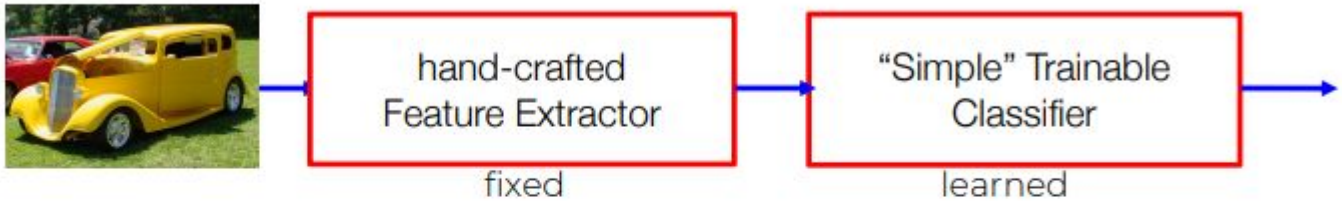
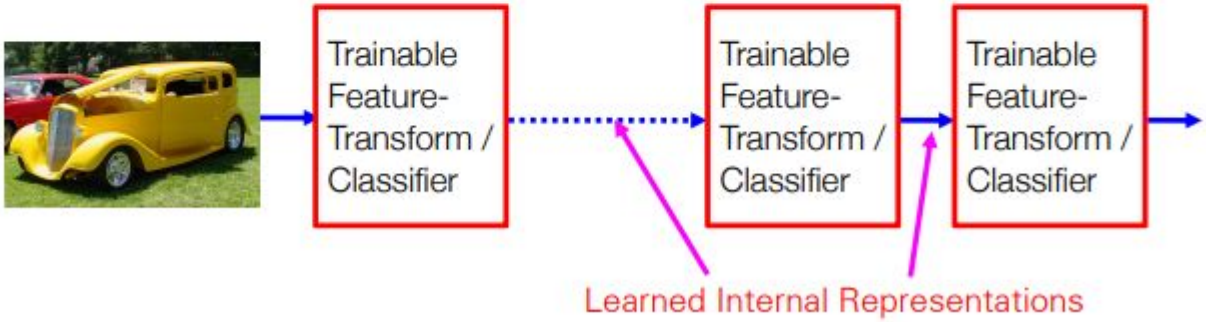
Convolutional Neural Network

So what is deep learning?

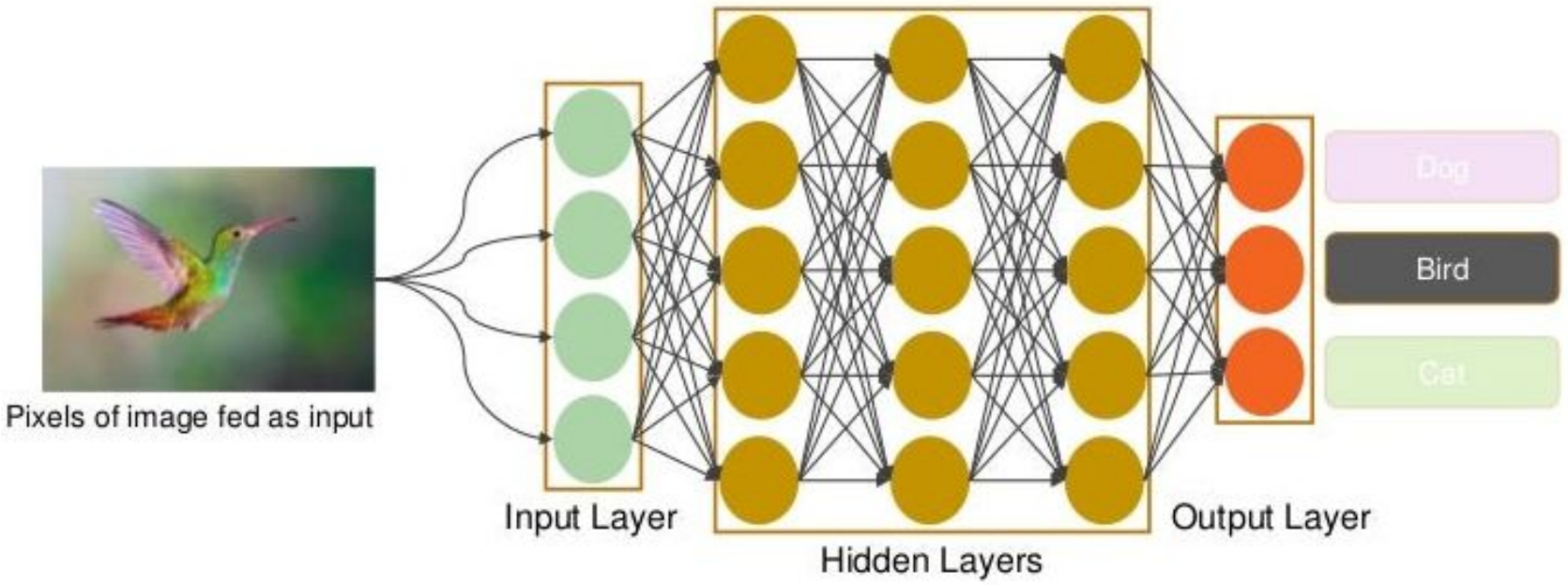
- "Shallow" models



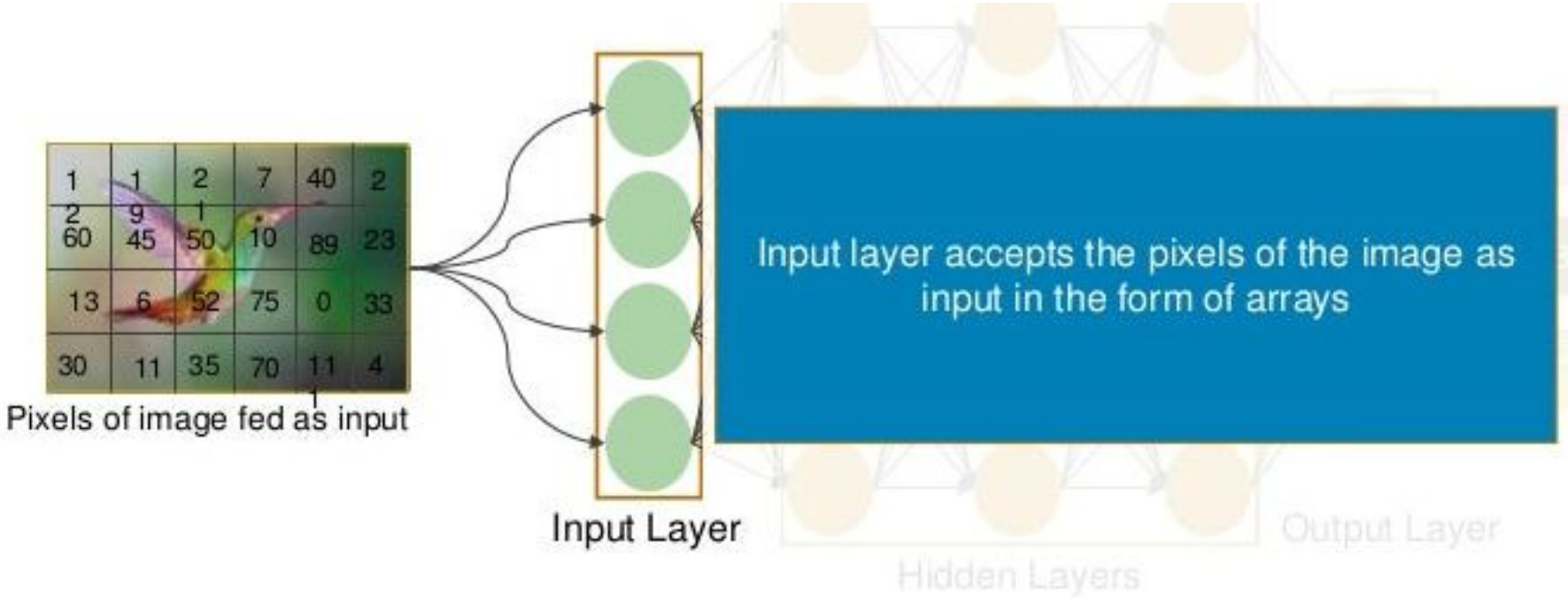
- Deep models



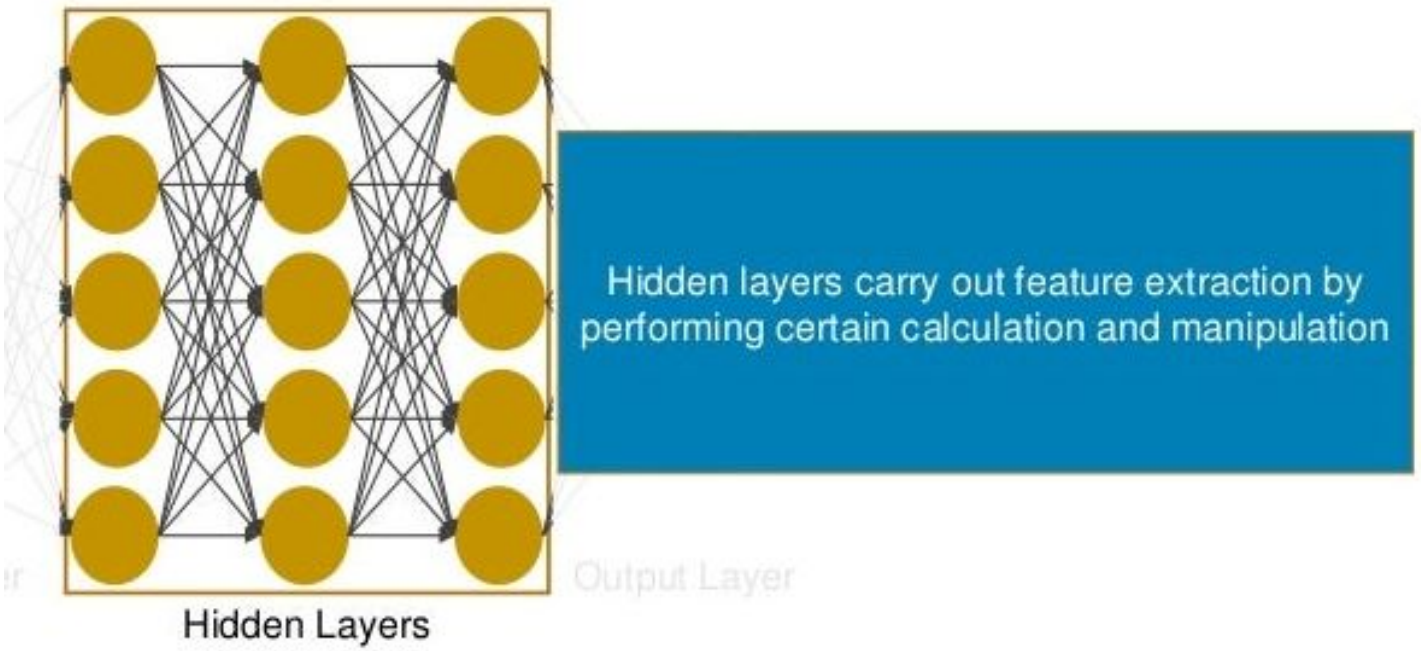
Outline

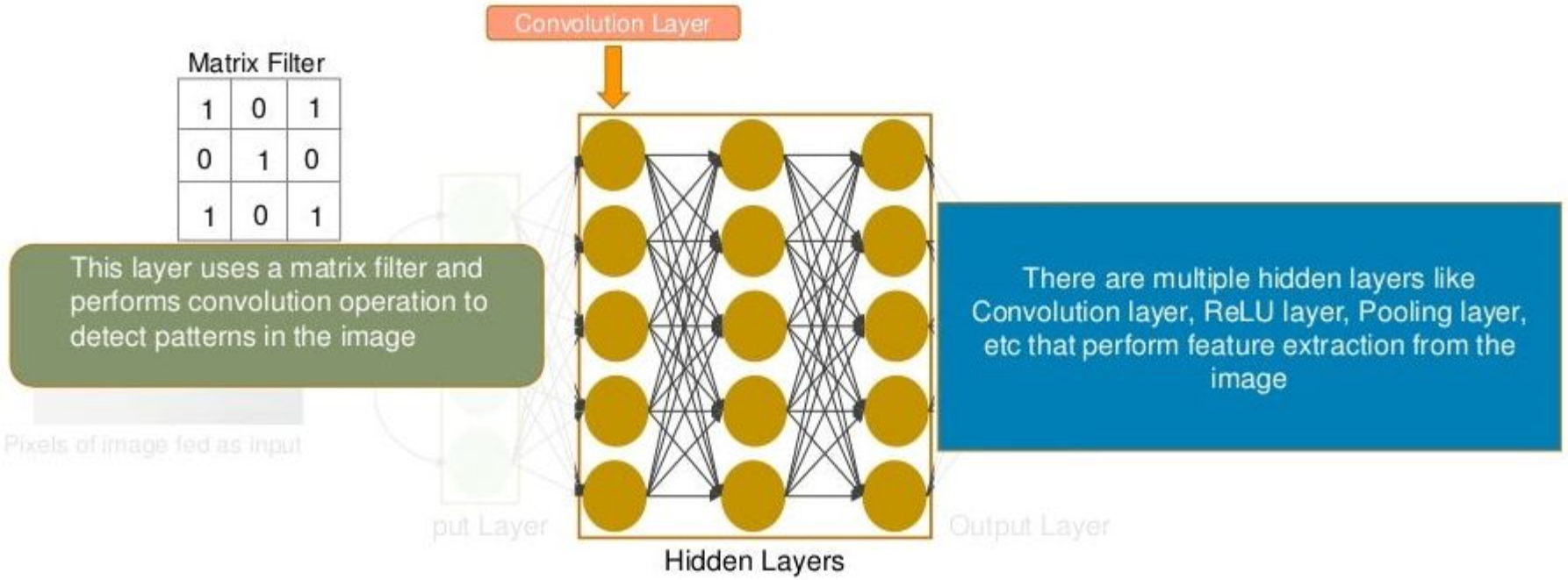

Pixels of image fed as input
Input Layer
Hidden Layers
Output Layer
Dog
Bird
Cat

Outline

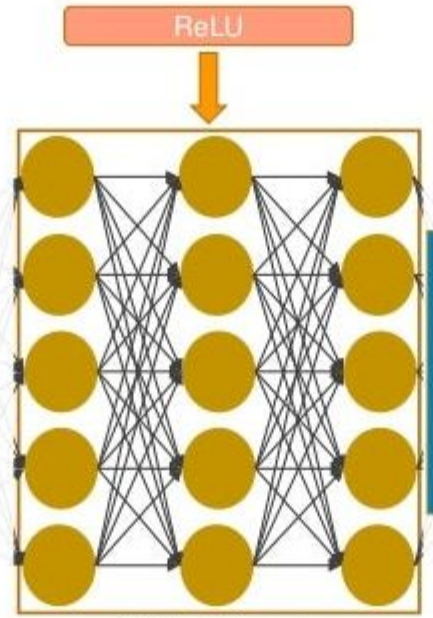

1 1 2 7 40 2
2 9 1
60 45 50 10 89 23
13 6 52 75 0 33
30 11 35 70 11 4
Pixels of image fed as input
Input Layer
Input layer accepts the pixels of the image as input in the form of arrays
Hidden Layers
Output Layer

Outline

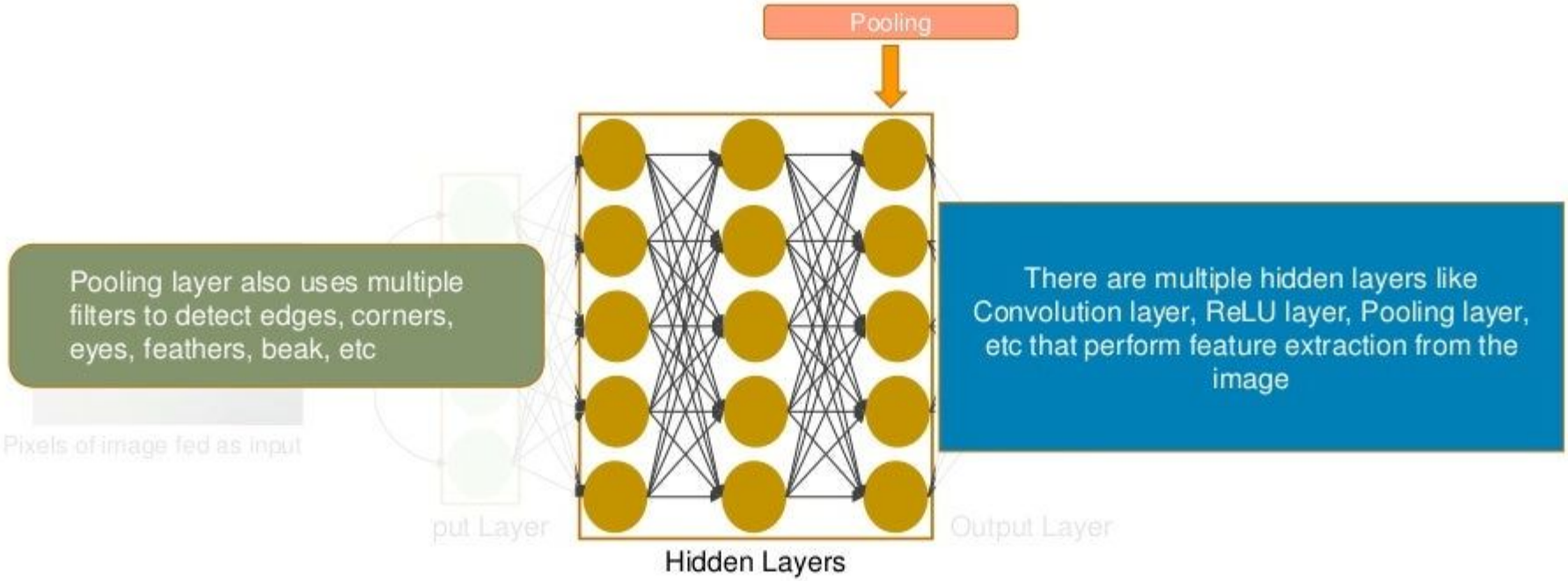

Hidden layers carry out feature extraction by performing certain calculation and manipulation
Output Layer
Hidden Layers

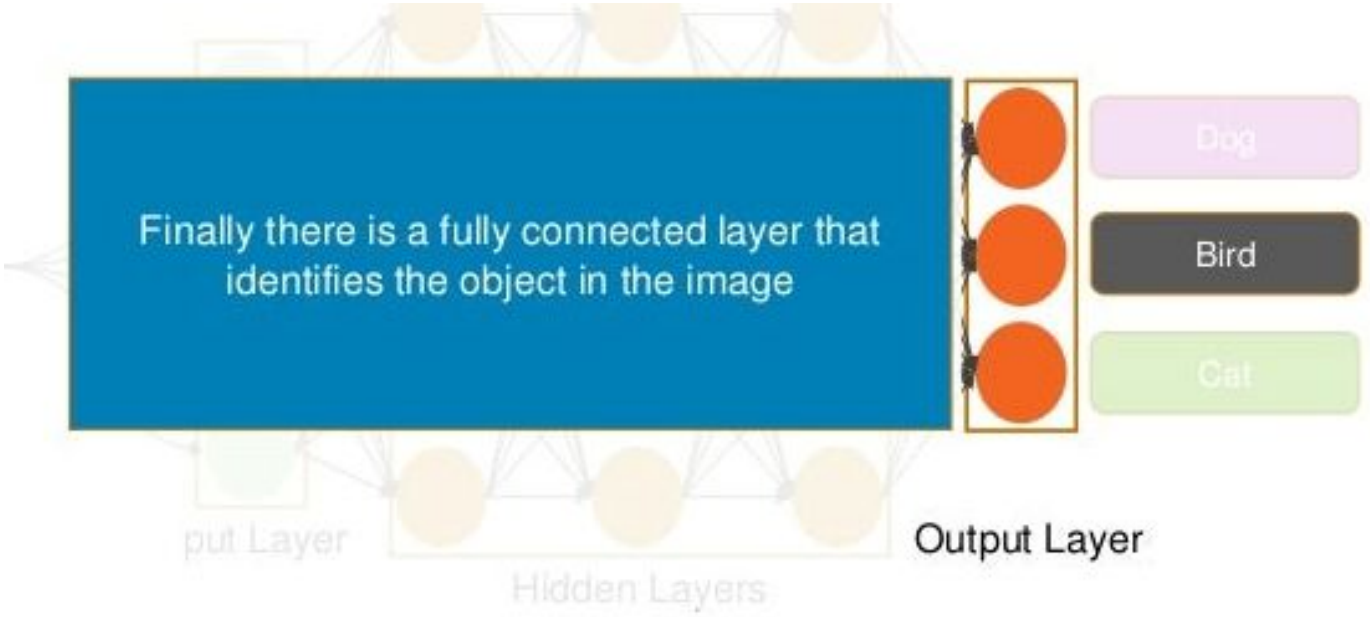

Convolution Layer
Matrix Filter
1 0 1
0 1 0
1 0 1
This layer uses a matrix filter and performs convolution operation to detect patterns in the image
There are multiple hidden layers like Convolution layer, ReLU layer, Pooling layer, etc that perform feature extraction from the image
Hidden Layers

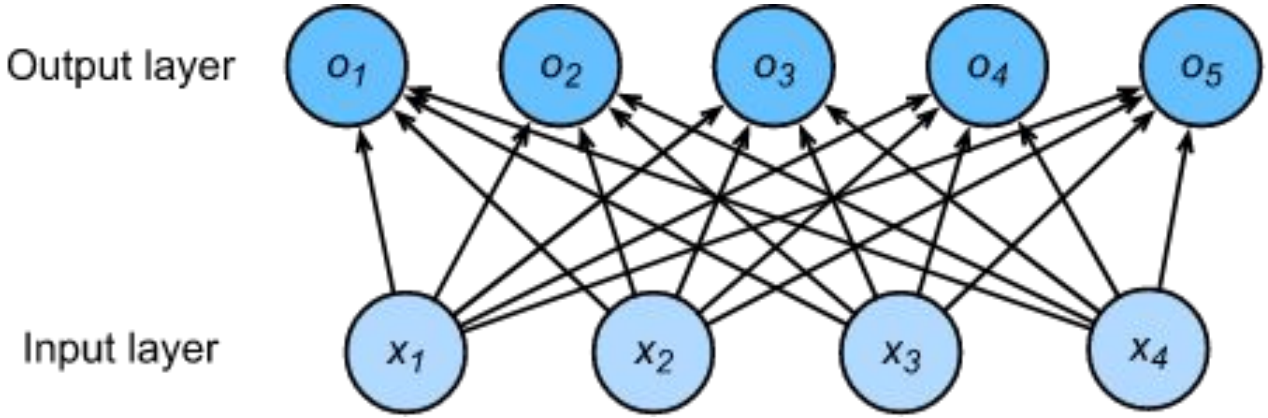
ReLU activation function is applied to the convolution layer to get a rectified feature map of the image



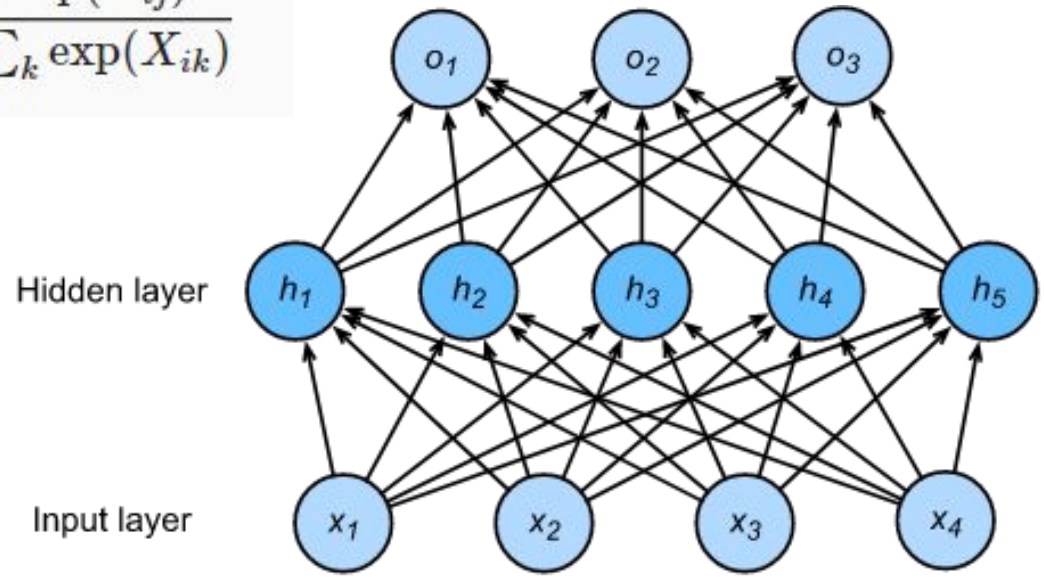
There are multiple hidden layers like Convolution layer, ReLU layer, Pooling layer, etc that perform feature extraction from the image


Pooling
Pooling layer also uses multiple filters to detect edges, corners, eyes, feathers, beak, etc
There are multiple hidden layers like Convolution layer, ReLU layer, Pooling layer, etc that perform feature extraction from the image
Pixels of image fed as input
put Layer
Output Layer
Hidden Layers


Finally there is a fully connected layer that identifies the object in the image
Dog
Bird
Cat
Output Layer
Hidden Layers



$$\mathrm{softmax}(\mathbf{X})_{ij} = \frac{\exp(X_{ij})}{\sum_k \exp(X_{ik})}$$



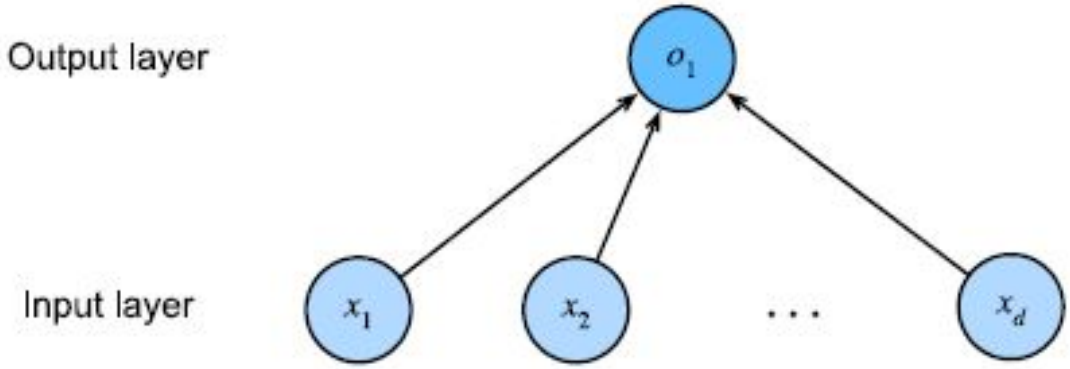


Fig. 3.1.2: Linear regression is a single-layer neural network.

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

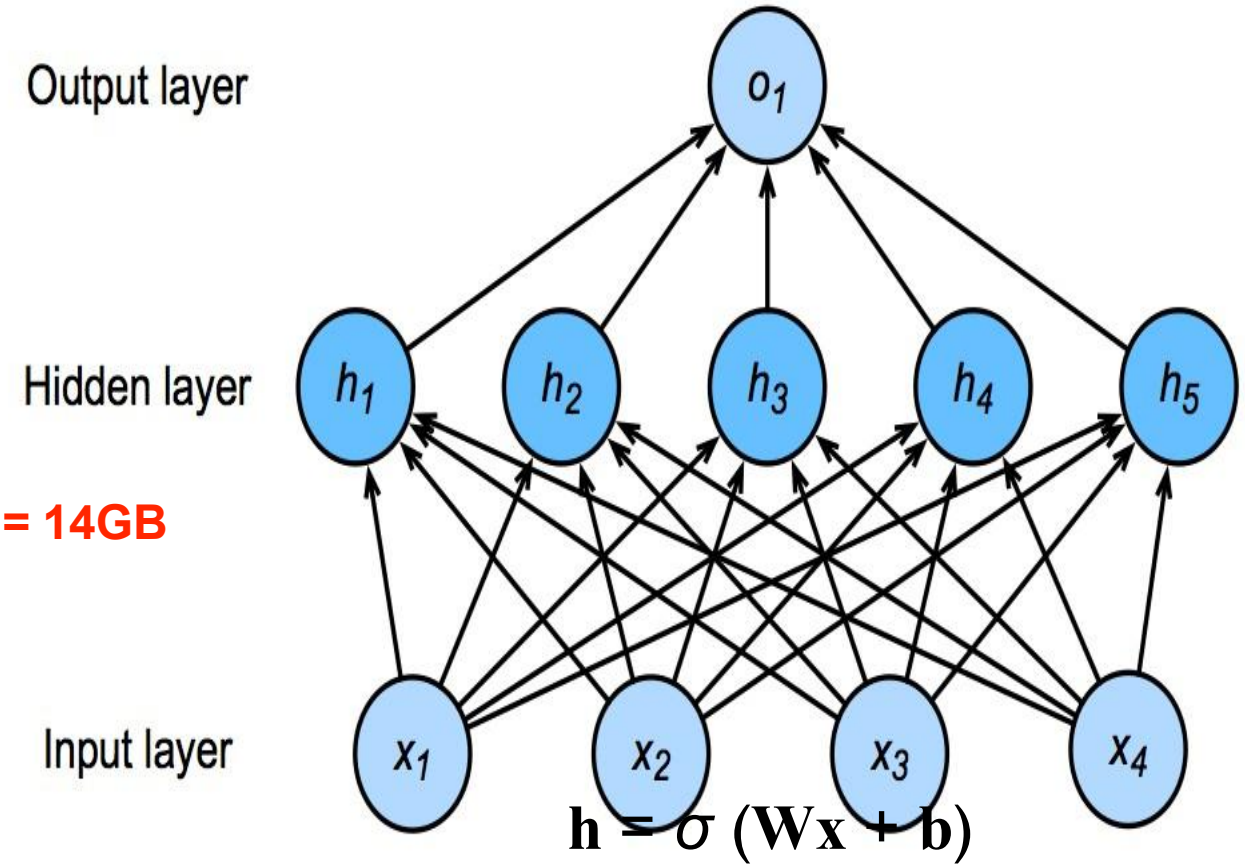
(900M dogs + 600M cats)

Flashback - Network with one hidden layer

100 neurons

3.6B parameters = 14GB

36M features



Where is Waldo?

- Translation Invariance
- Locality

Convolution

the *convolution* between two functions, say $f, g : \mathbb{R}^d \to \mathbb{R}$ is defined as

$$(f * g)(\mathbf{x}) = \int f(\mathbf{z}) g(\mathbf{x} - \mathbf{z}) d\mathbf{z}.$$

Discrete

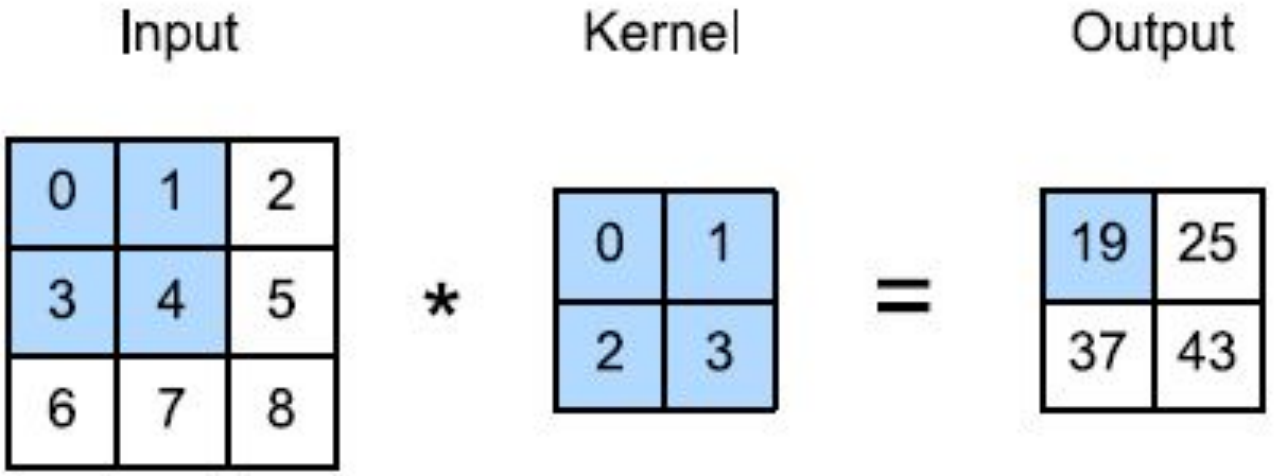
$$(f * g)(i) = \sum_a f(a) g(i - a).$$

Two dimension

$$(f * g)(i, j) = \sum_a \sum_b f(a, b) g(i - a, j - b).$$

Convolutions for Images

The Cross-Correlation Operation



Rethinking Dense Layers

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

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

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

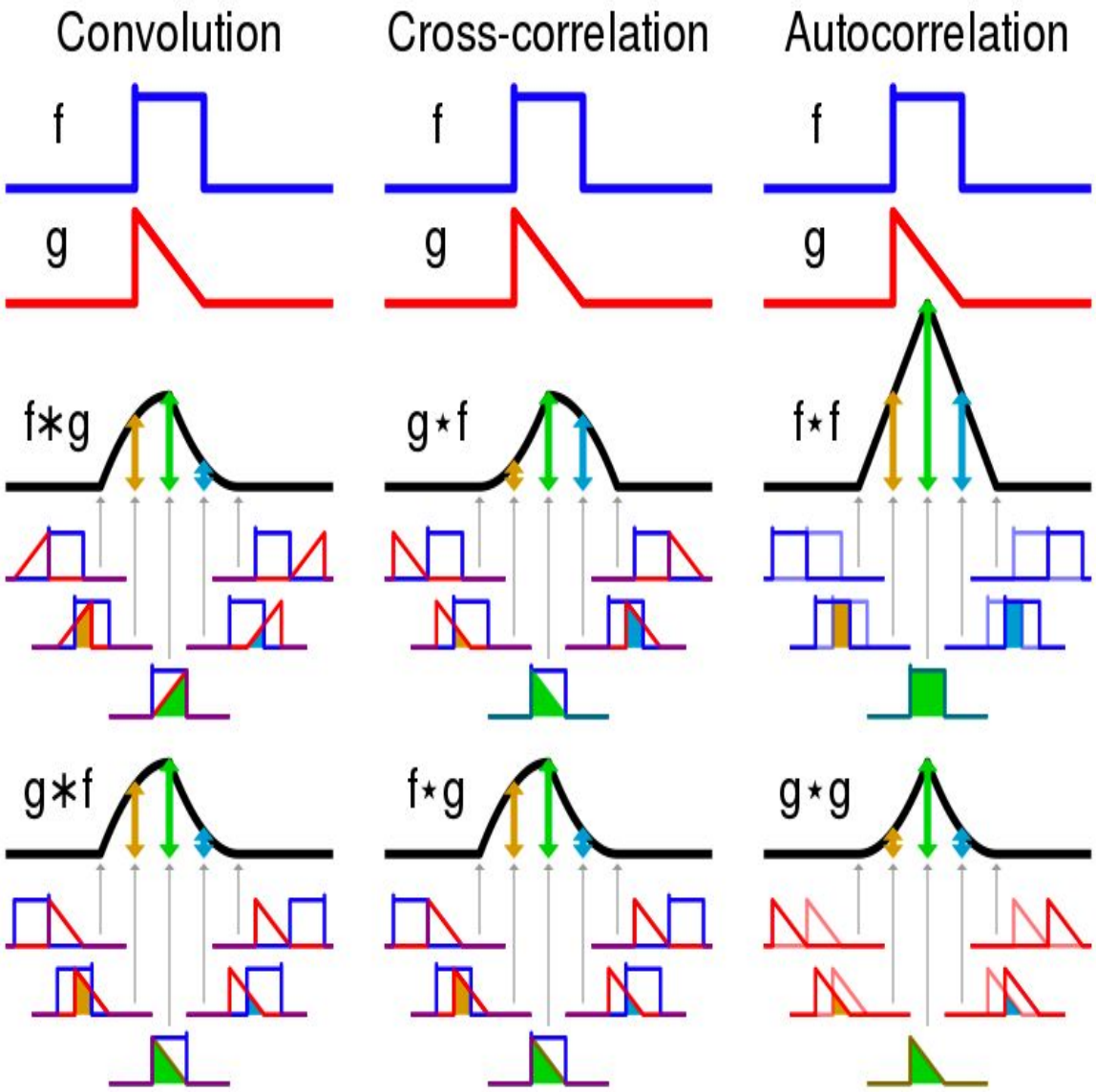
Idea #2 - Locality

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

- 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 $v_{a,b} = 0$

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

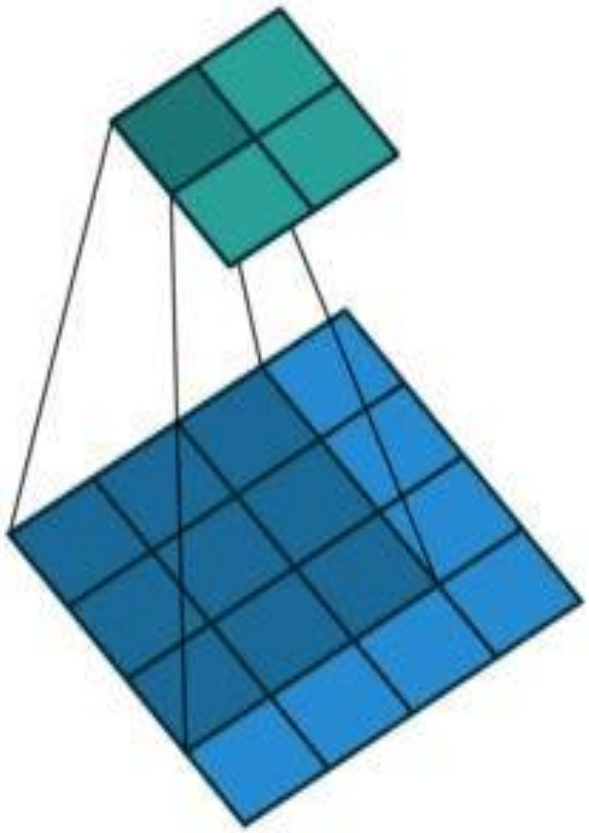
Convolution



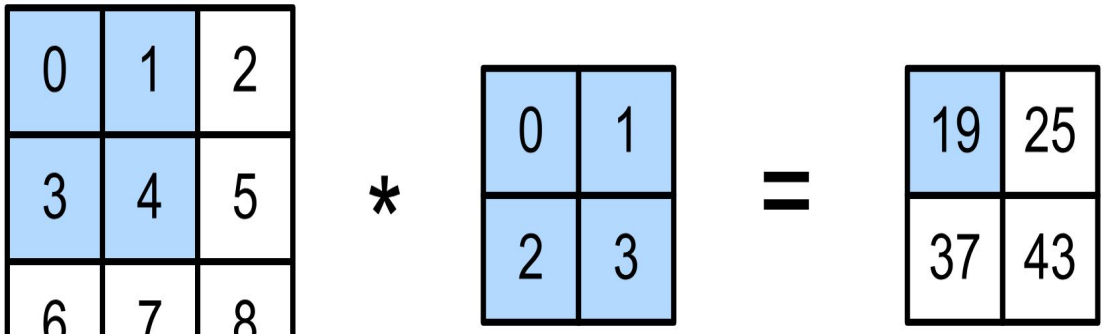
2-D Cross Correlation

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

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



2-D Convolution Layer



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

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

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

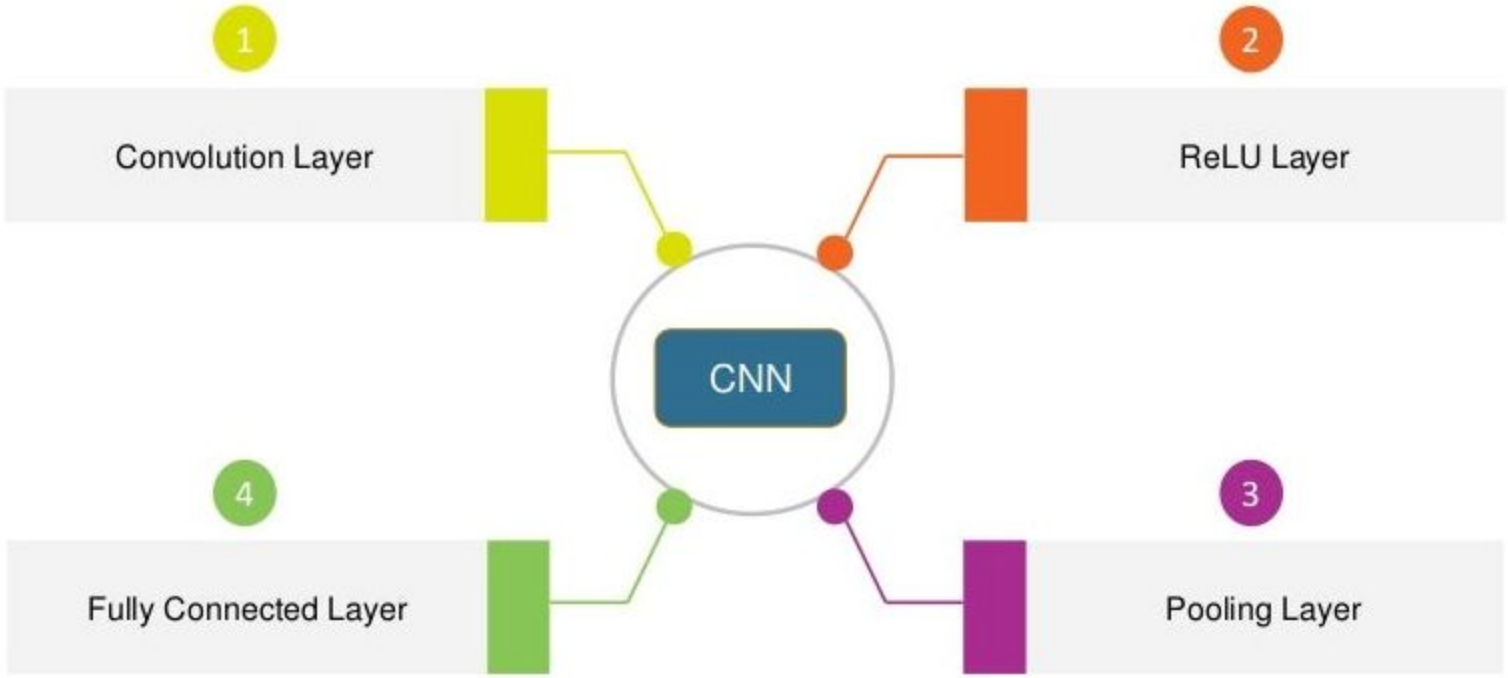
(Rob Fergus)

Feature Map and Receptive Field

The convolutional layer output is sometimes called a feature map, as it can be regarded as the learned representations (features) in the spatial dimensions (e.g., width and height) to the subsequent layer.

In CNNs, for any element x of some layer, its *receptive field* refers to all the elements (from all the previous layers) that may affect the calculation of x during the forward propagation.

Convolutions


1
Convolution Layer
2
ReLU Layer
CNN
4
Fully Connected Layer
3
Pooling Layer


Padding and Stride

Padding

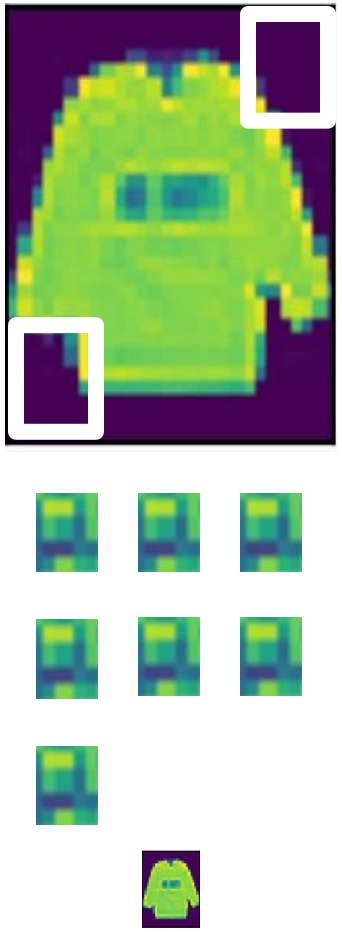
- Given a 32 x 32 input image
- Apply convolutional layer with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels

  obliterating any interesting information on the boundaries of the original image

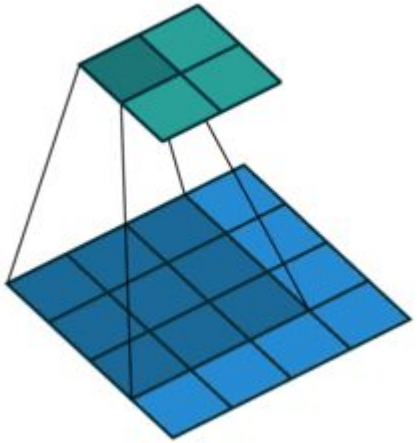
  - Shape reduces from $n_h \times n_w$ to

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

using kernel shape is $k_h \times k_w$

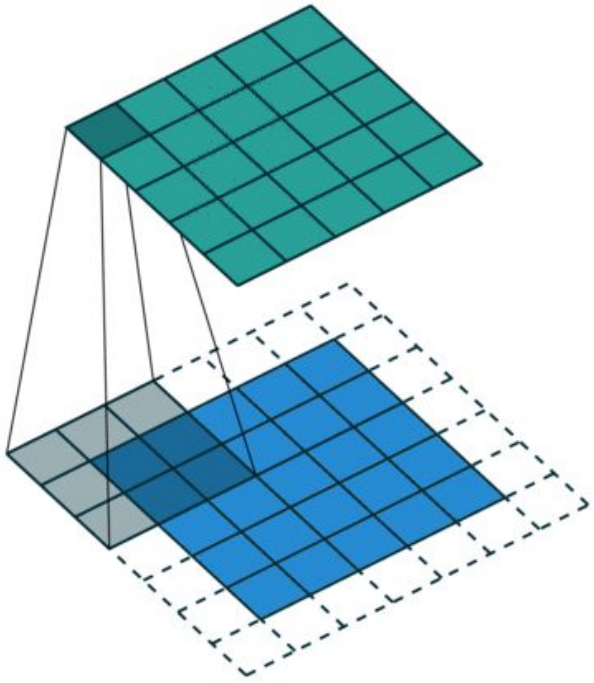


Convolutional Layer



We can also pad the edges of your images with 0-valued pixels as to fully scan the original image and preserve its complete dimensions.

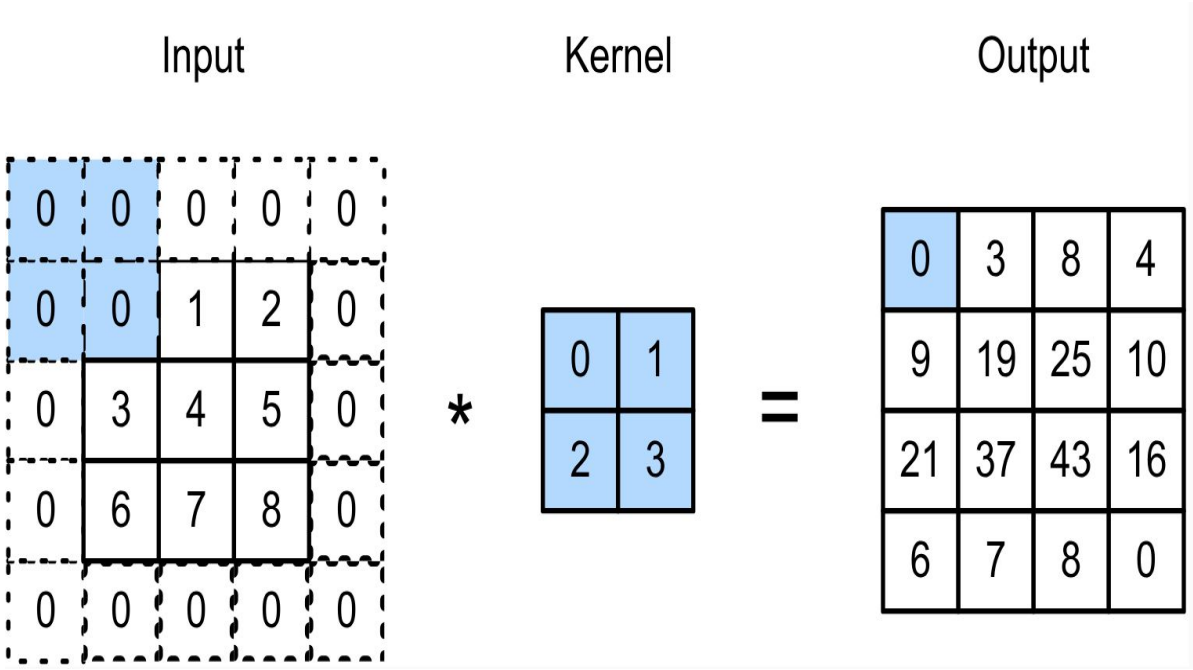
Convolution with padding

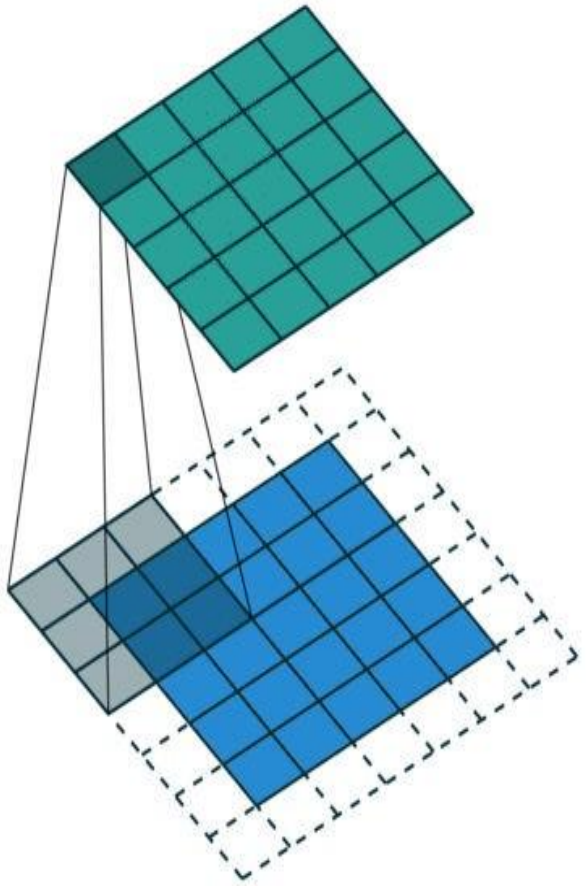


Convolution with padding

Padding

Padding adds rows/columns around input





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

Padding

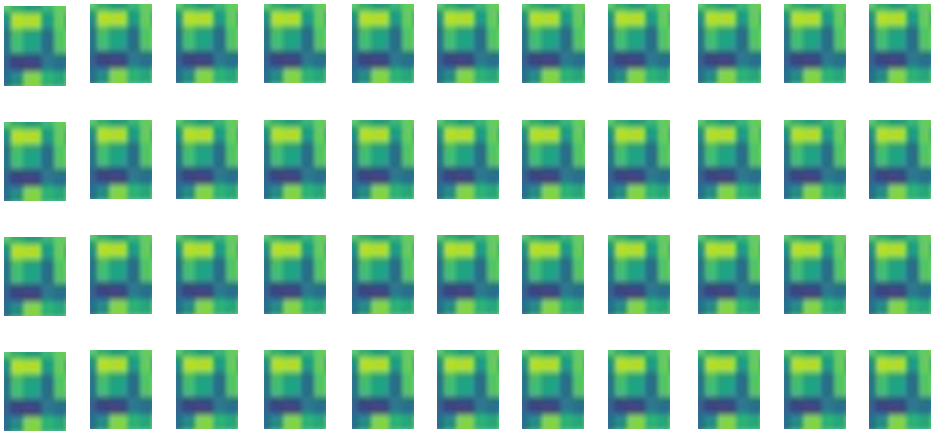
- Padding p_h rows and p_w columns, output shape will be

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

- 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

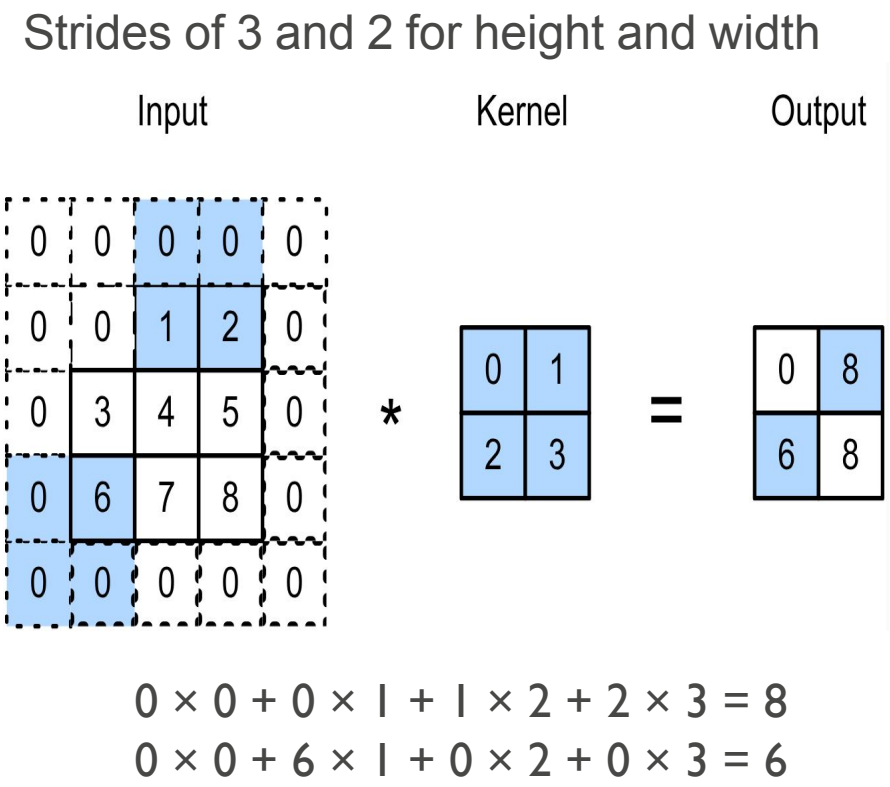
- Padding reduces shape linearly with #layers
  - Given a 224 x 224 input with a 5 x 5 kernel, needs 44 layers to reduce the shape to 4 x 4
  - Requires a large amount of computation

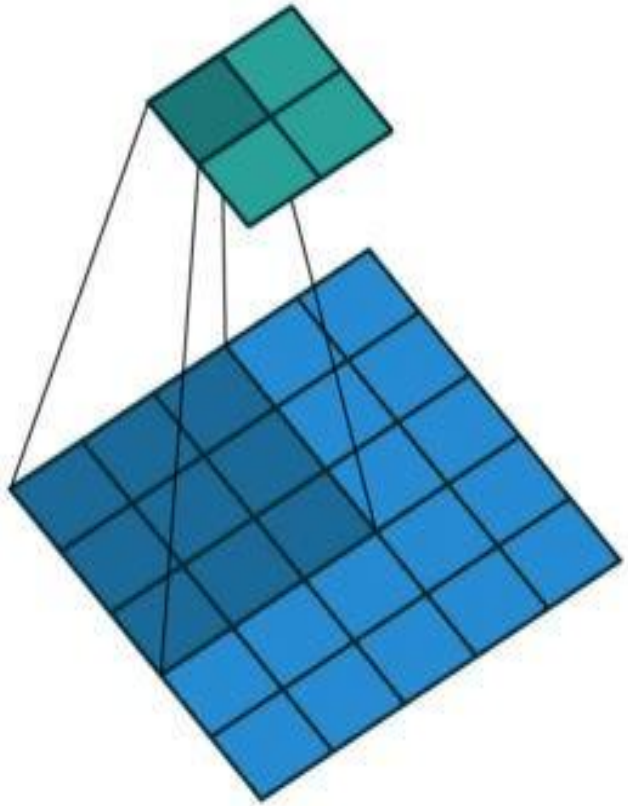


Stride

Sometimes, either for computational efficiency or because we wish to downsample, we move our window more than one element at a time, skipping the intermediate locations.

- Stride is the #rows/#columns per slide





Stride

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

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

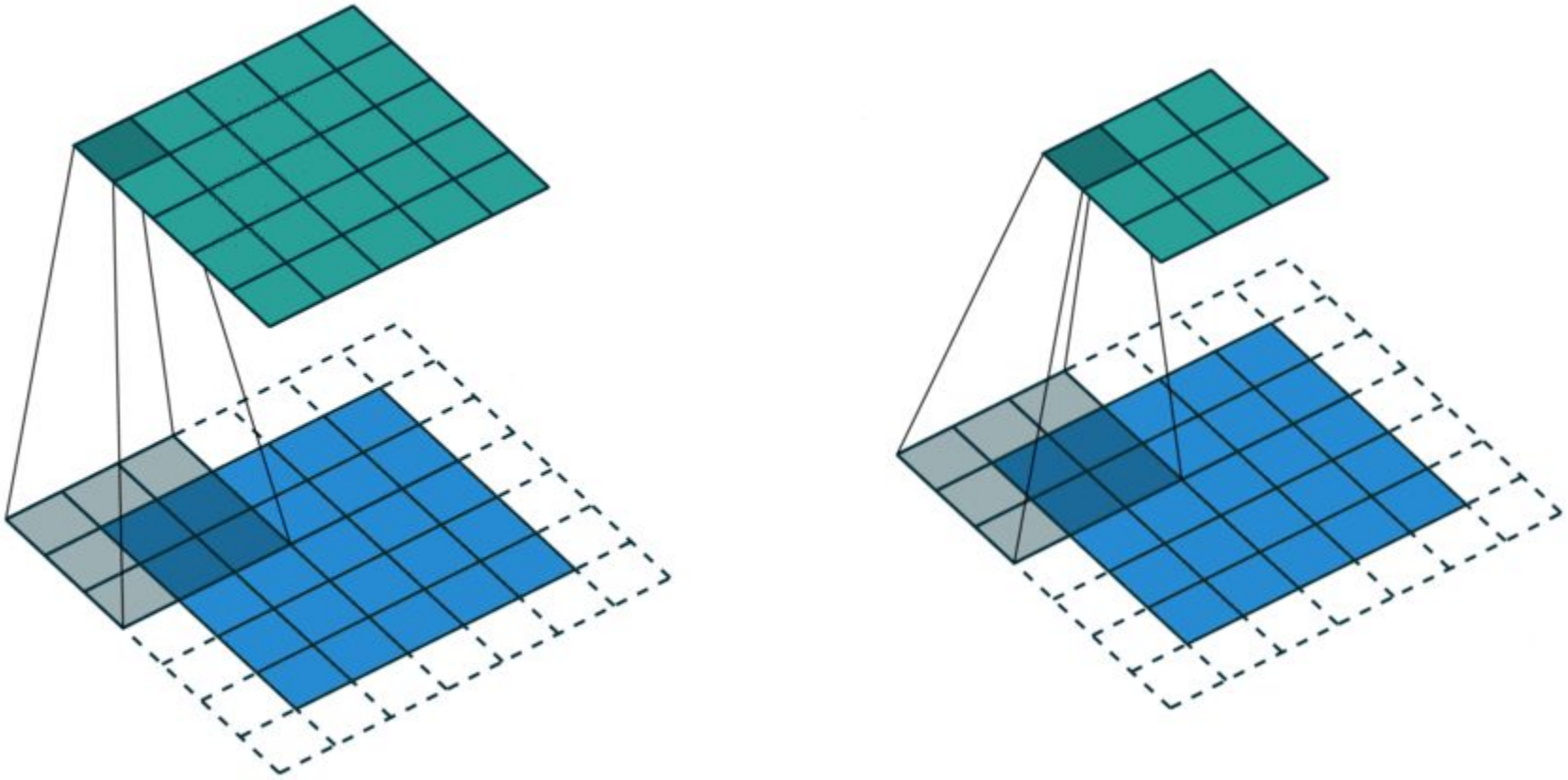
- With $p_h = k_h - 1$ and $p_w = k_w - 1$

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

- If input height/width are divisible by strides

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

Stride



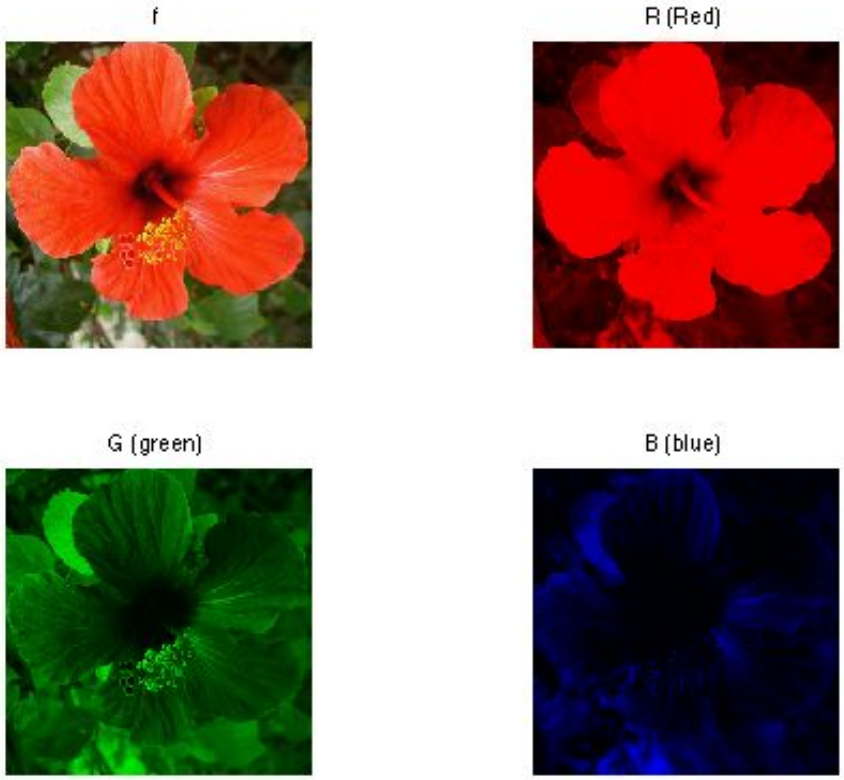

Multiple Input and
Output Channels

Multiple Input Channels

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

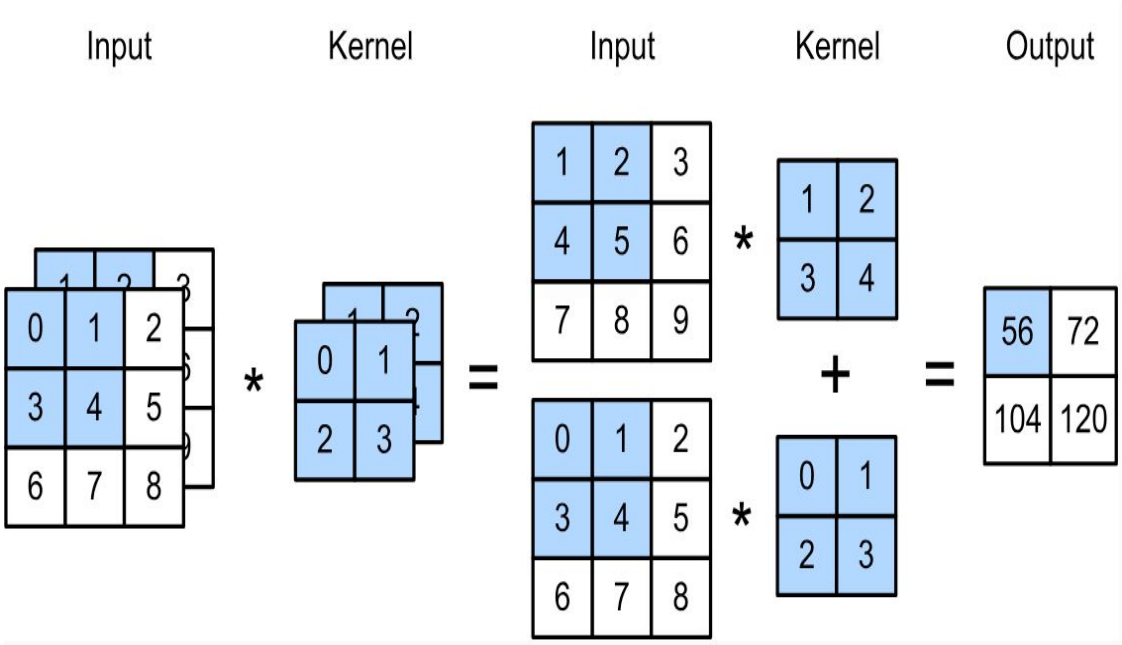
Multiple Input Channels

- 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



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

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

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

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

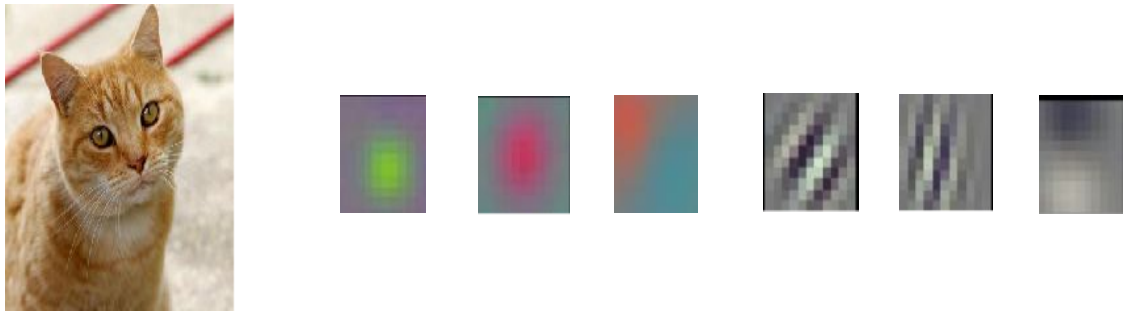
- Input $\mathbf{X} : c_i \times n_h \times n_w$
- Kernel $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- Output $\mathbf{Y} : c^o \times m^h \times m^w$

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

for $i = 1, \ldots, c_o$

Multiple Input/Output Channels

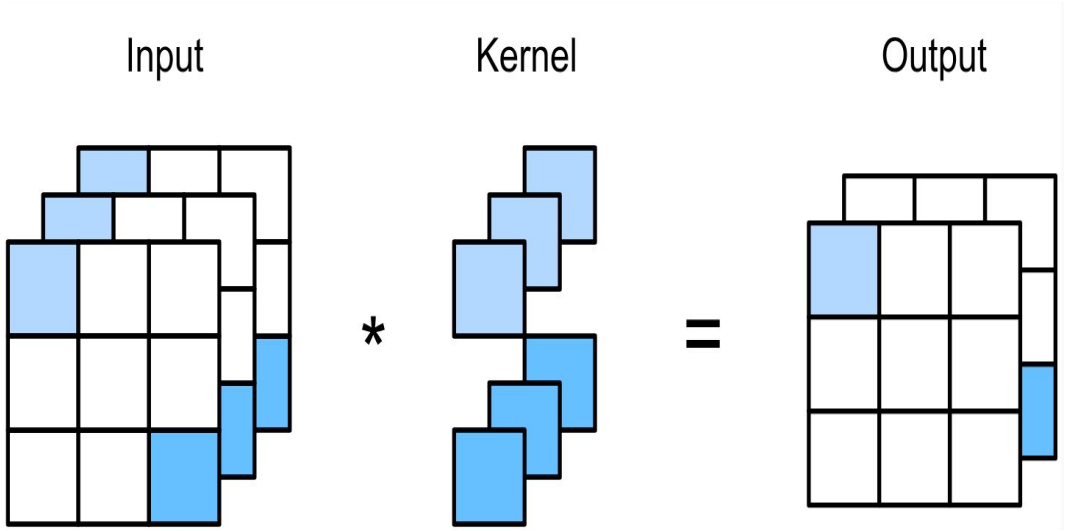
- Each output channel may recognize a particular pattern



- 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)

$c_i = c_o = 100$

$k_h = h_w = 5$

$m_h = m_w = 64$

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

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.
```

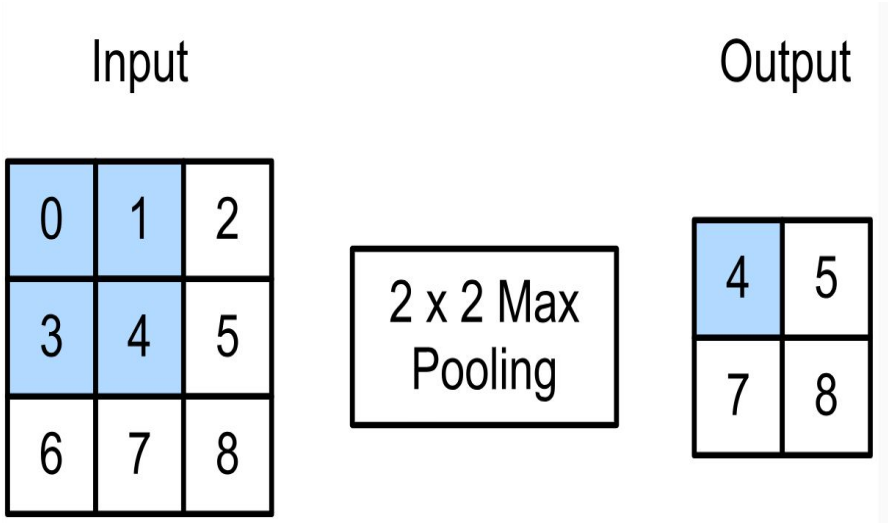
Y

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

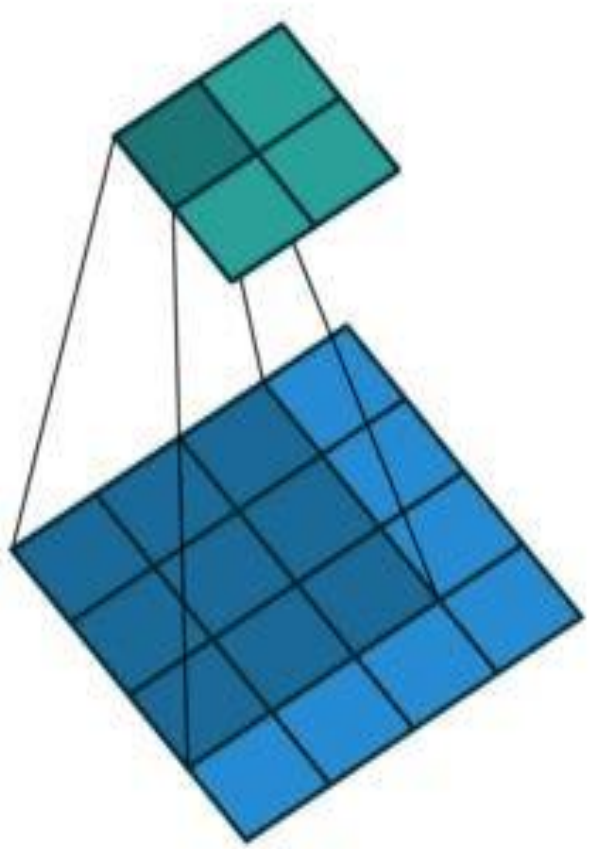
- We need some degree of invariance to translation
  - Lighting, object positions, scales, appearance vary among images

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

Vertical edge detection Conv output 2 x 2 max pooling

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

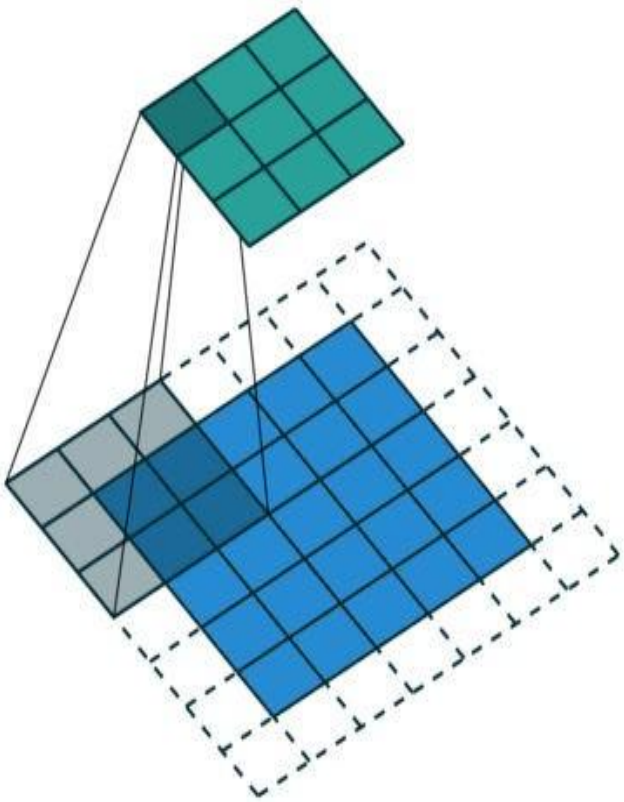
```
[[ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
```

```
[[ 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

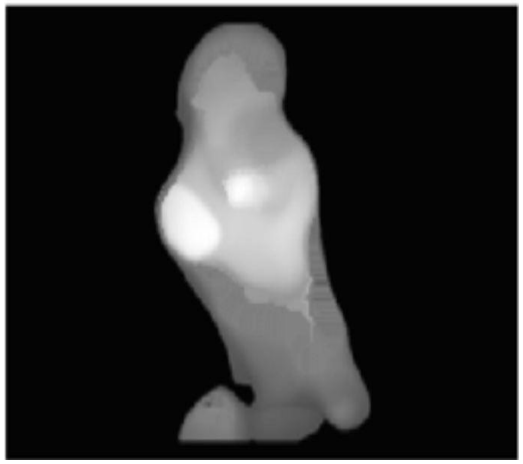


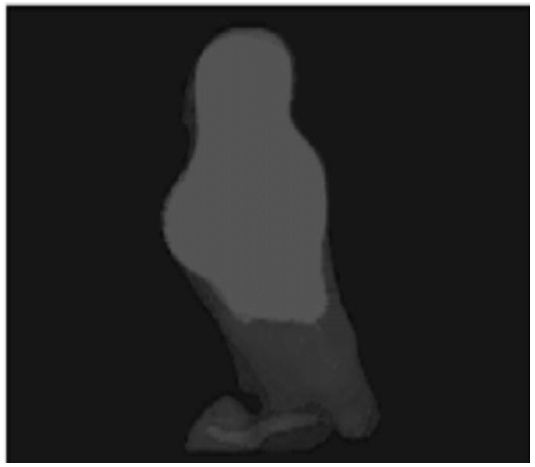
#output channels = #input channels

Average Pooling

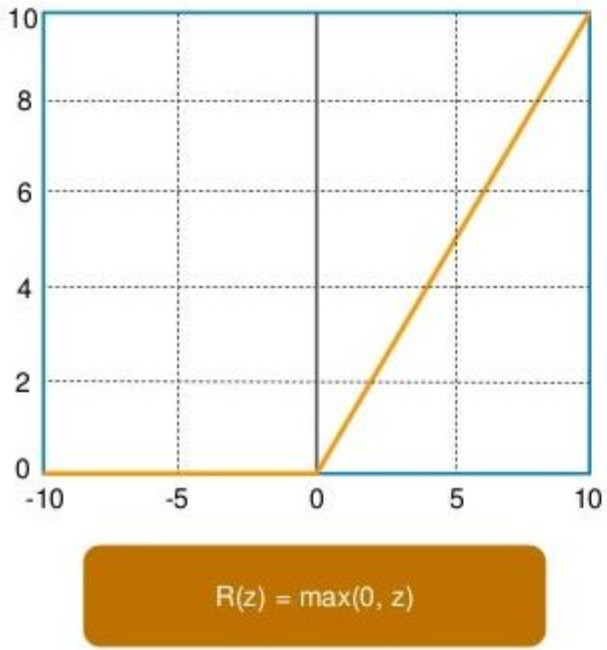
- Max pooling: the strongest pattern signal in a window
- Average pooling: replace max with mean in max pooling
  - The average signal strength in a window

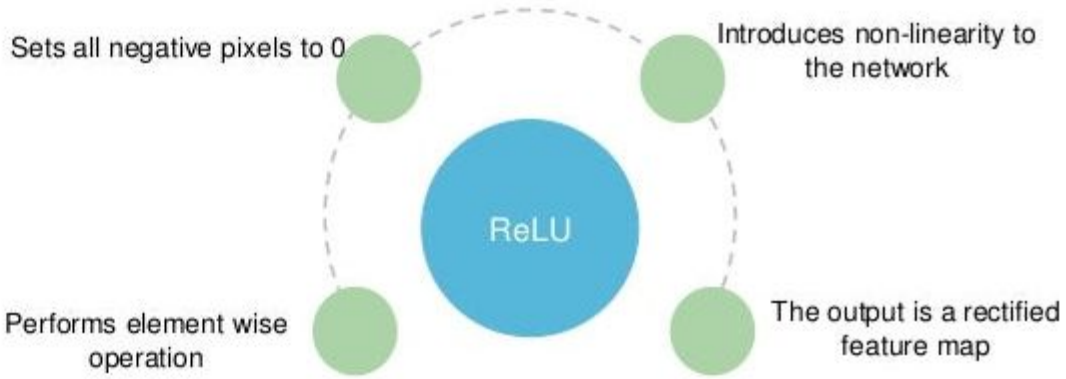
Max pooling Average pooling



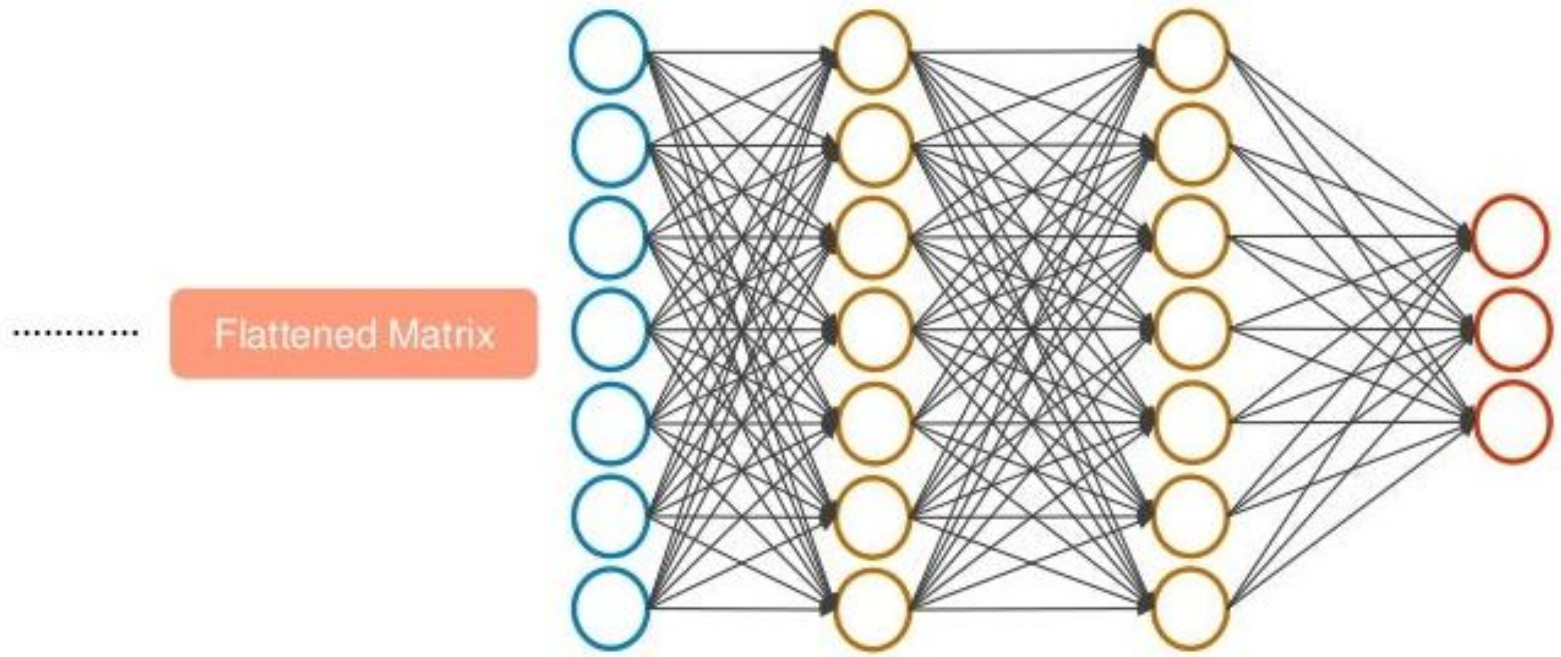


ReLU

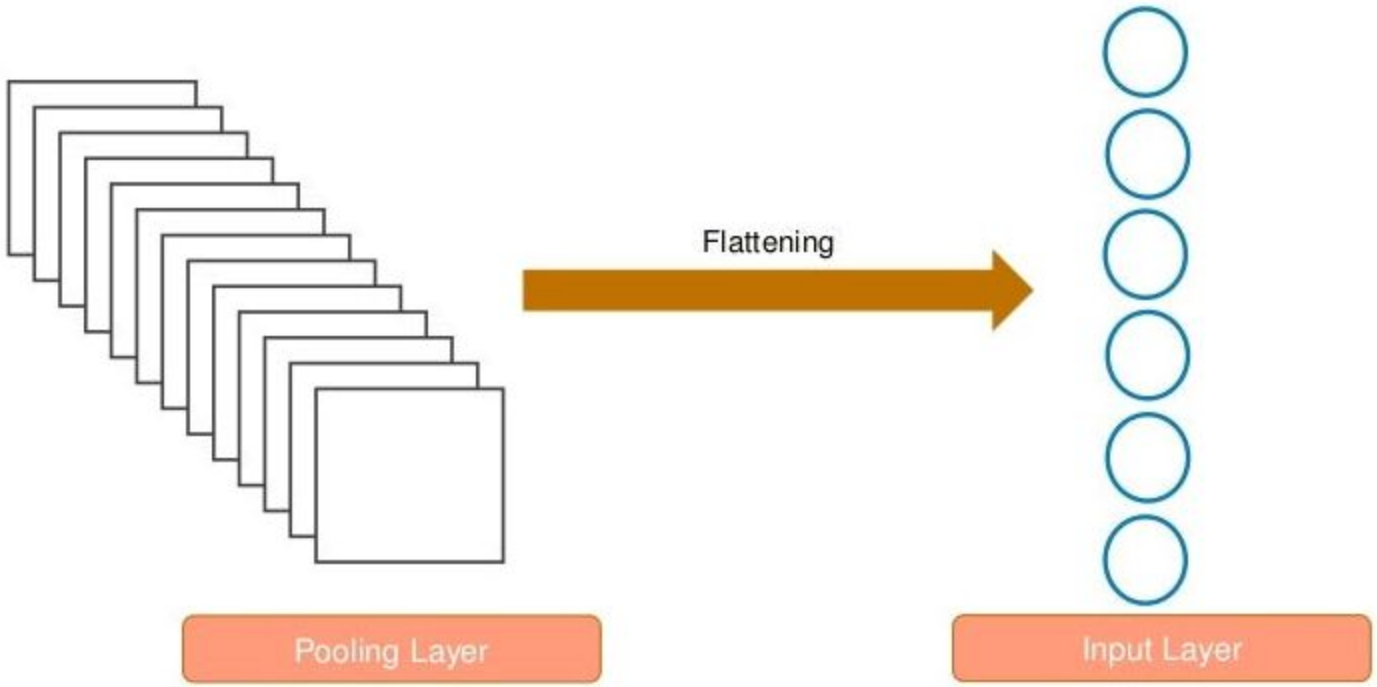

10
8
6
4
2
0
-10
-5
0
5
10
R(z) = max(0, z)


Sets all negative pixels to 0
Introduces non-linearity to the network
ReLU
Performs element wise operation
The output is a rectified feature map

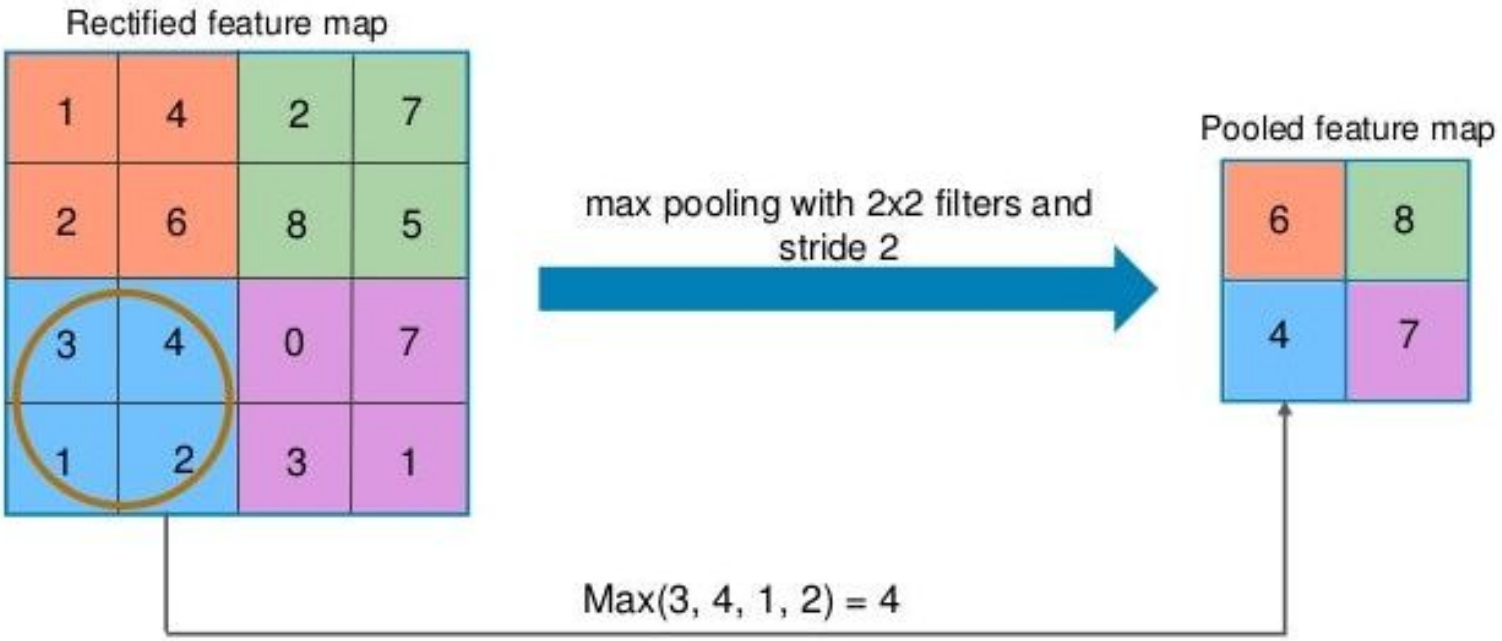
Fully connected layer



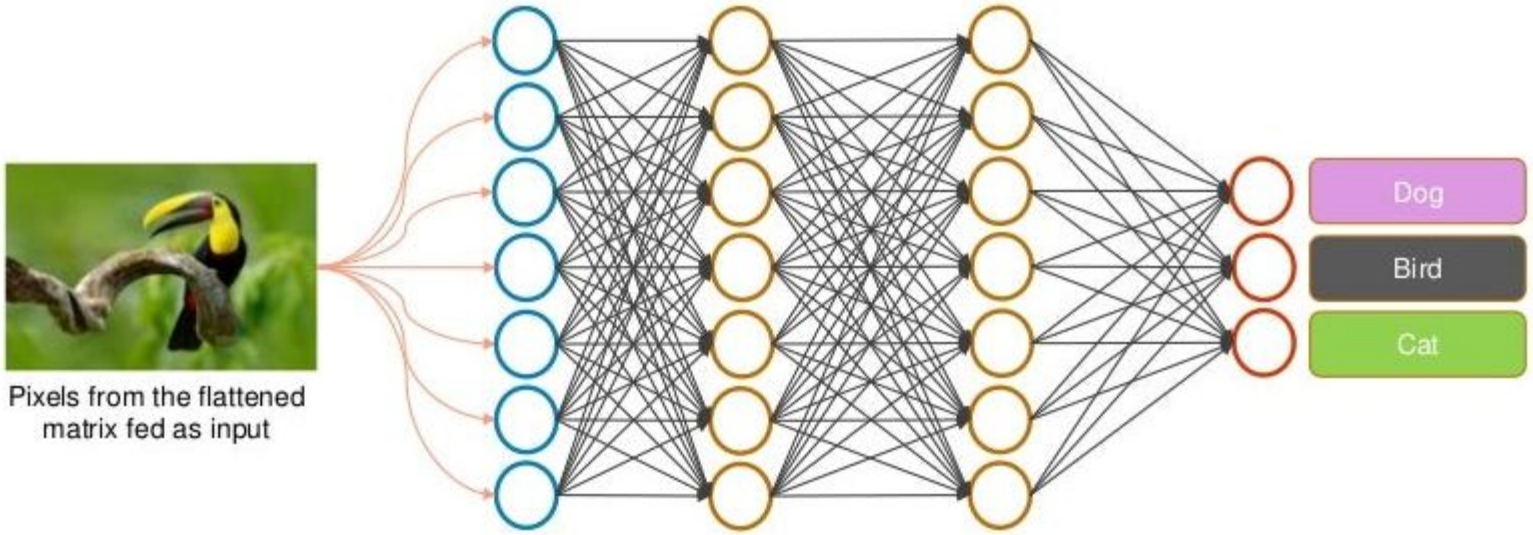
Flattening



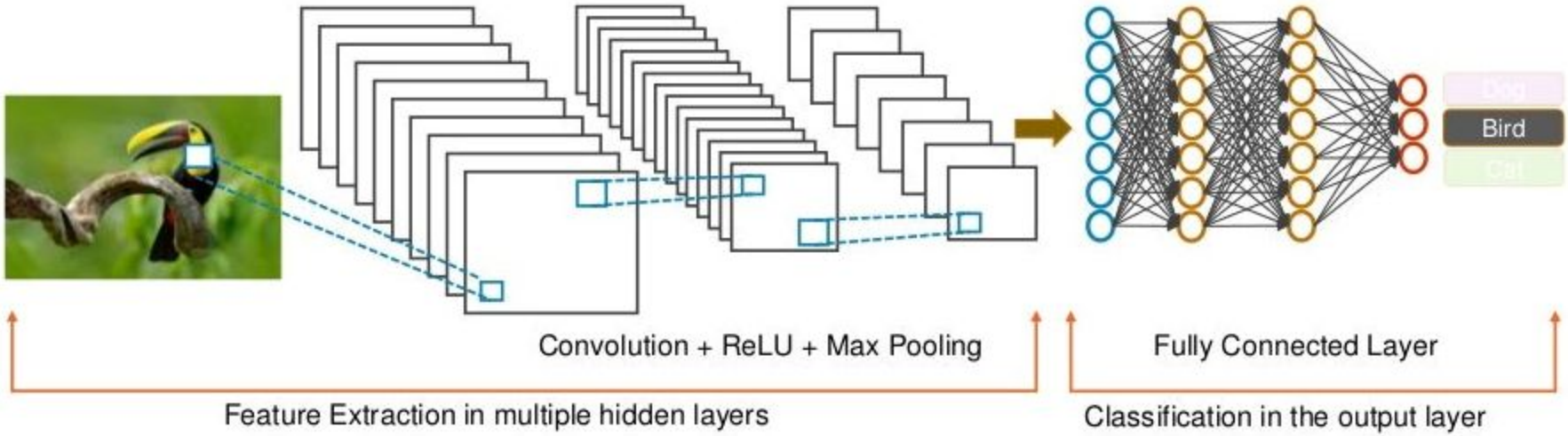
Max pooling

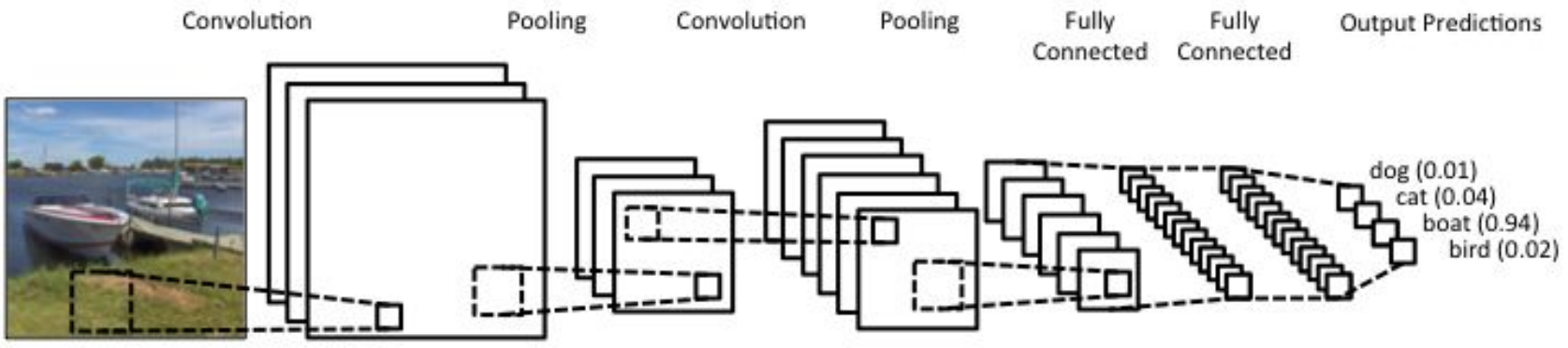

Rectified feature map
1 4 2 7
2 6 8 5
3 4 0 7
1 2 3 1
max pooling with 2x2 filters and stride 2
Pooled feature map
6 8
4 7
Max(3, 4, 1, 2) = 4

Fully Connected Layer

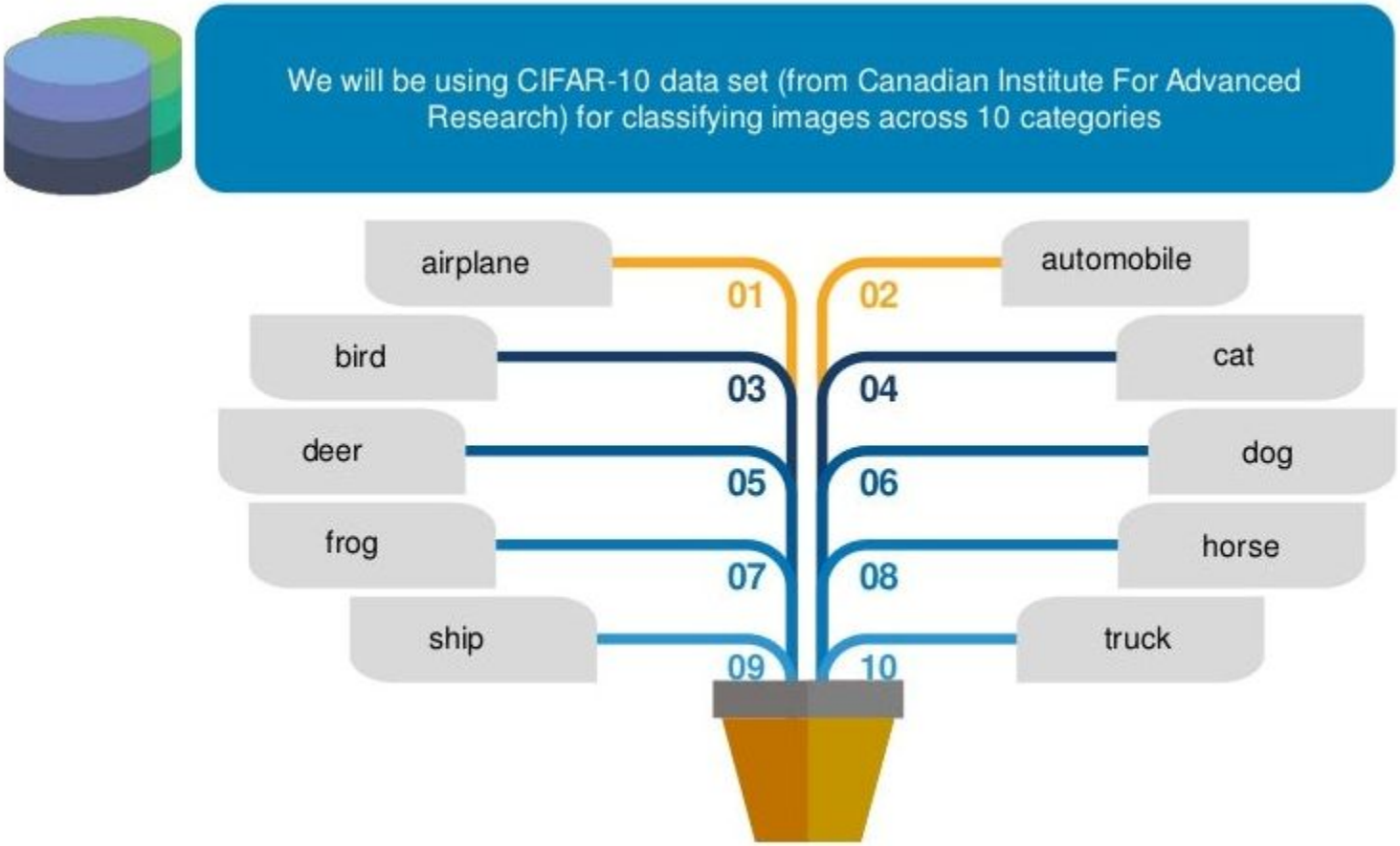


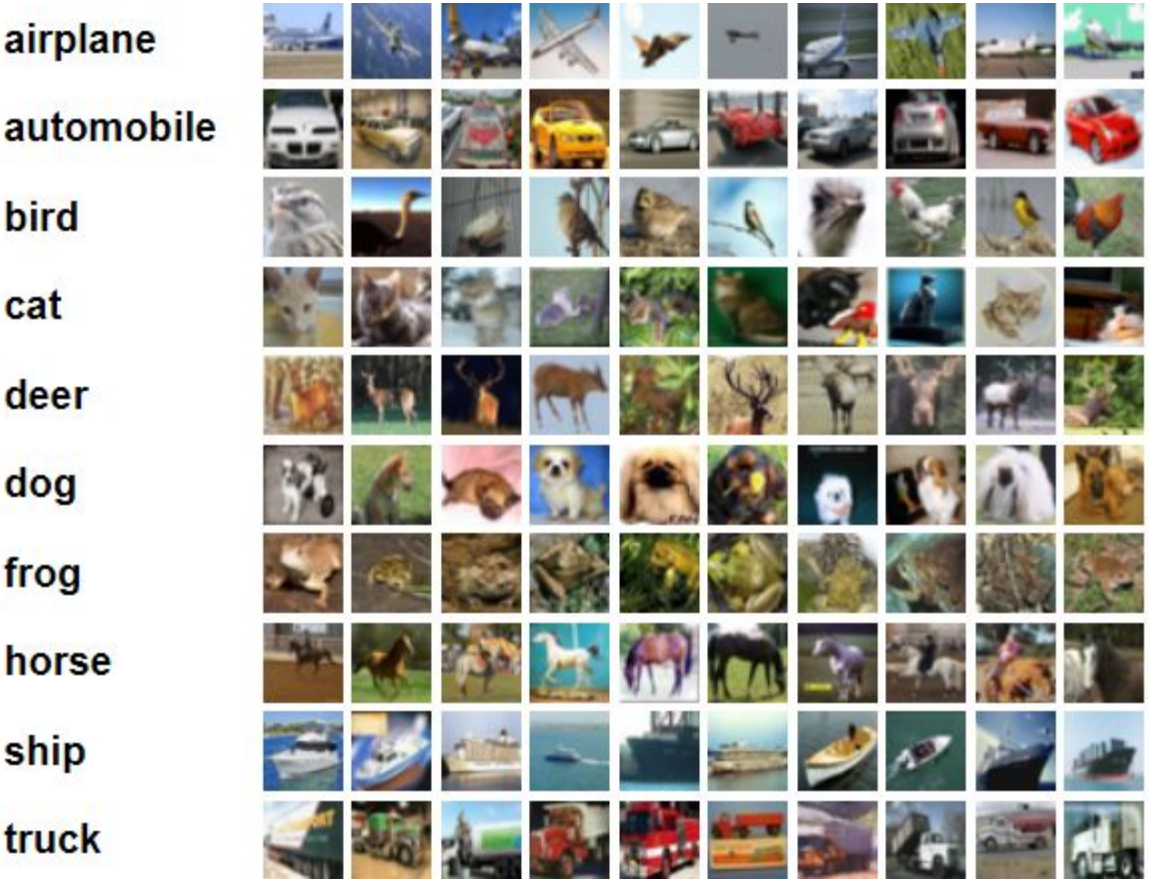
Final CNN Model



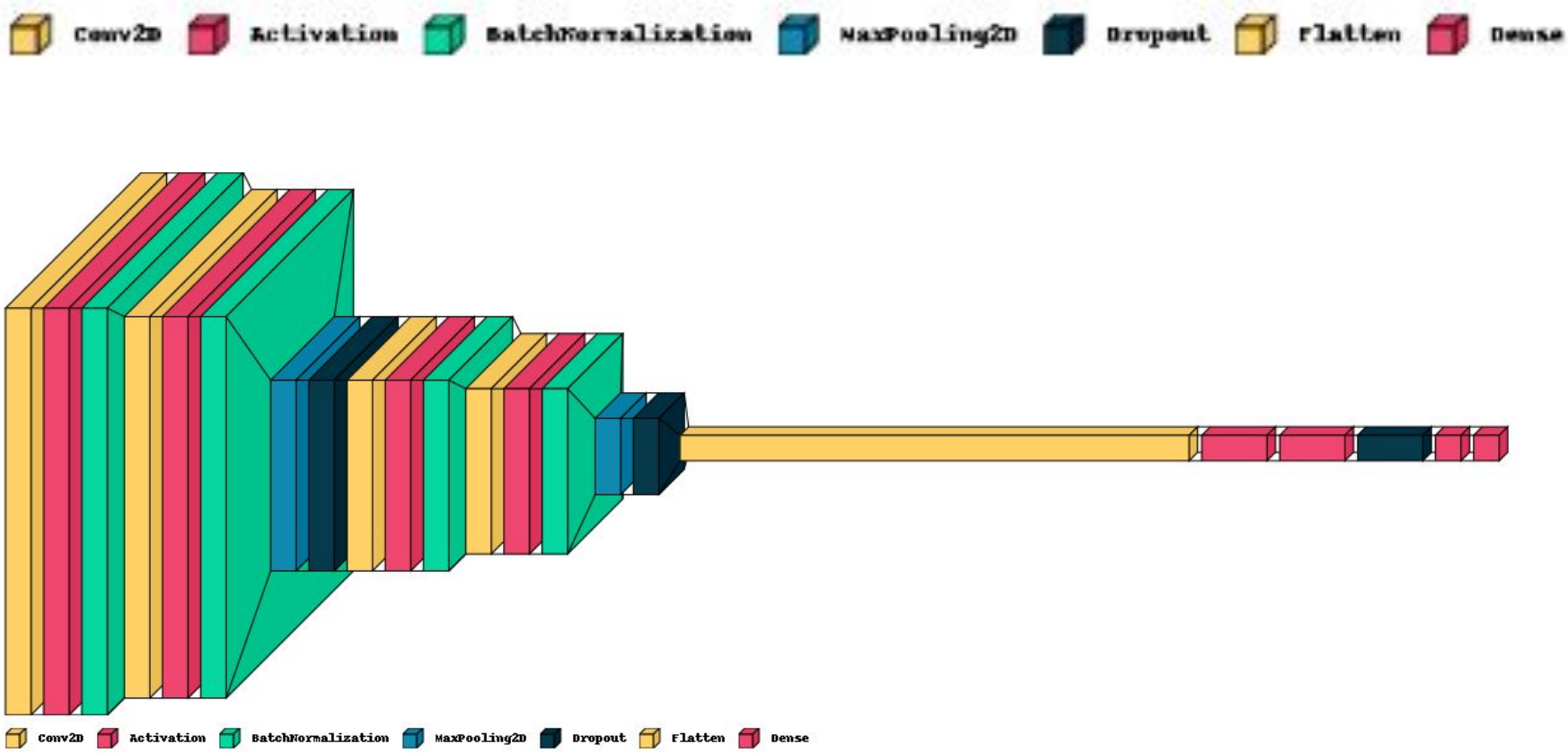

Convolution
Pooling
Convolution
Pooling
Fully Connected
Fully Connected
Output Predictions
dog (0.01)
cat (0.04)
boat (0.94)
bird (0.02)

Dataset


We will be using CIFAR-10 data set (from Canadian Institute For Advanced Research) for classifying images across 10 categories
airplane
01
automobile
02
bird
03
cat
04
deer
05
dog
06
frog
07
horse
08
ship
09
truck
10


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

Model


Conv2D
Activation
BatchNormalization
MaxPooling2D
Dropout
Flatten
Dense
Conv2D
Activation
BatchNormalization
MaxPooling2D
Dropout
Flatten
Dense

Model parameters

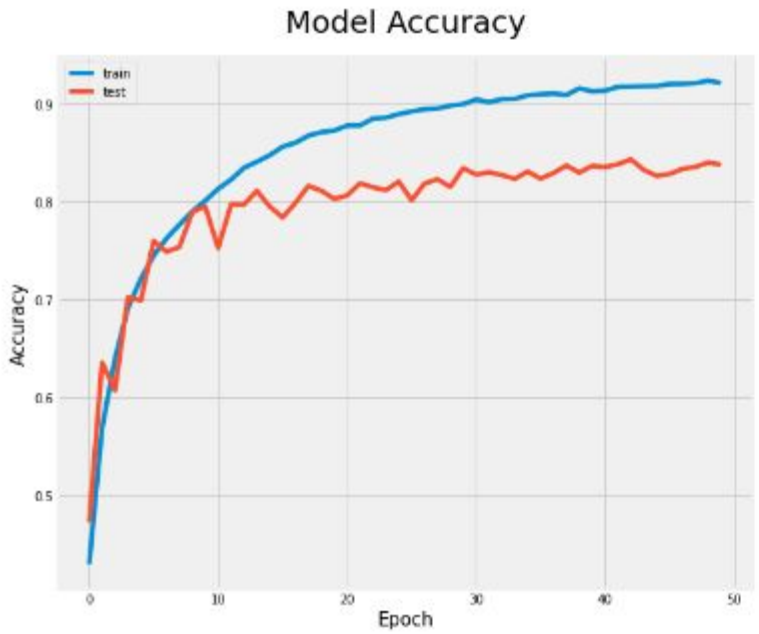
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 64)        1792
_________________________________________________________________
activation (Activation)      (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 64)        256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 64)        36928
_________________________________________________________________
```

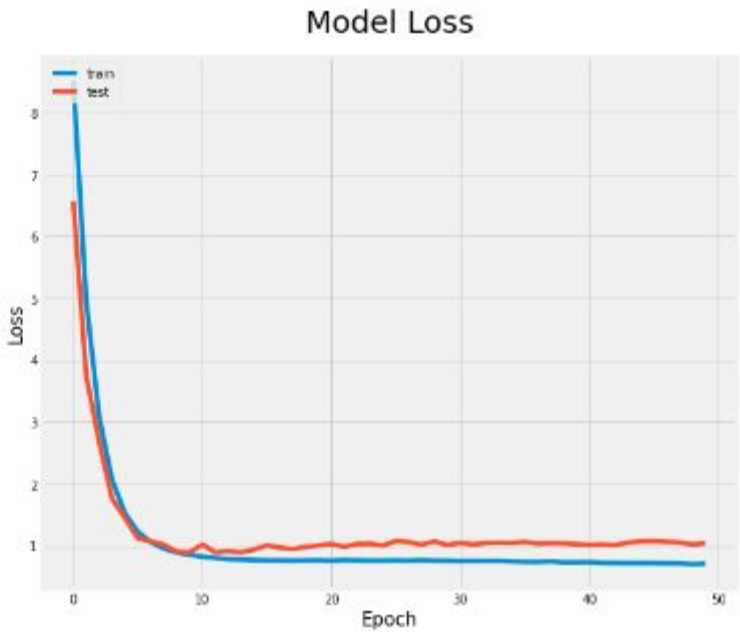
```
_________________________________________________________________
activation_4 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 2,626,634
Trainable params: 2,625,866
Non-trainable params: 768
_________________________________________________________________
None
```

Training

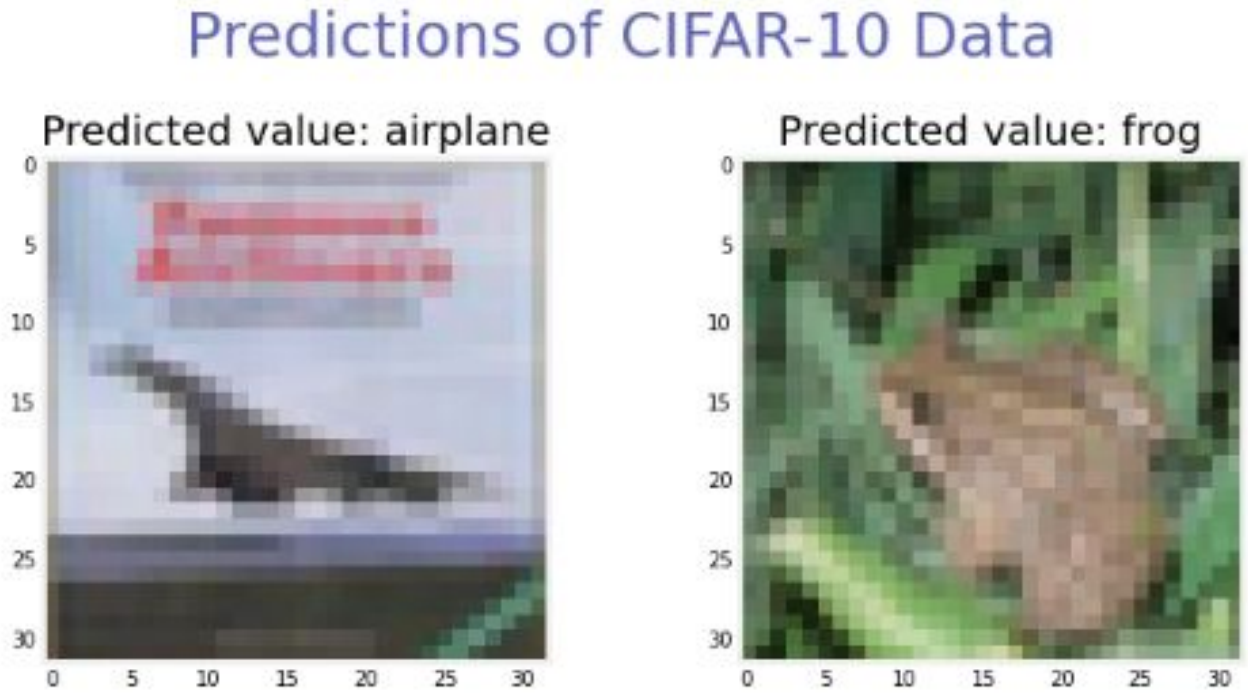
```
2022-06-20 09:25:04.301225: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimi
zation Passes are enabled (registered 2)
Epoch 1/50
12500/12500 [==============================] - 513s 41ms/step - loss: 4.2169 - accuracy: 0.3639 - val_loss: 4.6308
- val_accuracy: 0.1687
Epoch 2/50
12500/12500 [==============================] - 499s 40ms/step - loss: 2.0688 - accuracy: 0.5234 - val_loss: 2.8501
- val_accuracy: 0.2733
Epoch 3/50
 9207/12500 [=====================>........] - ETA: 2:02 - loss: 1.8332 - accuracy: 0.5930
```

Accuracy and Loss





Prediction


Predictions of CIFAR-10 Data
Predicted value: airplane
Predicted value: frog
0
5
10
15
20
25
30

Batch Normalization

The regularization techniques help to improve a model and allows it to converge faster.

We have several regularization tools at our end, some of them

- early stopping,
- dropout,
- weight initialization techniques,
- batch normalization.

The regularization helps in preventing the over-fitting of the model and the learning process becomes more efficient.

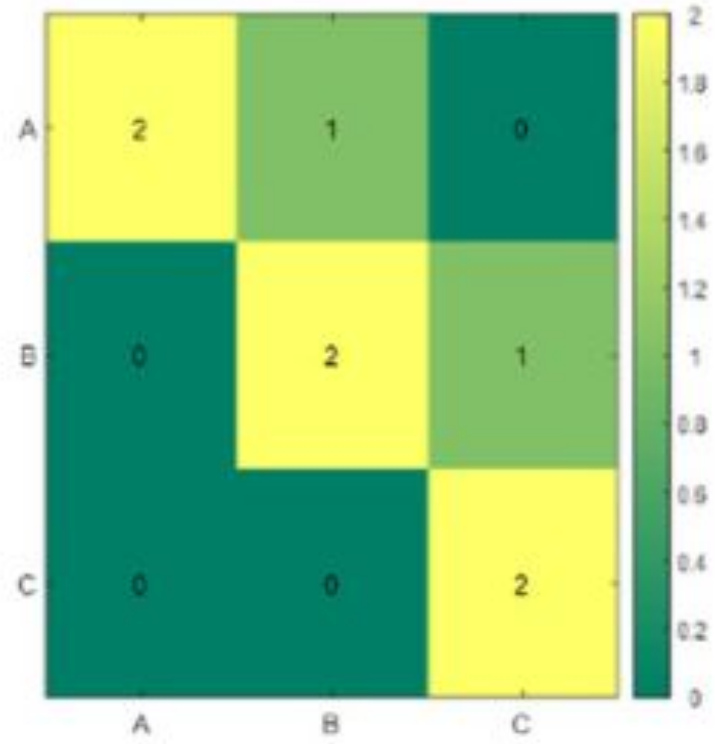
Regularization

Sometimes situation occurs when the model is performing very well on the training data but is unable to predict the test data accurately. The reason is your model is over fitting.

The solution to such a problem is regularization.

Pooling Notebook

```
cm =
     2     1     0
     0     2     1
     0     0     2
grp =
    A
    B
    C
```



Confusion Matrix

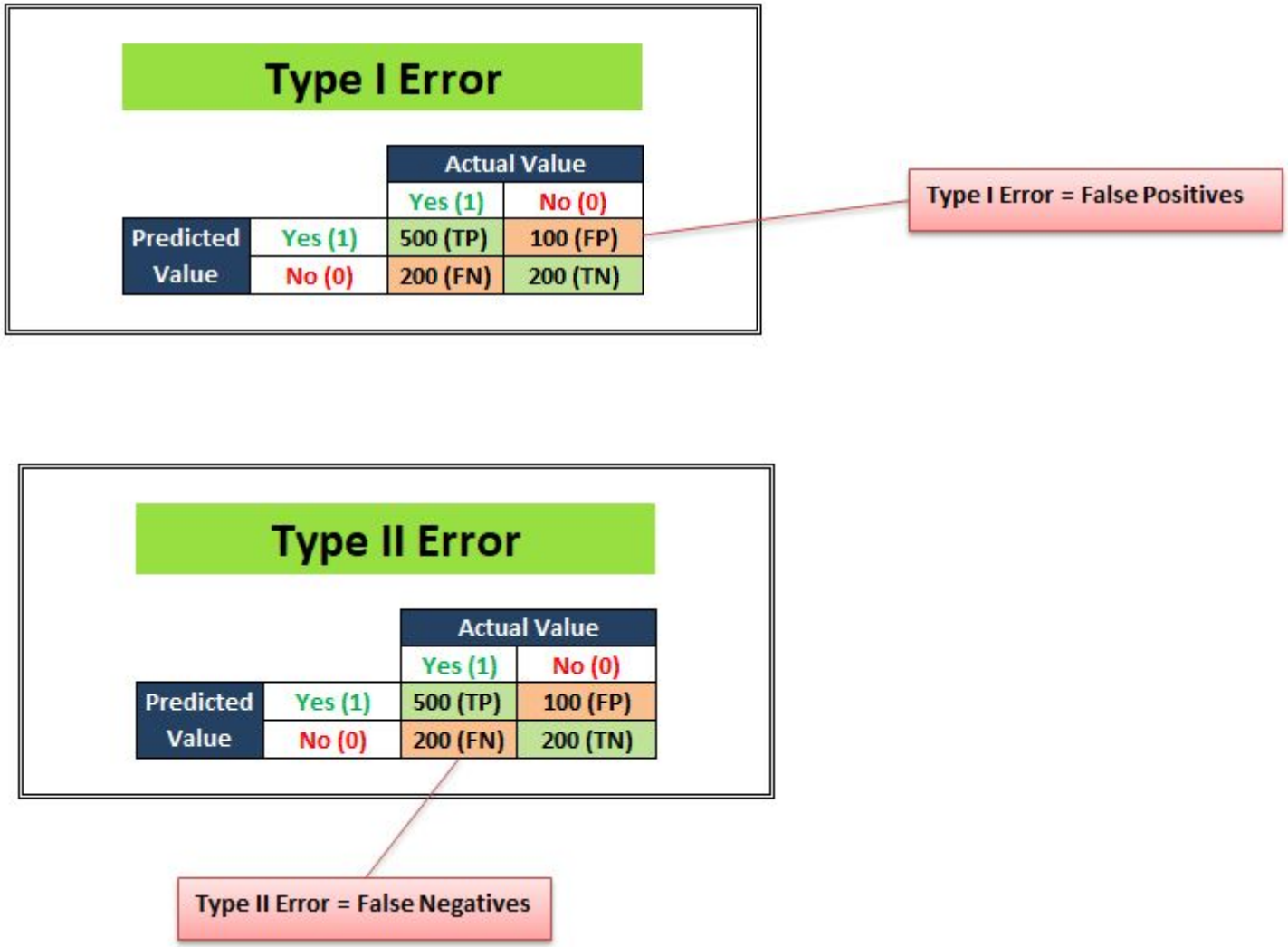
		Actual Value	
		Yes (1)	No (0)
Predicted Value	Yes (1)	TP	FP
	No (0)	FN	TN

TP= True Positive

FP= False Positive

FN= False Negative

TN= True Negative


Type I Error
Actual Value
Yes (1)
No (0)
Predicted Value
Yes (1)
No (0)
500 (TP)
100 (FP)
200 (FN)
200 (TN)
Type I Error = False Positives
Type II Error
Actual Value
Yes (1)
No (0)
Predicted Value
Yes (1)
No (0)
500 (TP)
100 (FP)
200 (FN)
200 (TN)
Type II Error = False Negatives

Accuracy

		Actual Value	
		Yes (1)	No (0)
Predicted Value	Yes (1)	500 (TP)	100 (FP)
	No (0)	200 (FN)	200 (TN)

Accuracy= (TP+TN)/(TP+FP+FN+TN)

=(500+200)/(500+100+200+200)

70%

Accuracy= (TP+TN)/Total customers

Receiver Operating Characteristics curve

- The values described are used to calculate different measurements of the quality of the test.

- The first one is **sensitivity, SE**, which is the probability of having a positive test among the patients who have a positive diagnosis.

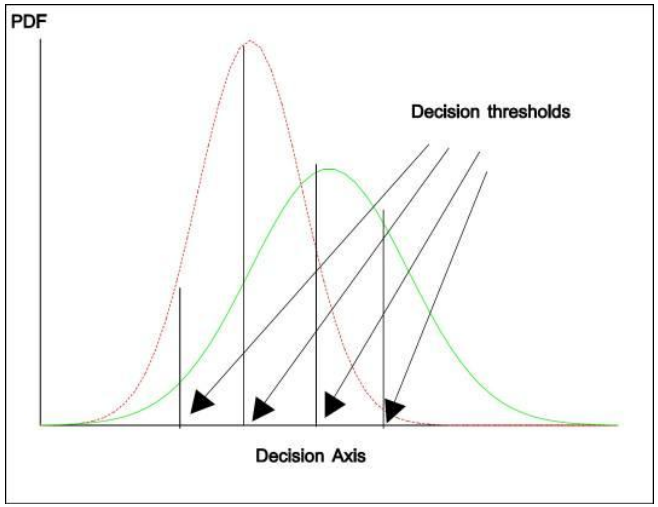
$$\mathrm{SE} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN}) = \mathrm{TP}/P.$$

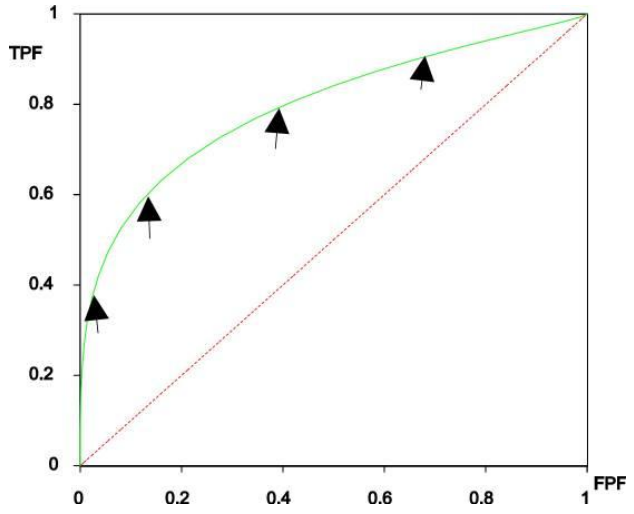
- **Specificity, SP**, is the probability of having a negative test among the patients who have a negative diagnosis.

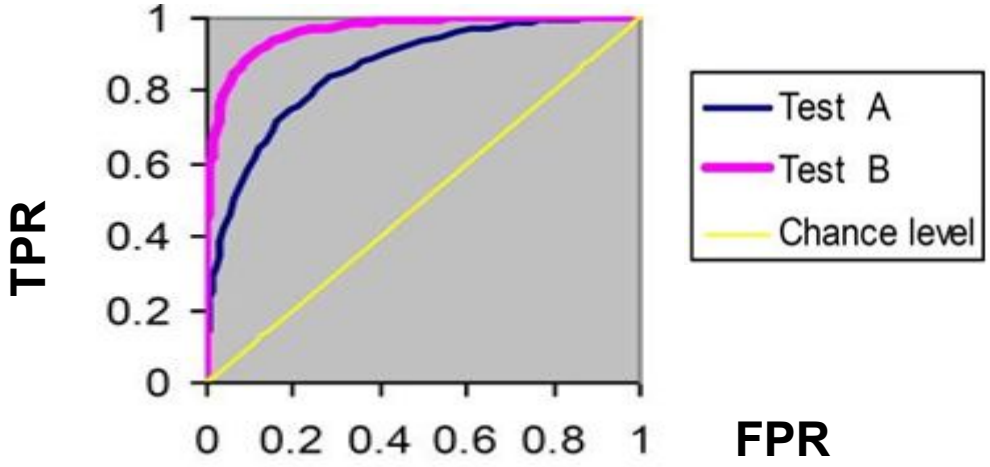
$$\mathrm{SP} = \mathrm{TN}/(\mathrm{FP} + \mathrm{TN}) = \mathrm{TN}/P'.$$

- **Efficiency** is defined as , EFF = TP + TN.

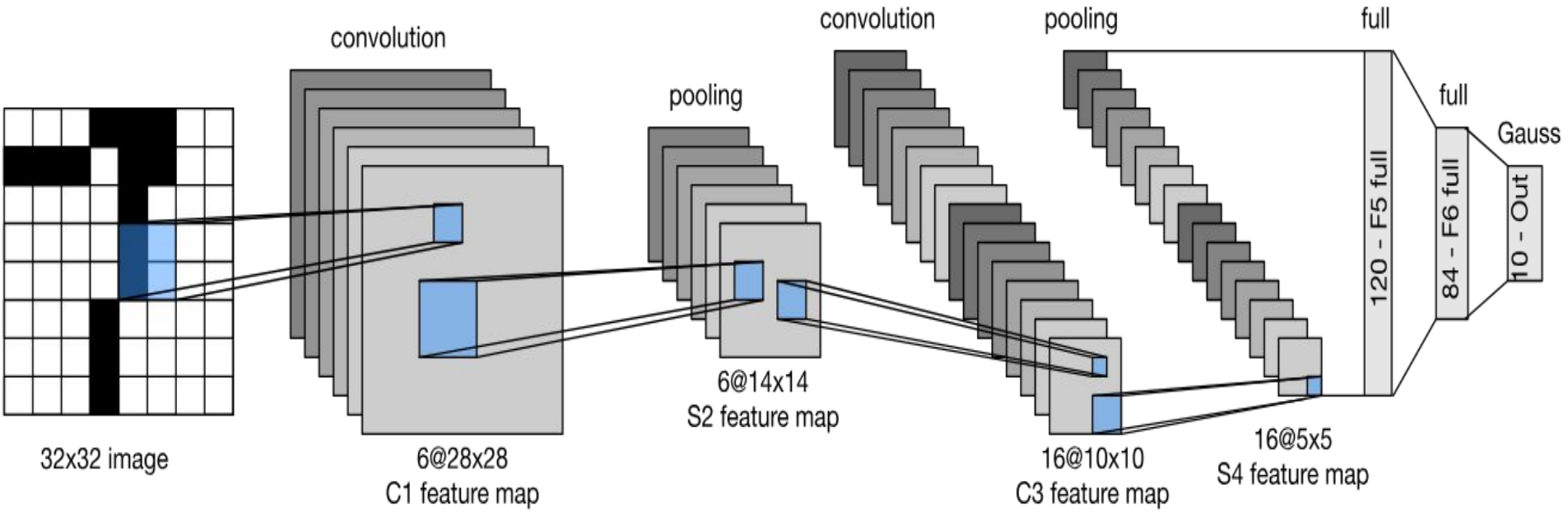
Receiver Operating Characteristics curve



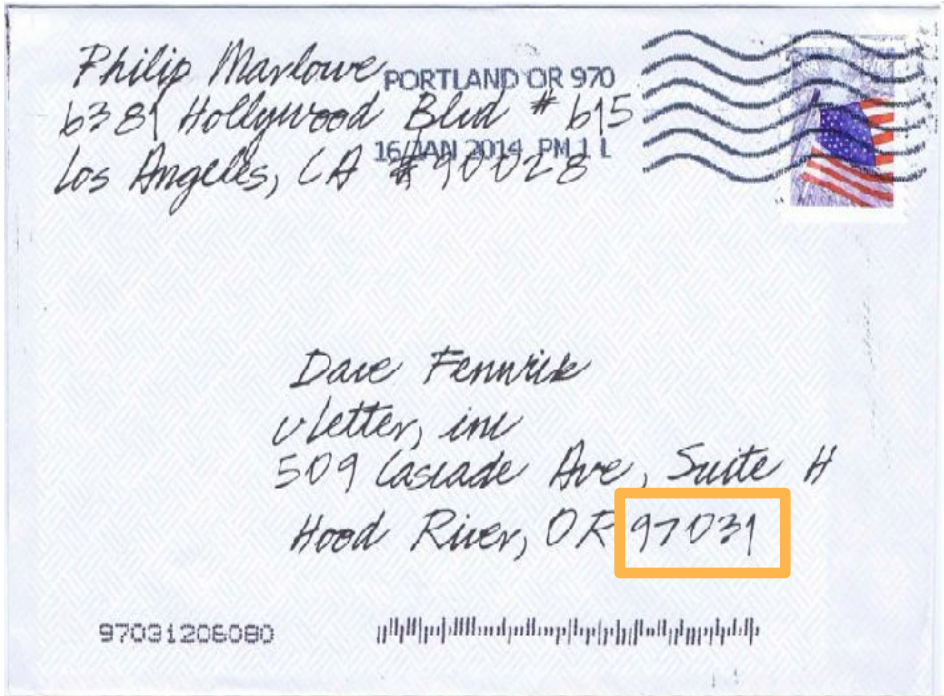


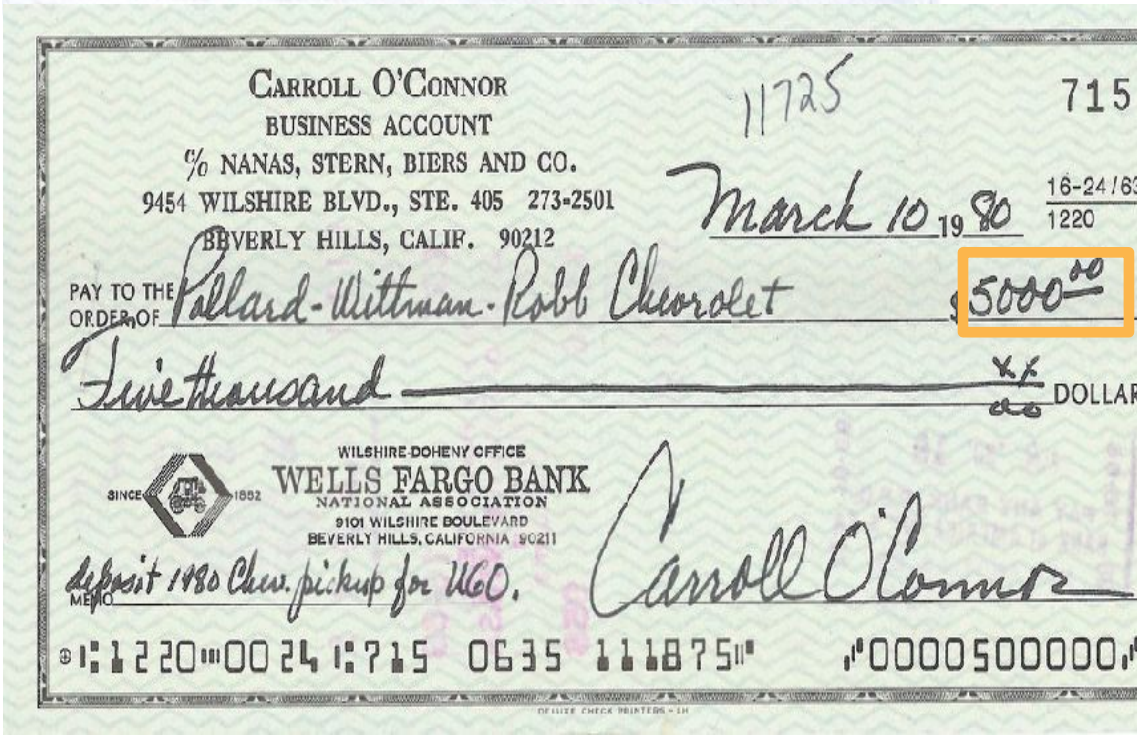


LeNet


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

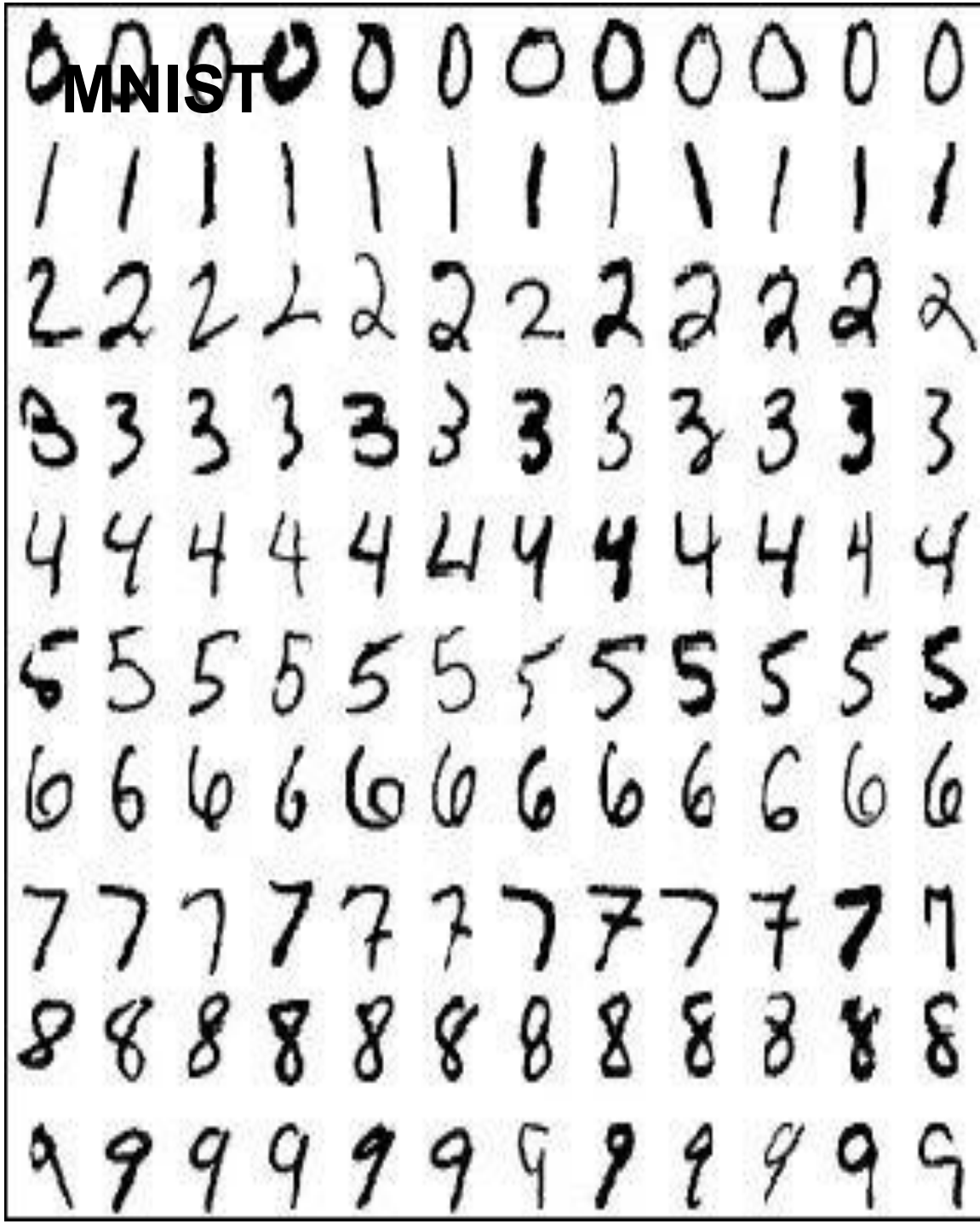
Handwritten Digit Recognition

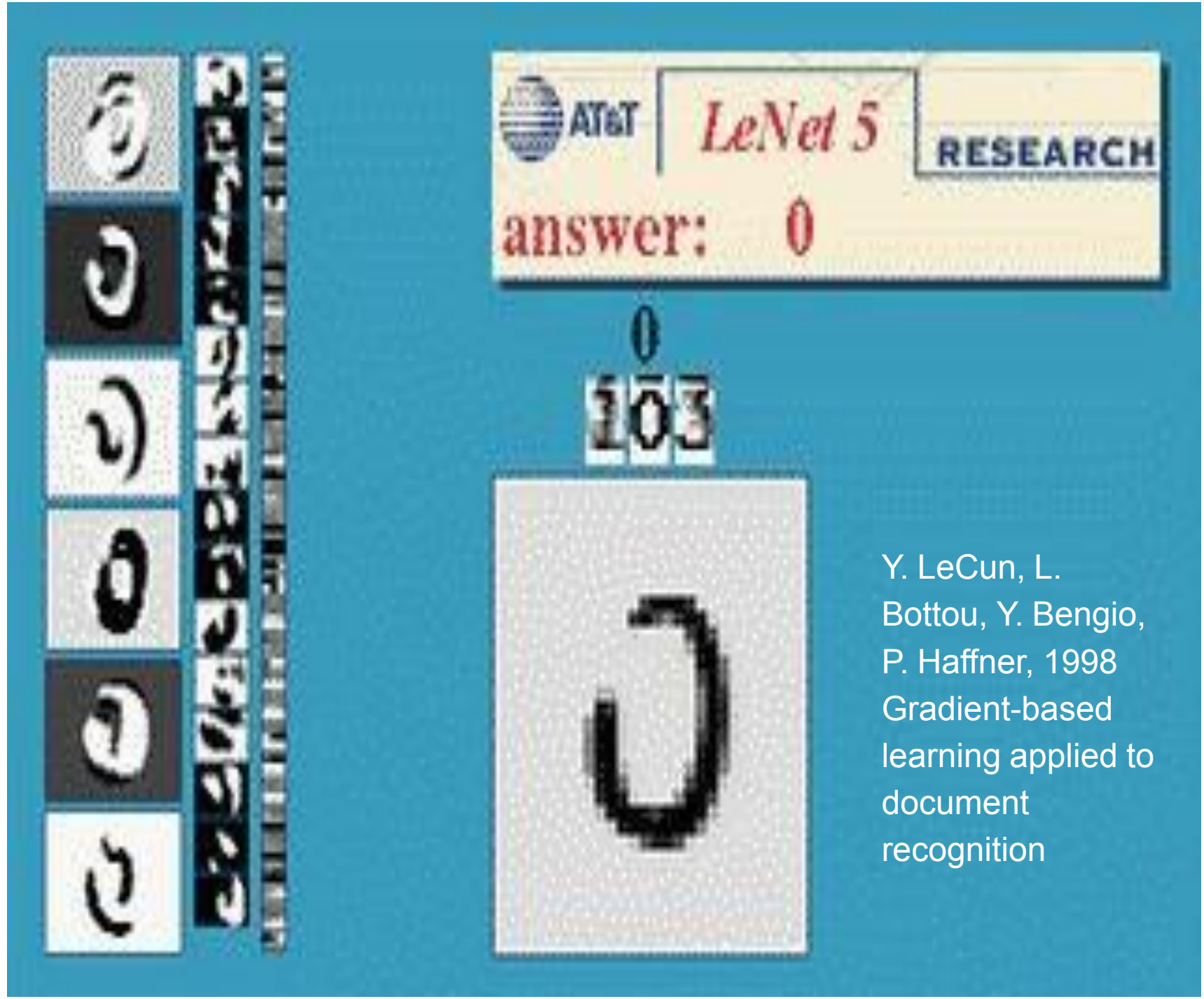




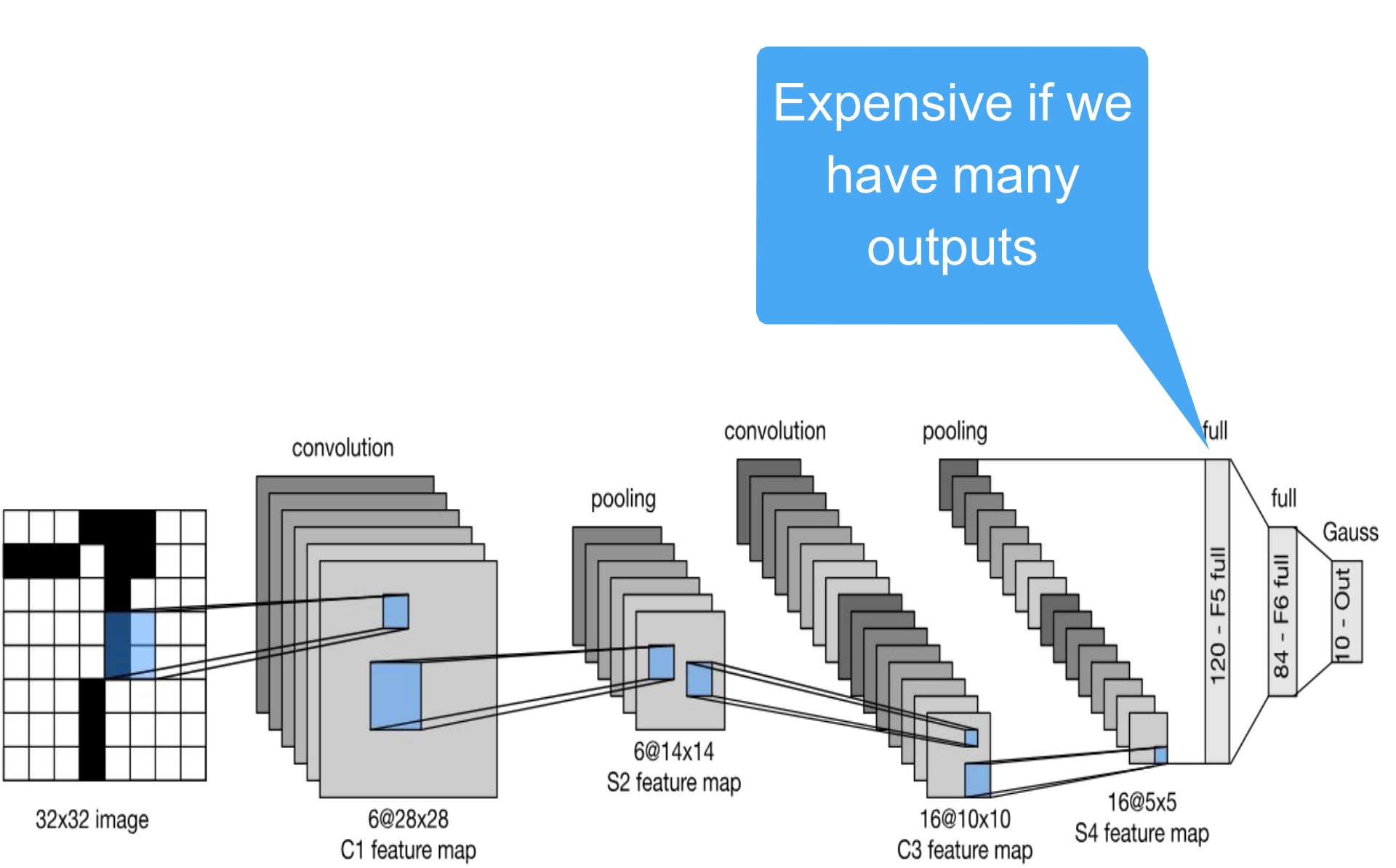
MNIST

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



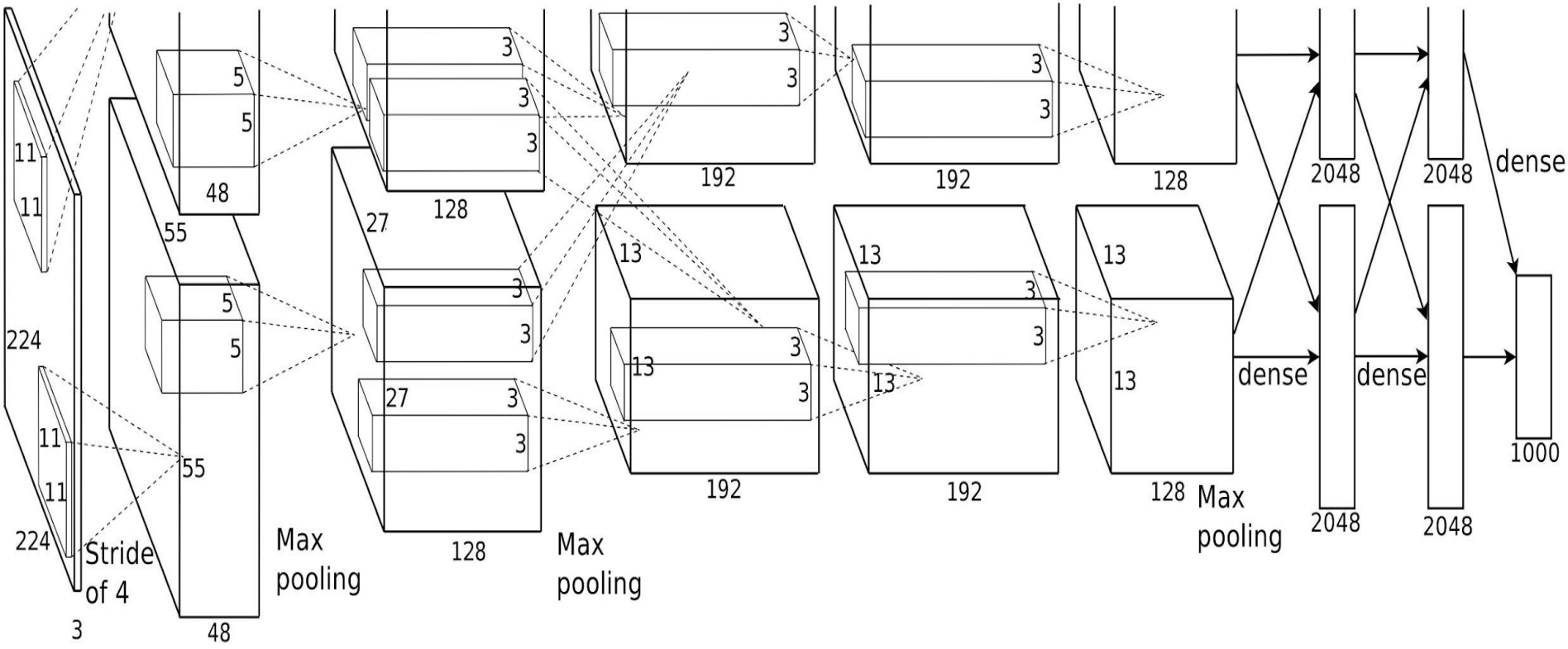


Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998 Gradient-based learning applied to document recognition


Expensive if we have many outputs
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

LeNet Notebook

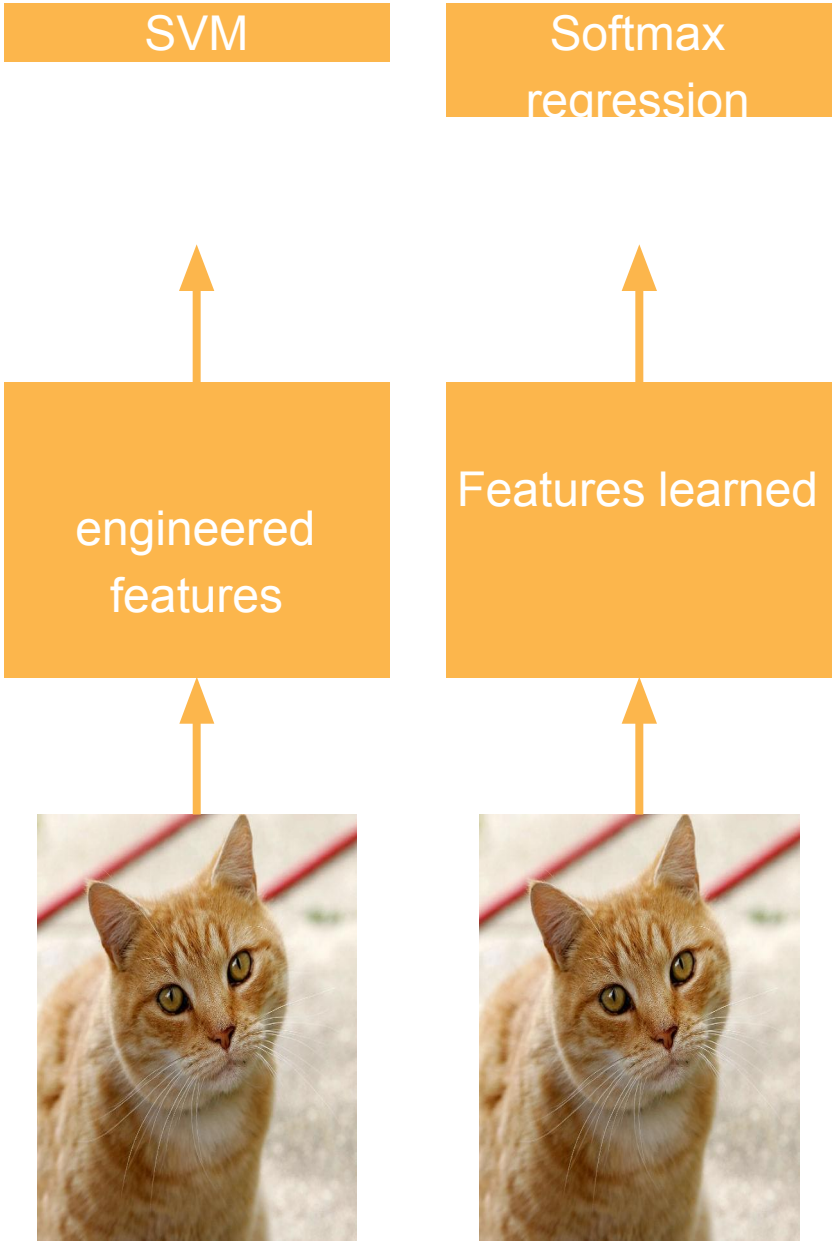
AlexNet


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

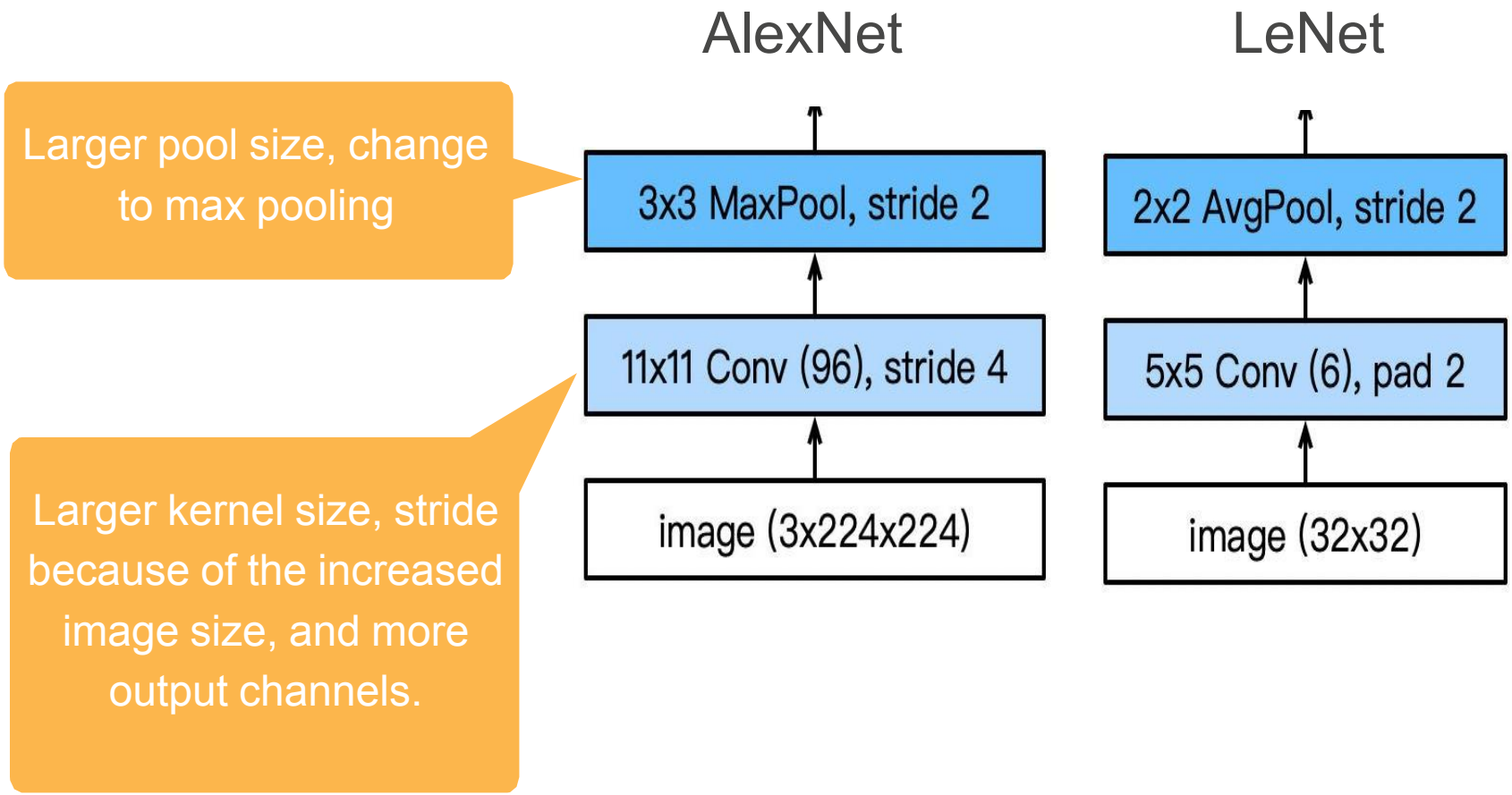
AlexNet

- AlexNet won ImageNet competition in 2012

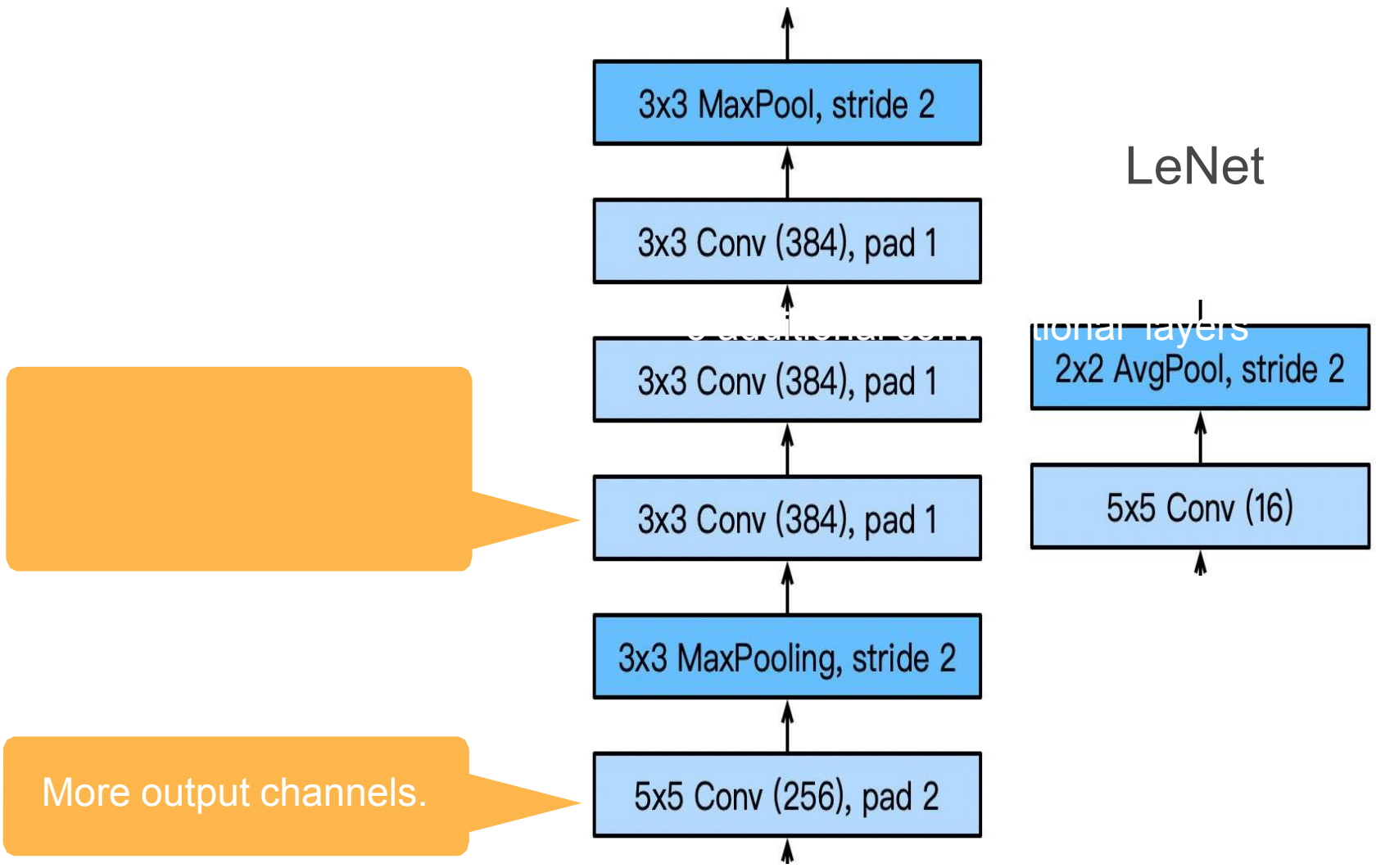
- Key modifications
  - Dropout (regularization)
  - ReLu (training)
  - MaxPooling
- Paradigm shift for computer vision



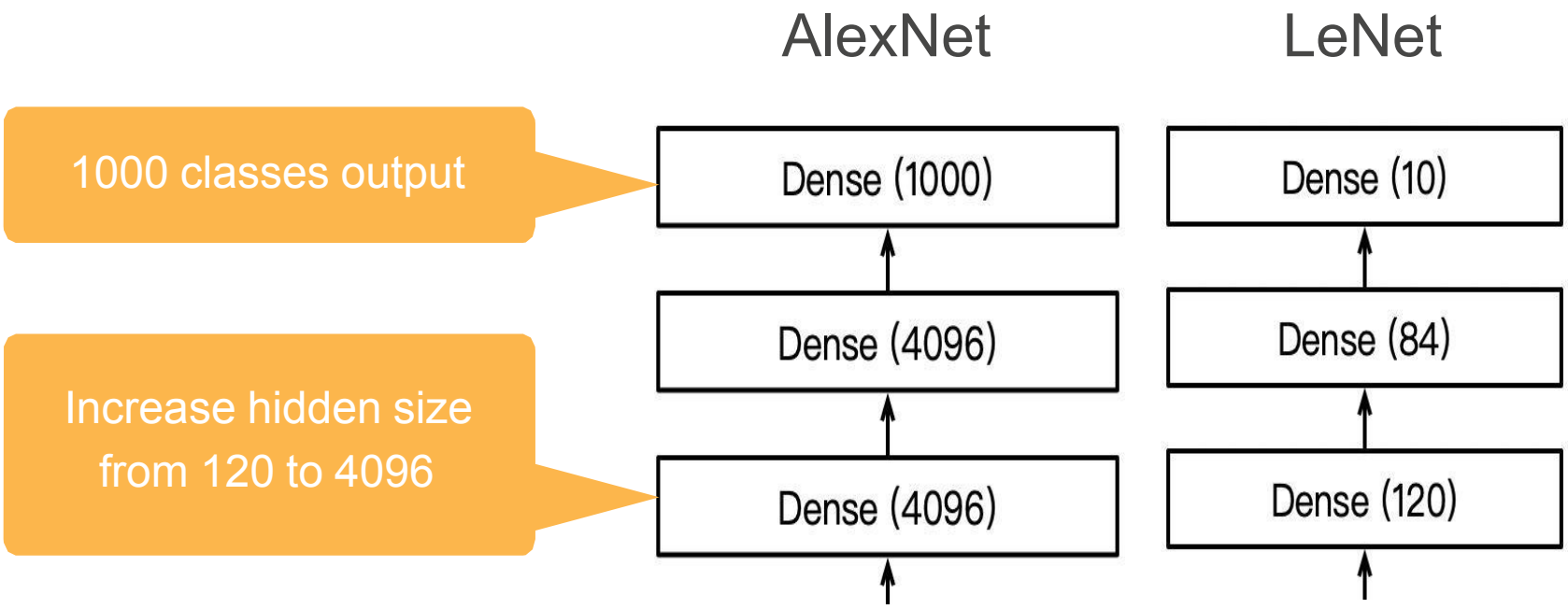
AlexNet Architecture



AlexNet Architecture

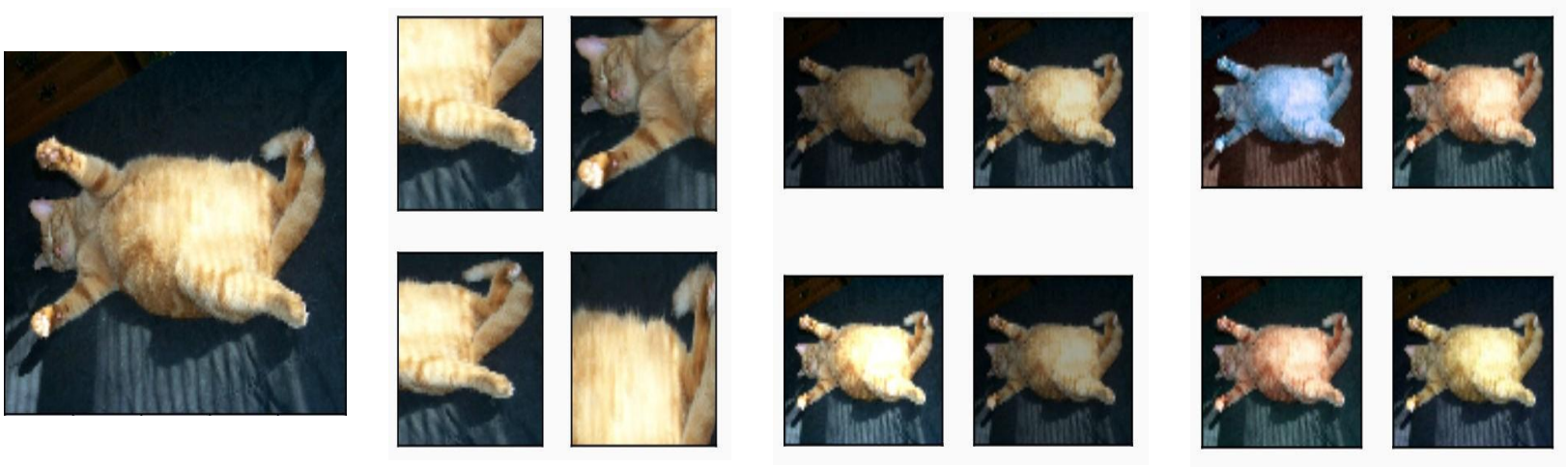


AlexNet Architecture



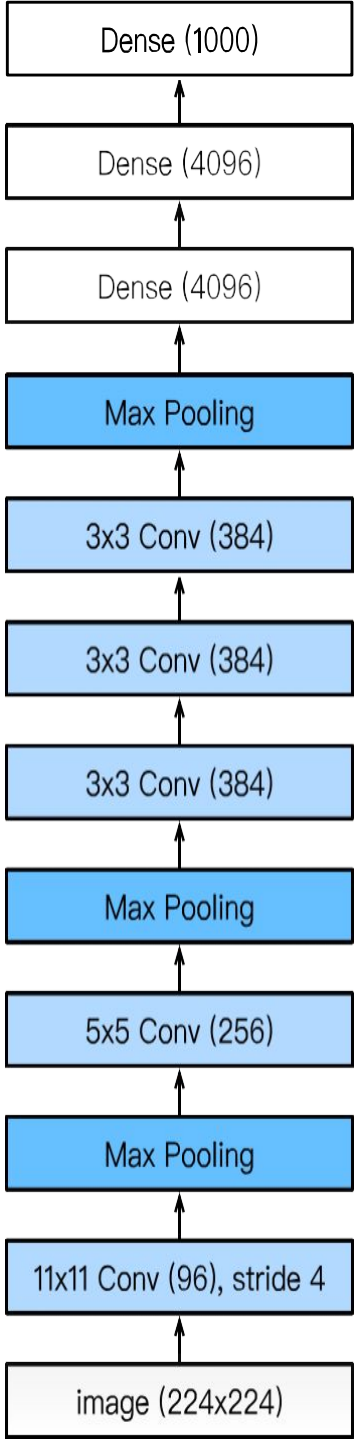
More Tricks

- 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



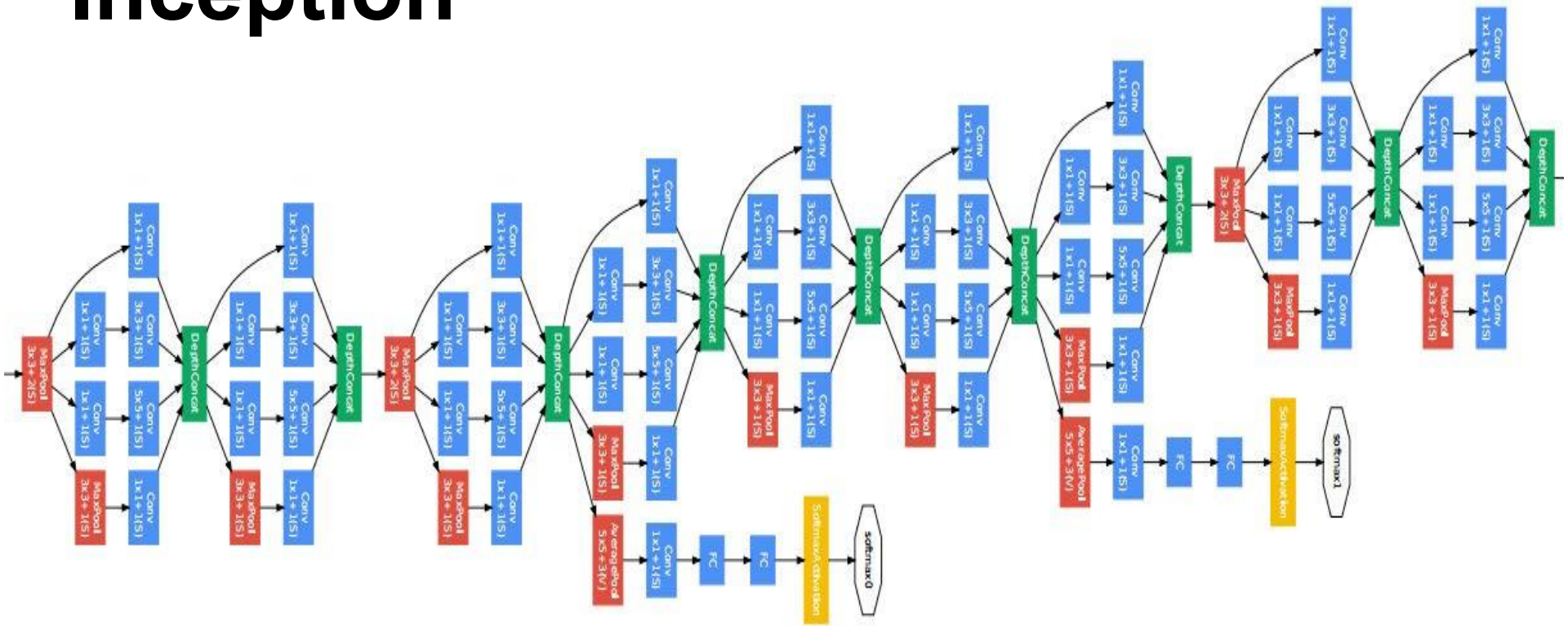
Complexity

	#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

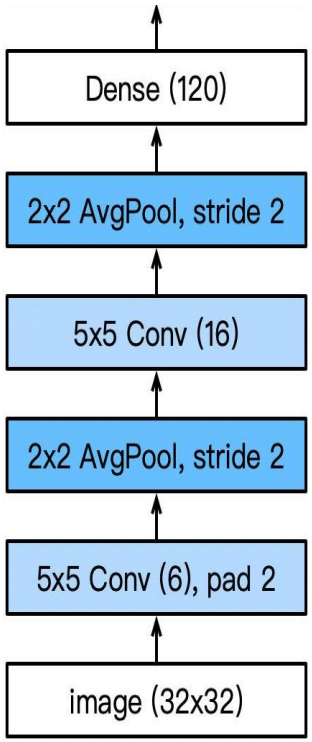


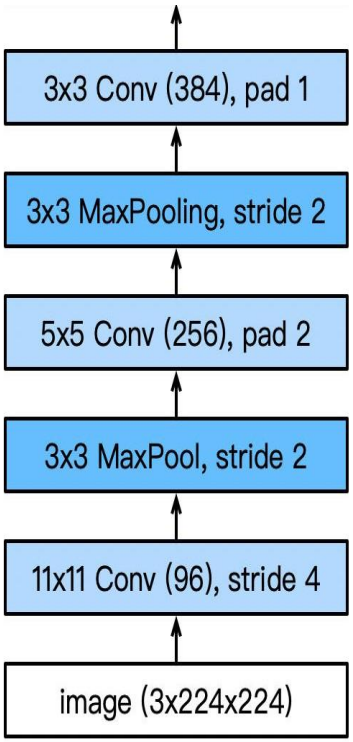
AlexNet Notebook

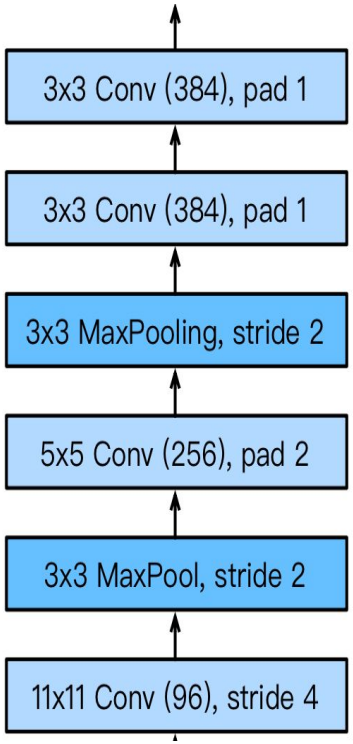
Inception

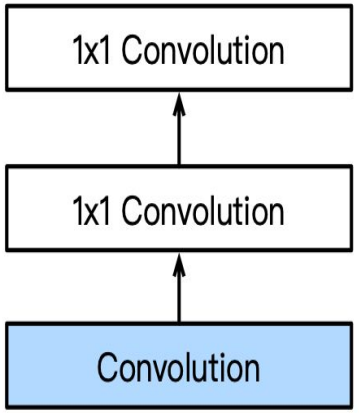

MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
DepthConcat
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
FC
SoftmaxActivation
softmax0
softmax1

Picking the best convolution …





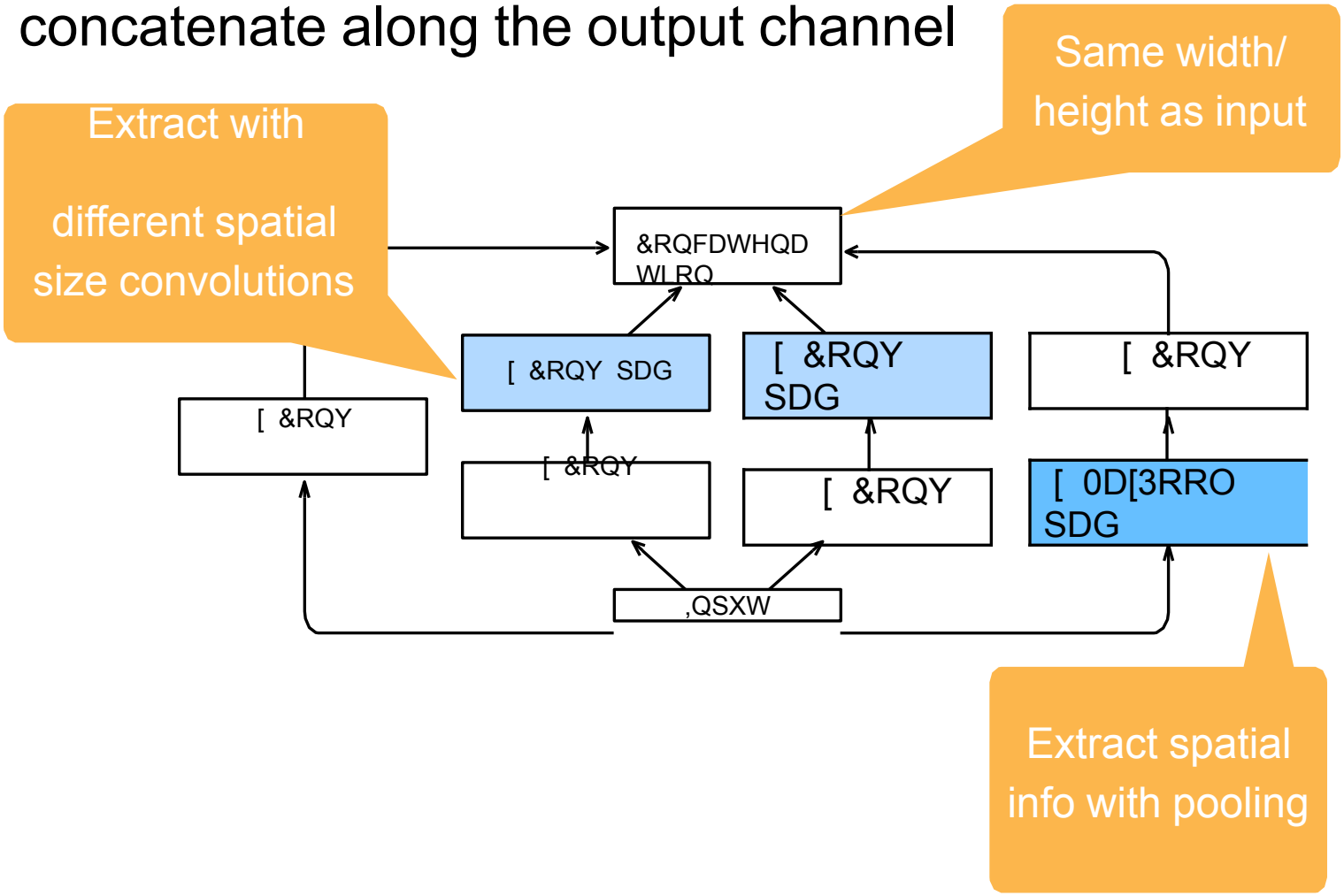




Why choose? Just pick them all.

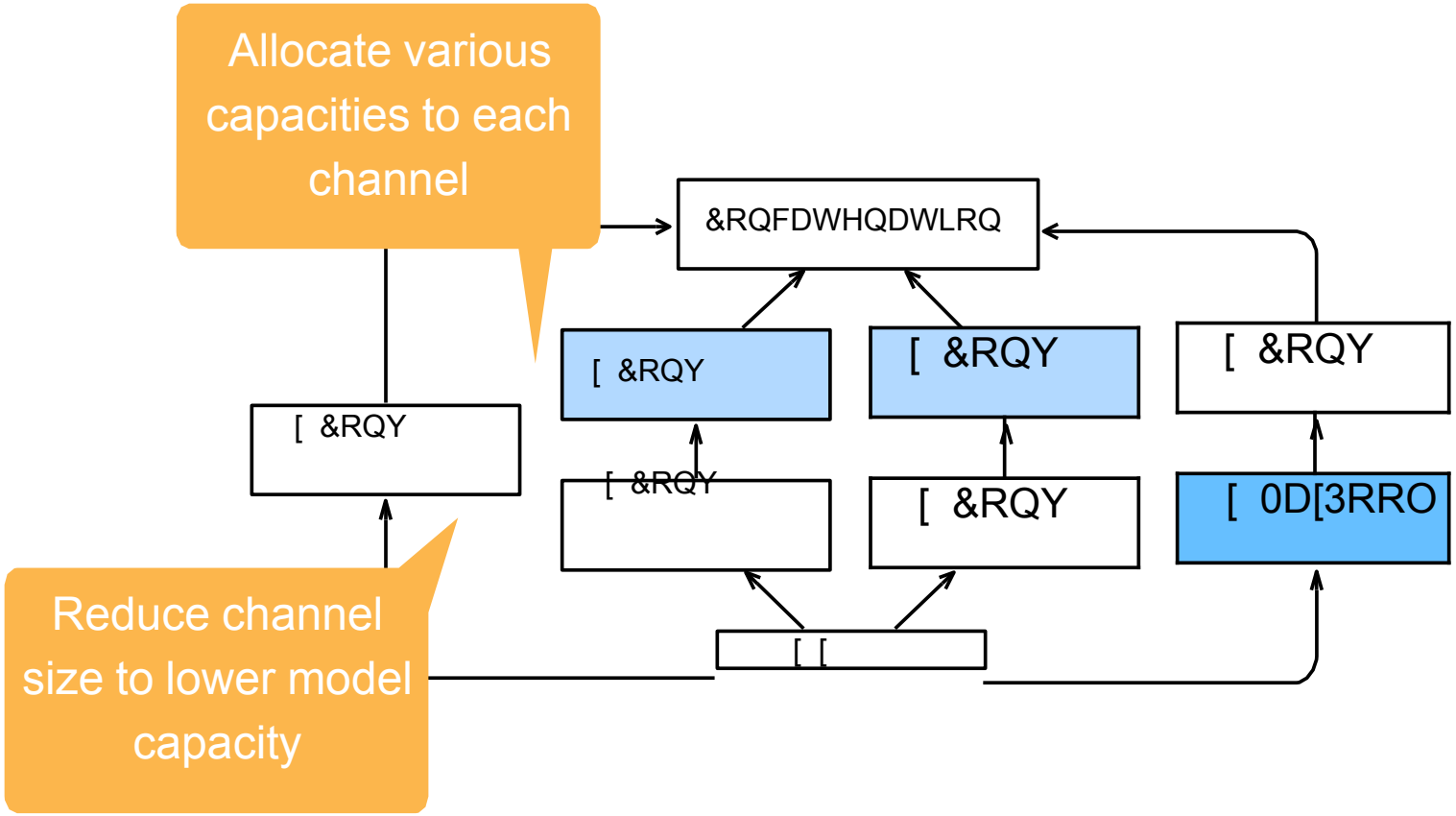
Inception Blocks

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



Inception Blocks

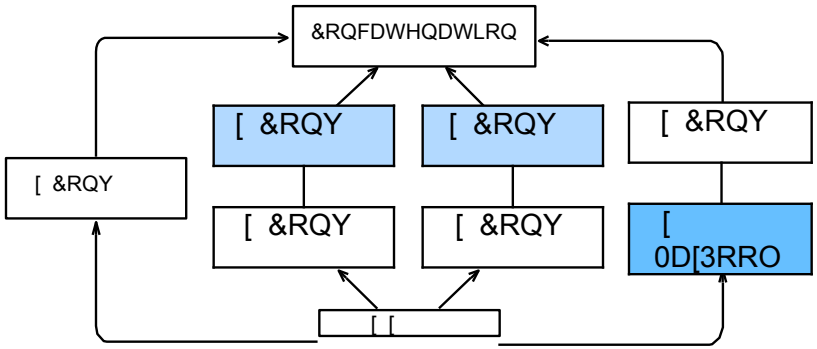
The first inception block with channel sizes specified



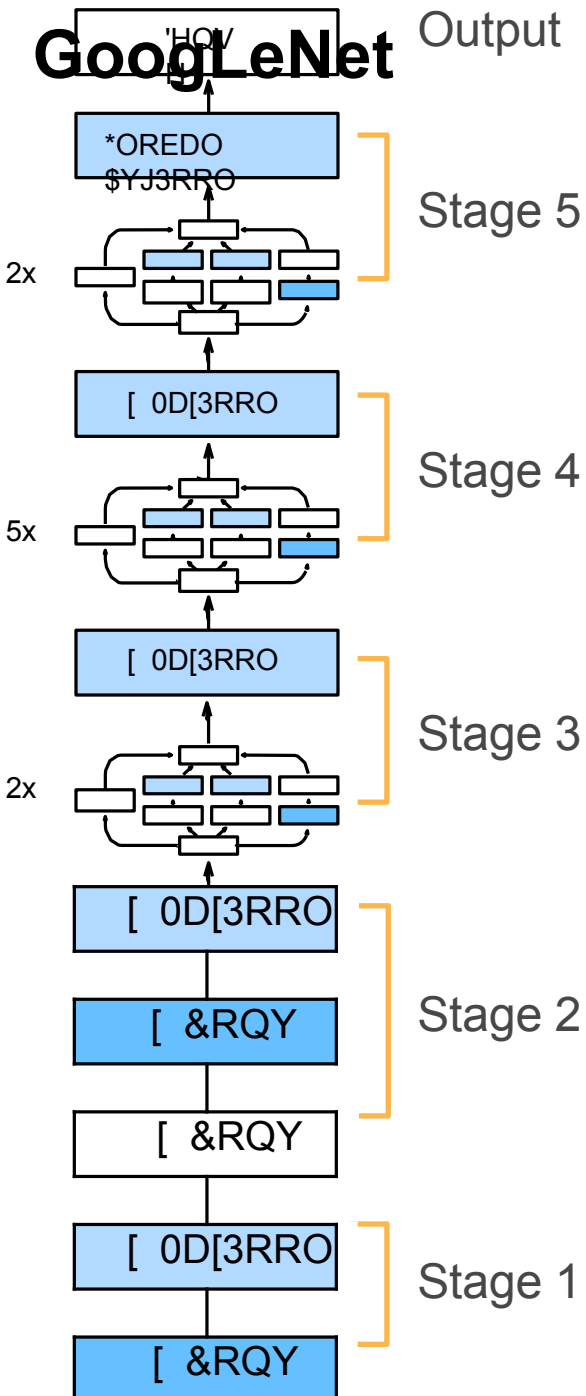
Inception Blocks

- 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

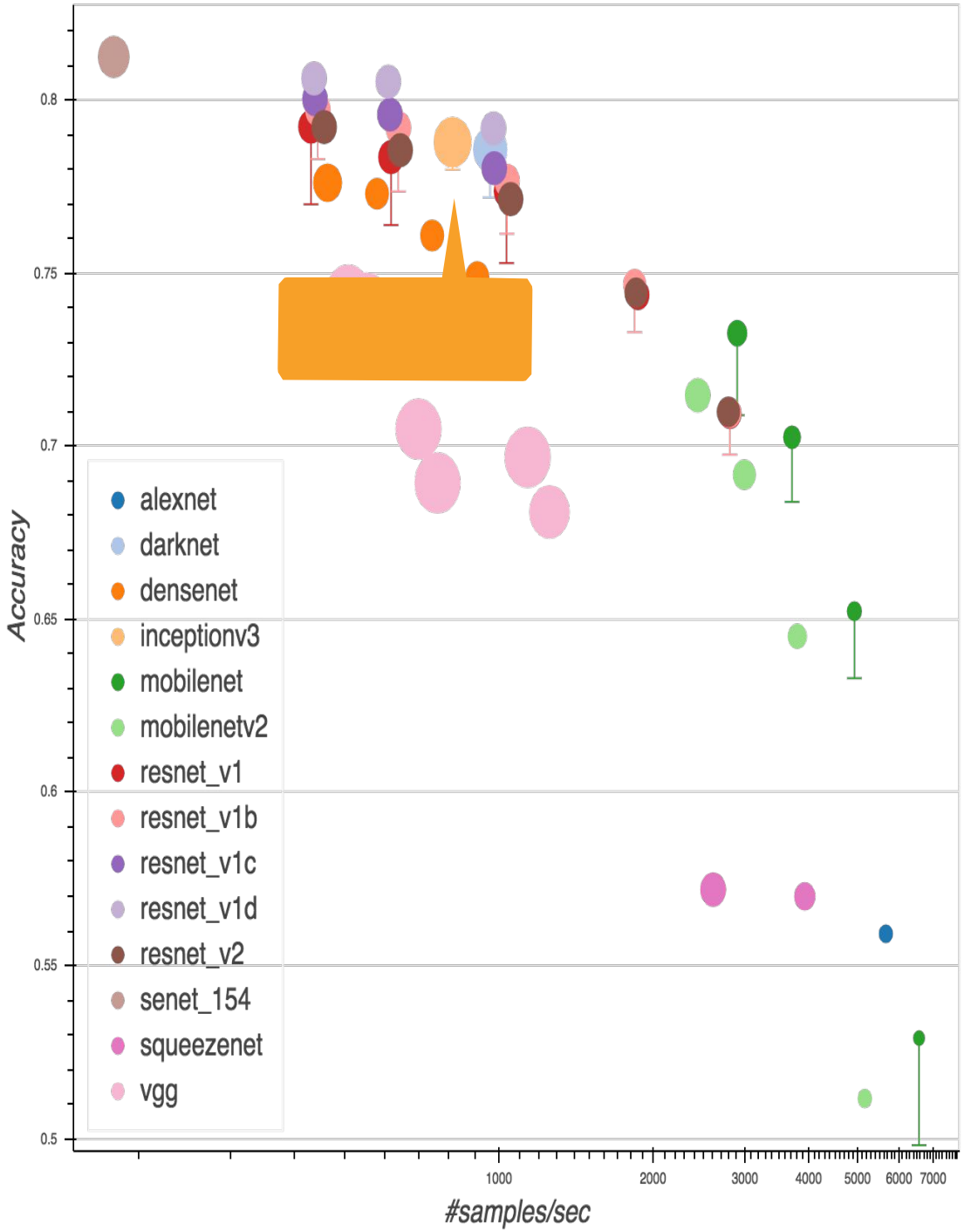


- 5 stages with 9 inceptions blocks



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

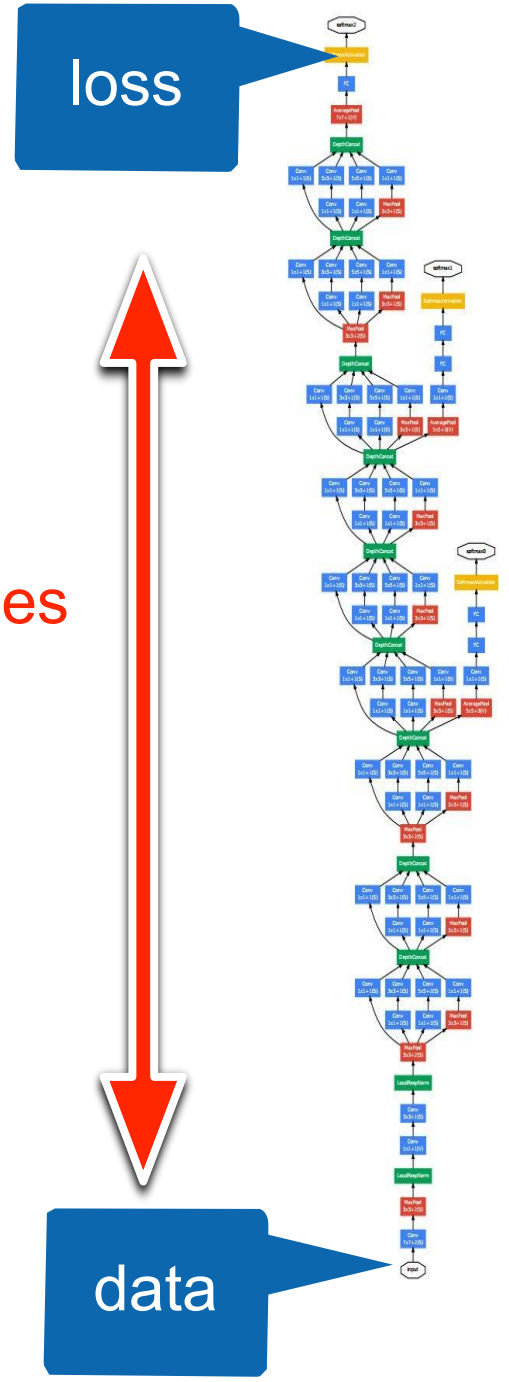
https://gluon-cv.mxnet.io/model_zoo/classification.html


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?



Batch Normalization

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

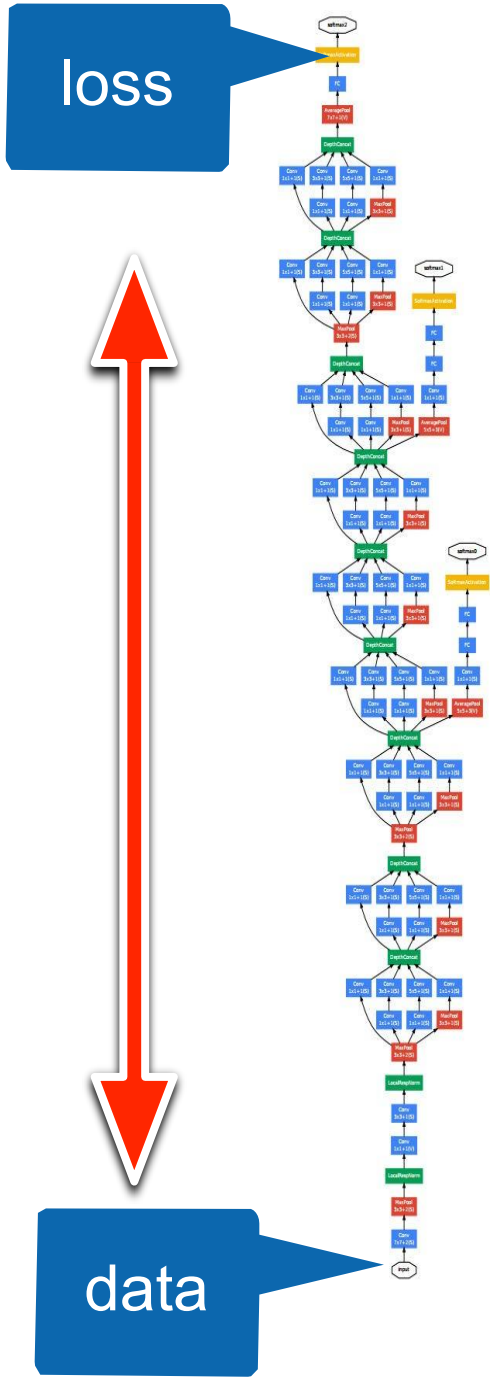
$$\mu_B = \frac{1}{|B|}\sum_{i \in B} x_i \quad \text{and} \quad \sigma_B^2 = \frac{1}{|B|}\sum_{i \in B}(x_i - \mu_B)^2 + \epsilon$$

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

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

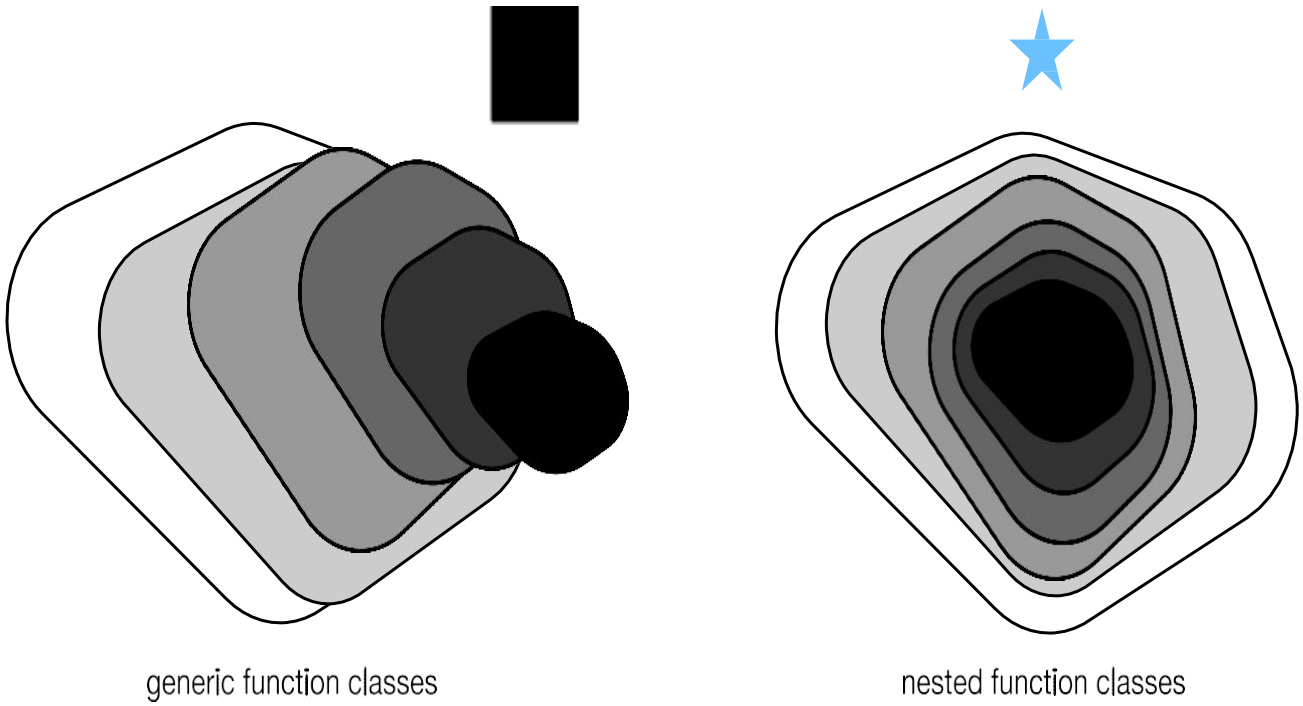
ERmanp diroicmal

ERmanp diroicmal

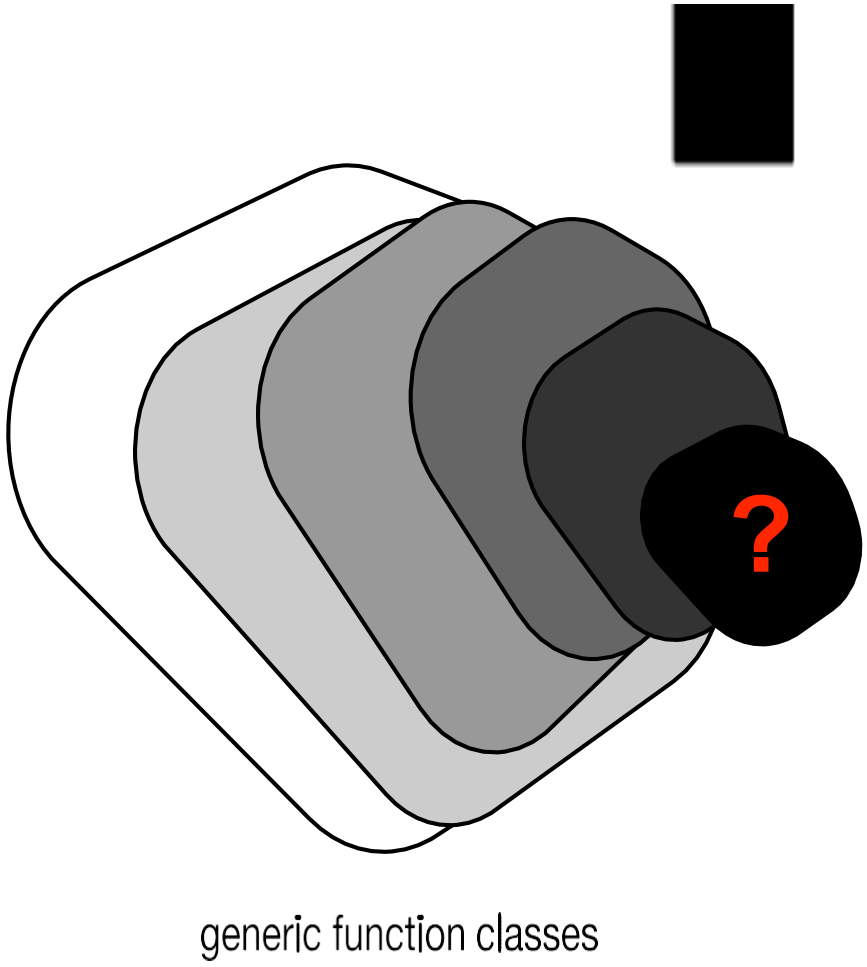
  - Random shift per minibatch
  - Random scale per minibatch

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

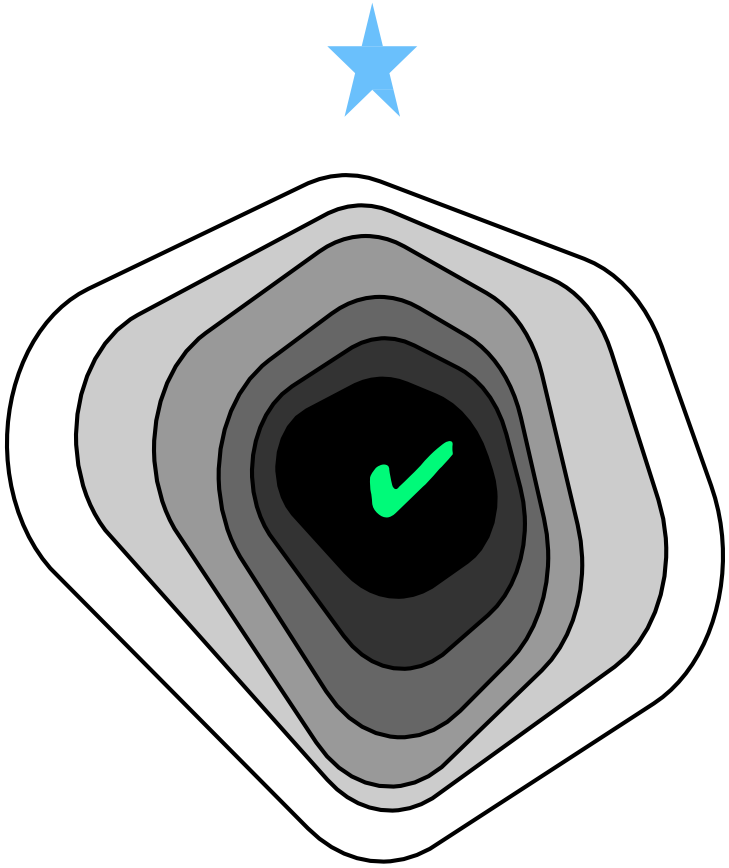
Residual Networks



Does adding layers improve accuracy?



generic function classes

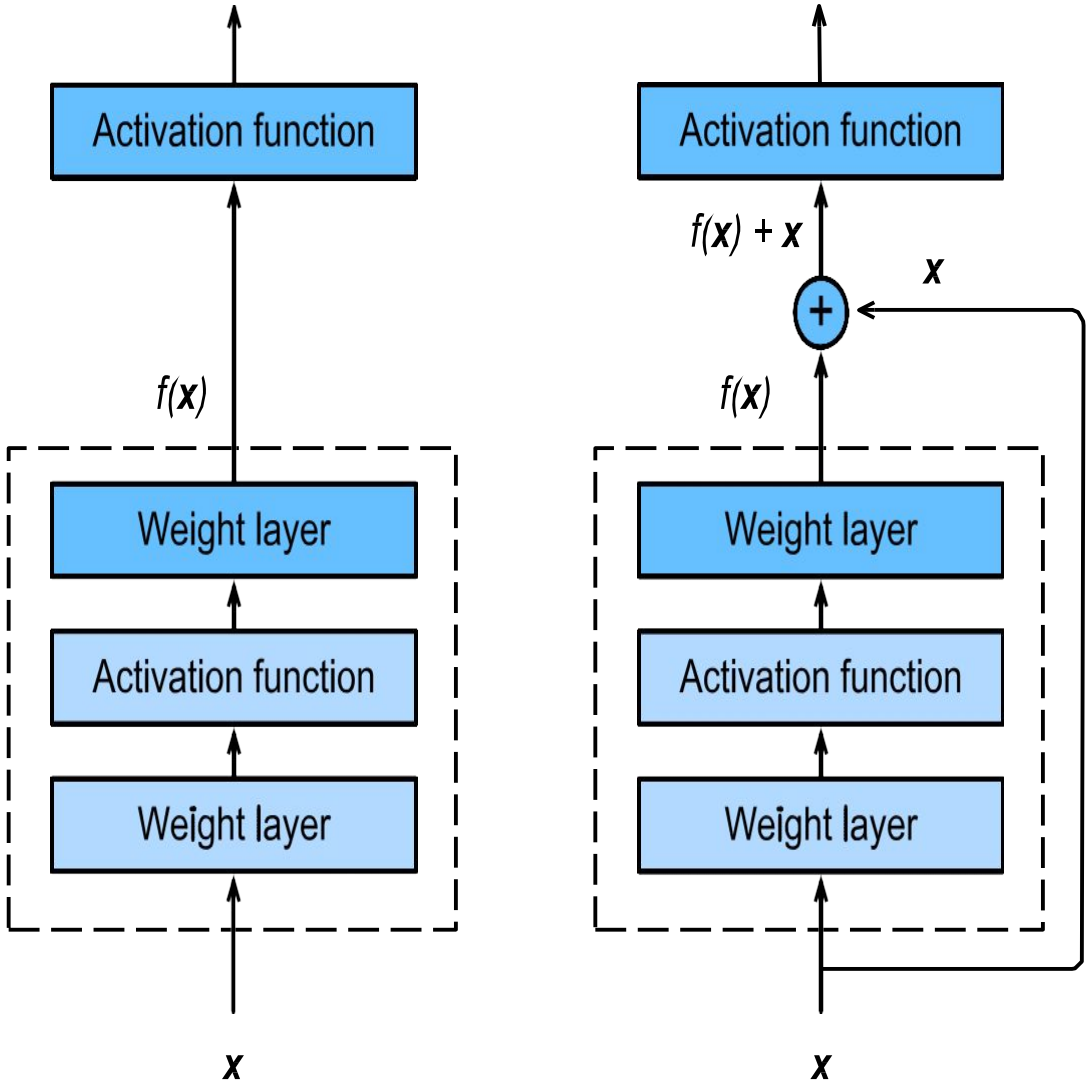


nested function classes

Residual Networks

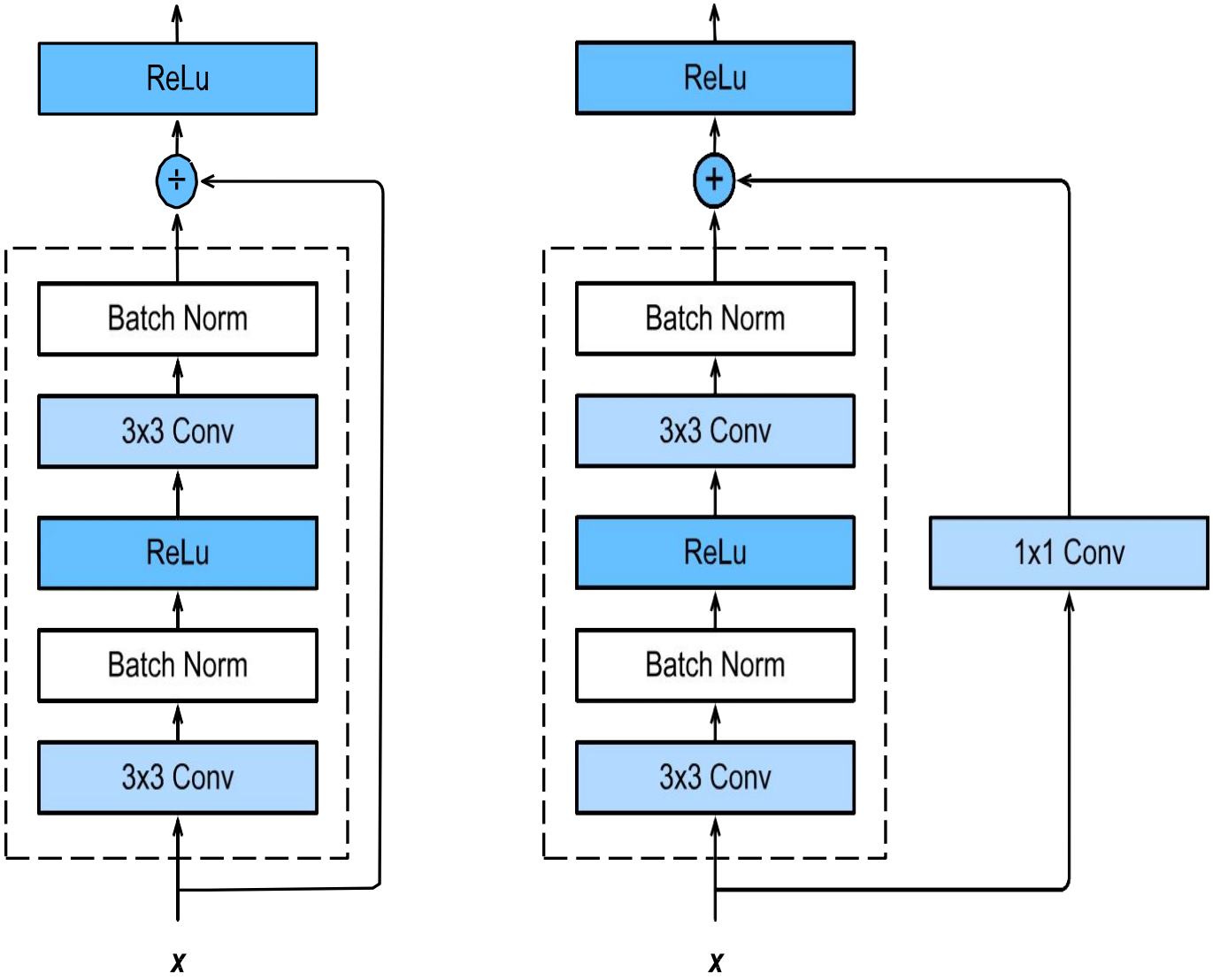
- Adding a layer **changes** function class
- We want to **add to** the function class
- 'Taylor expansion' style parametrization

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

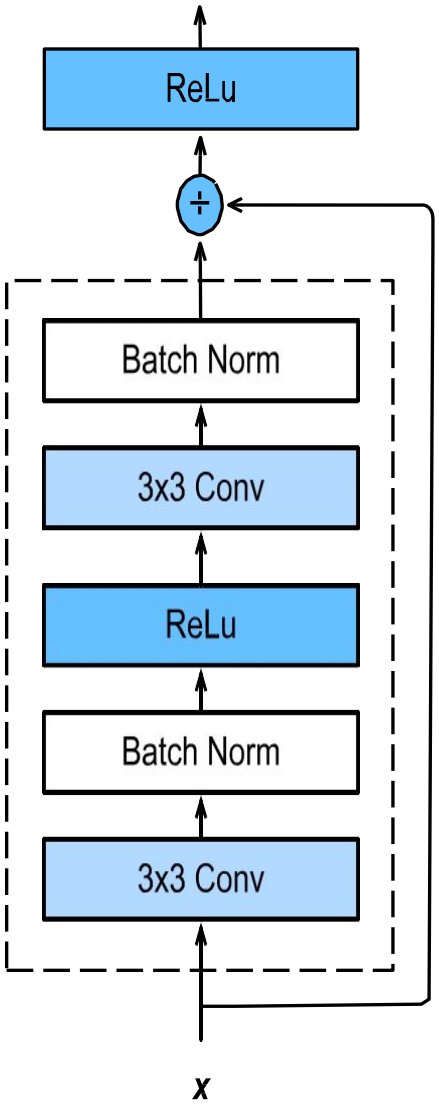


He et al., 2015

ResNet Block in detail

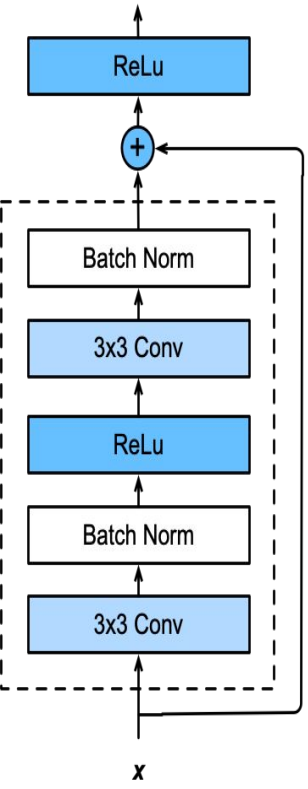


In code

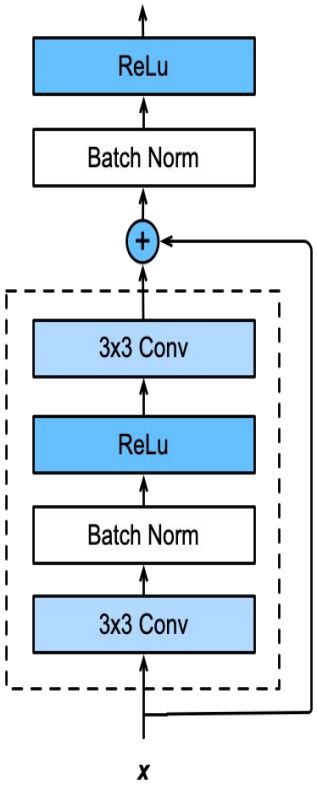



```
def forward(self, X):
Y = npx.relu(self.bn1(self.conv1(X))) Y =
self.bn2(self.conv2(Y))
if self.conv3:
X = self.conv3(X) return npx.relu(Y + X)
```

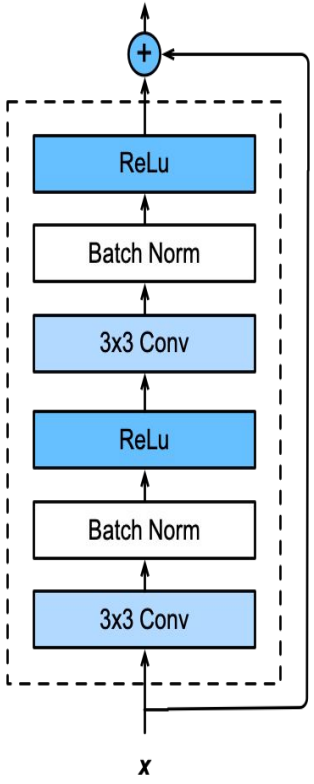
The many flavors of ResNet blocks



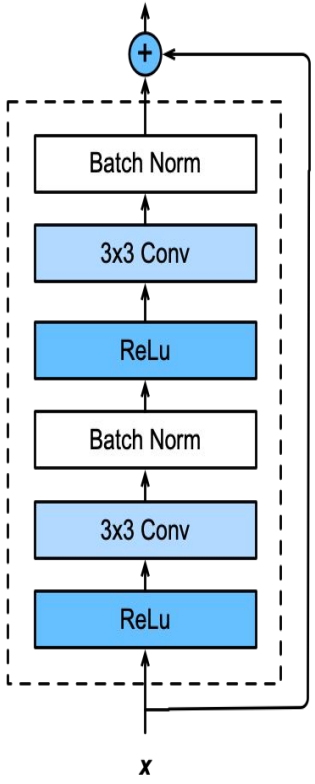
original



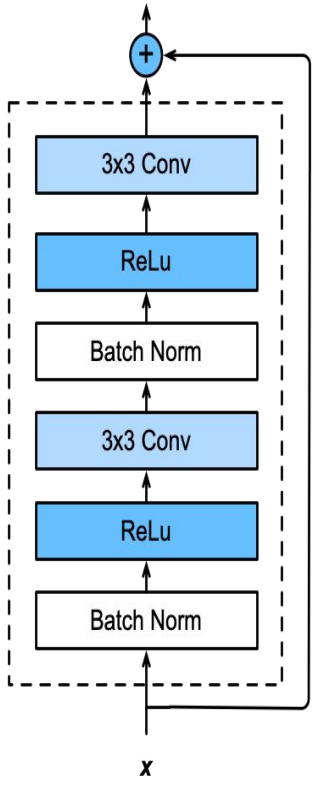
Batch Norm after addition



ReLu last



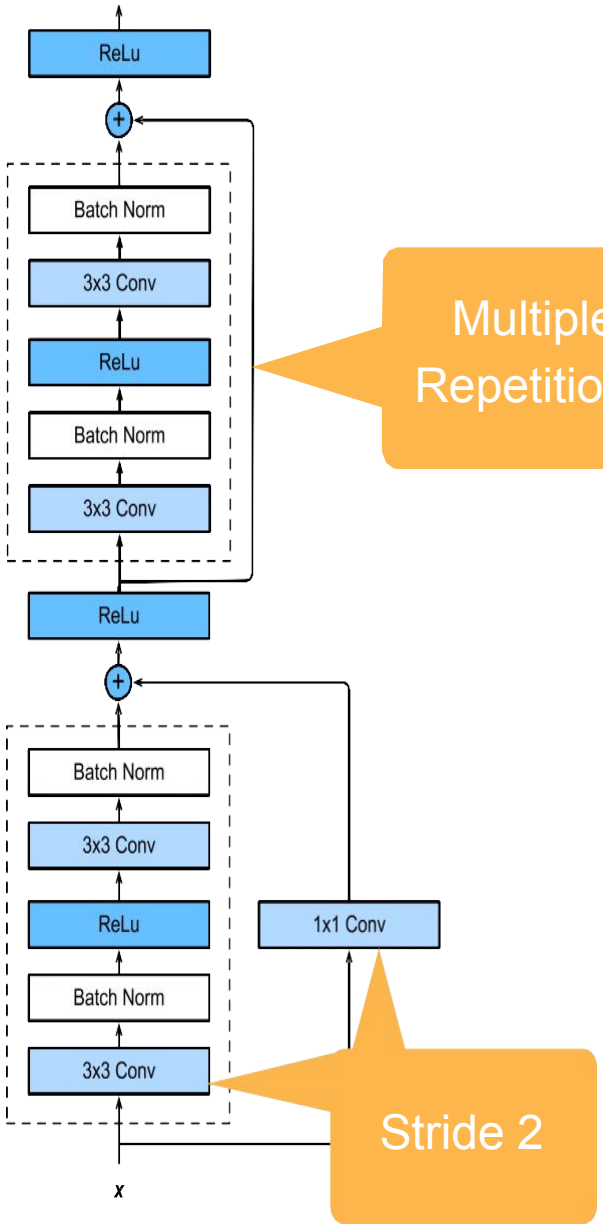
Batch Norm last



Pre-activation

Try every permutation

ResNet Module



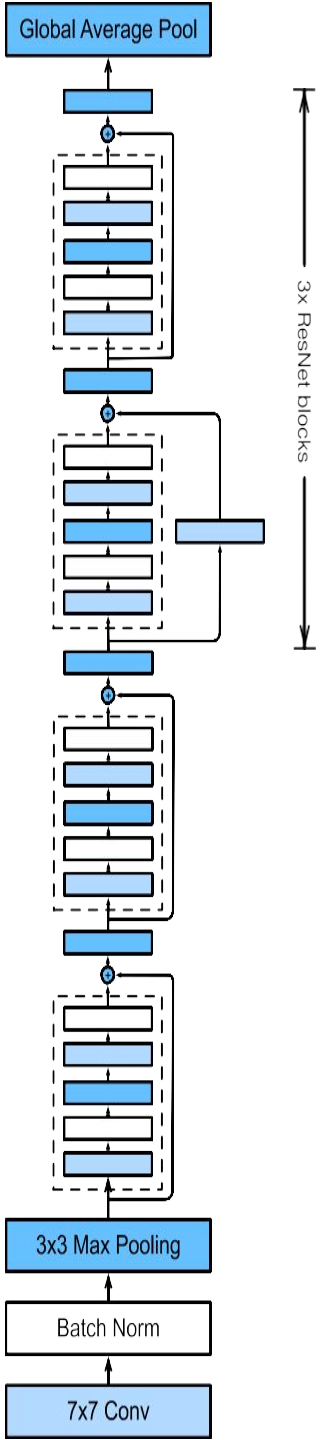
- Downsample per module (stride=2)
- Enforce some nontrivial nonlinearity per module (via 1x1 convolution)
- Stack up in blocks

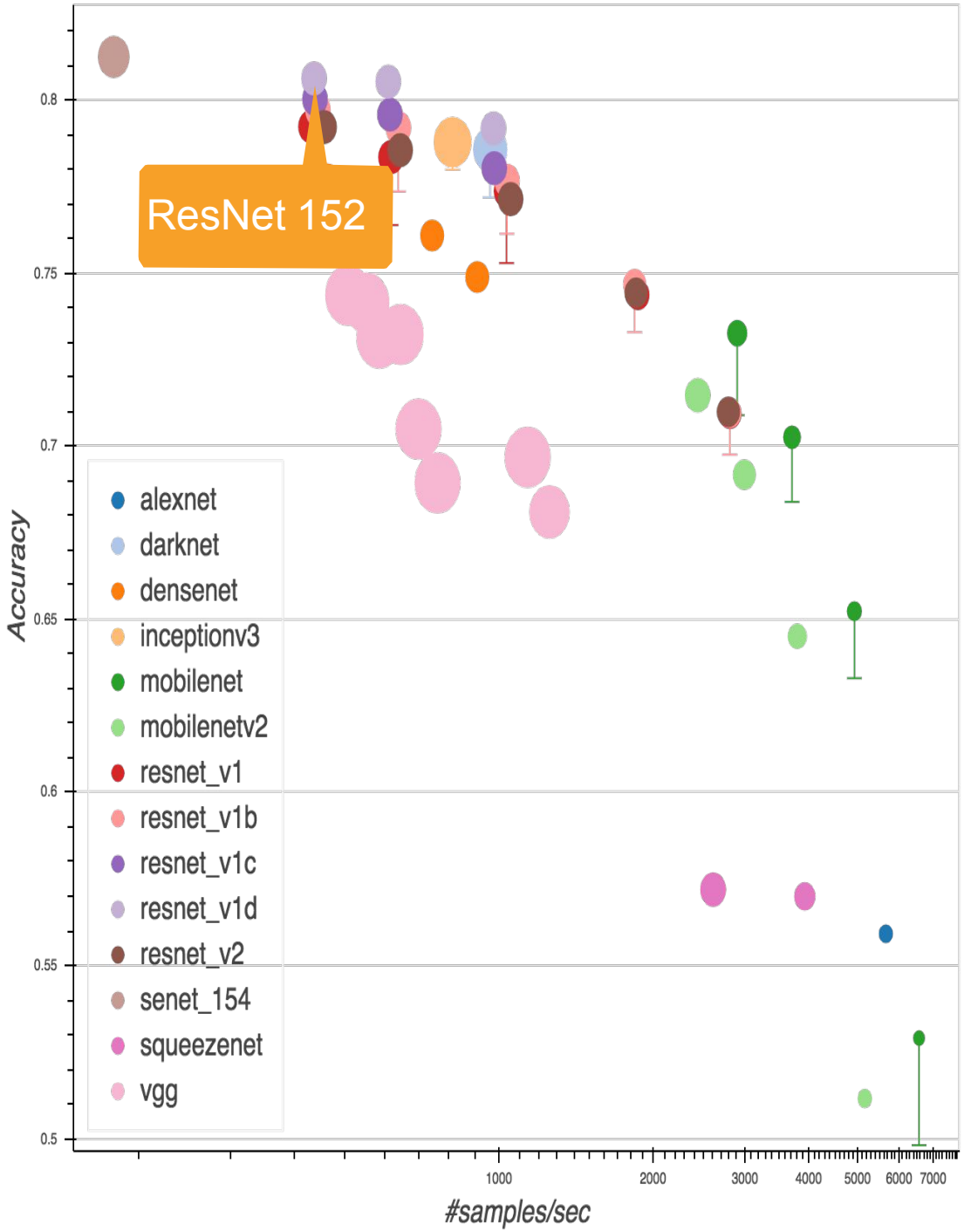
```
blk = nn.Sequential()
for i in range(num_residuals):
if i == 0 and not first_block:
        blk.add(Residual(num_channels,
            use_1x1conv=True, strides=2))
    else:
        blk.add(Residual(num_channels))
```

Putting it all together

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

… train it at scale …





GluonCV Model Zoo

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

Jupyter Notebook

More Ideas

DenseNet (Huang et al., 2016)

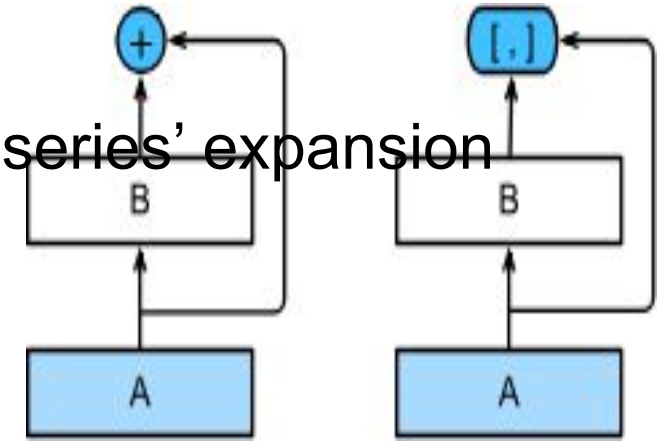
- ResNet combines x and f(x)
- DenseNet uses higher order 'Taylor series' expansion

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

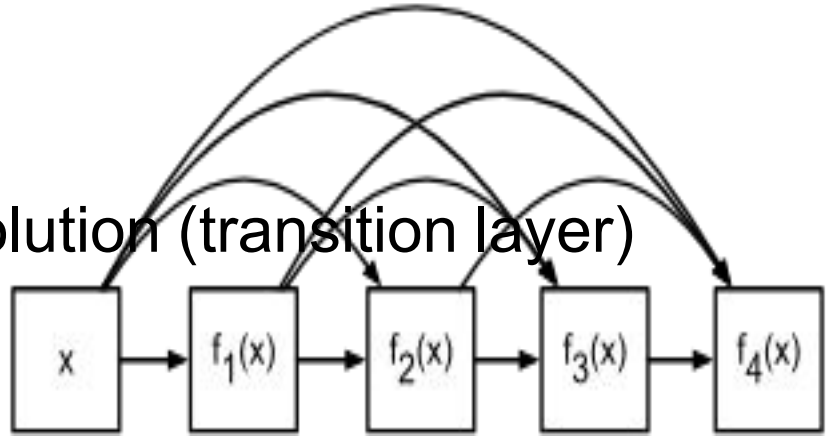
$$x_1 = x$$

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

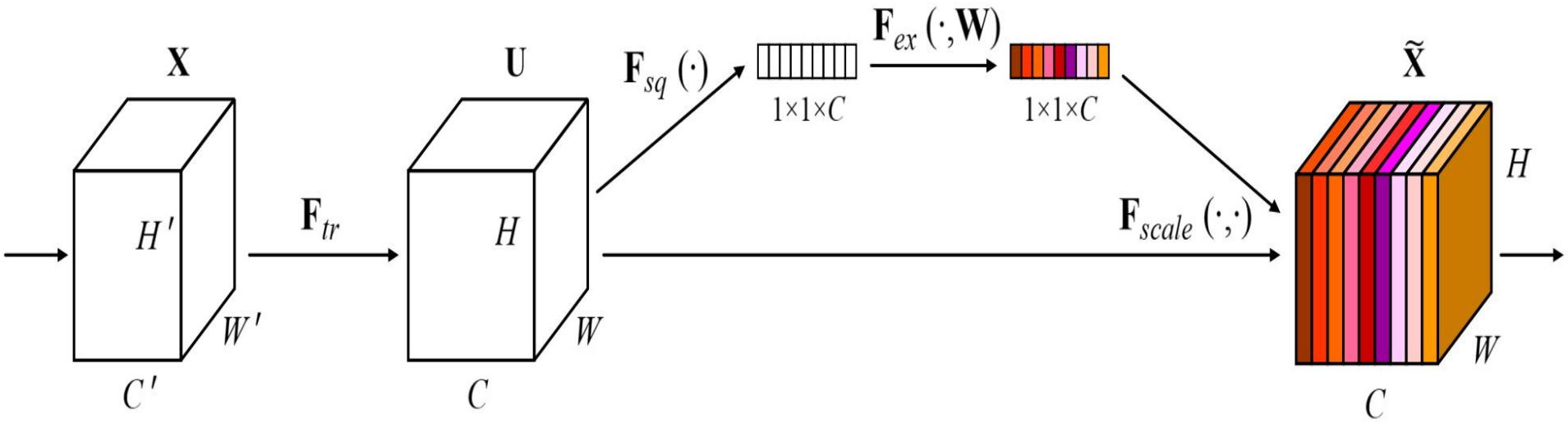
$$x_2 = [x, f_1(x), f_2([x, f_1(x)])]$$



- Occasionally need to reduce resolution (transition layer)



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



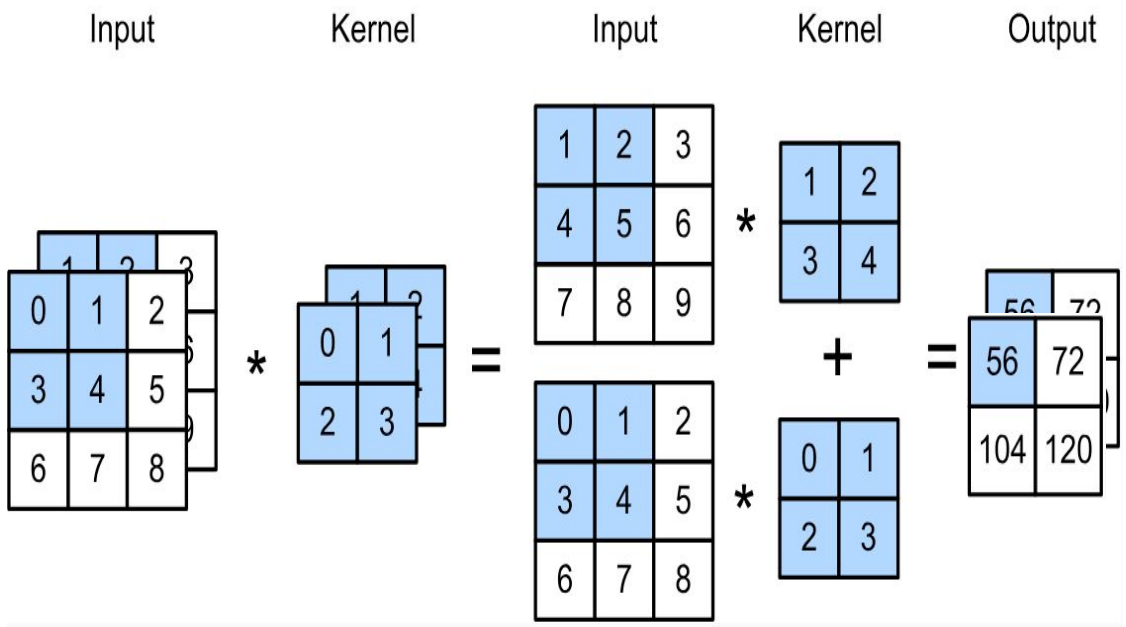
- Learn global weighting function per channel
- Allows for fast information transfer between pixels in different locations of the image

Separable Convolutions - all channels separate

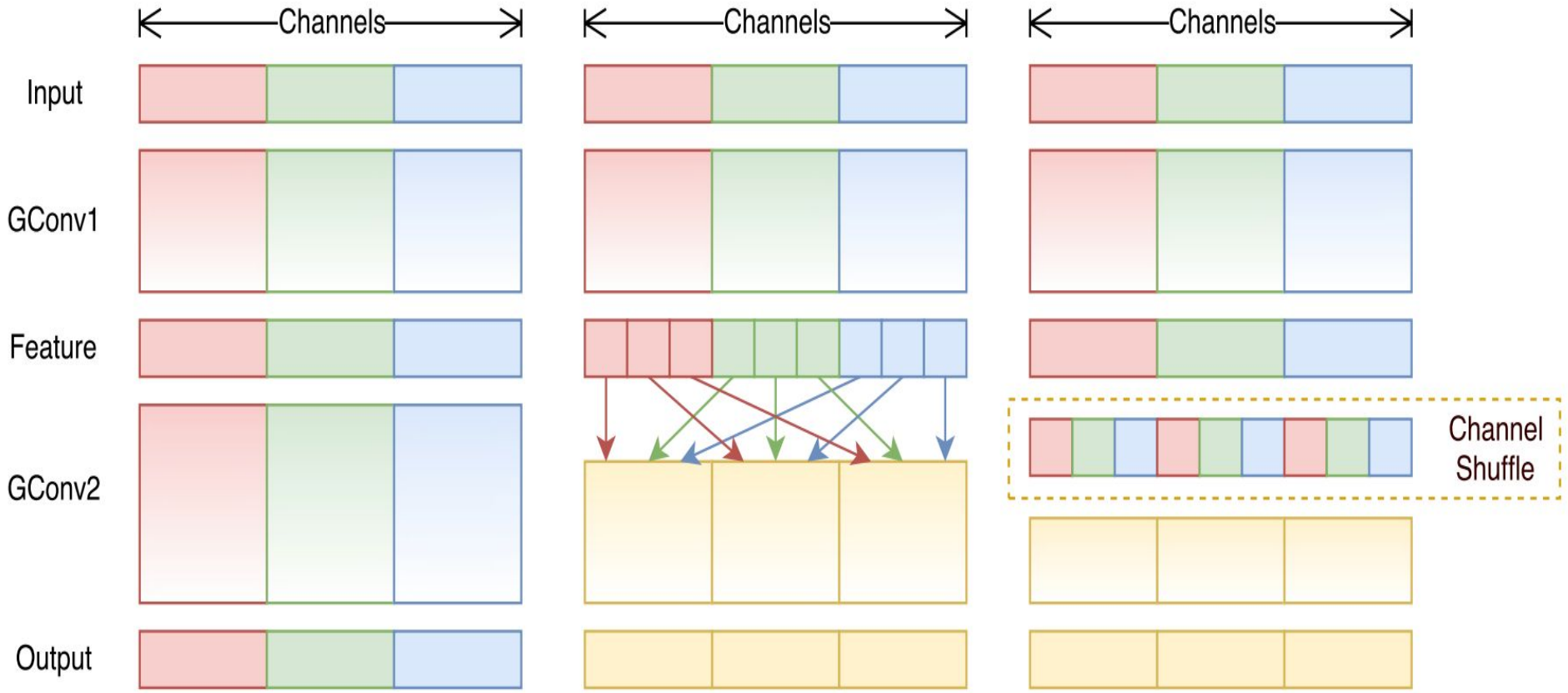
- **Parameters** $k_h \cdot k_w \cdot c_i \cdot c_o$
- **Computation** $m_h \cdot m_w \cdot k_h \cdot k_w \cdot c_i \cdot c_o$
- **Break up channels** to the extreme

  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)