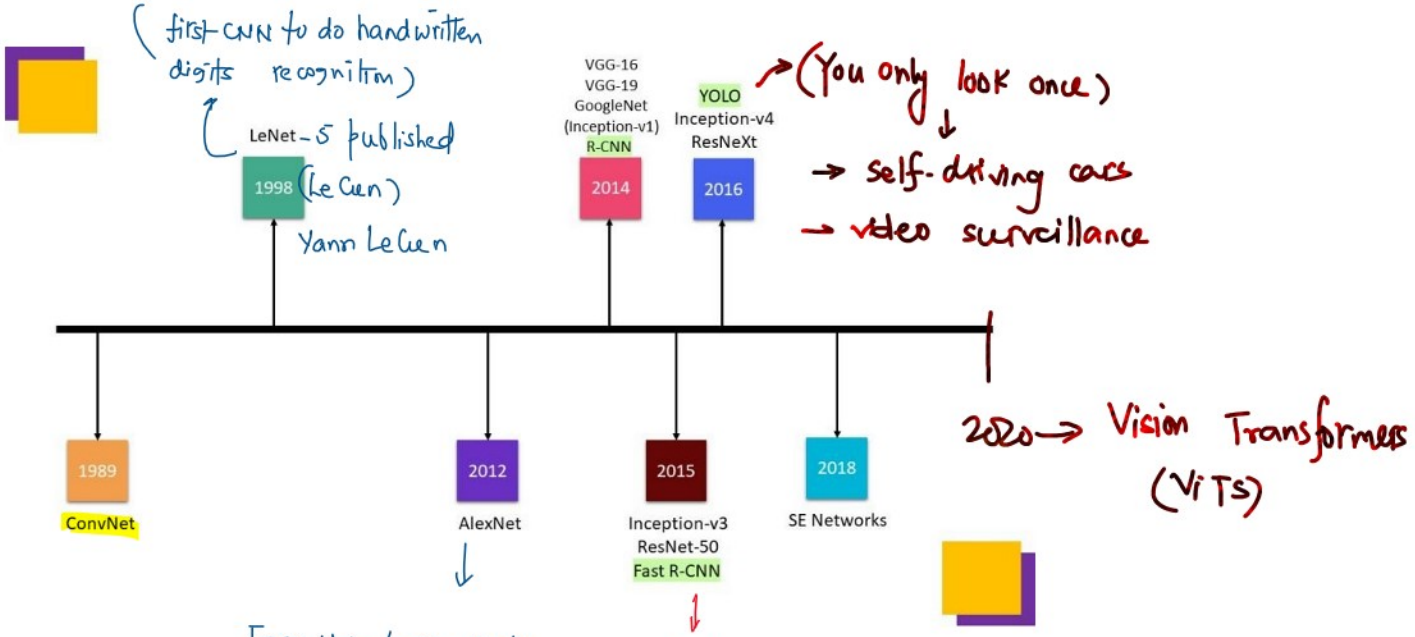
CNN Detailed Discussion

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)

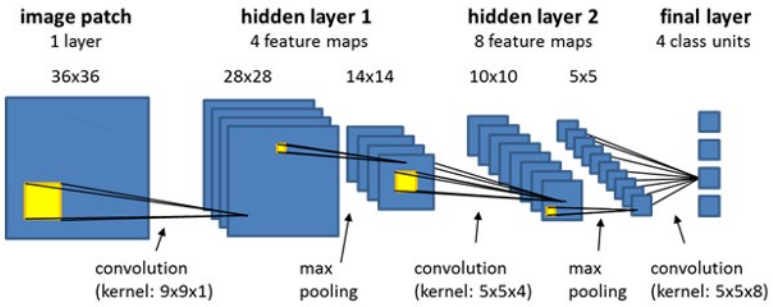
- ReLU activation function
- Dropout
- GPU acceleration

⇓

- [illegible] 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)

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

For color (RGB) images:

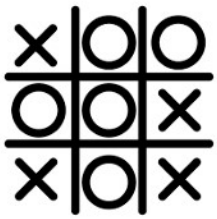
input tensor ⟶ (H x W x 3)

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

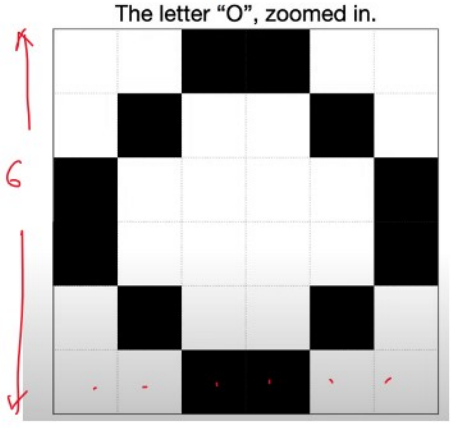
2. Convolutional Layer

- 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.

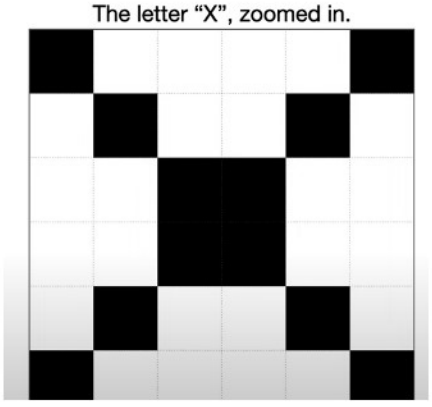
Tic- Tac- Toe

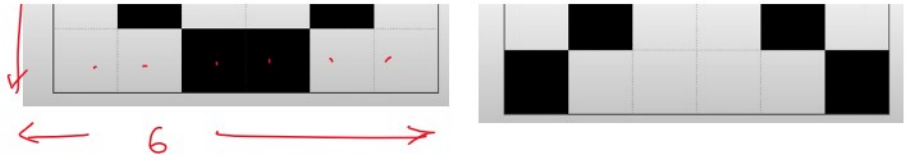


6x6 pixels

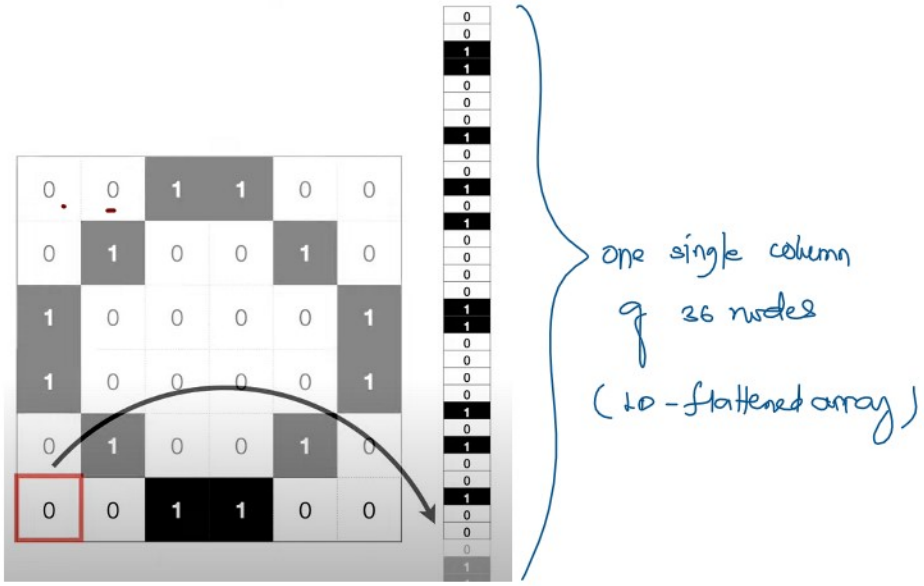


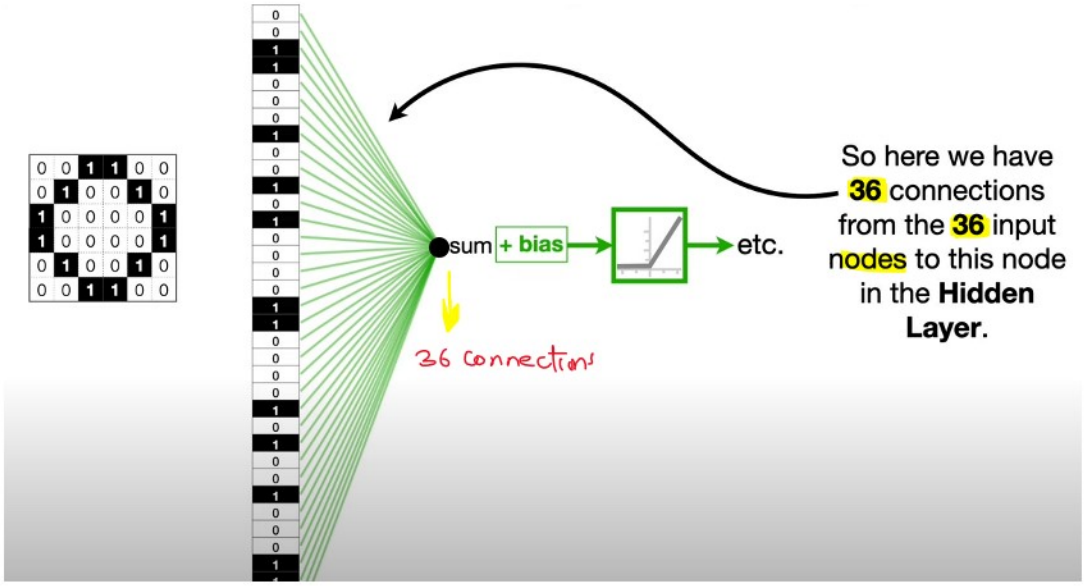
6x6 pixels

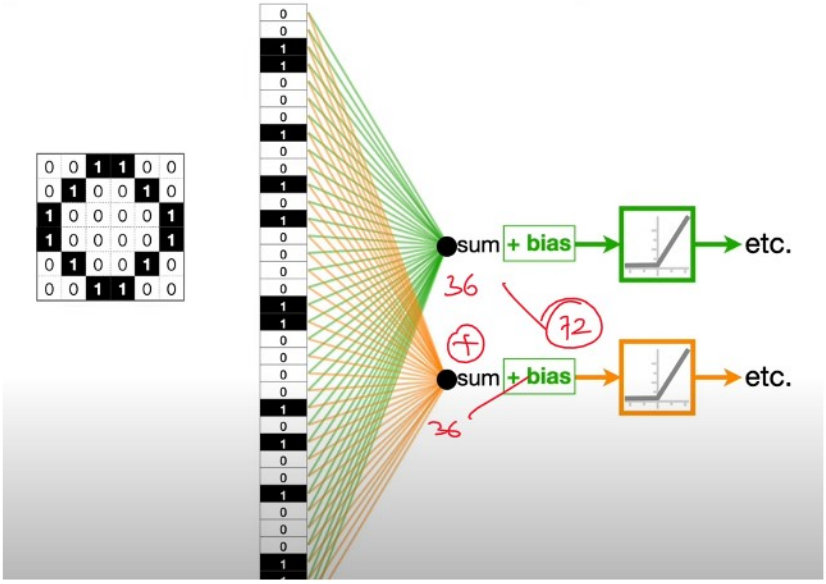




In artificial neural n/w,







1 Mega pixel → 1×10^6 pixels

$\sqrt{10^6} = 1000$ pixels → 1 mega pixel image.

1000 pixels

● → one neuron → 1 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.

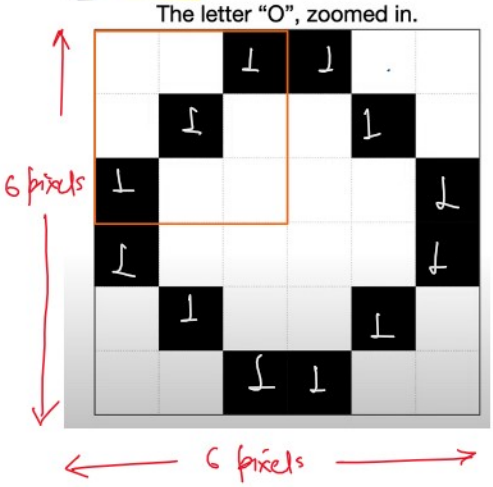
CNNs do three things to make image classification practical:

1) Reduce the no. of input nodes

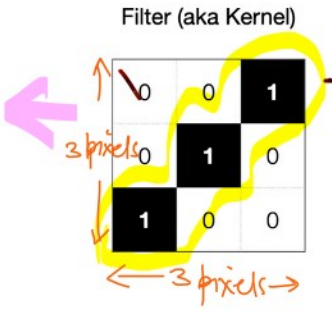
2) Tolerate / accommodate the marginal shifts in the pixels within the image.

3) Mind the correlations in the complex image.

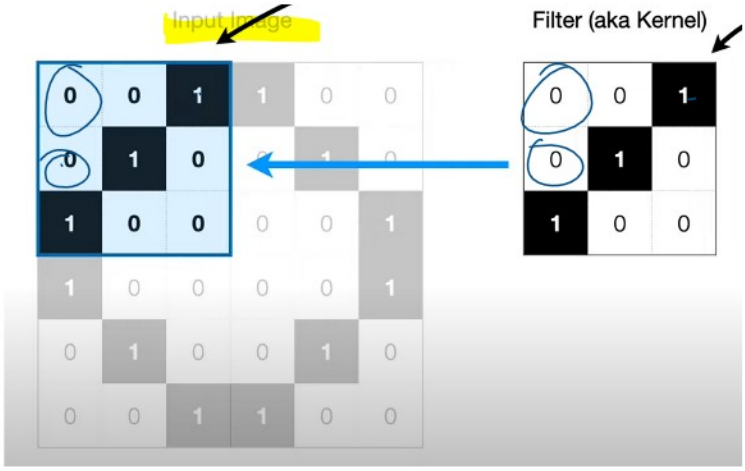
Step #①



CNN applies a filter to the input image.



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

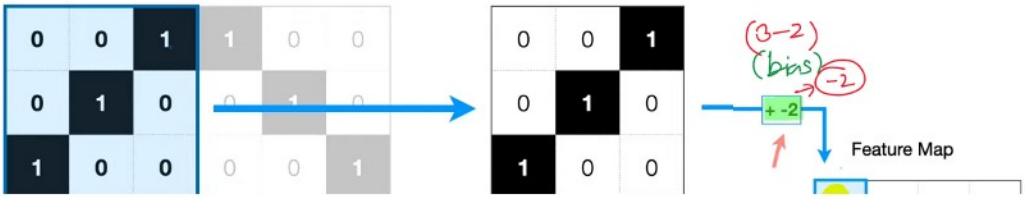


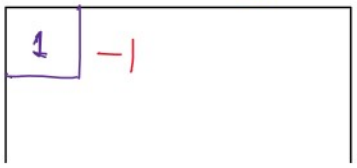
In convolution, dot product between the input and the filter in the overlapped area, and hence we can say that the filter is convolved with the input.

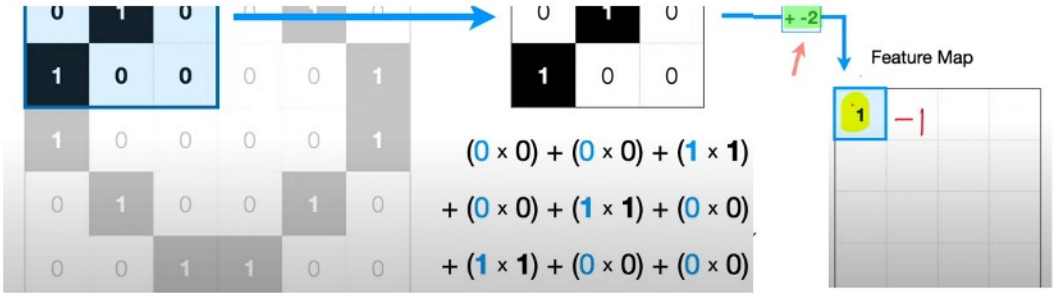
dot product → sum of products

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

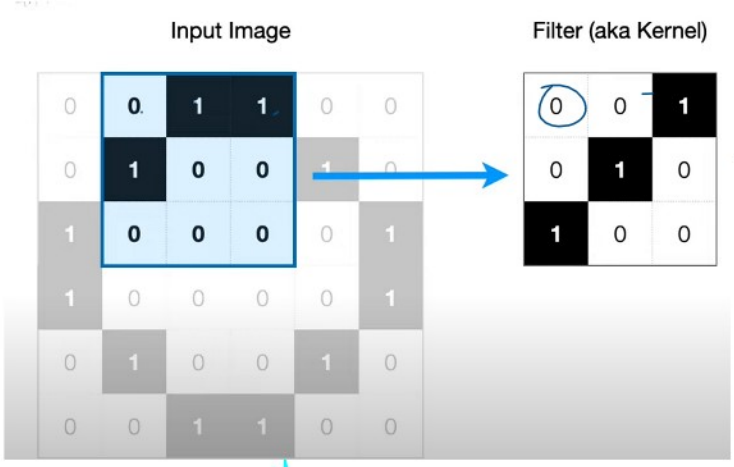
$= 3 \rightarrow$ bias $(-2) \rightarrow 1$







Next shift by a pixel:



(0x0) + (1x0) + (1x1)
+(1x0) + (0x1) + (0x0)
+(0x1) + (0x0) + (0x0) = 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	0	0	0
0	0	0	0
0	0	0	0
0	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.

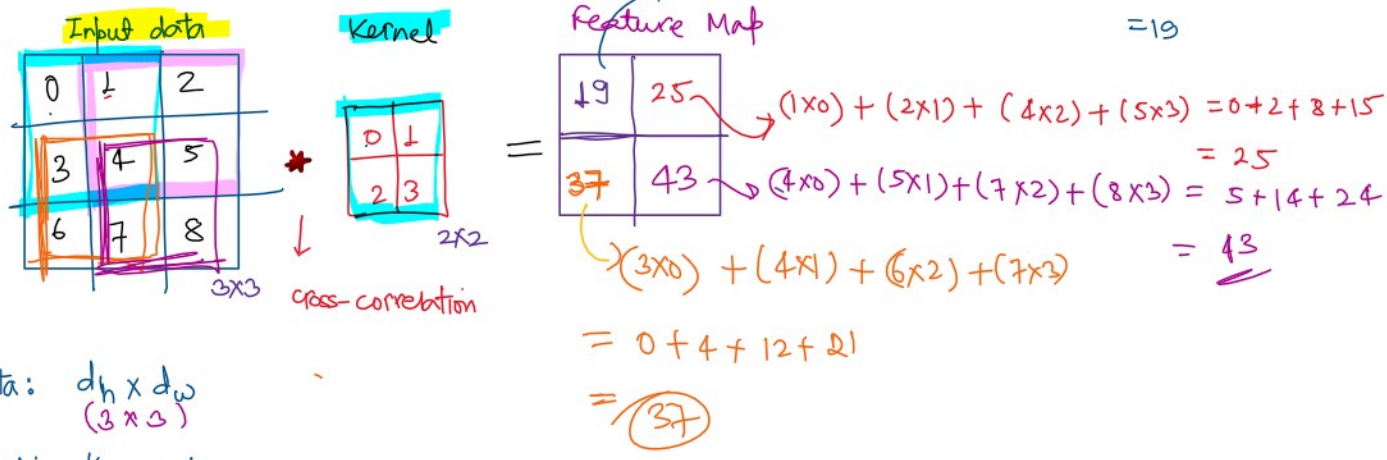
Key terms:

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: $d_h \times d_w$ (3 x 3)

kernel: $k_h \times k_w$ (2 x 2)

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.

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

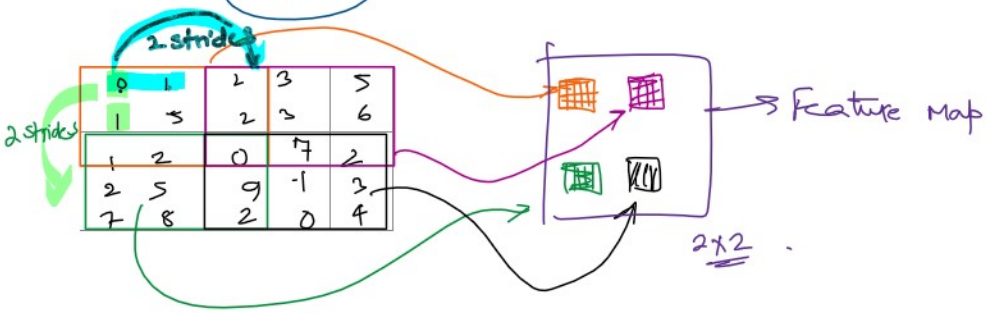
$= 3 \times 3$

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

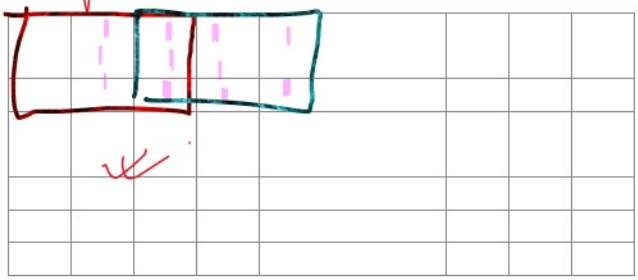
b) Prevents overfitting: With reduction in feature map size, it acts like a form of regularization

map size, it acts like a form of regularization as CNN model captures important insights from the the data.

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

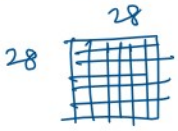
2. Faster Computation

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



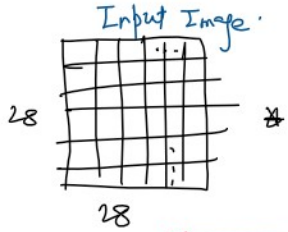
→ small size image

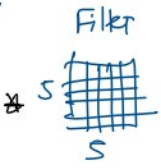
- Using stride 2 or more can cause excessive downsampling, leading to a significant reduction in spatial dimensions (size) of the feature map which in turn can result in the loss of important fine-grained information.

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 captured sufficient features.





$= ? \left(\frac{28-5}{2}+1\right) \times \left(\frac{28-5}{2}+1\right)$

$\left(\frac{23}{2}+1\right) \times \left(\frac{23}{2}+1\right)$

$11.5 \quad 11.5$

stride = 2

(12×12)

784 → 144

144/784=0.1837

excessive downsampling by 82%.

↓

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

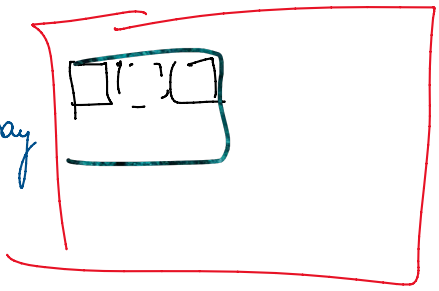
Best practices for large input sizes | images

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

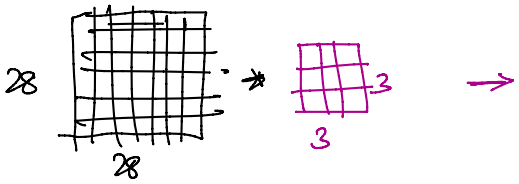
Smaller Input Sizes:



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. (3x3) 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



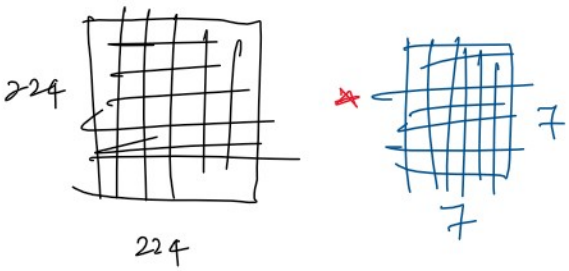
Larger Input Sizes:

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, 5x5 or 7x7) can be used in initial layers to capture more contextual information from the image and reduce the depth required for feature extraction.

In a 224x224 input image, a (7x7) 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 x 224) (7x7)

= 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 x 109

Practical Examples from CNN architectures

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)

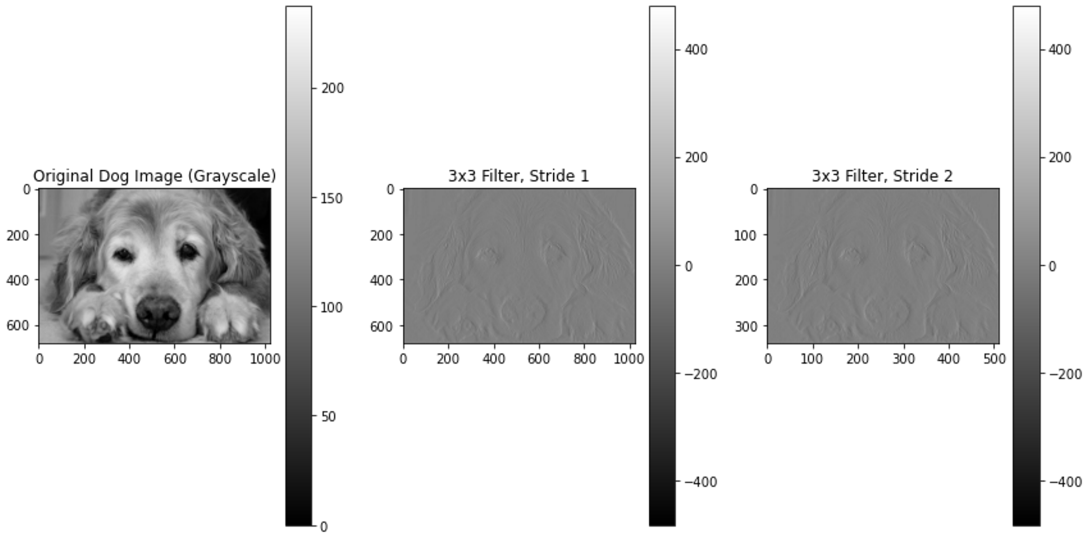
Kernel size: 3x3

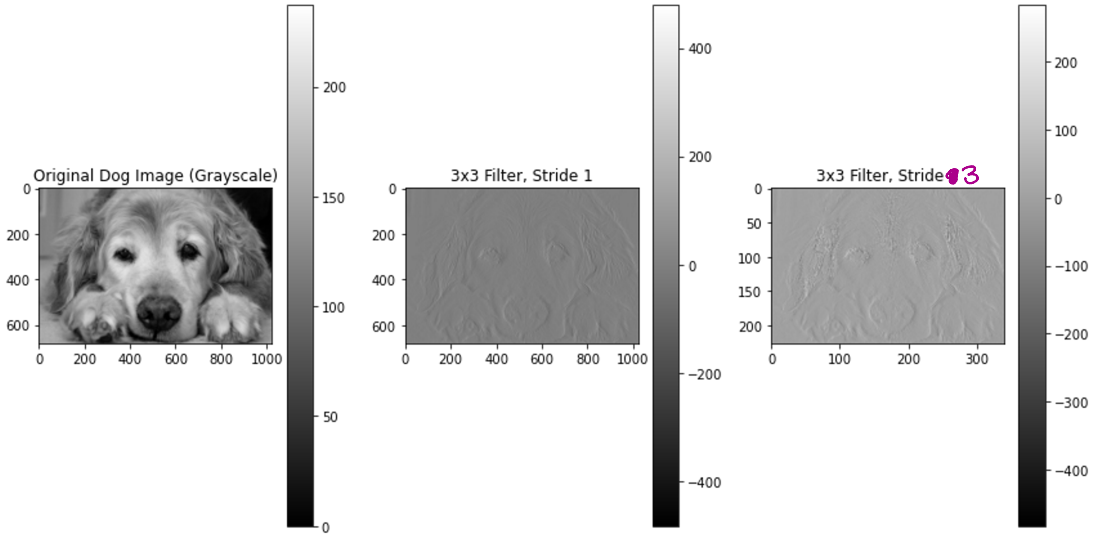
stride : (1) followed by (2)

4. ResNet

For large inputs:

Kernel size: 7x7	3x3
Stride: 2	1


Original Dog Image (Grayscale)
3x3 Filter, Stride 1
3x3 Filter, Stride 2


Original Dog Image (Grayscale)
3x3 Filter, Stride 1
3x3 Filter, Stride 3