An Introduction to Convolutional Neural Networks (CNNs)

A complete guide to understanding CNNs, their impact on image analysis, and some key strategies to combat overfitting for robust CNN vs deep learning applications.

Nov 14, 2023 · 14 min read

CONTENTS

What is a Convolutional Neural Network (CNN)?

A Convolutional Neural Network (CNN), also known as ConvNet, is a specialized type of deep learning algorithm mainly designed for tasks that necessitate object recognition, including image classification, detection, and segmentation. CNNs are employed in a variety of practical scenarios, such as autonomous vehicles, security camera systems, and others.

The importance of CNNs

There are several reasons why CNNs are important in the modern world, as highlighted below:

- CNNs are distinguished from classic machine learning algorithms such as **SVMs** and **decision trees** by their ability to autonomously extract features at a large scale, bypassing the need for manual feature engineering and thereby enhancing efficiency.

- The convolutional layers grant CNNs their translation-invariant characteristics, empowering them to identify and extract patterns and features from data irrespective of variations in position, orientation, scale, or translation.

- A variety of pre-trained CNN architectures, including VGG-16, ResNet50, Inceptionv3, and EfficientNet, have demonstrated top-tier performance. These models can be adapted to new tasks with relatively little data through a process known as fine-tuning.
- Beyond image classification tasks, CNNs are versatile and can be applied to a range of other domains, such as natural language processing, time series analysis, and speech recognition.

Inspiration Behind CNN and Parallels With The Human Visual System

Convolutional neural networks were inspired by the layered architecture of the human visual cortex, and below are some key similarities and differences:

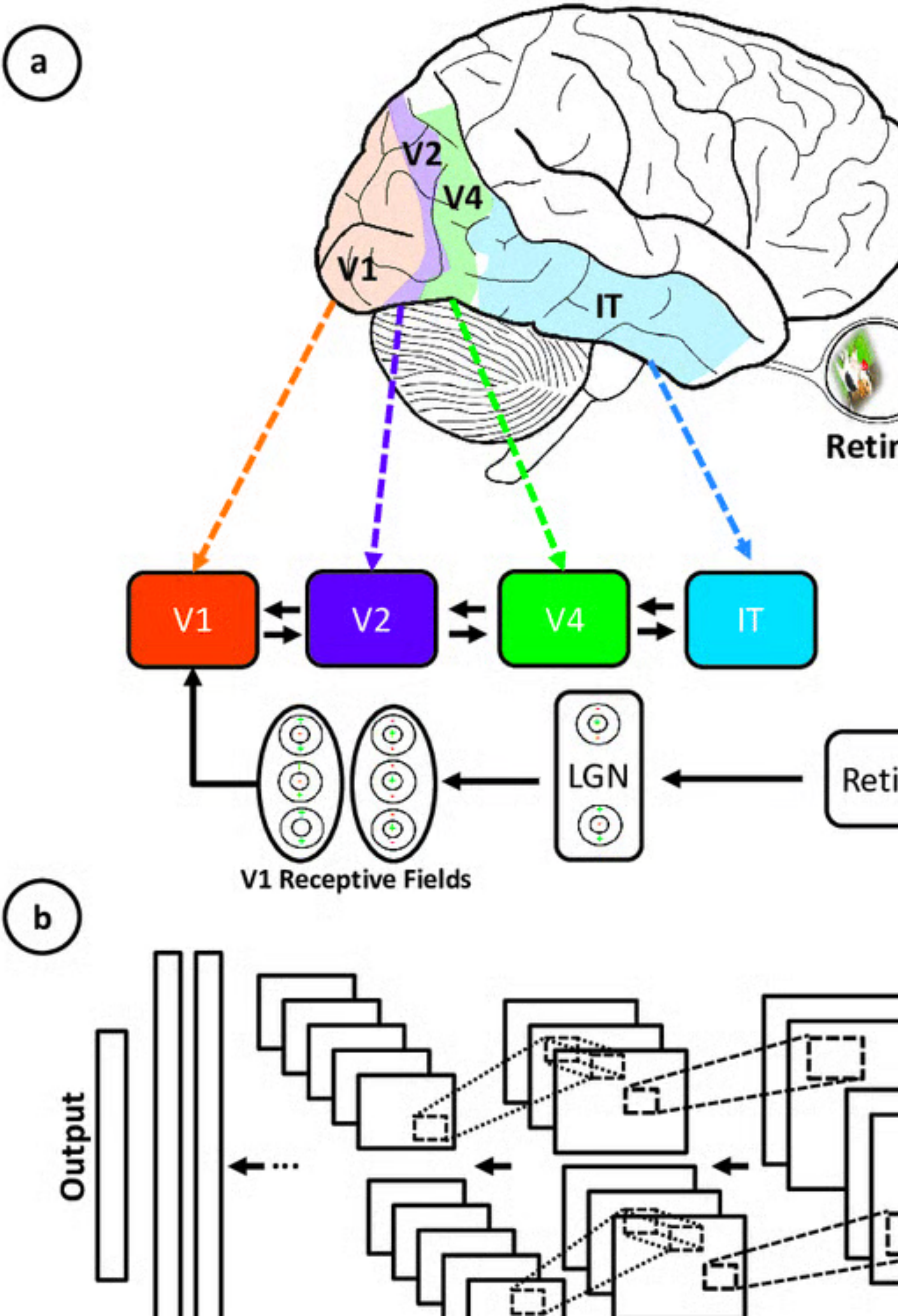

a
V2
V4
V1
IT
Retin
V1
V2
V4
IT
LGN
Reti
V1 Receptive Fields
b
Output

Illustration of the correspondence between the areas associated with the primary visual cortex and the layers in a convolutional neural network (source)

- **Hierarchical architecture:** Both CNNs and the visual cortex have a hierarchical structure, with simple features extracted in early layers and more complex features built up in deeper layers. This allows increasingly sophisticated representations of visual inputs.
- **Local connectivity:** Neurons in the visual cortex only connect to a local region of the input, not the entire visual field. Similarly, the neurons in a CNN layer are only connected to a local region of the input volume through the convolution operation. This local connectivity enables efficiency.
- **Translation invariance:** Visual cortex neurons can detect features regardless of their location in the visual field. Pooling layers in a CNN provide a degree of translation invariance by summarizing local features.
- **Multiple feature maps:** At each stage of visual processing, there are many different feature maps extracted. CNNs mimic this through multiple filter maps in each convolution layer.
- **Non-linearity:** Neurons in the visual cortex exhibit non-linear response properties. CNNs achieve non-linearity through activation functions like ReLU applied after each convolution.

CNNs mimic the human visual system but are simpler, lacking its complex feedback mechanisms and relying on supervised learning rather than unsupervised, driving advances in computer vision despite these differences.

Key Components of a CNN

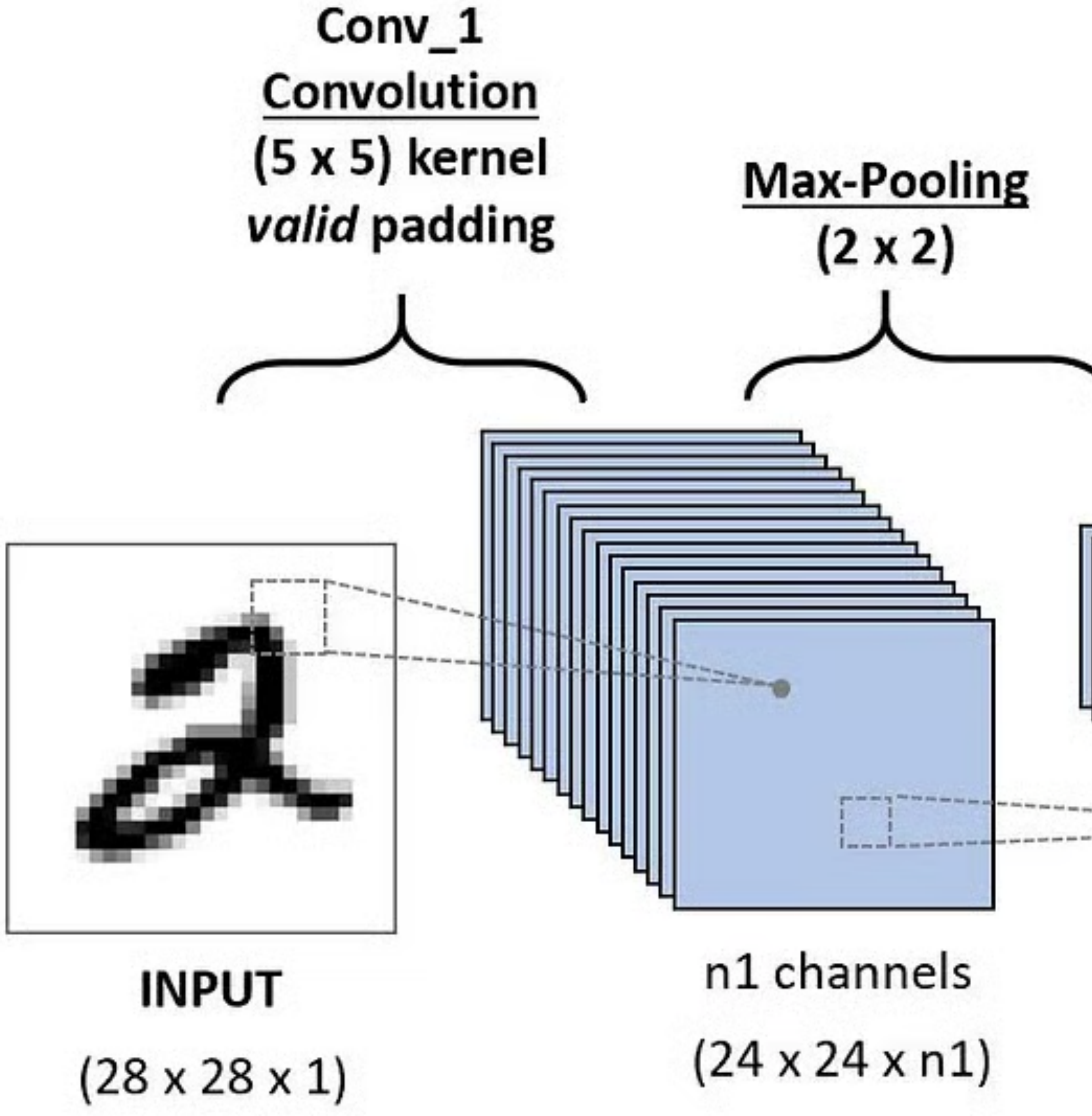
The convolutional neural network is made of four main parts.

But how do CNNs Learn with those parts?

They help the CNNs mimic how the human brain operates to recognize patterns and features in images:

- Convolutional layers
- Rectified Linear Unit (ReLU for short)
- Pooling layers
- Fully connected layers

This section dives into the definition of each one of these components through the example of the following example of classification of a handwritten digit.


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)

Architecture of the CNNs applied to digit recognition (source)

Convolution layers

This is the first building block of a CNN. As the name suggests, the main mathematical task performed is called convolution, which is the application of a sliding window function to a matrix of pixels representing an image. The sliding function applied to the matrix is called kernel or filter, and both can be used interchangeably.

In the convolution layer, several filters of equal size are applied, and each filter is used to recognize a specific pattern from the image, such as the curving of the digits, the edges, the whole shape of the digits, and more.

Put simply, in the convolution layer, we use small grids (called filters or kernels) that move over the image. Each small grid is like a mini magnifying glass that looks for specific patterns in the photo, like lines, curves, or shapes. As it moves across the photo, it creates a new grid that highlights where it found these patterns.

For example, one filter might be good at finding straight lines, another might find curves, and so on. By using several different filters, the CNN can get a good idea of all the different patterns that make up the image.

Let’s consider this 32x32 grayscale image of a handwritten digit. The values in the matrix are given for illustration purposes.

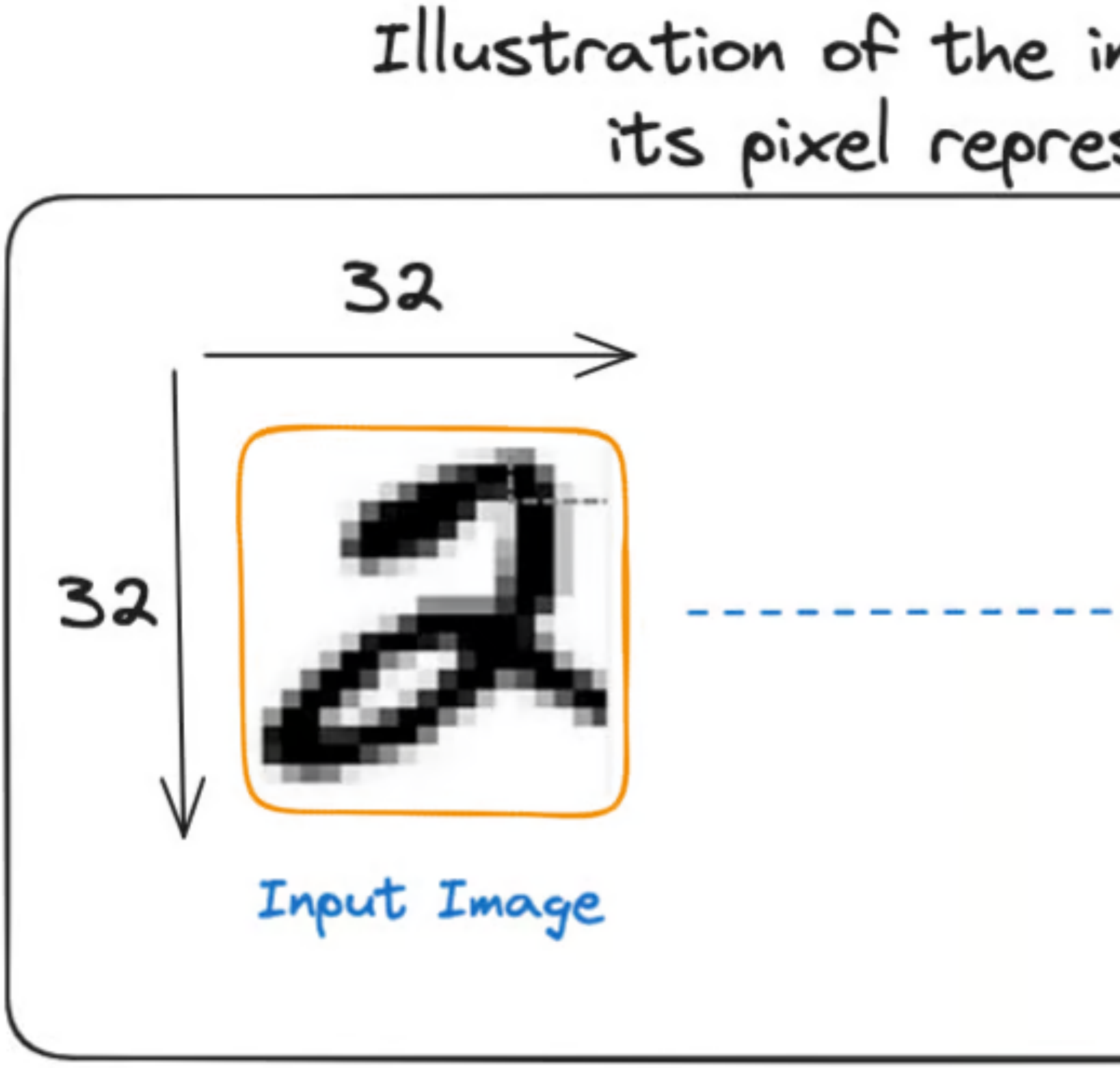


Illustration of the input image and its pixel representation

Also, let's consider the kernel used for the convolution. It is a matrix with a dimension of 3x3. The weights of each element of the kernel is represented in the grid. Zero weights are represented in the black grids and ones in the white grid.

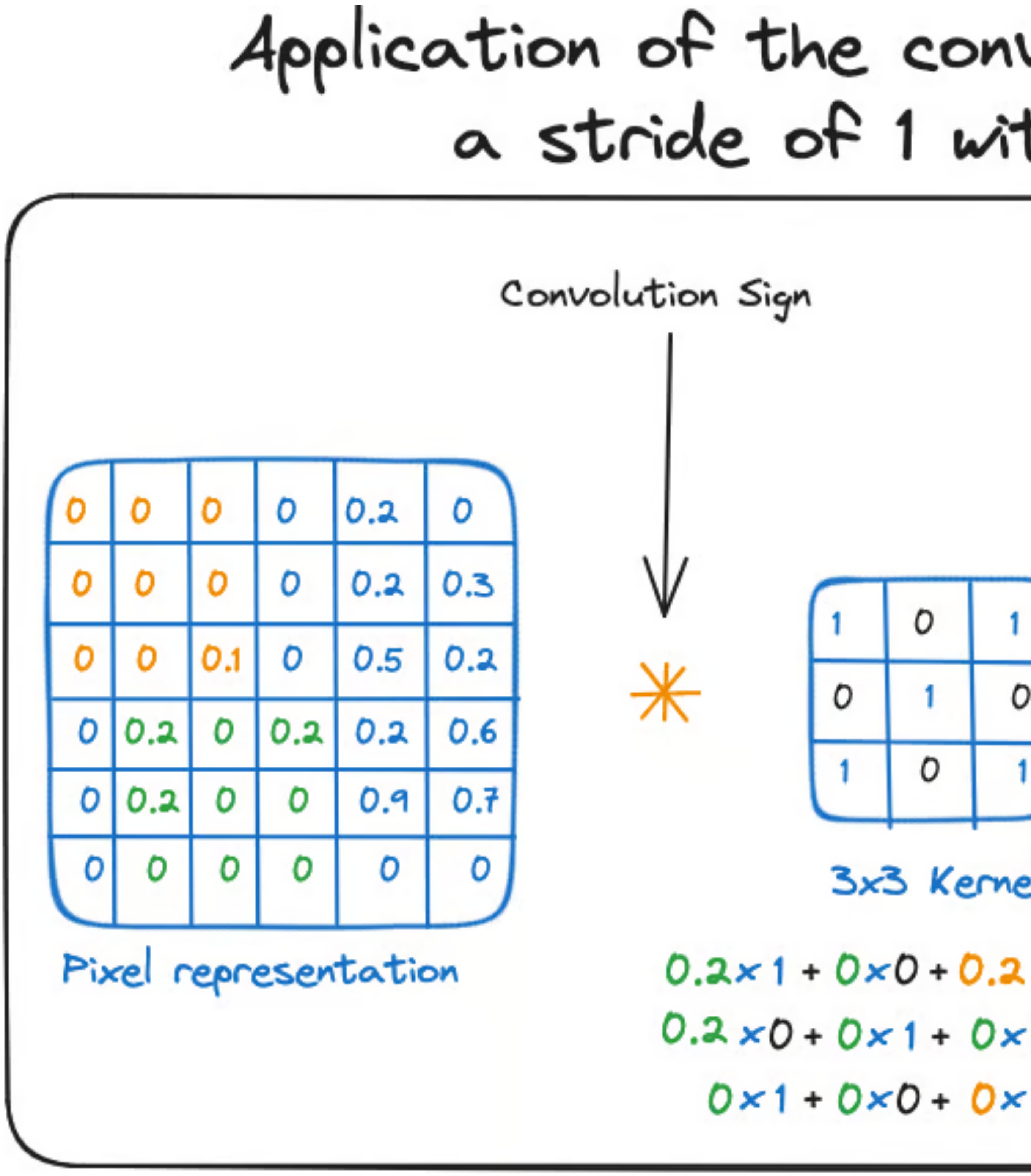
Do we have to manually find these weights?

In real life, the weights of the kernels are determined during the training process of the neural network.

Using these two matrices, we can perform the convolution operation by applying the dot product, and work as follows:

1. Apply the kernel matrix from the top-left corner to the right.
2. Perform element-wise multiplication.
3. Sum the values of the products.
4. The resulting value corresponds to the first value (top-left corner) in the convoluted matrix.
5. Move the kernel down with respect to the size of the sliding window.
6. Repeat steps 1 to 5 until the image matrix is fully covered.

The dimension of the convoluted matrix depends on the size of the sliding window. The higher the sliding window, the smaller the dimension.



0	0	0	0	0.2	0
0	0	0	0	0.2	0.3
0	0	0.1	0	0.5	0.2
0	0.2	0	0.2	0.2	0.6
0	0.2	0	0	0.9	0.7
0	0	0	0	0	0

1	0	1
0	1	0
1	0	1

Application of the convolution task using a stride of 1 with 3x3 kernel

Another name associated with the kernel in the literature is feature detector because the weights can be fine-tuned to detect specific features in the input image.

For instance:

- Averaging neighboring pixels kernel can be used to blur the input image.
- Subtracting neighboring kernel is used to perform edge detection.

The more convolution layers the network has, the better the layer is at detecting more abstract features.

Activation function

A ReLU activation function is applied after each convolution operation. This function helps the network learn non-linear relationships between the features in the image, hence making the network more robust for identifying different patterns. It also helps to mitigate the vanishing gradient problems.

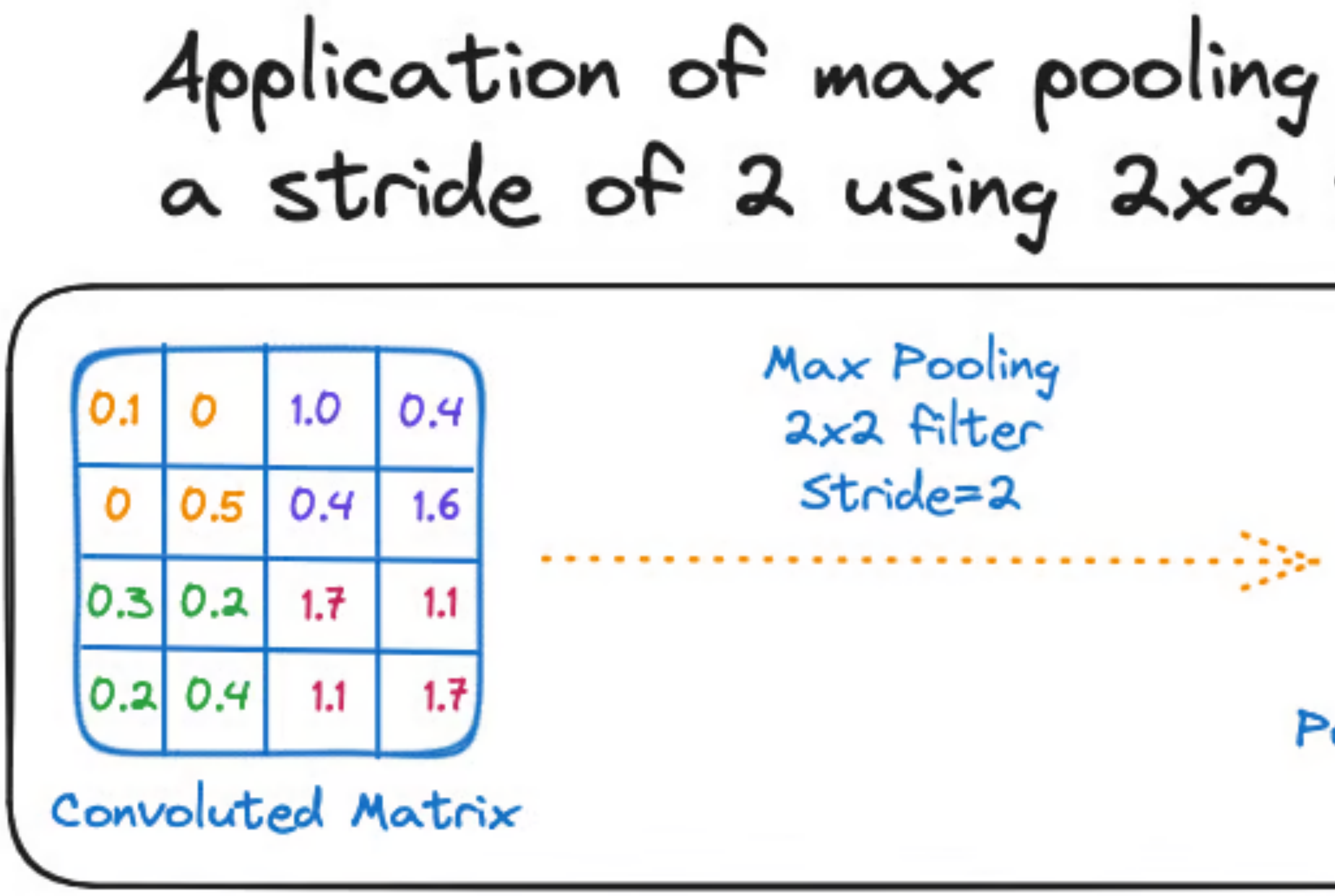
Pooling layer

The goal of the pooling layer is to pull the most significant features from the convoluted matrix. This is done by applying some aggregation operations, which reduce the dimension of the feature map (convoluted matrix), hence reducing the memory used while training the network. Pooling is also relevant for mitigating overfitting.

The most common aggregation functions that can be applied are:

- Max pooling, which is the maximum value of the feature map
- Sum pooling corresponds to the sum of all the values of the feature map
- Average pooling is the average of all the values.

Below is an illustration of each of the previous example:



0.1	0	1.0	0.4
0	0.5	0.4	1.6
0.3	0.2	1.7	1.1
0.2	0.4	1.1	1.7

Application of max pooling with a stride of 2 using 2x2 filter

Also, the dimension of the feature map becomes smaller as the pooling function is applied.

The last pooling layer flattens its feature map so that it can be processed by the fully connected layer.

Fully connected layers

These layers are in the last layer of the convolutional neural network, and their inputs correspond to the flattened one-dimensional matrix generated by the last pooling layer. ReLU activations functions are applied to them for non-linearity.

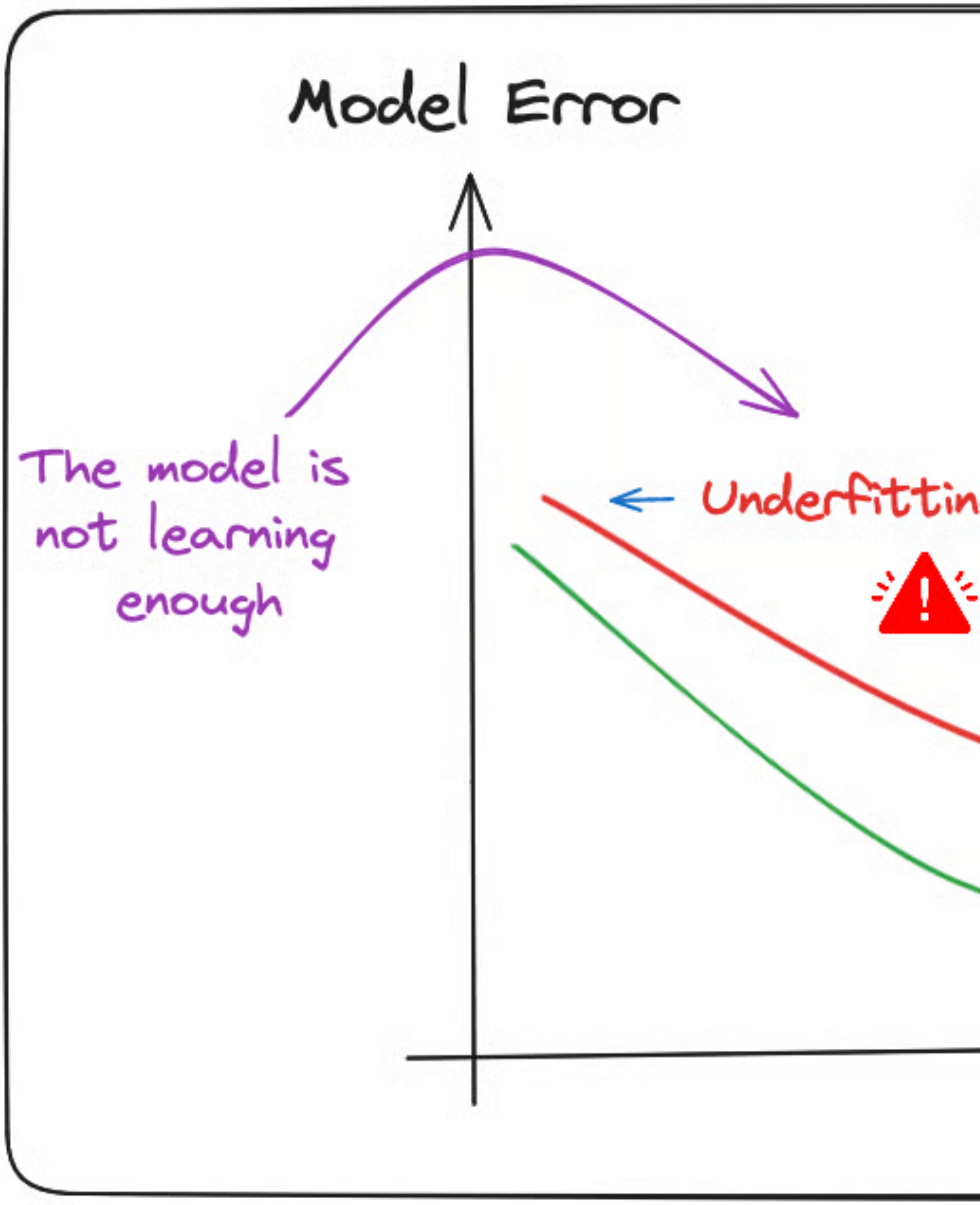
Finally, a softmax prediction layer is used to generate probability values for each of the possible output labels, and the final label predicted is the one with the highest probability score.

Overfitting and Regularization in CNNs

Overfitting is a common challenge in machine learning models and CNN deep learning projects. It happens when the model learns the training data too well (“learning by heart”), including its noise and outliers. Such a learning leads to a model that performs well on the training data but badly on new, unseen data.

This can be observed when the performance on training data is too low compared to the performance on validation or testing data, and a graphical illustration is given below:

Underfittin

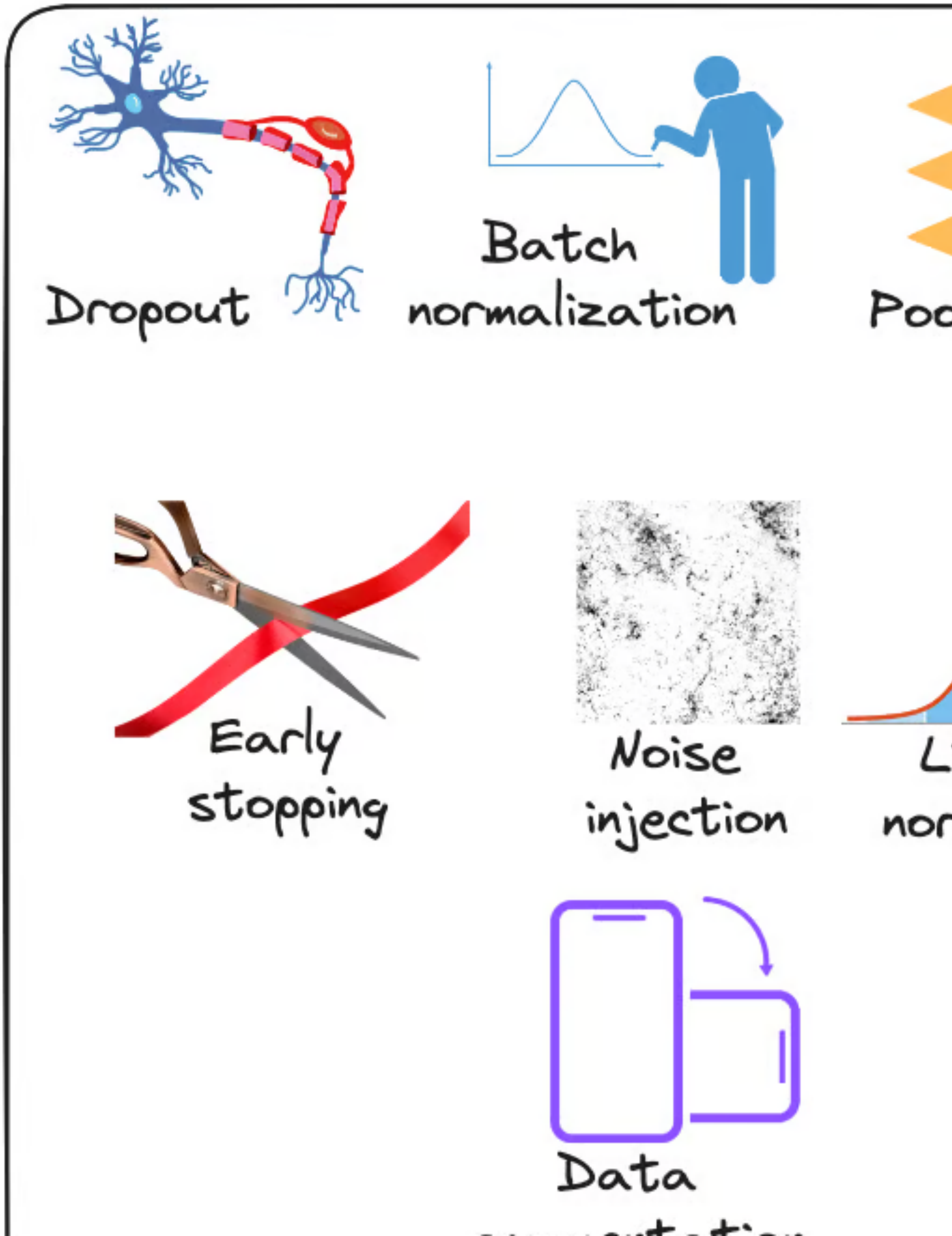


Underfitting Vs. Overfitting

Deep learning models, especially Convolutional Neural Networks (CNNs), are particularly susceptible to overfitting due to their capacity for high complexity and their ability to learn detailed patterns in large-scale data.

Several regularization techniques can be applied to mitigate overfitting in CNNs, and some are illustrated below:

7 Strategies to Mitiga
Overfitting in CNNs



Dropout

Batch normalization

Poo

Early stopping

Noise injection

L
nor

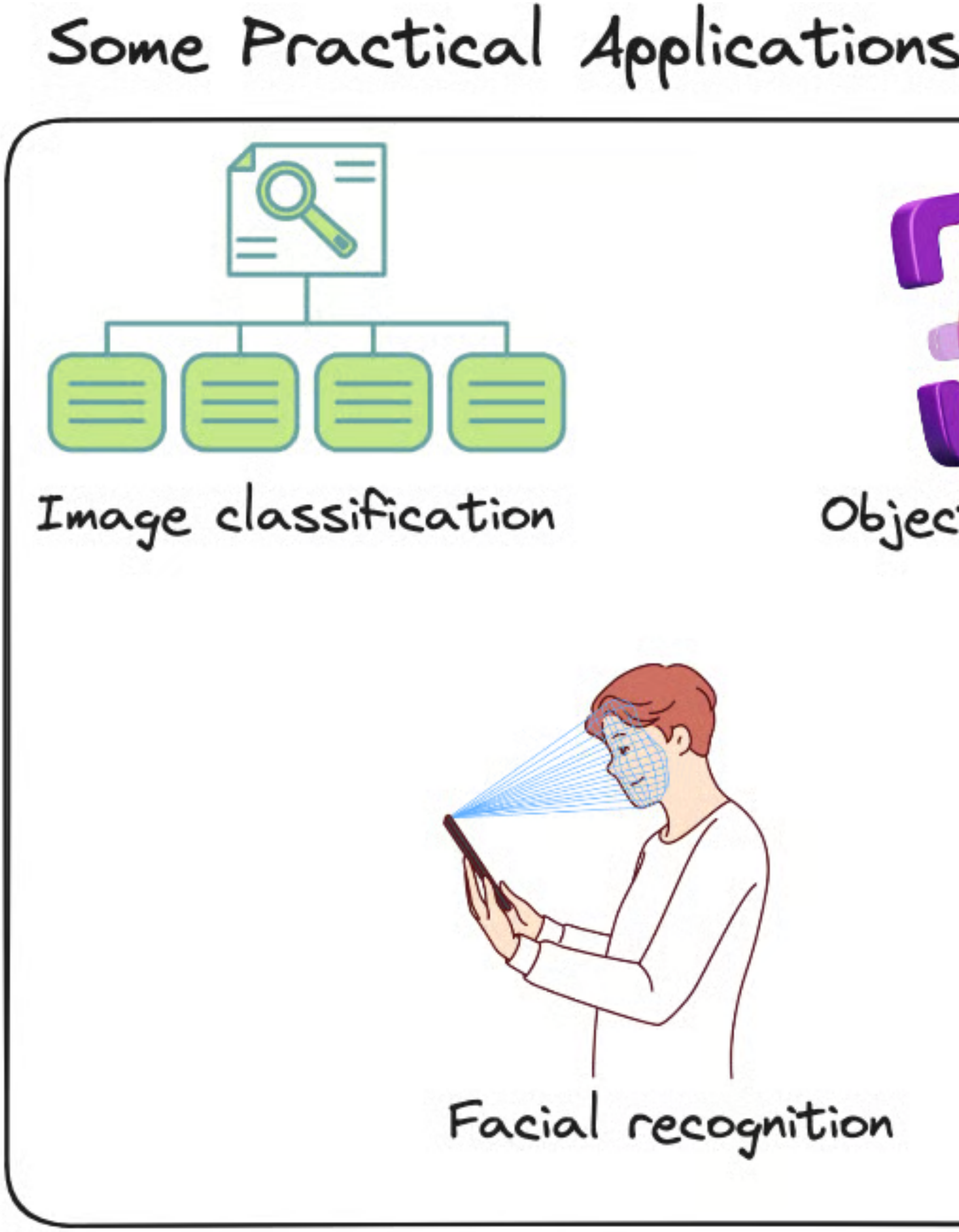
Data augmentation

7 strategies to mitigate overfitting in CNNs

- **Dropout:** This consists of randomly dropping some neurons during the training process, which forces the remaining neurons to learn new features from the input data.
- **Batch normalization:** The overfitting is reduced at some extent by normalizing the input layer by adjusting and scaling the activations. This approach is also used to speed up and stabilize the training process.
- **Pooling Layers:** This can be used to reduce the spatial dimensions of the input image to provide the model with an abstracted form of representation, hence reducing the chance of overfitting.
- **Early stopping:** This consists of consistently monitoring the model's performance on validation data during the training process and stopping the training whenever the validation error does not improve anymore.
- **Noise injection:** This process consists of adding noise to the inputs or the outputs of hidden layers during the training to make the model more robust and prevent it from a weak generalization.
- **L1 and L2 normalizations:** Both L1 and L2 are used to add a penalty to the loss function based on the size of weights. More specifically, L1 encourages the weights to be spare, leading to better feature selection. On the other hand, L2 (also called weight decay) encourages the weights to be small, preventing them from having too much influence on the predictions.
- **Data augmentation:** This is the process of artificially increasing the size and diversity of the training dataset by applying random transformations like rotation, scaling, flipping, or cropping to the input images.

Practical Applications of CNNs

Convolutional Neural Networks have revolutionized the field of computer vision, leading to significant advancements in many real-world applications. Below are a few examples of how they are applied.



Some practical applications of CNNs

- **Image classification:** Convolutional neural networks are used for image categorization, where images are assigned to predefined categories. One use of such a scenario is automatic photo organization in social media platforms.
- **Object detection:** CNNs are able to identify and locate multiple objects within an image. This capability is crucial in multiple scenarios of shelf scanning in retail to identify out-of-stock items.
- **Facial recognition:** this is also one of the main industries of application of CNNs. For instance, this technology can be embedded into security systems for efficient control of access based on facial features.

For a more hands-on implementation, our **Convolutional Neural Networks (CNN) with TensorFlow Tutorial** teaches how to construct and implement CNNs in Python with Tensorflow Framework 2.

Deep Learning Frameworks for CNNs

The rapid growth of deep learning is mainly due to powerful frameworks like Tensorflow, Pytorch, and Keras, which make it easier to train convolutional neural networks and other deep learning models.

Let’s have a brief overview of each framework.

TensorFlow

Tensorflow, Keras and Pytorch logos

Tensorflow

TensorFlow is an open-source deep learning framework developed by Google and released in 2015. It offers a range of tools for machine learning development and deployment. Our Introduction to Deep Neural Networks provides a complete guide to understanding deep neural networks and their significance in the modern deep learning world of artificial intelligence, along with real-world implementations in Tensorflow.

Keras

Keras is a high-level neural network framework in Python that enables rapid experimentation and development. It's open-source and can be used within other frameworks like TensorFlow, CNTK, and Theano. Our course, Image Processing with Keras in Python, teaches how to conduct image analysis using Keras with Python by constructing, training, and evaluating convolutional neural networks.

Pytorch

Released by Facebook's AI research division in 2017, it's designed for applications in natural language processing and is noted for its dynamic computational graph and memory efficiency. If you are interested in diving into Natural Language Processing, Our NLP with PyTorch: A Comprehensive Guide is a great starting point.

Each project is different, so the decision really depends on what characteristics are most important for a given use case. To help make better decisions, the following table provides a brief comparison of these frameworks, highlighting their unique features.

	Tensorflow	**Pytorch**	**Keras**
API Level	Both (High and Low)	Low	High
Architecture	Not easy to use	Complex, less readable	Simple, concise, readable

Datasets	Large datasets, high performance	Large datasets, high performance	Smaller datasets
Debugging	Difficult to conduct debugging	Good debugging capabilities	Simple network, so debugging is not often needed
Pretrained models?	Yes	Yes	Yes
Popularity	Second most popular of the three	Third most popular of the three	Most popular of the three
Speed	Fast, high-performance	Fast, high-performance	Slow, low performance
Written in	C++, CUDA, Python	Lua	Python

*Comparative table between Tensorflow, Pytorch and Keras (**source**)*

Conclusion

This article has provided a complete overview of what a CNN in deep learning is, along with their crucial role in image recognition and classification tasks.

It started by highlighting the inspiration drawn from the human visual system for the design of CNNs and then explored the key components that allow these networks to learn and make predictions.

The issue of overfitting was acknowledged as a significant challenge to CNNs' generalization capability. To mitigate this, a variety of relevant strategies to mitigate overfitting and improve CNNs overall performance were outlined.

Finally, some major deep learning CNN frameworks have been mentioned, along with the unique features of each one and how they compare to each other.

Eager to dive further into the world of AI, and machine learning? Take your expertise to the next level by enrolling in the **Deep Learning with PyTorch** course today.