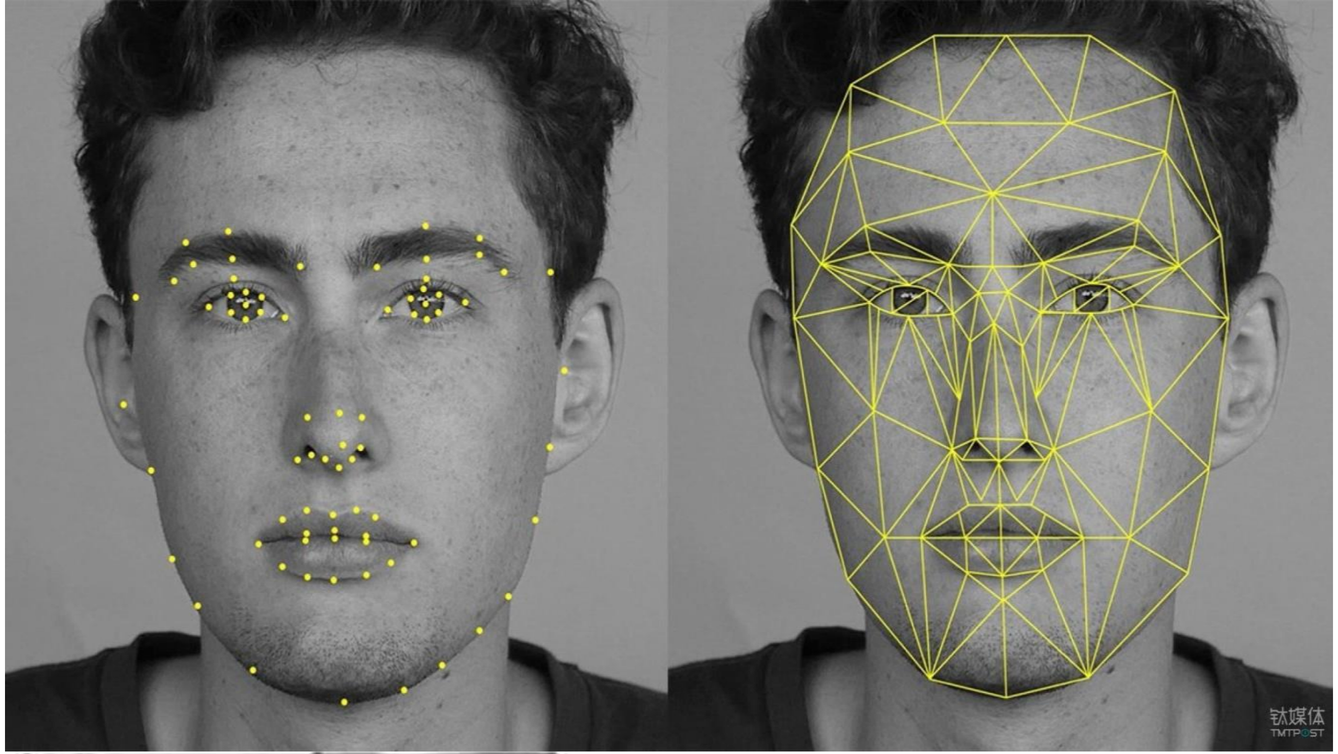
Convolutional Neural Networks

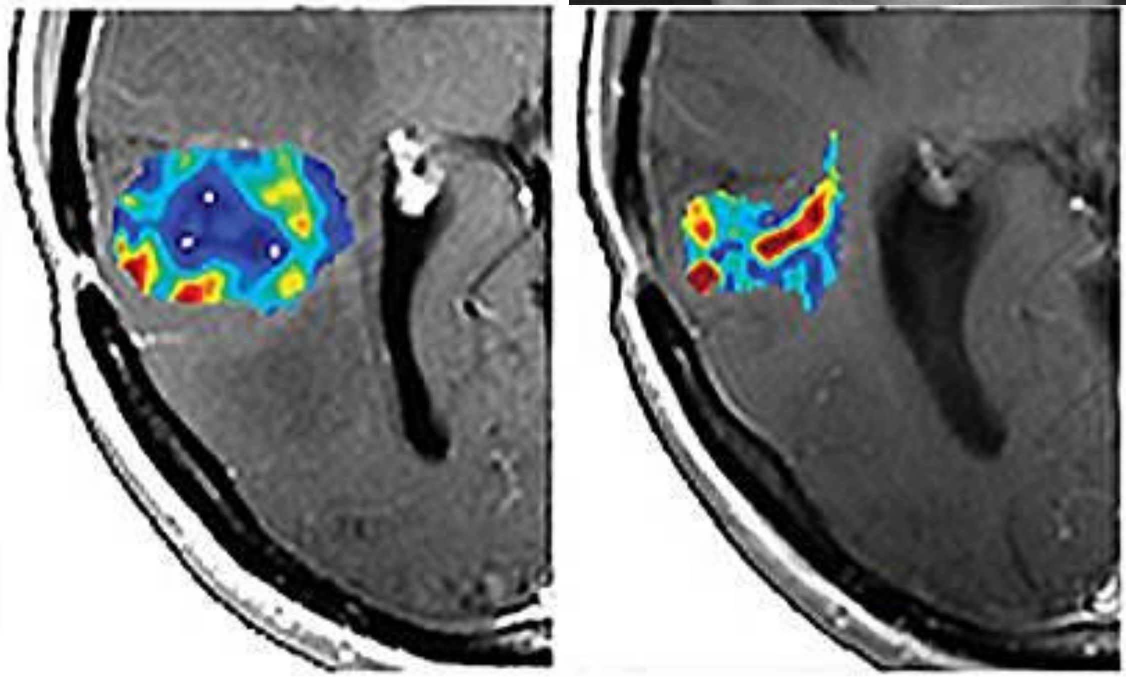
Computer Vision using Deep Learning

- Objective: To build a strong computer vision system using deep learning
- To discover from images
  - What is present
  - Where they are present
  - What actions are taking place
  - To predict and anticipate events

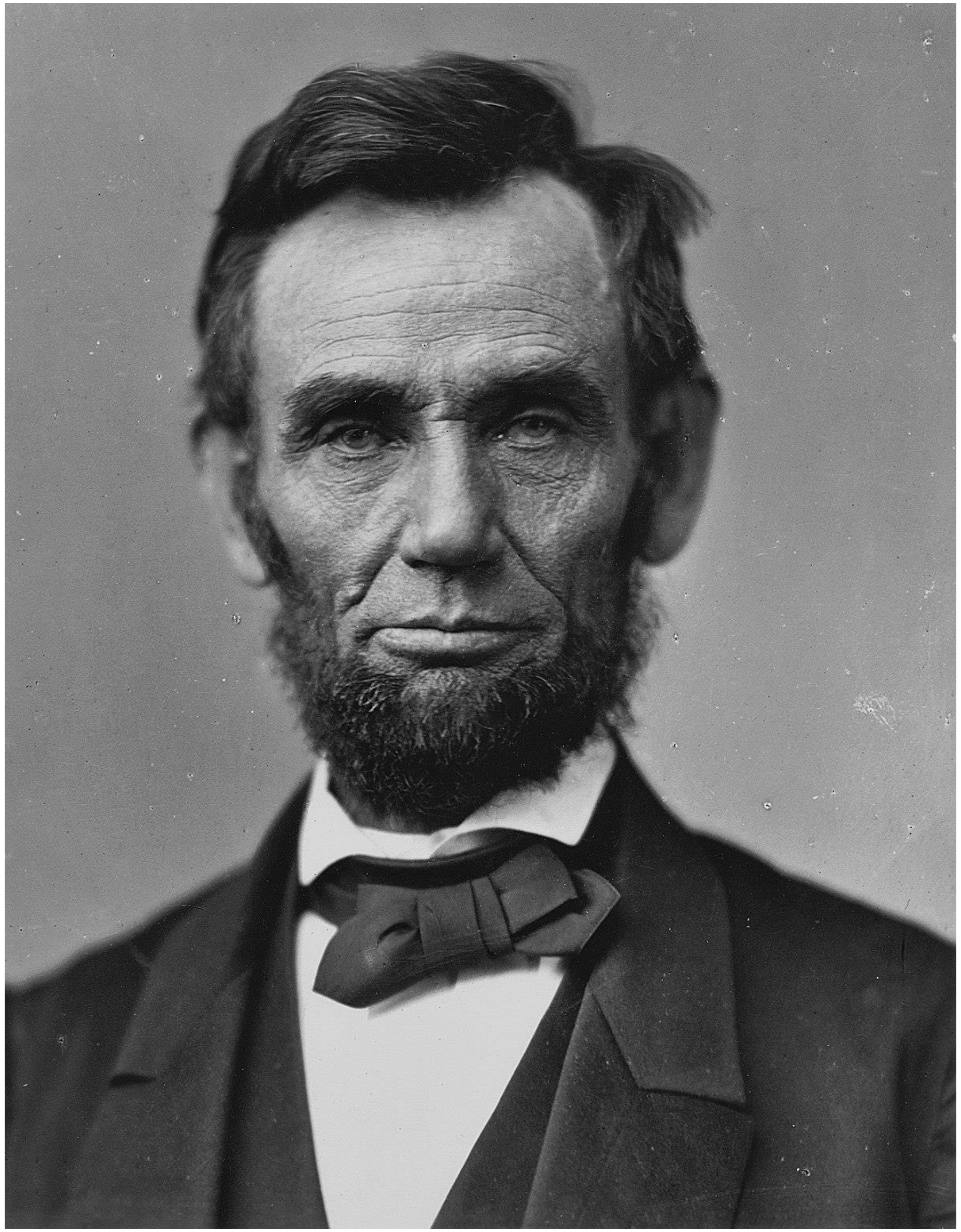

GAP

https://www.ptolemus.com/topics/autonomous-vehicles/

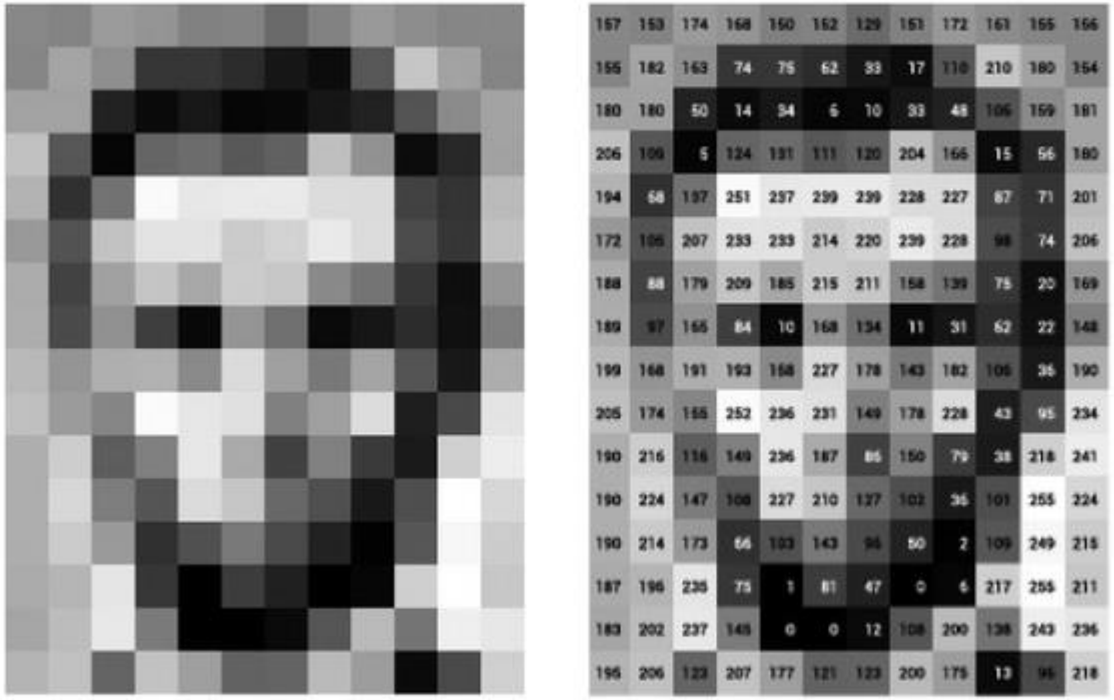




https://www.nih.gov/news-events/nih-research-matters/tumor-imaging-technique-tracks-responses-cancer-therapies
https://rapidapi.com/blog/top-facial-recognition-apis/



What computers see?



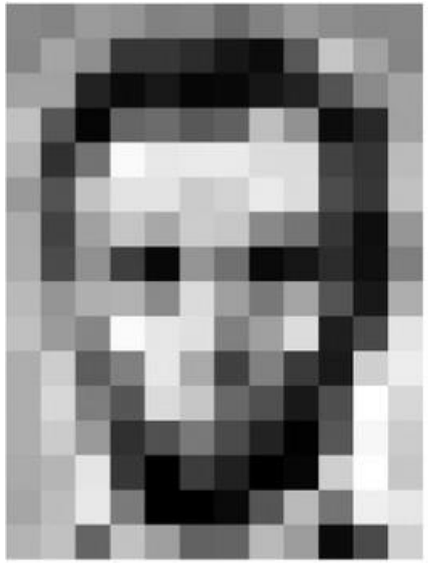
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Image is a matrix of numbers [0, 255]

Gray scale: 1080 x 1080

Colour (RGB): 1080 x 1080 x 3

Classification Task



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Washington	0.07
Lincoln	0.8
Obama	0.1
Trump	0.03

High Level Features

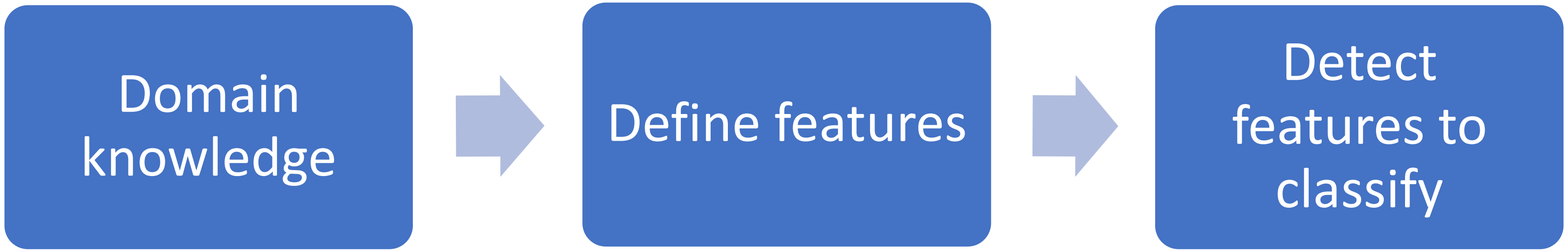
Key features in each image category

Eyes, nose,
mouth

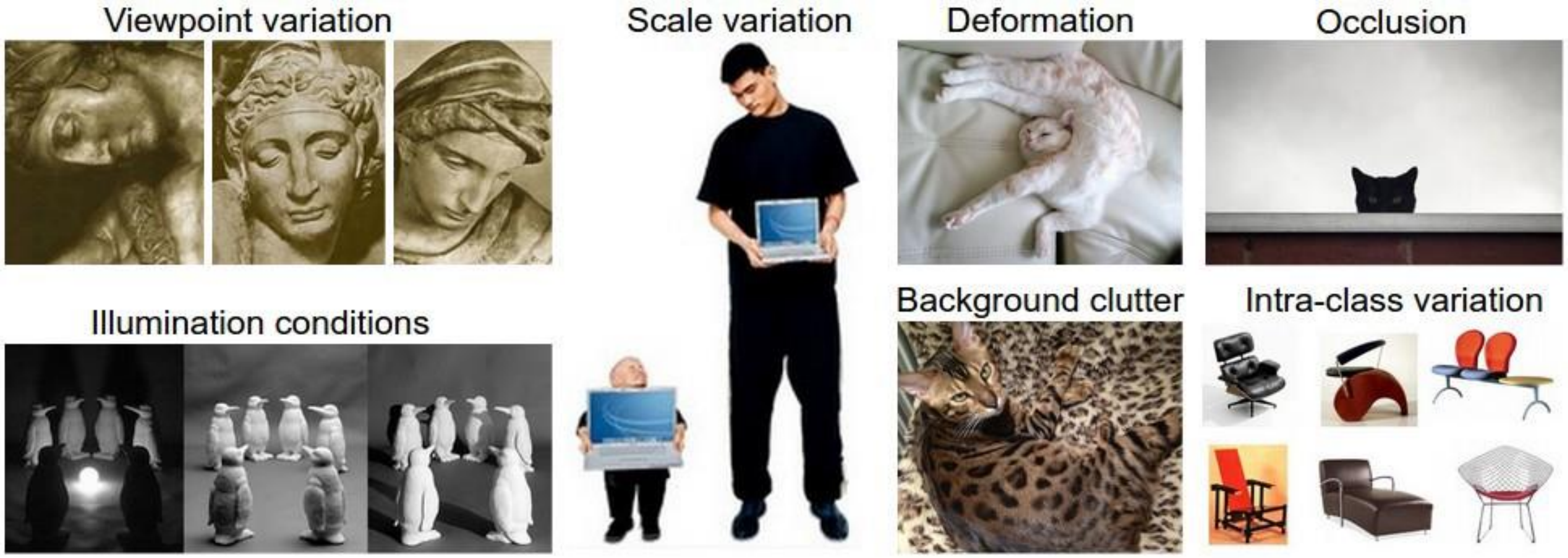
Wheels, license plate,
headlights

Windows, doors,
steps

Manual Feature Extraction



Problems????



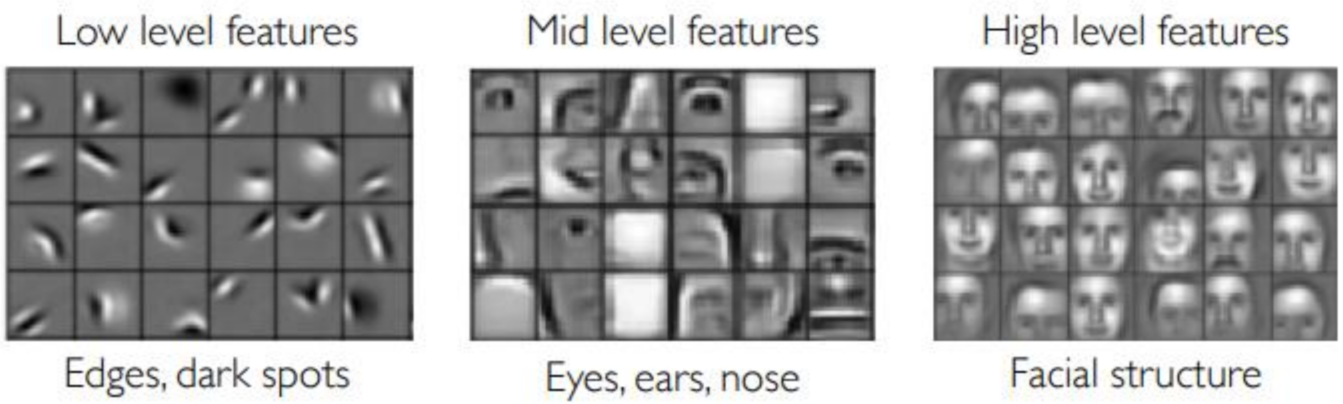
Classification pipeline needs to be invariant to these variations but be sensitive to pick out the inter-class variations. This is a challenging problem!!

Instead, can we extract and detect features automatically?

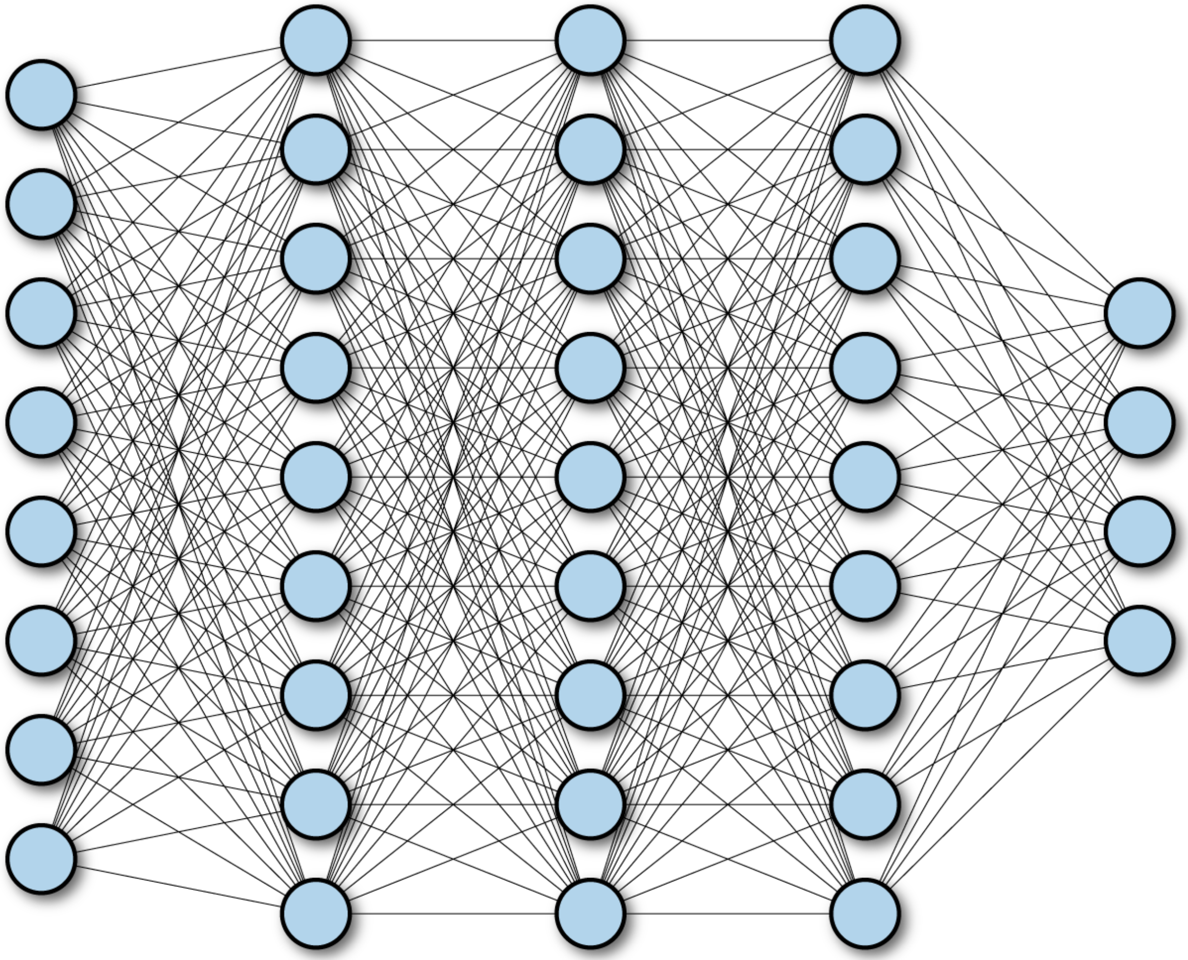
Learning Feature Representations

Learning heirarchy of features directly from images instead of hand-engineering

- Using neural network layers to learn



Fully Connected Neural Network

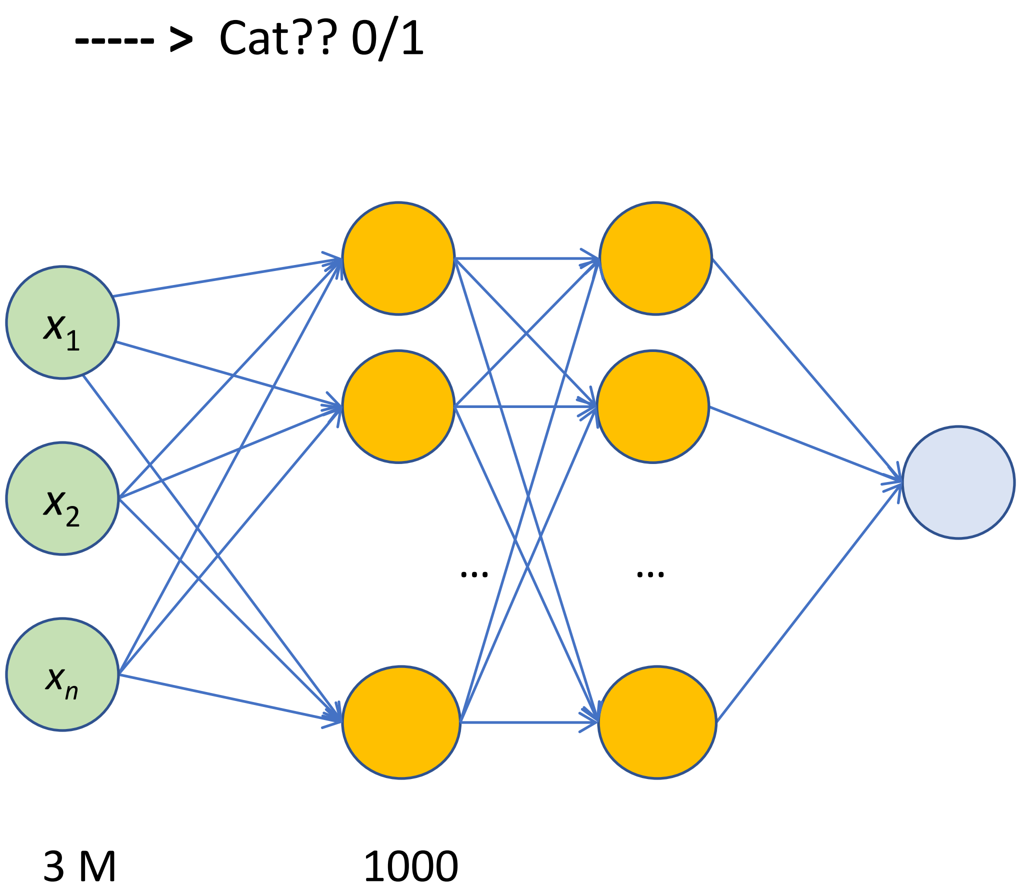


Deep Learning on Large Images

64*64*3 = 12288

1000*1000*3 image???
3 million inputs!!!

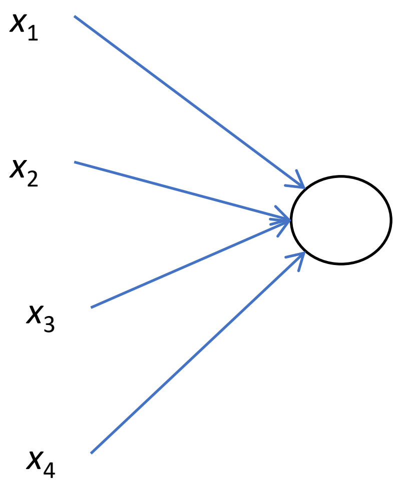
Matrix $\mathbf{W}^{[1]}$ will have
$3*10^6*1000 = 3*10^9$ parameters



Fully Connected Neural Network

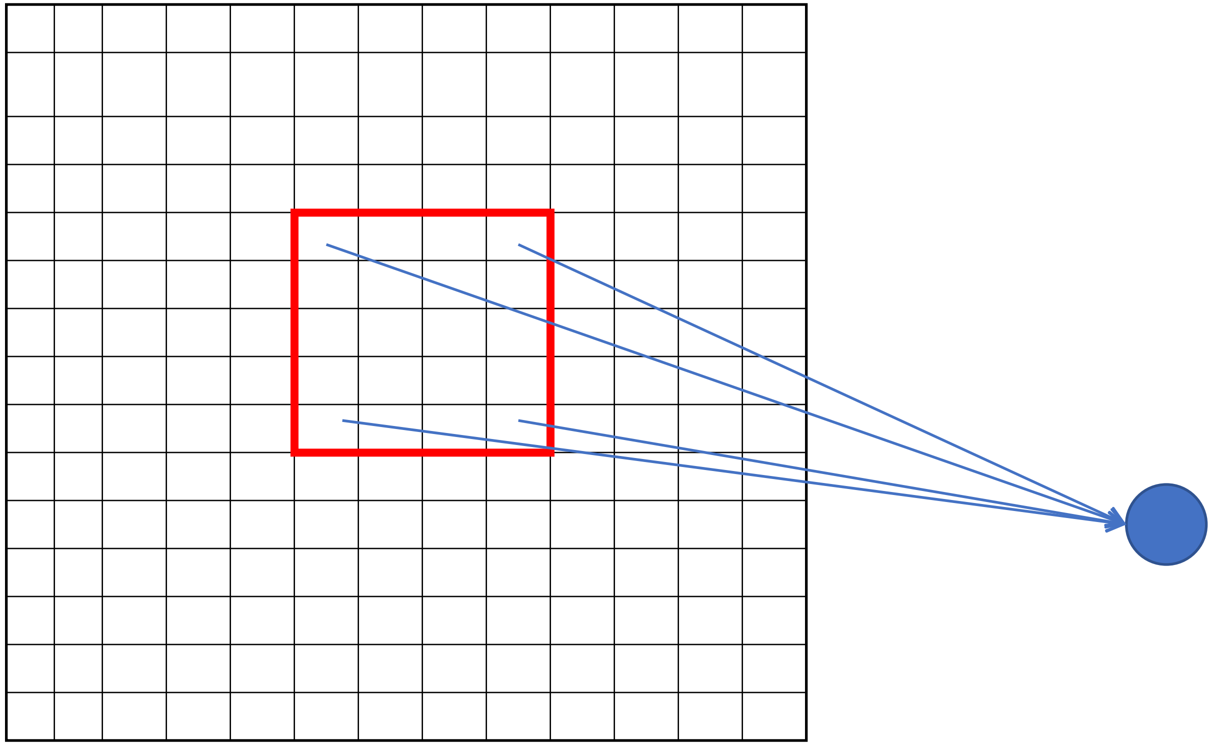
- Input: 2D image
  - Vector of pixel values
  - Flattened 2D image into a 1D vector to pass values to a fully connected layer

- Fully connected layer: Connects each neuron of hidden layer to each input value
  - No spatial information
  - And many parameters



How to pass spatial structure in image to the architecture??

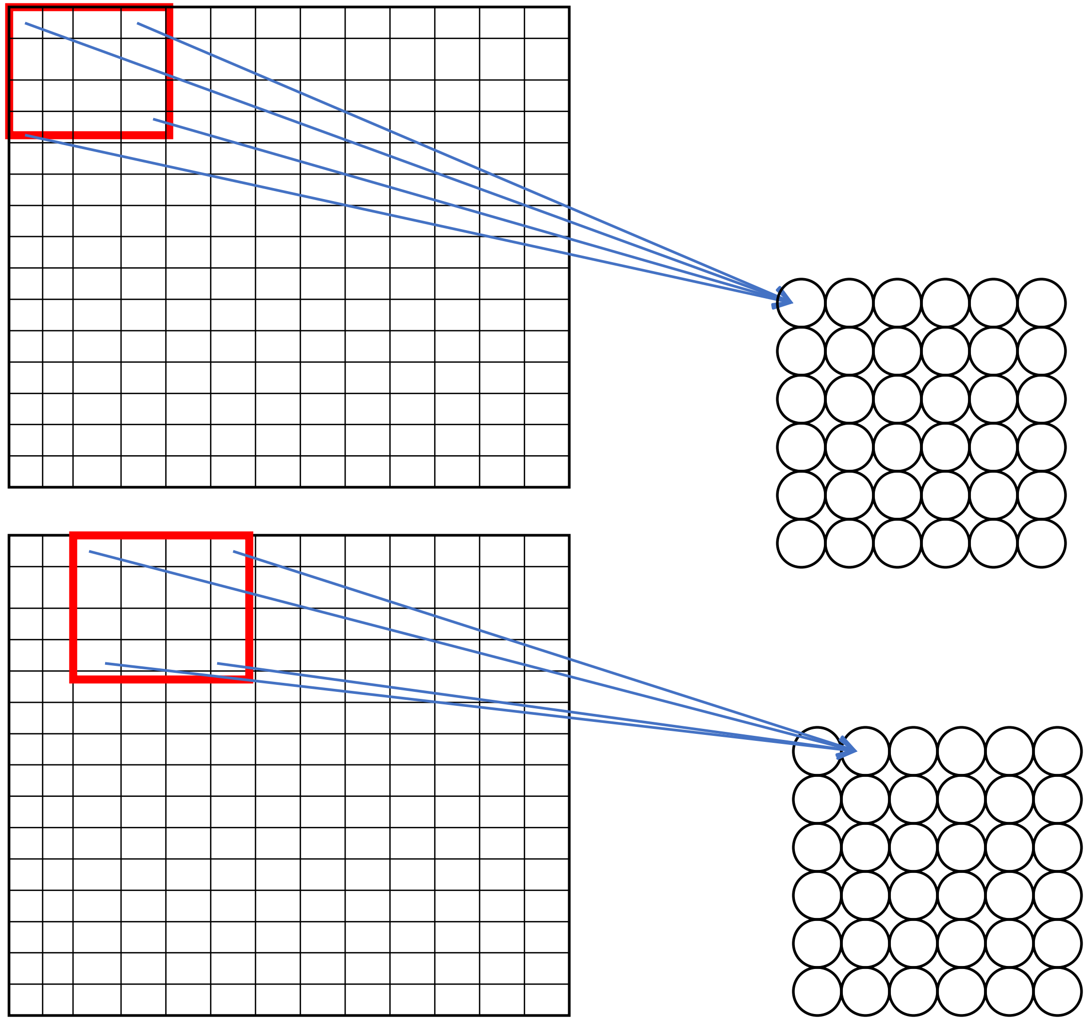
Using Spatial Structure



- Connect patches of input to neuron
- Neuron is connected to regions
- Neuron sees only these values
- Spatially close pixels are likely to be correlated to each other

Input: 2D image
Array of pixel values

Using Spatial Structure

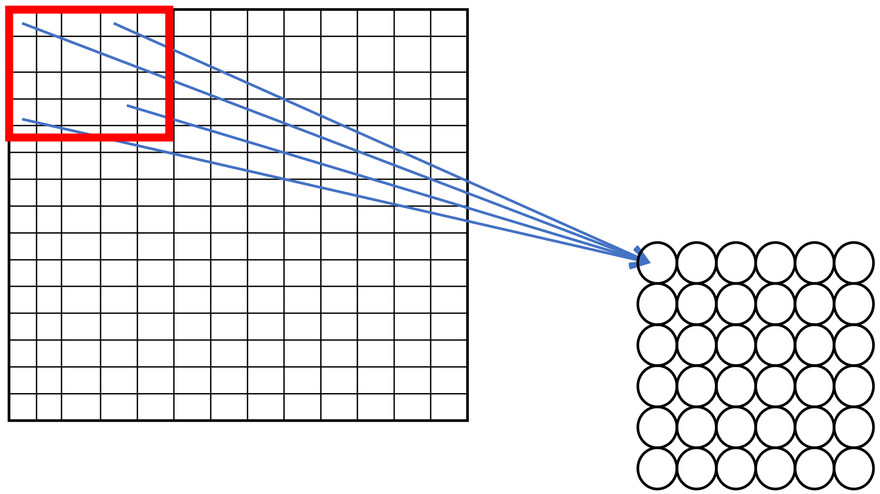


Connect patch in input layer to single neuron in subsequent layer

Use sliding window to define connections

How to **weigh** patch to detect features?

Feature Extraction with Convolution



Filter of size 4*4: 16 different weights

Apply same filter to 4*4 patches in input

Shift by ‘*s*’ pixels for next patch

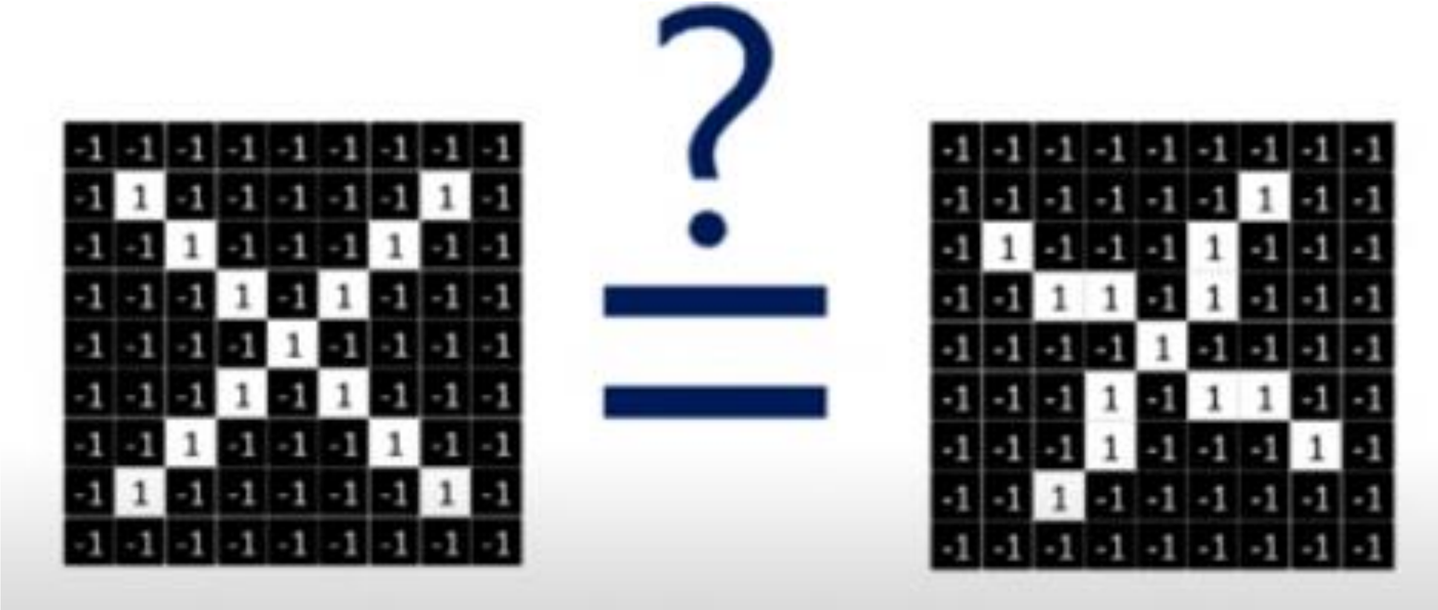
This operation is called **convolution**

1. **Apply set of weights (one filter) to extract local features**

2. **Use multiple filters to extract different features**

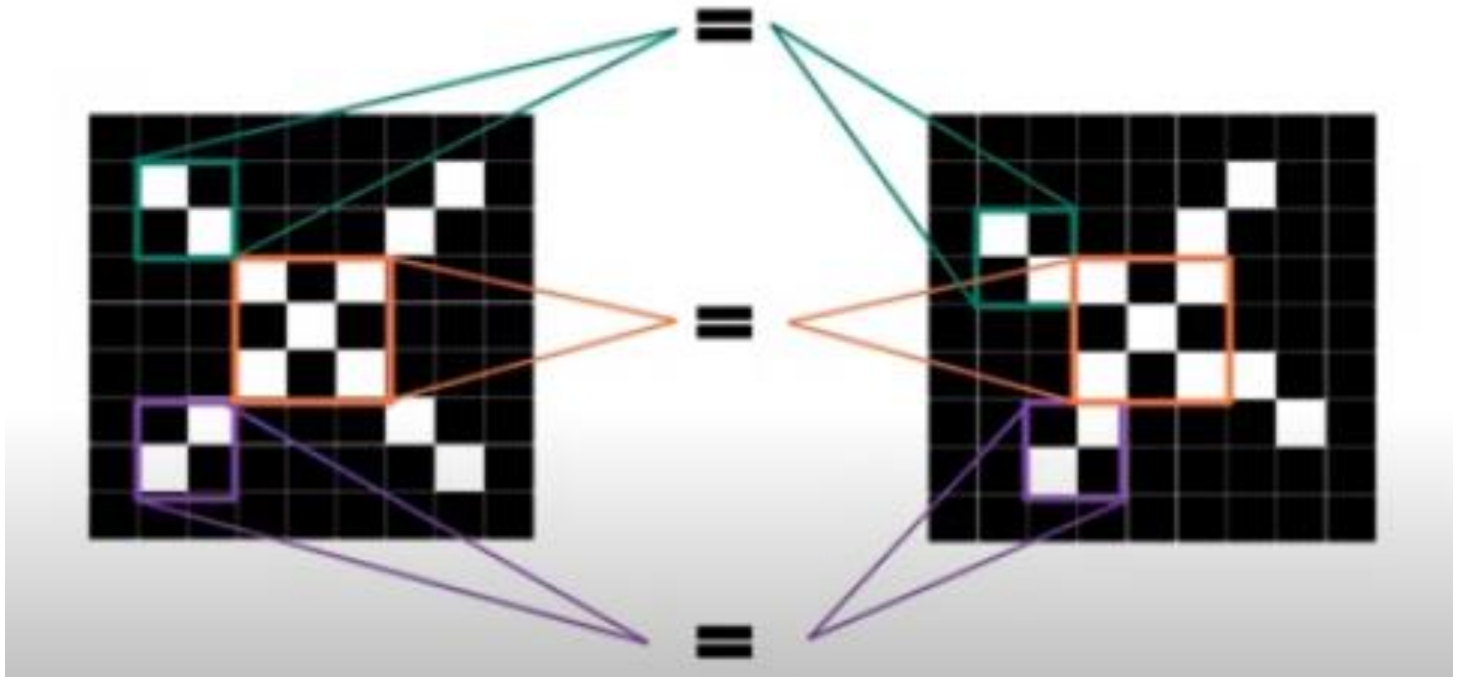
3. **Spatially share parameters of each filter**

Case study: X???



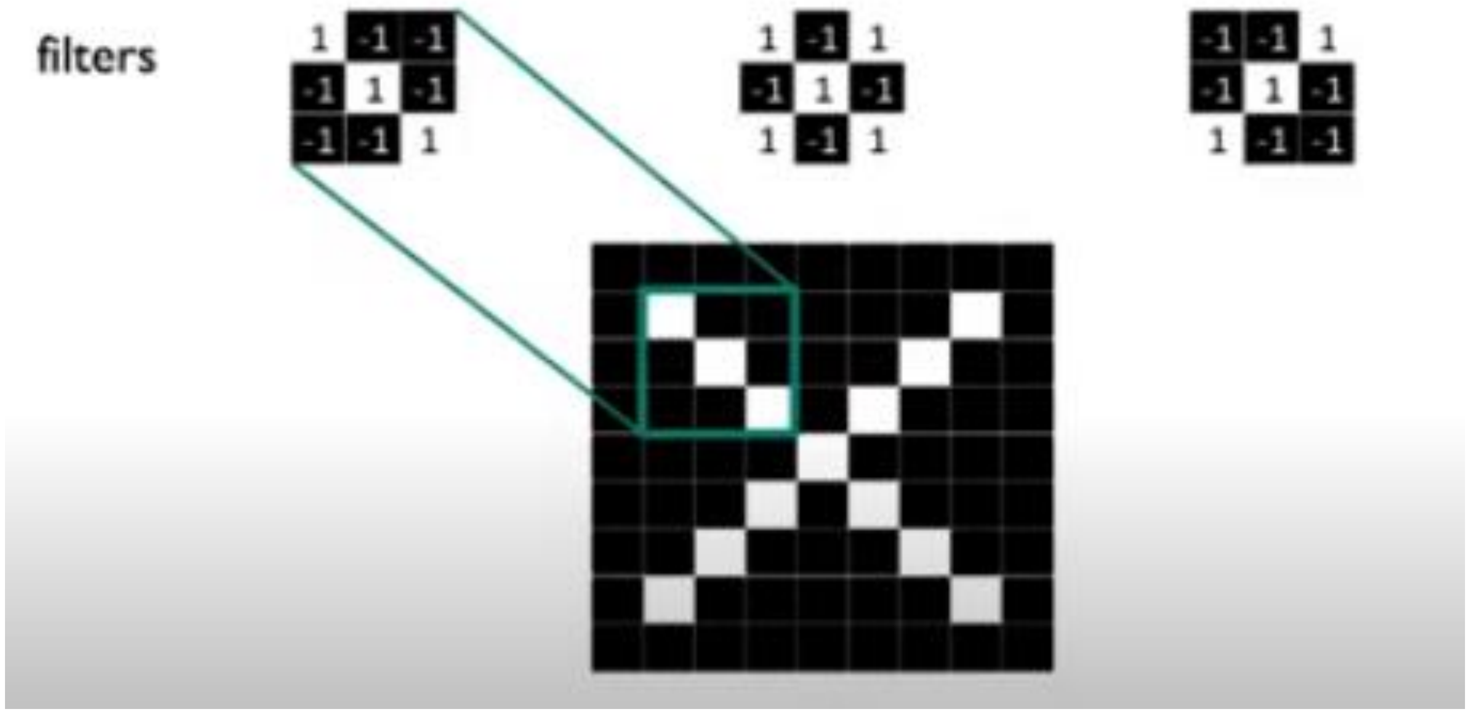
Classify **X** as an **X** even if it is shifted, scaled, rotated or deformed

Features of X

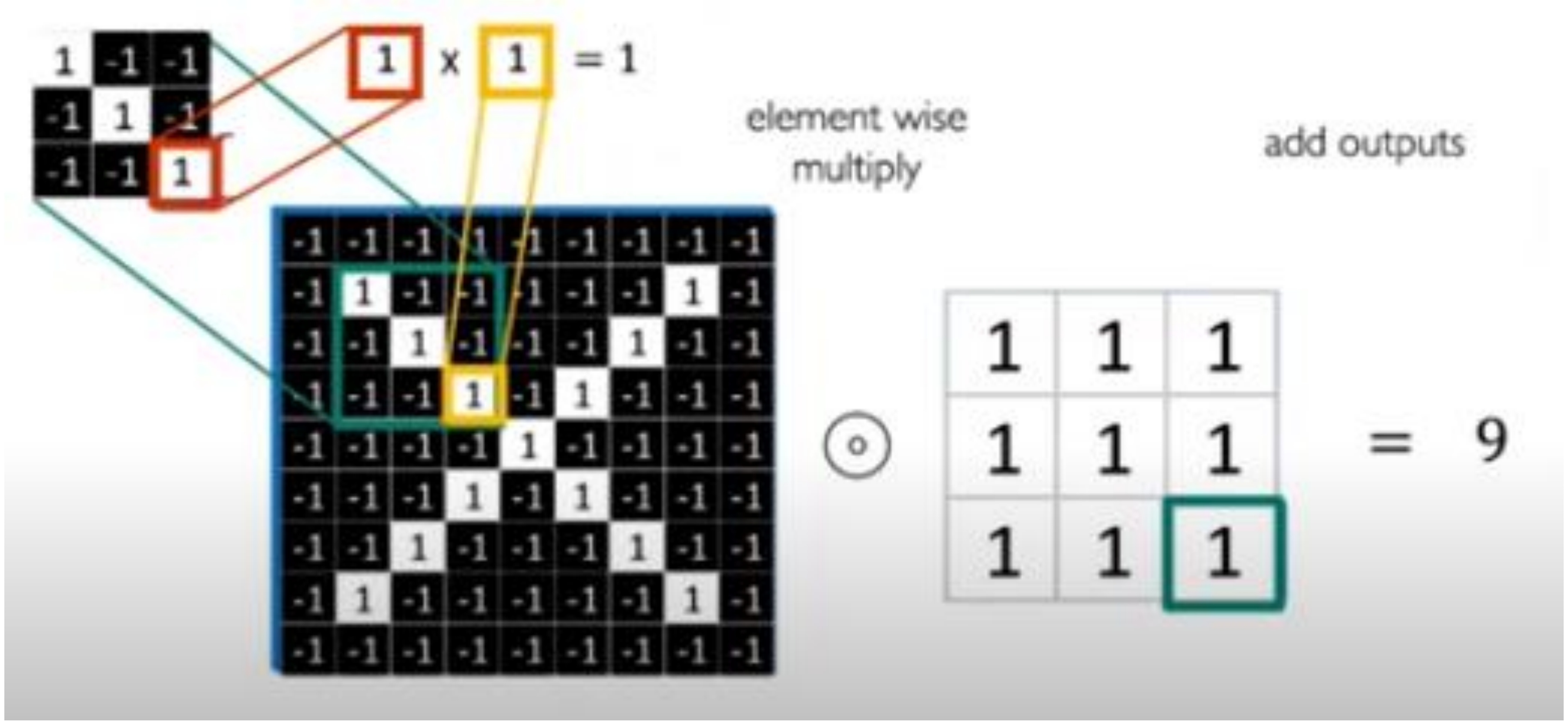


Comparing piece by piece

Filters to detect 'X' features



Convolution Operation



Convolution Operation

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Convolution of 5x5 image with 3x3 filter

Convolution Operation

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

*

1	0	1
0	1	0
1	0	1

=

4		

Feature Map

Convolution Operation

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

*

1	0	1
0	1	0
1	0	1

=

4	3	

Feature Map

Convolution Operation

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 3 & 4 \\ & & \\ & & \end{bmatrix}$$

Feature Map

Convolution Operation

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

*

1	0	1
0	1	0
1	0	1

=

4	3	4
2		

Feature Map

Convolution Operation

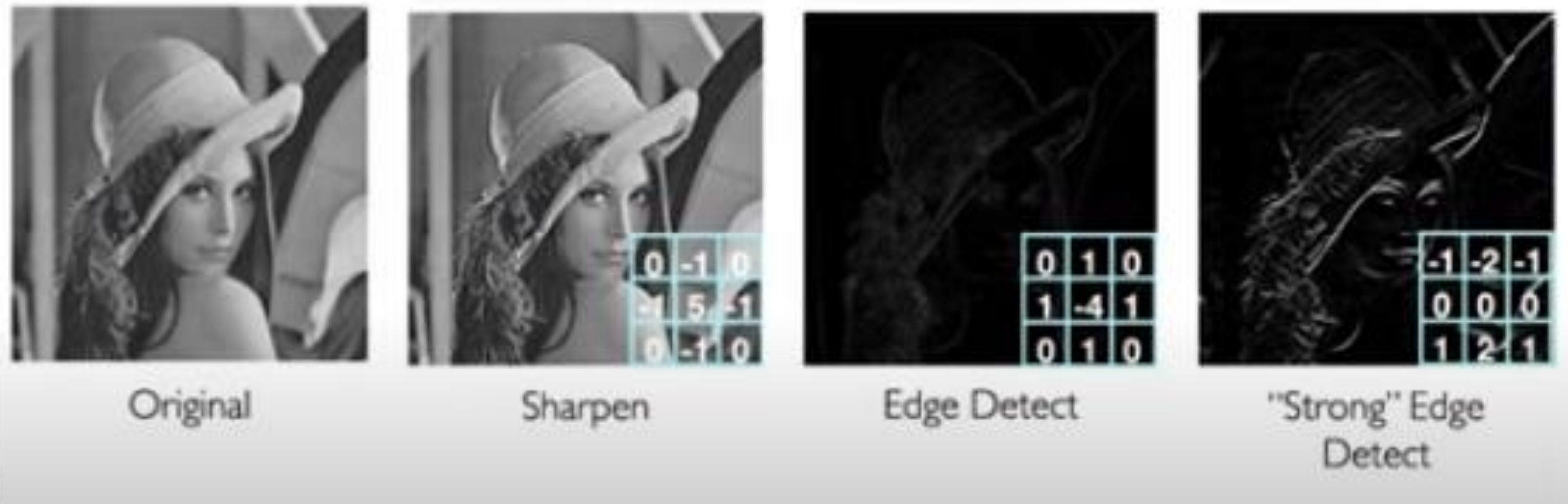
$$
\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}
*
\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}
=
\begin{bmatrix} 4 & 3 & 4 \\ 2 & 4 & 3 \\ 2 & 3 & 4 \end{bmatrix}
$$

Feature Map

Wherever pattern of filter is seen in image, feature map will have highest value

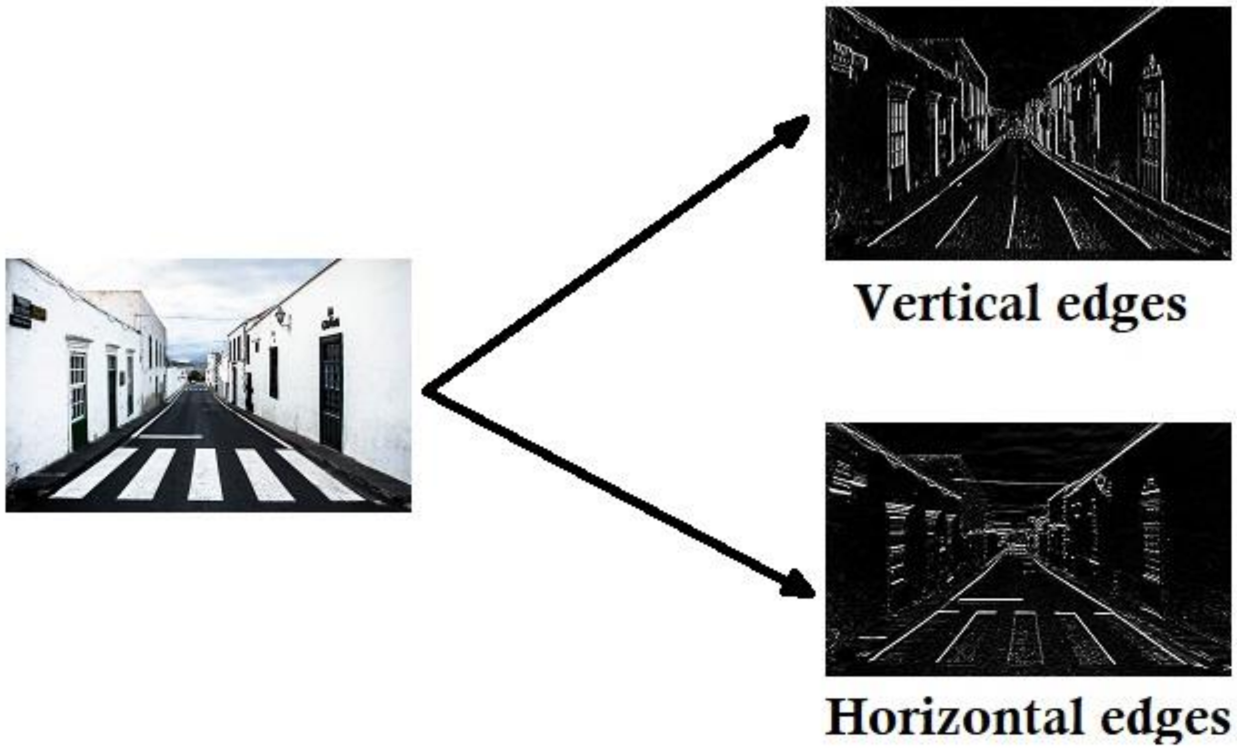
- **Technically, cross-correlation is being performed.**
- **Convolution actually involves horizontal and vertical flipping of the filter.**
- **Flipping allows for associativity property in some signal processing applications. Not really required in neural networks.**
- **So the term ‘convolution’ is used here by convention**

Feature Maps



The network learns the weights of the filters to detect various features

Computer Vision



https://datahacker.rs/edge-detection/

Vertical Edge Detection

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
3 & 0 & 1 & 2 & 7 & 4 \\ \hline
1 & 5 & 8 & 9 & 3 & 1 \\ \hline
2 & 7 & 2 & 5 & 1 & 3 \\ \hline
0 & 1 & 3 & 1 & 7 & 8 \\ \hline
4 & 2 & 1 & 6 & 2 & 8 \\ \hline
2 & 4 & 5 & 2 & 3 & 9 \\ \hline
\end{array}
*
\begin{array}{|c|c|c|}
\hline
1 & 0 & -1 \\ \hline
1 & 0 & -1 \\ \hline
1 & 0 & -1 \\ \hline
\end{array}
=
\begin{array}{|c|c|c|c|}
\hline
-5 & -4 & 0 & 8 \\ \hline
-10 & -2 & 2 & 3 \\ \hline
0 & -2 & -4 & -7 \\ \hline
-3 & -2 & -3 & -16 \\ \hline
\end{array}
$$

6*6 image

3*3 filter

Light to Dark Edge

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6x6 image

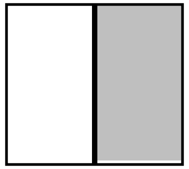
*

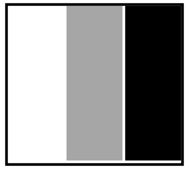
1	0	-1
1	0	-1
1	0	-1

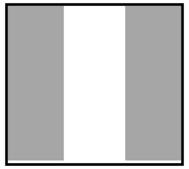
3x3 filter

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0







Dark to Light Edge

0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

1	0	-1
1	0	-1
1	0	-1

=

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0

Vertical and Horizontal Edges

1	0	-1
1	0	-1
1	0	-1

1	1	1
0	0	0
-1	-1	-1

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

1	1	1
0	0	0
-1	-1	-1

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Edge Detection Filters

1	0	-1
1	0	-1
1	0	-1

1	0	-1
2	0	-2
1	0	-1

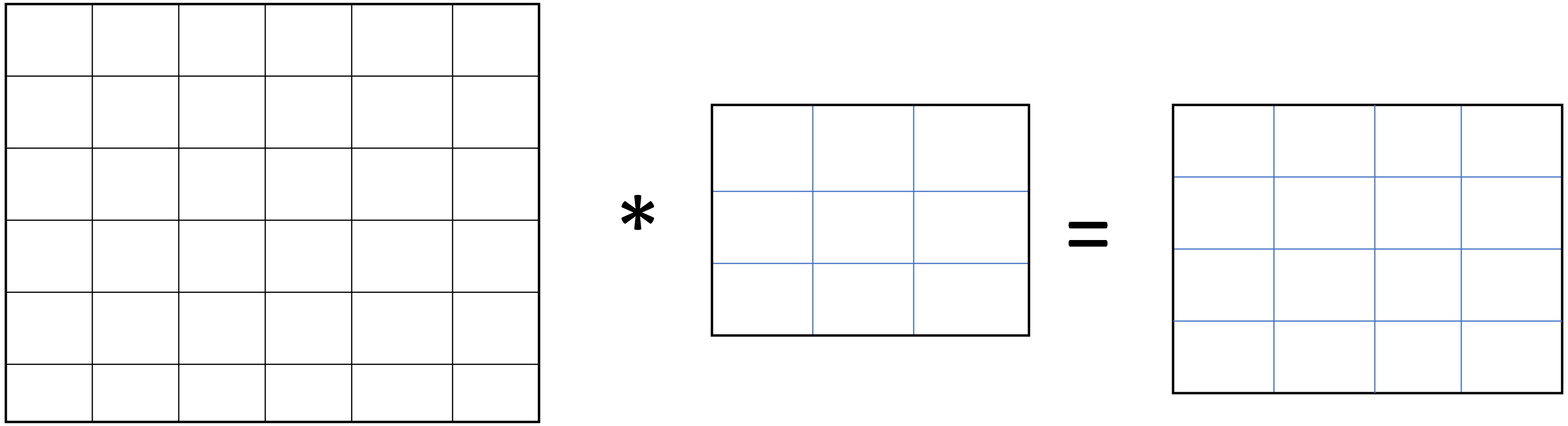
Sobel filter

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Learn the parameters

Filters can be of various types to detect edges at various angles

Output Feature Map



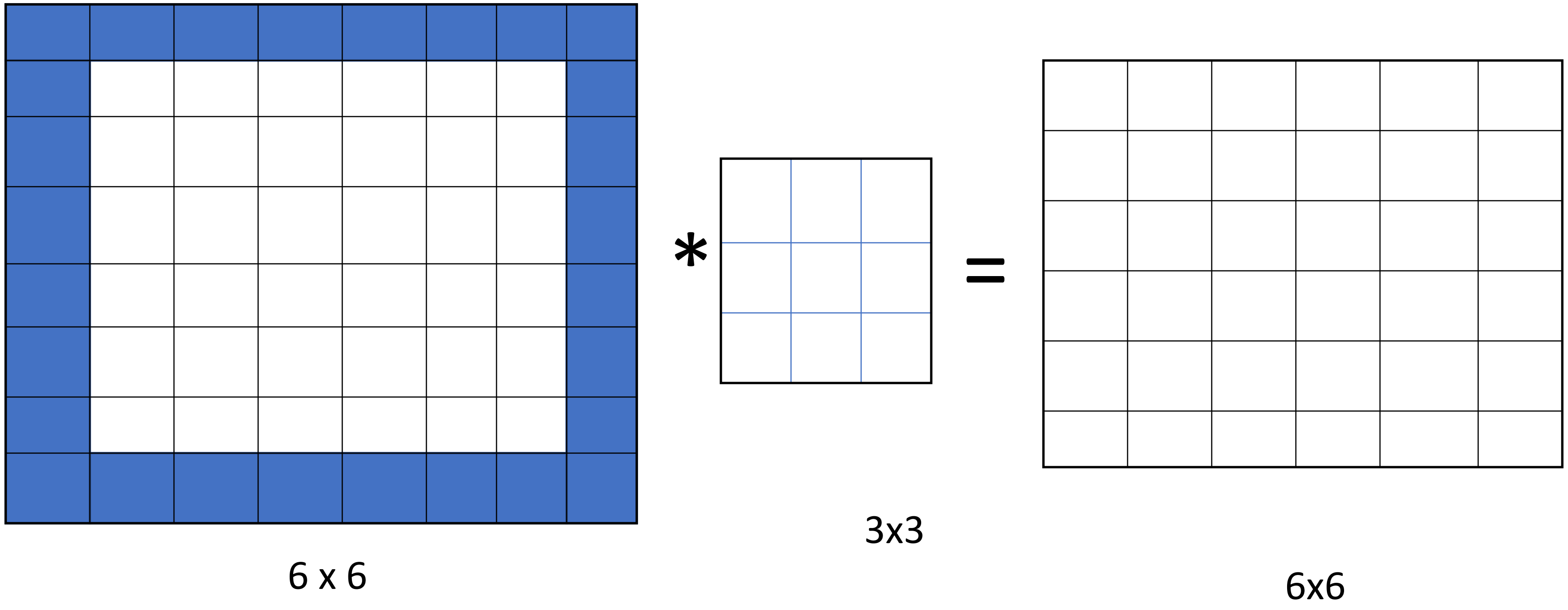
6 * 6   3 * 3   4 * 4

n * *n*   *f* * *f*

Output feature map size: (*n*-*f*+1)* (*n*-*f*+1)

- Output shrinks
- Corner/edge pixels contribute less to output
  - -Loss of information

Padding



Padding is usually with zeros
Let *p* be the padding (here, 1 pixel all around)

Output size = $(n+2p-f+1) * (n+2p-f+1)$

Valid and Same Convolutions

- 'Valid' convolution: No padding
- 'Same' convolution: Padding done so that output size is same as input size

$$n+2p-f+1 = n$$

$$\rightarrow \quad p = (f-1)/2$$

- By convention, f is usually odd
  - Filter has a central pixel

Strided convolution

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

*

3	4	4
1	0	2
-1	0	3

=

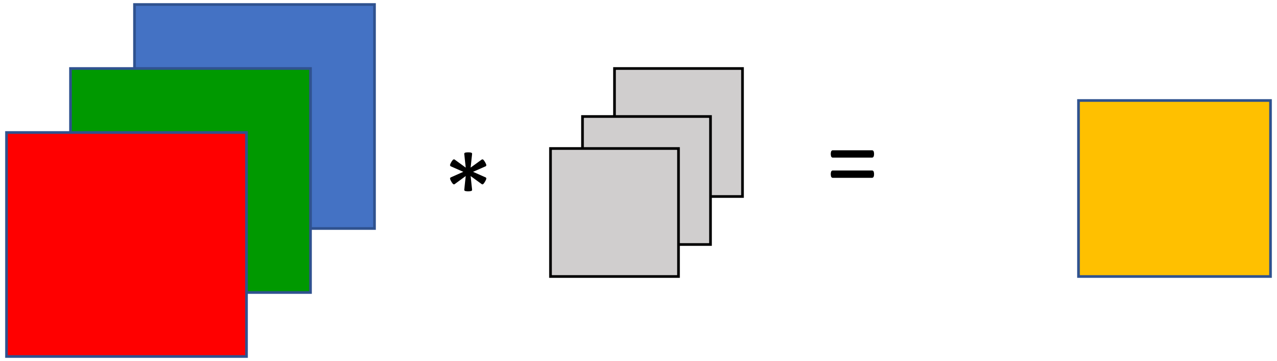
91	100	83
69	91	127
44	72	74

Stride (s): Step size of sliding window

Output size: $[((n+2p-f)/s)+1]*[((n+2p-f)/s)+1]$

- If not an integer, take floor value
- Filter must lie entirely within the image

Convolutions on RGB images



6 * 6 * 3

Height * width * no. of channels

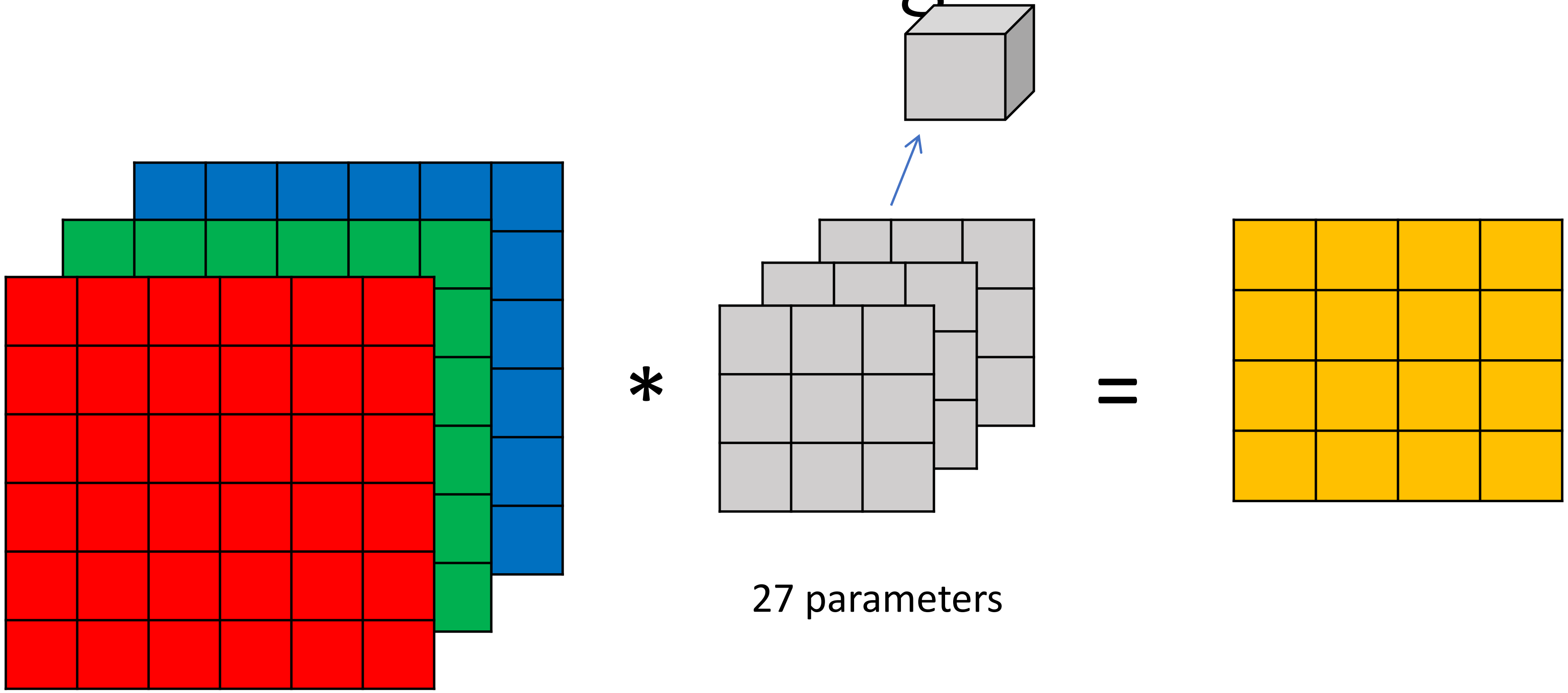
3 * 3 * 3

Height * width * no. of channels

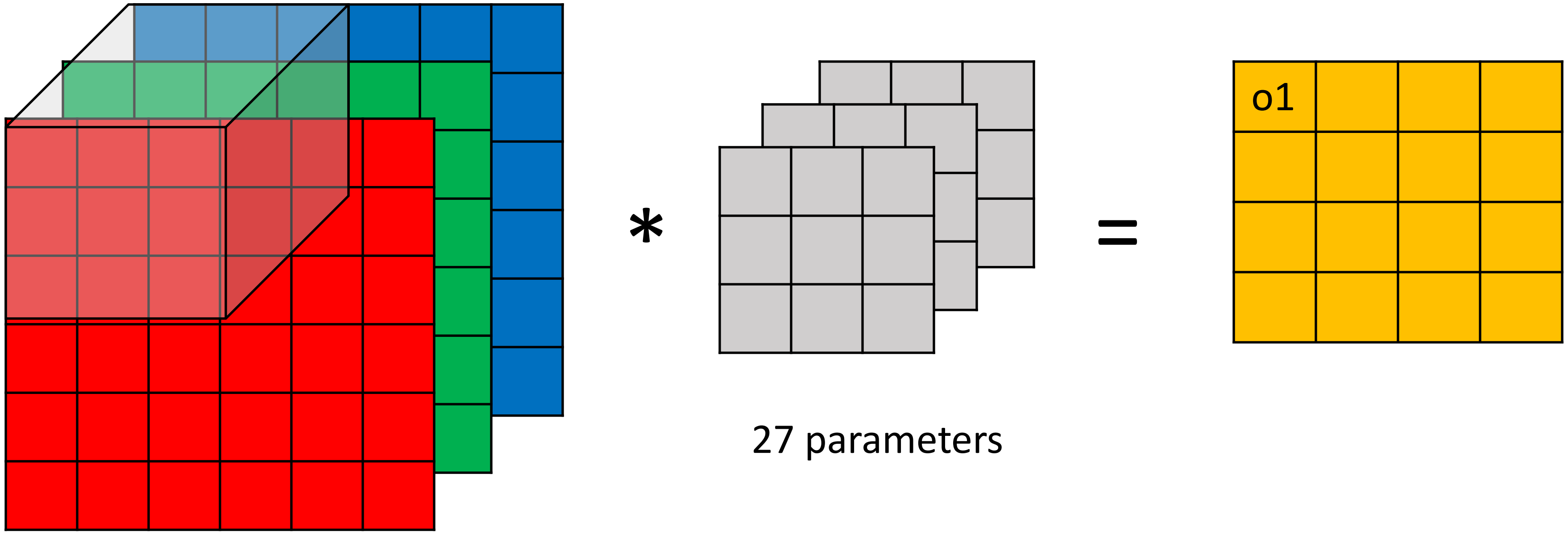
4 * 4 * 1

No. of channels in input = No. of channels in filter

Convolutions on RGB images

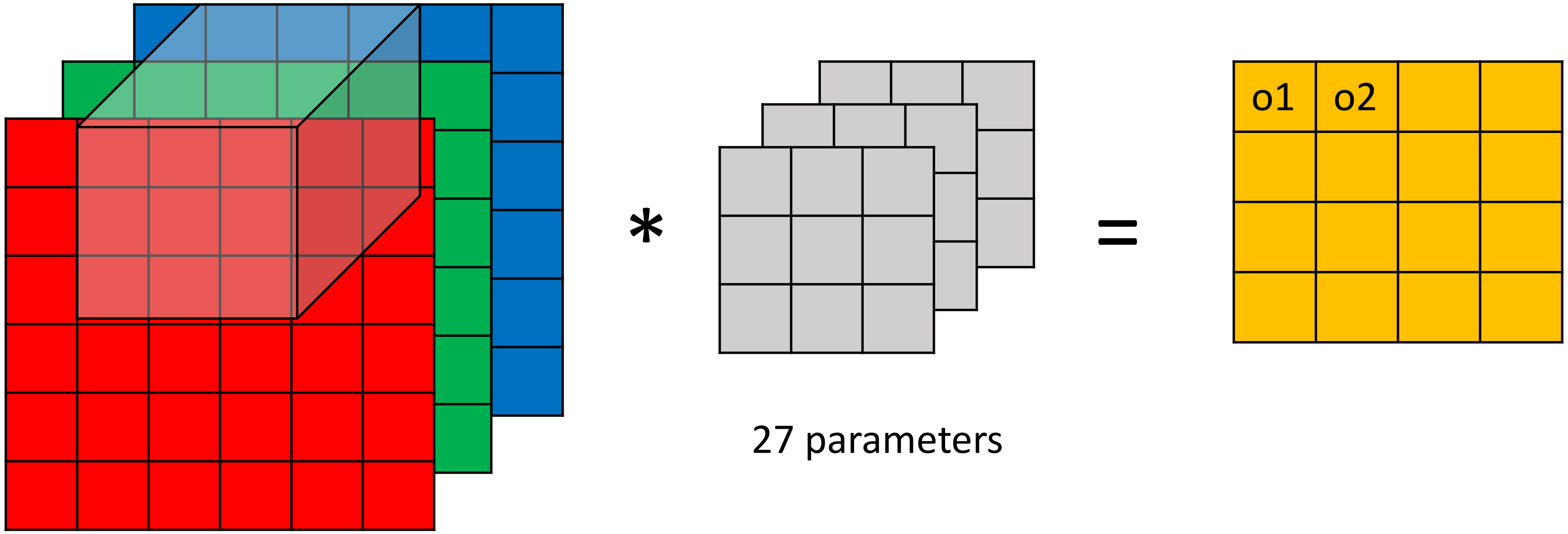


Convolutions on RGB images

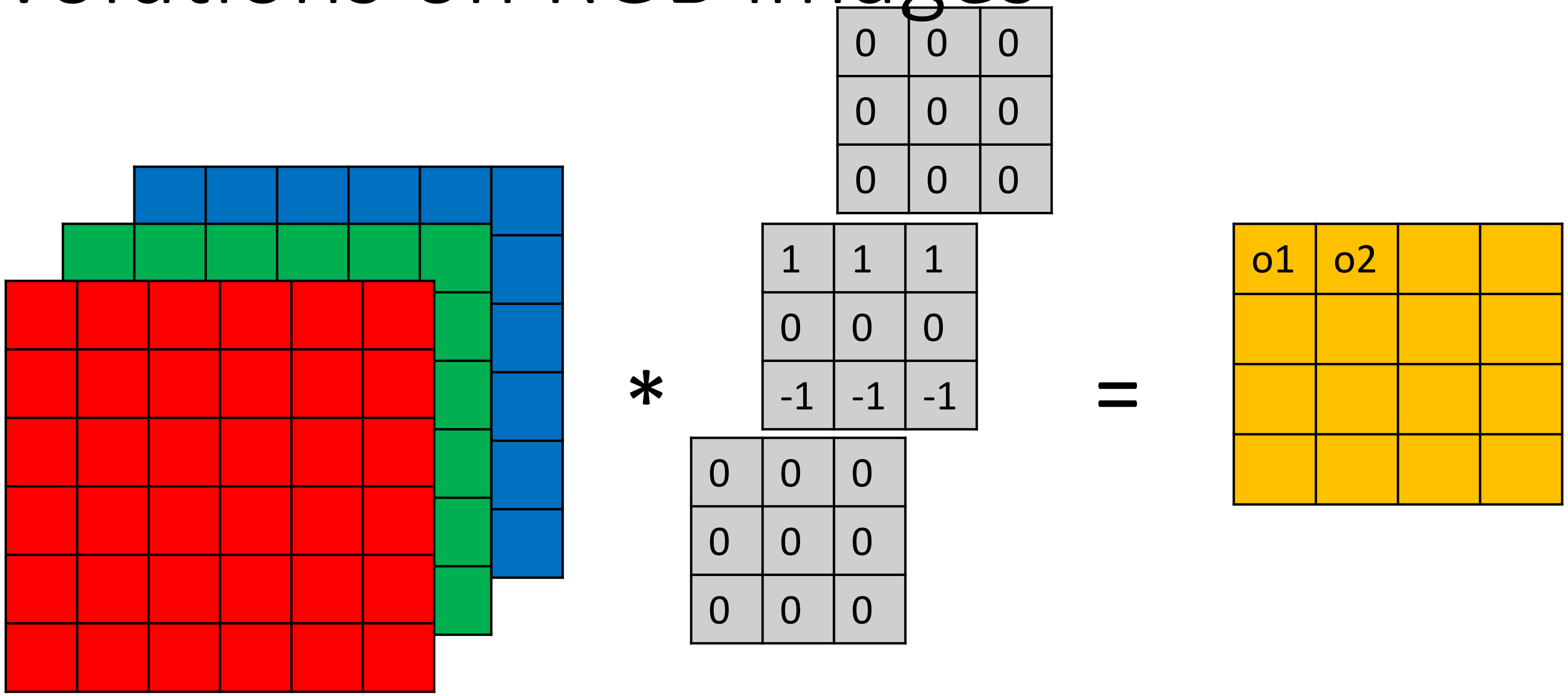


27 parameters

Convolutions on RGB images

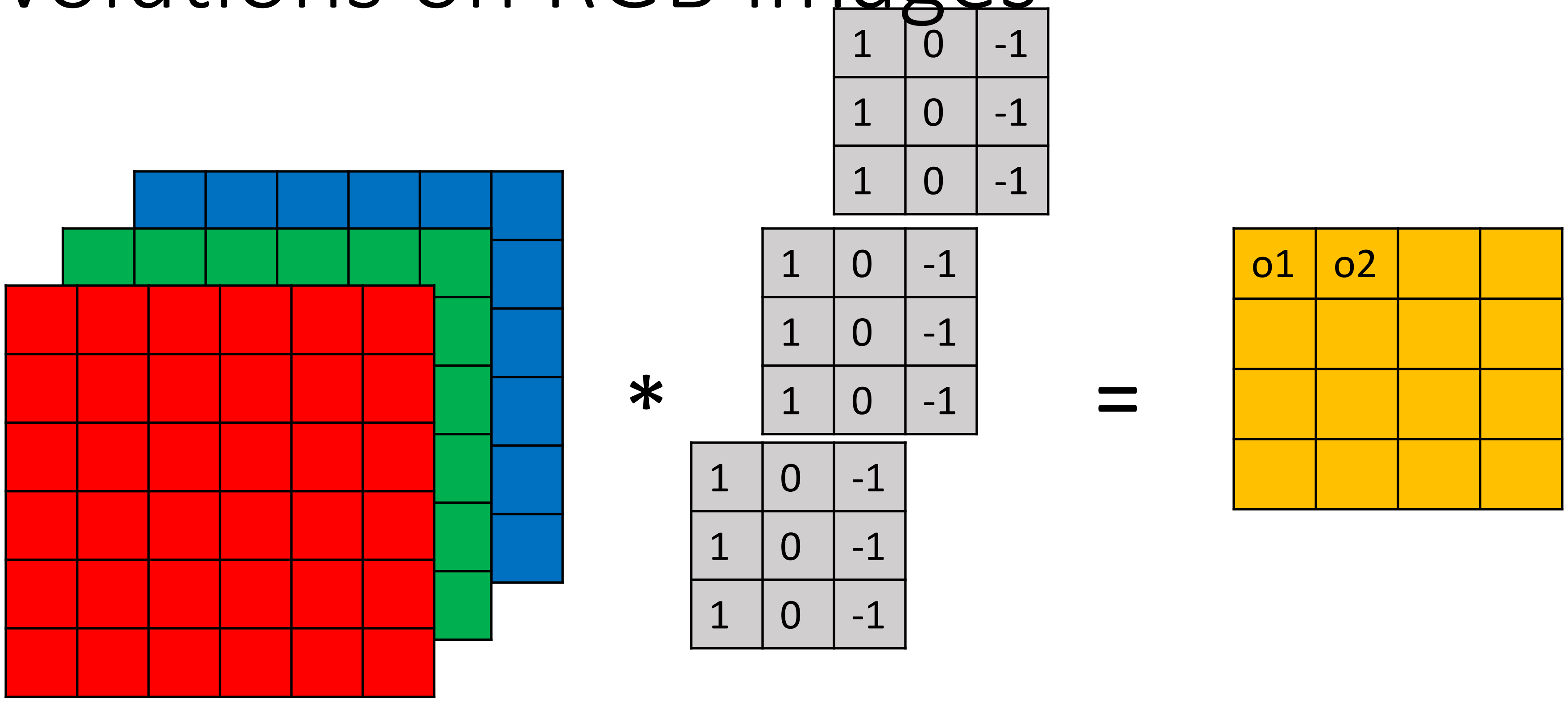


Convolutions on RGB images



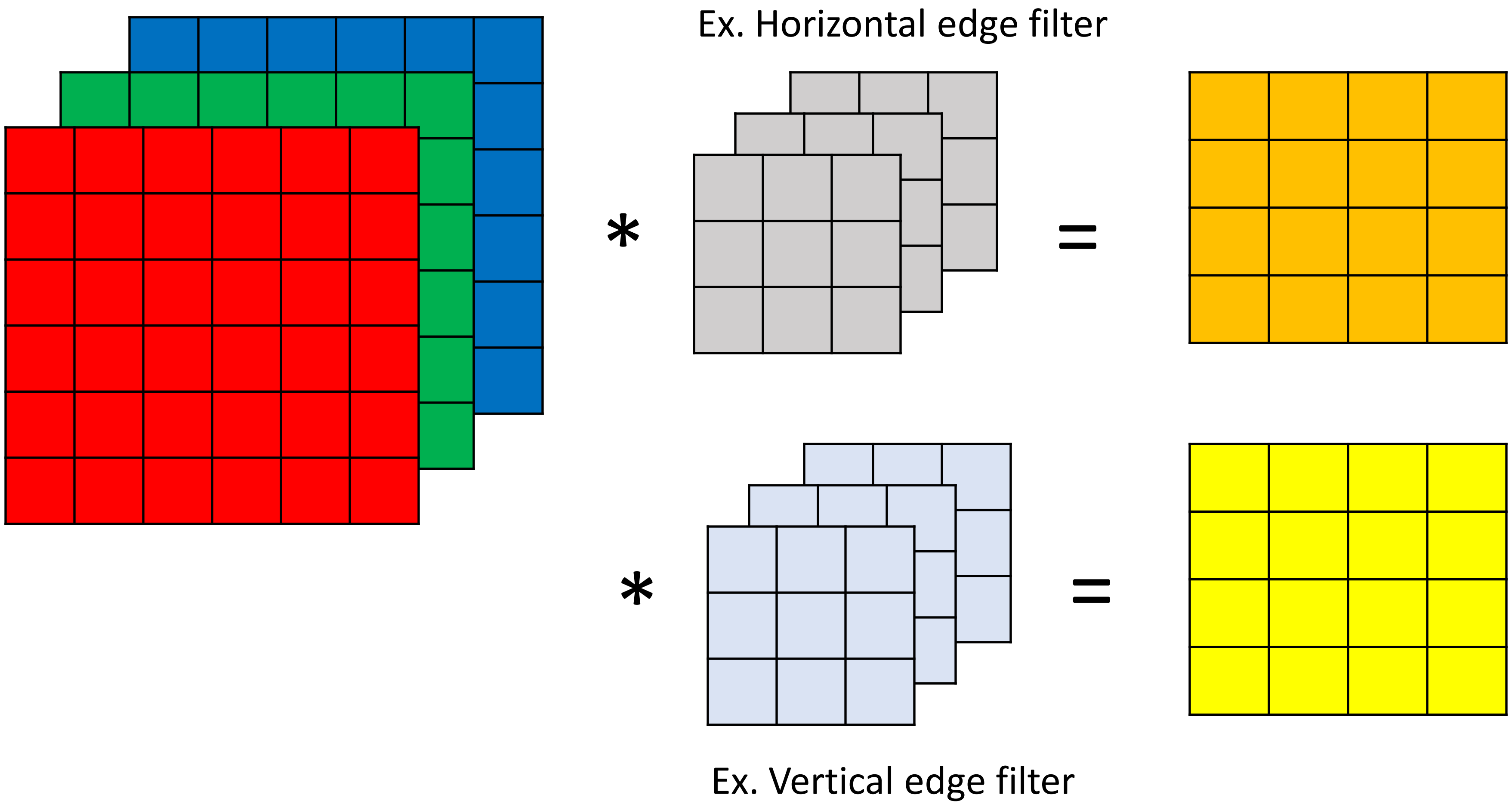
Example of filter if horizontal edges in Green channel are to be detected

Convolutions on RGB images

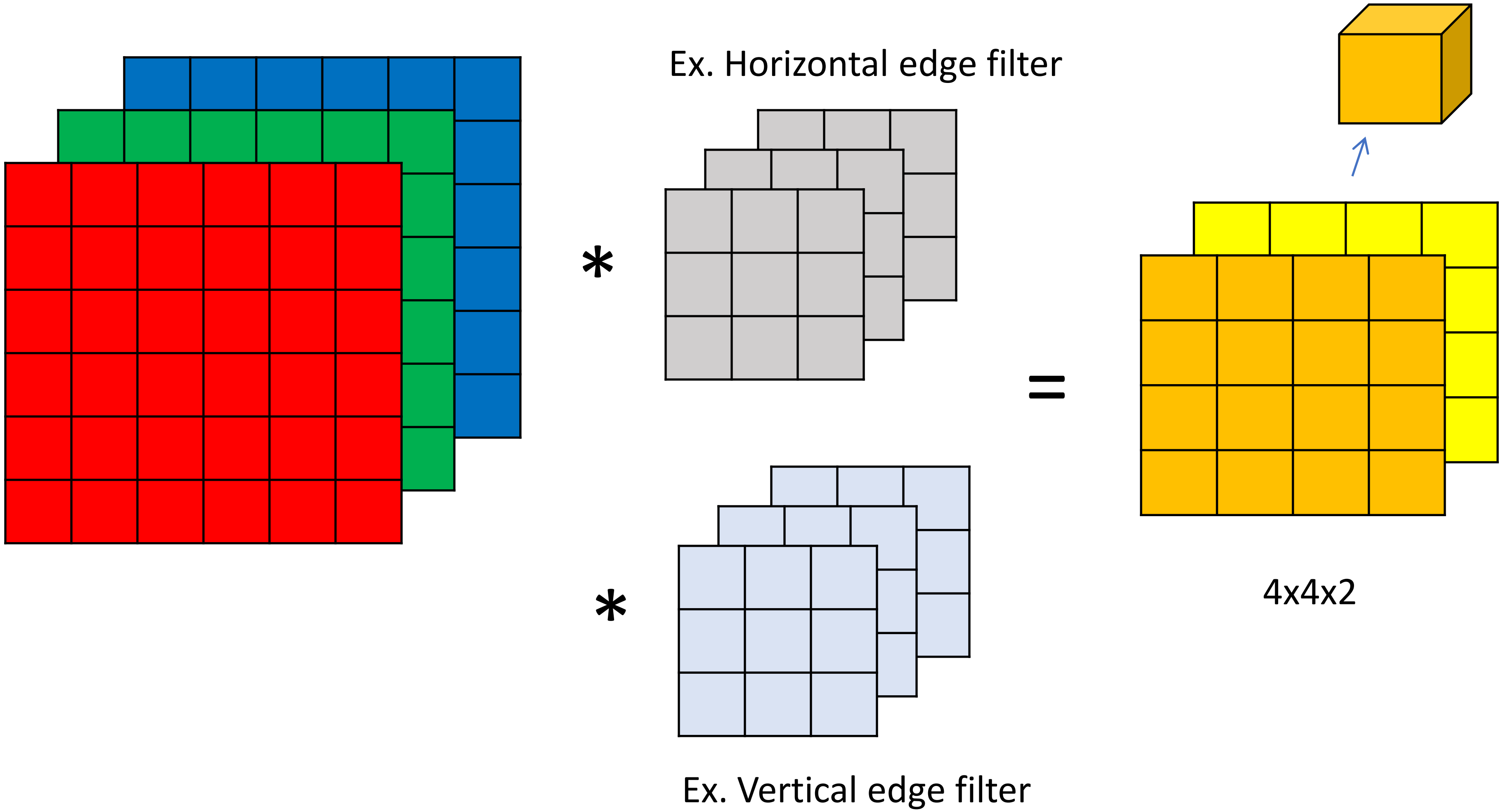


Example of filter if vertical edges in all channels are to be detected

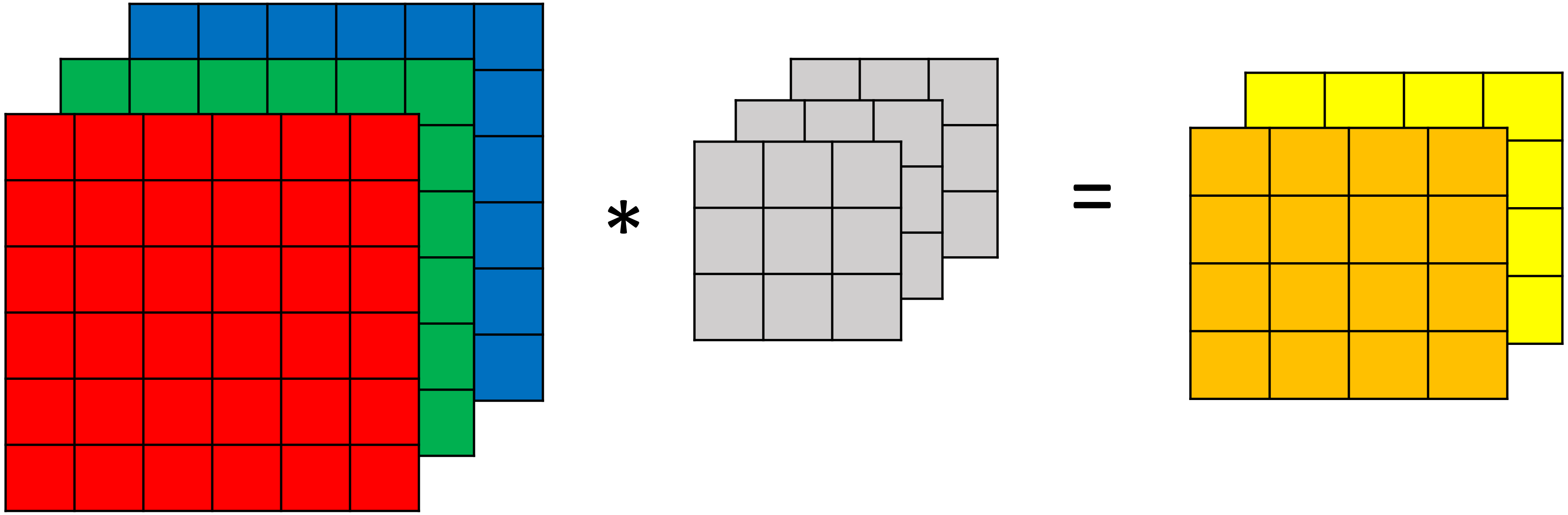
Multiple filters



Multiple filters



Multiple filters

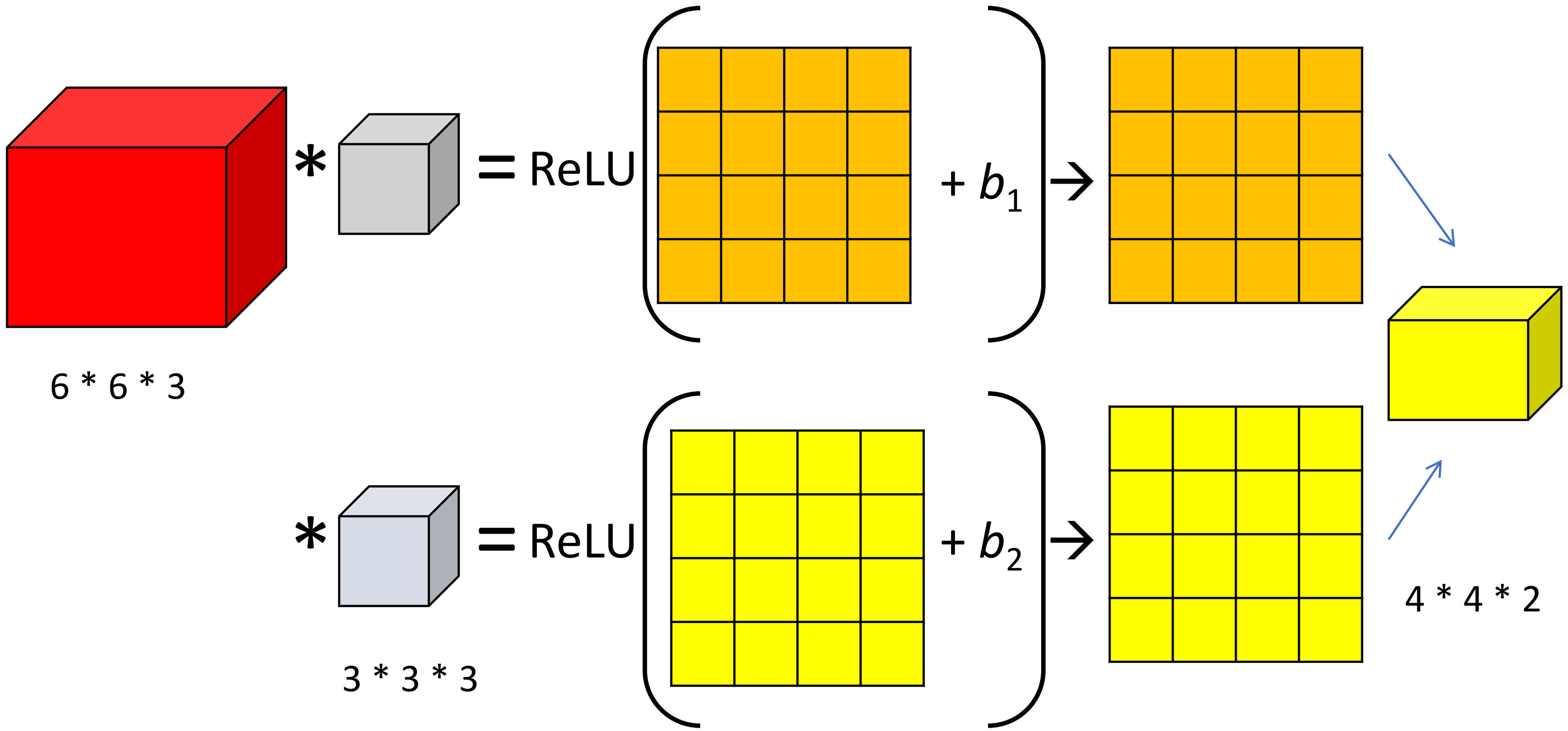


$n \times n \times n_c$ $\quad$ $f \times f \times n_c$

Output size: $((n - f)/s + 1) * ((n - f)/s + 1) * n_f$

n_f: no. of filters

One Layer of Convolution Network



$z = w\,x + b$

$a = g(z)$

No. of parameters in a layer

- Ex. If input size is 28*28*3 with zero padding, what will be output feature size (assume stride = 1) assuming 10 filters of size 5*5*3?

- How many parameters does that layer have?

  Each filter: 5*5*3 = 75 + 1 = 76 parameters

  For 10 filters = 76*10 = 760 parameters

Notation

- For layer *l* (convolution layer):

    $f^{[l]}$: filter size

    $p^{[l]}$: padding

    $s^{[l]}$: stride

    $n_c^{[l]}$: no. of filters

    Hyperparameters

    Each filter: $f^{[l]} * f^{[l]} * n_c^{[l-1]}$

    Weights: $f^{[l]} * f^{[l]} * n_c^{[l-1]} * n_c^{[l]}$

    Bias: $n_c^{[l]}$

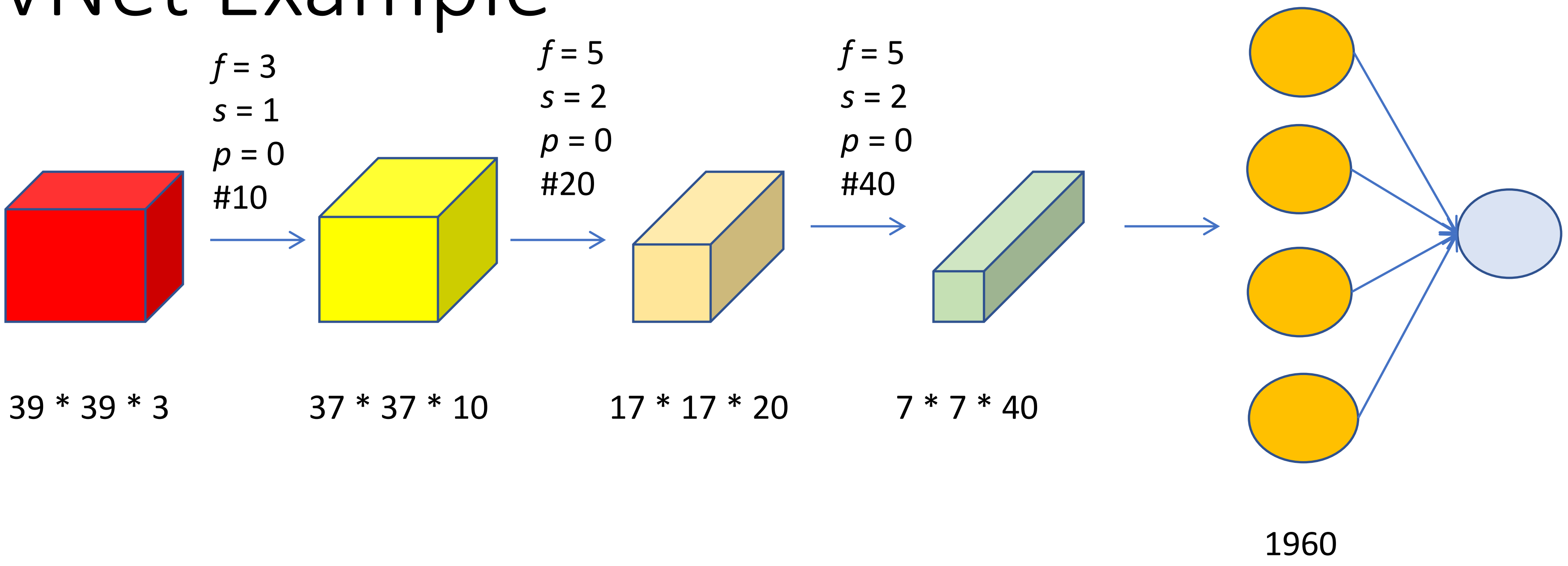
Input: $n_h^{[l-1]} * n_w^{[l-1]} * n_c^{[l-1]}$

Output: $n_h^{[l]} * n_w^{[l]} * n_c^{[l]}$

$$n_h^{[l]} = (n_h^{[l-1]} + 2p^{[l]} - f^{[l]})/s^{[l]} + 1$$

$$n_w^{[l]} = (n_w^{[l-1]} + 2p^{[l]} - f^{[l]})/s^{[l]} + 1$$

ConvNet Example



n_h, n_w decrease
n_c increases

Pooling

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

Max pooling
$f = 2$
$s = 2$

$\longrightarrow$

6	8
3	4

- Summarizes characteristics in an area of feature map produced by convolution layer
- Reduce size of representation to speed up computation
- Makes detected features more robust
- No parameters to learn!!

Max Pooling

Intuition for max pool: The large number probably represents some feature

Max pooling
$f = 3$
$s = 1$

1	3	2	2	5
5	6	7	8	1
3	2	1	0	7
1	2	3	4	9
1	1	2	9	4

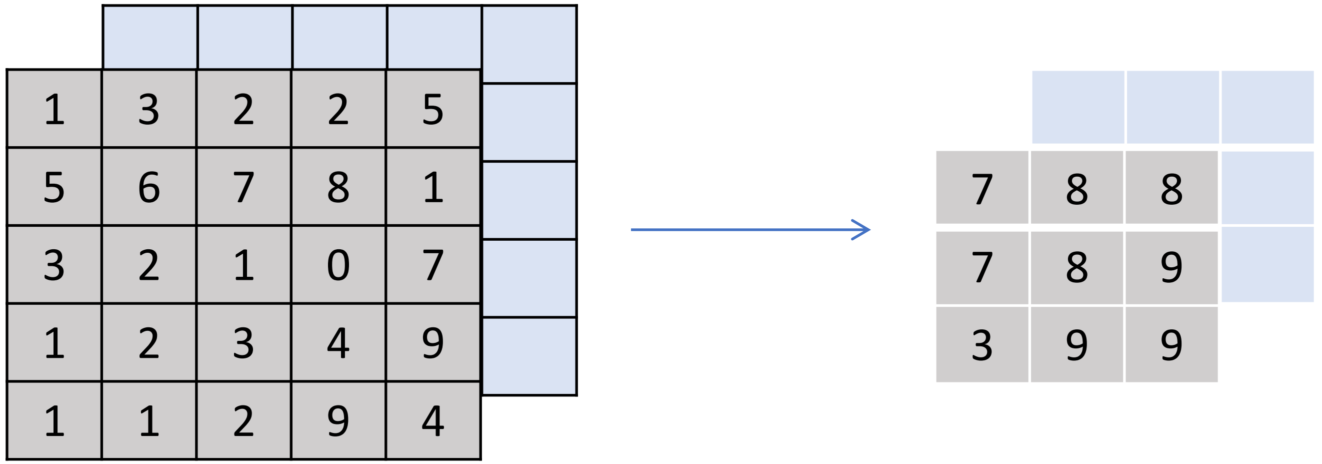
7		

7	8	8
7	8	9
3	9	9

1	3	2	2	5
5	6	7	8	1
3	2	1	0	7
1	2	3	4	9
1	1	2	9	4

7	8	

Max Pooling



5 x 5 x 2

3 x 3 x 2

Each channel is handled independently

Average Pooling

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

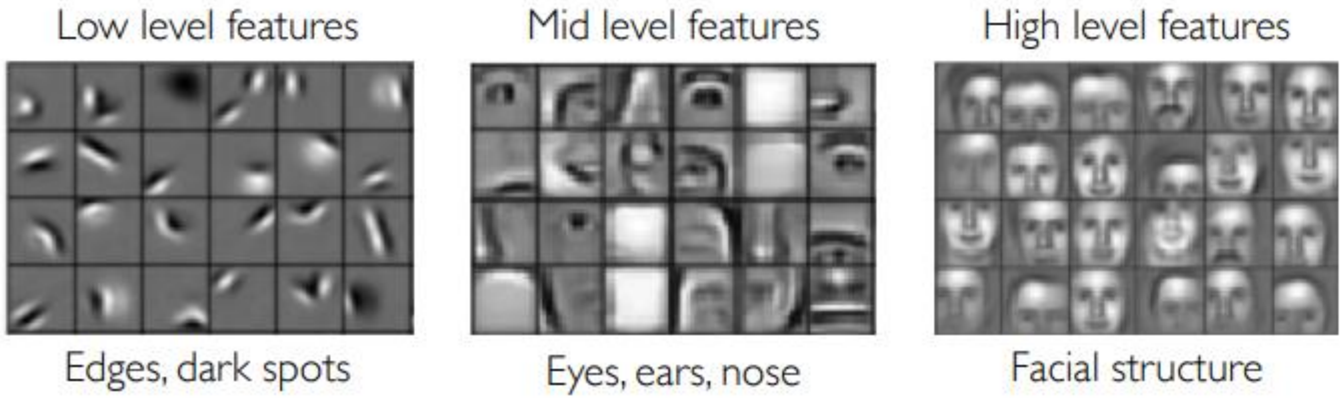
3.25	5.25
2	42

Pooling Output

- Assuming a *f*-dimensional pooling filter over each channel of feature map with stride *s*:
  - For a feature map having dimensions $n_h*n_w*n_c$, the dimensions of output obtained after a pooling layer is $((n_h - f)/s + 1) * ((n_w - f)/s + 1) * n_c$

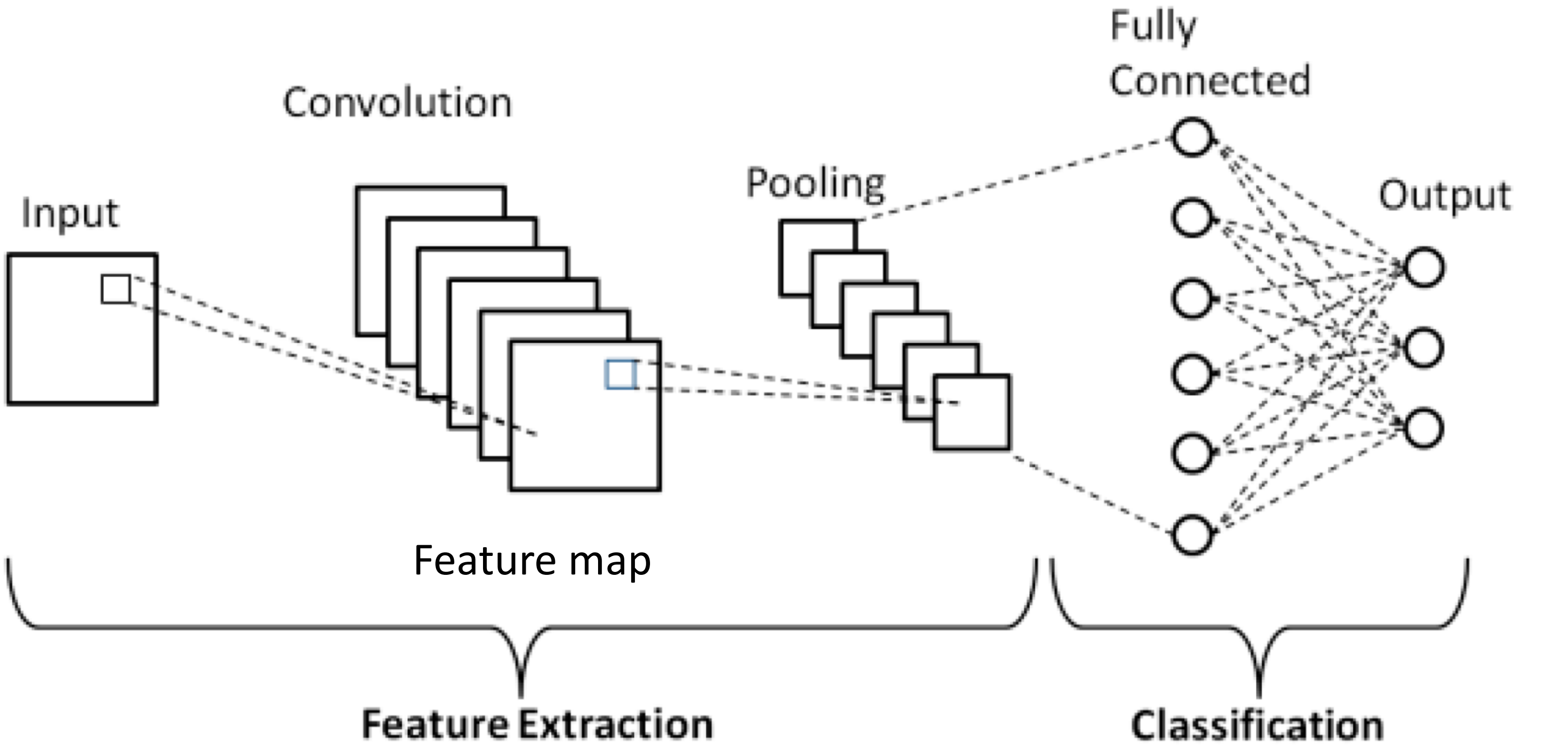
- Example: Input feature map of 4*4*3, a max pool filter of 2*2, with stride 2*2, what is size of output?

Representation Learning in Deep CNNs



Conv Layer 1 Conv Layer 2 Conv Layer 3

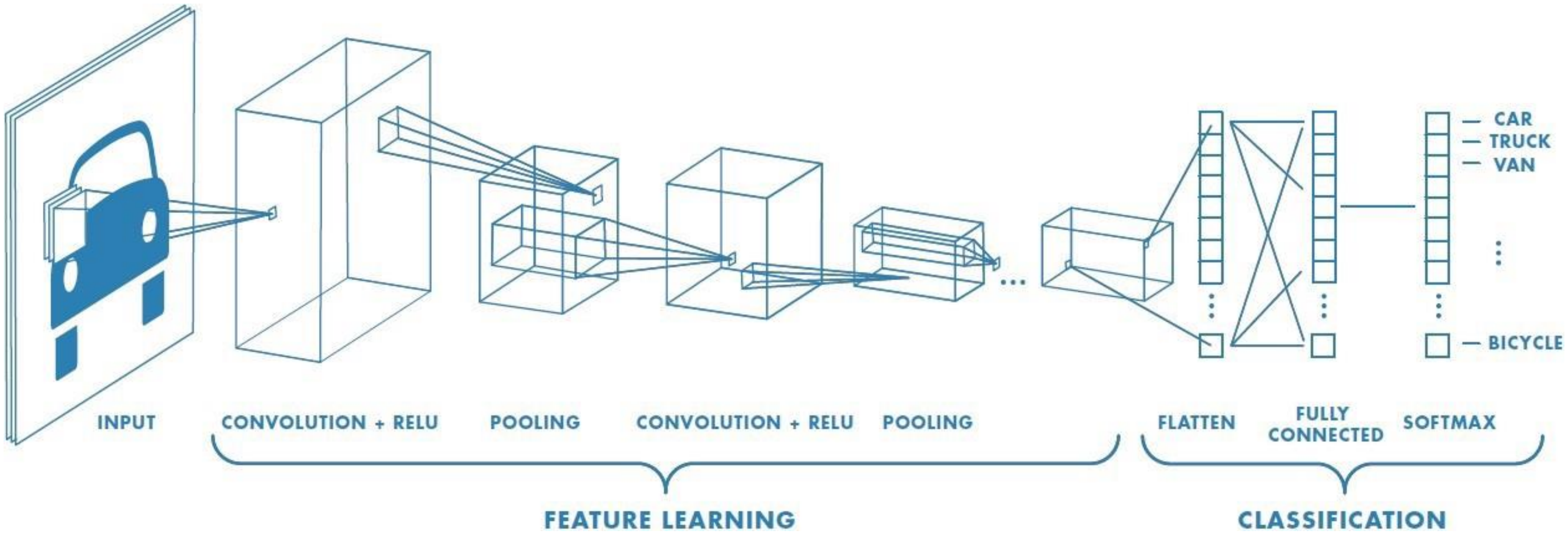
Convolutional Neural Network



1. Convolution – Apply filters to generate feature maps
2. Apply non-linearity - Often ReLU
3. Pooling – Downscaling operation on each feature map

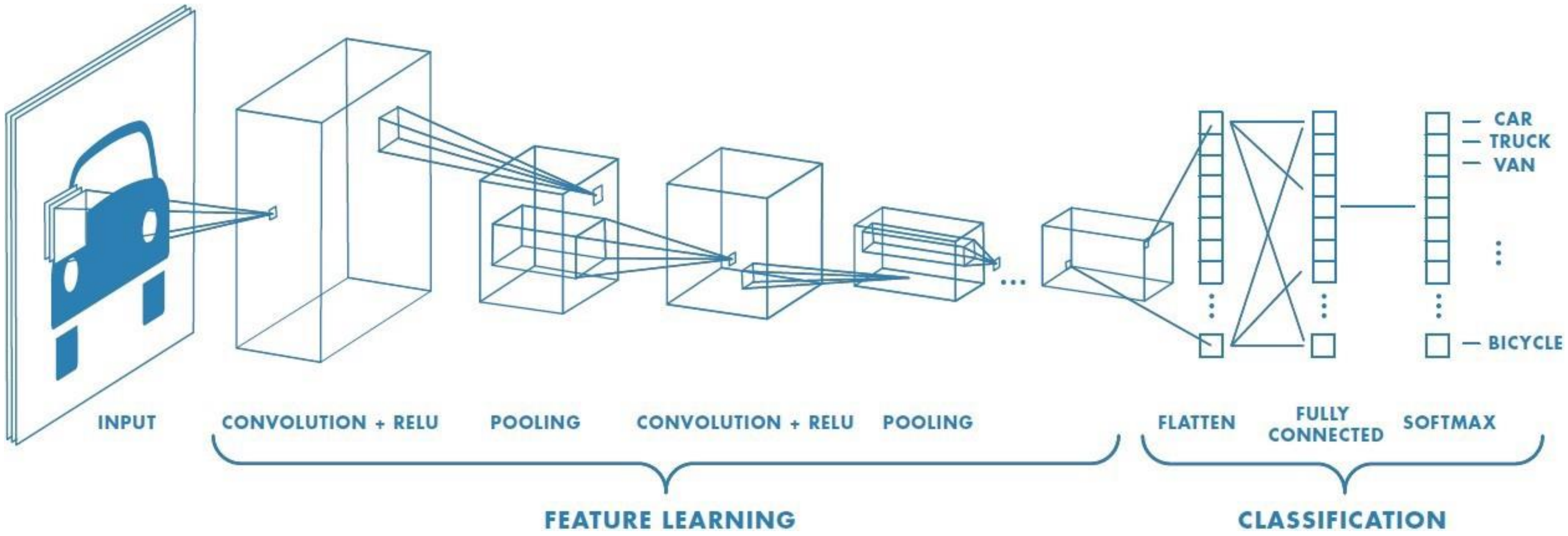
Train model with image data. Learn weights of filters in convolution layers

CNN for Classification: Feature Learning



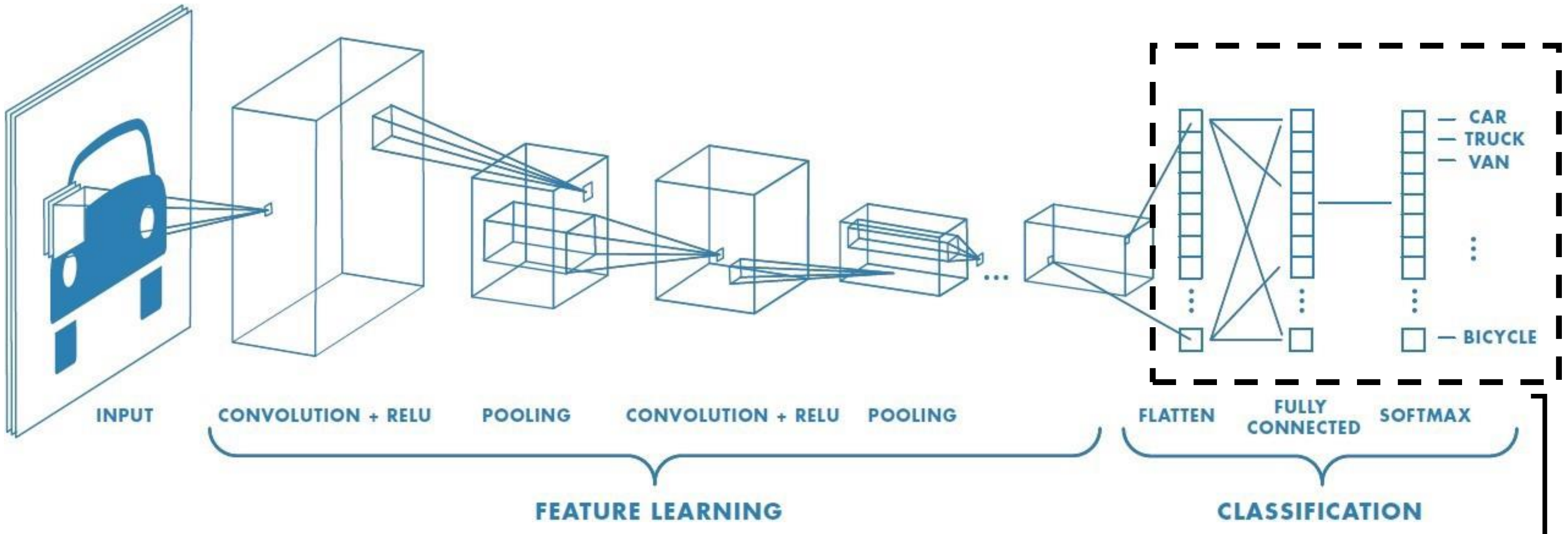
1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation
3. Reduce dimensionality and preserve spatial invariance with **pooling**

CNN for Classification: Class Probabilities



1. CONV and POOL layers output high level features of input
2. Fully connected layer uses these features for classifying input image
3. Express output as **probability** of image belonging to a particular class

CNN for Classification: Class Probabilities



Classification
Object detection
Segmentation

CNN Example

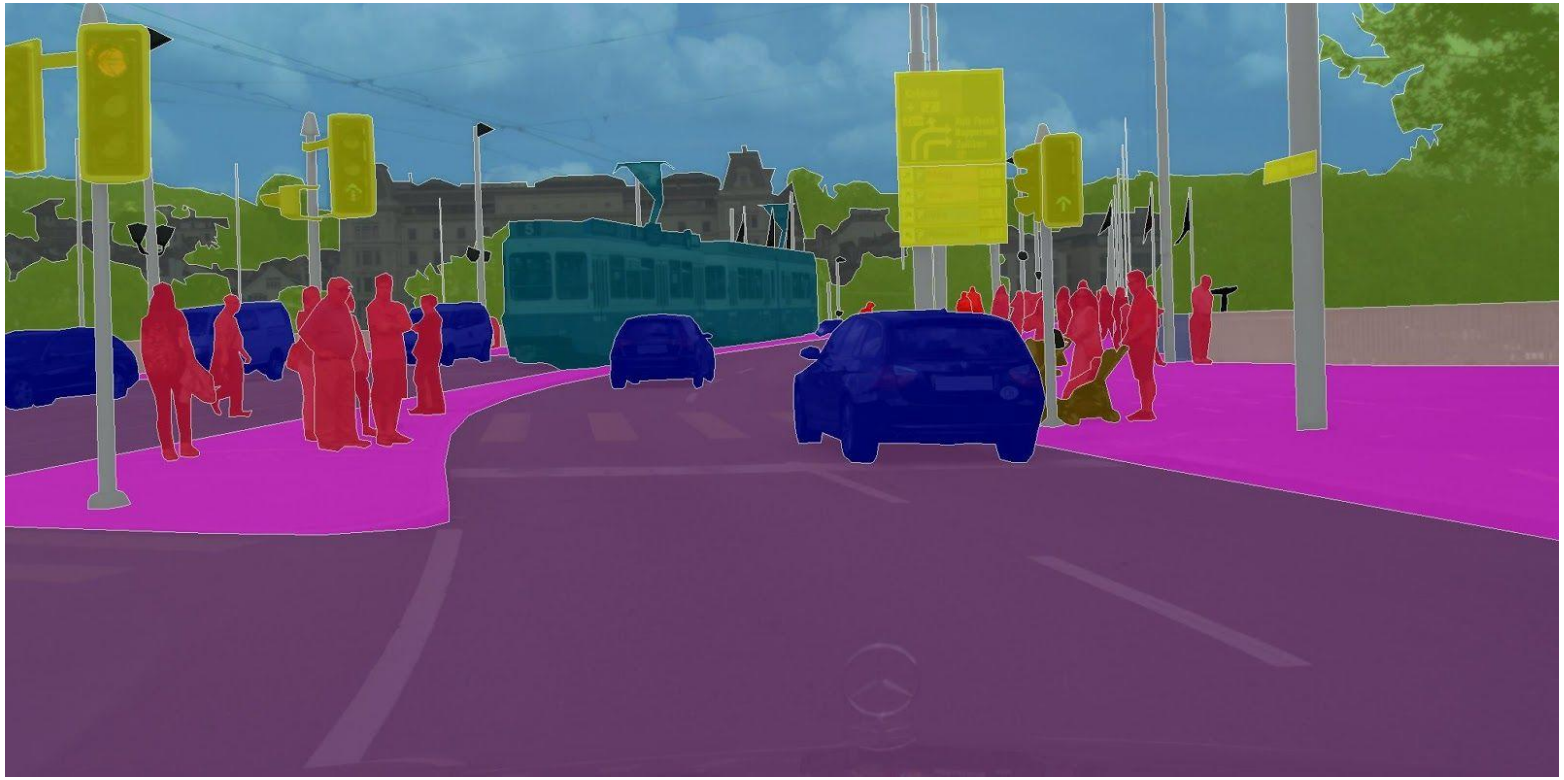
	Activation shape	Activation size	# parameters (assuming all channels of a filter have same weight)
Input:	(32, 32, 3)	3072	0
CONV1 (f = 5, s = 1, #f = 8)	(28, 28, 8)	6272	208
POOL1	(14, 14, 8)	1568	0
CONV2 (f = 5, s = 1, #f = 16)	(10, 10, 16)	1600	416
POOL2	(5, 5, 16)	400	0
FC3	(120, 1)	120	48,001
FC4	(84, 1)	84	10,081
Softmax	(10, 1)	10	841

Transposed Convolution

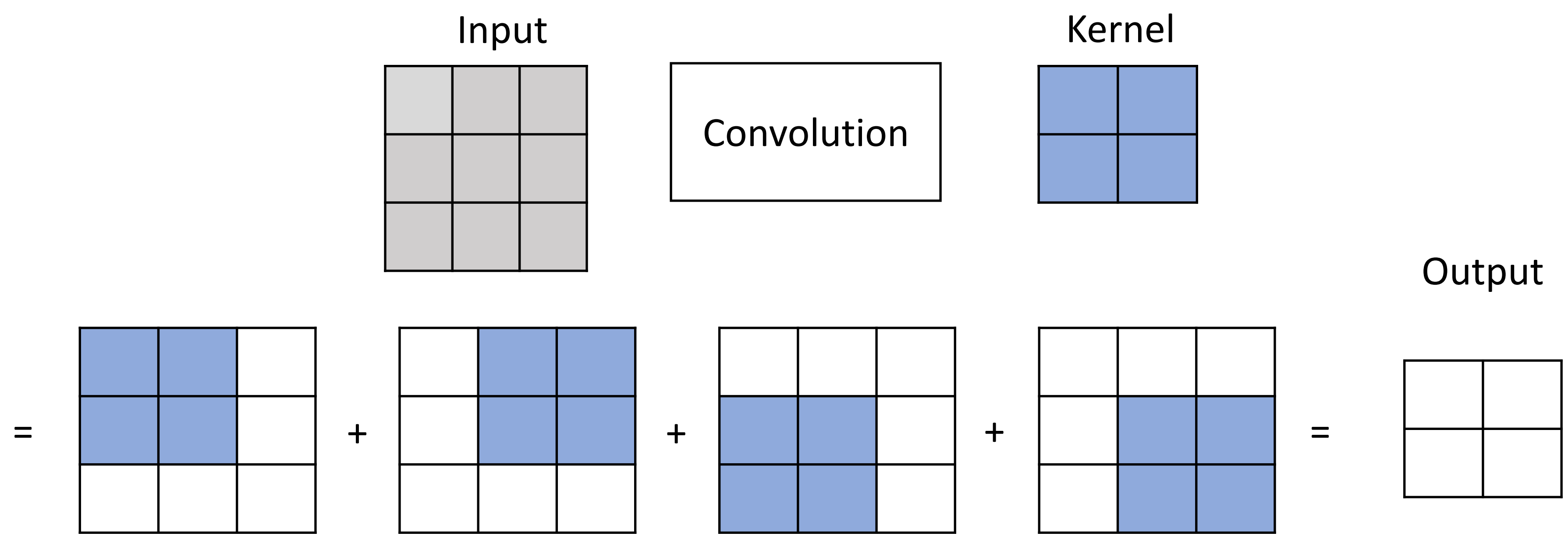
Introduction

- CNN layers, such as convolutional layers and pooling layers:
  - Typically reduce (downsample) spatial dimensions (height and width) of input, or keep them unchanged

- In semantic segmentation that classifies at pixel-level, it will be convenient if spatial dimensions of input and output are same
  - Ex., channel dimension at one output pixel can hold classification results for input pixel at same spatial position



https://medium.com/intro-to-artificial-intelligence/semantic-segmentation-udaitys-self-driving-car-engineer-nanodegree-c01eb6eaf9d



Convolution with a 2 * 2 kernel computed for a 3 * 3 input tensor → Output size becomes 2 * 2

Tranpose Convolution

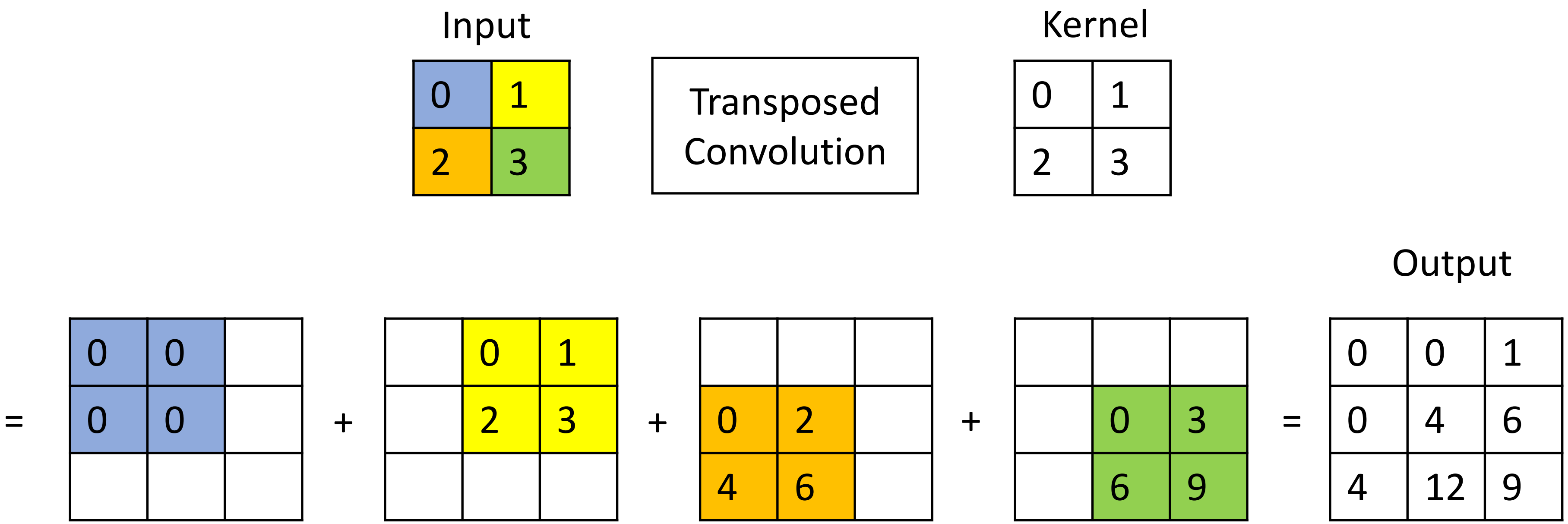
- Can use another type of CNN layers that can **increase** (upsample) spatial dimensions of intermediate feature maps

- *Transposed convolution*, also called *fractionally-strided convolution*, used for reversing down-sampling operations by convolution
  - For increasing resolution of input
  - In encoder-decoder architectures (on decoder side)

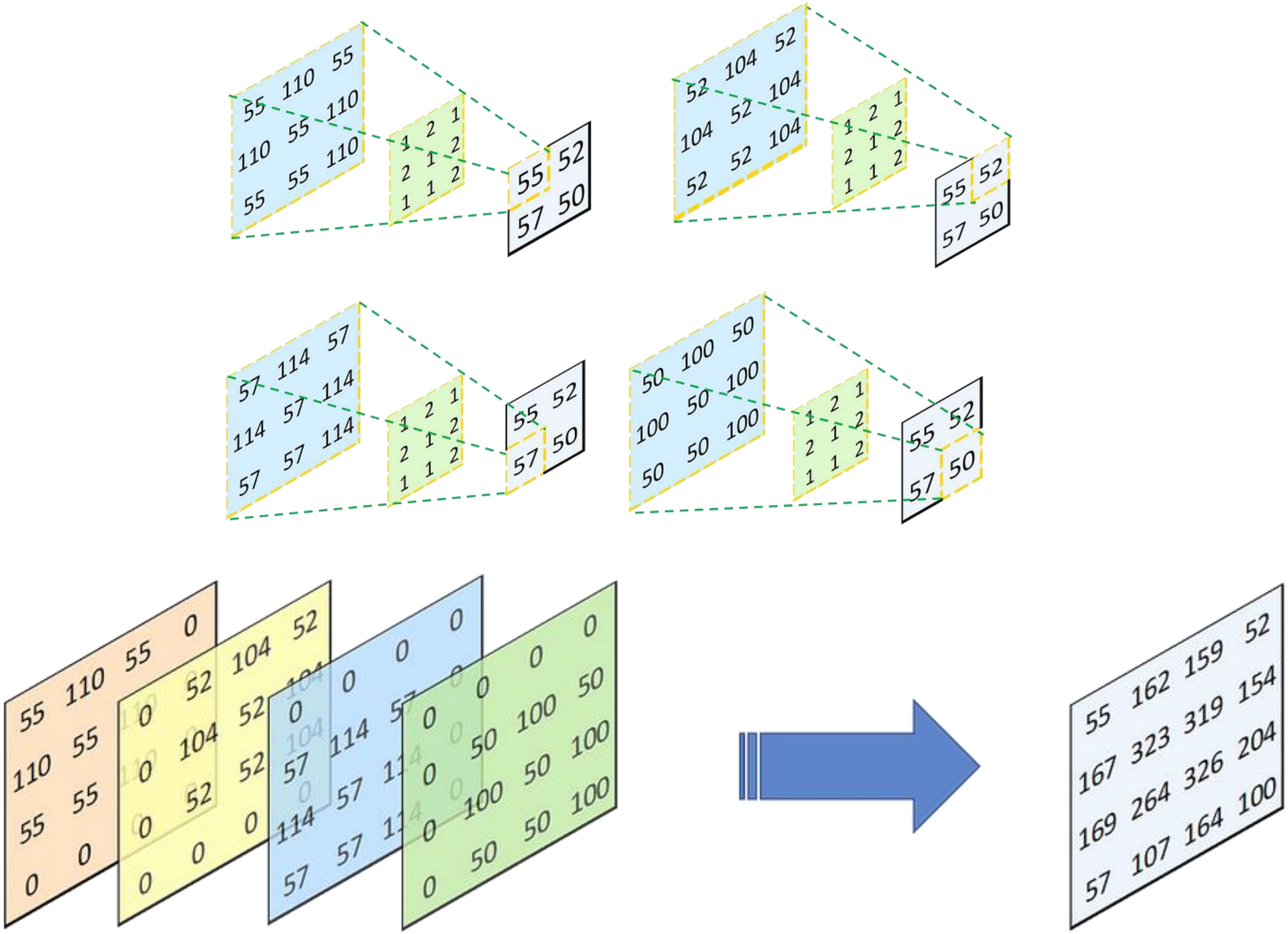
Transposed Convolution

Basic transposed convolution operation with stride of 1 and no padding

- Given a n_h*n_w input tensor and a k_h*k_w kernel
- Sliding kernel window with stride of 1 for n_w times in each row and n_h times in each column yields a total of $n_h n_w$ intermediate results
  - Each intermediate result is a ($n_h + k_h - 1$) *($n_w + k_w - 1$) tensor initialized as zeros
- To compute each intermediate tensor:
  - Each element in input tensor multiplied by kernel so that resulting k_h*k_w tensor replaces a portion in each intermediate tensor
  - Note: position of replaced portion in each intermediate tensor corresponds to position of element in input tensor used for computation
- All intermediate results summed over to produce output



Transposed convolution with a 2 * 2 kernel computed for a 2 * 2 input tensor with stride 1



https://towardsdatascience.com/

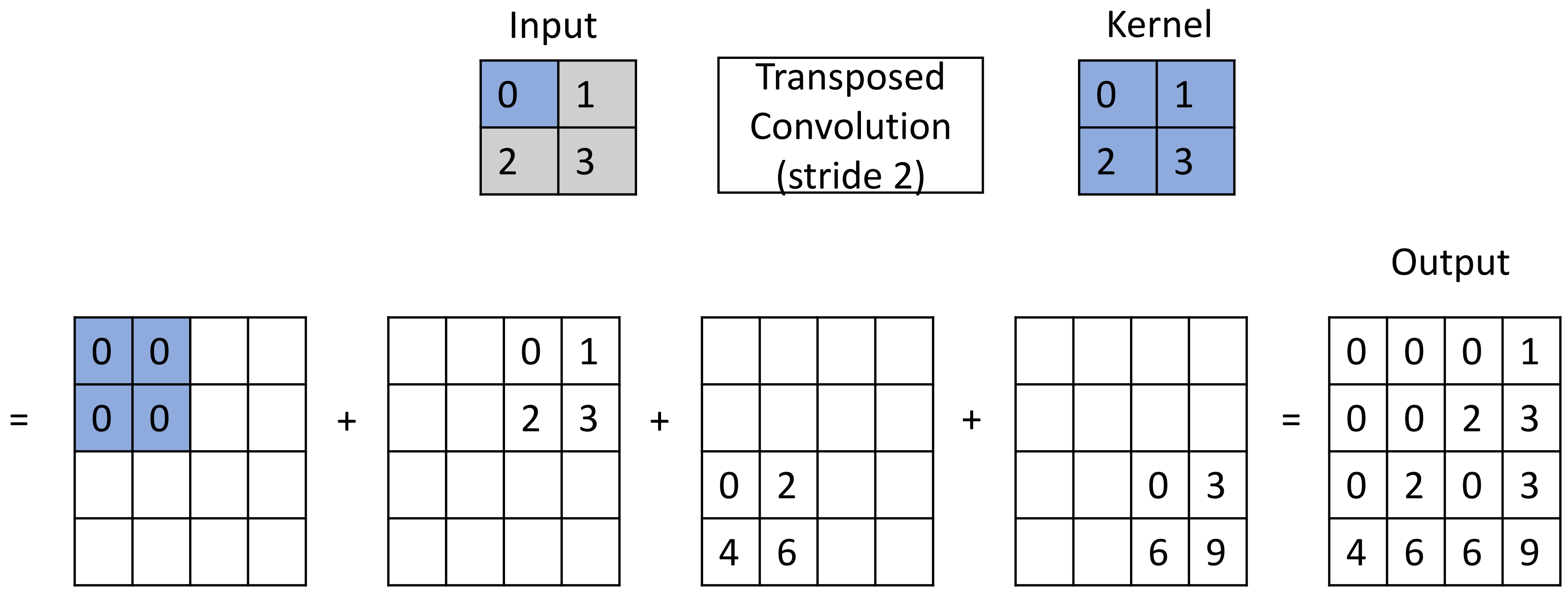
Transposed Convolution

- Regular convolution *reduces* input elements via the kernel
- Transposed convolution *broadcasts* input elements via kernel
  - Produces an output larger than the input

$$\mathbf{Y}[i : i + h, j : j + w] \mathrel{+}= \mathbf{X}[i, j\,] * \mathbf{K}$$

Padding, Strides

- Applied to output in transposed convolution
  - Different from regular convolution where padding is applied to input
- Ex., when specifying padding number on either side of height and width as 1:
  - First and last rows and columns will be removed from transposed convolution output
- Strides are specified for intermediate results (thus output), not for input



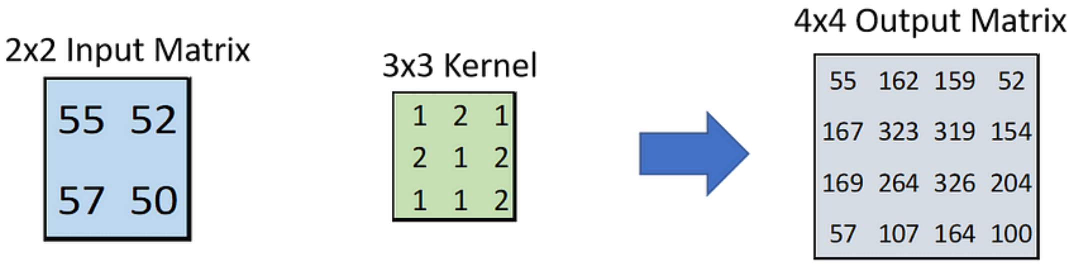
Transposed convolution with a 2 * 2 kernel computed for a 2 * 2 input tensor with stride 2

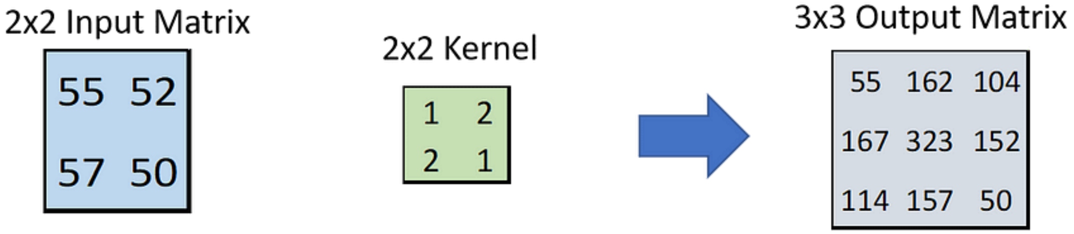
Notation

- Input: $n_h * n_w$
- Kernel: $k_h * k_w$
- Stride: (s_h, s_w)
- Padding: p
- Output: $O_h * O_w$
  - $O_h = (n_h - 1) * s_h + k_h - 2p$
  - $O_w = (n_w - 1) * s_w + k_w - 2p$

Kernel Size

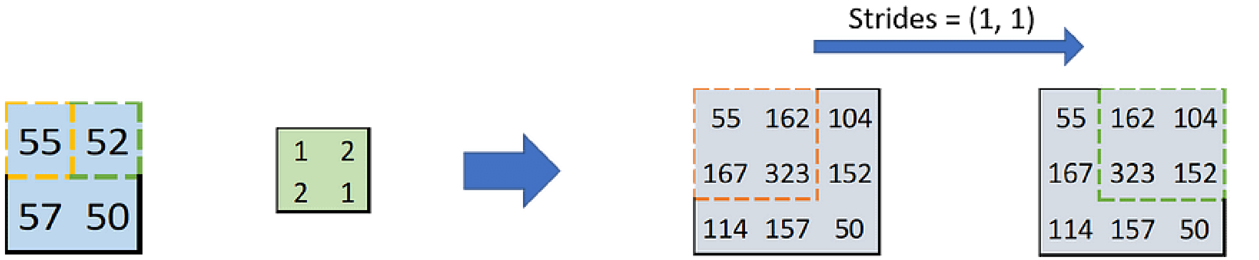
- When kernel size gets larger, we "disperse" every single number from input layer to a broader area
  - Larger the kernel size, larger the output matrix (if no padding is added)



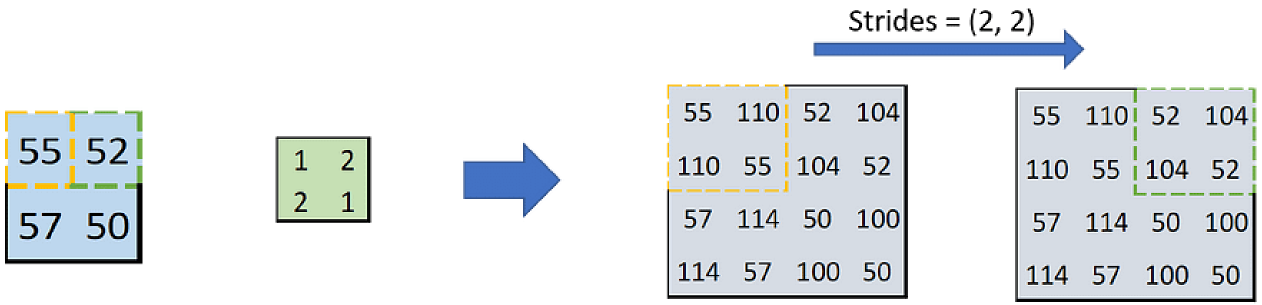


Strides

- Indicates how fast kernel moves on **output layer**
  - Kernel always move only one number at a time on input layer
  - Larger the strides, larger the output matrix (if no padding)



3x3 Output Matrix

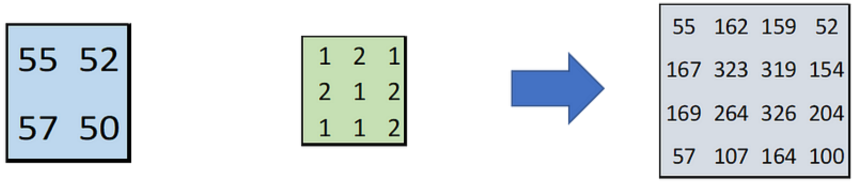


4x4 Output Matrix

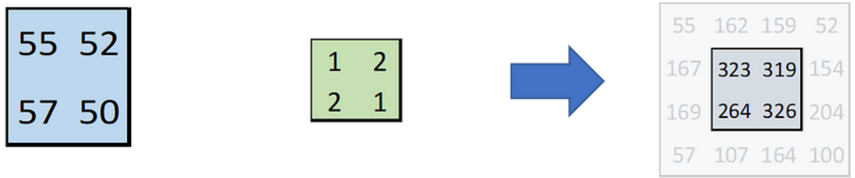
Padding

- Can be of type: “valid” and “same”
  - “valid”: output shape will be larger than input shape
  - “same”: output shape becomes input shape multiplied by stride
    - If this output shape is smaller than original output shape, only the very middle part of output is maintained
    - When padding = ‘same’ and stride = 1, output has same size as input

Padding: “valid”



Padding: “same”



Multiple Channels

- For multiple input and output channels, transposed convolution works in same way as regular convolution
- Suppose input has c_i channels, and transposed convolution assigns a $k_h \times k_w$ kernel tensor to each input channel
  - Will have a $c_i \times k_h \times k_w$ kernel for each output channel