

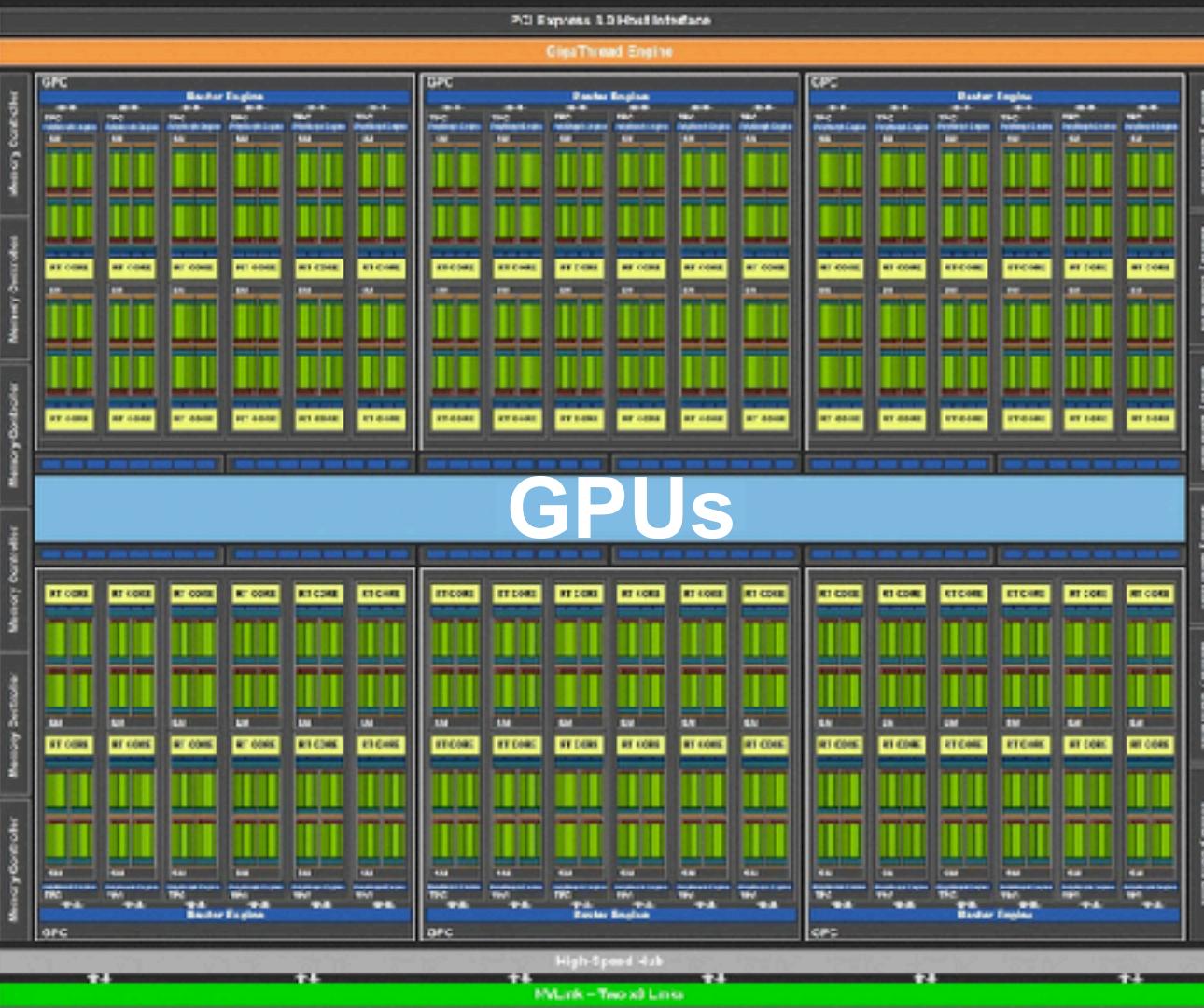
Convolutional Neural Networks

Rachel Hu and Zhi Zhang
Amazon AI

Outline

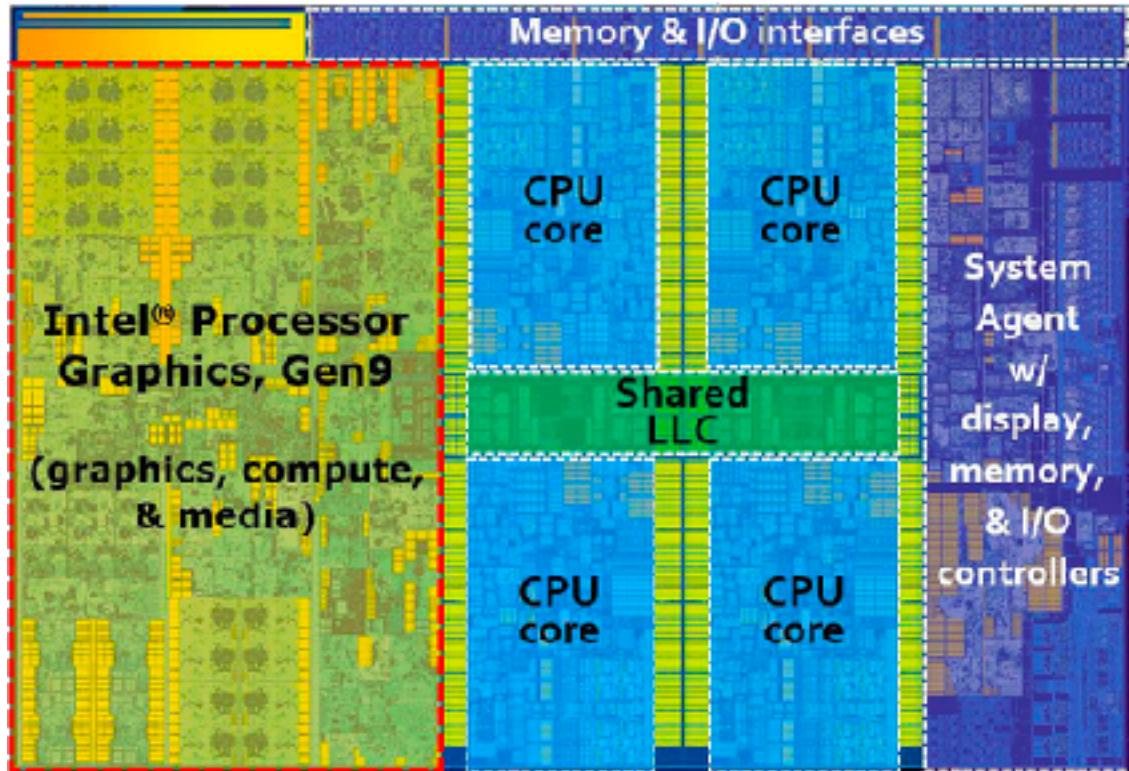
- **GPUs**
- **Convolutions**
- **Pooling, Padding and Stride**
- **Convolutional Neural Networks (LeNet)**
- **Deep ConvNets (AlexNet)**
- **Networks using Blocks (VGG)**
- **Residual Neural Networks (ResNet)**

NVIDIA Turing TU102

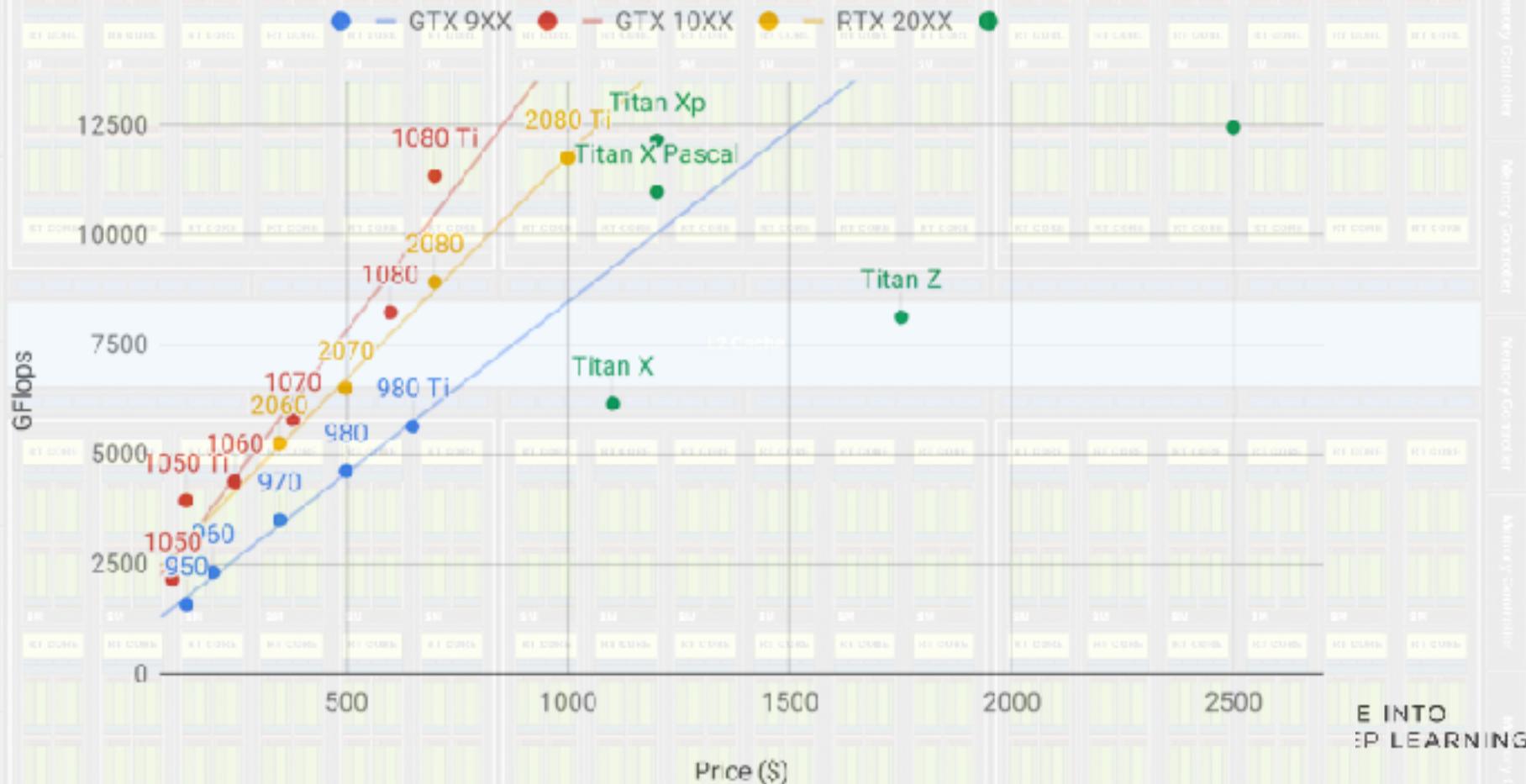


Intel i7-6700K

- 4 Physical cores
- Per core
 - 64KB L1 cache
 - 256KB L2 cache
- Shared 8MB L3 cache
- 30 GB/s to RAM



GPU performance

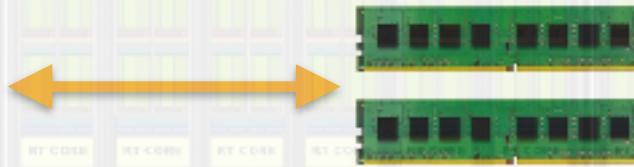


Highend Gaming / DeepLearning PC

Intel i7
0.15 TFLOPS



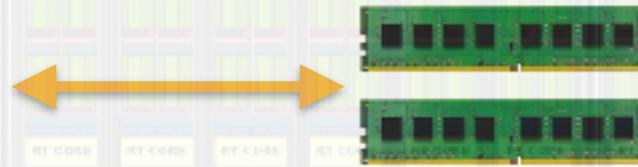
DDR4
32 GB



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

A large, dense crowd of people, mostly young adults, are dressed in red and white striped clothing, including hats and shirts, resembling the character Waldo from the children's book series. They are cheering and raising their hands in excitement. The background is dark, suggesting an outdoor event at night.

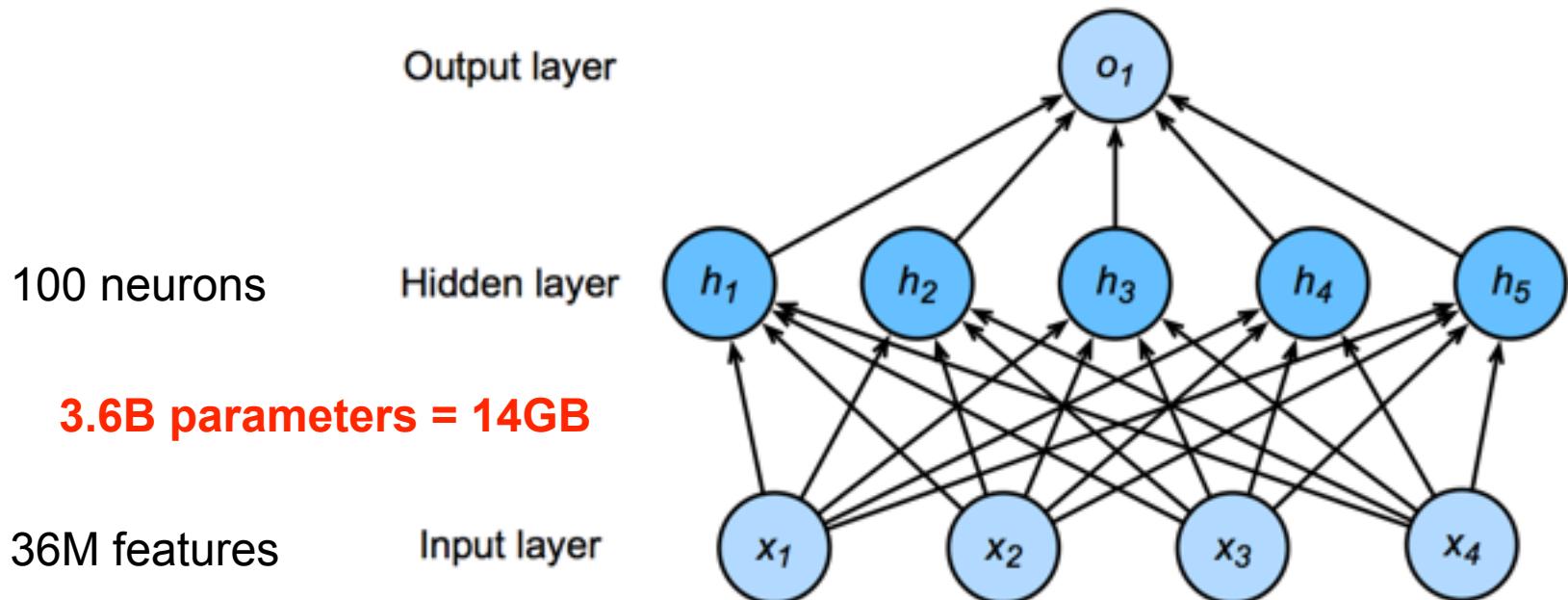
From fully connected
to convolutions

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



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



DIVE INTO
DEEP LEARNING

Where is Waldo?



Two Principles

- Translation Invariance
- Locality



Rethinking Dense Layers

- Reshape inputs and output into matrix (width, height)
- Reshape weights into 4-D tensors (h, w) to (h', w')

$$h_{i,j} = \sum_{k,l} w_{i,j,k,l} x_{k,l} = \sum_{a,b} v_{i,j,a,b} x_{i+a, j+b}$$

V is re-indexes W such as that

$$v_{i,j,a,b} = w_{i,j,i+a, j+b}$$

Idea #1 - Translation Invariance

$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$$

- A shift in x also leads to a shift in h
- v should not depend on (i,j) . Fix via $v_{i,j,a,b} = v_{a,b}$

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a,j+b}$$

That's a 2-D convolution

cross-correlation



DIVE INTO
DEEP LEARNING

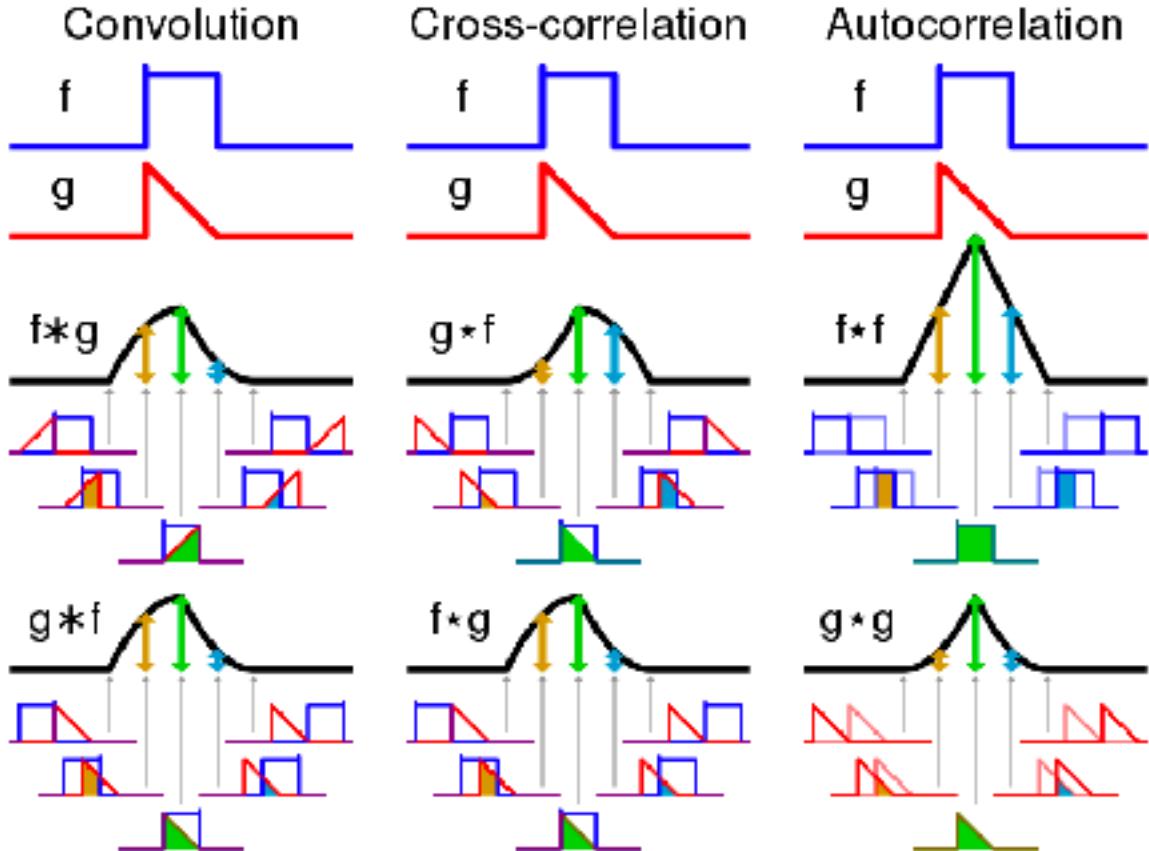
Idea #2 - Locality

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a, j+b}$$

- We shouldn't look very far from $x(i,j)$ in order to assess what's going on at $h(i,j)$
- Outside range $|a|, |b| > \Delta$ parameters vanish $v_{a,b} = 0$

$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a, j+b}$$

Convolution



2-D Cross Correlation

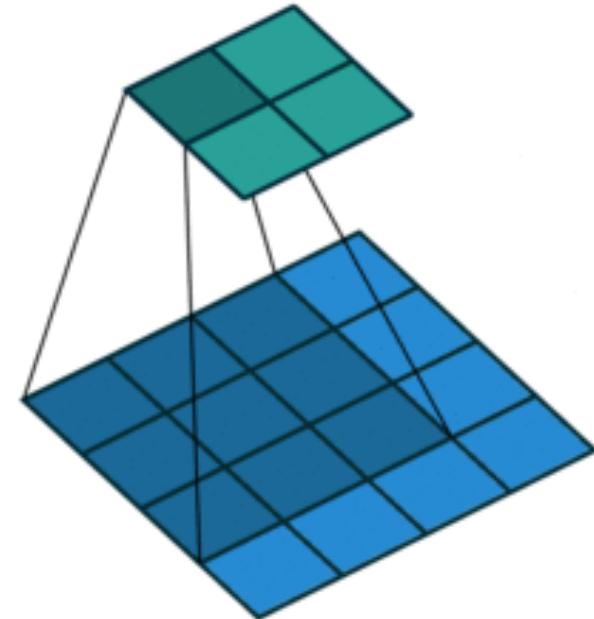
Input	Kernel	Output
$\begin{matrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{matrix}$	$* \quad \begin{matrix} 0 & 1 \\ 2 & 3 \end{matrix}$	$= \quad \begin{matrix} 19 & 25 \\ 37 & 43 \end{matrix}$

$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$

$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$

$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$

$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$



(vdumoulin@ Github)

2-D Convolution Layer

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

- $\mathbf{X} : n_h \times n_w$ 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

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

- \mathbf{W} and b are learnable parameters

Examples

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



Edge Detection



(wikipedia)

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



Sharpen

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



Gaussian Blur

Examples



(Rob Fergus)



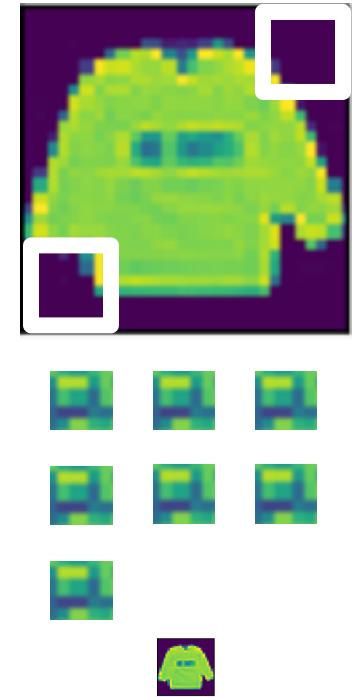
Convolutions Notebook



Padding and Stride

Padding

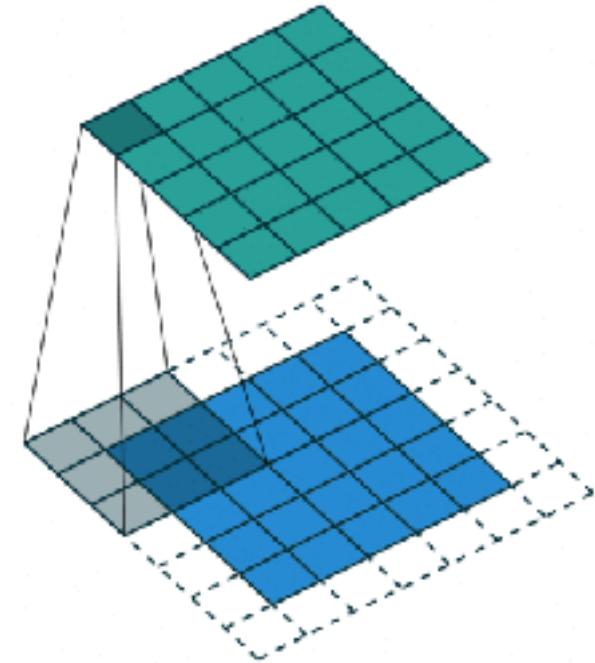
- Given a 32×32 input image
- Apply convolutional layer with 5×5 kernel
 - 28×28 output with 1 layer
 - 4×4 output with 7 layers
- Shape decreases faster with larger kernels
 - Shape reduces from $n_h \times n_w$ to
$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$



Padding

Padding adds rows/columns around input

Input					Kernel		Output					
0	0	0	0	0	*	0	1	=	0	3	8	4
0	0	1	2	0		2	3		9	19	25	10
0	3	4	5	0					21	37	43	16
0	6	7	8	0					6	7	8	0
0	0	0	0	0								



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

Padding

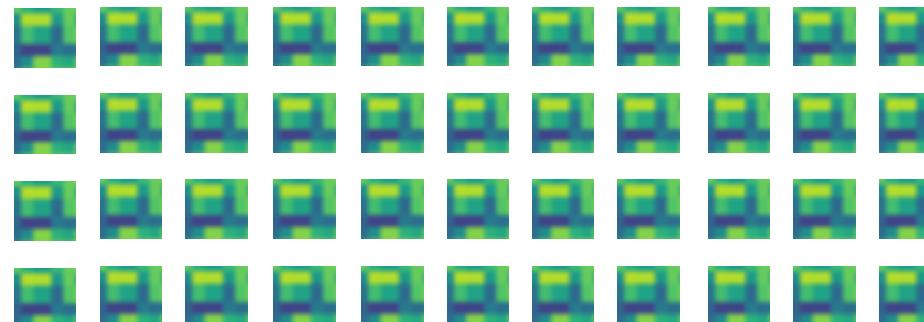
- Padding p_h rows and p_w columns, output shape will be

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

- 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

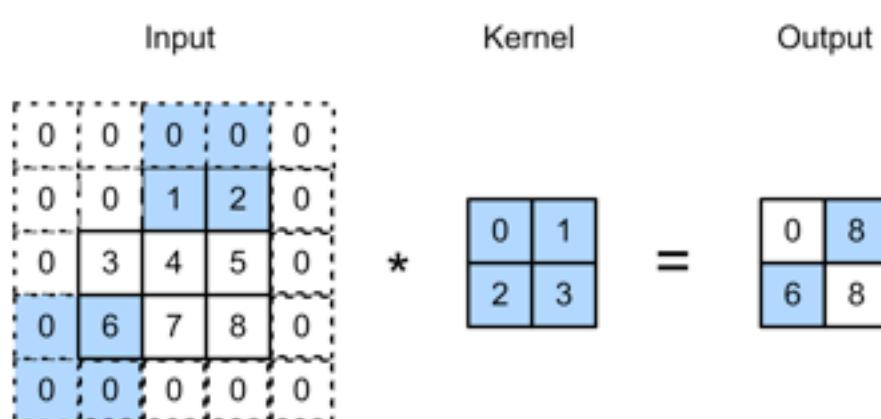
- Padding reduces shape linearly with #layers
 - Given a 224×224 input with a 5×5 kernel, needs 44 layers to reduce the shape to 4×4
 - Requires a large amount of computation



Stride

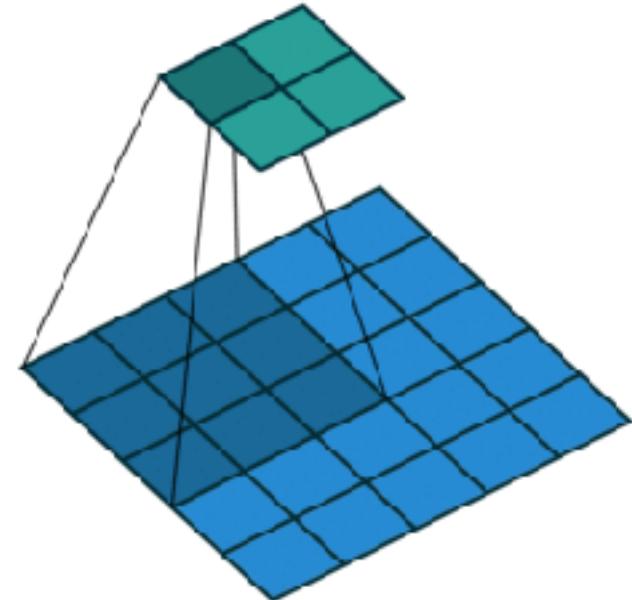
- Stride is the #rows/#columns per slide

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

- 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

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

An aerial photograph showing a complex network of water channels. The channels are narrow and filled with dark green water, creating a grid-like pattern across the landscape. They are bordered by thick, lush green vegetation, likely reeds or cattails, which grow along the banks. The perspective is from above, looking down the length of the channels, which converge towards the top and bottom of the frame.

*Multiple Input and
Output Channels*

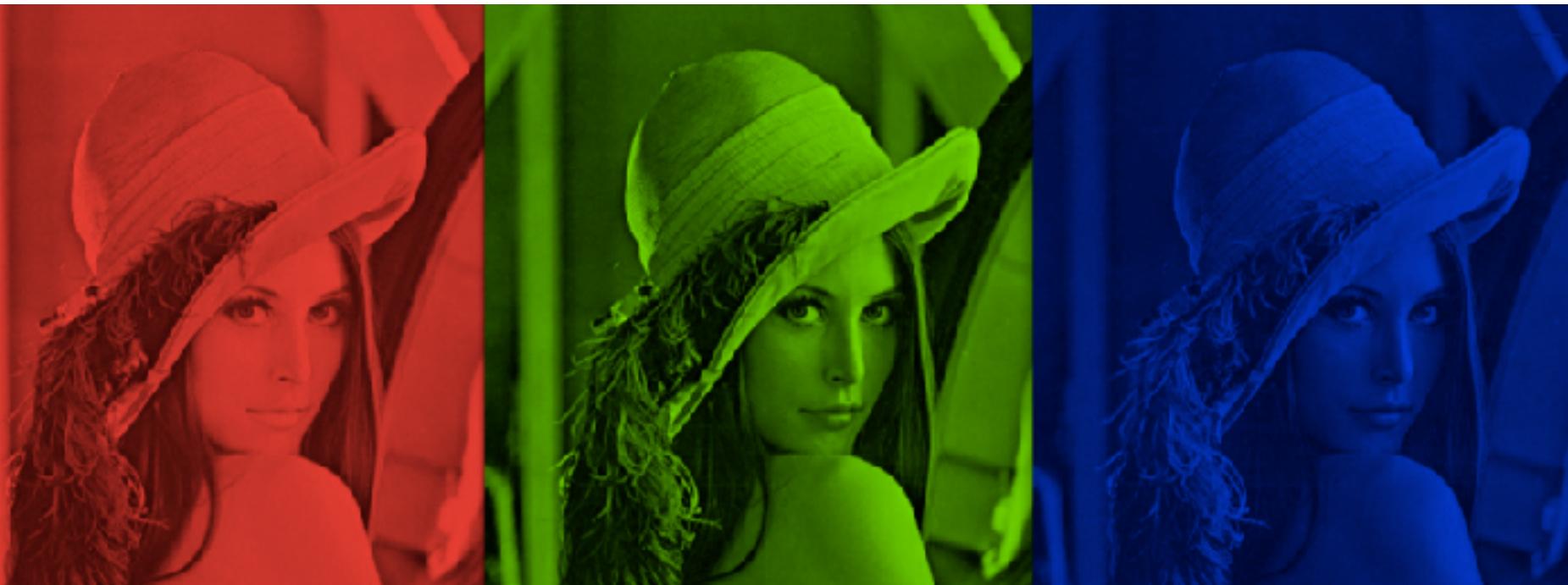
Multiple Input Channels

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



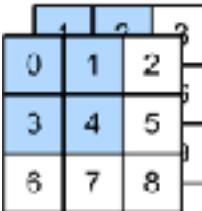
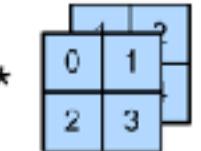
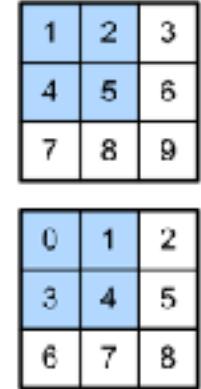
Multiple Input Channels

- 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

Input	Kernel	Input	Kernel	Output
	*		=	
			*	1 2 3 4
			+	56 72
			*	0 1 2 3
			=	104 120
				$(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4)$ $(0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3)$ $= 56$

Multiple Input Channels

- $\mathbf{X} : c_i \times n_h \times n_w$ input
- $\mathbf{W} : c_i \times k_h \times k_w$ kernel
- $\mathbf{Y} : m_h \times m_w$ output

$$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$$

Multiple Output Channels

- No matter how many inputs channels, so far we always get single output channel
- We can have multiple 3-D kernels, each one generates a output channel
- Input $\mathbf{X} : c_i \times n_h \times n_w$
- Kernel $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- Output $\mathbf{Y} : c_o \times m_h \times m_w$

$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:}$$

for $i = 1, \dots, c_o$

Multiple Input/Output Channels

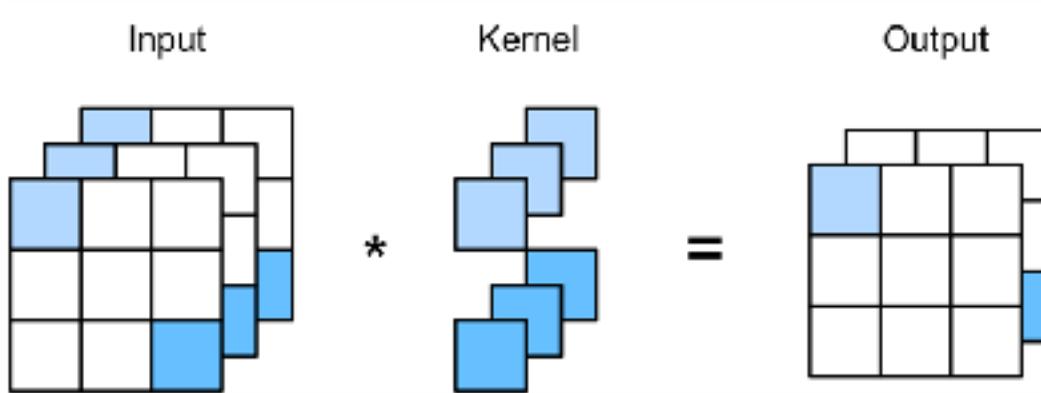
- Each output channel may recognize a particular pattern



- Input channels kernels recognize and combines patterns in inputs

1×1 Convolutional Layer

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



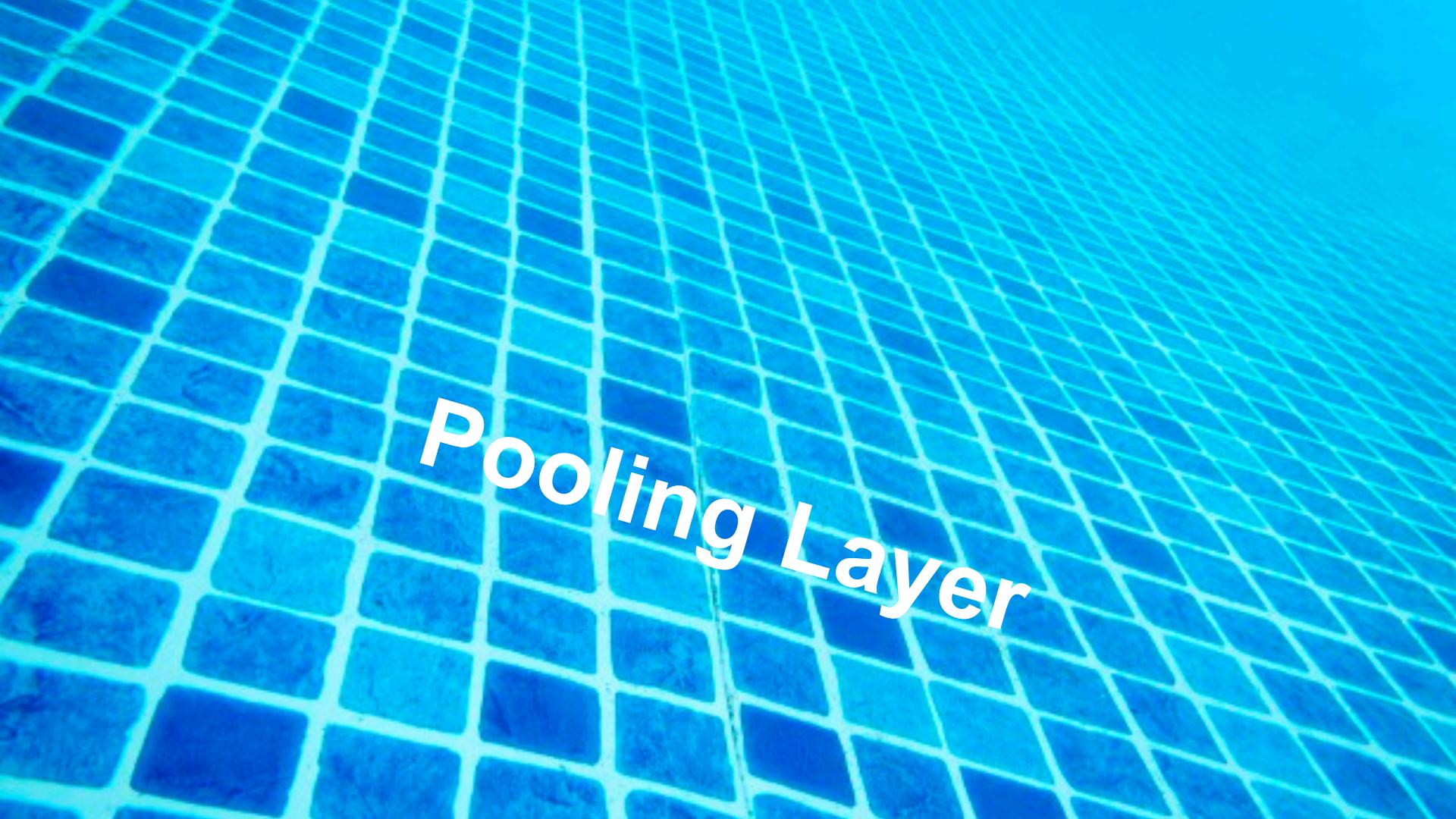
Equal to a dense layer with $n_h n_w \times c_i$ input and $c_o \times c_i$ weight.

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 $\mathbf{Y} : c_o \times m_h \times m_w$
- Complexity (number of floating point operations FLOP)
 $c_i = c_o = 100$
 $k_h = h_w = 5$
 $m_h = m_w = 64$

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

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

The background of the image is a swimming pool with blue square tiles on the bottom. The water is clear and has a slight blue tint. There are some small ripples and reflections on the surface of the water.

Pooling Layer

Pooling

- Convolution is sensitive to position
 - Detect vertical edges

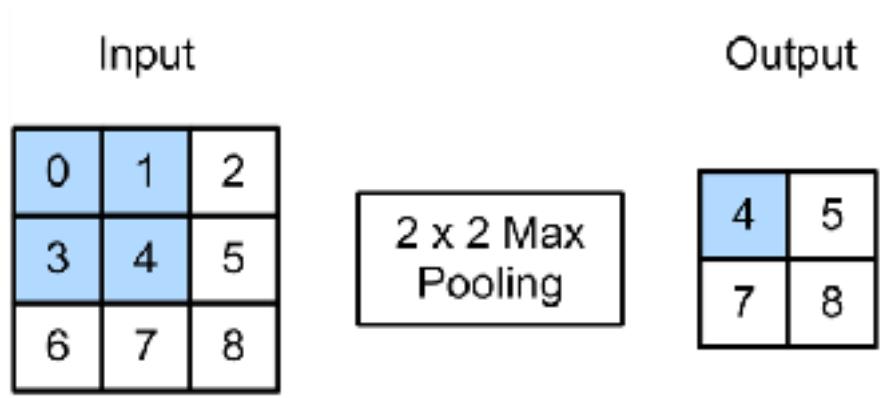
0 output with
1 pixel shift

$$\begin{array}{c} \times \\ X \end{array} \quad \begin{bmatrix} [1. & 1. & 0. & 0. & 0.] \\ [1. & 1. & 0. & 0. & 0.] \\ [1. & 1. & 0. & 0. & 0.] \\ [1. & 1. & 0. & 0. & 0.] \end{bmatrix} \quad Y \quad \begin{bmatrix} [0. & 1. & 0. & 0.] \\ [0. & 1. & 0. & 0.] \\ [0. & 1. & 0. & 0.] \\ [0. & 1. & 0. & 0.] \end{bmatrix}$$

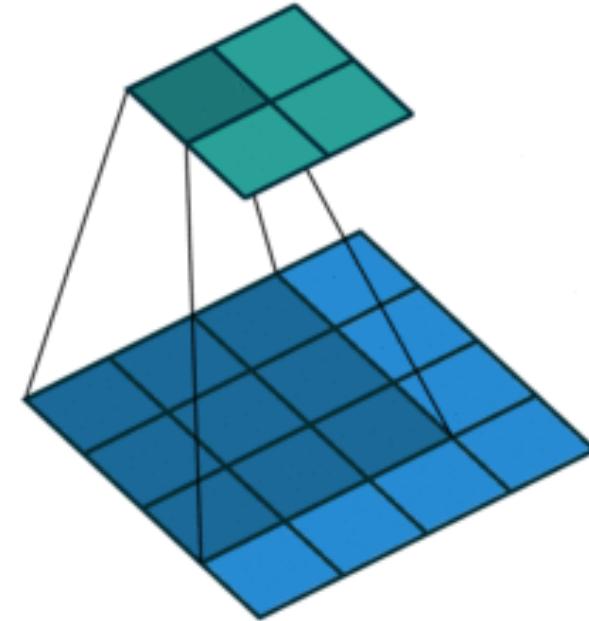
- We need some degree of invariance to translation
 - Lighting, object positions, scales, appearance vary among images

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

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

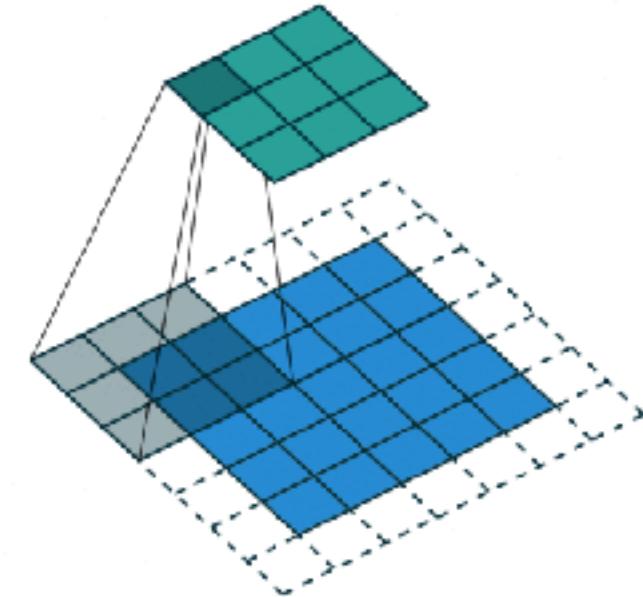
2 x 2 max pooling



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

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

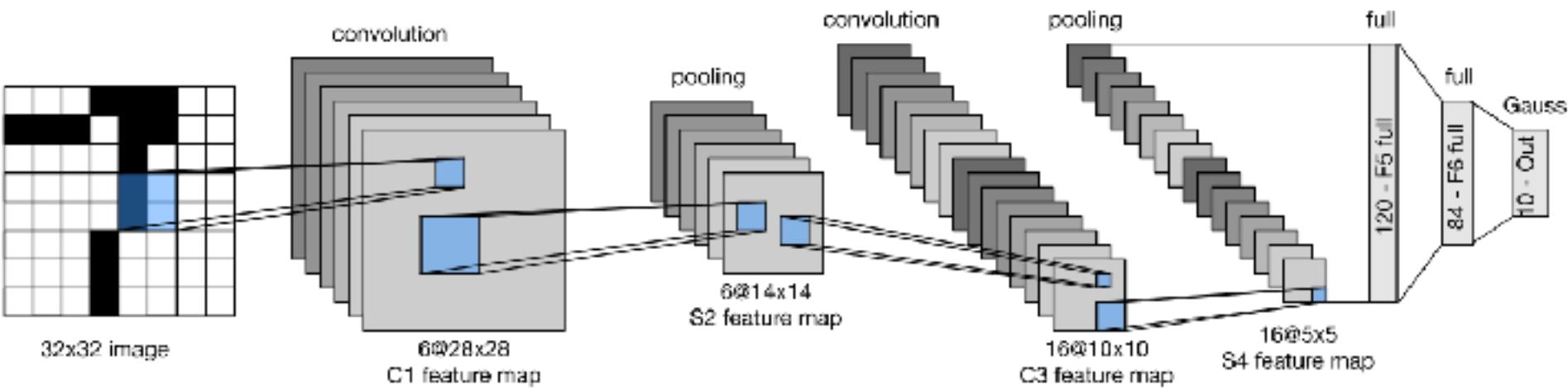


Average pooling

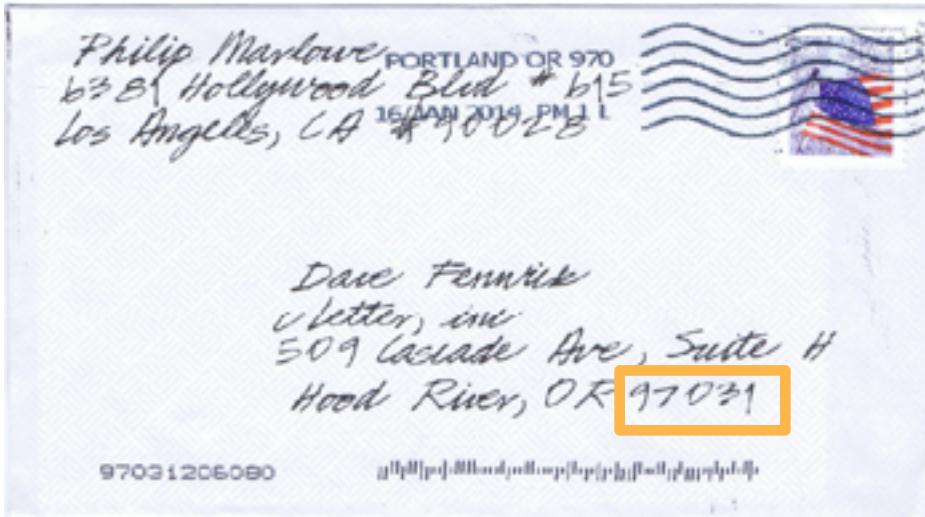


Pooling Notebook

LeNet



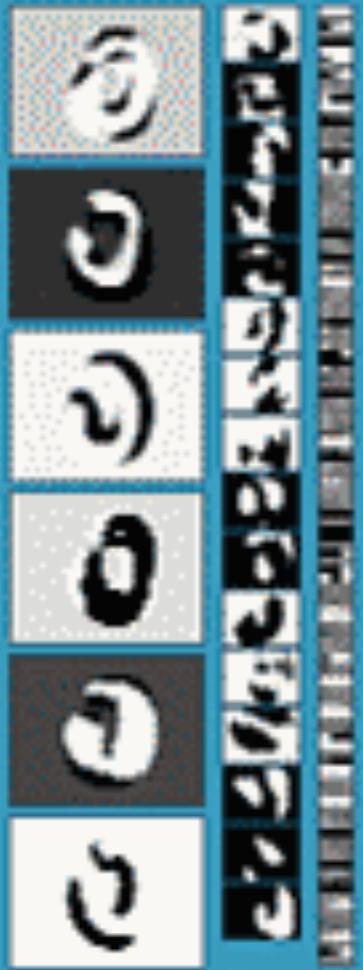
Handwritten Digit Recognition



MNIST

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

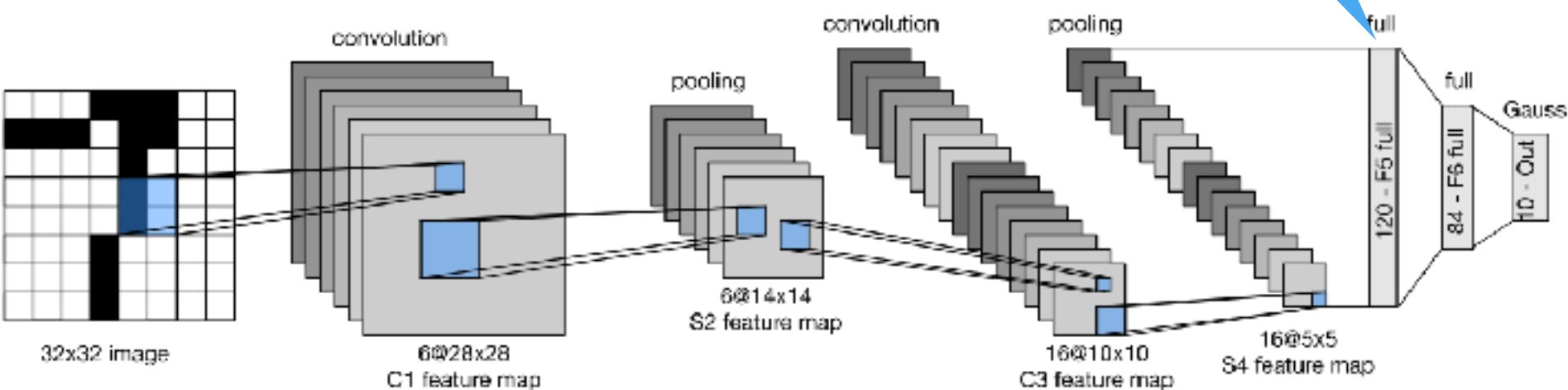




The image shows a screenshot of the LeNet-5 research software. At the top, the AT&T logo and the text "LeNet 5" are displayed in red, with "RESEARCH" in blue to the right. Below this, the word "answer:" is followed by a red "0". In the center, there is a digital representation of the digit "0" with the number "103" next to it. At the bottom, there is a 28x28 pixel grayscale image of a handwritten digit, which appears to be a '0'. The entire interface has a light beige background with a subtle grid pattern.

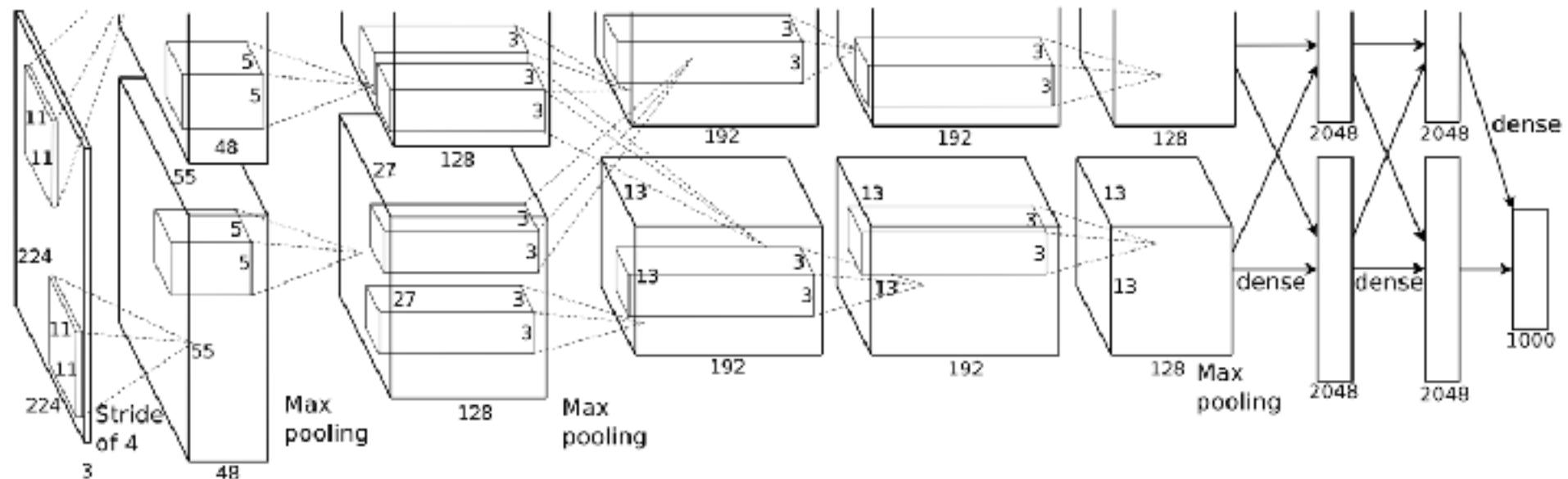
Y. LeCun, L.
Bottou, Y. Bengio,
P. Haffner, 1998
Gradient-based
learning applied to
document
recognition

Expensive if we
have many
outputs



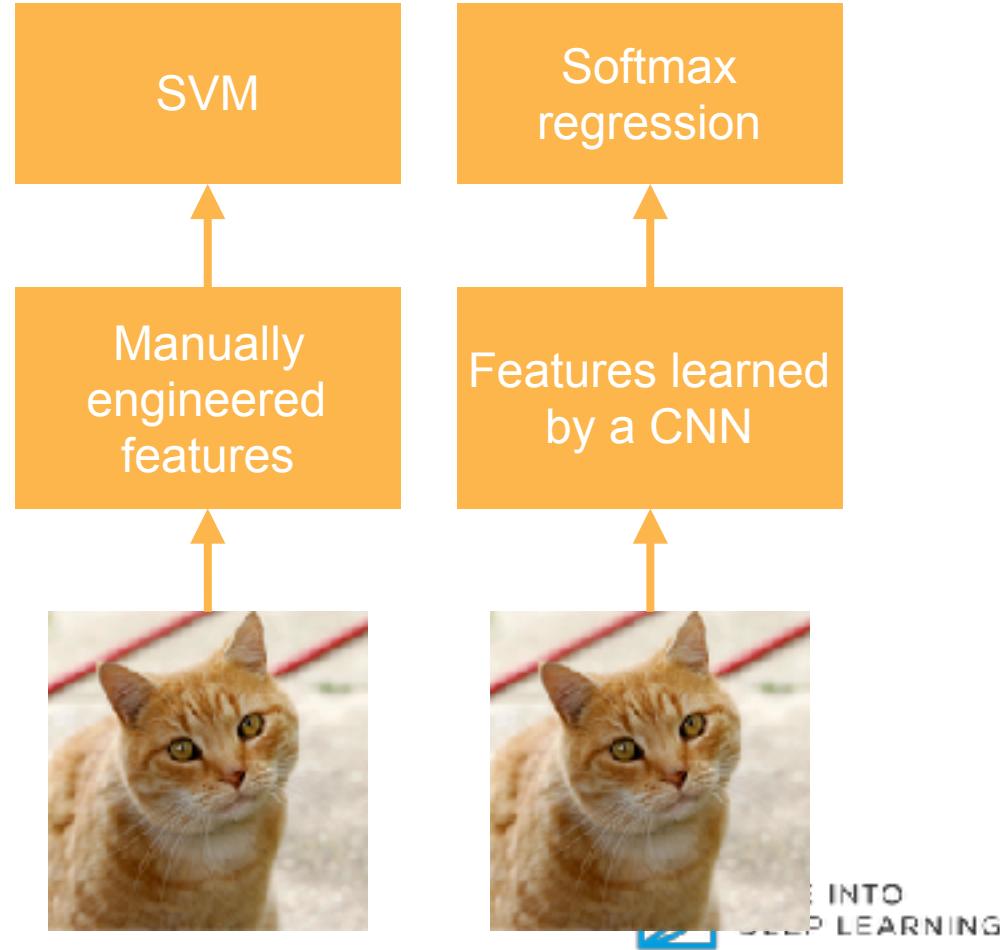
LeNet Notebook

AlexNet

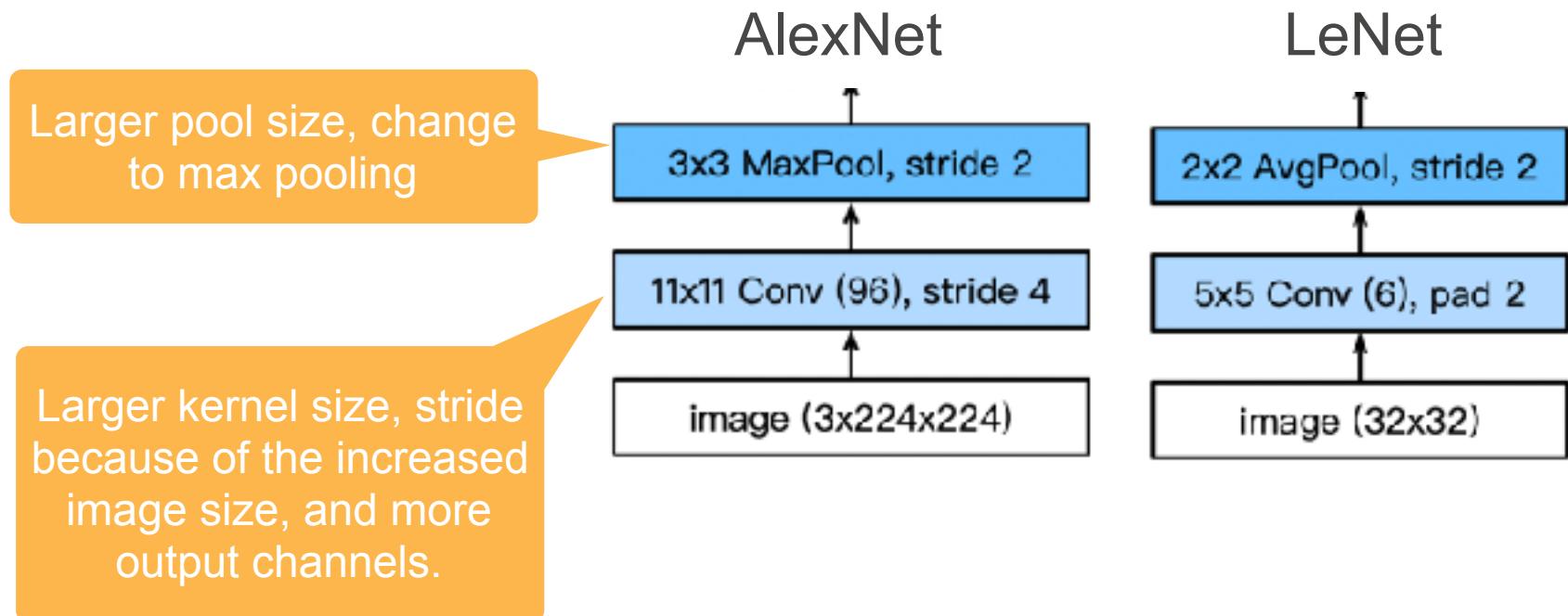


AlexNet

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

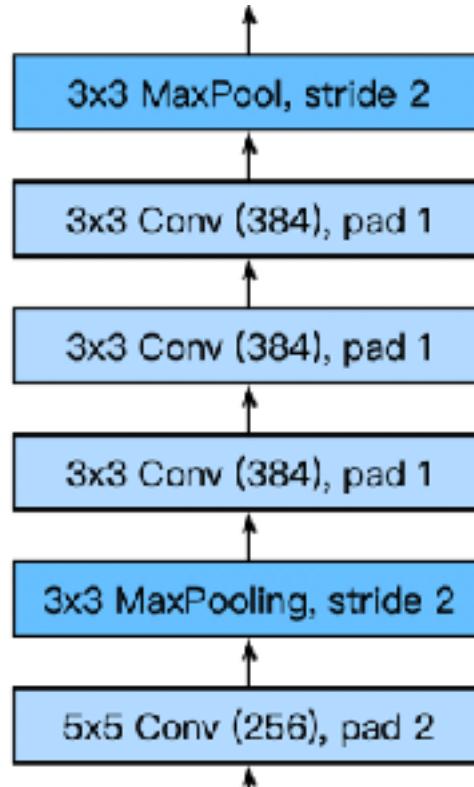


AlexNet Architecture

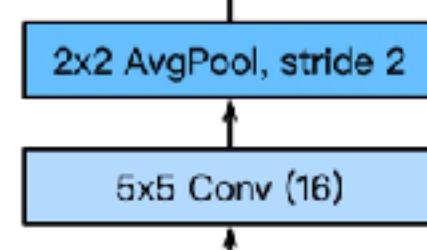


AlexNet Architecture

AlexNet



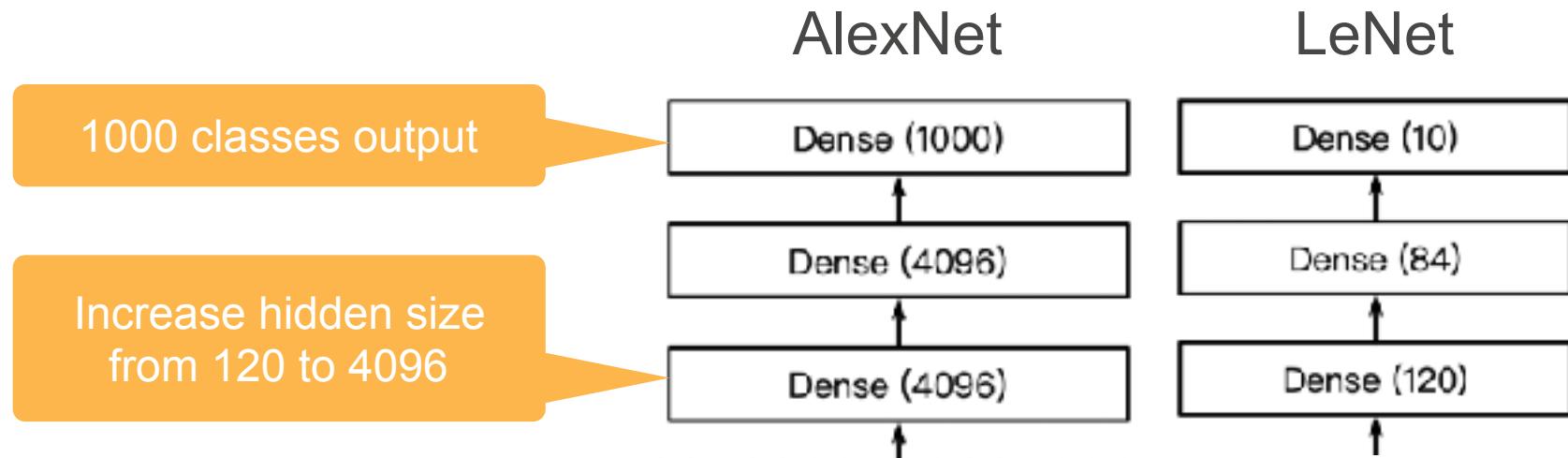
LeNet



3 additional convolutional layers

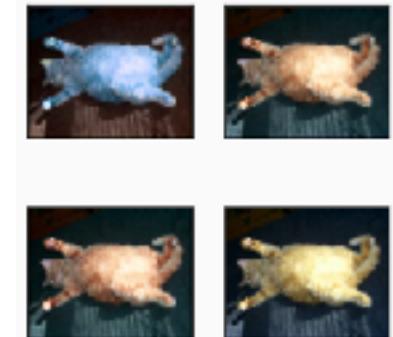
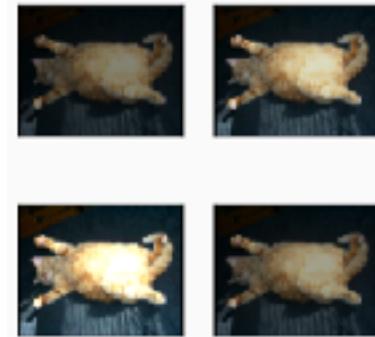
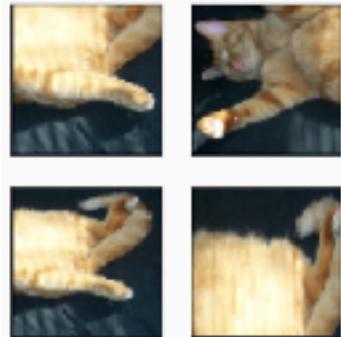
More output channels.

AlexNet Architecture



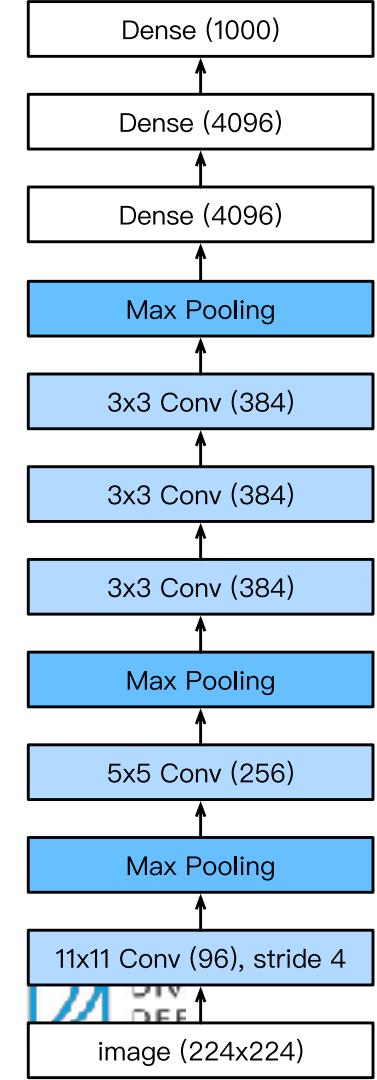
More Tricks

- 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



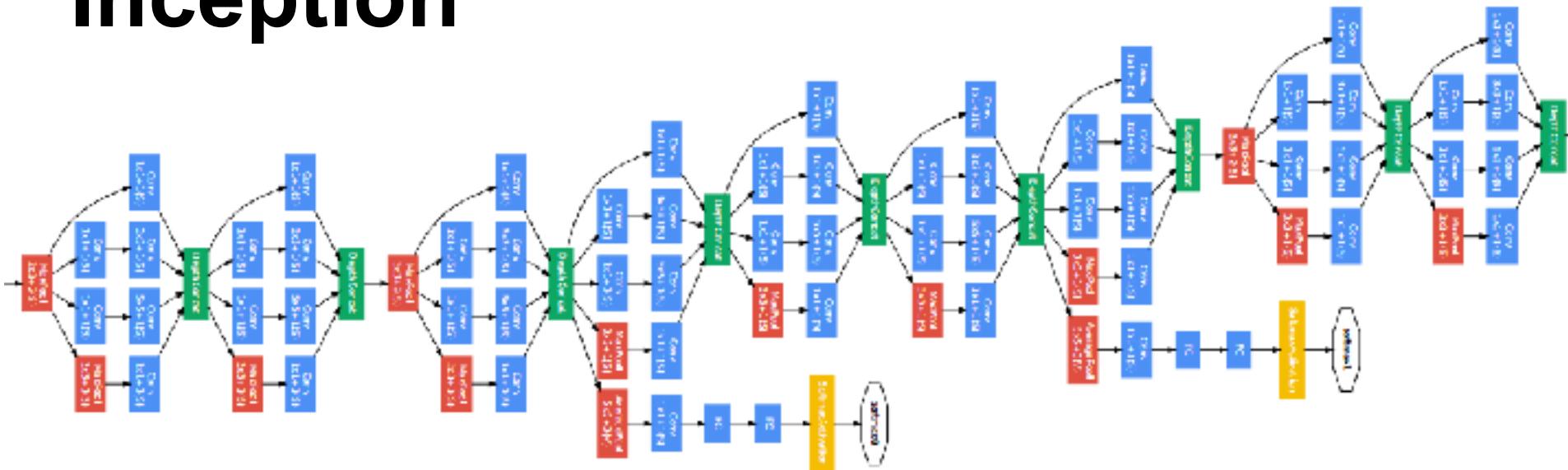
Complexity

	#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



Picking the best convolution ...

1x1

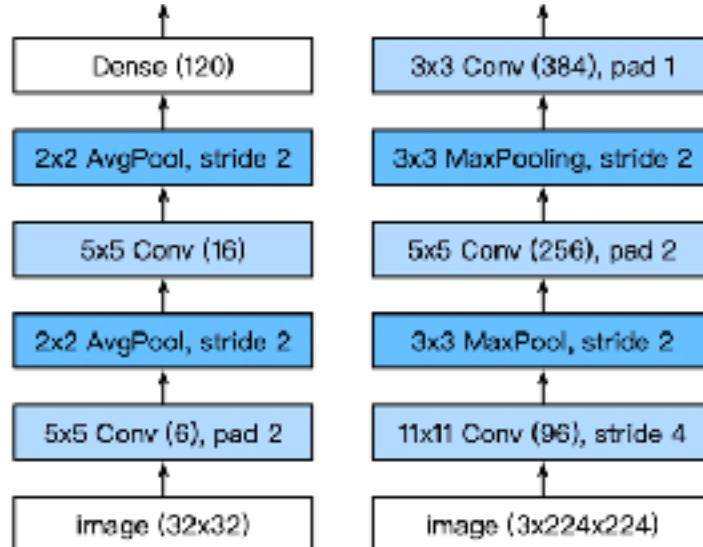
3x3

5x5

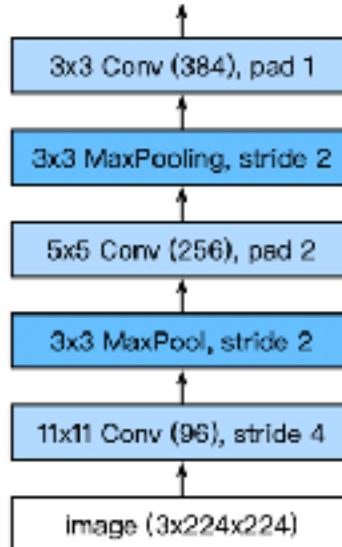
Max pooling

Multiple 1x1

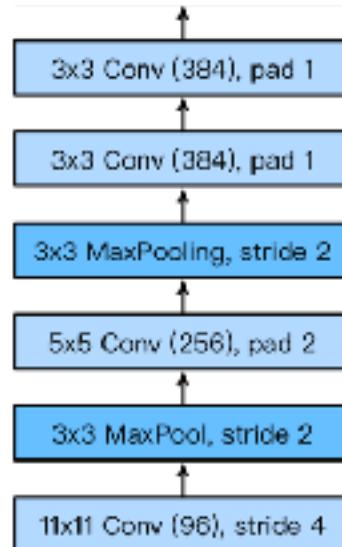
LeNet



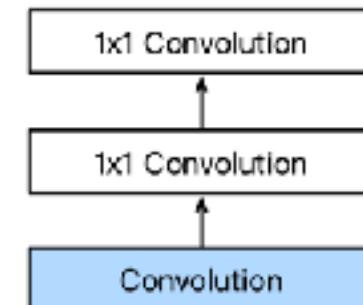
AlexNet



VGG



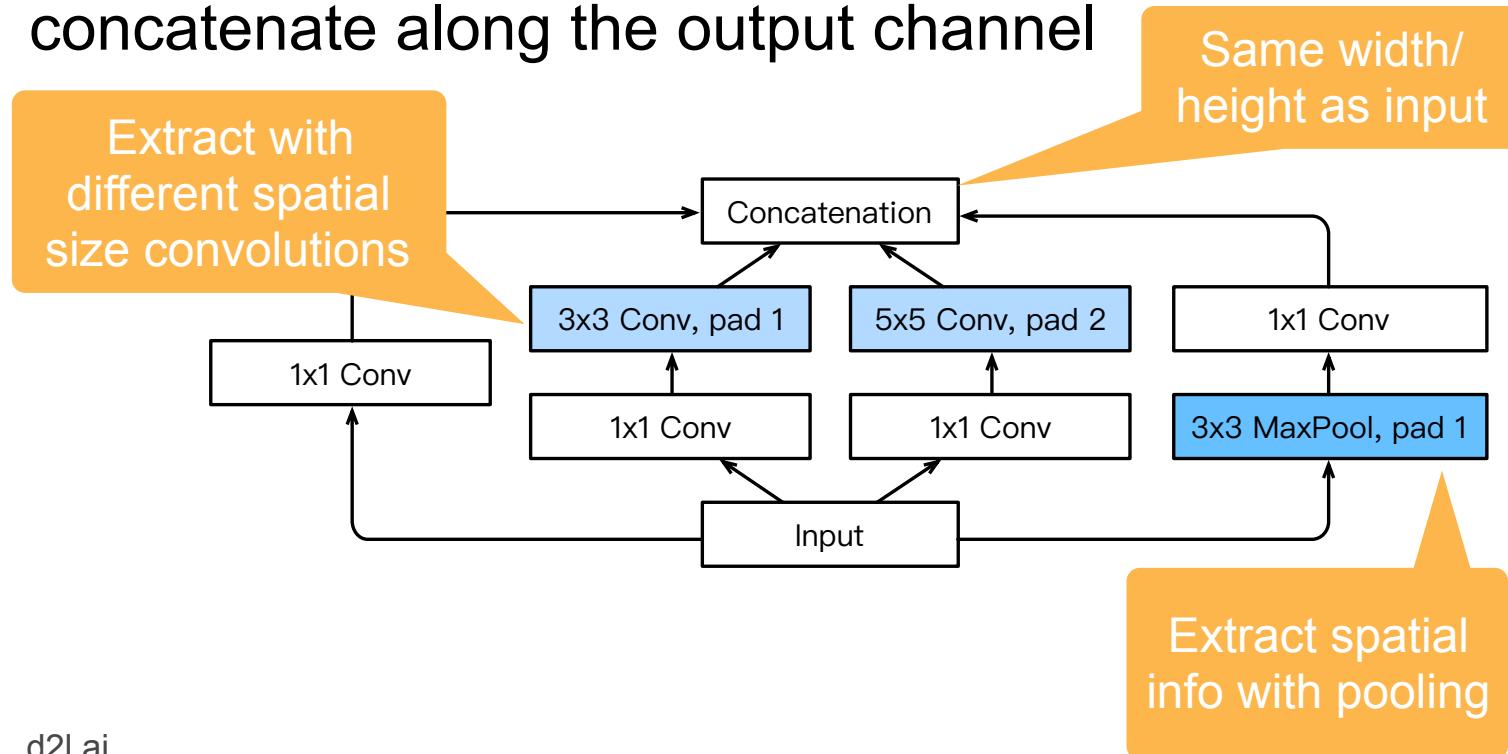
NiN



Why choose? Just pick them all.

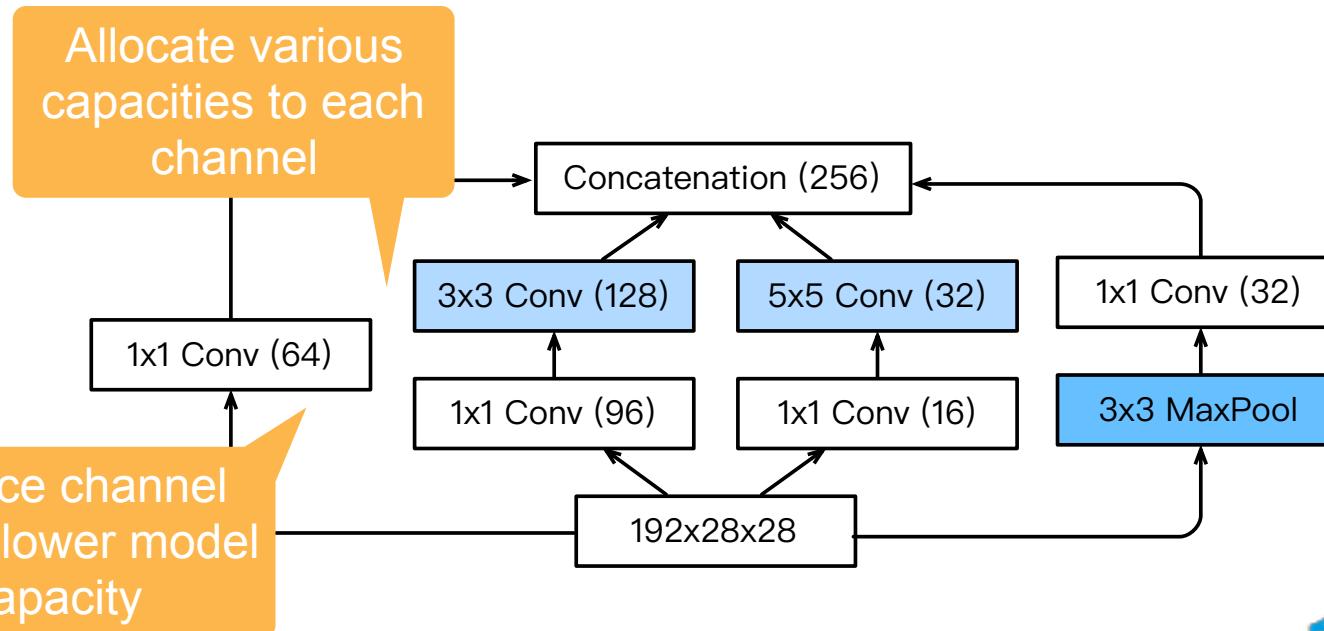
Inception Blocks

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



Inception Blocks

The first inception block with channel sizes specified

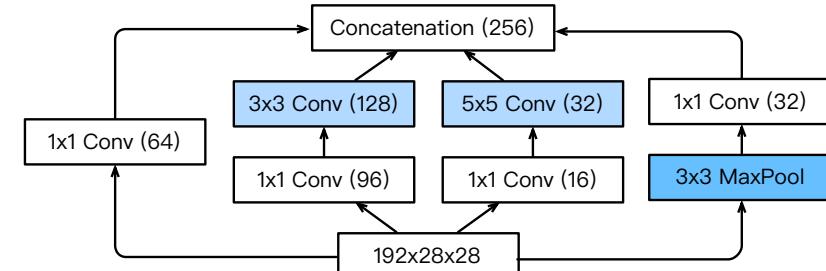


Inception Blocks

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

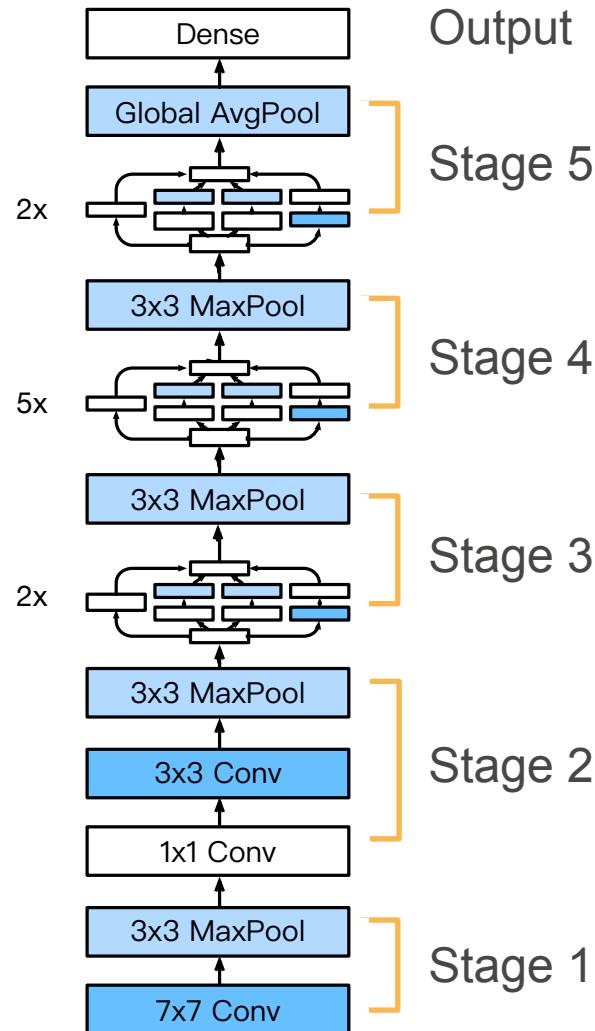
- Mix of different functions (powerful function class)
- Memory and compute efficiency (good generalization)

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



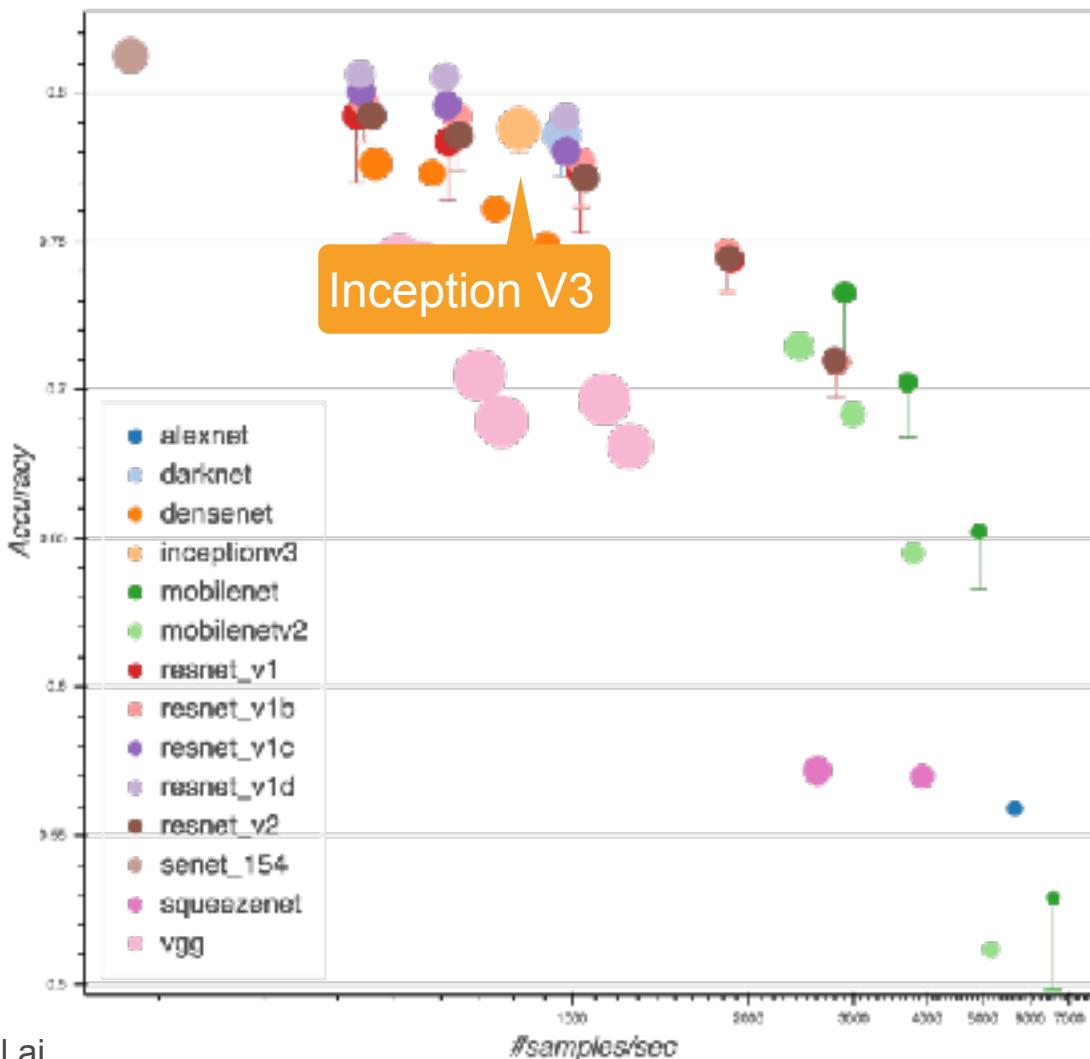
GoogLeNet

- 5 stages with 9 inception 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)

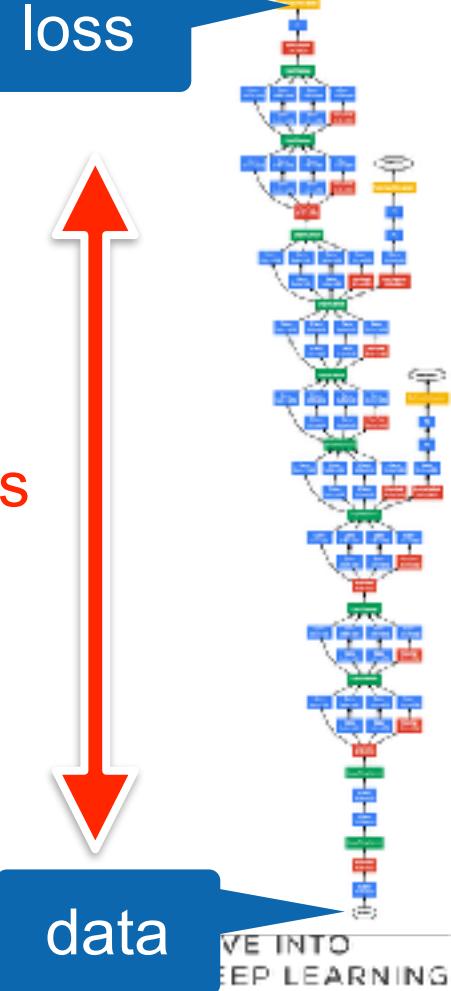


GluonCV Model Zoo
https://gluon-cv.mxnet.io/model_zoo/classification.html



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?



Batch Normalization

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

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

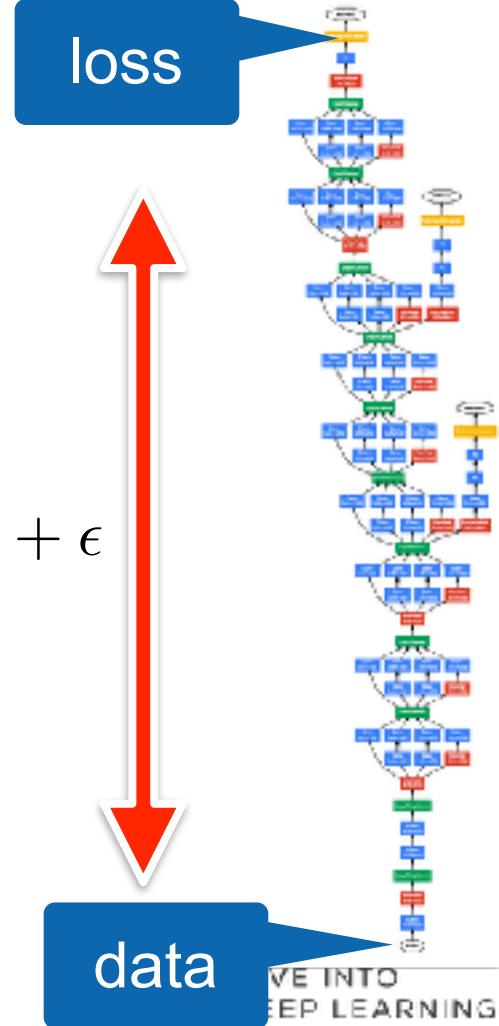
and adjust it separately

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

variance

mean

data



This was the original motivation ...

What Batch Norms really do

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

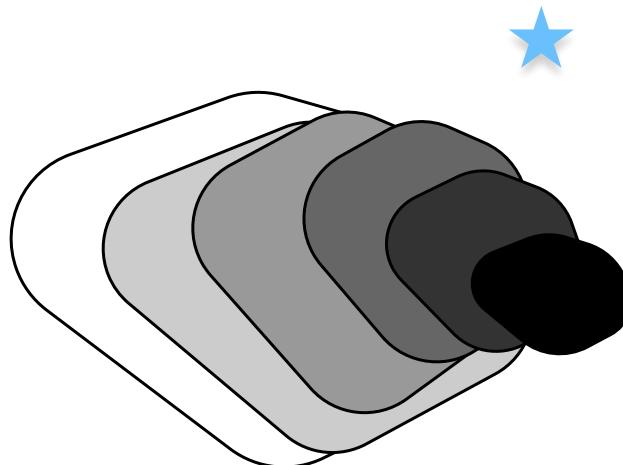
$$x_{i+1} = \gamma \frac{x_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

Random offset

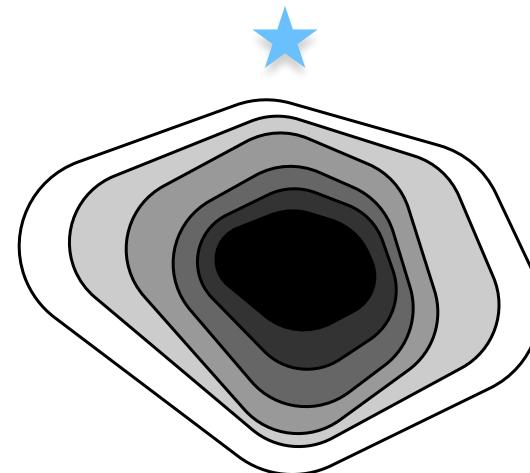
Random scale

- 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

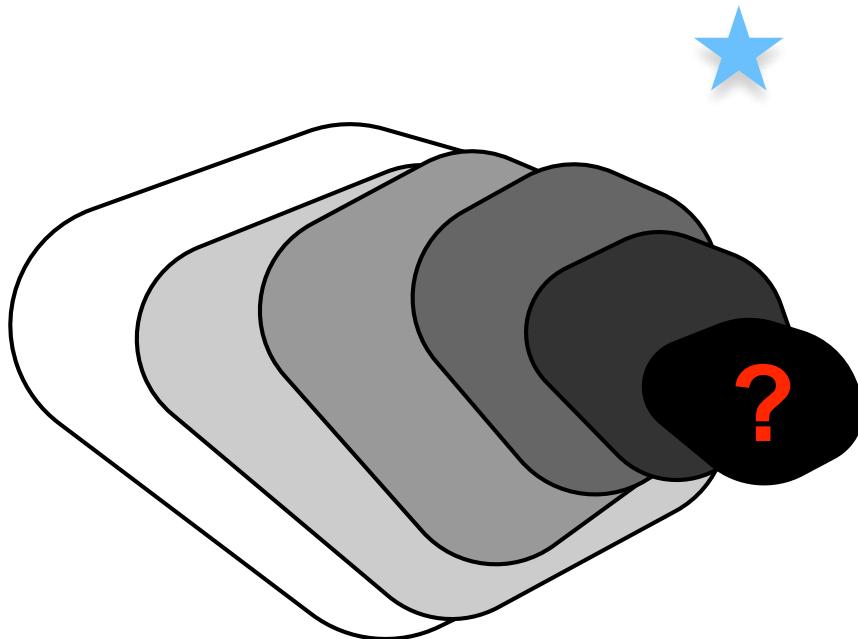


generic function classes

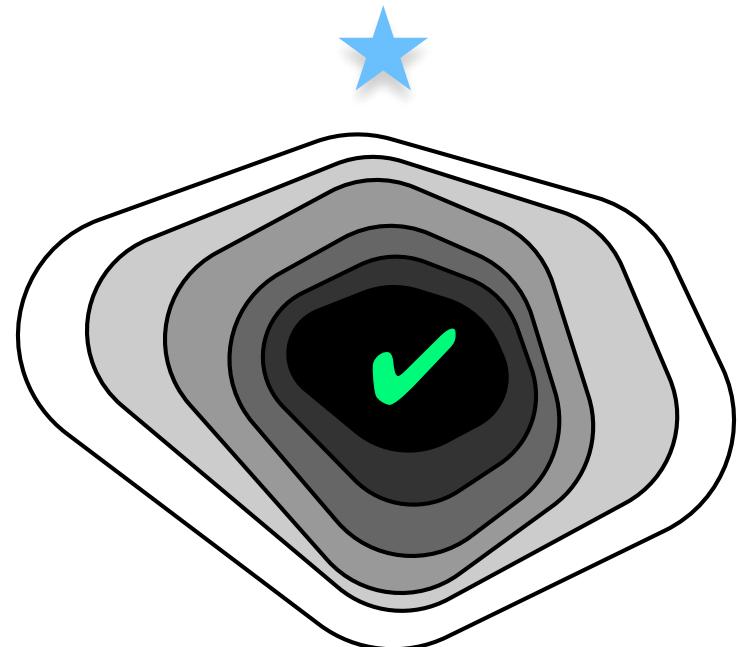


nested function classes

Does adding layers improve accuracy?



generic function classes

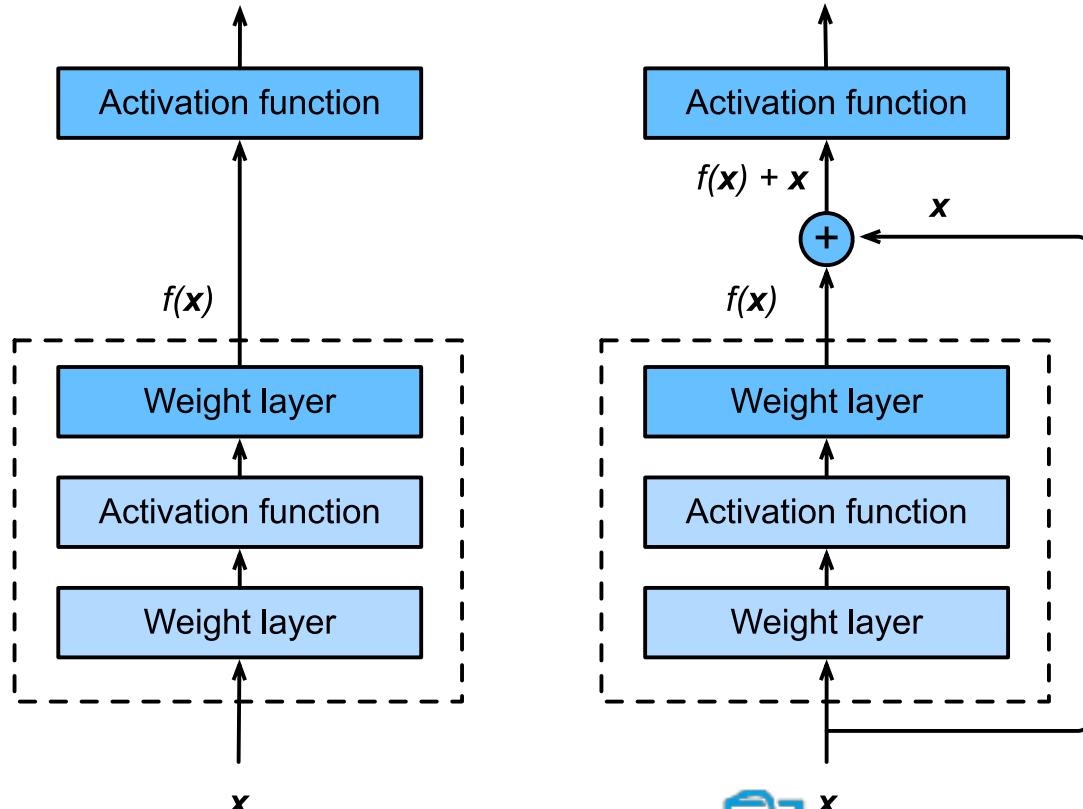


nested function classes

Residual Networks

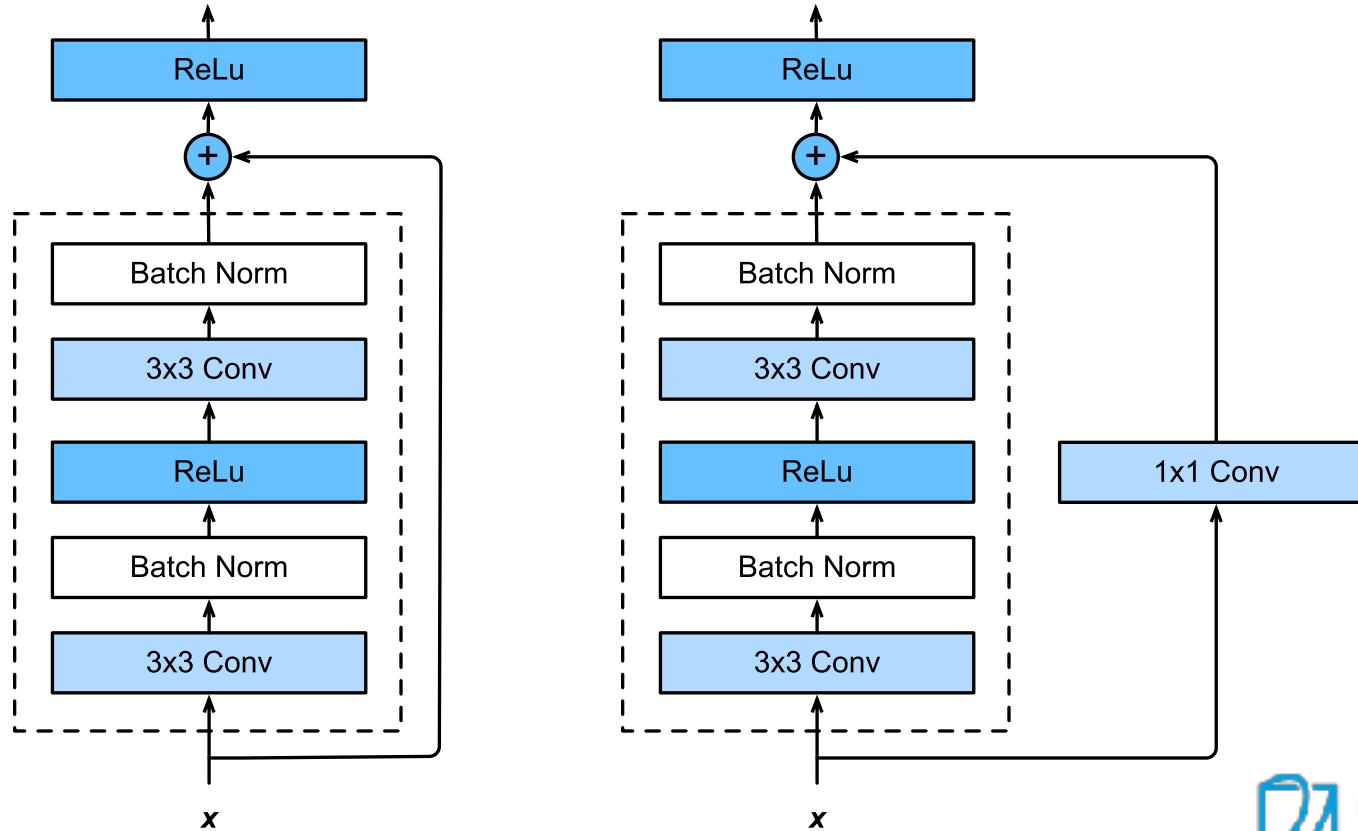
- Adding a layer **changes** function class
- We want to **add to** the function class
- ‘Taylor expansion’ style parametrization

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

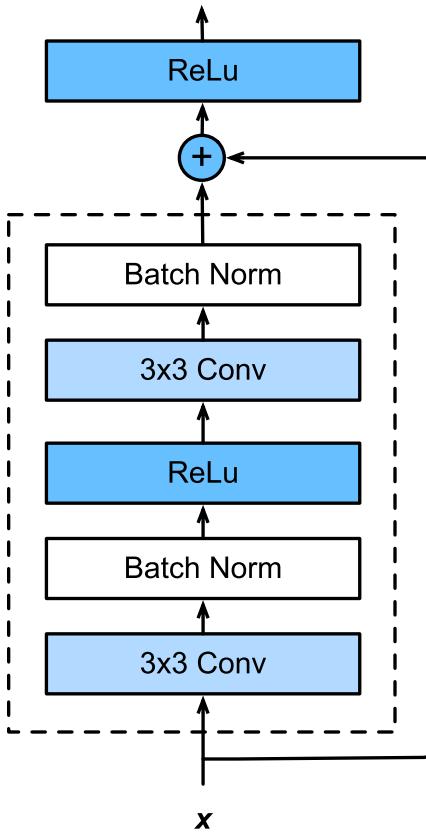


He et al., 2015

ResNet Block in detail

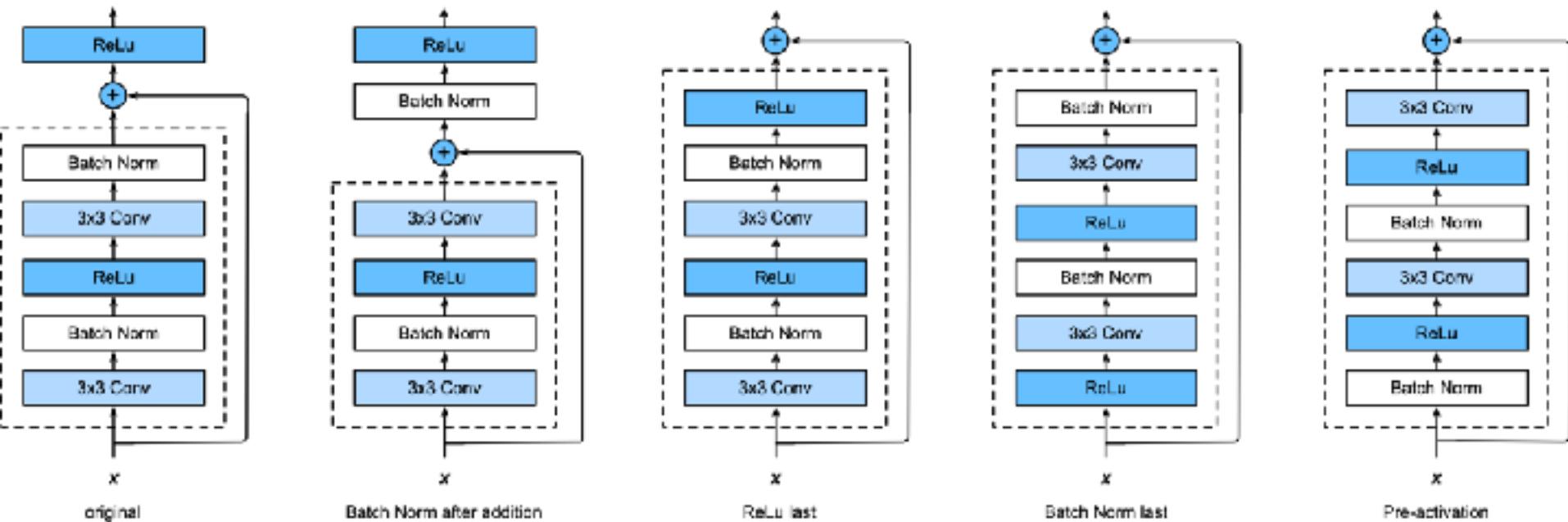


In code



```
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)
```

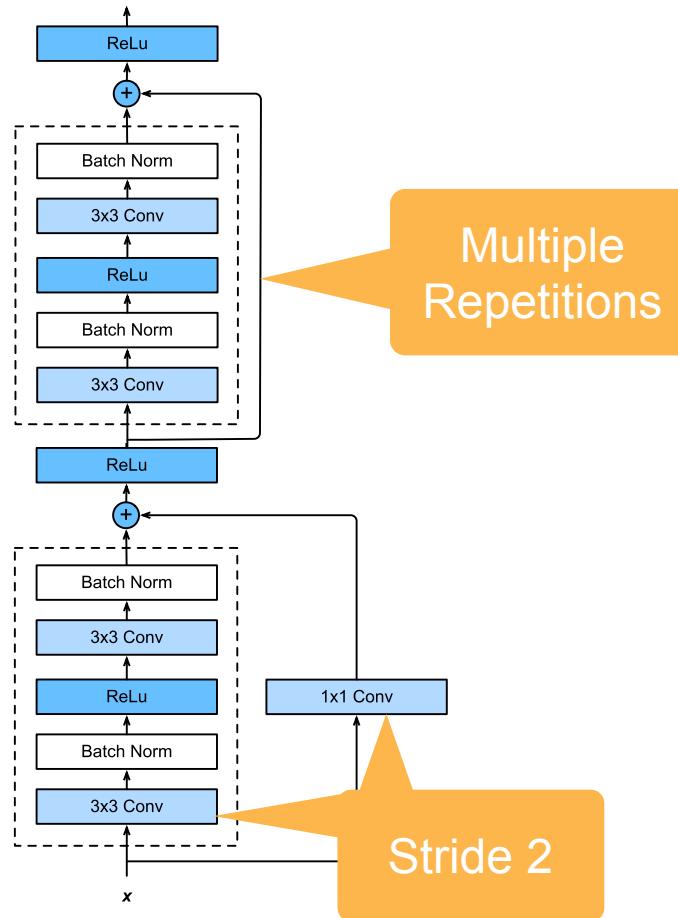
The many flavors of ResNet blocks



Try every permutation

DIVE INTO
DEEP LEARNING

ResNet Module



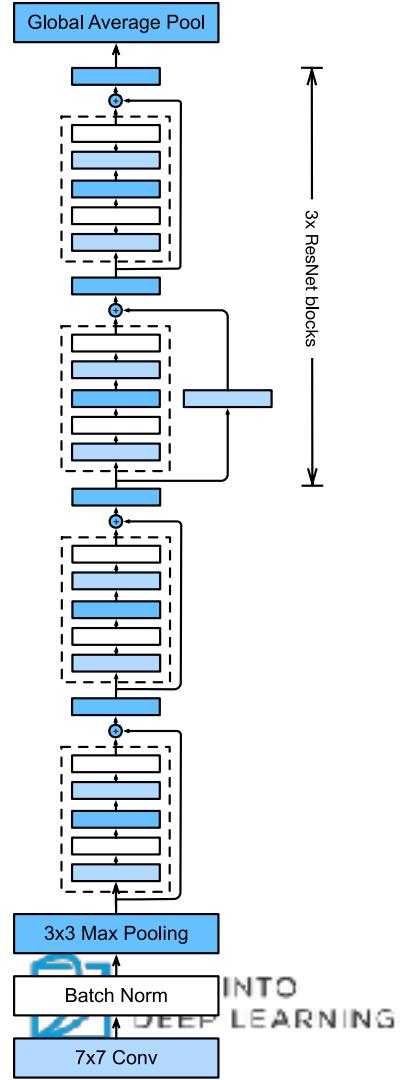
- Downsample per module (stride=2)
- Enforce some nontrivial nonlinearity per module (via 1x1 convolution)
- Stack up in blocks

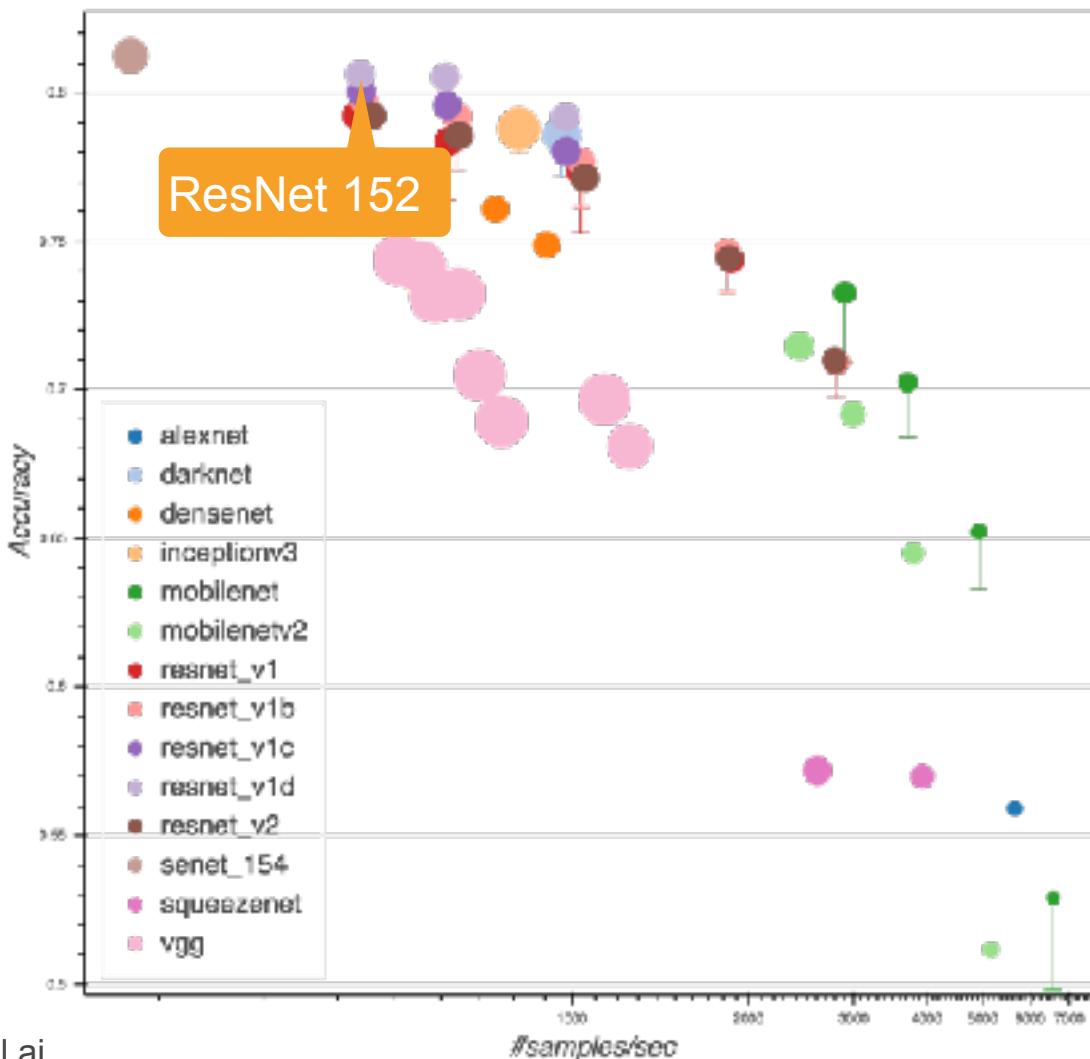
```
blk = nn.Sequential()
for i in range(num_residuals):
    if i == 0 and not first_block:
        blk.add(Residual(num_channels,
                         use_1x1conv=True, strides=2))
    else:
        blk.add(Residual(num_channels))
```

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 ...





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

Jupyter Notebook

More Ideas



DenseNet (Huang et al., 2016)

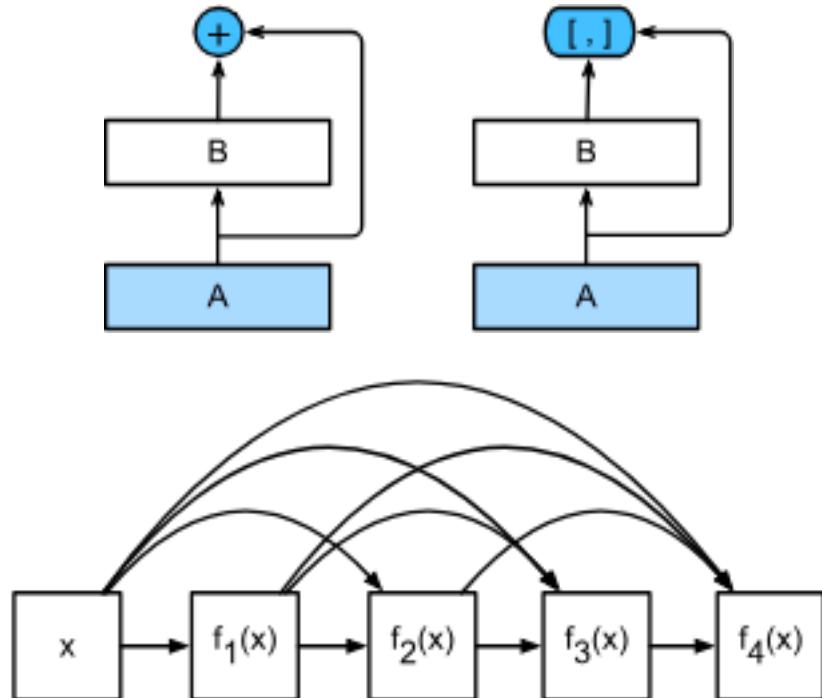
- ResNet combines x and $f(x)$
- DenseNet uses higher order ‘Taylor series’ expansion

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

$$x_1 = x$$

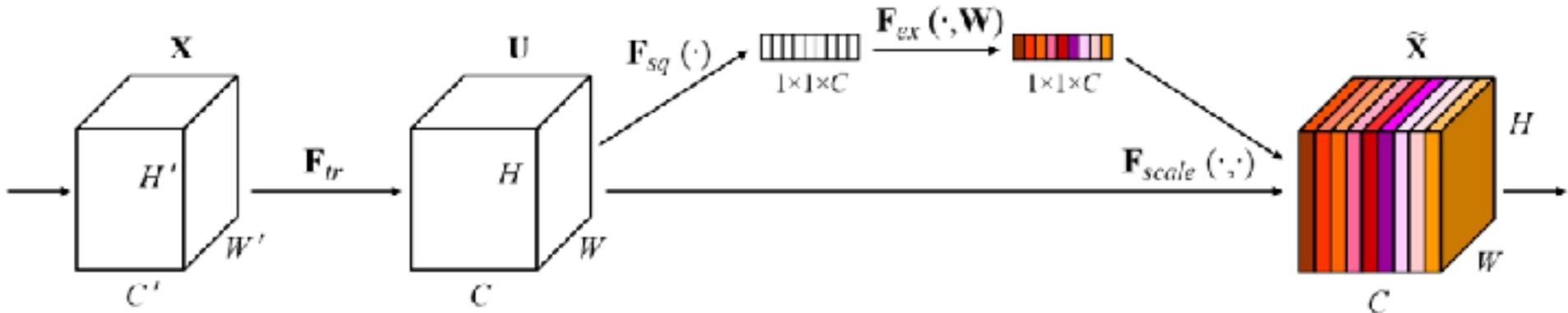
$$x_2 = [x, f_1(x)]$$

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



- Occasionally need to reduce resolution (transition layer)

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

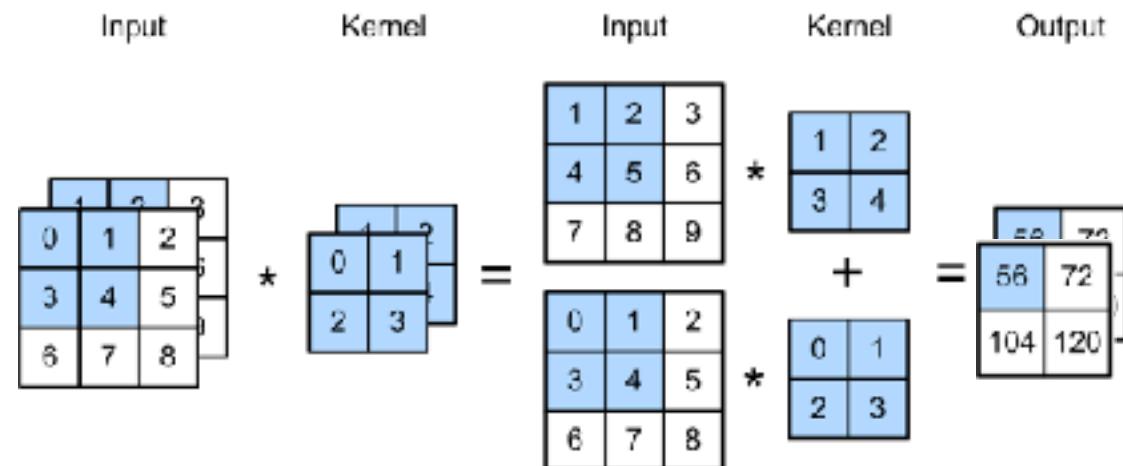


- Learn global weighting function per channel
- Allows for fast information transfer between pixels in different locations of the image

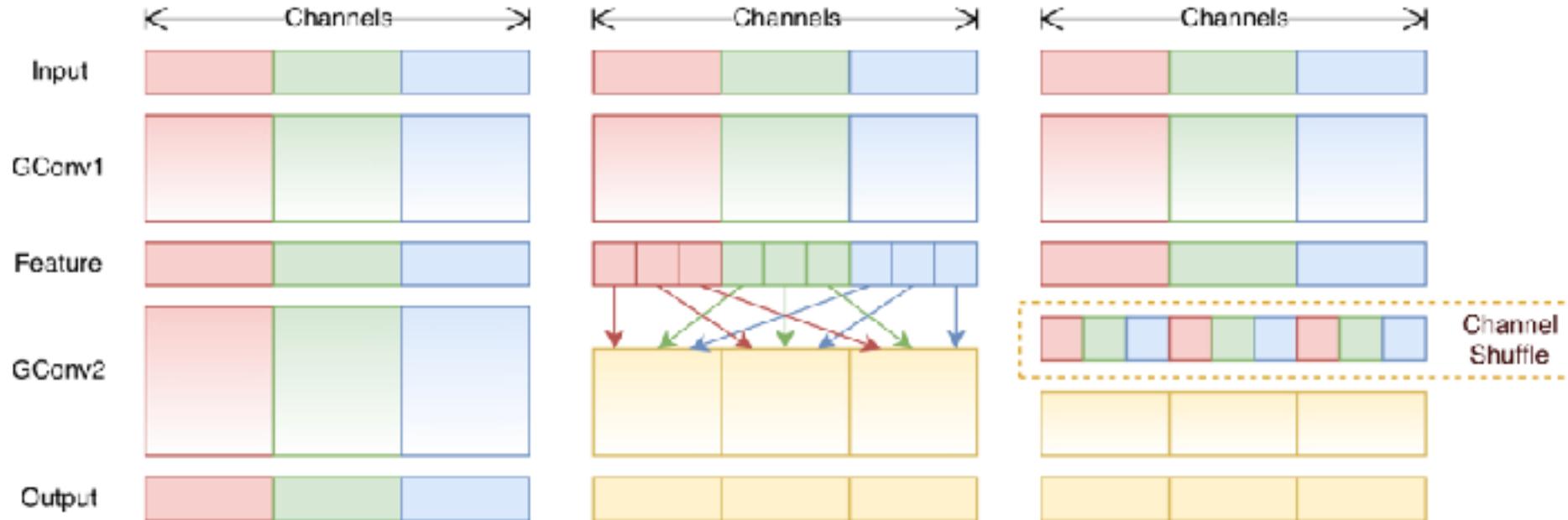
Separable Convolutions - all channels separate

- **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

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



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)**