Convolutional Neural Network

MLRS 2019

Rachel Hu

Amazon AI

Outline

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

aws

Image Classification

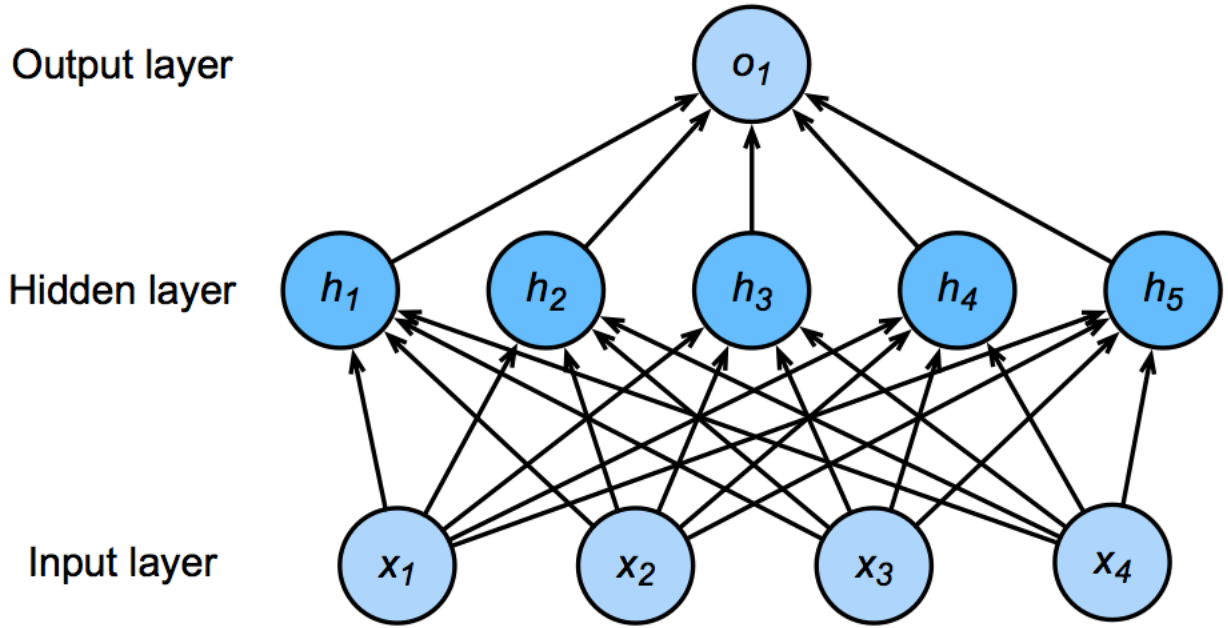
- Use a good camera
- RGB image has **36M** elements
- The model size of a single hidden layer MLP with a 100 neurons hidden size is **3.6B** parameters

aws

Flashback - Network with one hidden layer

100 neurons

36M features



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

Flashback - Network with one hidden layer

Output layer

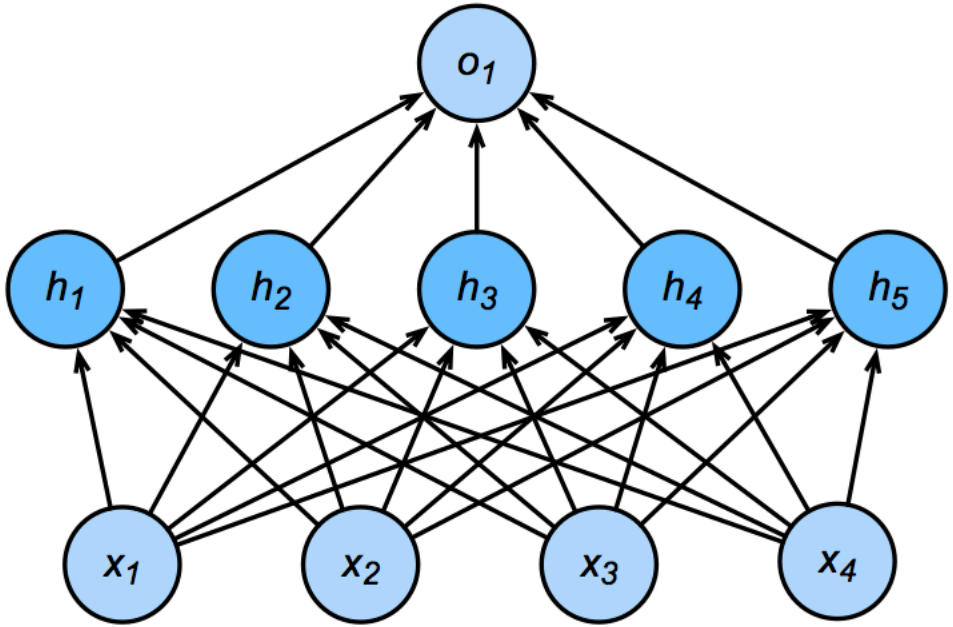
100 neurons

Hidden layer

3.6B parameters = 14GB

36M features

Input layer



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

aws


Can we reduce the number of the weights a bit?
numpy.d2l.ai
aws

Where is Waldo?

Two Principles

Two Principles

1. Translation Invariance:

Our vision systems should, in some sense, **respond similarly to the same object regardless of where it appears** in the image.

Two Principles

2. Locality:

Our visions systems should, in some sense, **focus on local regions**, without regard for what else is happening in the image at greater distances.

Rethinking Dense Layers

- Reshape the input, x and the hidden output, h into matrix (width, height)
- Reshape weights, w into 4-D tensors (h,w) to (h',w')

$$h_{i,j} = \sum_{k,l} w_{i,j,k,l} x_{k,l}$$

Rethinking Dense Layers

- Reshape the input, x and the hidden output, h into matrix (width, height)
- Reshape weights, w into 4-D tensors (h,w) to (h',w')

$$h_{i,j} = \sum_{k,l} w_{i,j,k,l} x_{k,l}$$

V is re-indexes W such as that

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

Rethinking Dense Layers

- Reshape the input, x and the hidden output, h into matrix (width, height)
- Reshape weights, w 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* should simply lead to a shift in *h*

Idea #1 - Translation Invariance

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

- A shift in *x* should simply lead to a shift in *h*
- However, in the Waldo case, we did not see a lot change in the output results. Hence, v should not depend on (i, j).

Idea #1 - Translation Invariance

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

- A shift in *x* should simply lead to a shift in *h*
- However, in the Waldo case, we did not see a lot change in the output results. Hence, v should not depend on (i, j).

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

Idea #1 - Translation Invariance

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

Idea #1 - Translation Invariance

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

- $v_{i,j,a,b} = v_{a,b}$

Idea #1 - Translation Invariance

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

- $v_{i,j,a,b} = v_{a,b}$

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

Idea #1 - Translation Invariance

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

- $v_{i,j,a,b} = v_{a,b}$

Reduced the dim by 36 million times smaller

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

Idea #2 - Locality

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

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)

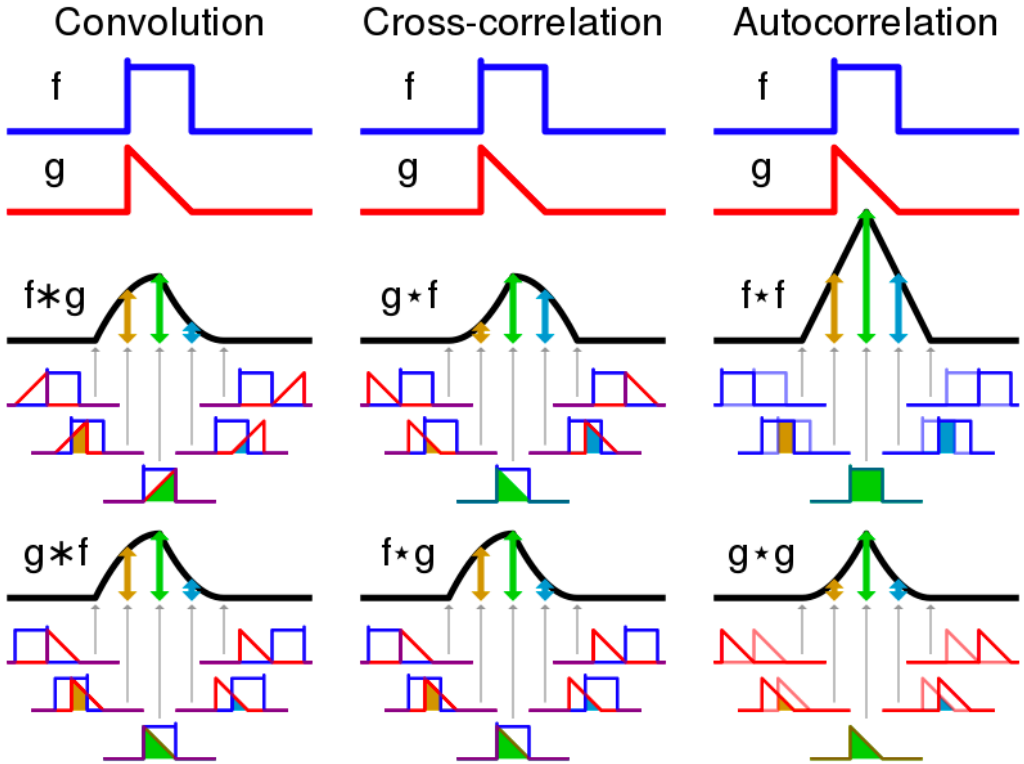
Idea #2 - Locality

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

- 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 $v_{a,b} = 0$

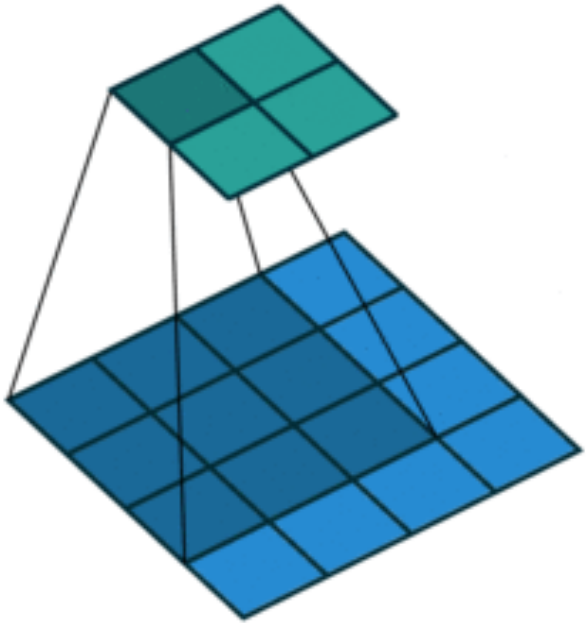
$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$

Convolution



2-D Cross Correlation

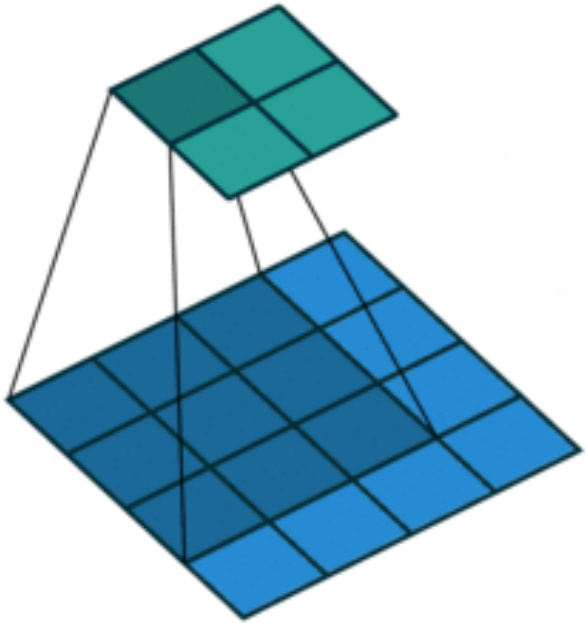
$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$



(vdumoulin@ Github)

2-D Cross Correlation

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

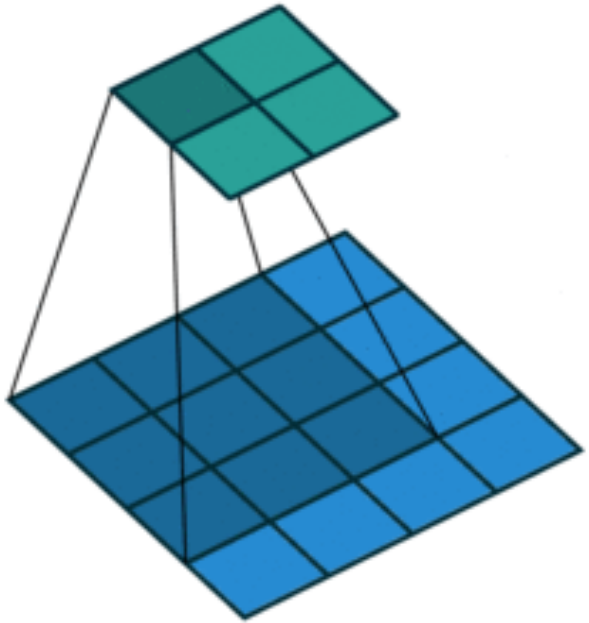


(vdumoulin@ Github)

2-D Cross Correlation

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

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

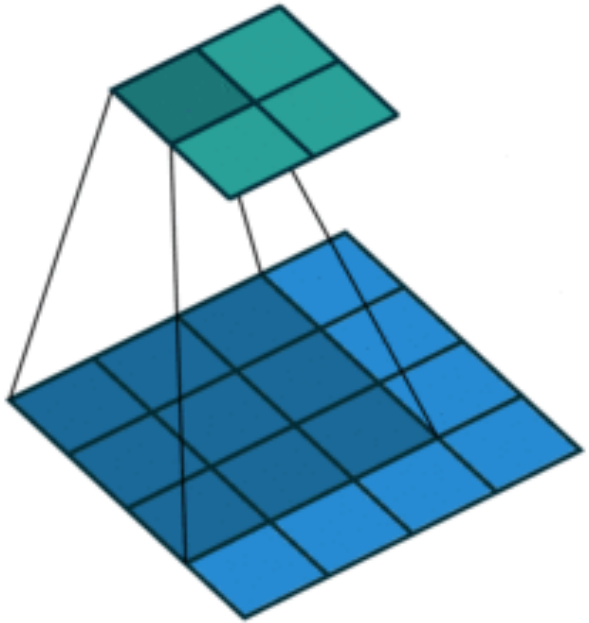


(vdumoulin@ Github)

2-D Cross Correlation

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

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



(vdumoulin@ Github)

2-D Convolution Layer

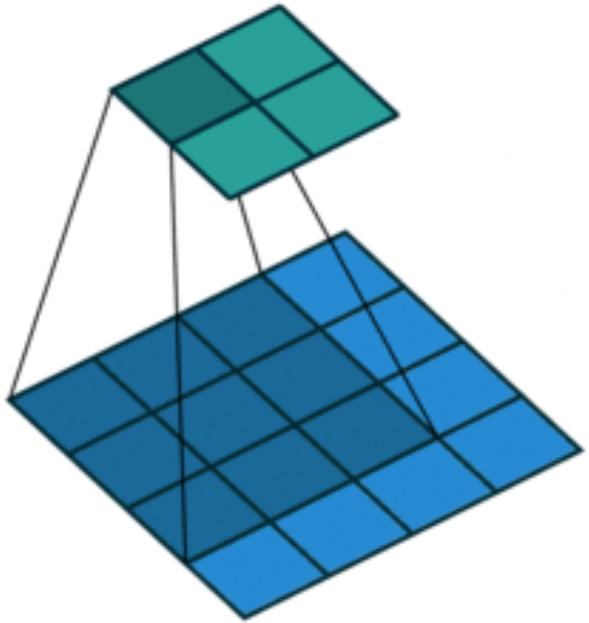
0	1	2
3	4	5
6	7	8

$*$

0	1
2	3

$=$

19	25
37	43



(vdumoulin@ Github)

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

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

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

2-D Convolution Layer

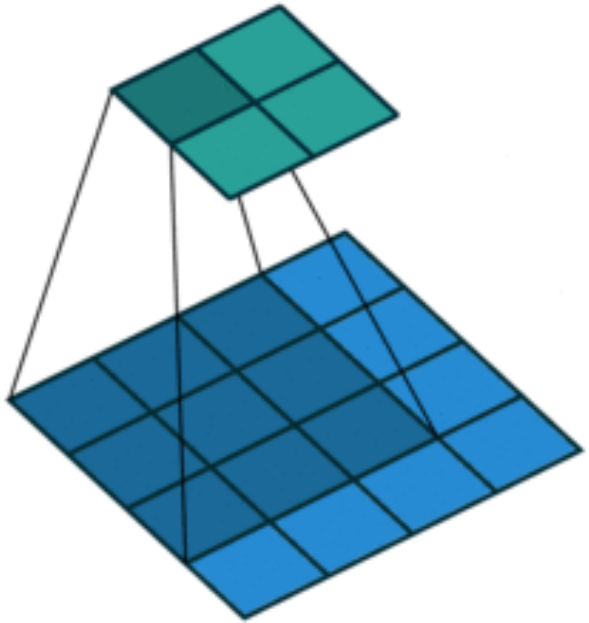
0	1	2
3	4	5
6	7	8

$*$

0	1
2	3

$=$

19	25
37	43



(vdumoulin@ Github)

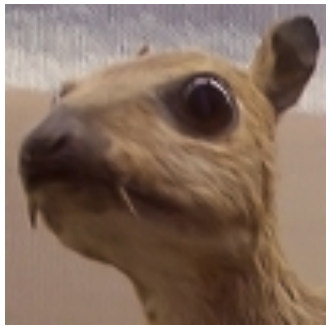
- $\mathbf{X}: n_h \times n_w$ input matrix
- $\mathbf{W}: k_h \times k_w$ kernel matrix
- b: scalar bias
- $\mathbf{Y}: (n_h - k_h + 1) \times (n_w - k_w + 1)$ output r

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

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

Examples

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



(wikipedia)

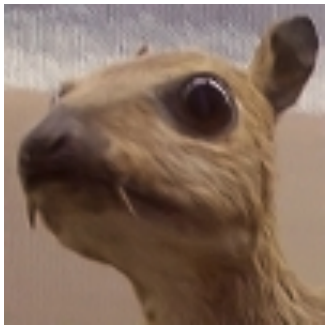
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

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

Examples

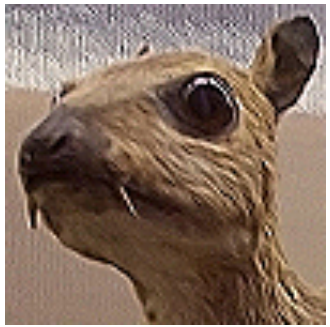
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection



(wikipedia)

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

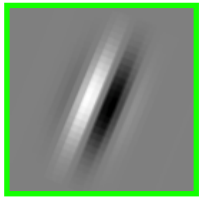


Sharpen

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

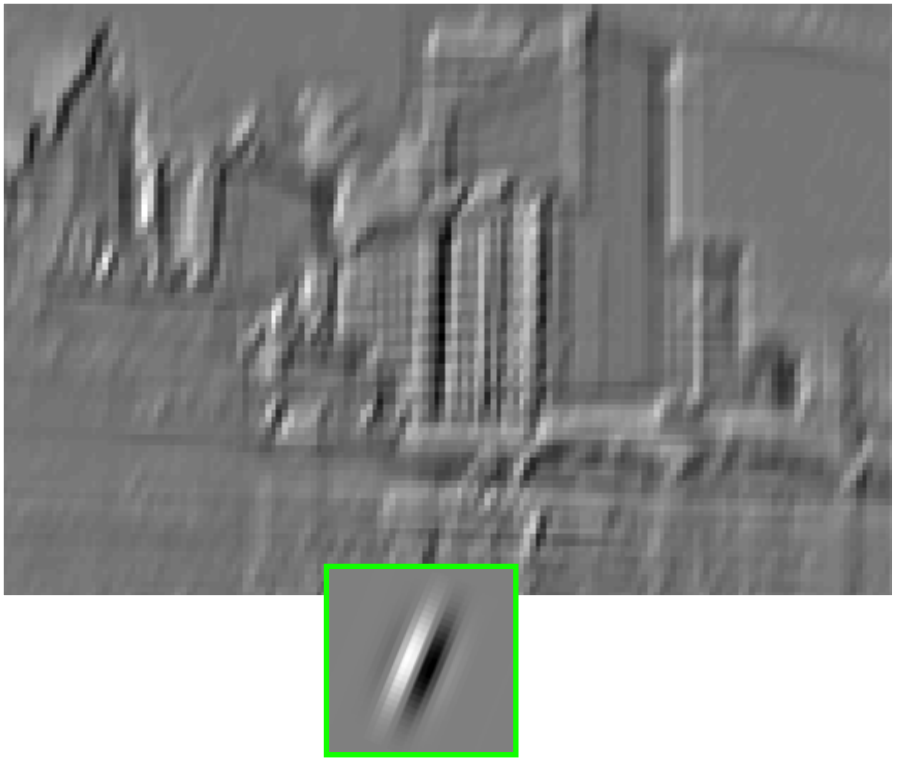
Gaussian Blur

Examples



(Rob Fergus)

Examples



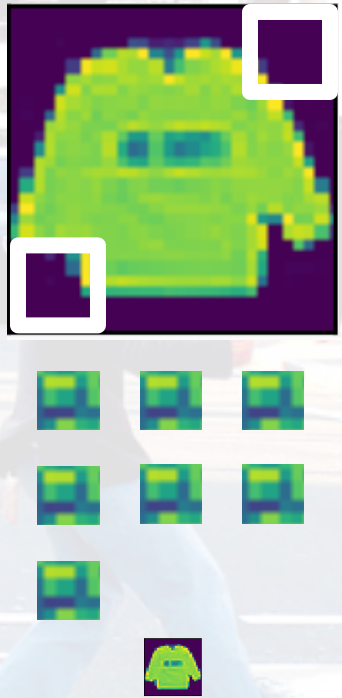
(Rob Fergus)

Convolutions Notebook


Padding and Stride

The Issue

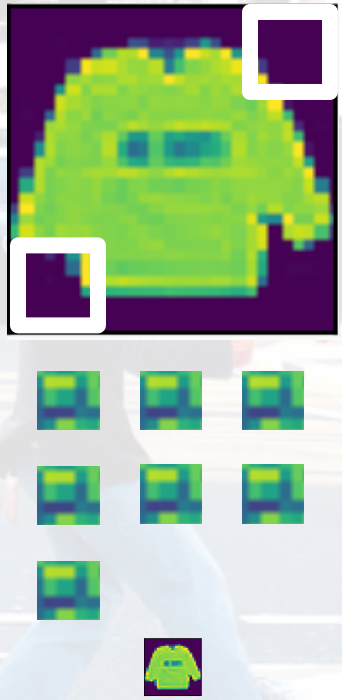
- Given a 32 x 32 input image
- Apply convolutional layer with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers



The Issue

- 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

Input

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

*

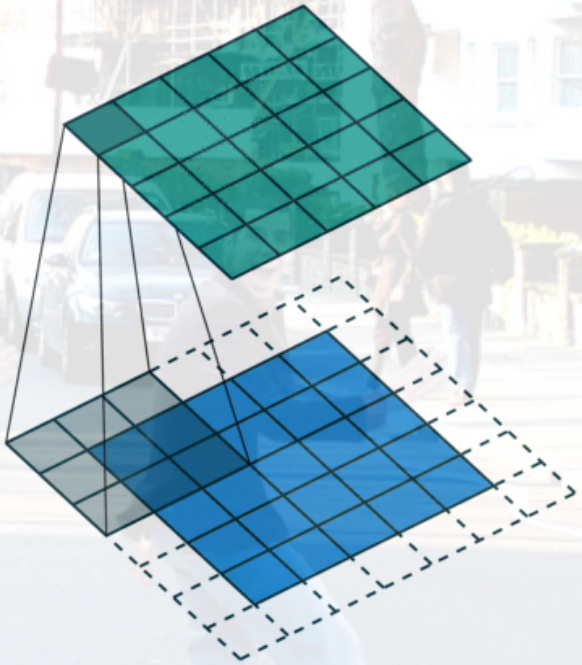
Kernel

0	1
2	3

=

Output

0	3	8	4
9	19	25	10
21	37	43	16
6	7	8	0



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

Padding

Padding adds rows/columns around input

Input

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

*

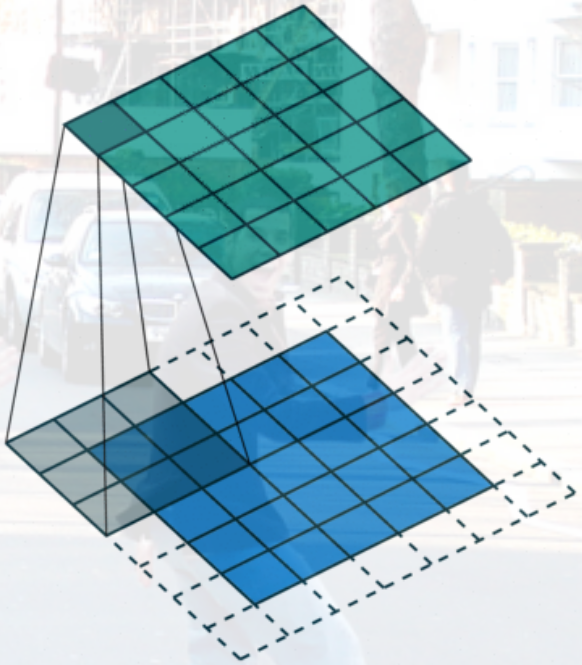
Kernel

0	1
2	3

=

Output

0	3	8	4
9	19	25	10
21	37	43	16
6	7	8	0



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

Padding

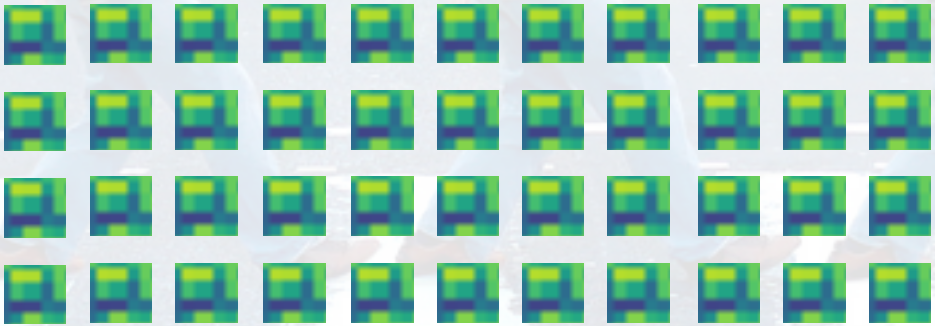
- Padding p_h rows and p_w columns, output shape will be

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

- 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

The Issue

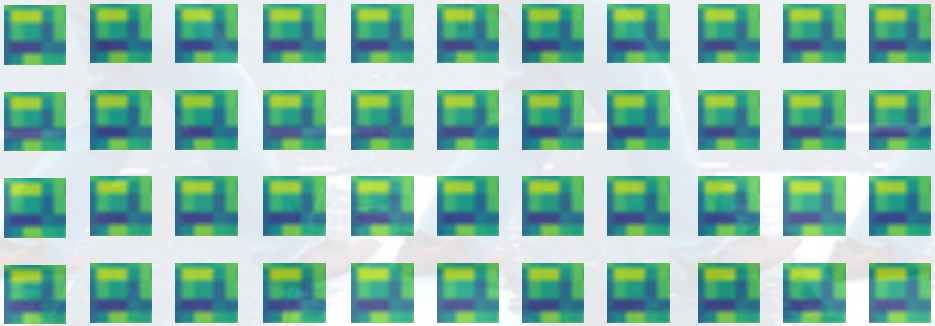
Too SLOW !



aws

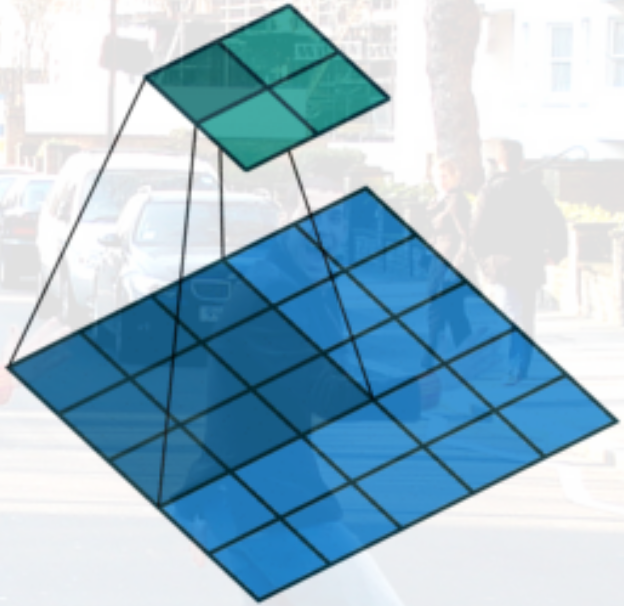
The Issue

- 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

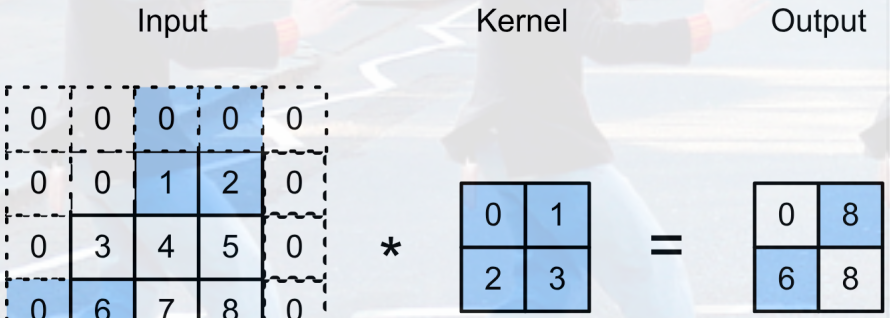


Stride

- Stride is the #rows/#columns per slide



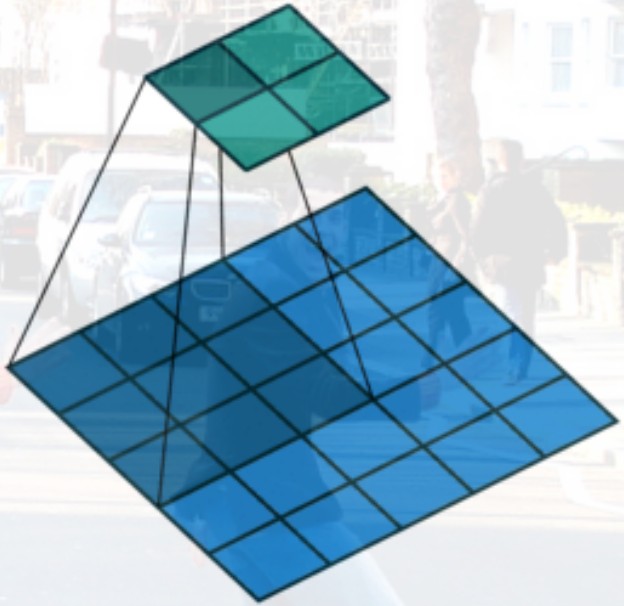
Strides of 3 for height and 2 for width



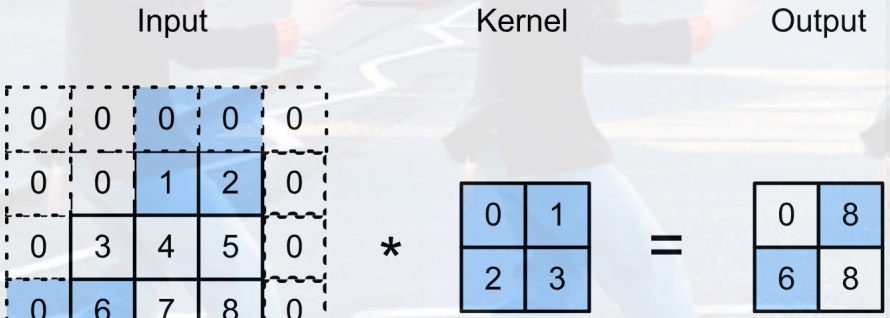
$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$

Stride

- Stride is the #rows/#columns per slide



Strides of 3 for height and 2 for 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

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

Stride

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

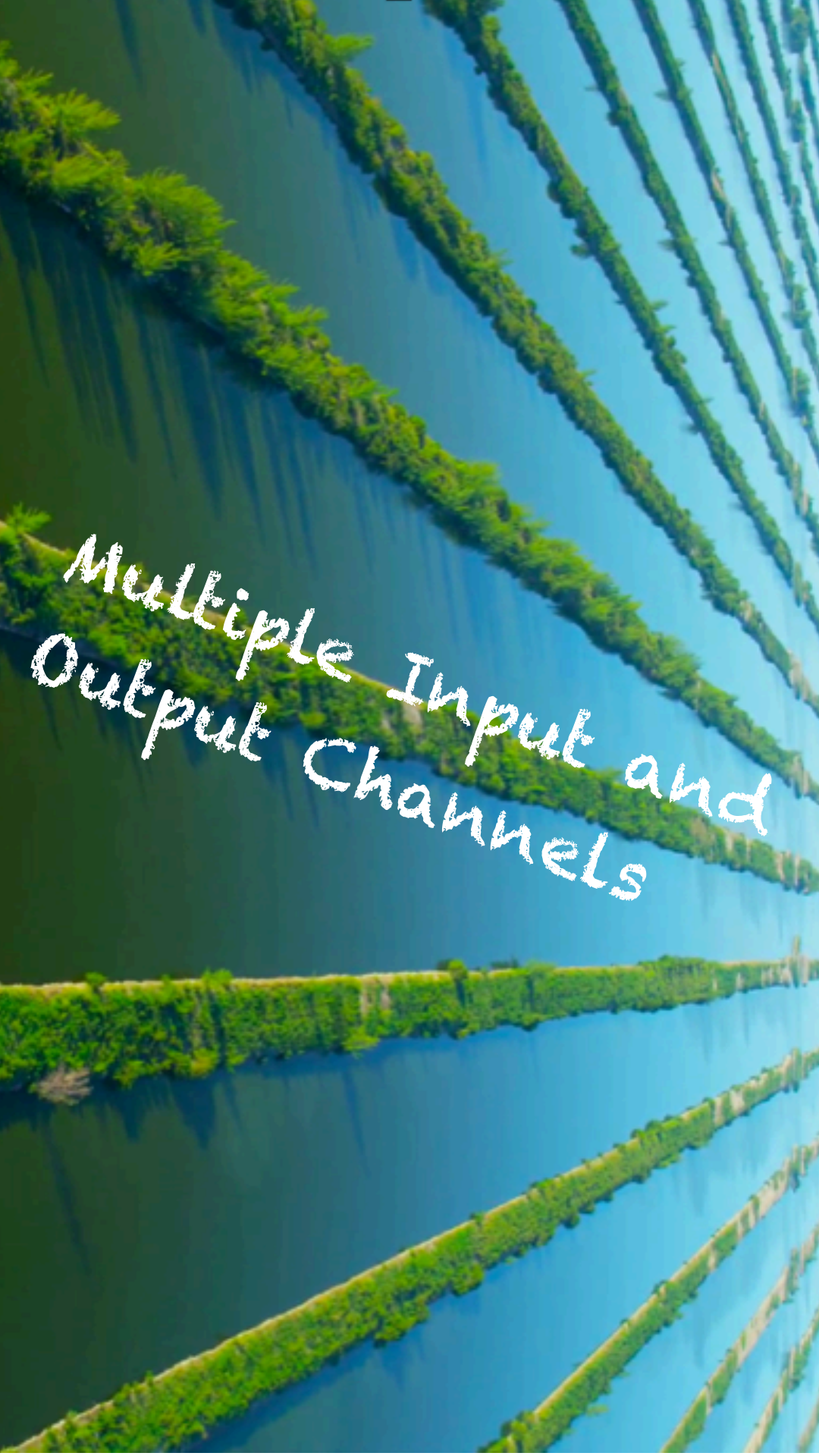
$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

- 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

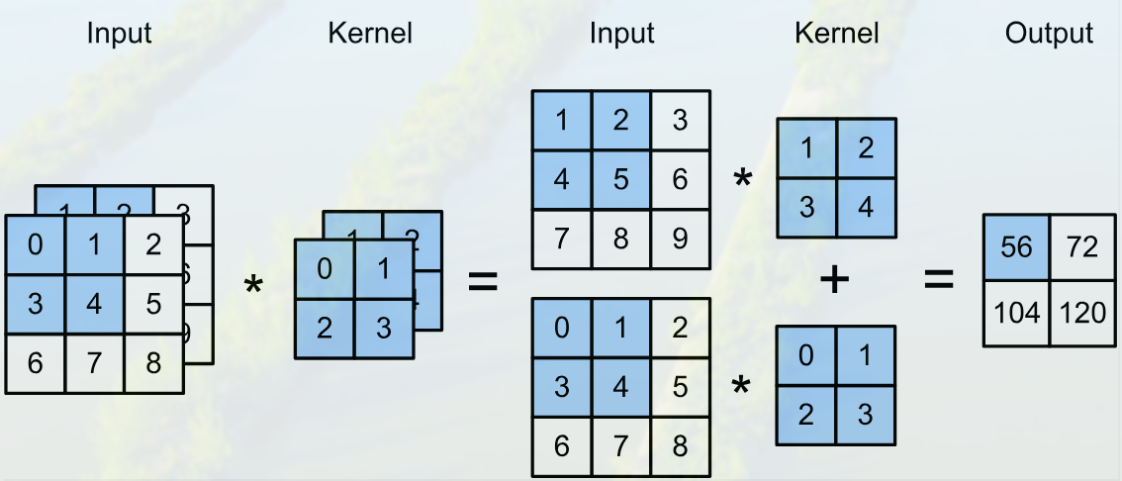
A lot of really interesting and challenging details back in 1970s…

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



$$(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4)$$
$$+(0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3)$$
$$= 56$$

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 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

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

So far we always get single output channel, no matter how many inputs channels

Multiple Input & Output 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

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

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

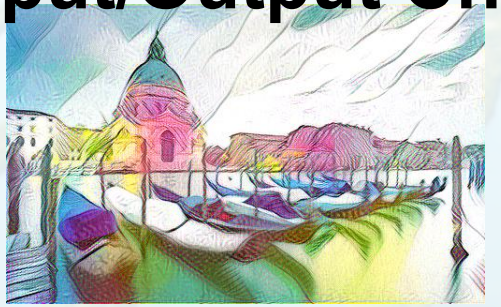
$$\mathbf{Y}_{j,:,:} = \mathbf{X} \star \mathbf{W}_{j,:,:,:}$$

for $j = 1, \ldots, c_o$

Why Multiple Input/Output Channels?

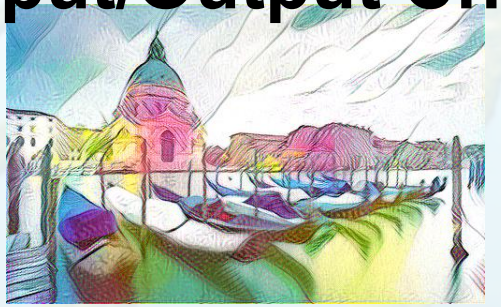
https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/

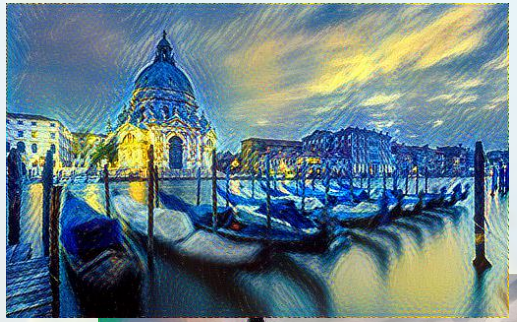
Why Multiple Input/Output Channels?



https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/

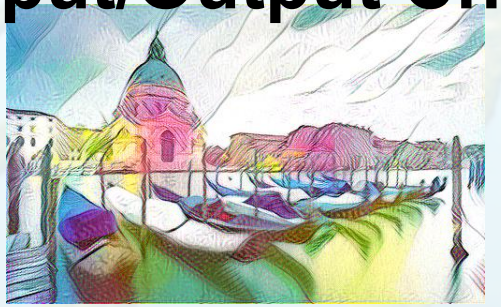
Why Multiple Input/Output Channels?





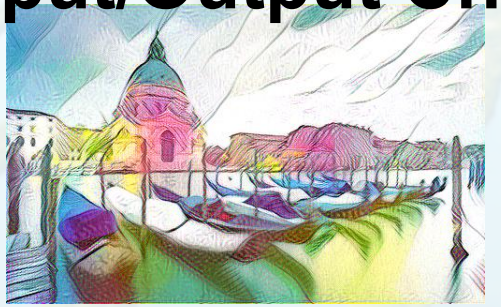
https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/

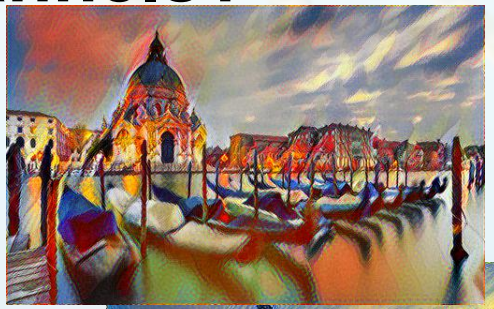
Why Multiple Input/Output Channels?

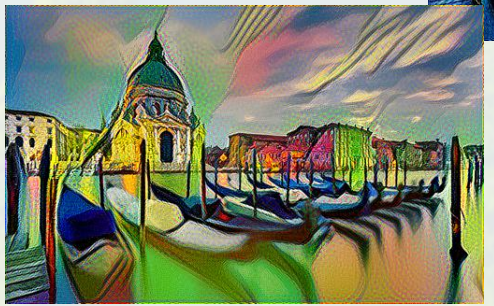


https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/

Why Multiple Input/Output Channels?

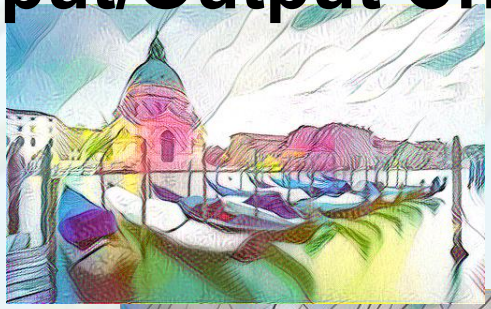


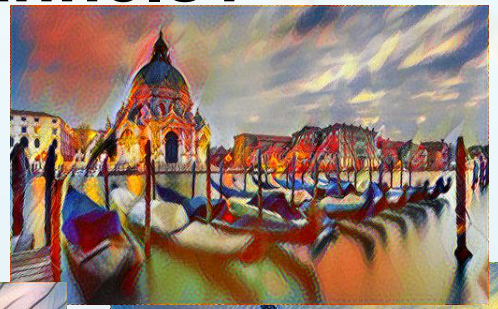




https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/

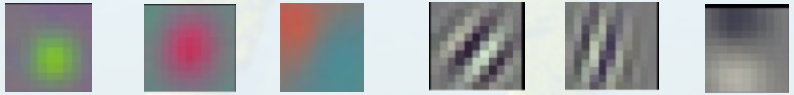
Why Multiple Input/Output Channels?





https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/

Why Multiple Input/Output Channels?



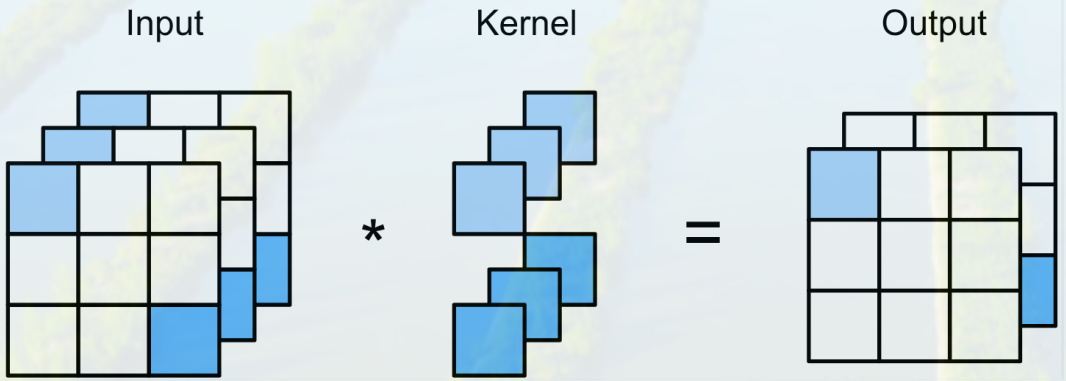
Why Multiple Input/Output Channels?

- Each input channel may recognize a particular pattern in inputs

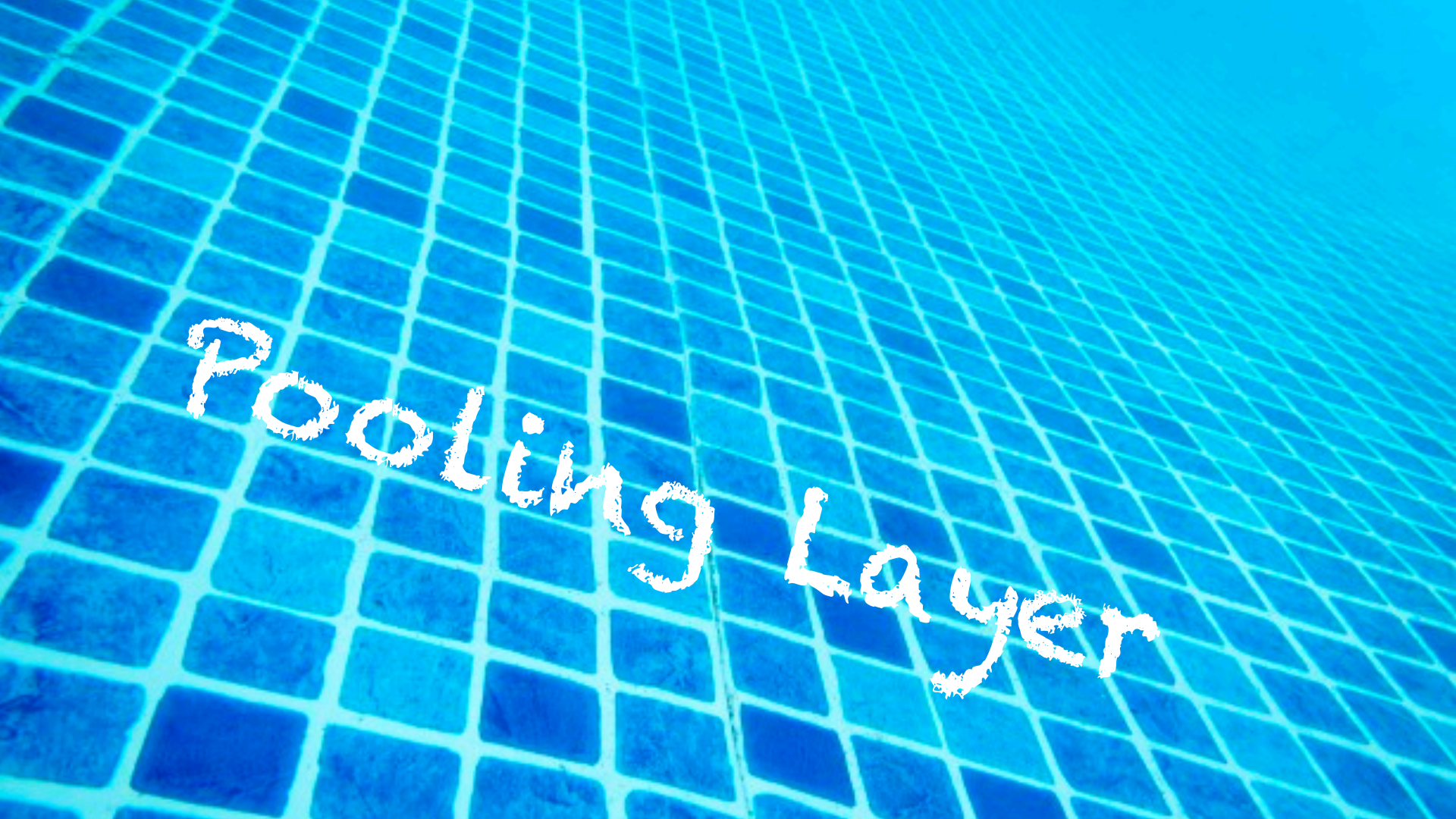
- Each output channel recognize and combines patterns to different outputs

1 x 1 Convolutional Layer

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

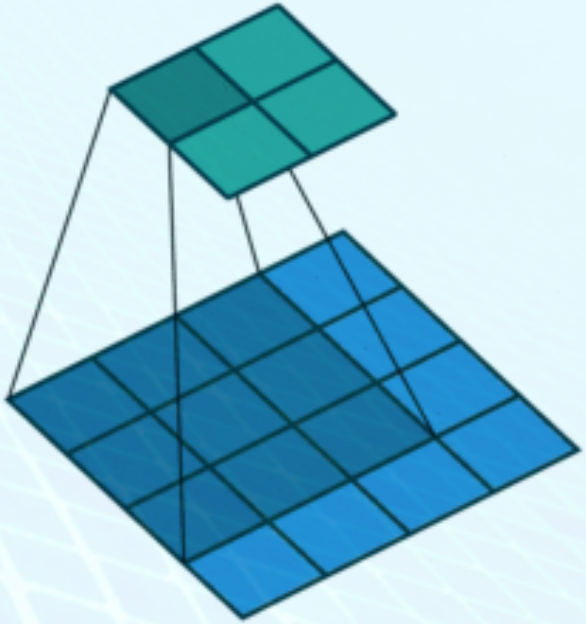


Equal to a dense layer with $n_h n_w \times c_i$ input and $c_o \times c_i$ weight.


Pooling Layer

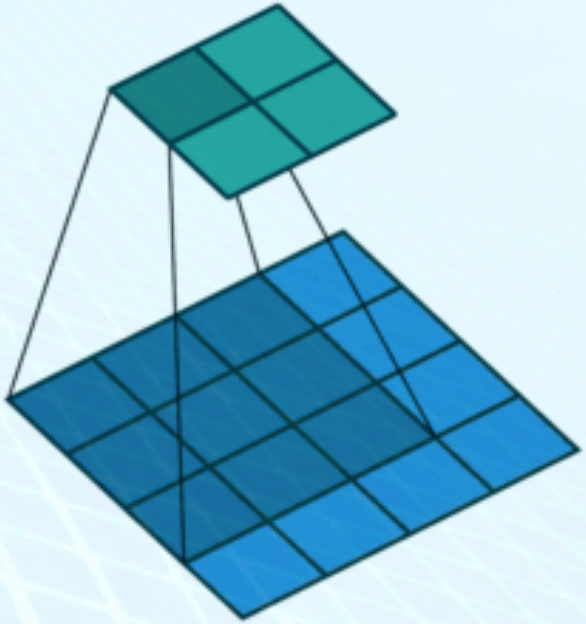
2-D Max Pooling

Like convolutional layers, pooling layers consist of a fixed-shape window that slides over all regions in the input according to its stride.



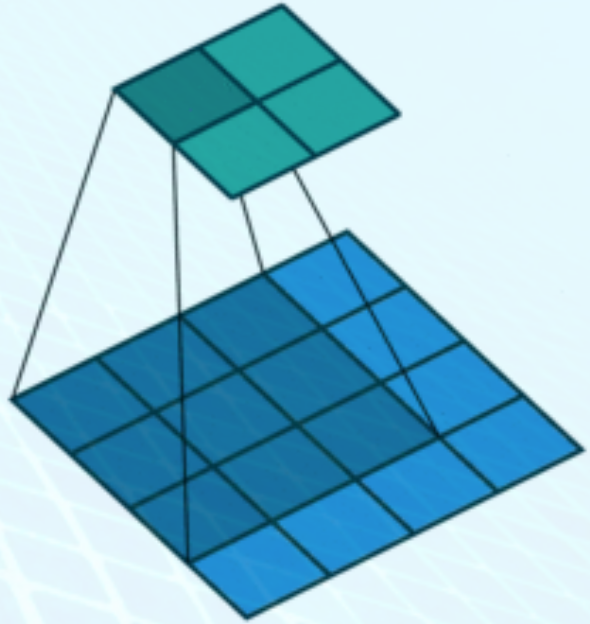
2-D Max Pooling

Like convolutional layers, pooling layers consist of a fixed-shape window that slides over all regions in the input according to its stride.



2-D Max Pooling

- Returns the maximal value in the pooling window



Input

0	1	2
3	4	5
6	7	8

2 x 2 Max Pooling

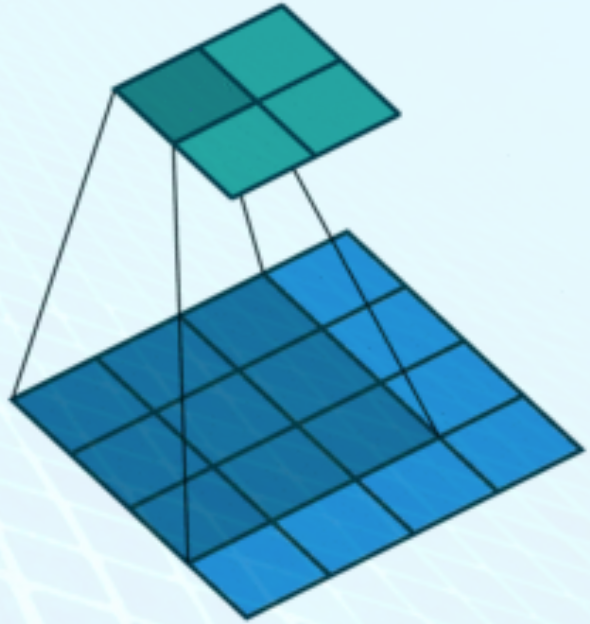
Output

4	5
7	8

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

2-D Max Pooling

- Returns the maximal value in the pooling window



Input

0	1	2
3	4	5
6	7	8

2 x 2 Max Pooling

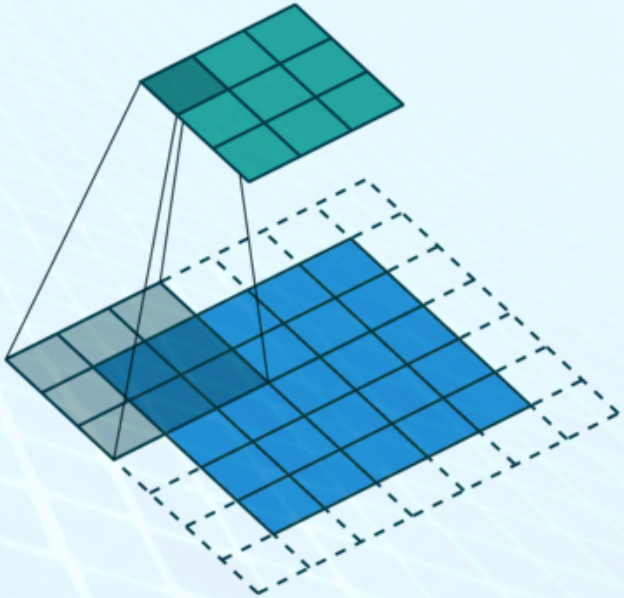
Output

4	5
7	8

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

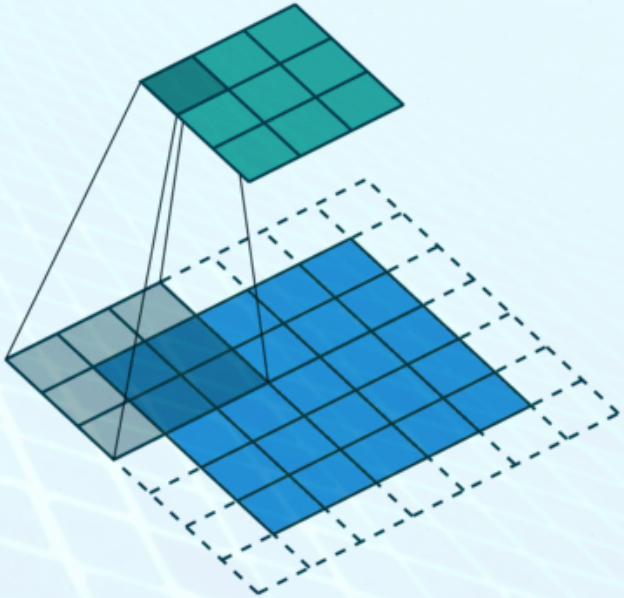
Padding, Stride, and Multiple Channels

- No learnable parameters
- Pooling layers can apply similar padding and stride as convolutional layers
- Apply pooling for each input channel to obtain the corresponding output channel
  #output channels = #input channels



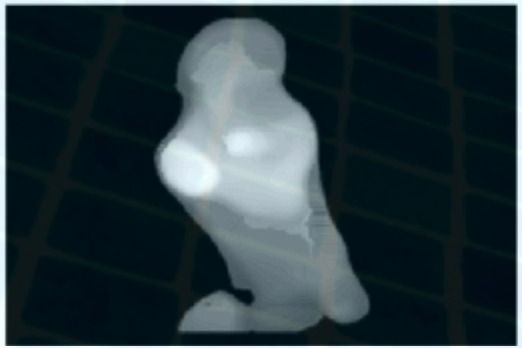
Padding, Stride, and Multiple Channels

- No learnable parameters
- Pooling layers can apply similar padding and stride as convolutional layers
- Apply pooling for each input channel to obtain the corresponding output channel
  #output channels = #input channels



Average Pooling

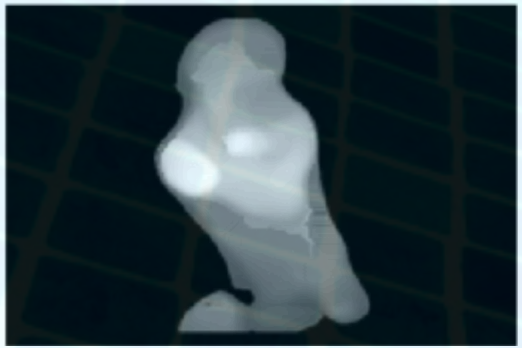
Max pooling



Average pooling

Average Pooling

Max pooling

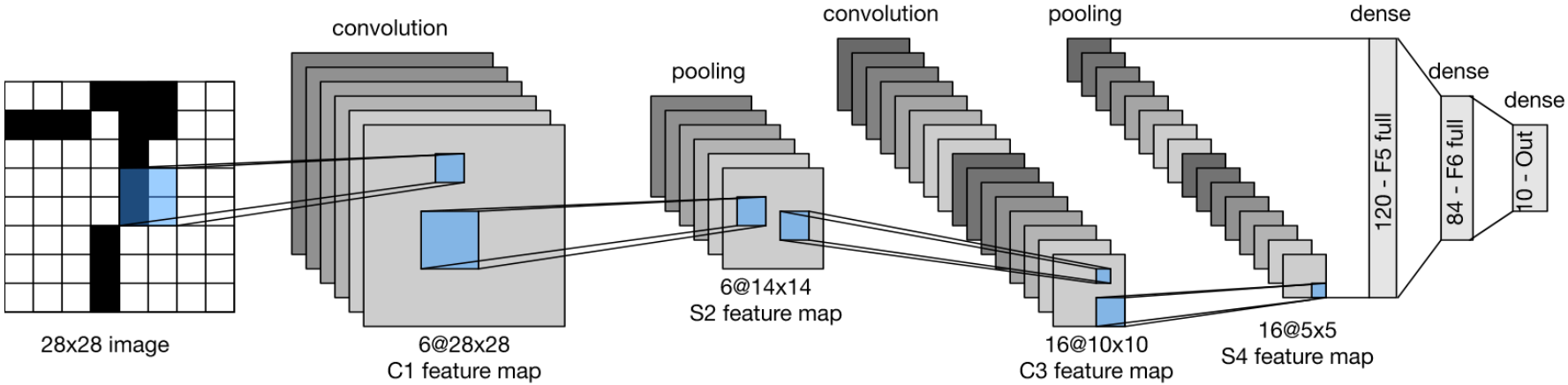


Average pooling

- 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

Pooling Notebook

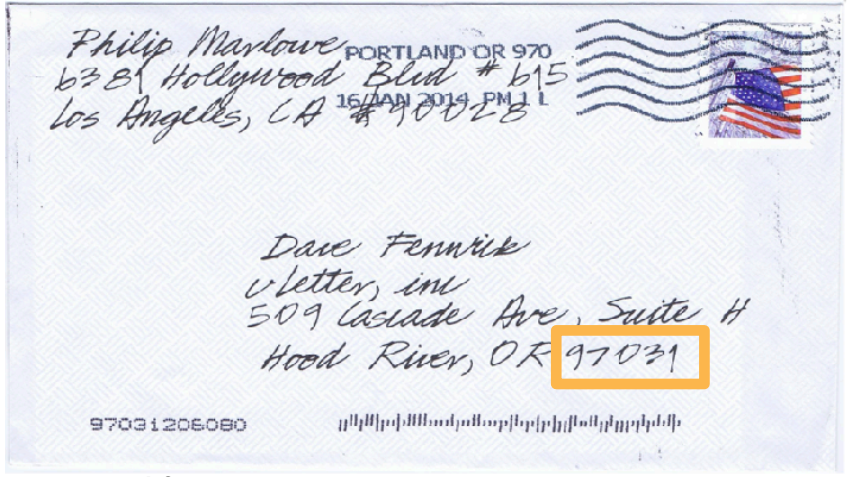
LeNet

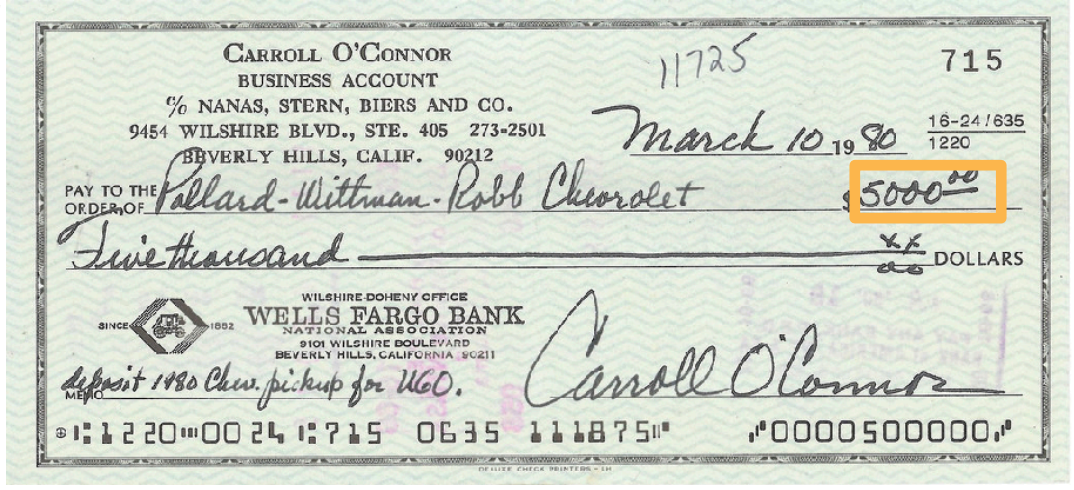


http://yann.lecun.com/exdb/lenet/

Why Handwritten Digit Recognition in 1990s?

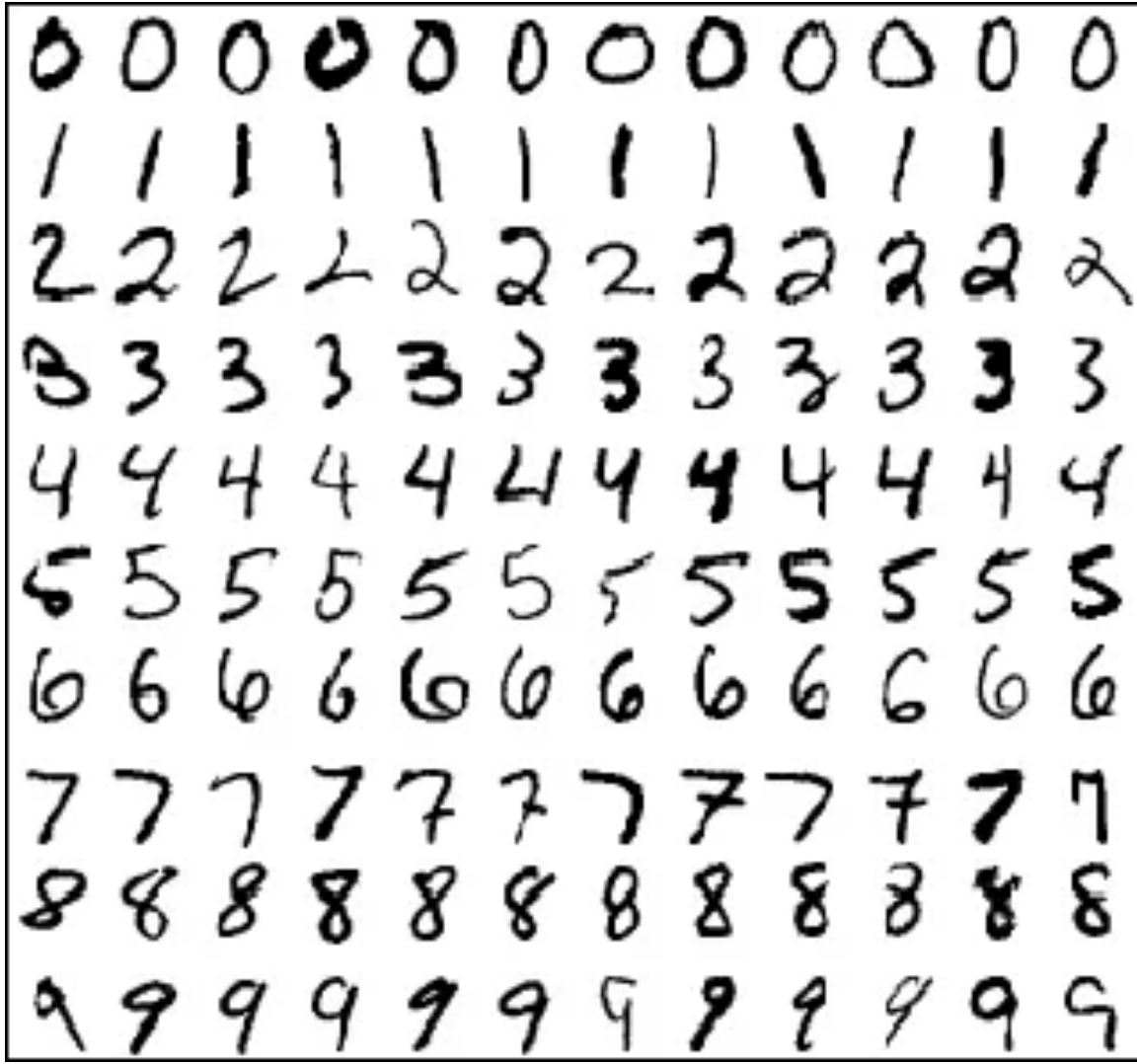
AT&T had a project in order to recognize handwritten characters for postal codes on letters and also recognize the dollar amounts on checks.





MNIST

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



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

aws

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

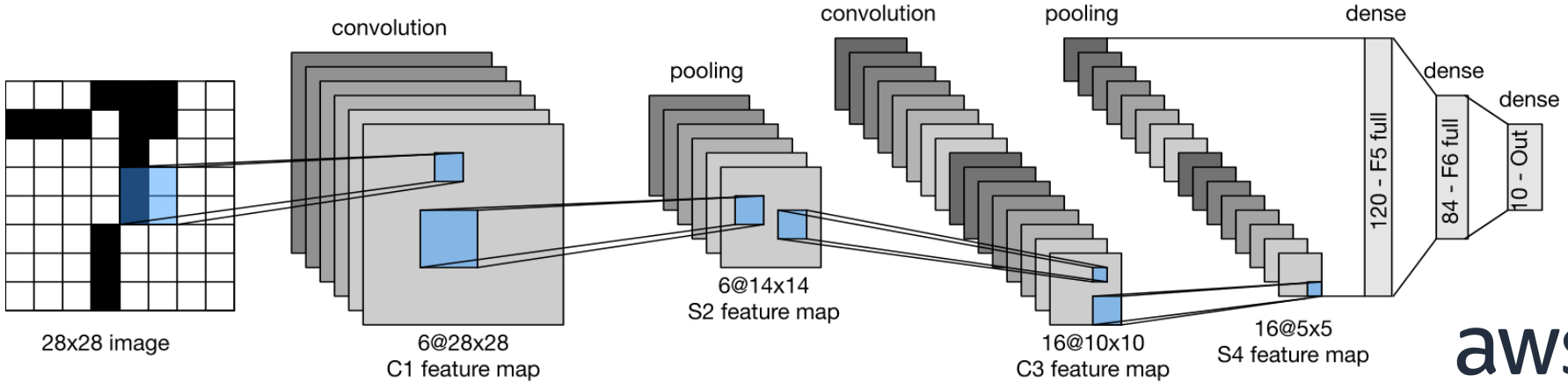
aws

LeNet

LeNet consists of two parts:

Part I. Convolution Block

- Convolution layer - To recognize the spatial patterns
  - 5×5 kernel
  - sigmoid activation function
- Average pooling layer - To reduce the dimensionality

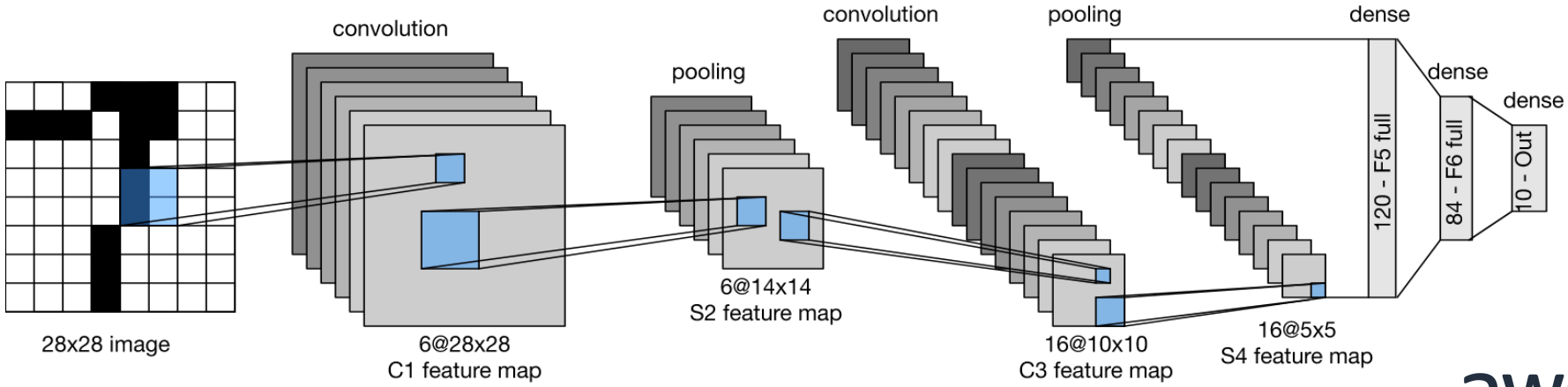


LeNet

LeNet consists of two parts:

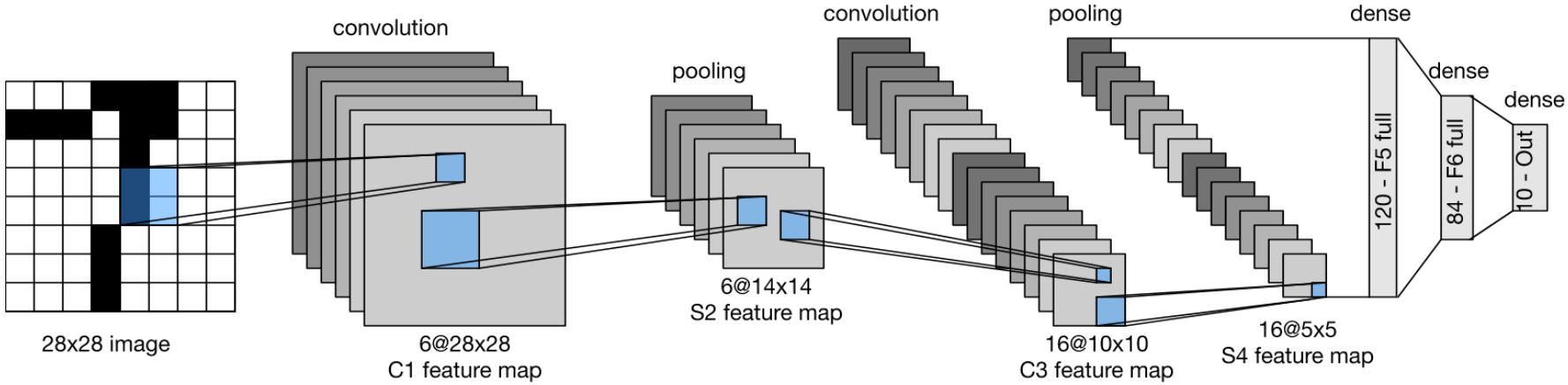
Part II. Fully-connected layers Block

- 3 fully-connected layers
  - with 120, 84, and 10 outputs, respectively



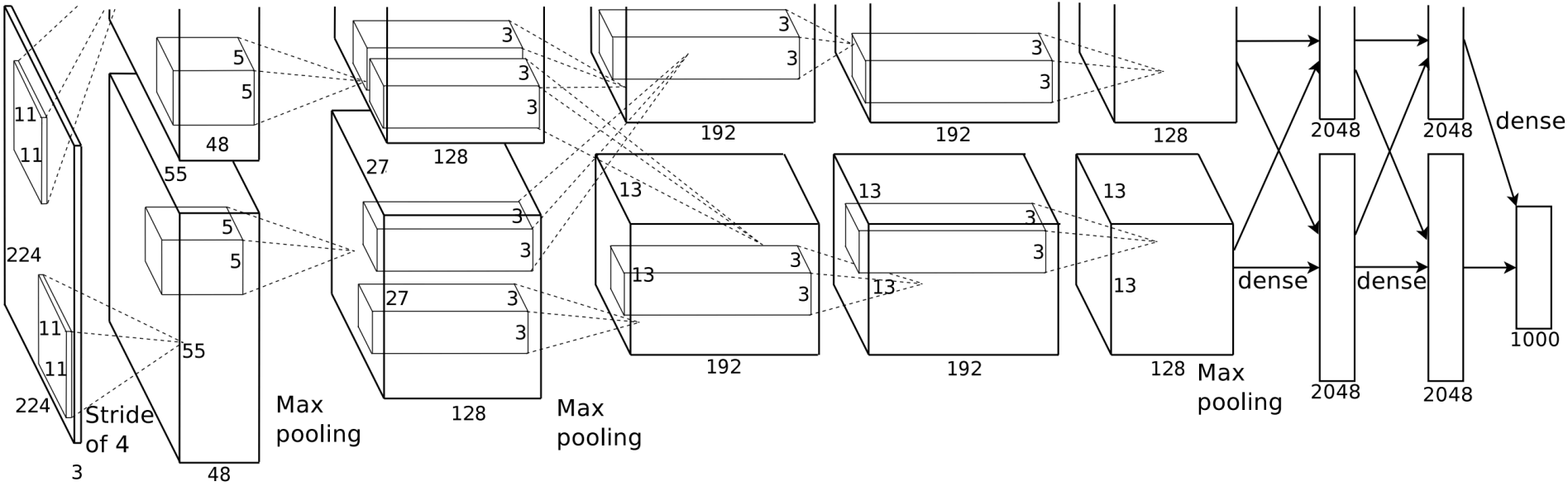
LeNet

Expensive if we have many outputs



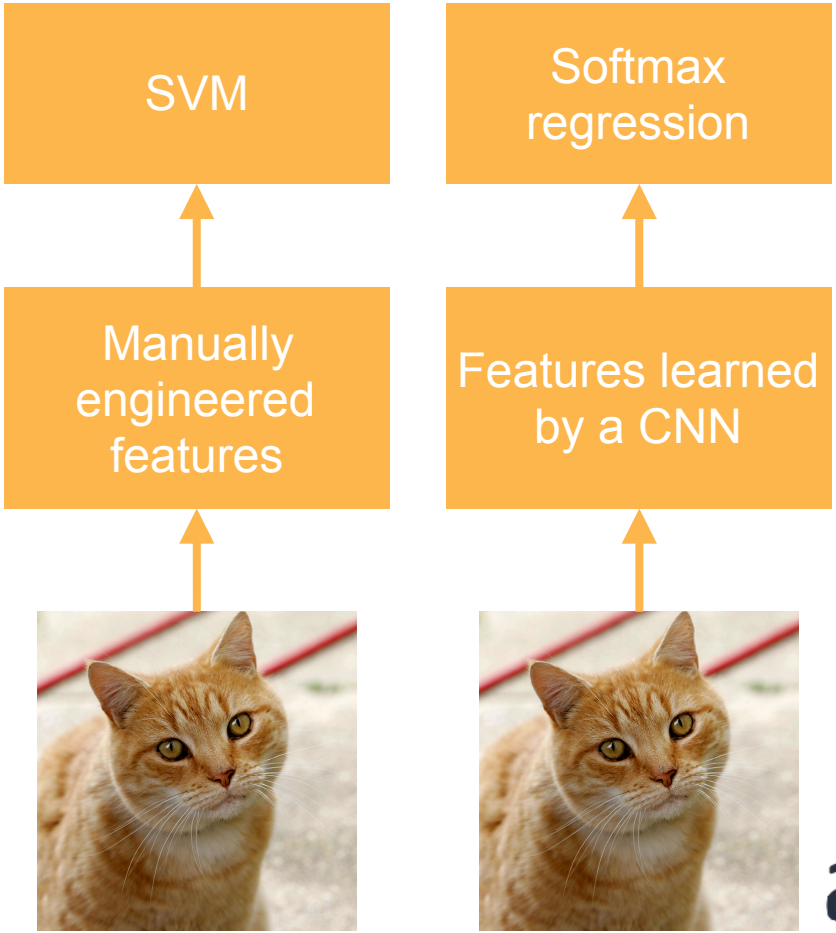
LeNet Notebook

AlexNet

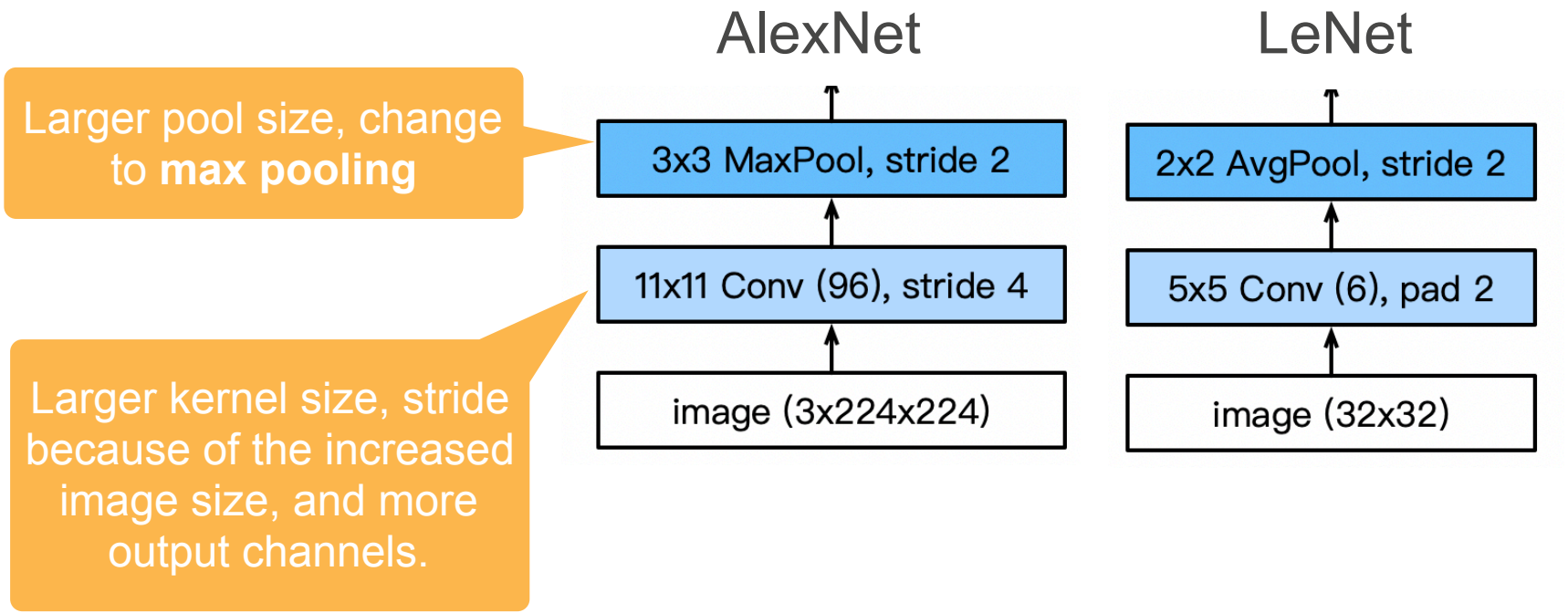

224
11
11
224
3
Stride
of 4
55
48
5
5
48
55
5
5
Max
pooling
27
128
3
3
3
128
27
3
3
3
Max
pooling
13
192
3
3
13
3
3
13
192
3
3
13
192
3
3
13
128
Max
pooling
dense
dense
2048
2048
2048
2048
dense
1000

AlexNet

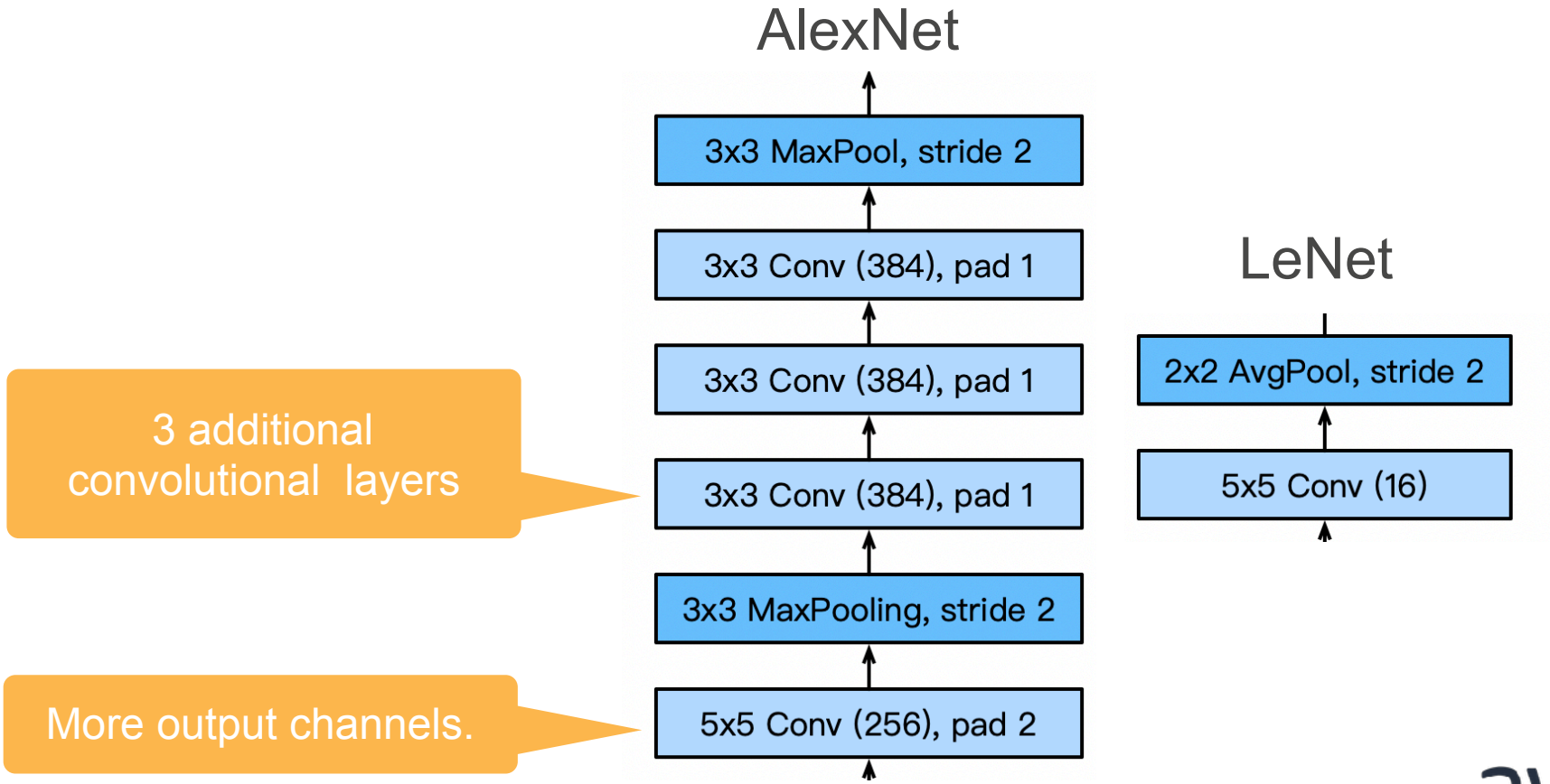
- AlexNet won ImageNet competition in 2012
- Deeper and bigger LeNet
- Key modifications
  - Dropout (regularization)
  - ReLu (training)
  - MaxPooling
- Paradigm shift for computer vision



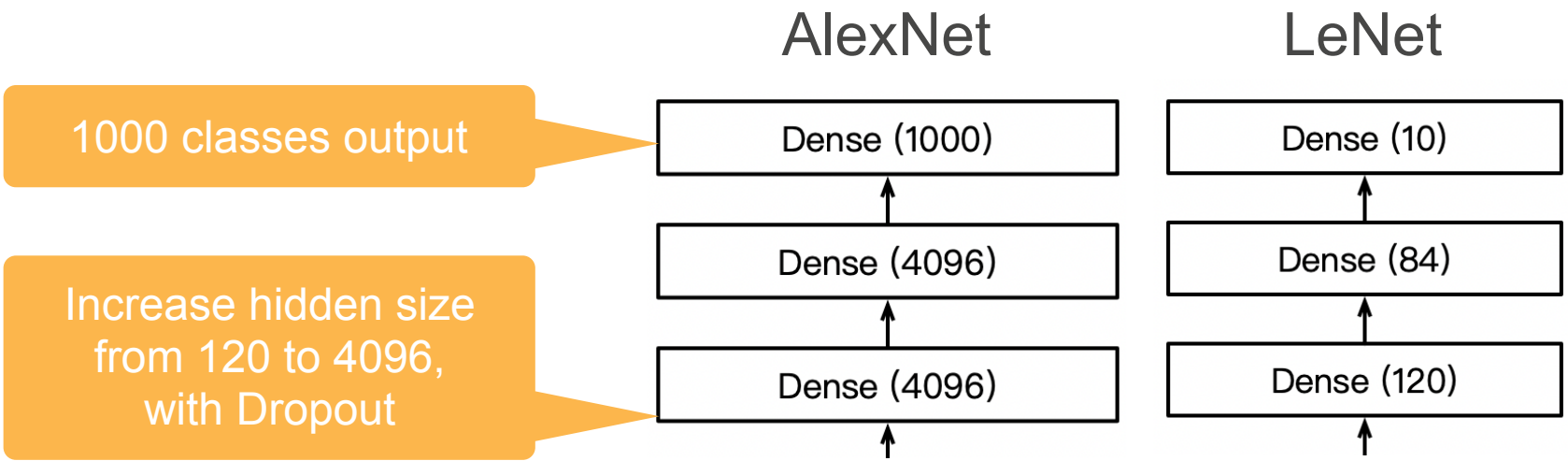
AlexNet Architecture



AlexNet Architecture

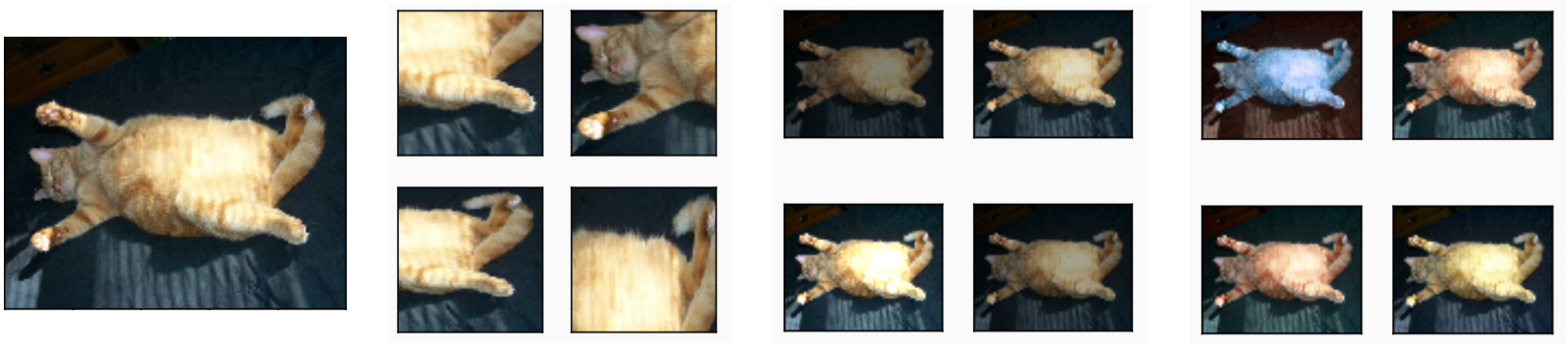


AlexNet Architecture



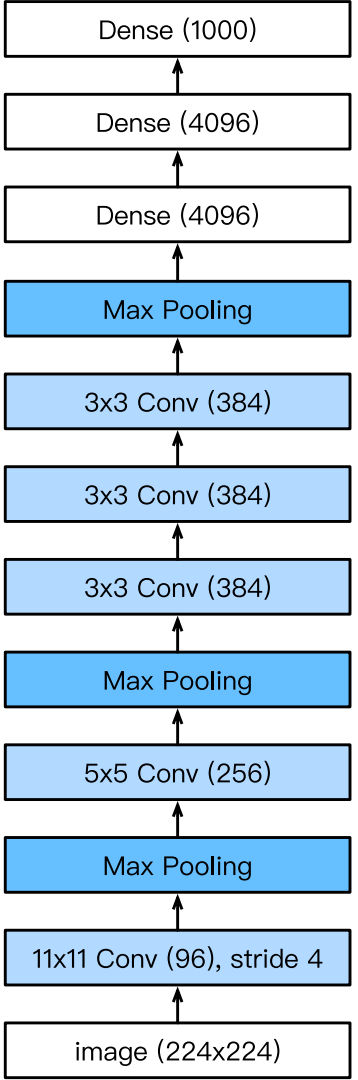
More Tricks

- 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



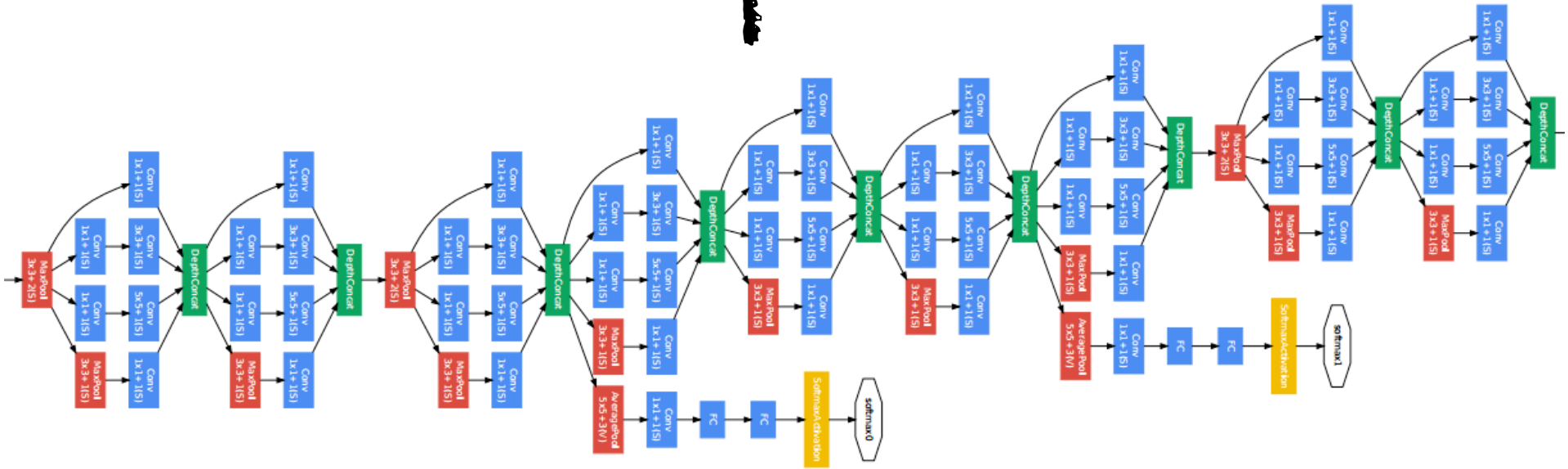
Complexity

	#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

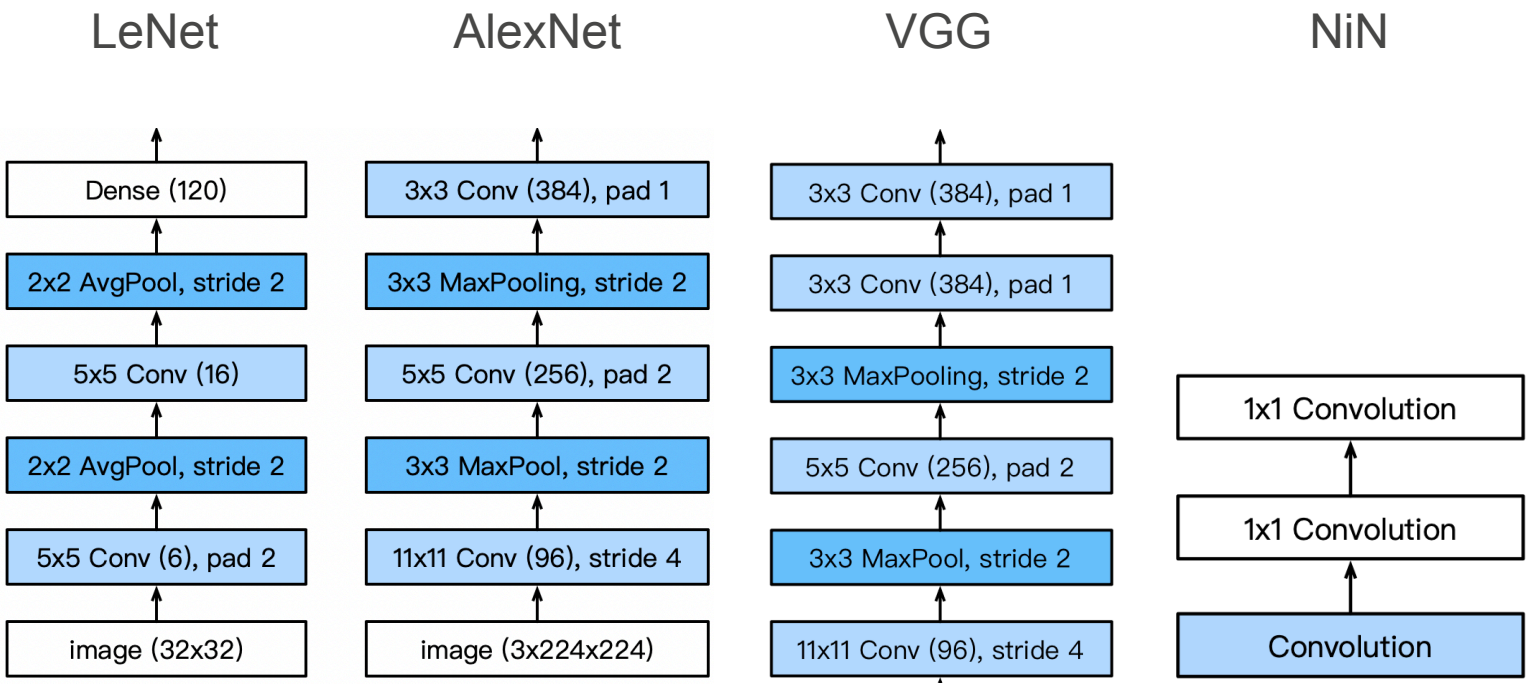


AlexNet Notebook

Inception


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

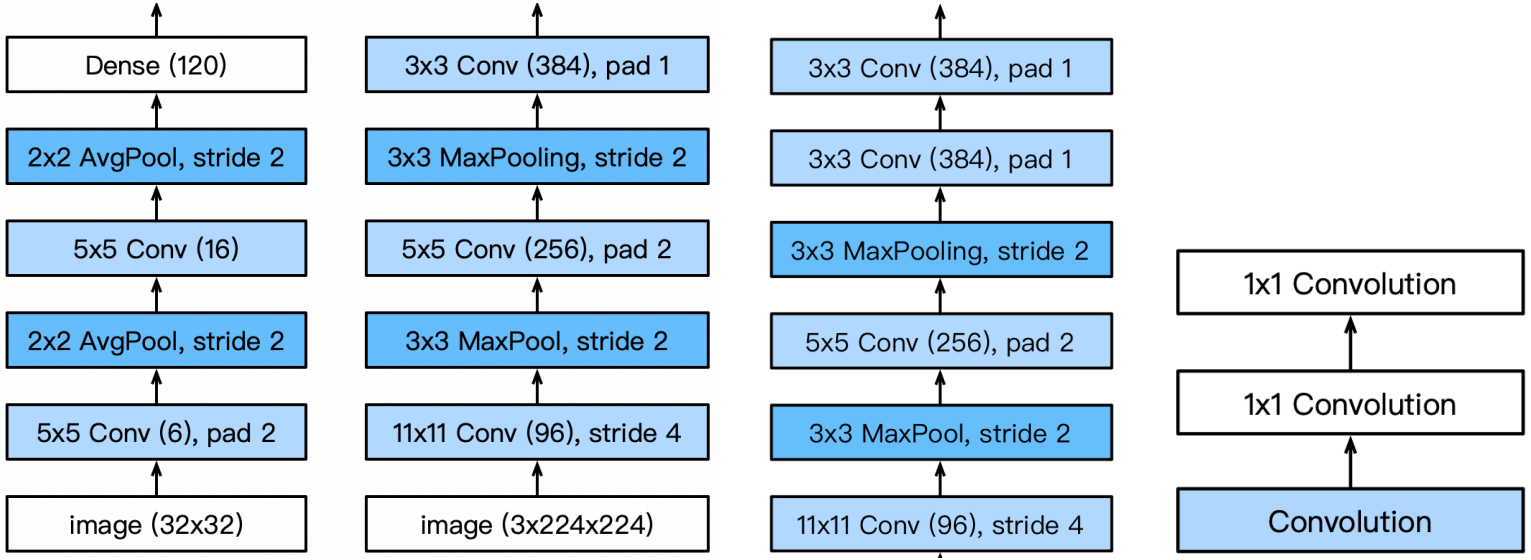
Picking the best convolution …



Picking the best convolution …

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

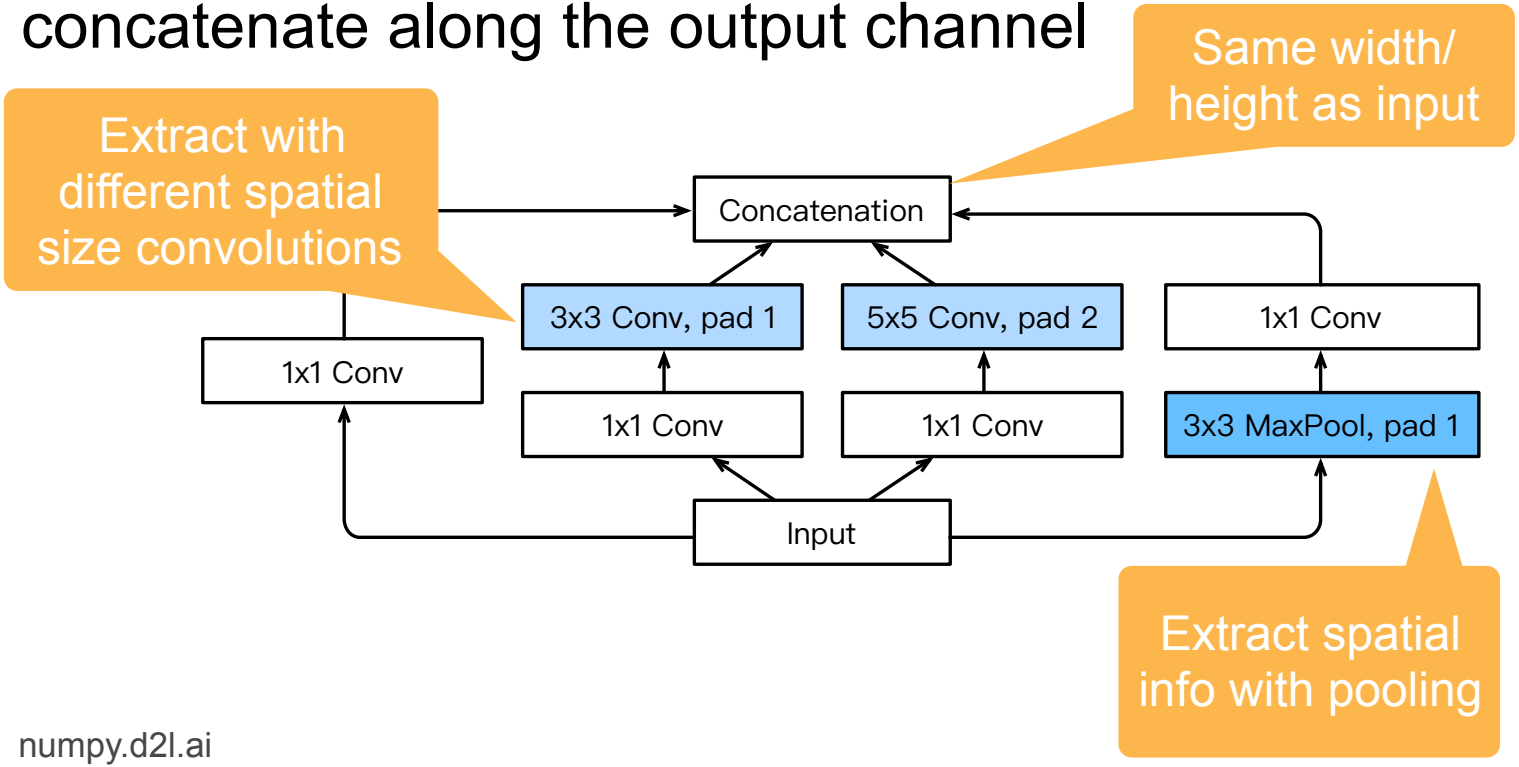
LeNet | AlexNet | VGG | NiN



Why choose? Just pick them all.

Inception Blocks

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

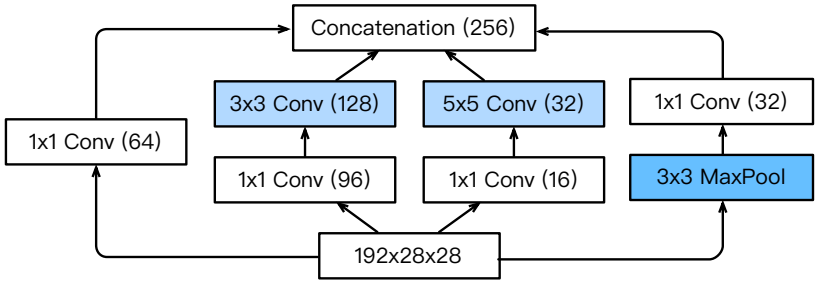


Inception Blocks

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

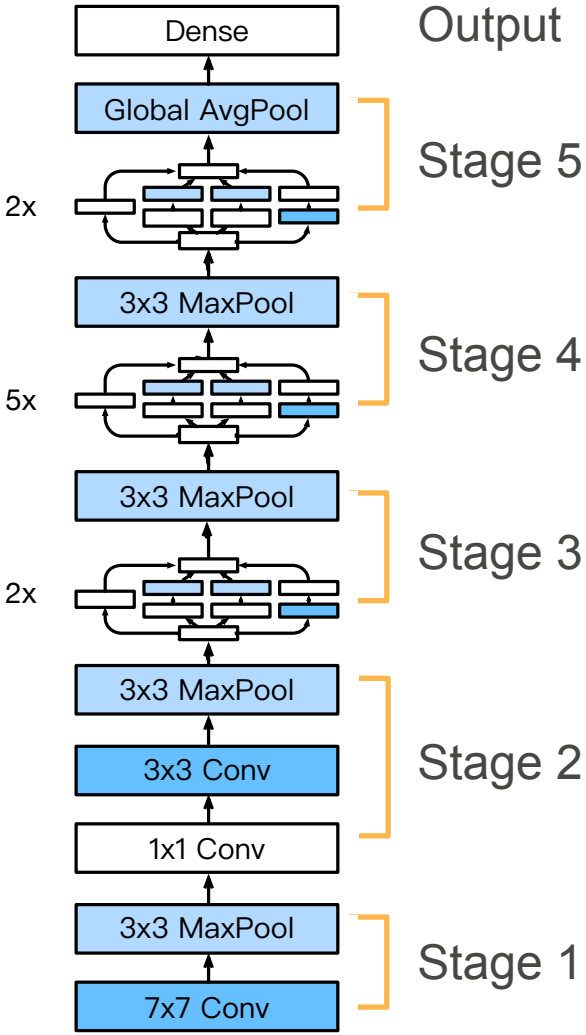
- Mix of different functions (powerful function class)
- Memory and compute efficiency (good generalization)

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



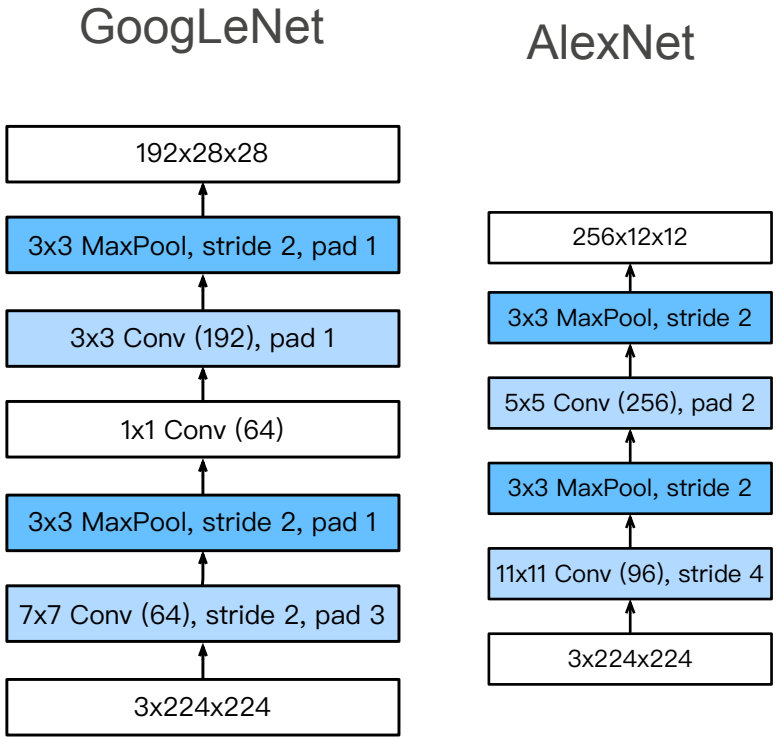
GoogLeNet

- 5 stages with 9 inceptions blocks

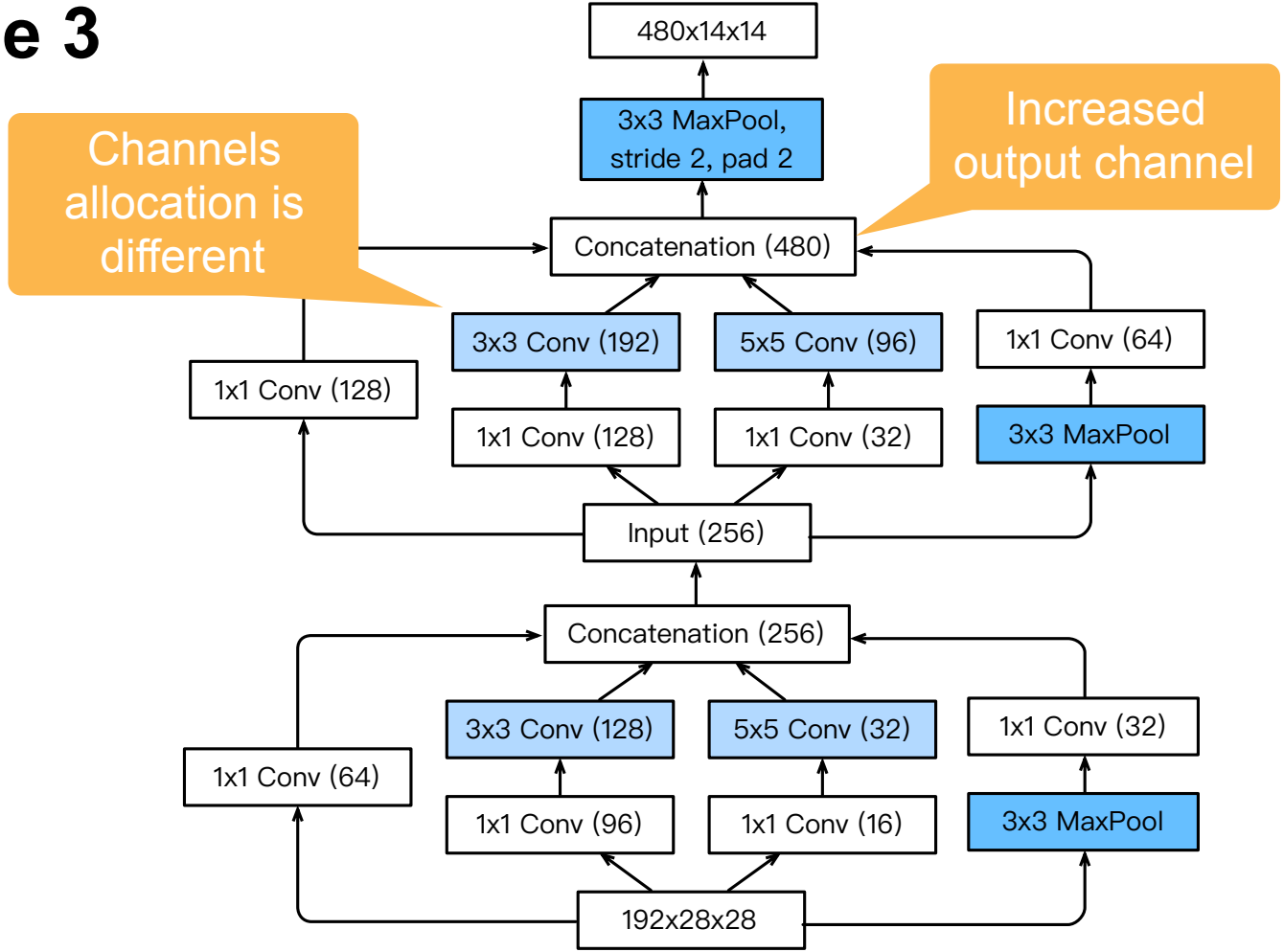


Stage 1 & 2

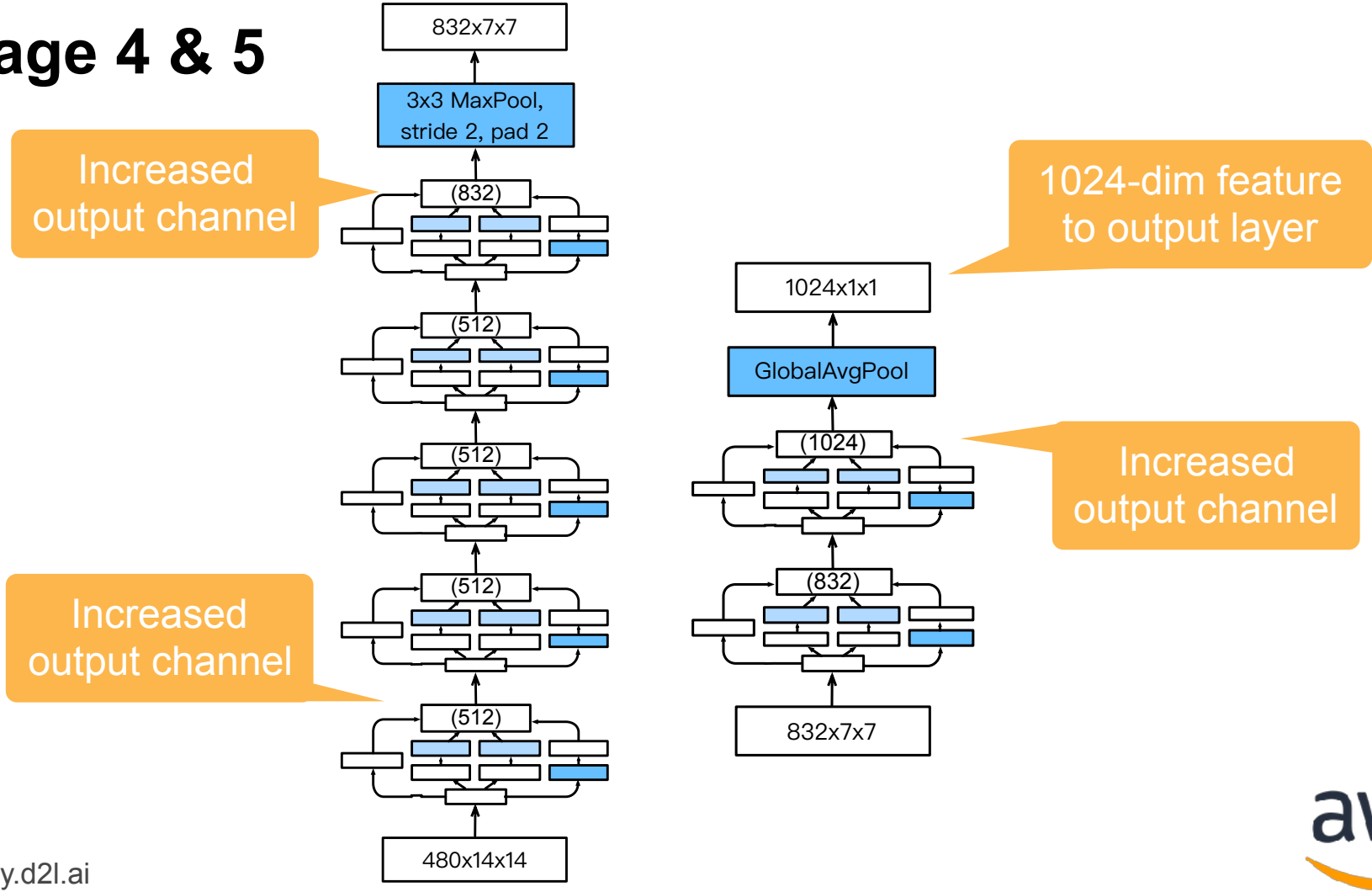
- Smaller kernel size and output channels due to more layers



Stage 3

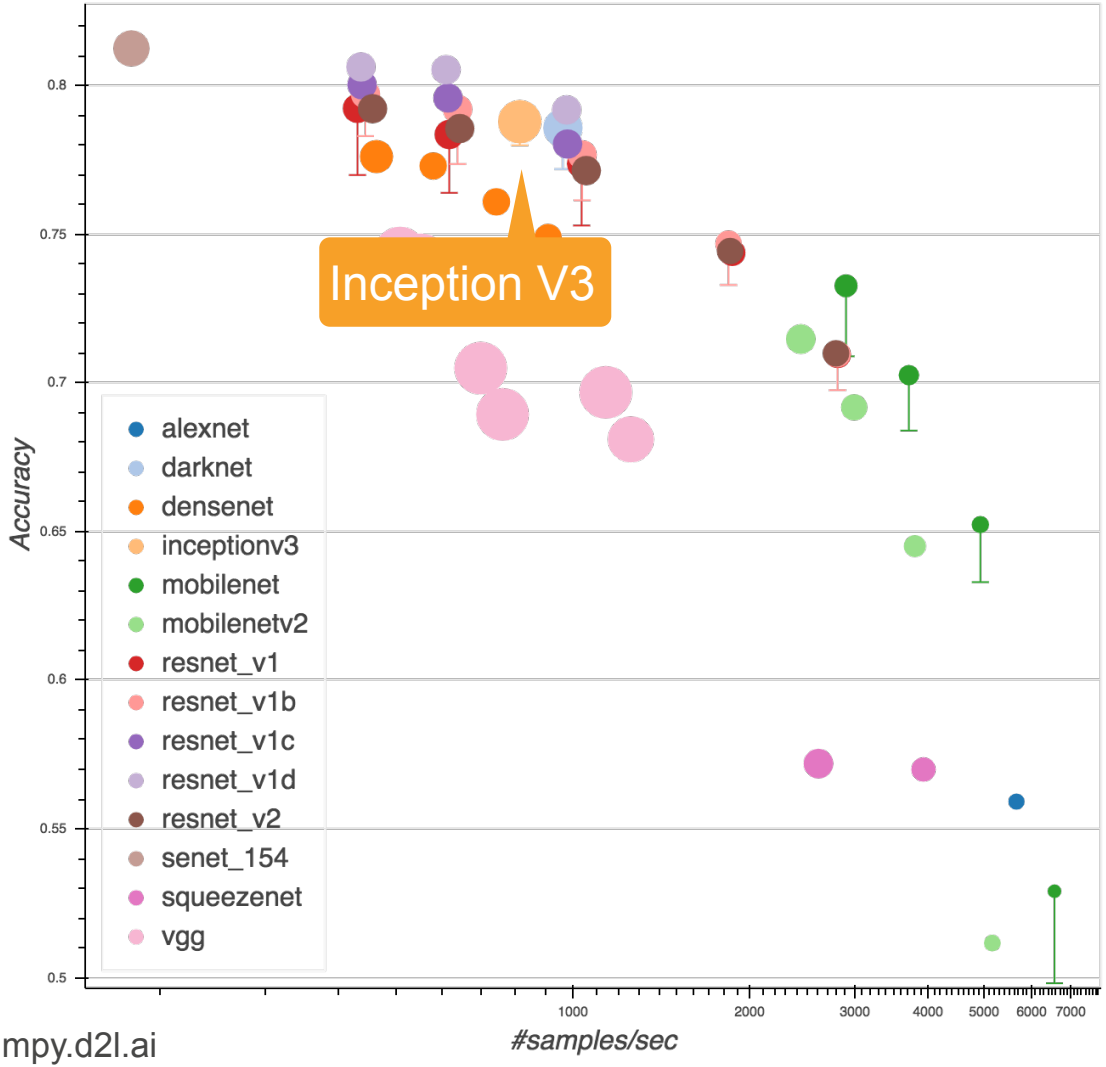

480x14x14
3x3 MaxPool, stride 2, pad 2
Increased output channel
Channels allocation is different
Concatenation (480)
3x3 Conv (192)
5x5 Conv (96)
1x1 Conv (64)
1x1 Conv (128)
1x1 Conv (128)
1x1 Conv (32)
3x3 MaxPool
Input (256)
Concatenation (256)
3x3 Conv (128)
5x5 Conv (32)
1x1 Conv (32)
1x1 Conv (64)
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)
(512)
(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)



GluonCV Model Zoo

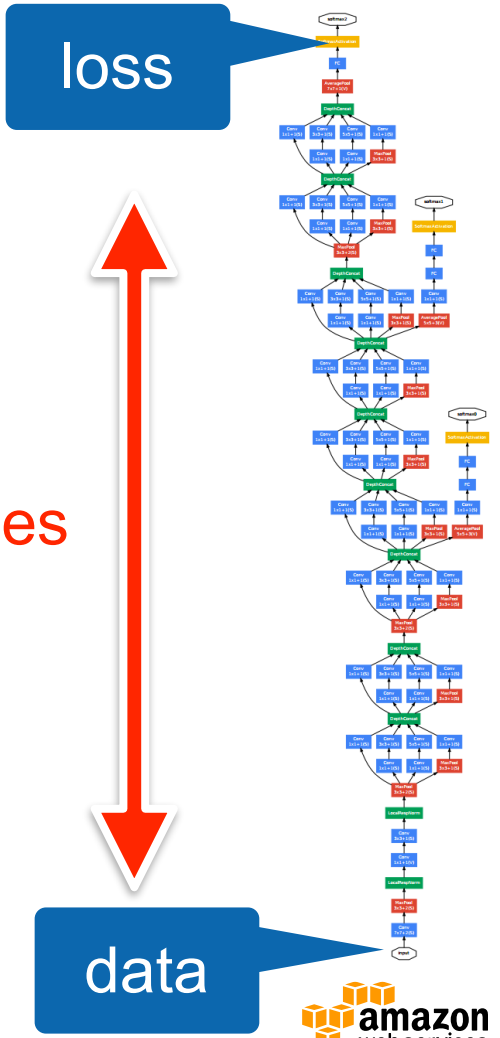
https://gluon-cv.mxnet.io/model_zoo/classification.html

aws


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?



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

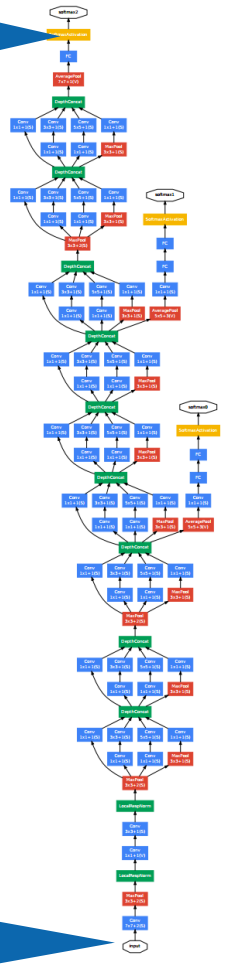
$$x_{i+1} = \gamma \frac{x_i - \mu_B}{\sigma_B} + \beta$$

mean

variance

loss

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

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

Empirical mean

Empirical std.

  - 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

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

Random offset

Random scale

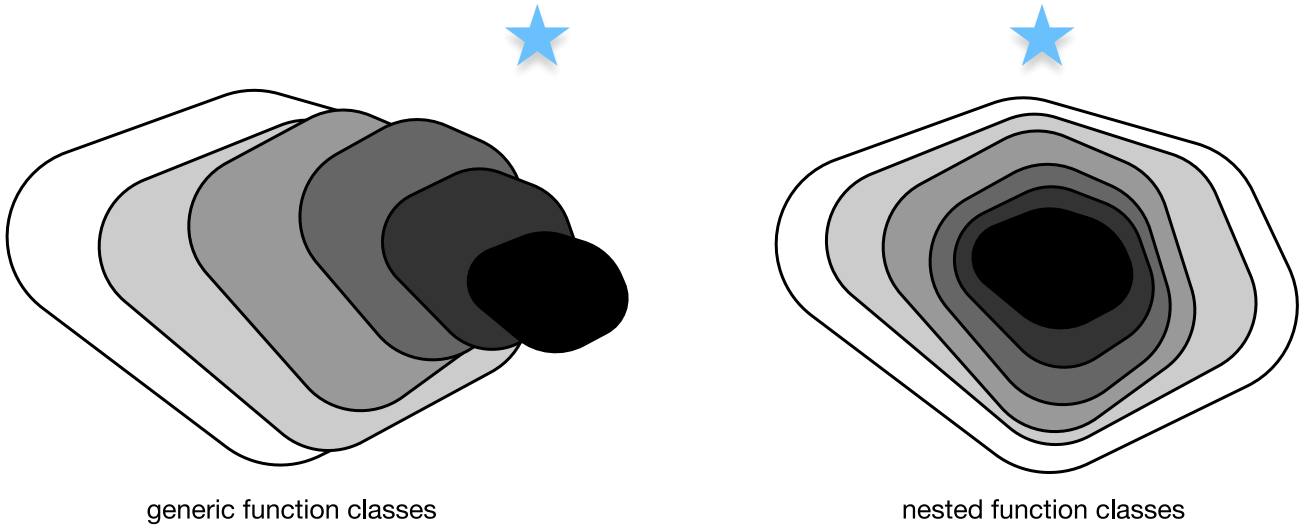
  - Random shift per minibatch
  - Random scale per minibatch
- No need to mix with dropout (both are capacity control)
- Ideal minibatch size of 64 to 256

Details

`gluon.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)

Residual Networks

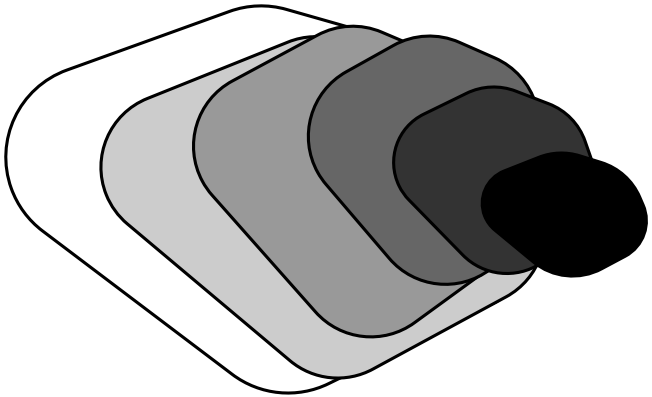


generic function classes

nested function classes

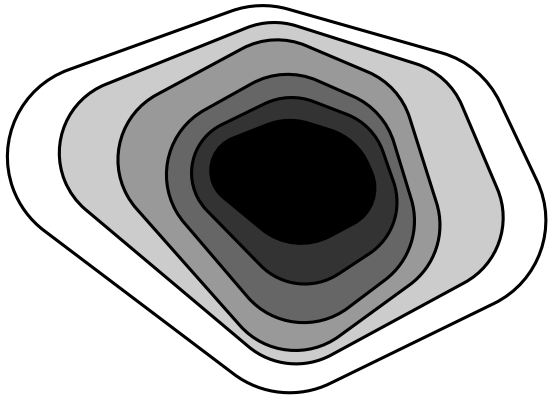
Does adding layers improve accuracy?

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



generic function classes

?



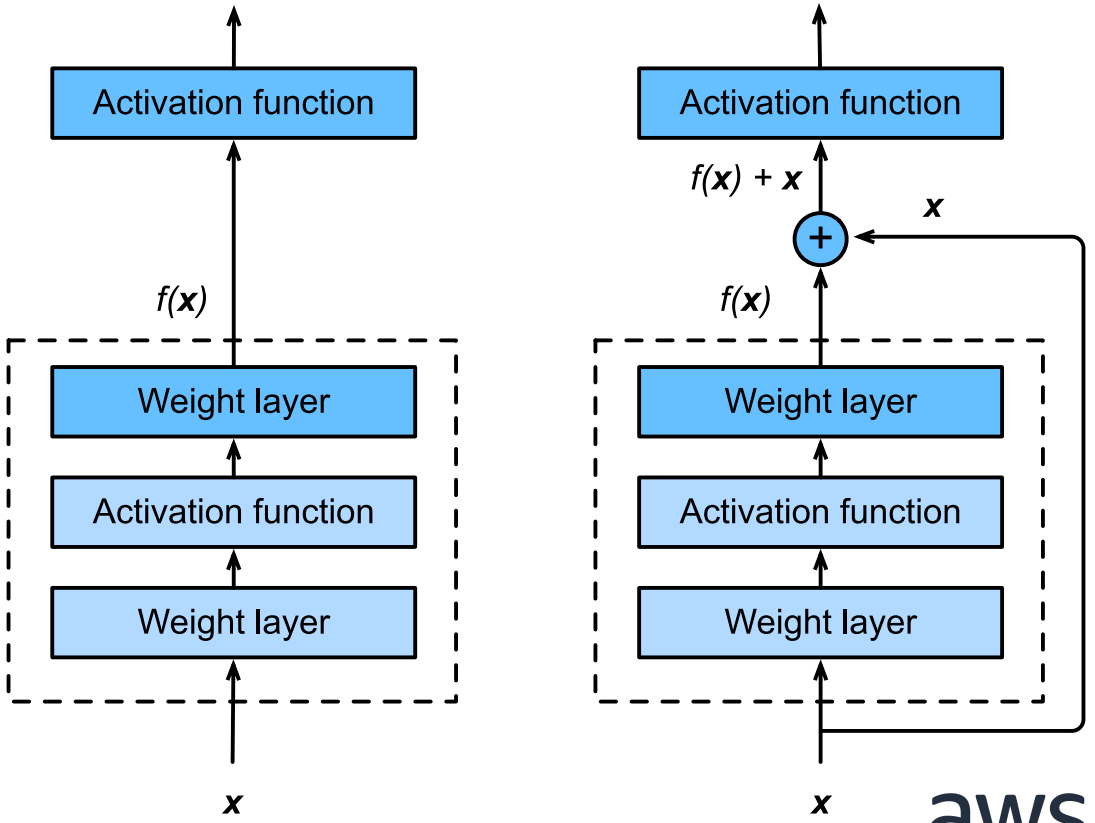
nested function classes

✔

Residual Networks

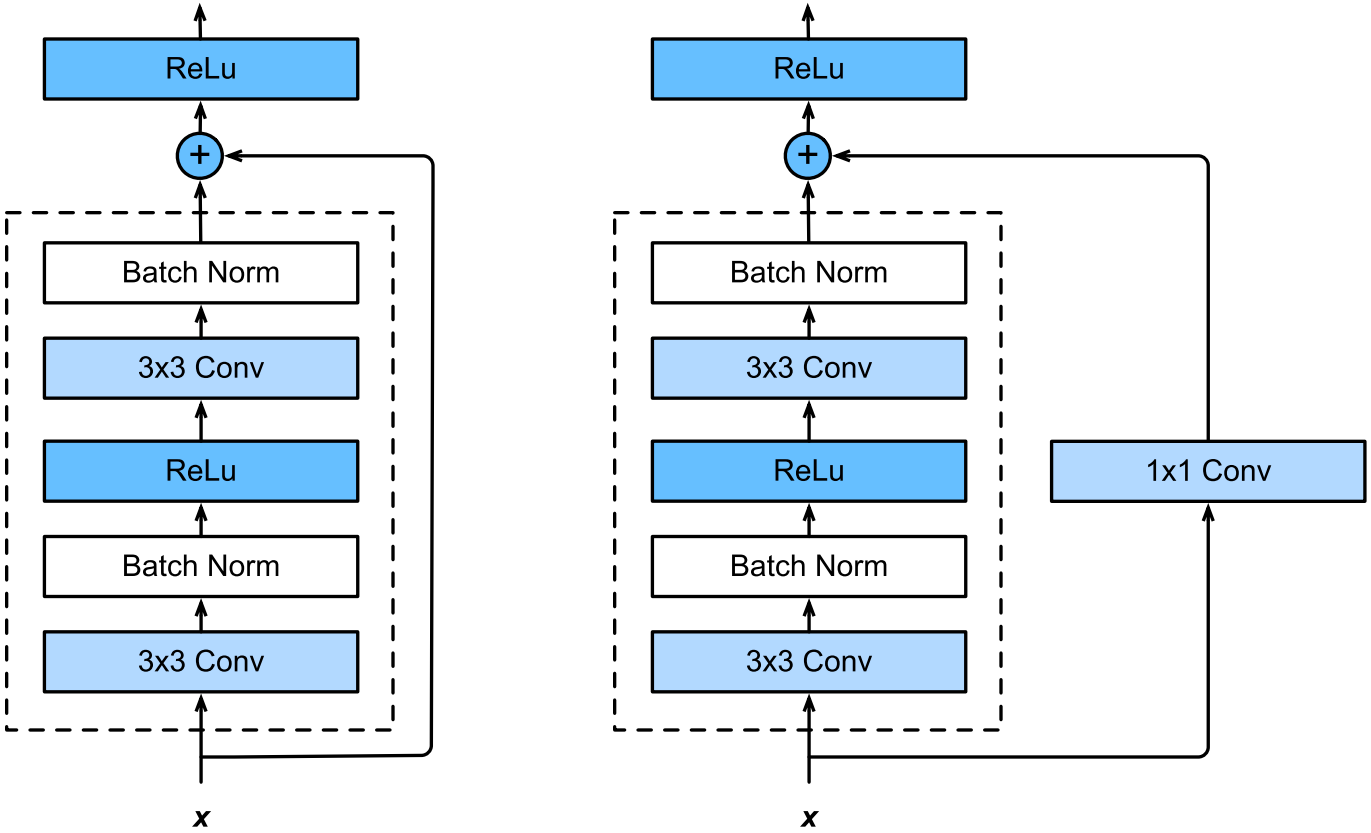
- Adding a layer **changes** function class
- We want to **add to** the function class
- 'Taylor expansion' style parametrization

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

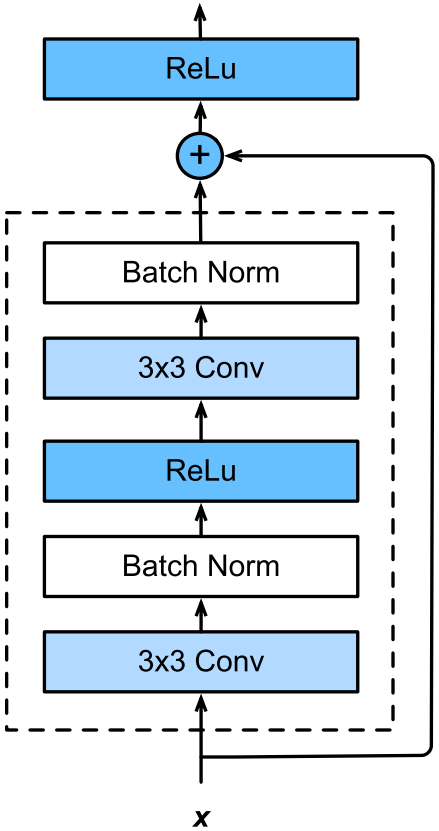


He et al., 2015

ResNet Block in detail

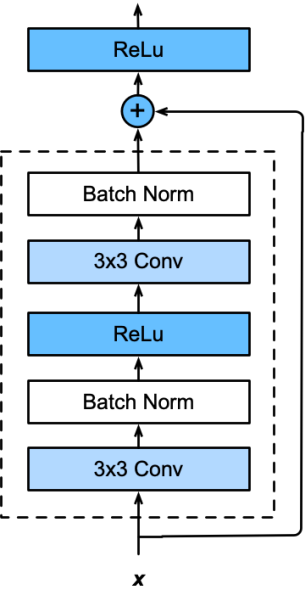


In code

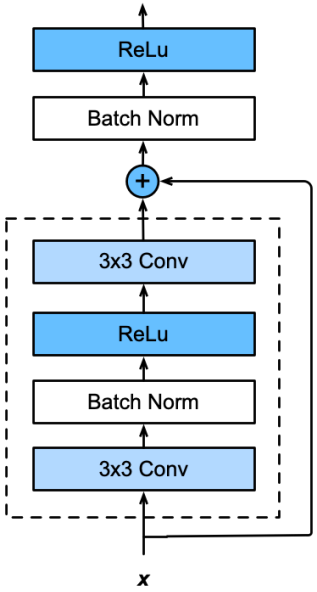



```
def forward(self, X):
  Y = npx.relu(self.bn1(self.conv1(X)))
  Y = self.bn2(self.conv2(Y))
  if self.conv3:
     X = self.conv3(X)
  return npx.relu(Y + X)
```

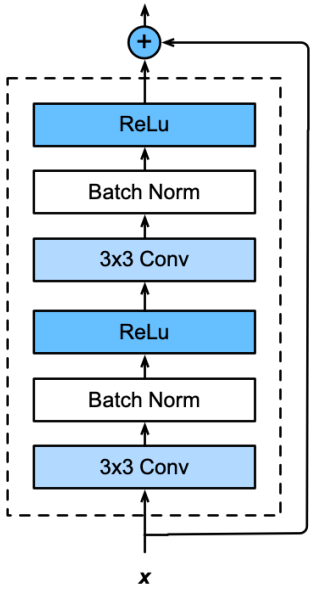
The many flavors of ResNet blocks



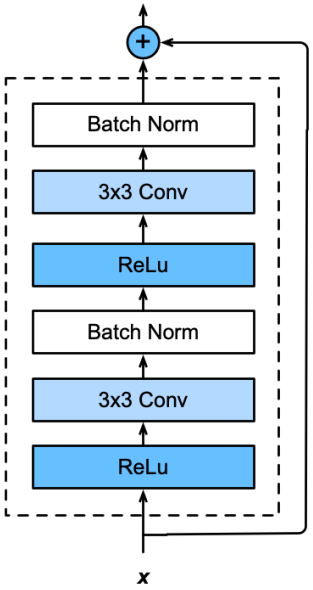
original



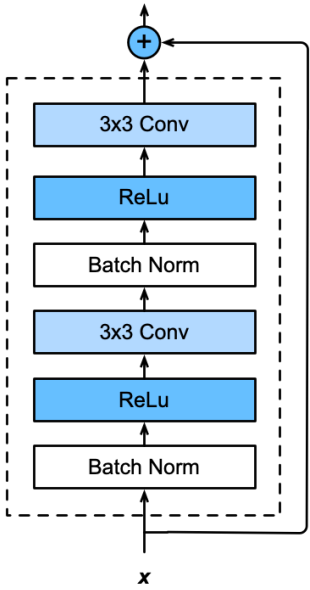
Batch Norm after addition



ReLu last



Batch Norm last



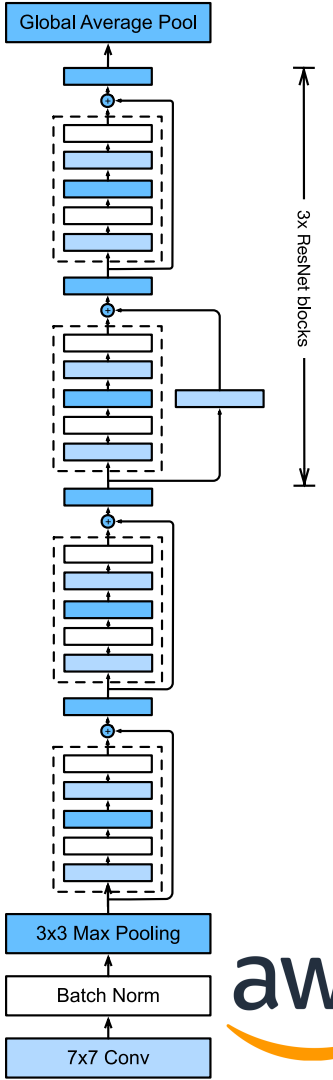
Pre-activation

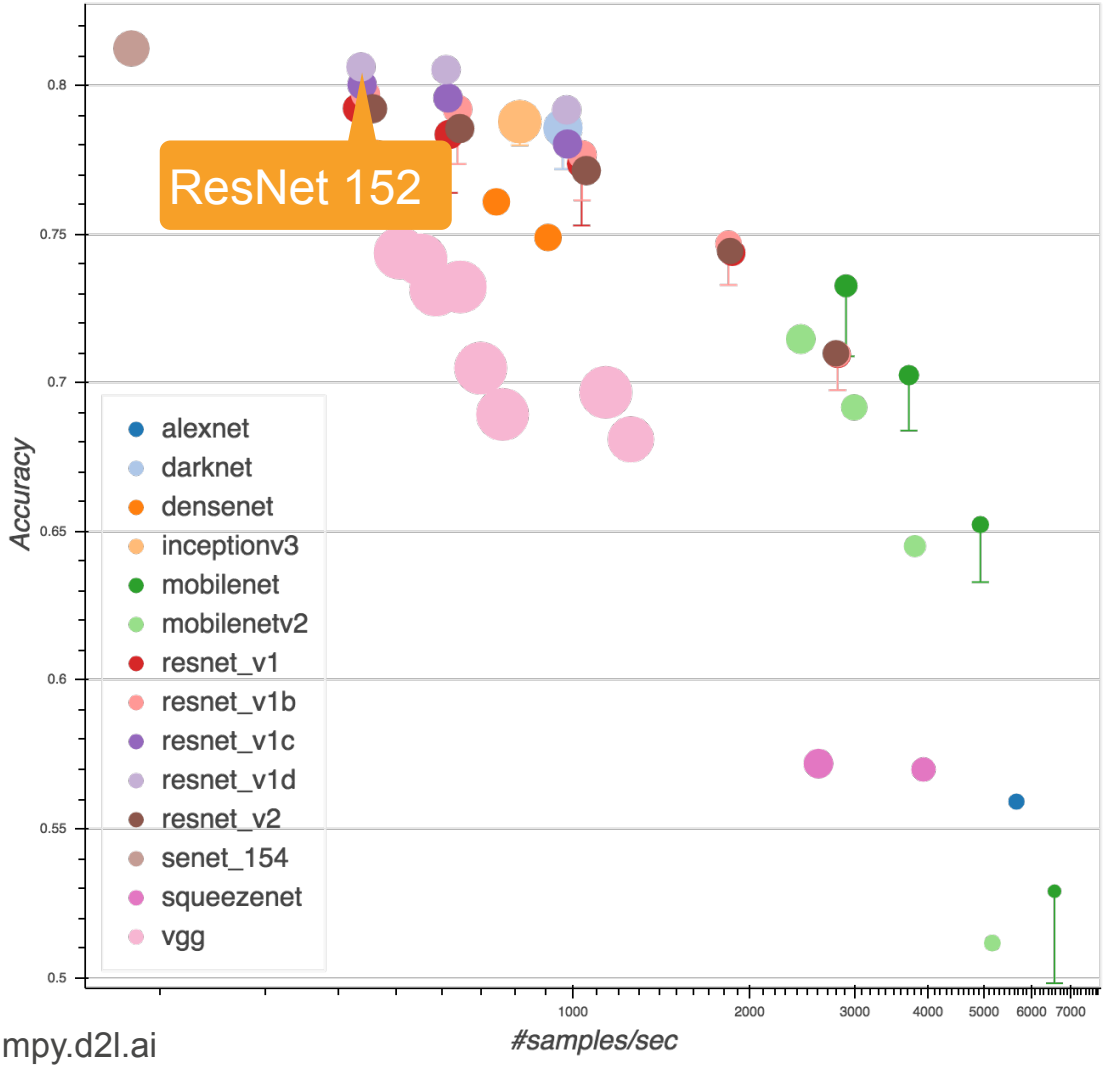
Try every permutation

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

aws

Jupyter Notebook

More Ideas

DenseNet (Huang et al., 2016)

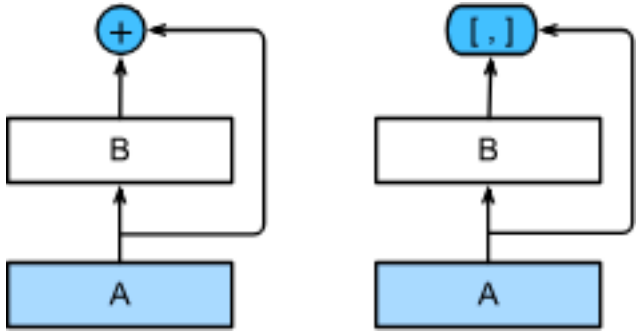
- ResNet combines x and f(x)
- DenseNet uses higher order 'Taylor series' expansion

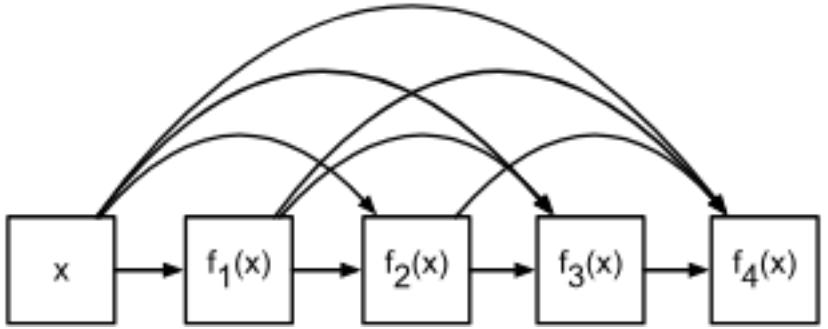
$$x_{i+1} = [x_i, f_i(x_i)]$$

$$x_1 = x$$

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

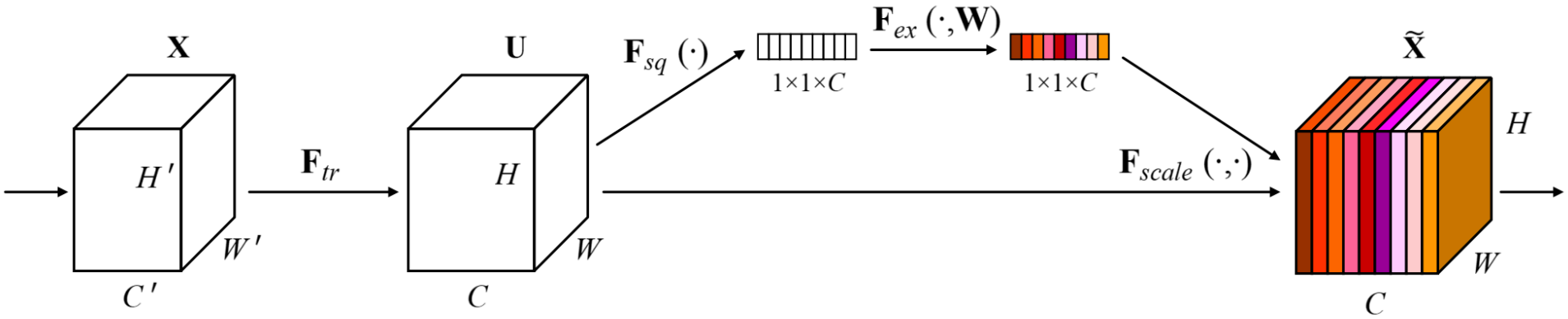
$$x_2 = [x, f_1(x), f_2([x, f_1(x)])]$$





aws

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

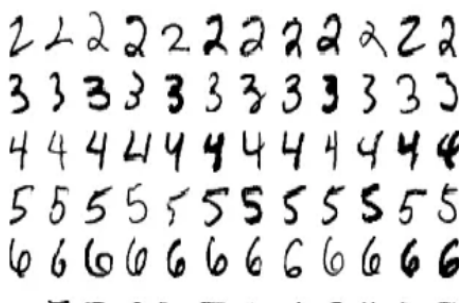


- Learn global weighting function per channel
- Allows for fast information transfer between pixels in different locations of the image


Fine Tuning

Labelling a Dataset is Expensive

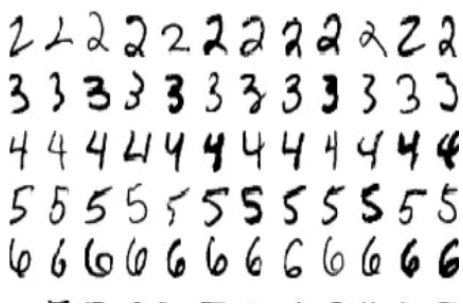
# examples	1.2M	50K	60K
# classes	1,000	100	10



My dataset

Labelling a Dataset is Expensive

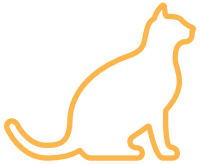
# examples	1.2M	50K	60K
# classes	1,000	100	10



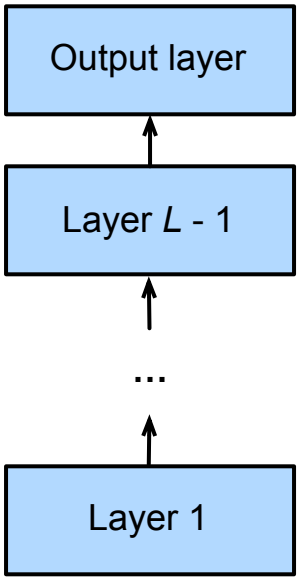
Can we reuse this?

My dataset

aws



Network Structure



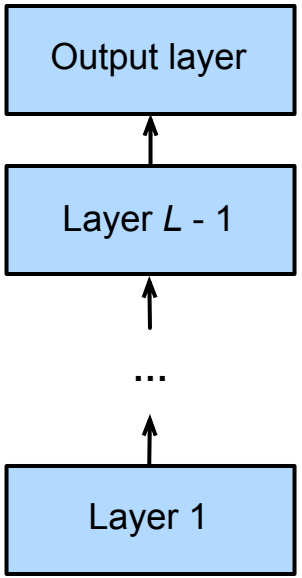
Softmax classifier

Feature extractor

Two components in deep network

- **Feature extractor to map raw pixels into linearly separable features.**
- Linear classifier for decisions

Fine Tuning



Don’t use last layer since classification problem is different

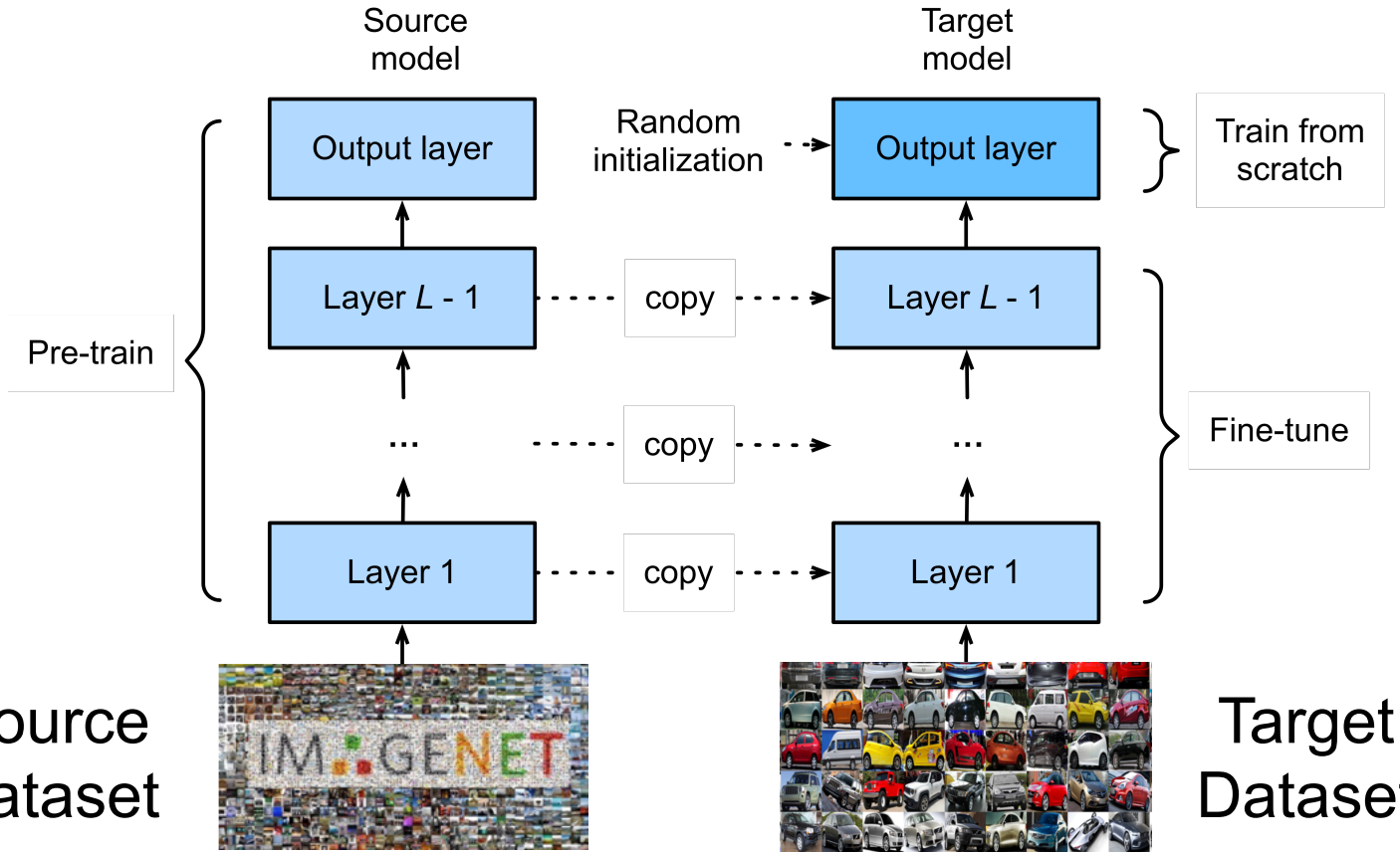
Likely good feature extractor for target

Source Dataset

Target Dataset

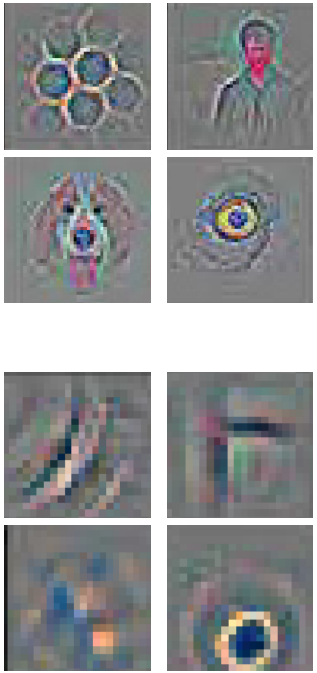
aws

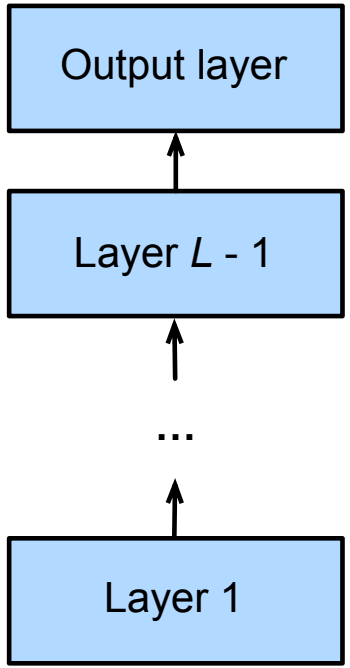
Weight Initialization for Fine Turning



Fix Lower Layers

- Neural networks learn hierarchical feature representations
  - Low-level features are universal
  - High-level features are more related to objects in the dataset
- **Fix the bottom layer parameters during fine tuning**
  (useful for regularization)

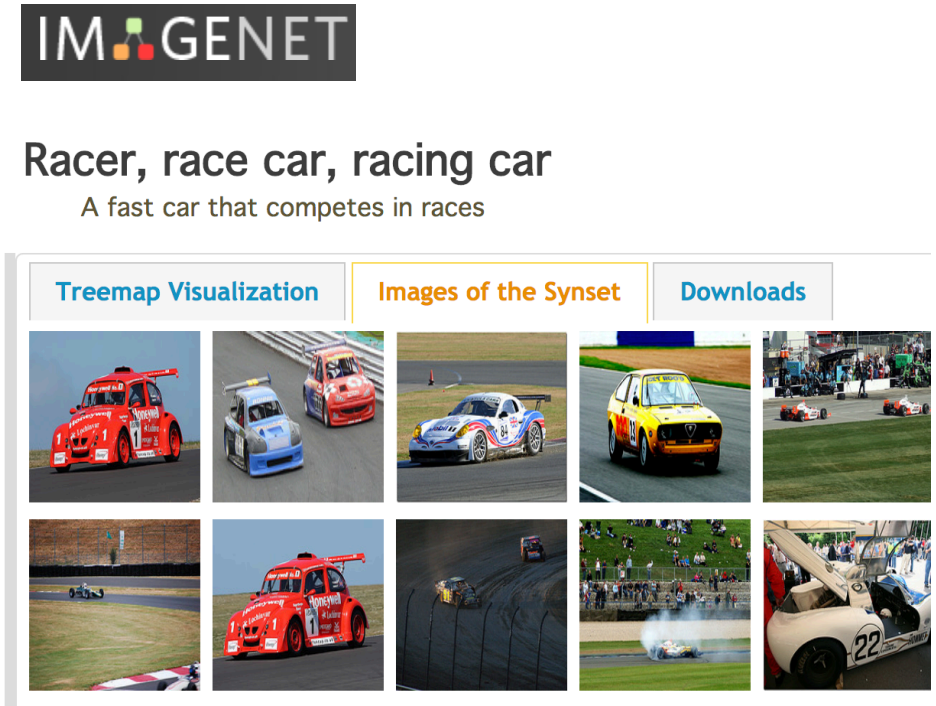




Re-use Classifier Parameters

Lucky break

- Source dataset may contain some of the target categories
- Use the according weight vectors from the pre-trained model during initialization



Fine-tuning Training Recipe

- Train on the target dataset as normal but with strong regularization
  - Small learning rate
  - Fewer epochs
- If source dataset is more complex than the target dataset, fine-tuning can lead to better models (source model is a good prior)

Fine-tuning Notebook