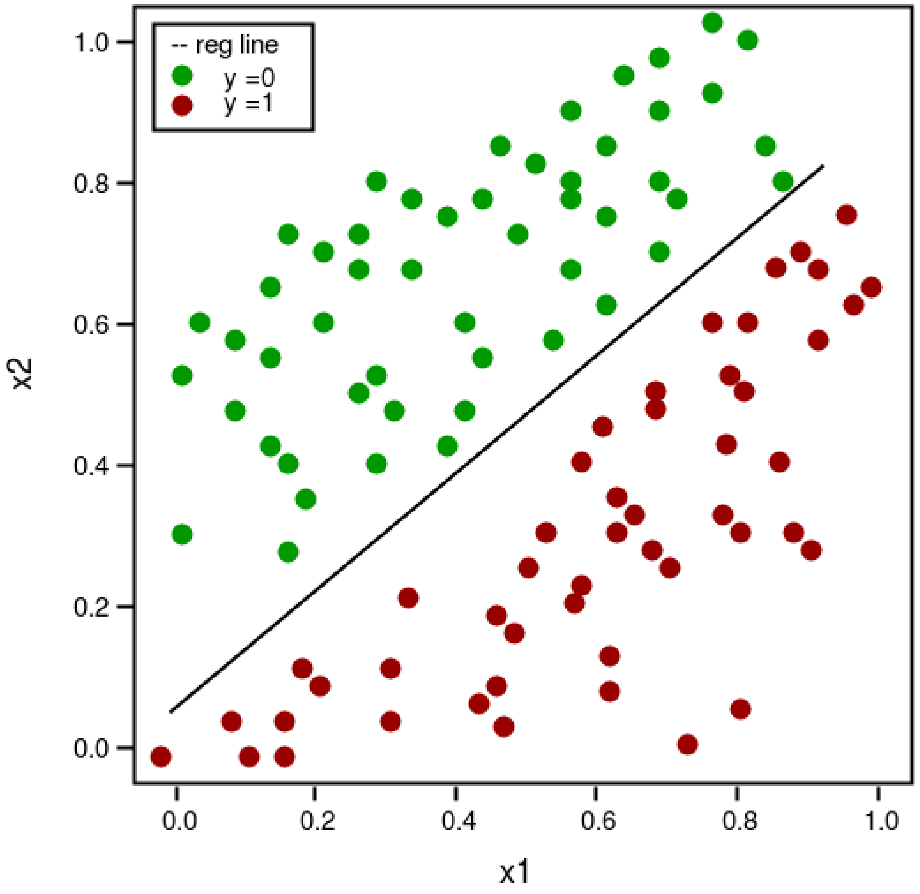
LECTURE 3: CONVOLUTIONAL NEURAL NETWORKS

University of Washington, Seattle

Fall 2025

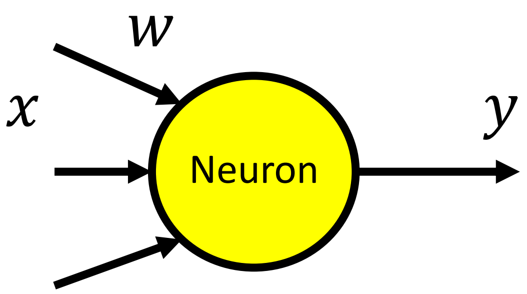
Previously in EE 596…

$$y = \sigma(\vec{w}^T \vec{x} + b)$$



Previously in EE 596...

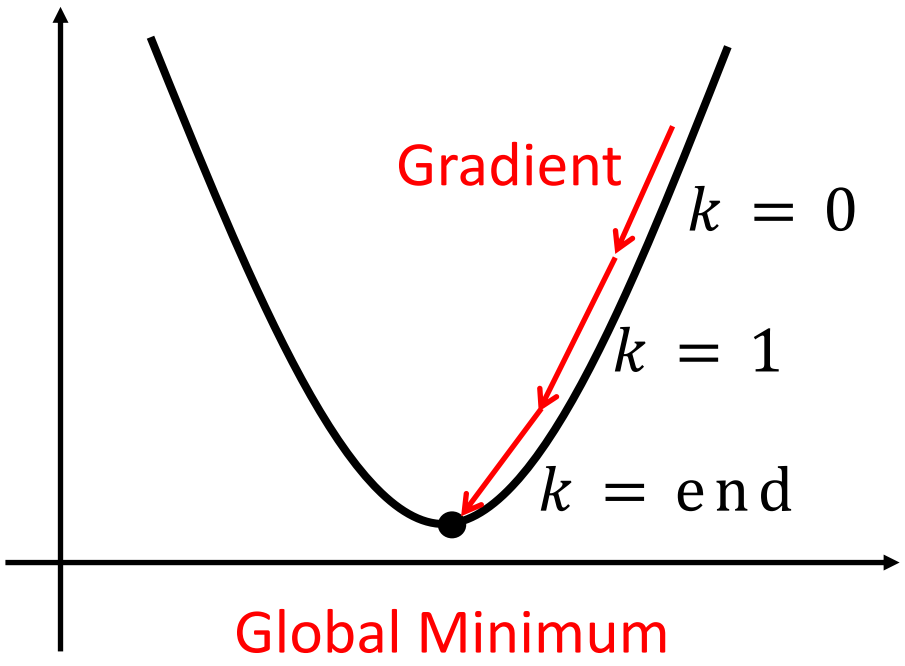
$$y = \sigma(\vec{w}^T \vec{x} + b)$$



$$J = L((\vec{w}, b), y)$$

$$\vec{w}_{k+1} = \vec{w}_k - \alpha \nabla_{\vec{w}} J(\vec{w}_k; b)$$

$$b_{k+1} = b_k - \alpha \frac{\partial}{\partial b} J(\vec{w}; b_k)$$



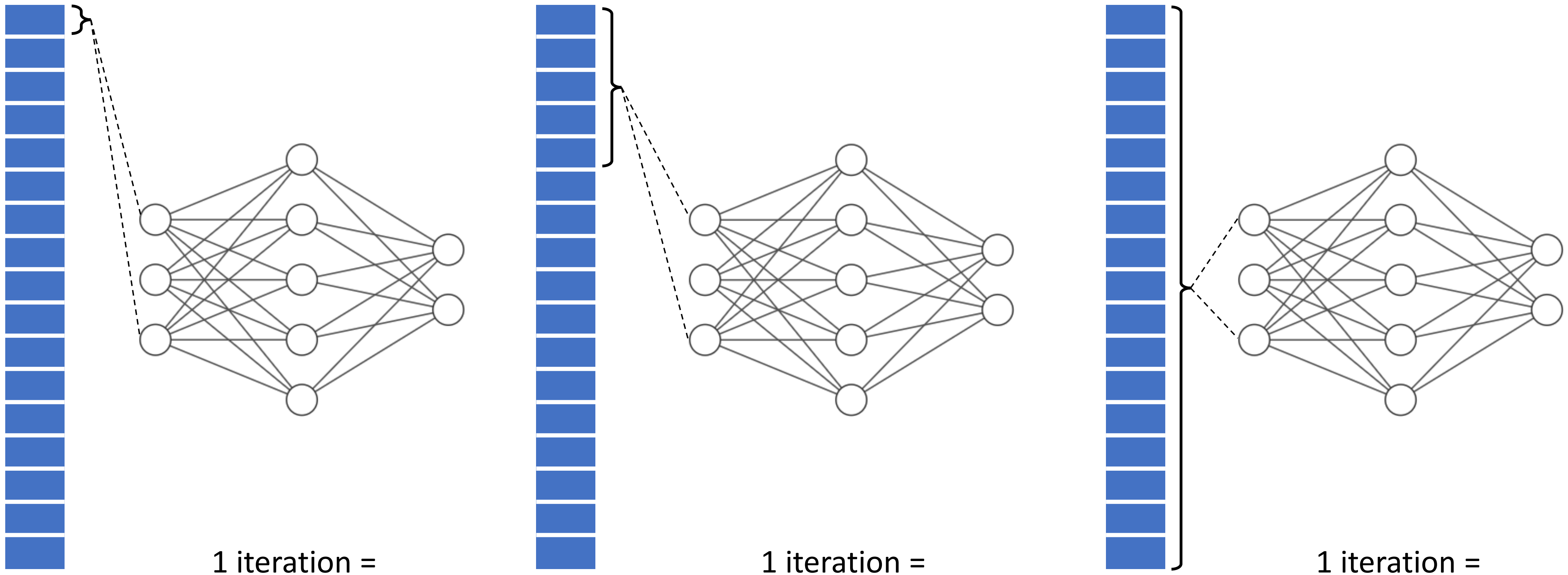
Previously in EE 596…

Training Dataset

SGD

Mini-batch

Batch GD

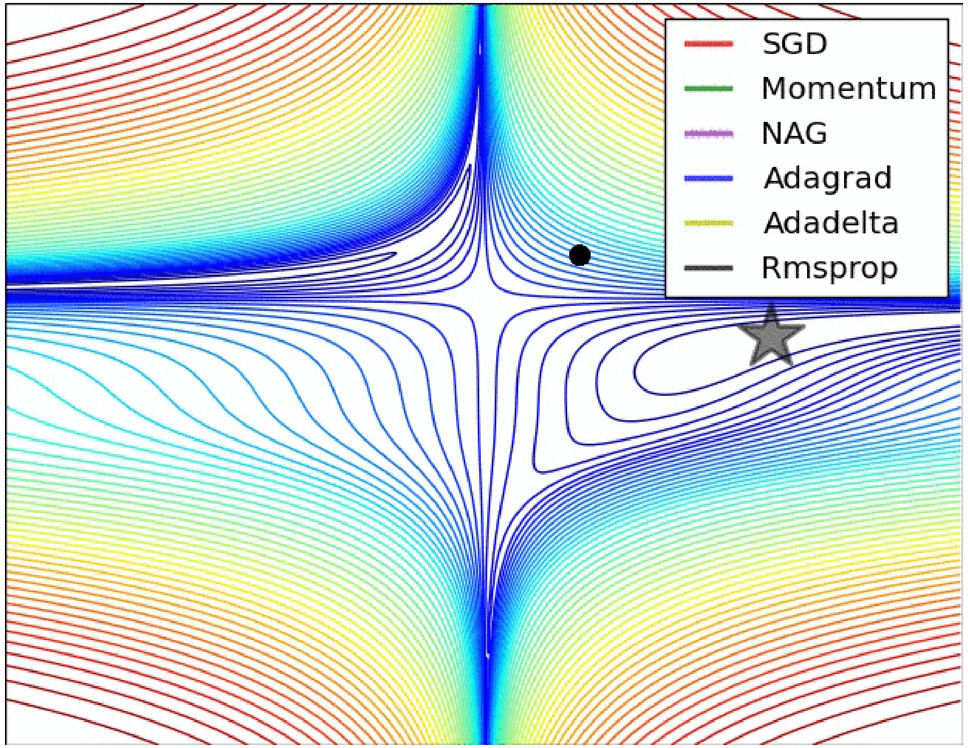


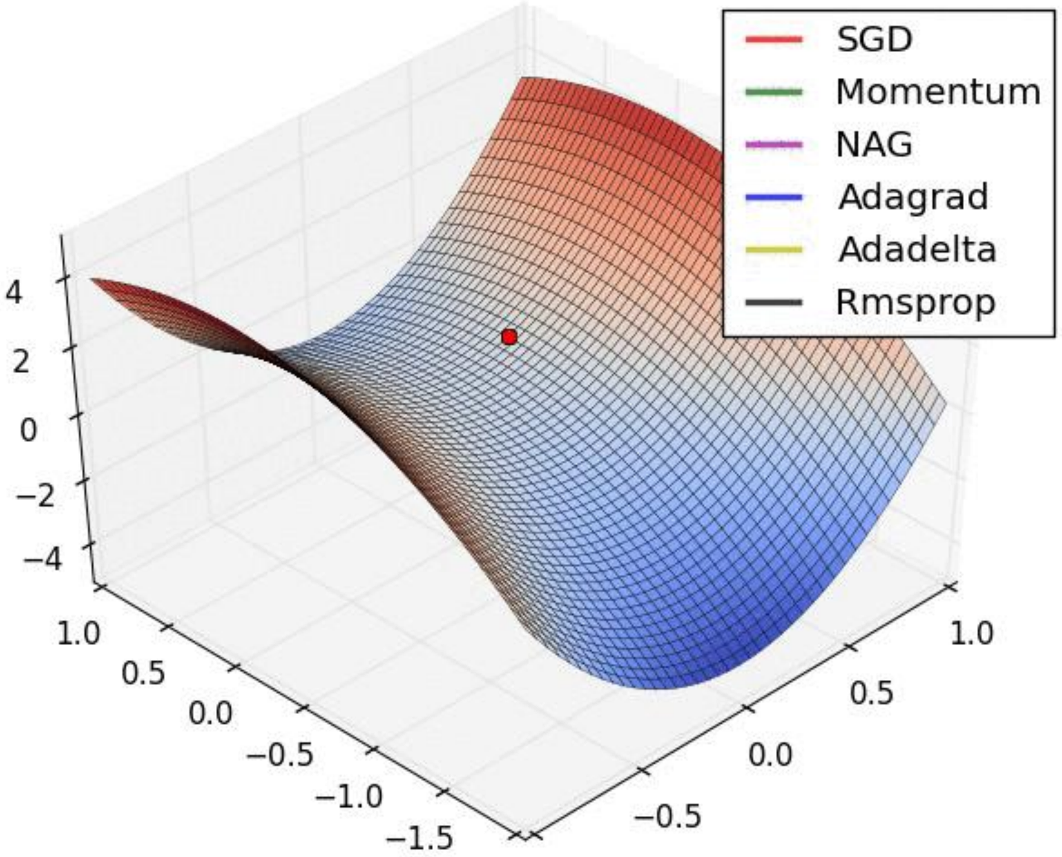
1 iteration =
Fwd/bwd pass 1 training sample

1 iteration =
Fwd/bwd pass n-training samples
(n < total # of training samples)

1 iteration =
Fwd/bwd pass full training samples

Previously in EE 596...





Previously in EE 596…

Optimizers

- Vanilla SGD
- Momentum
- AdaGrad
- RMSProp
- Adam

Balance

Optimization Techniques

- Data splitting (Train/Val/Test)
- Regularization
- Data normalization
- Batch-normalization
- Network initialization
- Hyperparameter tunings

OUTLINE

Part 1: Need for CNNs

- Limitation of MLP
- Convolution Layer

Part 2: Convolution Filters

- 2D convolution
- Stride
- Padding
- Volume convolutions

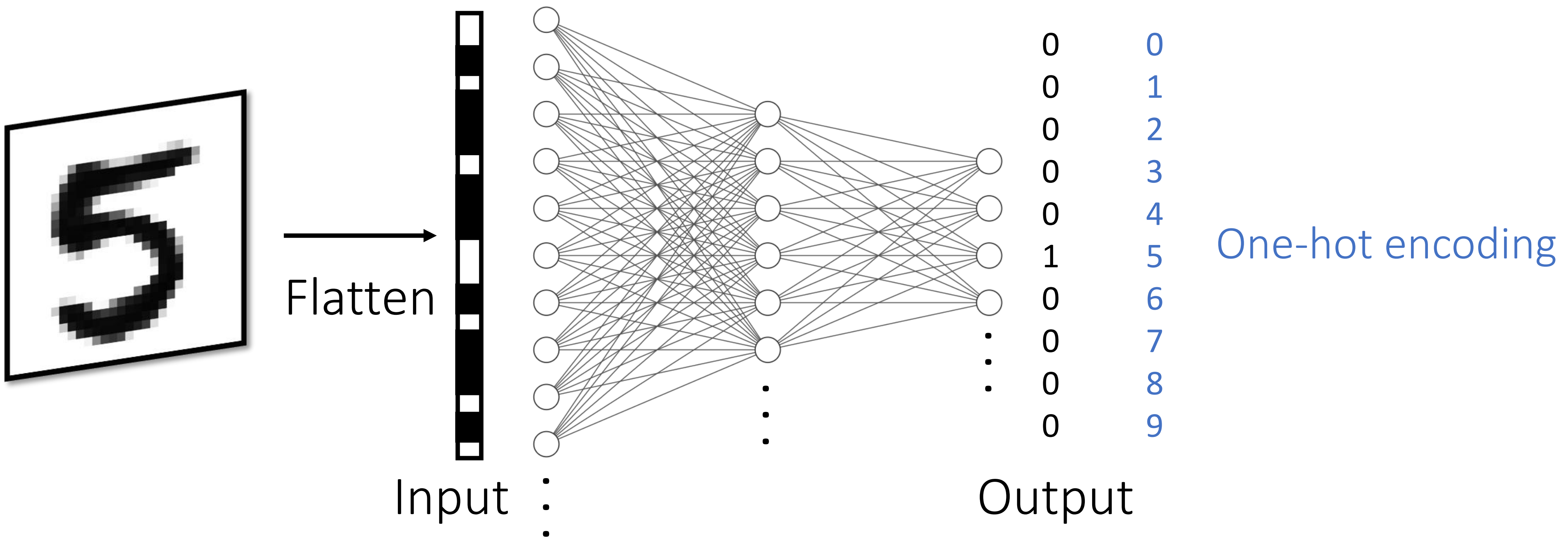
Part 3: Composing Convolutional Neural Networks

- Convolution Layer
- Pooling Layers
- Benefits and challenges of CNNs
- Historical CNN examples

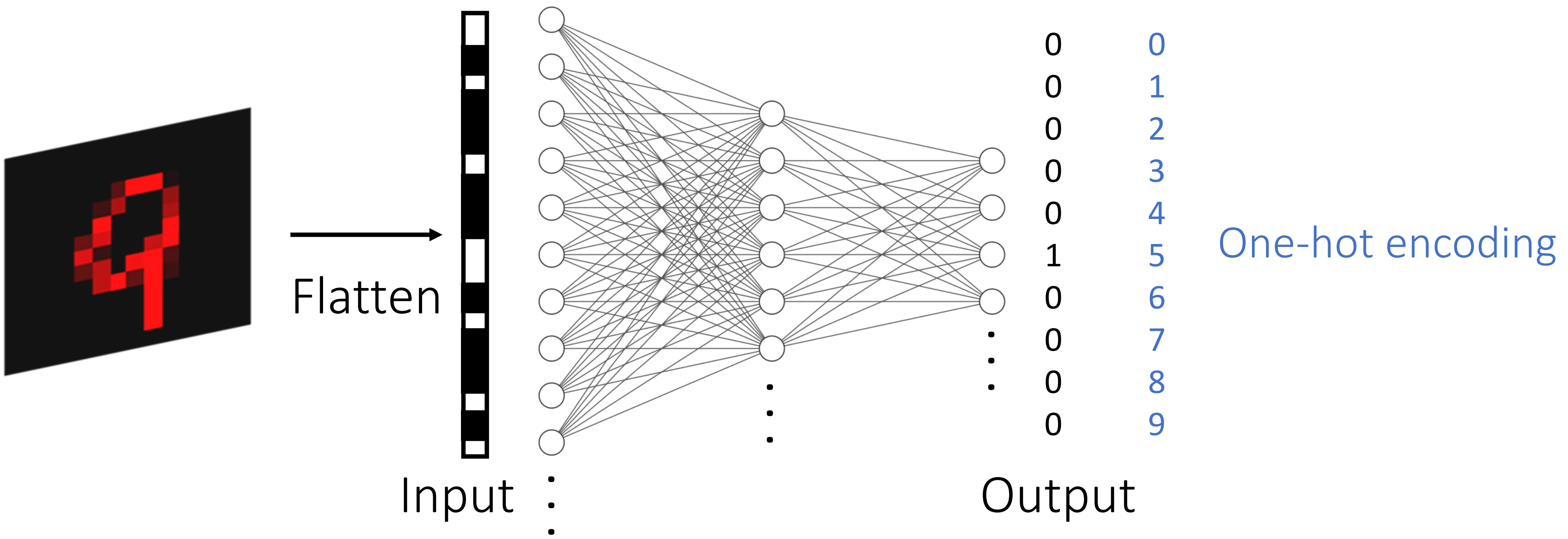
PART 1:

Need for CNNs

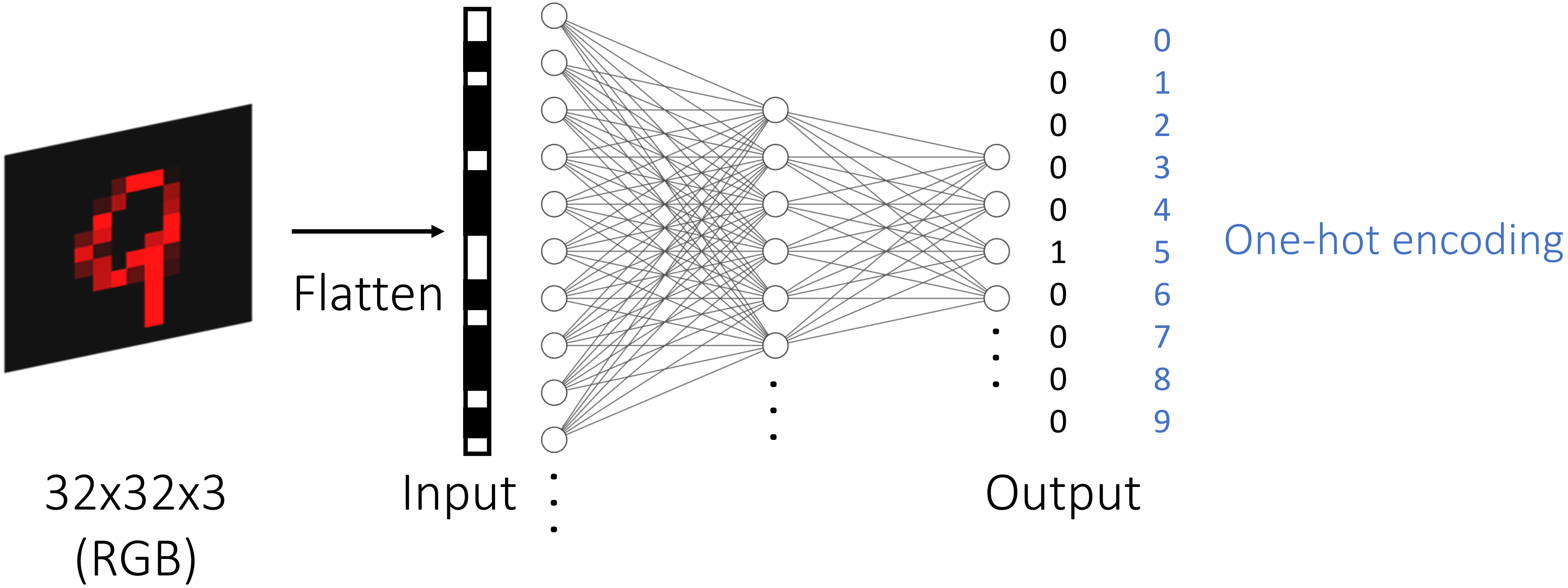
MLP for Image Classification (Lab 2)



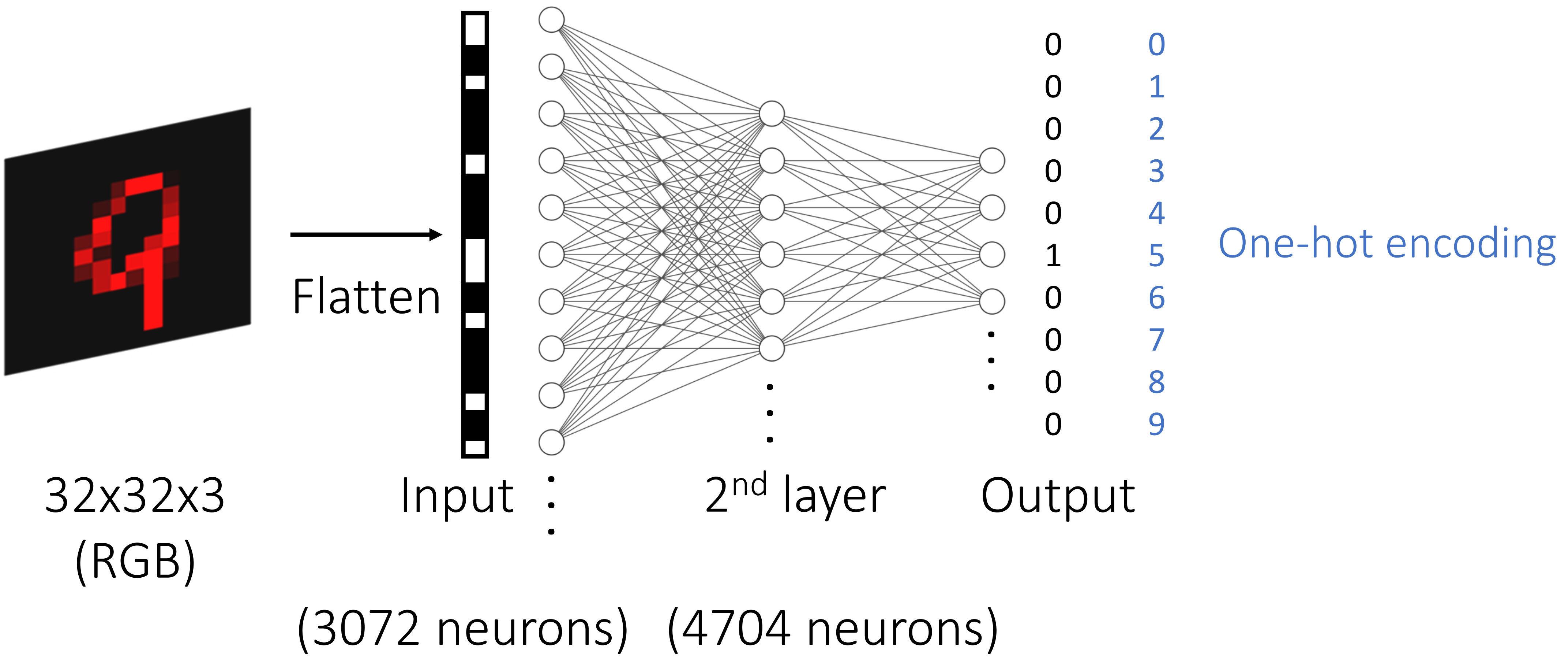
MLP for Image Classification (Lab 2)



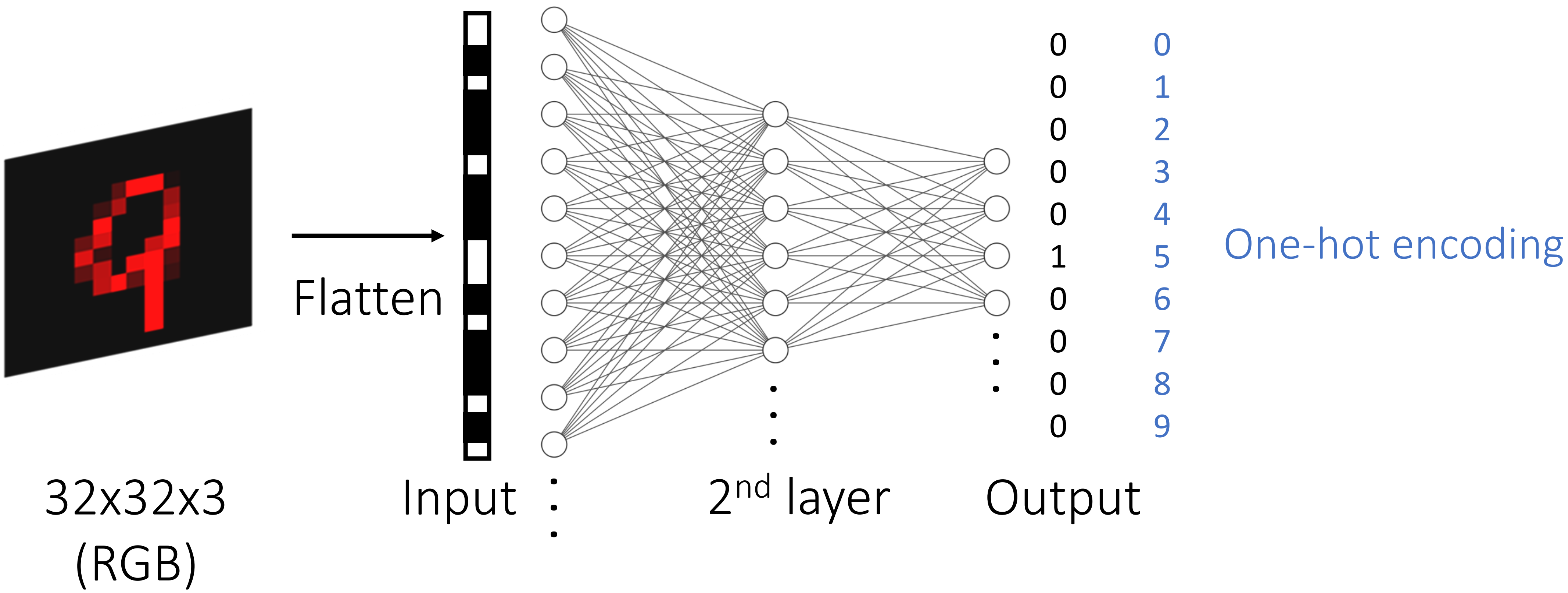
MLP for Image Classification (Lab 2)



MLP for Image Classification (Lab 2)



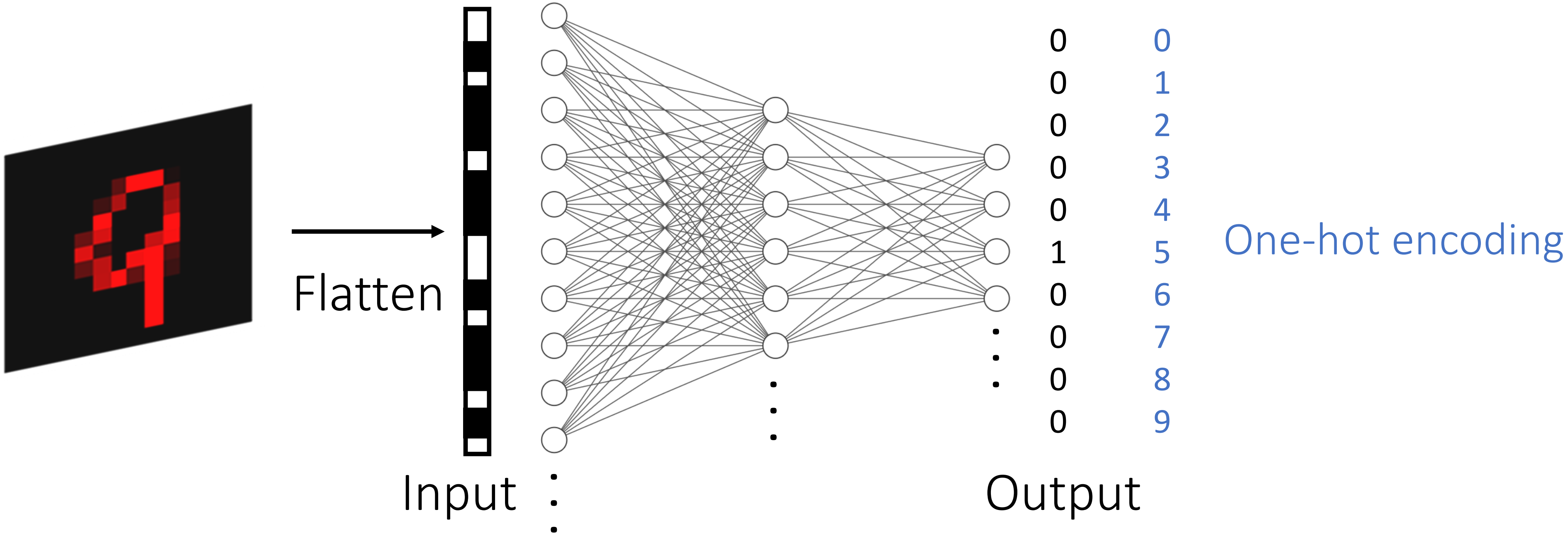
MLP for Image Classification (Lab 2)



(3072 neurons) (4704 neurons)

$$W^{[1]} = [n^{[l-1]}, n^{[l]}] = [3072, 4704] \approx 14M$$

MLP for Image Classification (Lab 2)

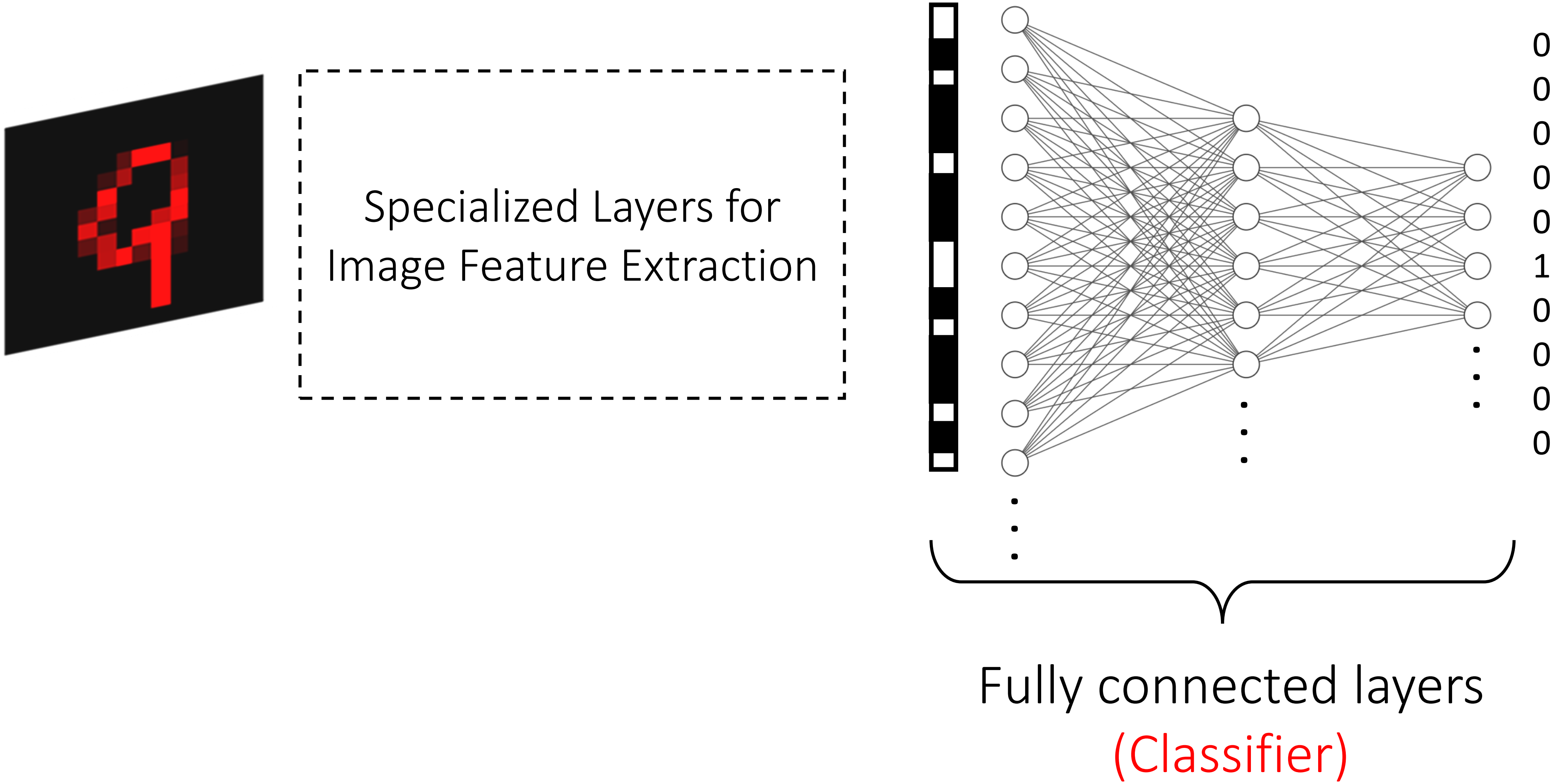


Great at Classification

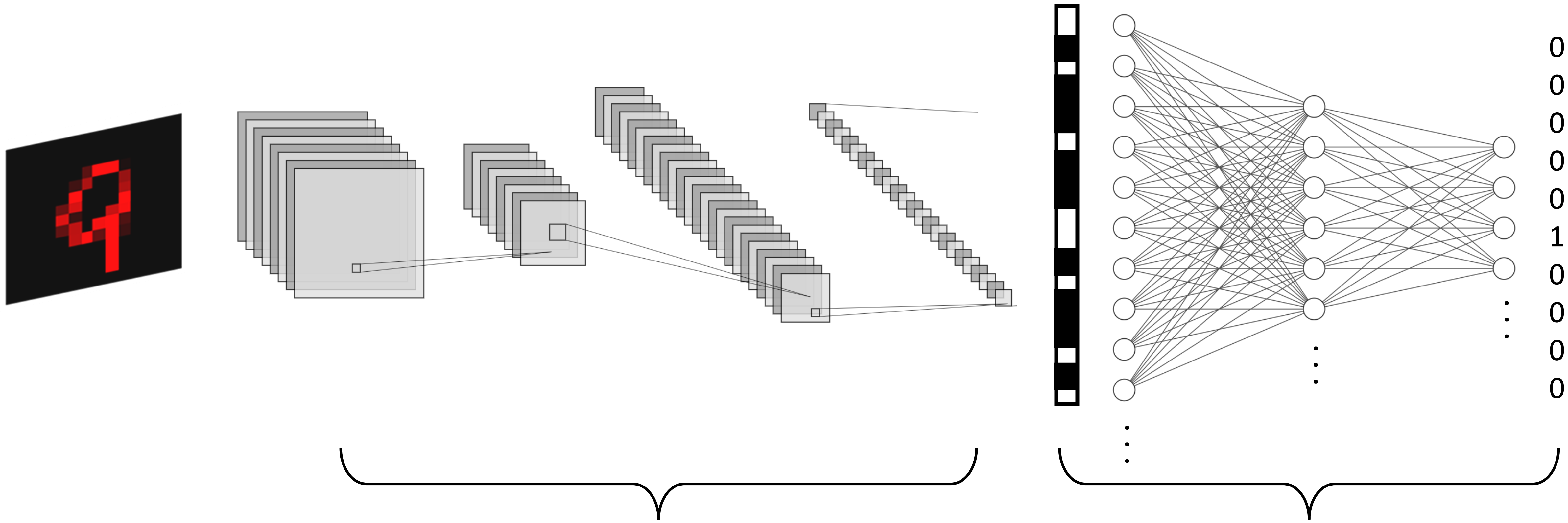
Not as good with Extracting image features

Too many parameters when Flattening images

Specialized Layers for Feature Extractions



Full CNN Architecture



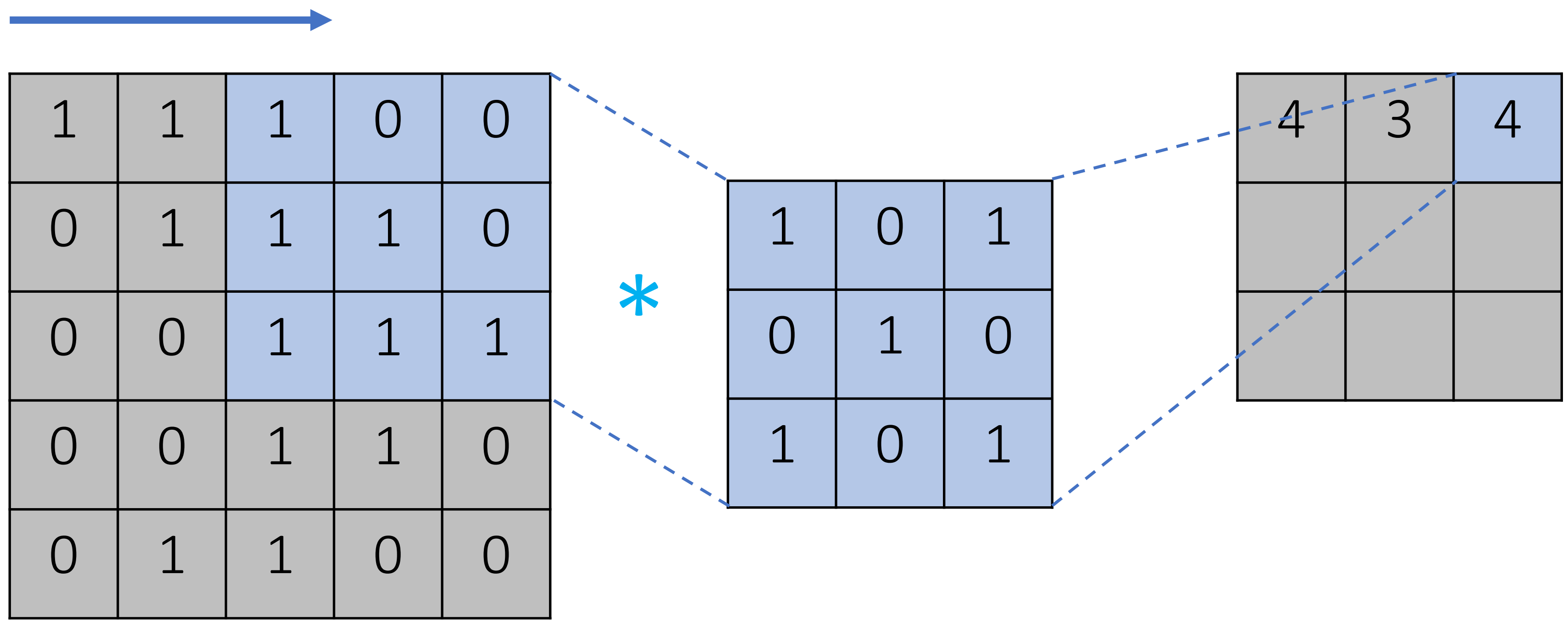
Convolution Layers + Pooling Layers
(Image feature extraction)

Fully connected layers
(Classifier)

PART 2:

Convolution Filters

Image Convolution



$$(1 * 1) + (0 * 0) + (0 * 1) +$$
$$(0 * 1) + (1 * 1) + (0 * 0) +$$
$$(1 * 1) + (0 * 0) + (1 * 1) +$$

Input Image
Layer $l-1$

Filter(Kernel)
W

Feature map
Layer l

Traditional Convolution Filters

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Traditional Convolution Filters

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

CNNs Learn these features instead of us guessing

Traditional Convolution Filters

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

CNNs Learn these features instead of us guessing

1000 filters

3x3=9*1000 = **9K** parameters

Traditional Convolution Filters

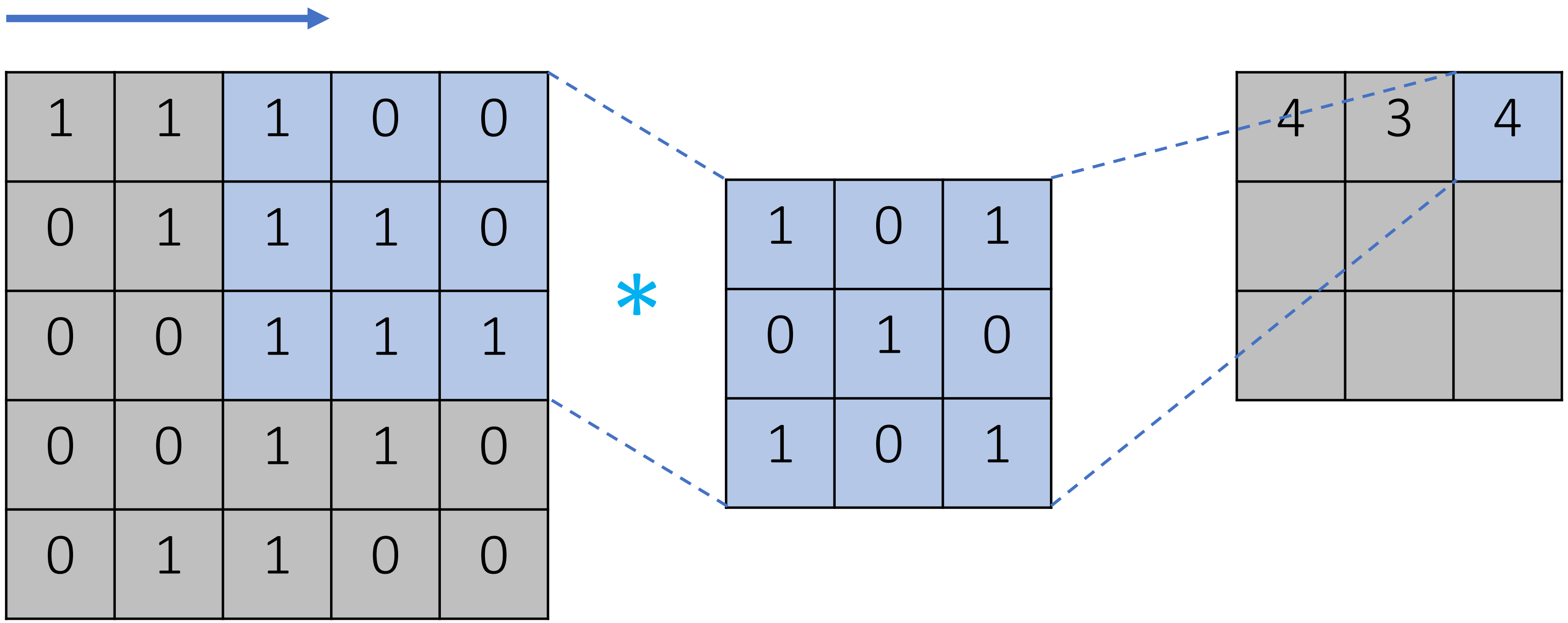
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

CNNs Learn these features instead of us guessing

1000 filters
3x3=9*1000 = **9K** parameters

14M vs 9k
Several orders of magnitude of difference in parameters

Convolution Dimensions

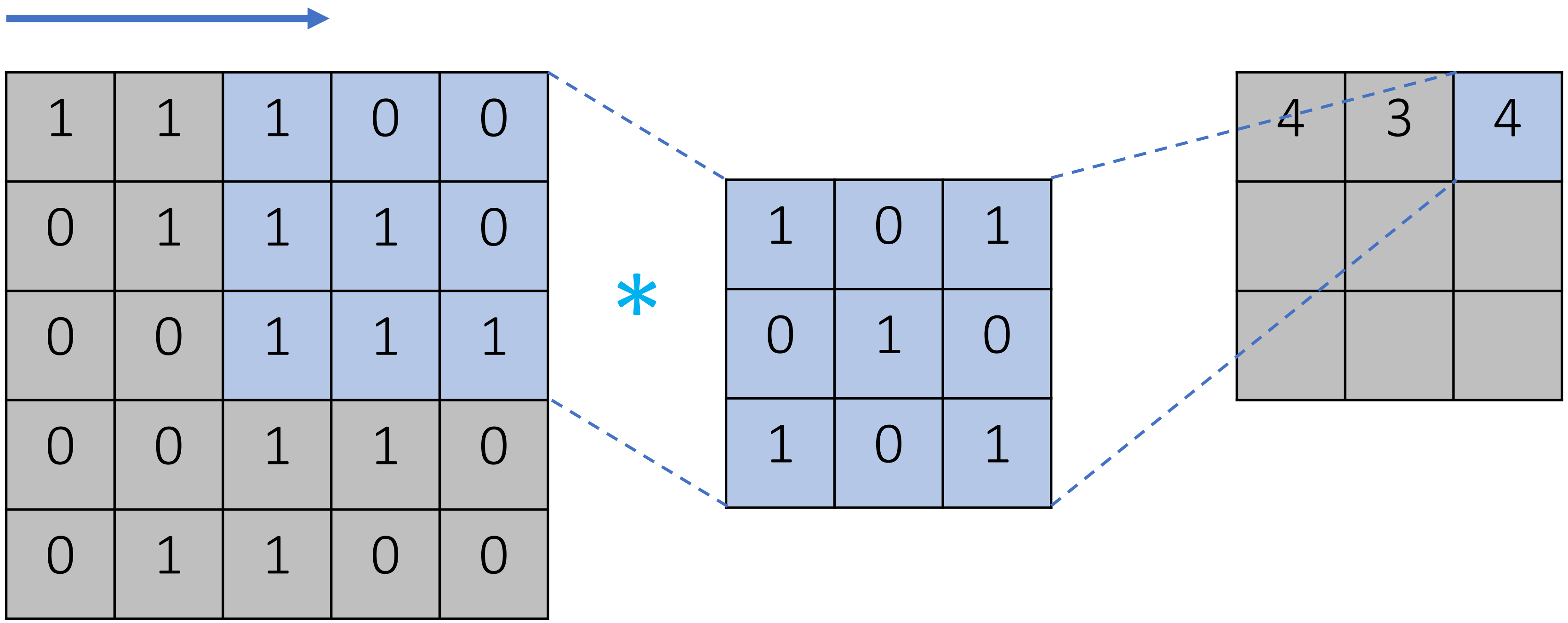


Input Image
(5 x 5)

Filter
(3 x 3)

Convoluted Feature
(3 x 3)

Convolution Dimensions



Input Image	Filter	Convoluted Feature
(5 x 5)	(3 x 3)	(3 x 3)
n x n	f x f	(n – f + 1) x (n – f + 1)

Stride

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input Image

*

1	0	1
0	1	0
1	0	1

Filter

4		

Convoluted Feature

Stride = 1

Stride

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input Image

*

1	0	1
0	1	0
1	0	1

Filter

4	3	

Convoluted Feature

Stride = 1

Stride

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input Image

*

1	0	1
0	1	0
1	0	1

Filter

4	3	4

Convoluted Feature

Stride = 1

Stride

Input = 5 x 5

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input Image

*

1	0	1
0	1	0
1	0	1

Filter

Output = 3 x 3

4	3	4
3		

Convoluted Feature

Stride = 1

Stride

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input Image

*

1	0	1
0	1	0
1	0	1

Filter

4	

Convoluted Feature

Stride = 2

Stride

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input Image

$*$

1	0	1
0	1	0
1	0	1

Filter

4	4

Convoluted Feature

Stride = 2

Stride

Input = 5 x 5

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input Image

*

1	0	1
0	1	0
1	0	1

Filter

Output = 2 x 2

4	4
2	

Convoluted Feature

Stride = 2

Padding

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input Image
(5x5)

Padding = 1

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Padded Image
(7x7)

Padding

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

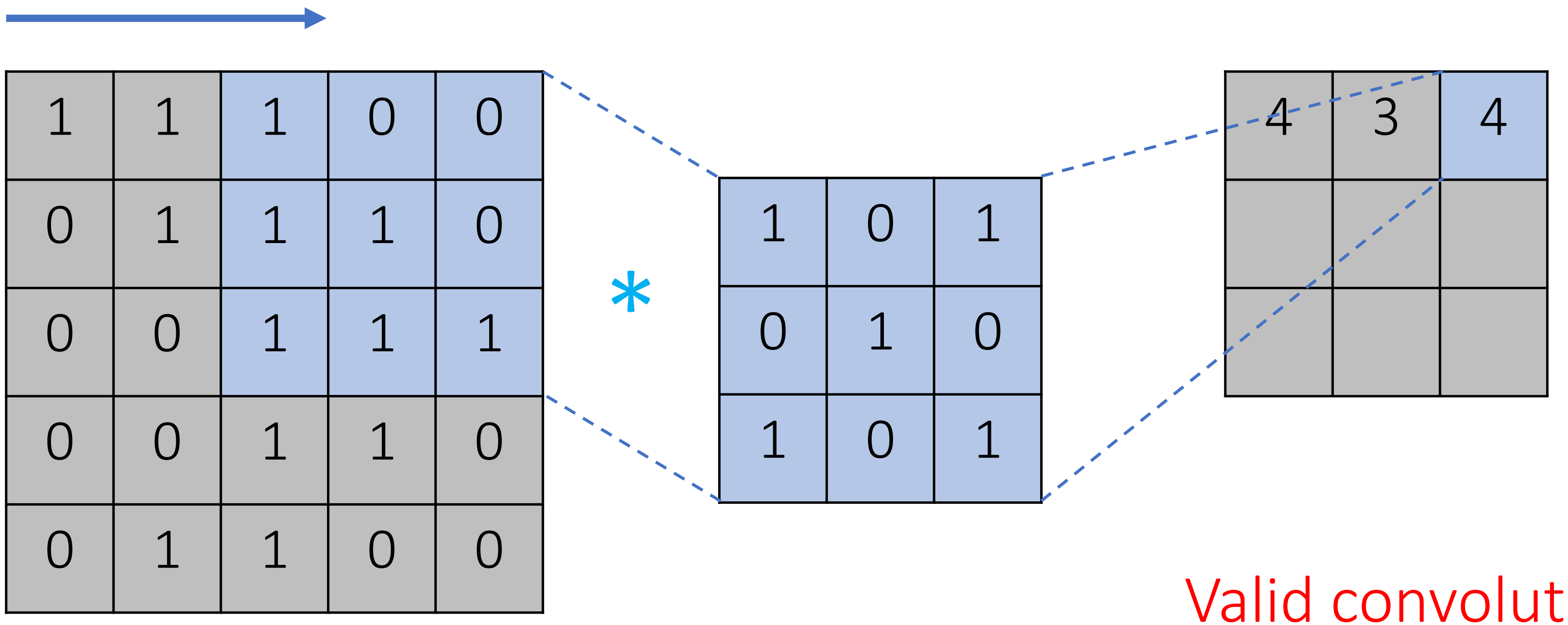
Input Image
(5x5)

Padding = 2

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0
0	0	0	0	1	1	1	0	0
0	0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Padded Image
(9x9)

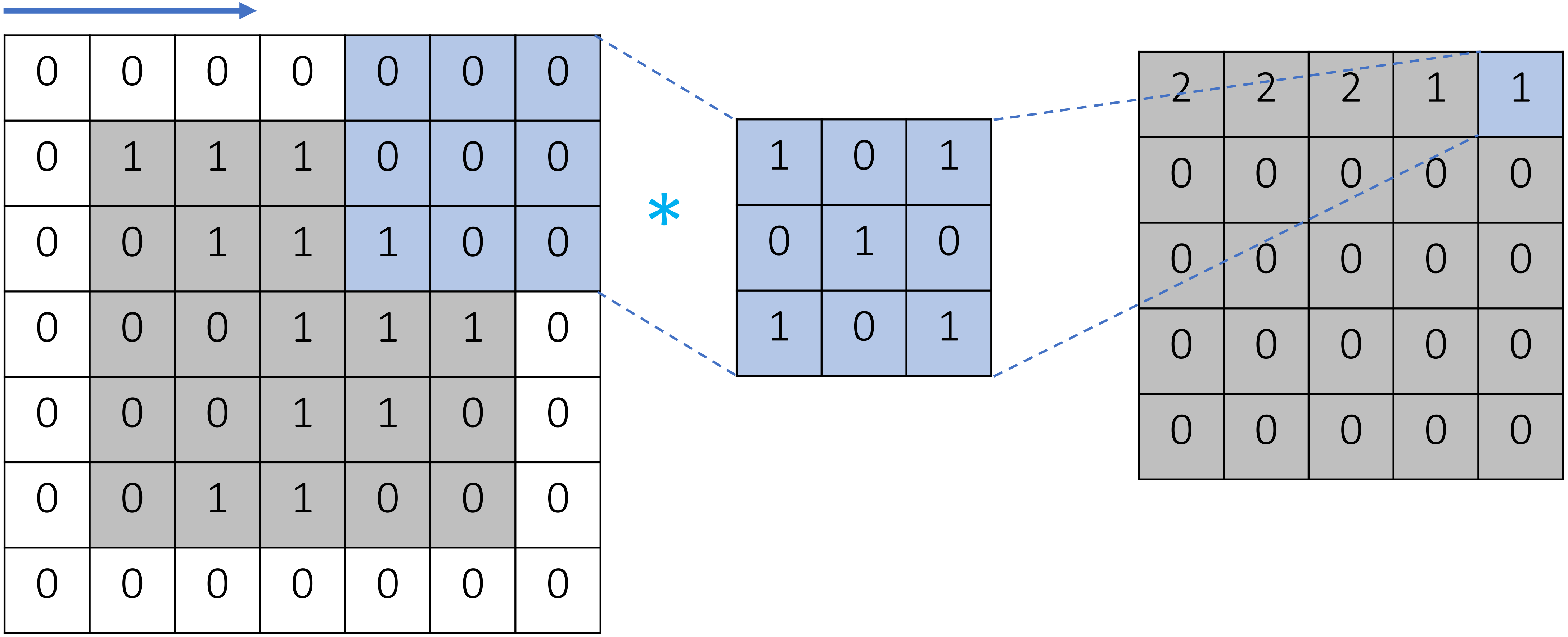
Convolution Dimensions



Valid convolution

Input Image	Filter	Convoluted Feature
(5 x 5)	(3 x 3)	(3 x 3)
n x n	f x f	(n – f + 1) x (n – f + 1)

Convolution Dimensions



Padded Input Image
(7 x 7)
n x n

Filter
(3 x 3)
f x f

Convoluted Feature
(5 x 5)
(n + 2p - f + 1) x (n + 2p – f + 1)

Convolution Dimensions

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Padded Input Image
(7 x 7)
n x n

$*$

1	0	1
0	1	0
1	0	1

$p = (f - 1)/2$

Filter
(3 x 3)
f x f

2	2	2	1	1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

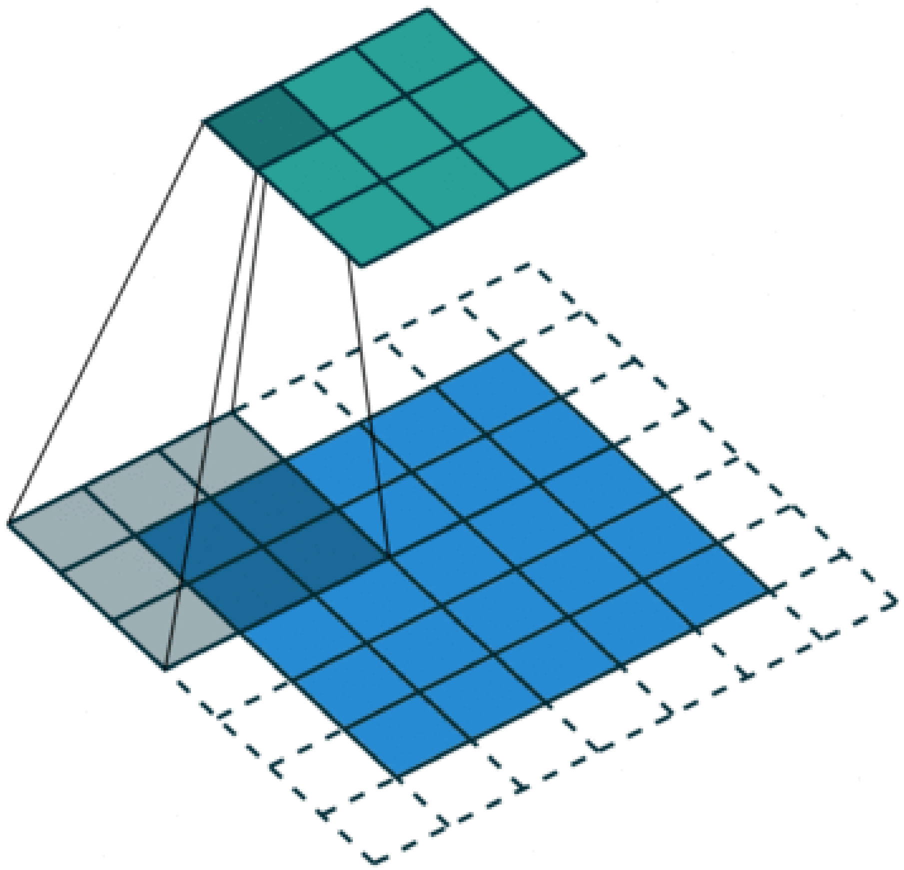
Same convolution

Convoluted Feature
(5 x 5)
(n + 2p - f + 1) x (n + 2p – f + 1)

Convolution Dimensions

Feature	Valid	Same
Padding	No	Yes
Output size	Smaller than input	Same size as input
Aim	Apply convolution on valid regions only	Preserve spatial dimensions

Generalized Dimensions



$$(n) * (n)$$

$$\downarrow$$

$$\left(\frac{n + 2p - f}{s} + 1\right) * \left(\frac{n + 2p - f}{s} + 1\right)$$

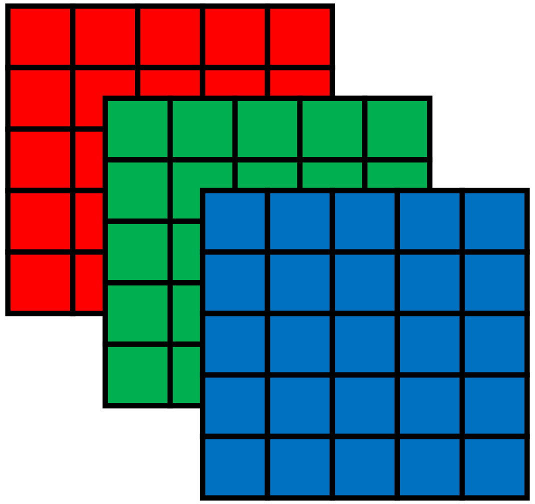
n: original image dimensions

p: padding size

f: filter dimension

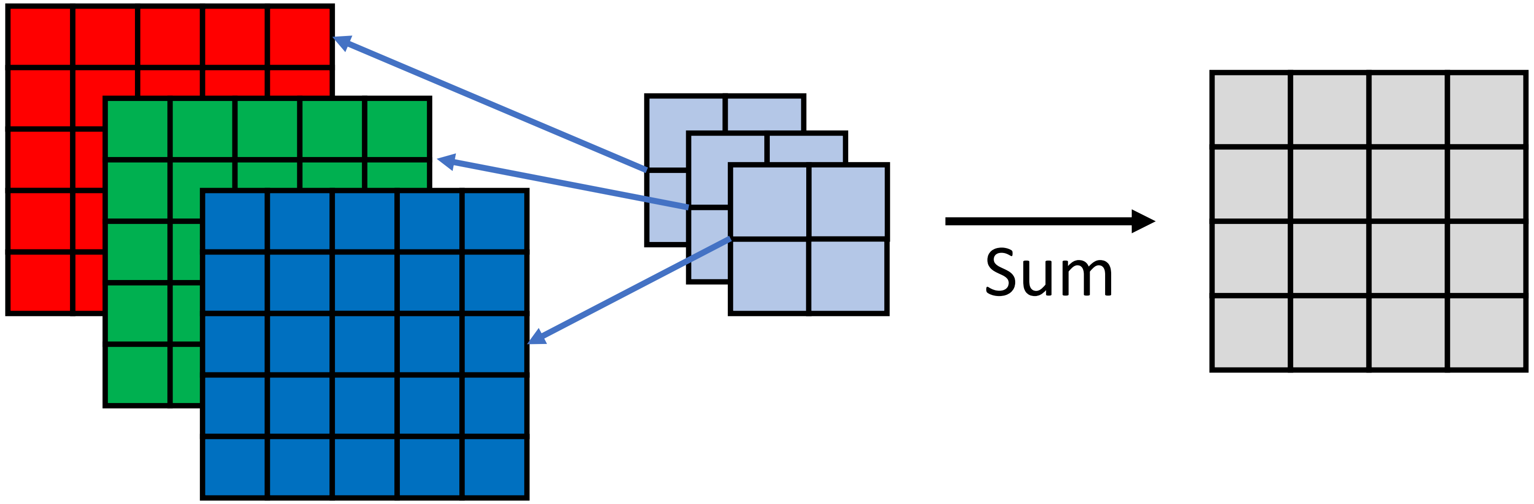
s: stride

Volume Convolution



Input
(5x5x3)

Volume Convolution



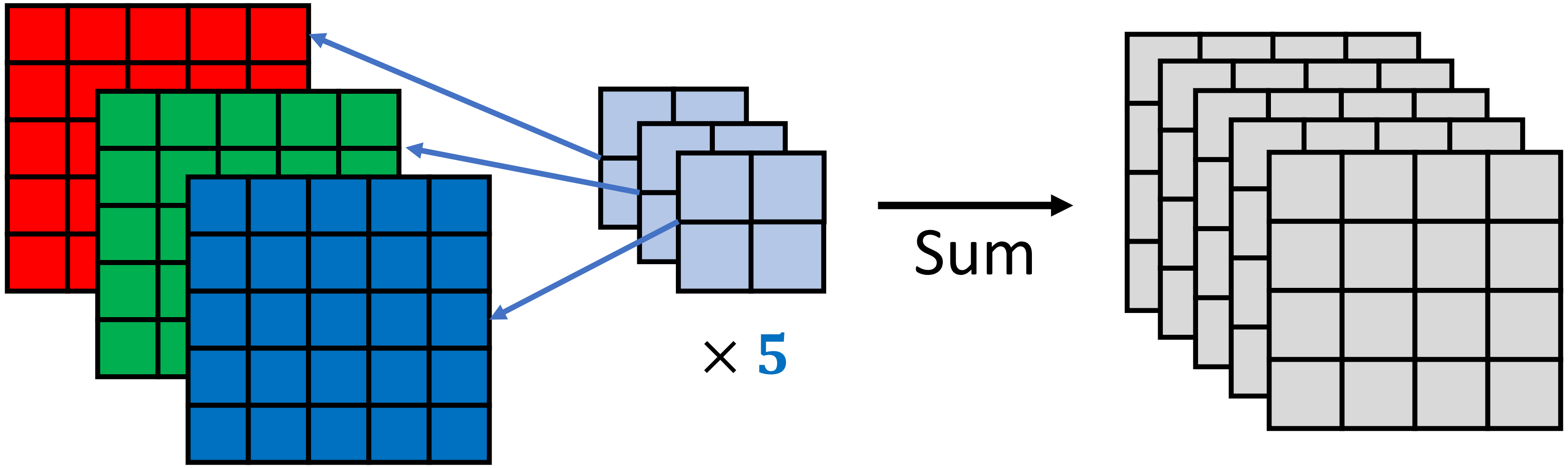
Input
(5x5x3)

Kernel
(2x2x3)

Output
(4x4x1)

(Height x Width x Channels)

Volume Convolution (multiple filters)



Input
(5x5x3)

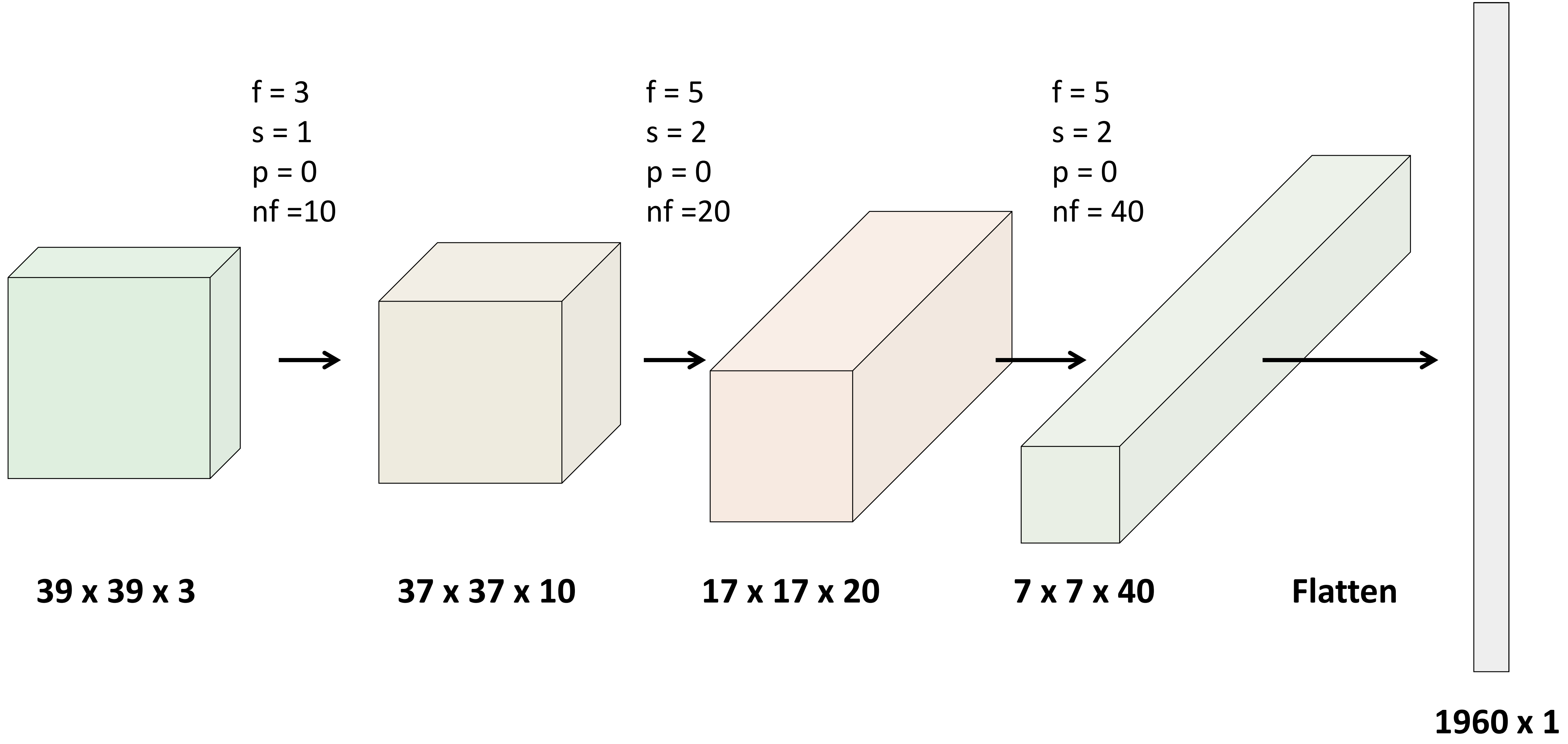
Kernels
(5x2x2x3)

Output
(5x4x4x1)

PART 3:

Composing CNNs

CNN example



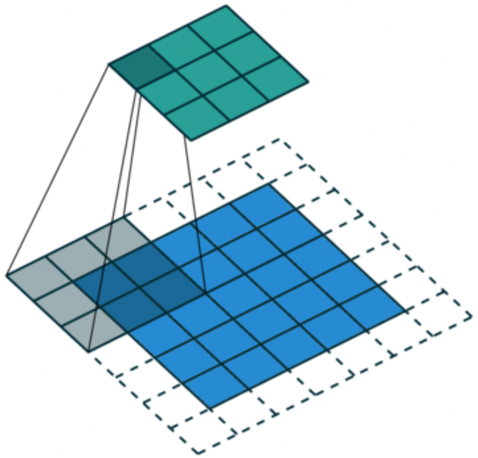
f = 3
s = 1
p = 0
nf =10
f = 5
s = 2
p = 0
nf =20
f = 5
s = 2
p = 0
nf = 40
39 x 39 x 3
37 x 37 x 10
17 x 17 x 20
7 x 7 x 40
Flatten
1960 x 1

Typical CNN Layers

- **Convolutional Layer (CONV)**
- Pooling Layer (POOL)
- **Fully Connected (FC)**

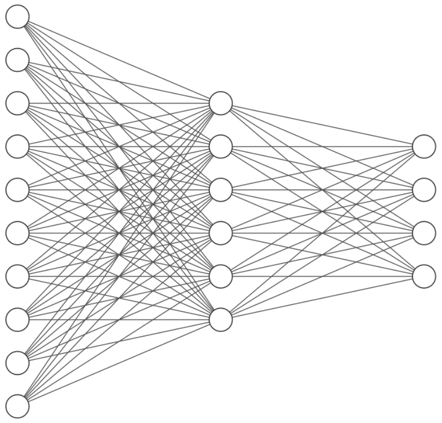
Typical CNN Layers

- **Convolutional Layer (CONV)**



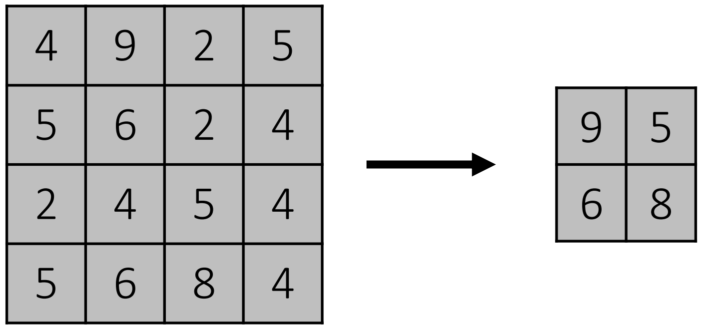
- Pooling Layer (POOL)

- **Fully Connected (FC)**



Typical CNN Layers

- Convolutional Layer (CONV)
- **Pooling Layer (POOL)**
- Fully Connected (FC)



Max Pooling and Average Pooling

4	9	2	5
5	6	2	4
2	4	5	4
5	6	8	4

→

9	5
6	8

Max pool

4	9	2	5
5	6	2	4
2	4	5	4
5	6	8	4

→

6.0	3.3
4.3	5.3

Average pool

Pooling Layers

1	2	5	10	7
0	5	1	4	0
0	4	9	3	15
0	0	2	4	5
0	7	2	0	0

Input Image

Max pool

Dim= 3 x 3

Stride = 1

9		

Pooled Feature

Pooling Layers

1	2	5	10	7
0	5	1	4	0
0	4	9	3	15
0	0	2	4	5
0	7	2	0	0

Input Image

Max pool

Dim= 3 x 3

Stride = 1

9	10	

Pooled Feature

Pooling Layers

1	2	5	10	7
0	5	1	4	0
0	4	9	3	15
0	0	2	4	5
0	7	2	0	0

Input Image

Max pool

Dim= 3 x 3

Stride = 1

9	10	15

Pooled Feature

Pooling Layers

1	2	5	10	7
0	5	1	4	0
0	4	9	3	15
0	0	2	4	5
0	7	2	0	0

Input Image

Avg pool

Dim= 3 x 3

Stride = 1

3		

Pooled Feature

Pooling Layers

1	2	5	10	7
0	5	1	4	0
0	4	9	3	15
0	0	2	4	5
0	7	2	0	0

Input Image

Avg pool

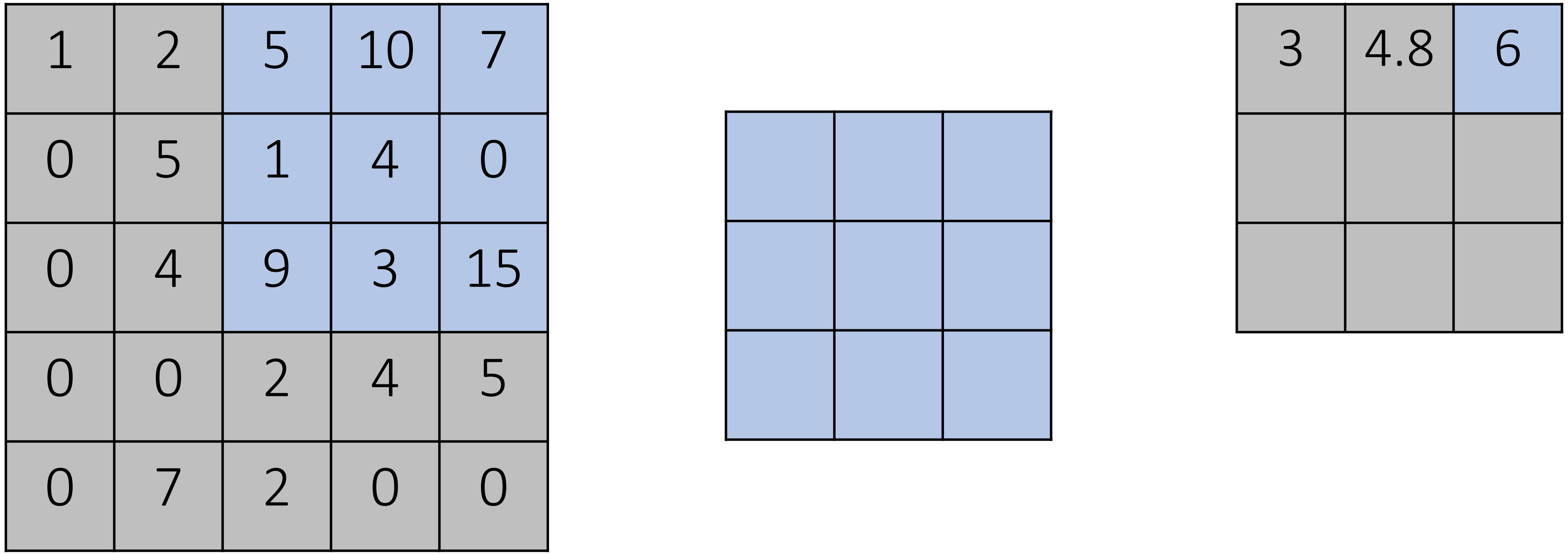
Dim= 3 x 3

Stride = 1

3	4.8	

Pooled Feature

Pooling Layers



Input Image

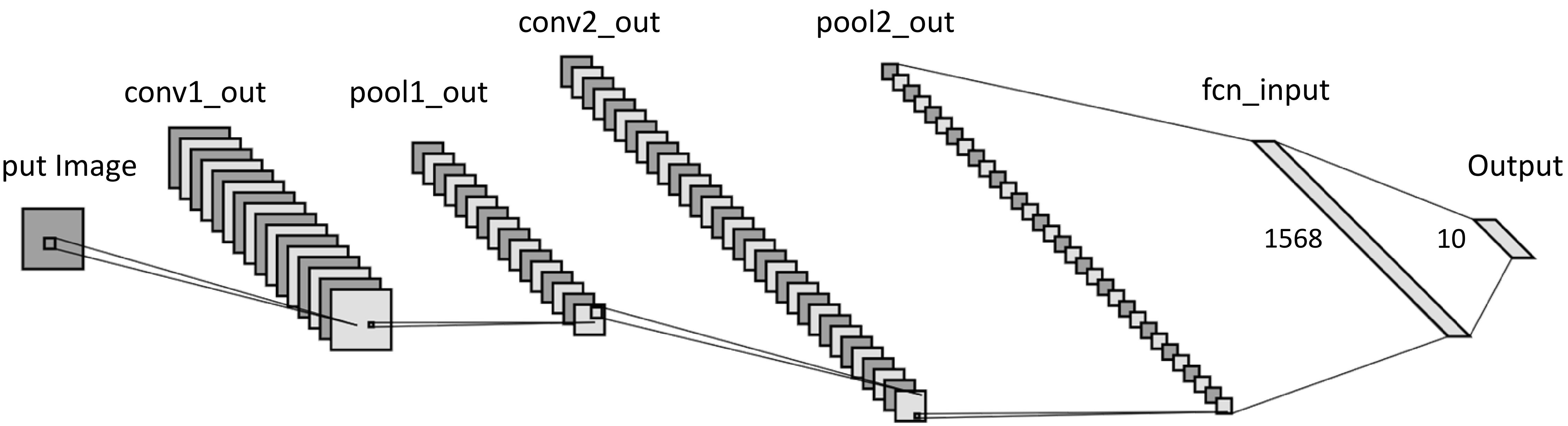
Avg pool

Pooled Feature

Dim= 3 x 3

Stride = 1

Full CNN example



self.cnn1	self.maxpool1	self.cnn2	self.maxpool2	Flatten	self.fc1
• In-channel # : 1 • Out-channel # : 16 • Kernel size : 5 • Stride = 1 • Padding = 2 • ReLU	Kernel size : 2	• In-channel # : 16 • Out-channel # : 32 • Kernel size : 5 • Stride = 1 • Padding = 2 • ReLU	Kernel size : 2		1568 → 10

Benefits of CNNs

Parameter Sharing

Filter can be useful in different parts of the input (image)

Sparsity of Connections

- In each layer each output value depends only on small number of inputs
  (Pixel at layer l only depend on subset of pixels in layer $l-1$)

- Translation invariance
  (network recognizes patterns regardless of its position in the image)

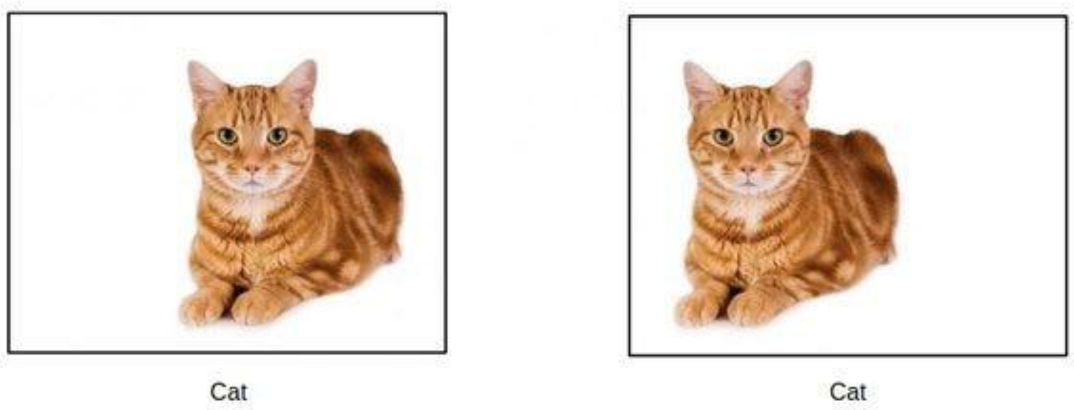
Benefits of CNNs

Sparsity of Connections

- In each layer each output value depends only on small number of inputs (Pixel at layer l only depend on subset of pixels in layer $l-1$)

- Translation invariance (network recognizes patterns regardless of its position in the image)

Image credit: Comet



Challenges of CNNs

Computational Complexity

Convolutions are expensive $O(N^2 n^4)$

Deeper Structure Needed

In each layer each output value depends only on small number of inputs (local)

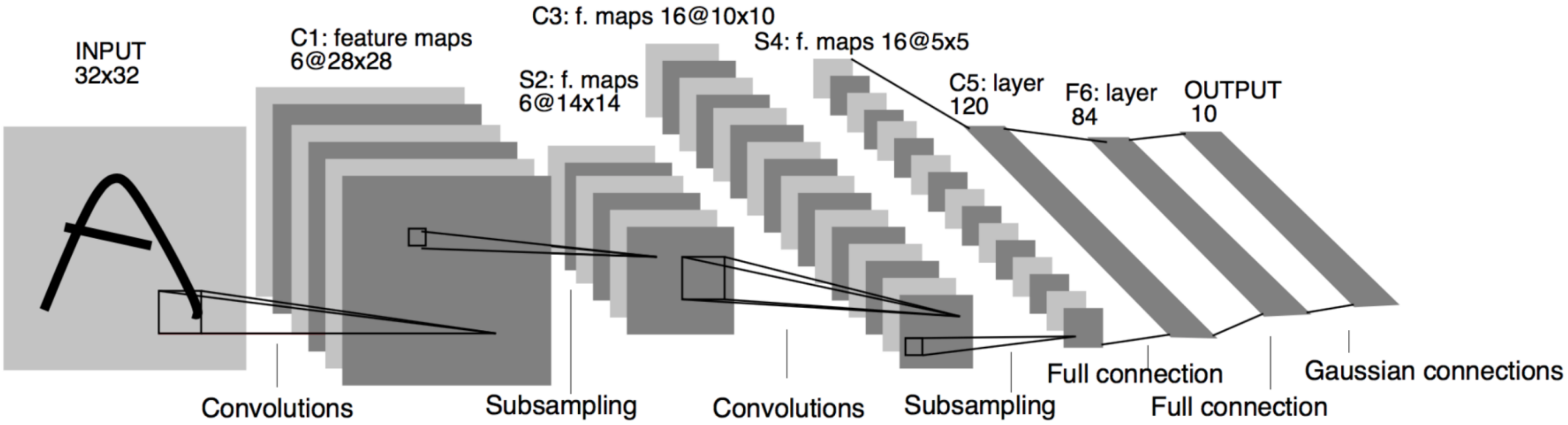
Popular CNN Architectures (LeNet 5)

Yann LeCun

Leon Bottou

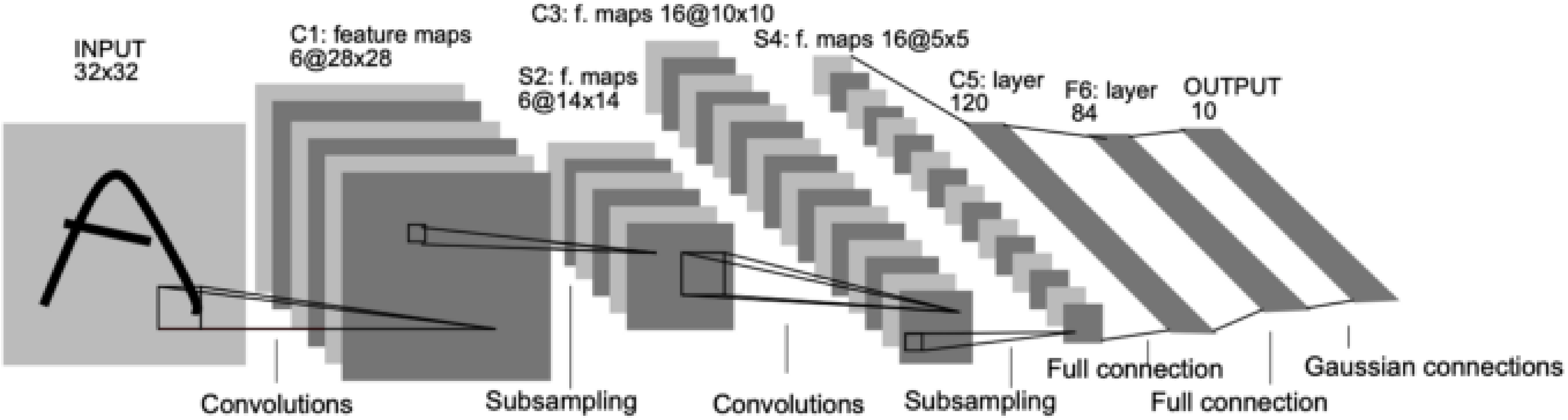
Yoshua Bengio

Patrick Haffner



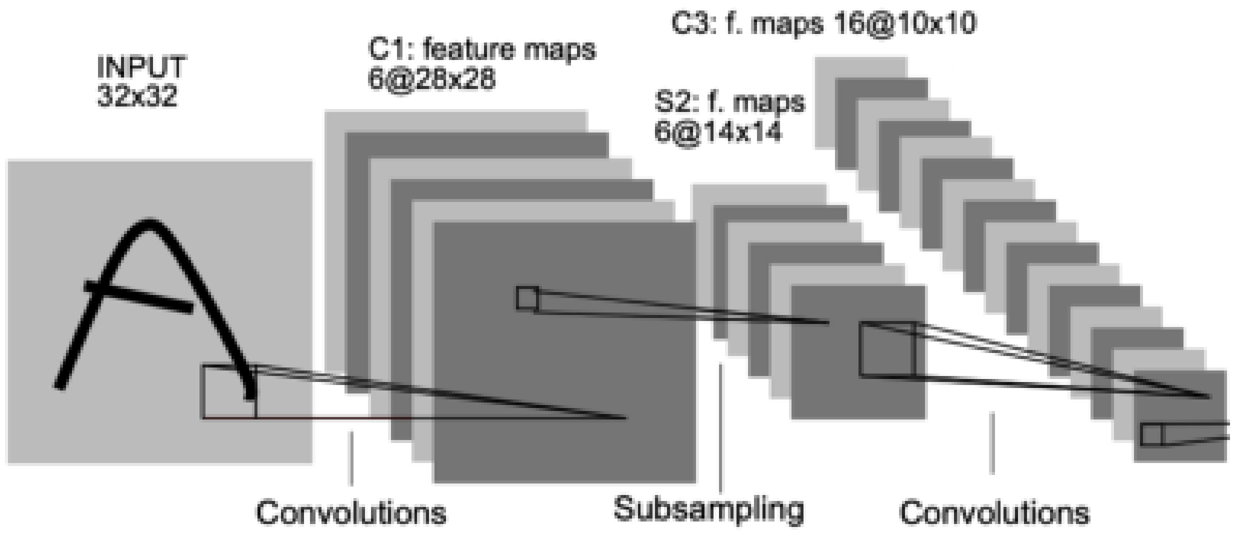
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

LeNet-5 (1998)



INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

LeNet-5 (1998)



Layer 1:

- Convolutional Layer with 6 kernels
- kernel size of 5x5
- Padding = 2, stride = 1

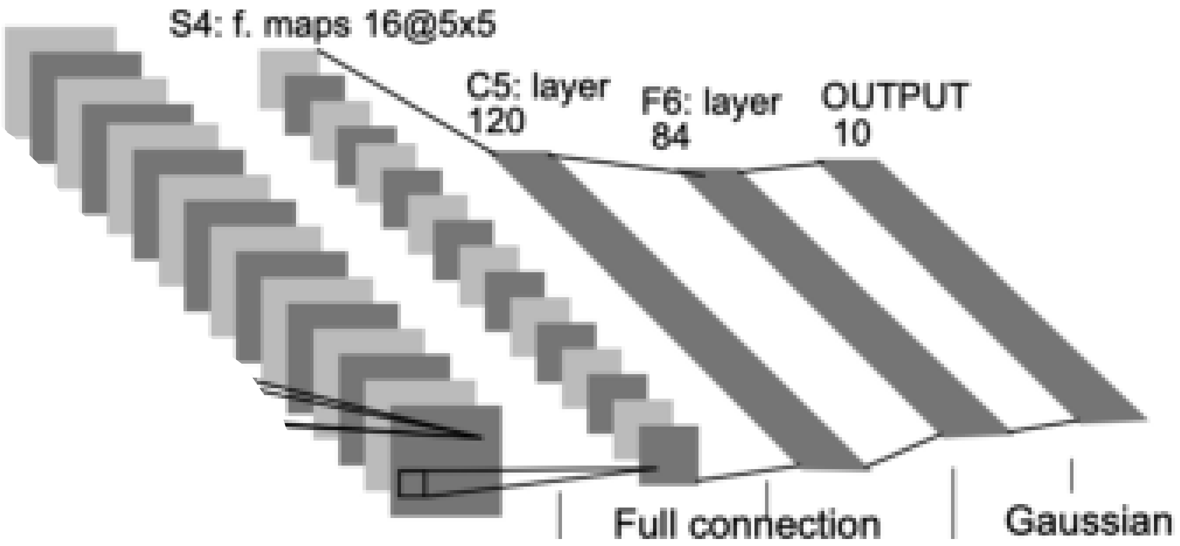
Layer 2:

- Average pooling (2x2 kernel)

Layer 3:

- Convolutional layer with 16 kernels
- kernel size of 5x5
- Padding = 0, stride = 1

LeNet-5 (1998)



Layer 4:

Average pooling (2x2 kernel)

Layer 5:

- Convolutional layer with 120 kernels
- Kernel size of 5x5
- Padding = 0, stride = 1

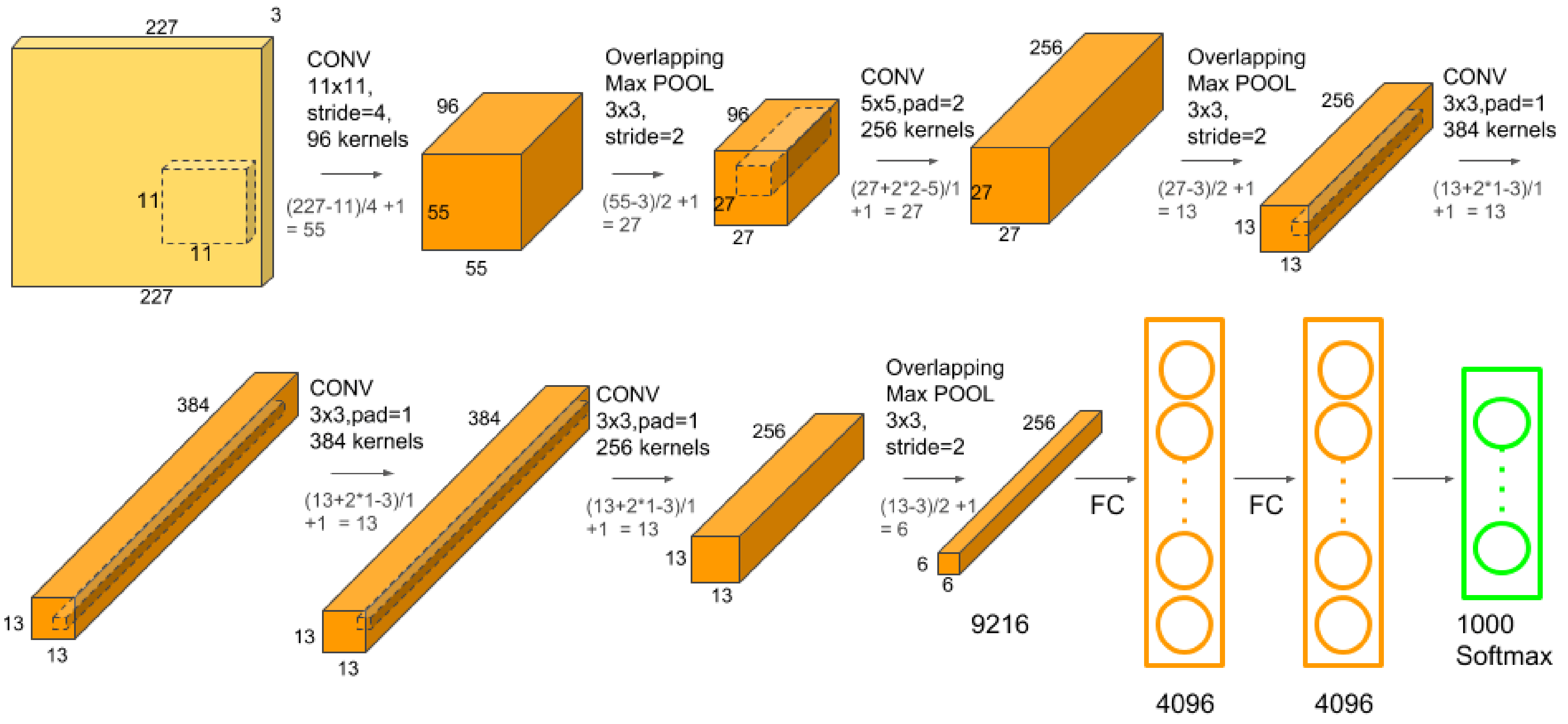
Layer 6:

- Fully Connected Layer
- Input dimension = 120
- Output dimension = 84

Layer 7:

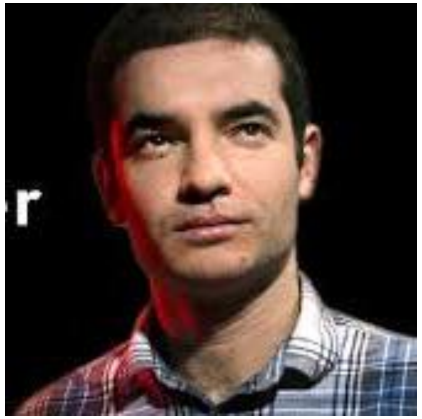
- Fully Connected Layer
- Input dimension = 84
- Output dimension = 10

AlexNet (2012)



AlexNet (2012)

Alex Krizhevsky

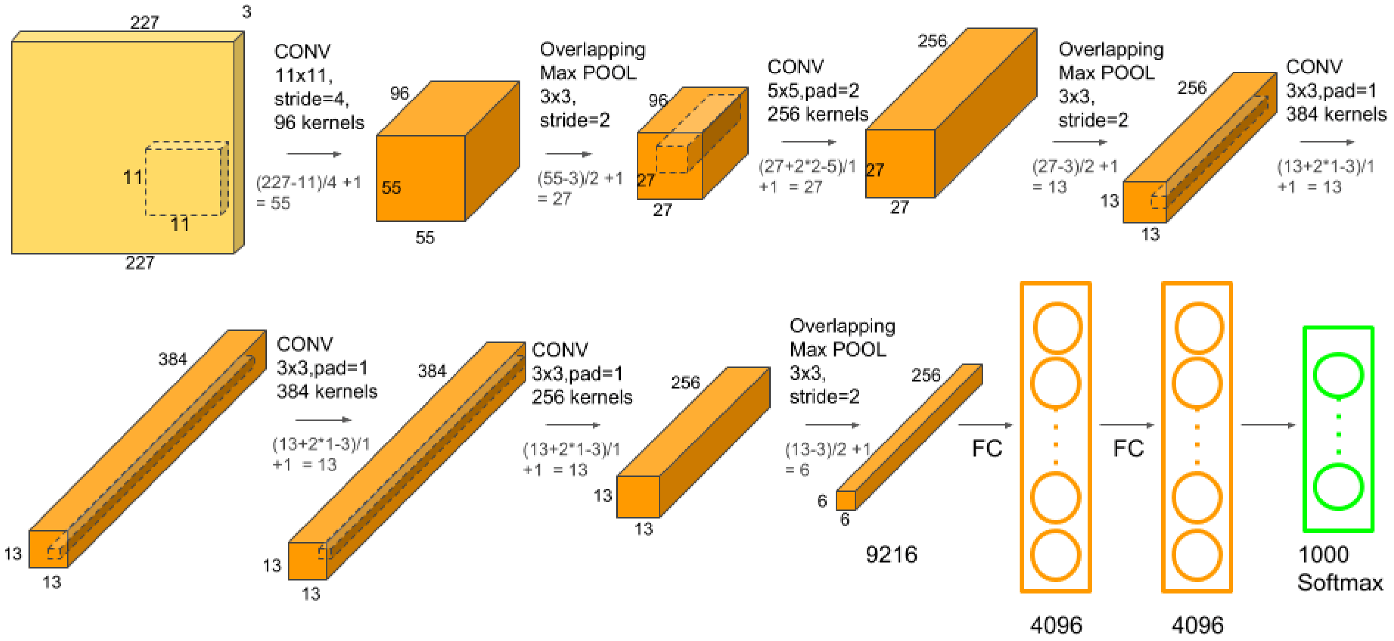


Ilya Sutskever

Geoffrey Hinton

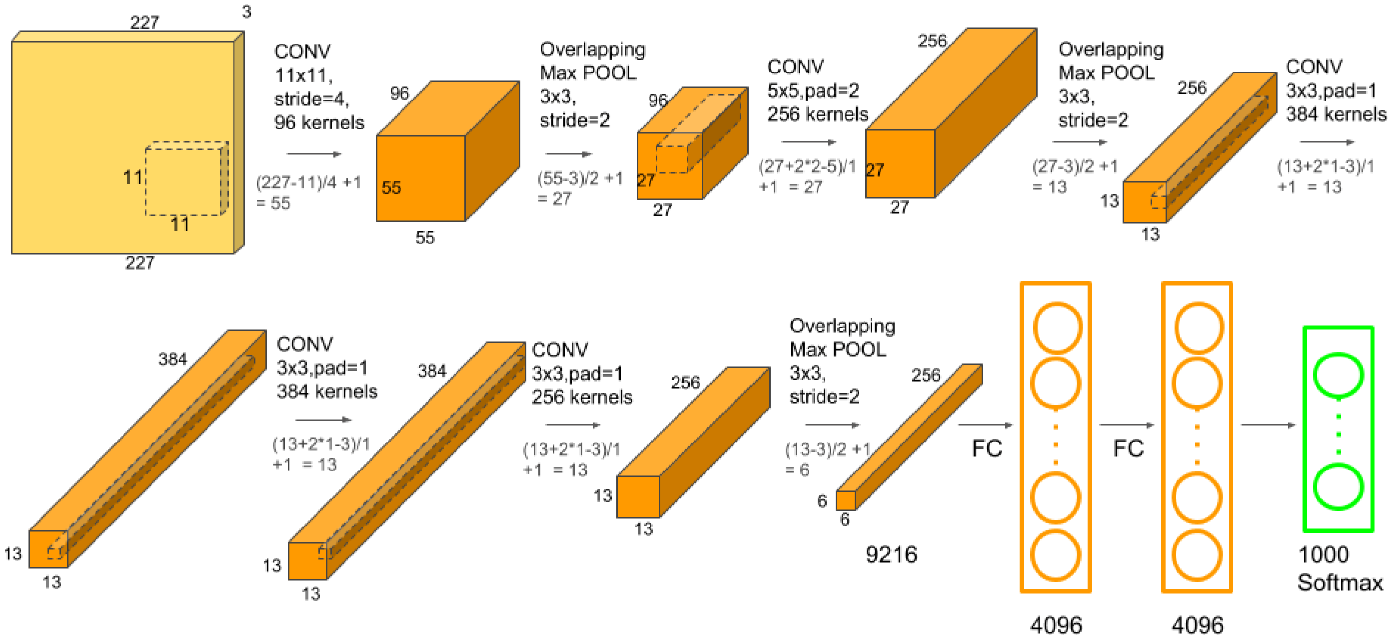
Krizhevsky et al., Imagenet classification with deep convolutional neural networks, 2012

Parameters (AlexNet)



Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096×1	37,748,736	4,096	37,752,832
FC-2	4096×1	16,777,216	4,096	16,781,312
FC-3	1000×1	4,096,000	1,000	4,097,000
Output	1000×1	0	0	0
Total				**62,378,344**

Parameters (AlexNet)



- Much bigger than LeNet (60M parameters)
- ReLU
- Multiple GPUs
- Local Response Normalization (LRN)

VGG-16 (2014)

CONV: f=3, s=1, same

POOL: f=2, s=2

Order: CCP CCP CCCP CCCP CCCP FFS

Nf: 2^6 2^7 2^8 2^9 2^9

~138 mil parameters

VGG-16 (2014)

CONV: f=3, s=1, same

POOL: f=2, s=2

Order: CCP CCP CCCP CCCP CCCP FFS

Nf: 2^6 2^7 2^8 2^9 2^9

~138 mil parameters

- Multiple convolution layers
- Smaller convolution filters
- Modularized architecture (VGG-19)