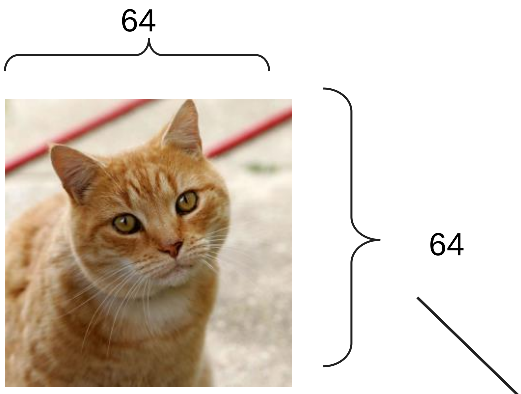
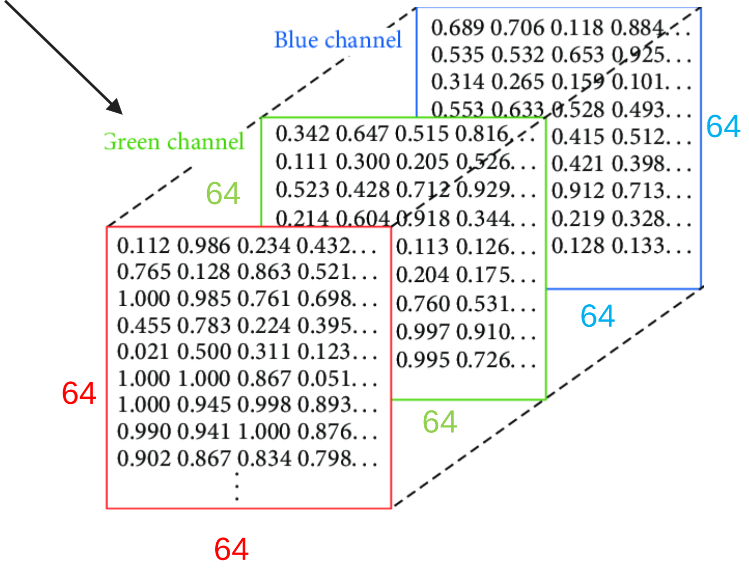
CNN (Convolutional Neural Network)

Background of CNN :

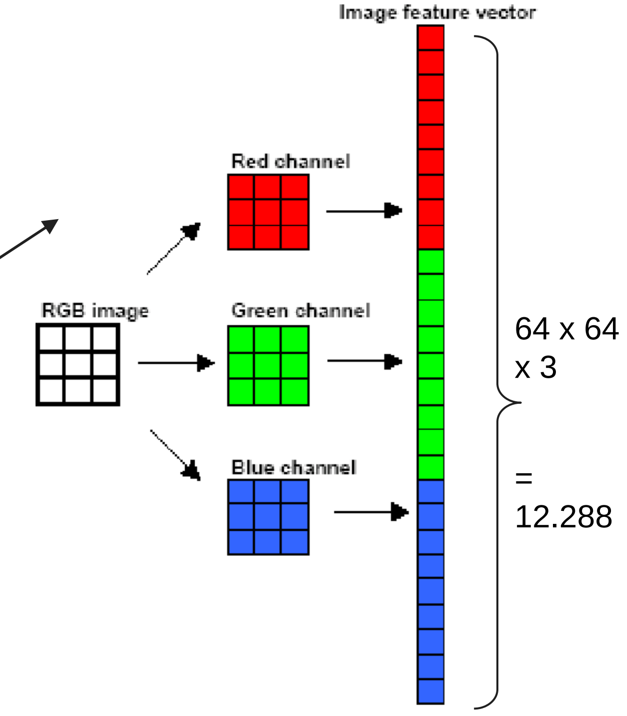




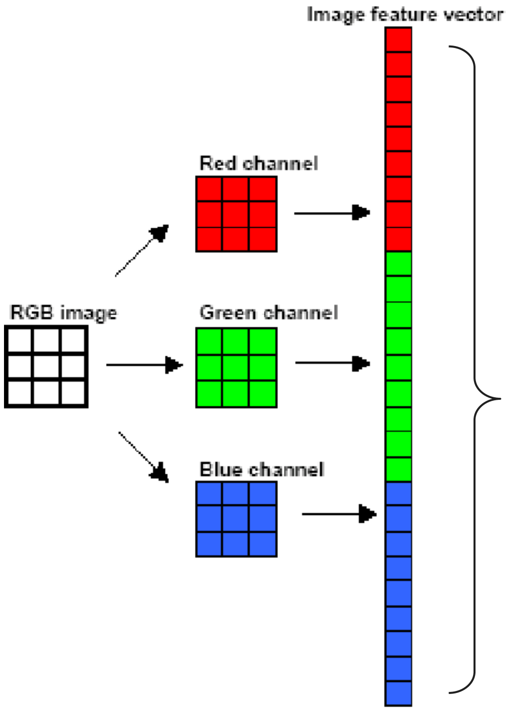
0.112 0.986 0.234 0.432...
0.765 0.128 0.863 0.521...
1.000 0.985 0.761 0.698...
0.455 0.783 0.224 0.395...
0.021 0.500 0.311 0.123...
1.000 1.000 0.867 0.051...
1.000 0.945 0.998 0.893...
0.990 0.941 1.000 0.876...
0.902 0.867 0.834 0.798...
⋮

0.342 0.647 0.515 0.816...
0.111 0.300 0.205 0.526...
0.523 0.428 0.712 0.929...
0.214 0.604 0.918 0.344...
0.113 0.126...
0.204 0.175...
0.760 0.531...
0.997 0.910...
0.995 0.726...

0.689 0.706 0.118 0.884...
0.535 0.532 0.653 0.925...
0.314 0.265 0.159 0.101...
0.553 0.633 0.528 0.493...
0.415 0.512...
0.421 0.398...
0.912 0.713...
0.219 0.328...
0.128 0.133...



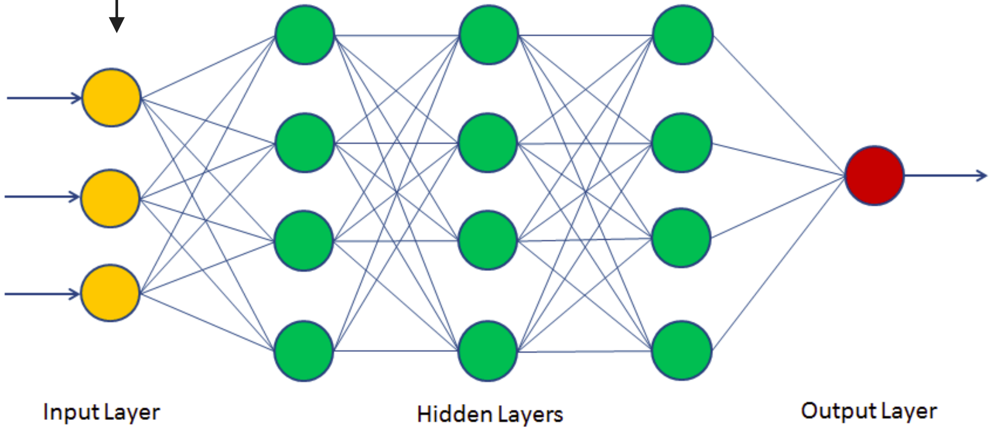
64 x 64 x 3 = 12.288



= 12.288

12288 nodes in input layer

No too large for parameters

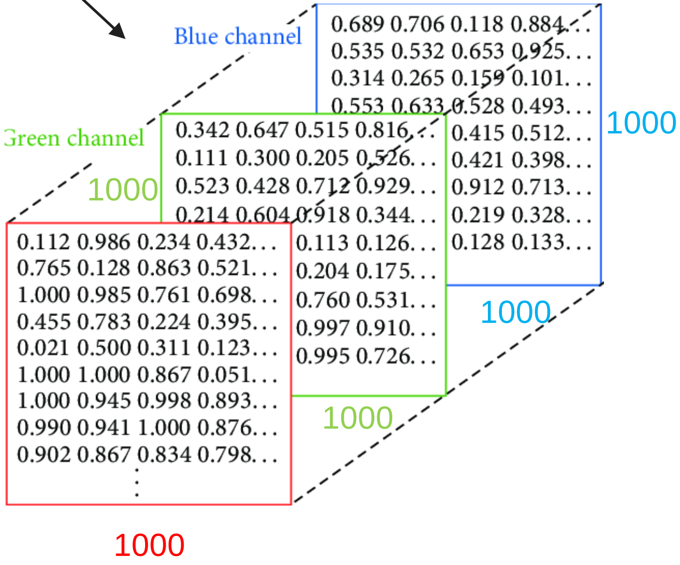


CNN (Convolutional Neural Network)

Background of CNN :

1000

1000



Blue channel

0.689 0.706 0.118 0.884...
0.535 0.532 0.653 0.925...
0.314 0.265 0.159 0.101...
0.553 0.633 0.528 0.493...
0.415 0.512...
0.421 0.398...
0.912 0.713...
0.219 0.328...
0.128 0.133...

1000

1000

Green channel

0.342 0.647 0.515 0.816...
0.111 0.300 0.205 0.526...
0.523 0.428 0.712 0.929...
0.214 0.604 0.918 0.344...
0.113 0.126...
0.204 0.175...
0.760 0.531...
0.997 0.910...
0.995 0.726...

1000

1000

0.112 0.986 0.234 0.432...
0.765 0.128 0.863 0.521...
1.000 0.985 0.761 0.698...
0.455 0.783 0.224 0.395...
0.021 0.500 0.311 0.123...
1.000 1.000 0.867 0.051...
1.000 0.945 0.998 0.893...
0.990 0.941 1.000 0.876...
0.902 0.867 0.834 0.798...
⋮

1000

1000

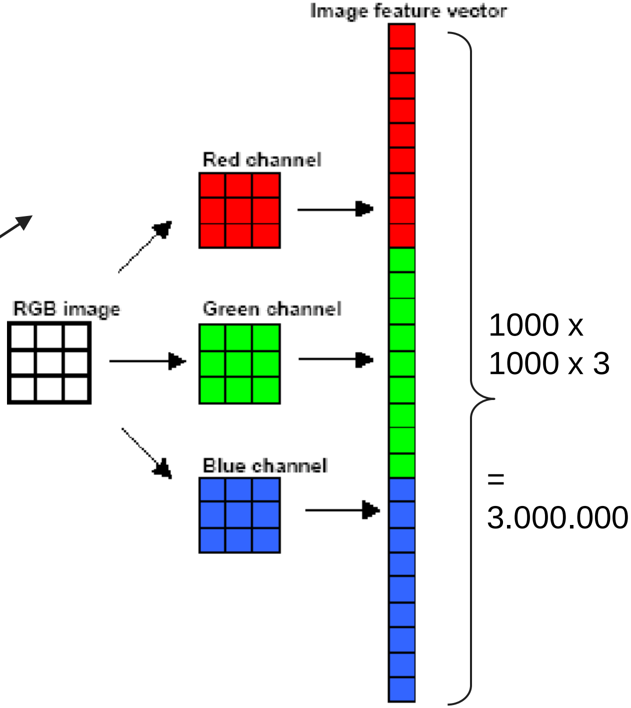


Image feature vector

Red channel

RGB image

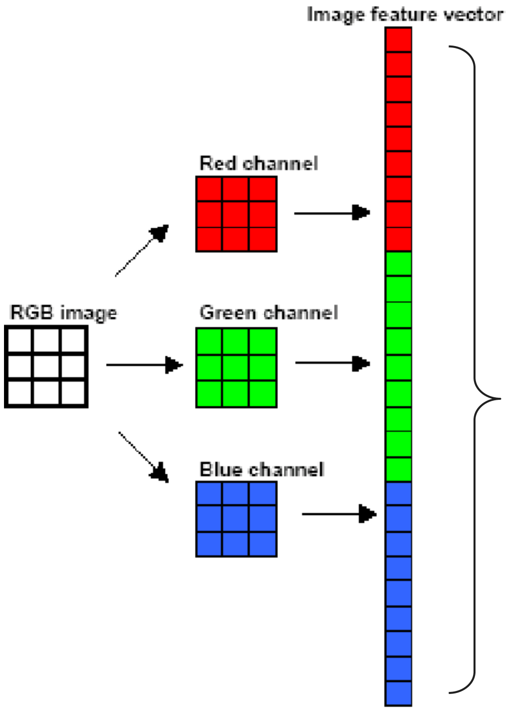
Green channel

Blue channel

1000 x
1000 x 3

=

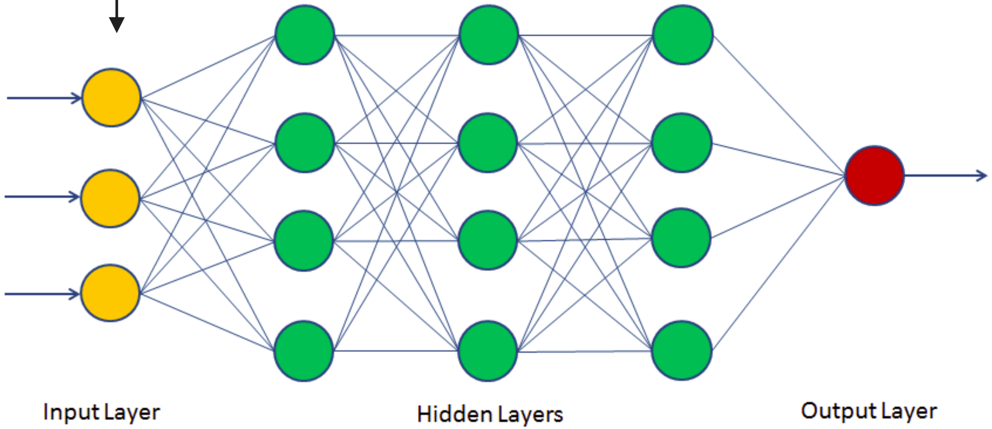
3.000.000



= 3.000.000

3 M nodes
in input layer

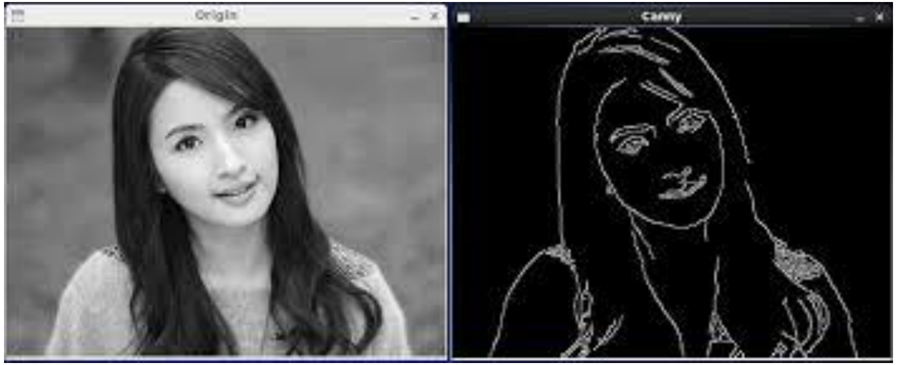
Computationally
expensive

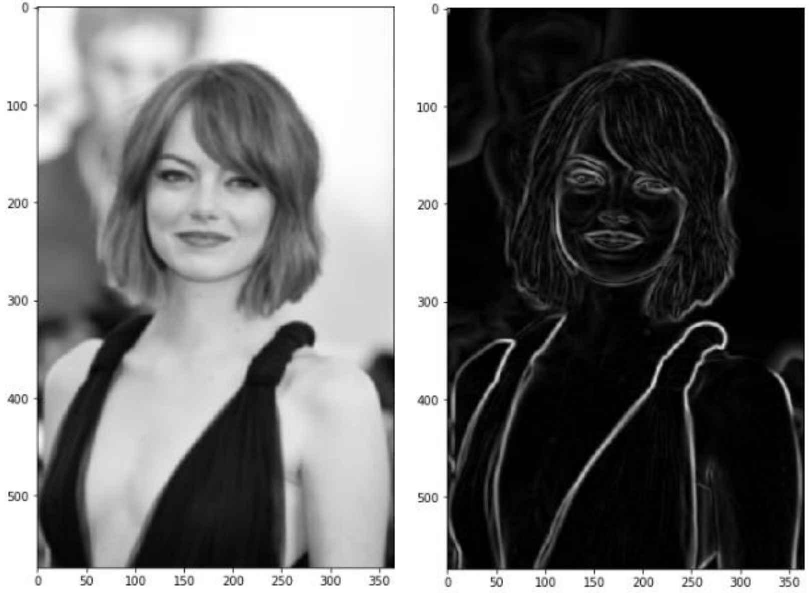


Large matrix might
get stuck

How CNN works :

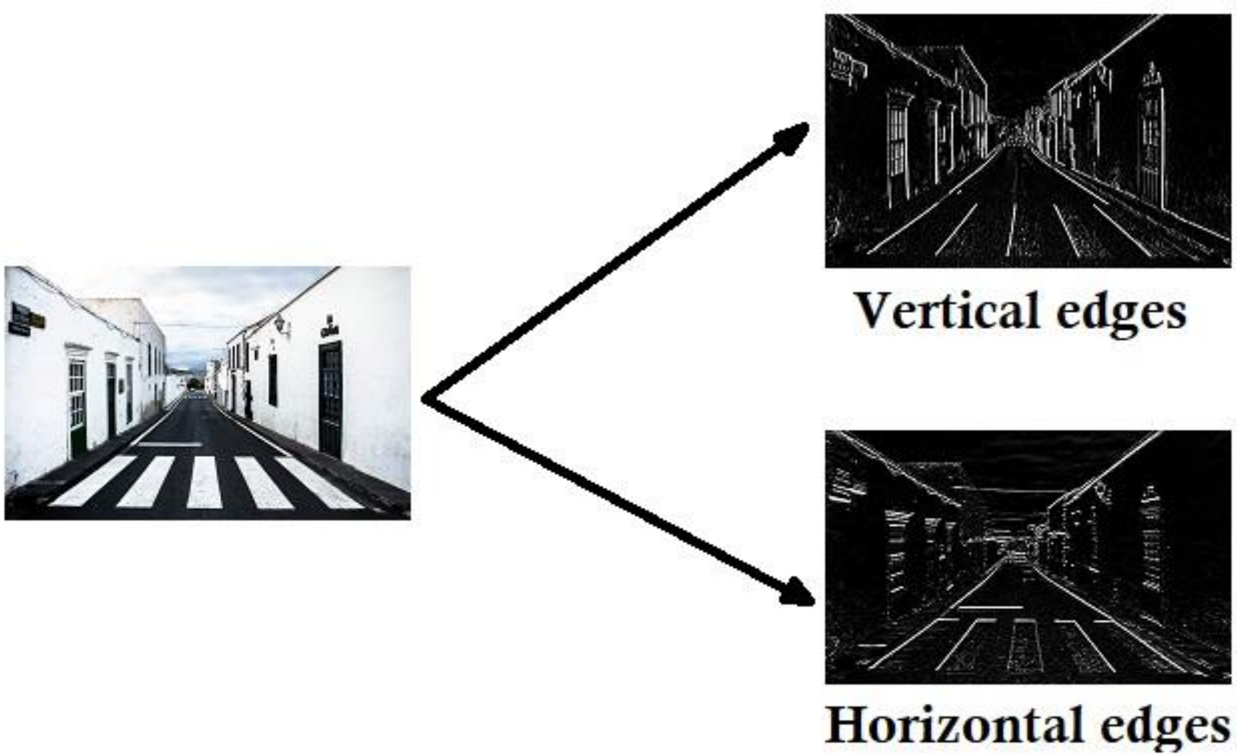
Edge detection



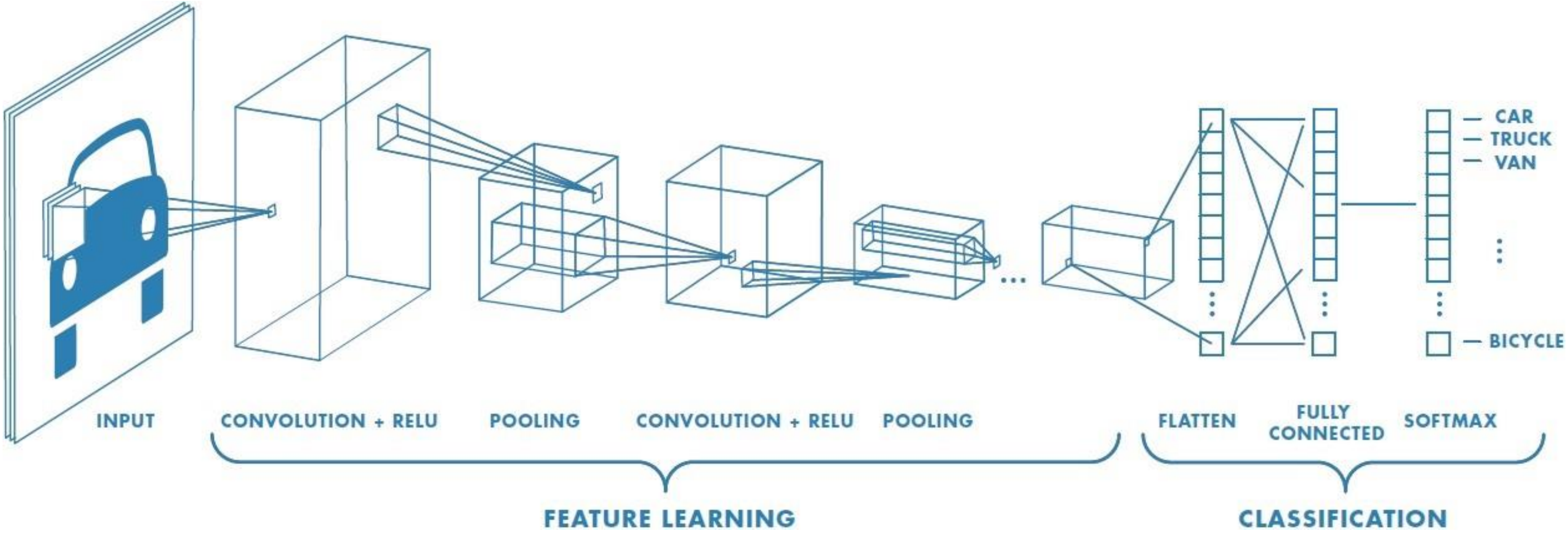


How CNN works :

Edge detection



CNN (Convolutional Neural Network)



Convolution steps

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

6 x6
Gray scale img

convolution

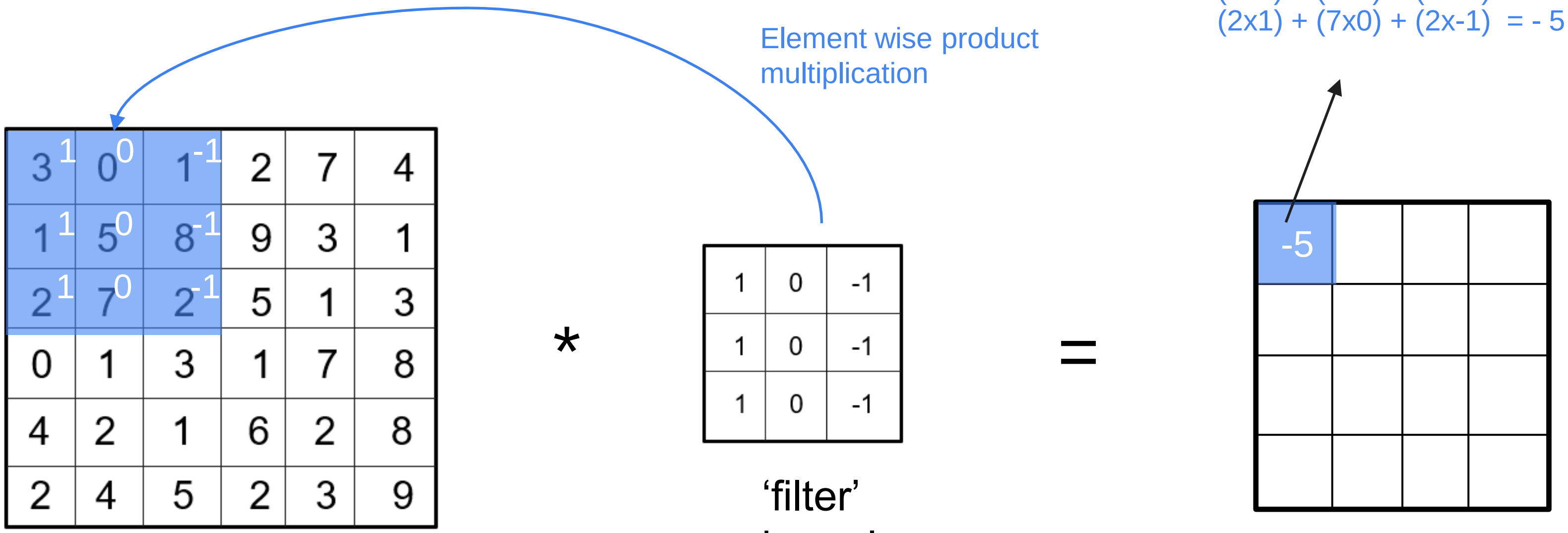
*

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

‘filter’
kernel

=

Convolution steps



(3x1) + (0x0) + (1x-1) +
(1x1) + (5x0) + (8x-1) +
(2x1) + (7x0) + (2x-1) = - 5

Element wise product multiplication

3^{1}	0^{0}	1^{-1}	2	7	4
1^{1}	5^{0}	8^{-1}	9	3	1
2^{1}	7^{0}	2^{-1}	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 x6
Gray scale img

*

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

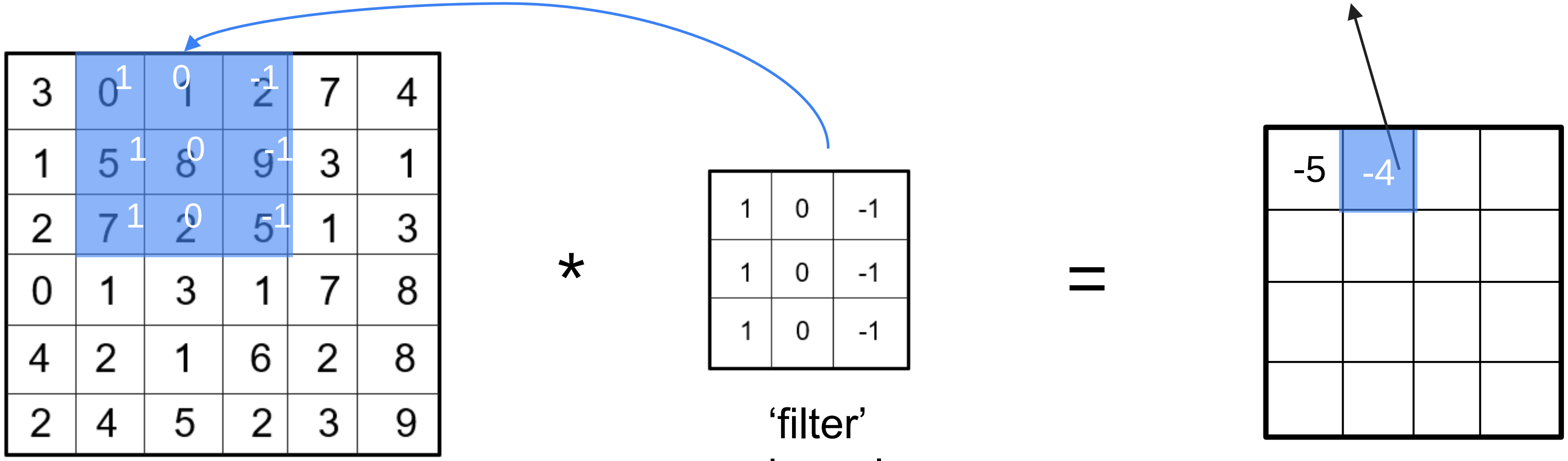
‘filter’
kernel

=

-5			

Convolution steps

(0x1) + (1x0) + (2x-1) +
(5x1) + (8x0) + (9x-1) +
(7x1) + (2x0) + (5x-1) = - 4



3	0^{1}	1^{0}	2^{-1}	7	4
1	5^{1}	8^{0}	9^{-1}	3	1
2	7^{1}	2^{0}	5^{-1}	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 x6
Gray scale img

*

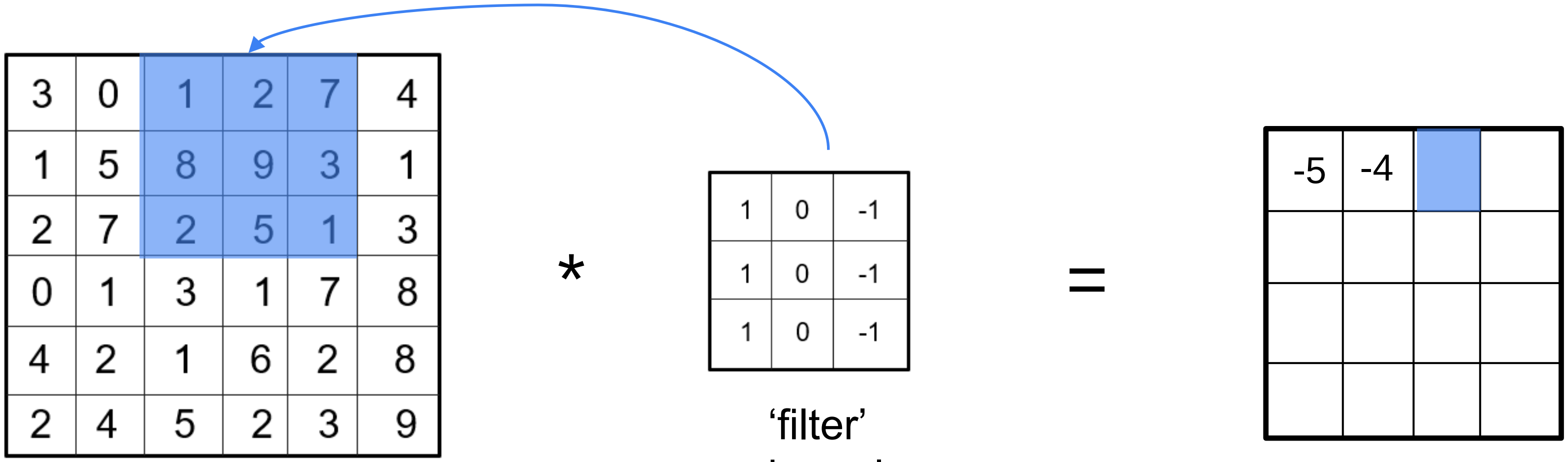
1	0	-1
1	0	-1
1	0	-1

‘filter’
kernel

=

-5	-4		

Convolution steps



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

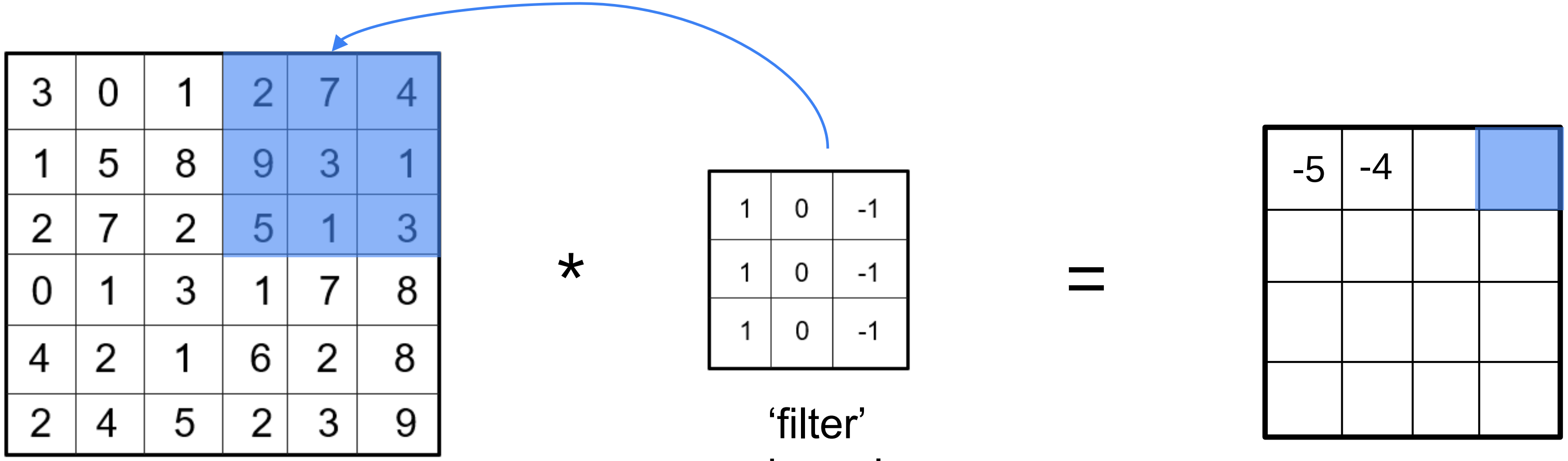
6 x6
Gray scale img

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

‘filter’
kernel

-5	-4		

Convolution steps



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

6 x6
Gray scale img

*

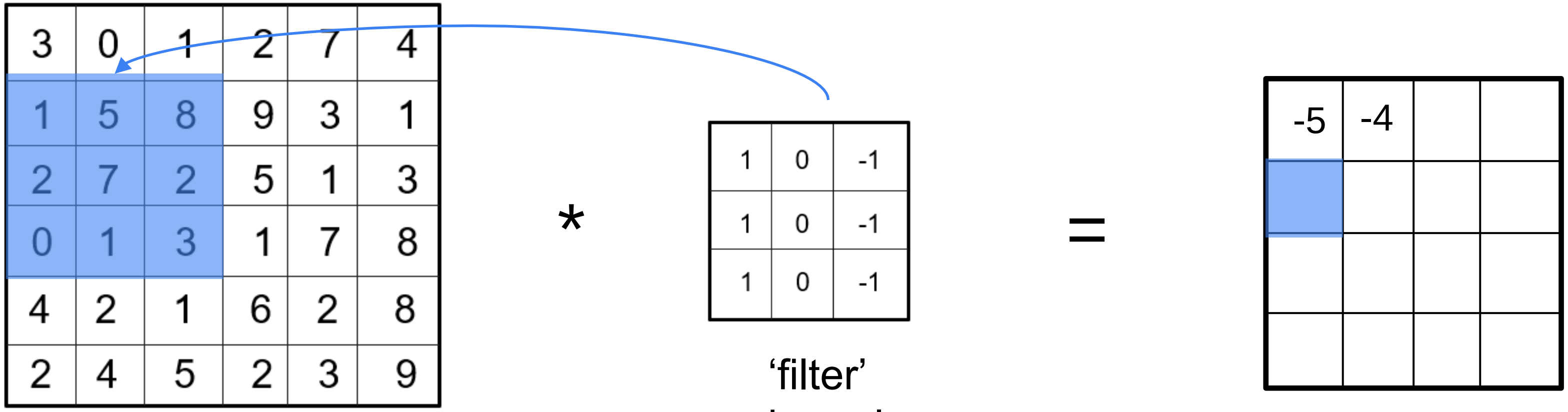
1	0	-1
1	0	-1
1	0	-1

‘filter’
kernel

=

-5	-4		

Convolution steps



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

6 x6
Gray scale img

*

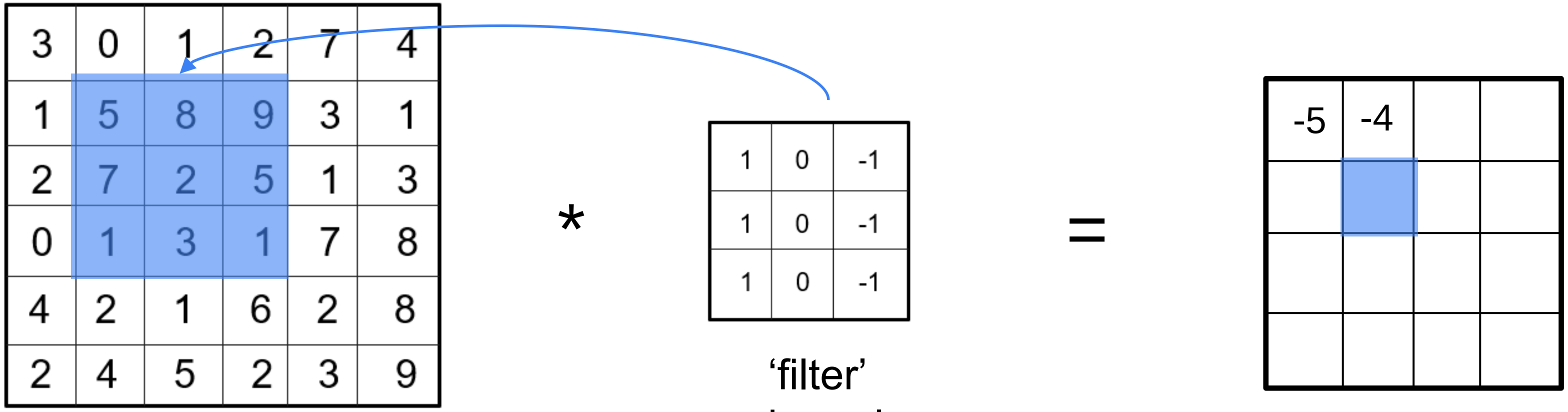
1	0	-1
1	0	-1
1	0	-1

‘filter’
kernel

=

-5	-4		

Convolution steps



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

*

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

=

-5	-4		

‘filter’
kernel

6 x6
Gray scale img

Convolution steps

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

6 x6
Gray scale img

*

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

‘filter’
kernel

=

Final result

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

Python = conv.forward
Tensorflow = tf.nn.conv2D
Keras = conv.2D

Simplified Example CNN

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

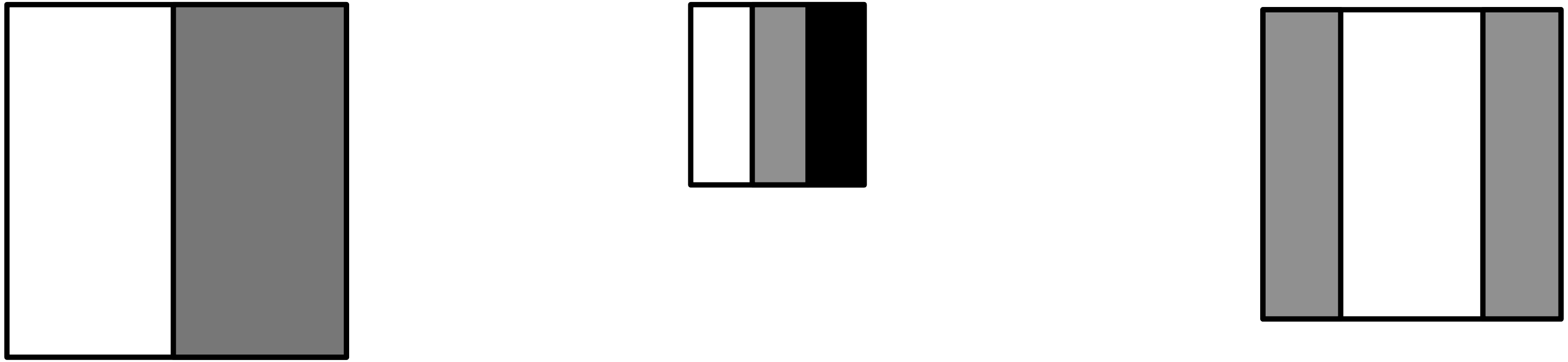
*

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

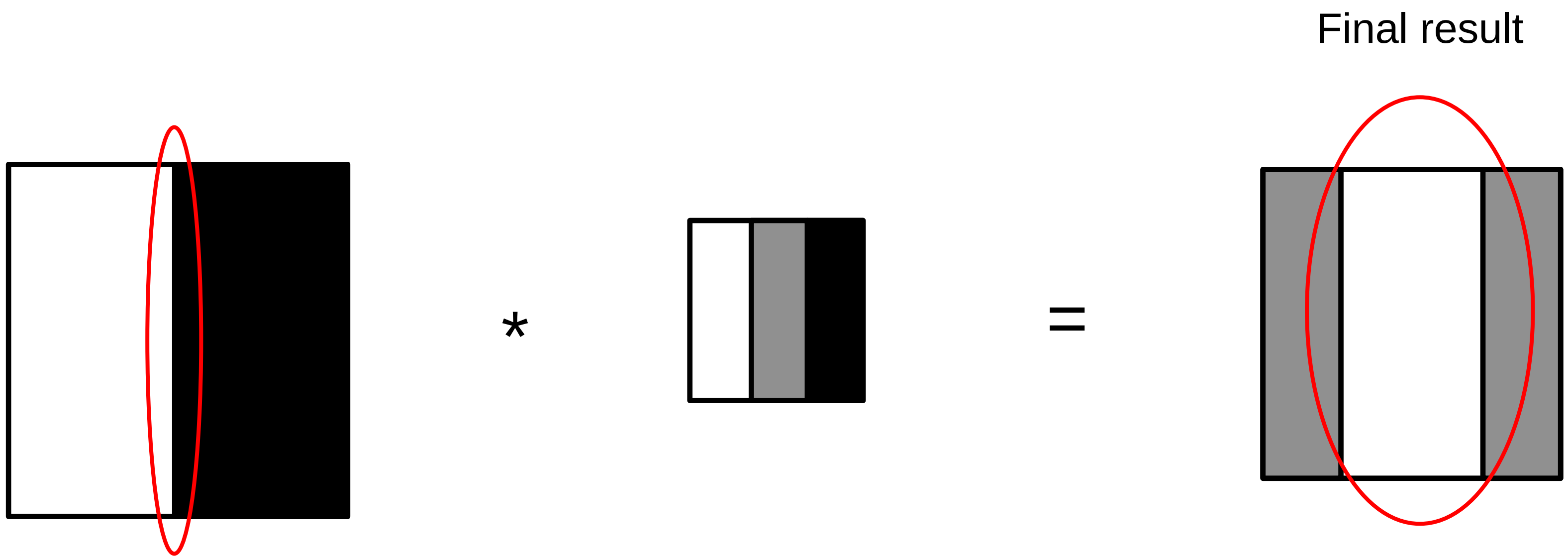
=

Final result

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

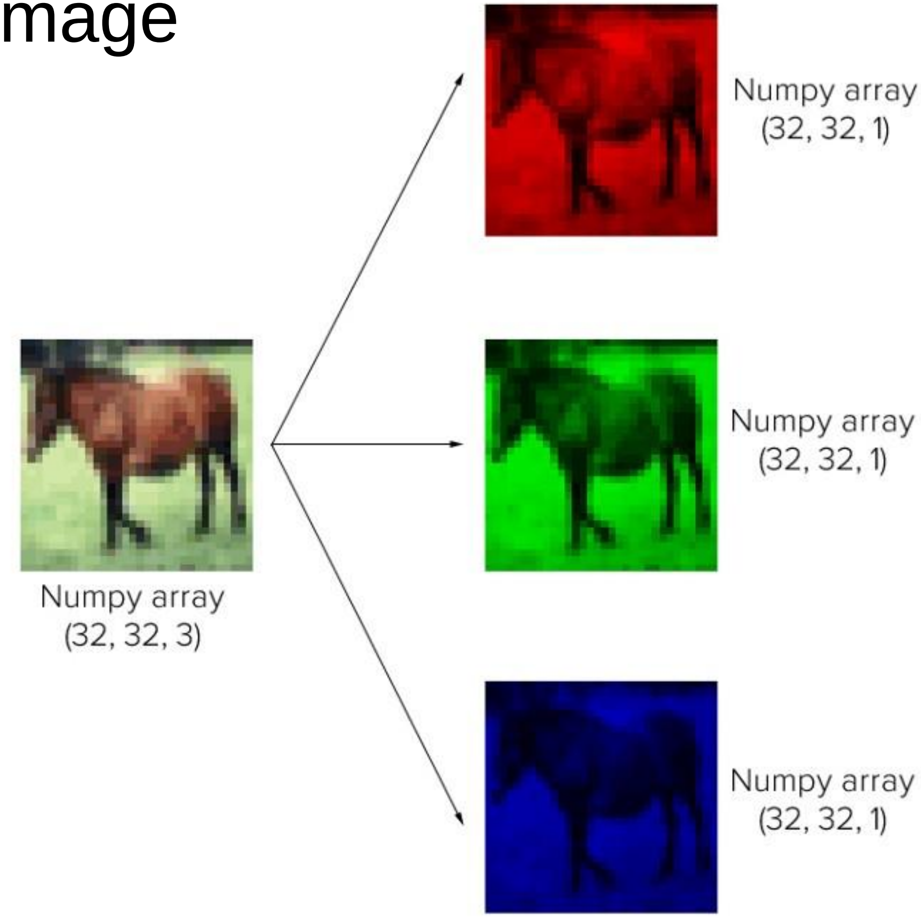


Simplified Example CNN



Brighter area = detects the vertical edge

CNN in RGB Image



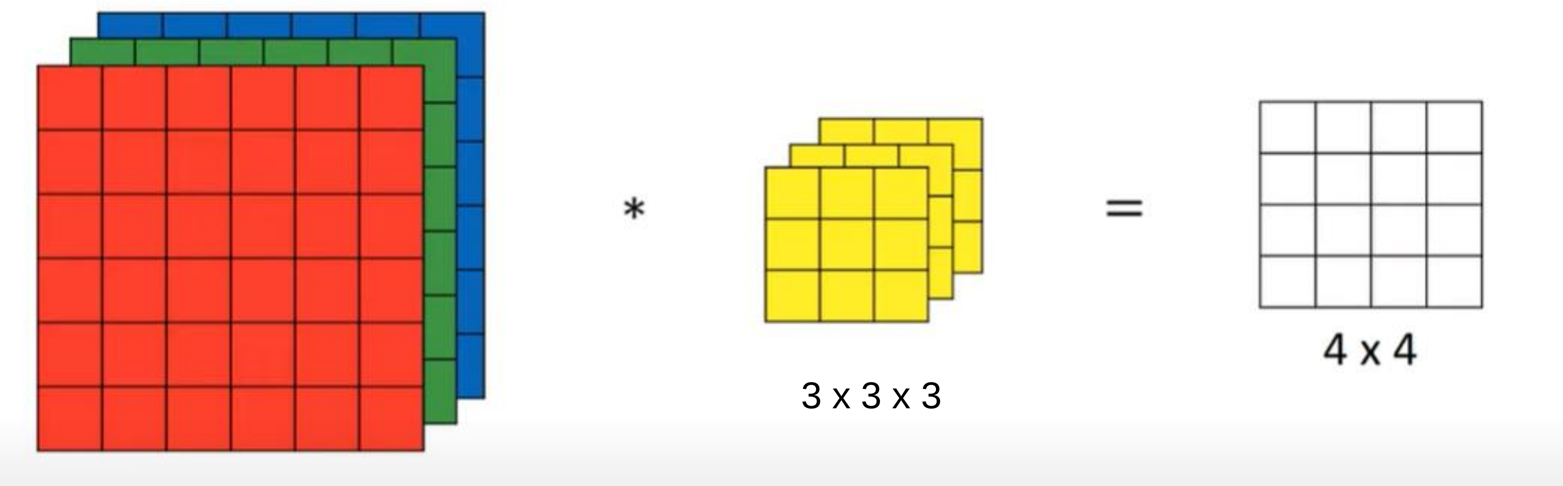
Numpy array
(32, 32, 3)

Numpy array
(32, 32, 1)

Numpy array
(32, 32, 1)

Numpy array
(32, 32, 1)

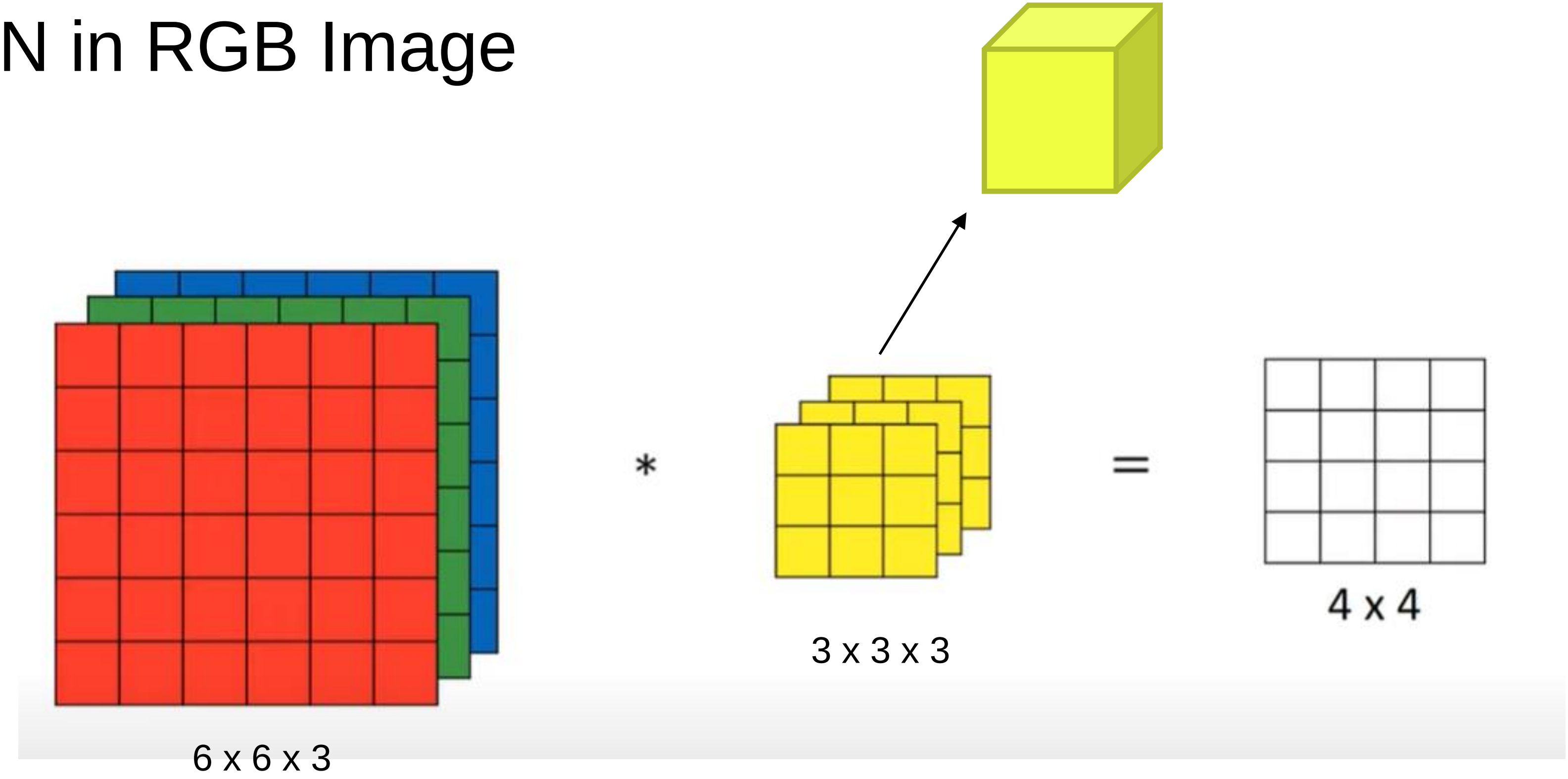
CNN in RGB Image



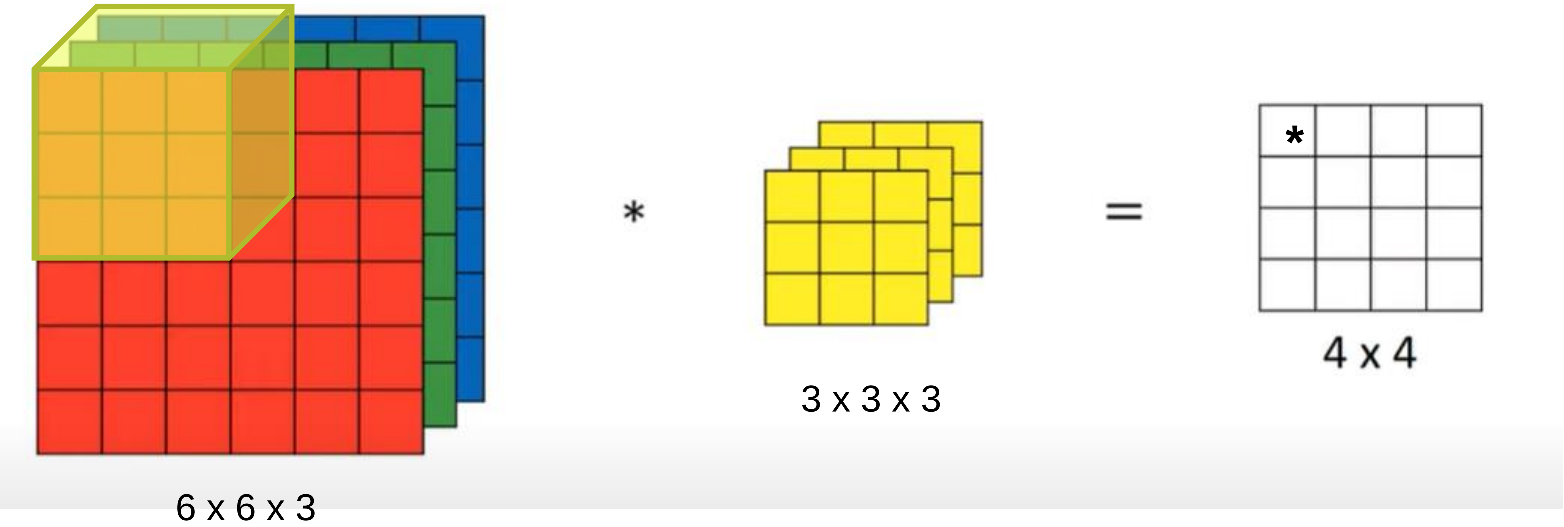
3 x 3 x 3

6 x 6 x 3

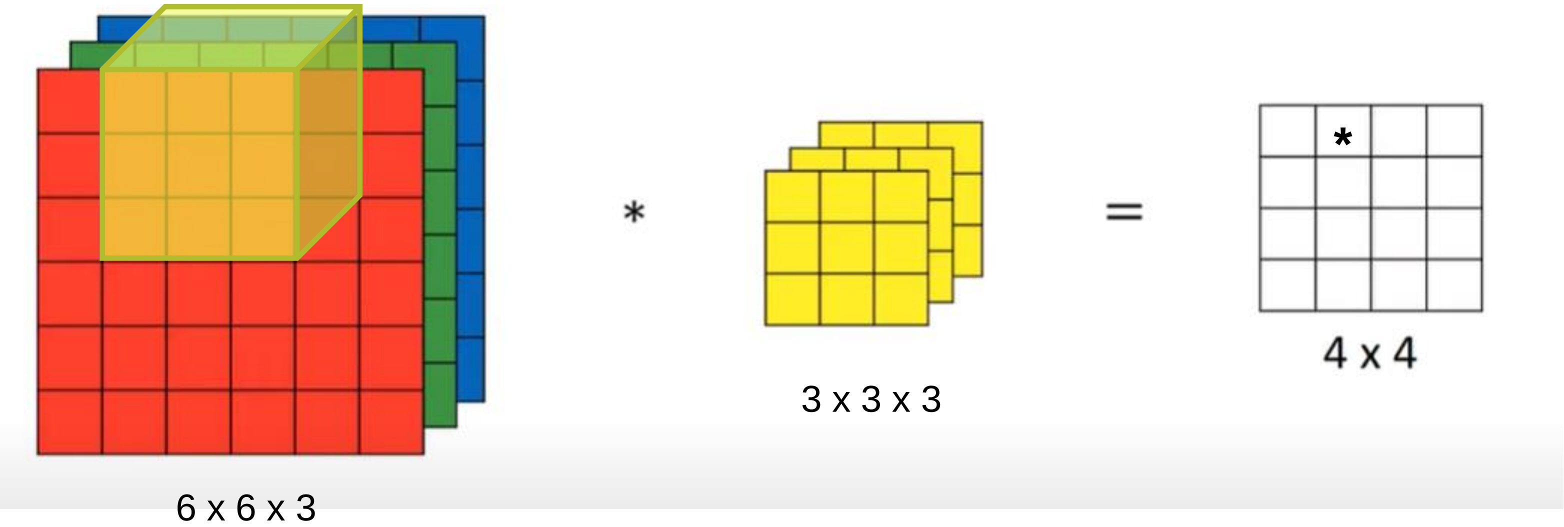
CNN in RGB Image



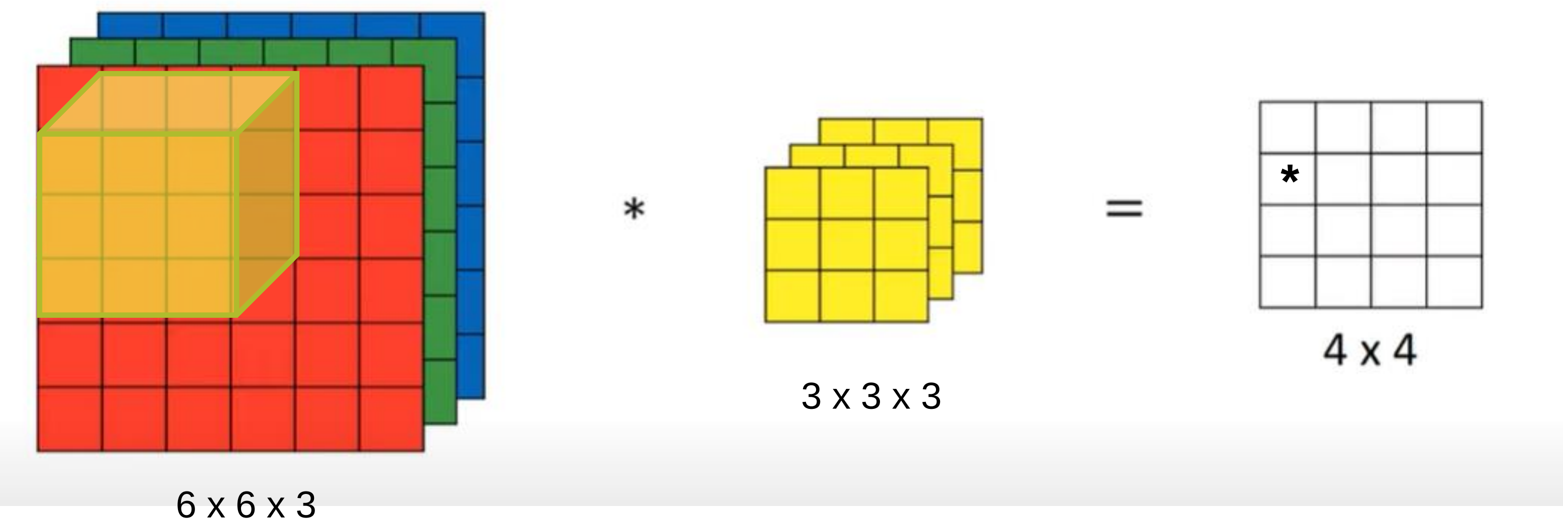
CNN in RGB Image



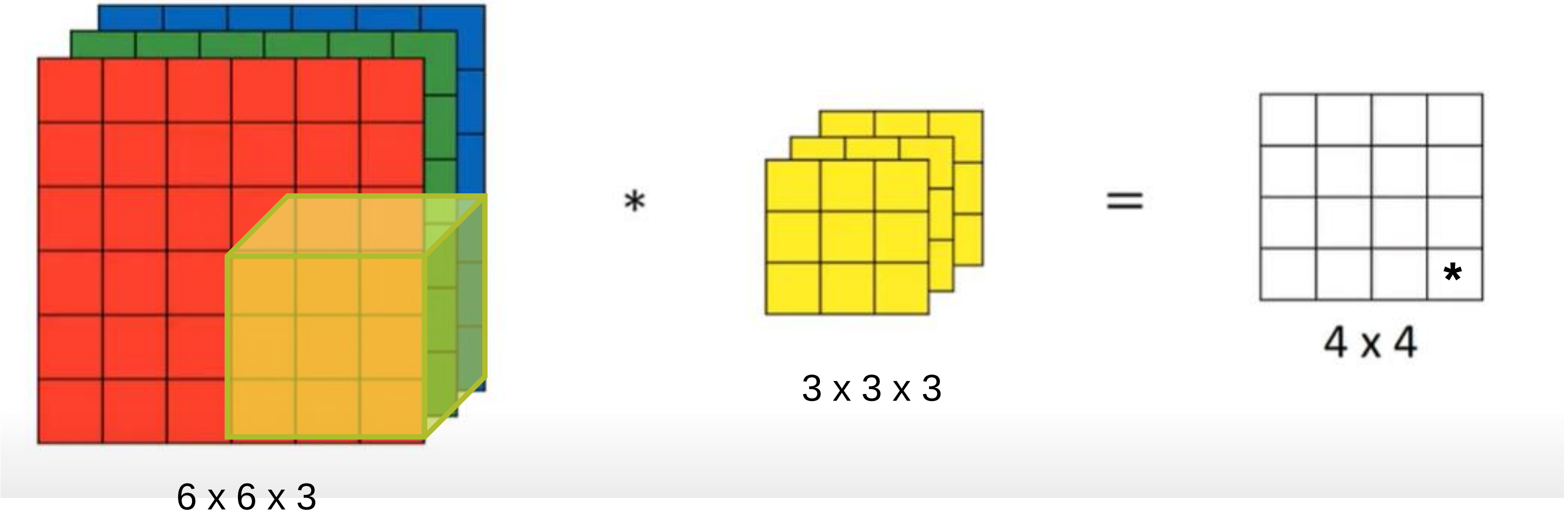
CNN in RGB Image



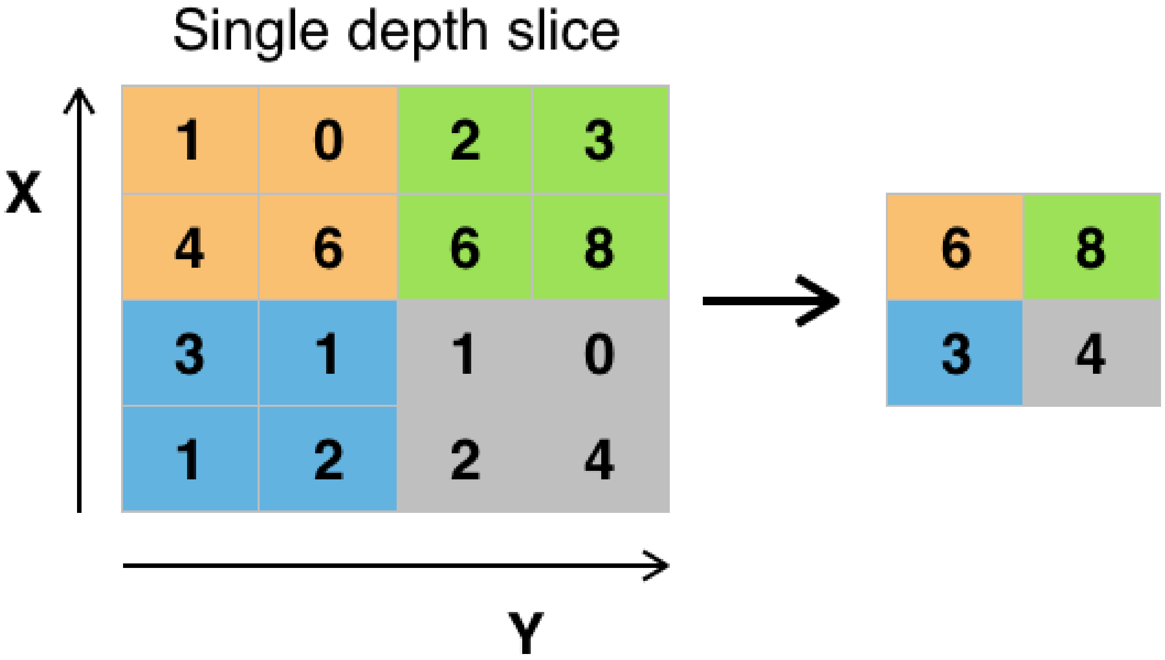
CNN in RGB Image



CNN in RGB Image



Max Pooling

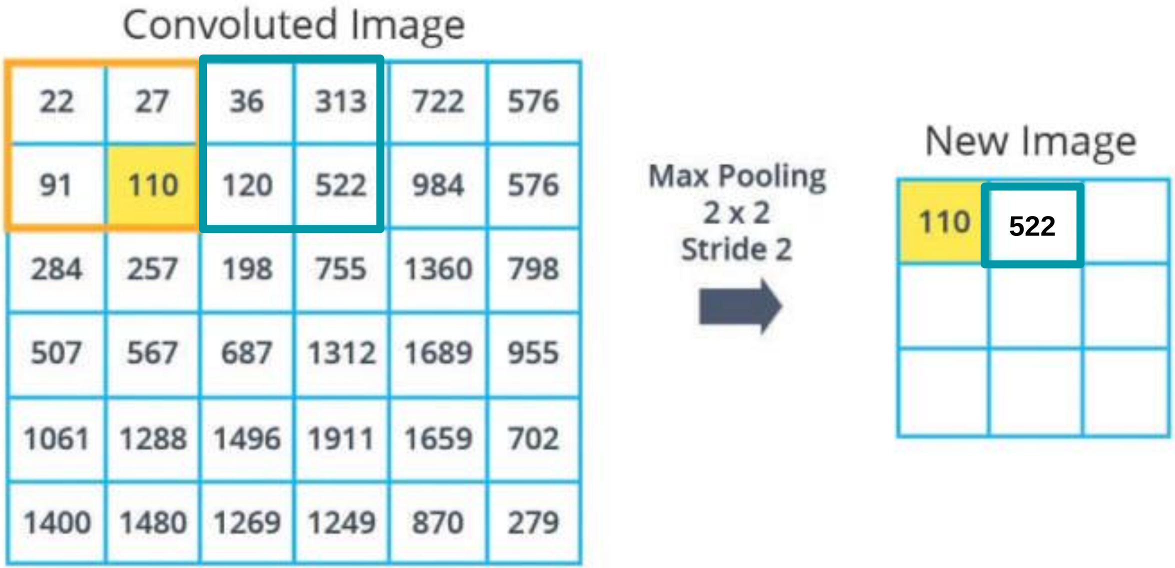


Single depth slice

1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

6	8
3	4

Max Pooling



Convoluted Image

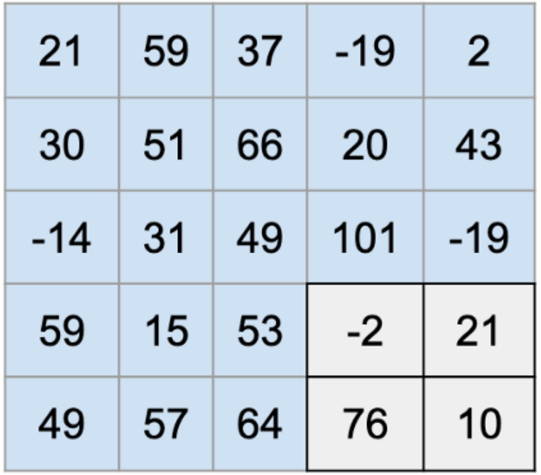
22	27	36	313	722	576
91	110	120	522	984	576
284	257	198	755	1360	798
507	567	687	1312	1689	955
1061	1288	1496	1911	1659	702
1400	1480	1269	1249	870	279

Max Pooling
2 x 2
Stride 2

New Image

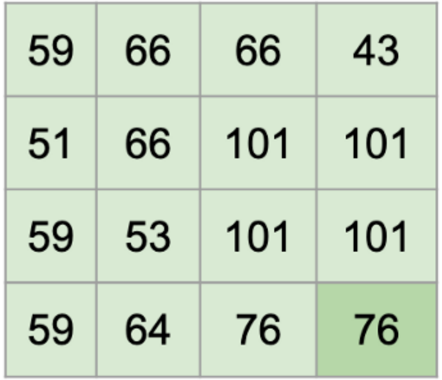
110	522	

Max Pooling



21	59	37	-19	2
30	51	66	20	43
-14	31	49	101	-19
59	15	53	-2	21
49	57	64	76	10

Convolved Feature



59	66	66	43
51	66	101	101
59	53	101	101
59	64	76	76

Max Pooled Feature