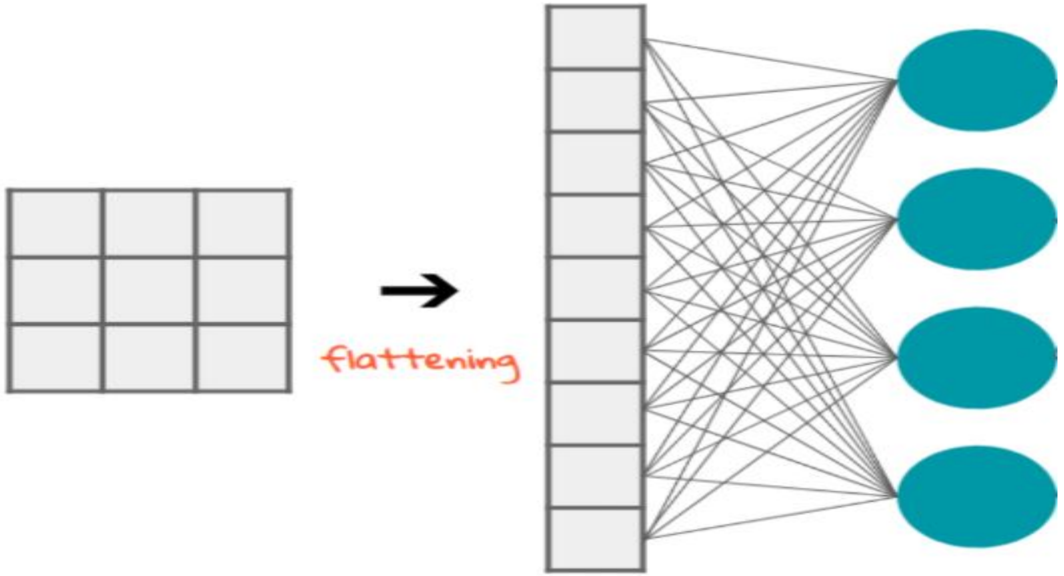
CNN

Why not use ANN?

- High computational cost
  → Millions of parameters for even small images
- Severe overfitting
  → Too many weights, not enough data
- Loss of important spatial information
  → Treats pixels as independent, ignores 2D structure

When we flatten 2D to 1D input weight increase and training time increase and it causes to loss of spatial arrangement of pixels



What is CNN

Convolutional Neural Networks (CNNs), also known as ConvNets, are a specialized type of neural network designed to process data with a known grid-like structure—such as images, videos, and time series.

A CNN typically consists of three main types of layers:

1. Convolutional layers – to extract features

2. Pooling layers – to reduce spatial dimensions

3. Fully connected (FC) layers – to classify the final features

CNNs learn hierarchical features across layers:

- Early layers detect low-level features (e.g., edges, corners, gradients)
- Middle layers combine these into mid-level features (e.g., eyes, noses, wheels, textures)
- Deep layers recognize high-level semantic objects (e.g., faces, cars, dogs)

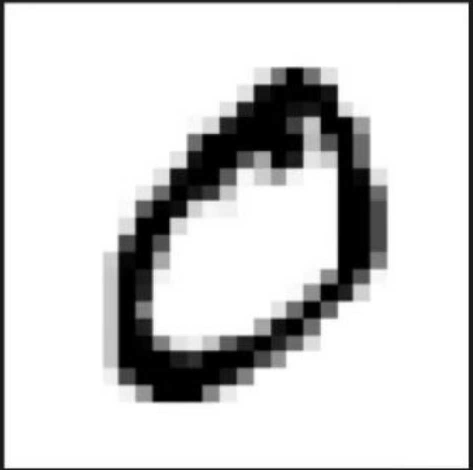
CNNs can process different types of image inputs, most commonly:

Grayscale images

- Single channel (intensity only)
- Shape: Height × Width × 1
- Example: Medical scans, handwritten digits (MNIST)

RGB images

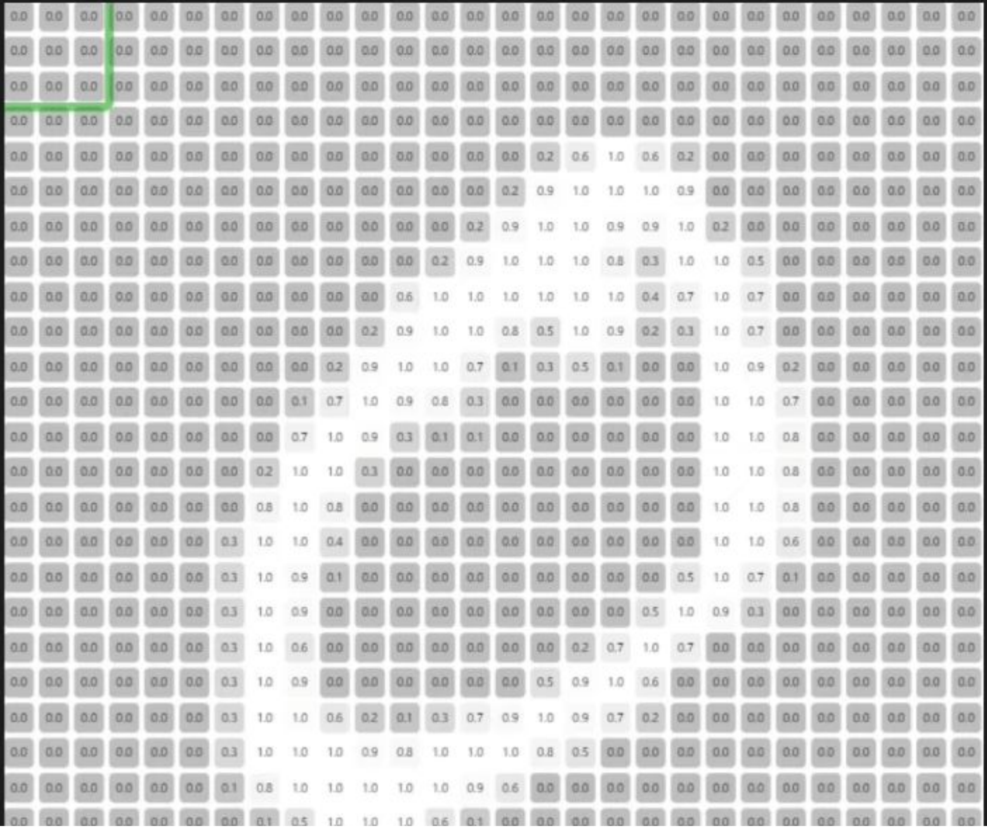
- Three color channels: Red, Green, Blue
- Shape: Height × Width × 3
- Example: Photos from cameras, web images (CIFAR-10, ImageNet



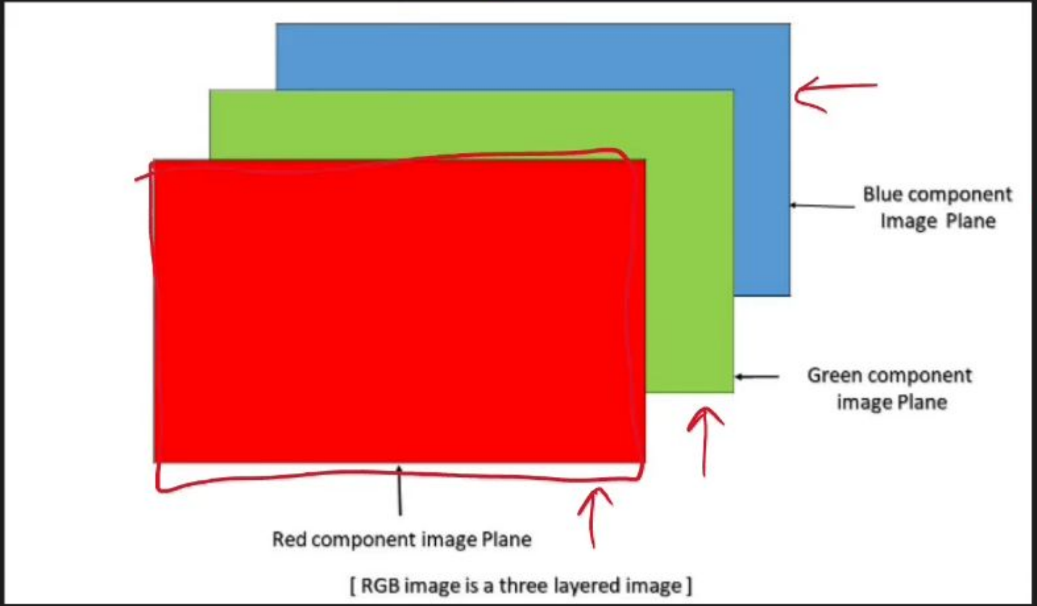
$\uparrow$

mnist $\rightarrow \underline{0}$

(28×28)



0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.6	1.0	0.6	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.9	1.0	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.9	1.0	1.0	0.9	0.9	1.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.9	1.0	1.0	1.0	0.8	0.3	1.0	1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	1.0	1.0	1.0	1.0	1.0	1.0	0.4	0.7	1.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.9	1.0	1.0	0.8	0.5	1.0	0.9	0.2	0.3	1.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.9	1.0	1.0	0.7	0.1	0.3	0.5	0.1	0.0	0.0	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.7	1.0	0.9	0.8	0.3	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.7	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	1.0	0.9	0.3	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.8	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	1.0	1.0	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.8	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	1.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.8	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.6	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	0.9	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	0.7	0.1	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	0.9	0.3	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.7	1.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.9	1.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.6	0.2	0.1	0.3	0.7	0.9	1.0	0.9	0.7	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	1.0	0.9	0.8	1.0	1.0	1.0	0.8	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.8	1.0	1.0	1.0	1.0	1.0	0.9	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.5	1.0	1.0	1.0	0.6	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

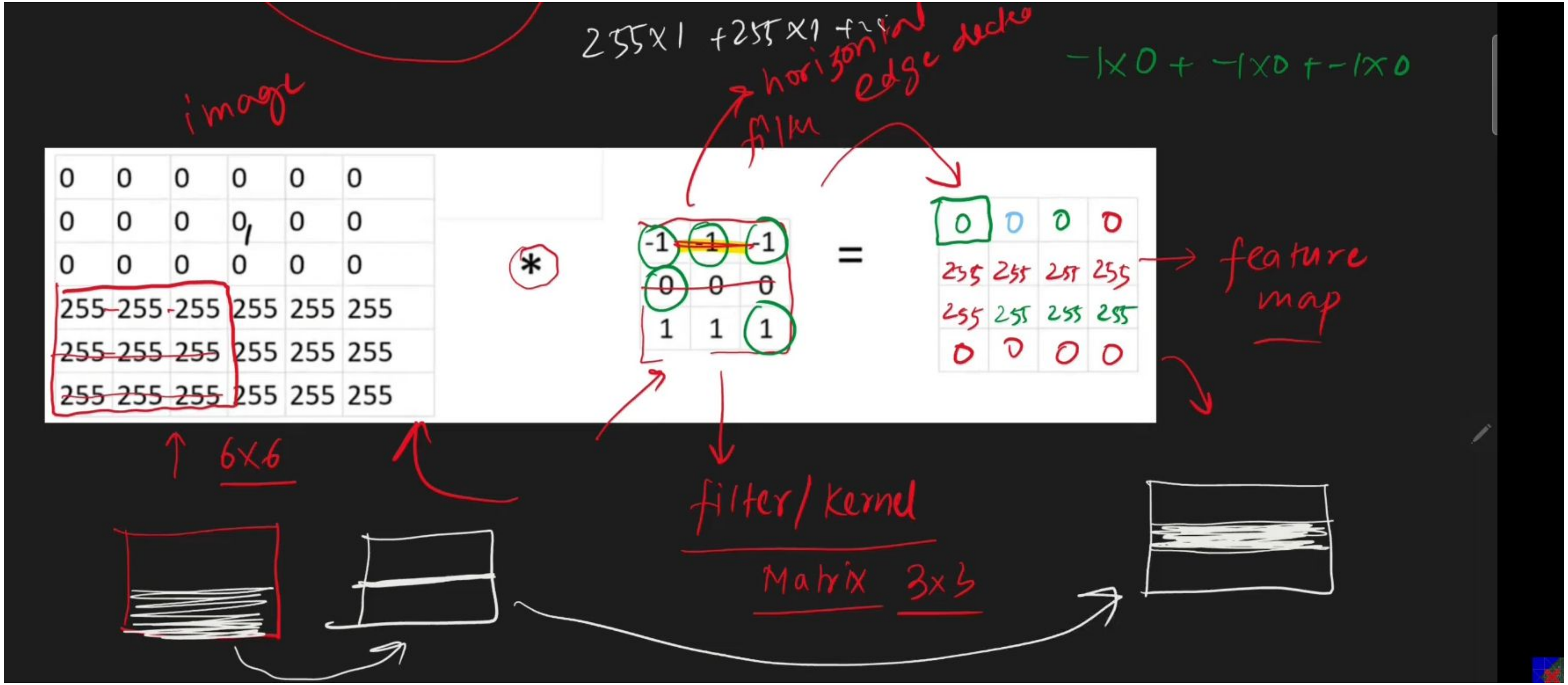

Blue component Image Plane
Green component image Plane
Red component image Plane

[RGB image is a three layered image]

$\boxed{228 * 228 * 3}$

Edge Detection in CNNs

In Convolutional Neural Networks (CNNs), edge detection emerges automatically in the early layers during training. The network uses small learnable filters (also called kernels) — typically 3×3 or 5×5 matrices of weights — that slide across the input image to produce feature maps. Initially, these filters contain random values, but through backpropagation and gradient descent, they adapt to detect patterns that help reduce the overall loss. Remarkably, without any human guidance, the first-layer filters often converge to detect fundamental visual primitives such as vertical edges, horizontal edges, diagonal lines, textures, and color contrasts. This self-organized feature learning is a core reason CNNs outperform traditional methods: instead of relying on handcrafted edge detectors (like Sobel or Canny), the network discovers the most useful filters for the task directly from data.



0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

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

0	0	0	0
255	255	255	255
255	255	255	255
0	0	0	0

https://deeplizard.com/resource/pavq7noze2

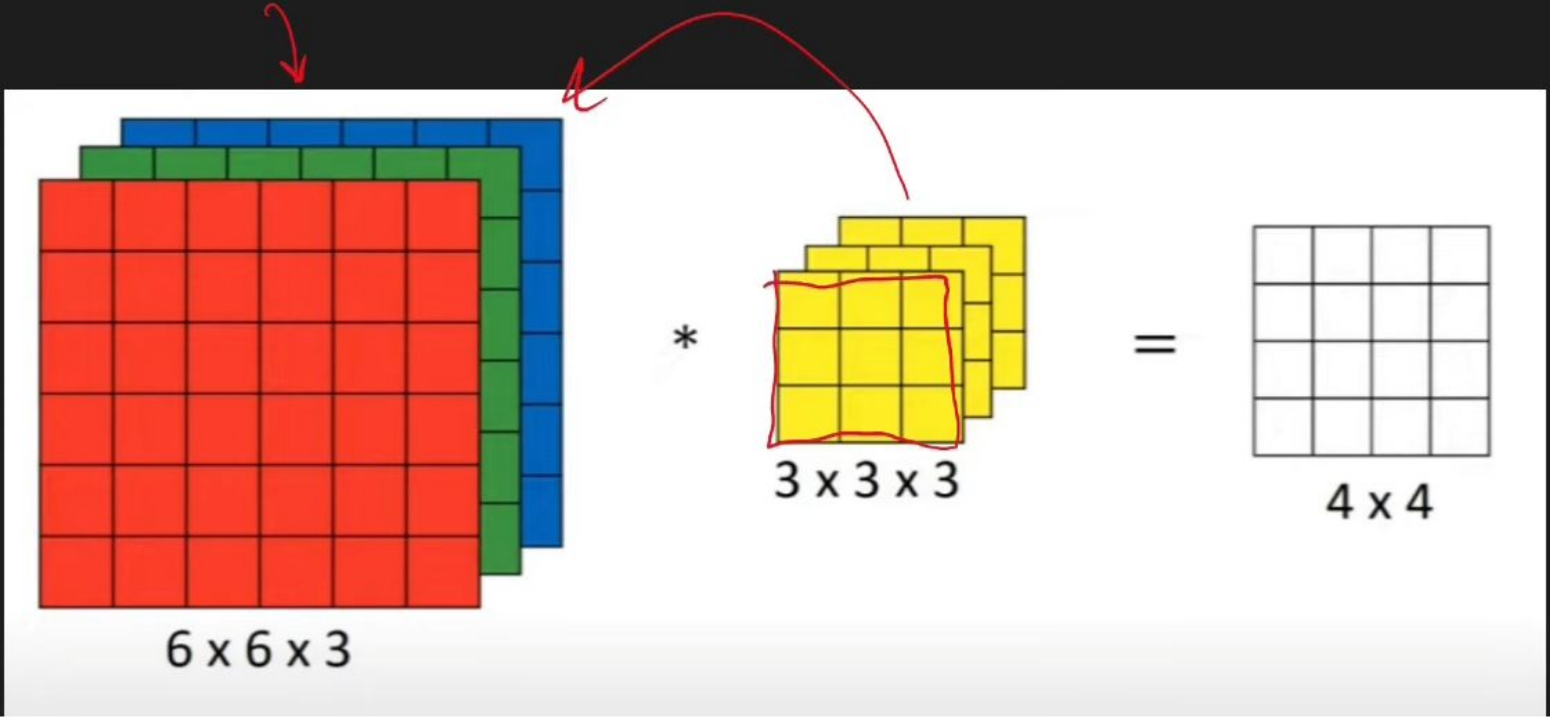
Image * filter=feature Image

(m x m) *(f x f) = (m-f+1)(m-f+1)

Working with RGB images

19 August 2022 16:54

$\underline{\boxed{3 \times 3} \times 3}$ $\underline{3 \text{ channel}}$



Multiple Filters

23 August 2022 08:24

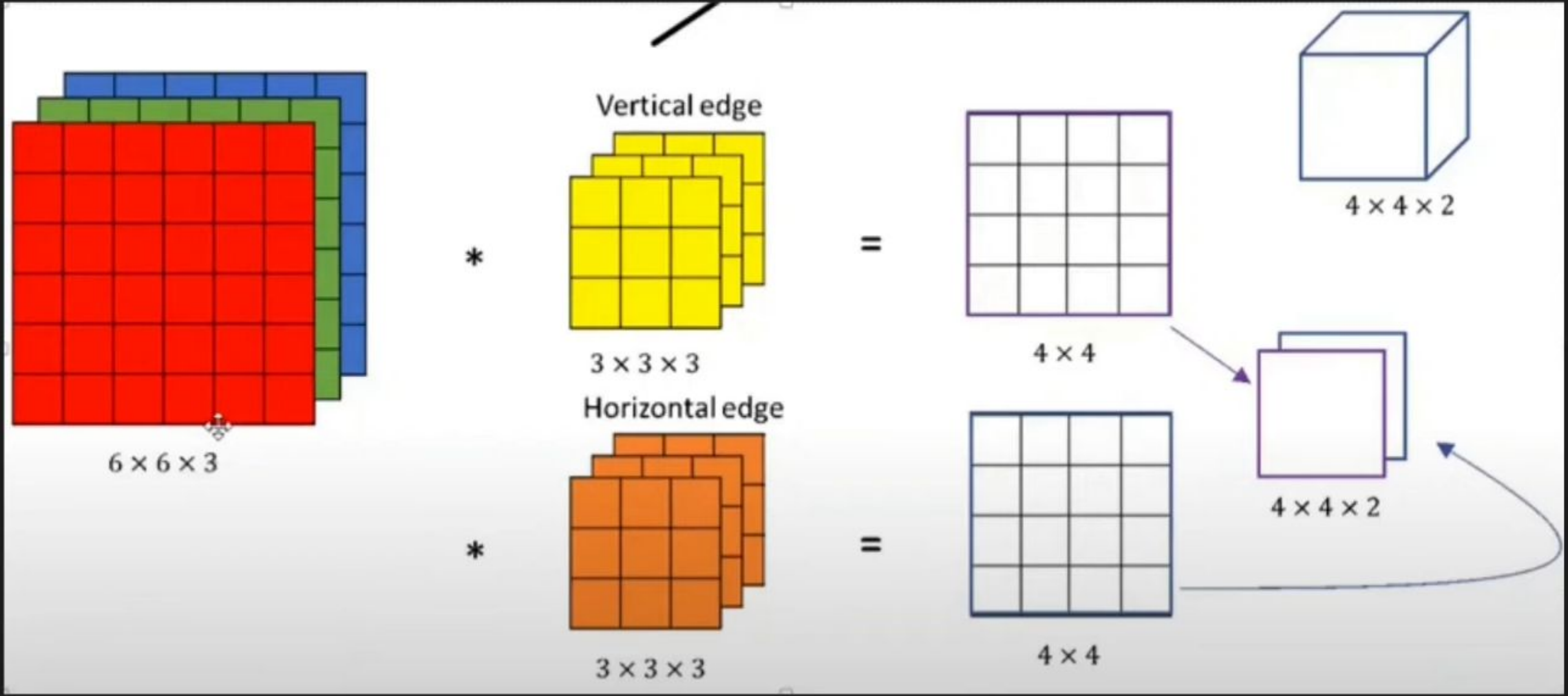


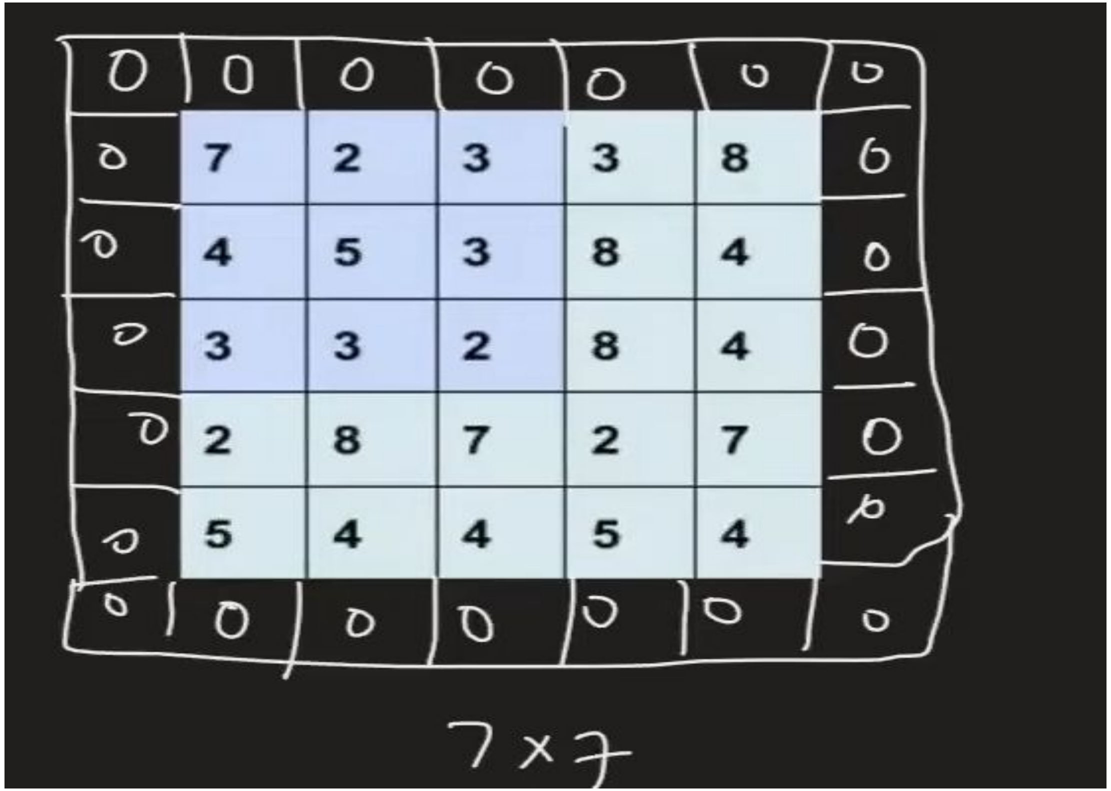
Image taken from Andrew NG's lecture

Padding

Cause size of image less than original and numbers aka pixel value and border doesn't have that much importance than others

So we increase the size of margin

5*5 to=7*7 which give 5*5 feature map



0	0	0	0	0	0	0
0	7	2	3	3	8	0
0	4	5	3	8	4	0
0	3	3	2	8	4	0
0	2	8	7	2	7	0
0	5	4	4	5	4	0
0	0	0	0	0	0	0

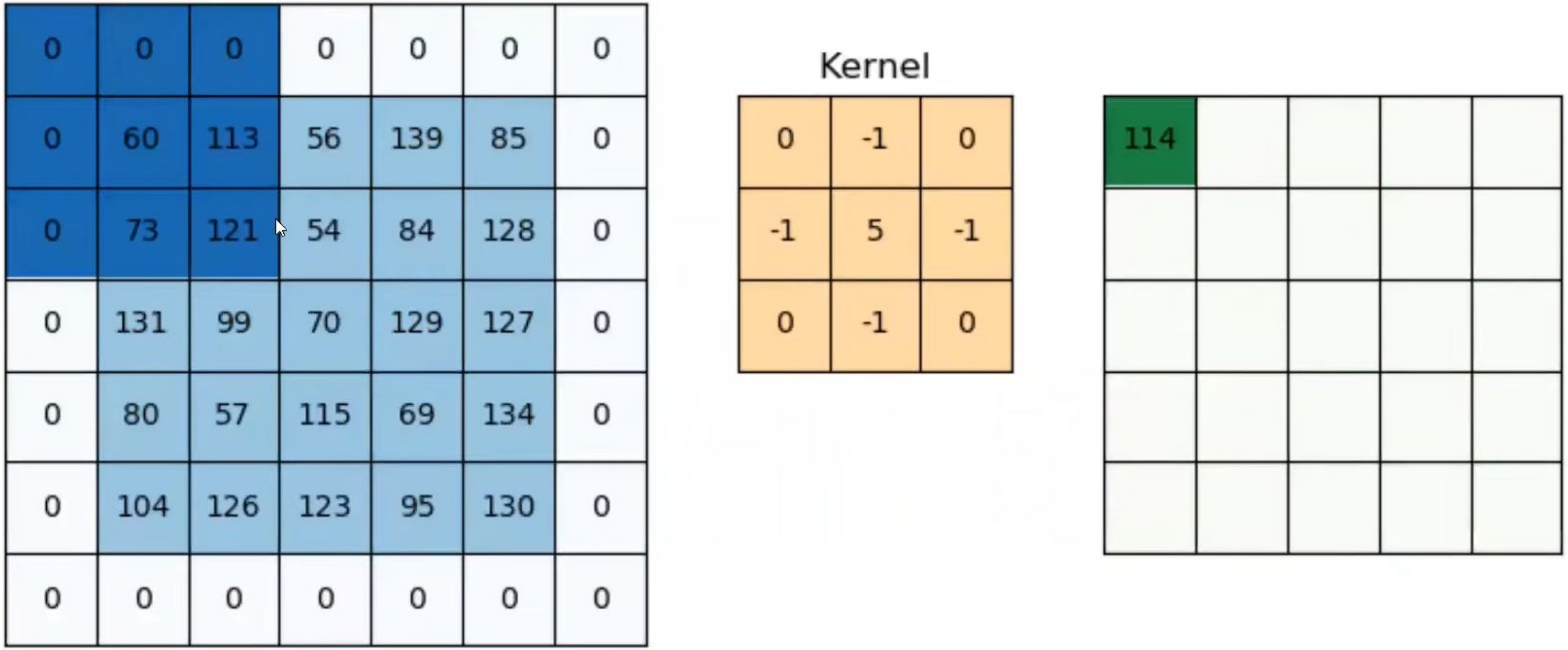
7×7

2 types of padding

Valid:no padding,

Same padding to get same number of feature image as input

https://colab.research.google.com/drive/1HBMLctcBnhvV6Rj62Zc8eAXERQw54I2H?usp=sharing



0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

Stride

Is the moment space or speed/shift of filter over image

When stride decrease feature map size decrease

$((n-f)/s)+1$=output image

Stride are used when only high lvl features are required

Pooling

Pooling (typically max pooling) is used to:

Reduce Spatial Dimensions

→ Shrinks the width and height of feature maps

Introduce Translation Invariance

→ Helps the network recognize features even if they shift slightly in the image

→ Example: A cat’s eye is still detected whether it’s 1 pixel left or right

Retain the Most Important Information

→ Max pooling keeps the strongest activation (e.g., the clearest edge response)

→ Discards less relevant details, acting as a form of controlled downsampling

Prevent Overfitting

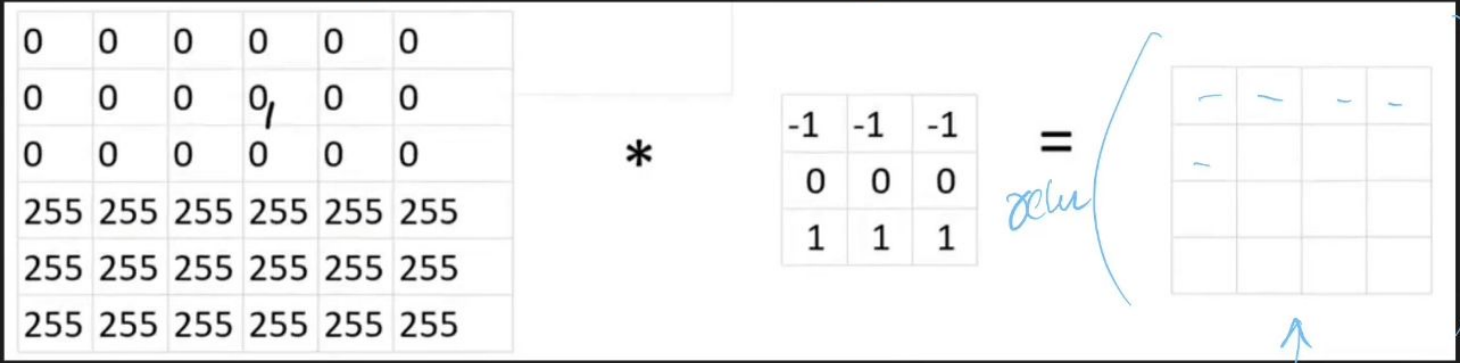
Types of pooling

https://medium.com/@abhishekjainindore24/pooling-and-their-types-in-cnn-4a4b8a7a4611

https://colab.research.google.com/drive/1F4F6Q9O-hPvCDeOWcqMUa5BuBOvuOBWc?usp=sharing

Pooling

01 September 2022 09:55



0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

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

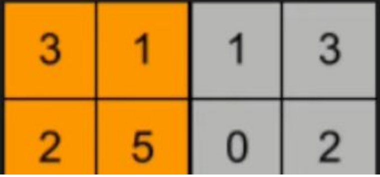
= relu(

-	-	-	-
-			

)

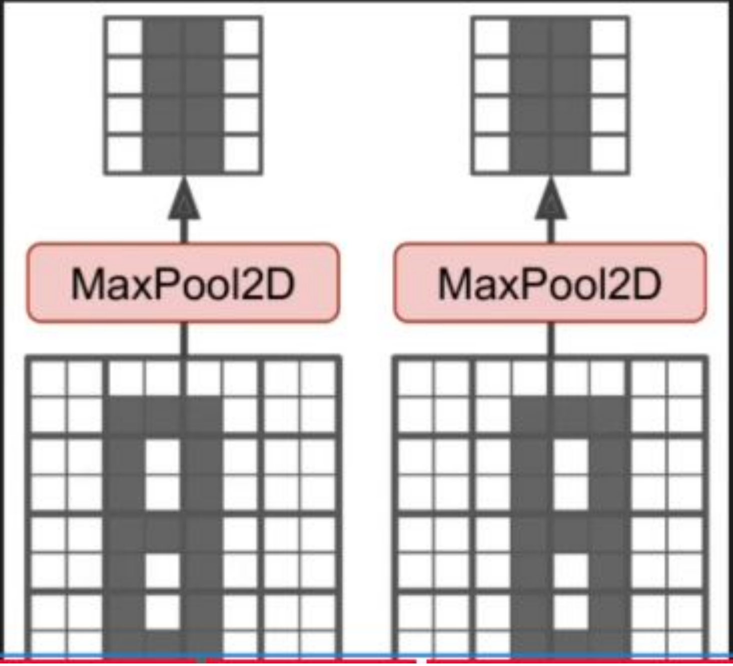
feature map (non-linear)

Max pooling
Min pooling
Avg pooling
L2 pooling
Global pooling

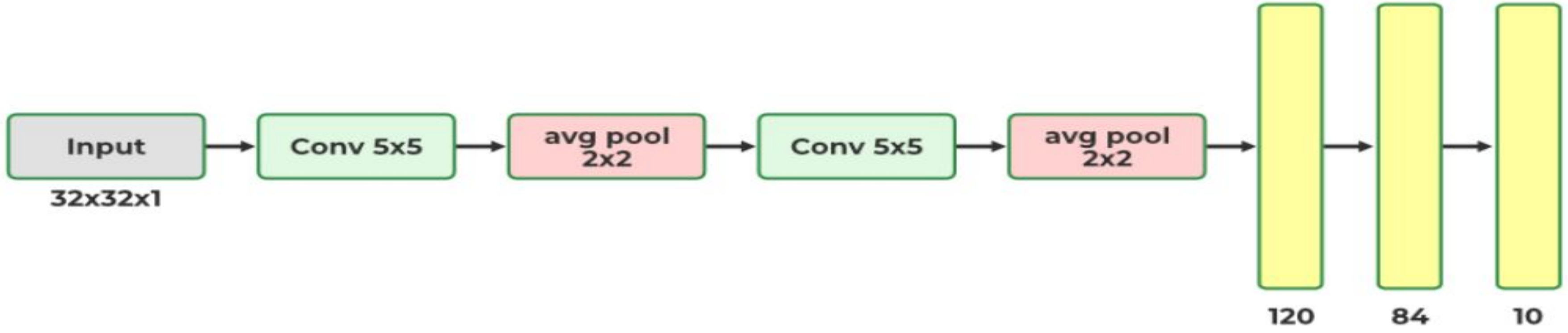


3	1	1	3
2	5	0	2

2) Translation invariance



https://www.geeksforgeeks.org/machine-learning/convolutional-neural-network-cnn-architectures/



Regularisation

ANN - weight / bias
↳ loss function

we add a penalty term in cost func

Cost = L + (penalty)

L2
L1

with penalty value will come near to zero

penalty value will come near to zero

Intuition behind Regularisation

learning rate

$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_o}$ - partial derivative of loss func

$= w_i - \eta \left(\frac{\partial L}{\partial w_i} + \lambda w_i \right)$

so weight decrease

~~so~~ model learn simple, more general

Dropout

we randomly drop node in layers

so architecture change,

2% increase accuracy

so dependec on nodes decrease

so weigh distribution increase

Batch Normallization

It make traning faster and stable

by Normalis training can go faster and stable)

hidden layer output → input layer

we normalize every ned individ

$z_{11} = wx + b$

$y = \text{sig}(z_{11})$

in batch Normal

① $z_{11} \to z_{11} \xrightarrow{\text{Normalize}} g(z_{11}^{N}) = a_{11}$

2 normal

② $z_{11} \to g(z_{11}) \to a_{11} \to a_{11}^{N}$

② $z_{11} \to g(z_{11}) \to a_{11} \to a_{11}^{N}$

$\frac{z_{11} - \mu}{\sigma} = z_{11}^{N}$

$\frac{z_{11} - \mu}{\sigma} = z_{11}^{N}$

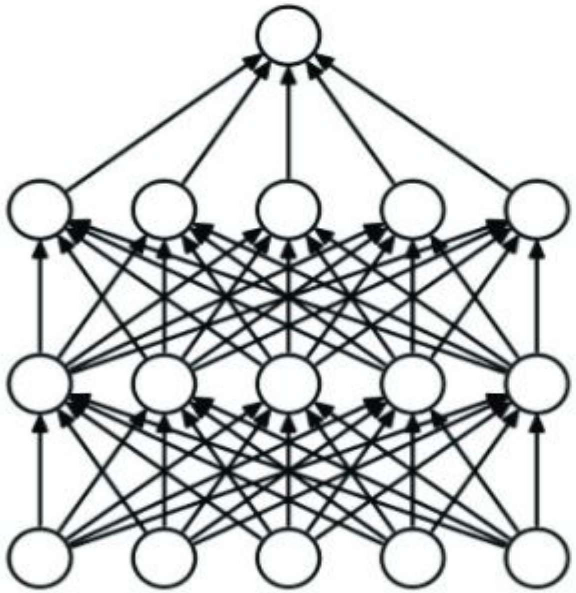
$\mu = \frac{1}{m} \sum_{i=1}^{m} z_{11}^{i}$ m = batch size

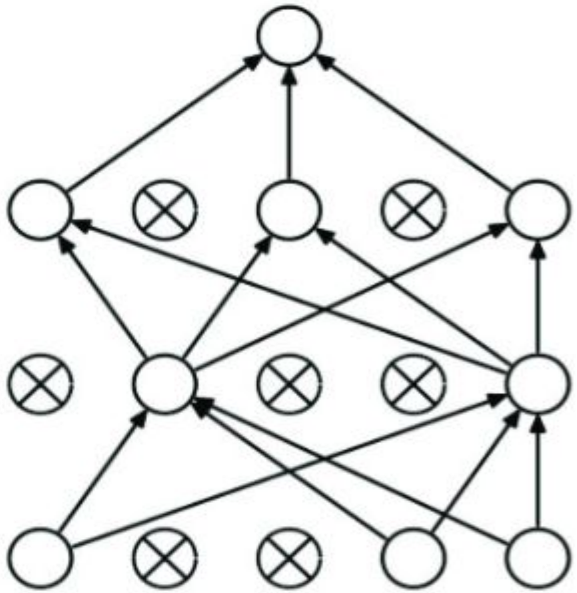
$\sigma = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (z_{11}^{i} - \mu_B)}$

https://colab.research.google.com/drive/1PObj5KrXLDDmHjoJ1x0bVmxAFbif5s7q?usp=sharing

https://colab.research.google.com/drive/1KyMLdV1yB0qVdS-1huxKMN9xVKhrfxGL?usp=sharing

https://colab.research.google.com/drive/1473vOd0ICPbRW-co_Rm-_TBXgeajkJZ_?usp=sharing#scrollTo=xMDnrLgHHAcP


(a) Standard Neural Network


(b) Neural Net with Dropout