CNN Detailed Discussion

28 September 2024 21:41

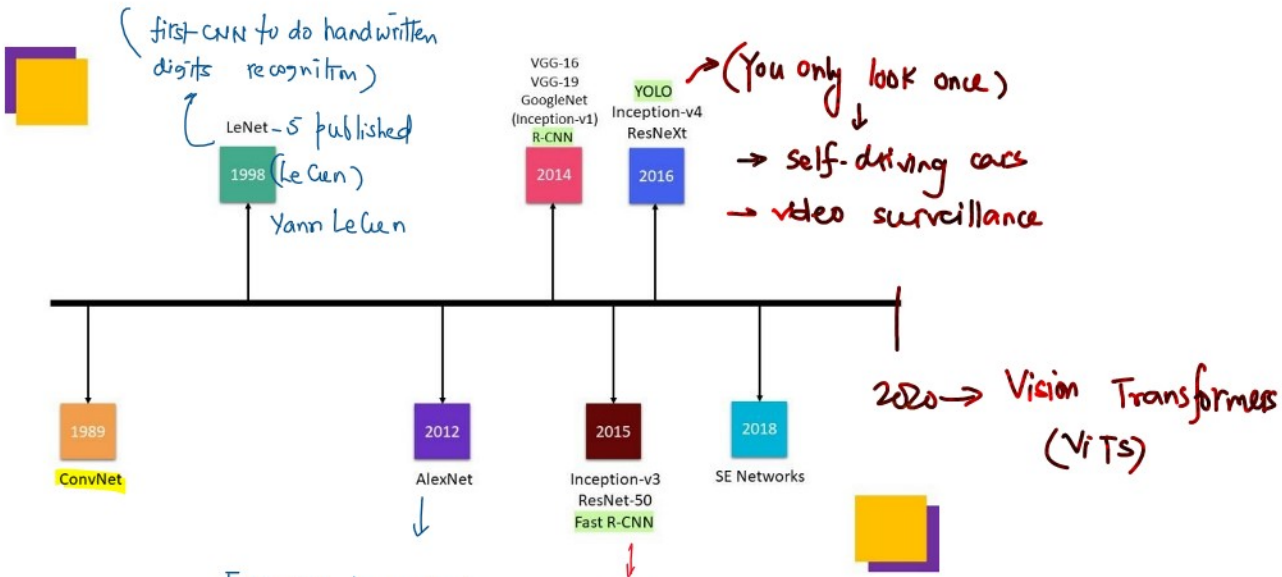
CNNs have proven very effective in image recognition and classification.

- Really successful in identifying faces, objects and traffic signs apart from powering vision in robots and self-driving cars.

a soccer player is kicking a soccer ball

a street sign on a pole in front of a building


a couple of giraffe standing next to each other



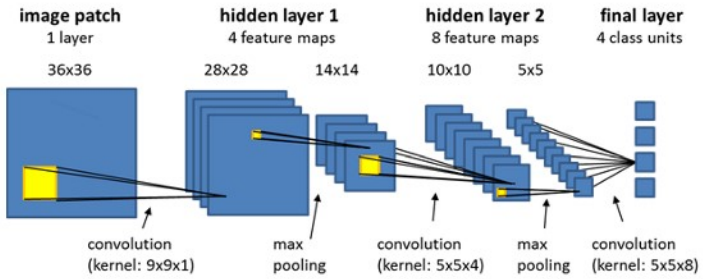
ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

Training (upto 152 layers)

(- ReLU activation function
- Dropout
- GPU acceleration)

⇓

(to make deep CNNs feasible for large-scale classification tasks)



① ReLU Activation
② Dropout
③ Batch Normalization
④ Transfer learning

} CNN (recent developments)

Architecture overview

A typical CNN architecture consists of several key layers:

1. Input Layer

- the input layer of a CNN receives the raw image data in the form of a multi-dimensional array (also known as Tensor)

For grayscale images

input Tensor → (H x W x 1)

* H: Height of the image (no. of pixels vertically)
* W: width of the image (no. of pixels horizontally)
* 1: single color channel - grayscale.

For color (RGB) images:

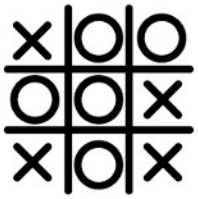
input tensor → (H x W x 3)

↳ 3 represents the three color channels - red, blue green.

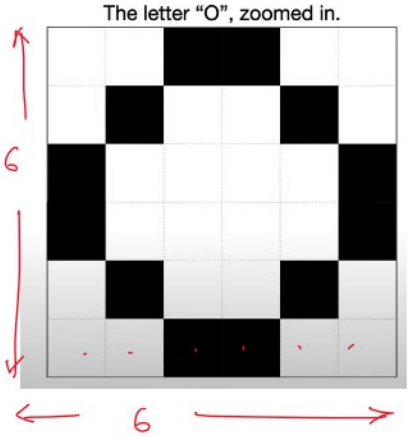
2. Convolutional Layer

- core building block of CNN
- it applies filters (also known as kernels) to extract features like edges, corners, textures etc
- the filters slide over the input and this process is called convolution and it generates a feature map as output

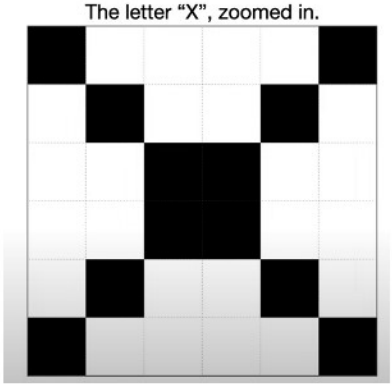
Tic- Tac- Toe



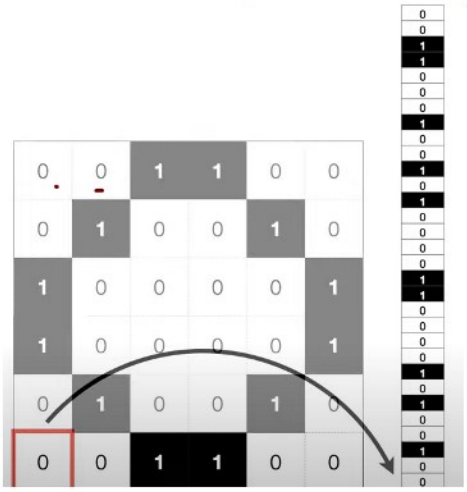
6x6 pixels



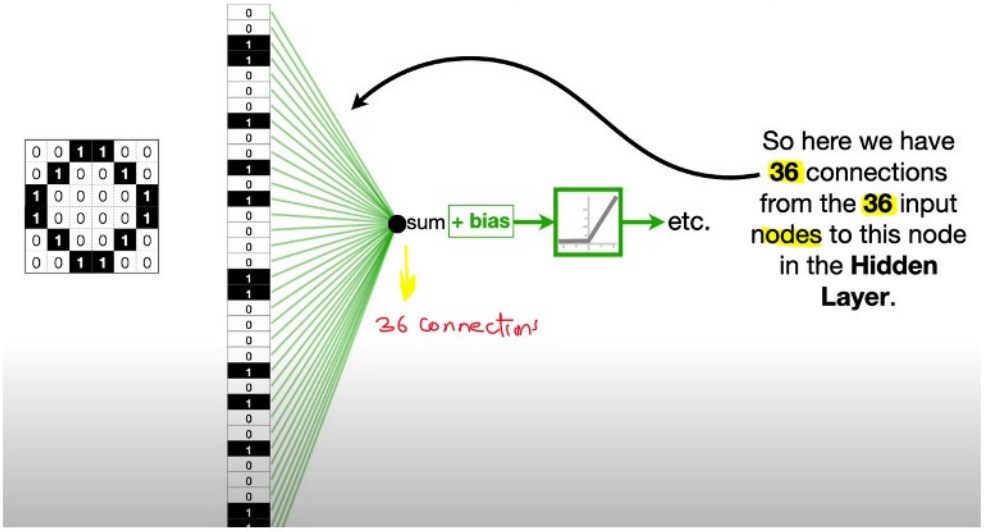
6x6 pixels

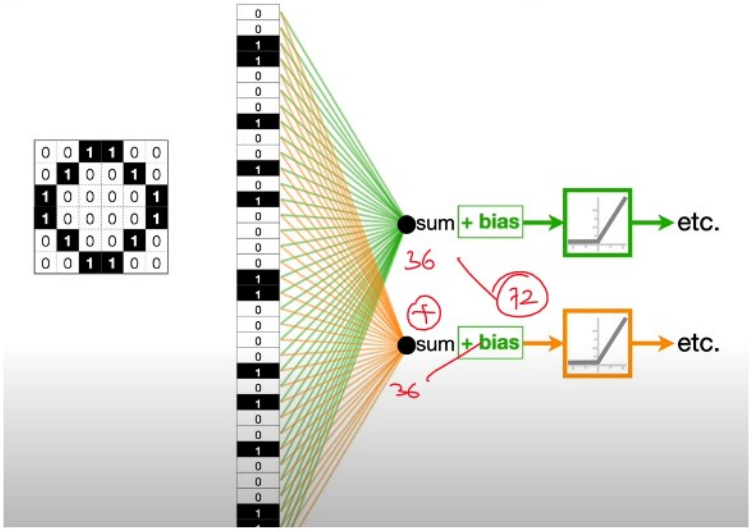


In artificial neural n/w,



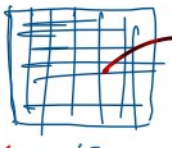
one single column of 36 nodes (1D - flattened array)





1 Mega pixel → 1×10^6 pixels

$\sqrt{10^6} = 1000$ pixels



→ 1 mega pixel image.

1000 pixels

● → one neuron → (input) (10^6 connections)

For a neural network having 1×10^6 (input) connections for a standard size image the model

needs to estimate 1 Million weights for each neuron which is computationally highly expensive.

↓

Applying neural networks (vanilla) is not the right approach as it doesn't scale well.

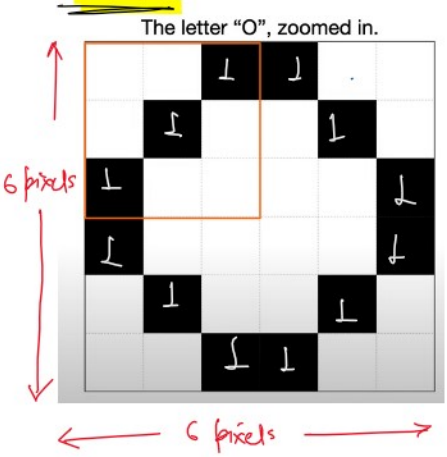
↓

CNN comes and simplify it and make it computationally less heavy.

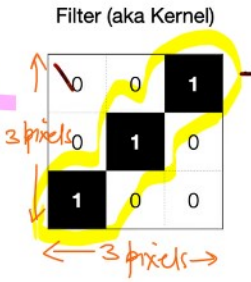
CNNs do three things to make image classification practical:

1) Reduce the no. of input nodes
2) Tolerate / accommodate the marginal shifts in the pixels within the image.
3) Mind the correlations in the complex image.

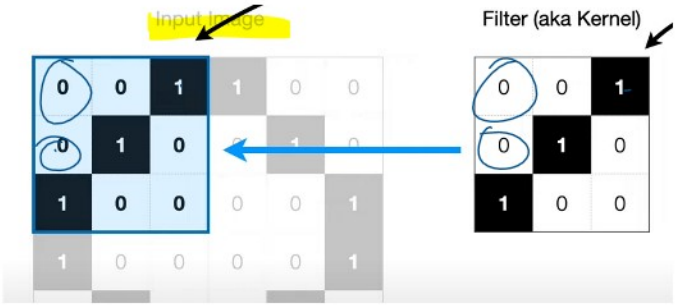
Step #①



CNN applies a filter to the input image.



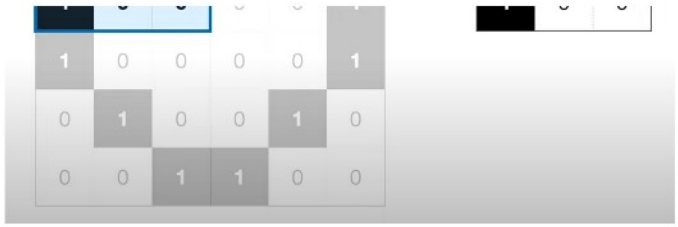
→ on off-diagonal, values are 1 and rest all zero



In convolution, dot product between the input and the filter in the overlapped area, and hence we can say that the filter is convolved with the input.

$(0\times0)+(0\times0)+(1\times1)$

dot product

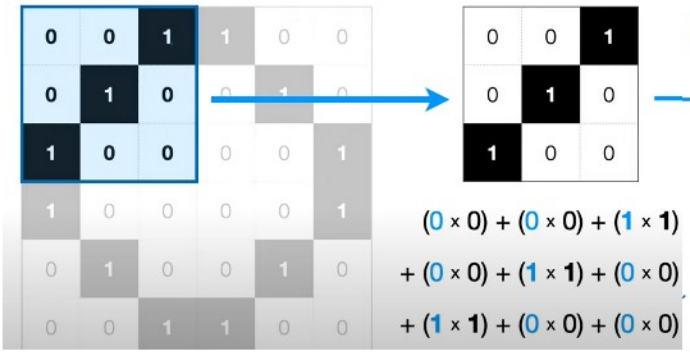


the input:

$(0\times0)+(0\times0)+(1\times1)$
$+(0\times0)+(1\times1)+(0\times0)$
$+(1\times1)+(0\times0)+(0\times0)$

dot product
↓
sum of products

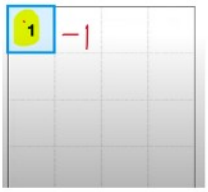
$= 3 \rightarrow$ bias $(-2) \rightarrow 1$

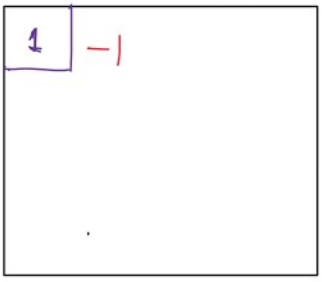


$(0\times0)+(0\times0)+(1\times1)$
$+(0\times0)+(1\times1)+(0\times0)$
$+(1\times1)+(0\times0)+(0\times0)$

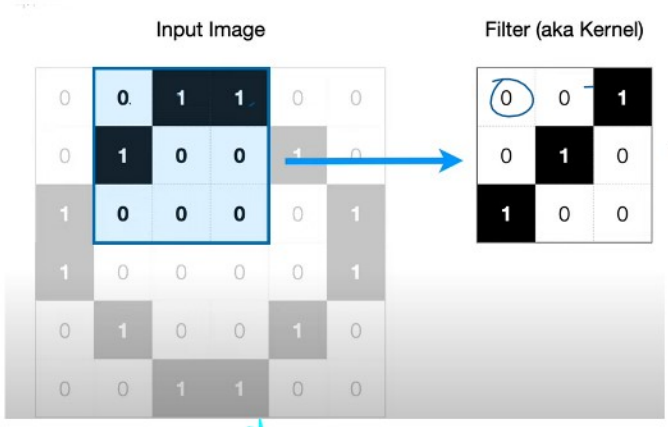
$(3-2)$ (bias -2)

Feature Map





Next shift by a pixel:



$(0\times0)+(1\times0)+(1\times1)$
$+(1\times0)+(0\times1)+(0\times0)$
$+(0\times1)+(0\times0)+(0\times0) = 1$

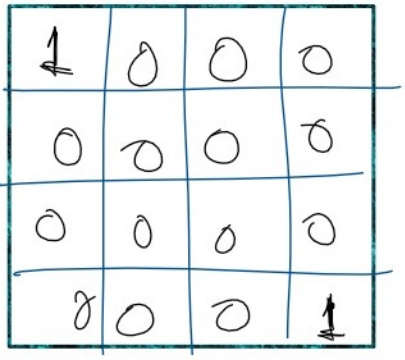
bias (-2)
↓
-1

(Final Feature Map)

Feature Map

1	-1	-2	-1
-1	-2	-1	-2
-2	-1	-2	-1
-1	-2	-1	**1**

ReLU →



-1	-2	-1	1

1	0	0	1

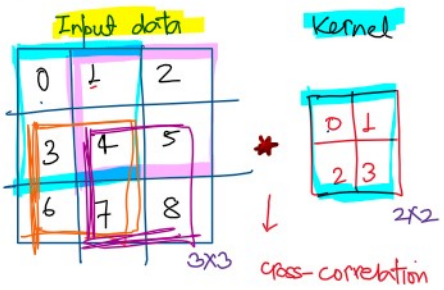
The convolutional layer is the core building block of CNN. It applies filters (also known as kernels) to the input data to extract features like edges, corners, textures etc.

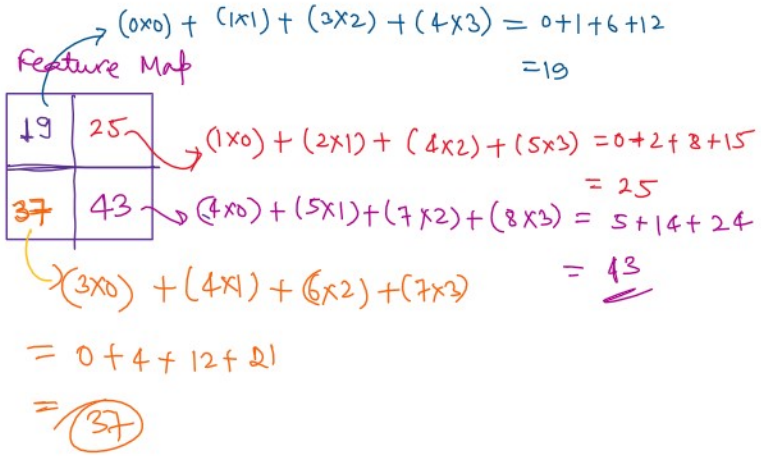
Key terms:

① Filter / Kernels – already covered.
- Basically filters are small matrices such as 3x3 or 5x5 which are convolved with the input image

② stride – it's the step size at which the filter moves over the input image.

A stride of 1 means the filter moves one pixel at a time.





data: $d_h \times d_w$ (3x3)

kernel: $K_h \times K_w$ (2x2)

Feature Map $= (d_h - K_h + 1) \times (d_w - K_w + 1)$

$= (3 - 2 + 1) \times (3 - 2 + 1)$

$= 2 \times 2$

Q1 For a 5x5 input image and 3x3 filter (kernel), and a stride of 1, the output feature map will be of the size 3x3.

$(5-3+1) \times (5-3+1)$

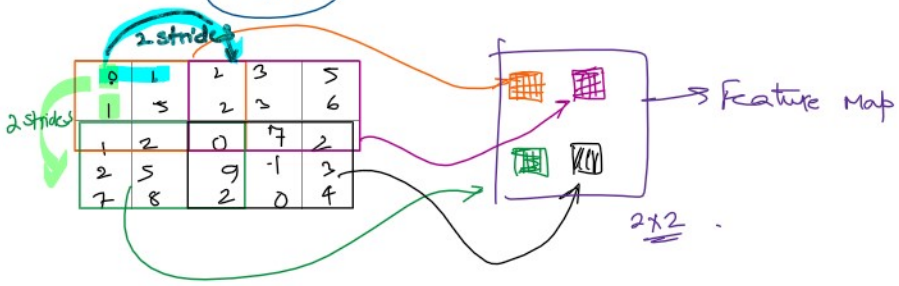
$= 3 \times 3$

Q: For the same question, what's the output feature map with stride = 2

$$\text{Feature Map} = \left[\left(\frac{d_h - k_h}{s}\right)+1\right] \times \left[\left(\frac{d_w - k_w}{s}\right)+1\right]$$

$$= \left(\frac{5-3}{2}+1\right) \times \left(\frac{5-3}{2}+1\right)$$

$= 2 \times 2$



Pro-Tip

Why do we use strides of 2 or more in CNN?

1. Reducing the spatial dimensions (downsampling)

Using strides of 2, the convolution filter moves two pixels at a time across the input image which reduces the size of the o/p feature map.

a) Fewer parameters: By reducing spatial dimensions of the feature map, no. of parameters in subsequent layers decreases

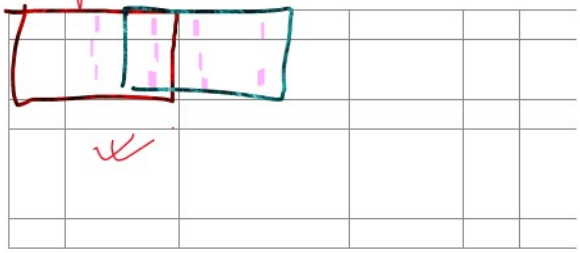
b) Prevents overfitting: With reduction in feature map size, it acts like a form of regularization as CNN model captures important insights from the

as CNN model captures important insights from the the data.

Note: In practice, the most commonly used strides in CNNs are 1 and 2

2. Faster Computation

Using a stride of 2 or more results in fewer convolution operations because the filter moves more quickly over the image, effectively reducing the no. of times convolution operation is applied.



Impact of input size on stride selection

MNIST dataset

→ small size image

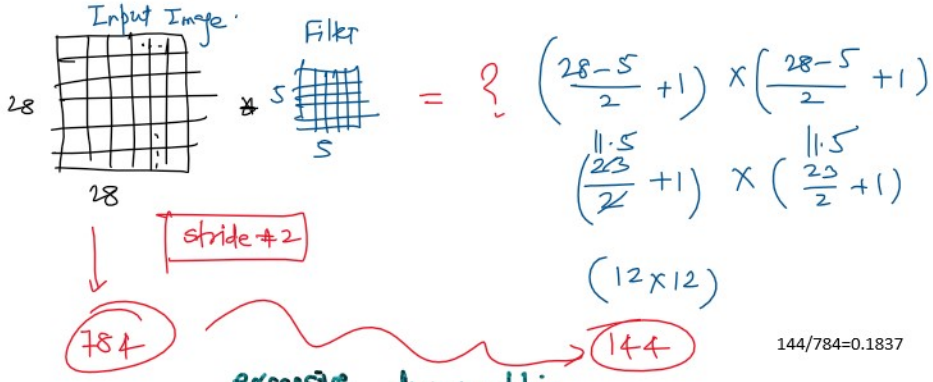
- Using stride 2 or more can cause excessive downsampling, leading to a significant reduction in spatial dimensions (size) of the feature map which in turn can result in the loss of important fine-grained information.

Best practices for small inputs | small size image

stride = 1 is the most recommended for smaller images → slide the filter over the image without skipping.

stride 2 can be used but not often in deeper layers to reduce the feature map size for computational efficiency, but only after initial layers have

efficiency, but only after initial layers have captured sufficient features.



excessive downsampling by 82%.

↓

this limits the n/w's ability to extract detailed features in later layers.

Best practices for large input sizes | images

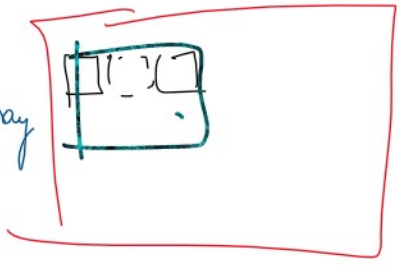
When the input image is large (224x224) – (ImageNet dataset), using stride 1 throughout the n/w may result in unnecessarily large feature map — leading to high computational costs and memory usage.

Hence, larger strides (stride = 2) can be used to efficiently downsample the input and reduce the size of the feature maps without losing too much information.

Impact of Input size on Kernel or Filter Selection

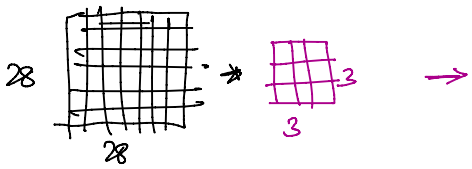
Smaller Input Sizes:

For smaller input sizes, using larger kernels e.g (7x7) may result in excessive smoothing, causing the n/w model to miss fine-grained features and small details.



t | a . Smaller kernels e.g. (3×3) are preferred when working with small input sizes because they focus on smaller local regions of the image

Smaller kernels are particularly effective in initial layers



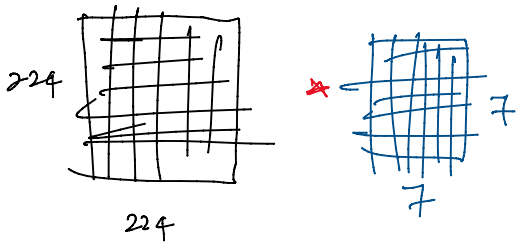
Larger Input Size:

For larger input sizes, smaller kernels may not be able to capture higher-level, broader patterns effectively because they only focus on small patches of the image.

| d

Larger kernels (e.g. 5×5 or 7×7) can be used in initial layers to capture more contextual information from the image and reduce the depth required for feature extraction.

In a 224×224 input image, a (7×7) kernel is often used in the first convolutional layer to capture larger patterns like textures and shapes — followed by smaller kernels in subsequent layers to refine the segments.



(224 × 224) (7×7)

Feature Map

$= \left(\frac{224-7}{2}+1\right) \times \left(\frac{224-7}{2}+1\right)$

$= \left(\frac{217}{2}+1\right) \times \left(\frac{217}{2}+1\right)$

$= 109 \times 109$

Pa r m| m N a h l clu

1. LeNet

For small inputs like (28x28)

Kernel size : 3x3

stride : 1

2. Alexnet

For large inputs like (224 x224)

Kernel size : 11 x11

stride : 4.

3. VGGNet

For large inputs :(224 x 224)

Kernel size : 3x3

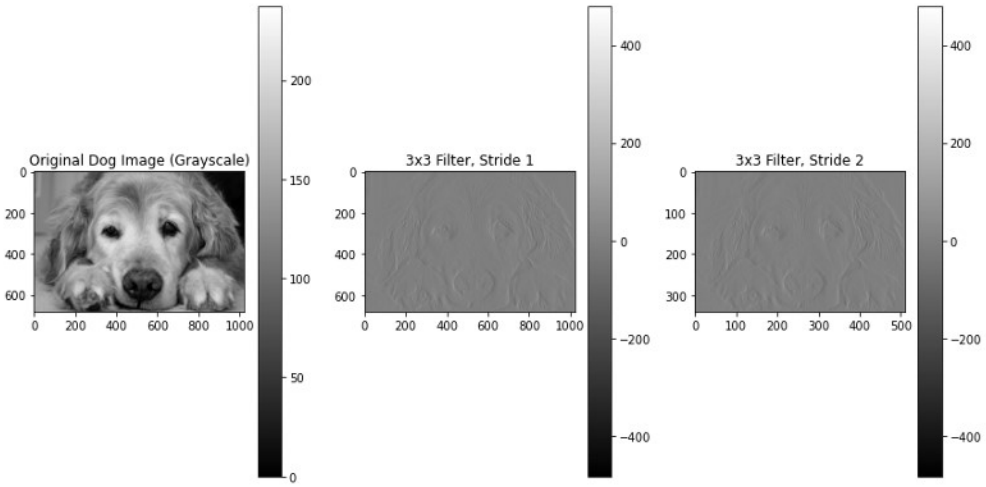
stride : (1) followed by (2)

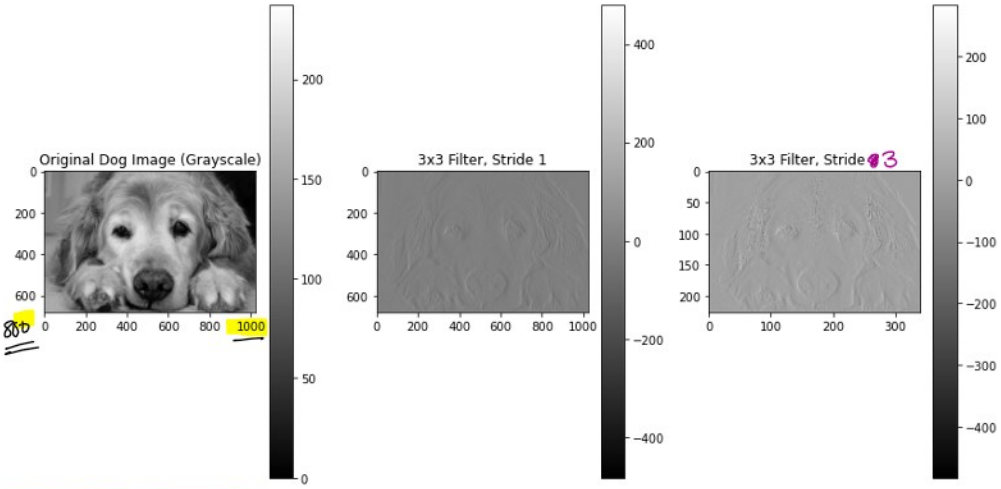
4. ResNet

For large inputs :

Kernel size : ~~7x7~~ 3x3

stride : ~~2~~ 1



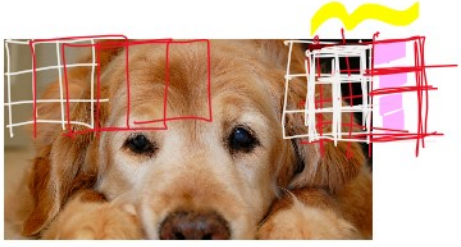


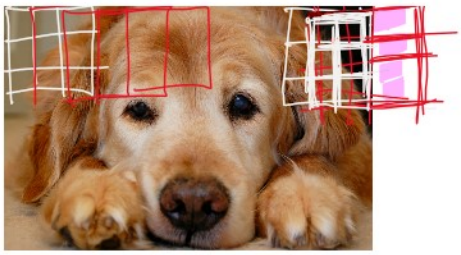
Concept of Padding in Convolutional Layer

In convolutional Neural Networks, padding is used to control how the convolution operation handles the boundaries of an image, ensuring that the filter

can process every part of the image, including the borders.

Q: Why do we need padding?





When a filter slides over an image, it starts at the top-left corner and moves across the image. However if there is no padding, the filter (kernel) cannot fully cover the pixels near the edges of the image, (boundaries) Due to this, it results in convolution output being smaller than original input image.

Padding is used to address this by adding extra pixels usually zeros around the borders of the image.

Reasons to use padding:

- Preserve information at the edges
- Control the output size — padding allows you to control whether the output feature map has the same size as the input or if it shrinks

Types of Padding

a) Valid Padding (No padding)

In valid padding, no padding is added to the input image.

Formula for calculating output size with valid padding:

$$\text{output size} = \frac{(\text{Input size} - \text{Filter size})}{\text{Stride}} + 1$$

where

Input size: size of the input image.

Filter size: size of the filter (kernel).

Stride: step size of the filter.

Stride: step size of the filter.

Example: Input Image: 5 x 5

Filter Image: 3 x 3

Stride : 1

$$\text{output size})_{\text{valid padding}} = \left(\frac{5-3}{1}+1\right) = 3$$

↓

size of the output : 3 x 3

With valid padding (no padding), the output feature map is smaller than the input image

(5 x 5) (3 x 3)

Note: Valid padding (No padding) is useful when the primary goal is to reduce the size of the feature map

b.) Same Padding (Zero padding)

In same padding, the input image is padded with zeros around the border so that the output feature map has the same spatial dimensions (same height and width) as the input image.

Formula: $\text{output size} = \left(\frac{\text{Input Size}}{\text{stride}}\right)$

In above formula, the output size = Input size but only stride = 1.

To achieve same padding,

$$P = \left(\frac{F-1}{2}\right)$$

where

P: is the amount of padding to add

F: is the filter size.

For a filter size 3x3, P is:

$$P = \frac{3-1}{2} = \frac{2}{2} = 1$$

→ it means that we need to add 1-pixel border of zeros around the image to ensure the output size remains the same as input size.

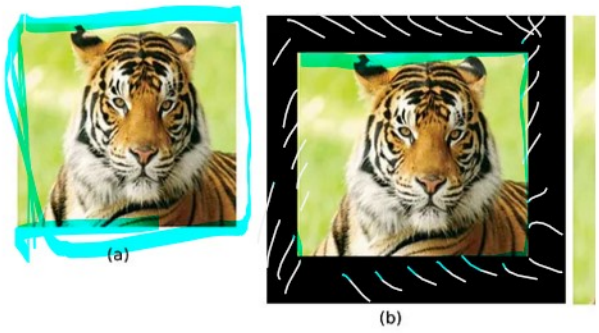
Example:

Input Image: 5x5

Filter size : 3x3

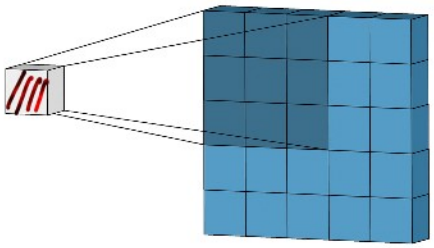
stride: 2

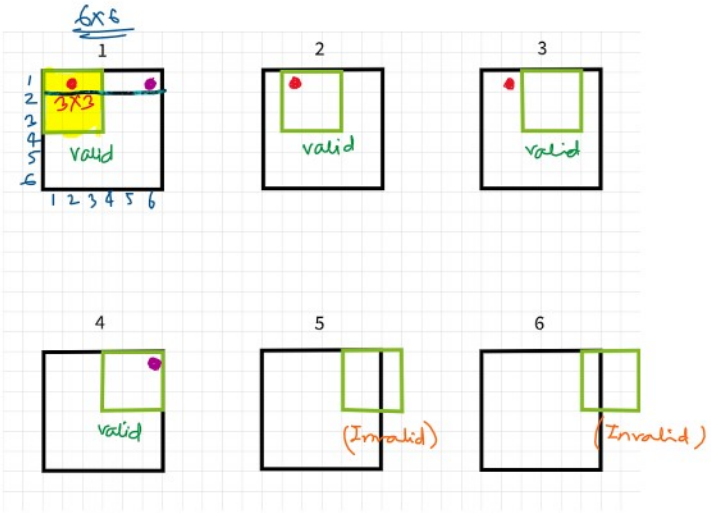
output size $= \frac{5}{1} = 5x5$



A small image (a) resized to 180 x 180 pixels using zero-padding

The Lost Pixels





To illustrate, consider a 6x6 pixel image convolved upon by a 3x3 filter. As you can see above, the first 4 convolutions fall within the image to produce 4 pixels for the first row, while the 5th and 6th instances fall out of bounds and hence ignored.

6x6 Image

3x3 filter

$$\text{Feature Map Size} = \left(\frac{6-3}{1} + 1\right)$$

$$= (3+1) = 4 \rightarrow 4\times4$$

Implication of lost Pixels

Losing 2 rows and 2 columns might not seem to have much effect with large image.

Feature Map = 4x4

Image size = (6 x 6)

UHD Image : (3840 x 2160)

= 2/2160=0.0009

0.0009x100 %.

= 0.009 %.

For RESNET architecture (128)

50 convolutional layers (3x3)

↓

Filter size.

1 layers → 2 rows

2 columns

50 layers → 2x50 = 100 rows

2x50 = 100 columns

Input image: (3840 X 2160) 3840 * 2160 = 8294400

↓

output image: (3740 X 2060) 3740*2060 = 7704400 7704400 /8294400 = 0.9289

1-0.9289=0.0711*100=**7.11%**

ROW	1- (3740/3840)=0.026*100 = 2.6%
COLUMN	1 - (2060/2160) = 0.0463 *100 = 4.63%

Total loss = 2.6 + 4.63 = 7.23 %

MNIST Database

Image size : 28 X 28 → 784 pixels

A CNN with 4 convolutional layers.

↓

rows = 4x2 = (8)

Columns = 4x2 = (8) } lost pixels

28 X 28 → 784 pixels

↓

20x20 → 400 pixels

1- (400/784) = 0.4898*100 = 48.98 %

[without padding]

original convolved

[with padding]

original convolved

dark black borders

3. Activation Layer (ReLU)

- Most commonly used activation function in CNNs is Rectified Linear Unit,

$$ReLU(x) = \max(0, x)$$

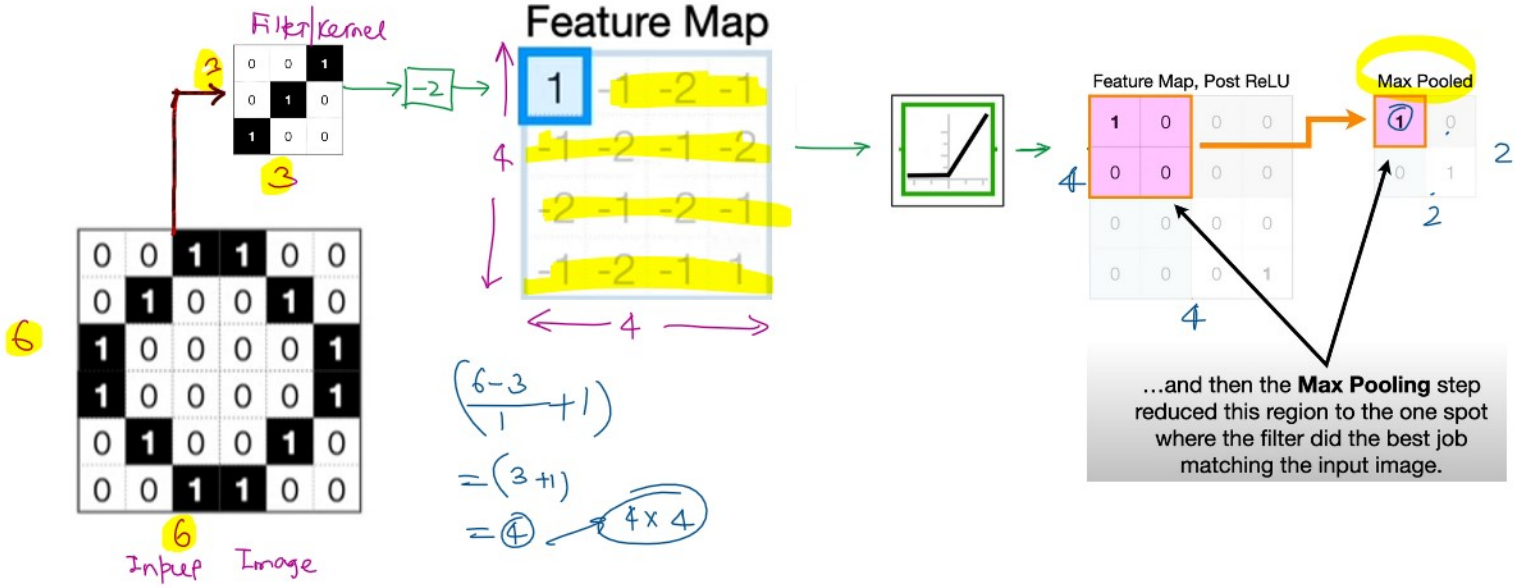
- ReLU sets all negative values to zero and keeps positive values unchanged, helping to prevent issues like vanishing gradients.

4. Pooling Layer (subsampling)

Pooling layer is used to reduce the spatial dimensions (height and width) of the feature maps, helping to computational cost, prevent overfitting.

How does pooling work?

Pooling operates on a small window (2x2) and aggregates information from these windows by either selecting maximum value (max pooling) or the average value (Mean/ Avg. Pooling)



$\left(\frac{6-3}{1}+1\right)$

$=(3+1)$

$=4$ → 4 x 4

Overfitting occurs when a model learns to perform very well on the training data but fails to generalize to unseen test data

Pooling helps with overfitting as it reduces the spatial dimensions of the feature maps ⟶ the number of

Pooling helps with overfitting as it reduces the spatial dimensions of the feature maps ⟶ the number of parameters and computations in the n/w is reduced.

↓

Fewer parameters mean the model is less likely to memorize the Training data

↓

model is less likely to be overfit

Example: 32 x 32 : Feature Map.

↓

After applying a 2x2 max pooling with stride # 2

$$\text{output size} = \left[\frac{(\text{Input size} - \text{Filter size} + 2 \times \text{padding})}{\text{stride}} + 1\right]$$

$$= \left[\frac{(32 - 3 + 2 \times 0)}{2} + 1\right]$$

$$= \left[\frac{29}{2} + 1\right] = 16$$

output size = (16 x 16)

32 x 32 —Max Pooling with stride # 2→ 16 x 16

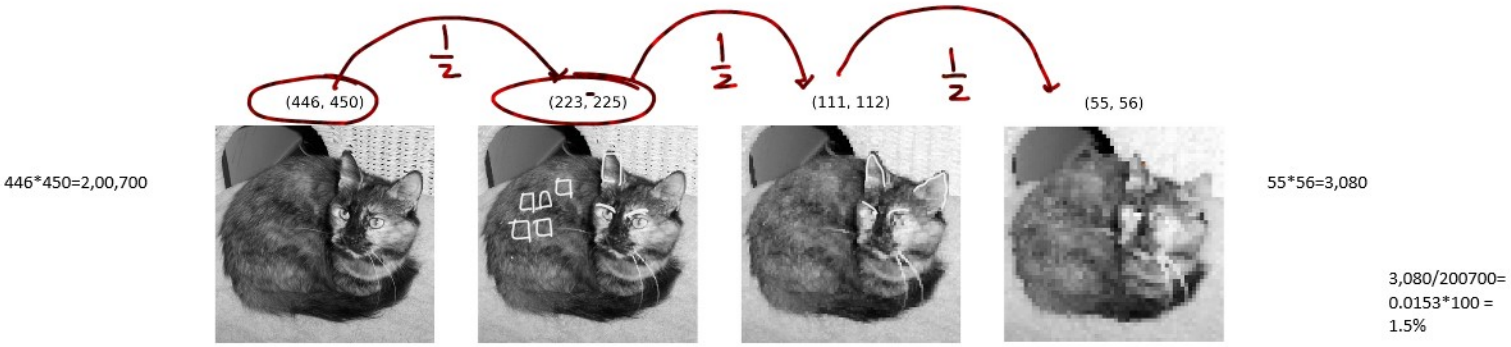
2x2

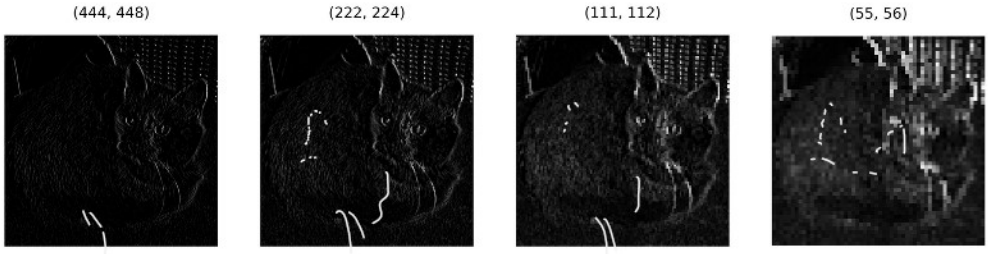
32 x 32 / 16 x 16 ≈ 4 times

Note:① After doing max pooling with stride # 2, spatial

Note:① After doing max pooling with stride #2, spatial resolution of the feature map reduces by 4 times, meaning the following layers have to process much smaller inputs, reducing the overall complexity of the network.

Note #② Pooling layers, particularly max pooling help the model to focus on the most important or prominent features within a feature map, such as edges or textures.
By selecting the maximum value within a pooling window, the neural n/w becomes less sensitive to small variations in the input image.





Max Pooling & Average Pooling Usecases

Max Pooling → it is often used because it tends to retain the most important features in the image.

Image.

Avg. Pooling :→ it is used in the cases where smoothness or overall representation is more important than retaining high-intensity features

© apc —

→ (Max) → (Max Performance delivered)

Topper in DL → Max → Top of the class

Avg → Smoothing out the performance.

Performances → Top Notch / good / bad / worse → averaged out.