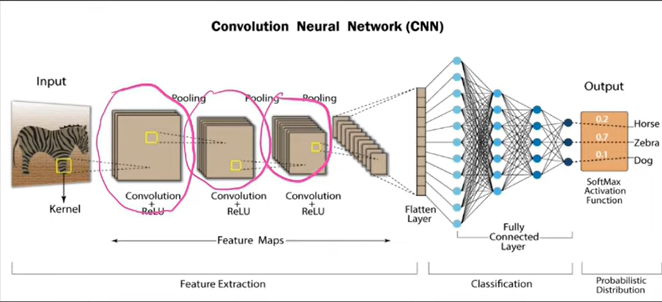
**CNN**

A Convolutional Neural Network (CNN) is a type of deep learning neural network mainly used for image data. It is used to process data in grid-like structure. It learns patterns such as:

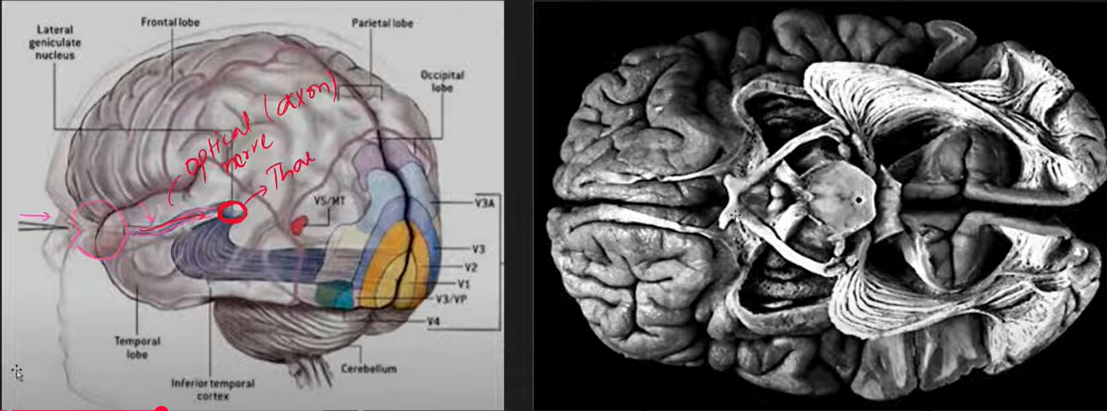
✔ edges  
✔ textures  
✔ shapes  
✔ objects

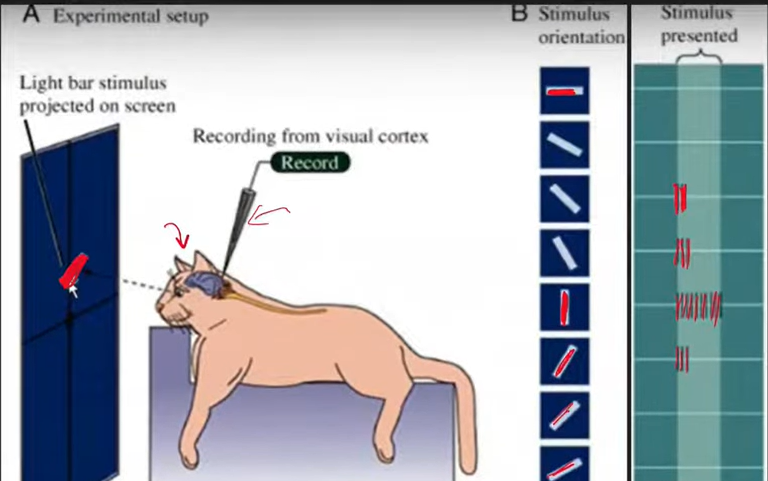


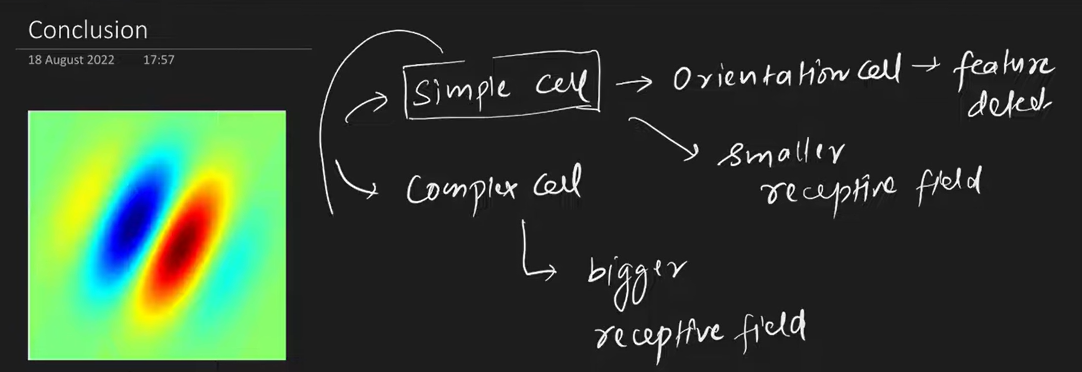
**Structure of CNN**

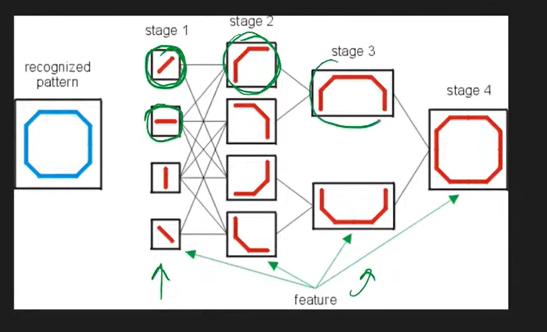
* It has convolution layers that perform convolution operation – helps to extract features from images
* Pooling Layer
* Fully connected layer

Inspiration from Human visual cortex









**Why ANN is not used?**

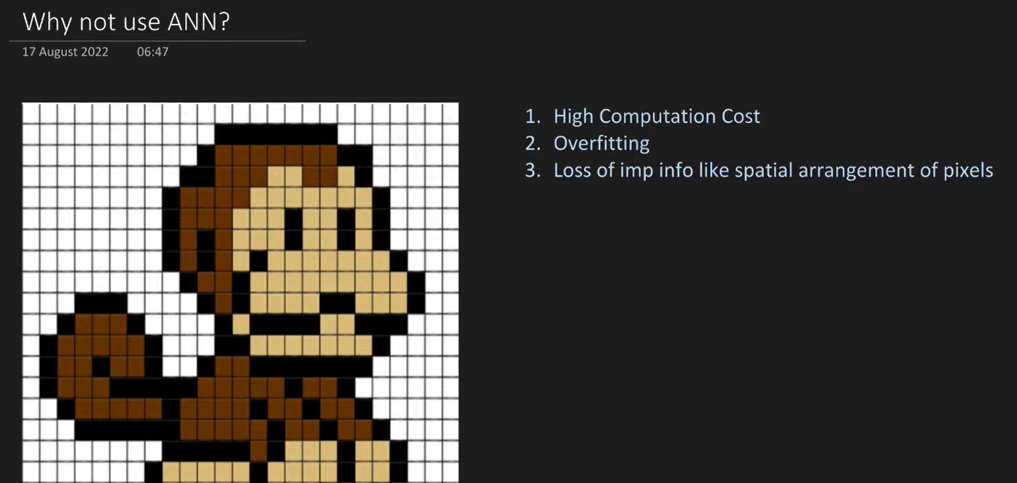


Image is a grid of 2D pixels

40 X 40 image with 100 nodes = 640000 weights needs to be trained

A regular ANN (fully connected) does **not scale well** for images.

Example: A color image of size 224 × 224 pixels has:

224 × 224 = 50,176 input values

A dense layer connecting that to just 1,000 neurons would need:

150,176 × 1,000 ≈ 150 million weights!

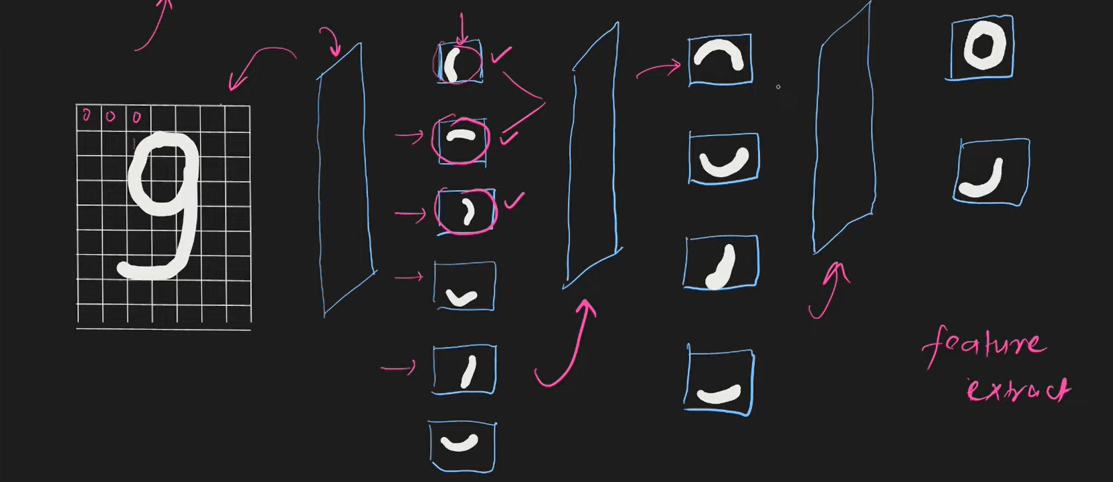
That is **too large + inefficient + prone to overfitting**

CNN solves this!

**Feature Detection in CNN**



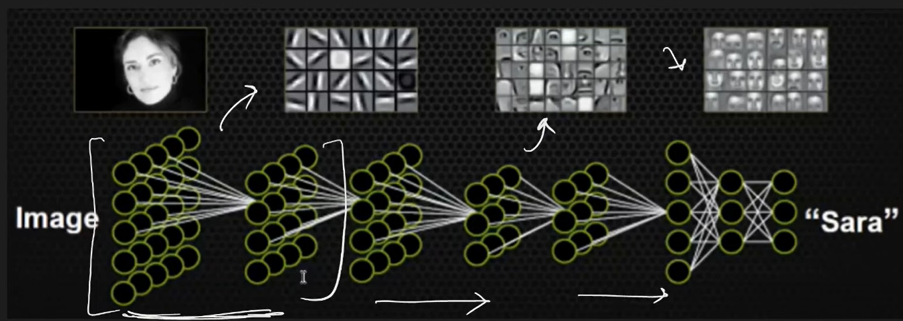




**CNNs are used for tasks like:**

* Image Classification (cat vs dog)
* Object Detection (detect a car in an image)
* Face Recognition (Face ID)
* Medical Image Analysis (MRI scans)
* Vision in Self-driving Cars
* Image Segmentation

**Convolution Operation**

****

Initial Layers detect primitive feature

Next layers detect complex features

Fully connected layers does tasks like classification, segmentation

Images – grayscale and colored

Grayscale – 1 channel 0-255

0 – black

255 – white

Colored – 3 channels

Red, Green, Blue

0 – 255

If red channel has 0 means no red color, if it is 255 means 100% red

A **convolution** uses a small filter (kernel) like **3×3**, **5×5**, etc., sliding over the image to detect features.

Filter slides over the pixels and outputs a feature map highlighting edges.

CNN learns filters automatically during training.

Edge is nothing but intensity change in an image



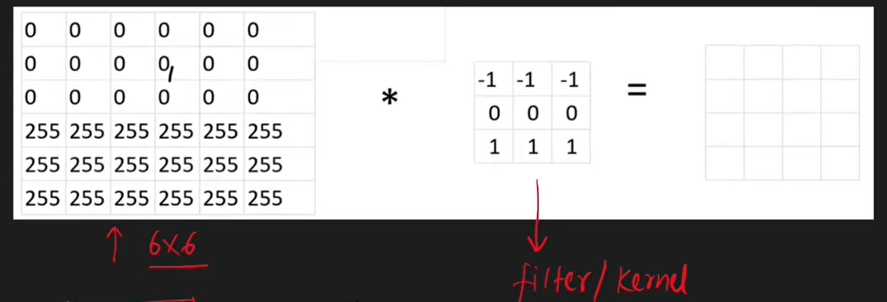
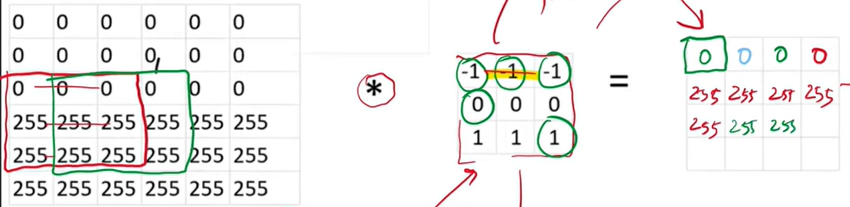
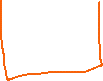
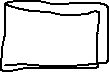


Image > Convolution operation > filter/kernel = feature map

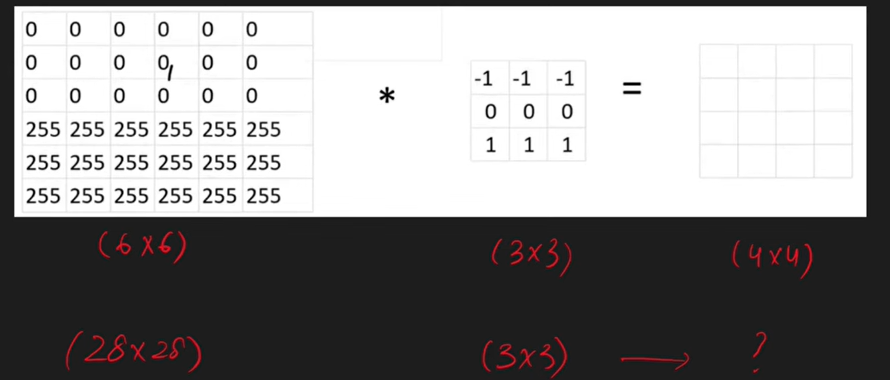




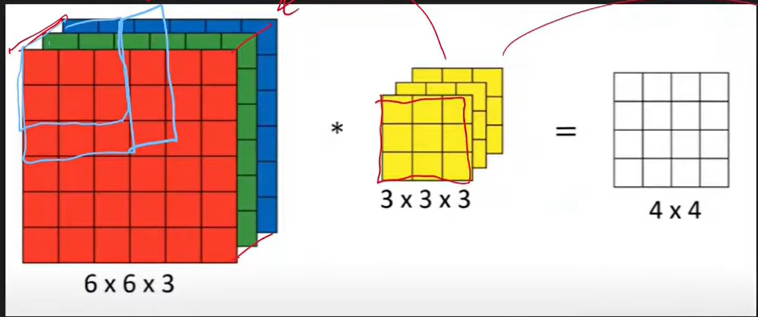


-1 \* 0 + -1 \* 0 + -1\*0 + 0 + 0 = 0

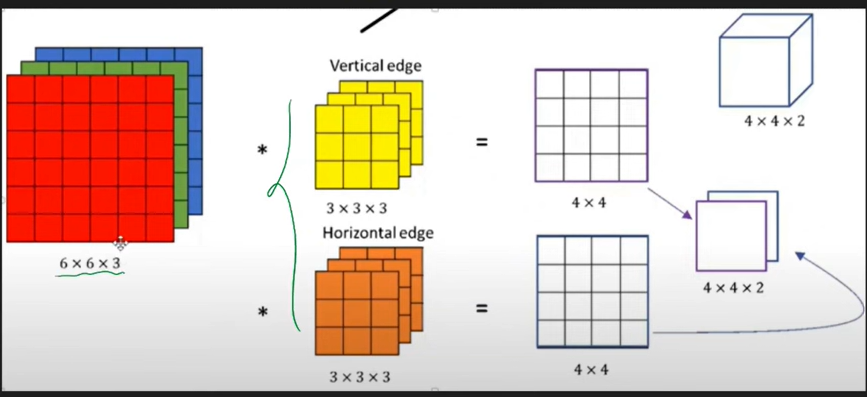




N X N m X m = (N – m + 1) X (N – m + 1)







Demo - <https://deeplizard.com/resource/pavq7noze2>

**Padding**

When you apply a convolution filter to an image, the output shrinks in size unless you add padding.

**Why does output shrink?**

**Example:**Input = 5×5 image  
Filter = 3×3

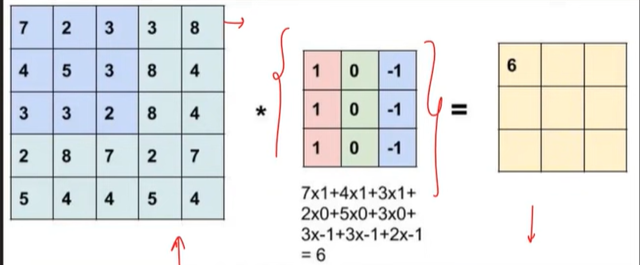
**Output size = (5−3)+1 = 3 → output will be 3×3**

So the image is shrinking and we are loosing information

border information is lost as well

**Padding fixes this**

Padding means adding extra pixels (usually 0s) around the image borders

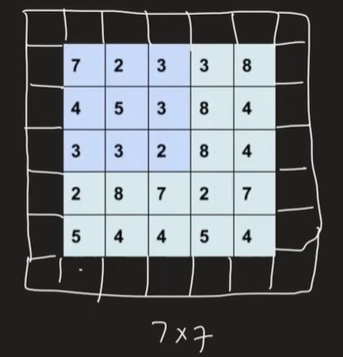


5 X 5 > 3 X 3 = 3 X 3

N – 3 + 1 =5

N – 2 = 5

**N = 7**



**Types of Padding in keras**

Two common types:

**(a) VALID Padding**

* No padding added
* Output **shrinks**
* Computation cheaper

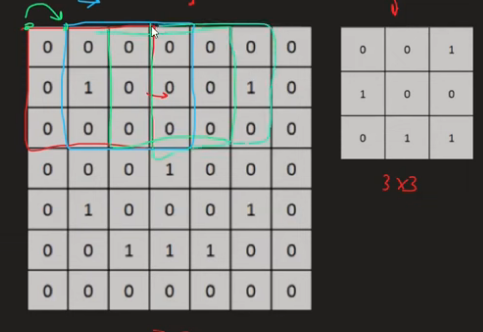
**(b) SAME Padding**

* Adds padding so output size stays same
* Output size ≈ Input size

**👍 Why Padding is useful**

1. Prevents loss of border information
2. Maintains input size for deeper layers
3. Allows using more filters without rapid shrinking

**Strides**

****

Stride = how many pixels you move the filter at each step.

Example

Stride = 1 → move step by 1 pixel  
Stride = 2 → skip 1 pixel each step (more aggressive downsampling)

Example:

Input: 5×5  
Filter: 3×3

Case 1: Stride 1  
Output = (5−3)/1 + 1 = 3 → 3×3

Case 2: Stride 2  
Output = (5−3)/2 + 1 = 2 → 2×2

So stride 2 downsamples (shrinks) the feature map.

**📌 Intuition Summary**

| **Concept** | **Intuition** |
| --- | --- |
| **Padding** | Adds border so output doesn't shrink |
| **Stride** | Controls how fast you move filter (affects output size) |

**🎯 Effect on Output Size**

Formula:

= 7 – 3 + 0/2 = 4/2 = 2 + 1= 3

Where:

| **Symbol** | **Meaning** |
| --- | --- |
| N | Input size |
| F | Filter size |
| P | Padding |
| S | Stride |
| Special Case | 7-3 = 4/2 = 2 + 1 = 3  6-3=3/2=1.5(floor) +1 =2 |

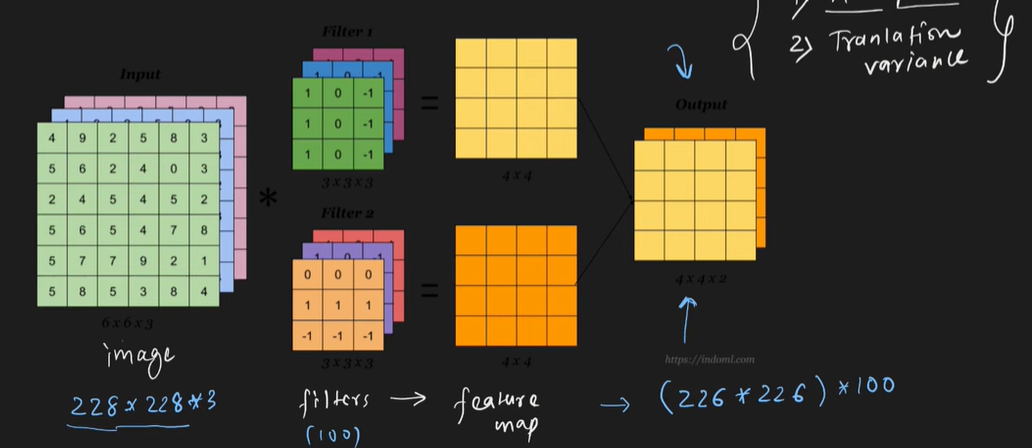
2 X 3 output

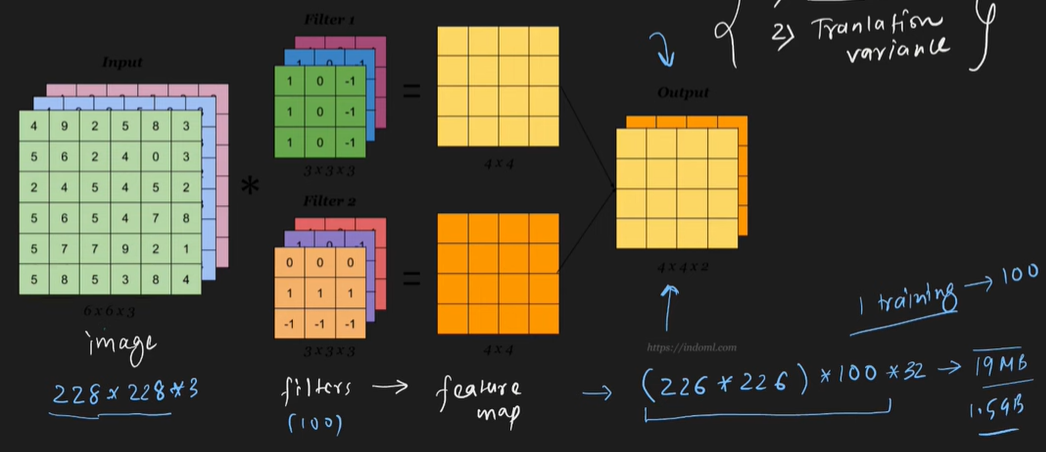


We use **strides** to **control how much we downsample (reduce)** the feature map so that:

✔ fewer computations  
✔ smaller memory usage  
✔ extract useful features without keeping every pixel – only keeps high level features

**Pooling Layer**





2 problem1

* 1. Memory Issues – need to reduce the size of feature map

Stride can solve this problem but there is another problem

* 1. Translation variance

Translation simply means **shifting an object in the image**.

Example: imagine a cat in an image:

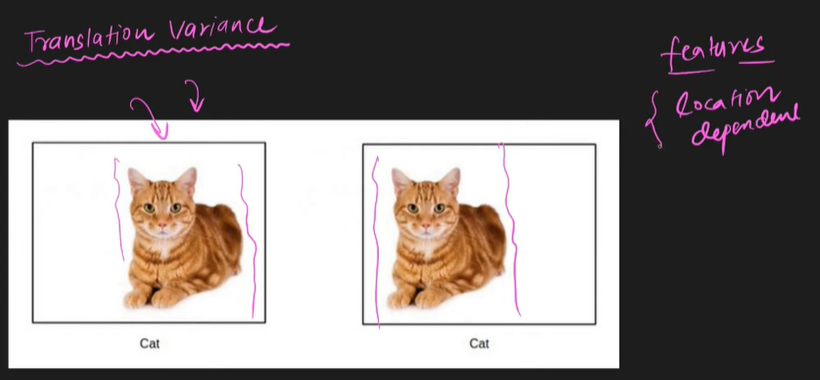
* in the center  
  vs.
* slightly to the left

It’s still a cat, but pixel positions change.

A standard convolution is **sensitive to exact spatial position** — so if the cat shifts a little, the feature map activations also shift.

This property is called:

**Translation variance** — model output changes when the input shifts slightly.



Location dependent

**1. Downsampling (Reduce Spatial Size)**

Large images → large feature maps → large computation

Pooling shrinks the feature maps:

Example:

Input: 32×32  
After pooling: 16×16

Downsampling helps in:

✔ reducing model size  
✔ reducing memory cost  
✔ speeding up training

**2. Provides Translation Invariance**

Meaning → small movements in the input image do **not** change the pooled output much.

Example:

If a cat moves slightly in the image, pooling prevents the feature map from completely changing.

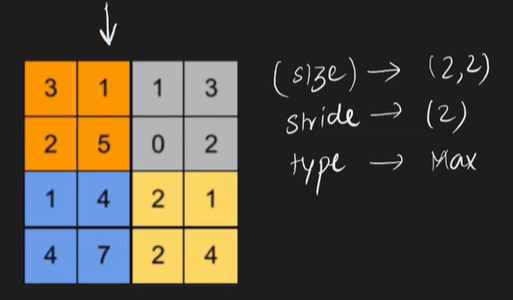
This makes CNNs robust to:

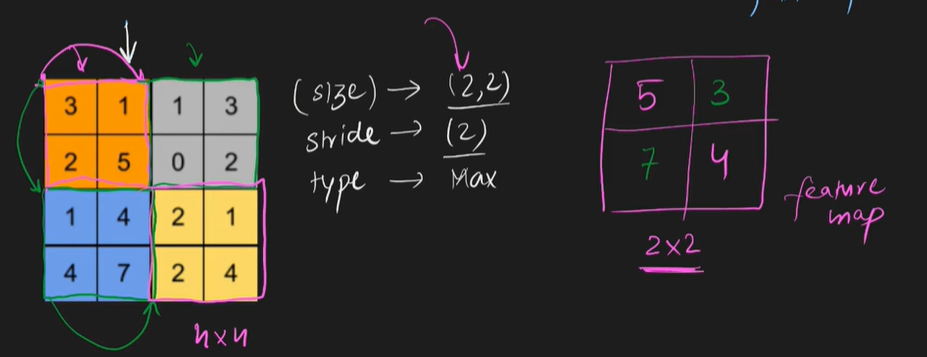
✔ shifts  
✔ distortions  
✔ image noise

Pooling is applied after convolution

Strides are applied during convolution

A **pooling layer** reduces the size of the feature map by summarizing information from a small window (e.g., 2×2), without adding learnable weights.





Two common types:

**1. Max Pooling**

Takes the **maximum** value in the window.

Example:

Window:

[ 1 3

2 4 ]

Max pooling → **4**

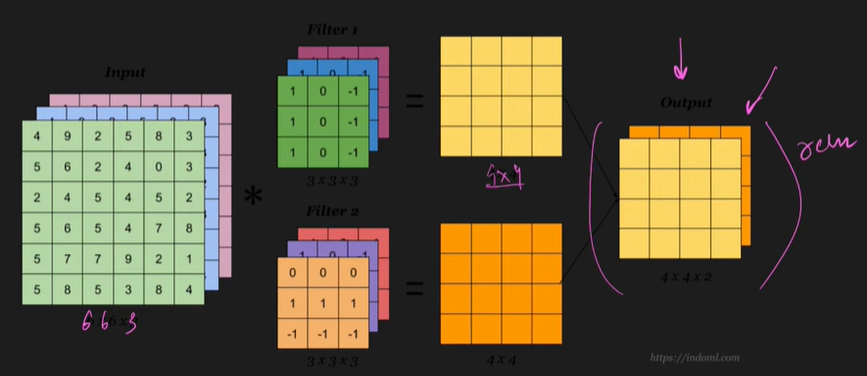
**2. Average Pooling**

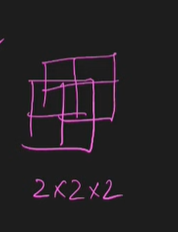
Takes the **average** of the window.

**3. Min Pooling**

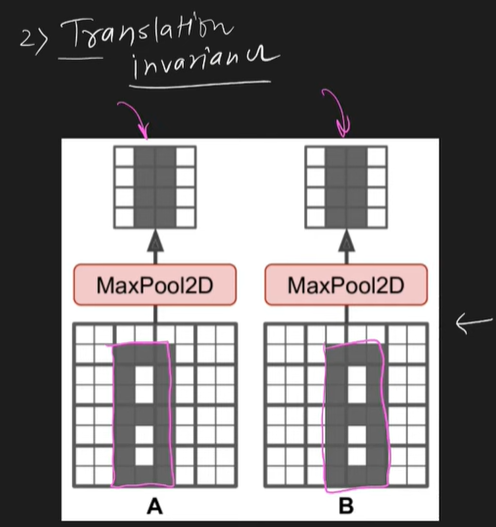
* Takes the **min** value in the window.

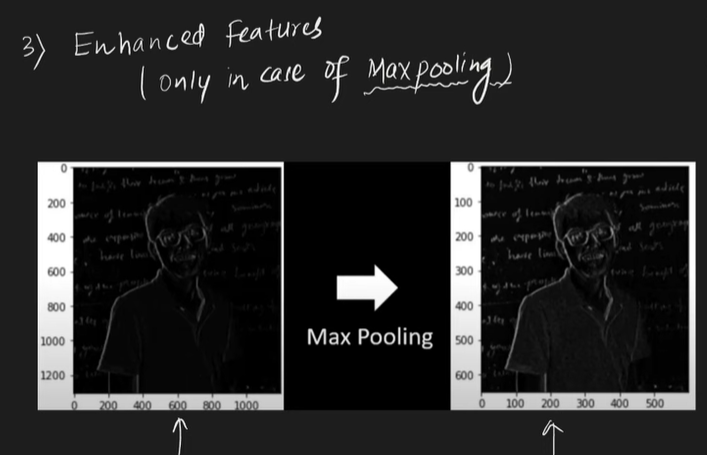
**On RGBA**

****

****

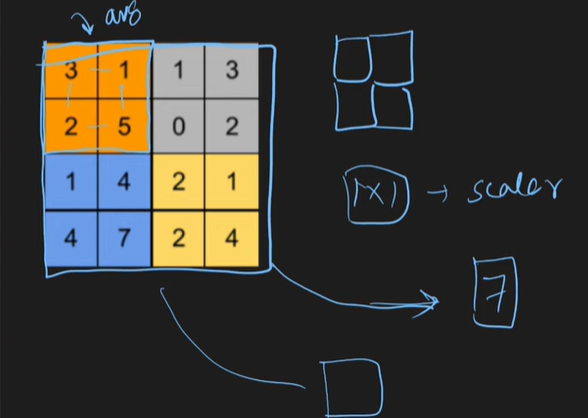
****

****

****

****

**Global Pooling**

****

It is sometimes used as a replacement for flatten

Disadvantages

* 1. Loose a lot of information
  2. Not good **for image segmentation**

**ANN VS CNN**

Problems with ANN

* 1. More weights > High computation cost
  2. Spatial disorientation
  3. Overfitting

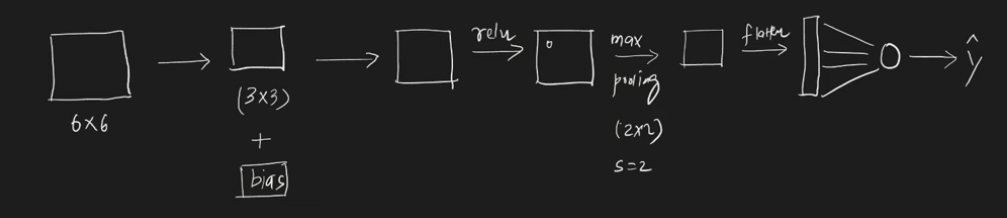
In CNN the trainable parameters remain same regardless the size of the input image

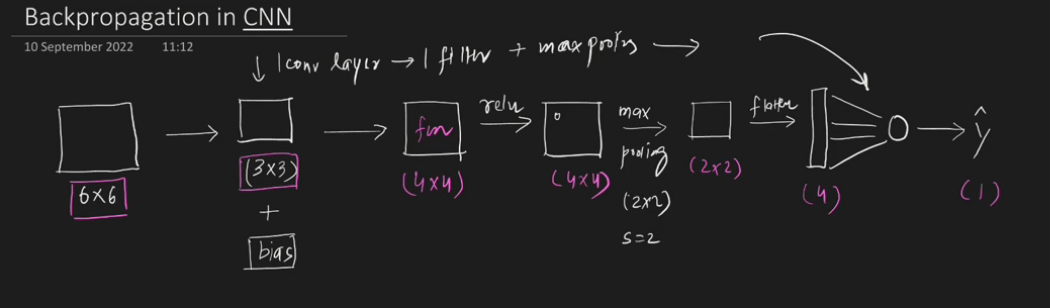
So, it solves computation and overfitting problem

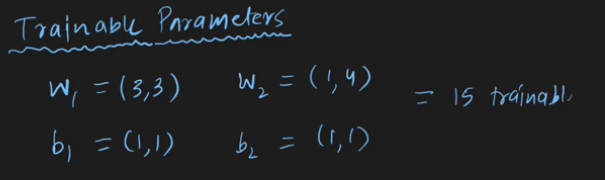
**Calculating the trainable parameters**

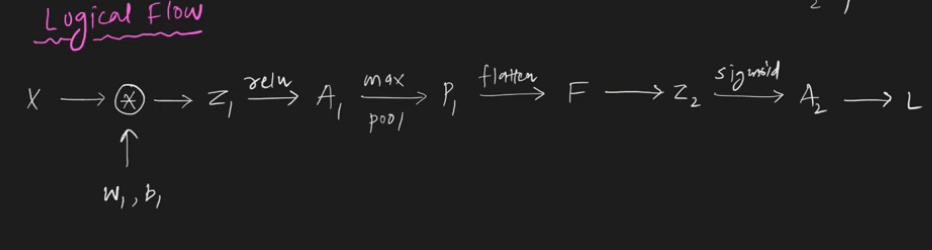
* Conv layers: parameters = (kernel\_size \* kernel\_size \* input\_channels + bias) \* output\_channels
* Dense layers: parameters = (input\_units + 1) \* output\_units
* MaxPooling & Flatten layers: **no parameters**

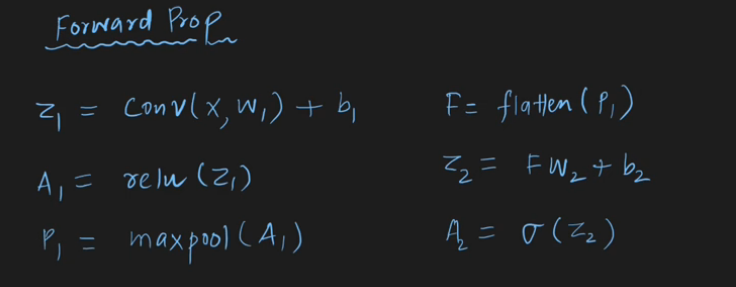
**Backpropagation**











**1️⃣ What is Backpropagation?**

Backpropagation is just the **process of teaching the CNN to get better**.

* Think of it as **giving feedback to the network** after it makes a prediction.
* The network **learns from its mistakes** by adjusting its “knobs” (weights and biases).

**2️⃣ How it works in CNNs (Step by Step)**

**Step 1: Forward Pass**

* You give the CNN an image (e.g., a cat).
* The CNN processes the image through **convolution layers, pooling layers, and dense layers**.
* It predicts **what it thinks the image is** (e.g., dog = 0.8, cat = 0.2).
* Then we **compare** the prediction with the **actual label**.

Forward pass = CNN **making a guess**.

**Step 2: Calculate the Mistake (Loss)**

* The network looks at its prediction and sees **how wrong it is**.
* This “wrongness” is called **loss** or **error**.

Example: CNN predicted 0.8 dog but it’s actually a cat → loss is high.

**Step 3: Backward Pass (Backpropagation)**

* CNN now needs to **fix itself**.
* It asks: “Which parts of me caused this mistake?”

In a CNN:

1. **Dense layers**: Look at each neuron and see how it contributed to the wrong answer.
2. **Convolution layers**: Look at each filter and see if it detected the right features (edges, shapes).
3. **Pooling layers**: Simply pass the “blame” back to the right locations.

Backpropagation = CNN **blaming each weight for the mistake**.

**Step 4: Adjust the Knobs (Weights & Biases)**

* Each neuron and filter has **weights and biases**.
* The CNN changes these slightly in the **direction that reduces the error**.
* This is repeated for **many images**, and slowly the CNN **gets better at recognizing cats and dogs**.

Analogy:

* Forward pass → CNN guesses what’s in the image.
* Loss → Teacher says “this is wrong by this much.”
* Backprop → CNN looks at all its internal parts and says, “oh, I need to tweak these knobs a bit.”
* Repeat → CNN becomes more accurate.

**Step 5: Repeat**

* The CNN goes through **thousands of images**, repeatedly doing:  
  **Guess → Check Mistake → Adjust → Guess again**.
* Eventually, the CNN learns the **patterns that define cats and dogs**.

**🧩 What is a Fully Connected Layer in CNN?**

A **Fully Connected (FC) layer**, also called a **dense layer**, is the layer where **every neuron is connected to every neuron in the previous layer**.

It's usually placed **towards the end of the CNN**.

**🔍 Role of CNN vs FC Layer (Intuition)**

CNN has **two major parts**:

**1. Feature Extraction (Convolution + Pooling)**

These layers learn to detect **patterns in the image**, such as:

✔ edges  
✔ corners  
✔ textures  
✔ shapes  
✔ objects

This outputs a **feature map** (spatial features).

**2. Classification (Fully Connected Layers)**

The feature map is then **flattened** into a vector and fed to FC layers to:

✔ combine features  
✔ make final decision / prediction

**. Components of a CNN Architecture**

A typical CNN has these layers:

**(A) Convolutional Layer**

Responsible for feature extraction  
Formula output size:

(W − F + 2P) / S + 1

Where:  
W=width, F=filter size, P=padding, S=stride

**(B) Activation Function (ReLU)**

**(C) Pooling Layer (Downsampling)**

Reduces spatial dimensions & computation

Types:

* Max Pooling ✔ (most used)
* Average Pooling

Example: 2×2 max pool picks max value

**(D) Flatten Layer**

Converts 2D feature maps → 1D vector

**(E) Fully Connected Layer**

Performs classification (softmax / sigmoid)

**📦 6. Typical CNN Architecture Example**

Example for Image Classification:

Input Image (224×224×3)

↓

Conv Layer + ReLU

↓

Max Pooling

↓

Conv Layer + ReLU

↓

Max Pooling

↓

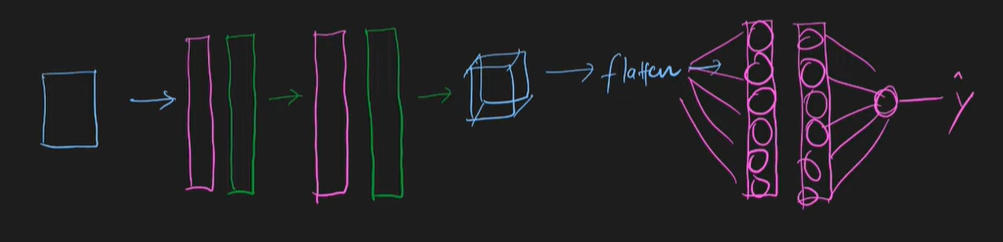
Flatten

↓

Fully Connected Layer

↓

Softmax Output (e.g., 10 classes)



**📉 7. Hyperparameters in CNN**

Common hyperparameters:

✔ Filter size (3×3, 5×5)  
✔ Number of filters (32, 64, 128…)  
✔ Stride (1,2)  
✔ Padding (same, valid)  
✔ Pool size (2×2)  
✔ Learning rate  
✔ Batch size  
✔ Epochs

**🧪 9. Training a CNN: What happens?**

CNN trains via **backpropagation**, adjusting filters to reduce loss.

Loss used for image classification often:

* Categorical Cross Entropy

Optimizer often:

* Adam / SGD

**🧠 10. Advantages of CNN**

✔ Works well on images  
✔ Learns features automatically  
✔ Fewer parameters than fully connected  
✔ Translation invariance  
✔ High accuracy

**⚠️ 11. Disadvantages / Challenges**

✖ Data hungry (needs large datasets)  
✖ Computationally expensive (requires GPU)  
✖ Poor performance on non-visual sequential data (unless modified)