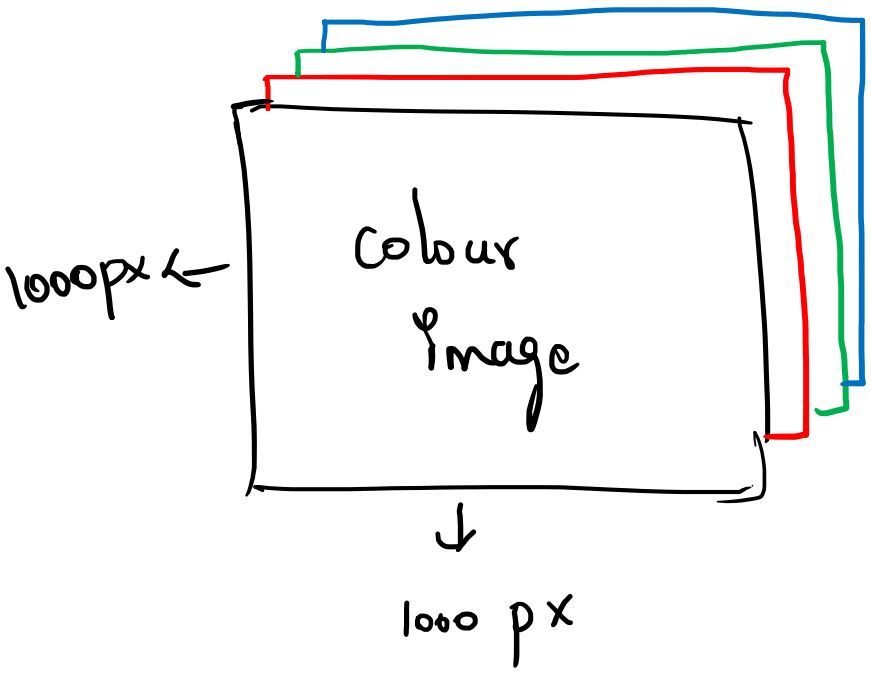
Convolutional Neural Network: (CNN) ⇒ images, videos

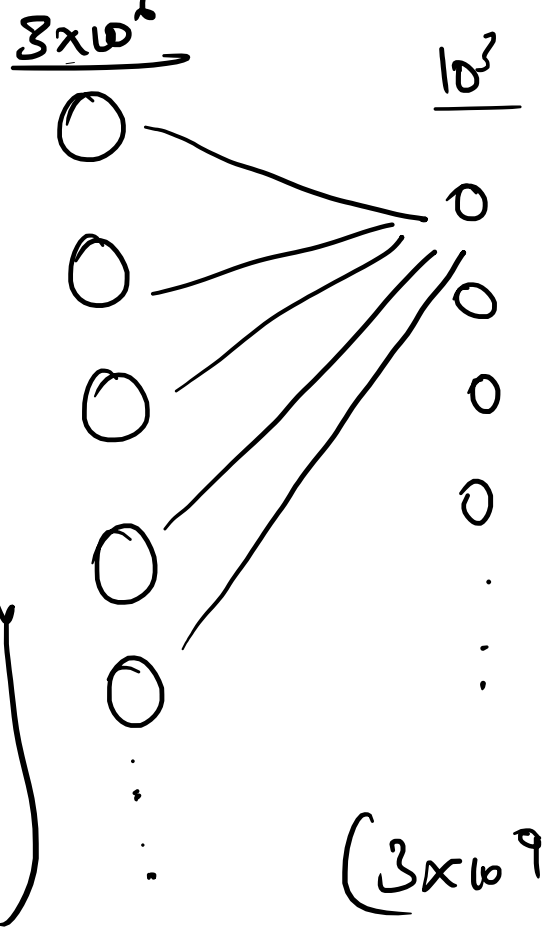
The architecture of CNN is completely different from ANN.

$(10^3 \times 10^3 \Rightarrow 10^6)$



(3×10^6)

Since it creates more neurons on each layer it cost too much for training and deployment



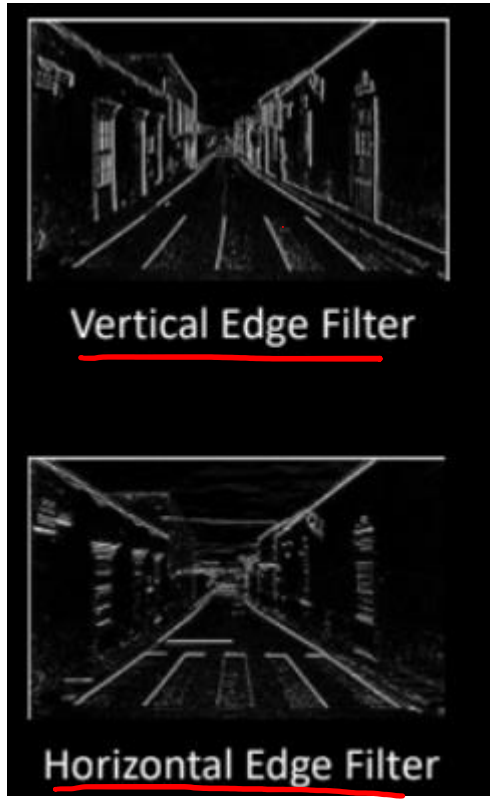
$(3 \times 10^9 + \text{bias})$

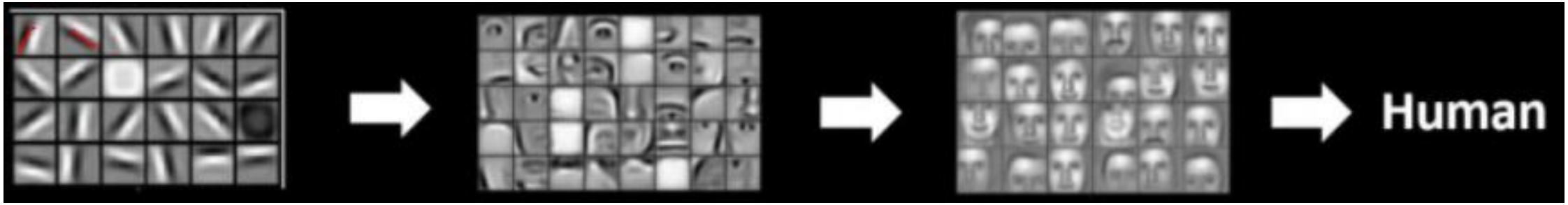
⇒ Images are made up of pixels:

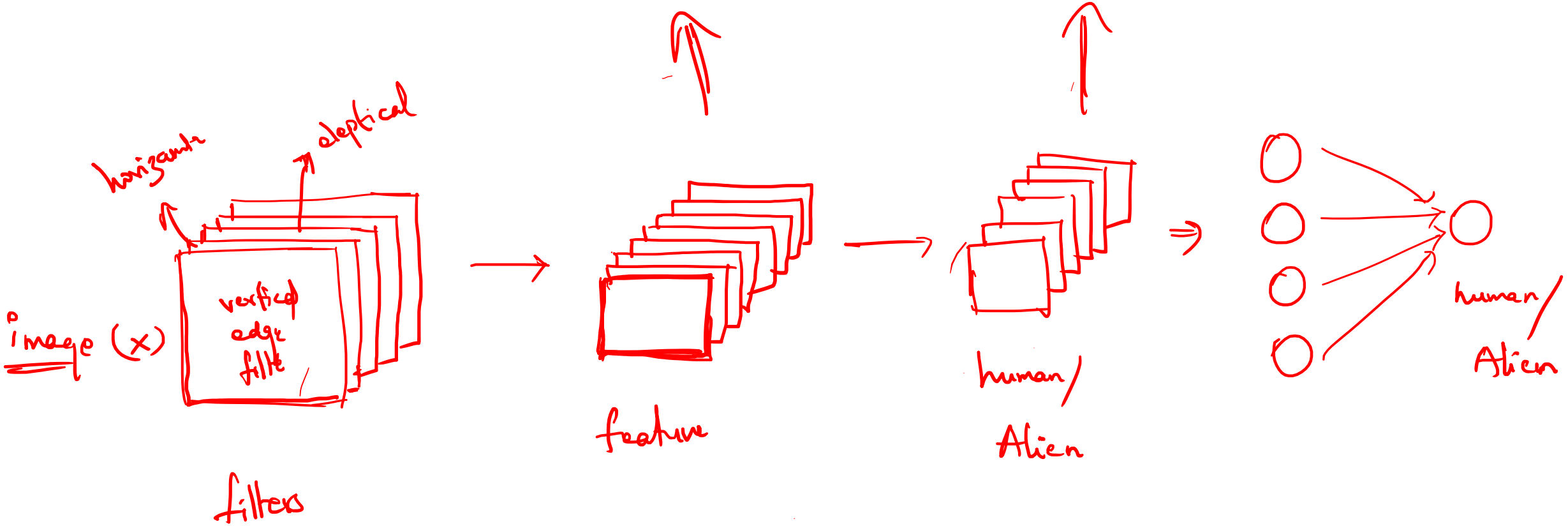
⇒ pixels are the input for CNN

Filters:

image




Human


horizontal
diagonal
vertical edge filter
image (x)
filters
feature
human/
Alien
human/
Alien

(n x n)

Grey scale

1	6	9	10	2	8
2	5	1	8	4	2
3	7	4	9	10	3
9	8	3	6	7	9
8	0	9	4	7	2
9	10	12	6	9	8

{6 x 6 pixel}

(f x f)

{vertical edge filter}

(x)

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

{3 x 3 pixel}

(window)

⇒

-8	-9	-2	14
6	-3	-13	9
4	-4	-8	5
2	2	1	-3

{4 x 4 pixel}

$(n \times n) \times (f \times f) = (n-f+1) \times (n-f+1)$

$(6 \times 6) \times (3 \times 3) = (6-3+1) \times (6-3+1) \Rightarrow (4 \times 4)$

Input image (x) filters

{Grey scale} → {horizontal filter, vertical edge} ⇒ {output image with layers and filters}

↓ ↓ ↓

$(n \times n) \times 1$ $(f \times f) \times c$ $(n-f+1) \times (n-f+1) \times c$

↓ ↓ ↓

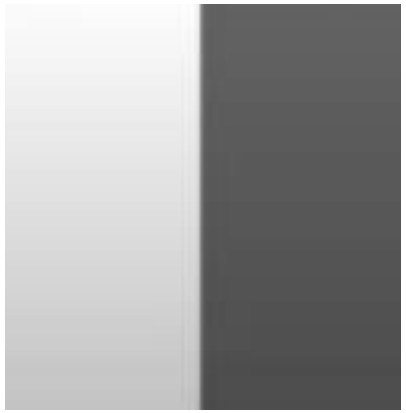
1-layer {# of filters} {# filters}

Grey scale image → 1 layer

Colour image → 3 layer (RGB)

(6×6 px)

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



x

(3×3 px)

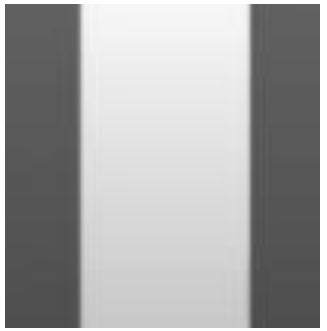
1	0	−1
1	0	−1
1	0	−1

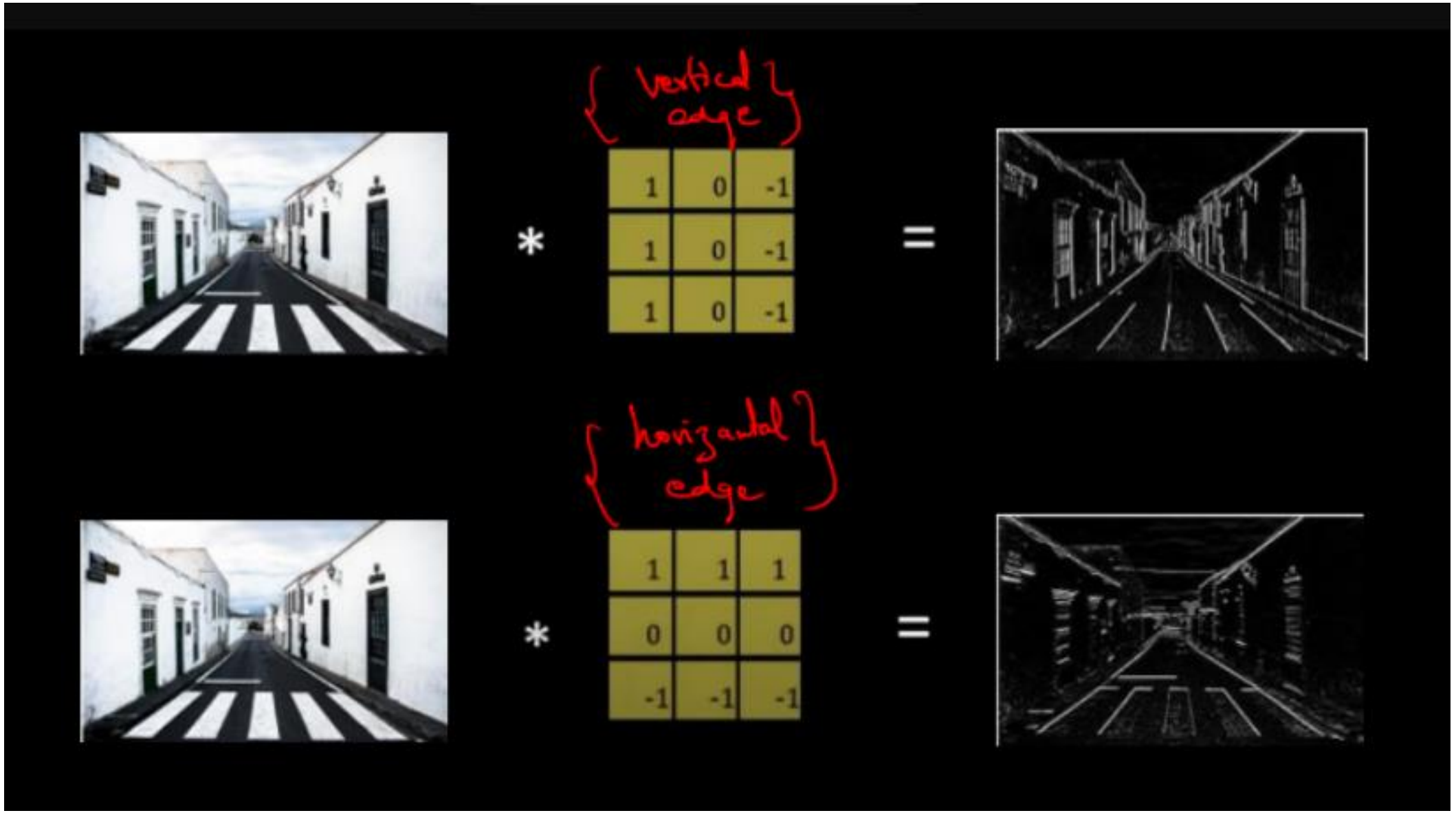
{ vertical edge filter }

⇒

(4×4 px)

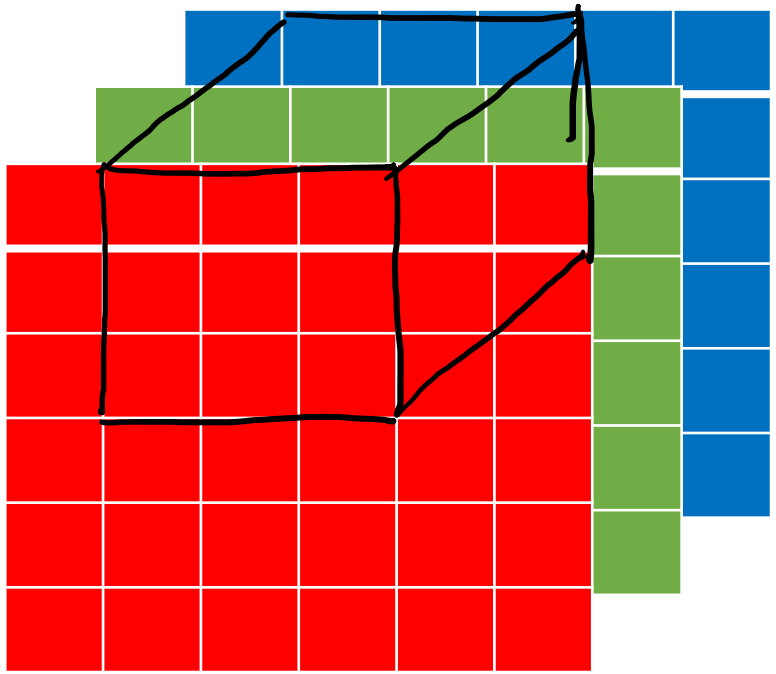
0	3	3	0
0	3	3	0
0	3	3	0
0	3	3	0




{ vertical edge }
1 0 -1
1 0 -1
1 0 -1
*
=
{ horizontal edge }
1 1 1
0 0 0
-1 -1 -1
*
=

$\Rightarrow$ (layer-1) + (layer-2) + (layer-3) = 1

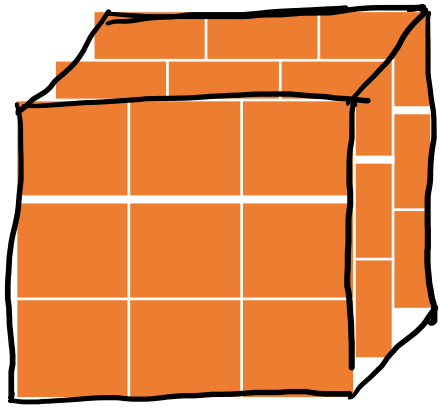
(Colour image)



$n \times n \times 3$

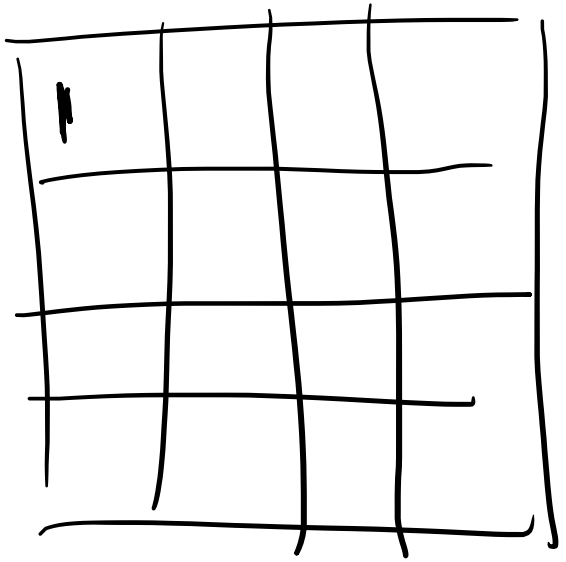
(x)

{vertical edge filter}

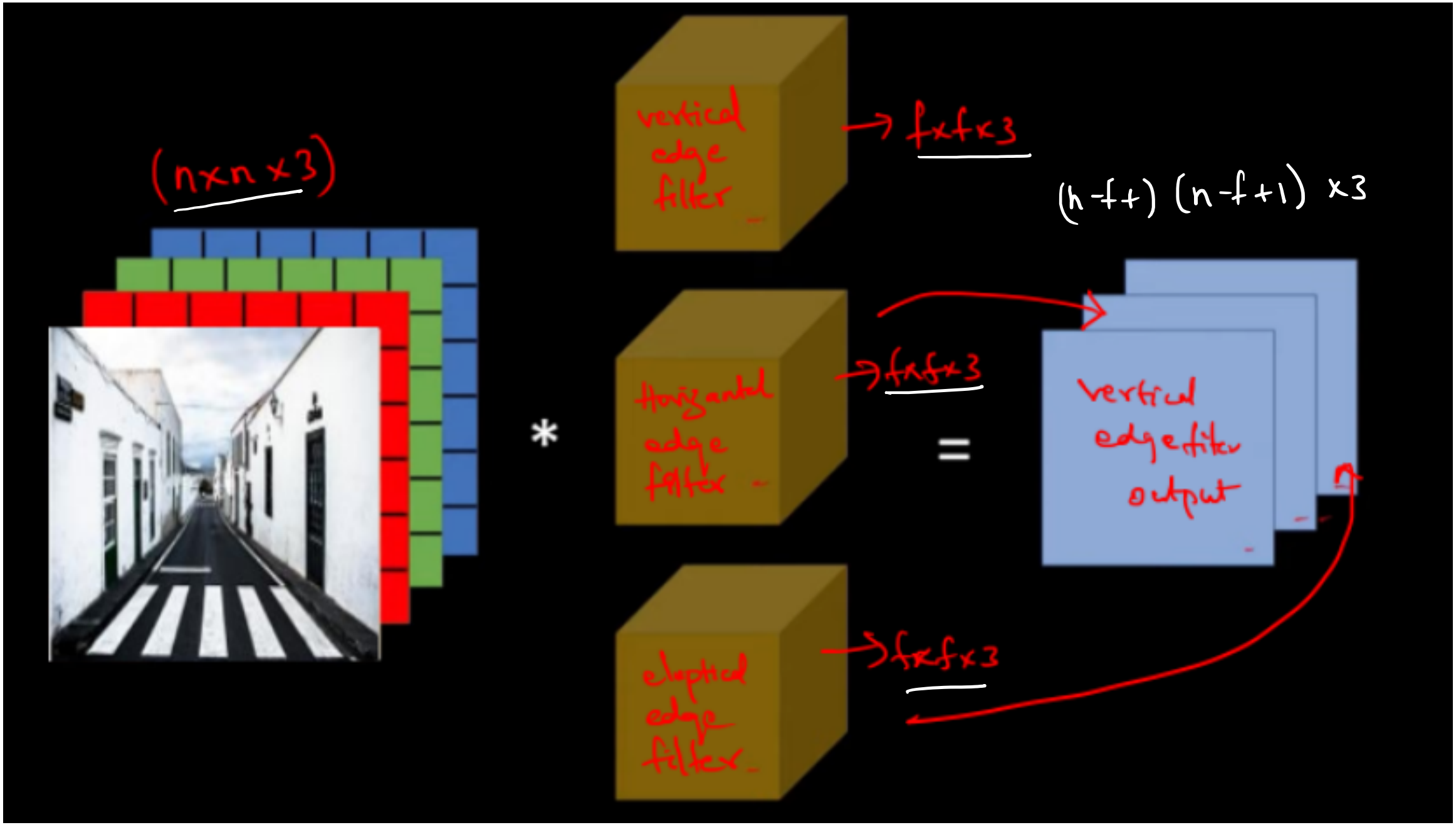


$f \times f \times 3$

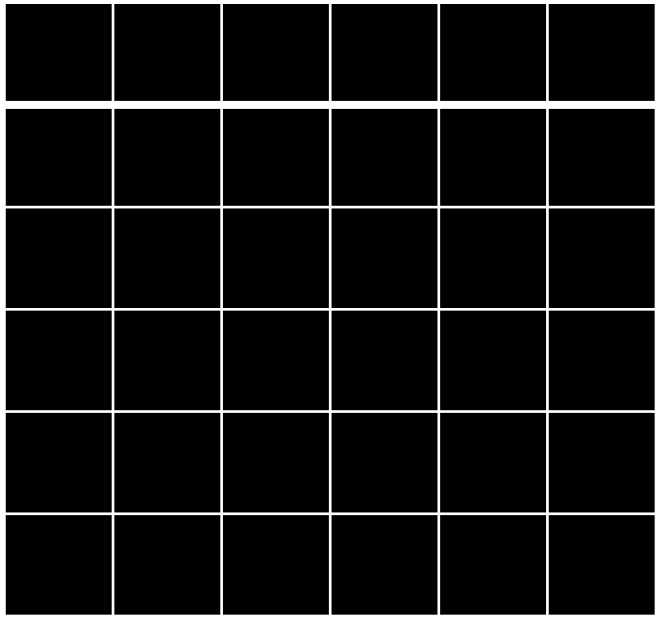
$\Rightarrow$



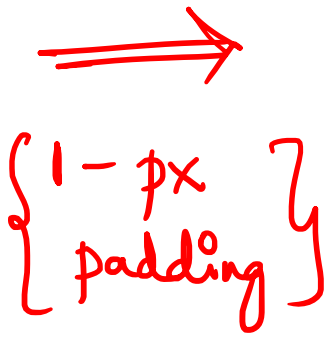
$(n-f+1) \times (n-f+1) \times 1$

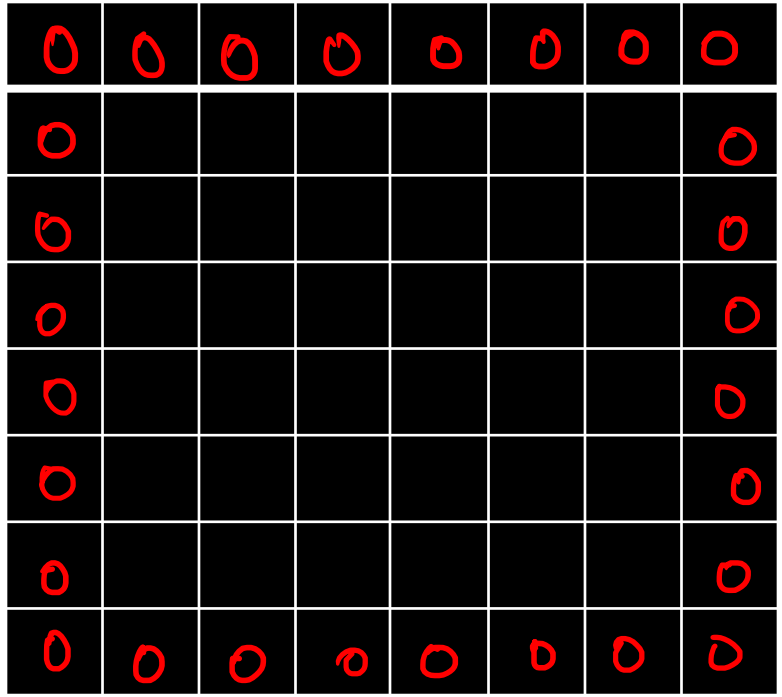

(n x n x 3)
*
vertical edge filter
f x f x 3
(n-f+1) (n-f+1) x 3
horizontal edge filter
f x f x 3
=
vertical edge filter output
elliptical edge filter
f x f x 3

Padding: ⇒ padding is used to retain the output px w.r.t input px



{6x6 px}





{8 x 8 px}

padded image * vertical filter ⇒ output

$\{8 \times 8\} * \{3 \times 3\} \Longrightarrow \{6 \times 6\}$

$(n \times n) * (f \times f) = (n-f+1)\ (n-f+1)$

$= (8-3+1)\ (8-3+1)$

$= (6 \times 6) \Rightarrow$ orginal image size

Types:

1. valid convolution ⇒ No padding

2. Same Convolution ⇒ $n' = n + 2p$ ⇒ (equ-1)

$(6 \times 6) \xrightarrow{[1-p\times]} (8 \times 8)$

$n \Downarrow$ p(?) $\Downarrow n'$

$\Rightarrow (n' - f + 1) = n$

$(n' \times n')\ (f \times f) = (n' - f + 1)\ (n' - f + 1)$

Apply equ-1 on above

$\Rightarrow (n + 2p - f + 1) = n$

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

$\Rightarrow 2p = f - 1 \Rightarrow p = \{(f-1)/2\} \Rightarrow$ padding size formula

$\{6 \times 6\} \times \{3 \times 3\} \Rightarrow \{n \times n\} \times \{f \times f\}$

$\Rightarrow \{n' - f + 1\} = n$

$\Rightarrow \{n + 2p - f + 1\} = n$

$\Rightarrow \{6 + 2p - 3 + 1\} = 6$

$\Rightarrow 2p = 6 - 6 + 3 - 1$

$\Rightarrow 2p = 2$

$\Rightarrow p = 1$

$\{6 \times 6\} \rightarrow$ 1-px padding $\rightarrow$?

$\Rightarrow n' \times n' = (n + 2p)(n + 2p)$

$\Rightarrow n' \times n' = (6 + 2(1))(6 + 2(1))$

$\Rightarrow n' \times n' = 8 \times 8$

Python: $\Rightarrow$ keras.conv2d(x, W1, padding = "valid" or "same")