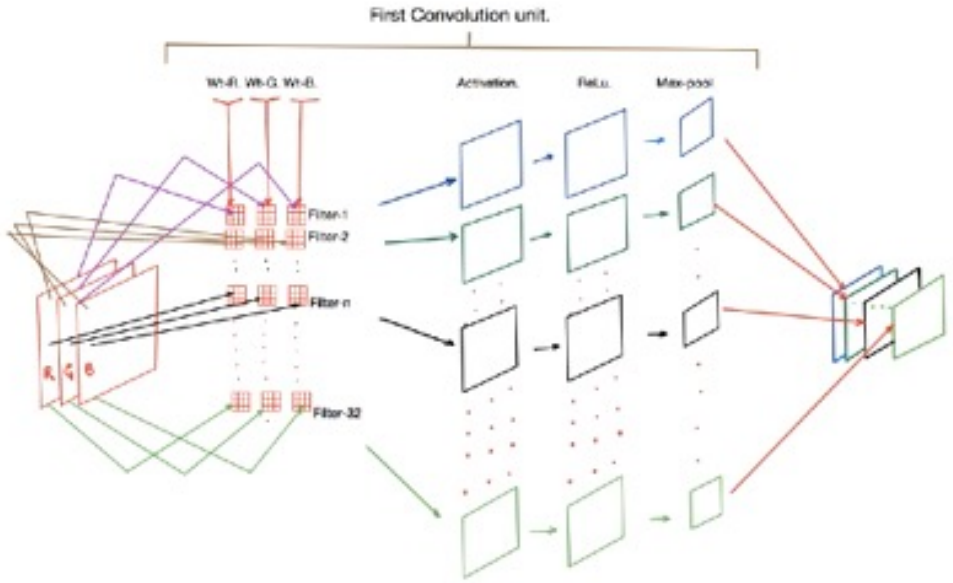
DEEP LEARNING – Convolutional Neural Network

Trainer: Dr. Darshan Ingle.

What is Convolution?

- A Convolutional Neural Network is a “Neural Network with Convolution”
- So in order to understand CNN, we must understand Convolution.



Is Convolution Mysterious?

Not at all.

Addition	+
Subtraction	−
Multiplication	×
Division	÷

✓ (1)

✓ (2)

⇒ Convolution.

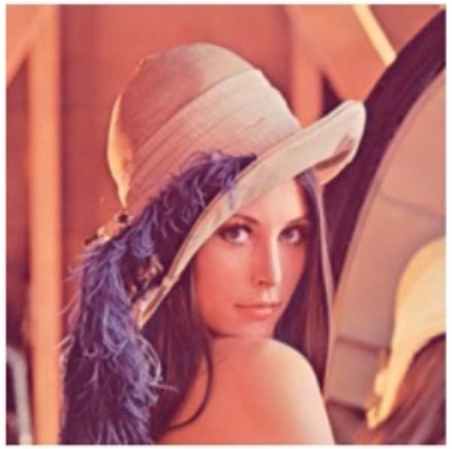
Understanding Convolution without Math

- Lets try to understand Convolution qualitatively by examples.

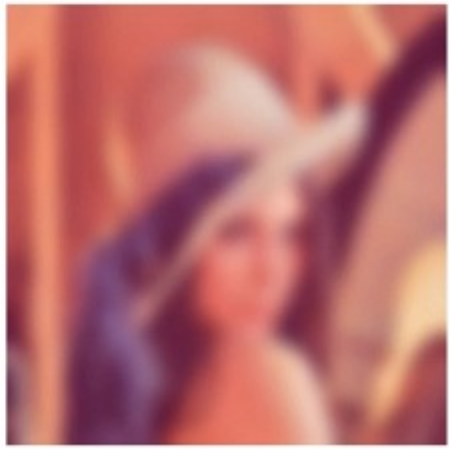
Blurring

Input Image * Filter (Kernel) = Output Image

Grandfather's spectacles



*



Understanding Convolution without Math

edge detection

Input Image * Filter (Kernel) = Output Image

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

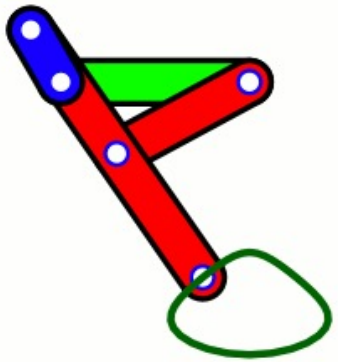
- https://www.codingame.com/playgrounds/2524/basic-image-manipulation/filtering
- https://www.javatpoint.com/dip-high-pass-vs-low-pass-filters

Basically Convolution = Image Modifier

- This is our first perspective on Convolution.
- Eg: it is merely a feature transformation on an image.
- Now the question is " What make one convolution (blur) different from another (edge detection) ?
- Answer is "The Filter".

- Next question: How do we design Filter?

The Mechanics of Convolution



Image

0	10	10	0
20	30	30	20
10	20	20	10
0	5	5	0

4x4

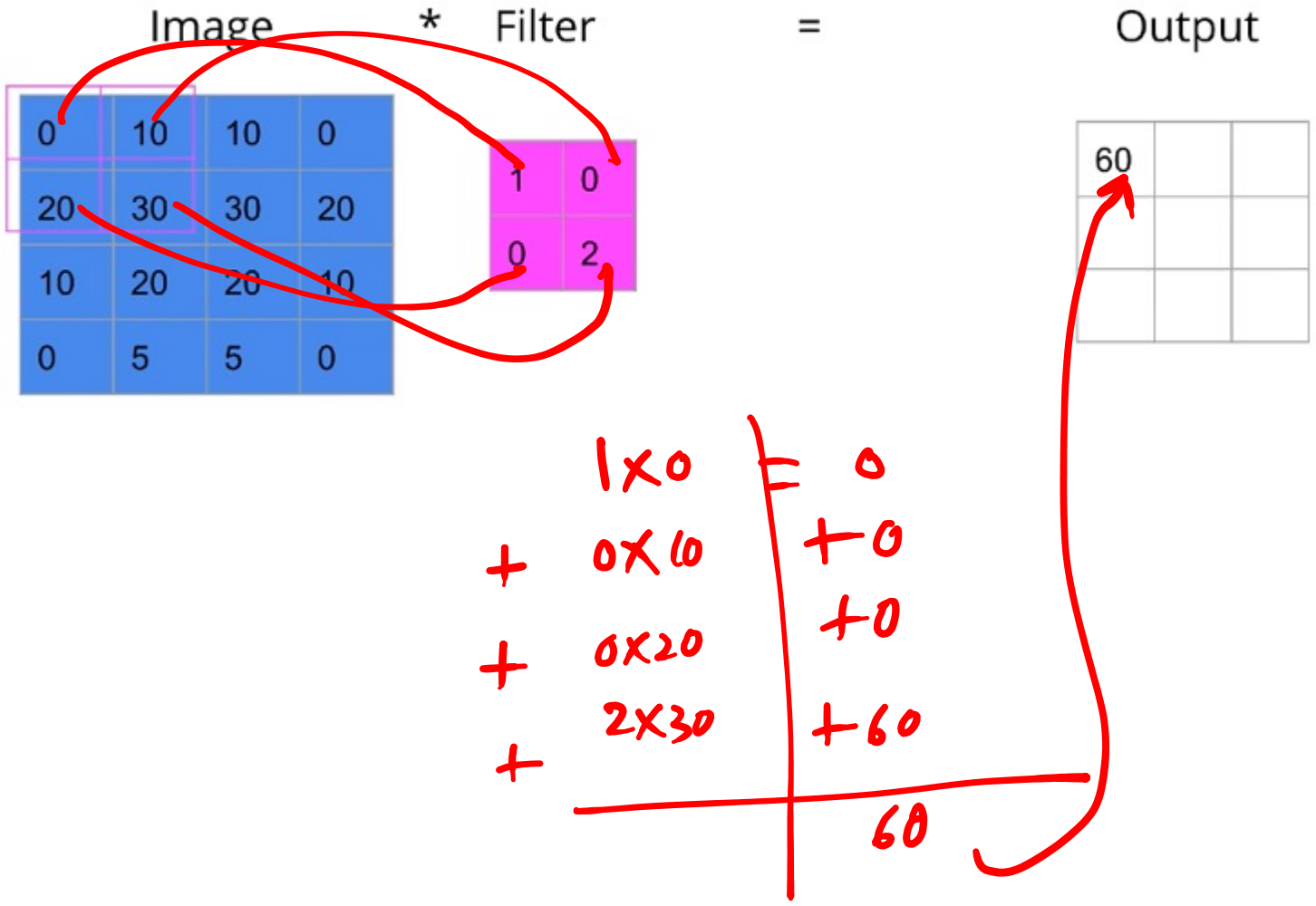
*

Filter

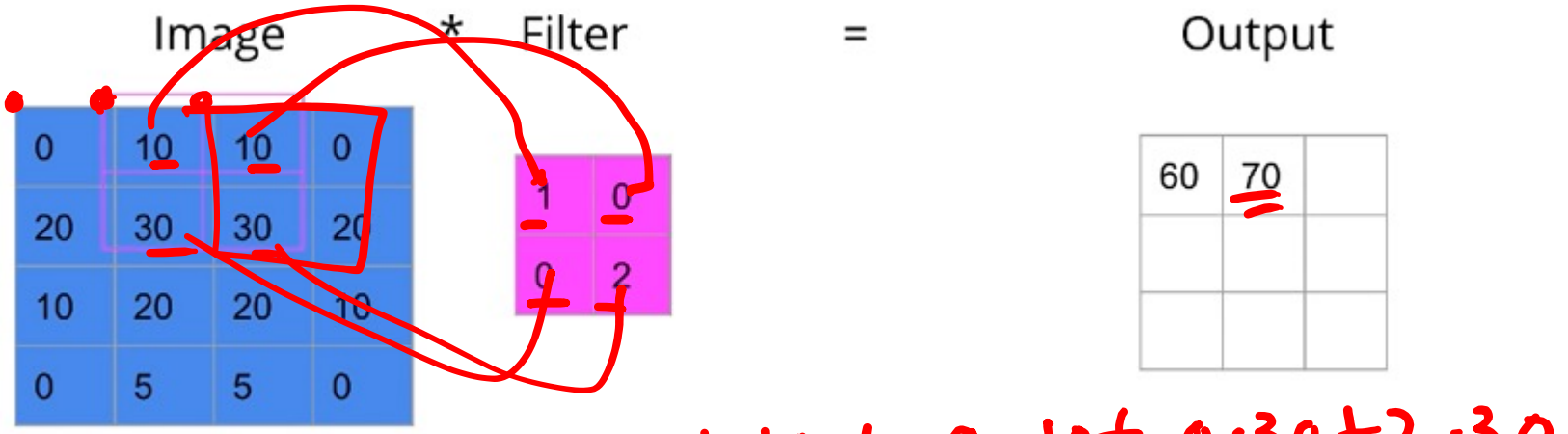
1	0
0	2

2x2

The Mechanics of Convolution



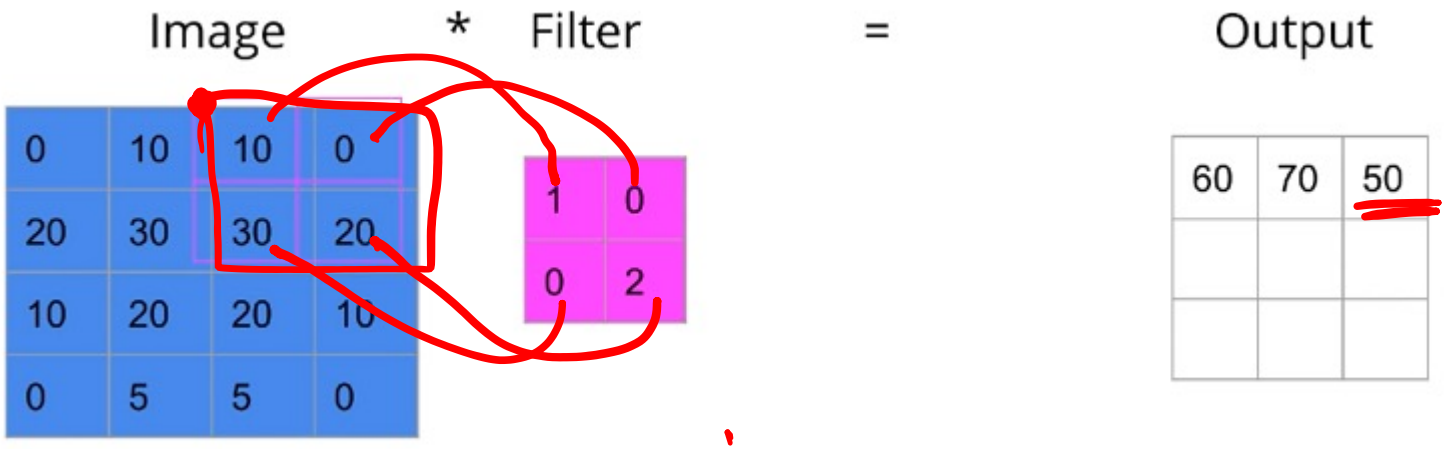
The Mechanics of Convolution



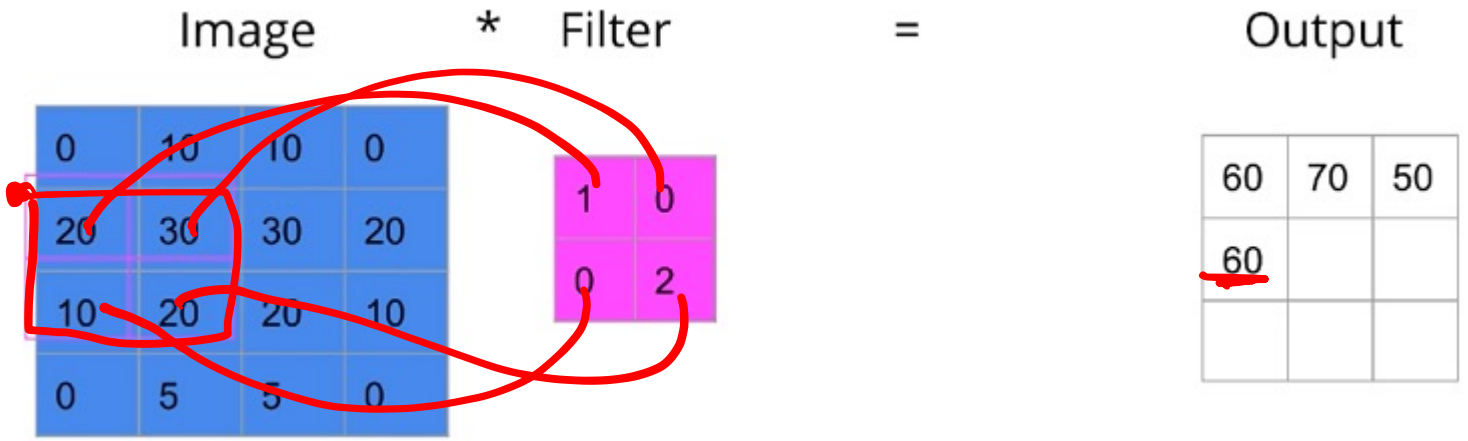
$1 \cdot 10 + 0 \cdot 10 + 0 \cdot 30 + 2 \cdot 30$

$= 70$

The Mechanics of Convolution



The Mechanics of Convolution



The Mechanics of Convolution

Try the rest yourself!

Image * Filter = Output

0	10	10	0
20	30	30	20
10	20	20	10
0	5	5	0

4x4

1	0
0	2

2x2

60	70	50
60	70	50
20	30	20

3x3

Note

Image * Filter = Output

0	10	10	0
20	30	30	20
10	20	20	10
0	5	5	0

1	0
0	2

60	70	50
60	70	50
20	30	20

3x3

Input : 4

Kernel / Filter : 2

o/p: 3

I/p = N

k = k

o/p = (N-k+1)

= 4 - 2 + 1

= 3

Note

N k

```
Given: input_image, kernel
output_height = input_height - kernel_height + 1
output_width = input_width - kernel_width + 1
```

N-k+1

N-k+1

Are images always square? No (eg:-Computer, TV screen, Mobile)

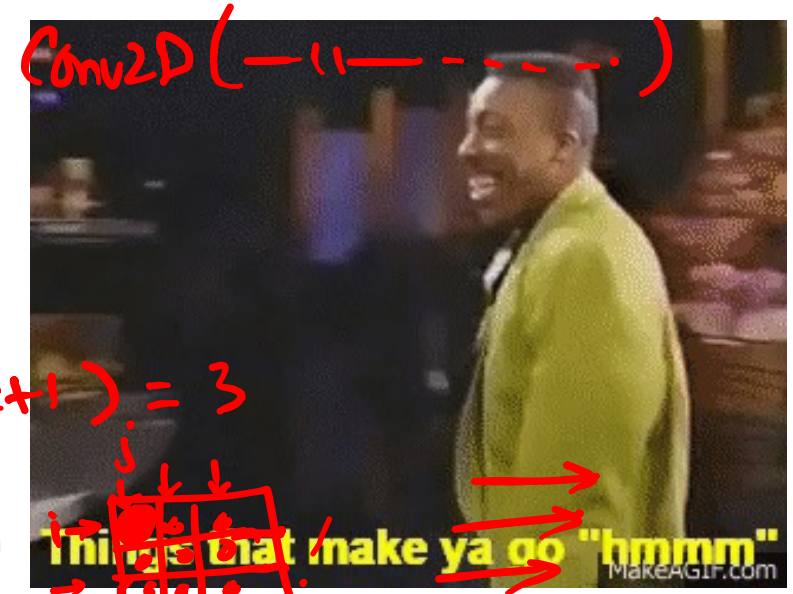
Some NN use square images for convenience. eg: MNIST

Kernels / Filters are always almost square.

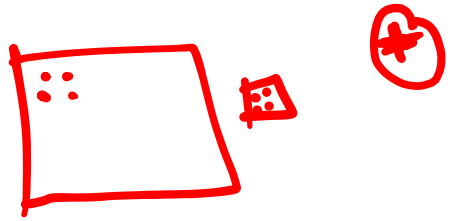
Pseudocode for Convolution

raw python code

```
Given: input_image, kernel
ouput_height = input_height - kernel_height + 1
output_width = input_width - kernel_width + 1
output_image = np.zeros((output_height, output_width))
for i in range(0, output_height):
  for j in range(0, output_width):
    for ii in range(0, kernel_height):
      for jj in range(0, kernel_width):
        output_image[i,j] += input_image[i+ii,j+jj] * kernel[ii,jj]
```



Convolution Equation



- This explains how to calculate (i,j)th entry of the output.

- Well TensorFlow does all these things for us automatically. Then why are we studying this?

- Answer: And the answer is that this will help you immensely ~~and~~ in understanding the different perspectives on convolution that we are going to discuss later.

Convolution on Wikipedia

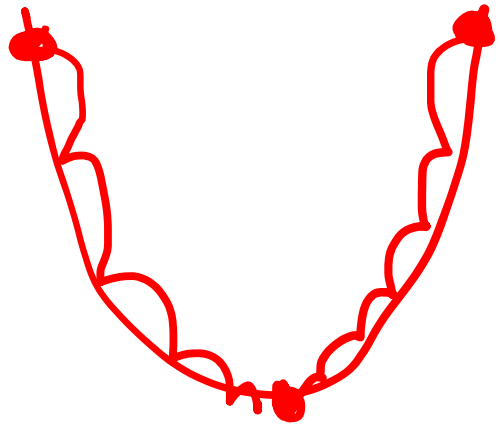
- Moreover if you're a curious person and you go on Wikipedia to read about convolution you'll see something very similar to this.

$$Z(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X(m,n) Y(i-m, j-n)$$

- In this example you can think of X as the filter y as the input image and Z as the output image.

Convolution on Wikipedia

$$Z(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X(m,n) Y(i-m, j-n)$$



- Now you might notice something weird about this equation from Wikipedia which is that instead of plus signs we have minus signs.
- Why is that?
- Am I wrong?

No. In the end, it doesn't make a difference bcz the filters will be learned using G.D (i.e. Automatically).
∴ It doesn't matter if the filter is reversed (not).

What is “mode”?

```
convolve2d(A, np.fliplr(np.flipud(w)), mode='valid')
```

- The movement of the filter is bounded by the edges of the image. The output is therefore always smaller than the input.

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

N=5

K=3

3X3

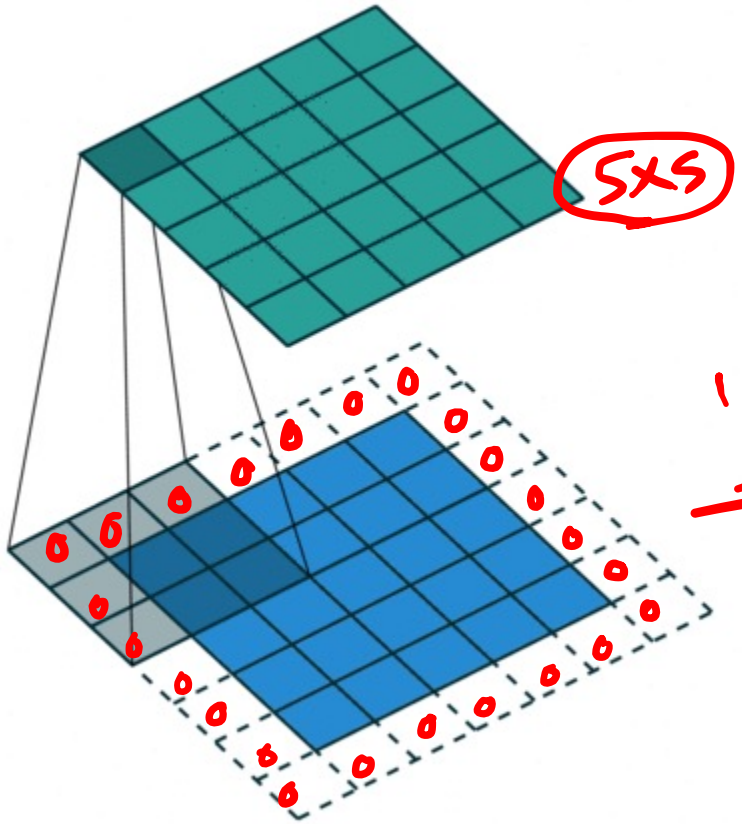
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

5-3+1

= 3

Padding (“same” mode)

- What if we want the output to be of the **same size** as the input image?
- Then we can use padding i.e. add imaginary zeros around the input.



'same'

5X5

ds (pad)

N= 7X7

k= 3

N-k+1= 7-3+1

=5

Even more padding! (“full “ mode)

- We could extend the filter further and still get non-zero outputs.
- This is not very common these days.
- “full” padding”
  i. Input Length = N
  ii. Kernel length = K
  iii. Output length = N+K-1

Summary of Modes

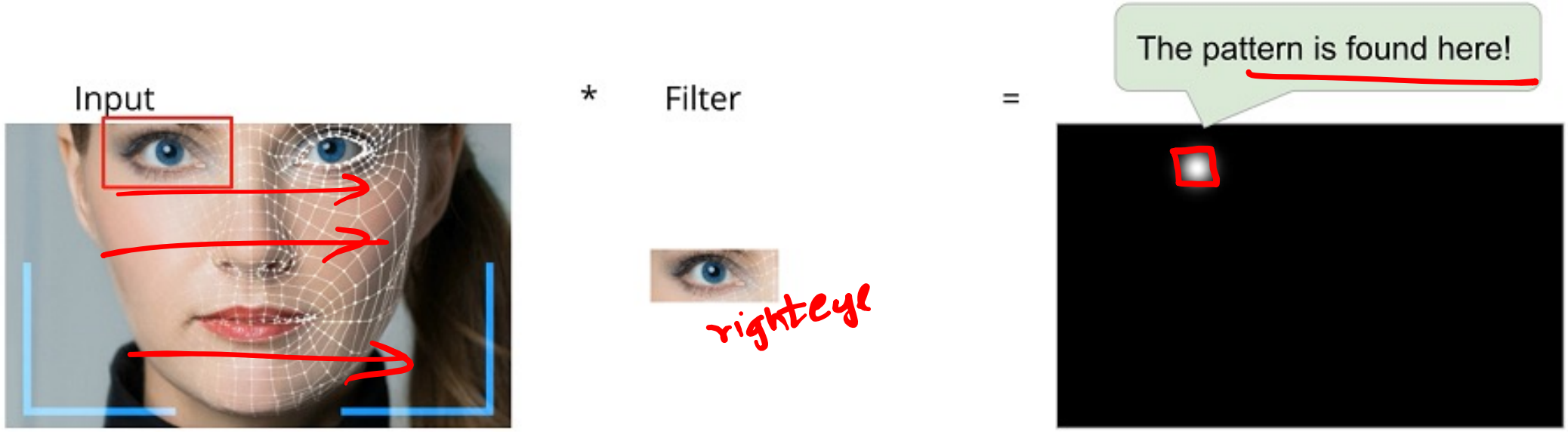
- Input length = N
- Kernel length = K

Mode	Output Size	Usage
Valid	N - K + 1	Typical
Same	N	Typical
Full	N + K - 1	Atypical

A new perspective on Convolution

https://i.insider.com/5e6f8386235c180f1533f1c2?width=800&format=jpeg&auto=webp

Pattern Finder.



How to view Convolution as Matrix Multiplication?

- Lets see 1D Convolution first because its easy to understand and 2D Convolution is just going to be a generalization of this.

Input image: $a = [a_1, a_2, a_3, a_4]$

Filter: $w = [w_1, w_2]$

Output image: $b = a * w = [a_1w_1 + a_2w_2, a_2w_1 + a_3w_2, a_3w_1 + a_4w_2]$

Convolution

$$a = (a_1, a_2, a_3, a_4)$$

$$w = (w_1, w_2)$$

$$b = a * w = (a_1w_1 + a_2w_2, a_2w_1 + a_3w_2, a_3w_1 + a_4w_2)$$

1D Convolution in general

$$b_i = \sum_{i'=1}^{K} a_{i+i'} w_{i'}$$

- **Note:** It is very same as 2D Convolution's equation, just without 2nd index

Matrix Multiplication

- Can we do the same operation using Matrix Multiplication.
- YES!!!
- HOW????????

Transpose

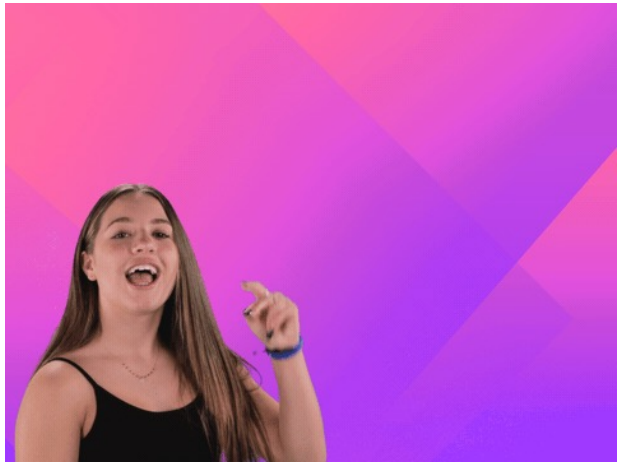
$$\begin{pmatrix} a_1w_1 + a_2w_2 \\ a_2w_1 + a_3w_2 \\ a_3w_1 + a_4w_2 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & 0 & 0 \\ 0 & w_1 & w_2 & 0 \\ 0 & 0 & w_1 & w_2 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

b w a

So what do we understand?

- By repeating the same filter again and again, we can do convolution without actually doing convolution.

$$\begin{pmatrix} a_1w_1 + a_2w_2 \\ a_2w_1 + a_3w_2 \\ a_3w_1 + a_4w_2 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & 0 & 0 \\ 0 & w_1 & w_2 & 0 \\ 0 & 0 & w_1 & w_2 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$



Problem with Matrix Multiplication?

Problem with Matrix Multiplication?

① Repeat filter multiplications.

② ∴ Result matrix takes up a lot more space than original filter which was just 2 element array originally.

$$\begin{pmatrix} a_1w_1 + a_2w_2 \\ a_2w_1 + a_3w_2 \\ a_3w_1 + a_4w_2 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & 0 & 0 \\ 0 & w_1 & w_2 & 0 \\ 0 & 0 & w_1 & w_2 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

3×4

Lets replace Matrix Multiplication with Convolution

Big & slow

Small & fast.

Parameter Sharing / Weight Sharing

Inside a neuron,

$$a = w^T \cdot x$$

where a = o/p activation

x = input feature vector

w = weight matrix.

If we use the same weights again, we could have less parameters, use up less RAM, & make the computation more efficient.

Why do this?

Consider a fully connected ANN.

eg: MNIST. Images r Grayscale. 28X28 = 784 features
i.e. 784 sized input vector.

eg: Larger image in color
CIFAR-10: 32X32X3 = 3072-sized input vector.

Note: 32X32 is very modest size.

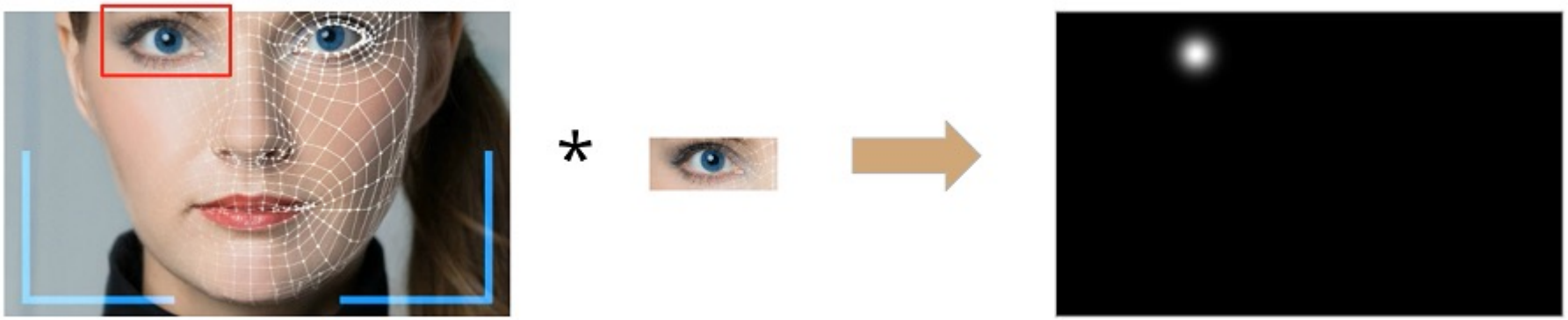
eg: Modern CNN such as VGG: (Images of size 224X224, If u use a full weight matrix then u would hv 1,50,528 features.

eg: Modern HD images with resolution 1280X720 = 2.8 million features.

Why do this?

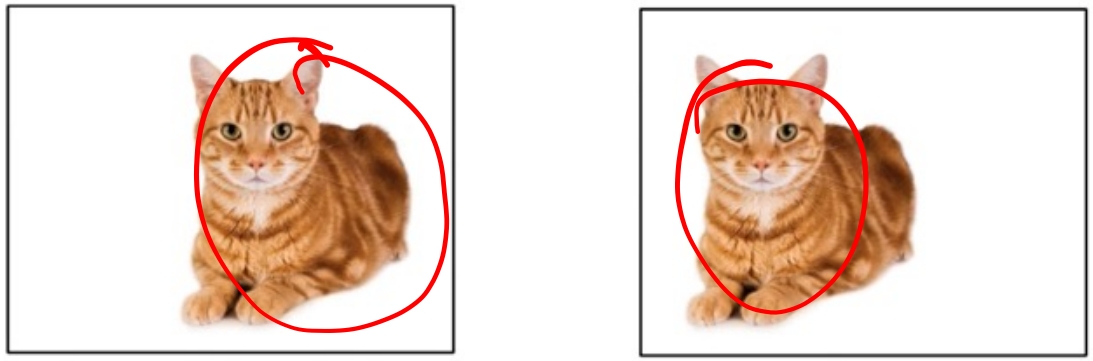
Convolution is actually Correlation & filter is a pattern finder.

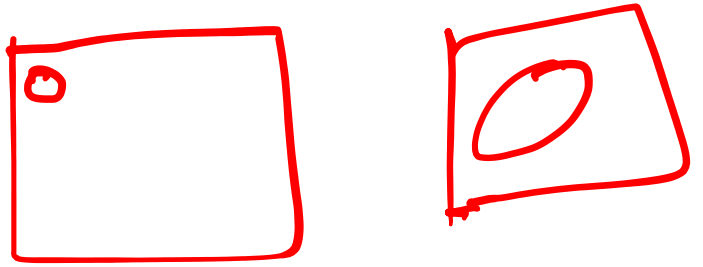
∴ we want the same filter to look at all locations on the image i.e. Translational Invariance.



Translational Invariance

- Suppose we are building a Dog vs cat classifier





Only difference is cat is in a different position.

Translational Invariance

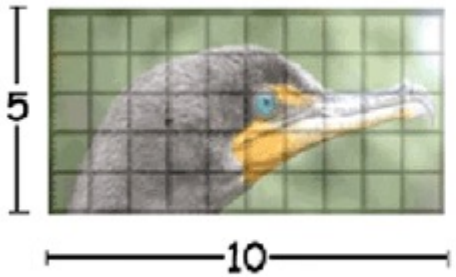
Cat

Cat

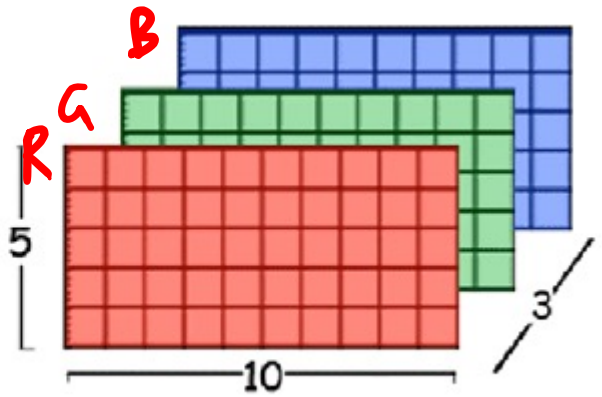
Shared pattern finder.—

Convolution on Color Images

Images are 3D objects: Height x Width x Color

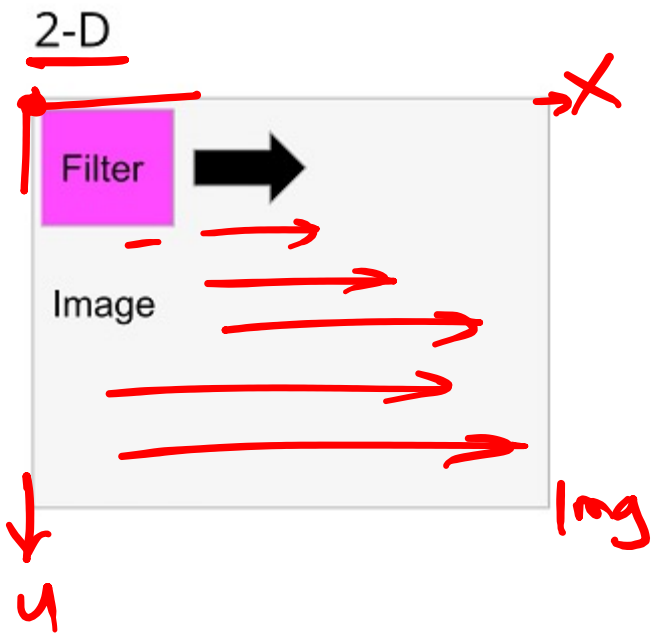


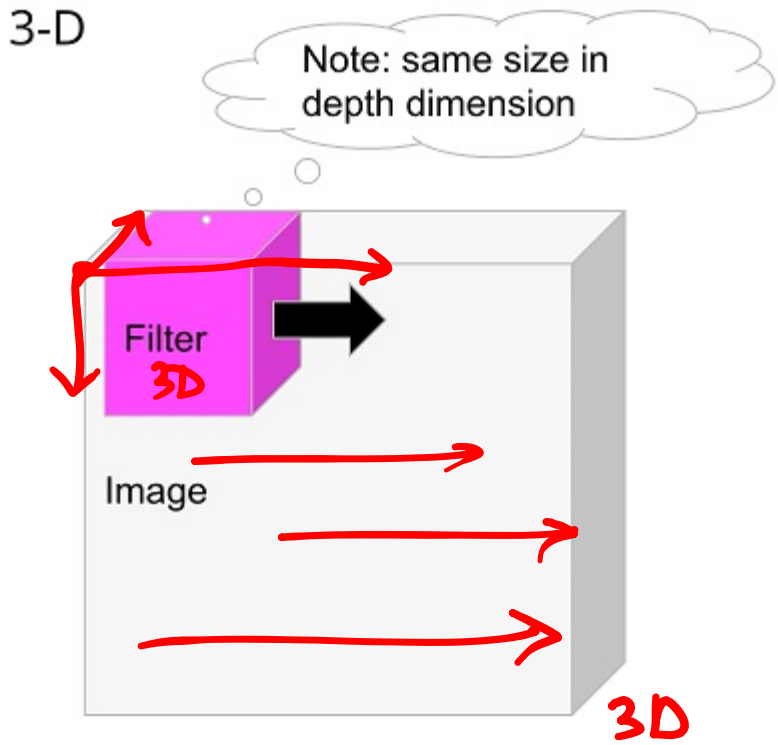
Original Color Image



Matlab RGB Matrix

Convolution on Color Images

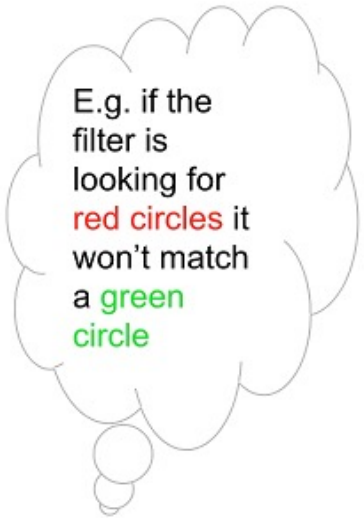




Convolution on Color Images

2-D (2-D "dot product") - a grayscale pattern-finder

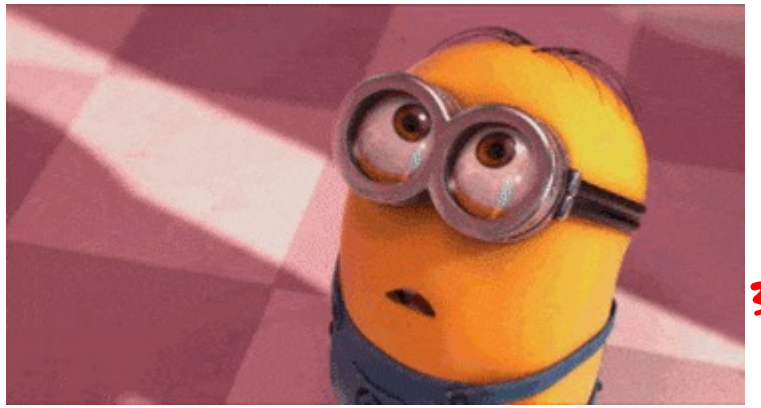
$$(A * w)_{ij} = \sum_{i'=1}^{K} \sum_{j'=1}^{K} A(i+i', j+j') w(i', j')$$



3-D (3-D "dot product") - a color pattern-finder

$$(A * w)_{ij} = \sum_{c=1}^{3} \sum_{i'=1}^{K} \sum_{j'=1}^{K} A(i+i', j+j', c) w(i', j', c)$$

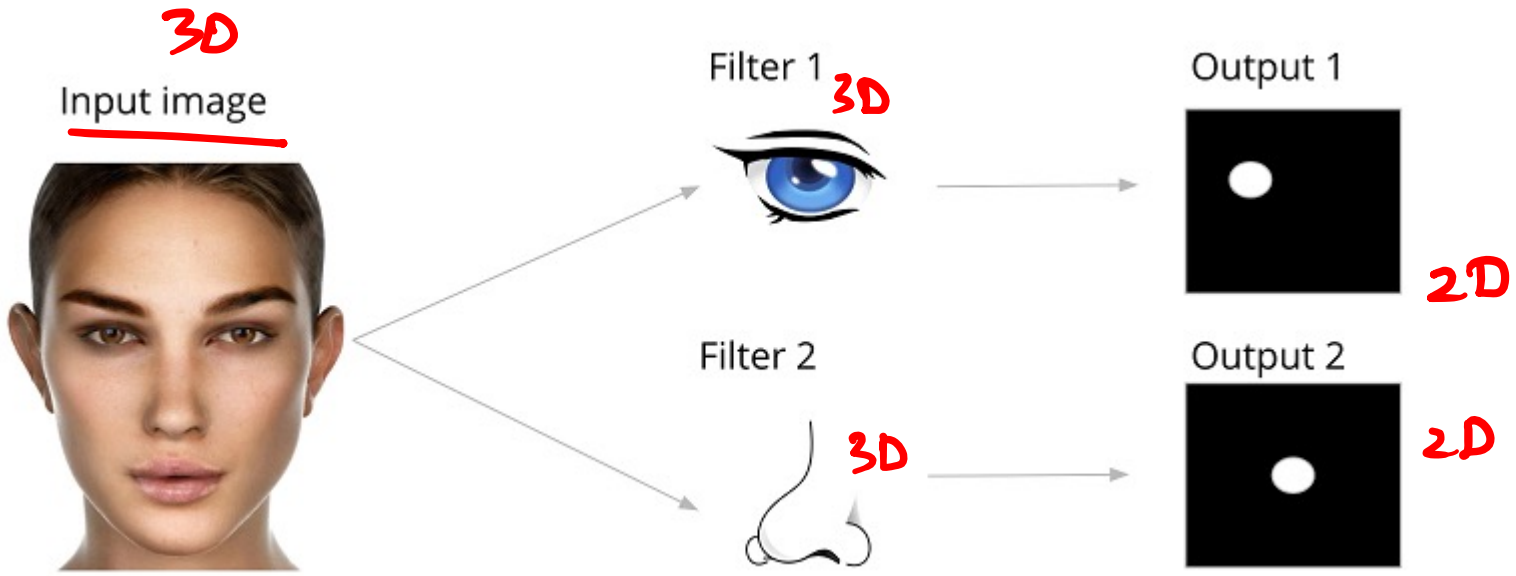
What more?



- Input Image: H x W x 3, which is a 3D vector
- Kernel: K x K x 3, which is a 3D vector
- Output Image: (H-K+1) x (W-K+1), which is a 2D vector
- We know that Neural Networks have a repeating structures (i.e. they have “uniformity”)
- Ex: For a Dense layer, if the input is 1D, then the output is also 1D and hence can be fed from one layer to another.
- But, in our case, we see that the output is 2D and input is 3D.
- Question: So how do we solve this?
- Answer: Lets see this now.

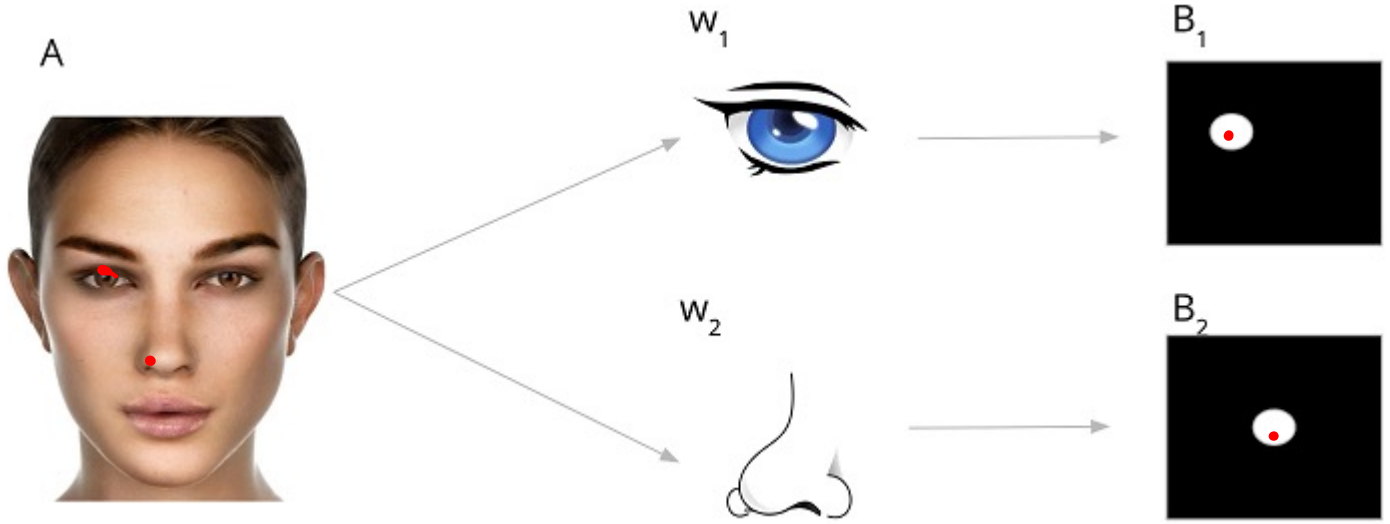
$$(A * w)_{ij} = \sum_{c=1}^{3} \sum_{i'=1}^{K} \sum_{j'=1}^{K} A(i+i', j+j', c) w(i', j', c)$$

Multiple Features

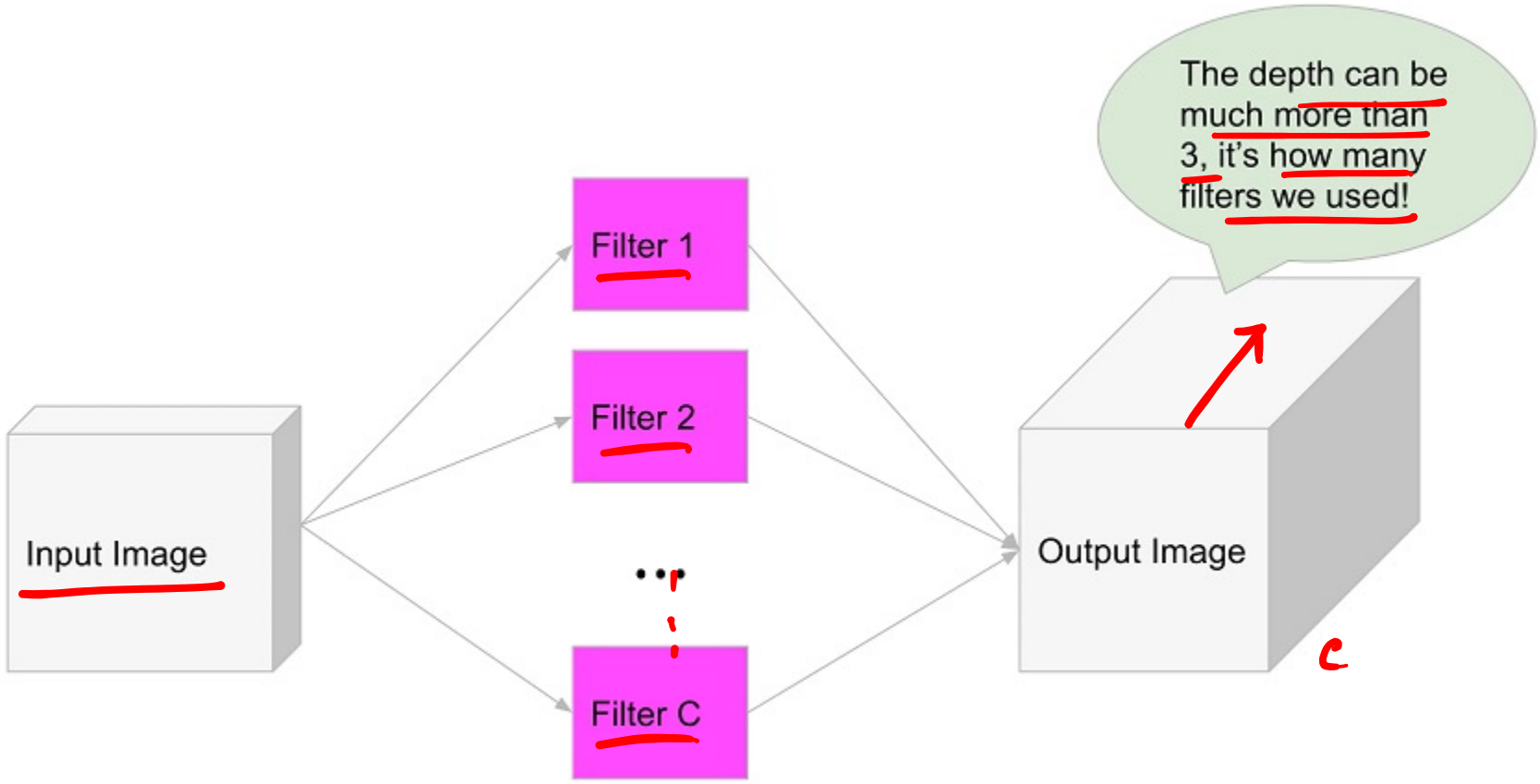


Multiple Features

- Consider we have an image A with dimensions: H x W x 3
- If we use the “same” mode, and apply one filter, then we get output, lets say B1 with dimensions: H x W
- If we use the “same” mode, and apply one more filter, then we get output, lets say B2 with dimensions: H x W
- If we stack up B1 and B2 together, we get a new output image, lets say B with dimensions: H x W x 2, i.e. a 3D image as our original image



Multiple Features



Convolution in Deep Neural Network

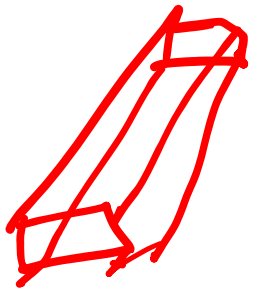
- Lets vectorize this operation, we don't need to do each color convolution separately

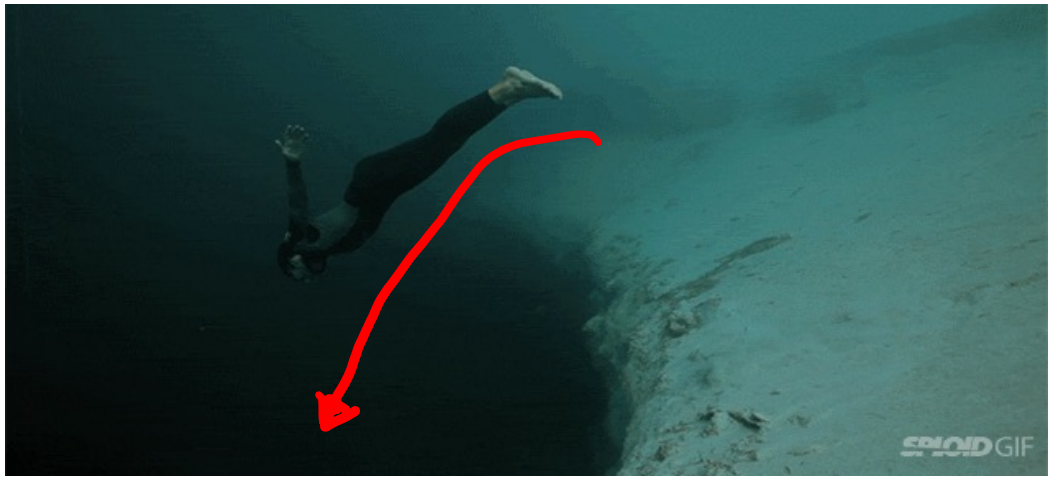
$$B = A * w$$

$$shape(A) = H \times W \times C_1$$

$$shape(w) = C_1 \times K \times K \times C_2$$

$$shape(B) = H \times W \times C_2$$





$$B(i,j,c) = \sum_{i'=1}^{K} \sum_{j'=1}^{K} \sum_{c'=1}^{C_1} A(i+i', j+j', c') w(c', i', j', c)$$

So What is this 3rd dimension in the Output image?

Its certainly not the color as it was in the input img.

H x W x (arbitary #)

no. of feature maps / no. of Channels

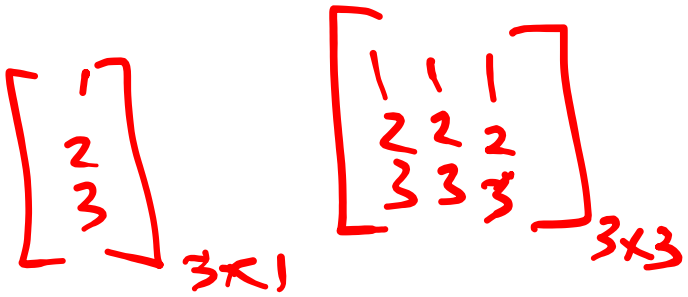
Convolution Layer

$$Conv\ layer: \ \sigma(W * x + b)$$

$$Dense\ layer: \ \sigma(W^T x + b)$$

Shape of the bias

- In a Dense layer, if $W^T x$ is a vector of size M, b is also a vector of size M
- In a Conv layer, b **does not** have the same shape as $W * x$ (a 3-D image)
- Technically, this is not allowed by the rules of matrix arithmetic
- But the rules of broadcasting (in Numpy code) allow it
- If $W * x$ has the shape H x W x C_2, then b is a vector of size C_2
  - One scalar per feature map



$$Conv\ layer: \ \sigma(W * x + b)$$

$$Dense\ layer: \ \sigma(W^T x + b)$$

How much do we save?

32-5+1 = 28

CONVOLUTION!

Input Img: 32 x 32 x 3

Filter : 3 x 5 x 5 x 64 (i.e. 64 feature maps)

o/p: 28 x 28 x 64 (assuming 'valid' mode)

∴ # parameters (ignore bias) = 3 x 5 x 5 x 64 = 4800

How much do we save?

MATRIX MULT:

flattened I/p Img: $32 \times 32 \times 3 = 3072$

Flattened o/p Vector $= 28 \times 28 \times 64 = 50176$

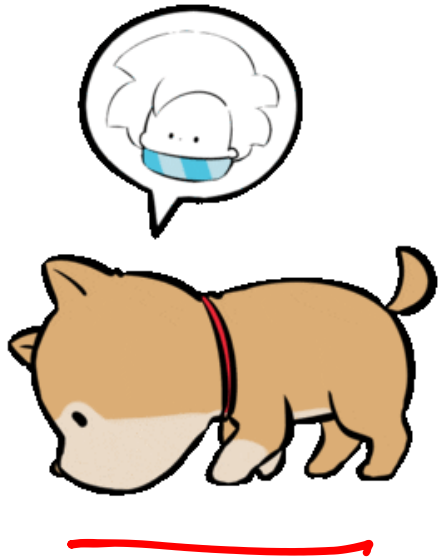
Weight Matrix: $3072 \times 50176 = 154,140,672 \approx 154$ million.

$\Rightarrow$ Compared to convolution, we hv ≈ 32000 times more parameters!

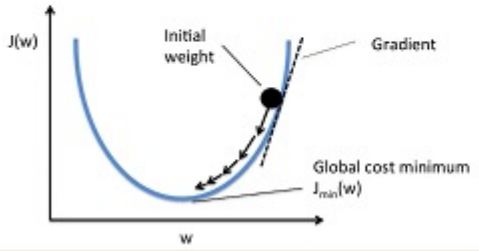
↓ high RAM

↓ Suboptimal performance.

How are Convolution filters found?



- Since convolution is just a part of some neural network layer, it's easy to conceive of how the filters will be found
- Initially, we looked at convolution as an image *modifier* (blur, edge)
- Now, we see it as a pattern finder / shared-parameter matrix multiplication / feature transformer
- In other words, W will be found the same as before, automatically!
- Still gradient descent, `model.fit()`



$$Conv\ layer: \ \sigma(W * x + b)$$

$$Dense\ layer: \ \sigma(W^T x + b)$$