

CNN

Convolutional Neural Network is the type of deep learning model that is especially powerful for working with images, videos etc.

💡 Why "Convolutional"?



Because it uses a mathematical operation called **convolution** to extract **features** (like edges, textures, shapes) from input data—especially images.

Input Image Patch:

```
[1, 2, 0]  
[4, 5, 6]  
[7, 8, 9]
```

And a 3x3 filter (also called kernel):

```
csharp  
Filter:      Kernel  
[1, 0, -1]  
[1, 0, -1]  
[1, 0, -1]
```

Purpose of Convolution =

Detect edges, corners, texture

💡 Convolution means:

Multiply each value from the image and the filter element-wise, then sum them up.

markdown

Result:

$$\begin{aligned} &= 1*1 + 2*0 + 0*(-1) \\ &+ 4*1 + 5*0 + 6*(-1) \\ &+ 7*1 + 8*0 + 9*(-1) \\ \\ &= 1 + 0 + 0 + 4 + 0 - 6 + 7 + 0 - 9 = -3 \end{aligned}$$

That -3 becomes a single pixel in the output feature map.

You move the filter over the whole image to generate more values.

🔍 Key Components of a CNN:

1. Input Layer – Raw image data (e.g., 224x224 pixels).
2. Convolutional Layer – Applies filters to detect patterns (like edges or corners). → (Kernel)
3. Activation Function – Usually ReLU, adds non-linearity.
4. Pooling Layer – Reduces dimensionality (e.g., max pooling).
5. Fully Connected Layer – Final layer(s) to make predictions. → Flattening
6. Output Layer – Gives final result (e.g., classification label like "cat" or "dog").

💡 What CNNs Are Used For:

- Image classification (e.g., recognizing objects in photos)
- Face recognition
- Self-driving cars (lane detection, obstacle recognition)

🧠 Simple Analogy:

Think of CNNs like a brain that first **detects edges**, then **shapes**, then **objects** — like how humans recognize patterns step by step.

Main concept

is to mimim how human brain analyses images

Cerebral cortex → Visual (layers) cortex

CNN TERMINOLOGY

Q: What is an image?

Image is a grid of pixel values.

→ For B & W image → each pixel = 0 - 255

0 = Black, 255 = White

Single channel (one value for each pixel)

e.g., 5×5 image = 5×5 vector

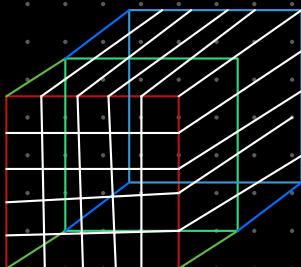
→ For a colored image (RGB)

Each pixel has 3 values (3 channels)

e.g., 5×5 image = $\underline{5 \times 5 \times 3}$ tensor

Imp.

R G B



$5 \times 5 \times 3 \rightarrow 3$ channel

◆ For Grayscale Images:

- Pixel value = 0 → black
- Pixel value = 255 → white
- Values in between represent shades of gray
 - 127 ≈ medium gray

◆ For RGB Images:

Each pixel has three channels:

- Red (R), Green (G), Blue (B)
- Each ranges from 0 (no intensity) to 255 (full intensity)

Example:

- (0, 0, 0) → Black (no light at all)
- (255, 255, 255) → White (full intensity of all colors)
- (255, 0, 0) → Red
- (0, 255, 0) → Green
- (0, 0, 255) → Blue

Q: What is Tensor?

A tensor is a container that can hold data in dimensions (any dimensions)

12
34

Examples:

- 0D Tensor → Scalar → 7
- 1D Tensor → Vector → [3, 5, 7]
- 2D Tensor → Matrix → [[1, 2], [3, 4]]
- 3D Tensor → Stack of matrices → Like a color image [Height x Width x Channels]

STEPS

→ Step I ; Input image pixel info

Step II ; Normalise → divide by 255

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

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	1	1	1

Step III ;

Convolution Operation in CNN to detect edges, corners and texture.

stride = 1 means
Right by 1 skip

I/P

$$\begin{array}{c}
 \text{stride=1} \\
 \begin{array}{|c|c|c|c|c|c|c|} \hline
 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 \end{array} \\
 6 \times 6 \times 1
 \end{array}
 \quad
 \begin{array}{c}
 n=6 \quad f=3 \\
 * \quad \begin{array}{|c|c|c|} \hline
 +1 & 0 & -1 \\ \hline
 +2 & 0 & -2 \\ \hline
 +1 & 0 & -1 \\ \hline
 \end{array} \\
 \text{filter} \quad 3 \times 3 \quad \text{filters / Kernel} \\
 \text{vertical edge filters}
 \end{array}
 \quad
 \begin{array}{c}
 0/p=4 \\
 \begin{array}{|c|c|c|c|} \hline
 0 & -4 & -4 & 0 \\ \hline
 0 & -4 & -4 & 0 \\ \hline
 0 & -4 & -4 & 0 \\ \hline
 0 & -4 & -4 & 0 \\ \hline
 \end{array} \\
 4 \times 4 \quad O/P
 \end{array}$$

Imp

Q, how come we got O/p as 4x4?

so we have formula ;

$$\hookrightarrow n - f + 1 \quad \begin{matrix} n \\ \downarrow \\ \text{Input} \end{matrix} \quad \begin{matrix} f \\ \downarrow \\ \text{filter} \end{matrix} \Rightarrow 6 - 3 + 1 = 4$$

Filters = are used to get info out of images

Q, what is padding ; → As seen above $6 \times 6 \rightarrow 4 \times 4$

padding is used to prevent the loss of image information (pixels) during convolution operations.

✓ In short: Padding adds extra pixels (usually zeros) around the image to preserve size and information during convolution.

$$\begin{array}{c}
 \begin{array}{|c|c|c|c|c|c|c|} \hline
 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline
 0 & & & & & & & \\ \hline
 \end{array} \\
 n=6 \quad p=1
 \end{array}
 \quad
 \begin{array}{c}
 f=3 \\
 * \quad \begin{array}{|c|c|c|} \hline
 +1 & 0 & -1 \\ \hline
 +2 & 0 & -2 \\ \hline
 +1 & 0 & -1 \\ \hline
 \end{array} \\
 \boxed{n-f+2p+1} \\
 6-3+2(1)+1=6
 \end{array}
 \quad
 \begin{array}{c}
 6 \times 6 \\
 = \\
 \begin{array}{|c|c|c|c|c|c|} \hline
 0 & -4 & -4 & 0 \\ \hline
 0 & -4 & -4 & 0 \\ \hline
 0 & -4 & -4 & 0 \\ \hline
 0 & -4 & -4 & 0 \\ \hline
 \end{array}
 \end{array}$$

New formula

$$\begin{aligned}
 &= n + 2p - f + 1 \\
 &= 6 + 2(1) - 3 + 1 \\
 &= 8 - 3 + 1 = 6
 \end{aligned}$$

Zero Padding

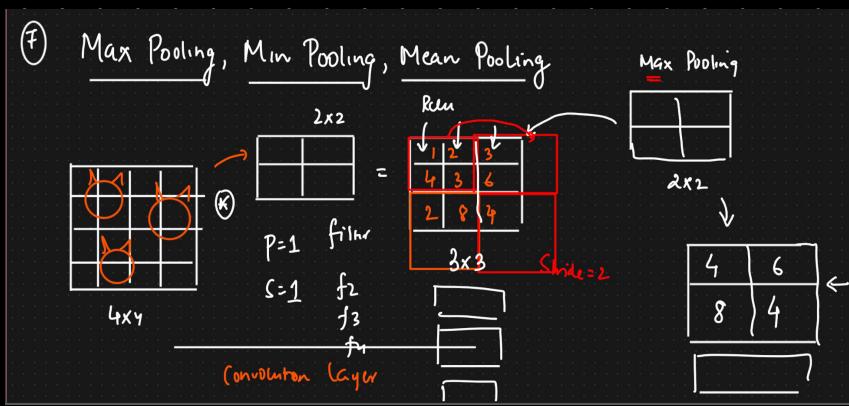
Pads with zeros around the image. Most common.

Reflect Padding

Pads with reflected values of the border.

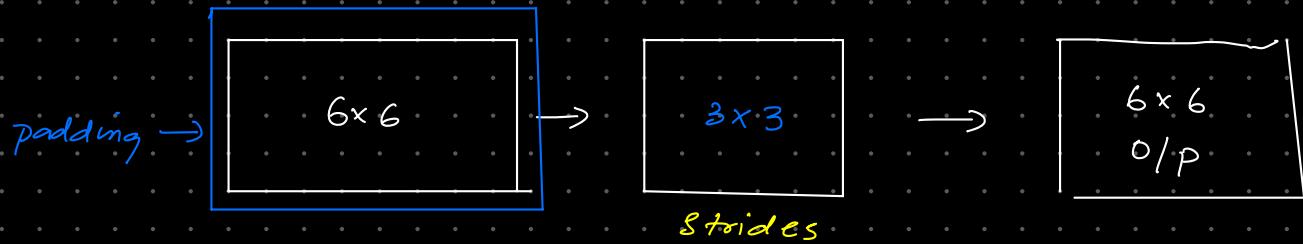
POOLING = is a technique in CNN to reduce dimensionality

Types =



Conclusion

1. Input Image
2. Normalization $(0-1)$ where $1 = 255$
3. Convolution method (apply filter (kernel))



4. Activation function (ReLU)
 5. Pooling \rightarrow to reduce dimensionality
- Max, Min or Mean pooling

6. Flattening \rightarrow elongating pooling output

$$\begin{array}{|c|c|} \hline 2 & 5 \\ \hline 1 & 6 \\ \hline \end{array} = \begin{array}{c} 2 \\ 5 \\ 1 \\ 6 \end{array}$$

7. Finally \rightarrow ANN classification (Dense layers).

CONCLUSION

✓ Step-by-Step: Your CNN Understanding

✓ 1. Input Image

Let's say a grayscale image: 6×6 matrix of pixel values (0–255)

✓ Correct.

✓ 2. Normalization

Normalize pixel values: $255 \rightarrow 1, 0 \rightarrow 0$

✓ Correct. Sometimes values are scaled between $0\text{--}1$ or $-1\text{--}1$.

✓ 3. Apply Convolution (Filter/Kernel 3x3)

To detect edges, textures, corners, etc.

✓ Correct.

Input Image (6×6) → Normalize →

Convolution (3x3 filter) →

ReLU →

Pooling (e.g., Max Pooling) →

Flatten →

Fully Connected Layers (ANN) →

Output with Activation (sigmoid/softmax)

✗ 4. You said: Output will be 6×6

Not always — usually the output size shrinks (unless you use padding).

Formula (without padding):

arduino

Copy Edit

$$\begin{aligned} \text{Output size} &= (\text{Input size} - \text{Kernel size}) + 1 \\ &= (6 - 3) + 1 = 4 \\ \rightarrow \text{So, output} &= 4 \times 4 \end{aligned}$$

✓ But if padding = same, then output stays 6×6 .

V.v. imp

? CONFUSION: Why ReLU after Convolution?

Let's clarify:

✓ After each convolution, we apply ReLU (or another activation function).

🔍 Why?

Because convolution may produce negative values, and ReLU keeps only the positive ones:

lua

Copy Edit

$$\text{ReLU}(x) = \max(0, x)$$

This introduces non-linearity into the model. Without this, your entire CNN would just be one big linear operation, and it wouldn't be able to learn complex patterns.

So yes — we apply ReLU right after convolution, before pooling.

✓ 5. Pooling Layer (e.g., Max Pooling)

To reduce dimensions, keep important features

✓ Correct. Typically reduces size (e.g., $4 \times 4 \rightarrow 2 \times 2$)

✓ 6. Flatten the Output

Convert the 2D/3D tensor to 1D vector for input into fully connected (ANN) layer

✓ Correct.

✓ 7. Apply Fully Connected (Dense) Layers

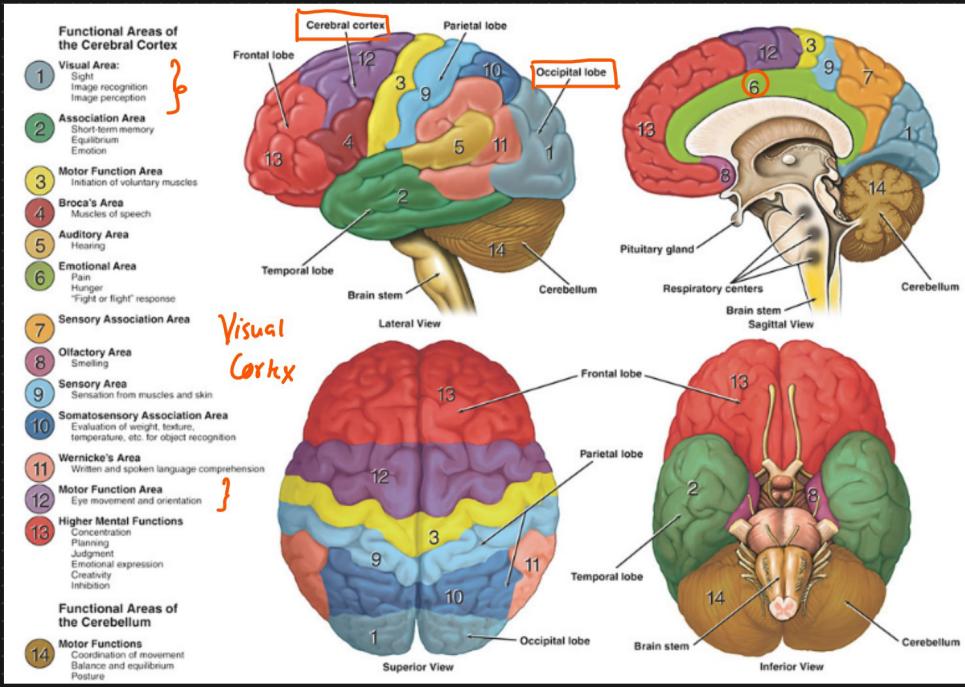
Feed into ANN, apply activation functions (like ReLU, softmax, etc.)

✓ Correct.

✓ Final Layer Activation:

- For binary classification: sigmoid
- For multi-class classification: softmax

Convolutional Neural N/w



<https://www.dana.org>

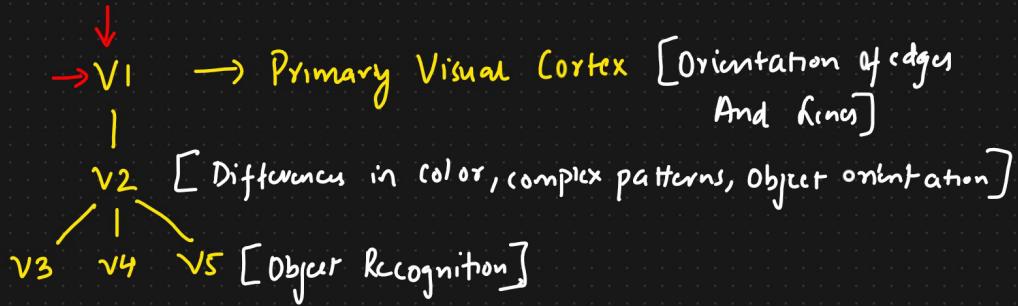


Dataset : I/p features O/p

② CNN : I/p ⇒ Images Eg: Image classification,
 Object Detection, Segmentation

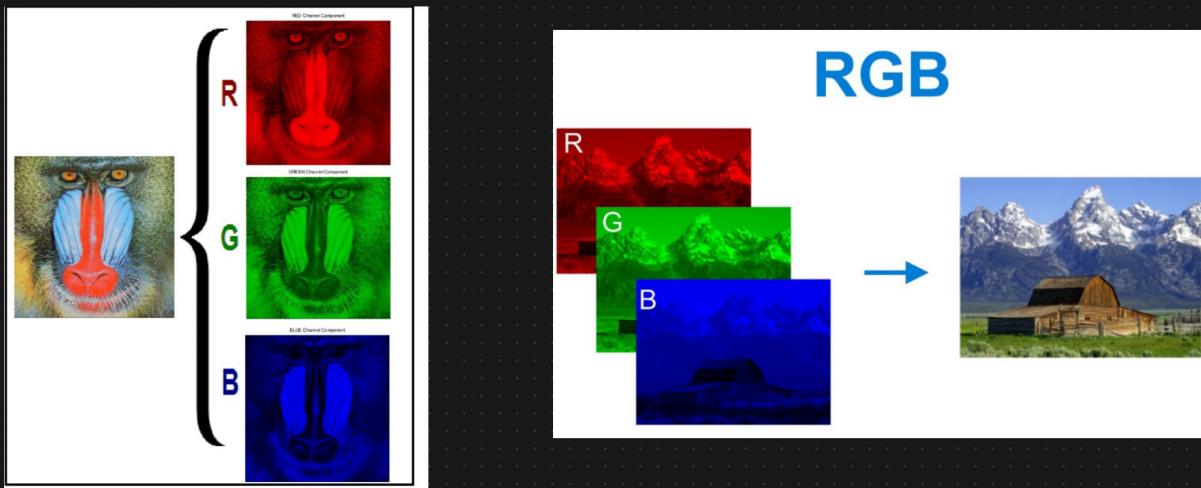
② Cerebral Cortex And Visual Cortex

Visual Cortex (V1-V5) [Region of the brain that receives, integrates and processes visual information relayed from the retinas].

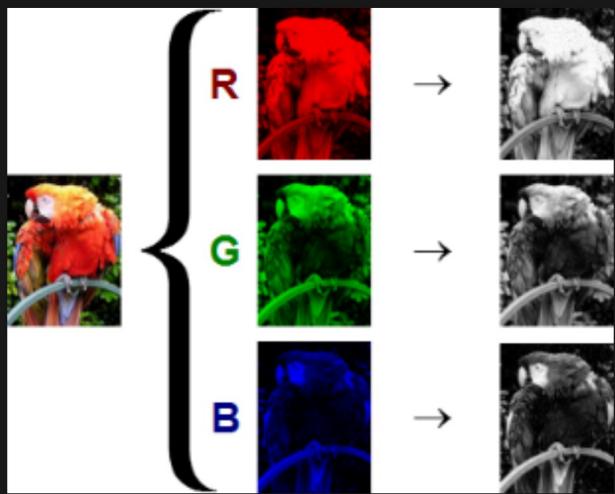


Visualize the Image

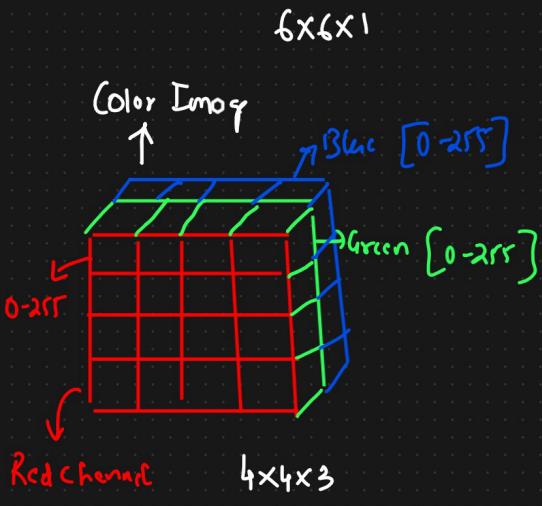
③ RGB Images And Gray Scale Images



<https://www.researchgate.net/>



$0-255 \rightarrow$ Gray Scale Image.



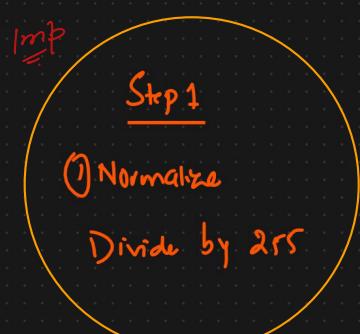
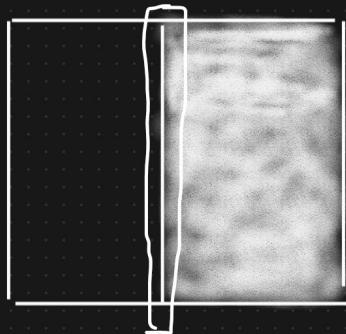
<https://commons.wikimedia.org/>

④ Convolution Operation In CNN

$\rightarrow (0,1)$

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

\Rightarrow



+1	+2	-1
0	0	0
-1	-2	-1

$6 \times 6 \times 1$

Convolution operation

$s=1$

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	1	1	1

$n=6$

$f=3$

$o/p = 4$

+1	0	-1
+2	0	-2
+1	0	-1

0	-4	-4	0
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0

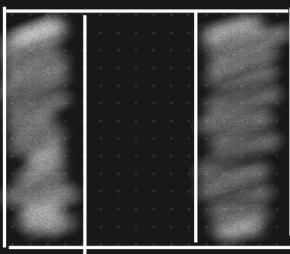
3×3

filters

Vertical edge filters

4×4

$$\begin{aligned} h - f + 1 &= \\ &= 6 - 3 + 1 = 4 \end{aligned}$$



arr	0	0	2rr
2rr	0	0	2rr
2rr	0	0	2rr
2rr	0	0	2rr

⑤ Padding In CNN

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

6×6

$n=6$

8×8

$f=3$

+1	0	-1
+2	0	-2
+1	0	-1

0	-4	-4	0	
0	-4	-4	0	
0	-4	-4	0	
0	-4	-4	0	

6×6

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

$$3 + 2p + 1 = 6$$

$$2p = 6 - 4$$

$$p = \frac{2}{2} = 1$$

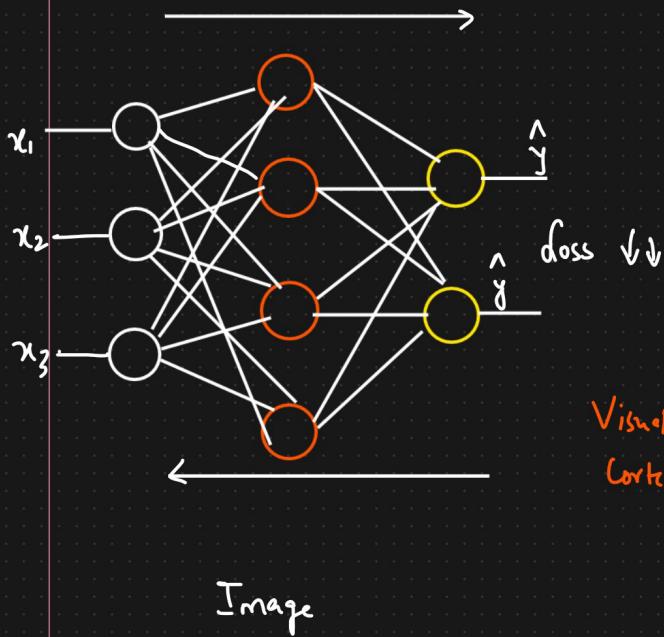
7×7

3×3

7×7

How much padding you need to apply?

⑥ Operation of CNN Vs ANN



$$z = w^T x_i + b$$

$$\text{ReLU}(z)$$

Vision
Cortex



→ ReLU operation $\max(0, x)$

$$\begin{array}{c} \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\ \begin{array}{|c|c|c|c|} \hline 0 & -4 & -4 & 0 \\ \hline 0 & -4 & -4 & 0 \\ \hline 0 & -4 & -4 & 0 \\ \hline 0 & -4 & -4 & 0 \\ \hline \end{array} \end{array}$$

$$\begin{array}{c} \begin{array}{|c|c|c|c|} \hline - & - & - & - \\ \hline - & - & - & - \\ \hline - & - & - & - \\ \hline - & - & - & - \\ \hline \end{array} \end{array}$$

*

$$\begin{array}{|c|c|c|} \hline +1 & 0 & -1 \\ \hline +2 & 0 & -2 \\ \hline +1 & 0 & -1 \\ \hline \end{array}$$

f_1

$$\begin{array}{|c|c|} \hline & \\ \hline & \\ \hline \end{array}$$

f_2

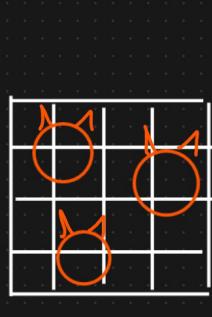
f_3

\vdots

f_n

Convolution Layer

⑦ Max Pooling, Min Pooling, Mean Pooling



2×2

$P=1$ filter
 $S=1$

Convolution Layer

$$\begin{array}{c} \text{ReLU} \\ \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 3 & 6 \\ \hline 2 & 8 & 4 \\ \hline \end{array} \end{array}$$

3×3 $S_{\text{stride}}=2$

Max Pooling

$$\begin{array}{|c|c|} \hline & \\ \hline & \\ \hline \end{array}$$

2×2

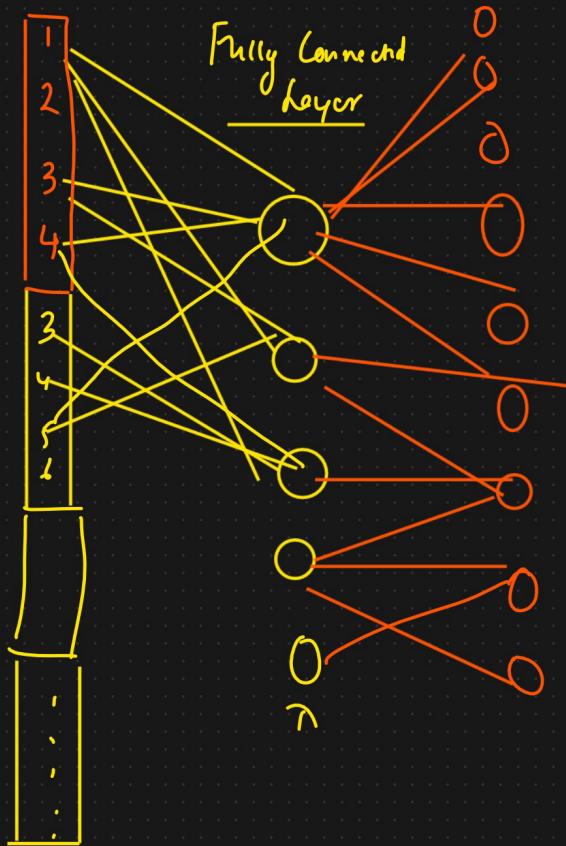
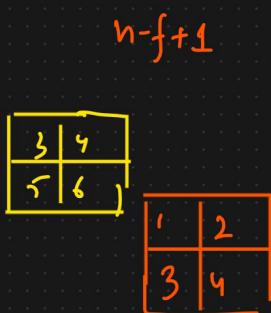
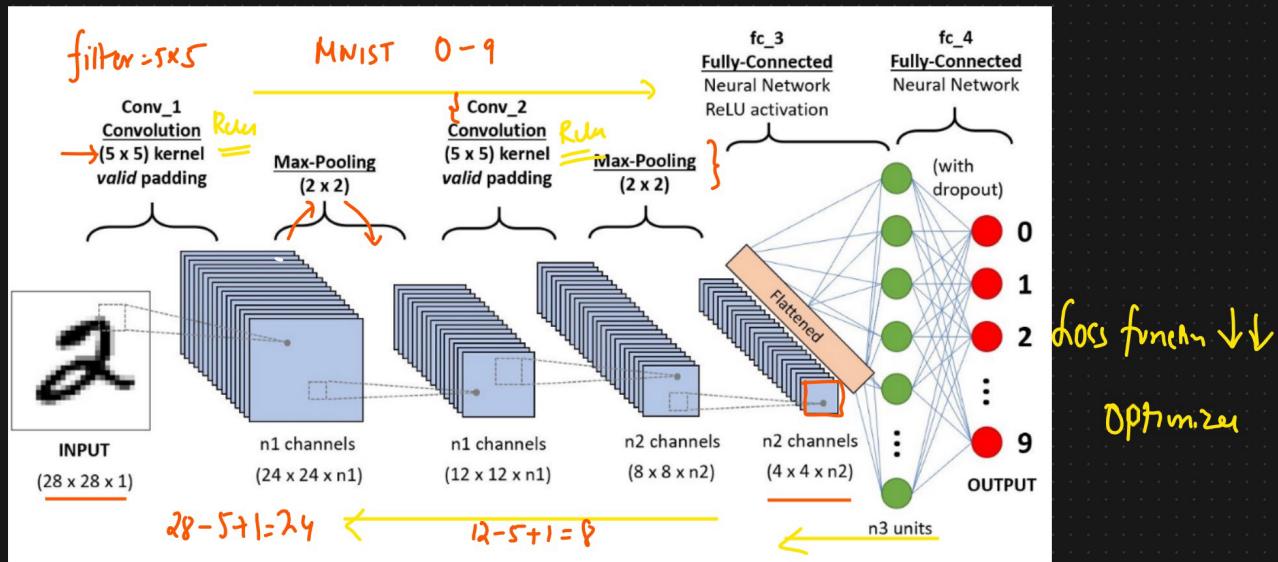
\downarrow

$$\begin{array}{c} \begin{array}{|c|c|} \hline 4 & 6 \\ \hline 8 & 4 \\ \hline \end{array} \end{array}$$

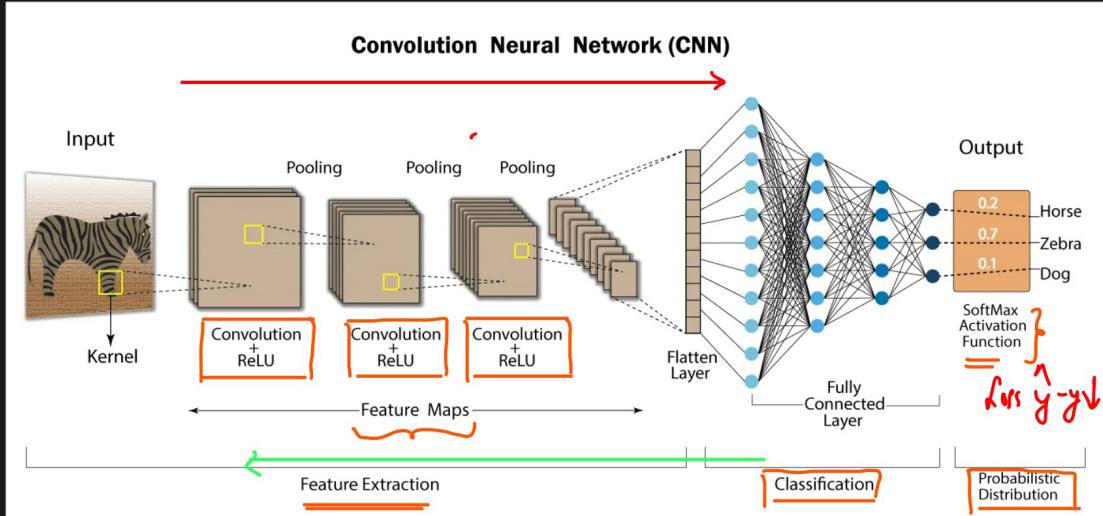
$\boxed{\quad}$

⑦ Location Invariant

⑧ Fully Connected Layer In CNN [Flattened Layer]



④ CNN Complete Example



<https://developersbreach.com/convolution-neural-network-deep-learning/>

0	0	0		
0	0	0		
0	0	0		
0	0	0		
0	0	0		

*

+1	0	-1
+2	0	-2
+1	0	-1

0	-4	-4	0
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0