Dive into Deep Learning in 1 Day

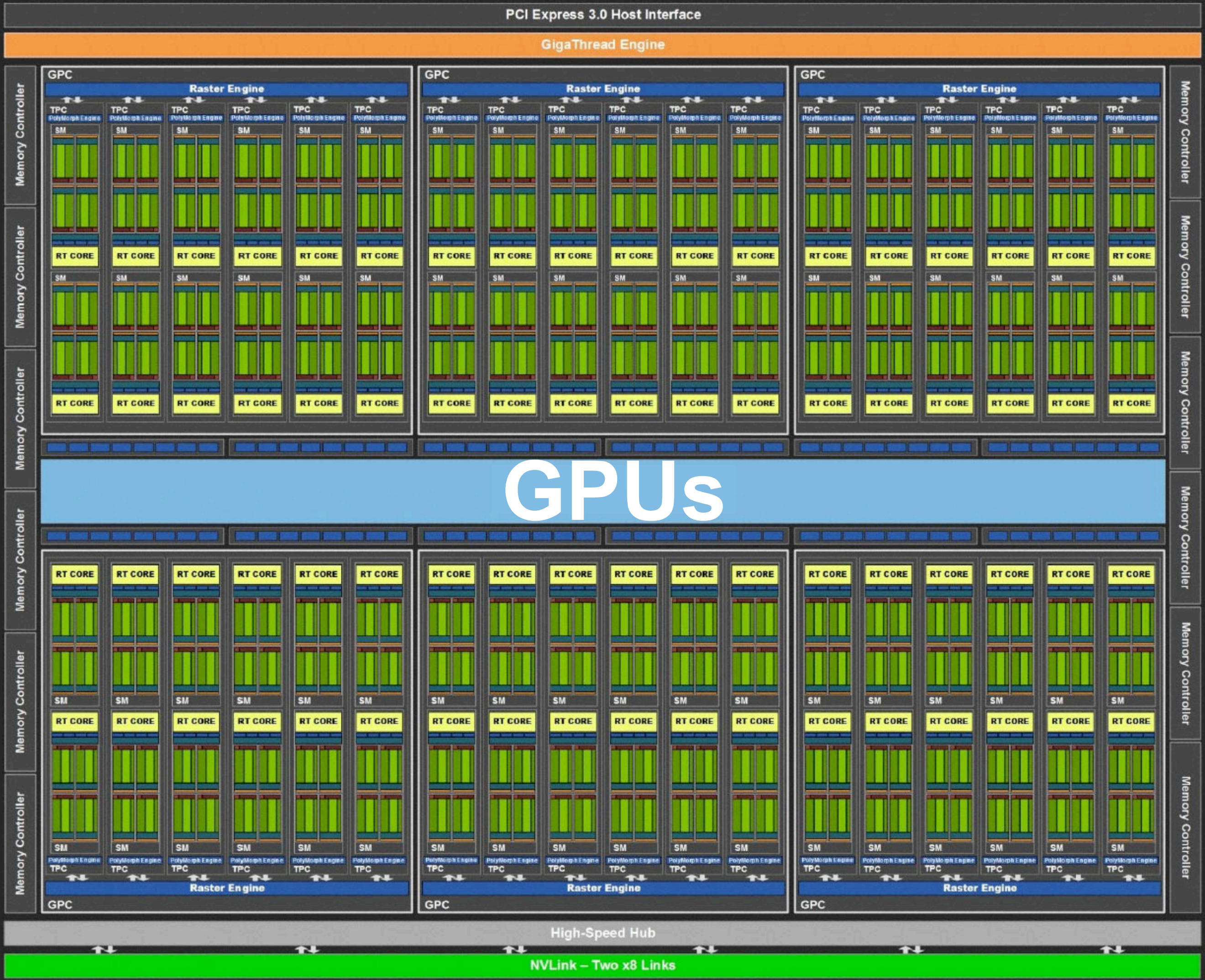
1 Basics · **2 Convnets** · 3 Computation · 4 Sequences

ODSC 2019

Alex Smola
http://courses.d2l.ai/odsc2019/

Outline

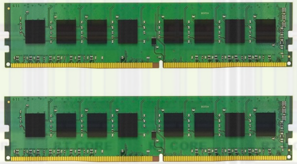
- GPUs
- Convolutions
- Pooling, Padding and Stride
- Convolutional Neural Networks (LeNet)
- Deep ConvNets (AlexNet)
- Networks using Blocks (VGG)
- Residual Neural Networks (ResNet)



NVIDIA Turing TU102

Highend Gaming / DeepLearning PC

Intel i7
0.15 TFLOPS

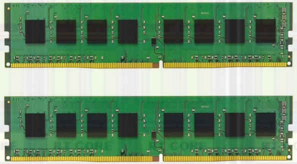


DDR4
32 GB

Nvidia Titan RTX
12 TFLOPS (130TF for FP16 TensorCores)
24 GB

Highend Gaming / DeepLearning PC

Intel i7
0.15 TFLOPS



DDR4
32 GB

`ctx = npx.cpu()`

`x.copyto(ctx)`

Nvidia Titan RTX
12 TFLOPS (130TF for FP16 TensorCores)
24 GB

`ctx = npx.gpu(0)`

GPU Notebook


From fully connected
to convolutions

Classifying Dogs and Cats in Images

- Use a good camera
- RGB image has 36M elements
- The model size of a single hidden layer MLP with a 100 hidden size is 3.6 Billion parameters
- Exceeds the population of dogs and cats on earth
  (900M dogs + 600M cats)

Flashback - Network with one hidden layer

Output layer

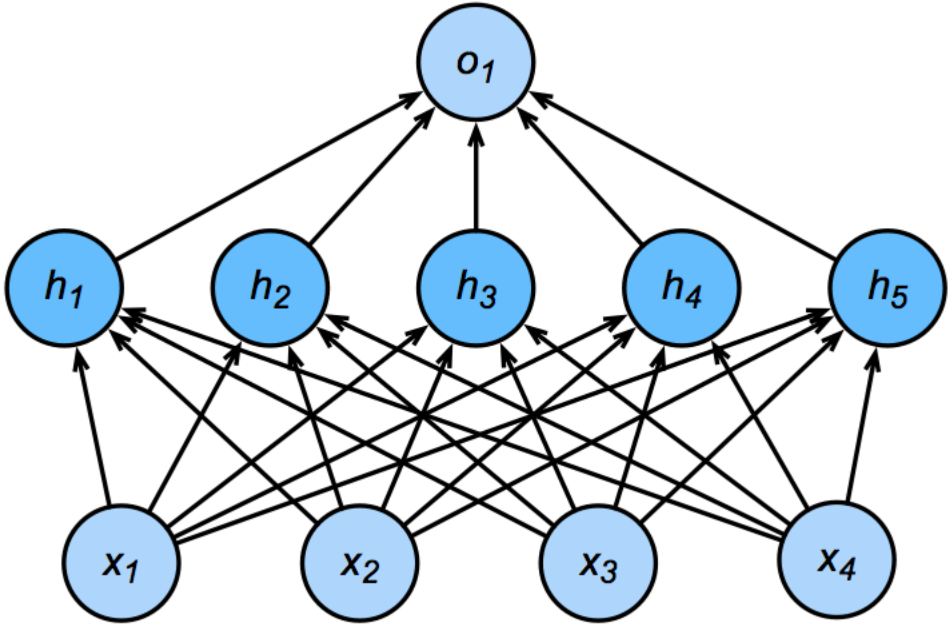
100 neurons

Hidden layer

3.6B parameters = 14GB

36M features

Input layer



$$\mathbf{h} = \sigma\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)$$

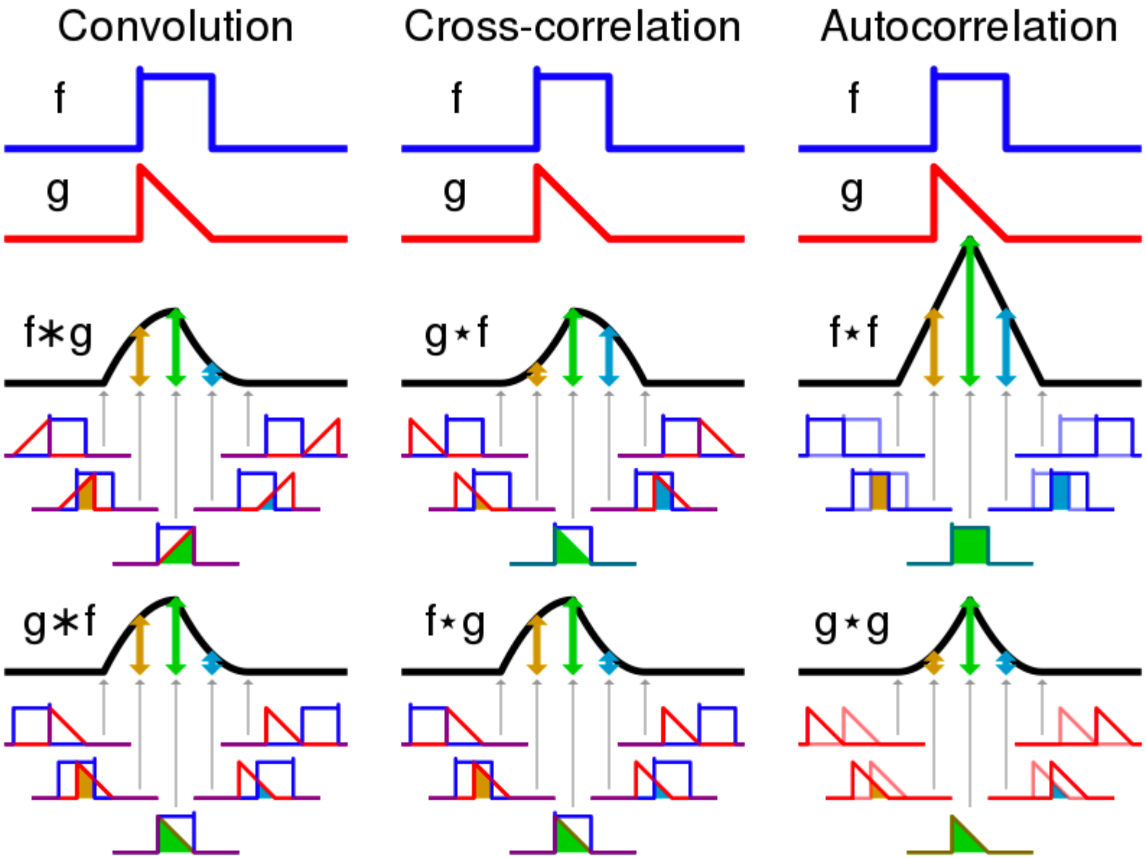
Where is Waldo?

numpy.d2l.ai

Two Principles

- Translation Invariance
- Locality

Convolution

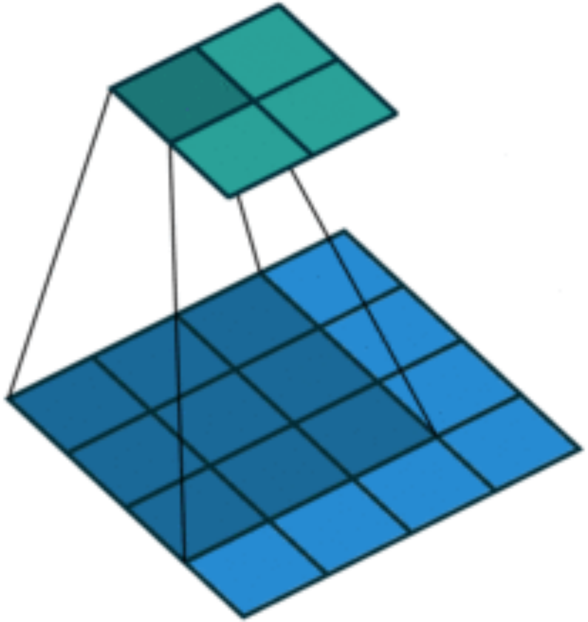


2-D Cross Correlation

Input, Kernel, Output

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$$

$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

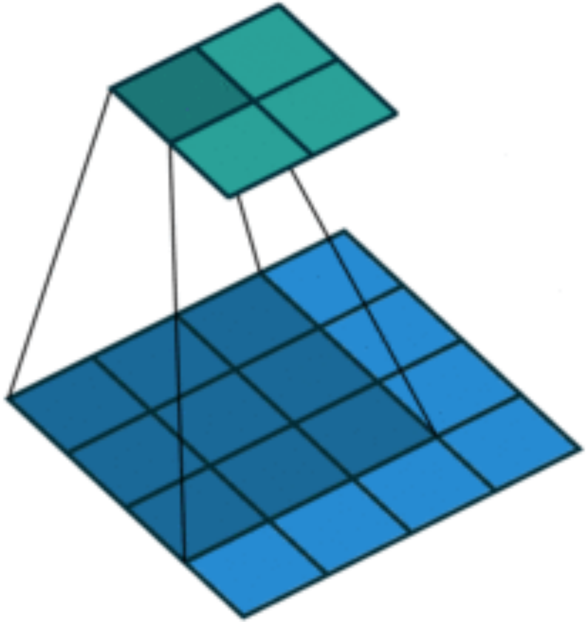


(vdumoulin@ Github)

2-D Cross Correlation

Input		Kernel		Output
$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$	$*$	$\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$	$=$	$\begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$

$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$



(vdumoulin@ Github)

2-D Convolution Layer

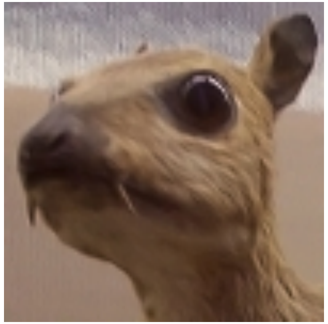
$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$$

- $\mathbf{X} : n_h \times n_w$ input matrix
- $\mathbf{W} : k_h \times k_w$ kernel matrix
- b: scalar bias
- $\mathbf{Y} : (n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

- **W** and *b* are learnable parameters

Examples

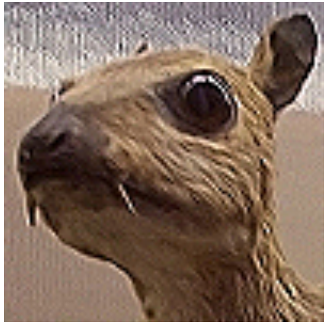


(wikipedia)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



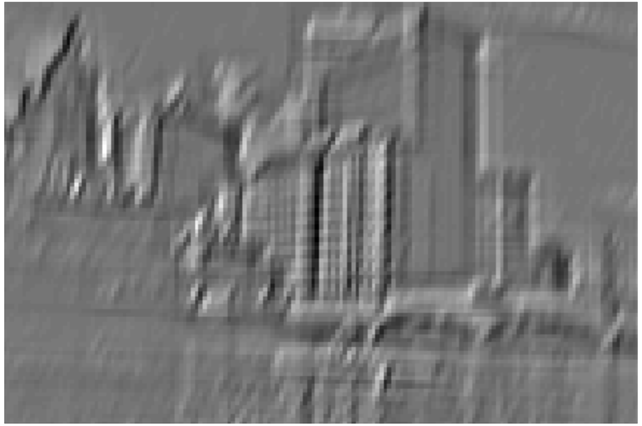
Sharpen

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian Blur

Examples

(Rob Fergus)



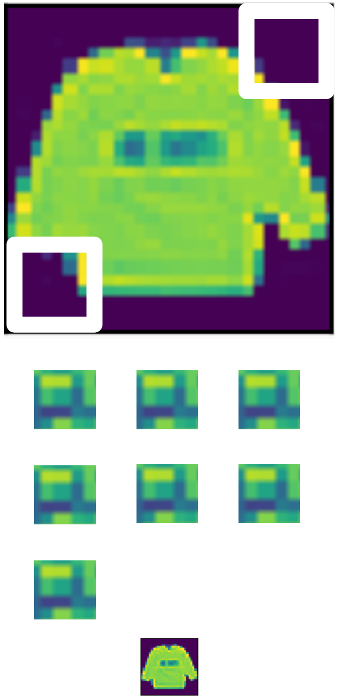
Convolutions Notebook


Padding and Stride

Padding

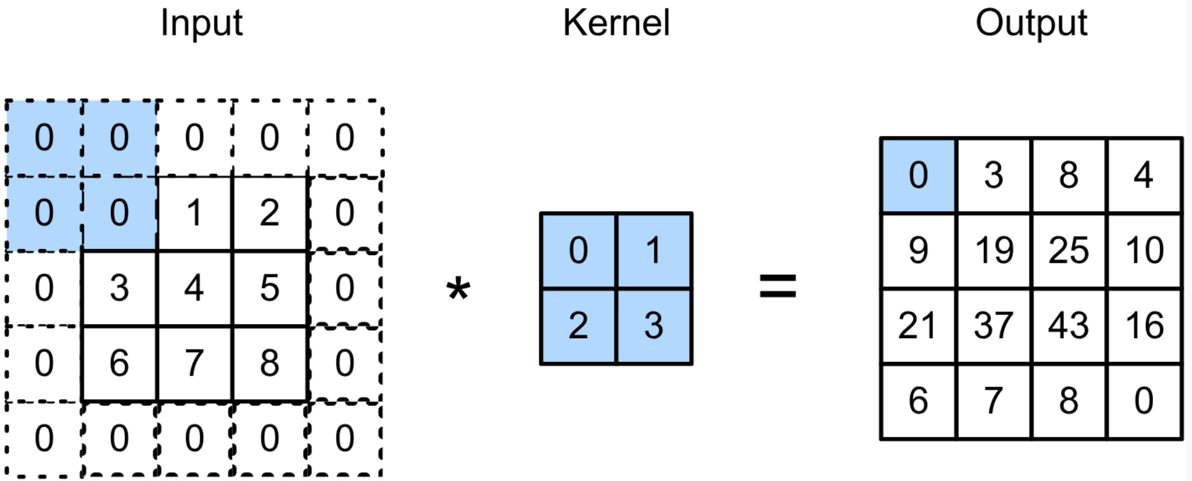
- Given a 32 x 32 input image
- Apply convolutional layer with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels
  - Shape reduces from $n_h \times n_w$ to

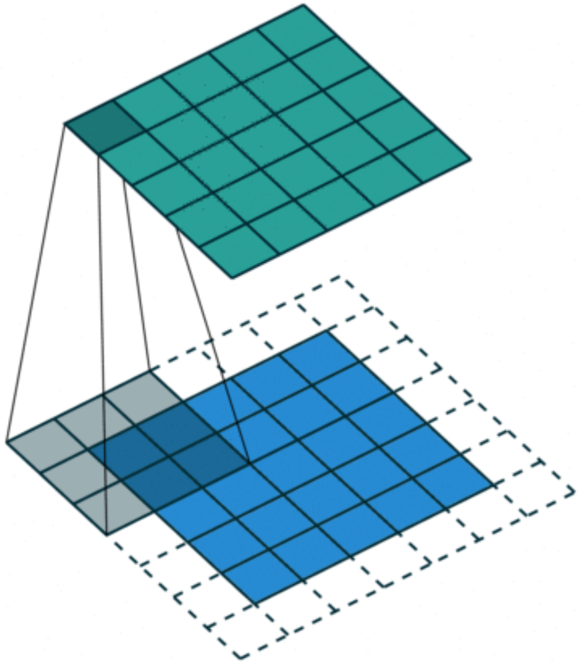
$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$



Padding

Padding adds rows/columns around input

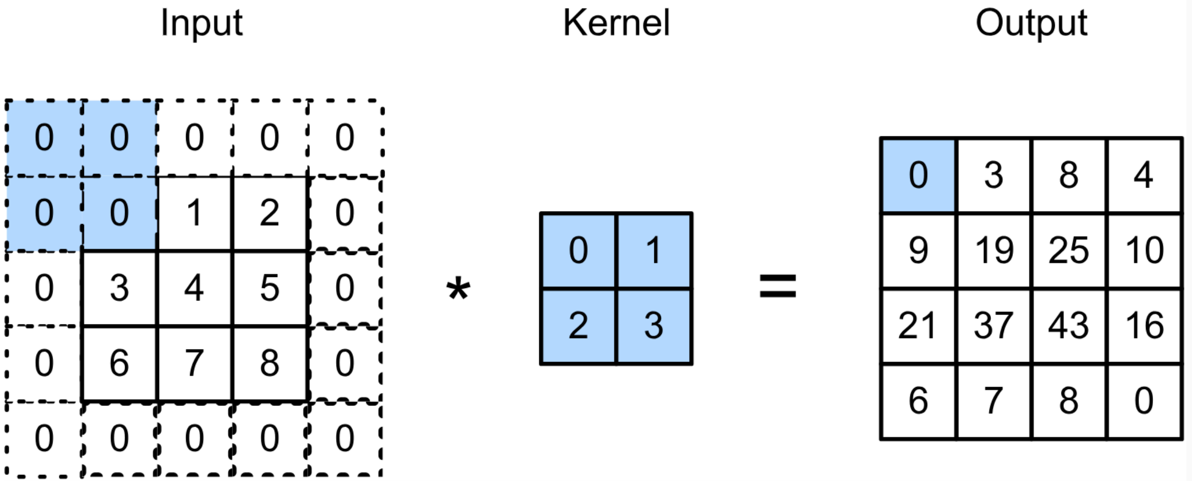


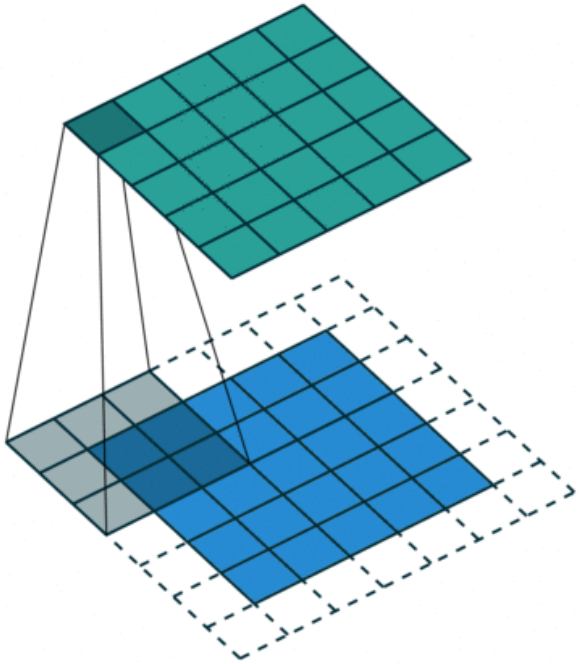


$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

Padding

Padding adds rows/columns around input





$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

Padding

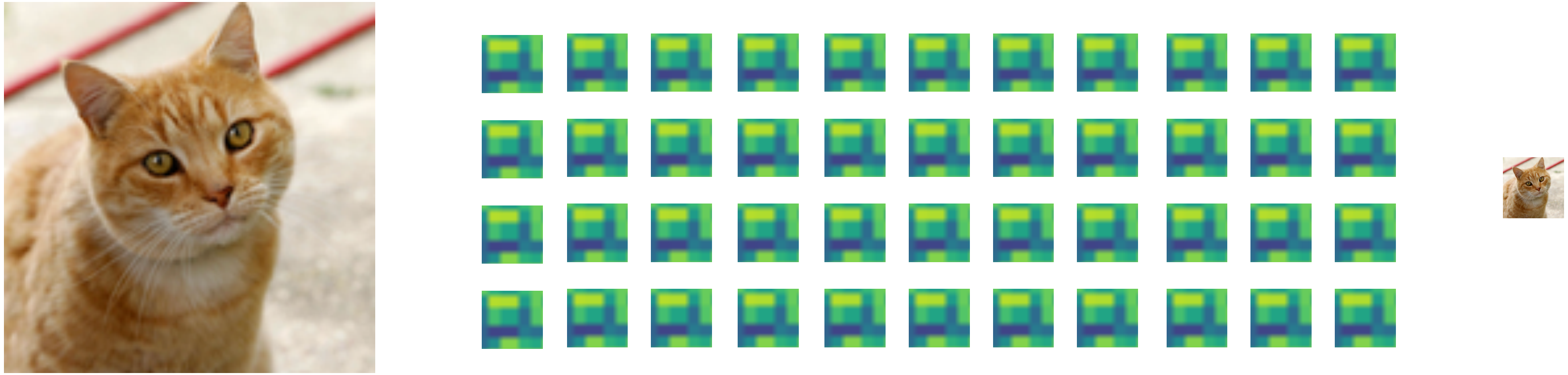
- Padding p_h rows and p_w columns, output shape will be

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

- A common choice is $p_h = k_h - 1$ and $p_w = k_w - 1$
  - Odd k_h : pad $p_h/2$ on both sides
  - Even k_h : pad $\lceil p_h/2 \rceil$ on top, $\lfloor p_h/2 \rfloor$ on bottom

Stride

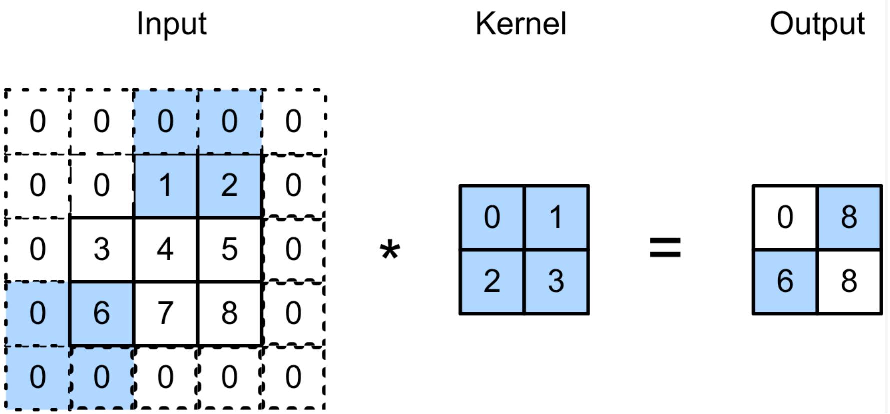
- Padding reduces shape linearly with #layers
  - Given a 224 x 224 input with a 5 x 5 kernel, needs 44 layers to reduce the shape to 4 x 4
  - Requires a large amount of computation



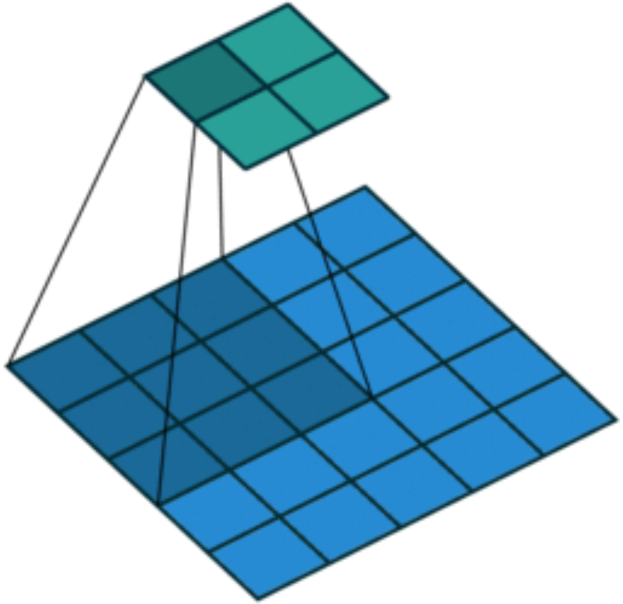
Stride

- Stride is the #rows/#columns per slide

Strides of 3 and 2 for height and width



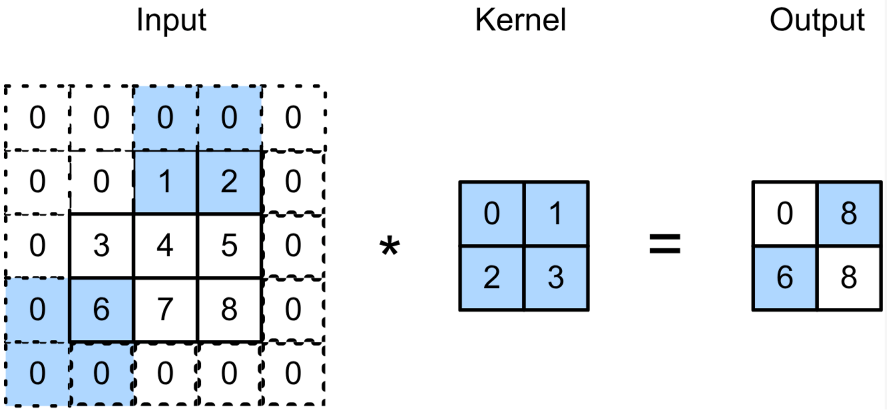
$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$



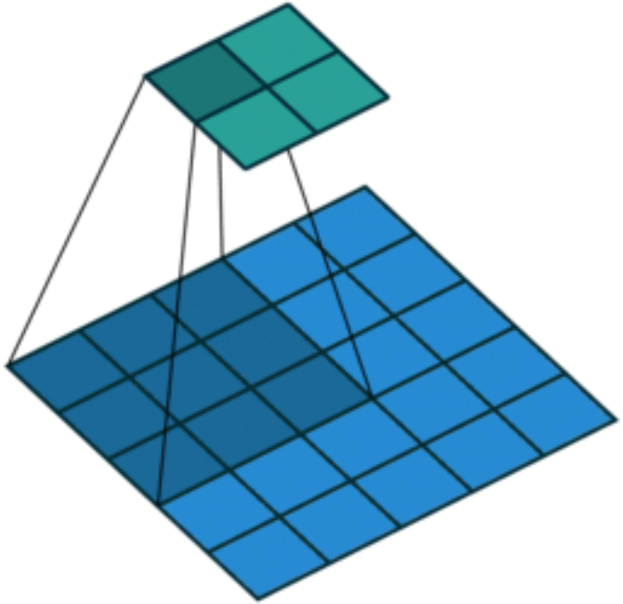
Stride

- Stride is the #rows/#columns per slide

Strides of 3 and 2 for height and width



$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$



Stride

- Given stride s_h for the height and stride s_w for the width, the output shape is

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

- With $p_h = k_h - 1$ and $p_w = k_w - 1$

$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$$

- If input height/width are divisible by strides

$$(n_h/s_h) \times (n_w/s_w)$$


Multiple Input and
Output Channels

Multiple Input Channels

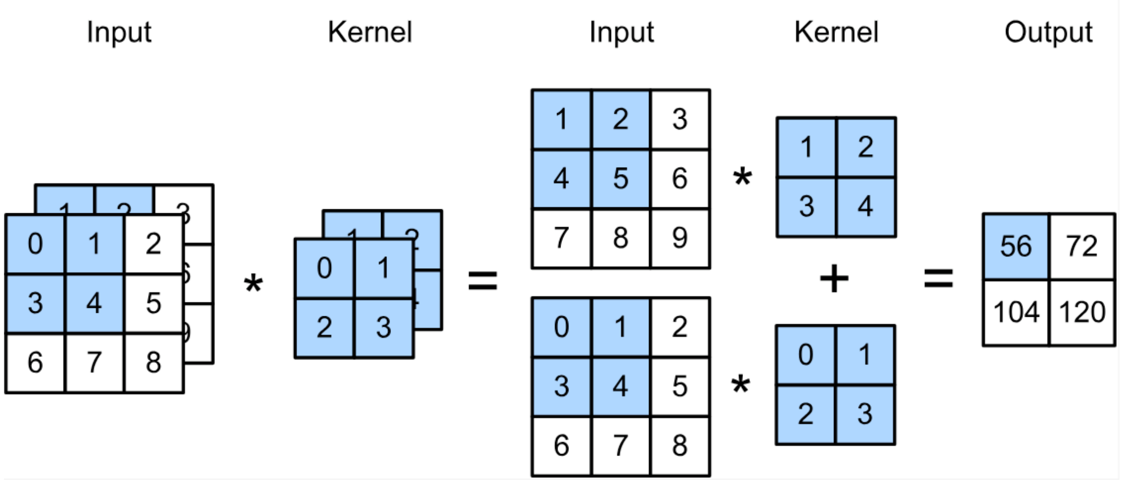
- Color image may have three RGB channels
- Converting to grayscale loses information

Multiple Input Channels

- Color image may have three RGB channels
- Converting to grayscale loses information

Multiple Input Channels

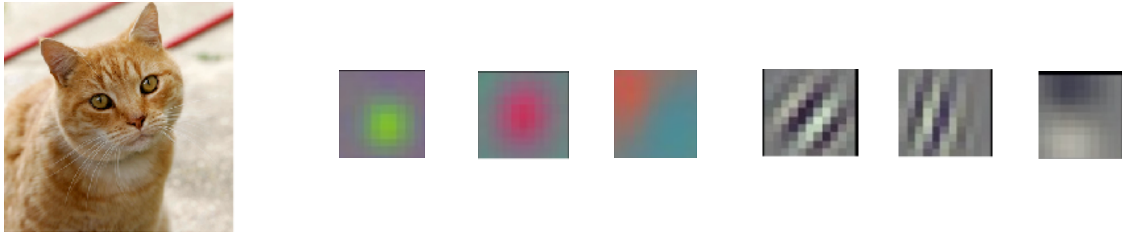
- Have a kernel for each channel, and then sum results over channels



$$
\begin{aligned}
&(1\times 1 + 2\times 2 + 4\times 3 + 5\times 4)\\
&+(0\times 0 + 1\times 1 + 3\times 2 + 4\times 3)\\
&= 56
\end{aligned}
$$

Multiple Input/Output Channels

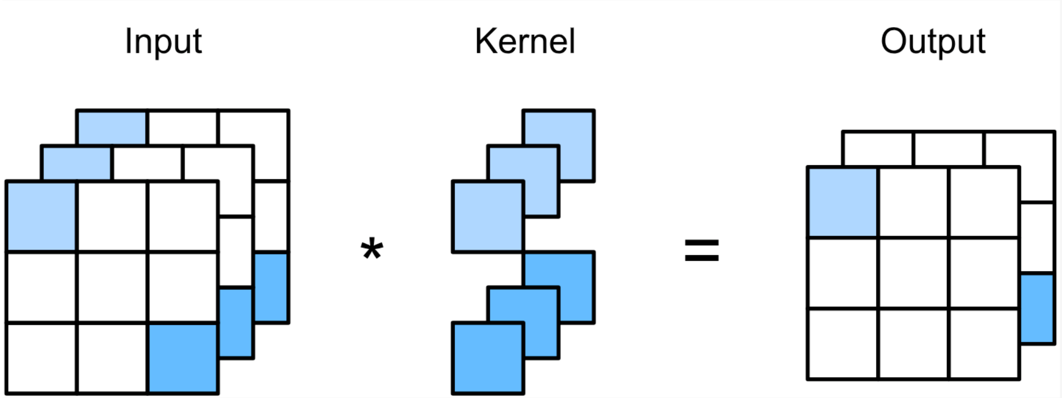
- Each output channel may recognize a particular pattern



- Input channels kernels recognize and combines patterns in inputs

1 x 1 Convolutional Layer

$k_h = k_w = 1$ is a popular choice. It doesn't recognize spatial patterns, but fuse channels.



Equal to a dense layer with $n_h n_w \times c_i$ input and $c_o \times c_i$ weight.

2-D Convolution Layer Summary

- Input $\mathbf{X} : c_i \times n_h \times n_w$
- Kernel $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- Bias $\mathbf{B} : c_o \times c_i$
- Output $\mathbf{Y} : c_o \times m_h \times m_w$

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + \mathbf{B}$$

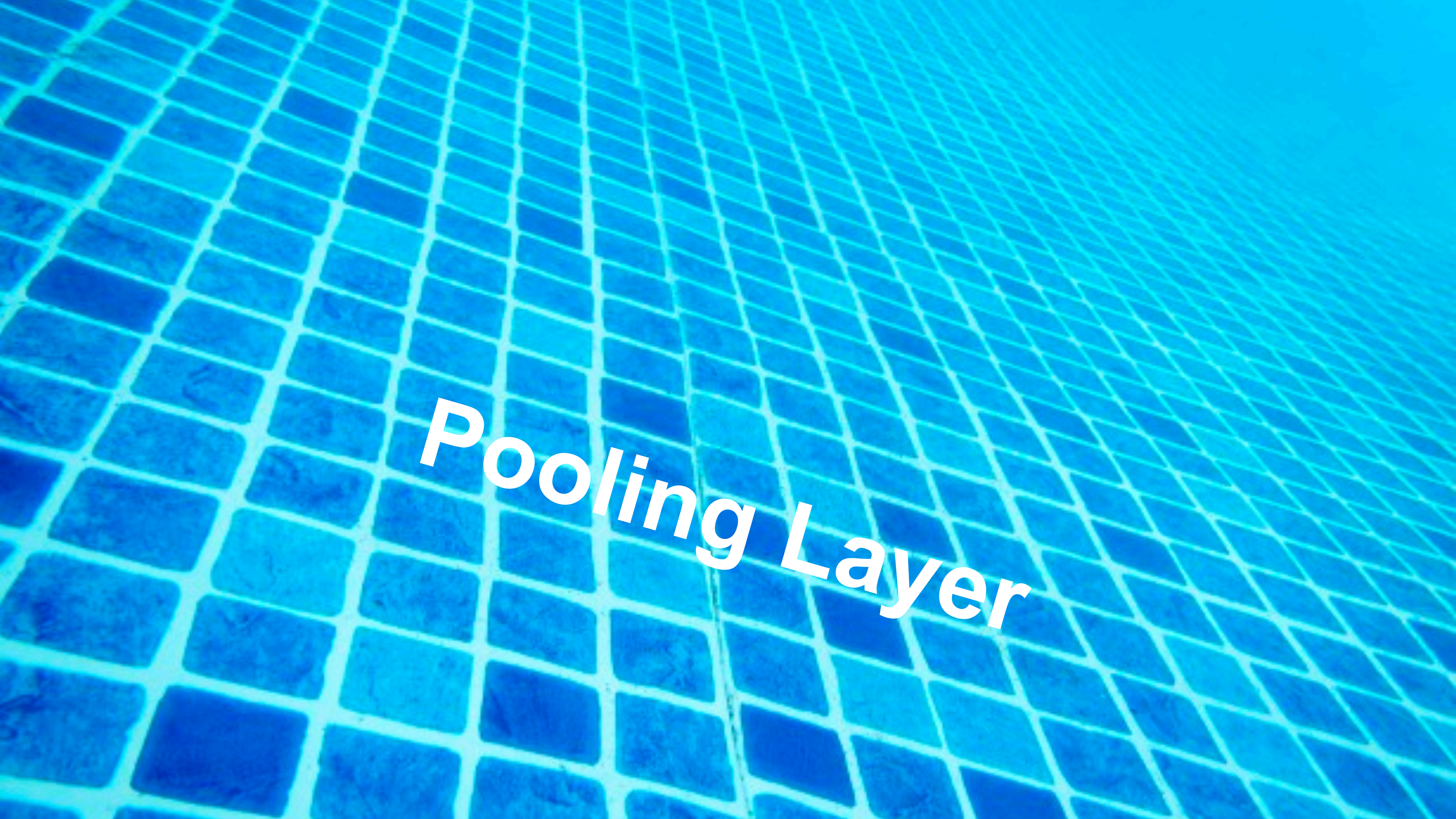
- Complexity (number of floating point operations FLOP)

$$c_i = c_o = 100 \qquad k_h = h_w = 5 \qquad m_h = m_w = 64$$

$$O(c_i c_o k_h k_w m_h m_w)$$

1GFLOP

- 10 layers, 1M examples: 10PF
(CPU: 0.15 TF = 18h, GPU: 12 TF = 14min)



Pooling Layer

Pooling

- Convolution is sensitive to position
  - Detect vertical edges

0 output with
1 pixel shift

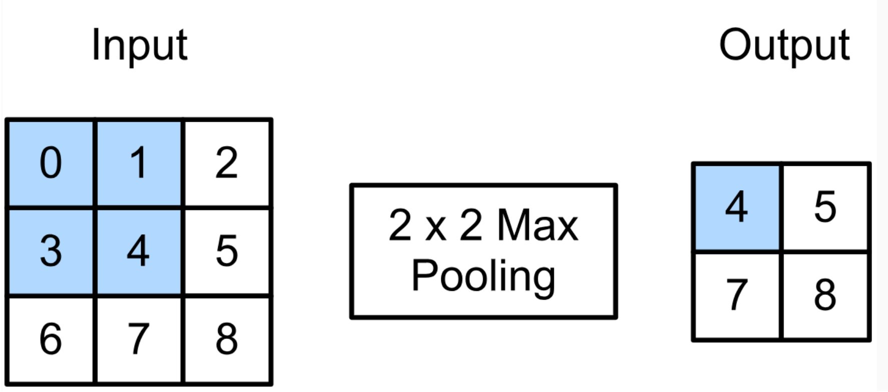
X
```
[[1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
```

Y
```
[[ 0.   1.   0.   0.
 [ 0.   1.   0.   0.
 [ 0.   1.   0.   0.
 [ 0.   1.   0.   0.
```

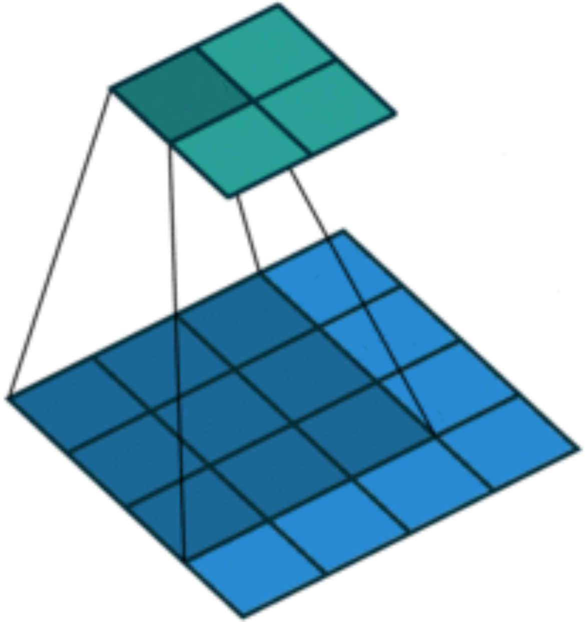
- We need some degree of invariance to translation
  - Lighting, object positions, scales, appearance vary among images

2-D Max Pooling

- Returns the maximal value in the sliding window

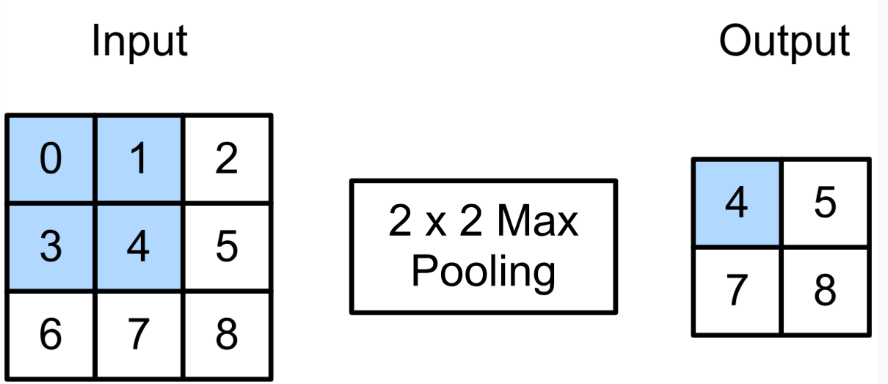


$$\max(0,1,3,4) = 4$$

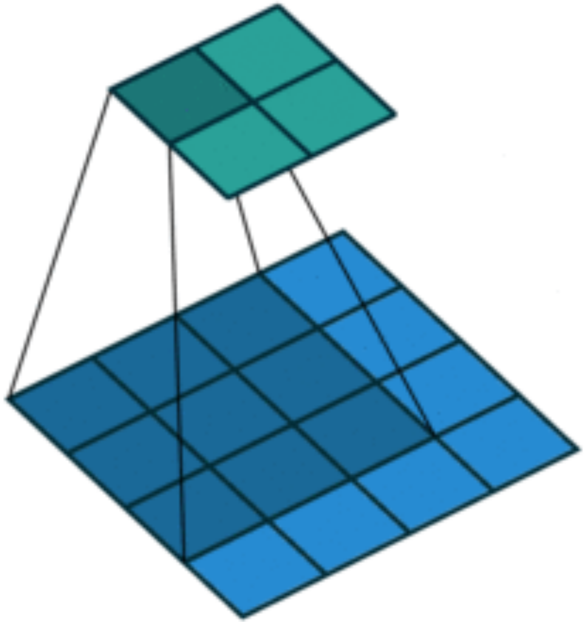


2-D Max Pooling

- Returns the maximal value in the sliding window



$$\max(0,1,3,4) = 4$$



2-D Max Pooling

- Returns the maximal value in the sliding window

Vertical edge detection

```
[[1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
```

Conv output

```
[[ 0.   1.   0.   0.
 [ 0.   1.   0.   0.
 [ 0.   1.   0.   0.
 [ 0.   1.   0.   0.
```

2 x 2 max pooling

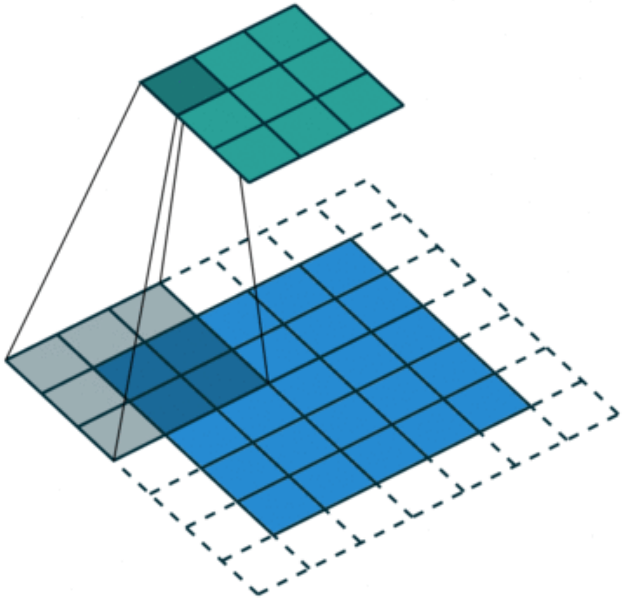
```
[[  1.   1.   1.   0.
 [  1.   1.   1.   0.
 [  1.   1.   1.   0.
 [  1.   1.   1.   0.
```

Tolerant to 1 pixel shift

Padding, Stride, and Multiple Channels

- Pooling layers have similar padding and stride as convolutional layers
- No learnable parameters
- Apply pooling for each input channel to obtain the corresponding output channel

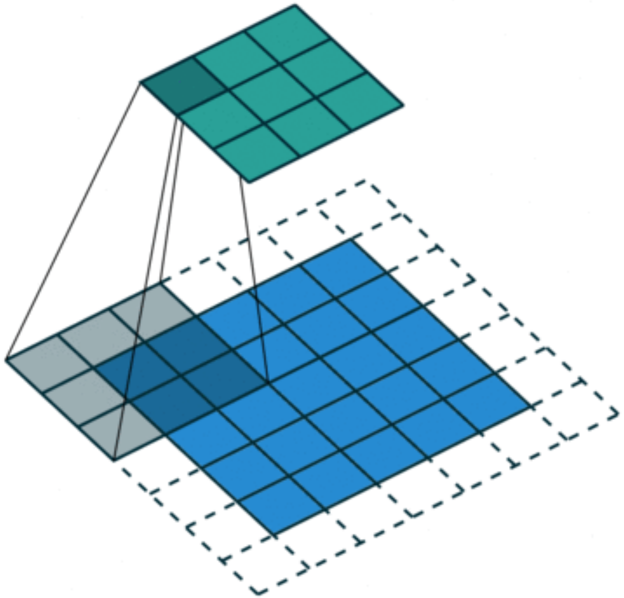
#output channels = #input channels



Padding, Stride, and Multiple Channels

- Pooling layers have similar padding and stride as convolutional layers
- No learnable parameters
- Apply pooling for each input channel to obtain the corresponding output channel

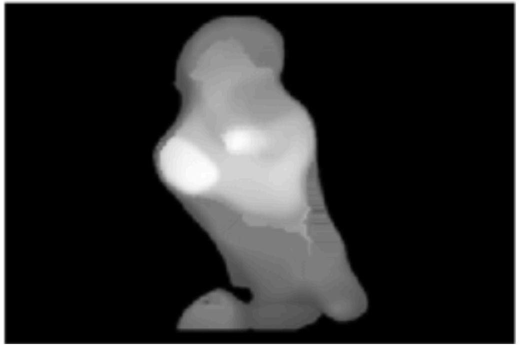
#output channels = #input channels



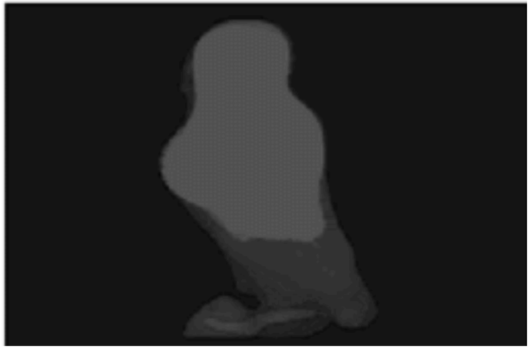
Average Pooling

- Max pooling: the strongest pattern signal in a window
- Average pooling: replace max with mean in max pooling
  - The average signal strength in a window

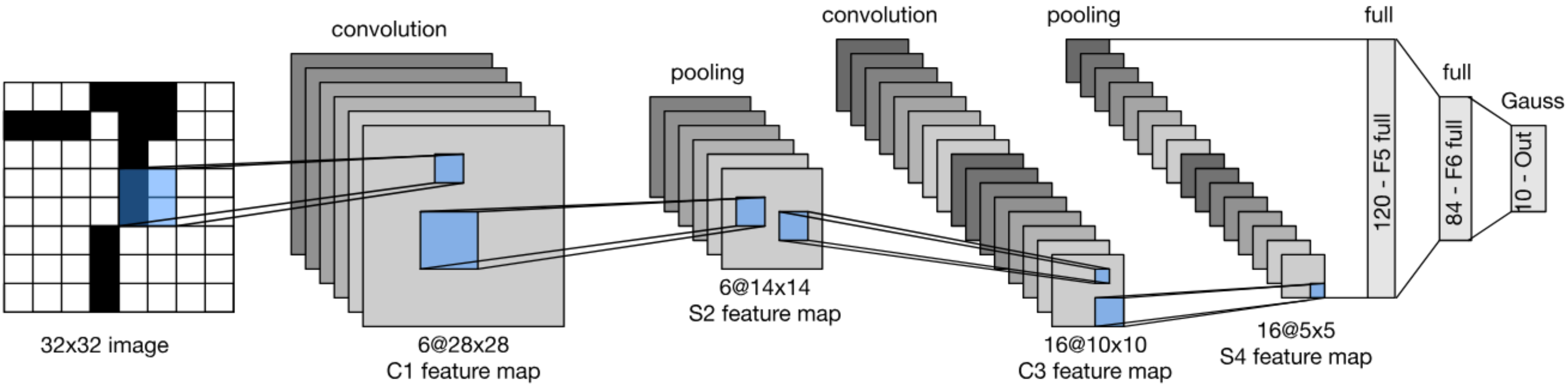
Max pooling



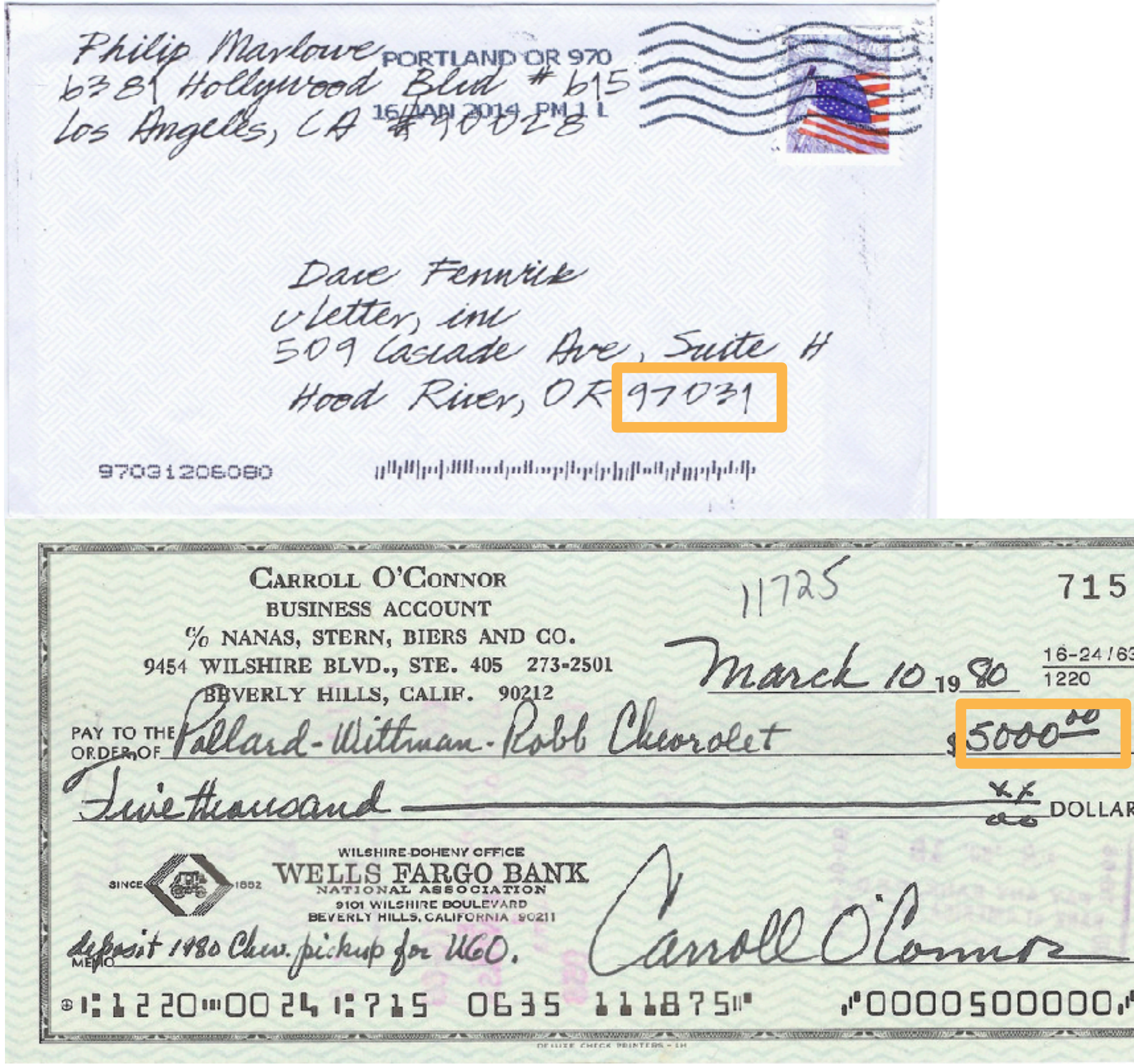
Average pooling



LeNet

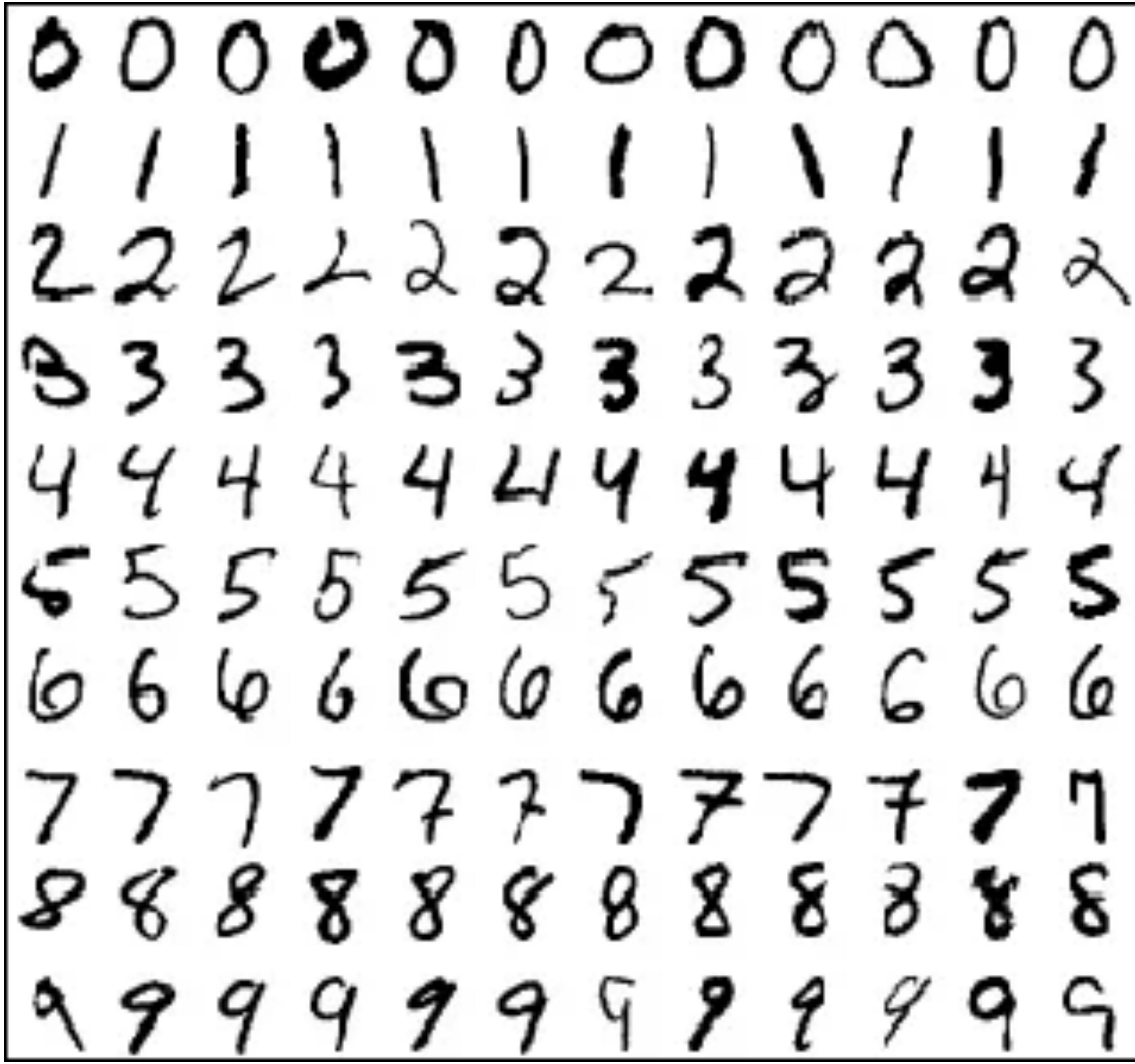

convolution
pooling
convolution
pooling
full
full
Gauss
120 - F5 full
84 - F6 full
10 - Out
32x32 image
6@28x28
C1 feature map
6@14x14
S2 feature map
16@10x10
C3 feature map
16@5x5
S4 feature map

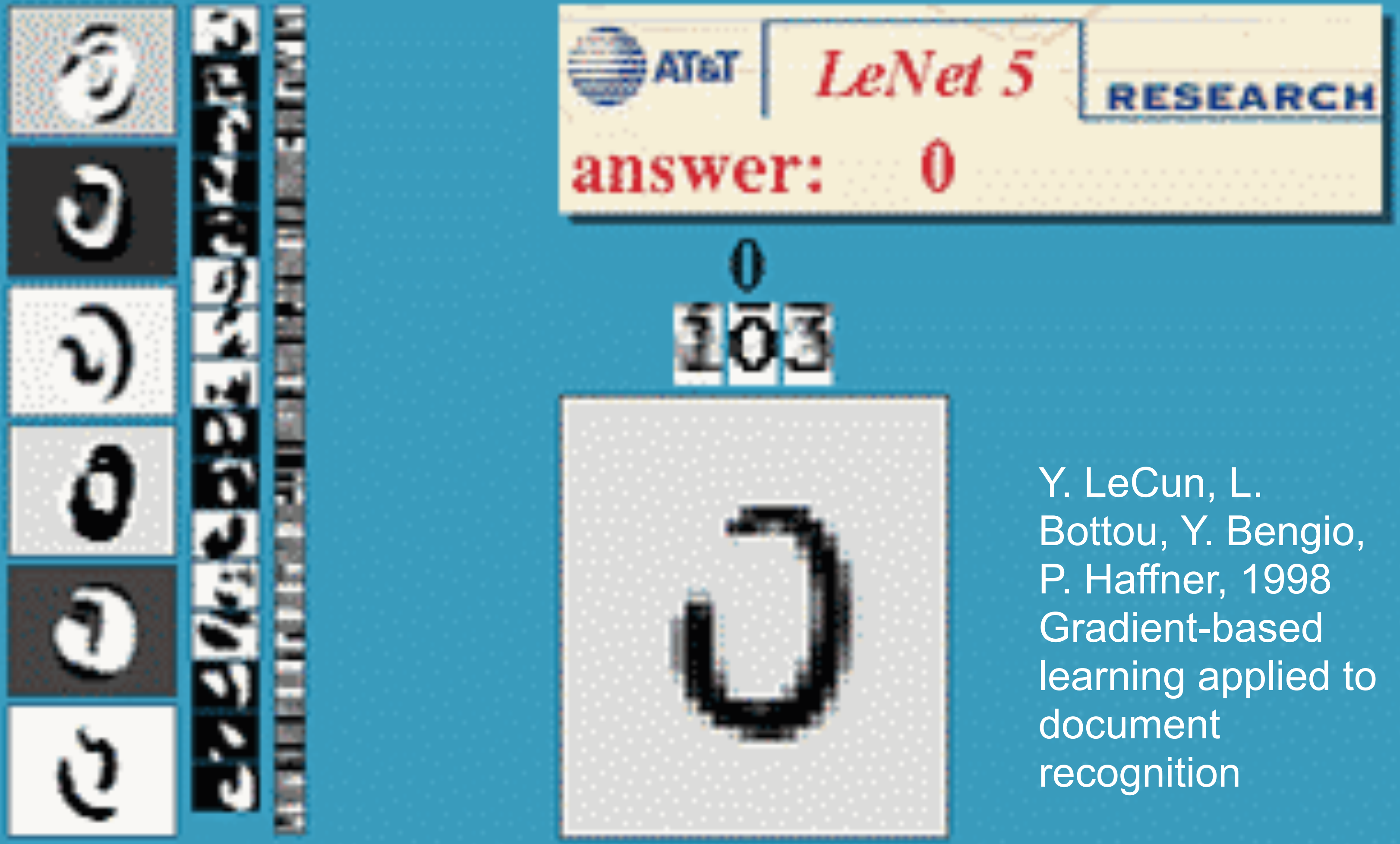
Handwritten Digit Recognition



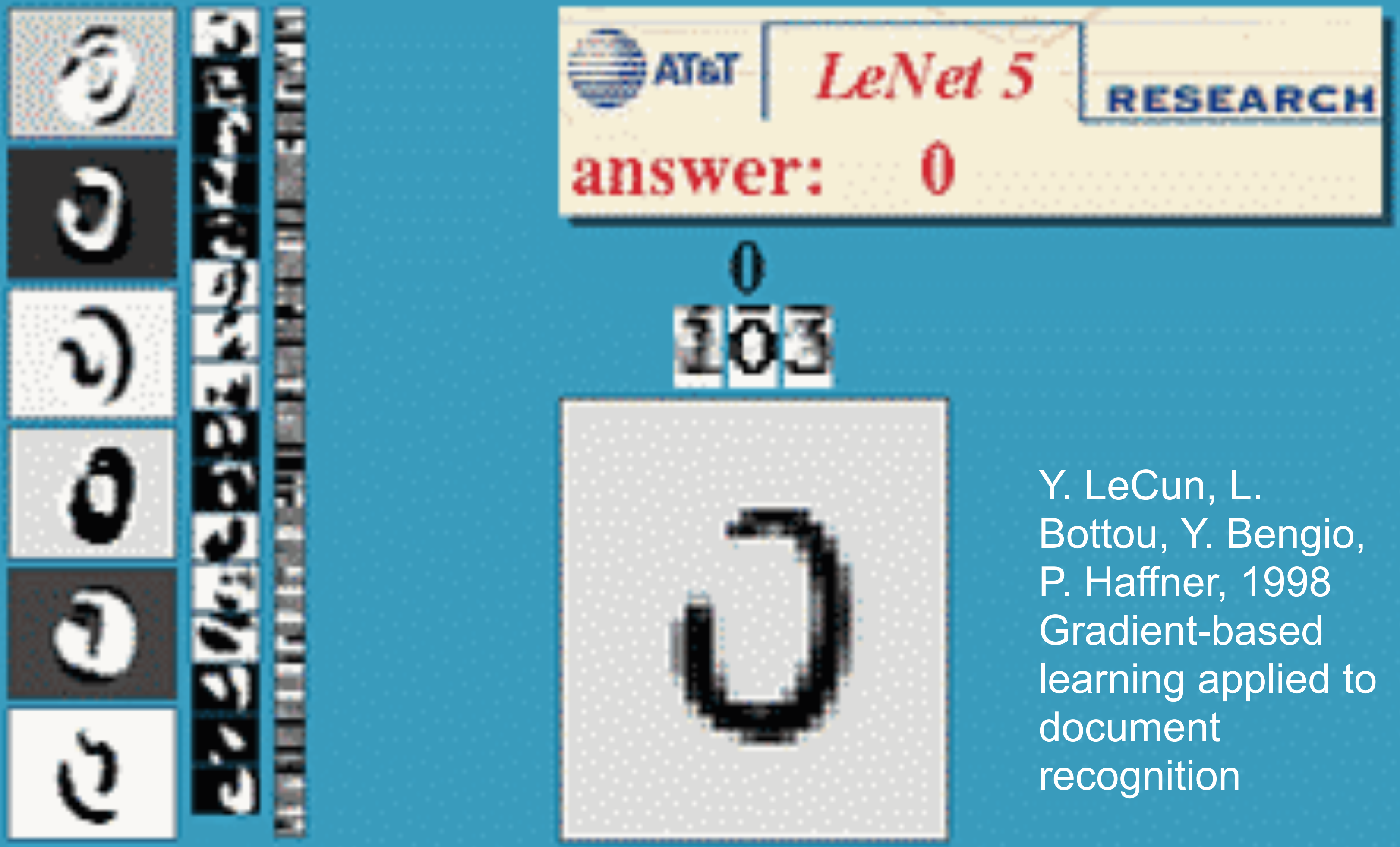
MNIST

- Centered and scaled
- 50,000 training data
- 10,000 test data
- 28 x 28 images
- 10 classes



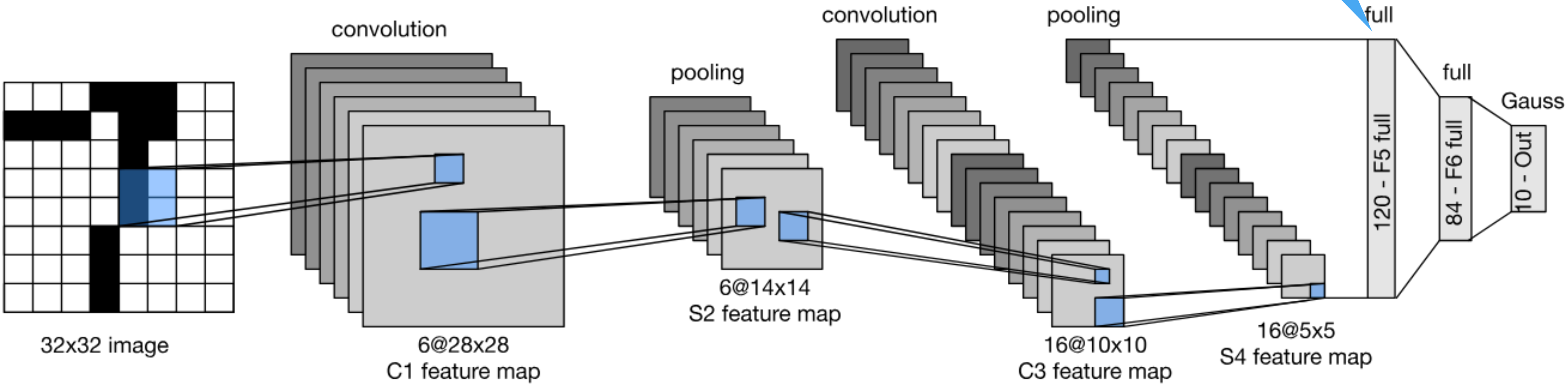


Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998 Gradient-based learning applied to document recognition



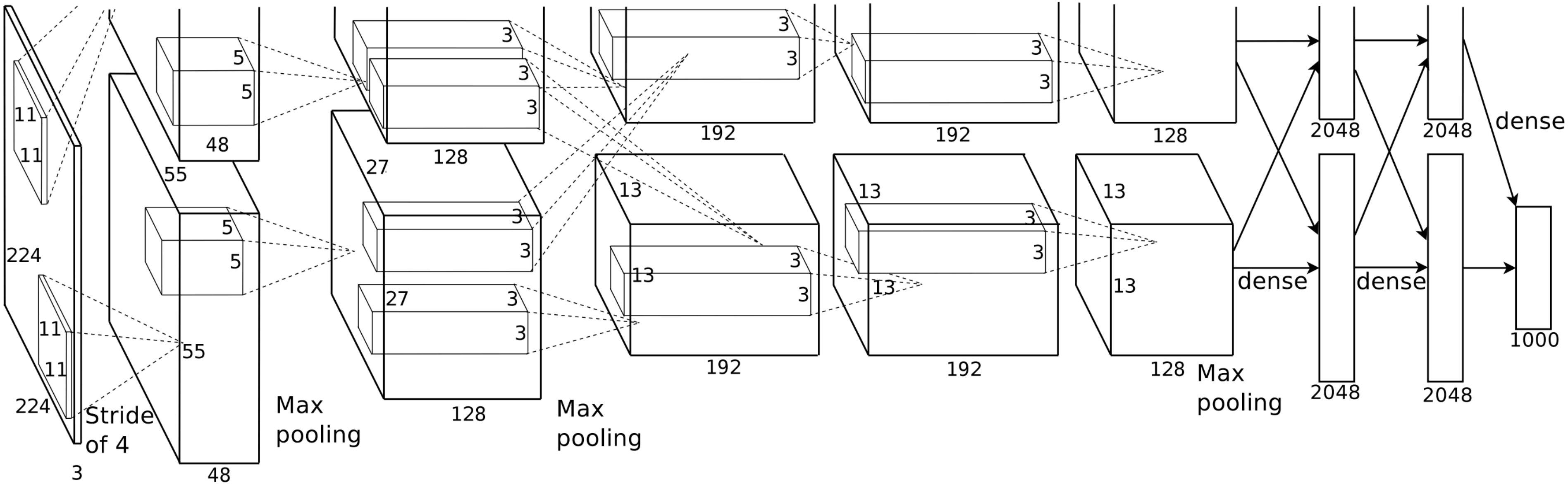
Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998 Gradient-based learning applied to document recognition

Expensive if we have many outputs



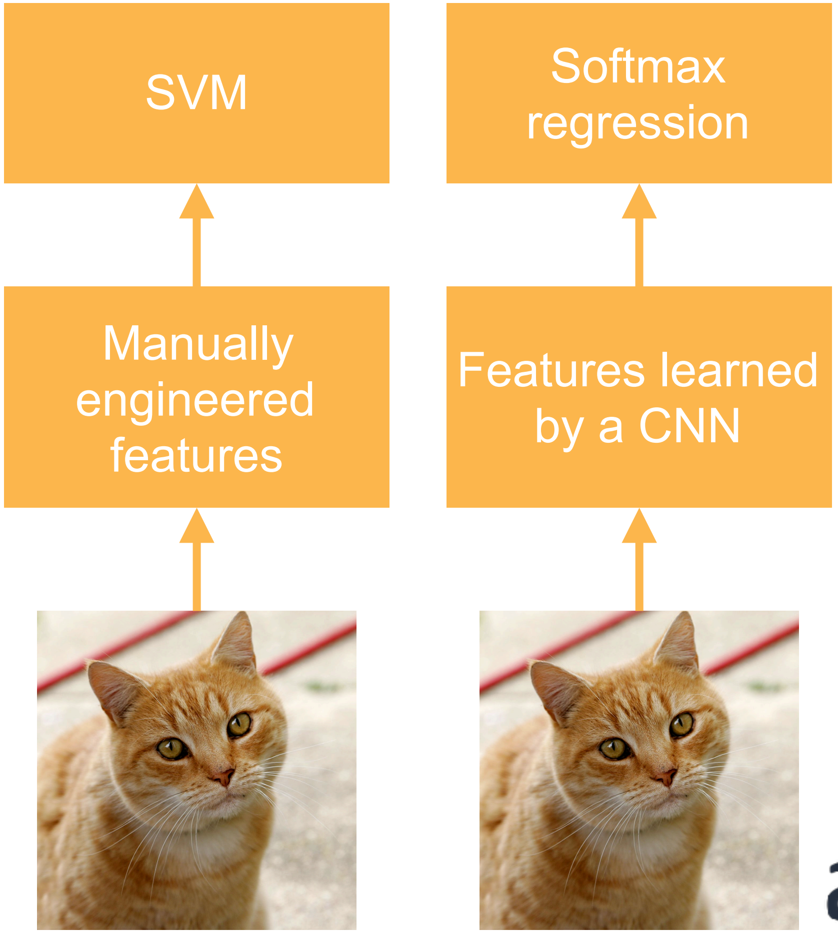
LeNet Notebook

AlexNet


224
224
11
11
11
11
3
Stride of 4
55
55
48
48
5
5
5
5
Max pooling
27
27
128
128
3
3
3
3
3
3
3
3
Max pooling
13
13
13
13
13
13
13
192
192
192
192
128
128
Max pooling
dense
dense
dense
2048
2048
2048
2048
1000

AlexNet

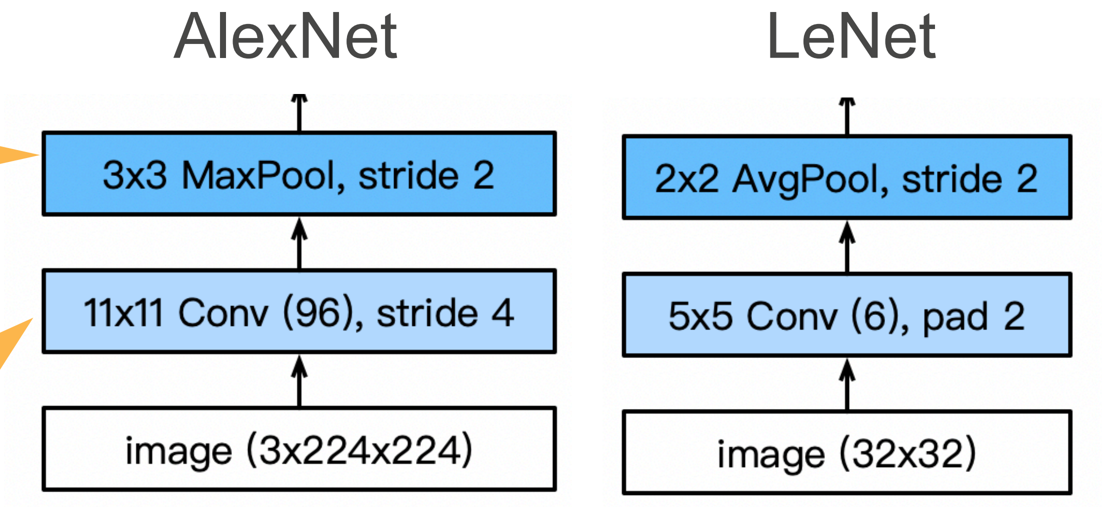
- AlexNet won ImageNet competition in 2012
- Deeper and bigger LeNet
- Key modifications
  - Dropout (regularization)
  - ReLu (training)
  - MaxPooling
- Paradigm shift for computer vision



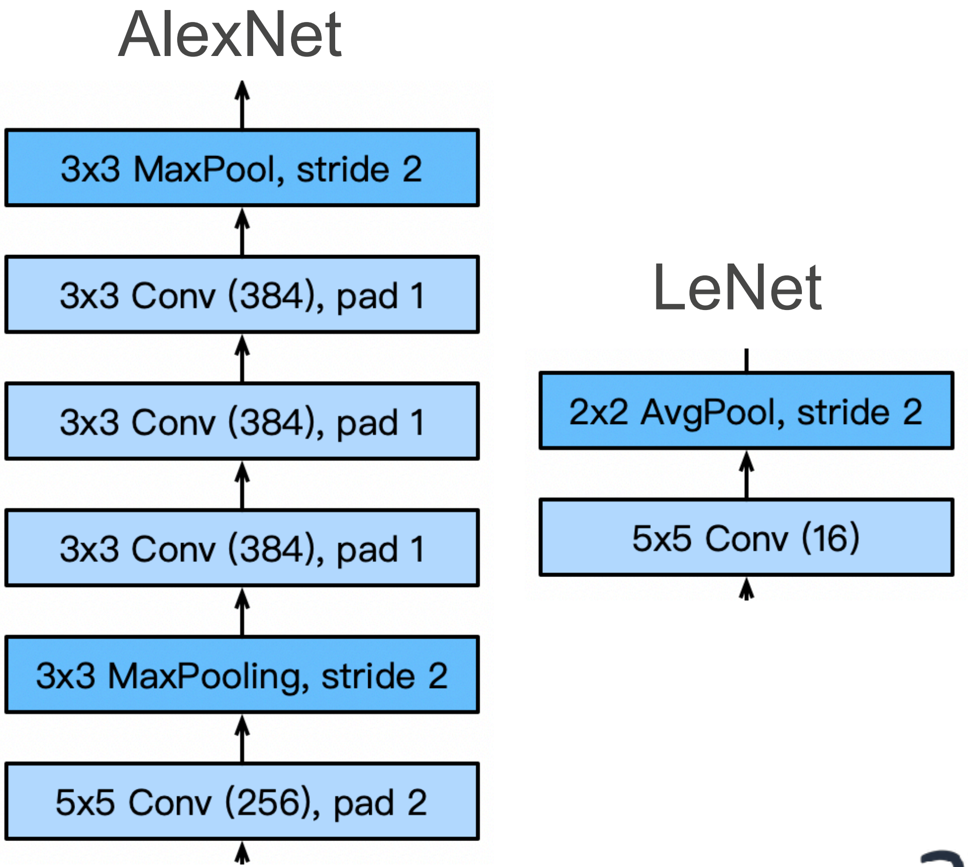
AlexNet Architecture

Larger pool size, change to max pooling

Larger kernel size, stride because of the increased image size, and more output channels.



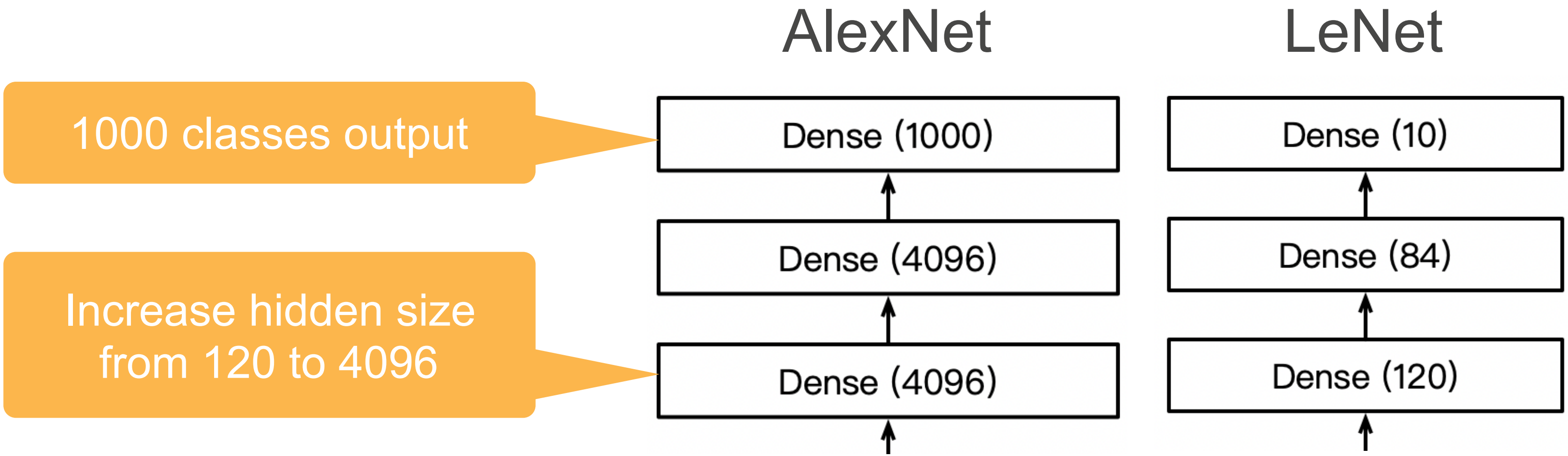
AlexNet Architecture



3 additional convolutional layers

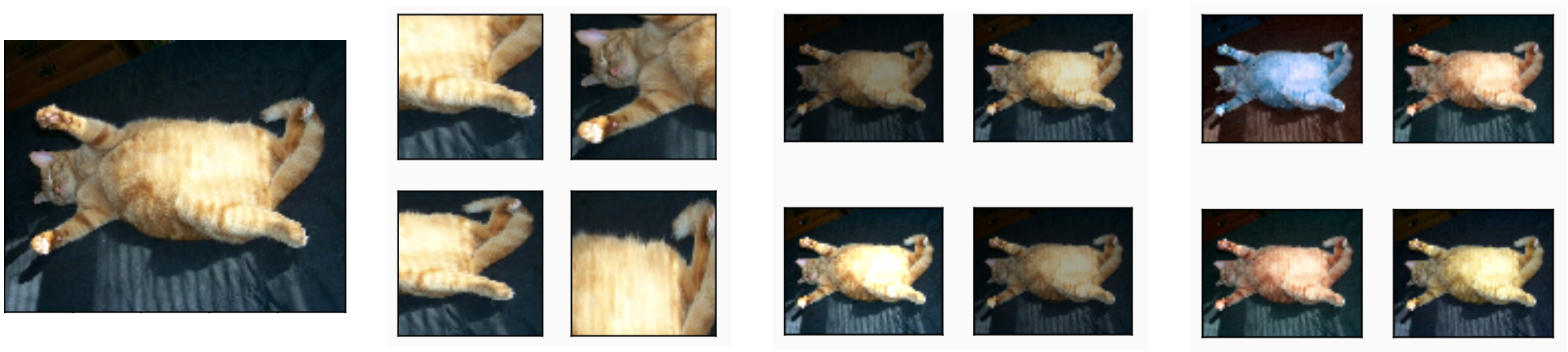
More output channels.

AlexNet Architecture



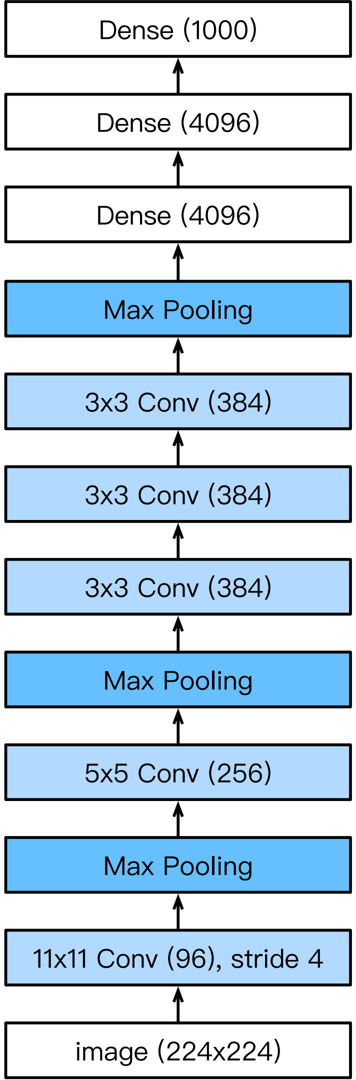
More Tricks

- Change activation function from sigmoid to ReLu (no more vanishing gradient)
- Add a dropout layer after two hidden dense layers (better robustness / regularization)
- Data augmentation



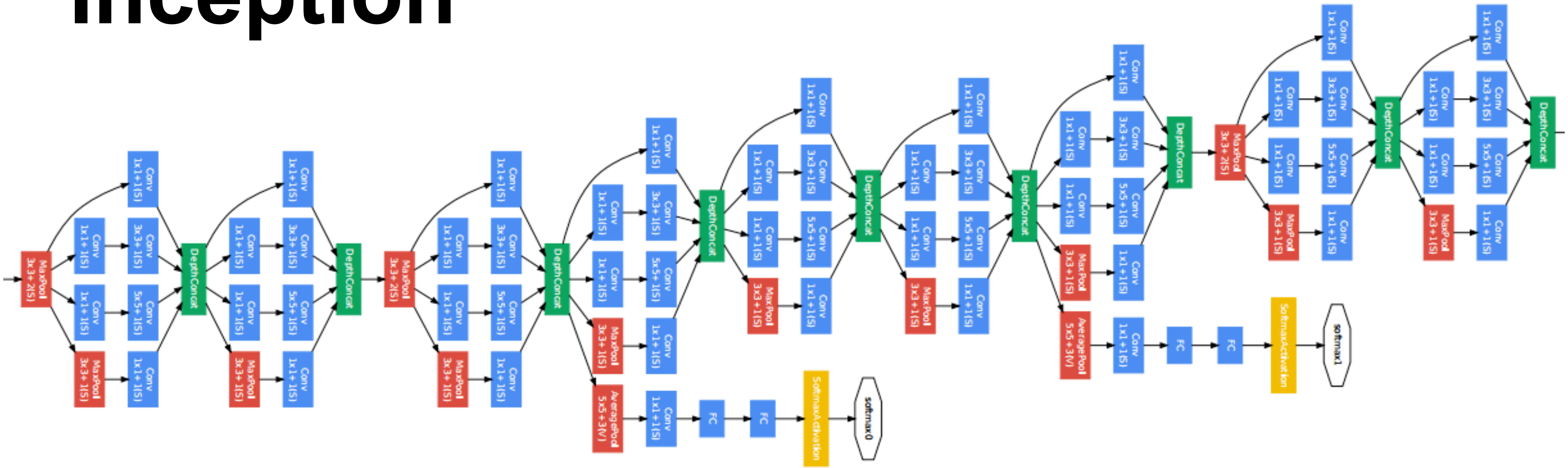
Complexity

	#parameters		FLOP	
	AlexNet	LeNet	AlexNet	LeNet
Conv1	35K	150	101M	1.2M
Conv2	614K	2.4K	415M	2.4M
Conv3-5	3M		445M	
Dense1	26M	0.48M	26M	0.48M
Dense2	16M	0.1M	16M	0.1M
Total	46M	0.6M	1G	4M
Increase	11x	1x	250x	1x

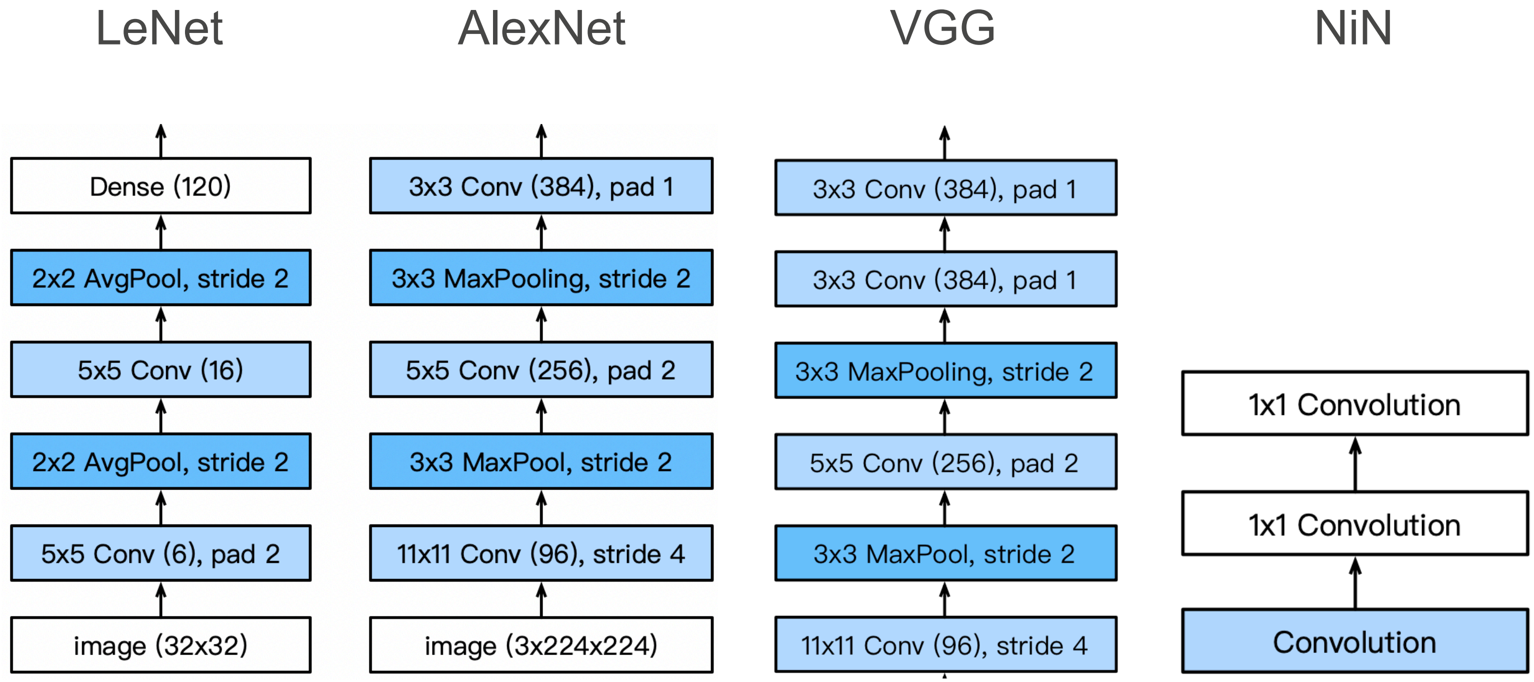


AlexNet Notebook

Inception


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

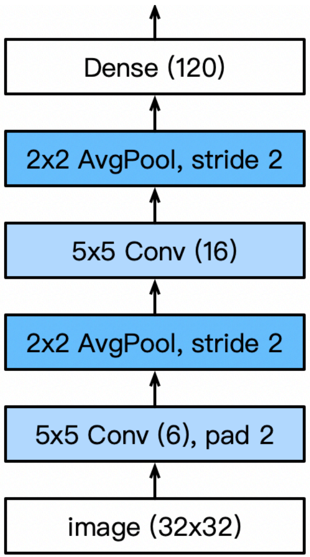
Picking the best convolution …

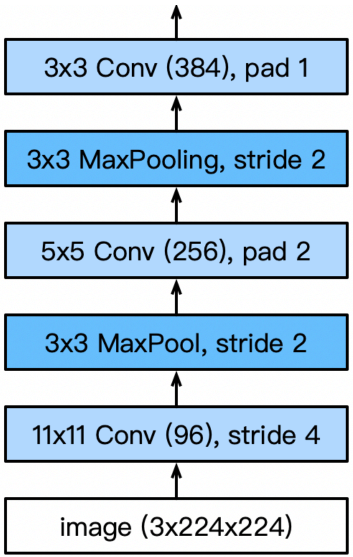


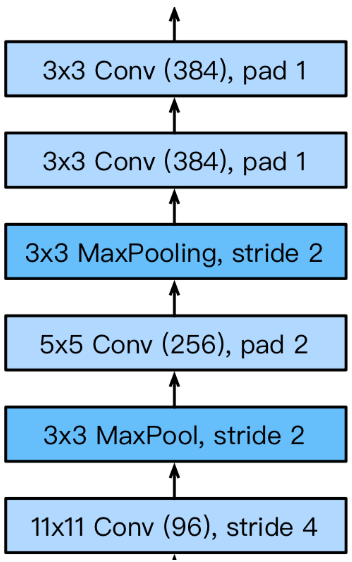
Picking the best convolution …

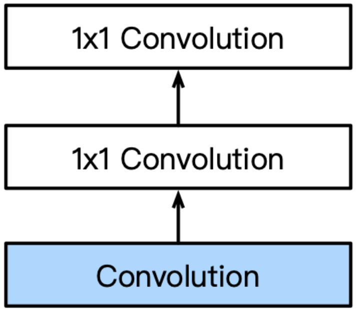
1x1 | 3x3 | 5x5 | Max pooling | Multiple 1x1

LeNet | AlexNet | VGG | NiN









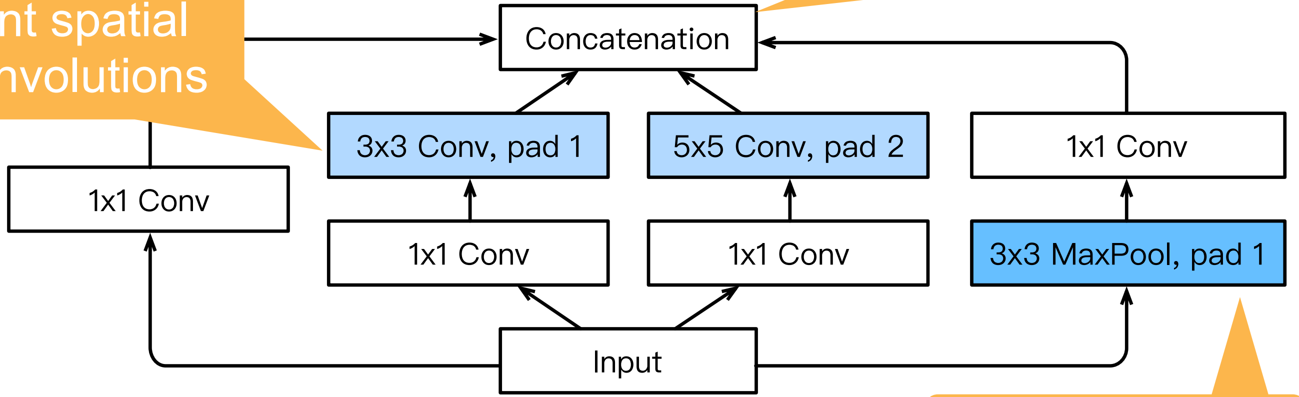
Why choose? Just pick them all.

Inception Blocks

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

Same width/ height as input

Extract with different spatial size convolutions



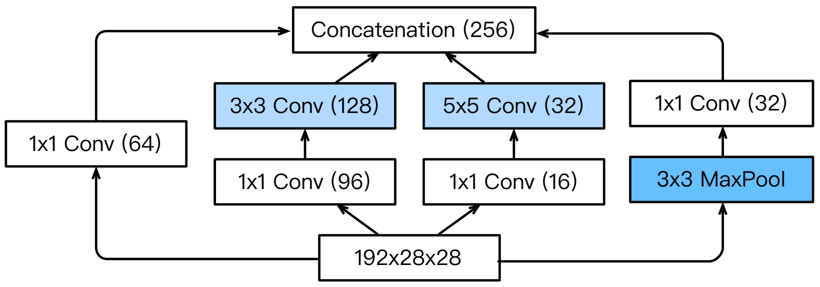
Extract spatial info with pooling

Inception Blocks

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

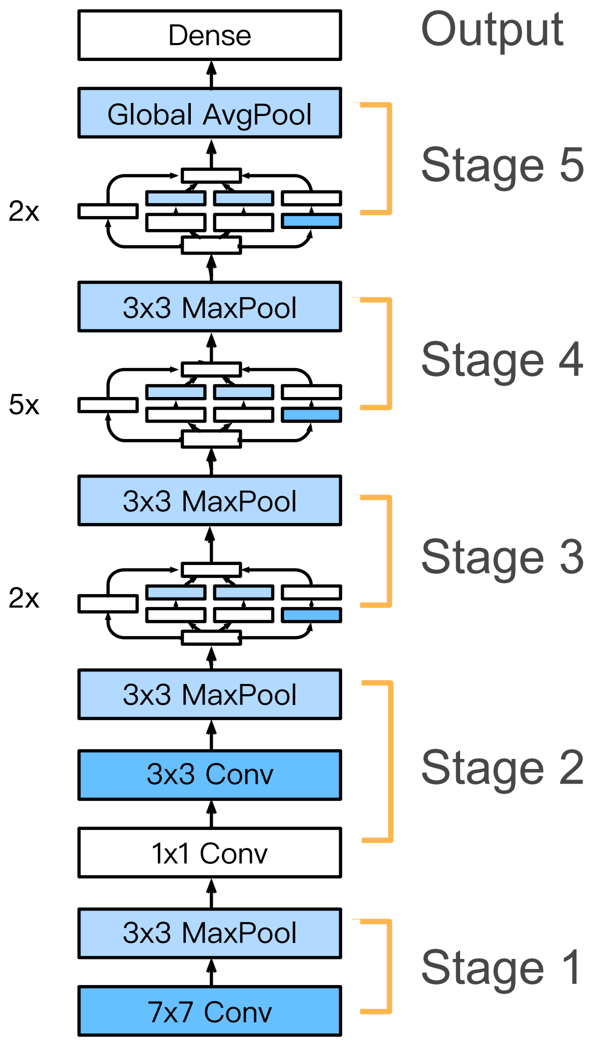
- Mix of different functions (powerful function class)
- Memory and compute efficiency (good generalization)

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



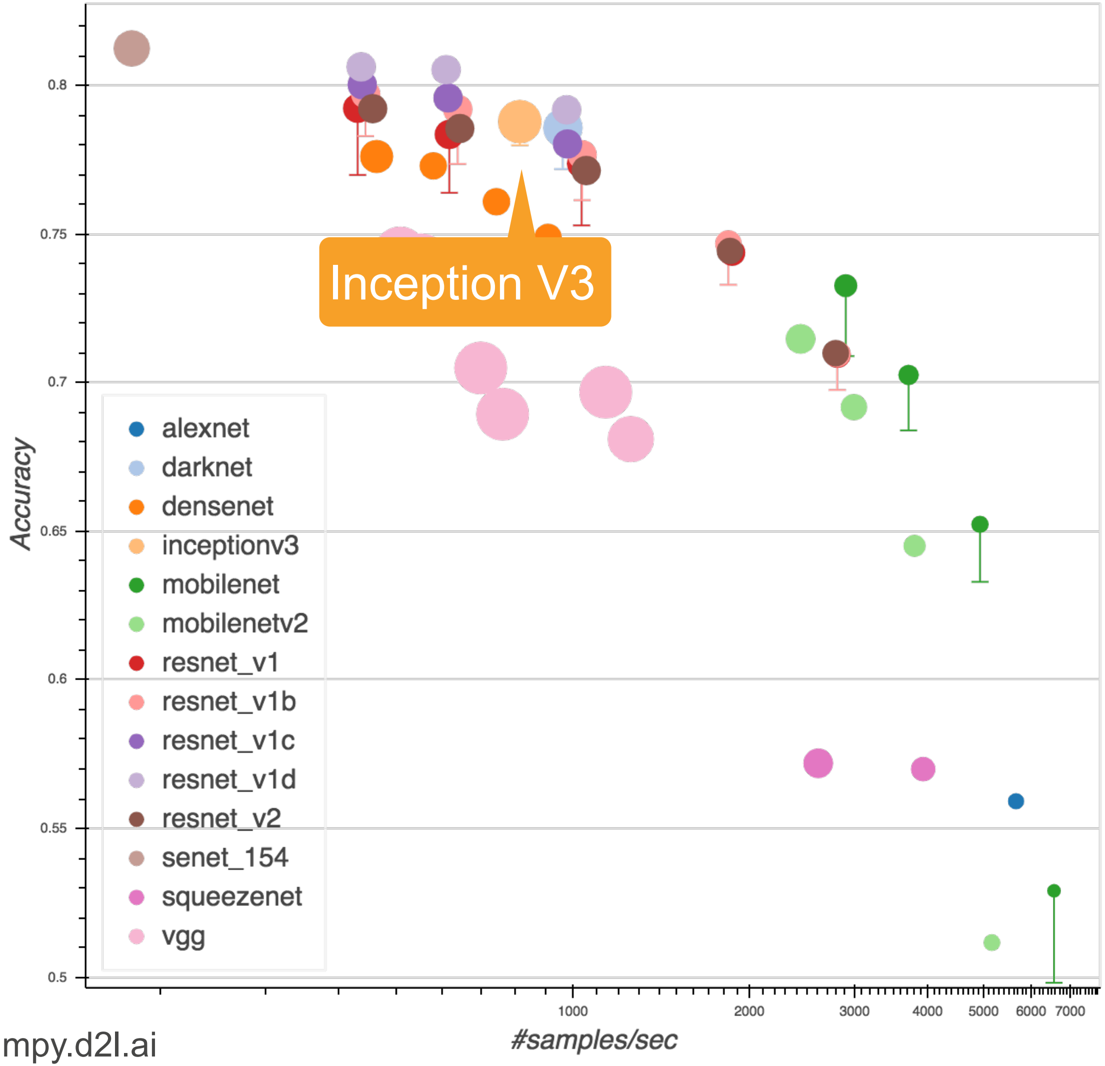
GoogLeNet

- 5 stages with 9 inceptions blocks



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)



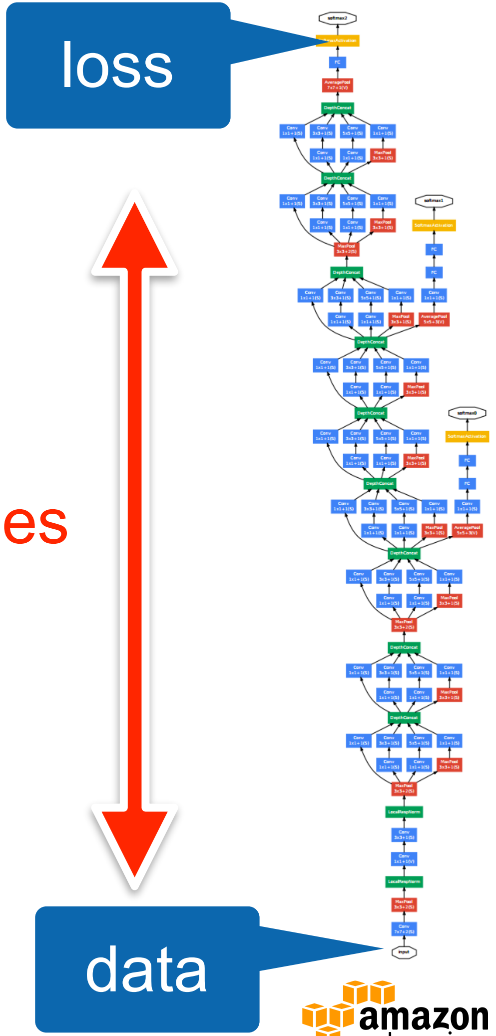
GluonCV Model Zoo

https://gluon-cv.mxnet.io/model_zoo/classification.html


SMALL BATCH NORM
• PACIFICA, CALIFORNIA •
Batch # 22716-06
Hand Made All Natural
BLACKBERRY JAM

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?



amazon web services

Batch Normalization

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

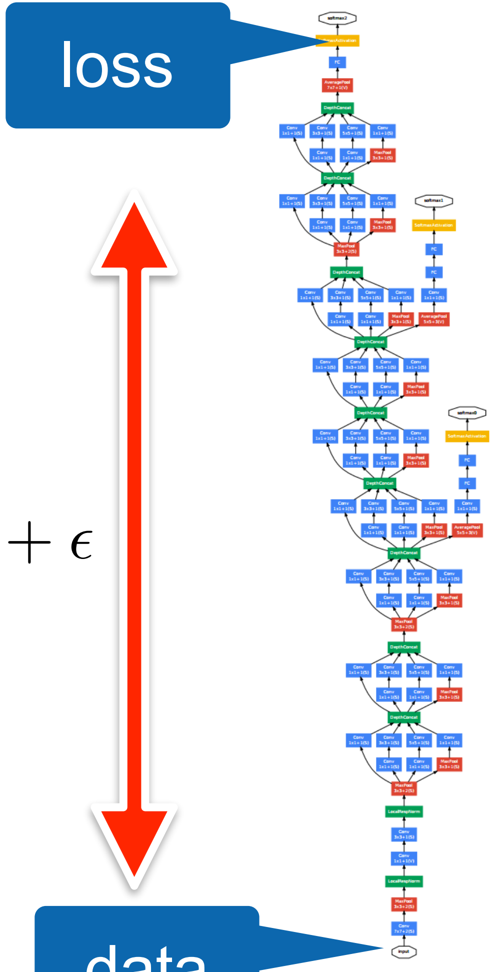
$$\mu_B = \frac{1}{|B|}\sum_{i \in B} x_i \text{ and } \sigma_B^2 = \frac{1}{|B|}\sum_{i \in B}(x_i - \mu_B)^2 + \epsilon$$

and adjust it separately

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

mean

variance



This was the original motivation …

What Batch Norms really do

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

Empirical mean

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

Empirical variance

  - 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

What Batch Norms really do

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

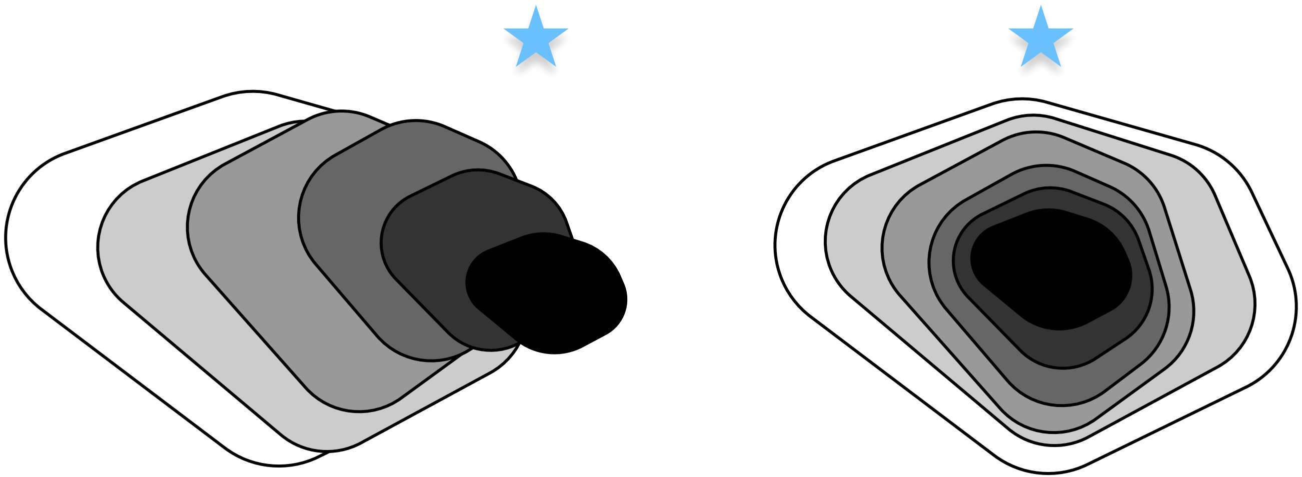
Random offset

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

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

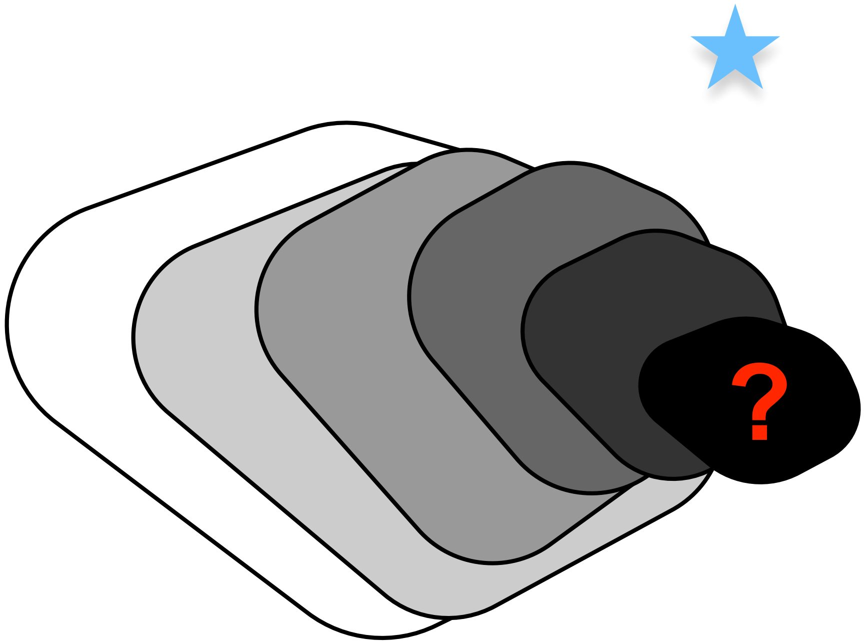
Residual Networks



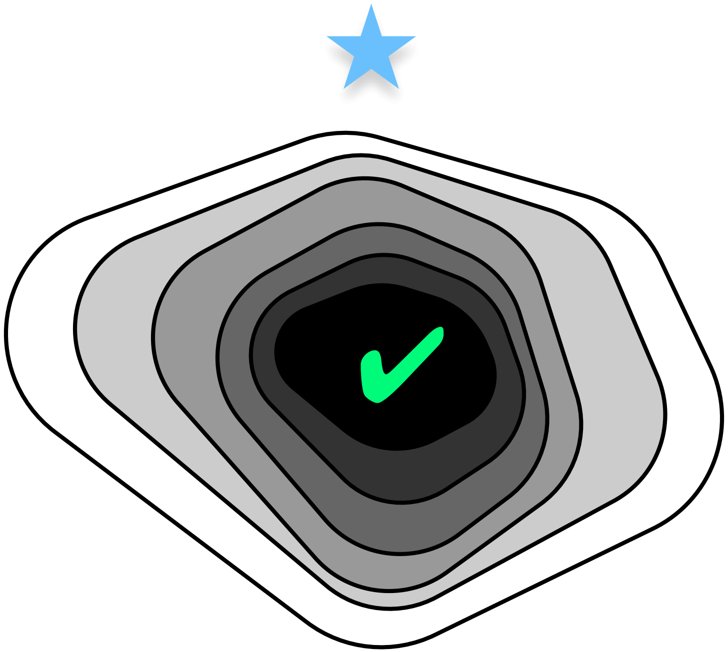
generic function classes

nested function classes

Does adding layers improve accuracy?



generic function classes

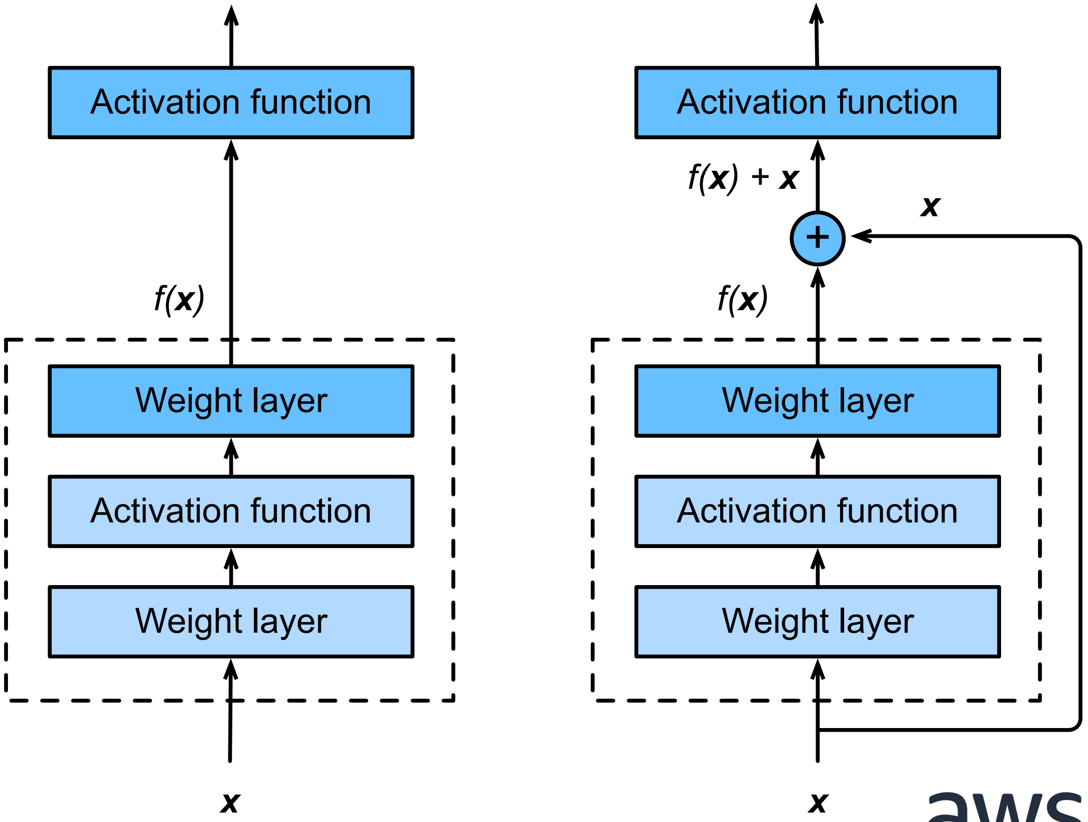


nested function classes

Residual Networks

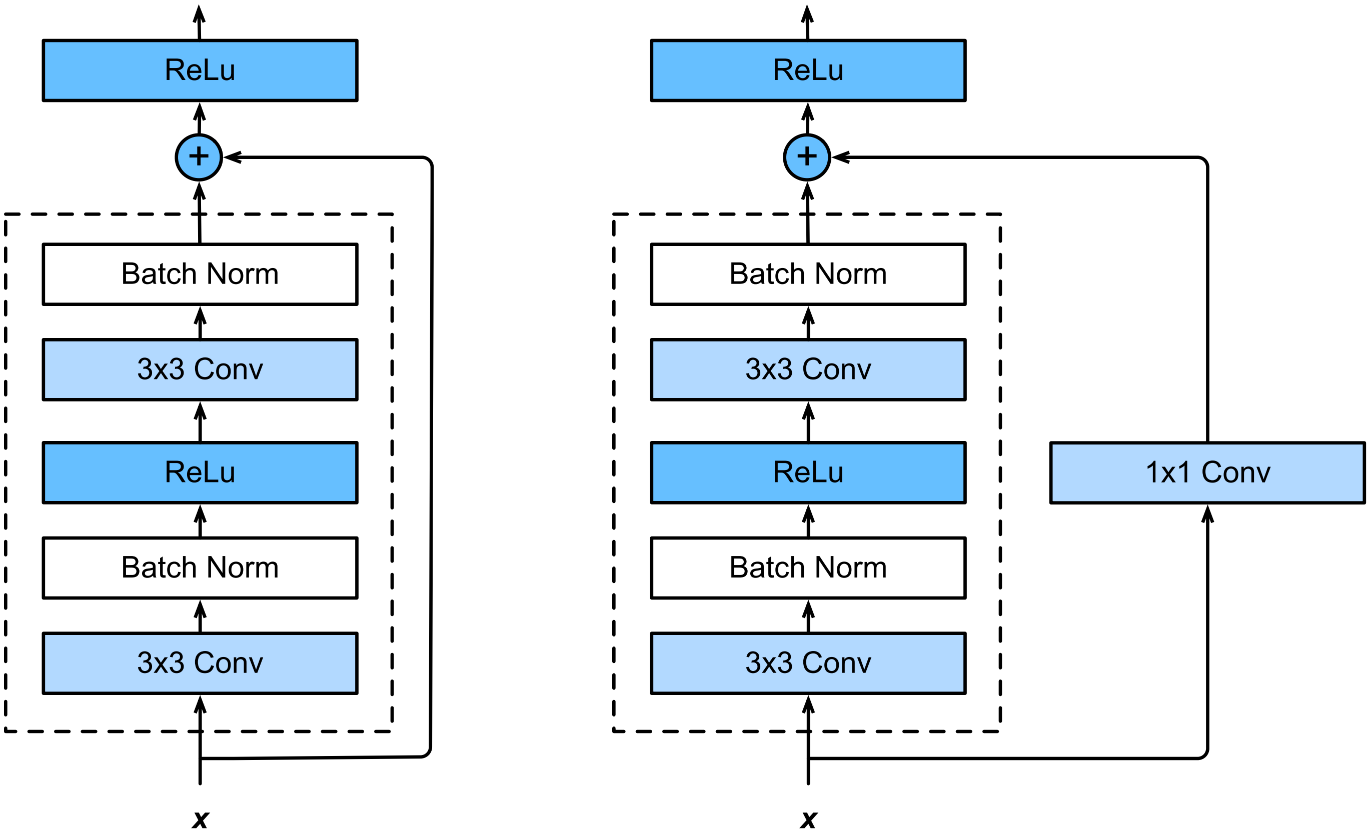
- Adding a layer **changes** function class
- We want to **add to** the function class
- ‘Taylor expansion' style parametrization

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

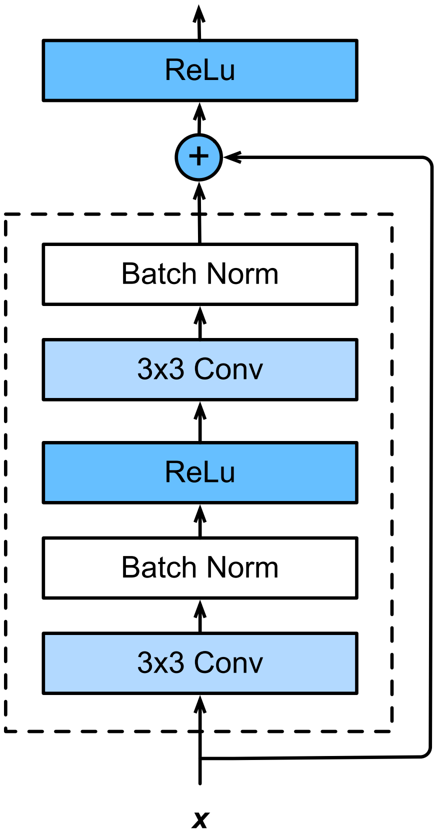


He et al., 2015

ResNet Block in detail

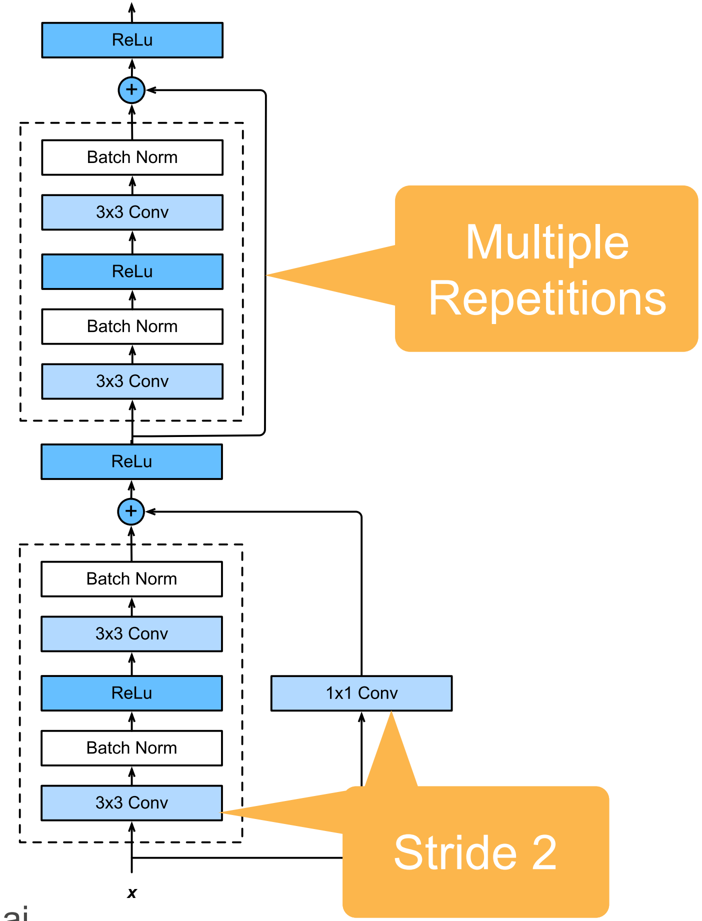


In code




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

ResNet Module



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

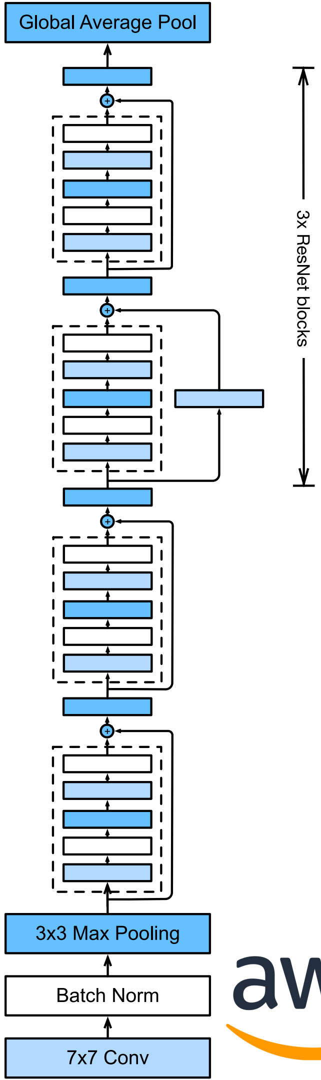
```
blk = nn.Sequential()
for i in range(num_residuals):
    if i == 0 and not first_block:
        blk.add(Residual(num_channels,
            use_1x1conv=True, strides=2))
    else:
        blk.add(Residual(num_channels))
```

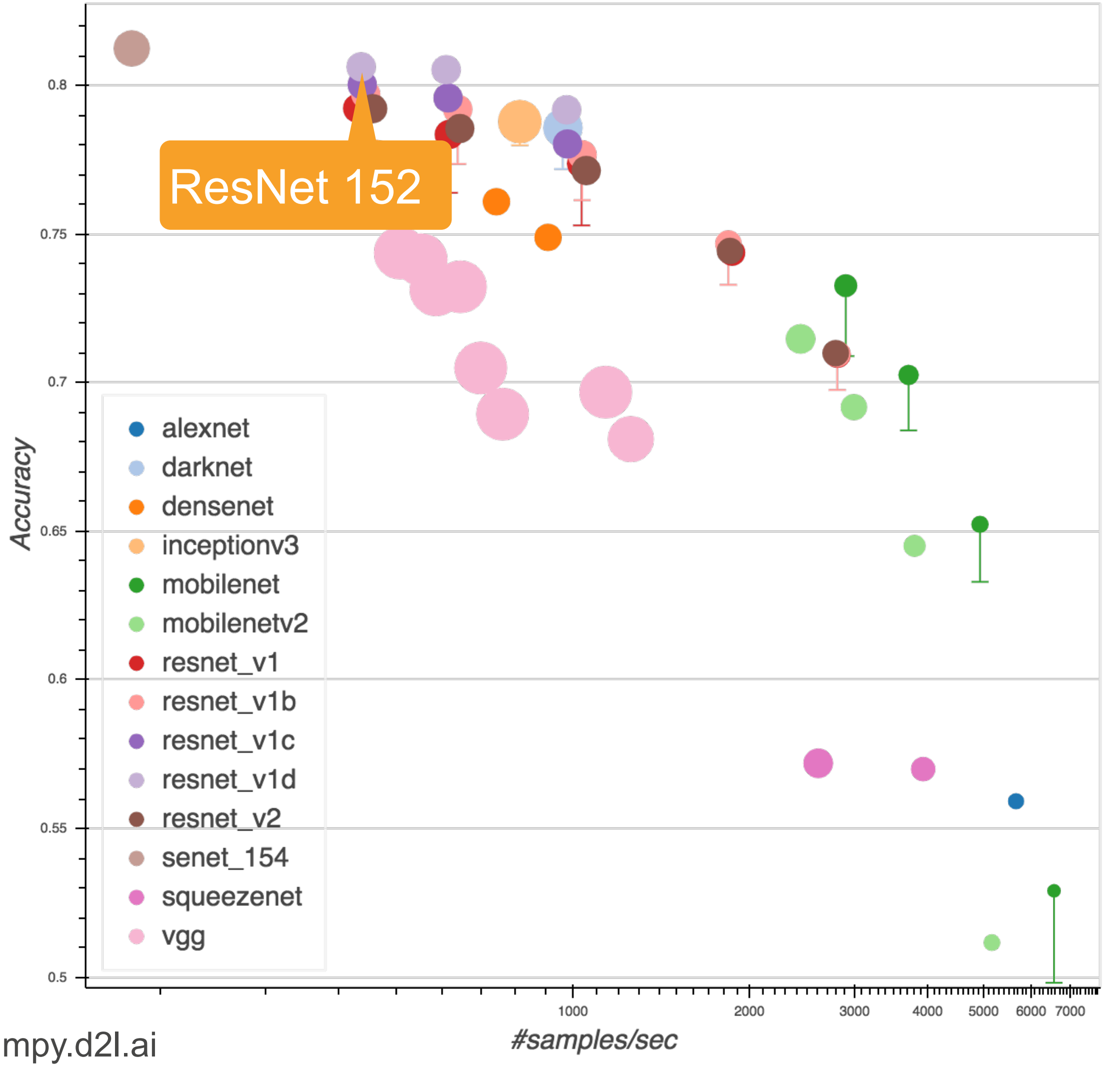
numpy.d2l.ai

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 …





GluonCV Model Zoo

https://gluon-cv.mxnet.io/model_zoo/classification.html

Jupyter Notebook

More Ideas

DenseNet (Huang et al., 2016)

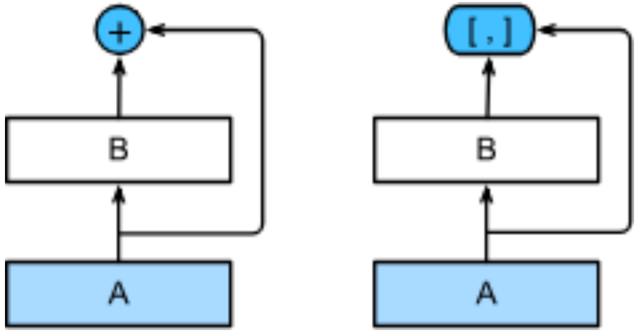
- ResNet combines x and f(x)
- DenseNet uses higher order 'Taylor series' expansion

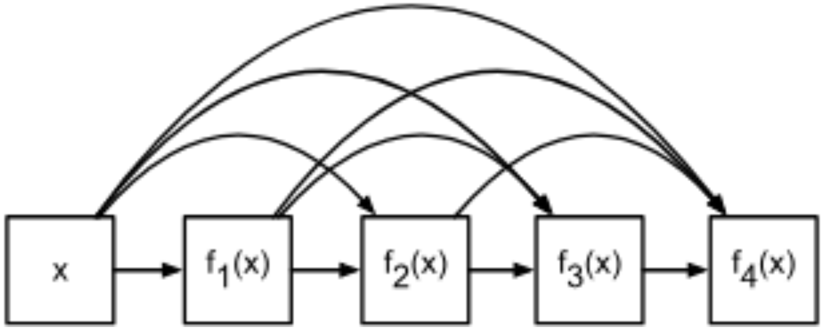
$$x_{i+1} = [x_i, f_i(x_i)]$$

$$x_1 = x$$

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

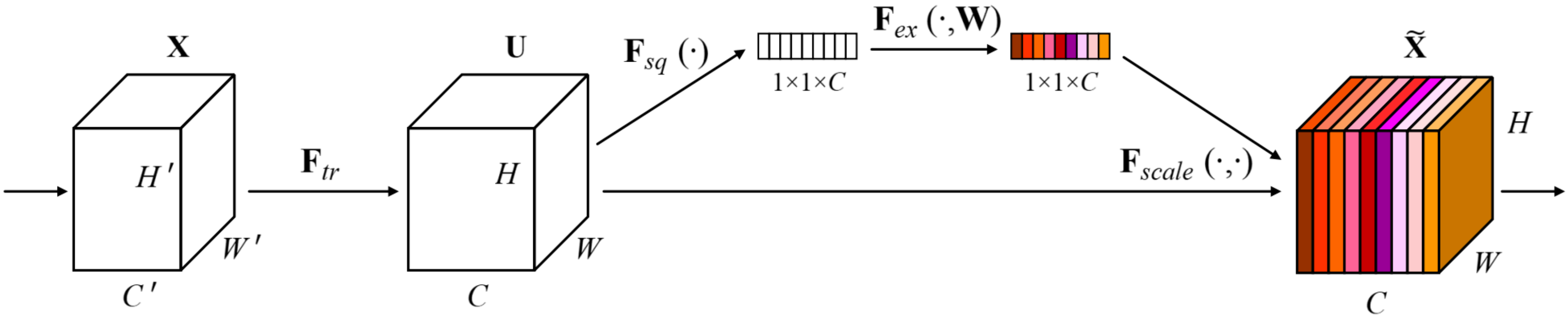
$$x_2 = [x, f_1(x), f_2([x, f_1(x)])]$$





- Occasionally need to reduce resolution (transition layer)

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

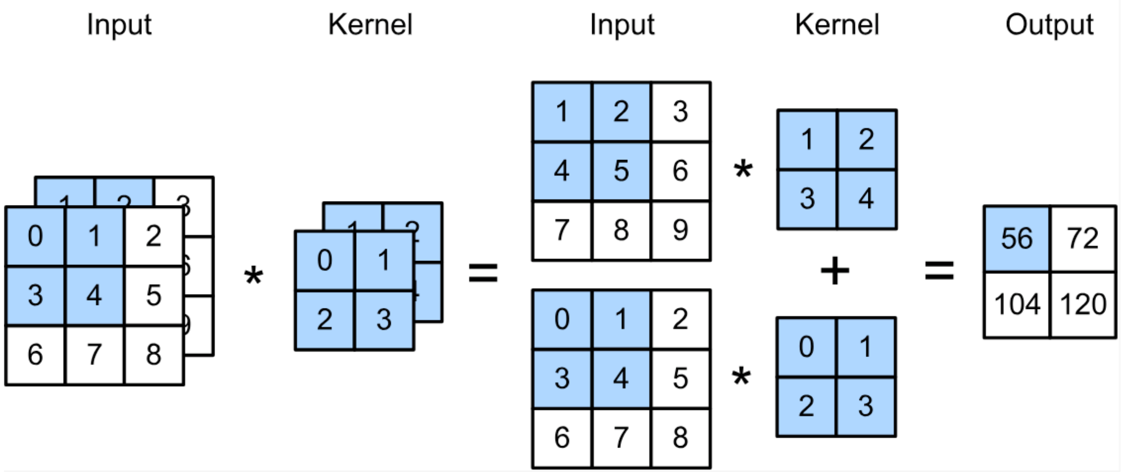


- Learn global weighting function per channel
- Allows for fast information transfer between pixels in different locations of the image

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

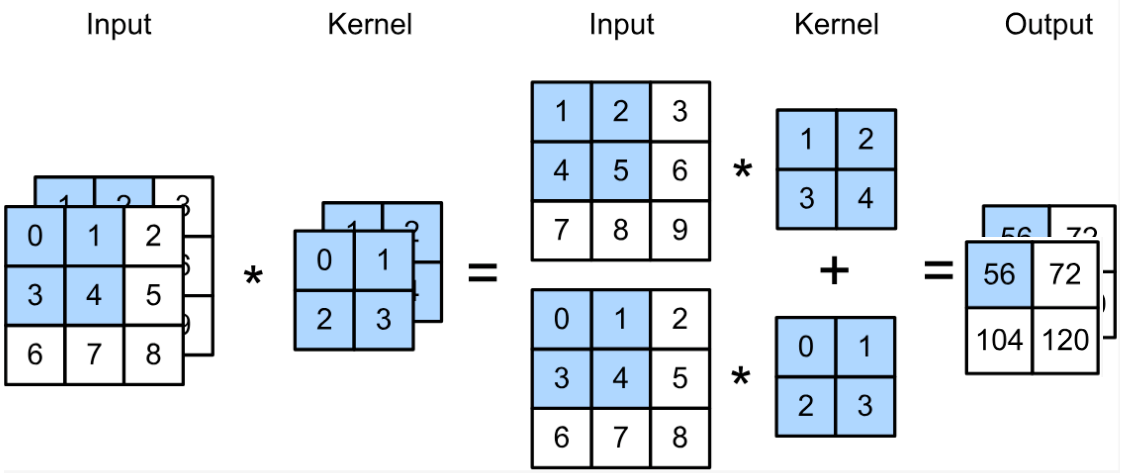
$$m_h \cdot m_w \cdot k_h \cdot k_w \cdot c$$



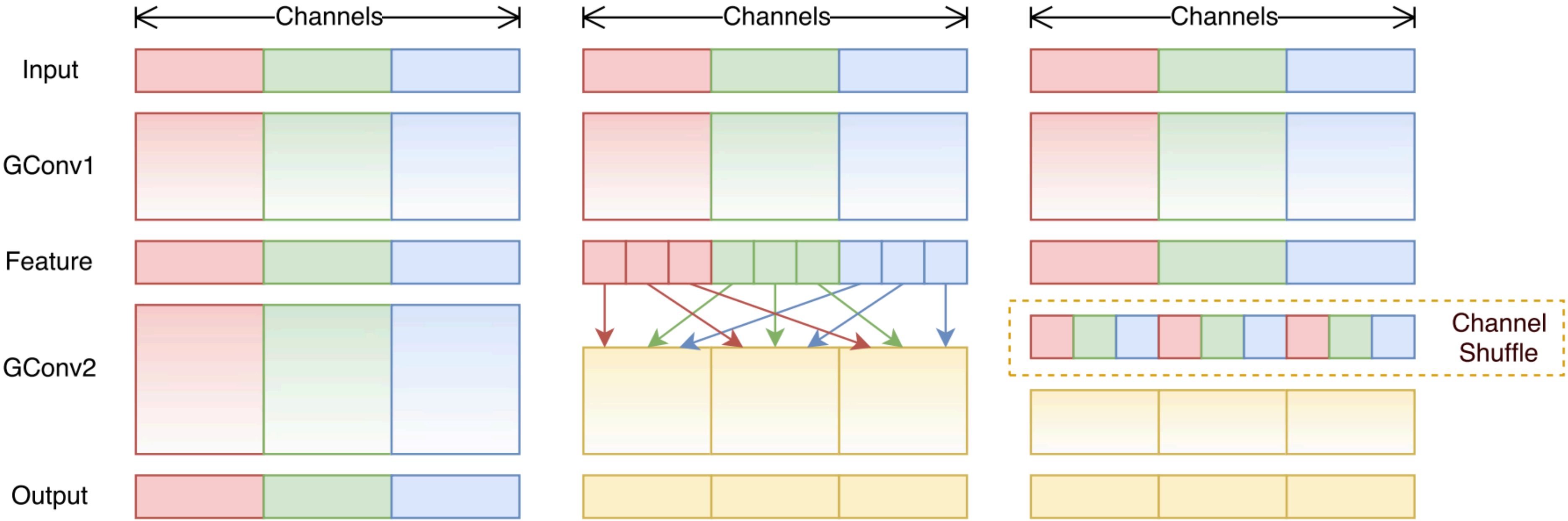
Separable Convolutions - all channels separate

- **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

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



ShuffleNet (Zhang et al., 2018)



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

Outline

- GPUs
- Convolutions
- Pooling, Padding and Stride
- Convolutional Neural Networks (LeNet)
- Deep ConvNets (AlexNet)
- Networks using Blocks (VGG)
- Residual Neural Networks (ResNet)