What is "mode"?

valid | same | full

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

i/p

$\boxed{N = 5}$

$K = 3$

$N-K+1$

$= 5-3+1$

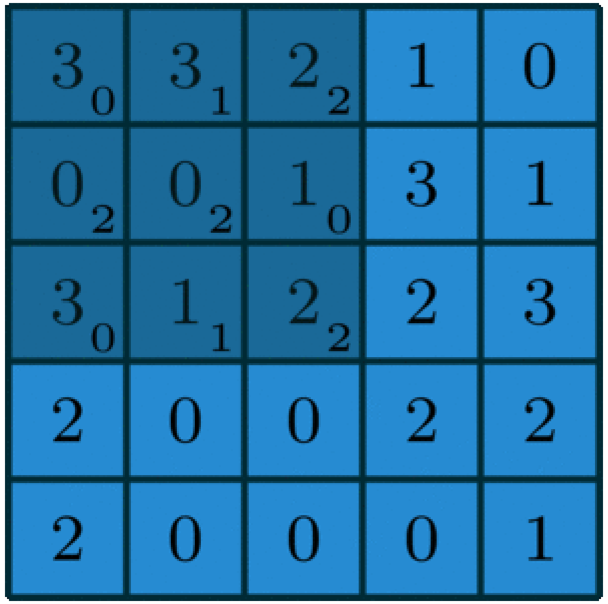
$= \boxed{3}$

o/p shrunk in size

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

Filter?

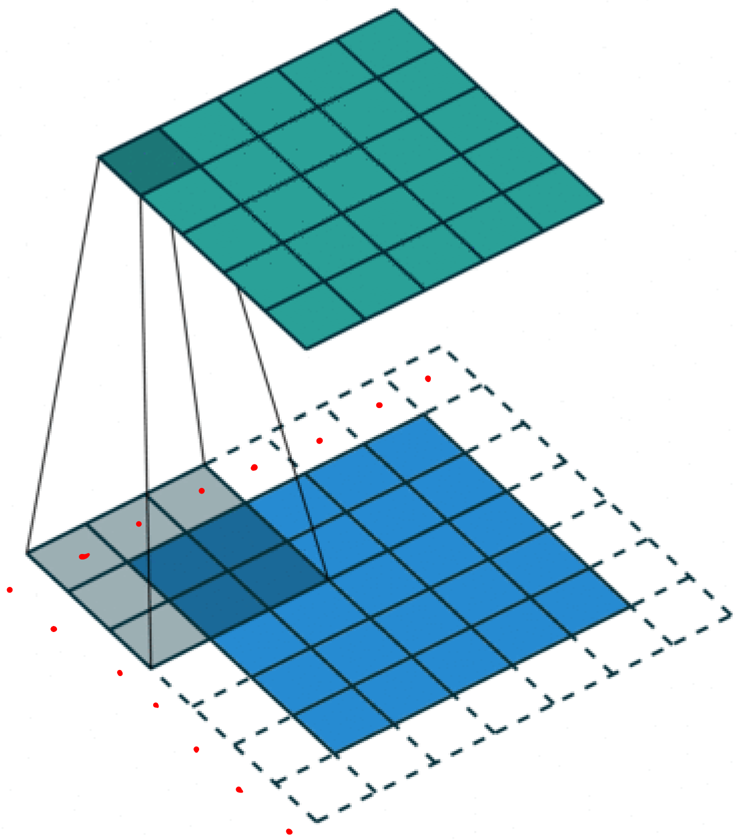
0	1	2
0	1	2
0	1	2



Output Image

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.



Input = N = 5 (→ 7)

Filter = K = 3

o/p = 5

N-K+1

= 7 - 3 + 1

= ~~3~~ 4 + 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”
  1. Input Length = N   5
  2. Kernel length = K   3
  3. Output length = N+K-1 $5+3-1$ $= 8-1$ $= 7$

Summary of Modes

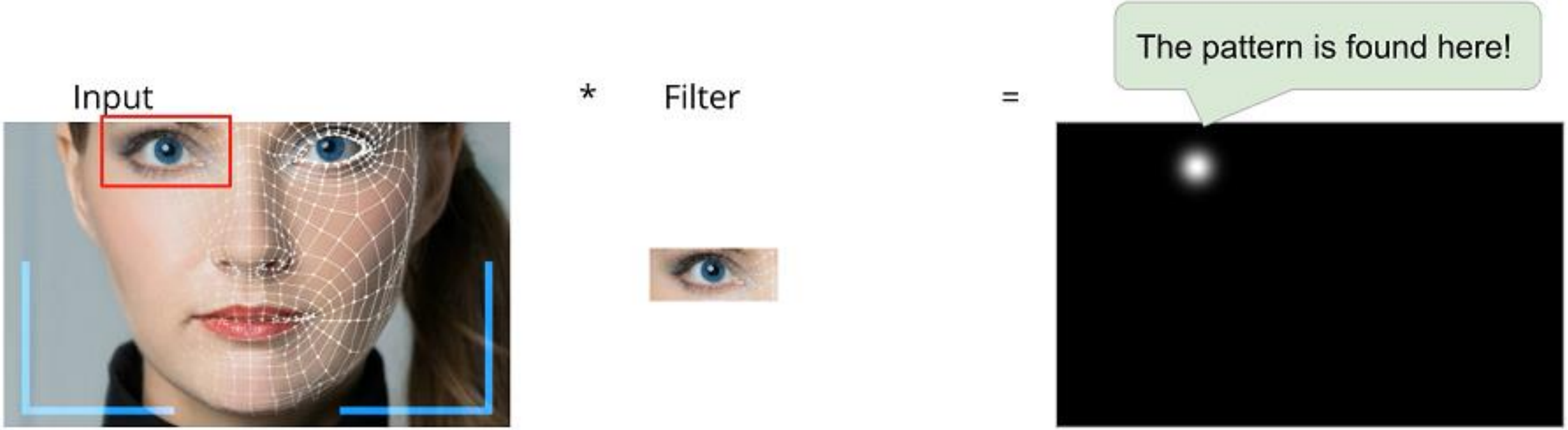
- Input length = N
- Kernel length = K

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

A new perspective on Convolution

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

Sliding Pattern Finder that passes through the entire image.



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

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

$$w = (w_1, w_2)$$

CONVOLUTION

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

1D Convolution in general

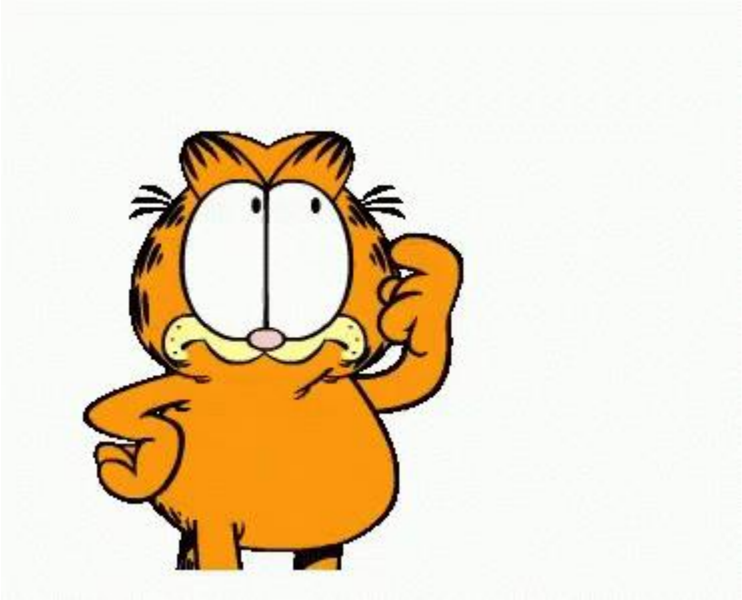
$$b_i = \sum_{i'=1}^{K} a_{i+i'} w_{i'}$$

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

Matrix Multiplication

1D Array

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



Matrix Multiplication

Kernel * I/p

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

b w a

So what do we understand?

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

$$\begin{pmatrix} a_1 w_1 + a_2 w_2 \\ a_2 w_1 + a_3 w_2 \\ a_3 w_1 + a_4 w_2 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & 0 & 0 \\ 0 & w_1 & w_2 & 0 \\ 0 & 0 & w_1 & w_2 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

Trainer: Dr. Darshan Ingle.

Problem with Matrix Multiplication?

Problem with Matrix Multiplication?

1. filter gets repeated multiple times.
2. It takes up a lot more space
3. Filter w was vector of size 2.
   In Matrix mult, w becomes a 2D Matrix of size $\underline{\underline{3 \times 4}}$.

$$\begin{pmatrix} a_1 w_1 + a_2 w_2 \\ a_2 w_1 + a_3 w_2 \\ a_3 w_1 + a_4 w_2 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & 0 & 0 \\ 0 & w_1 & w_2 & 0 \\ 0 & 0 & w_1 & w_2 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

Lets replace Matrix Multiplication with Convolution

MATRIX Mult. is BIG & SLOW.

∴ We will go for Convolution which is SMALL & FAST.

Parameter Sharing / Weight Sharing

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

whr a = o/p Activation

x = i/p feature vector

W = weight matrix.

~~Whn~~ What if I use the same weights (w_1, w_2) again & again?

ADV:

1. less parameters
2. use lesser RAM
3. Computation becomes far more efficient.

∴ Convolution ~~takes~~ saves both SPACE & TIME by using less weights.

Why do this?

Consider a fully connected ANN

Ex: MNIST dataset

$28 \times 28 = 784$-sized input vector

what if the image size is slightly larger & it is a color image?.

Ex: CIFAR-10 : $32 \times 32 \times 3 = 3072$-sized input vector.

Modern CNN such VGG : 224×224

If we are using full weight matrix, the # i/p features = 1,50,528 features

Modern HD image : 1280×720 = 2.8 million features.

Too Large for a NN.

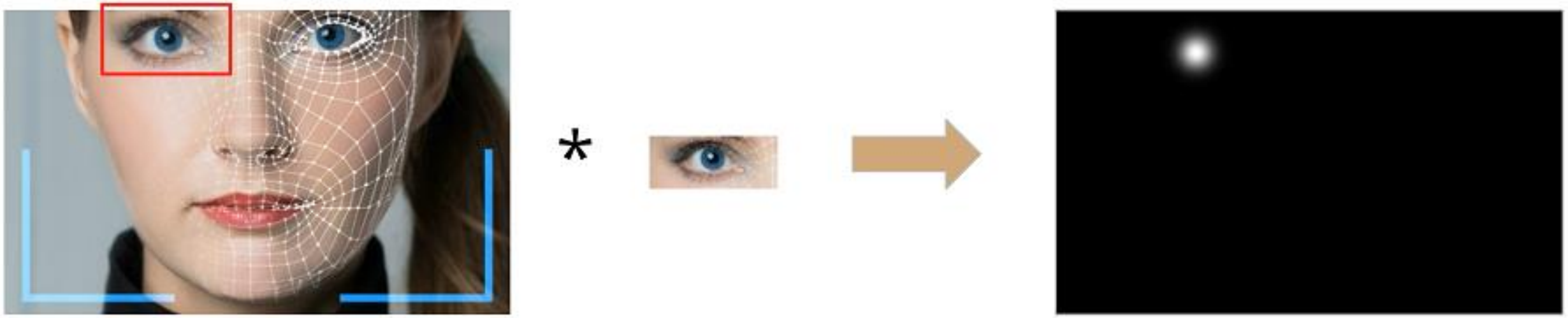
Why do this?

Convolution ≡ Corelation.

FILTER ≡ Pattern Finder.

We want the same filter to look at all locations on the image.

↳ This is called as Translational Invariance.



Translational Invariance

- Suppose we are building a Dog vs cat classifier

Cat

Cat

If we use a fully connected NN, then the NN has to learn the weights for each of these positions separately.

& yet after that, NN wont generalize that well bcz if we come across the same cat in a different position, NN shall fail to recognize it.

Translational Invariance

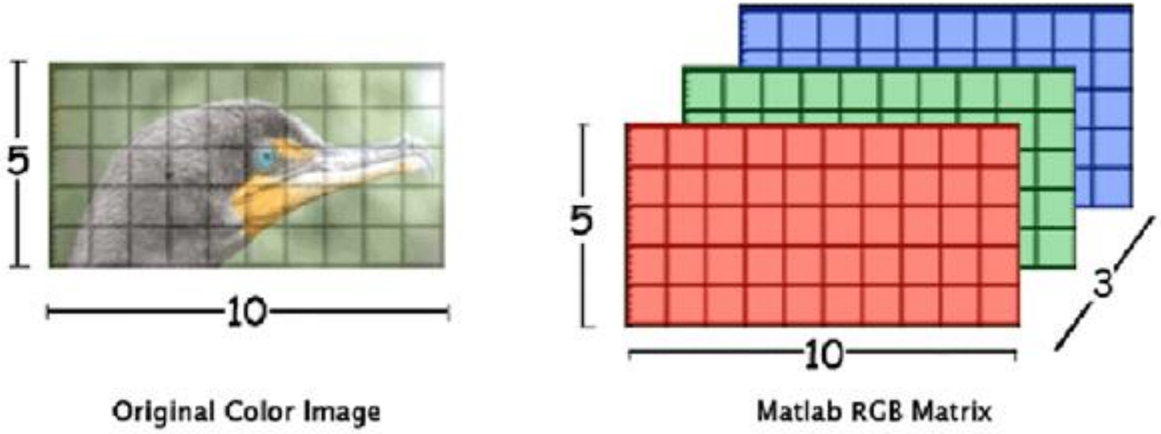
Cat

Cat

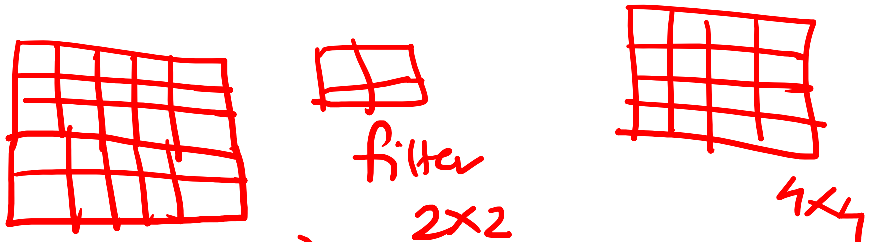
∴ Having a Shared pattern finder makes more sense.

Convolution on Color Images

Color: 3D : H × W × C (Ht × Width × Color)



Convolution on Color Images

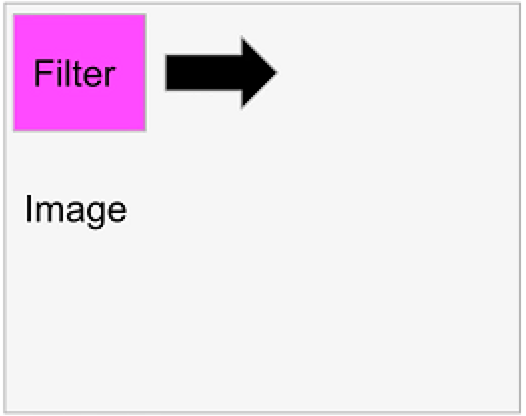


Input (5x5)

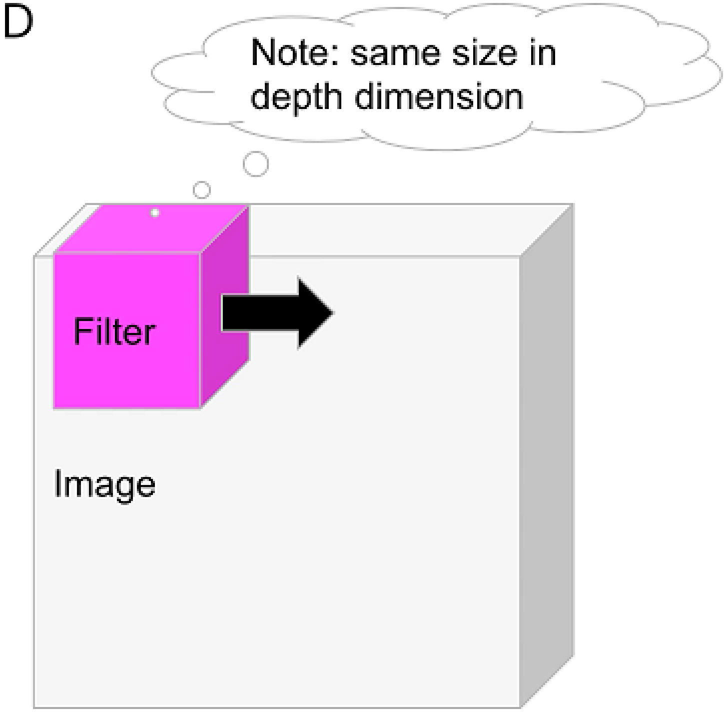
filter 2x2

4x4

2-D



3-D



o/p Image

$N-k+1$

$5-2+1$

$= 4$

Convolution on Color Images

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

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

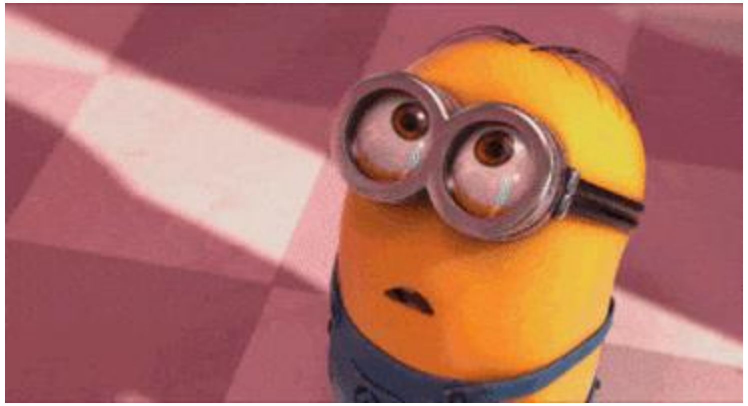
E.g. if the filter is looking for red circles it won't match a green circle

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?

3D



2D

I/P = N (2D)

Filter = K

O/P: N-K+1 (2D)

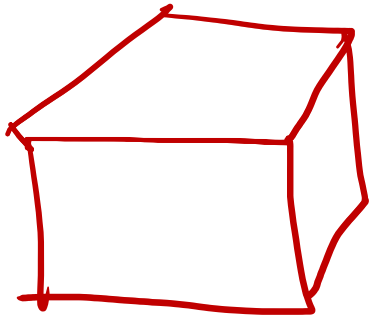
- Input Image: H x W x 3, which is a 3D vector
- Kernel: K x K x 3, which is a 3D vector

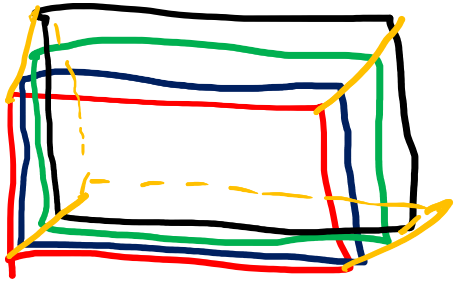
$\boxed{\begin{array}{l} \text{I/P} = 3D \\ \text{O/P} = 2D \end{array}}$

- 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.

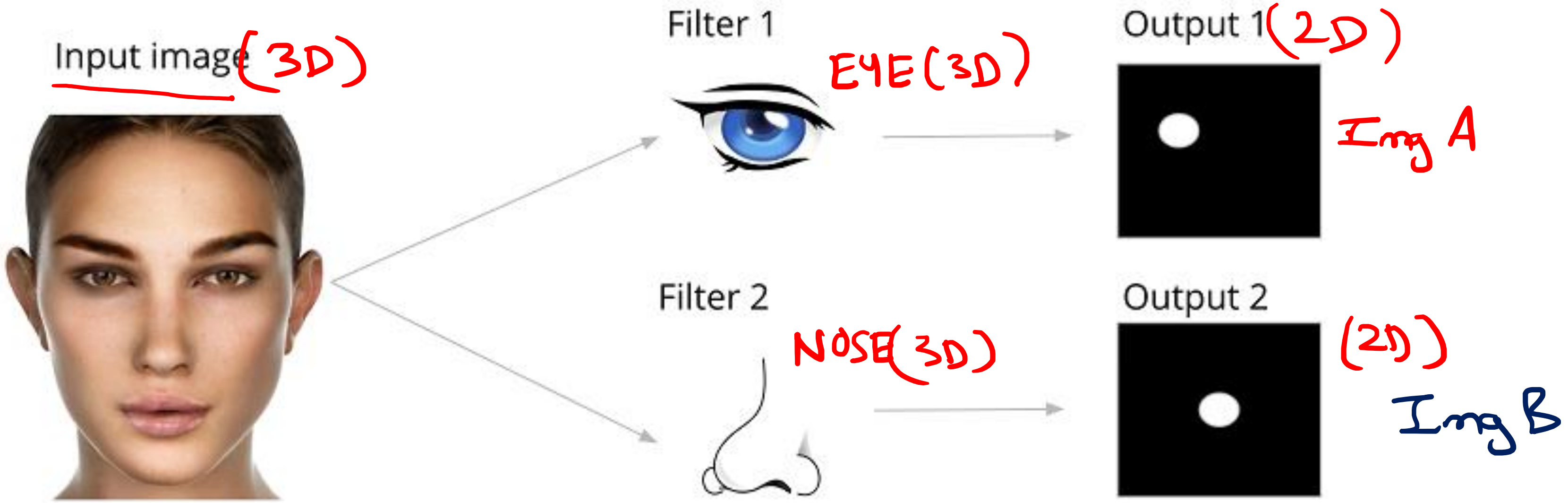
$$(A * w)_{ij} = \sum_{c=1}^{3} \sum_{i'=1}^{K} \sum_{j'=1}^{K} A(i+i', j+j', c)w(i', j', c)$$

Multiple Features



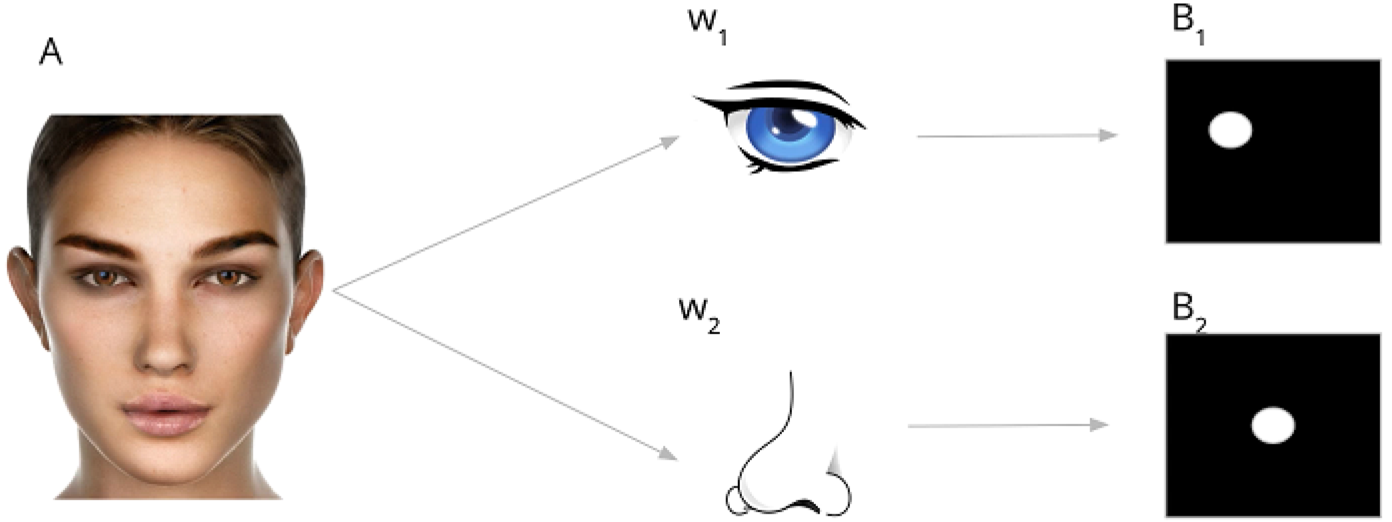


eye, nose, lips,
eyebraws

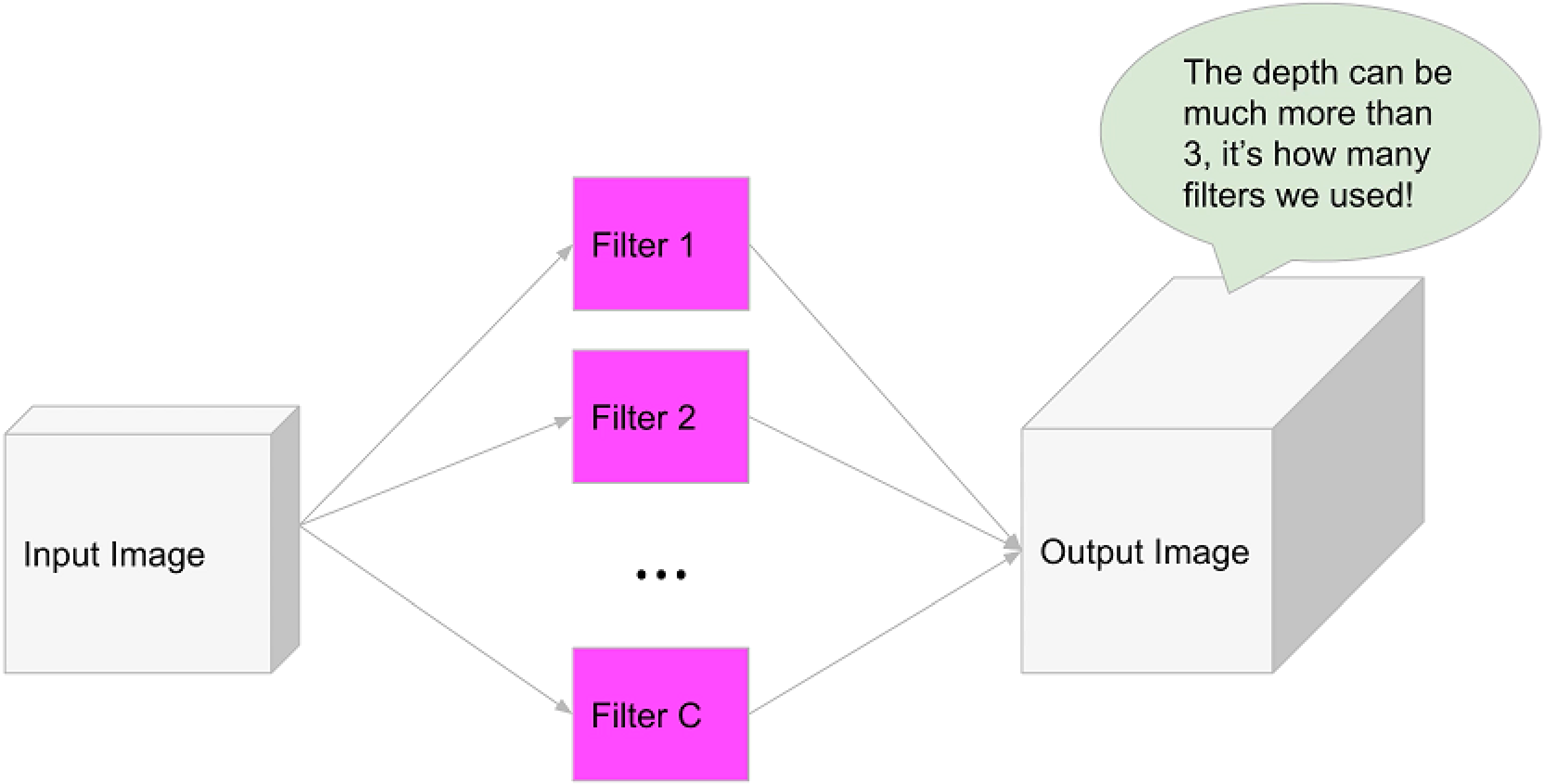


Multiple Features

- Consider we have an image A with dimensions: H x W x 3 3D
- If we use the "same" mode, and apply one filter, then we get output, lets say B1 with dimensions: H x W 2D
  eye
- If we use the "same" mode, and apply one more filter, then we get output, lets say B2 with dimensions: H x W 2D
  nose
- 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

$$B = \overset{\text{convolve}}{A * w}$$

A = Input Img , w = filter

B = Output Img

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

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

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

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

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

3rd is certainly not the color.

I/p to NN : $H \times W \times 3$

After subsequent convolutions, we get $H \times W \times$ (arbitary #)

3rd dimension = feature maps.

Convolution Layer

convolution

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

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

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 $H \times W \times C_2$, 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?

CONVOLUTION

Has convolution saved my time & space?

Input Img: $32 \times 32 \times 3$

Filter: $3 \times 5 \times 5 \times 64$ (i.e. 64 feature maps)

Output Img: $28 \times 28 \times 64$ (assuming mode = 'valid')

why 28? $N-k+1$

$H-k+1 = 32-5+1 = 28$

$\therefore$ # of parameters (ignore the bias) $= 3 \times 5 \times 5 \times 64 = 4800$

How much do we save?

MATRIX MULTIPLICATION

Flattened Input Img = $32 \times 32 \times 3 = 3072$

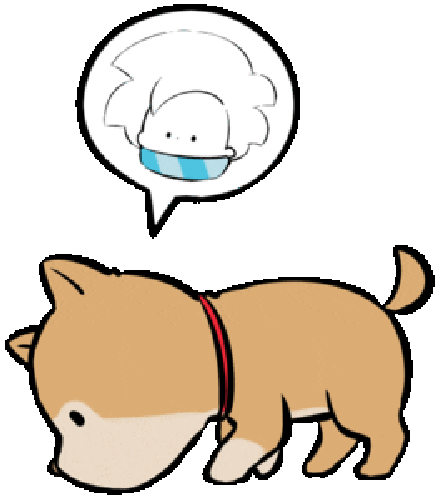
— " — O/p Img = $28 \times 28 \times 64 = 50176$

Weights Matrix = $3072 \times 50176 = 1,54,140,672 \approx 154$ million

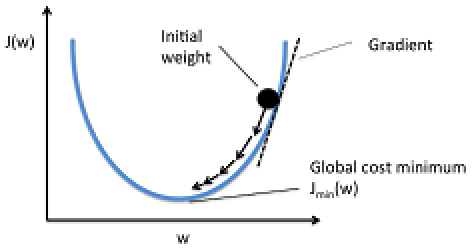
Compared to Convolution, we see that it takes ~ 32000 times more parameters.

$\therefore$ Huge RAM. $\rightarrow$ Suboptimal

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`()

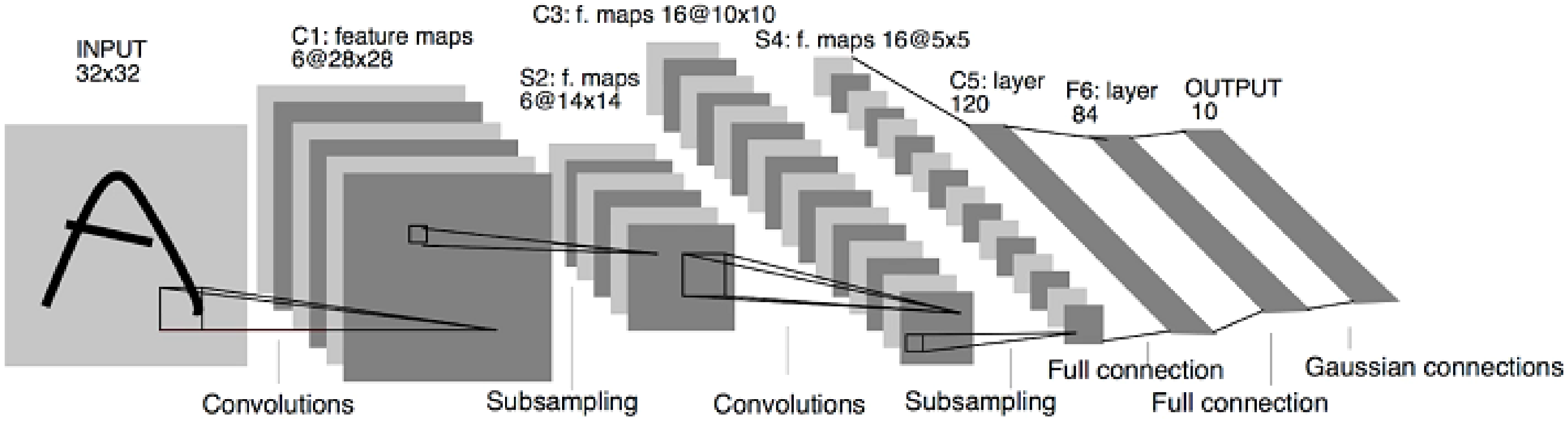


$$Conv\ layer: \ \sigma(W * x + b)$$
$$Dense\ layer: \ \sigma(W^T x + b)$$

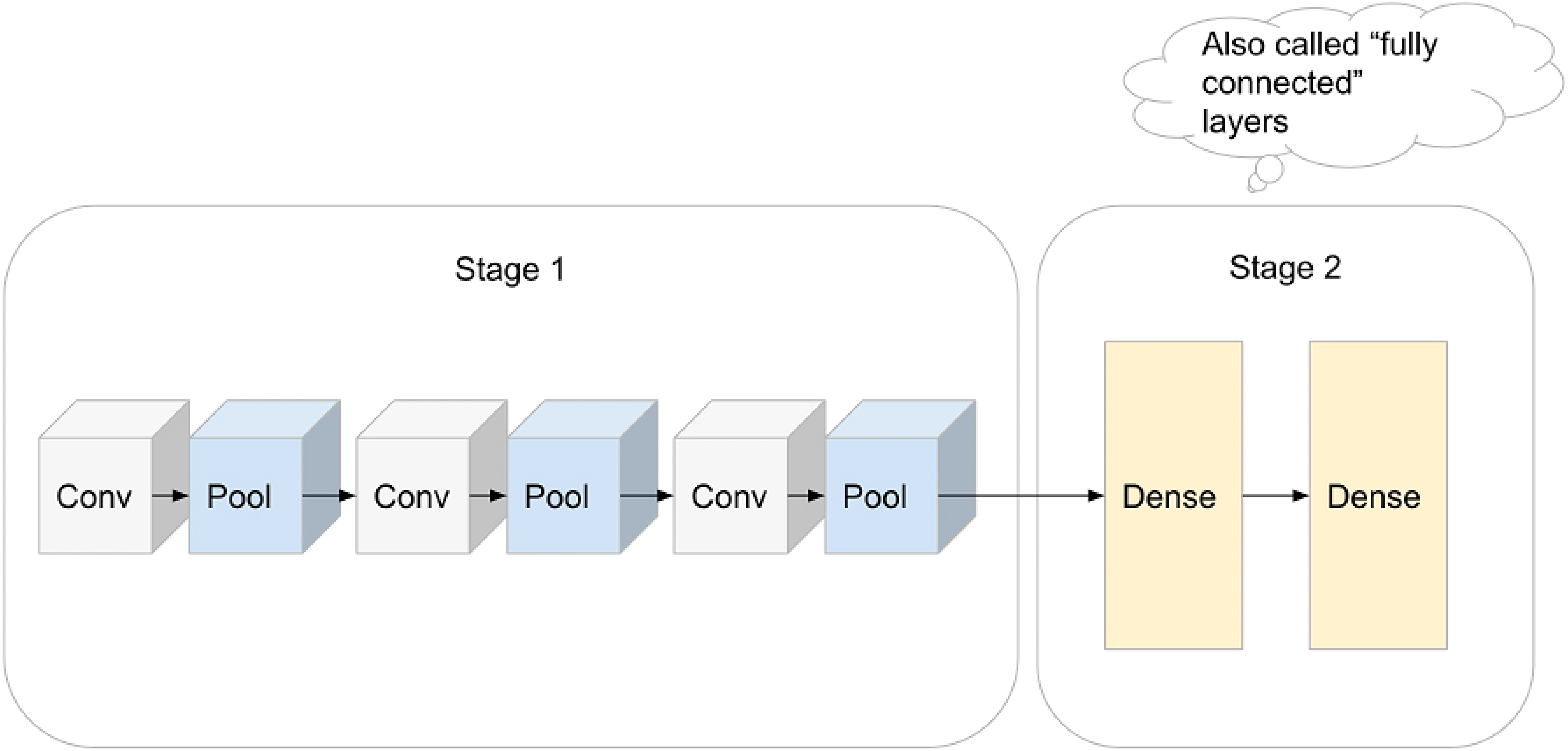
CNN Architecture

CNN Architecture

- Now that you understand how exactly a convolution layer works including the bias term and activation function we can now consider the architecture of a convolution~~or~~al neural network and why it's that way.
- So as a little bit of a history lesson, modern CNN is essentially all originated from the same model, the LeNet.
- This is named after Yann LeCun, one of the original Deep Learning pioneers, along with Geoff Hinton and Yoshua Bengio.

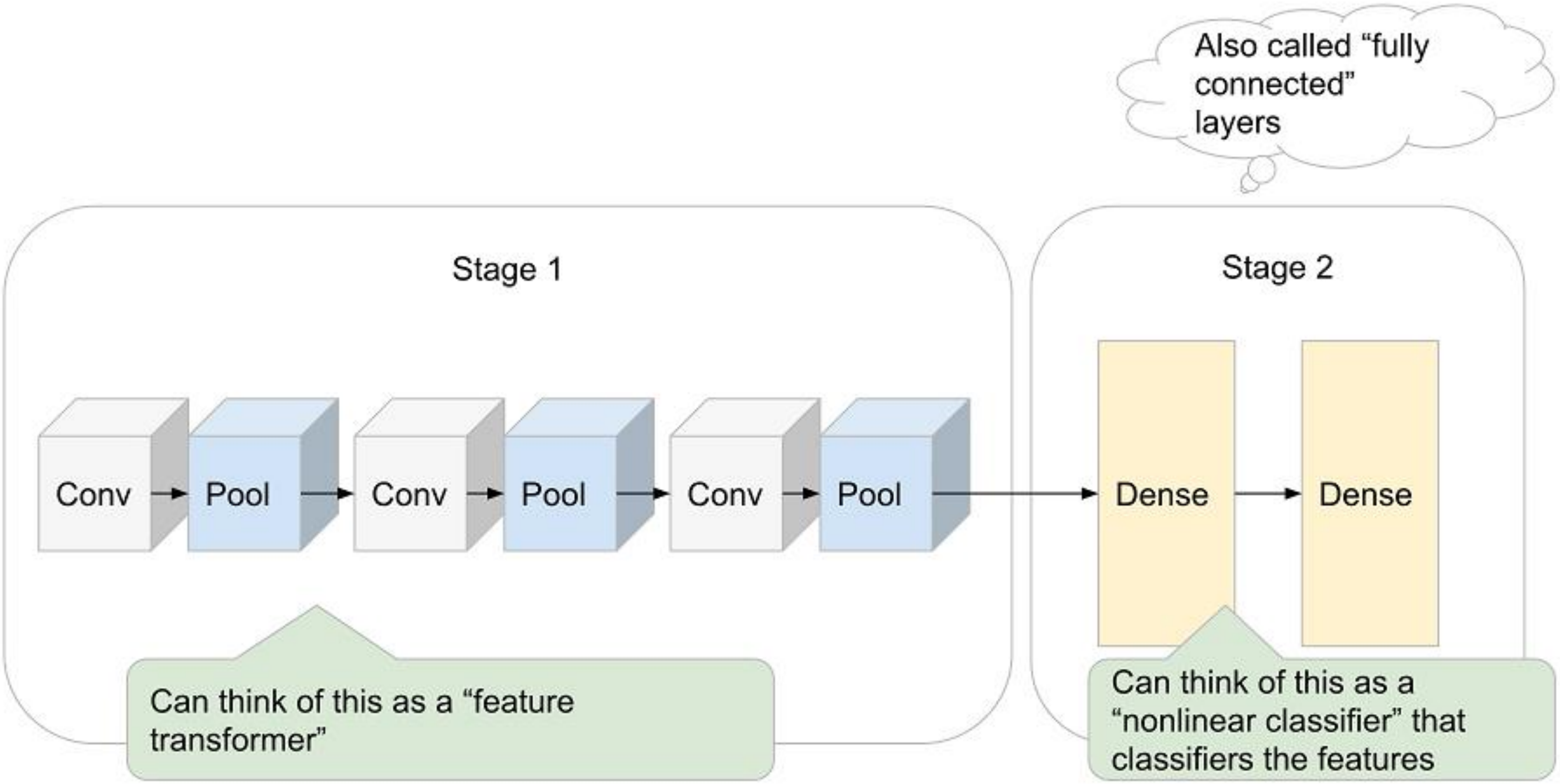


Typical CNN



Also called “fully connected” layers
Stage 1
Stage 2
Conv
Pool
Conv
Pool
Conv
Pool
Dense
Dense

Typical CNN


Also called "fully connected" layers
Stage 1
Stage 2
Conv
Pool
Conv
Pool
Conv
Pool
Dense
Dense
Can think of this as a "feature transformer"
Can think of this as a "nonlinear classifier" that classifiers the features

Pooling means DownSampling

i.e. output a smaller image from a bigger one.

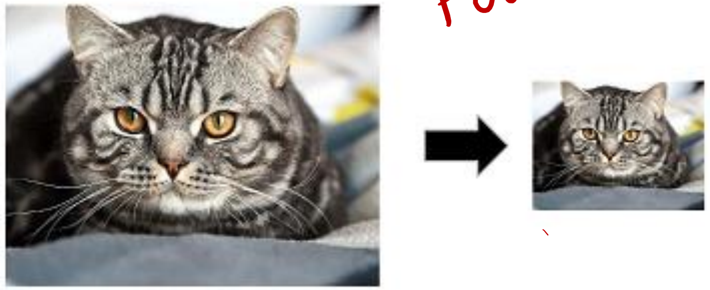
eg: If i/p image = 100 × 100

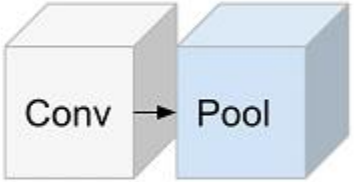
then if pool_size = 2

then my o/p image = 50 × 50

∴ $\boxed{\text{DownSample by 2.}}$

POOLING





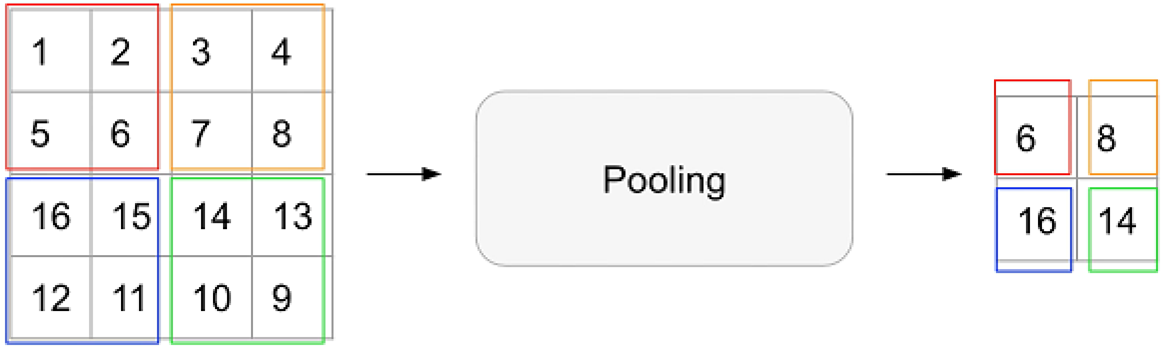
Types of Pooling

- There are two types of Pooling:

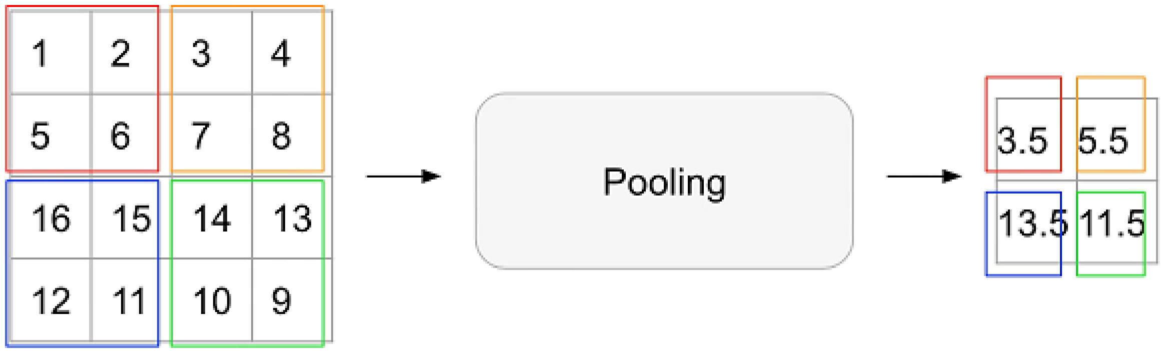
1. Max pooling
2. Average pooling

- Which one to use is a Hyperparameter choice.

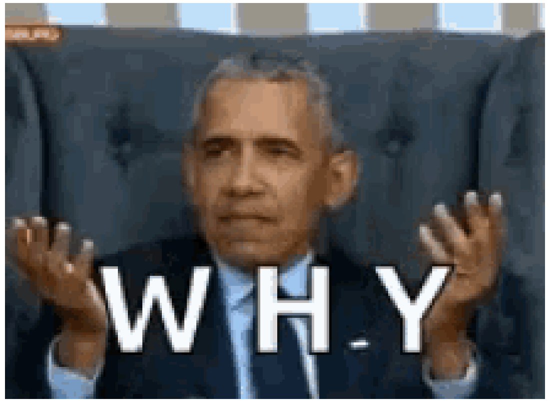
Max Pooling



Average Pooling



Why to use Pooling?



- Practical: If we shrink the image, we have less data to process.

- Translational Invariance: I don't care where in the image the feature occurred, I just care that it did.