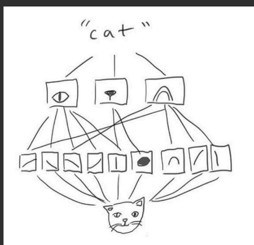
- Unlike neural networks, who recognise global patterns, convolutional networks learn local patterns
- Patterns are transition invariants, convolutional networks recognise pattern everywhere in the image.

## Hierachy of images




## Padding

- Pixels at the **edges and corners** get processed fewer times compared to central pixels, leading to less information being extracted from them.

**Solution?**

- **Padding**: Adding extra pixels (usually zeros) around the image ensures that edge and corner features are better captured.

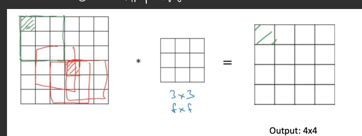
**Padding** in convolutional neural networks (CNNs) refers to adding extra pixels (usually zeros) around the input image before applying the convolution operation. This helps in:

1. **Preserving spatial dimensions** – Ensures the output size remains the same as the input.
2. **Handling edge features** – Allows filters to process edge pixels effectively.
3. **Avoiding information loss** – Prevents shrinking of the image after multiple convolutions.

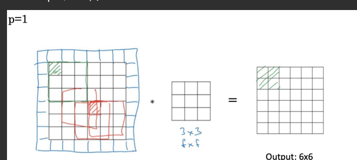
Common types:

- **Valid padding (no padding)** – Reduces image size.
- **Same padding (zero-padding)** – Keeps image size unchanged.

NO PADDING




PADDING




Suppose the kernel move 1px at time.

Let:

$m :=$ image input size $(x, y)$

$f :=$ kernel size

the output is of size:

$$m - f + 1 \times m - f + 1$$

Given that $f \geq 2 \Longrightarrow m - f + 1 \leq m$

$\Longrightarrow$ so the output image is smaller

To keep the same dimension we add padding

$$m + 2p - f + 1 \times m + 2p - f + 1$$

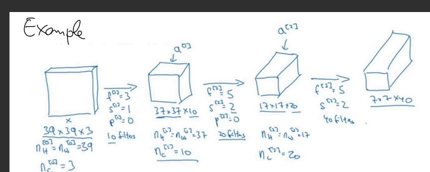
so to have the same dimension:

$$m + 2p - f + 1 = m \Longrightarrow p = \frac{f-1}{2}$$

## Strided convolution

- Kernel moves s pixels a time
- If the kernel moves s pixels a time, the output dimension is:

$$\left\lfloor \frac{(n+2p-f)}{s} + 1 \right\rfloor \times \left\lfloor \frac{(n+2p-f)}{s} + 1 \right\rfloor$$




## Pooling

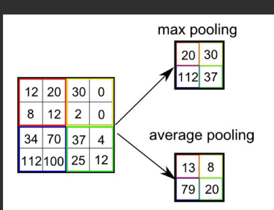
- decrease the dimension of one

Hyper parameters:

$f :=$ dimension filter

$s :=$ stride

Average, max

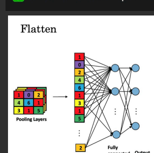


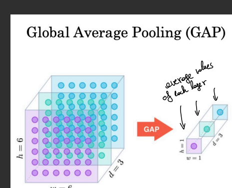

## Flatten

Layer to convert multi dimensional tensor into 1D vector

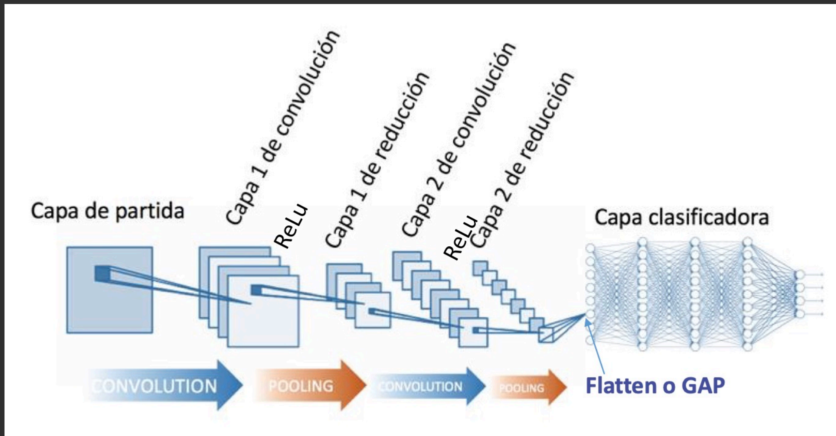
**Why is Flatten Needed?**

✅ Bridges Convolutional and Fully Connected Layers
✅ Prepares Data for Classification Tasks
✅ Maintains Feature Representation in a Vector Form



# Example




Compute number of parameters for each convolutional layer

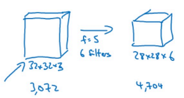
**Formula for the Number of Parameters:**

$$\text{Parameters} = (f \times f \times C_{\text{in}} + 1) \times C_{\text{out}}$$

Where:

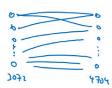
- $f \times f$ = **Filter size** (e.g., $3 \times 3$, $5 \times 5$)
- $C_{\text{in}}$ = **Number of input channels** (e.g., 3 for an RGB image)
- $C_{\text{out}}$ = **Number of output channels (filters)**
- **+1** accounts for the **bias** associated with each filter.

Why CNNs are used?




5x5x3+1=76 parámetros por filtro
76x6 filtros =456 parámetros

Con una fully connected:




3072x4704 =14.5 millones de parámetros

Using only 9 parameters it output 16 values and analyses the whole image

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

to compute a specific out pixel it uses a subset of values

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

# ResNet

- Solves vanishing gradient problem using skip connection.

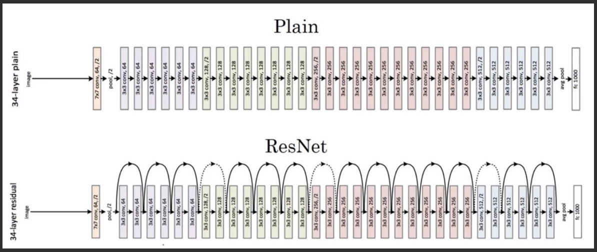
**Why Was ResNet Introduced?**

As neural networks get **deeper**, they: ❌ **Suffer from vanishing gradients** → Training becomes difficult.

❌ **Lose feature information in deeper layers** → Performance degrades.

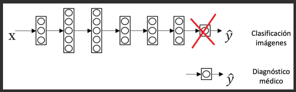
❌ **Face degradation problem** → Accuracy saturates and may decrease.

🚀 **ResNet solves this by introducing "Residual Learning" using skip connections!**




# Transfer learning

- use a pre-trained network to solve a different task.

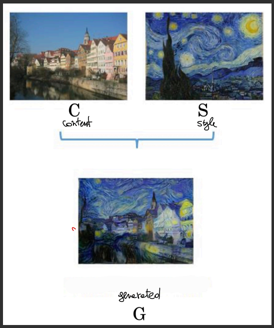



- In this example we can train the network using medical images
- Fine tuning

When can be used to transform A in B?

- Same input type for A and B
- More data in A that B
- Low level characteristics of A are usefull for B

# Neural Style transfer



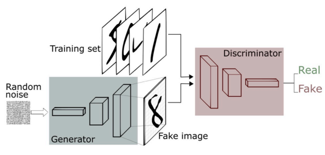

$$\boxed{J(G) = \alpha\, J_{content}(C,G) + \beta\, J_{style}(S,G)}$$

$\alpha, \beta$ hyper parameters that decides if the new image will looks like more C or G

- **Content Cost Function:** $J_{\text{content}}$
- We select a layer $l$ in the middle of the network (neither too early nor too late).
- We use a **pre-trained convolutional network**.
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation values at layer $l$ for the original image $C$ and the generated image $G$, respectively.
- We want $J_{\text{content}}$ to measure how similar these two activations are.
  - If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, the images will have a similar content.
- Thus, we define $J_{\text{content}}$ as:

$$J_{\text{content}}(C,G) = \frac{1}{2}||a^{[l](C)} - a^{[l](G)}||^2$$

# Generative Adversarial Network




# Vulnerability in vision systems