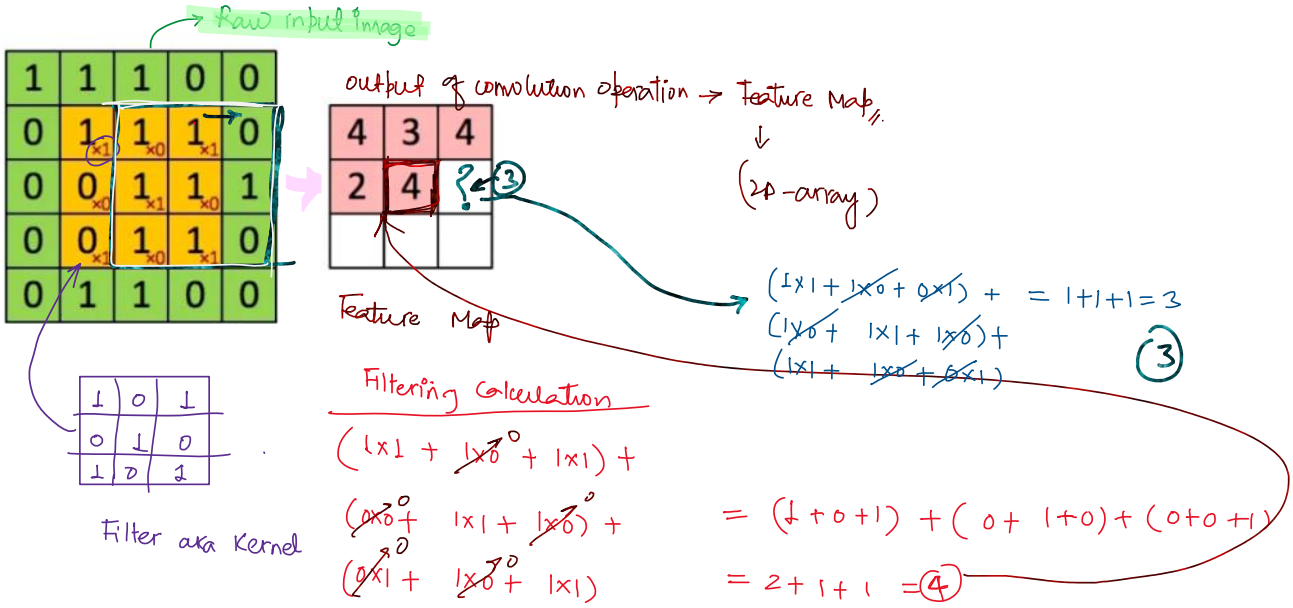
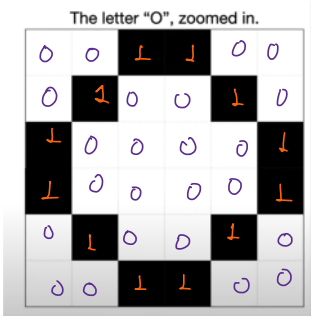
CNN Model Architecture Contd.

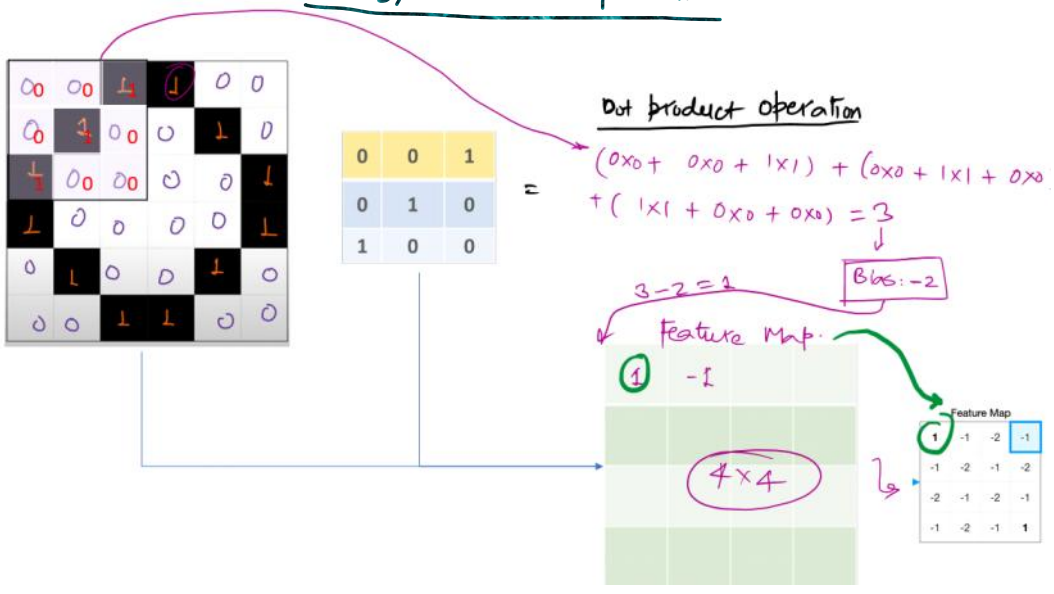
15 November 2025 10:14

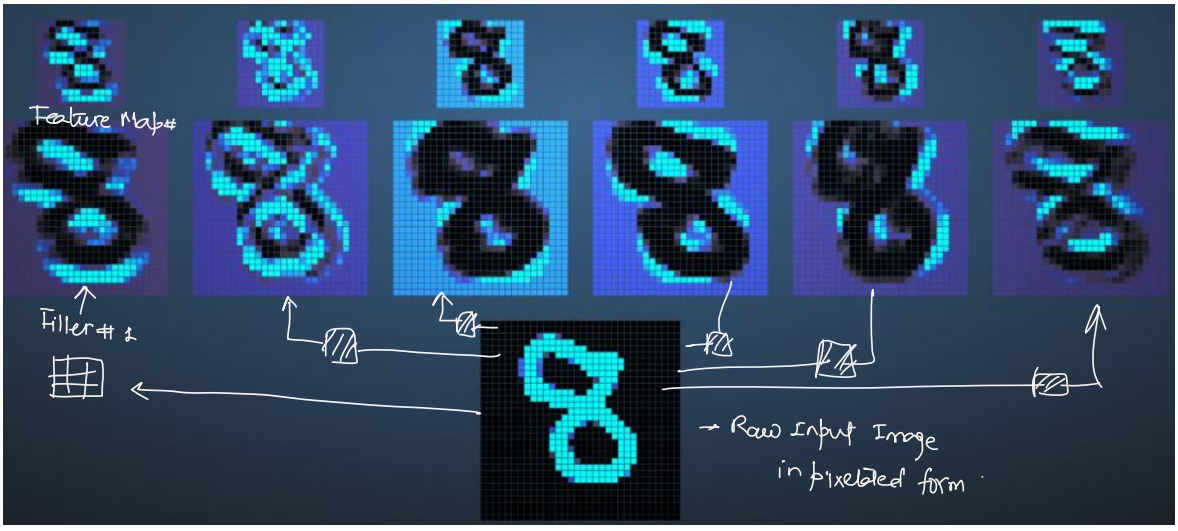
How does convolution work?



Tic-Tac-Toe





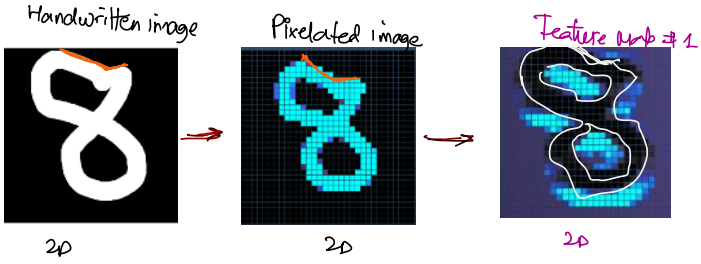


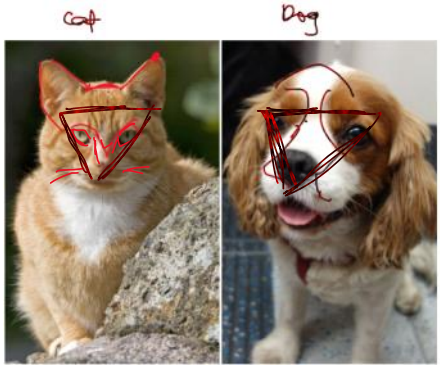
CONVOLUTION / Filtering operation

In simple terms, convolution is like sliding a small filter aka kernel over an input image → so basically a mathematical operation [dot product → sumproduct of cell overlapped with filter values values]

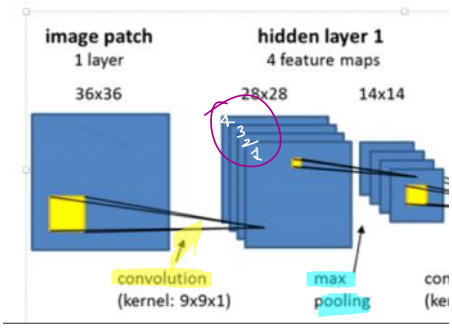
[to extract specific features such as edges, fine lines, strokes, corners, curves etc.]

to form shapes → patterns





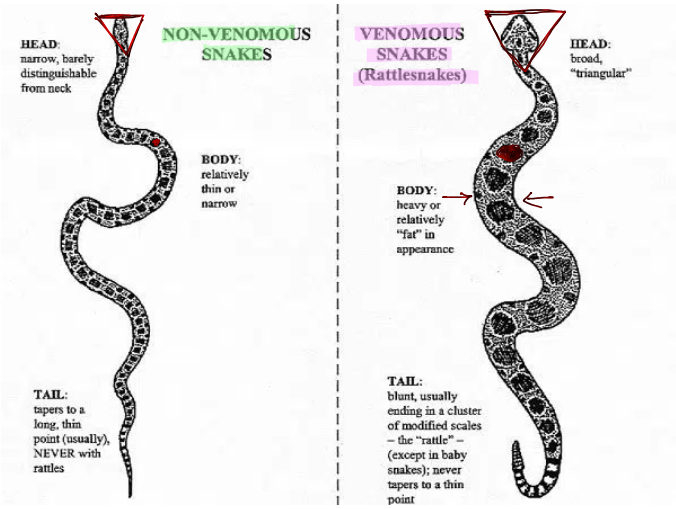
→ convolution



Note: convolutional layer applies a set of filters or kernels (4 filters) to an input image and filters slide over the image and generate a set of feature maps as the output. (4 feature maps)

CNN leverages convolution to:

- preserve the spatial relationship in the image
- efficiently learn patterns from the local regions of the image
- reduce the computational load compared to ANNs.



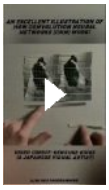
Intuitive Idea behind CNN

↳ convolution still with downsampling.

Original Image

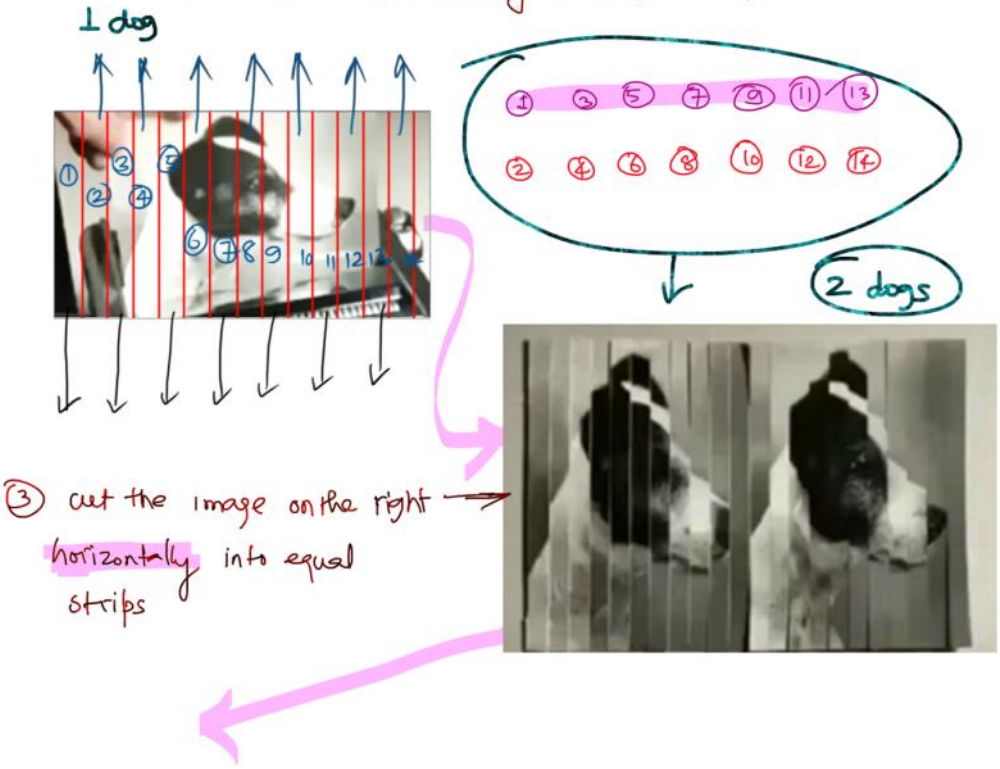
An excellent illustration of how CNN work! #artificialintelligence #deeplearning

https://www.youtube.com/shorts/N6NBT-n9mmo

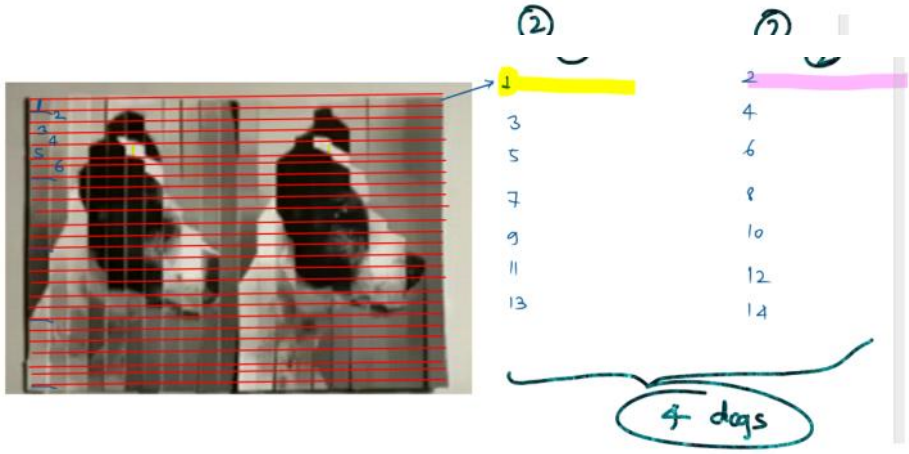


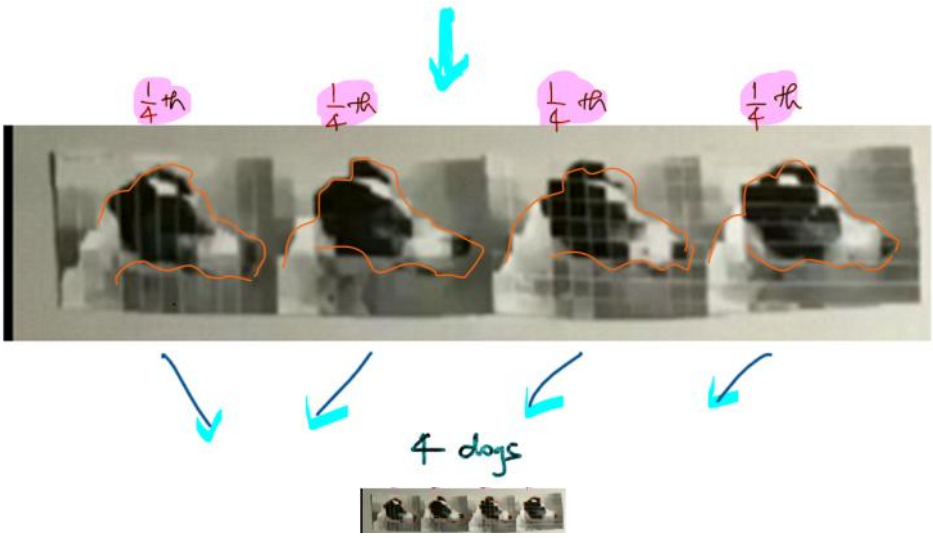
Original Image

2. cut the original dog's image vertically into equal strips



(3) cut the image on the right horizontally into equal strips



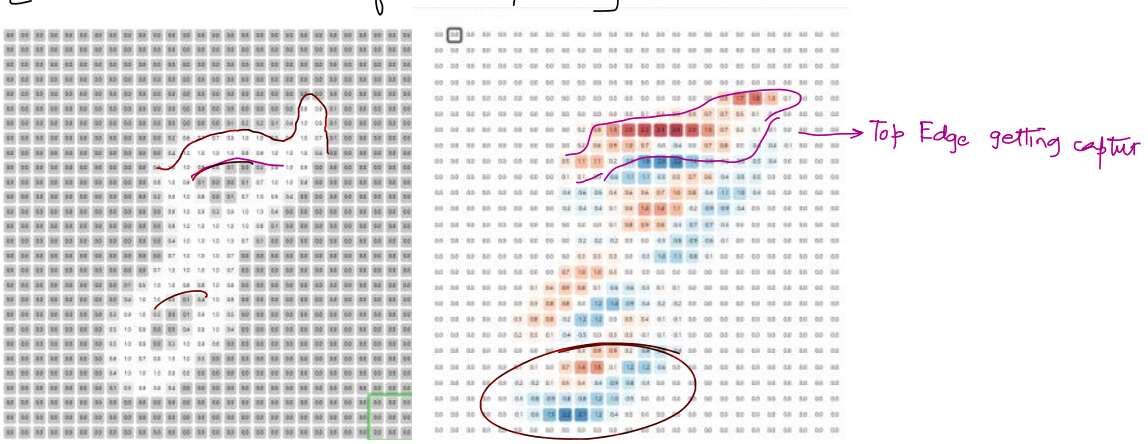


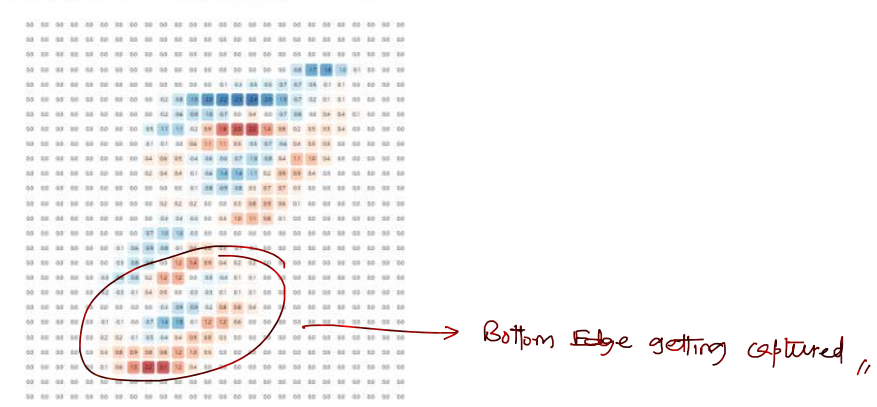
categorized / classified as a

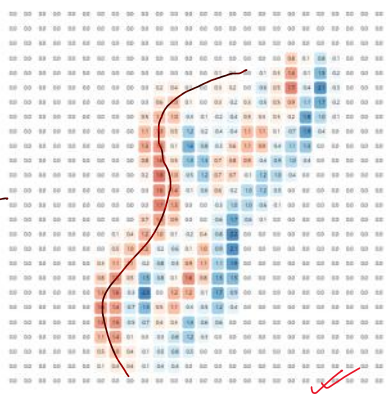
DOG

https://deeplizard.com/resource/pavq7noze2

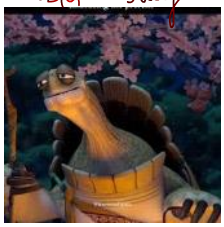
[Mathematical Interpretation of filtering operation]







'Master Oogway'

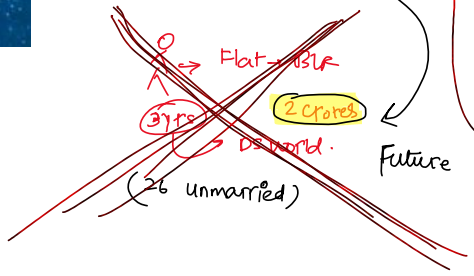


Past.

Future

Present

+ (Phone has to be in silent) → [Push calls] →



[exceptions + important calls allow → filtering]

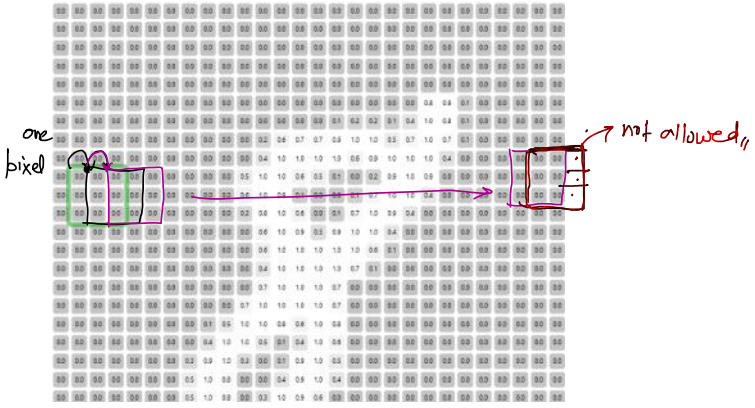
*** Pb-tib

Pic-tip

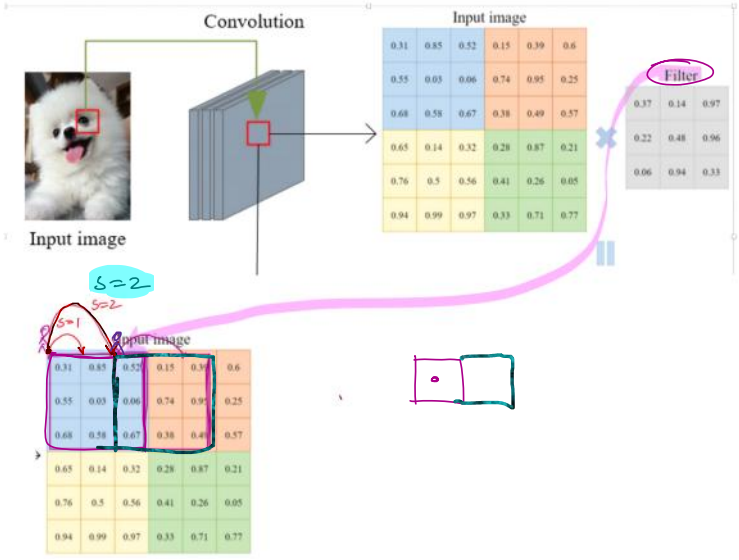
STRIDE & PADDING

Stride: Stride is a hyperparameter in CNN that determines the no. of pixels by which the filter (aka kernel) moves across the input image during the convolution operation.

Default setting S = 1 → Filter moves one pixel at a time



Stride # S = 2 ← Filter jumps by 2 pixels at a time → reducing the overlap and hence the filter map size



Why stride matters ?

- small (s = 1) → high resolution output with more details captured at the granular level → computation would be slower.
- large (s = 2 or more) 2, 3 or 4 → small resolution output → feature maps' size would be smaller → better downsampling ↓

It is faster in computation

— may loose fine details of the input image

* For any small input images (upto 100 x 100 pixels) → stride # 2 is the most recommended.
* In some cases or large input images (upto 224 x 224 pixels) → stride # 2 or more can also be used but less frequently and generally in deeper layers to reduce the feature size for computational efficiency while not losing on essential features of the image.

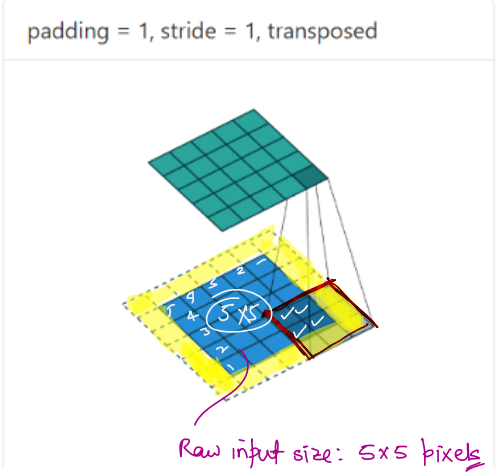
Padding

— margin / some room → to the images

In such cases, it is evident that padding is needed.

Padding is a technique used in CNNs to adjust the spatial dimensions of the input image

- it involves adding extra row(s) and/or column(s) pixels usually with zero around the border of the input image



Before padding

p=0 and s=1

size of the feature map??

Formula for feature map dimensions

Height of feature map

$$H_{out} = \left[\frac{H_{in} - K + 2p}{s}\right] + 1$$

where $[\cdot]$ → Greatest Integer Function (Floor)

$[2.3] \rightarrow 2$

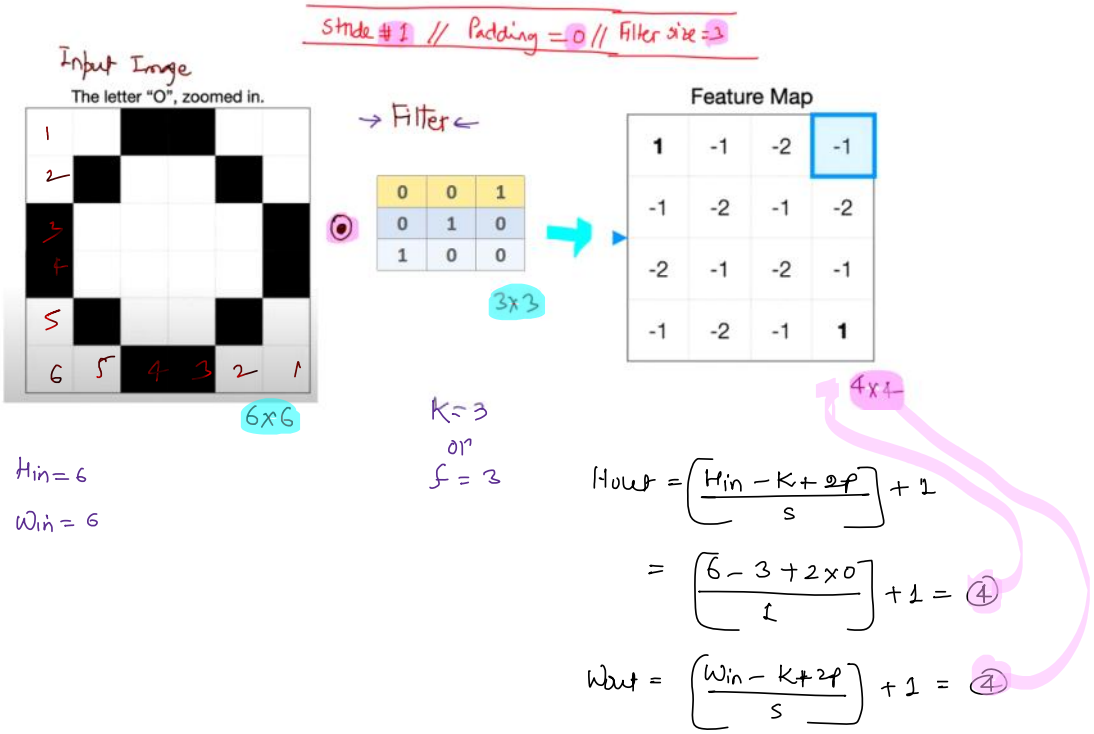
$[3.6] \rightarrow 3$

Width of the feature map

$$W_{out} = \left[\frac{W_{in} - K + 2p}{s}\right] + 1$$

where

- **Input dimensions**
  - H_{in}: Height of the input image
  - W_{in}: Width —"— —"—
- **Filter parameters**
  - K or f: size of the filter (assuming square matrix of order K)
  - s: stride of the filter
  - p: Padding for the filter
- **No. of filters**
  - F: no. of filter applied → which determines the depth of the output



$H_{in} = 6$

$W_{in} = 6$

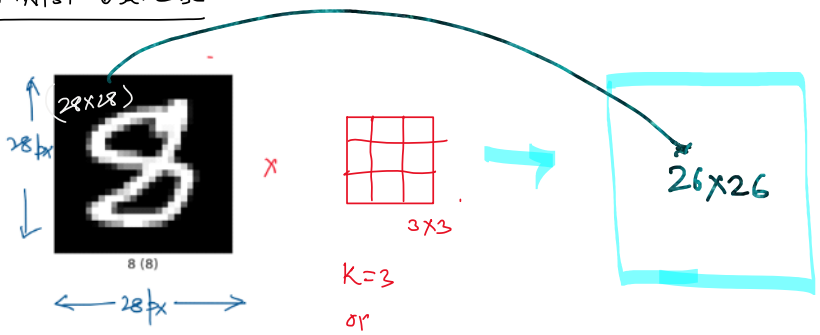
$K = 3$ or $f = 3$

$$H_{out} = \left[\frac{H_{in} - K + 2p}{s}\right] + 1 = \left[\frac{6 - 3 + 2\times 0}{1}\right] + 1 = 4$$

$$W_{out} = \left[\frac{W_{in} - K + 2p}{s}\right] + 1 = 4$$

Feature Map dimensions : 4 X 4

MNIST Use-case

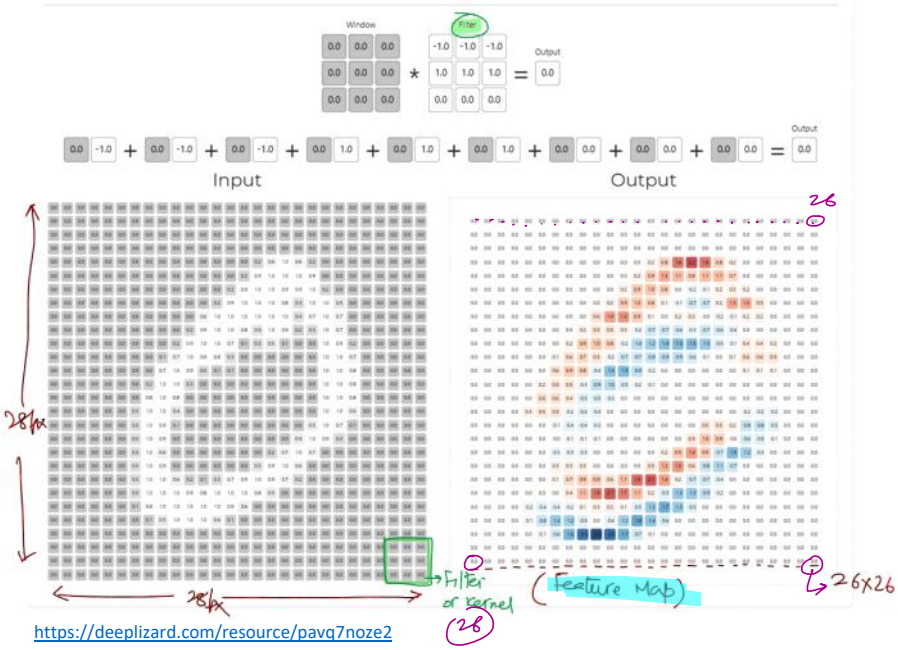


K=3
or
f=3

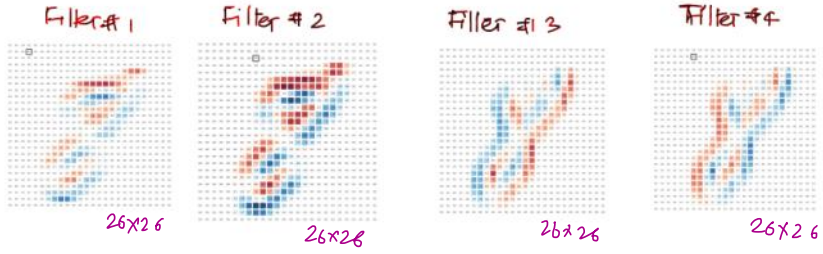
Assume s=1 and p=0,

$H_{out} = \left[\frac{28-3+2\times0}{1}\right] + 1 = 26$

$W_{out} = 26$



https://deeplizard.com/resource/pavq7noze2



Why is padding so important ?

Joydeep - but how do I understand that we need padding ? In training there will be many images

Padding is crucial to :

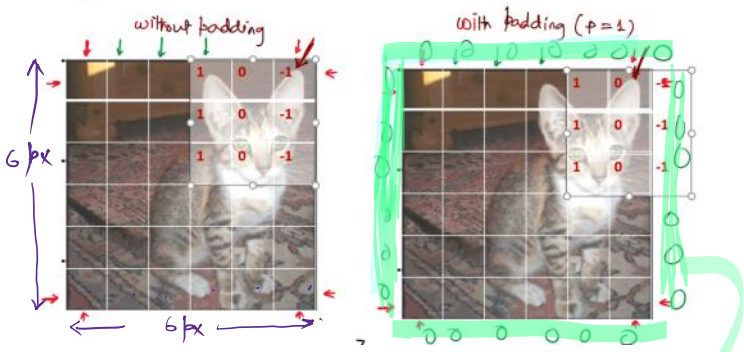
a) Preserve input dimensions

Without padding, the o/p dimensions (Feature Map (F_{map}) size) of a convolutional layer decrease after each convolution operation step

Padding helps to maintain the spatial size of the input

operation step

Padding helps to maintain the spatial size of the input specially when using convolutional layers.



input image : 6×6

filter : 3×3

Fmap : $\left[\frac{6-3+2\times0}{1}\right]+1 = 4$

Fmap : 4×4

with padding:

Fmap size: $\left[\frac{6-3+2\times1}{1}\right]+1$

$= \left[\frac{3+2}{1}\right]+1 = 6$

$= 6 \times 6$

b) Retain the edge information (Border/edge cases)

Without padding, edge pixels along the border lines are used less frequently during the convolution step

↓

causing loss of boundary information

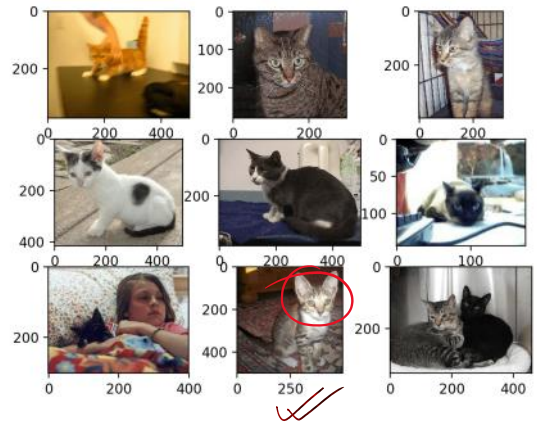
↓

Hence, padding ensures filters can process the edges properly.

Note: Without padding, edge pixels are under-represented and the model may perform poorly for objects near the border of the image.

Hence, padding assures that every pixel including those on the edges too, is processed by filter(s) in a proper way.

Task: Write a function to find the images where objects are border / edge cases?



Padding types:

- Valid: No padding → output is smaller than the input

Fmap: Feature Map

- Same: Padding added to keep the output (Fmap) same as the input

Input size = Fmap size

6x6 → 6x6 no change → same.

Best Practices

- Use 'valid' padding when reducing size is intentional
- Use **'same' padding** for conv layers to maintain spatial size (especially when stacking many layers)

Use-cases

* Image classification & segmentation → (same Padding)
* Autoencoders → (valid Padding) → (reduce the size of the array)

↳ Autoencoders are a special type of neural networks that learn to compress data into a compact form and then reconstruct it to closely match the original input

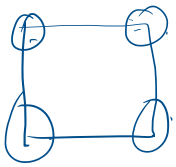
- facial recognition
- anomaly detection

Joydeep -How do I understand that we need padding or not? In training data, there are many images and how can be sure about finding the % of such border or edge cases?

Task: Write a function to find the images where objects are border / edge cases?

Read all the images → convert to grayscale → to simplify processing
Create Binary Mask using adaptive threshold

Read all the images → convert to grayscale → to simplify processing
Create Binary Mask using adaptive threshold
Extract all contours from the binary image
↳ select the largest contour
↓
object.
For the largest contour, x, y, w, H.



Pooling Layers

- also known as sub-sampling

Downsampling vs Oversampling vs sub-sampling

Downsampling : is a technique to reduce the size of the data usually by decreasing:
- no. of samples
- frequency
- dimensionality
- resolution

Why downsampling → a) Faster Training
b) Avoid bias towards the majority class.

Oversampling: increasing the number of samples for the minority class
- Random oversampling
  - duplicating minority samples.
- SMOTE (Synthetic Minority Oversampling technique)
  - Generates new synthetic samples by interpolation

Why oversampling: a) helps to solve class imbalance.
b) No data loss

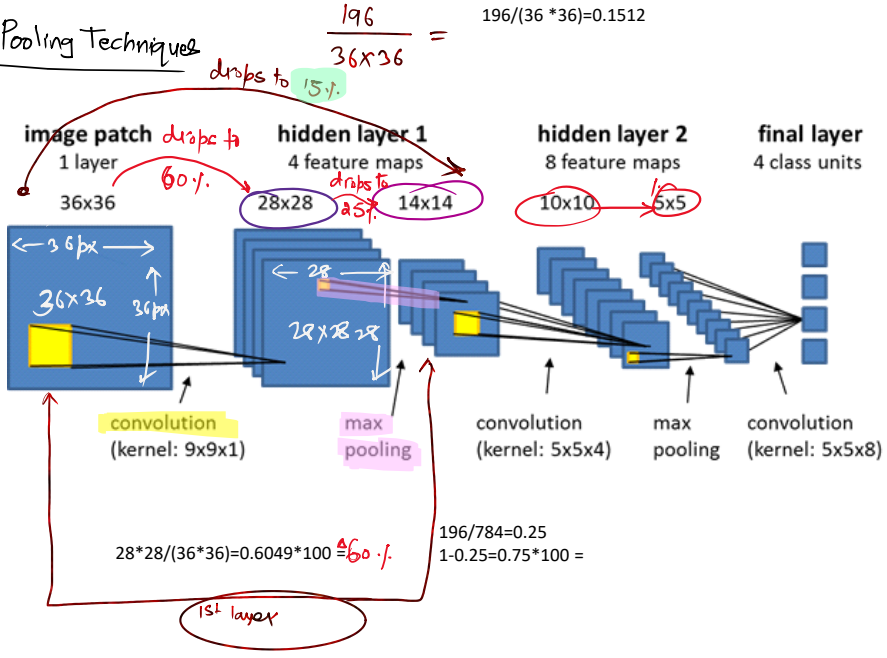
Sub-sampling → reducing spatial or temporal resolution using the below pooling techniques
- Max Pooling
- Avg. Pooling

Avg. Pooling

Min. Pooling

Pooling Techniques

196/(36 *36)=0.1512



Why is pooling so important ?

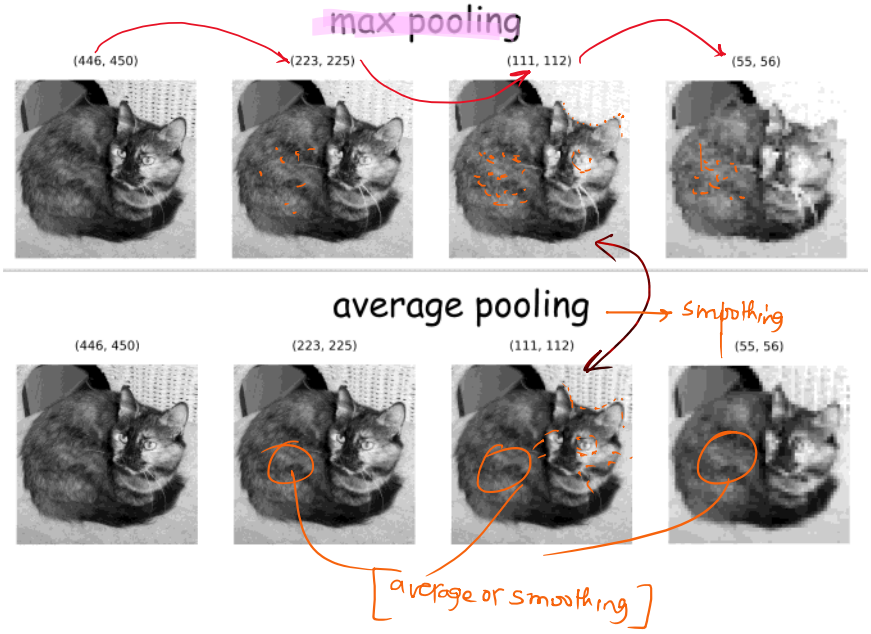
- Pooling reduces the computational cost and improves the efficiency by reducing the no. of pixels
  ↓
  reducing the no. of trainable parameters

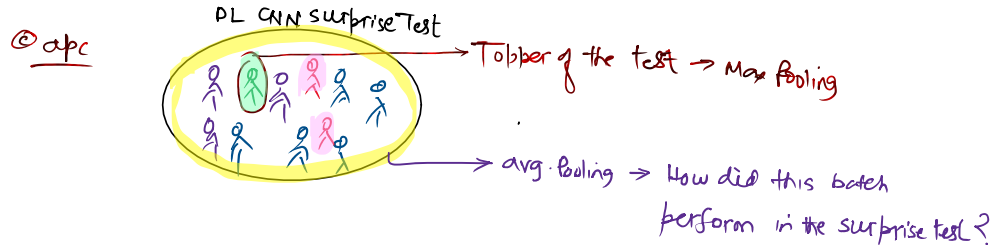
- Pooling prevents overfitting as well → by retaining only significant features

Note: Pooling is a crucial layer in CNNs that reduces the spatial dimensions (width & height) of feature map(s) while retaining the essential information

Pooling is a sub-sampling technique applied to feature map(s) to reduce their size. It works by summarizing a region of feature map into a single value (within overlapping grid)

- → Max. Pooling
- → Mean/Avg. Pooling
- → Minimum Pooling ✗

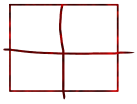


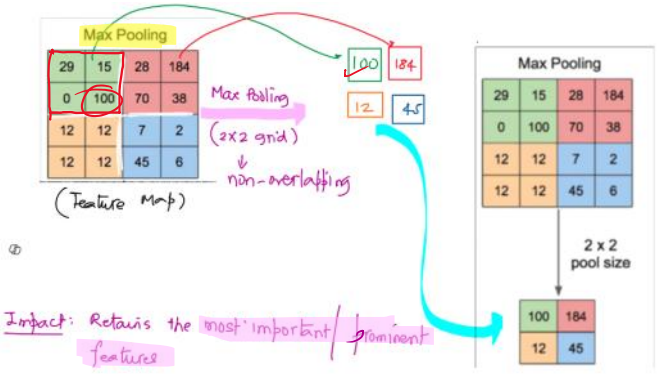


① Max Pooling

- divide the feature map(s) into non-overlapping regions
- and outputs the maxm value from each region

[2x2 grid/ window]



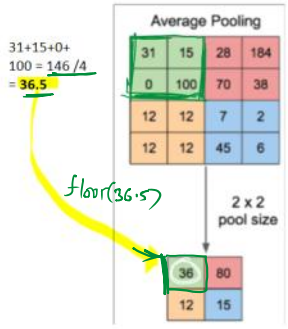


2. Mean/Avg. Pooling

- divides the input (feature map) into non-overlapping regions using 2x2 grid.
- and outputs the avg. (mean) value from the each non-overlapping

using 2x2 grid.

- and outputs the avg. (mean) value from the each non-overlapping region



Impact: Mean/Avg. pooling operation smoothens the feature by averaging

In max pooling, petal edges and flower center become very sharp and bright

In avg/mean pooling technique, the model tries to learn overall shapes/patterns in the flower rather than sharp edges.

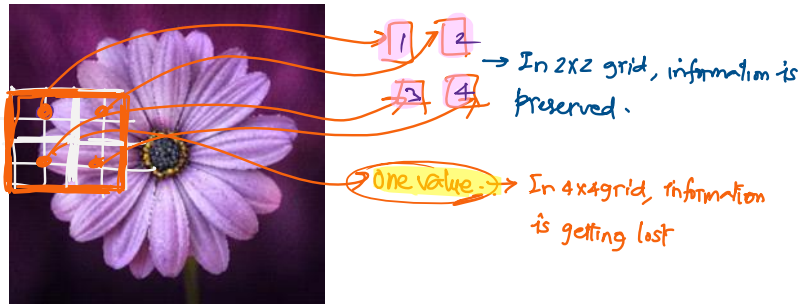
In CNN, max pooling technique is used to detect strong features or edges.

Best practices for pooling layer

Pooling window size 2x2 grid is very common for the below reasons:

- it is computationally efficient
- it reduces

- it is computationally efficient
- it reduces feature map size by exactly half in both the dimensions → width, height
- It balances down-sampling speed and information retention.



Gaurav - so, when we compress image online, does it apply pooling ?

No → online image compression doesn't use max or mean pooling techniques

↓

JPEG uses chroma subsampling (4:2:0)

- luminance (brightness) kept at full resolution
- color channels stored at half resolution