Machine Learning



Machine learning (ML) is a subdomain of artificial intelligence (AI) that focuses on developing systems that learn—or improve performance—based on the data they ingest. Artificial intelligence is a broad word that refers to systems or machines that resemble human intelligence. Machine learning and AI are frequently discussed together, and the terms are occasionally used interchangeably, although they do not signify the same thing. A crucial distinction is that, while all machine learning is AI, not all AI is machine learning.

## Features of Machine learning

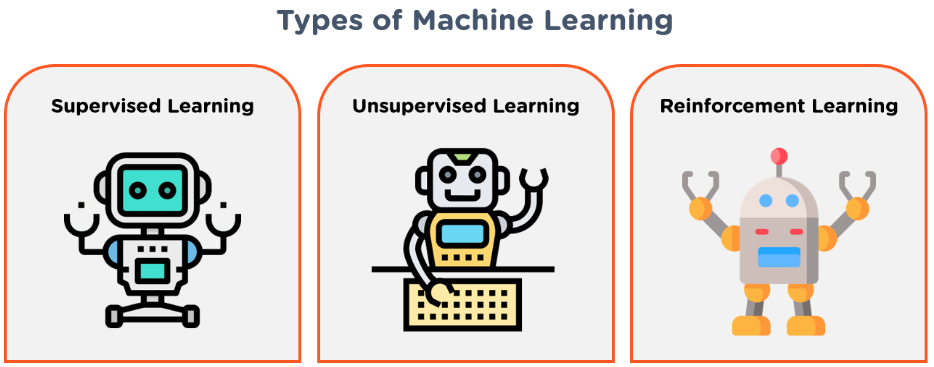
* Machine learning is data driven technology. Large amount of data generated by organizations on daily bases. So, by notable relationships in data, organizations makes better decisions.
* Machine can learn itself from past data and automatically improve.
* From the given dataset it detects various patterns on data.
* For the big organizations branding is important and it will become more easy to target relatable customer base.
* It is similar to data mining because it is also deals with the huge amount of data.

# Types of Machine Learning

There are many types of machine learning you can use in your application. The ML type is determined by many criteria such as the type and amount of data you have, what are you going to do with the ML model , how you are going to train your ML model etc…

The main categorization of ML is as follows.

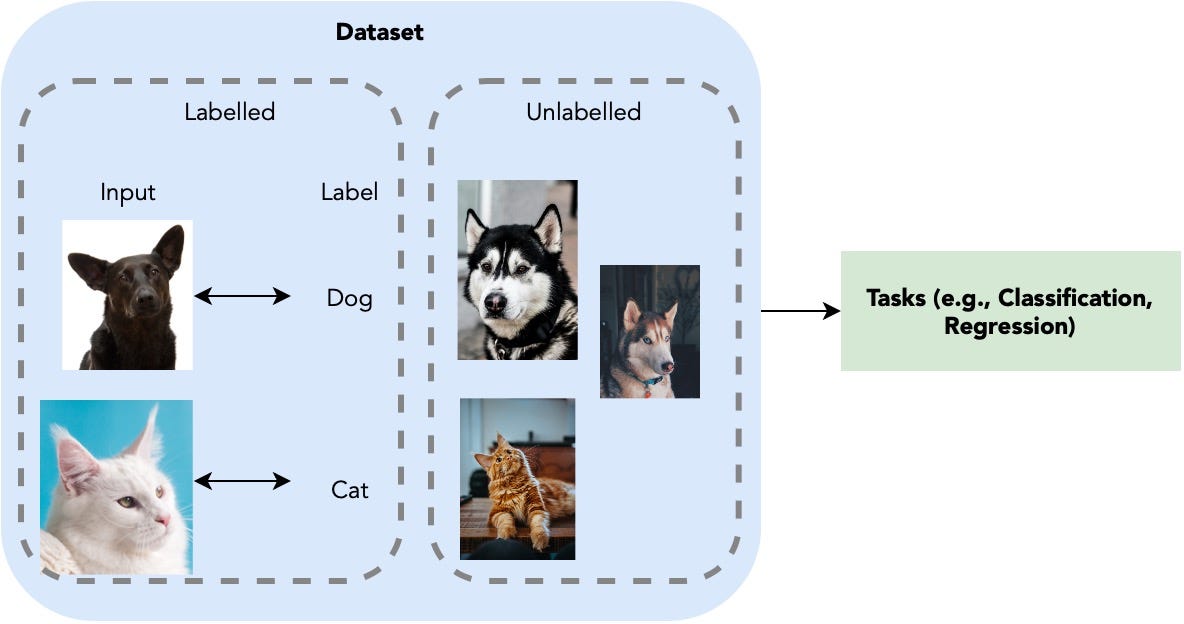
* Is the ML model is training under human supervision (supervised, unsupervised, semi supervised, and Reinforcement Learning)



### **Supervised machine learning**

[Supervised learning](https://www.ibm.com/topics/supervised-learning), also known as supervised machine learning, is defined by its use of labeled datasets to train algorithms to classify data or predict outcomes accurately. As input data is fed into the model, the model adjusts its weights until it has been fitted appropriately. This occurs as part of the cross validation process to ensure that the model avoids [overfitting](https://www.ibm.com/topics/overfitting) or [underfitting](https://www.ibm.com/topics/underfitting). Supervised learning helps organizations solve a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox. Some methods used in supervised learning include neural networks, naïve bayes, linear regression, logistic regression, random forest, and support vector machine (SVM).

EX OF S ML :



## **How Supervised Learning Works?**

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

The working of Supervised learning can be easily understood by the below example and diagram:

\uppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon. Now the first step is that we need to train the model for each shape.

* If the given shape has four sides, and all the sides are equal, then it will be labelled as a **Square**.
* If the given shape has three sides, then it will be labelled as a **triangle**.
* If the given shape has six equal sides then it will be labelled as **hexagon**.

Now, after training, we test our model using the test set, and the task of the model is to identify the shape.

The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.



### **Unsupervised machine learning**

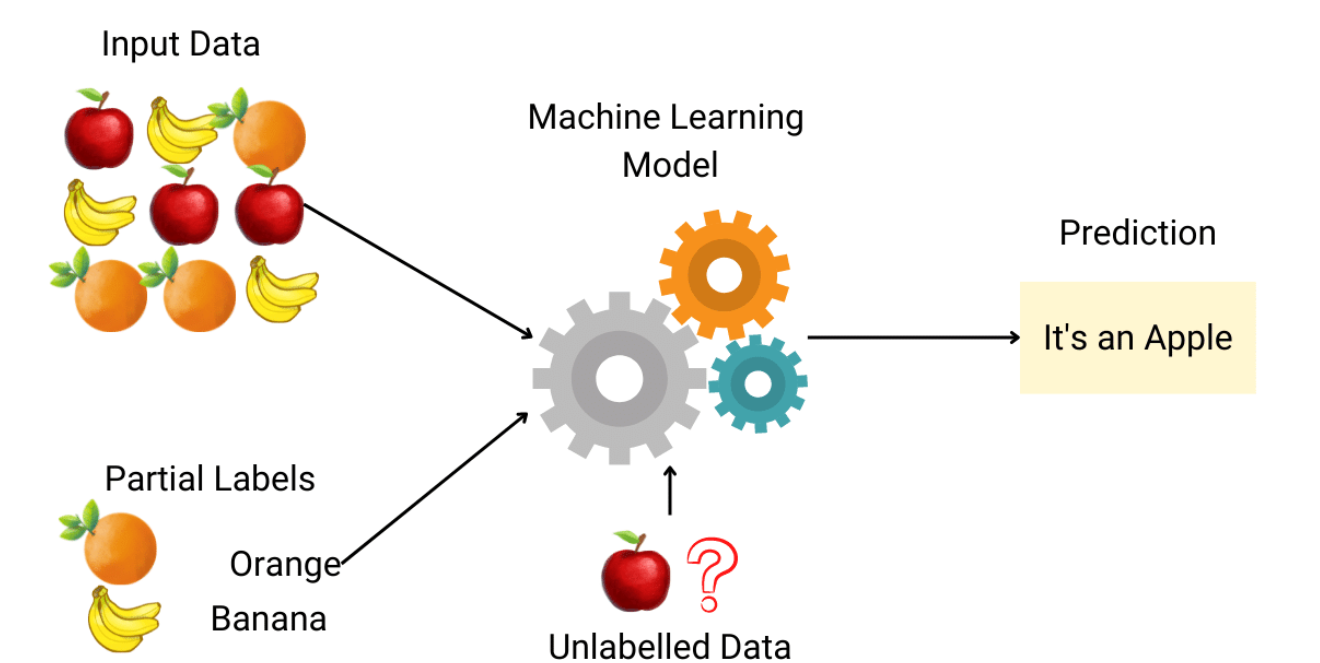
[Unsupervised learning](https://www.ibm.com/topics/unsupervised-learning), also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. This method’s ability to discover similarities and differences in information make it ideal for exploratory data analysis, cross-selling strategies, customer segmentation, and image and pattern recognition. It’s also used to reduce the number of features in a model through the process of dimensionality reduction. Principal component analysis (PCA) and singular value decomposition (SVD) are two common approaches for this. Other algorithms used in unsupervised learning include neural networks, k-means clustering, and probabilistic clustering methods.

Examples:



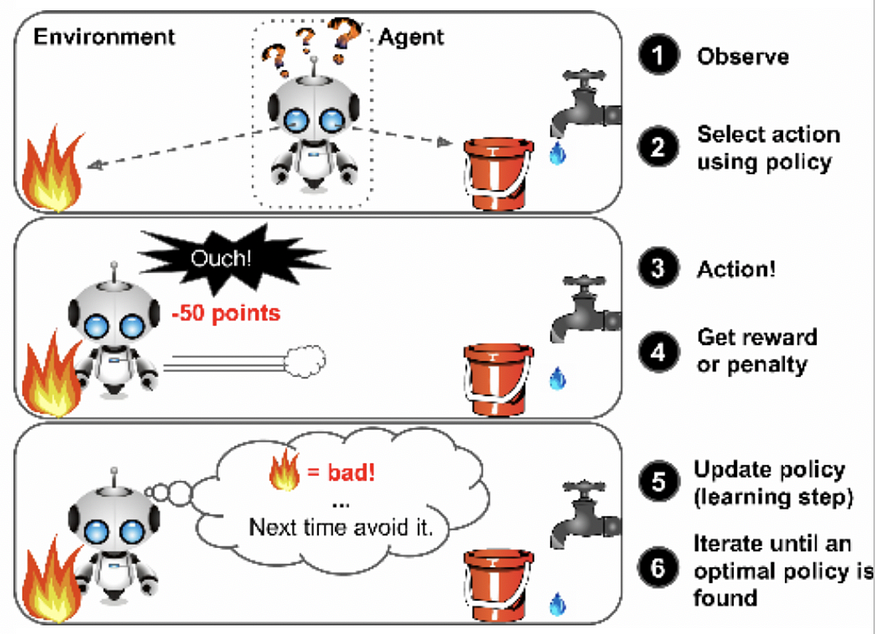
### **Semi-supervised learning**

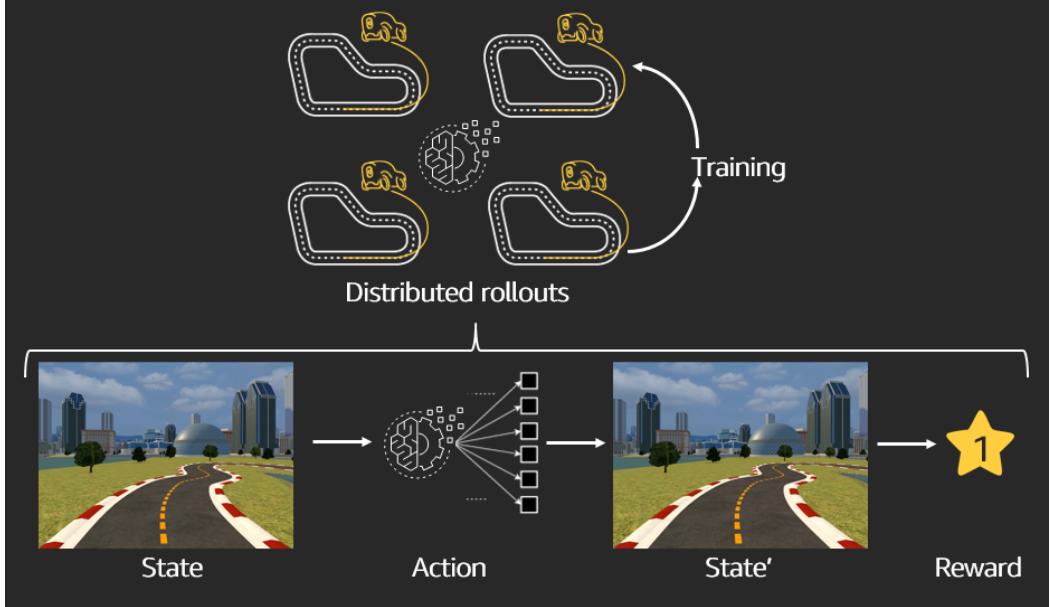
Semi-supervised learning offers a happy medium between supervised and unsupervised learning. During training, it uses a smaller labeled data set to guide classification and feature extraction from a larger, unlabeled data set. Semi-supervised learning can solve the problem of not having enough labeled data for a supervised learning algorithm. It also helps if it’s too costly to label enough data.



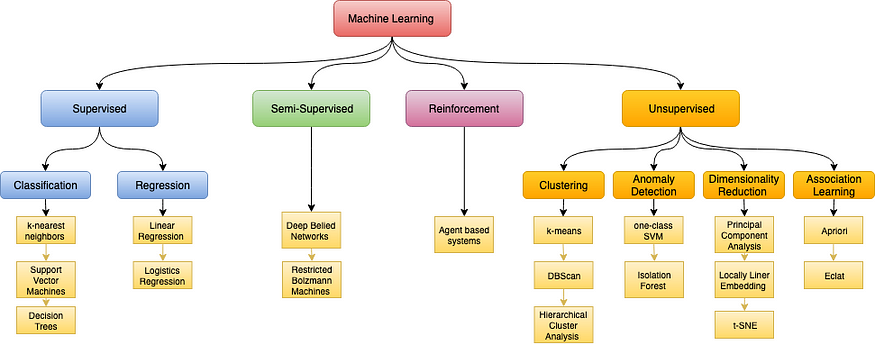
**Reinforcement Learning**

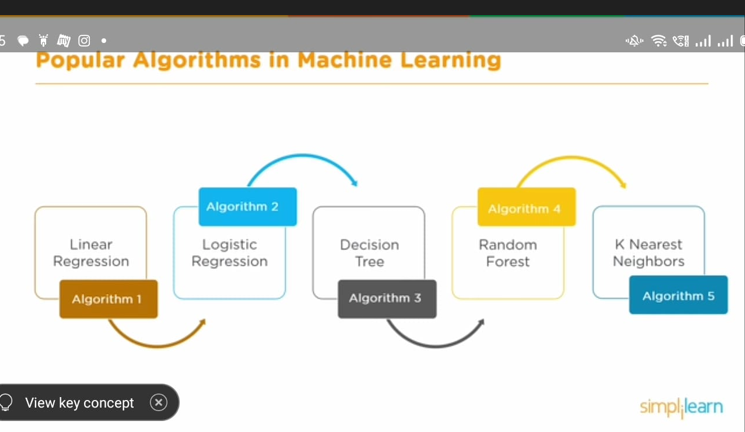
Reinforcement Learning is much different. Here the system is exposed to a situation where it can take an action. According to the action taken, it will be rewarded or penalized. The system then updates it’s policy with the actions it should and shouldn’t take. This will be continued until it finds the optimal action for a situation. Robots generally use Reinforcement Learning to learn how to walk.

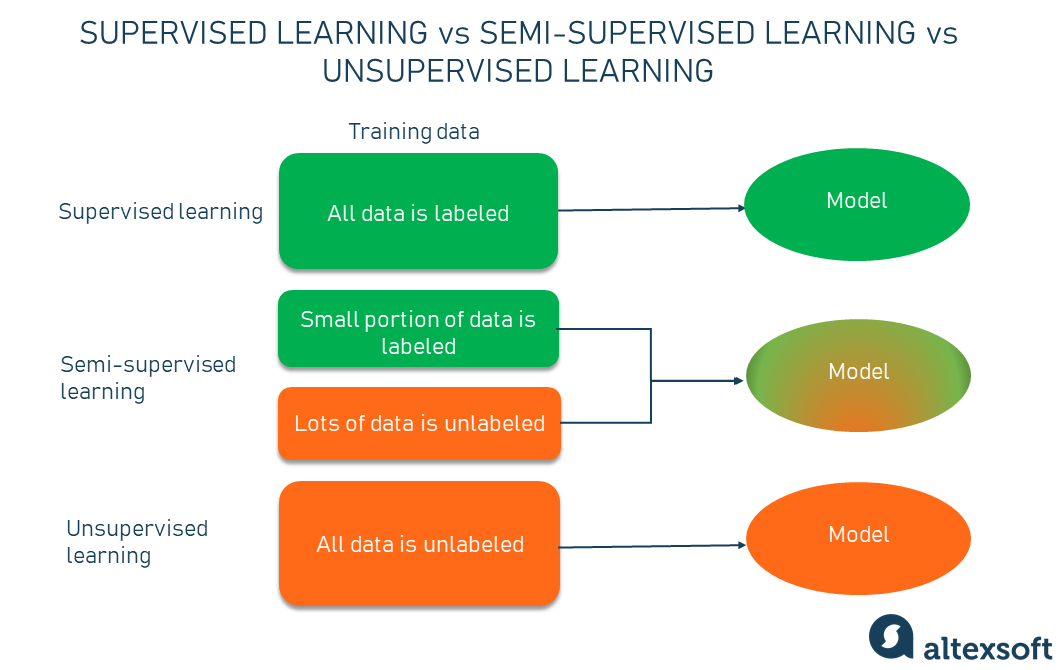


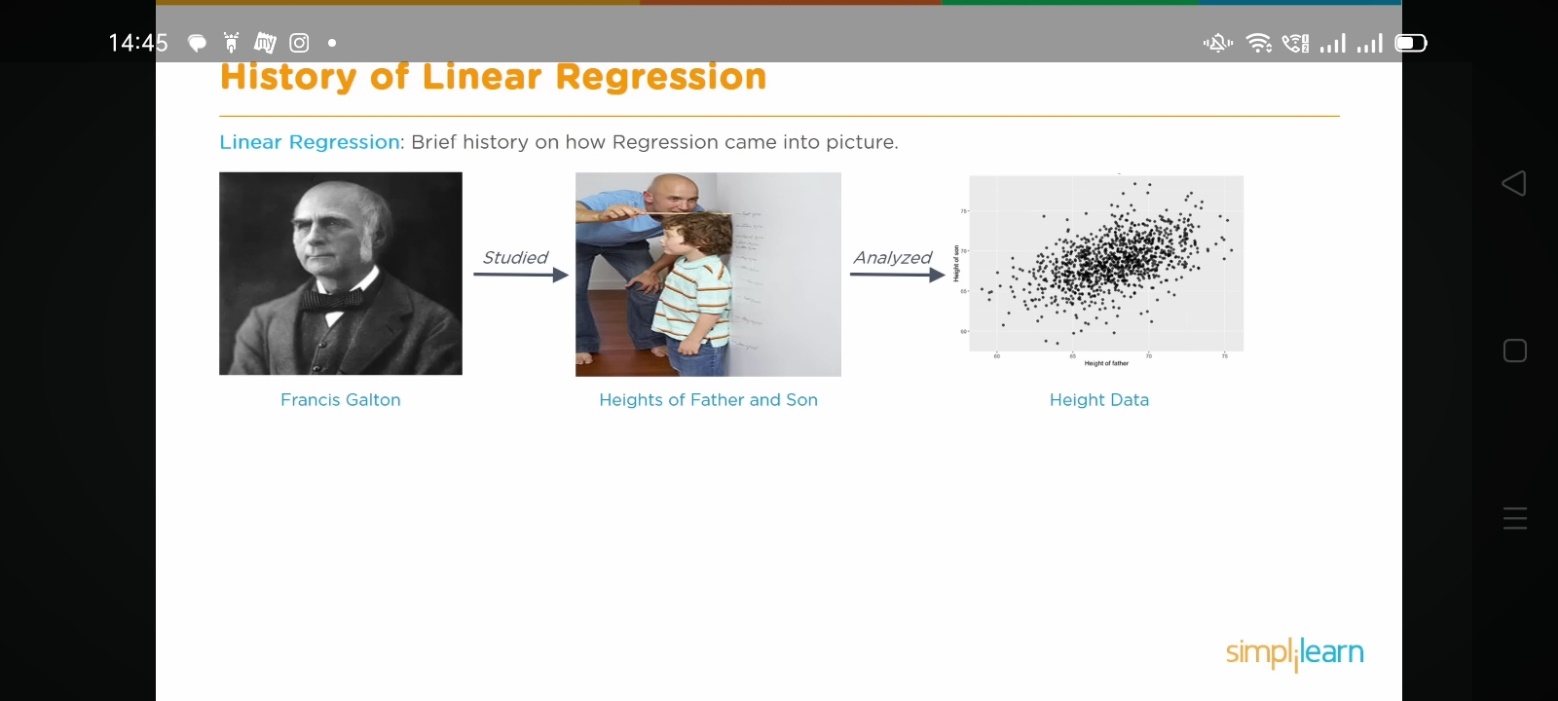


ML ALGORITHAMS









Machine learning tools:

SCIKIT-LEARN:

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

## **Features**

Rather than focusing on loading, manipulating and summarising data, Scikit-learn library is focused on modeling the data. Some of the most popular groups of models provided by Sklearn are as follows −

**Supervised Learning algorithms** − Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree etc., are the part of scikit-learn.

**Unsupervised Learning algorithms** − On the other hand, it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks.

**Clustering** − This model is used for grouping unlabeled data.

**Cross Validation** − It is used to check the accuracy of supervised models on unseen data.

**Dimensionality Reduction** − It is used for reducing the number of attributes in data which can be further used for summarisation, visualisation and feature selection.

**Ensemble methods** − As name suggest, it is used for combining the predictions of multiple supervised models.

**Feature extraction** − It is used to extract the features from data to define the attributes in image and text data.

**Feature selection** − It is used to identify useful attributes to create supervised models.

**Open Source** − It is open source library and also commercially usable under BSD license.

# Scikit Learn - Modelling Process

## **Dataset Loading**

A collection of data is called dataset. It is having the following two components −

**Features** − The variables of data are called its features. They are also known as predictors, inputs or attributes.

* **Feature matrix** − It is the collection of features, in case there are more than one.
* **Feature Names** − It is the list of all the names of the features.

**Response** − It is the output variable that basically depends upon the feature variables. They are also known as target, label or output.

* **Response Vector** − It is used to represent response column. Generally, we have just one response column.
* **Target Names** − It represent the possible values taken by a response vector.

Scikit-learn have few example datasets like **iris** and **digits** for classification and the **Boston house prices** for regression.

### **Example**

Following is an example to load **iris** dataset −

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

feature\_names = iris.feature\_names

target\_names = iris.target\_names

print("Feature names:", feature\_names)

print("Target names:", target\_names)

print("\nFirst 10 rows of X:\n", X[:10])

## **Splitting the dataset**

To check the accuracy of our model, we can split the dataset into two pieces-**a training set** and **a testing set**. Use the training set to train the model and testing set to test the model. After that, we can evaluate how well our model did.

### **Example**

The following example will split the data into 70:30 ratio, i.e. 70% data will be used as training data and 30% will be used as testing data. The dataset is iris dataset as in above example.

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size = 0.3, random\_state = 1

)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

### **Output**

(105, 4)

(45, 4)

(105,)

(45,)

As seen in the example above, it uses **train\_test\_split()** function of scikit-learn to split the dataset. This function has the following arguments −

* **X, y** − Here, **X** is the **feature matrix** and y is the **response vector**, which need to be split.
* **test\_size** − This represents the ratio of test data to the total given data. As in the above example, we are setting **test\_data = 0.3** for 150 rows of X. It will produce test data of 150\*0.3 = 45 rows.
* **random\_size** − It is used to guarantee that the split will always be the same. This is useful in the situations where you want reproducible results.

## **Train the Model**

Next, we can use our dataset to train some prediction-model. As discussed, scikit-learn has wide range of **Machine Learning (ML) algorithms** which have a consistent interface for fitting, predicting accuracy, recall etc.

### **Example**

In the example below, we are going to use KNN (K nearest neighbors) classifier. Don’t go into the details of KNN algorithms, as there will be a separate chapter for that. This example is used to make you understand the implementation part only.

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size = 0.4, random\_state=1

)

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

classifier\_knn = KNeighborsClassifier(n\_neighbors = 3)

classifier\_knn.fit(X\_train, y\_train)

y\_pred = classifier\_knn.predict(X\_test)

# Finding accuracy by comparing actual response values(y\_test)with predicted response value(y\_pred)

print("Accuracy:", metrics.accuracy\_score(y\_test, y\_pred))

# Providing sample data and the model will make prediction out of that data

sample = [[5, 5, 3, 2], [2, 4, 3, 5]]

preds = classifier\_knn.predict(sample)

pred\_species = [iris.target\_names[p] for p in preds] print("Predictions:", pred\_species)

### **Output**

Accuracy: 0.9833333333333333

Predictions: ['versicolor', 'virginica']

## **Model Persistence**

Once you train the model, it is desirable that the model should be persist for future use so that we do not need to retrain it again and again. It can be done with the help of **dump** and **load** features of ***joblib*** package.

Consider the example below in which we will be saving the above trained model (classifier\_knn) for future use −

from sklearn.externals import joblib

joblib.dump(classifier\_knn, 'iris\_classifier\_knn.joblib')

The above code will save the model into file named iris\_classifier\_knn.joblib. Now, the object can be reloaded from the file with the help of following code −

joblib.load('iris\_classifier\_knn.joblib')

## **Preprocessing the Data**

As we are dealing with lots of data and that data is in raw form, before inputting that data to machine learning algorithms, we need to convert it into meaningful data. This process is called preprocessing the data. Scikit-learn has package named **preprocessing** for this purpose. The **preprocessing** package has the following techniques −

## **Binarisation**

This preprocessing technique is used when we need to convert our numerical values into Boolean values.

### **Example**

import numpy as np

from sklearn import preprocessing

Input\_data = np.array(

[2.1, -1.9, 5.5],

[-1.5, 2.4, 3.5],

[0.5, -7.9, 5.6],

[5.9, 2.3, -5.8]]

)

data\_binarized = preprocessing.Binarizer(threshold=0.5).transform(input\_data)

print("\nBinarized data:\n", data\_binarized)

In the above example, we used **threshold value** = 0.5 and that is why, all the values above 0.5 would be converted to 1, and all the values below 0.5 would be converted to 0.

### **Output**

Binarized data:

[

[ 1. 0. 1.]

[ 0. 1. 1.]

[ 0. 0. 1.]

[ 1. 1. 0.]

]

## **Mean Removal**

This technique is used to eliminate the mean from feature vector so that every feature centered on zero.

### **Example**

import numpy as np

from sklearn import preprocessing

Input\_data = np.array(

[2.1, -1.9, 5.5],

[-1.5, 2.4, 3.5],

[0.5, -7.9, 5.6],

[5.9, 2.3, -5.8]]

)

#displaying the mean and the standard deviation of the input data

print("Mean =", input\_data.mean(axis=0))

print("Stddeviation = ", input\_data.std(axis=0))

#Removing the mean and the standard deviation of the input data

data\_scaled = preprocessing.scale(input\_data)

print("Mean\_removed =", data\_scaled.mean(axis=0))

print("Stddeviation\_removed =", data\_scaled.std(axis=0))

### **Output**

Mean = [ 1.75 -1.275 2.2 ]

Stddeviation = [ 2.71431391 4.20022321 4.69414529]

Mean\_removed = [ 1.11022302e-16 0.00000000e+00 0.00000000e+00]

Stddeviation\_removed = [ 1. 1. 1.]

## **Scaling**

We use this preprocessing technique for scaling the feature vectors. Scaling of feature vectors is important, because the features should not be synthetically large or small.

### **Example**

import numpy as np

from sklearn import preprocessing

Input\_data = np.array(

[

[2.1, -1.9, 5.5],

[-1.5, 2.4, 3.5],

[0.5, -7.9, 5.6],

[5.9, 2.3, -5.8]

]

)

data\_scaler\_minmax = preprocessing.MinMaxScaler(feature\_range=(0,1))

data\_scaled\_minmax = data\_scaler\_minmax.fit\_transform(input\_data)

print ("\nMin max scaled data:\n", data\_scaled\_minmax)

### **Output**

Min max scaled data:

[

[ 0.48648649 0.58252427 0.99122807]

[ 0. 1. 0.81578947]

[ 0.27027027 0. 1. ]

[ 1. 0.99029126 0. ]

]

## **Normalisation**

We use this preprocessing technique for modifying the feature vectors. Normalisation of feature vectors is necessary so that the feature vectors can be measured at common scale. There are two types of normalisation as follows −

### **L1 Normalisation**

It is also called Least Absolute Deviations. It modifies the value in such a manner that the sum of the absolute values remains always up to 1 in each row. Following example shows the implementation of L1 normalisation on input data.

### **Example**

import numpy as np

from sklearn import preprocessing

Input\_data = np.array(

[

[2.1, -1.9, 5.5],

[-1.5, 2.4, 3.5],

[0.5, -7.9, 5.6],

[5.9, 2.3, -5.8]

]

)

data\_normalized\_l1 = preprocessing.normalize(input\_data, norm='l1')

print("\nL1 normalized data:\n", data\_normalized\_l1)

### **Output**

L1 normalized data:

[

[ 0.22105263 -0.2 0.57894737]

[-0.2027027 0.32432432 0.47297297]

[ 0.03571429 -0.56428571 0.4 ]

[ 0.42142857 0.16428571 -0.41428571]

]

### **L2 Normalisation**

Also called Least Squares. It modifies the value in such a manner that the sum of the squares remains always up to 1 in each row. Following example shows the implementation of L2 normalisation on input data.

### **Example**

import numpy as np

from sklearn import preprocessing

Input\_data = np.array(

[

[2.1, -1.9, 5.5],

[-1.5, 2.4, 3.5],

[0.5, -7.9, 5.6],

[5.9, 2.3, -5.8]

]

)

data\_normalized\_l2 = preprocessing.normalize(input\_data, norm='l2')

print("\nL1 normalized data:\n", data\_normalized\_l2)

### **Output**

L2 normalized data:

[

[ 0.33946114 -0.30713151 0.88906489]

[-0.33325106 0.53320169 0.7775858 ]

[ 0.05156558 -0.81473612 0.57753446]

[ 0.68706914 0.26784051 -0.6754239 ]

]

## **What is Matplotlib?**

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Matplotlib was created by John D. Hunter.

Matplotlib is open source and we can use it freely.

Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

## **Plotting x and y points**

The plot() function is used to draw points (markers) in a diagram.

By default, the plot() function draws a line from point to point.

The function takes parameters for specifying points in the diagram.

Parameter 1 is an array containing the points on the **x-axis**.

Parameter 2 is an array containing the points on the **y-axis**.

If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

## 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.

## nspiration 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:

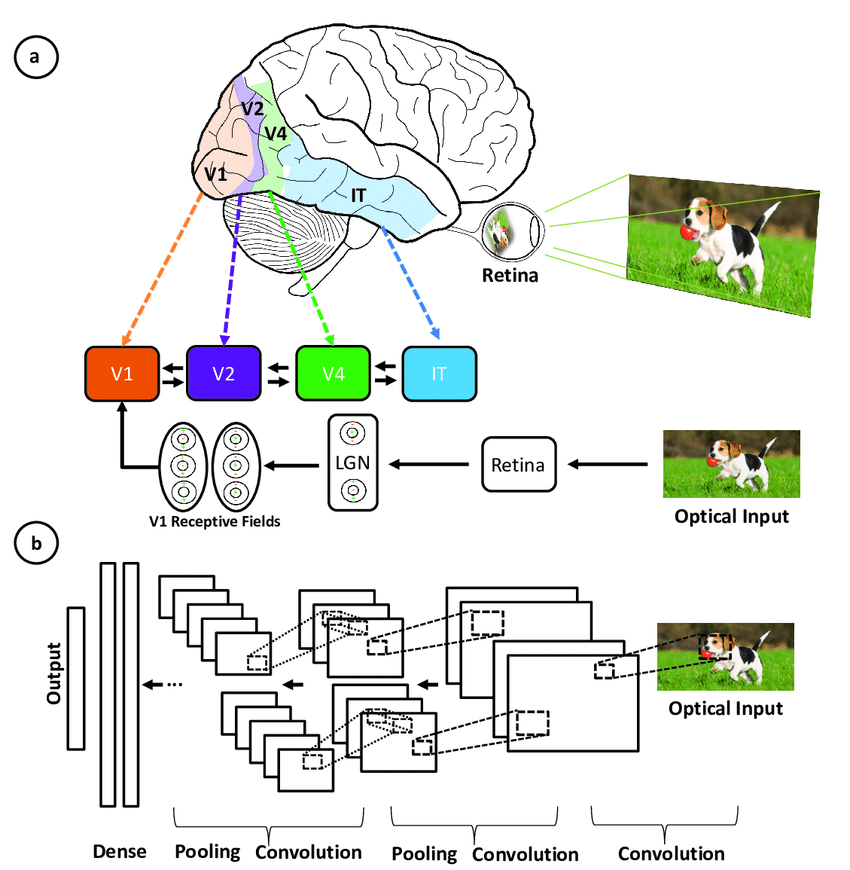


Illustration of the correspondence between the areas associated with the primary visual cortex and the layers in a convolutional neural network ([***source***](https://www.researchgate.net/figure/2-Illustration-of-the-corrispondence-between-the-areas-associated-with-the-primary_fig7_317679065))

* **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
* input volume through the convolution operation. This local 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 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.

## Key Components of a CNN

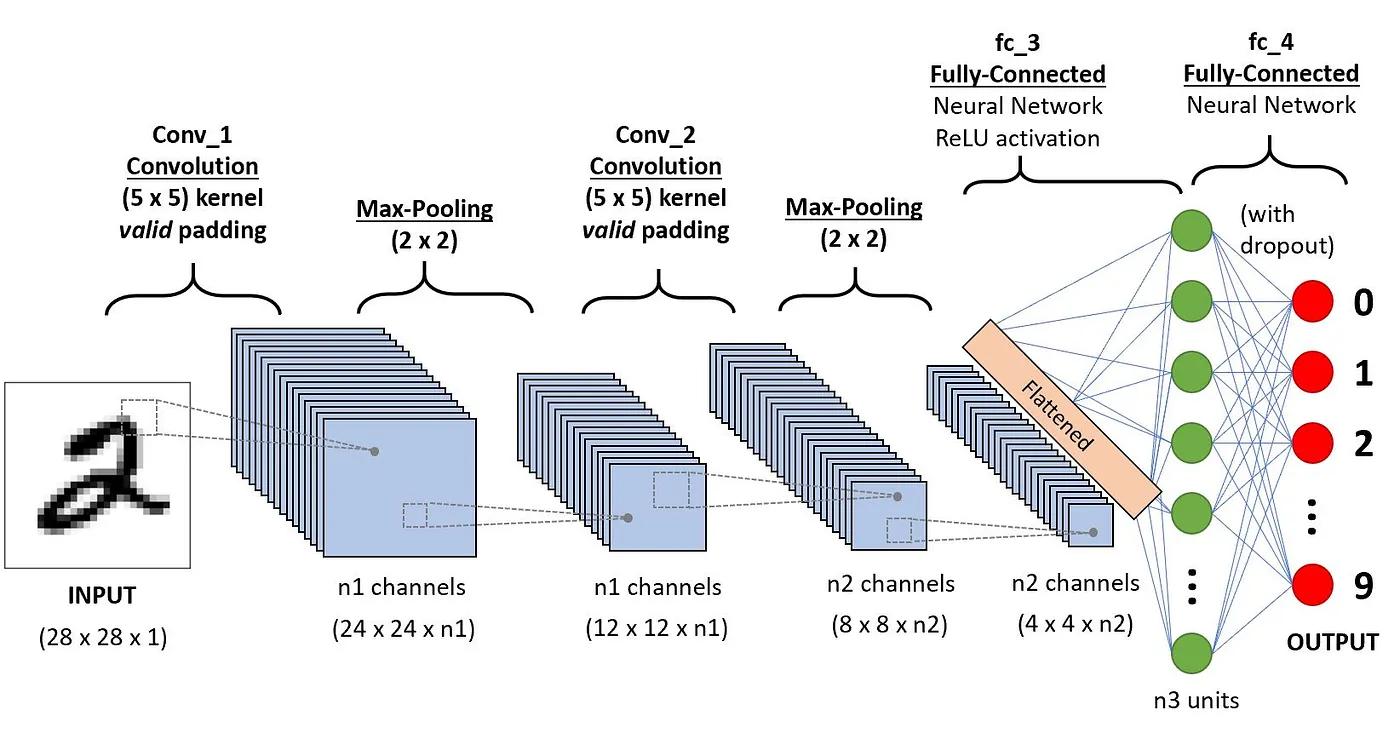
The convolutional neural network is made of four main parts.

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.



Architecture of the CNNs applied to digit recognition ([***source***](https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53))

### 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.

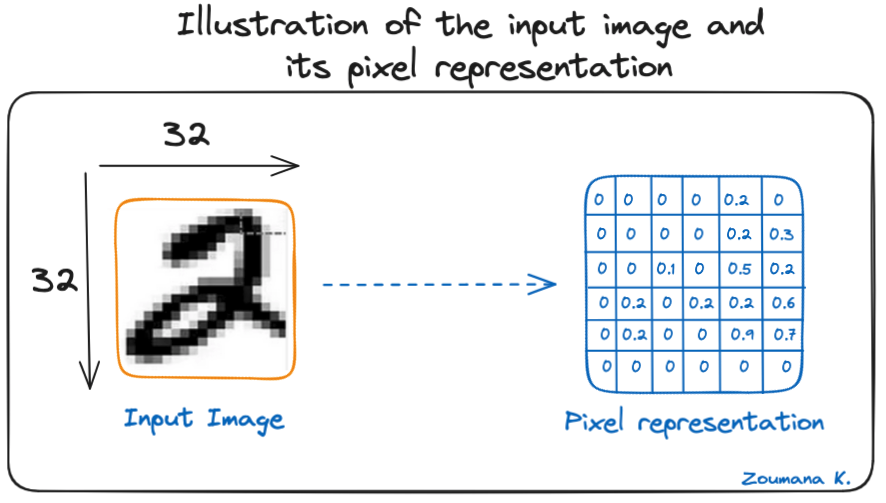


Illustration of the input image and its pixel representation

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.

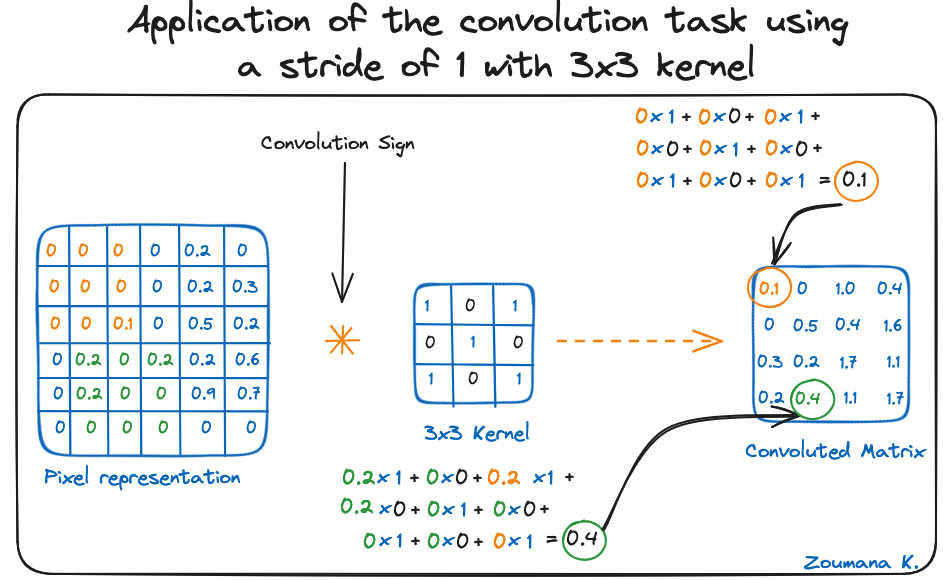
**Do we have to manually find these weights?**

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.



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.

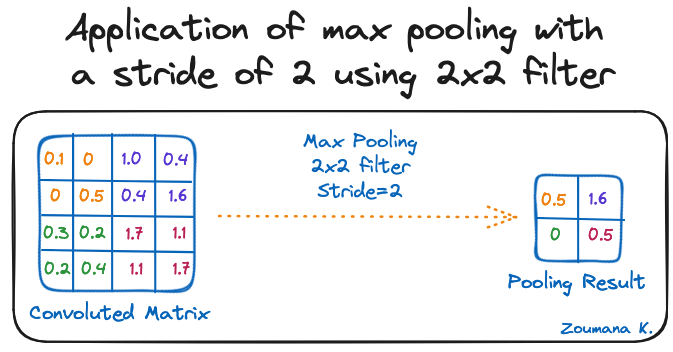
### Pooling layer

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:



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

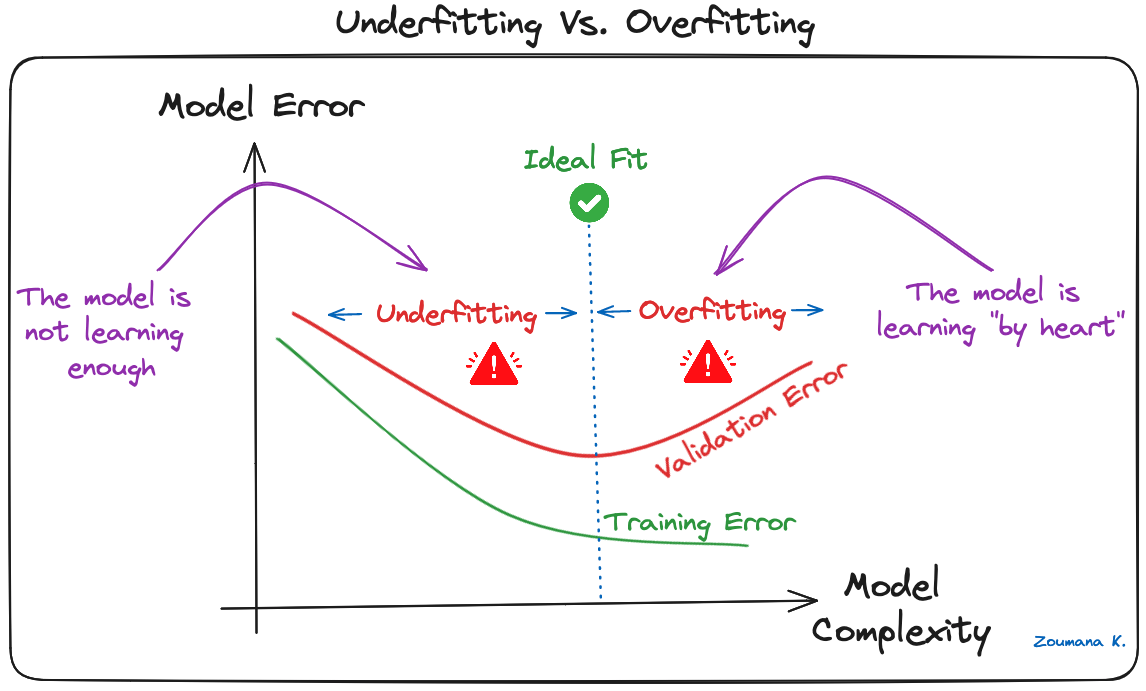
These layers are in the last layer of the convolutional neural network, and their inputs correspond to the flattened one-dimensional matrix generated by the last pooling layer. ReLU activations functions are applied to them for non-linearity.

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.

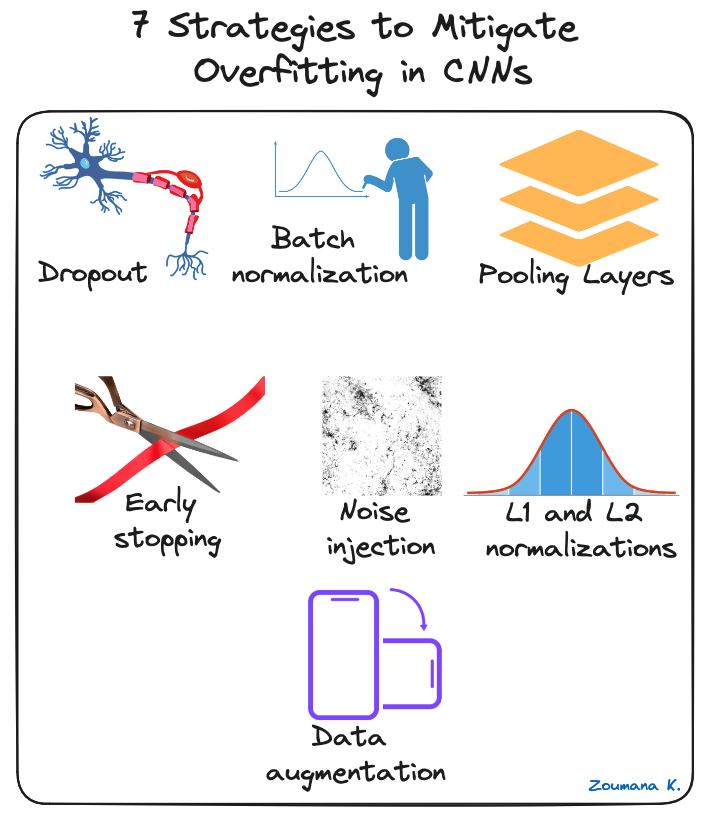
This can be observed when the performance on training data is too low compared to the performance on validation or testing data, and a graphical illustration is given below:



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:

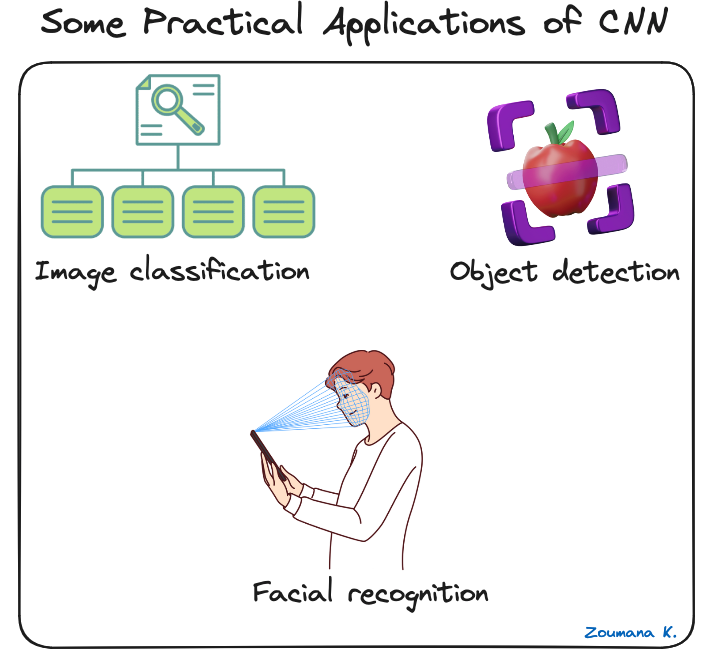


7 strategies to mitigate overfitting in CNNs

* **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**](https://www.datacamp.com/tutorial/cnn-tensorflow-python) 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, 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**](https://www.datacamp.com/tutorial/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**](https://www.datacamp.com/courses/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**](https://www.datacamp.com/tutorial/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.

