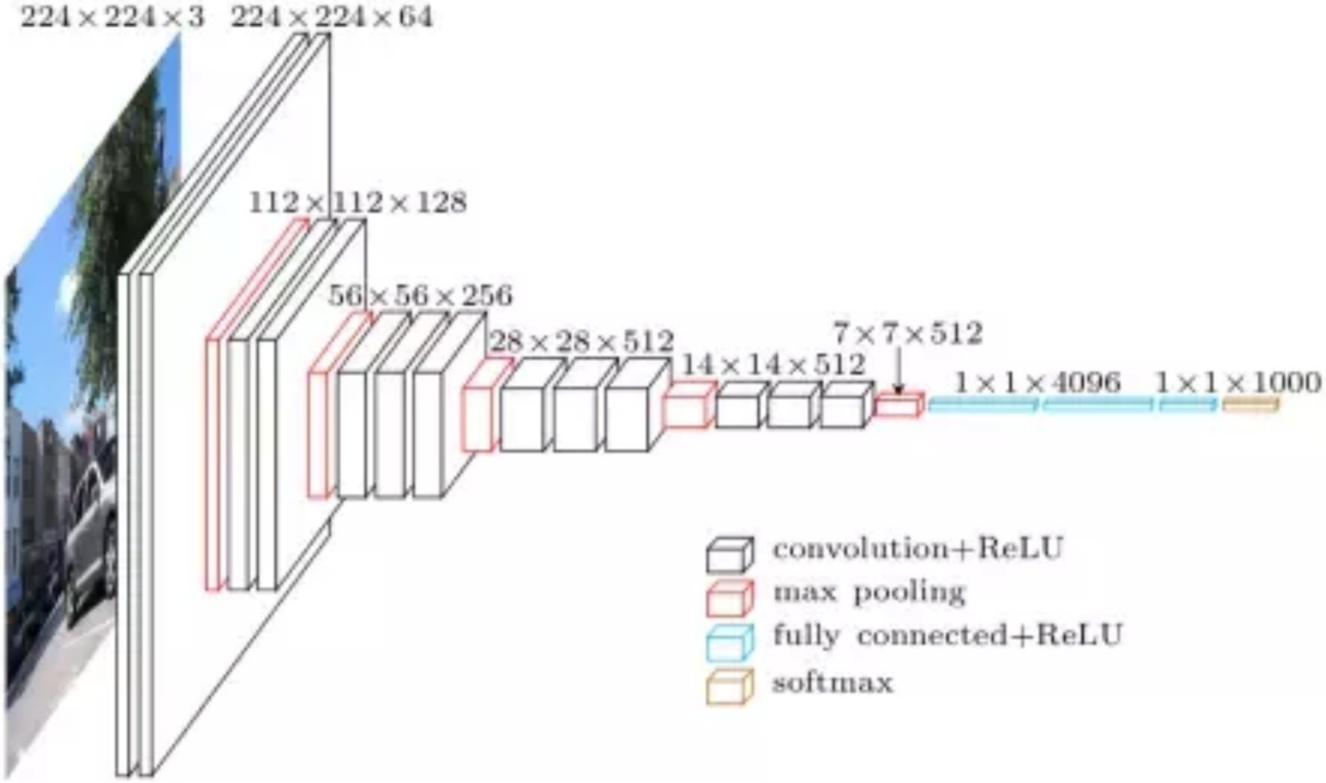
CNN_Models_Part_1

VGG

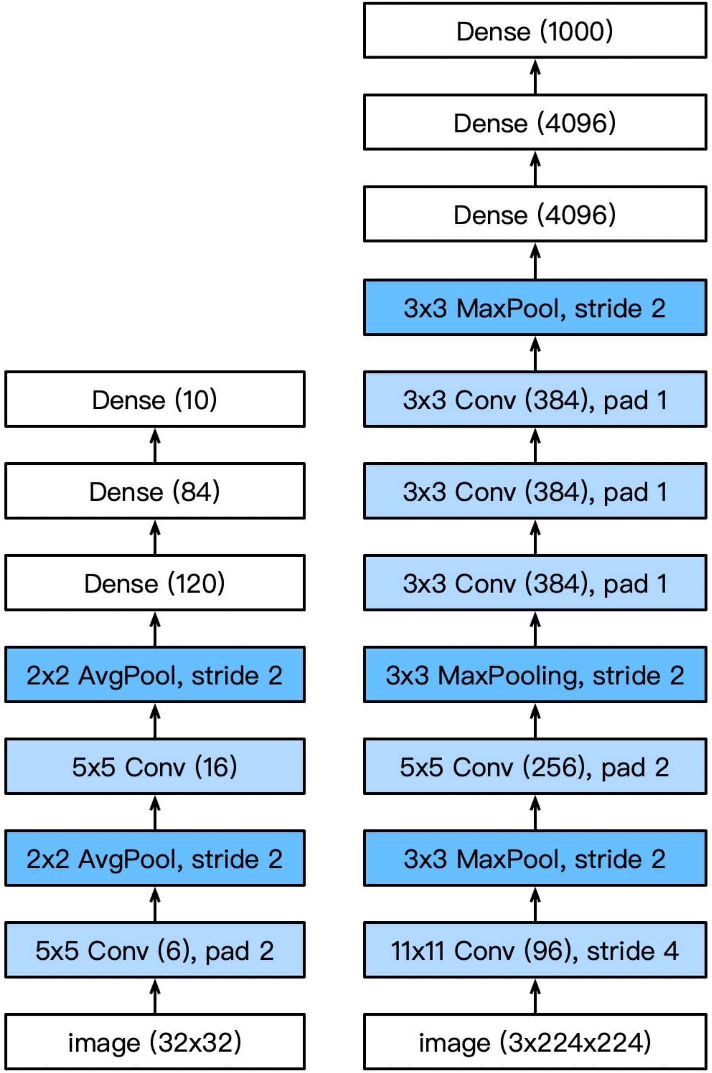


Slide credit: Alex Smola and Mu Li (Berkley)

VGG

- AlexNet is deeper and bigger than LeNet to get performance
- Go even bigger & deeper?
- Options
    - More dense layers (too expensive)
    - **More** convolutions
    - Group into **blocks**

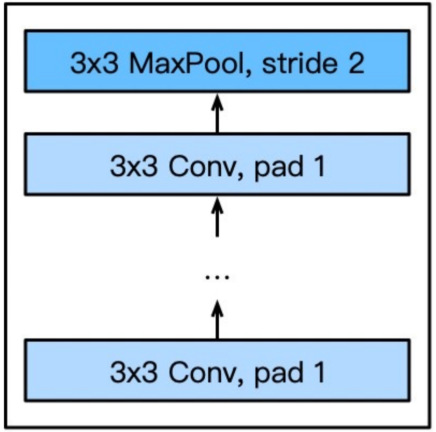
Slide credit: Alex Smola and Mu Li (Berkley)



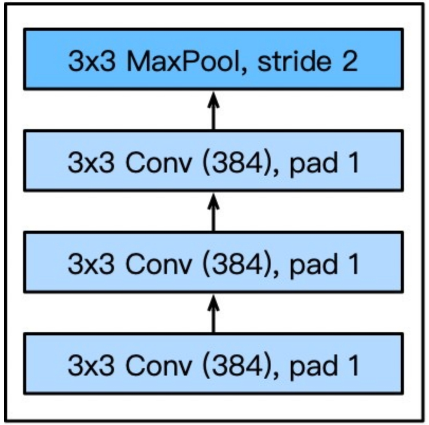
VGG Blocks

- Deeper vs. wider?
  - 5x5 convolutions
  - 3x3 convolutions (more)
  - **Deep & narrow better**
- VGG block
  - *3x3* convolutions (pad 1) **(n layers, m channels)**
  - 2x2 max-pooling (stride 2)

VGG block



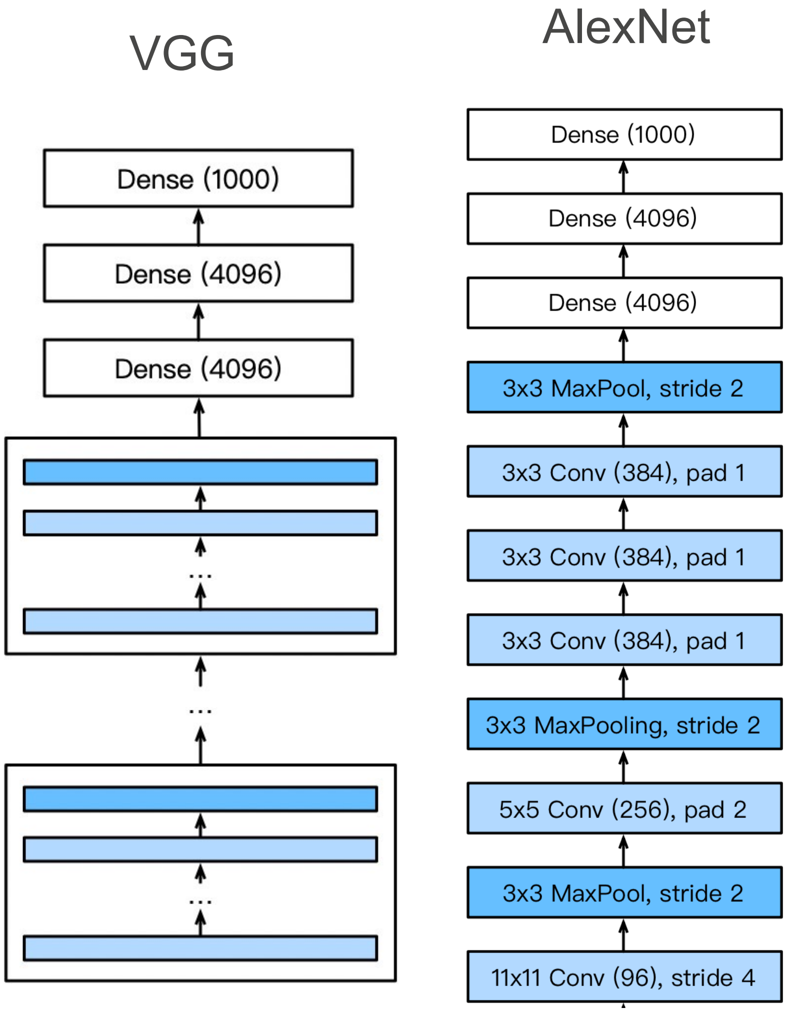
Part of AlexNet



Slide credit: Alex Smola and Mu Li (Berkley)

VGG Architecture

- Multiple VGG blocks followed by dense layers
- Vary the repeating number to get different architectures, such as VGG-16, VGG-19, …



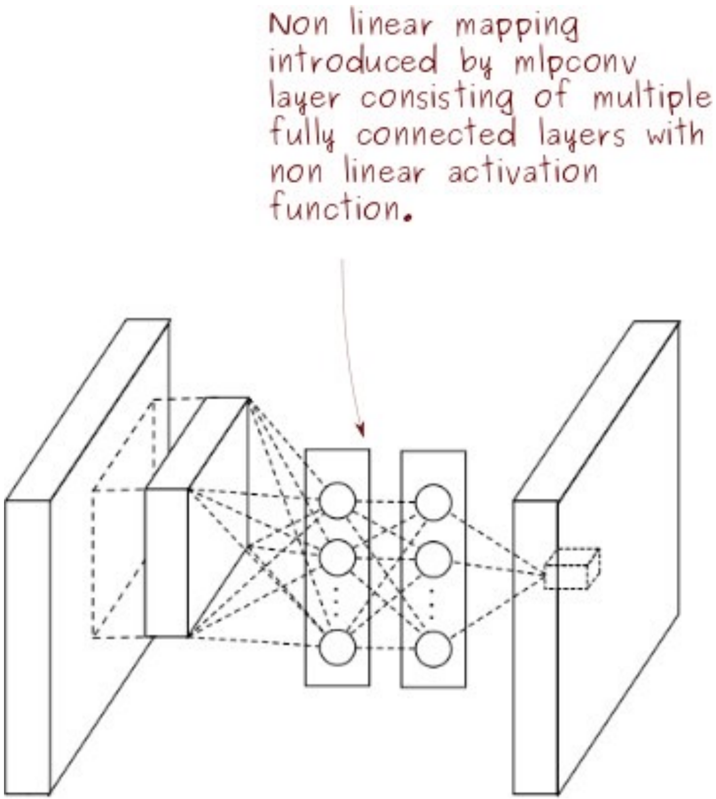
Slide credit: Alex Smola and Mu Li (Berkley)

Progress

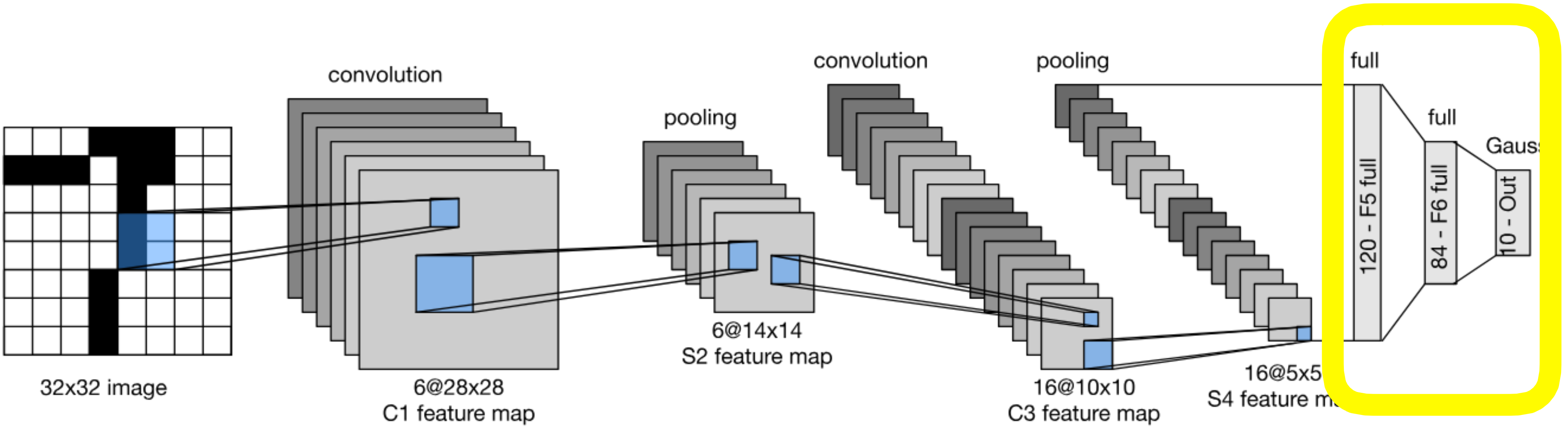
- LeNet (1995)
  - 2 convolution + pooling layers
  - 2 hidden dense layers
- AlexNet
  - Bigger and deeper LeNet
  - ReLu, Dropout, preprocessing
- VGG
  - Bigger and deeper AlexNet (repeated VGG blocks)

Slide credit: Alex Smola and Mu Li (Berkley)

Network in Network

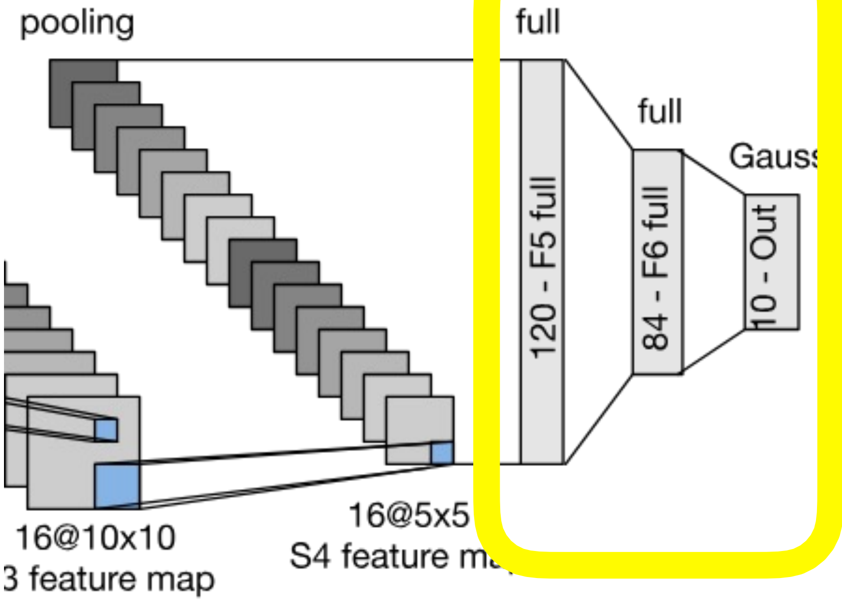


The Curse of the Last Layer(s)



Slide credit: Alex Smola and Mu Li (Berkley)

The Last Layer(s)



- Convolution layers need relatively few parameters

$$c_i \times c_o \times k^2$$

- Last layer needs many parameters for n classes

$$c \times m_w \times m_h \times n$$

- LeNet 16x5x5x120 = 48k
- AlexNet 256x5x5x4096 = 26M
- VGG 512x7x7x4096 = 102M

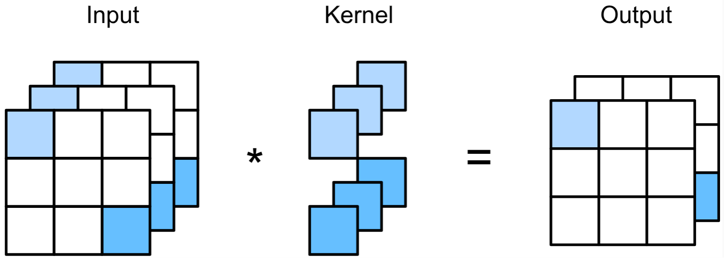
VGG parameters

```
sequential1 output shape: (1, 64, 112, 112)
sequential2 output shape: (1, 128, 56, 56)
sequential3 output shape: (1, 256, 28, 28)
sequential4 output shape: (1, 512, 14, 14)
sequential5 output shape: (1, 512, 7, 7)
dense0 output shape:      (1, 4096)
dropout0 output shape:    (1, 4096)
dense1 output shape:      (1, 4096)
dropout1 output shape:    (1, 4096)
dense2 output shape:      (1, 10)
```

Slide credit: Alex Smola and Mu Li (Berkley)

Breaking the Curse of the Last Layer

- Key Idea
  - **Get rid of the fully connected last layer(s)**
  - Convolutions and pooling reduce resolution (e.g. stride of 2 reduces resolution 4x)
- Implementation details
  - Reduce resolution progressively
  - Increase number of channels
  - Use **1x1 convolutions** (they only act per pixel)

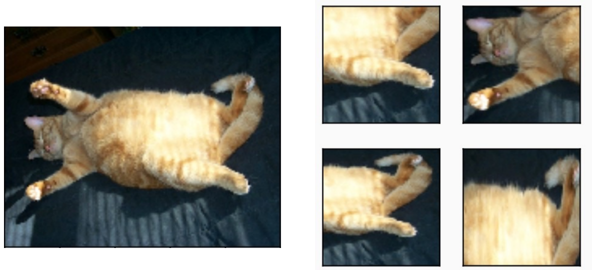


- **Global average pooling in the end**

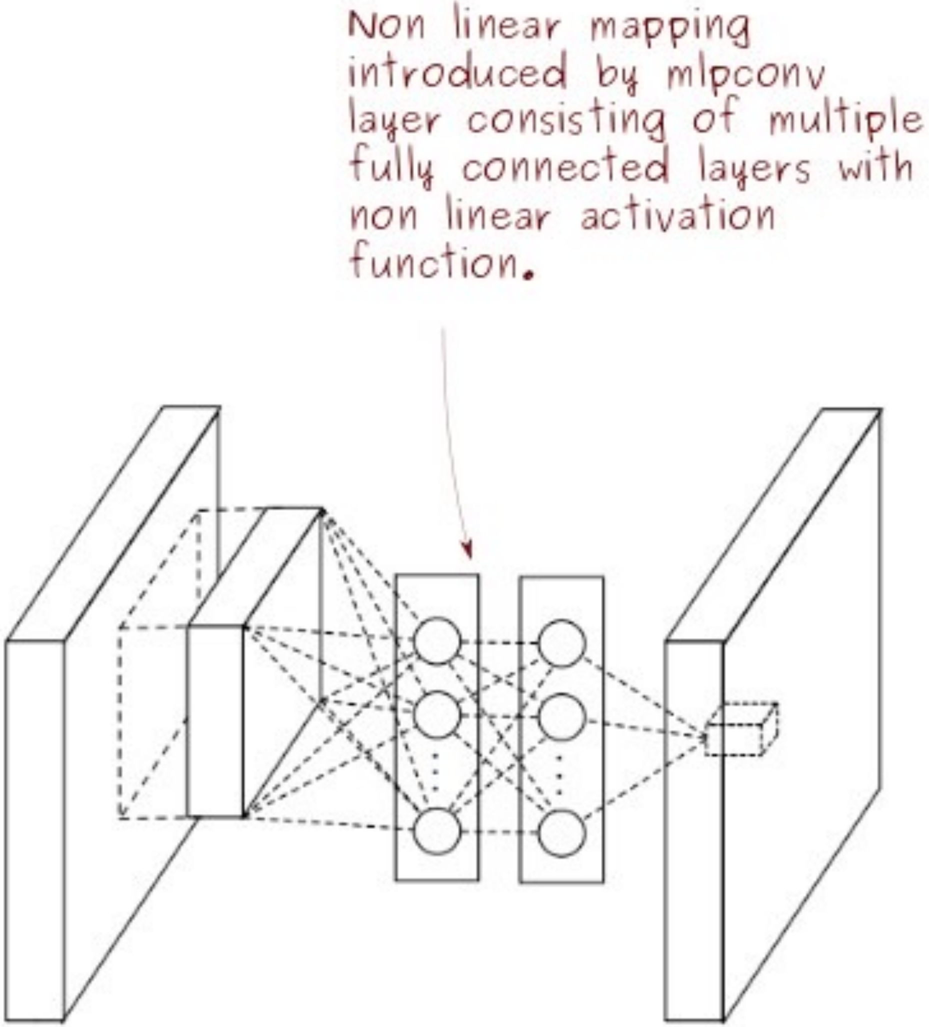
Slide credit: Alex Smola and Mu Li (Berkley)

What's a 1x1 convolution anyway?

- Extreme case
  1x1 image with n channels
- Equivalent to MLP
- Pooling allows for translation invariance of detection (e.g. 5x5)

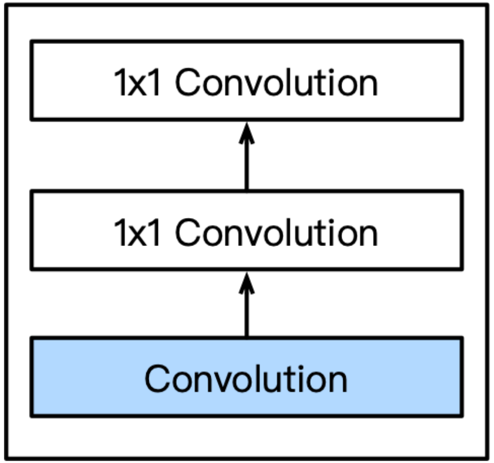


Slide credit: Alex Smola and Mu Li (Berkley)



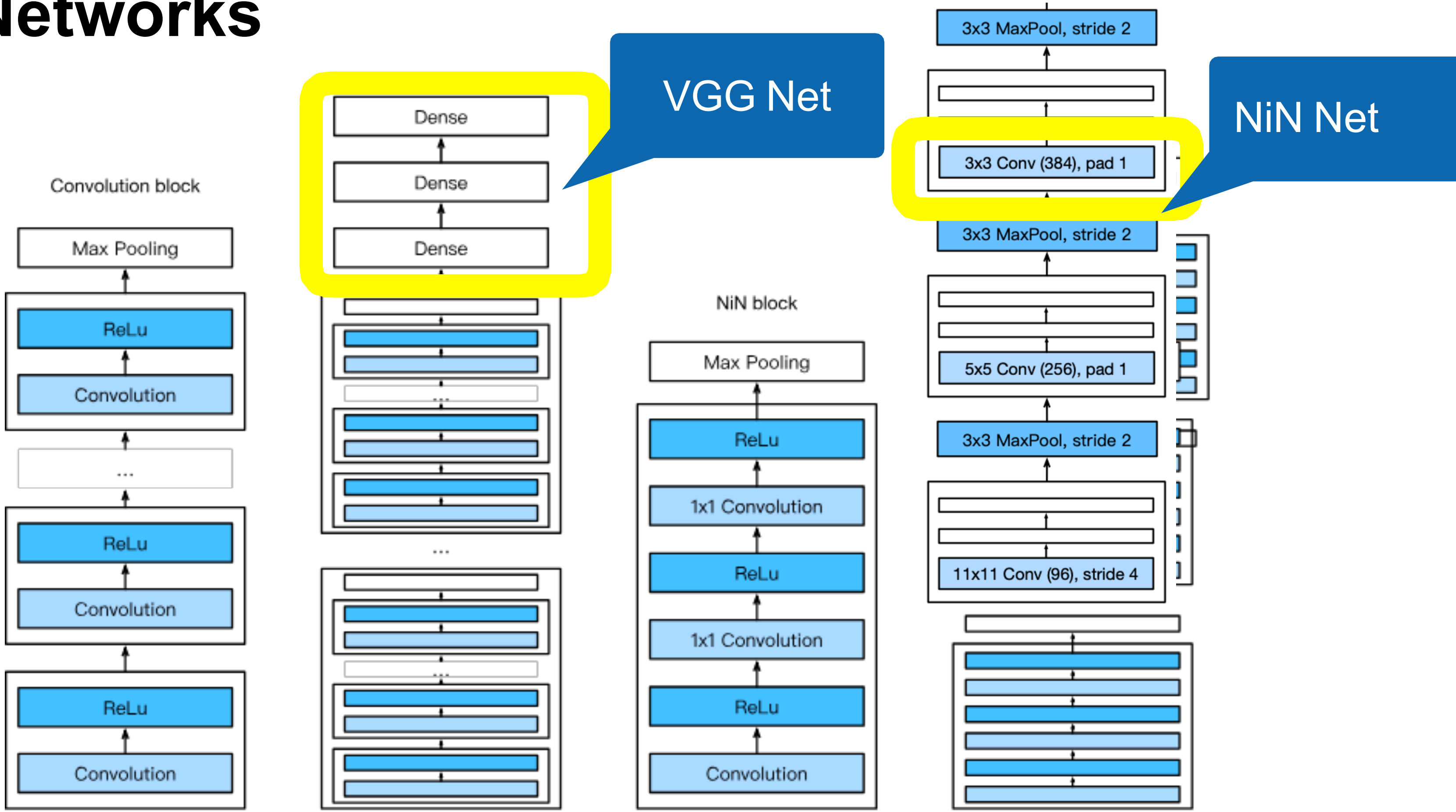
NiN Block

- A convolutional layer
  - kernel size, stride, and padding are hyper-parameters
- Following by two 1x1 convolutions
  - 1 stride and no padding, share the same output channels as first layer
  - Act as dense layers



Slide credit: Alex Smola and Mu Li (Berkley)

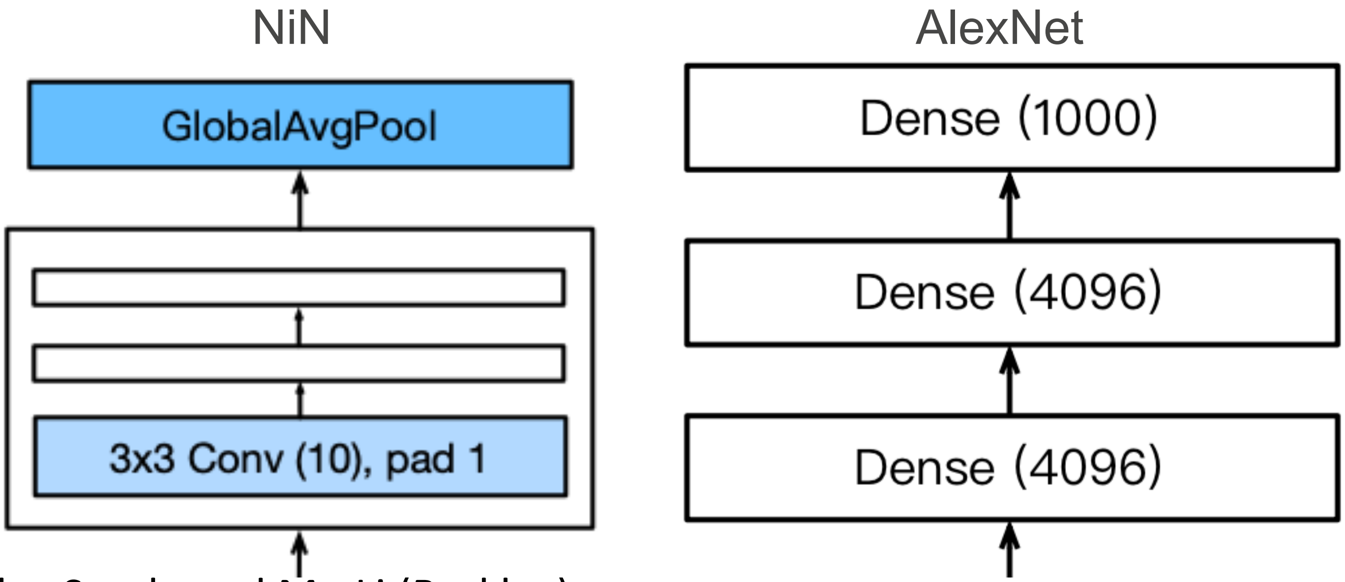
NiN Networks



Slide credit: Alex Smola and Mu Li (Berkley)

NiN Last Layers

- Replaced AlexNet's dense layers with a NiN block
- Global average pooling layer to combine outputs



Slide credit: Alex Smola and Mu Li (Berkley)

Summary

- Reduce image resolution progressively
- Increase number of channels
- Global average pooling for given number of classes

```
sequential1 output shape: (96, 54, 54)
pool0 output shape:       (96, 26, 26)
sequential2 output shape: (256, 26, 26)
pool1 output shape:       (256, 12, 12)
sequential3 output shape: (384, 12, 12)
pool2 output shape:       (384, 5, 5)
dropout0 output shape:    (384, 5, 5)
sequential4 output shape: (10, 5, 5)
pool3 output shape:       (10, 1, 1)
flatten0 output shape:    (10)
```

NIN dimension reduction

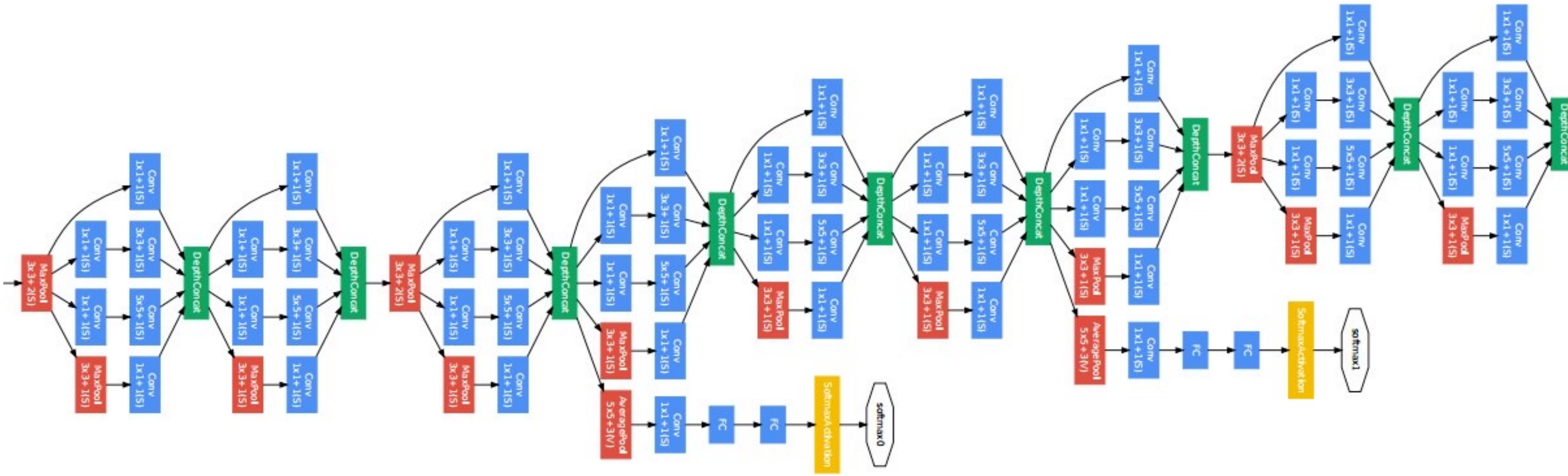
Slide credit: Alex Smola and Mu Li (Berkley)

Summary

- **LeNet** (the first convolutional neural network)
- **AlexNet**
  - More of everything
  - ReLu, Dropout, Invariances
- **VGG**
  - Even more of everything (narrower and deeper)
  - Repeated blocks
- **NiN**
  - 1x1 convolutions + global pooling instead of dense

Slide credit: Alex Smola and Mu Li (Berkley)

Inception

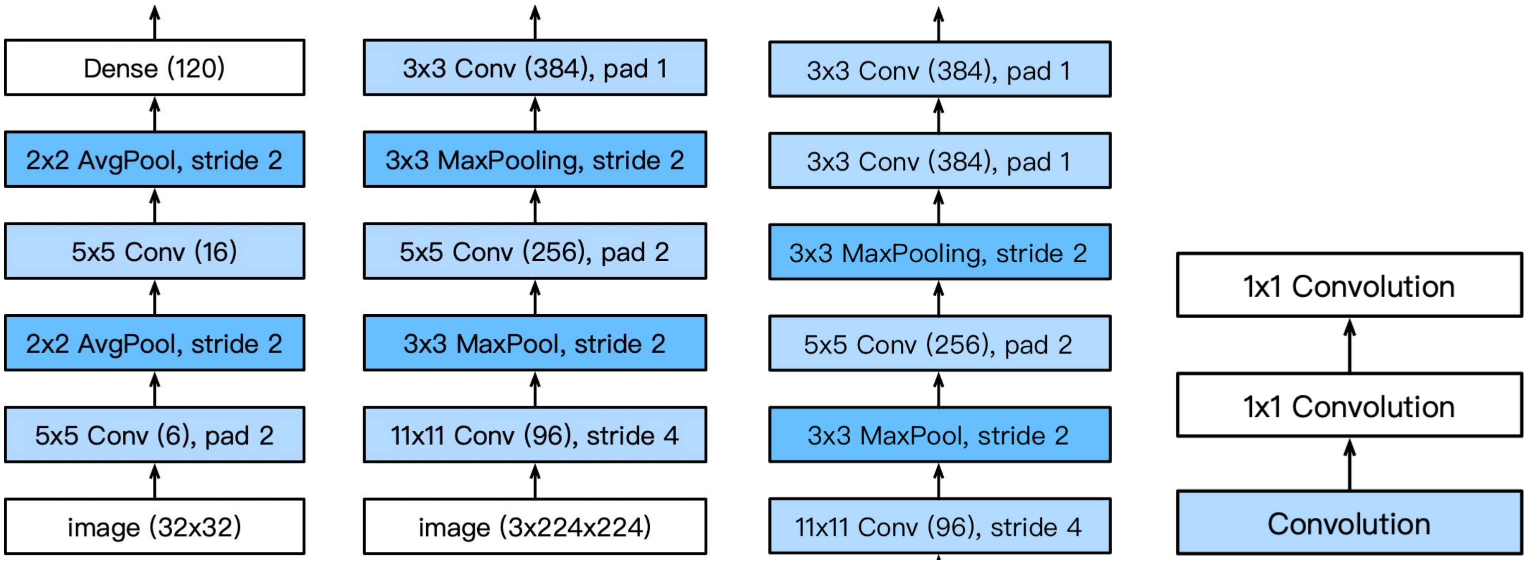


Slide credit: Alex Smola and Mu Li (Berkley)
Slide credit: Alex Smola and Mu Li (Berkley)

Picking the best convolution …

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

LeNet | AlexNet | VGG | NiN



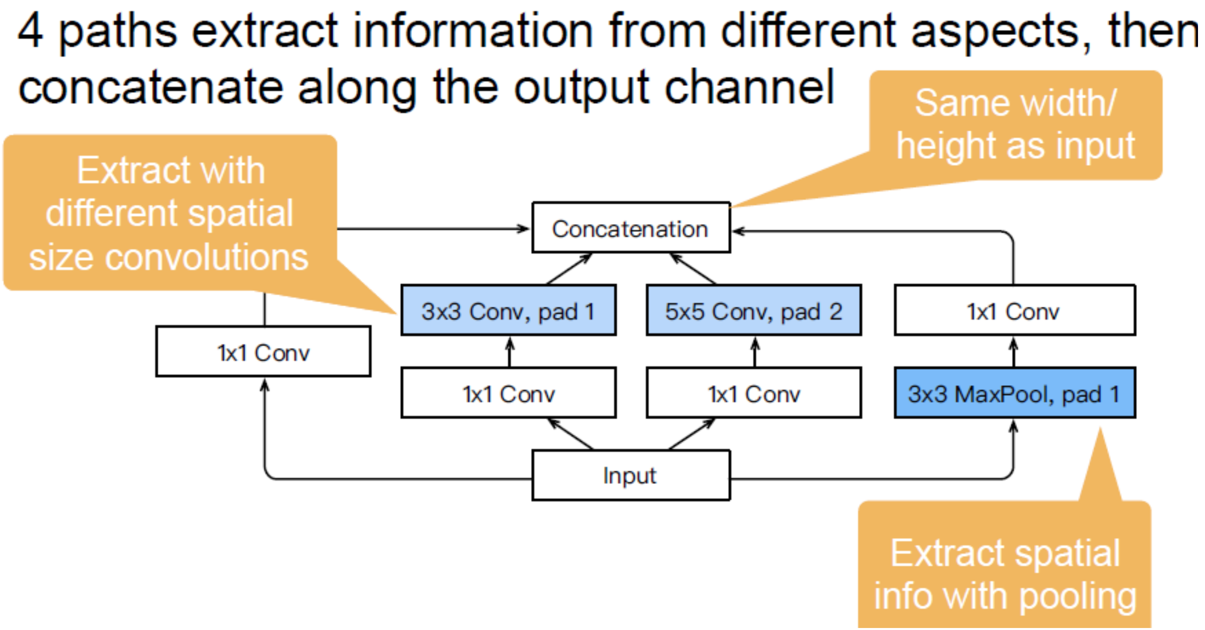
Slide credit: Alex Smola and Mu Li (Berkley)
Slide credit: Alex Smola and Mu Li (Berkley)

Why choose? Just pick them all.

Slide credit: Alex Smola and Mu Li (Berkley)
Slide credit: Alex Smola and Mu Li (Berkley)

Inception Blocks

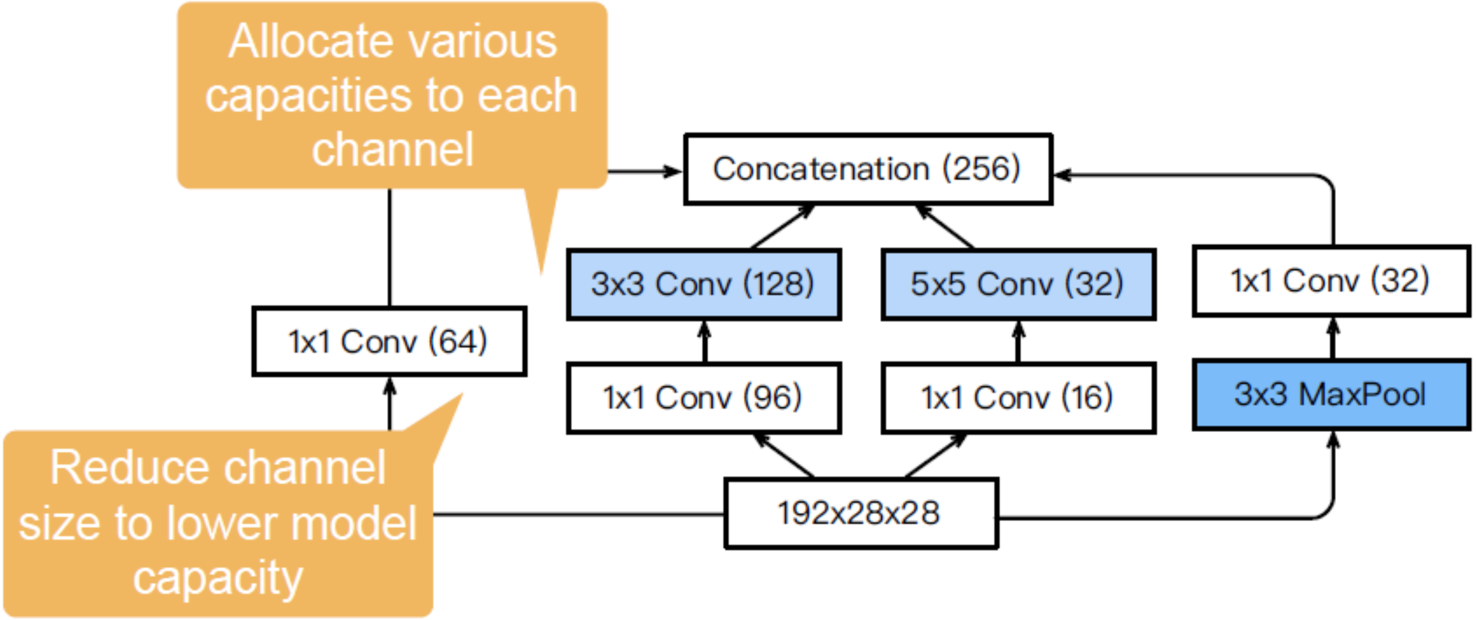
4 paths extract information from different aspects, then concatenate along the output channel



Slide credit: Alex Smola and Mu Li (Berkley)

Inception Blocks

The first inception block with channel sizes specified



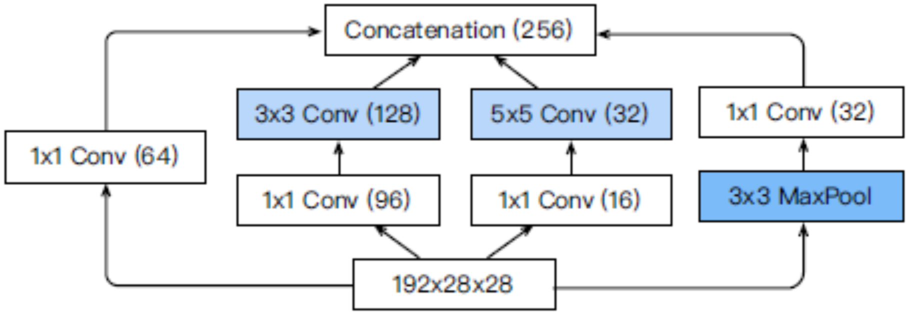
Slide credit: Alex Smola and Mu Li (Berkley)

Inception Blocks

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

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

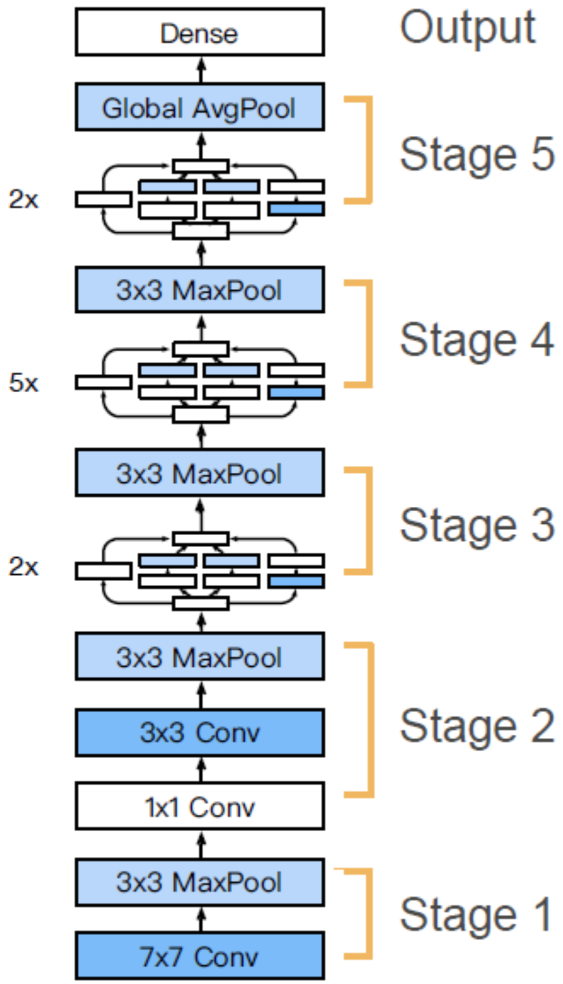
	#parameters
Inception	0.16 M
3x3 Conv	0.44 M
5x5 Conv	1.22 M



Slide credit: Alex Smola and Mu Li (Berkley)

GoogLeNet

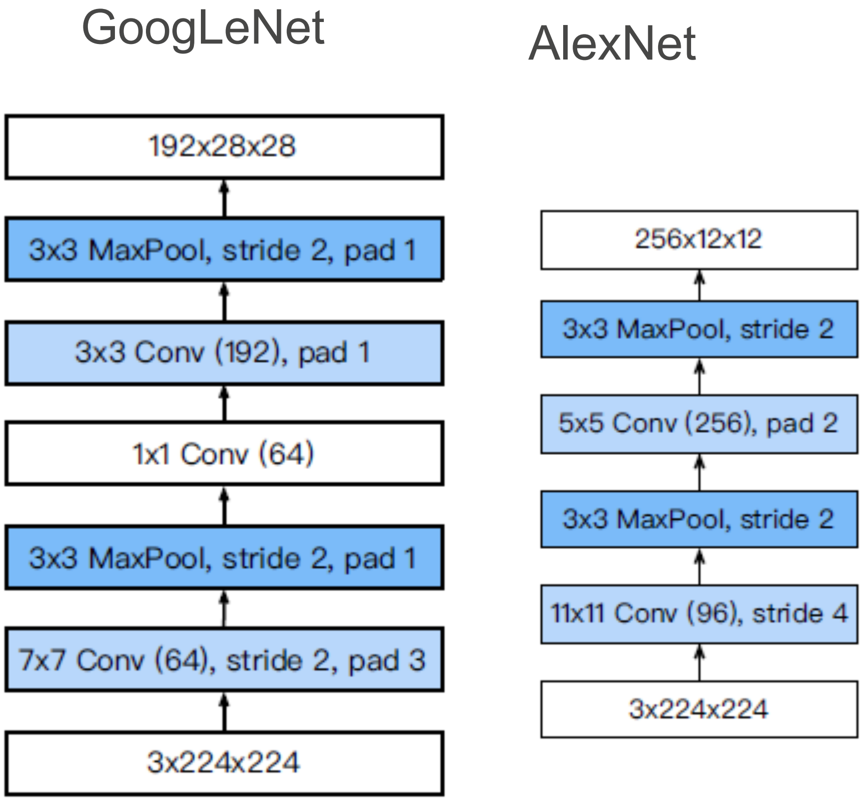
- 5 stages with 9 inceptions blocks



Slide credit: Alex Smola and Mu Li (Berkley)

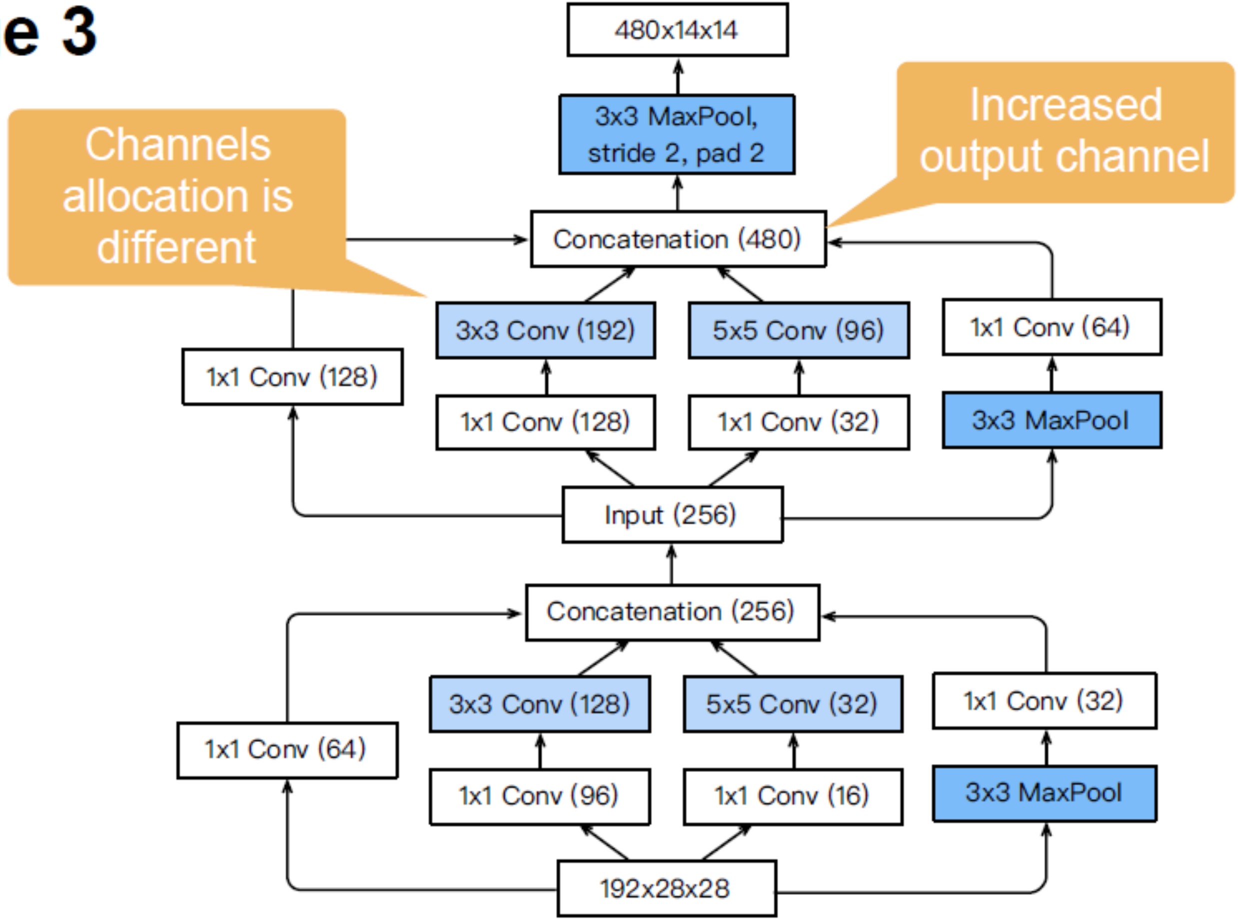
Stage 1 & 2

- Smaller kernel size and output channels due to more layers



Slide credit: Alex Smola and Mu Li (Berkley)

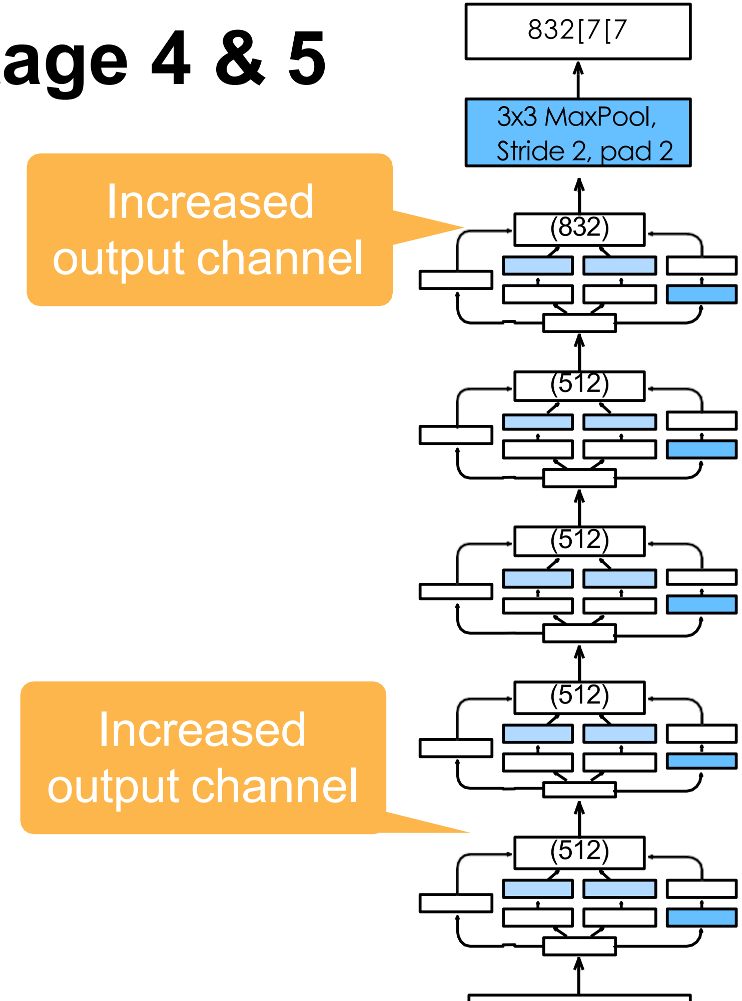
Stage 3

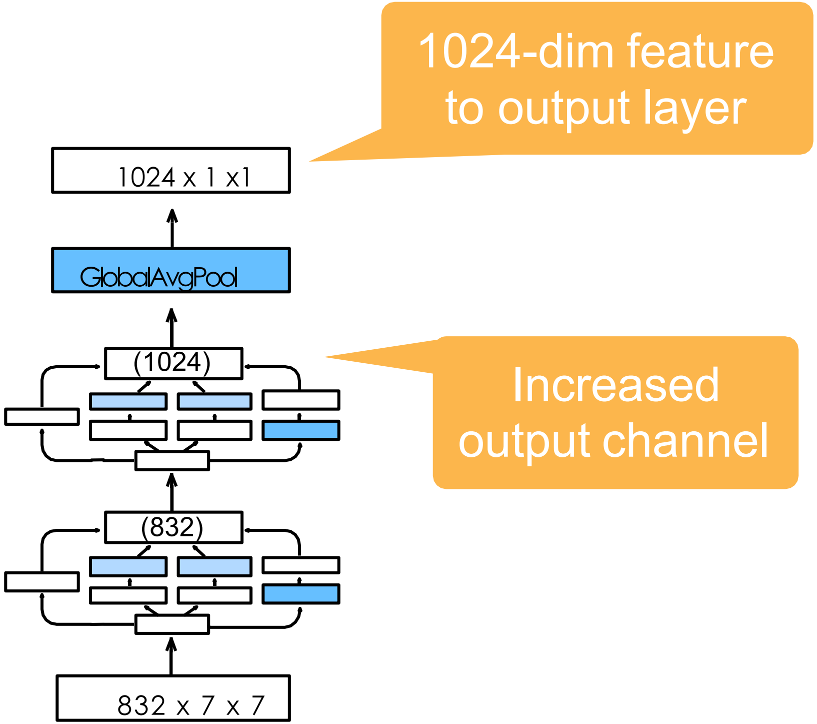


Slide credit: Alex Smola and Mu Li (Berkley)

Stage 4 & 5

Slide credit: Alex Smola and Mu Li (Berkley)





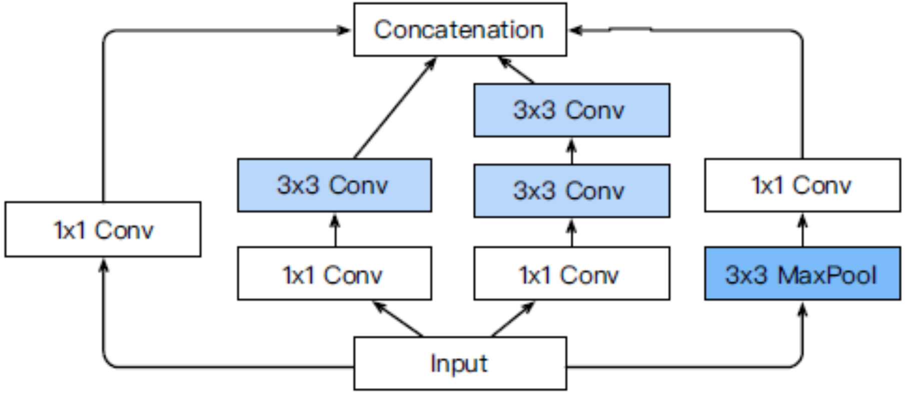
Slide credit: Alex Smola and Mu Li (Berkley)

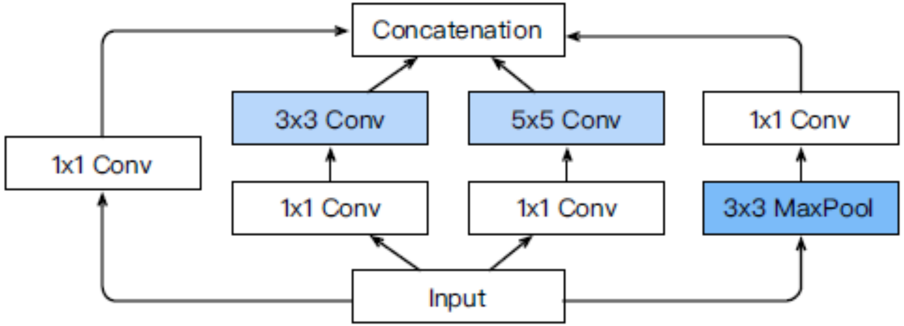
The many flavors of Inception Networks

- 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)

Slide credit: Alex Smola and Mu Li (Berkley)

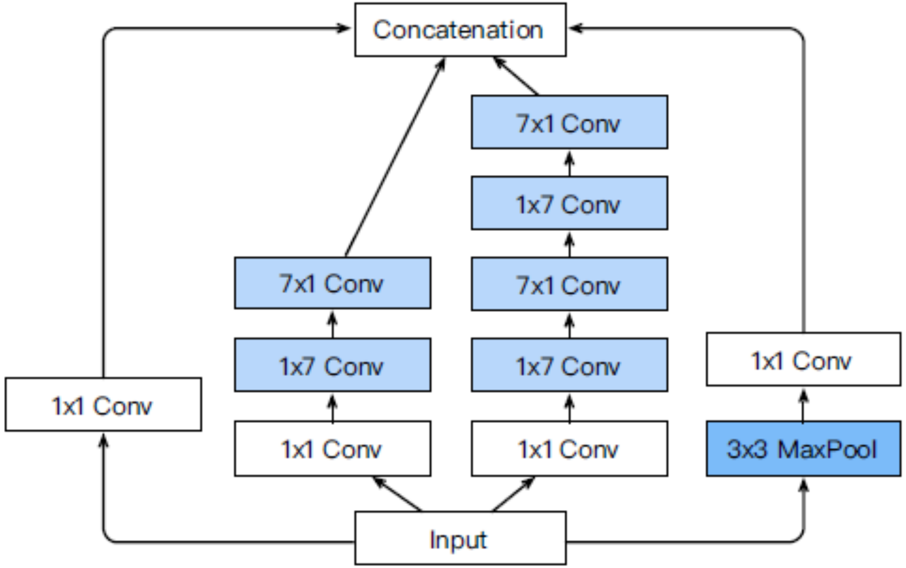
Inception V3 Block for Stage 3

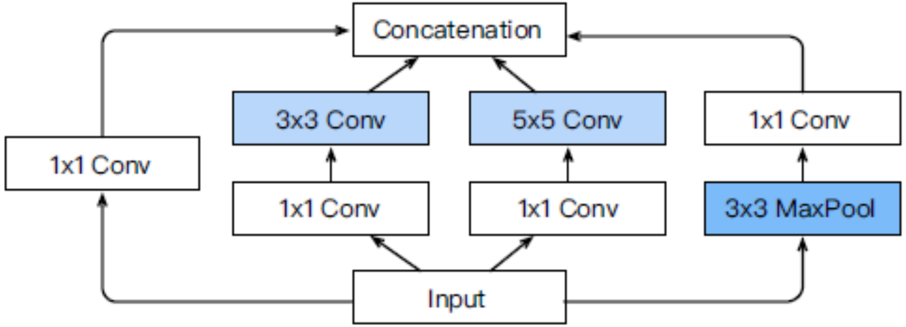




Slide credit: Alex Smola and Mu Li (Berkley)

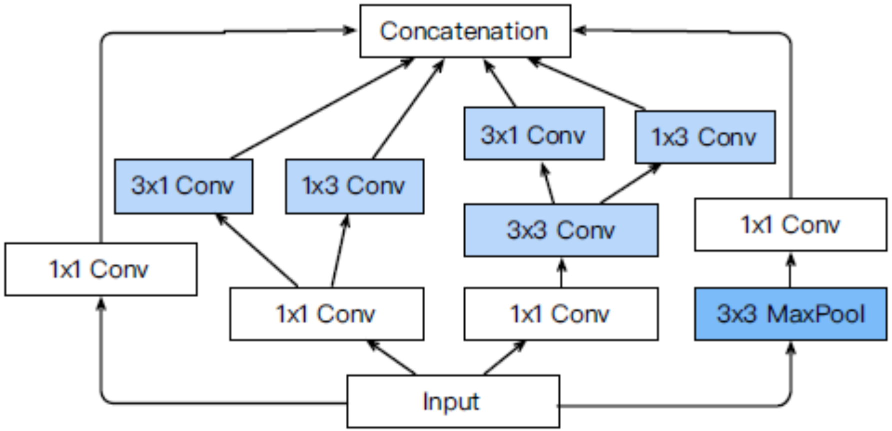
Inception V3 Block for Stage 4

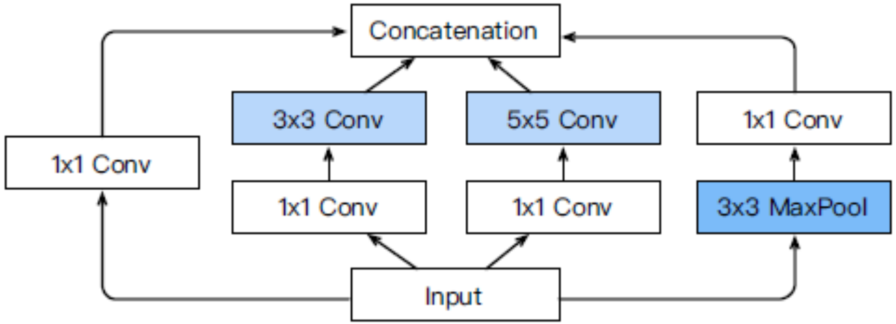




Slide credit: Alex Smola and Mu Li (Berkley)

Inception V3 Block for Stage 5



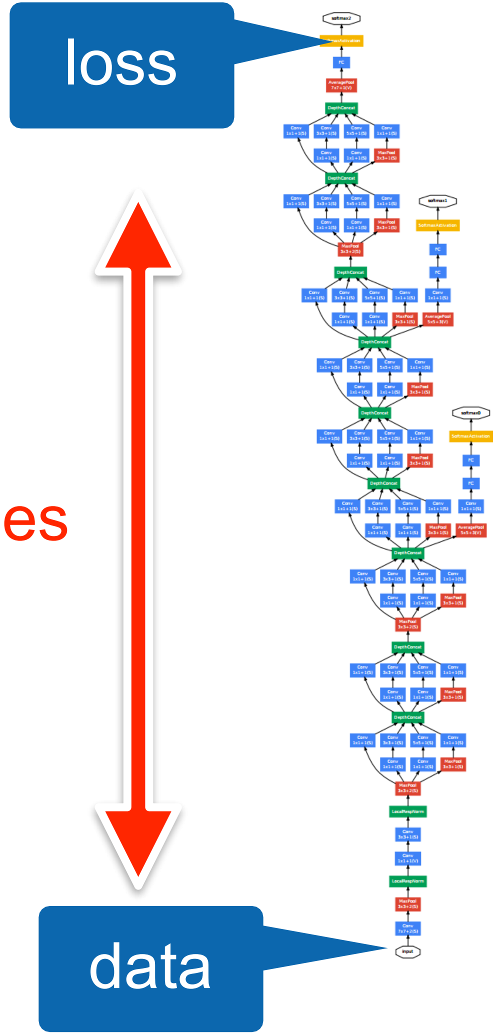


Slide credit: Alex Smola and Mu Li (Berkley)

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 last layers while learning first layers?



Slide credit: Alex Smola and Mu Li (Berkley)

Batch Normalization

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

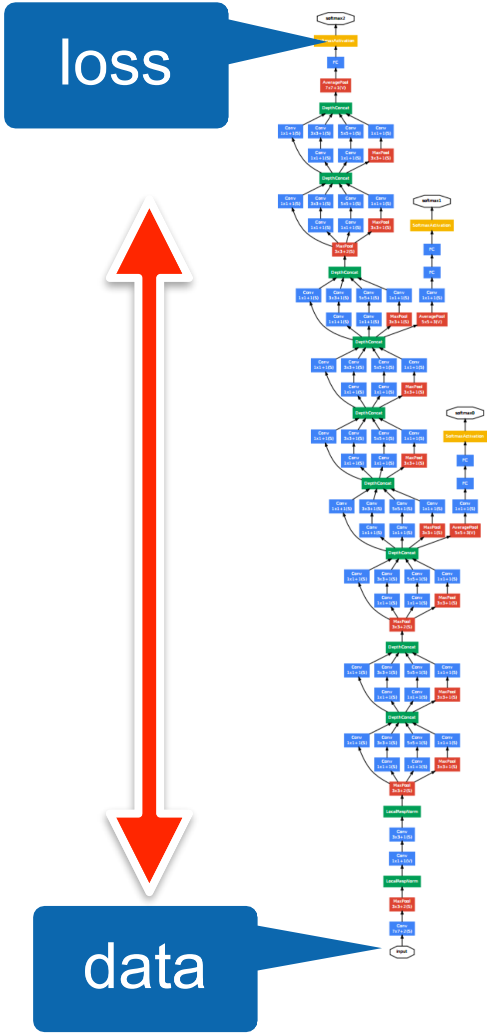
$$\mu_B = \frac{1}{|B|}\sum_{i \in B} x_i \text{ and } \sigma_B^2 = \frac{1}{|B|}\sum_{i \in B}(x_i - \mu_B)^2 + \epsilon$$

and adjust it separately

mean

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

variance



Slide credit: Alex Smola and Mu Li (Berkeley)

This was the original motivation …

Slide credit: Alex Smola and Mu Li (Berkley)

What Batch Norms really do

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

Random offset

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

Random scale

  - 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

Slide credit: Alex Smola and Mu Li (Berkley)

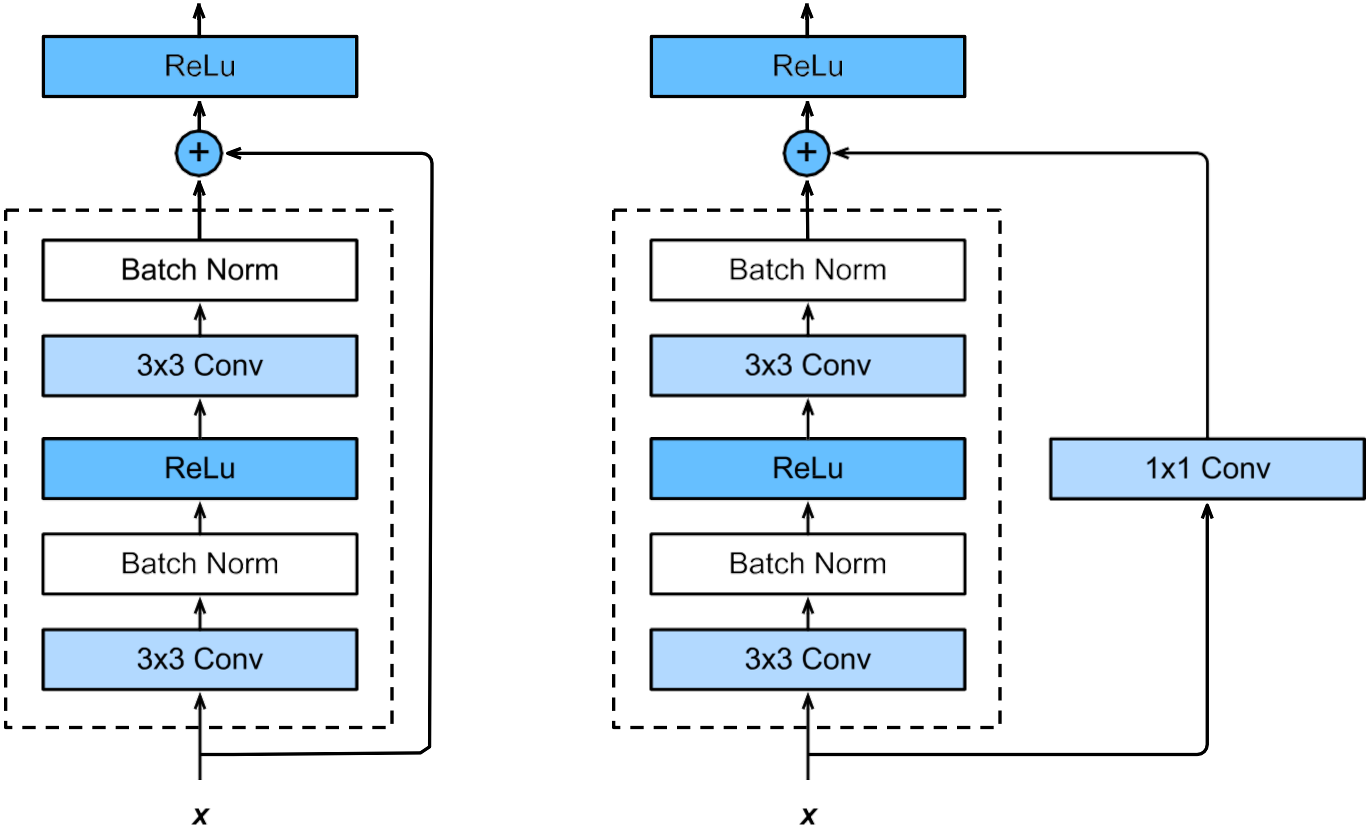
Details

`torch.nn.BatchNorm(…)`

- **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)

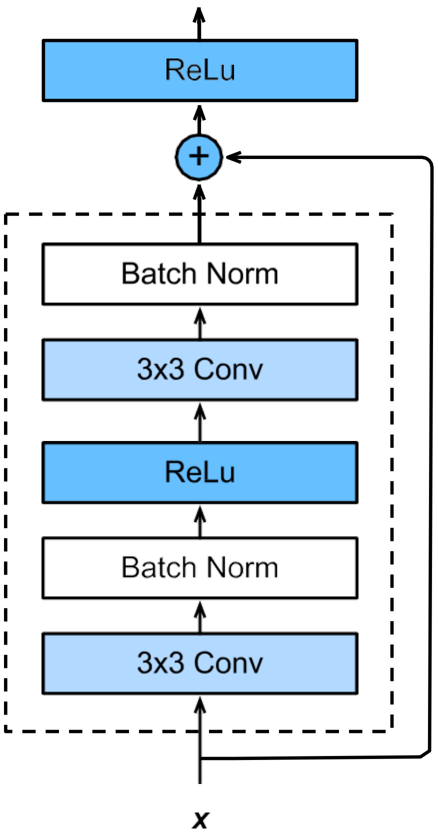
Slide credit: Alex Smola and Mu Li (Berkley)

ResNet Block in detail



Slide credit: Alex Smola and Mu Li (Berkley)

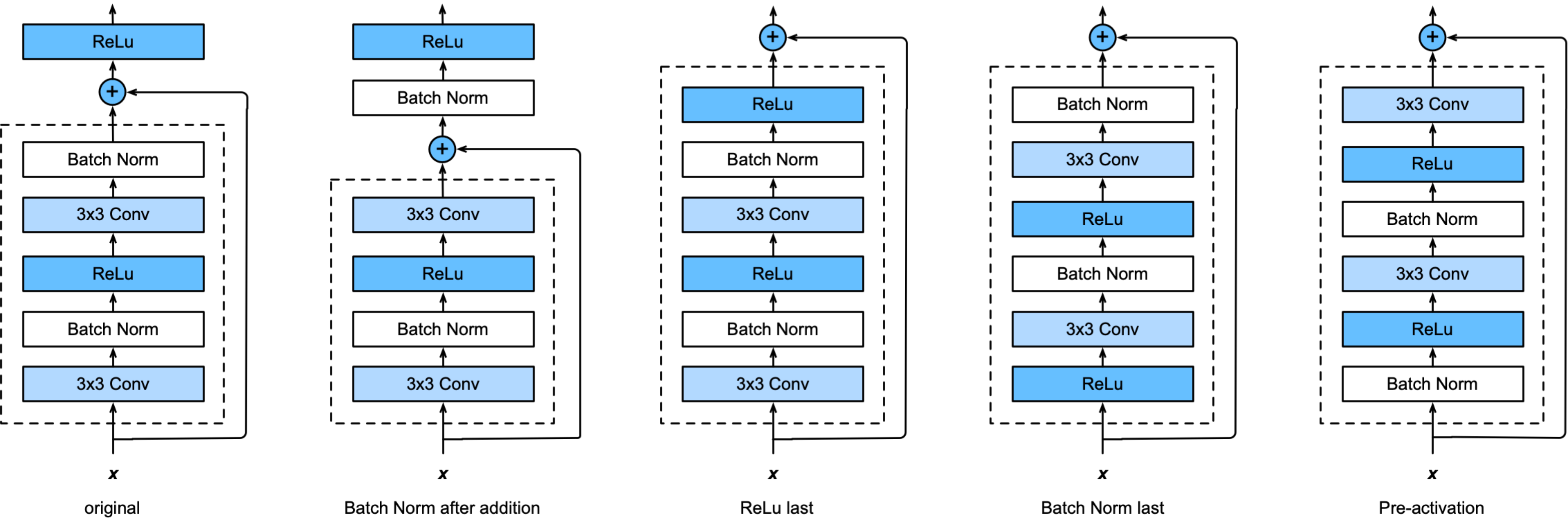
In code




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

Slide credit: Alex Smola and Mu Li (Berkley)

The many flavors of ResNet blocks



Try every permutation

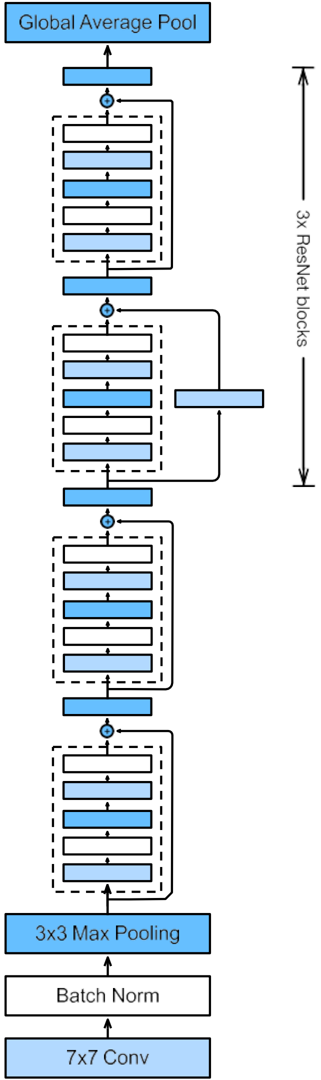
Slide credit: Alex Smola and Mu Li (Berkley)

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 …

Slide credit: Alex Smola and Mu Li (Berkley)



Acknowledgement

All the slides in this presentation have been borrowed/inspired from course taught by Alex Smola and Mu Li (Berkley) in 2019. I would like to thank Alex Smola and Mu Li for generously sharing their course material and allowing me to use their material.