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):

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

kernel

➡️ Convolution means:
Multiply each value from the image and the filter **element-wise**, then **sum them up.**

```
Result:
= 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
```

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

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

Purpose of Convolution:
Detect edges, corners, texture

🔍 Key Components of a CNN:

1. **Input Layer** – Raw image data (e.g., 224×224 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 mimic how human brain analyzes images

Cerebral cortex → visual cortex (Layers).

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

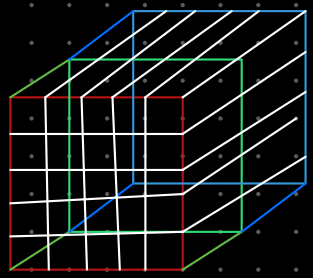
→ For a colored image (RGB)

Each pixel has 3 values (3 channel)

R G B

e.g., 5×5 image = 5×5×3 tensor

Imp.



5×5×3 → 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 n Dimensions (any Dimensions)

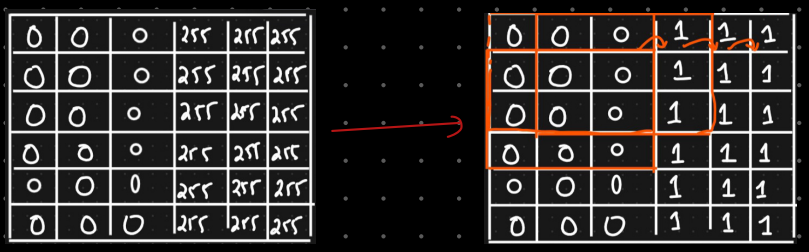
🔢 Examples:

- **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

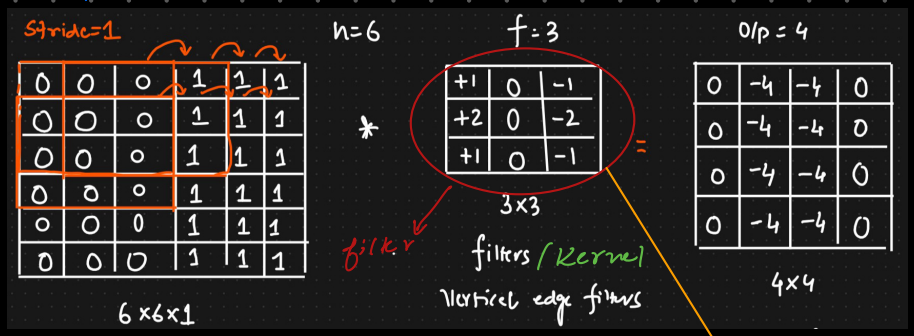
→ **Step I**; Input image pixel info

Step II; Normalize ⟶ Divide by 255



Step III; Convolution operation in CNN; to detect edges, corners and texture.

Stride = 1 means Right by 1 step.



This one is Vertical edge filter, we can similarly many other filter for texture corner etc.

Imp

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

So we have formula ;

↳ $n - f + 1$ (n: input, f: filter)

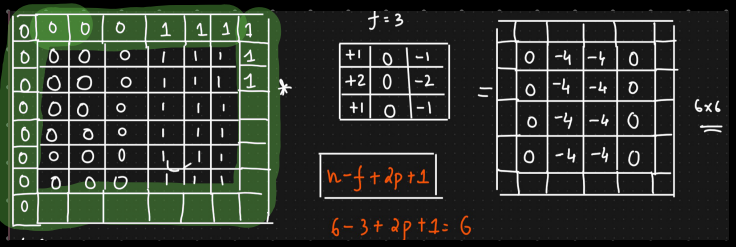
$\Rightarrow 6 - 3 + 1 = 4$

Filters : are used to get info out of images

Q// What is padding ; → As seen above 6x6 → 4x4

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.



New formula

$= n + 2p - f + 1$

$= 6 + 2(1) - 3 + 1$

$= 8 - 3 + 1 = 6$

$n = 6$

$p = 1$

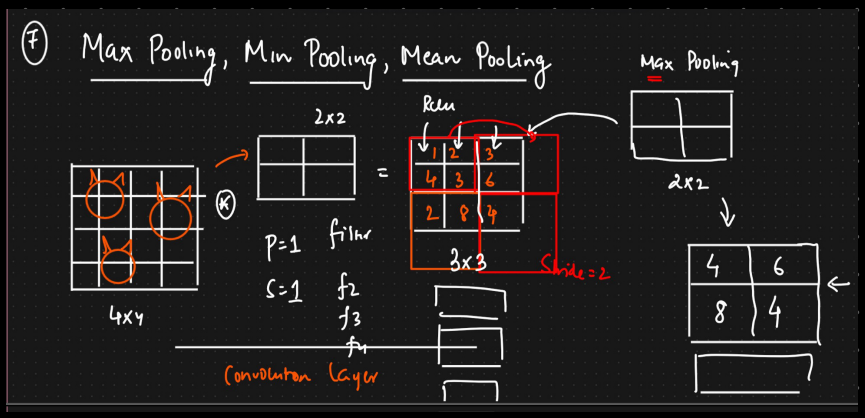
$f = 3$

Types

Zero Padding	Pads with zeros around the image. Most common.
Reflect Padding	Pads with reflected values of the border.

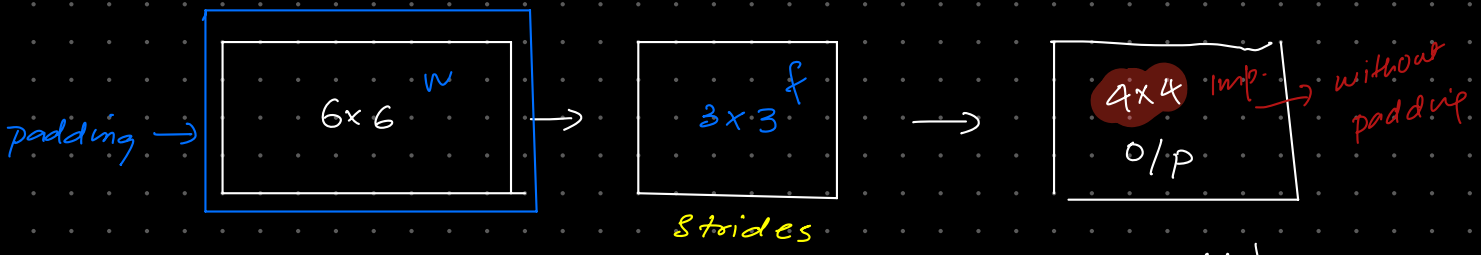
POOLING = is a technique in CNN to reduce dimensionality

Types = ① Max Pooling, Min Pooling, Mean Pooling



Conclusion

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



4. Activation function (Relu)
5. Pooling → to reduce dimensionality
   Max, Min or Mean Pooling
6. Flattening → elongating pooling output

2	5
1	6

= 2, 5, 1, 6

7. Finally → ANN classification (Dense Layers).

Conclusion

✅ Step-by-Step: Your CNN Understanding

✅ 1. Input Image

> Let's say a grayscale image: `6x6` matrix of pixel values (0–255)

✓ Correct.

✅ 2. Normalization

> Normalize pixel values: `255 → 1`, `0 → 0`

✓ Correct. Sometimes values are scaled between `0–1` or `-1 to 1`.

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

> To detect edges, textures, corners, etc.

✓ Correct.

```
Input Image (6x6) → 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 6x6

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

Formula (without padding):

arduino — Copy — Edit

```
Output size = (Input size − Kernel size) + 1
            = (6 − 3) + 1 = 4
→ So, output = 4x4
```

✅ But if **padding = same**, then output stays 6x6.

Wo padding
8 x 8
(2x2 padd)
= 8 − 3 + 1
= 6 (same).

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

```
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., 4x4 → 2x2)

✅ 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

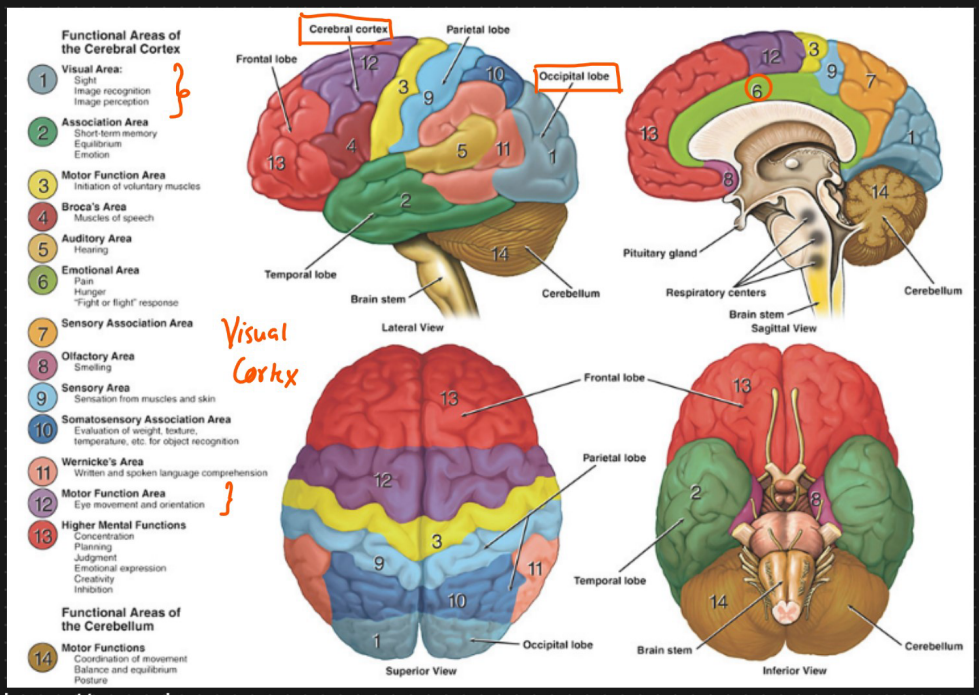
> Feed into ANN, apply activation functions (like **ReLU**, **softmax**, etc.)

✓ Correct.

✅ Final Layer Activation:

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

Convolutional Neural N/w



https://www.dana.org

① ANN → Supervised Learning → [Classification, Regression]

Dataset : I/p features O/p

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

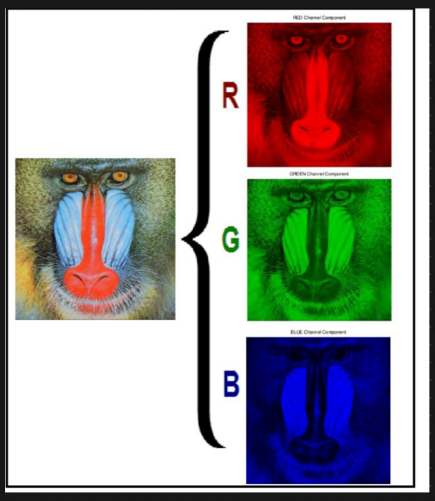
② Cerebral Cortex And Visual Cortex

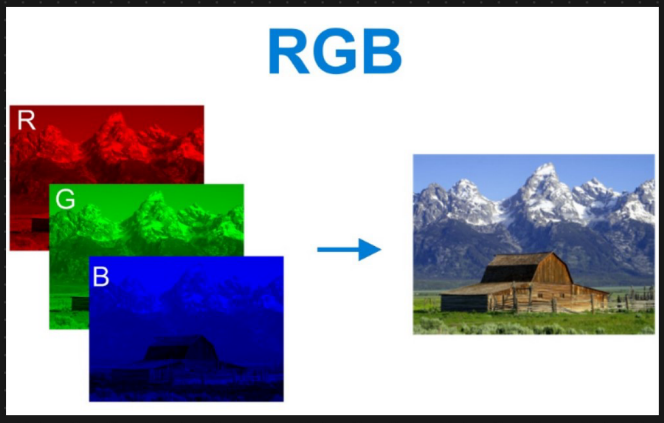
Visual Cortex (V1-V5) [Region of the brain that receives, integrates and processes visual information relayed from the retinas].

- V1 → Primary Visual Cortex [Orientation of edges And Lines]
- V2 [Differences in color, complex patterns, object orientation]
- V3, V4, V5 [Object Recognition]

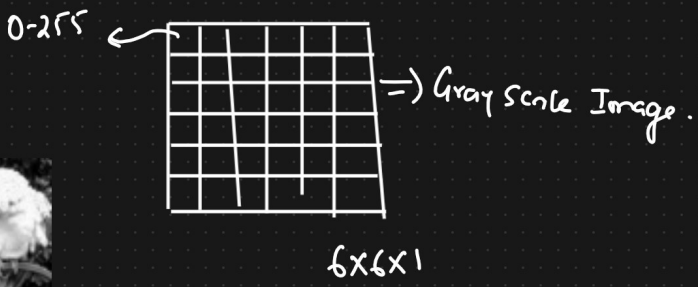
Visualize the Image

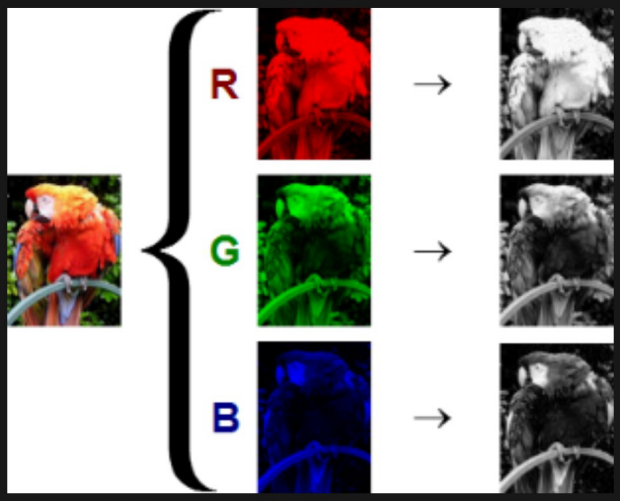
③ RGB Images And Gray Scale Images



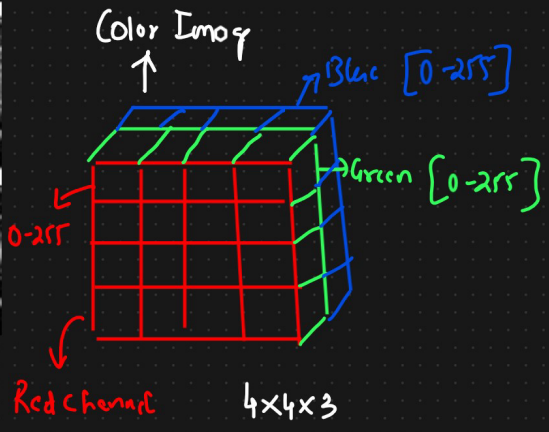


https://www.researchgate.net/





https://commons.wikimedia.org/

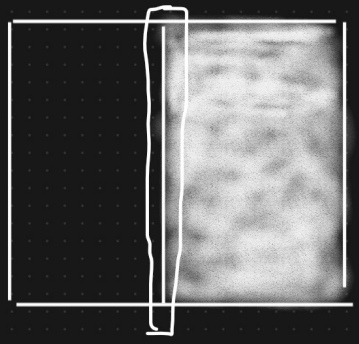


④ Convolution Operation In CNN

→ (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

6X6X1



Imp

Step 1

(1) Normalize

Divide by 255

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

Imp Convolution operation

Stride=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

6 x 6 x 1

n = 6

*

f = 3

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

3 x 3

filters

Vertical edge filters

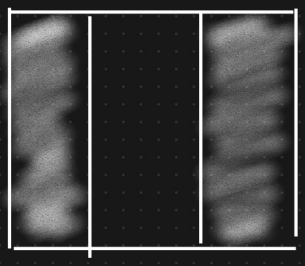
=

o/p = 4

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

4 x 4

$n - f + 1 = 6 - 3 + 1 = 4$



⇐

2rr	0	0	2rr
2rr	0	0	2rr
2rr	0	0	2rr
2rr	0	0	2rr

⇓

⑤ Padding In CNN

① Zero padding

② Neighbour padding

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

*

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 x 6

6x6 n=6 8x8

$n - f + 2p + 1$

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

$3 + 2p + 1 = 6$

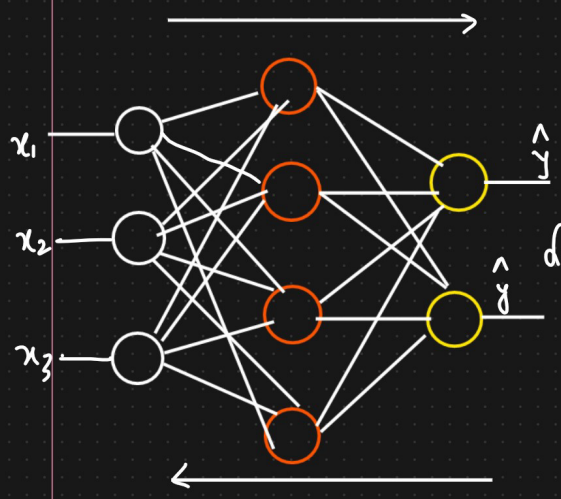
$2p = 6 - 4$

$p = 2/2 = 1$

7x7 3x3 7x7

How much padding you need to apply?

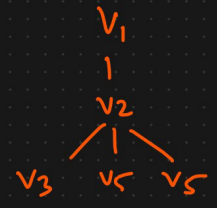
⑥ Operation of CNN Vs ANN



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

$$relu(z)$$

Visual Cortex



Image

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

$*$

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

f_1

f_2

f_3

f_n

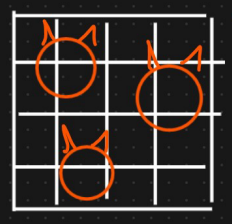
$=$

→ Relu operation $max(0, x)$

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

Convolution Layer

⑦ Max Pooling, Min Pooling, Mean Pooling



4x4

$\circledast$

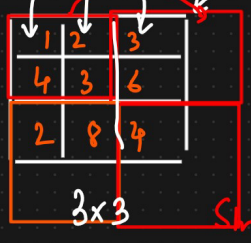
2x2 filter

P=1

S=1

$=$

Relu



3x3

Stride = 2

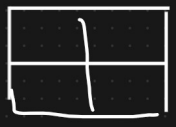
f_2

f_3

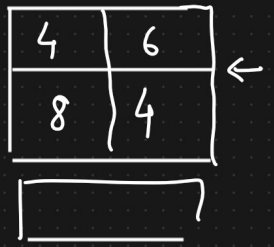
f_n

Convolution Layer

Max Pooling

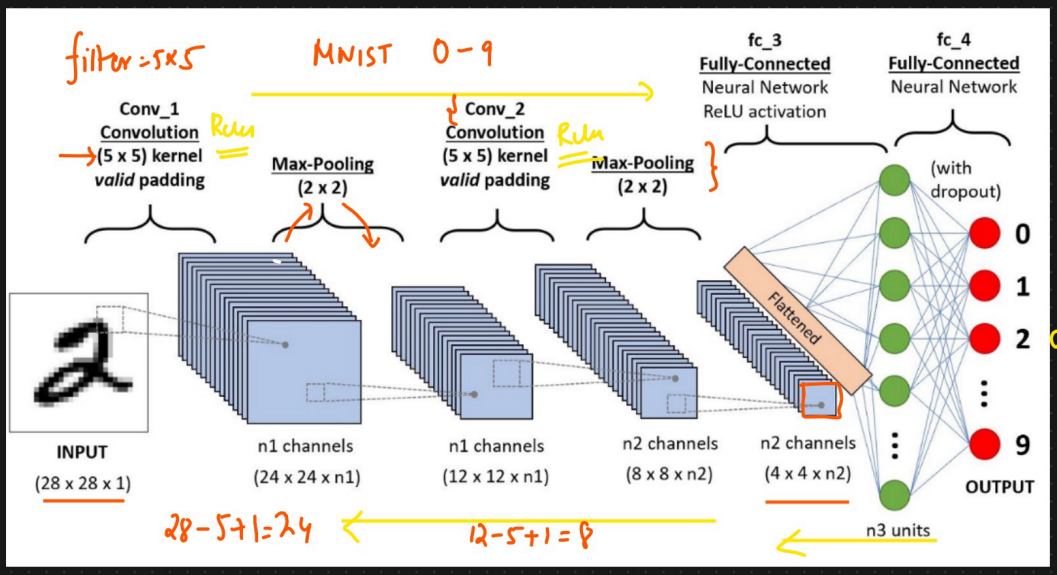


2x2



⑦ Location Invariant

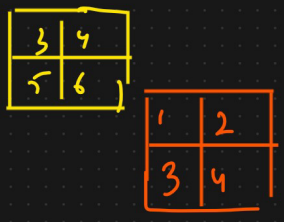
⑧ Fully Connected Layer In CNN [Flattened Layer]

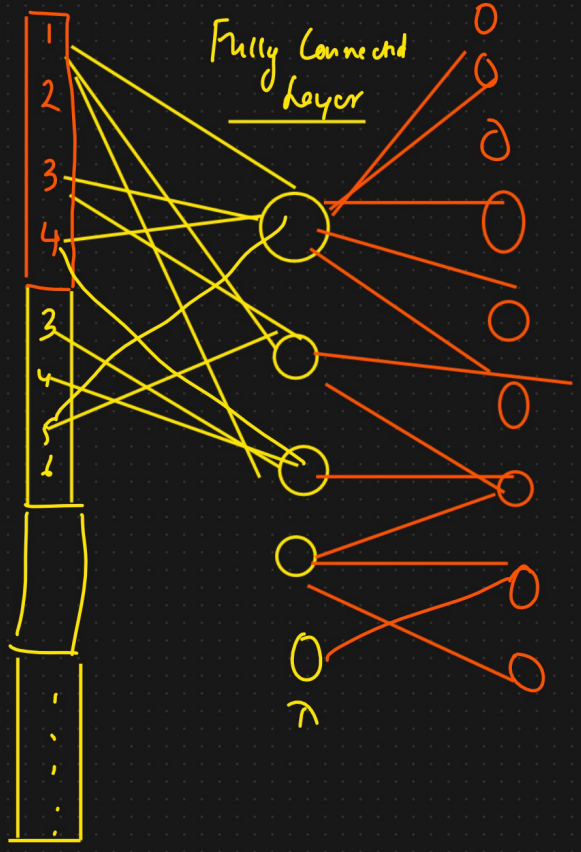


loss function ↓↓

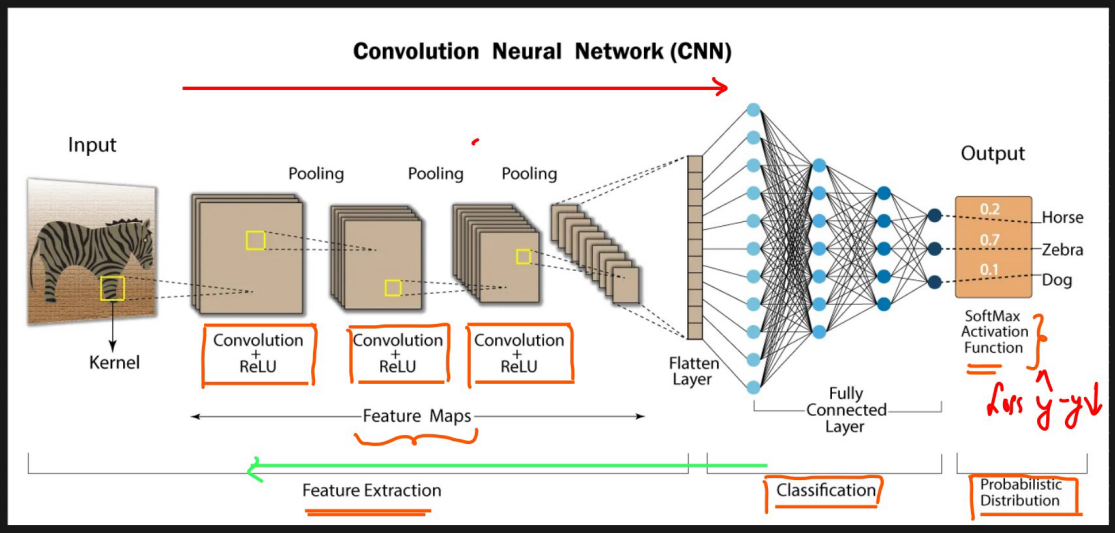
Optimizer

n-f+1





⑨ CNN Complete Example



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