Introdução à Aprendizagem Automática (IAA)

SUSANA BRÁS

SUSANA.BRAS@UA.PT

IAA – L10

Deep Learning: why, what, when.

Convolution Neural Network

- Basic building blocks: convolution (filter, stride, padding), pooling, fully connected
- Classical CNN

Transfer Learning: what, why

Image data augmentation

Hierarchical layer learning

Data Matrix

matrix X (mxn)	feature x_1	feature x_2	.....	feature x_n	Target Y
Example 1	$x_1^{(1)}$	$x_2^{(1)}$		$x_n^{(1)}$	$Y^{(1)}$
Example 2	$x_1^{(2)}$	$x_2^{(2)}$		$x_n^{(2)}$	$Y^{(2)}$
…					
Example i	$x_1^{(i)}$	$x_2^{(i)}$		$x_n^{(i)}$	$Y^{(i)}$
…					
…					
Example m	$x_1^{(m)}$	$x_2^{(m)}$		$x_n^{(m)}$	$Y^{(m)}$

x – input vector of features, attributes
y – output vector of labels, ground truth, target
m - number of training examples
n – number of features
$h_\theta(x)$ - model (hypothesis)
θ - vector of model parameters

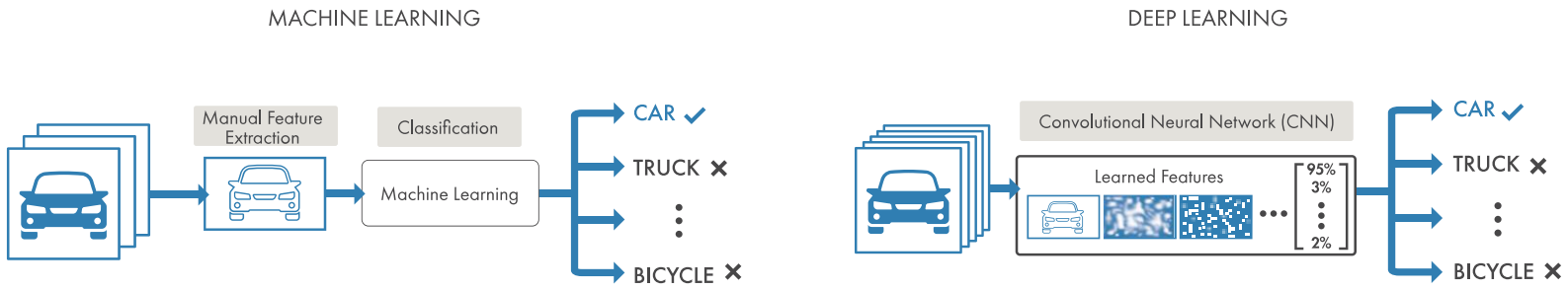
Training set: data matrix X (m rows, n columns)

Deep Neural Networks (DNN)

Deep Learning

Deep learning is a powerful subset of machine learning that utilizes artificial neural networks with multiple layers to learn from data and perform complex tasks.

Learning from Data:
Unlike traditional machine learning, deep learning models can learn directly from data, automatically extracting features and patterns without explicit feature engineering.



Deep Learning vs. Conventional Machine Learning: Which to Choose?

Deep Learning

What are the benefits of using deep learning models?

•**Can learn complex relationships between features in data:** This makes them more powerful than traditional machine learning methods, by eliminating the need for manual feature extraction.

•**Large dataset training:** This algorithms can scale with the amount of data and computational power available. As more data becomes available, these models continue to improve and adapt, leading to better performance.

•**Data-driven learning:** DL models can learn in a data-driven way, requiring less human intervention to train them, increasing efficiency and scalability. These models learn from data that is constantly being generated, such as data from sensors or social media.

• **Versatility:** Deep learning is applicable across various domains, from image and speech recognition to time-series forecasting and robotics. Its versatility makes it a valuable tool for solving complex problems in diverse fields.

Challenges of using deep learning models:

•**Computational Resources:** Training deep learning models can be resource-intensive, requiring powerful hardware.

•**Data requirements:** Deep learning models require large amounts of data to learn from, making it difficult to apply deep learning to small dataset problems.

•**Overfitting:** DL models may be prone to overfitting. This means that they can learn the noise in the data rather than the underlying relationships.

•**Bias:** These models can potentially be biased, depending on the training data. This can lead to unfair or inaccurate predictions.

•**Interpretability:** DL models are often considered "black boxes" due to their complexity, making it challenging to interpret their decision-making processes. This lack of transparency can be problematic, especially in critical applications.

•**Ethical Concerns:** The use of deep learning raises ethical concerns, including issues related to privacy, bias, and fairness. Ensuring that models are trained on diverse and representative data is crucial to addressing these concerns

CNN - basic building blocks

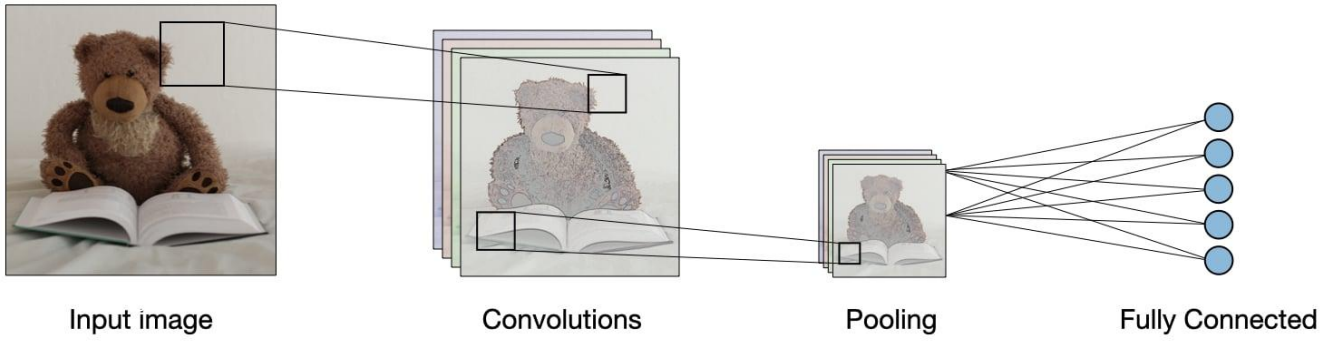
CNN

Convolutional Neural Networks (CNNs), are a type of deep learning algorithm specifically designed for processing and analyzing data with a grid-like structure, such as images. They excel at extracting **spatial** features and patterns from visual data.

Convolutional Layers:
These layers apply a filter (kernel) across the input, learning spatial hierarchies of features from low-level to high-level patterns.

Pooling Layers:
These layers down-sample the feature maps, reducing dimensionality and computational cost while preserving important features.

Fully Connected Layers:
These layers connect all neurons from the previous layer to all neurons in the current layer, enabling classification or prediction based on the extracted features.

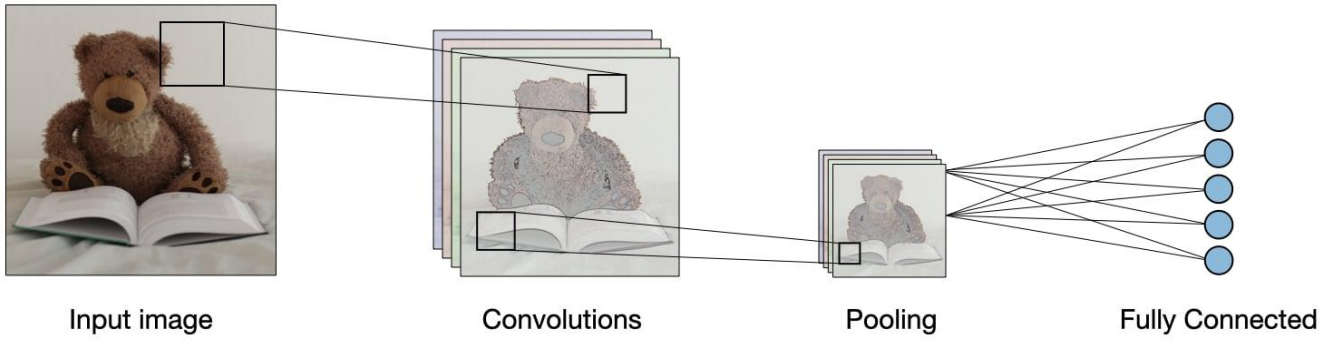


CNN

Feature Extraction:
CNNs learn to automatically extract meaningful features from data, such as edges, textures, and objects, without explicit feature engineering.

Hierarchical Feature Learning:
CNNs learn hierarchical representations of features, where each layer builds upon the previous layer's output, creating a progressively more complex understanding of the input.

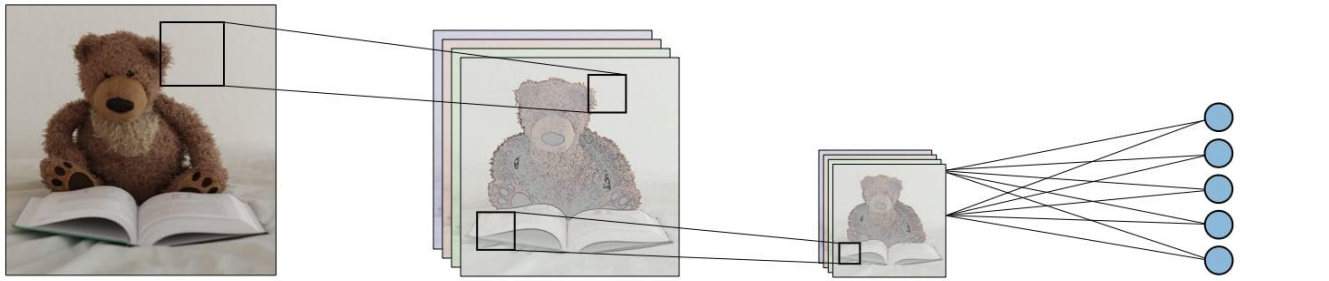
Applications:
In essence, CNNs leverage the power of deep learning and specialized architecture to efficiently analyze and understand patterns in grid-like data, making them a powerful tool for a wide range of applications.



CNN

Important Parameters in Convolution:

- Filter Dimension
- Stride
- Zero-Padding

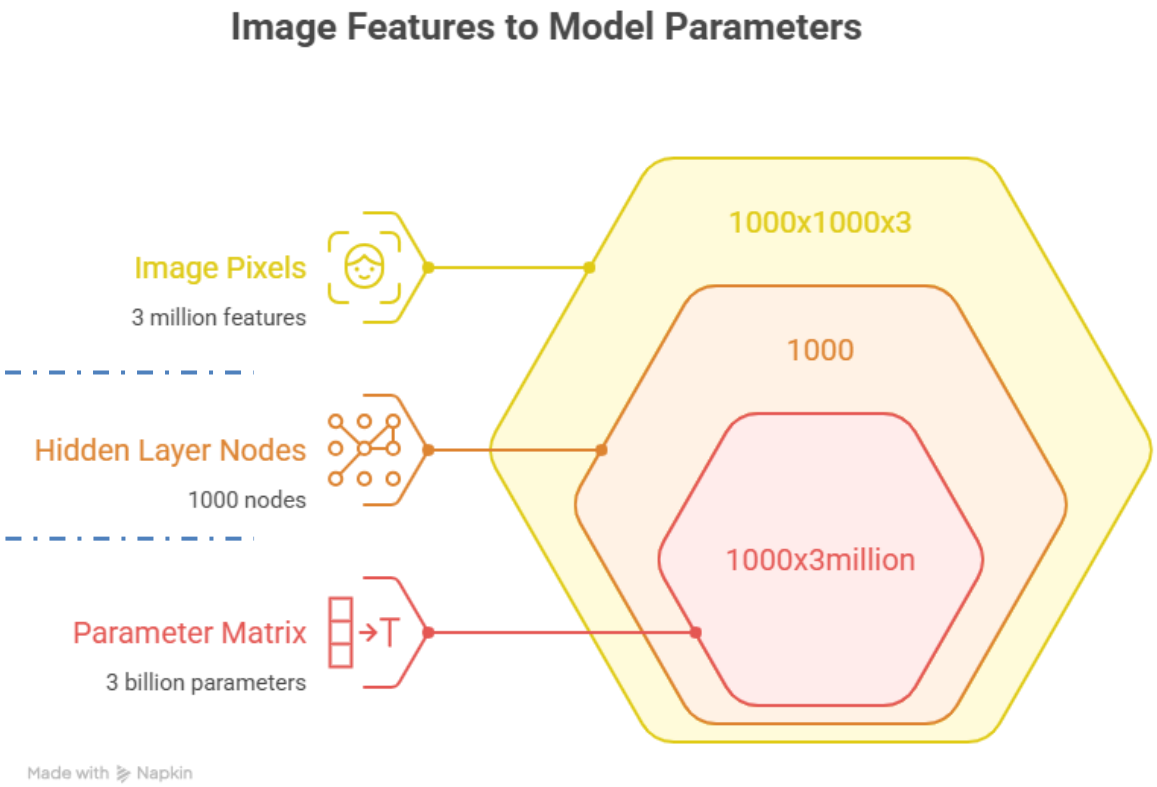


Why Convolution Learning?

Imagine an image with **1000x1000x3** (RGB) pixels, resulting in **3 million** features (inputs).

Assuming that the first hidden layer has **1000** nodes.

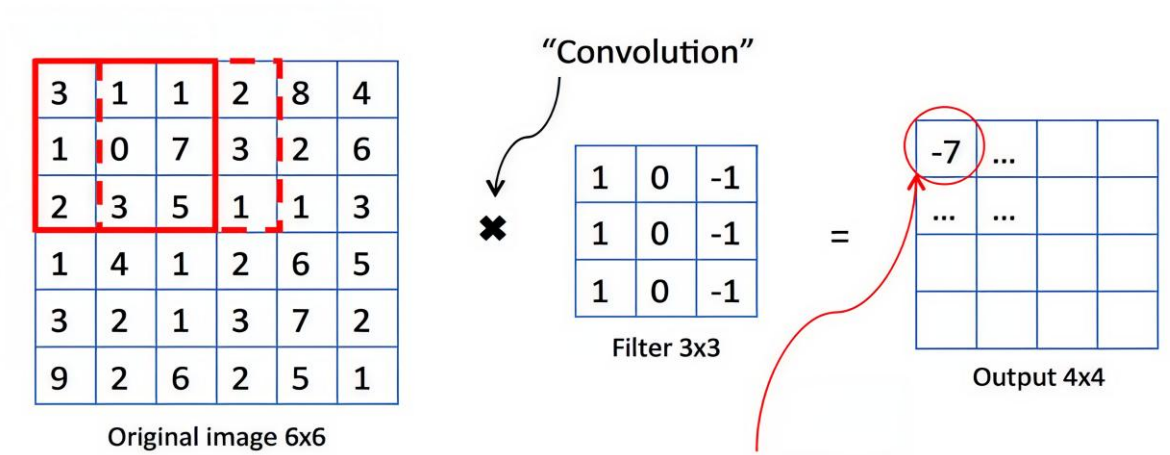
This implies that the parameter matrix between the input and the hidden layer has (1000x3million) 3billion parameters.



1st problem: Difficult to get enough data to prevent model overfitting
2nd problem: Computational (memory) requirements to train such networks are not feasible.

Why Convolution Learning?

Solution: implement convolution operation



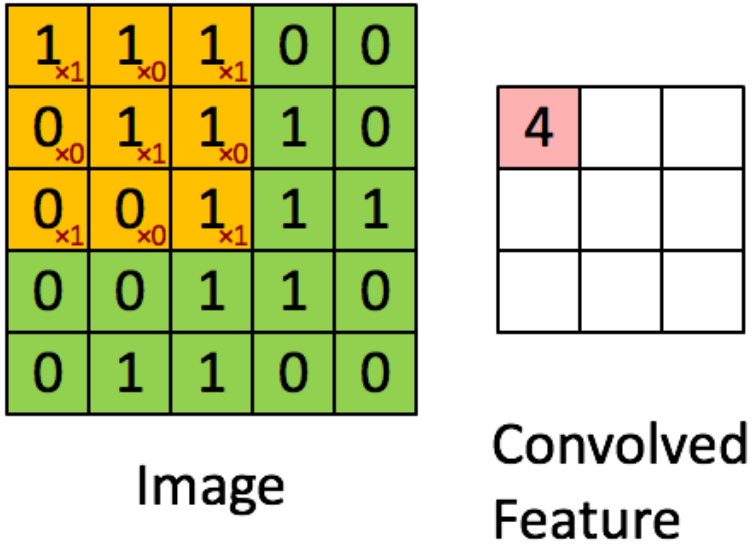
Result of the element-wise product and sum of the filter matrix and the original image

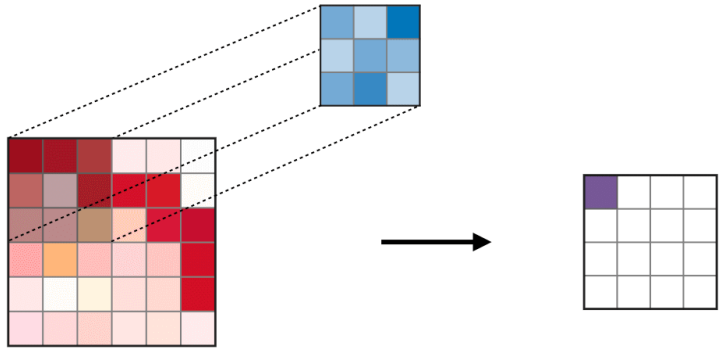
Convolution

The convolution layer (CONV) uses filters that perform convolution operations as it scans the input for its dimensions.

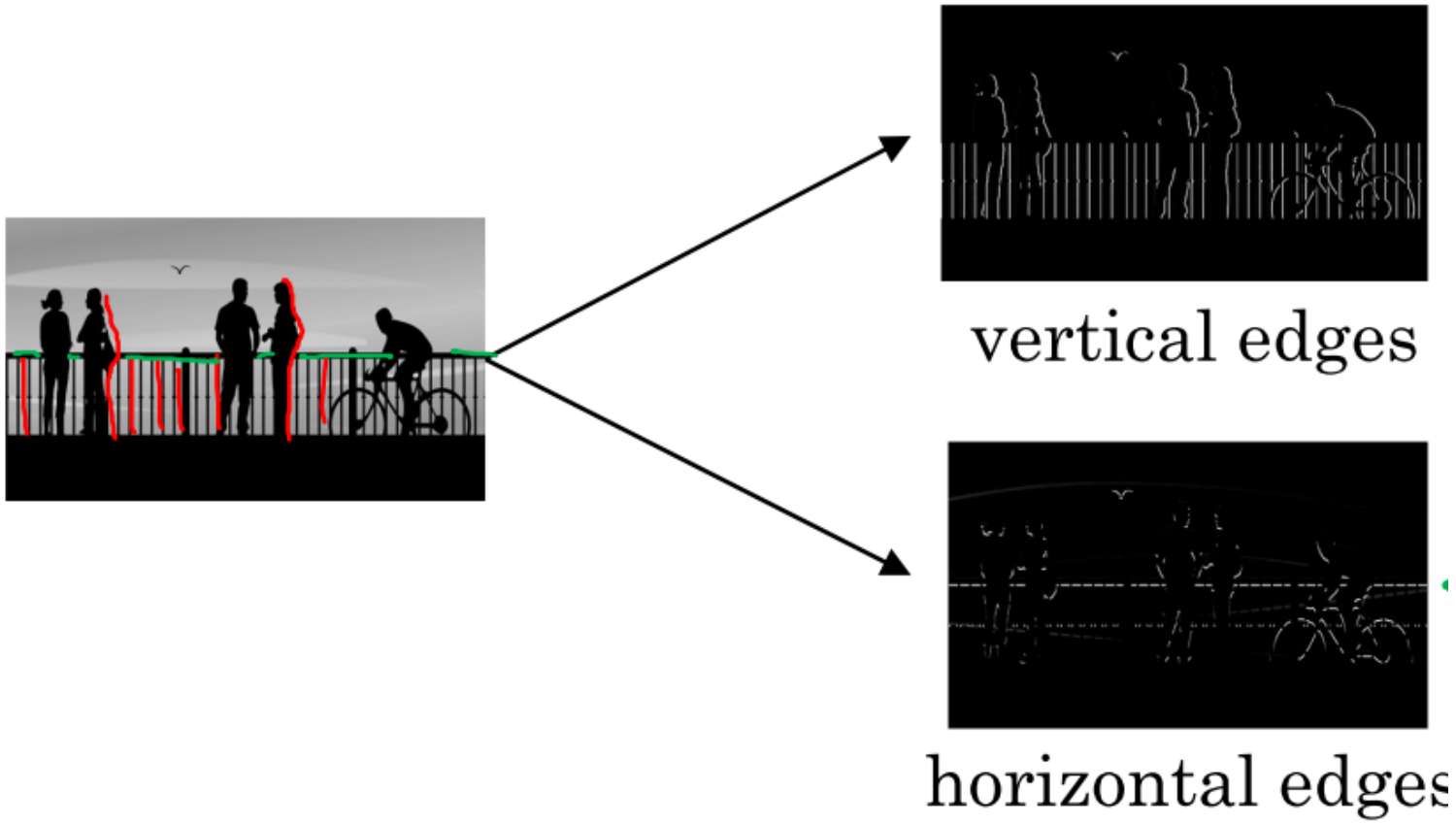
Its hyperparameters include the filter size F and stride S.

The resulting output O is called a ***feature map*** **or** ***activation map***.





Goal: Detect horizontal / vertical edges



Example: Vertical edges detection

Goal: Detection of bright to dark transition (+30)

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

*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

*

Hand-picked convolutional filters (kernels)

Horizontal edge detector

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

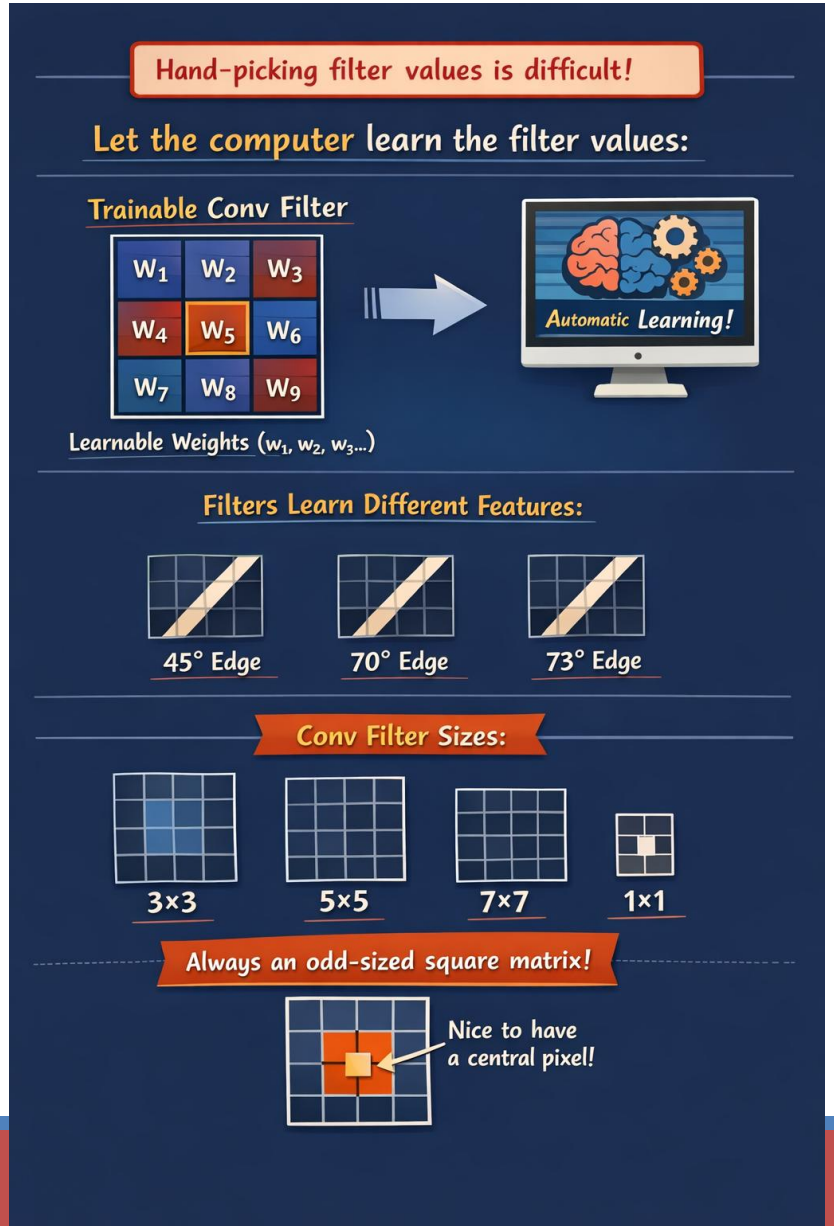
Sobel filter

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Sharr filter

$$\begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$$

Convolutional filters (kernels)



NOTES:

- ✓ The filter values should be treated as trainable parameters (w) and the computer will learn from them.
- ✓ Other than vertical and horizontal edges can learn information from different angles (e.g., 45º, 70º, 73º) and become more robust than hand-picking values.
- ✓ By convention, the conv filter is a square matrix with an odd size (typically 3x3, 5x5, 7x7, also 1x1).
- ✓ A central pixel facilitates the padding.

Padding

$$\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{bmatrix}$$

Example: Take a 6 × 6 image, apply a 3 × 3 conv filter, get a 4 × 4 output matrix, because we shift the filter one row down or one column right, and therefore we have 4 × 4 possible positions for the 3 × 3 filter to appear in the 6 × 6 input matrix.

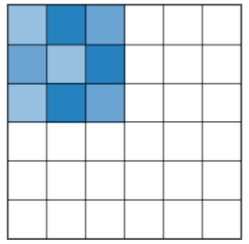
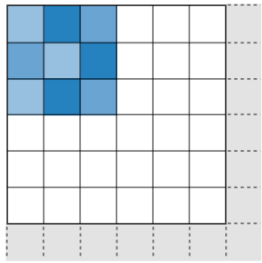
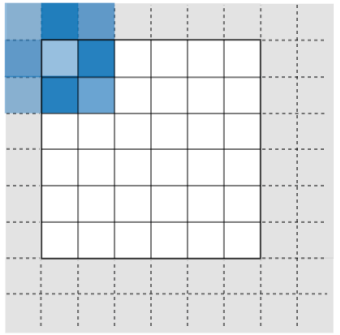
In general, given an $n \times n$ input matrix and $f \times f$ filter matrix, the convolution operation will compute an $(n-f+1) \times (n-f+1)$ output matrix by applying one pixel up/down left/right rule.

1st problem: Shrink the matrix size as we continue applying convolution. The image will get very small if we have many convolution layers.

2nd problem: Pixels at the corners of the image are used only once, while pixels in the centre of the image are used several times. This is uneven, loose of information.

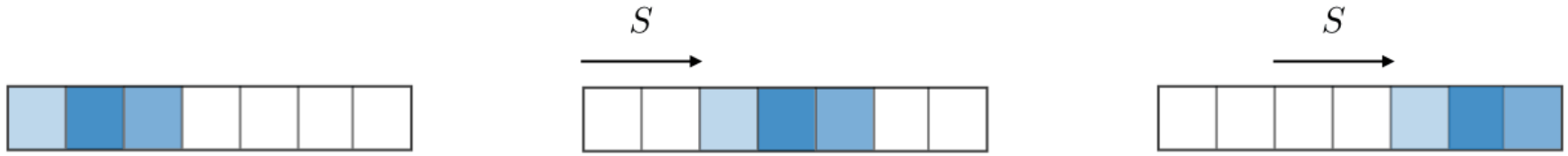
Padding

Zero-padding denotes the process of adding *P* zeroes to each side of the boundaries of the input. This value can either be manually specified or automatically set through one of the three modes detailed below.

Mode	Valid	Same	Full
Illustration			
Purpose	• No padding • Drops last convolution if dimensions do not match	• Padding such that feature map size has original size. • Output size is mathematically convenient • Also called 'half' padding	• Maximum padding such that end convolutions are applied on the limits of the input • Filter 'sees' the input end-to-end

Stride

For a convolutional or a pooling operation, the stride S denotes the number of pixels by which the window moves after each operation.



Increasing the stride will allow the filter to cover a **larger area of the input image**, which can be useful for capturing **more global features**.

In contrast, **lowering** the stride will capture **finer and more local details**.

In addition, increasing the stride will control **overfitting** and **reduce computational efficiency** as it will reduce the spatial dimensions of the feature map.

Strided Convolution

$$\begin{bmatrix} 2 & 3 & 7 & 4 & 6 & 2 & 9 \\ 6 & 6 & 9 & 8 & 7 & 4 & 3 \\ 3 & 4 & 8 & 3 & 8 & 9 & 7 \\ 7 & 8 & 3 & 6 & 6 & 3 & 4 \\ 4 & 2 & 1 & 8 & 3 & 4 & 6 \\ 3 & 2 & 4 & 1 & 9 & 8 & 3 \\ 0 & 1 & 3 & 9 & 2 & 1 & 4 \end{bmatrix} * \begin{bmatrix} 3 & 4 & 4 \\ 1 & 0 & 2 \\ -1 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 91 & 100 & 83 \\ 69 & 91 & 127 \\ 44 & 72 & 74 \end{bmatrix}$$

In general, given an $n \times n$ input matrix and an $f \times f$ filter with padding p and stride s, the convolution operation will compute an output matrix with size:

$$\lfloor \frac{n+2p-f}{s} + 1 \rfloor \text{ by } \lfloor \frac{n+2p-f}{s} + 1 \rfloor$$

If the formula computes a non-integer value, the closest lower integer should be chosen.

Ex.:
no padding (p=0) , stride s=2
(7 + 0 − 3)/2 + 1 = 4/2 + 1 = 3 => 3 ×3 matrix

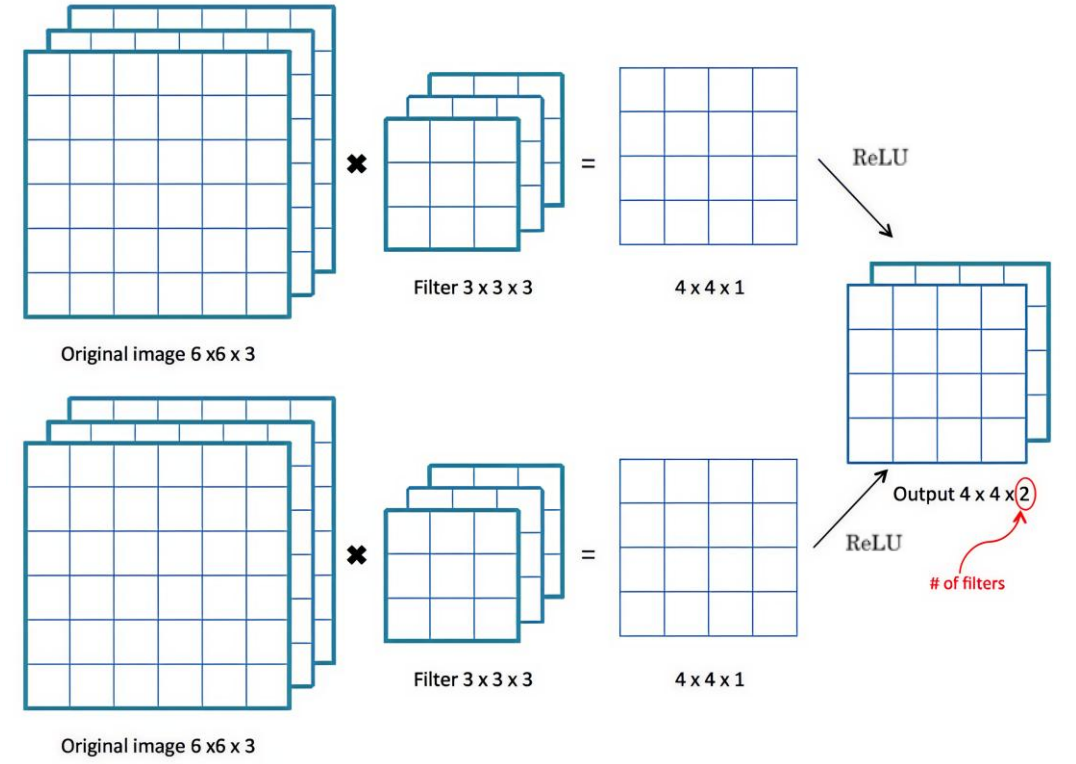
Multiple Conv Filters (3D filters)

RGB images have 3 dimensions: height, width, and number of channels (3D volume).

The conv filter will also be a 3D volume: *f × f × 3* (number of channels is the same for image and filter)

*$(n \times n \times n_c)$ image * $(f \times f \times n_c)$ filter*

(n-f+1) x (n-f+1) x n_filters (no padding)



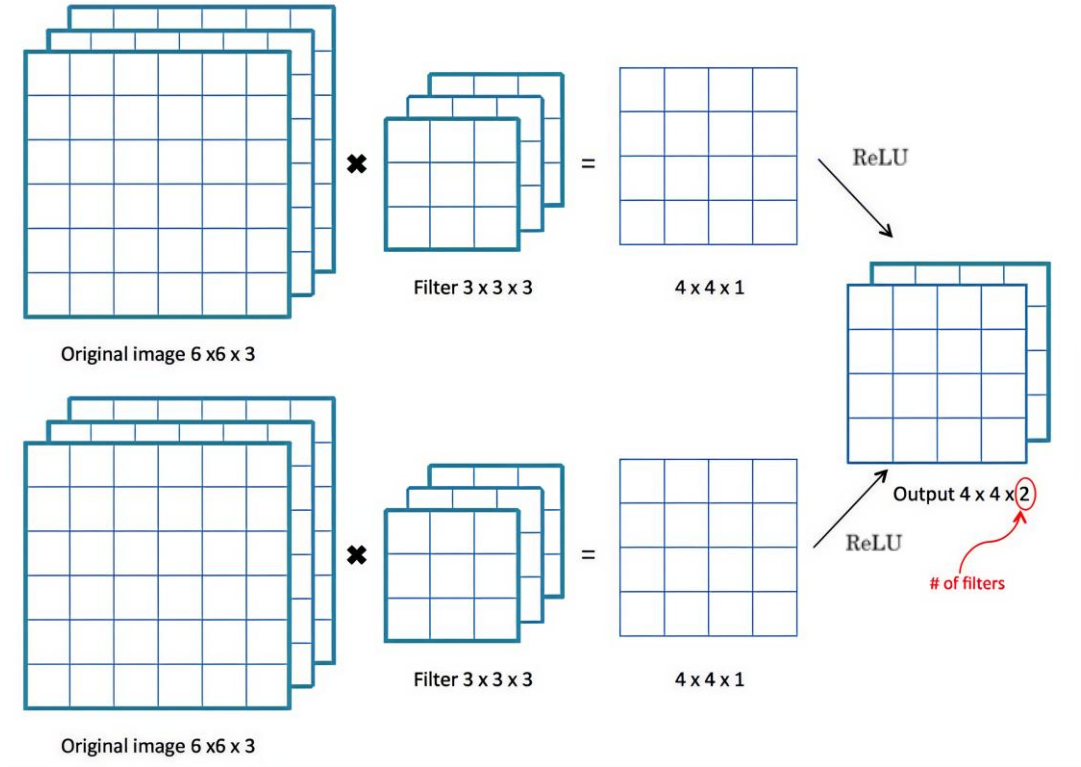
Feature maps

ONE CONV LAYER OF CNN

Different 3D filters (kernels) are applied to the 3D input image and the result matrices are stacked to form a 3D output volume.

After the convolution operation the result is passed through an activation function (e.g. ReLU, or sigmoid, linear, etc.).

The outputs of the conv layers are known as feature maps.



Feature maps

Number of parameters in one layer

Example:

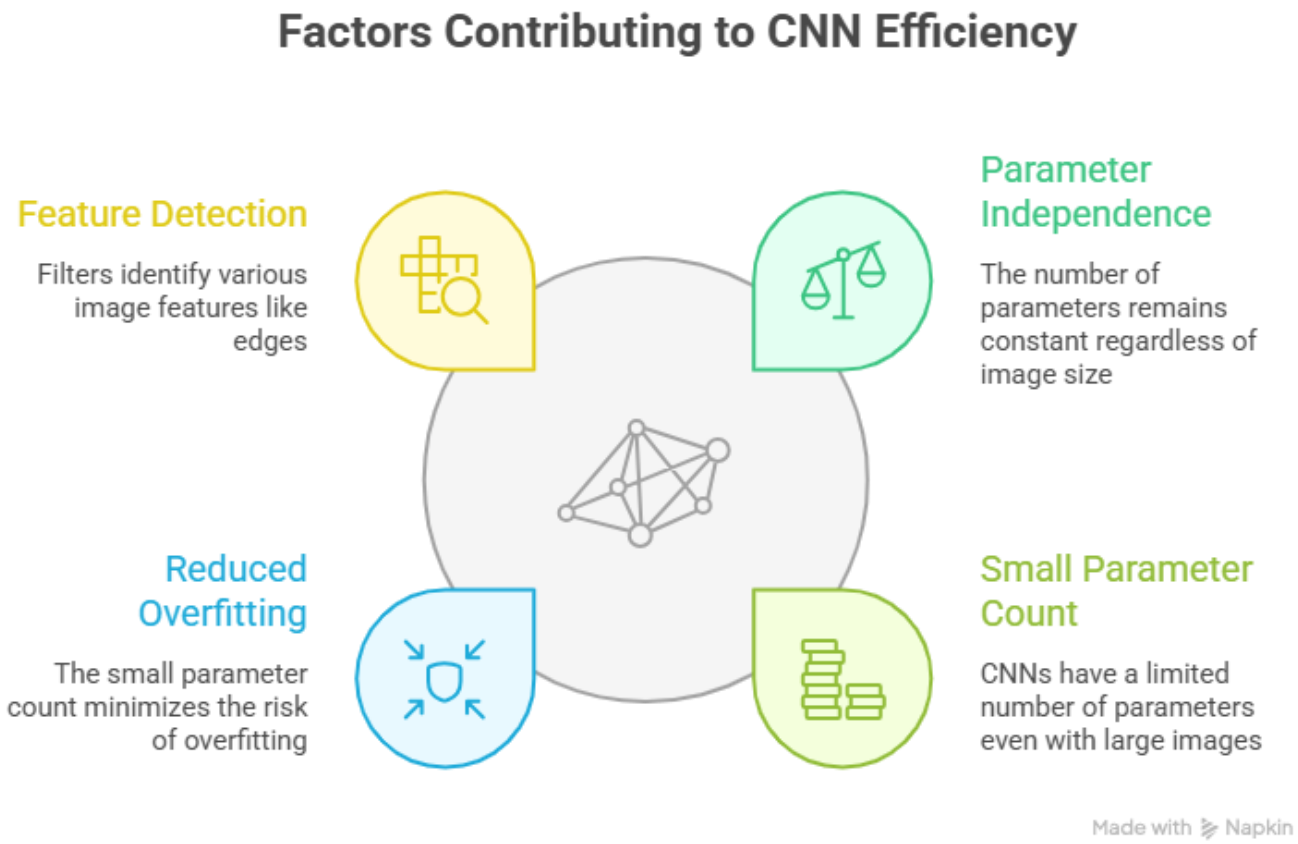
If you have **10 filters** that are ***3 × 3 × 3*** , in **one** layer of a CNN, how many parameters does that layer have?

Answer:

3 × 3 × 3 =27 + 1 (bias) = 28 x 10 (filters)

This implies a total of **280** parameters.

Number of parameters in one layer

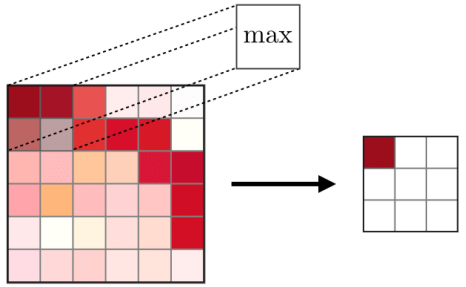
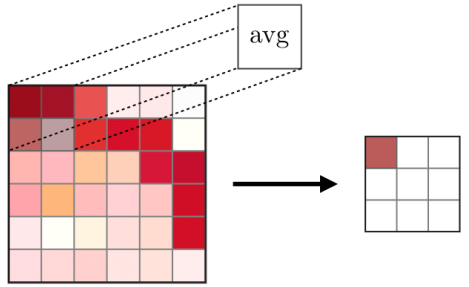


Pooling

The pooling layer is a downsampling operation, typically applied after a convolution layer.

Pooling operation reduces the size of the representation to speed up the computation and make the features more robust.

In particular, max and average pooling are pooling strategies where the maximum and average value is taken, respectively.

Type	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration		
Comments	• Preserves detected features • Most commonly used	• Downsamples feature map • Used in LeNet

Summary

The size of the regions (f) and the stride (s) are hyper-parameters.
Common choice f =2, s=2 has the effect of shrinking the height and width of the representation by a factor of 2.

There are no parameters to learn.

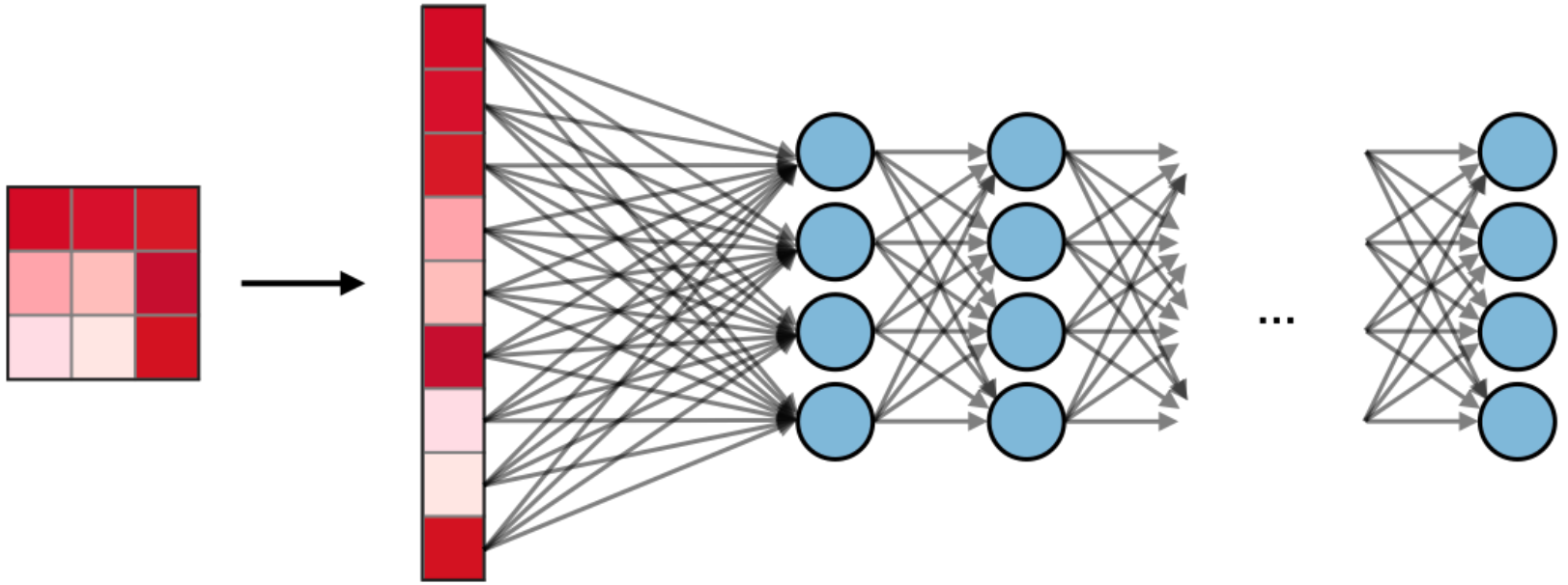
Max pooling is much often used than average pooling.

There is no theoretical proofs why pooling works well.
It is just a fact in practice that this approach works well with some data sets.

Fully Connected

The fully connected layer operates on a flattened input where each input is connected to all neurons.

If present, FC layers are usually found towards the end of CNN architectures.



Convolution vs Fully Connected

Why Convolutional Layers (vs Fully Connected)?
Key advantages: Parameter sharing, Sparse connections

Why does this matter?
Example

- Input: **32 × 32 × 3 RGB image** → 3072 inputs
- Using **6 filters (5 × 5 × 3)**
- Output: **28 × 28 × 6** → 4704 neurons

If Fully Connected...
To produce the same output:

- Each neuron connects to all inputs
- Parameters per neuron: **3072**

Total: **4704 × 3072 ≈ 14.4 million** → Inefficient for such structured data

Important Note:

- FC layers ignore spatial structure
- Conv layers explicitly exploit it

We are comparing **dense vs local connectivity**

Core Ideas
Parameter sharing

- Same filter applied across locations
- Detects the same feature anywhere

Sparse connections

- Each neuron sees only a **local region (receptive field)**

Inductive Bias (Critical Concept)
Convolutional layers assume:

- **Locality** → nearby pixels are related
- **Translation invariance** → features can appear anywhere → these are **assumptions about the data**

Why This Matters

- Fewer parameters are a consequence
- The real gain is: better generalization, faster learning, less data required

Quantitative Contrast

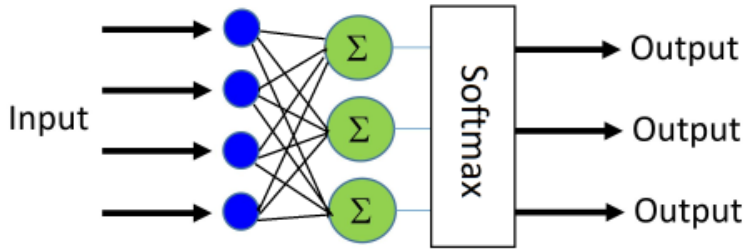
- Conv layer: **~456 parameters**
- Fully connected: **~14000000 parameters**

Takeaway
Convolutional layers are not just efficient; they encode *assumptions* about images.

Softmax

The softmax step can be seen as a generalized logistic function that:

- ✓ takes as input a vector of scores x∈R
- ✓ outputs a vector of probability p∈R through a softmax function at the end of the architecture.



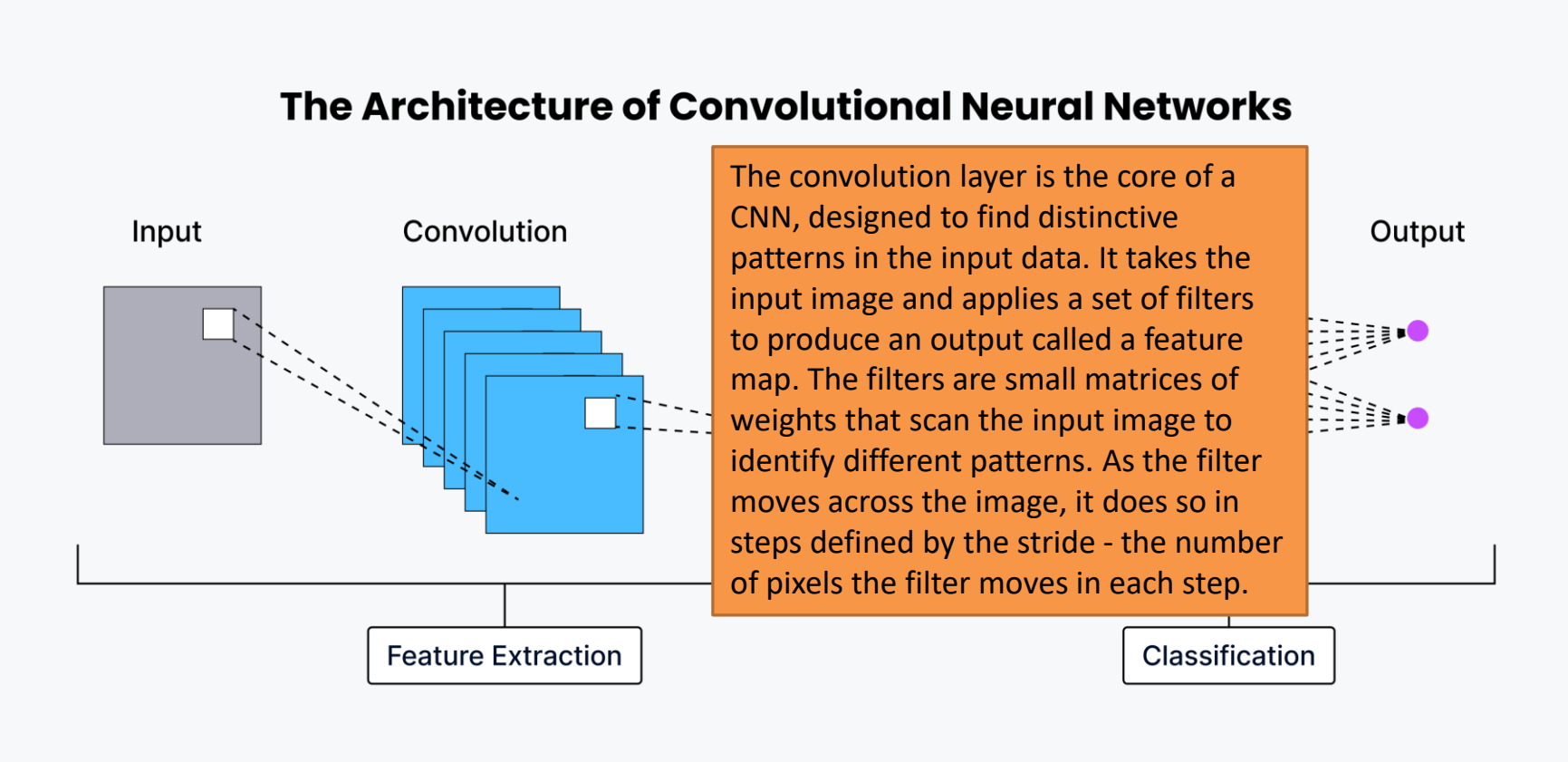
Softmax Layer (SL) estimates the probability that an example $x^{(i)}$ belongs to each of the k classes (j=1,2,..k).

$$p(y^{(i)} = j|x^{(i)};\theta) = \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}$$

SL outputs *k* dimensional vector with estimated probability for each class k:

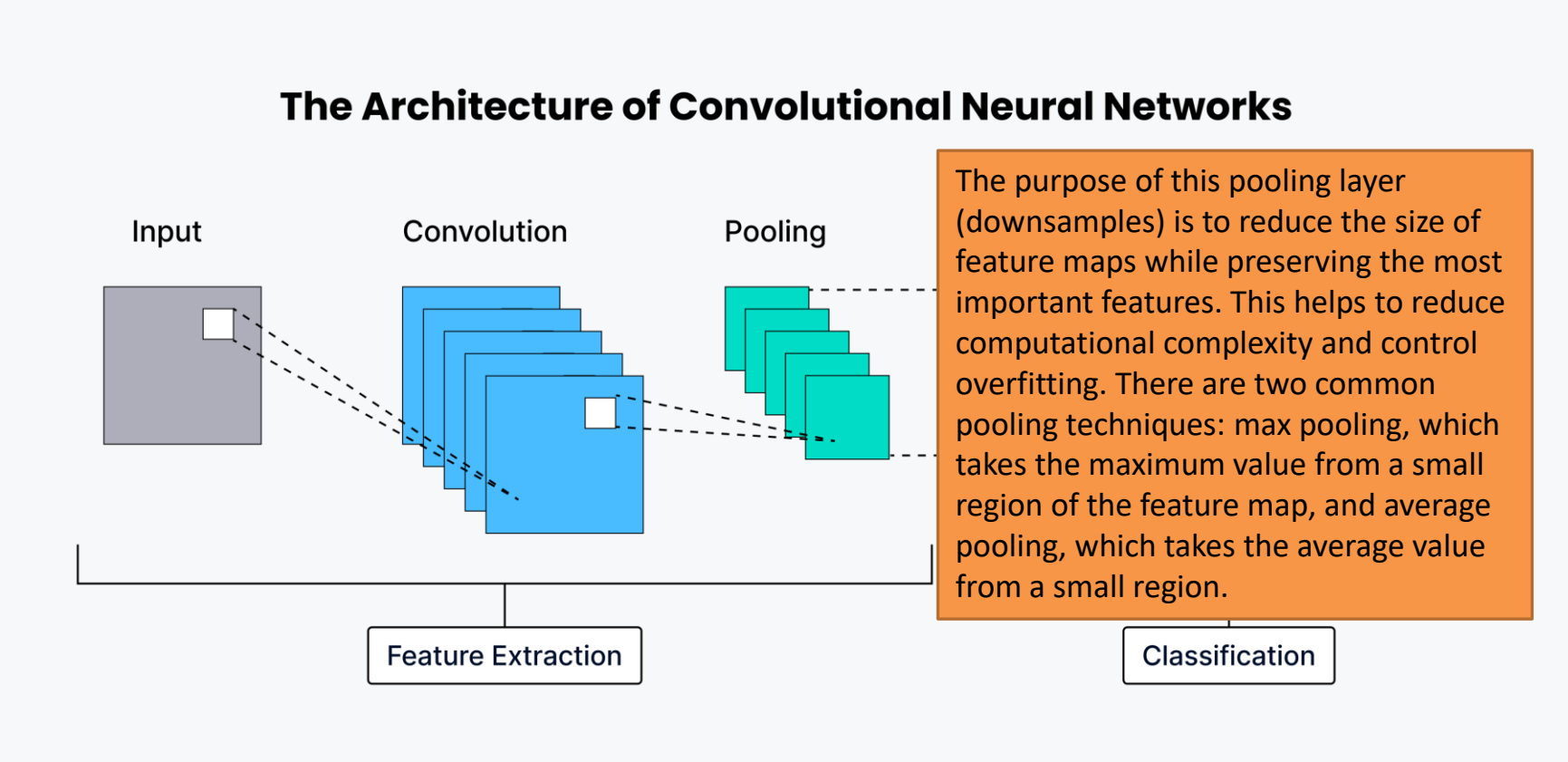
$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)};\theta) \\ p(y^{(i)} = 2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

Summary of CNN

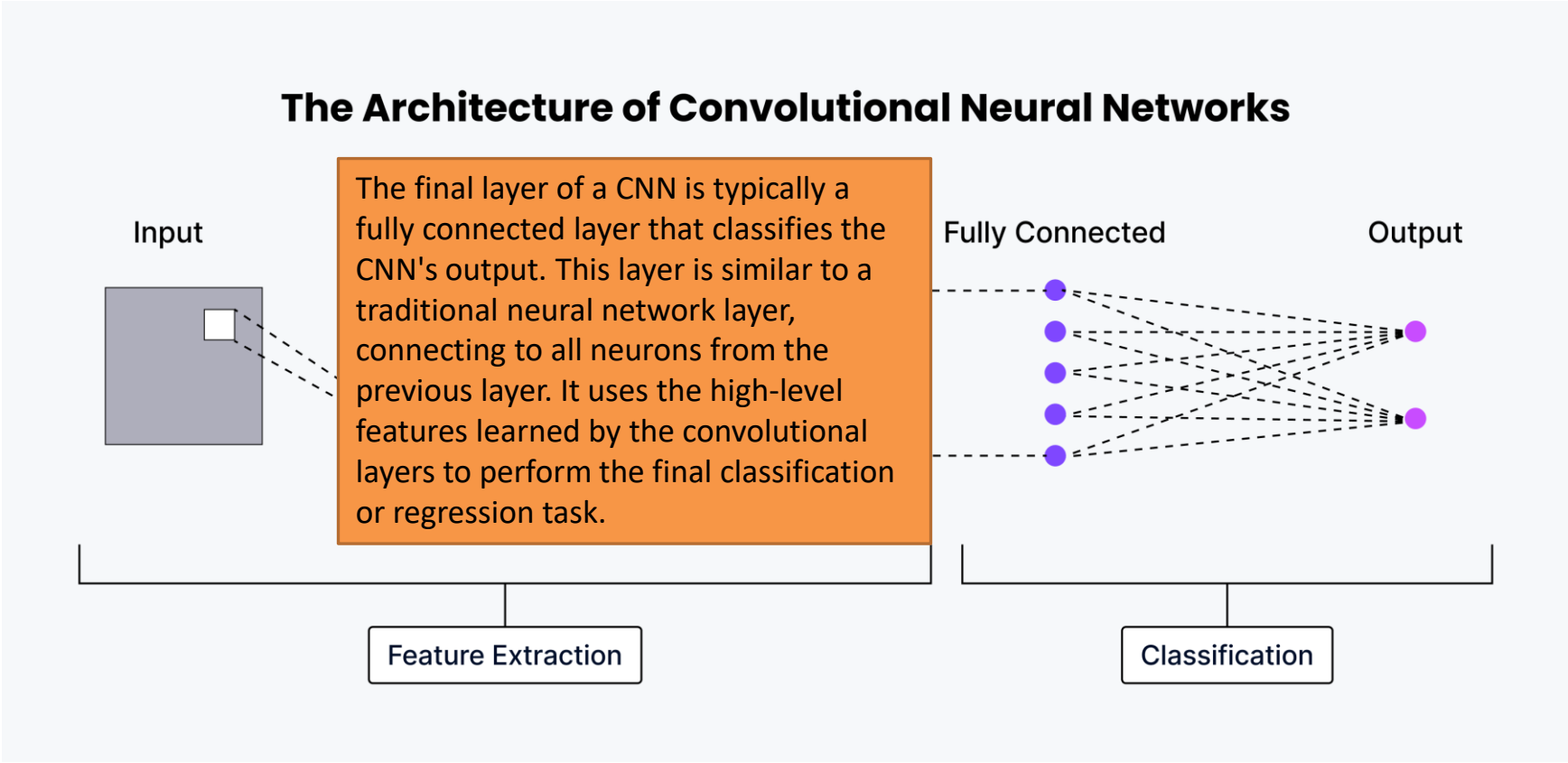


The convolution layer is the core of a CNN, designed to find distinctive patterns in the input data. It takes the input image and applies a set of filters to produce an output called a feature map. The filters are small matrices of weights that scan the input image to identify different patterns. As the filter moves across the image, it does so in steps defined by the stride - the number of pixels the filter moves in each step.

Summary of CNN



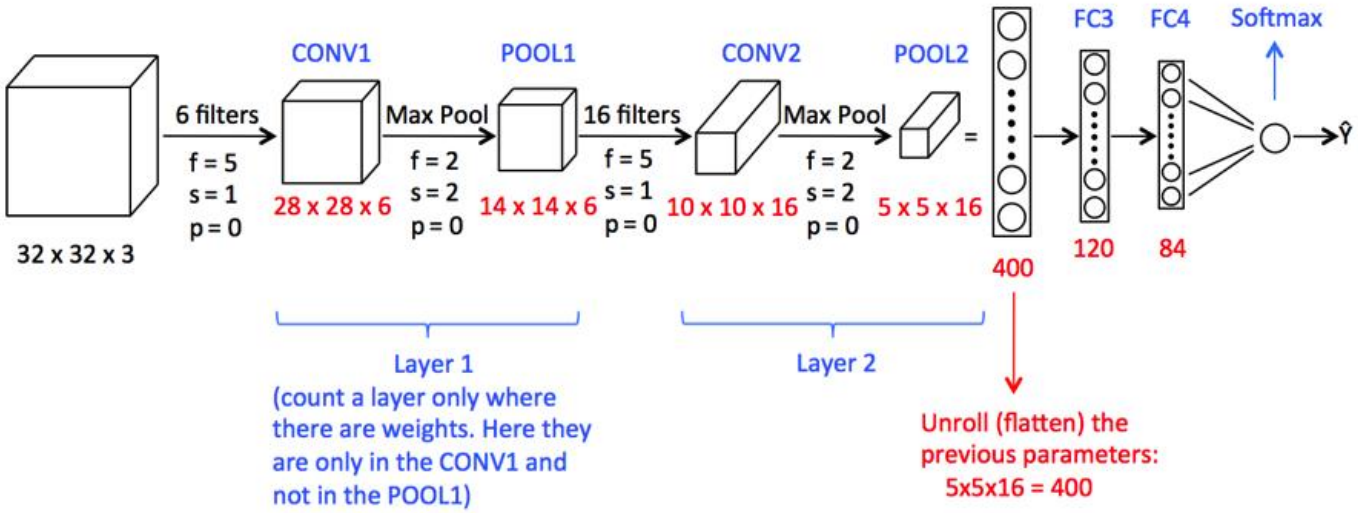
Summary of CNN



Classical CNN

LeNet-5, AlexNet, VGG

Classical CNN Example: LeNet5*



*LeCun et al., 1998, "Gradient-based learning applied to document recognition".

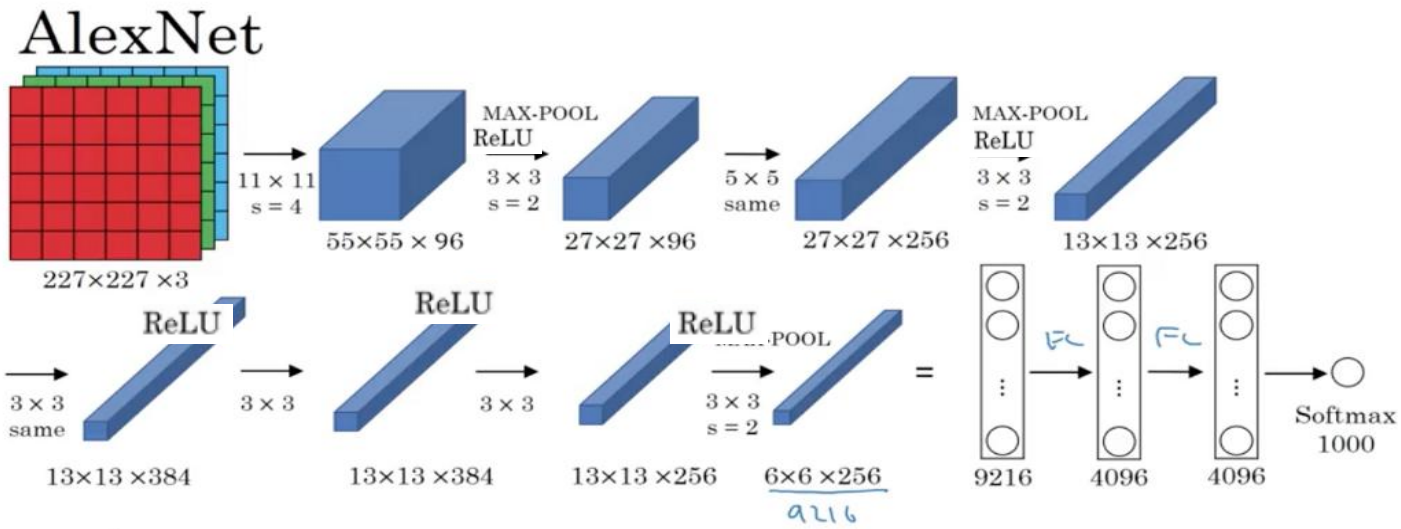
Original LeNet5:

- ✓ applied to handwritten digit recognition (grey scale images)
- ✓ Avg pooling, no padding, softmax classifier
- ✓ ReLU and sigmoid/tanh neurons in the Fully Connected (FC) layers.

NOTES:
The activation function is always present after the convolution, even if it is not drawn on the CNN diagram.

General trend: CNNs start with large image, then height and width gradually decrease as it goes deeper in the network, where as the number of channels increase.

AlexNet*

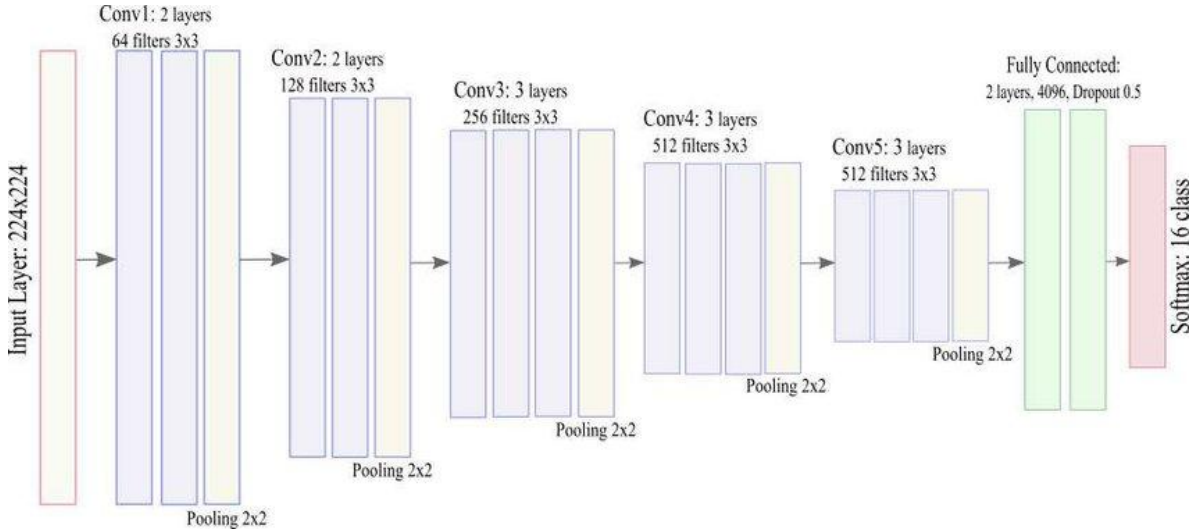


* Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, 2012, ImageNet classification with deep convolutional neural networks.

ImageNet:

- ✓ 5 conv layers, 3 FC layers with softmax output
- ✓ 60 million parameters in total.

VGG-16*



* Karen Simonyan, Andrew Zisserman, 2015, Very Deep Convolutional Networks for Large-Scale Image Recognition

VGG-16:

- ✓ 16 layers
- ✓ 138 million parameters.
- ✓ Unified architecture:
  - ✓ All conv layers: (3x3) filters, s=1, same;
  - ✓ All MaxPool =2x2, s=2.

VGG-19 is a larger version of VGG-16, both have similar performance.

Transfer Learning

What is Transfer Learning (and why we use it)

Transfer Learning: Use knowledge from a **pre-trained model** to solve a new task

Why do we need it?
Training deep networks from scratch:

- Requires **large datasets**
- Requires **high computational cost** (weeks, GPUs), which is often impractical

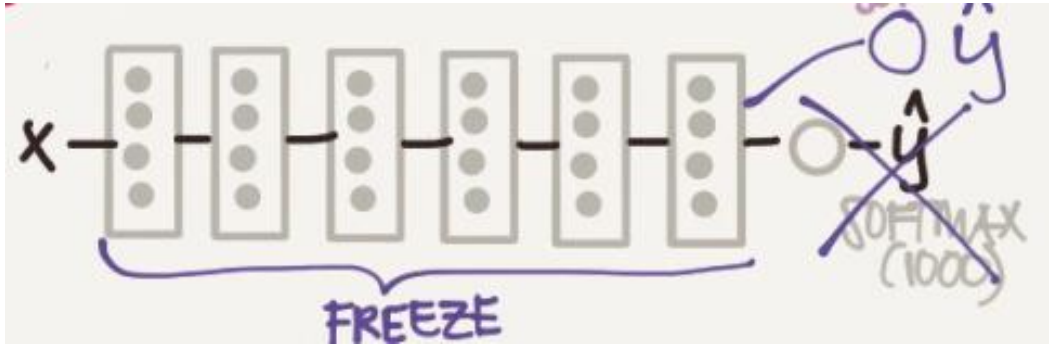
Core Idea: Instead of training from scratch:

- Start from a model trained on a **large dataset** (e.g., ImageNet)
- Reuse its **learned representations**

Key Intuition
Early layers learn: **Generic features** (edges, textures, shapes)
Later layers learn: **Task-specific features**
So, reuse general features, adapt the rest

Takeaway:
Transfer Learning

- Reduces data requirements
- Speeds up training
- Improves performance on small datasets



How Transfer Learning Works

Typical Workflow Example

- Task: 3 classes (car, pedestrian, neither)
- Start from model trained on 1000 classes

Step 1 — Replace Output Layer

- Remove original softmax (1000 classes)
- Add new softmax (3 classes)

Step 2 — Freeze Layers

Train only the final layer:

- Earlier layers → **frozen**
- Last layer → **trained**

Works well for **small datasets**

Step 3 — Fine-tuning

If more data is available:

- Unfreeze some deeper layers
- Train part of the network

More flexibility, higher risk of overfitting

Step 4 — Full Fine-tuning

With large datasets:

- Initialize with pre-trained weights
- Train **all layers**

Better than random initialization

Engineering Trick (Optional)

- Precompute features from frozen layers
- Train only the classifier

Faster, but less flexible

Rule of Thumb

- Small dataset → freeze most layers
- Large dataset → fine-tune more layers

When Transfer Learning Fails

1. Domain Mismatch
Pre-trained model learned from: Natural images (e.g., ImageNet)
New task: Medical images, satellite data, etc. In these cases, features may not transfer
Problem: Early layers are no longer “generic enough”.

2. Task Mismatch
Original task: Object classification
New task: segmentation, detection, regression
Here, the representation may not be appropriate

3. Negative Transfer
Transfer learning can **hurt performance** if the model is biased toward irrelevant patterns.
Example: Texture-based features reused in shape-based task

4. Too Small Dataset
Final layer may still overfit
Fine-tuning becomes unstable

5. Over-Freezing
Freezing too many layers, the model cannot adapt, producing an underfitting result.

6. Over-Fine-Tuning
Unfreezing too much, may destroy useful pre-trained features, which may produce overfitting or catastrophic forgetting.

Underlying Issue
All failures stem from: **Mismatch between source and target distributions**

Takeaway
Transfer learning works when:

- Features are **relevant**
- Data distributions are **similar enough**

Underlying Principle

Why TL Can Fail: $P_{source}(x) \approx P_{target}(x)$

Transfer learning assumes, that the source and target distributions are **similar enough**

If This Assumption Breaks

- Learned features are no longer relevant
- Model transfers **biases instead of knowledge**

This leads to **negative transfer**

Interpretation

- TL is not about reusing a model
- It is about reusing **useful representations**

Takeaway

Transfer learning works when:

- Features generalise across domains
- Features are **relevant**
- Data distributions are **similar enough**

what convolutional layers learn

What are learning Conv Layers *

Suppose you scan through your entire training set and the 9 image patches that maximize that unit's activation are identified.

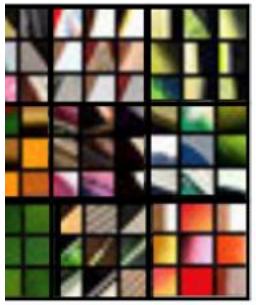
This particular hidden unit seems to be activated (to see) by edges or lines.

Repeat for other units in Layer 1, produce 9 different representative neurons, and for each of them, the 9 image patches that maximally activate them.

The trained hidden units in Layer 1 respond to relatively simple features such as edges or a particular shade of colour.

* ref. Zeiler&Fergus, 2013 Visualizing and understanding convolutional networks.

Visualizing deep layers: Layer 2



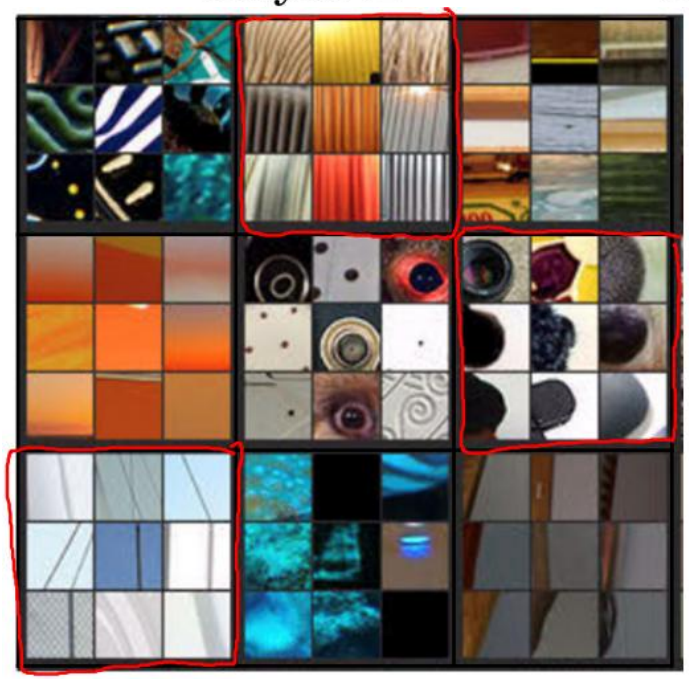
Layer 1

Layer 2

Layer 3

Layer 4

Layer 5

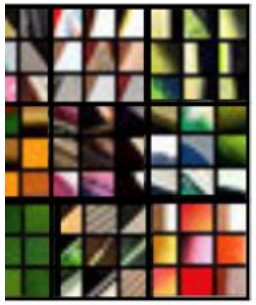


<u>In deeper layers, the hidden units see larger regions of the image.</u>

The units in Layer 2 are activated from more complex shapes and patterns, such as vertical lines with texture, rounder shapes at the left part of the image, and so on.

The features that Layer 2 is detecting are getting more detailed.

Visualizing deep layers: Layer 3


Layer 1


Layer 2

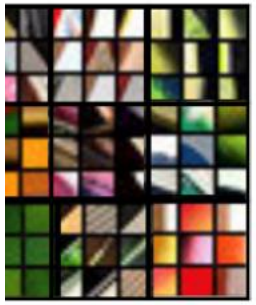

Layer 3


Layer 4


Layer 5

In Layer 3, the hidden units start detecting part of cars, irregular texture, even people, other shapes difficult to figure out what they are, but it is clearly starting to detect more complex patterns.

Visualizing deep layers: Layer 4


Layer 1

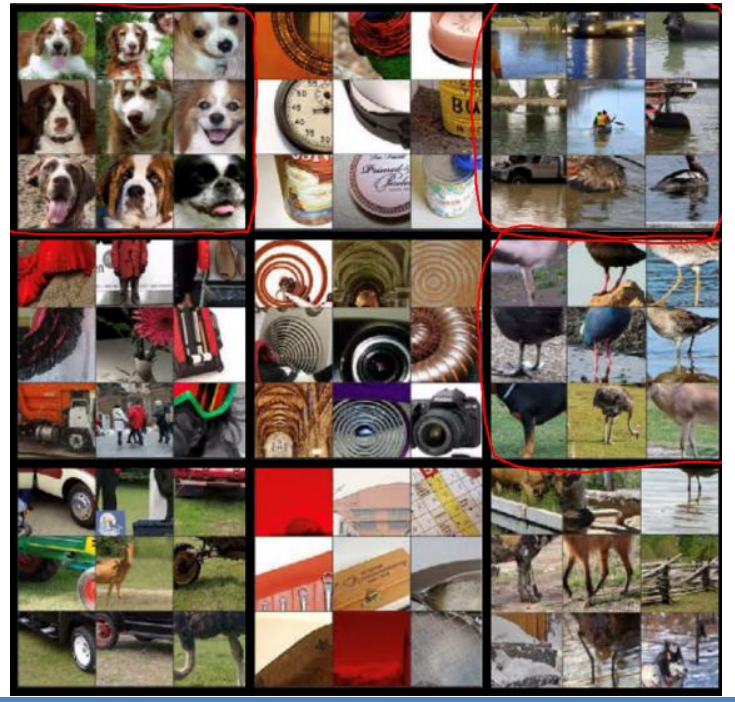

Layer 2


Layer 3

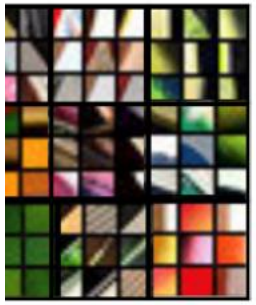

Layer 4


Layer 5

In Layer4, one unit seems to be a dog detector, but all dogs are somehow similar, other unit detects water, other unit detects legs of birds and so on. Detected patterns are even more complex then in Layer3.



Visualizing deep layers: Layer 5


Layer 1


Layer 2

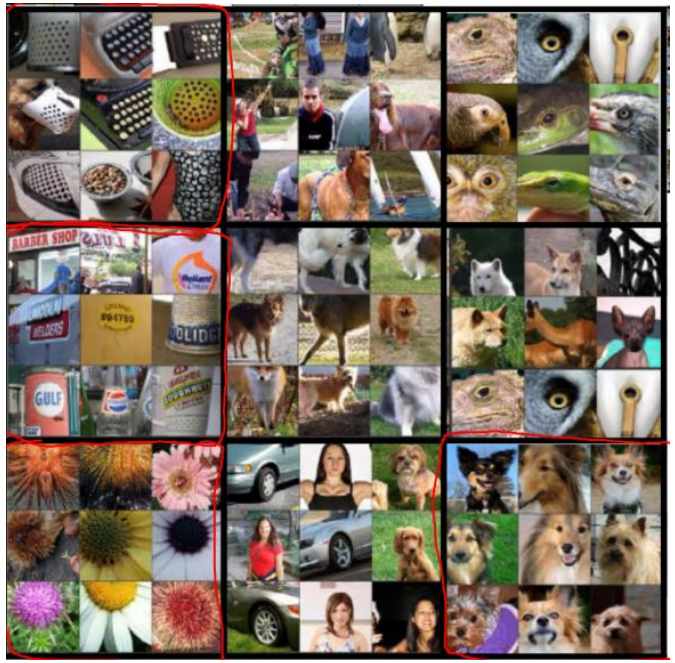

Layer 3


Layer 4


Layer 5

In Layer 5:

- one neuron seems to be a dog detector, but the detected dogs seem to be more varied;
- other detects flowers;
- other neuron seems to detect keyboard-like texture, or maybe lots of dots against the background;
- one neuron may detect text; it's always hard to be sure.



Advantages of CNNs

1.Good at detecting patterns and features in images, videos, and audio signals.

2.Robust to translation, rotation, and scaling invariance.

3.End-to-end training, no need for manual feature extraction.

4.Can handle large amounts of data and achieve high accuracy.

Disadvantages of CNNs

1.Computationally expensive to train and require a lot of memory.

2.Can be prone to overfitting if not enough data or proper regularization is used.

3.Requires large amounts of labeled data.

4.Interpretability is limited, it’s hard to understand what the network has learned.