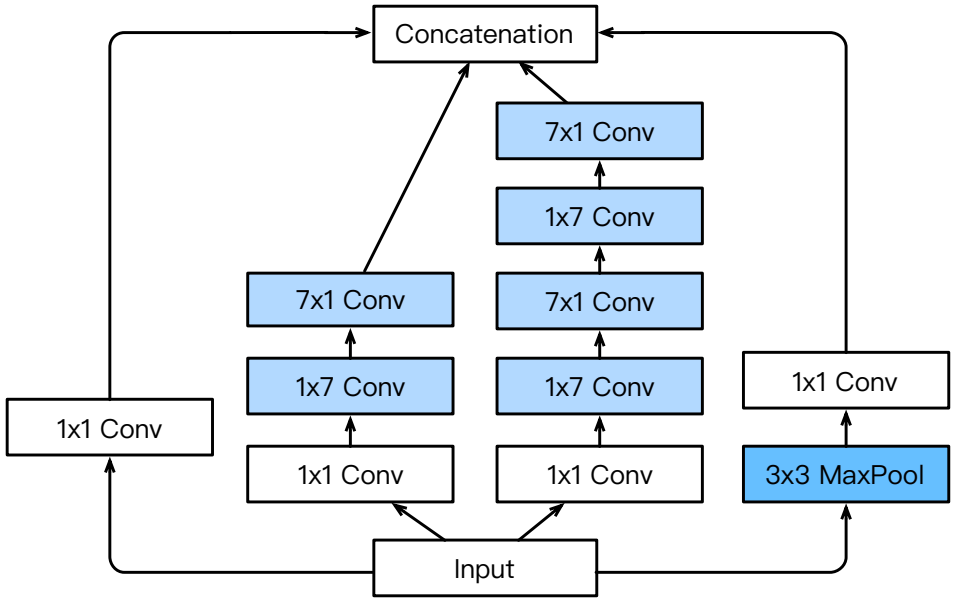
- Inception-BN (v2) - Add batch normalization
- Inception-V3 - Modified the inception block
  - Replace 5x5 by multiple 3x3 convolutions
  - Replace 5x5 by 1x7 and 7x1 convolutions
  - Replace 3x3 by 1x3 and 3x1 convolutions
  - Generally deeper stack
- Inception-V4 - Add residual connections (more later)

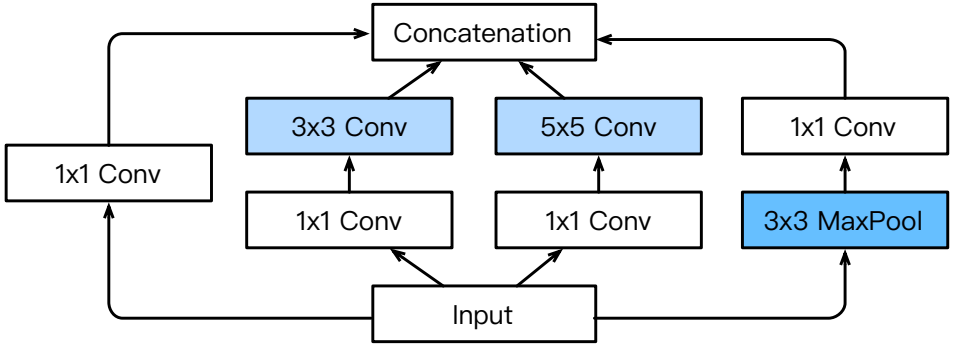
# Inception V3 Block for Stage 3



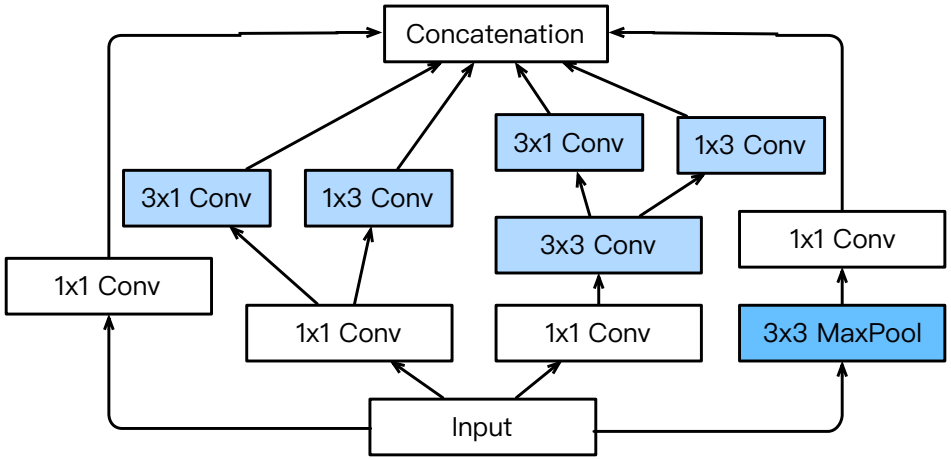



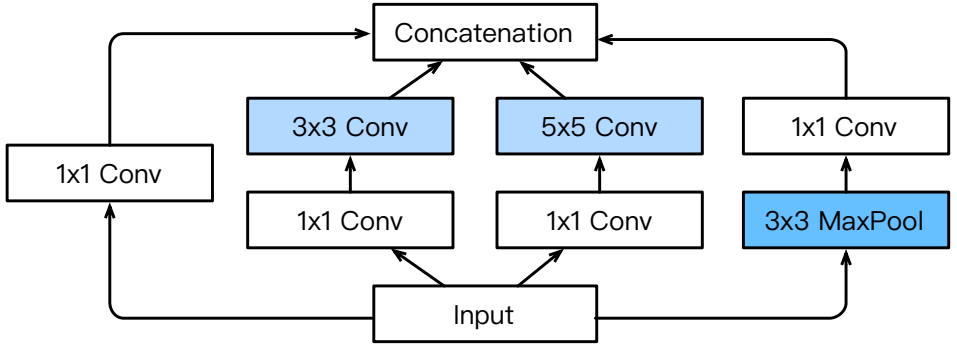
# Inception V3 Block for Stage 4

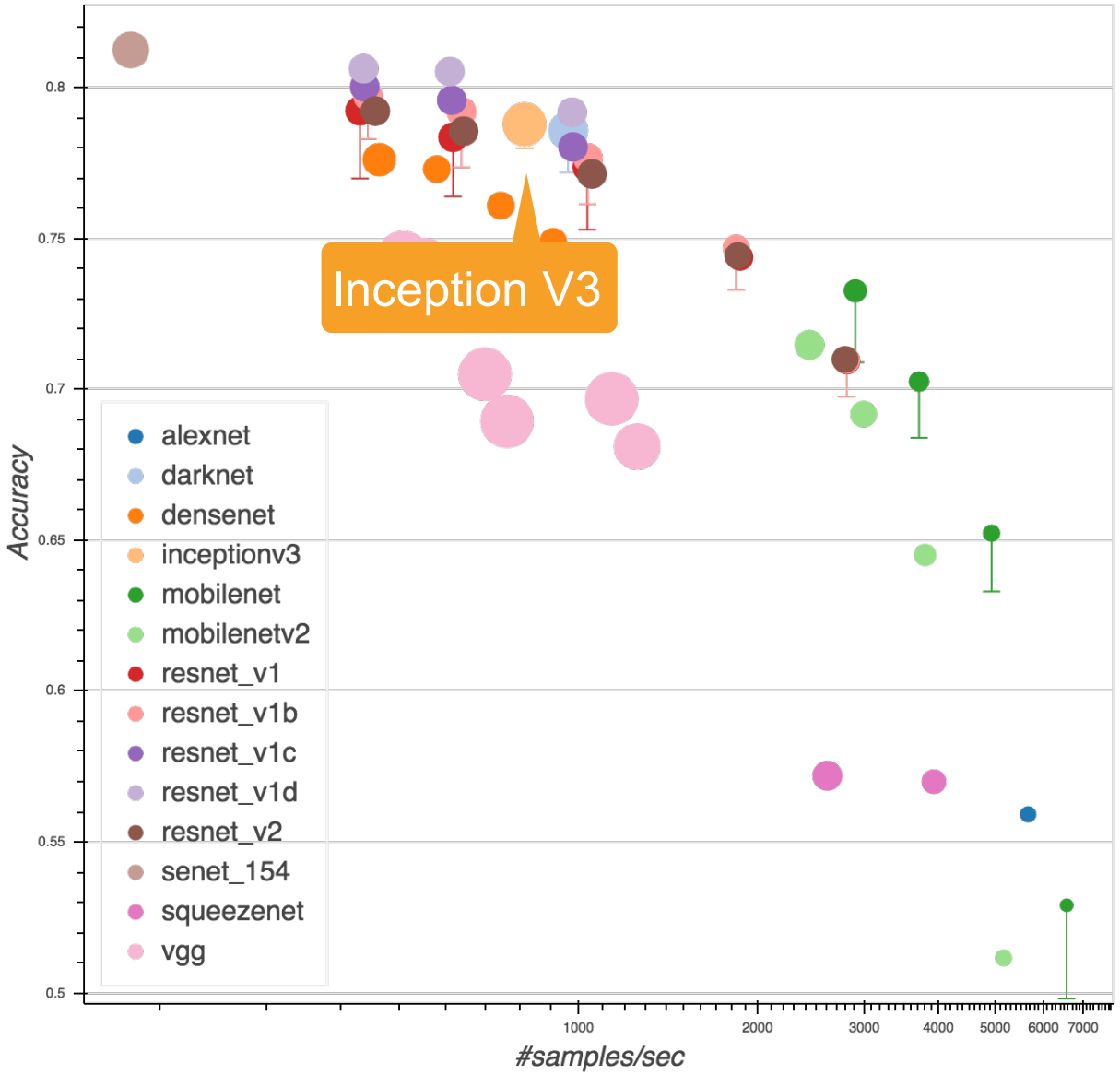


# Inception V3 Block for Stage 5



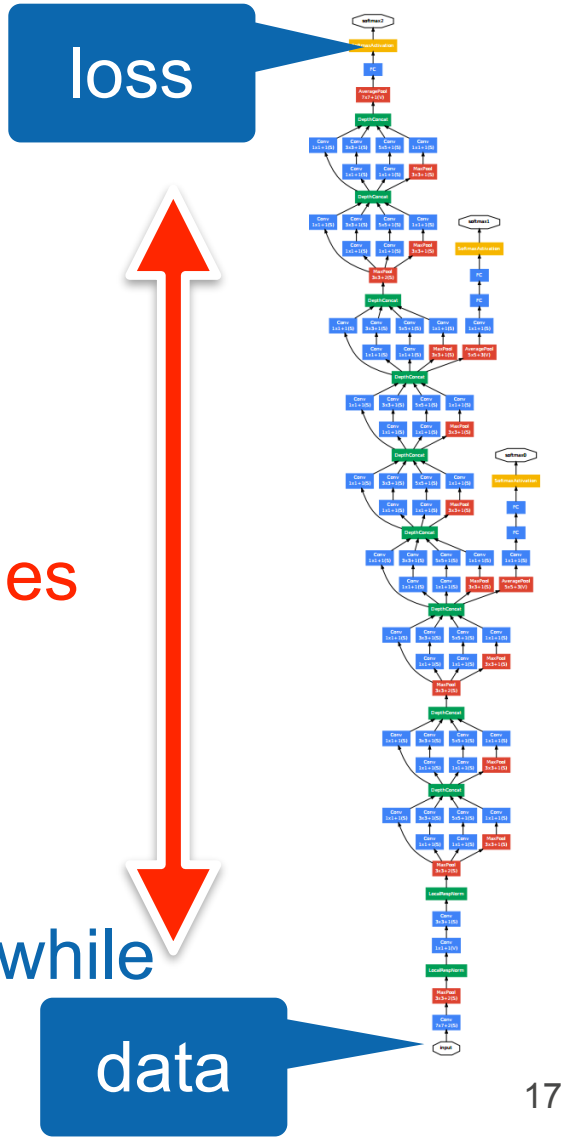





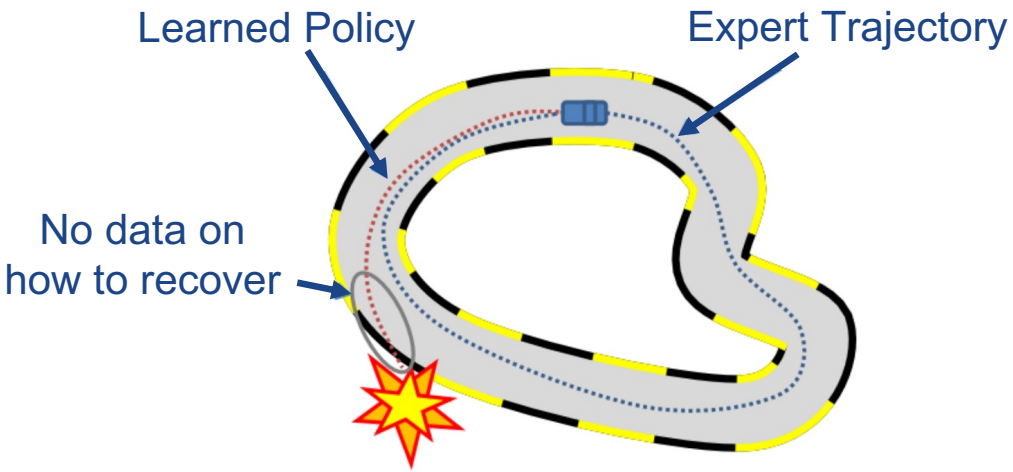

Use FP 16 to speed up
Less accuracy

# Batch Normalization

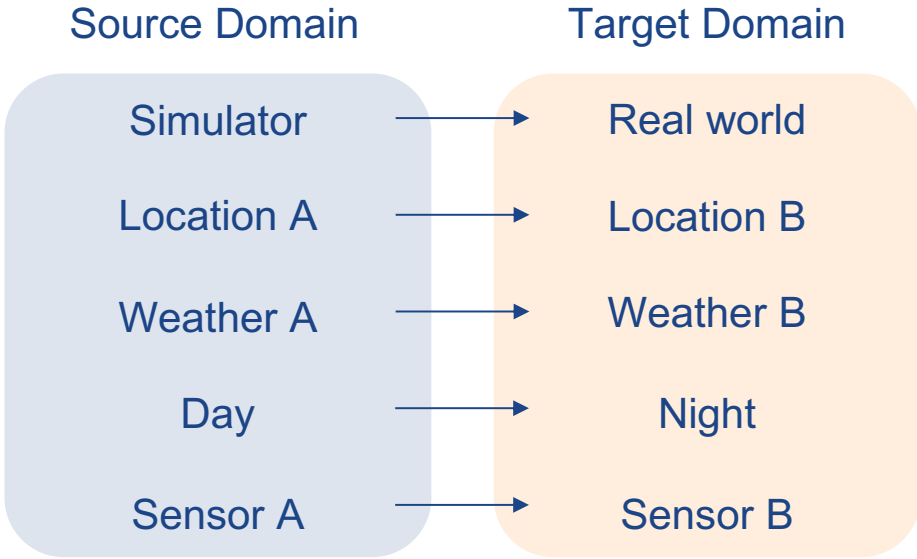
- Loss occurs at last layer
  - Last layers learn quickly
- Data is inserted at bottom layer
  - Bottom layers change - **everything** changes
  - Last layers need to relearn many times
  - Slow convergence
- This is like covariate shift
  Can we avoid changing the last layers (top) while learning first layers?



# Covariate Shift




**(b) Covariate Shift**



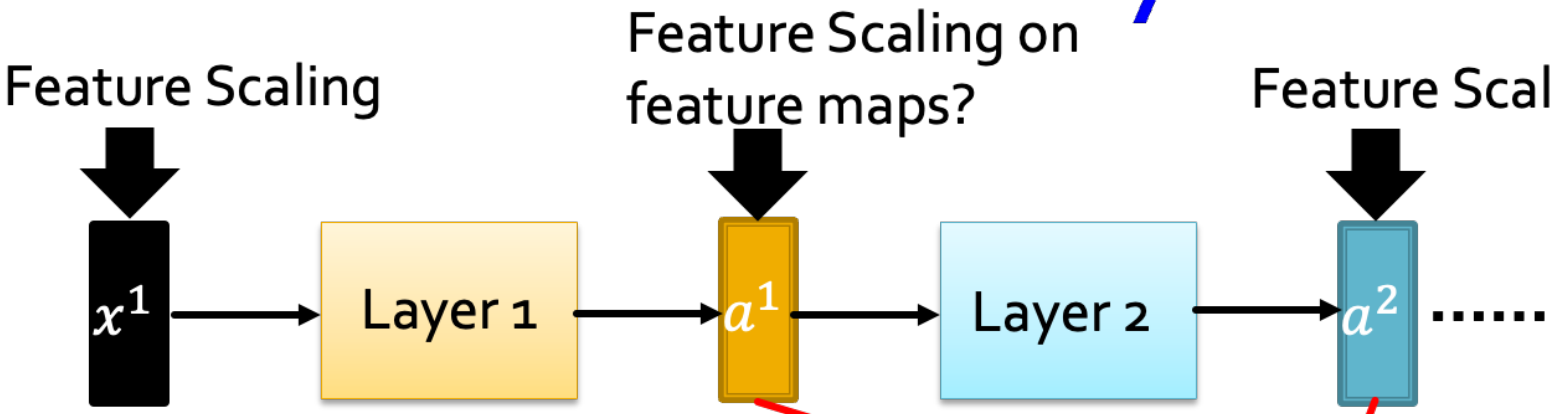

**(c) Domain Adaptation**

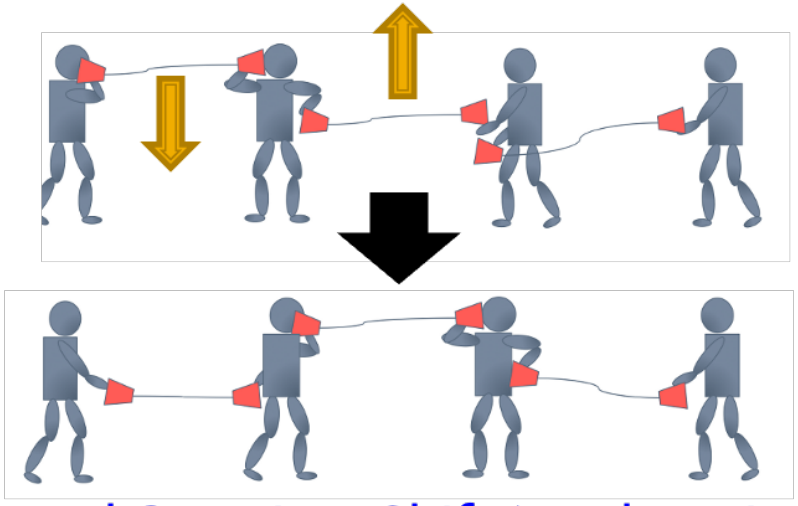
# Covariate Shift


Training data for distinguishing cats and dogs.


Test data for distinguishing cats and dogs.

# How about Hidden Layer?



Difficulty: their statistics change during the training
Up stream affect downstream ...

Batch normalization

Smaller learning rate can be helpful, but the training would be slower.

Internal Covariate Shift (use learning rate to control)

# Batch Normalization

- Can we avoid changing last layers while learning first layers?
- Fix mean and variance

$$\mu_B = \frac{1}{|B|}\sum_{i \in B} x_i \text{ and } \sigma_B^2 = \frac{1}{|B|}\sum_{i \in B}(x_i - \mu_B)^2 + \epsilon$$

and adjust it separately

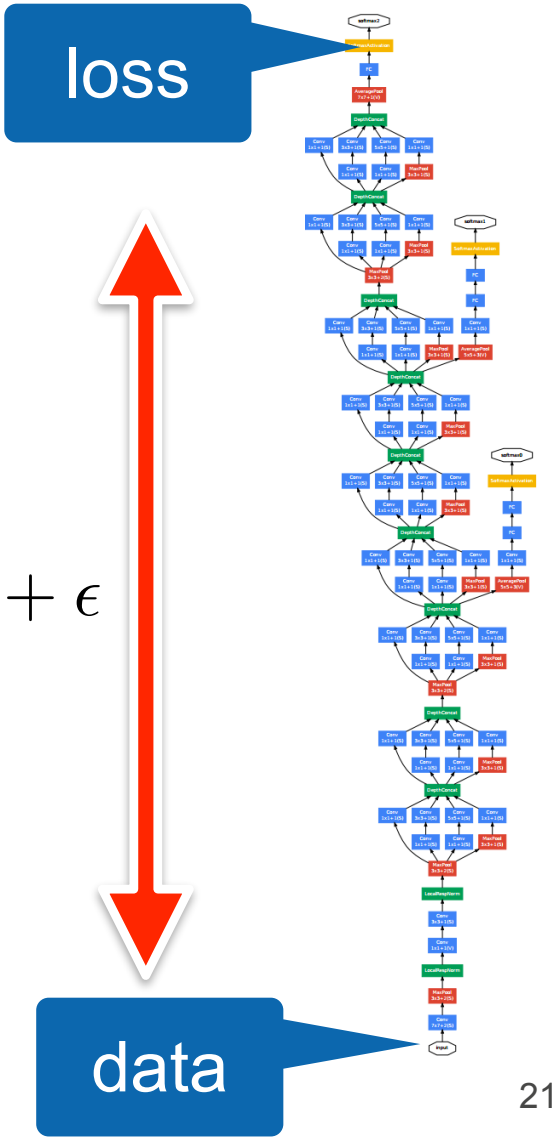
```
self.gamma = nn.Parameter(torch.ones(shape))
self.beta = nn.Parameter(torch.zeros(shape))
```

$$x_{i+1} = \gamma \frac{x_i - \mu_B}{\sigma_B} + \beta$$

mean

variance

Here $\gamma$ and $\beta$ are rescaling parameters.



# This was the original motivation …

# What Batch Norms really do

- Doesn't really reduce covariate shift (Lipton et al., 2018)
- Regularization by noise injection

$$x_{i+1} = \gamma \frac{x_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

Random offset

Random scale

This is sweet spot of sigmoid

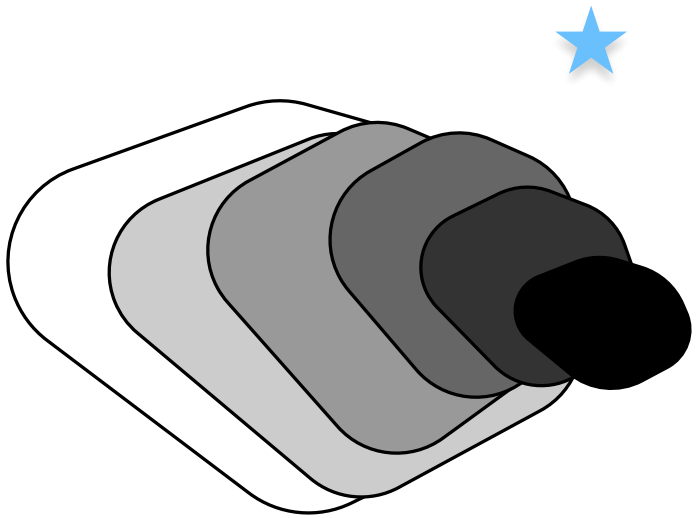
  - Random shift per minibatch
  - Random scale per minibatch
- No need to mix with dropout (both are capacity control)
- Ideal minibatch size of 64 to 256

# Details

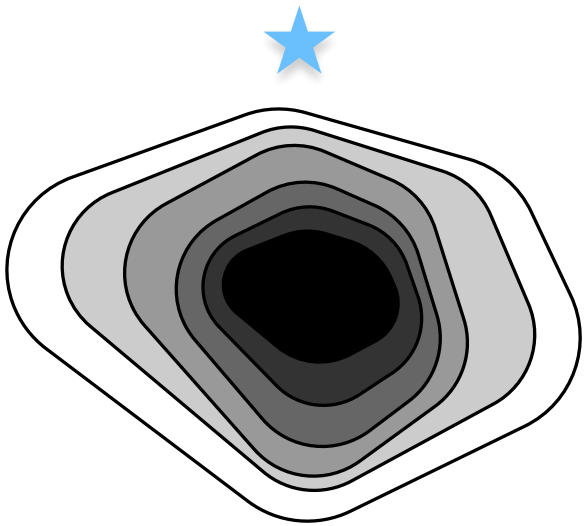
`torch.nn.BatchNorm1/2d`

- **Dense Layer**
  One normalization for all
- **Convolution**
  One normalization per channel
- Compute **new mean and variance** for every minibatch
  - Effectively acts as regularization
  - Optimal minibatch size is ~128
    (watch out for parallel training with many machines)
    - Multiple trainers might need to exchange empiral mean and variance.

# Residual Networks



generic function classes

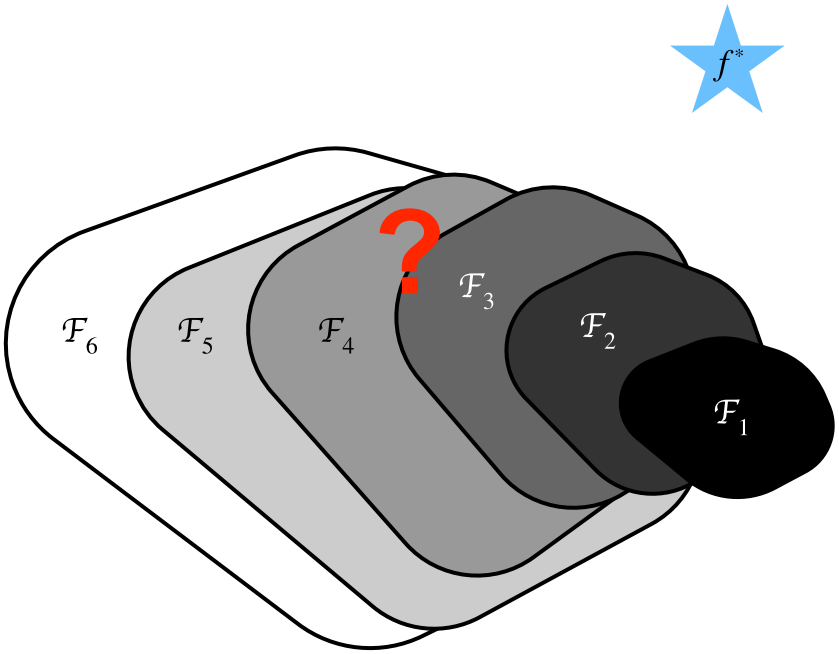


nested function classes

Add more layers mean a different class of functions, Deep Learning

You add more constrains/stronger assumptions, color becomes dark in statistical learning.

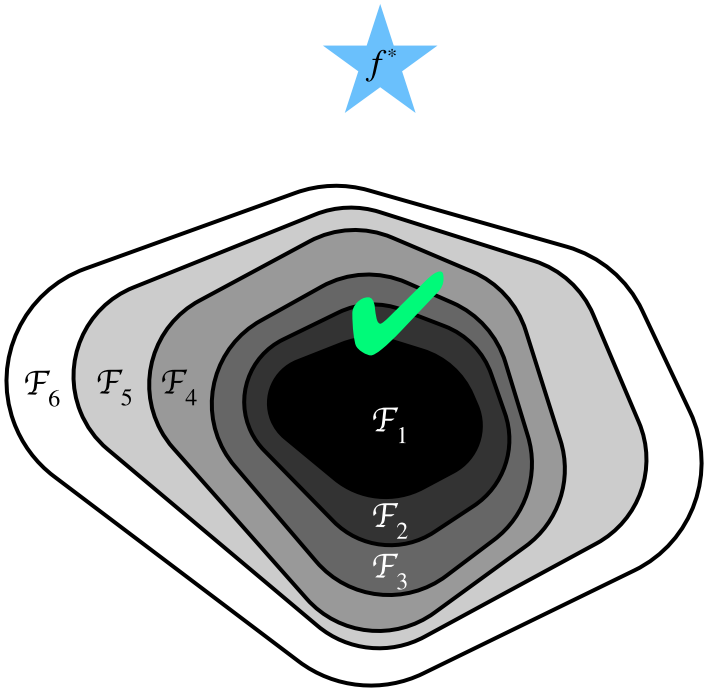
# Does adding layers improve accuracy?




Non-nested function classes

$When \mathcal{F}_1 \nsubseteq \mathcal{F}_2 \nsubseteq \cdots \nsubseteq \mathcal{F}_6,$

a larger function class does not always move closer to the "truth" function $f^*$.



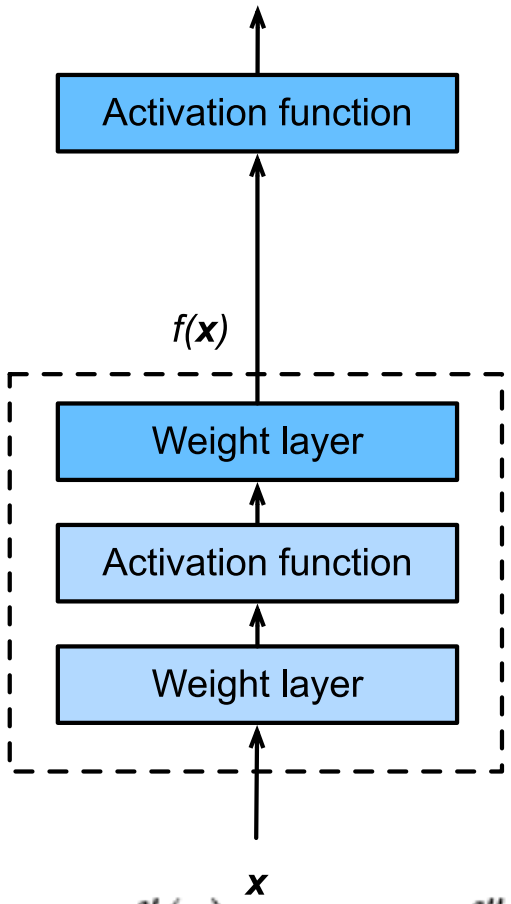

Nested function classes

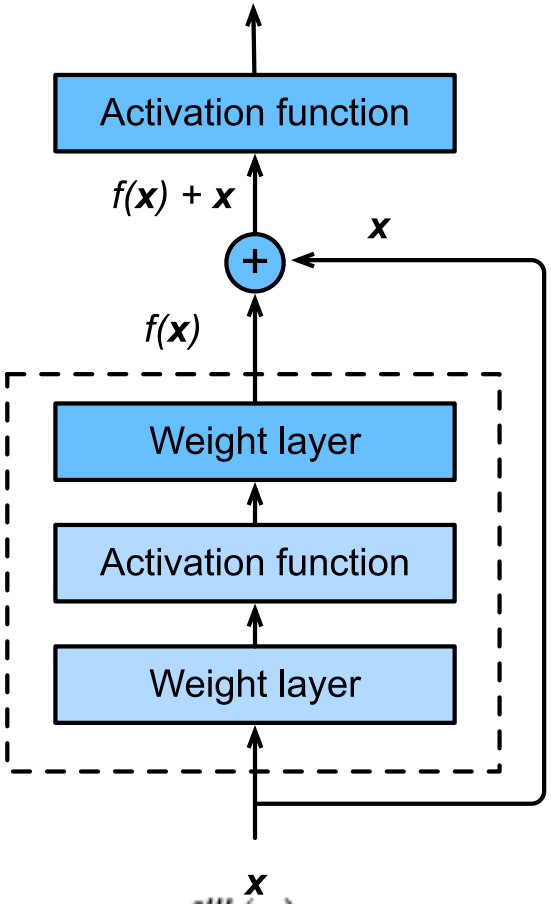
$When \mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \cdots \subseteq \mathcal{F}_6,$

a larger function class always move closer to the "truth" function $f^*$.

# Residual Networks

- Adding a layer **changes** function class
- We want to **add to** the function class
- 'Taylor expansion' style parametrization
- (Add additional layers , learn high-order functions.



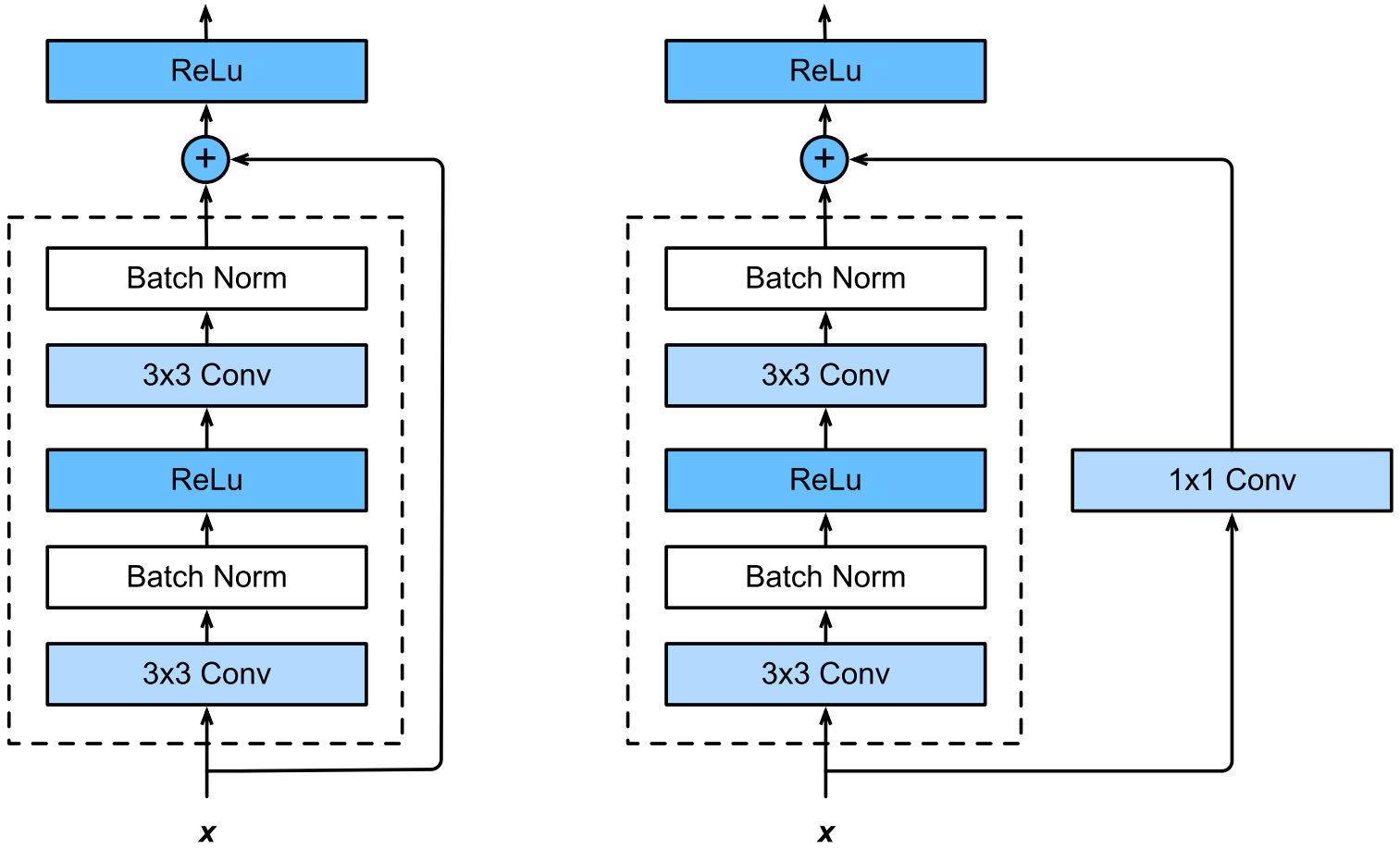




$$f(x) = x + g(x)$$

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots,$$

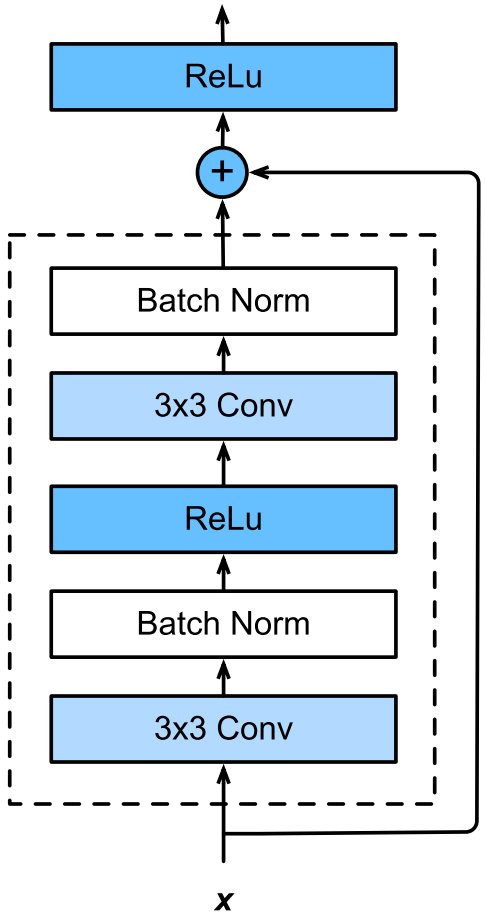
He et al., 2015

# ResNet Block in detail



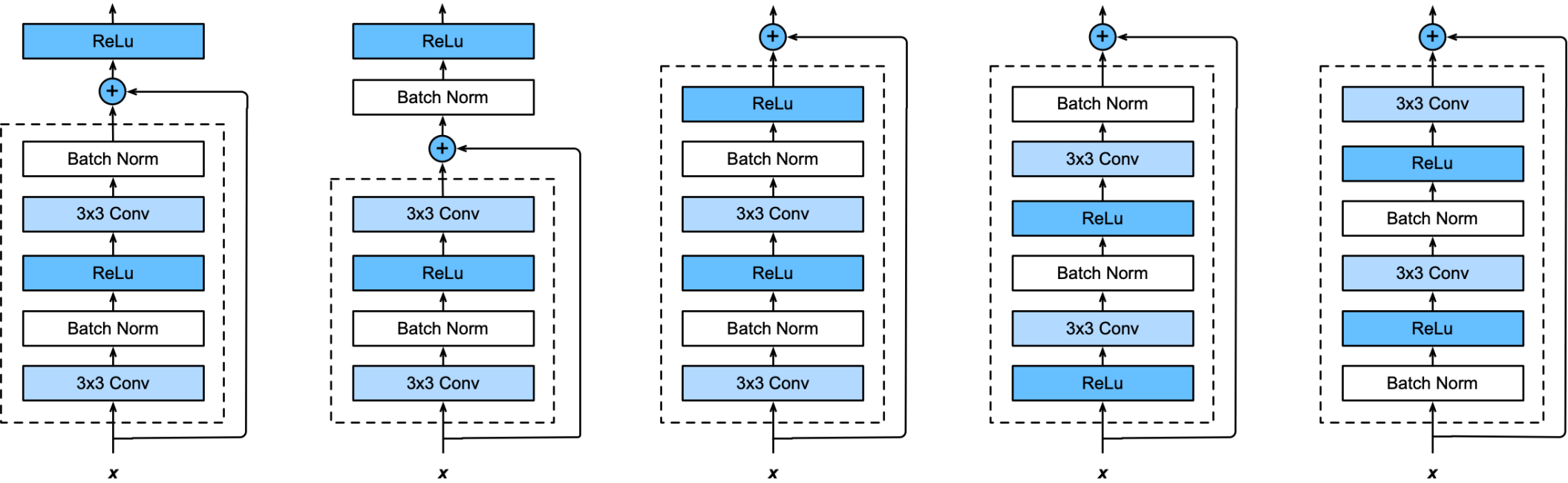

This one looks better, learn the identify functions

# In code



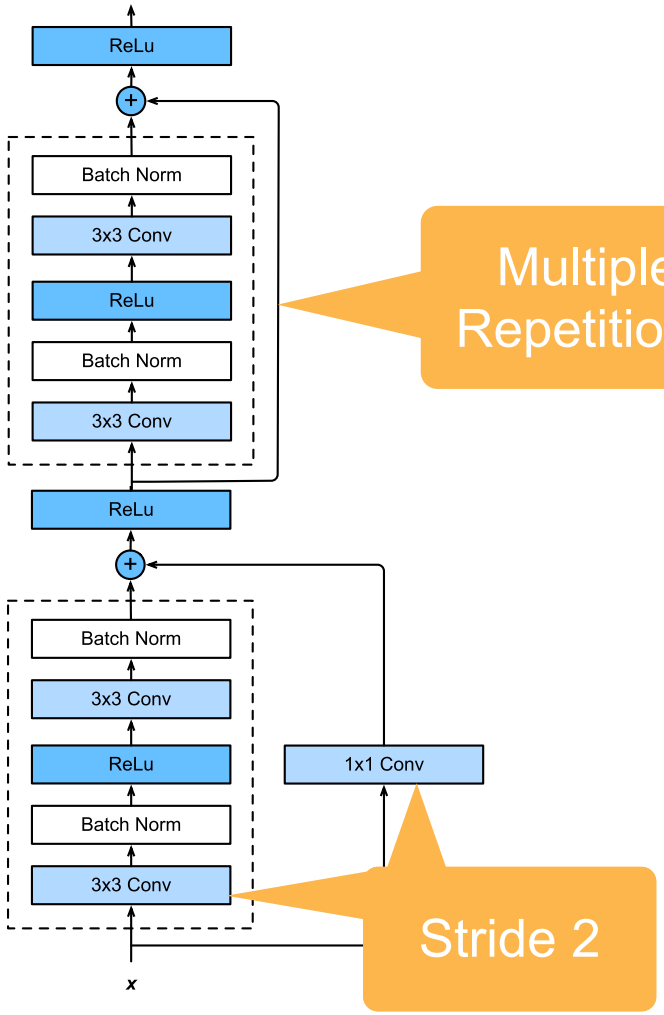

```
def forward(self, X):
    Y = self.bn1(self.conv1(X))
    Y = nd.relu(Y)
    Y = self.bn2(self.conv2(Y))
    if self.conv3:
        X = self.conv3(X)
    return nd.relu(Y + X)
```

# The many flavors of ResNet blocks




Try every permutation

# ResNet Module



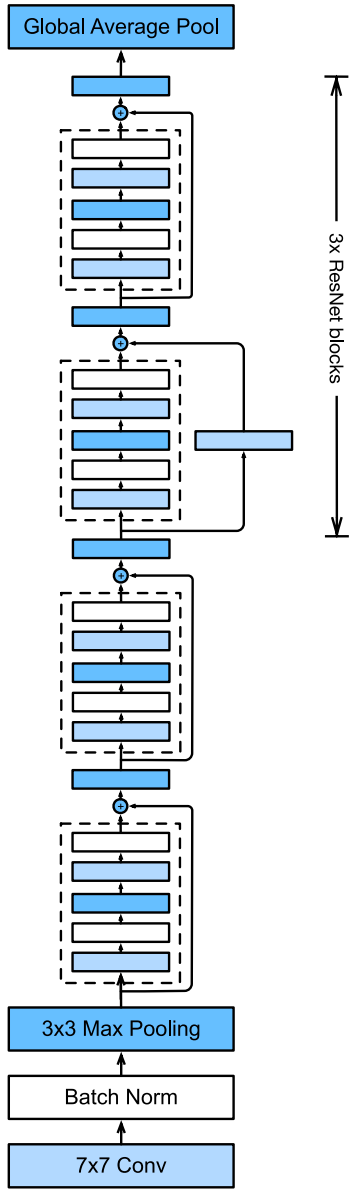

- Downsample per module (stride=2)
- Enforce some nontrivial nonlinearity per module (via 1x1 convolution)
- Stack up in blocks

```
blk = nn.Sequential()
for i in range(num_residuals):
   if i == 0 and not first_block:
      blk.add(Residual(num_channels,
         use_1x1conv=True, strides=2))
   else:
      blk.add(Residual(num_channels))
```

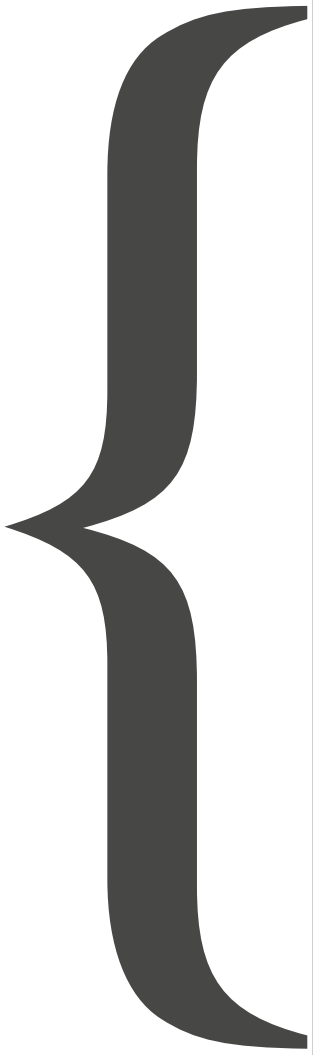
# Putting it all together

- Same block structure as e.g. VGG or GoogleNet
- Residual connection to add to expressiveness
- Pooling/stride for dimensionality reduction
- Batch Normalization for capacity control

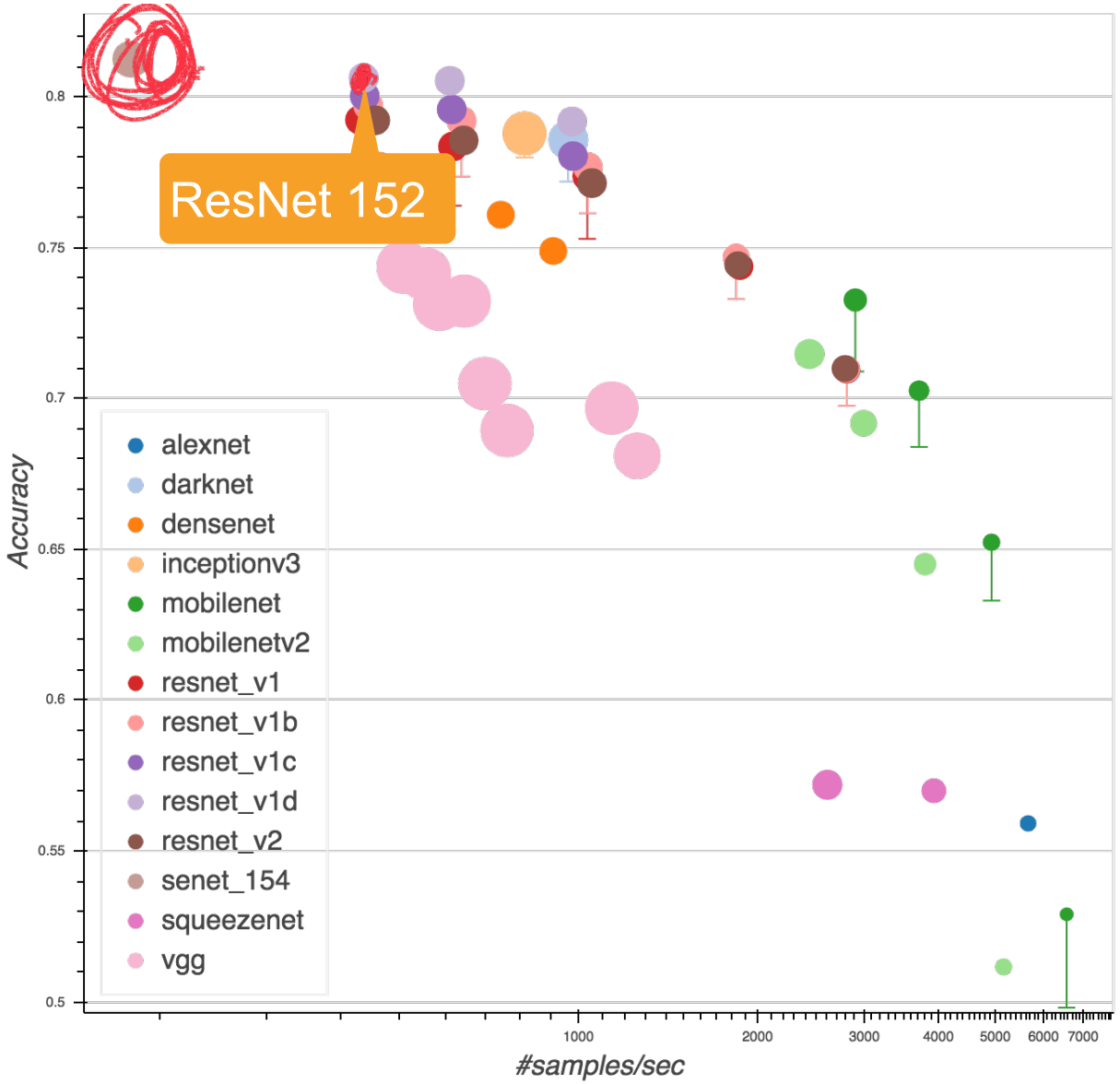
… train it at scale …



18 layers, Reset-18

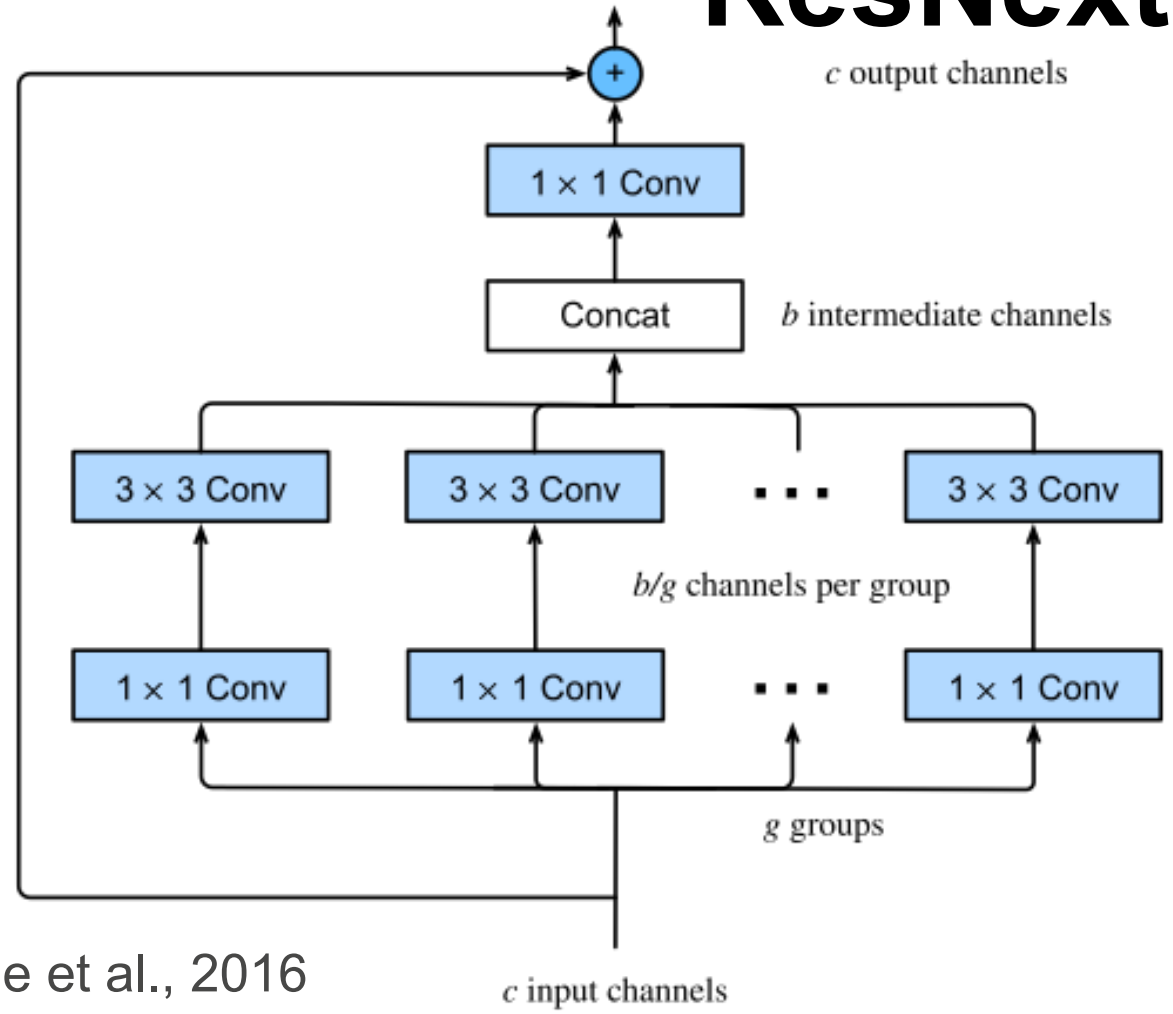


```yaml
Conv1: 7x7, 64, stride 2, padding 3
Max Pool: 3x3, stride 2, padding 1

ResBlock1:
  Block1a: Conv3x3, 64 -> Conv3x3, 64
  Block1b: Conv3x3, 64 -> Conv3x3, 64

ResBlock2:
  Block2a: Conv3x3, 128, stride 2 -> Conv3x3, 128
  Block2b: Conv3x3, 128 -> Conv3x3, 128

ResBlock3:
  Block3a: Conv3x3, 256, stride 2 -> Conv3x3, 256
  Block3b: Conv3x3, 256 -> Conv3x3, 256

ResBlock4:
  Block4a: Conv3x3, 512, stride 2 -> Conv3x3, 512
  Block4b: Conv3x3, 512 -> Conv3x3, 512

Global Average Pool
FC: 1000
```

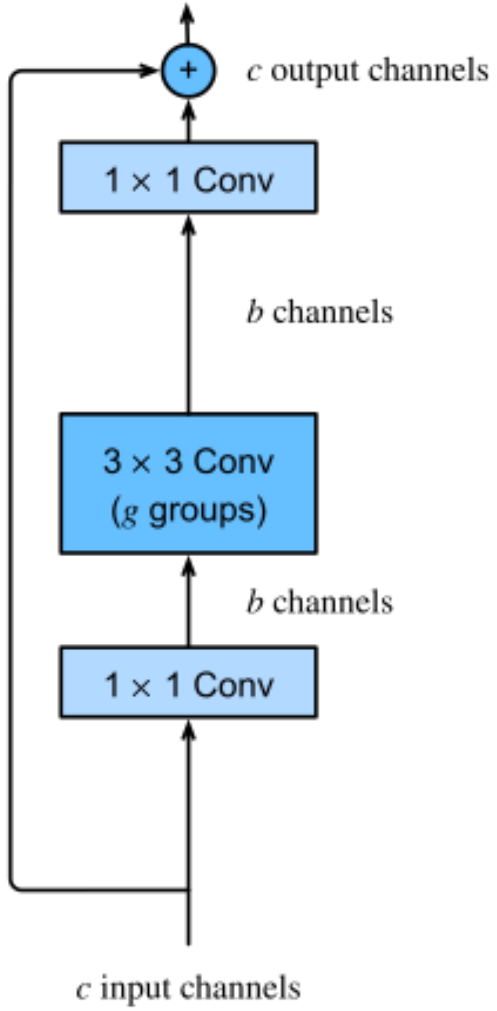



**Resnet is smaller, but high-performance.**

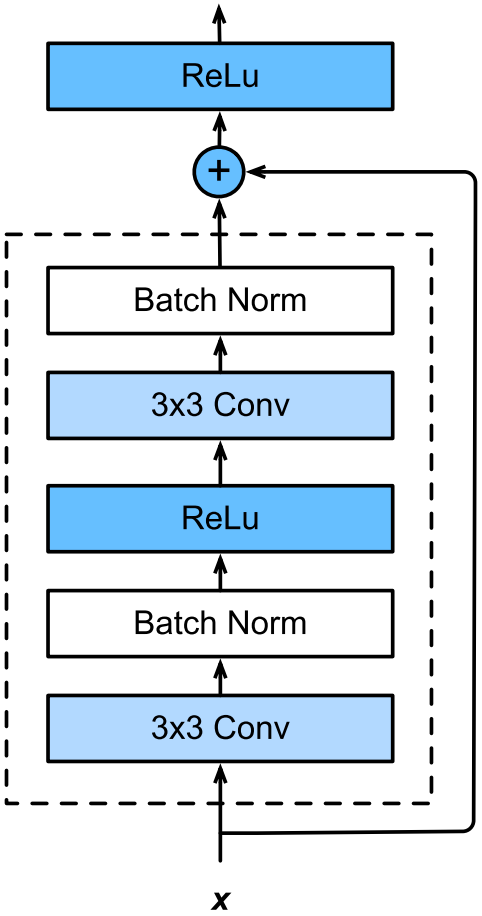
# ResNext




Xie et al., 2016



# Reducing the cost of Convolutions




- **Parameters**

$$k_h \cdot k_w \cdot c_i \cdot c_o$$

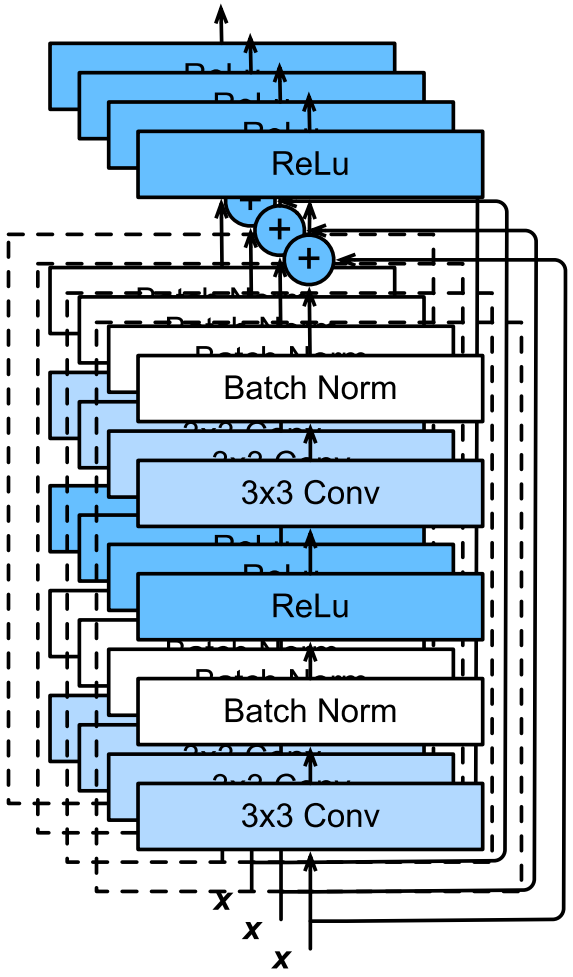
- **Computation**

$$m_h \cdot m_w \cdot k_h \cdot k_w \cdot c_i \cdot c_o$$

- **Slicing convolutions** (Inception v4) e.g. 3x3 vs. **1x5** and **5x1**
- **Break up channels** (mix only within)

$$m_h \cdot m_w \cdot k_h \cdot k_w \cdot \frac{c_i}{b} \cdot \frac{c_o}{b} \cdot b$$

# Reducing the cost of Convolutions




- **Parameters**

$$k_h \cdot k_w \cdot c_i \cdot c_o$$

- **Computation**

$$m_h \cdot m_w \cdot k_h \cdot k_w \cdot c_i \cdot c_o$$

- **Slicing convolutions** (Inception v4) e.g. 3x3 vs. **1x5** and **5x1**
- **Break up channels** (mix only within)

$$m_h \cdot m_w \cdot k_h \cdot k_w \cdot \frac{c_i}{b} \cdot \frac{c_o}{b} \cdot b$$

| stage | output | ResNet-50 | **ResNeXt-50 (32×4d)** |
|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | 7×7, 64, stride 2 |
| conv2 | 56×56 | 3×3 max pool, stride 2 | 3×3 max pool, stride 2 |
| | | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times3$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C=32 \\ 1\times1, 256 \end{bmatrix} \times3$ |
| conv3 | 28×28 | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times4$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C=32 \\ 1\times1, 512 \end{bmatrix} \times4$ |
| conv4 | 14×14 | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times6$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C=32 \\ 1\times1, 1024 \end{bmatrix} \times6$ |
| conv5 | 7×7 | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times3$ | $\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C=32 \\ 1\times1, 2048 \end{bmatrix} \times3$ |
| | 1×1 | global average pool<br>1000-d fc, softmax | global average pool<br>1000-d fc, softmax |
| # params. | | **25.5**×$10^6$ | **25.0**×$10^6$ |
| FLOPs | | **4.1**×$10^9$ | **4.2**×$10^9$ |

# RexNext budget

- Slice blocks into 32 sub-blocks
- Can use more dimensions
- Higher accuracy

```
nn.Conv2D(group_width=width,
          kernel_size=3,
          groups=cardinality)
```

Xie et al., 2016

# More Ideas

# DenseNet (Huang et al., 2016)

- ResNet combines x and f(x)
- DenseNet uses higher order 'Taylor series' expansion

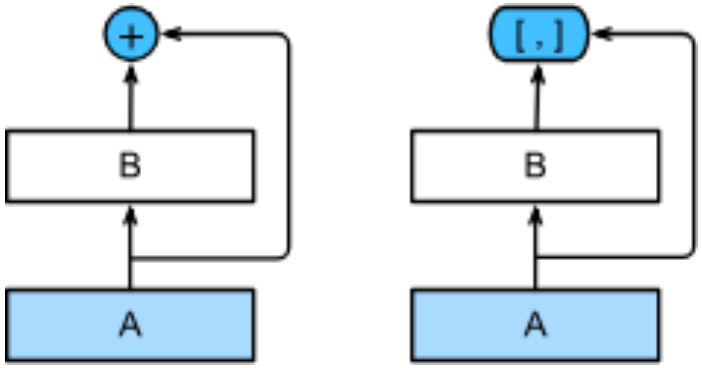
$$x_{i+1} = [x_i, f_i(x_i)]$$

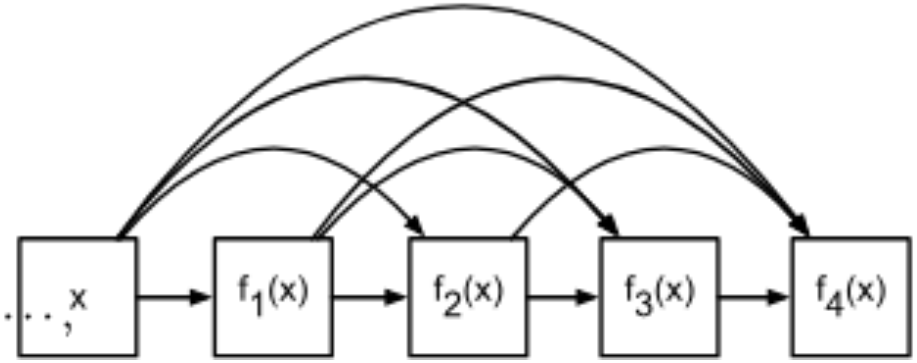
$$x_1 = x$$

$$x_2 = [x, f_1(x)]$$

$$x_3 = [x, f_1(x), f_2([x, f_1(x)])]$$

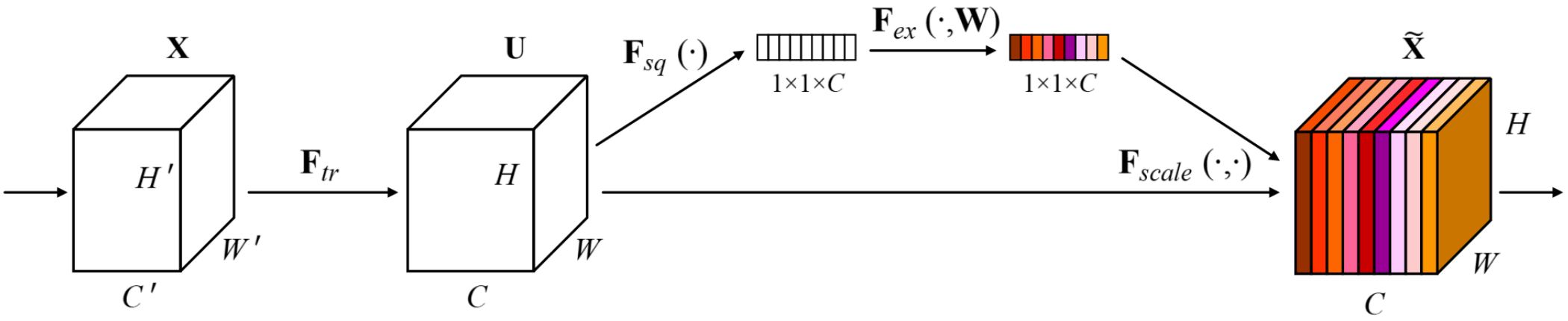
$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots$$



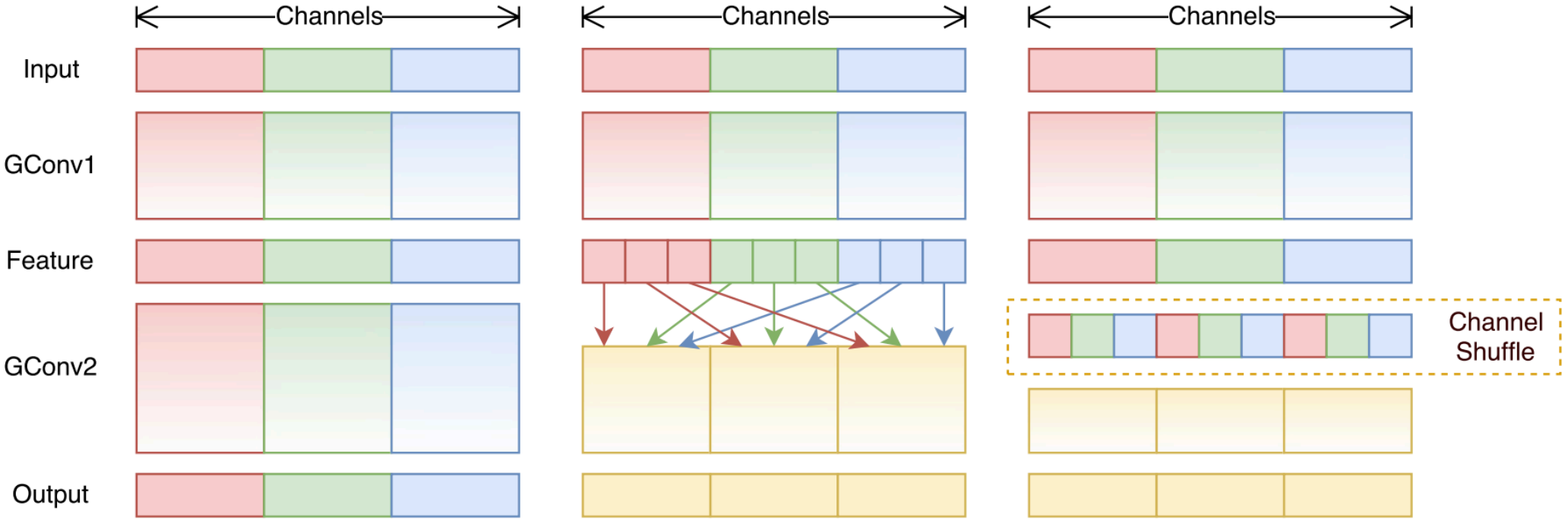

- Occasionally need to reduce resolution (transition layer)

# Squeeze-Excite Net (SENets Hu et al., 2017)




- Learn global weighting function per channel
- Learn channel that might correspond to one object (cat, dog, others)
- Allows for fast information transfer between pixels in different locations of the image (It takes multi-layers to propogate information from one location to another one).

# ShuffleNet (Zhang et al., 2018)

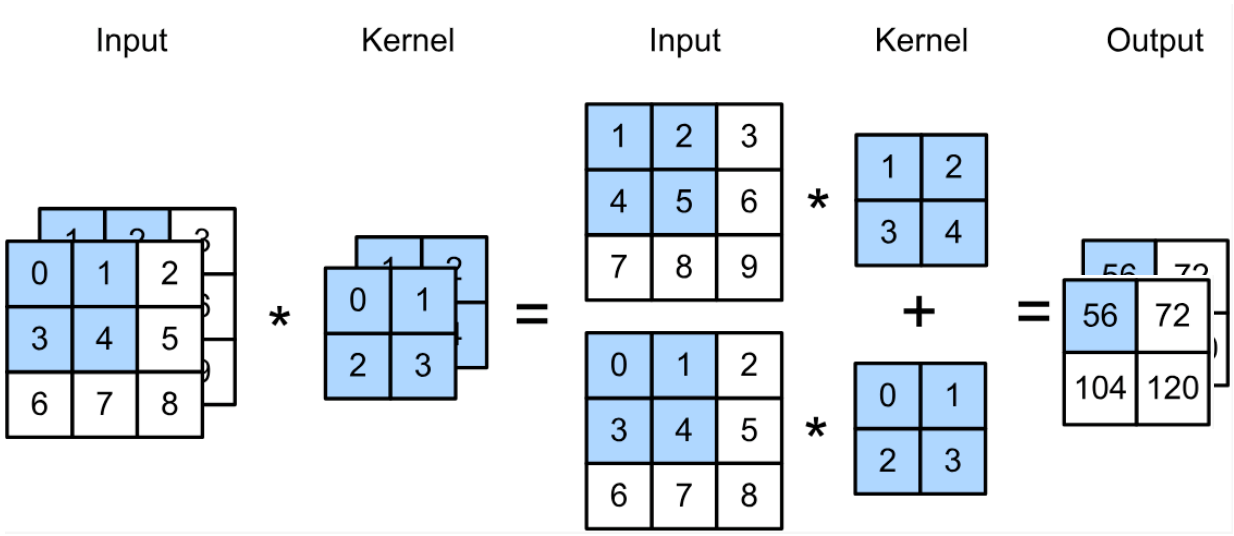



- ResNext breaks convolution into channels. (Stove pipe)
- ShuffleNet mixes by grouping (very efficient for mobile)

# Separable Convolutions - all channels separate

- **Parameters** $k_h \cdot k_w \cdot c_i \cdot c_o$
- **Computation** $m_h \cdot m_w \cdot k_h \cdot k_w \cdot c_i \cdot c_o$
- **Break up channels** to the extreme
  No mixing between channels

$$m_h \cdot m_w \cdot k_h \cdot k_w \cdot c$$



# Summary

- **Inception**
  - Inhomogeneous mix of convolutions (varying depth)
  - Batch norm regularization
- **ResNet**
  - Taylor expansion of functions
  - **ResNext** decomposes convolutions
- **Zoo**
  DenseNet, SENets Network, ShuffleNet, Separable Convolutions, …