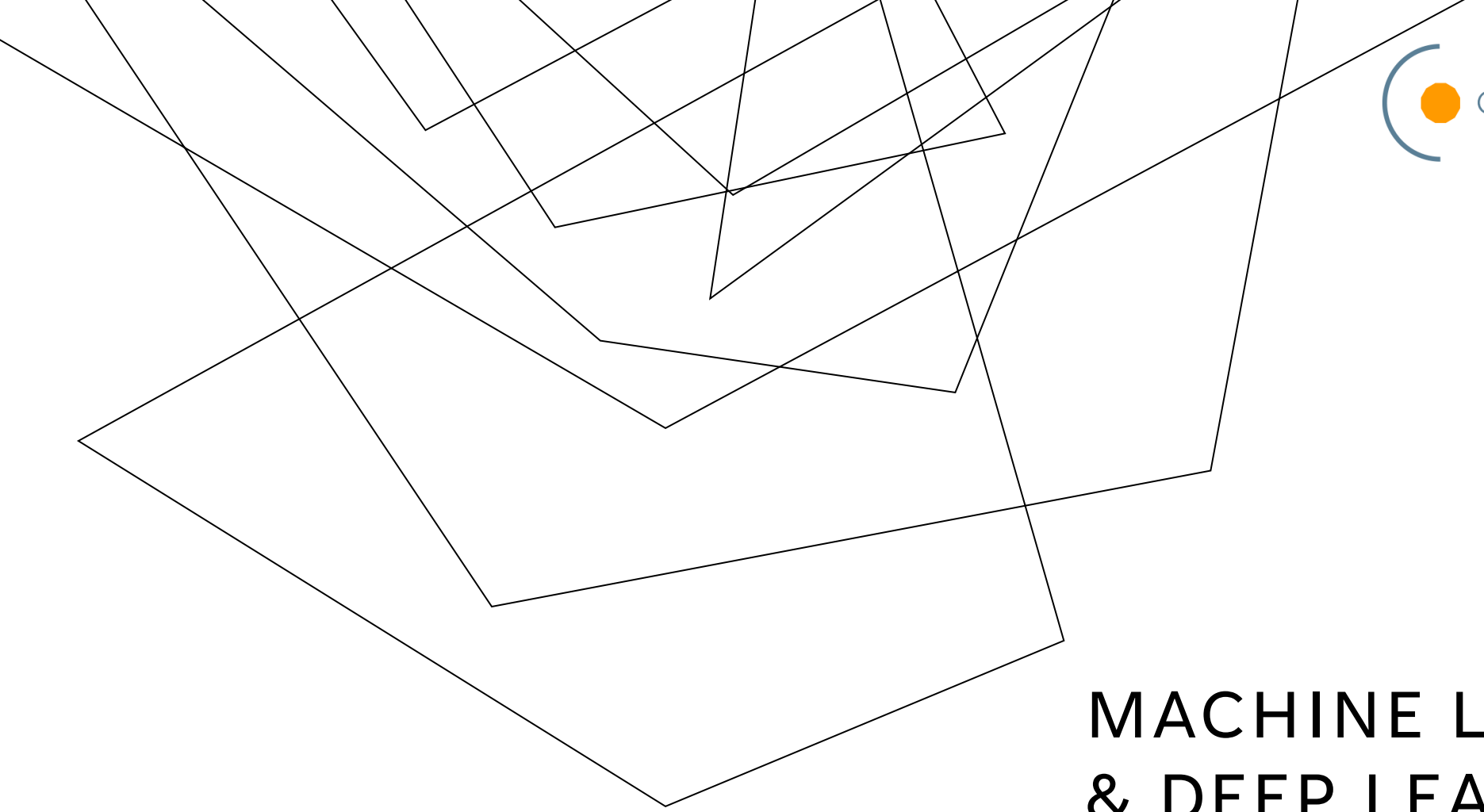


centralelille

ufr3s faculté d'ingénierie et management de la santé

Université de Lille

MACHINE LEARNING & DEEP LEARNING COURSE – MIAS M2

Idriss JAIRI

Idriss.jairi@univ-lille.fr

COURSE PLAN

Session 1: Classical Machine Learning

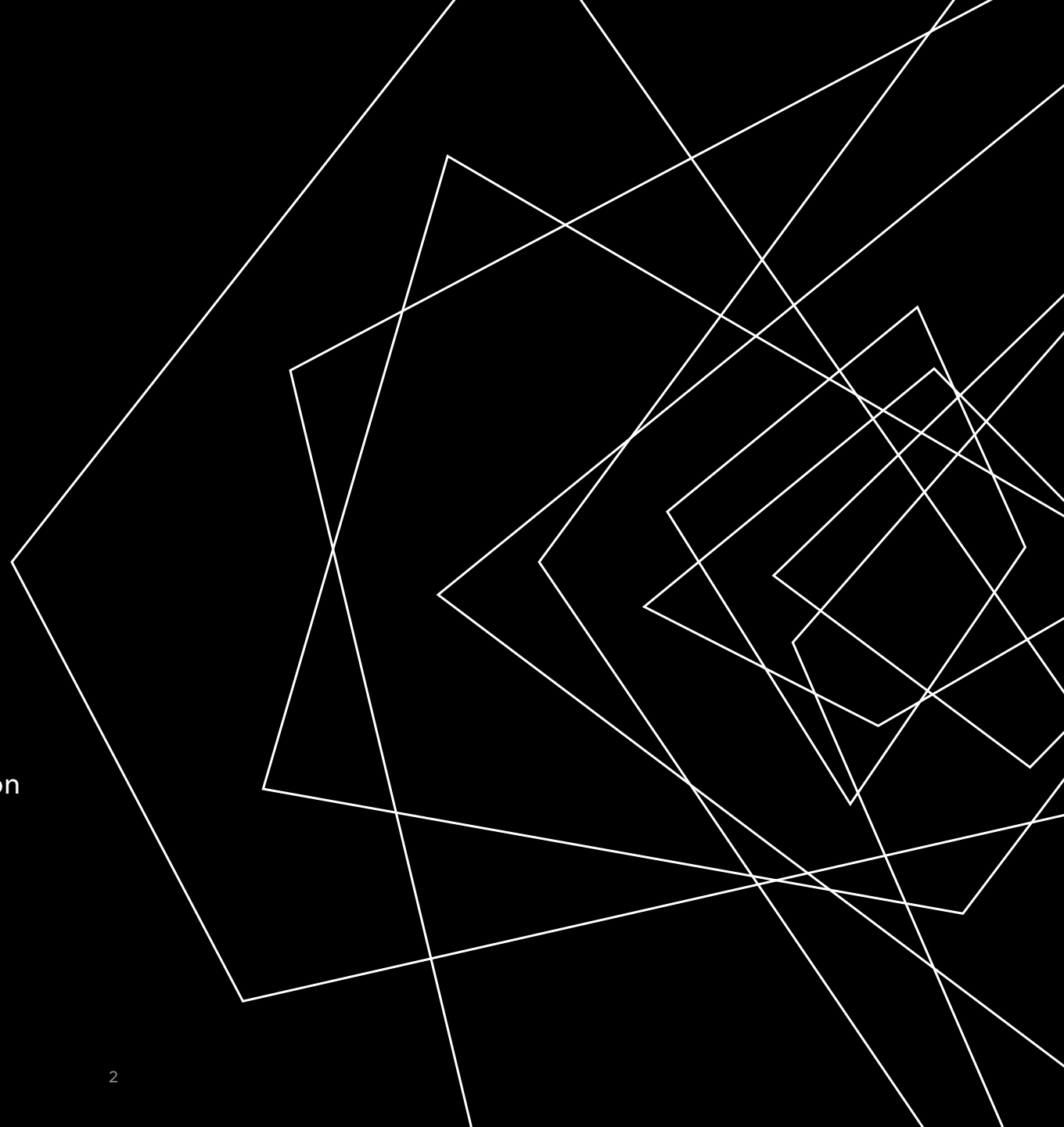
Session 2: Deep Learning

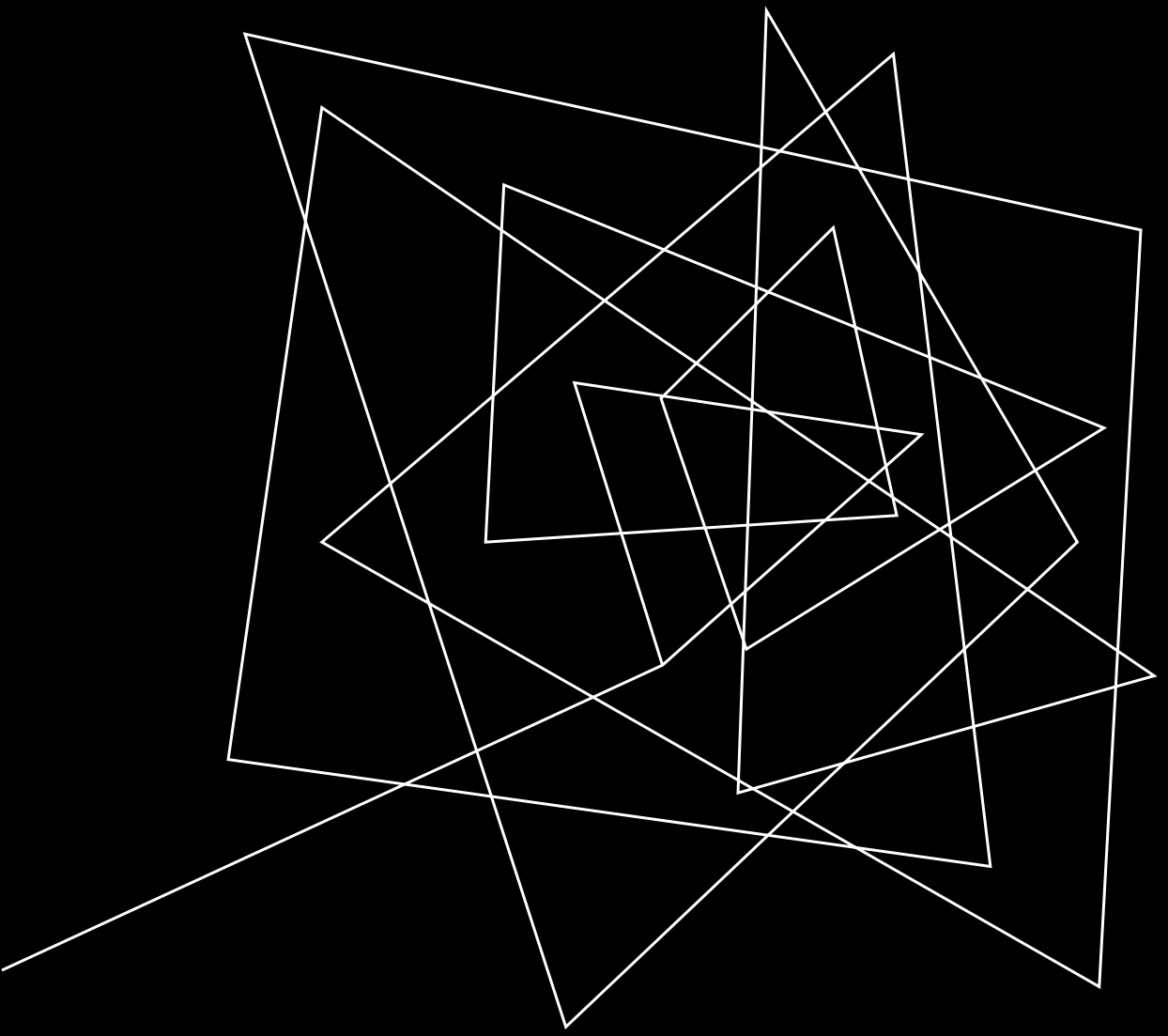
Session 3: Deep Learning for Computer Vision

Session 4: YOLO for Object Detection

Session 5: U-NET for Biomedical Image Segmentation

Session 6: Reinforcement Learning





SESSION 1: MACHINE LEARNING

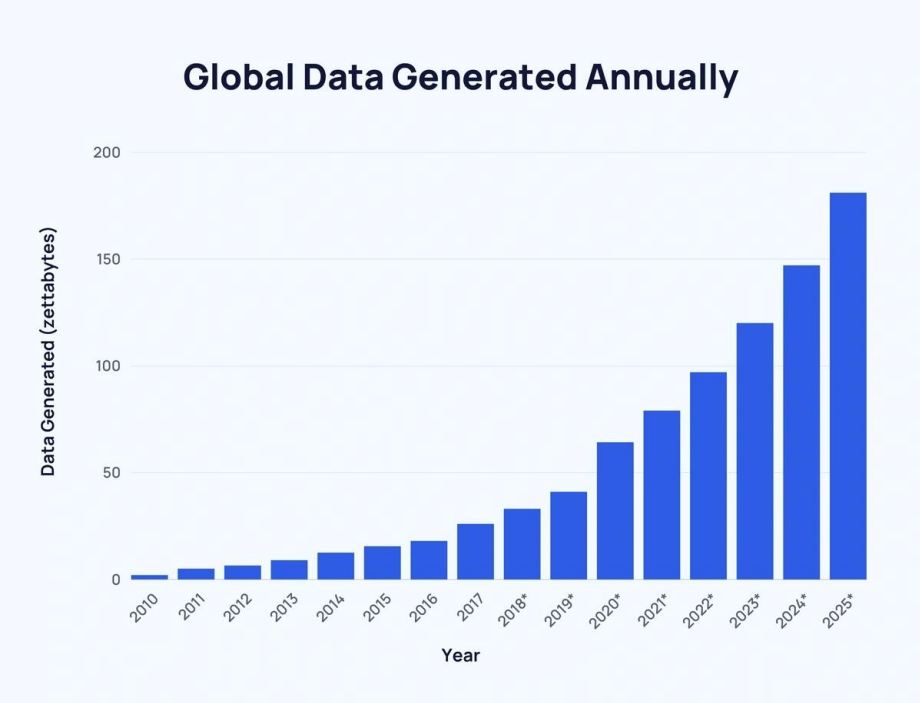
Introduction to AI, Supervised Learning, Unsupervised Learning

Introduction: Why has Artificial Intelligence (AI) become so popular?

Big tech companies invest a lot of effort and money in AI

Scientific research and impressive results

Introduction: Why has Artificial Intelligence (AI) become so popular? <Increased volumes of data>

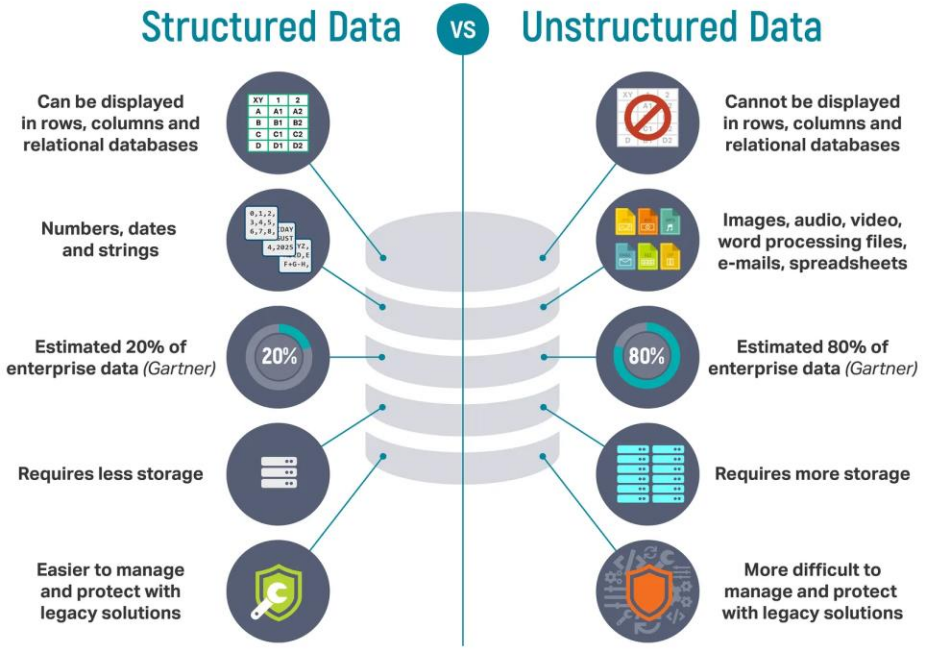


Source: https://explodingtopics.com/blog/data-generated-per-day

Note: 1 Zettabyte = 10^{12} Gigabyte

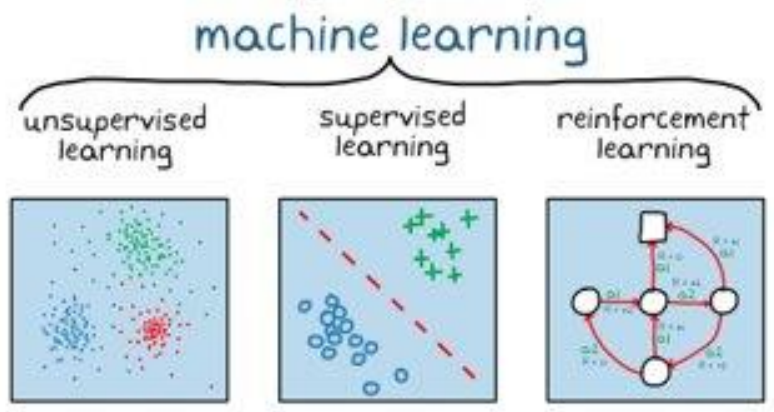
Introduction: Why has Artificial Intelligence (AI) become so popular? <Increased volumes of data>

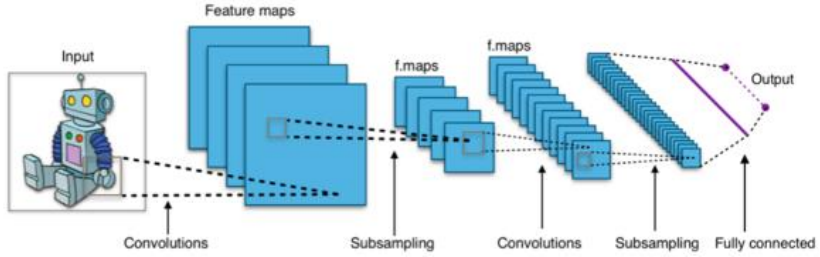
- **Data Availability**: Vast amounts of structured and unstructured data are available for training and testing AI models, which is crucial for their learning and decision-making processes.
- **Data Collection and Storage Technologies**: Innovations in data collection methods, sensors, and storage technologies have made it possible to capture and store vast quantities of data. This is a critical component for training and refining AI algorithms.



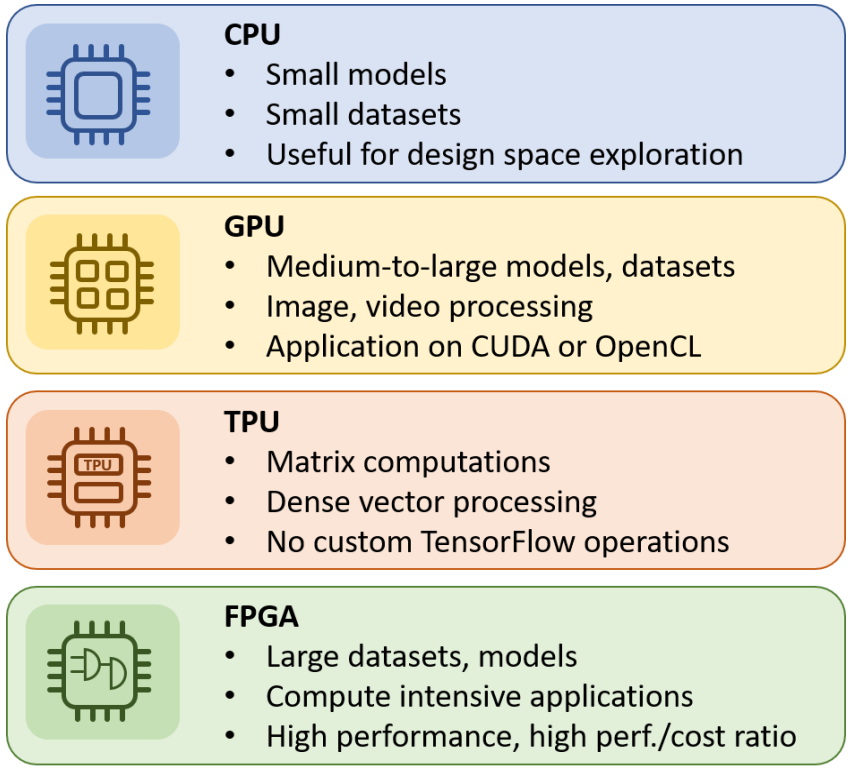
Introduction: Why has Artificial Intelligence (AI) become so popular? <Advanced algorithms>

1. **Machine Learning Algorithms**: Breakthroughs in machine learning algorithms, particularly in areas like deep learning, have significantly improved the performance of AI systems. Deep learning, in particular, has proven to be highly effective in tasks like image recognition, natural language processing, and more.
2. **Transfer Learning and Pretrained Models**: Techniques like transfer learning, where a model trained on a large dataset for one task is adapted for another related task, have accelerated progress in various domains. Pretrained models, which are models that have been trained on large datasets and then fine-tuned for specific tasks, have become a powerful tool for many applications.
3. **Reinforcement Learning and Generative Models**: Advances in reinforcement learning have enabled AI systems to learn through interaction with their environment, making them well-suited for tasks like autonomous control. Generative models, such as Generative Adversarial Networks (GANs), have enabled the creation of realistic synthetic data, which has numerous applications.



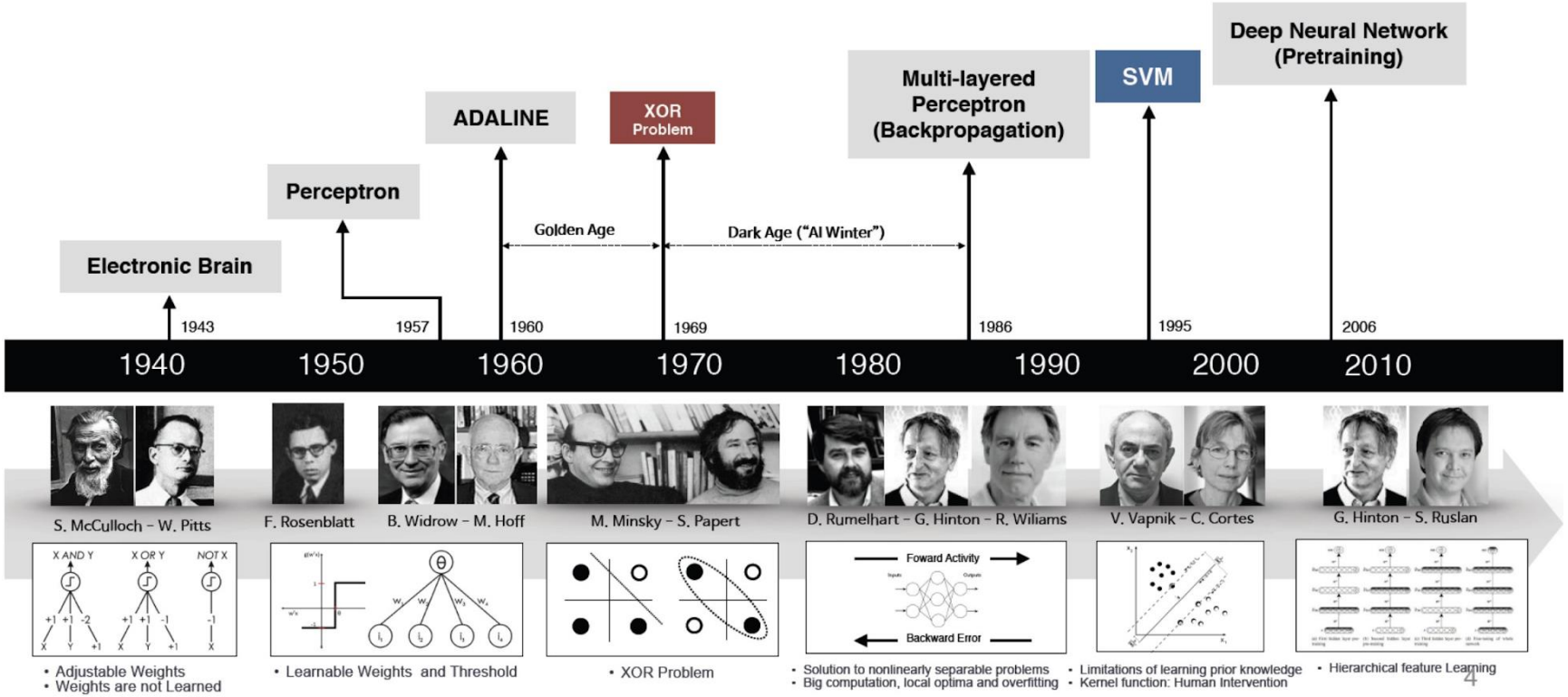


Introduction: Why has Artificial Intelligence (AI) become so popular? <Advancements in Computing Power>



- **CPUs** (Central Processing Units), **GPUs** (Graphics Processing Units), and **TPUs** (Tensor Processing Units) are different types of processors designed for specific types of computational tasks.
- **Specialized Hardware:** The development of specialized hardware like Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) has further accelerated the training and deployment of AI models, especially in deep learning applications.

Introduction: A Brief History of AI



Introduction: A Brief History of AI
<From 1940s to Now>

- **1943: Warren McCulloch and Walter Pitts create a mathematical model of a neural network.**
- **1949**: Donald Hebb proposes a learning rule for neural networks, known as Hebbian learning.
- **1950: Alan Turing introduces the "Turing Test" as a way to evaluate a machine's ability to exhibit intelligent behavior.**
- **1951**: Marvin Minsky and Dean Edmonds build the first neural network computer, SNARC.
- **1956: John McCarthy organized the Dartmouth Conference, which is considered the birth of AI as a field of study. "The term artificial intelligence was first coined by John McCarthy in 1956 "**
- **1967: Frank Rosenblatt develops the perceptron, a simplified model of a biological neuron, which becomes one of the earliest machine learning algorithms. It lays the groundwork for later developments in neural networks.**
- **1960:** John McCarthy develops the programming language LISP, which becomes widely used in AI research.
- **1966:** Shakey the robot, developed at Stanford Research Institute, demonstrates basic problem-solving abilities.
- **1966:** ELIZA is an early natural language processing computer program, the first program that allowed some kind of plausible conversation between humans and machines.

Note: There were two major AI winters (Dark Age of AI) approximately **1974–1980** and **1987–2000**

Introduction: A Brief History of AI <From 1940s to Now>

- **1968:** Terry Winograd develops SHRDLU, a natural language processing system.
- **1970: Expert systems, which are rule-based AI systems, gain popularity. "Expert systems were among the first truly successful forms of artificial intelligence (AI) software"**
- **1972:** The MYCIN system, developed at Stanford, becomes one of the first expert systems used for medical diagnosis.
- **1980:** The first commercial expert system, XCON, is deployed at Digital Equipment Corporation.
- **1986: The concept of backpropagation, a key algorithm for training artificial neural networks, is rediscovered.**
- **1997: IBM's Deep Blue defeats world chess champion Garry Kasparov in a six-game match.**
- **2002:** Roomba, a domestic robot developed by iRobot, is introduced, showcasing advancements in robotics and AI.
- **2010: Deep learning techniques, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), gain prominence.**
- **2011 : IBM's Watson wins the Jeopardy! game show, demonstrating natural language processing and question-answering capabilities.**

Introduction: A Brief History of AI
<From 1940s to Now>

- **2013: Google's DeepMind develops a deep learning algorithm (Q-Learning) that learns to play Atari 2600 video games at a human-level performance. "Playing Atari with Deep Reinforcement Learning".**
- **2016: AlphaGo, a program developed by DeepMind, defeats world Go champion Lee Sedol in a five-game match.**
- 2018: GPT-1 (Generative Pre-trained Transformer) by OpenAI showcases powerful language generation capabilities.
- 2019: OpenAI introduces GPT-2, a large-scale language model
- 2020: OpenAI introduces GPT-3, a large-scale language model.
- **2021: AlphaFold is an artificial intelligence program developed by DeepMind, which performs predictions of protein structure.**
- **2022: Generative Pre-trained Transformer 3.5 (GPT-3.5) is a sub class of GPT-3 Models created by OpenAI in 2022**
- **2023: Generative Pre-trained Transformer 4 (GPT-4) is a multimodal large language model created by OpenAI, and the fourth in its series of GPT foundation models**

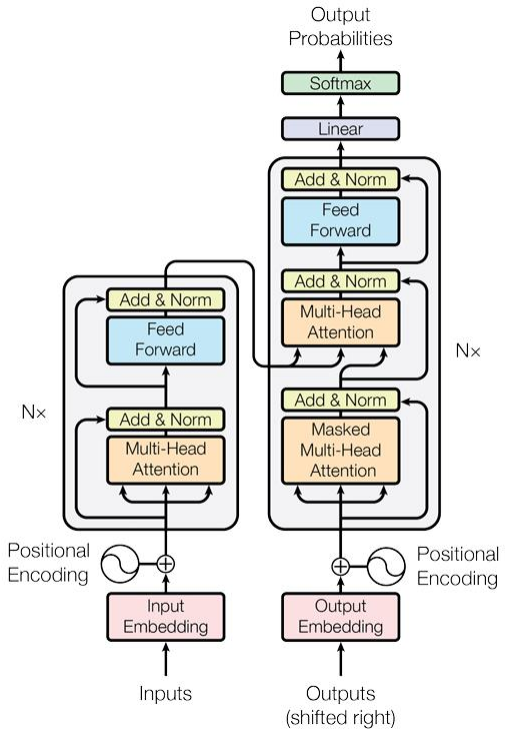
Introduction: A Brief History of AI

1997: Deep Blue IBM chess computer beats Garry Kasparov (Chess Grandmaster)

2011: IBM-Watson Defeats Humans in "Jeopardy!"

2016: Google's AlphaGo (Developed by DeepMind) beats Go master Lee Se-dol.
Full Documentary: AlphaGo – The Movie

Artificial Intelligence: New Trends
<Large Language Models (LLMs)>



Large Language Models (LLMs)

Fig. The Transformer - Model Architecture.
Paper Link: [Attention is All you Need]

Artificial Intelligence: New Trends
<Text to Image Models>

Source: Midjourney.com

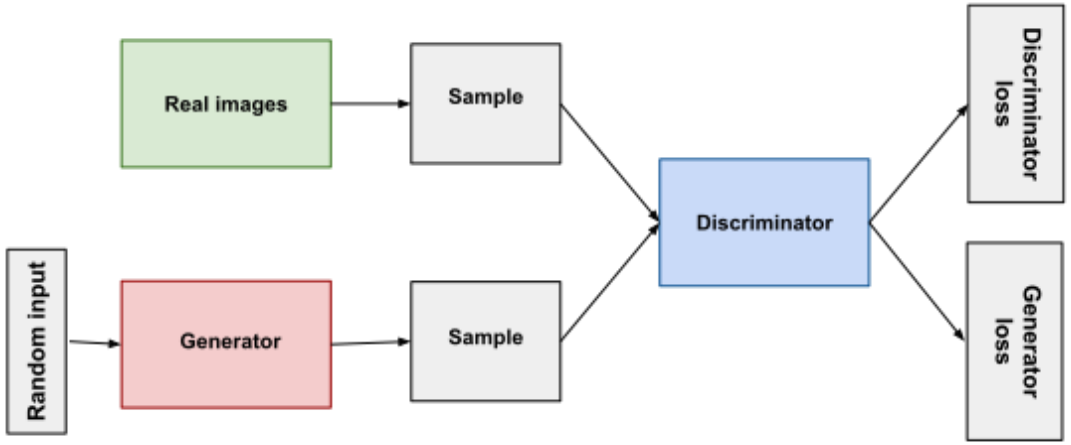
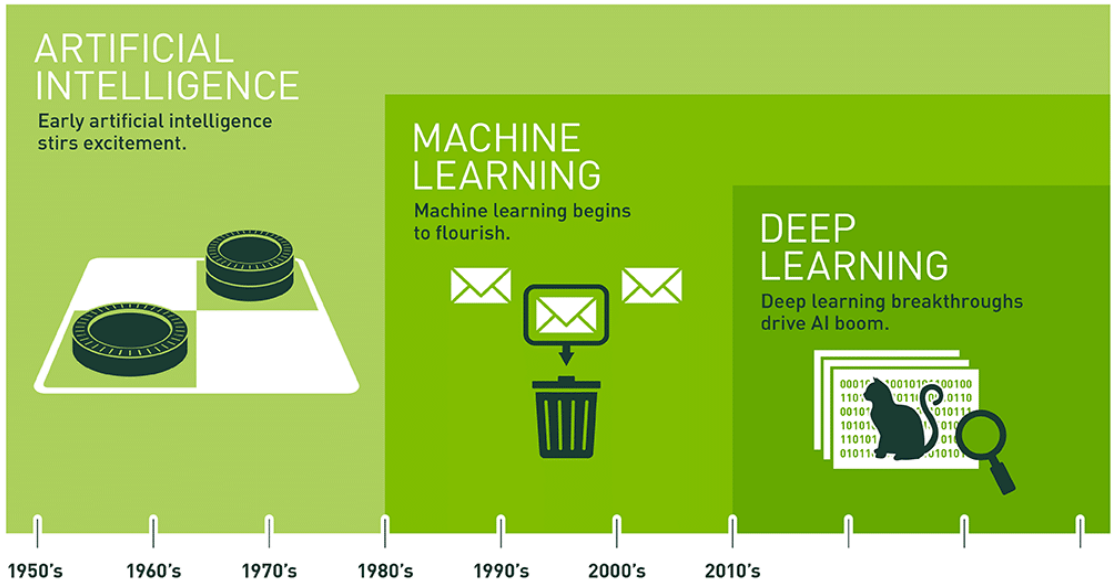


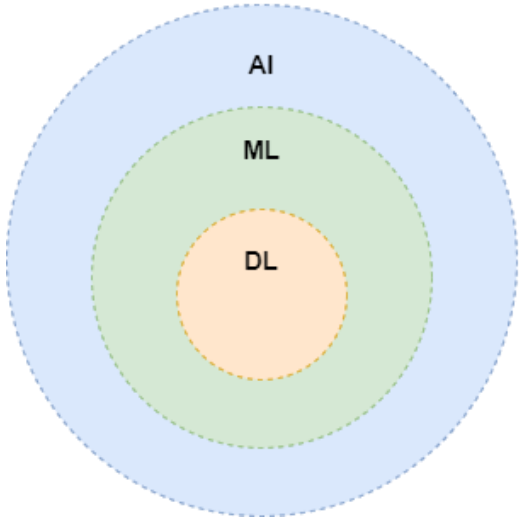
Fig. Overview of Generative Adversarial Networks (GANs)

Source: DALL-E 2

Artificial Intelligence, Machine Learning, and Deep Learning.
<What is the difference?>



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.



Artificial Intelligence

Class of problems we can solve when computers think/act like humans

Machine Learning

The science of getting computers to learn without being explicitly programmed. Data + Algorithms

Deep Learning

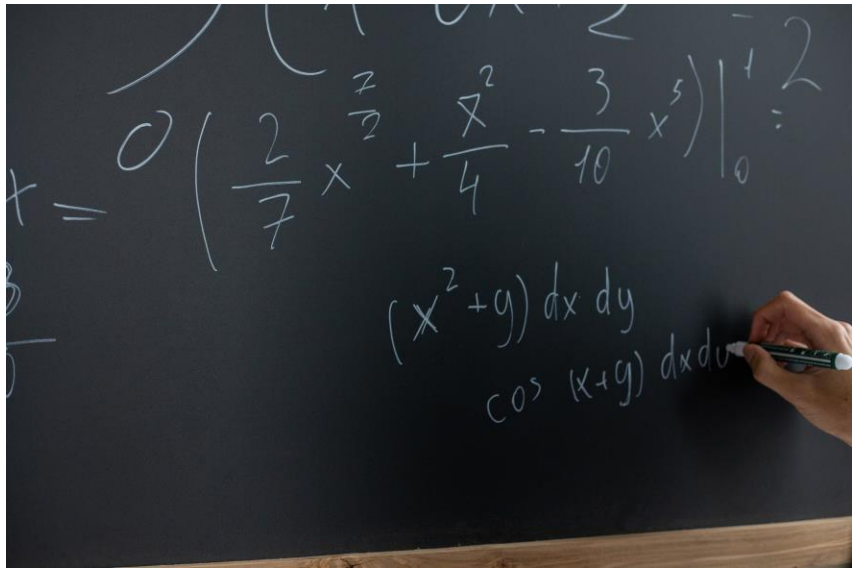
Subset of machine learning that is based on neural networks to mimic the human brain. Generally outperforms classical machine learning on unstructured data

Source: blogs.nvidia.com

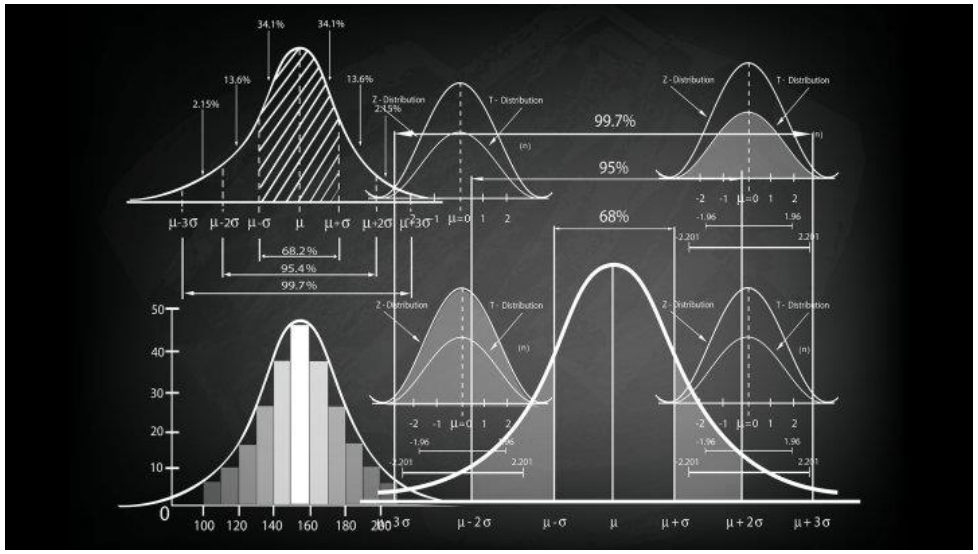
Introduction: The Importance of Mathematics

Mathematics are essential for understanding and working with machine learning algorithms. These include:

1. **Linear Algebra:**
   - Vectors and Matrices: Vectors and matrices are fundamental for representing and manipulating data. They are used extensively in tasks like transformations, regression, and neural networks.
   - Matrix Operations: Operations like addition, multiplication, and inversion of matrices are essential for various machine learning algorithms.
2. **Calculus:**
   - Derivatives and Gradients: Calculus is crucial for optimization algorithms. Understanding derivatives allows for finding the optimal parameters of a model.
   - Integration: Useful in probability theory, which is essential for many machine learning models.

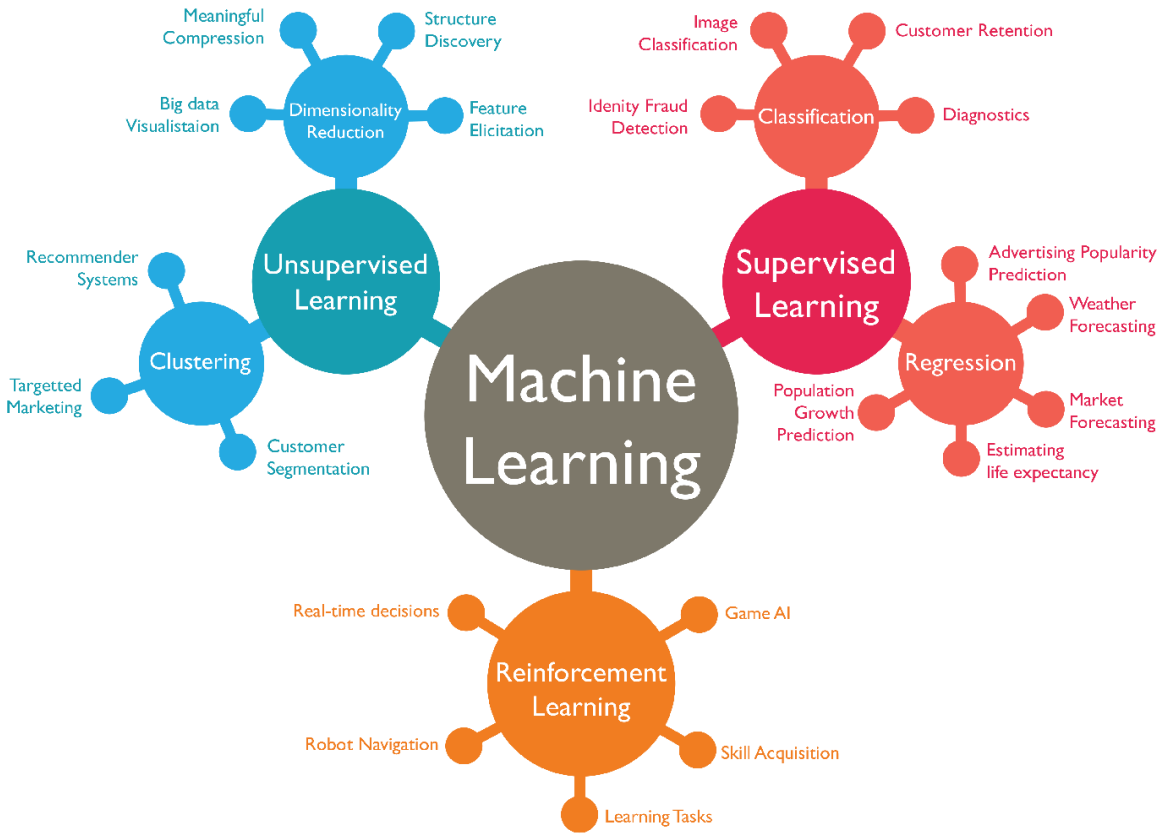


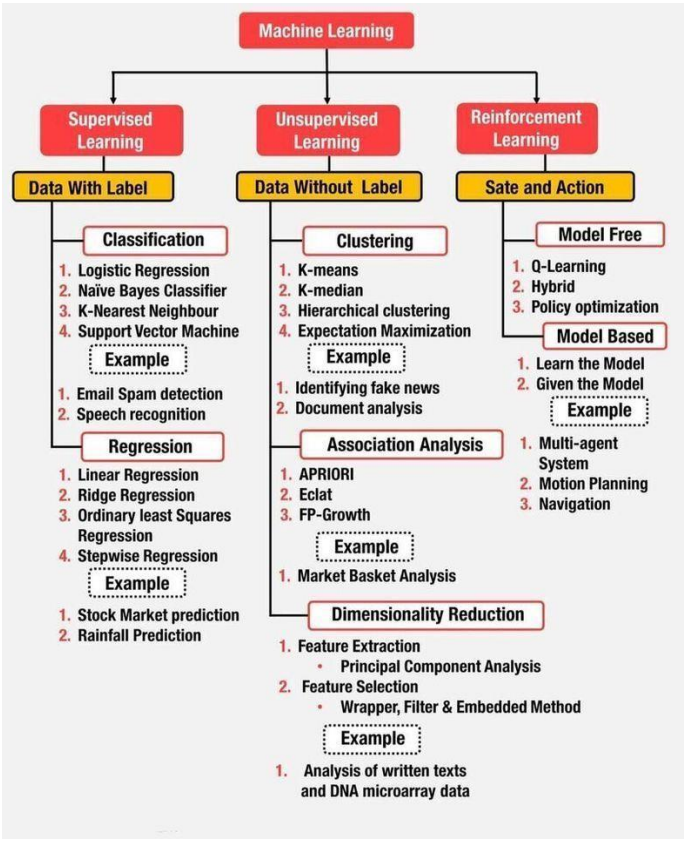
Introduction: The Importance of Mathematics



1. **Probability and Statistics:**
   - Probability Distributions: Understanding different probability distributions is crucial for modeling uncertainty and making predictions.
   - Bayesian Inference: It's used in Bayesian methods, which are important in areas like Bayesian networks and probabilistic programming.
   - Hypothesis Testing and Confidence Intervals: Used for evaluating the significance of results and estimating uncertainties.
2. **Optimization:**
   - Gradient Descent: A widely used optimization algorithm for training machine learning models.
   - Convex Optimization: Important for problems where the objective function is convex, which is common in machine learning.
3. **Information Theory:**
   - Entropy, Mutual Information: These concepts are used in feature selection, dimensionality reduction, and understanding the information content of data.

Machine Learning: Types of Machine Learning

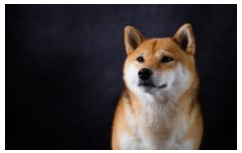




Machine Learning: Types of Machine Learning

Supervised Learning

Input Data: Labeled Data



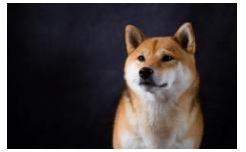
Label: Dog

Label: Cat

Label: Dog

Unsupervised Learning

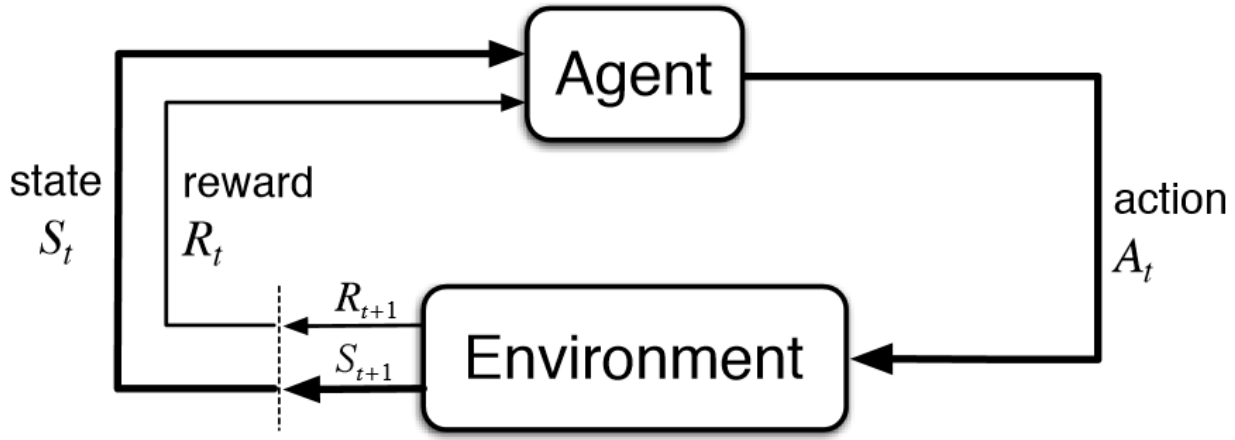
Input Data: Unlabeled Data



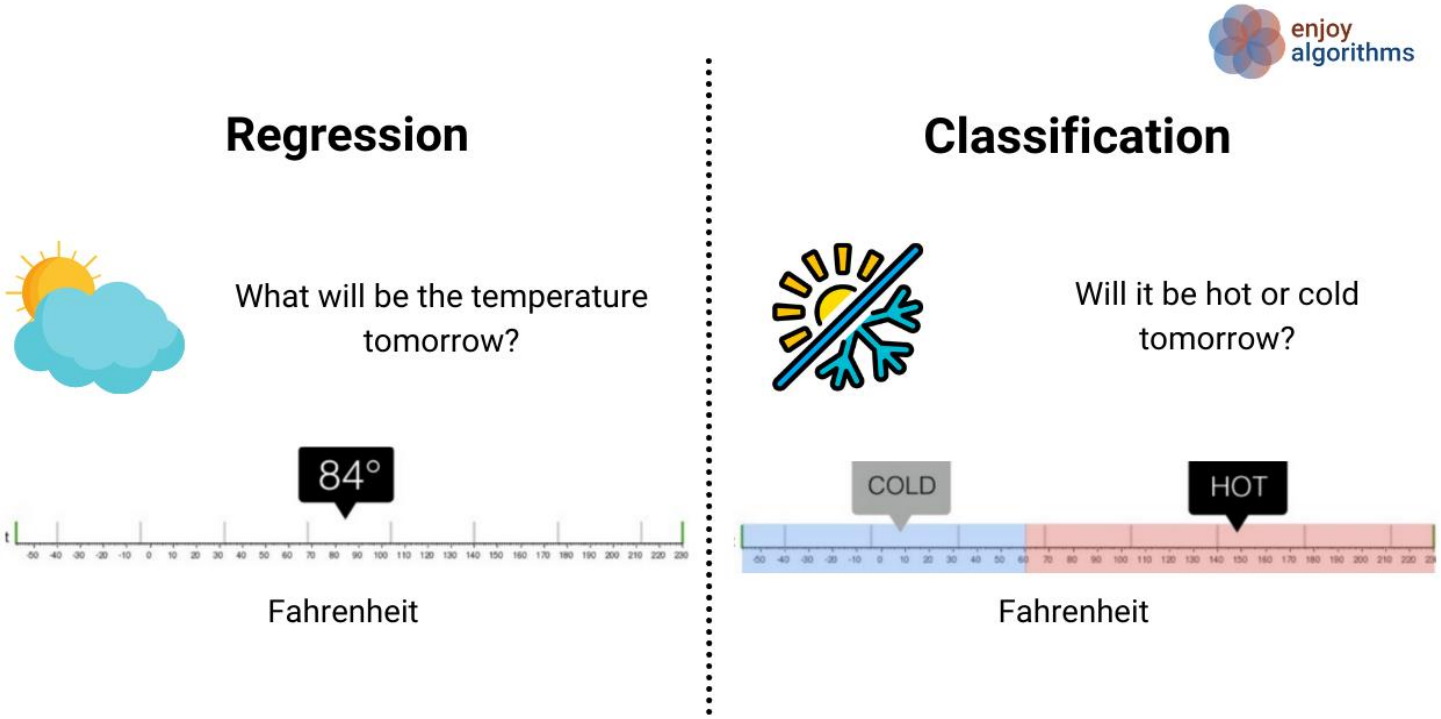
Machine Learning: Types of Machine Learning

Reinforcement Learning

The typical framing of a Reinforcement Learning (RL) scenario: an agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent.



Machine Learning: Types of Supervised Learning



Source: enjoyalgorithms.com

Machine Learning: Supervised Learning Algorithms

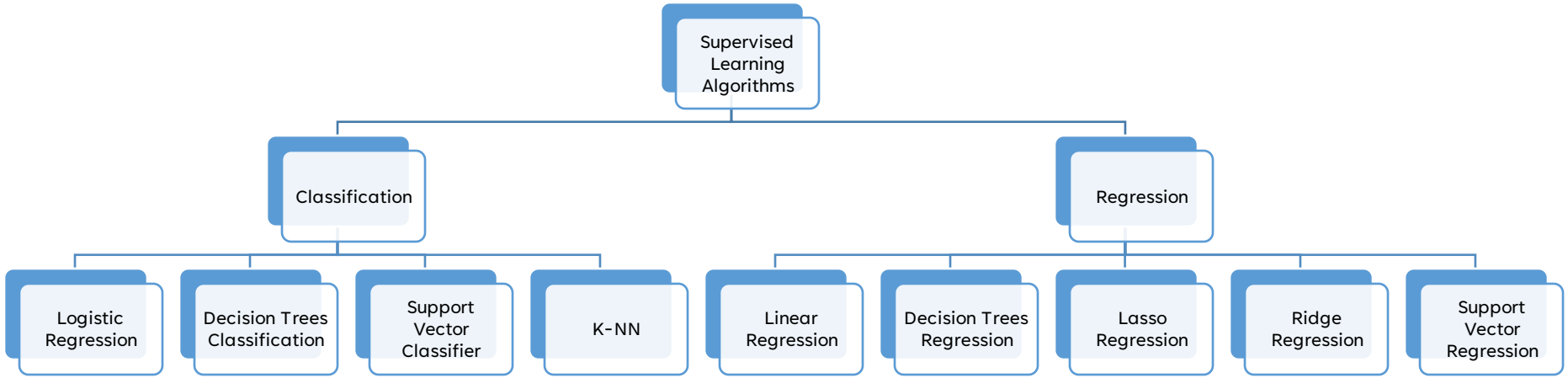
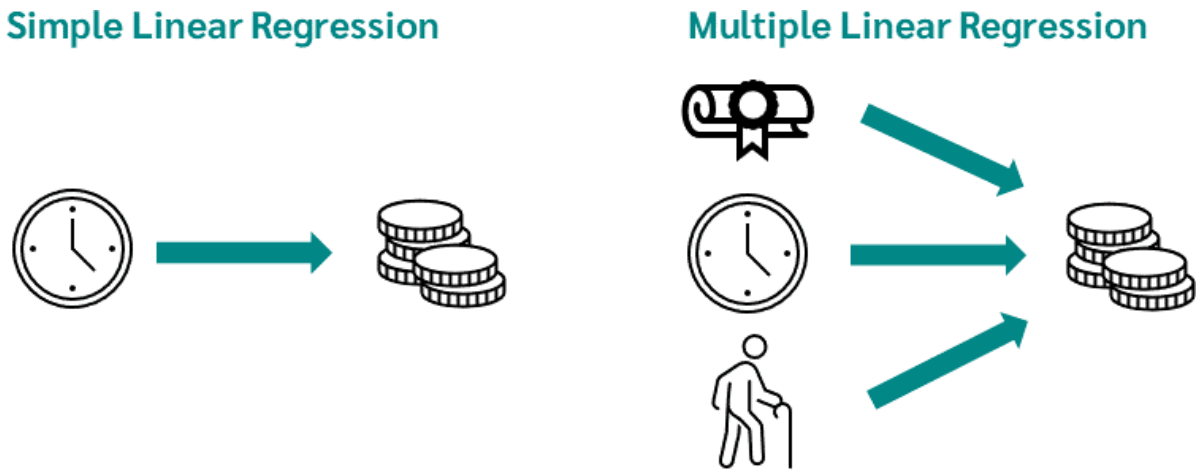


Fig. Some supervised learning algorithms

Supervised Learning Algorithms: Linear Regression

- Linear regression is a fundamental statistical and machine learning technique used to model the relationship between a dependent variable (target) and one or more independent variables (features)
- Depending on whether there are one or more independent variables, a distinction is made between simple and multiple linear regression analysis.
- The output **y** can be calculated from a linear combination of the input variables **X**



Supervised Learning: Linear Regression

$$\hat{y} = b \cdot x + a$$

Estimated dependent variable

Slope

Independent variable

y intercept

Mathematical model for simple linear regression

Simple Linear Regression

$$\hat{y} = b \cdot x + a$$

Multiple Linear Regression

$$\hat{y} = b_1 \cdot x_1 + b_2 \cdot x_2 + \ldots + b_k \cdot x_k + a$$

From simple linear regression to multiple linear regression

Supervised Learning: Linear Regression

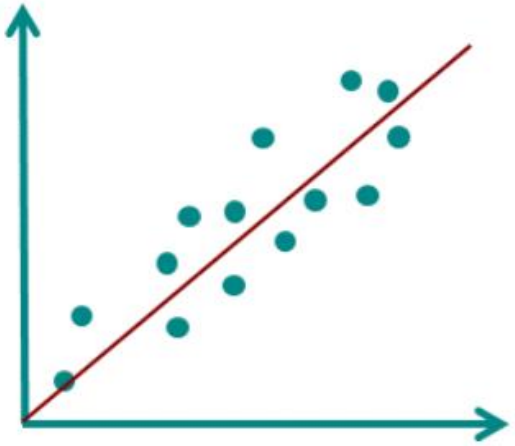
Simple Linear Regression

$$\hat{y} = b \cdot x + a$$

Multiple Linear Regression

$$\hat{y} = b_1 \cdot x_1 + b_2 \cdot x_2 + \ldots + b_k \cdot x_k + a$$

Question: How can we find the optimal values for **a** and **b** that fit the data?

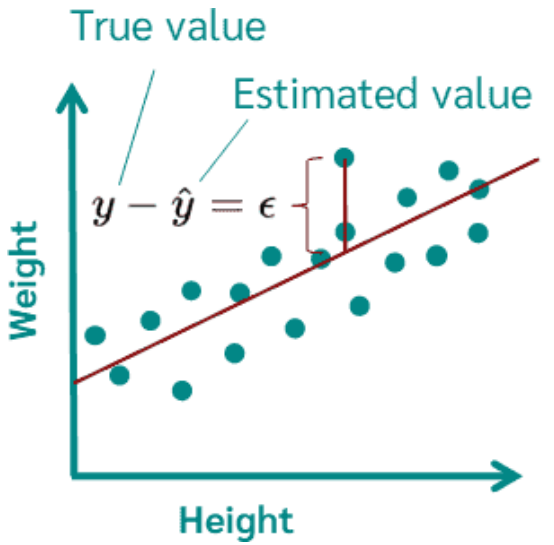


Supervised Learning: Linear Regression
<Ordinary Least Squares and Normal Equations>

Using a mathematical approach called **Least Squares (Ordinary Least Squares)** we can find the values of **a** and **b** that **minimizes** the **error epsilon ε**

$$a, b = \min_{a,b} \sum_{i=1}^{n} \hat{\epsilon}_i^2$$

$$\epsilon = y - \widehat{y} = y - (bX + a)$$



Error epsilon

$$y = b \cdot x + a + \epsilon$$

Example of simple linear regression

Supervised Learning: Linear Regression
<Ordinary Least Squares and Normal Equations>

Normal equations are **equations obtained by setting equal to zero the partial derivatives of the sum of squared errors** (least squares); normal equations allow one to estimate the parameters of a multiple linear regression.

$$LS = \sum_{i=1}^{n} (Y_i - \hat{Y})^2 = \sum_{i=1}^{n} (Y_i - a - bX_i)^2$$

Setting $\frac{\partial LS}{\partial a} = 0$ and $\frac{\partial LS}{\partial b} = 0$

$$a = \overline{Y} - b\overline{X} \qquad b = \frac{\sum_{i=1}^{n}(x_i - \overline{X})(y_i - \overline{Y})}{\sum_{i=1}^{n}(x_i - \overline{X})^2}$$

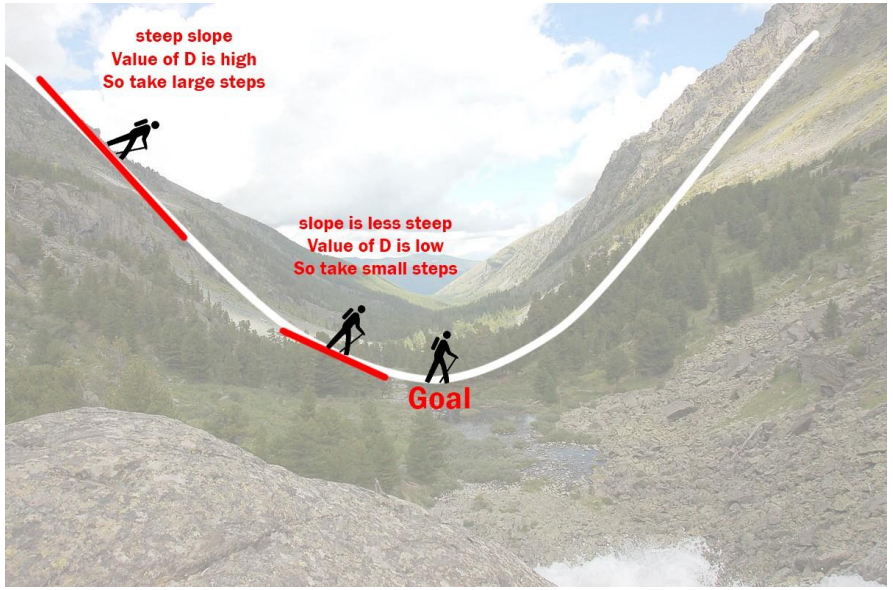
$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i, \quad \bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$$

If you cannot solve it by yourself, you can check this useful link: http://seismo.berkeley.edu/~kirchner/eps_120/Toolkits/Toolkit_10.pdf

Supervised Learning: Linear Regression
<Gradient Descent>

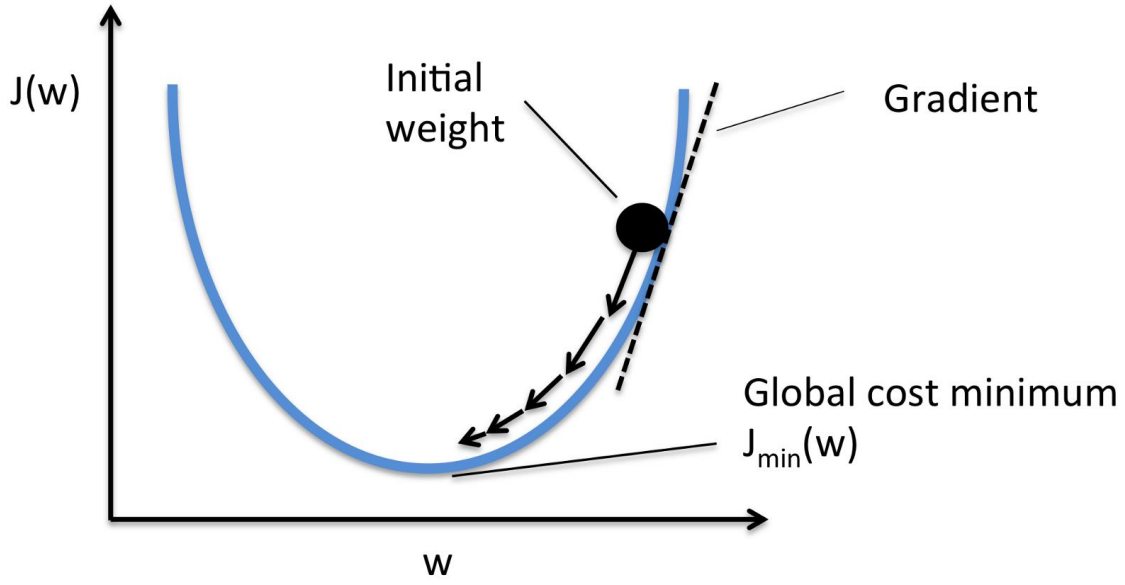
There is another alternative to the normal equations, using an optimization-based approach called **"Gradient Descent"**

- **Definition:** Gradient descent is an iterative optimization algorithm used for finding the minimum of a function. It's widely used in machine learning and deep learning for training models. The basic idea behind gradient descent is to take steps in the direction of steepest decrease in the function, i.e., in the direction of the negative gradient.
- **Objective:** Given a function f(x) that we want to minimize (Loss function/Cost function), gradient descent aims to find the value of x that minimizes f(x).



Gradient Descent: Iterative optimization algorithm

Supervised Learning: Linear Regression
<Gradient Descent>



Gradient Descent: Iterative optimization algorithm

Gradient: The gradient of a function f(x) is a vector that points in the direction of the steepest increase in the function. The negative gradient points in the direction of the steepest decrease.

$$w_i := w_i - \alpha \nabla J(w_i)$$

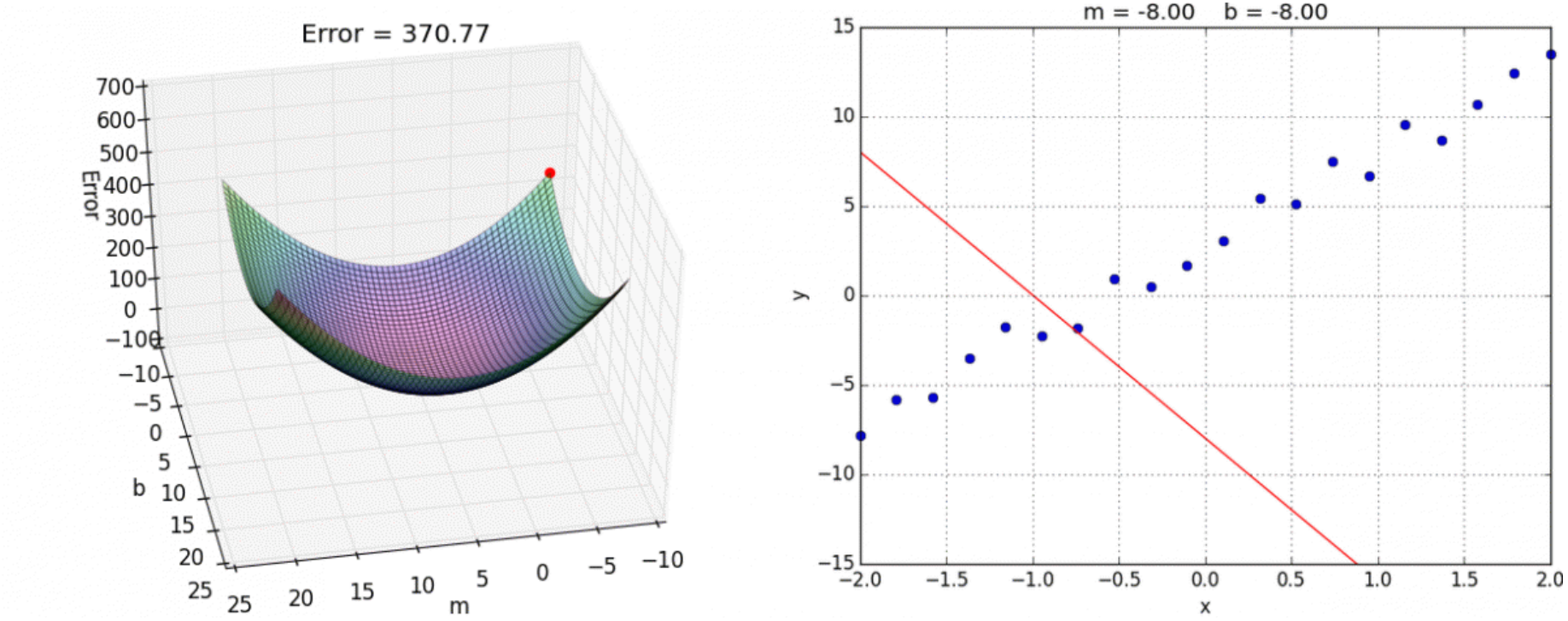
Supervised Learning: Linear Regression
<Gradient Descent>

$$\hat{\alpha}, \hat{\beta} = \min_{\alpha,\beta} \sum_{i=1}^{n} \hat{\varepsilon}_i^2 = \min_{\alpha,\beta} \sum_{i=1}^{n} (Y_i - \beta - \alpha X_i)^2$$

- **Step 1:** Start with a random initialization of the coefficients (α and β) (Set the values to 0 for example)
- **Step 2:** Repeat "K" times this "descending" using the learning rate alpha (but in order to avoid confusion α, we will name it gamma(γ))

$$\beta_k = \beta_{k-1} - \gamma \frac{1}{n} \sum_{i=1}^{n} (\beta_{k-1} + \alpha_{k-1} X_i - Y_i)$$
$$\alpha_k = \alpha_{k-1} - \gamma \frac{1}{n} \sum_{i=1}^{n} (\beta_{k-1} + \alpha_{k-1} X_i - Y_i) \cdot X_i$$

Supervised Learning: Linear Regression
<Gradient Descent>



Source: https://alykhantejani.github.io/a-brief-introduction-to-gradient-descent/

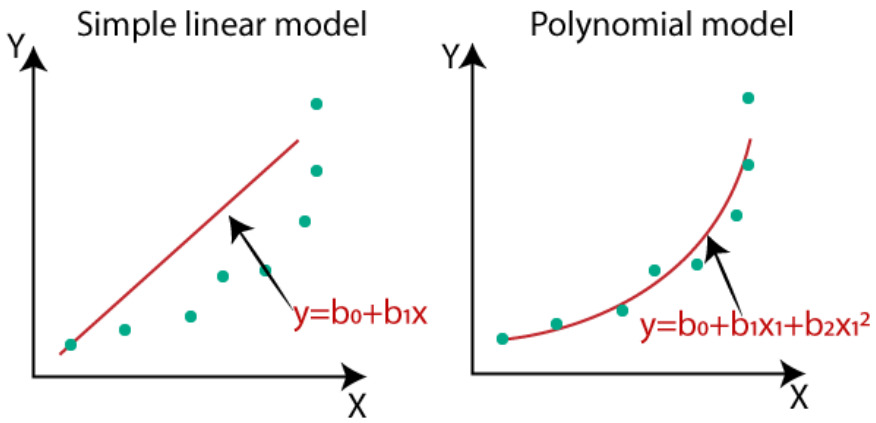
Supervised Learning: Linear Regression
<Gradient Descent>

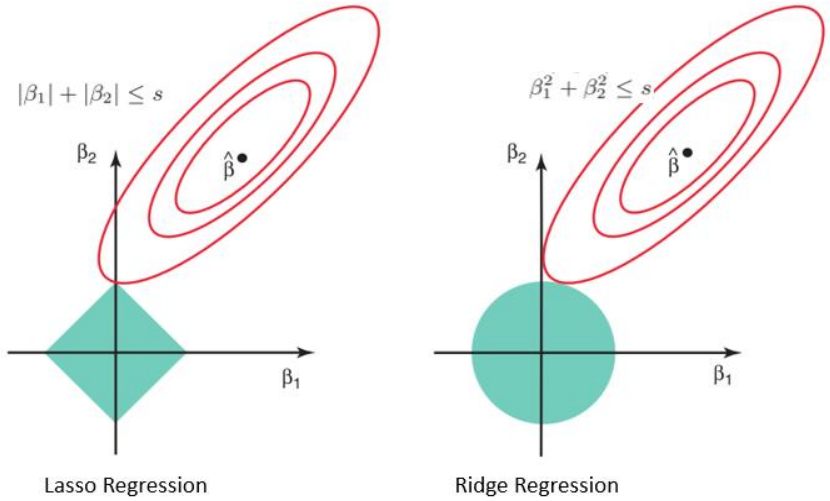
Gradient descent algorithm

$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$
$$(\text{for } j = 1 \text{ and } j = 0)$$
$$\}$$

- **Gradient:** The gradient of a function **f(x)** is a vector that points in the direction of the steepest increase in the function. The negative gradient points in the direction of the steepest decrease.

Supervised Learning: Advanced Types of Regression





Polynomial Regression: In which, we describe the relationship between the independent variable x and the dependent variable y using an nth-degree polynomial in x

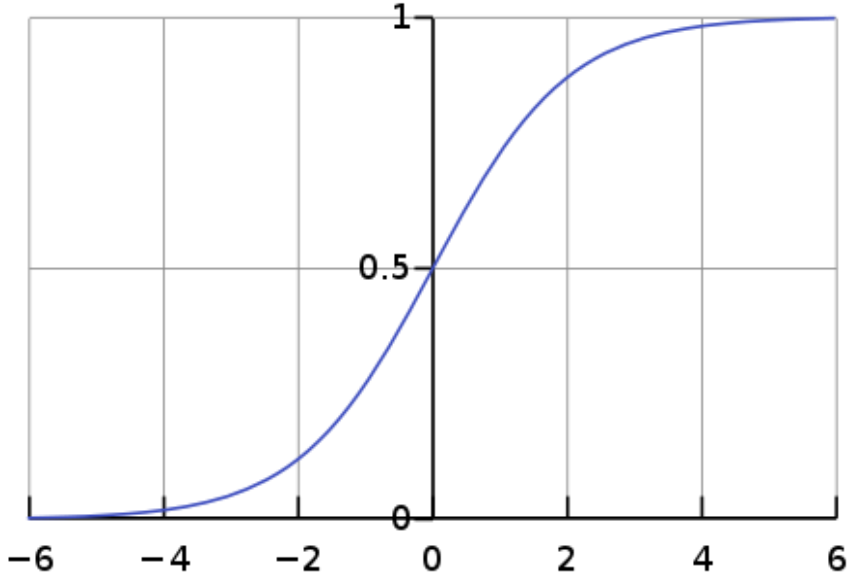
Lasso and **Ridge regression** are both techniques used in **regression analysis** to handle the problem of **overfitting** and to improve the **generalization** of the model. They do this by **adding** a **penalty** term to the standard linear regression **cost function**.

Supervised Learning: Linear Regression <DEMO>

DEMO: Session 1 – Linear Regression

Supervised Learning Algorithms: Logistic Regression

- **Logistic regression** is a statistical model used for analyzing datasets where there are one or more **independent variables (features)** that can be used to predict the outcome of a categorical **dependent variable (target)**.
- **Binary outcome:** It's particularly useful when the dependent variable is binary, meaning it has only two possible outcomes (e.g., 0 or 1, yes or no, true or false).
    - **Note:** For **multi-class classification** tasks (i.e., more than two classes), logistic regression can be extended using techniques like **one-vs-all** (also known as **one-vs-rest**) or softmax regression.
- **Why there is the "regression" word in a classification algorithm?** Despite its name, logistic regression is primarily used for classification rather than regression. **Logistic regression** gives a continuous value of **P(Y=1)** for a given input **X**, which is later converted to **Y=0** or **Y=1** based on a threshold value.
- **Sigmoid function:** It is the core of logistic regression (also known as the logistic function) to model the relationship between the independent variables (features) and the probability of the outcome. The sigmoid function "**squashes**" the output to be between 0 and 1, which makes it suitable for representing probabilities. It is defined as follows:



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

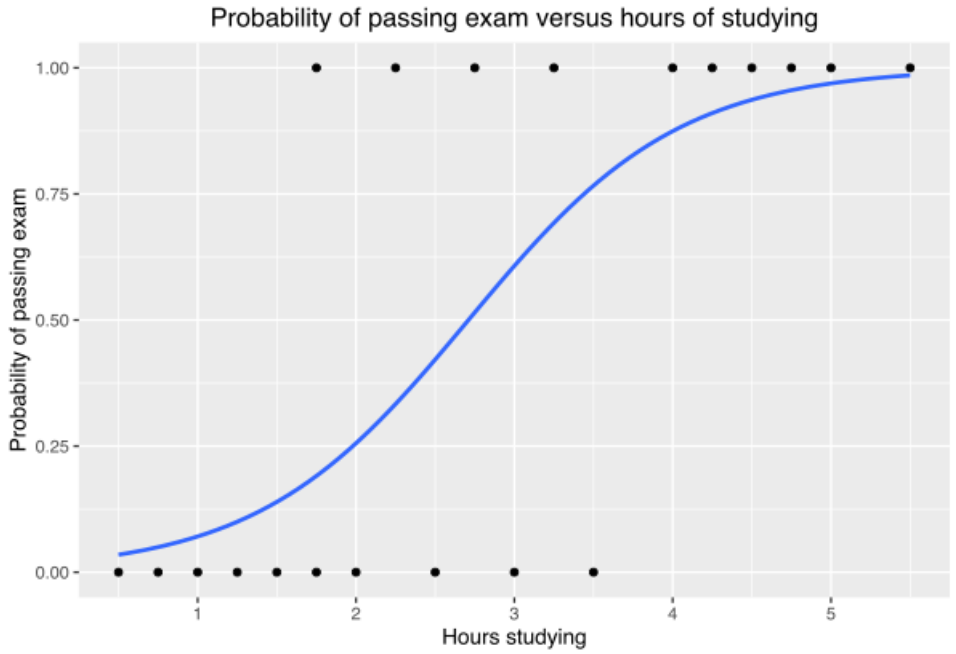
where "x" is the linear combination of the input features and their corresponding coefficients.

Supervised Learning Algorithms: Logistic Regression

Instead of fitting a line to the data (as in Linear Regression), Logistic Regression fits an “S” shaped logistic function called the Sigmoid Function.

$$P(y = 1) = \frac{1}{1 + \exp(-z)}$$

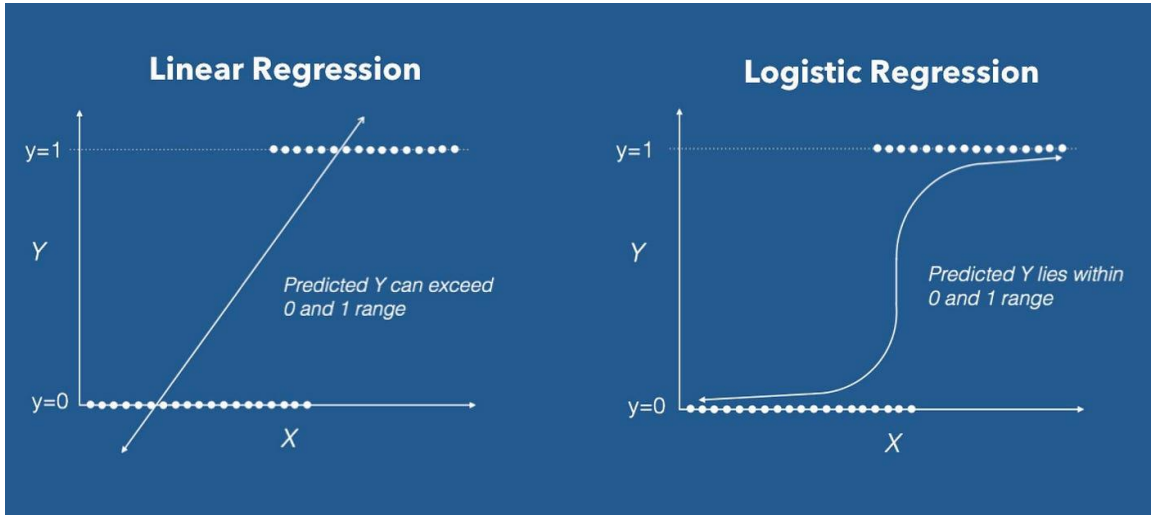
$$P(y = 0) = 1 - P(y = 1)$$



Supervised Learning Algorithms: Logistic Regression

The difference between **logistic regression** and **linear regression**

- One big difference between linear regression and logistic regression is how the line is fit to the data.
- In **linear regression**, we fit the line using least squares, In other words, we find the line that minimizes the sum of the squares of residuals (errors).
- **Logistic regression** does not have the same concept as residual, so it cannot use least squares. Thre is not a closed-form solution to solve the logistic regression
- **Logistic regression** uses something called "Maximum likelihood estimation" or "gradient descent", to estimate the coefficients of the logistic regression.

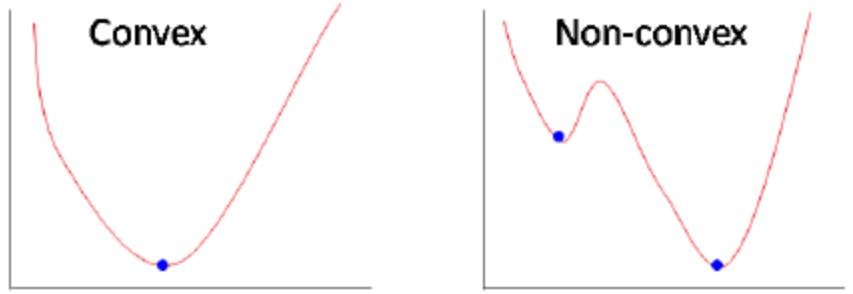


Note: For logistic regression, there is no longer a closed-form solution, due to the nonlinearity of the logistic sigmoid function.

Supervised Learning Algorithms: Logistic Regression
<How to train a logistic regression algorithm?>

Note: As usual, before training any machine learning algorithm we need to define a Loss/Cost Function.

As we have seen before, **linear regression** uses **Least Squared Error** as a loss function that gives a **convex** loss function and then we can complete the optimization by finding its vertex as a global minimum. However, for logistic regression, the hypothesis is changed, the Least Squared Error will result in a **non-convex** loss function with local minimums by calculating with the sigmoid function applied to the model's output.



Supervised Learning Algorithms: Logistic Regression
<How to train a logistic regression algorithm?>

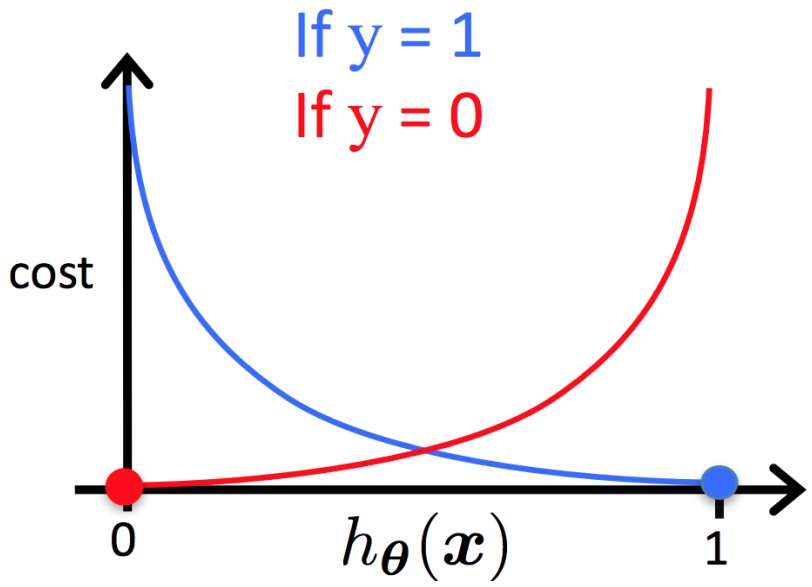
Note: As usual, before training any machine learning algorithm we need to define a Loss/Cost Function.

Due to the problem of non-convex if we apply the same loss function used for linear regression, therefore, the cost function for logistic regression is going to be defined as follows:

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Where,

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^T X)}$$



Supervised Learning Algorithms: Logistic Regression
<Gradient Descent>

As you can see, If y = 1 and *h(x)* = 1, the cost = 0. But if y = 1 and *h(x)* = 0, we will penalize the learning algorithm by a very large cost, cost = +*inf.*
The two defined functions for cost function can be compressed into a single function as follow:

$$J(\theta) = -\frac{1}{m}\sum\left[y^{(i)}\log(h\theta(x(i))) + \left(1-y^{(i)}\right)\log(1-h\theta(x(i)))\right]$$

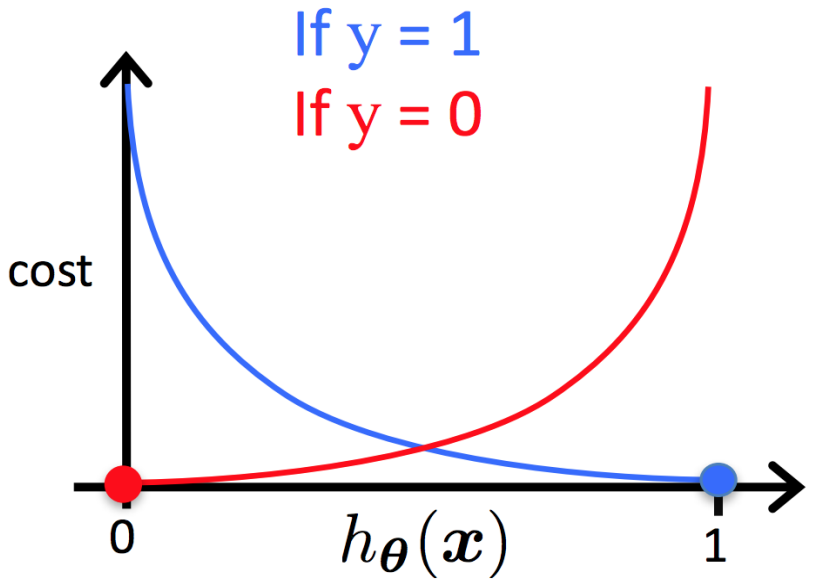
To minimize our cost function *J(θ)*, we are going to use the gradient descent algorithm.

Repeat {

$$\theta_j := \theta_j - \alpha\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

(simultaneously update all θ_j)

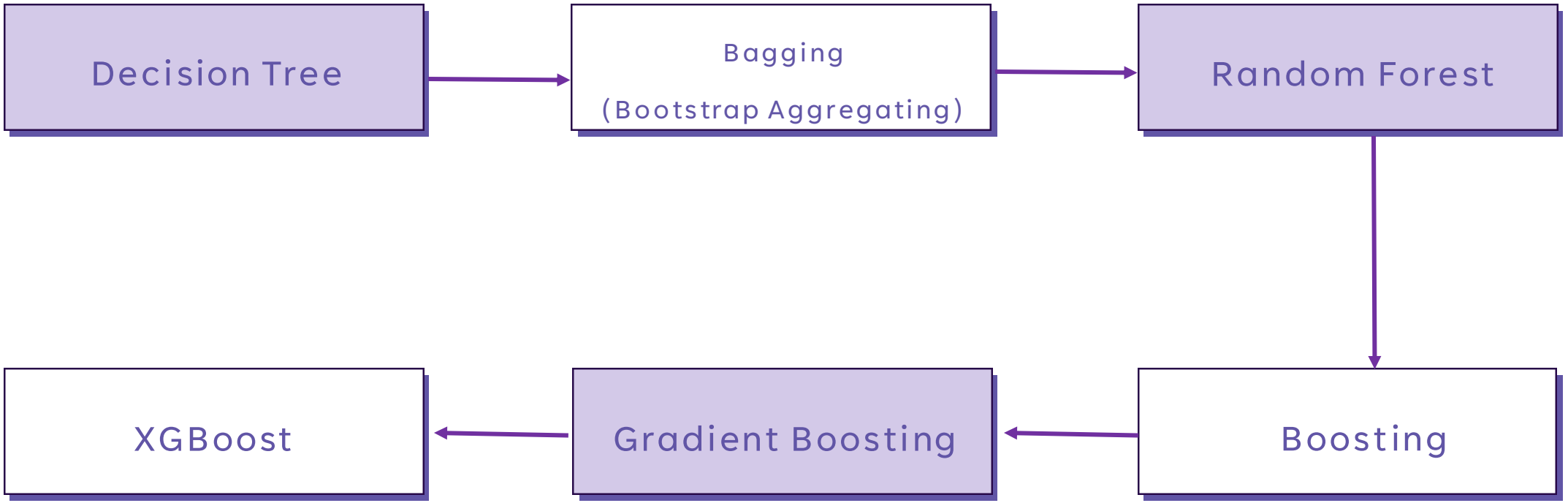
}



Supervised Learning: Logistic Regression <DEMO>

DEMO: Session 1 – Logistic Regression

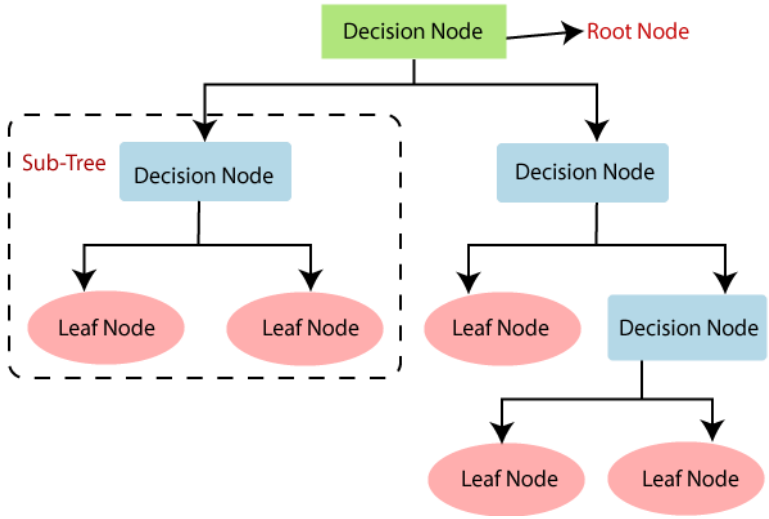
Supervised Learning Algorithms: Tree-Based Algorithms and Ensemble Methods



Supervised Learning Algorithms: Tree-Based Algorithms and Ensemble Methods

These methods can be used for both regression and classification problems.

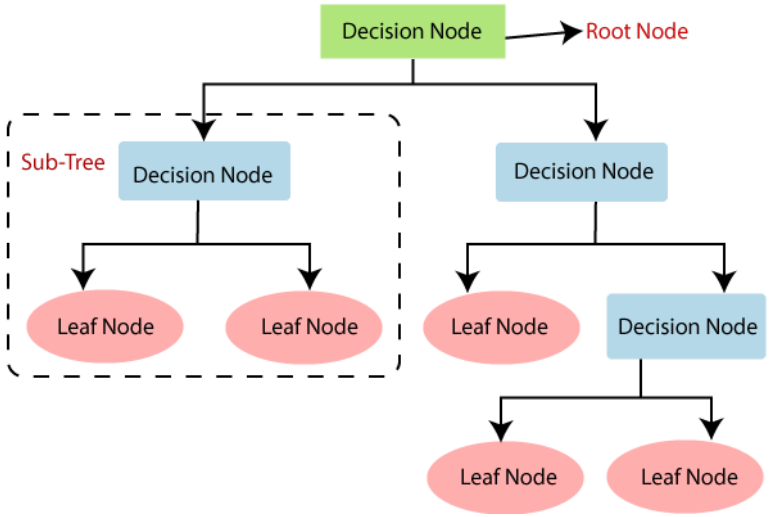
- **CART:** Classification and Regression Trees (CART), commonly known as decision trees, can be represented as **binary/non-binary trees**. They have the advantage to be very interpretable.
- **Bagging:** Bootstrap + Aggregating, is the ensemble technique used by random forest. Bagging chooses a random sample/random subset from the entire data set. Hence each model is generated from the samples (Bootstrap Samples) provided by the Original Data with replacement known as row sampling.
- **Random Forest:** It is a tree-based technique that uses a high number of decision trees built out of randomly selected sets of features. Contrary to the simple decision tree, it is highly uninterpretable, but its generally good performance makes it a popular algorithm.
- **Boosting:** The idea of boosting methods is to combine several weak learners to form a stronger one. The main ones are gradient boosting (e.g., XGBoost) and Adaptive boosting (AdaBoost)



Supervised Learning Algorithms: Decision Trees

Decision trees (also called Classification And Regression Trees "CART") are a type of machine learning algorithm that makes decisions by splitting data into subsets based on certain features. They are used for both classification (assigning a label to an item) and regression (predicting a numerical value).

1. Structure: Imagine a flowchart where each node represents a decision based on a feature, and the branches represent the possible outcomes or further decisions.
2. How They Work:
    1. Root Node: This is the starting point where the entire dataset is. It's the feature that is most effective at splitting the data.
    2. Internal Nodes: These nodes split the data based on certain conditions (e.g., if a value is greater than a certain threshold).
    3. Leaf Nodes: These are the final nodes that do not split further. They provide the final prediction or classification.



Supervised Learning Algorithms: Decision Trees <Splitting Criteria>

Question: How can we choose the best decision nodes?
Answer: Using splitting criteria, the model chooses the best features to split the data based on some criteria (e.g., Gini impurity for classification, Mean Squared Error for regression).
Splitting Criteria:

- **For classification tasks:** Gini Impurity, Information Gain, Entropy, Gini Gain. (and more)
- **For regression tasks:** Mean squared error, mean absolute error, and variance reduction.

Supervised Learning Algorithms: Decision Trees for Classification

1. **Gini Impurity:**
   - Gini impurity is a measure of how mixed the classes are in a given dataset or subset of data.
   - It ranges from 0 to 0.5, where 0 indicates that a node contains only samples of a single class, and 0.5 indicates that the samples are evenly distributed among the classes.
   - The decision tree algorithm aims to minimize the Gini impurity when making splits.

The formula for Gini impurity for a node is:

$$Gini(t) = 1 - \sum_{i=1}^{c} (p_i)^2$$

2. **Entropy:**
   - Entropy measures the disorder or uncertainty in a set of data.
   - Like Gini impurity, it also ranges from 0 to 1, with 0 indicating perfect order and 1 indicating maximum disorder.
   - Decision trees aim to minimize entropy when making splits.

The formula for entropy for a node is:

$$Entropy(t) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

Supervised Learning Algorithms: Decision Trees for Classification

1. **Information Gain:**
   - Information Gain is used to determine the best feature to split the data at each node in a decision tree.
   - It quantifies how much information a feature gives us about the classes.
2. **Gini Gain:**
   - Gini Gain is similar to Information Gain, but it uses Gini impurity as the measure instead of entropy.
   - It's calculated similarly to Information Gain but using Gini impurity instead of entropy.

Information gain is calculated as the difference between the entropy (or Gini impurity) before and after the split:

$$IG(D, A) = H(D) - \sum_{v \in Values(A)} \frac{|D_v|}{|D|} H(D_v)$$

where D is the dataset at the current node, A is the feature being considered for the split, D_v is the subset of D for which feature A has value v, and H represents entropy or Gini impurity.

Supervised Learning Algorithms: Decision Trees for Regression

1. **Mean Squared Error (MSE):**
   - The MSE measures the average squared difference between predicted and actual values. It quantifies the accuracy of a regression model.
   - In the context of decision trees, the MSE is used to evaluate the quality of a split. The split that minimizes the MSE is chosen.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

2. **Mean Absolute Error (MAE):**
   - The MAE is similar to MSE but measures the average absolute difference between predicted and actual values, rather than squared differences.
   - MAE is less sensitive to outliers compared to MSE.

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Supervised Learning Algorithms: Decision Trees for Regression

Variance Reduction:

- In regression tasks, the goal is often to minimize the variance of the target variable within each node.
- The variance reduction is used as the splitting criterion, and it measures how much the variance of the target variable is reduced after the split.
- The formula for variance reduction is specific to regression tasks and involves the calculation of variances.
- The formula for variance reduction in the context of regression trees is as follows:
  - Given a node with a dataset **D** containing **n** samples, where $\mathbf{y_i}$ represents the target variable values for each sample, the variance reduction is calculated as:

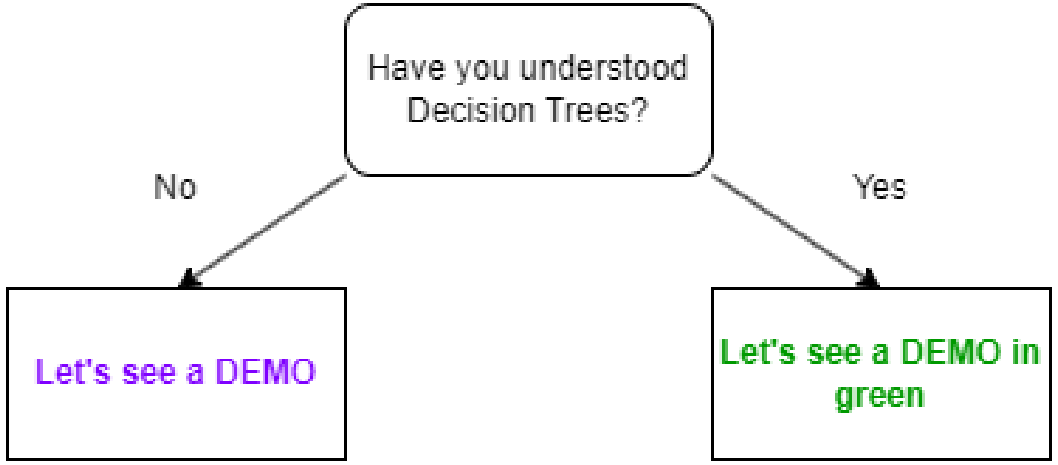
$$\text{Variance Reduction}(D) = \text{Var}(D) - \sum_{i=1}^{m} \frac{|D_i|}{|D|} \times \text{Var}(D_i)$$

$$\text{Var}(D) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \bar{y})^2$$

- m is the number of child nodes resulting from the split.
- Di represents the dataset in the i-th child node after the split.
- |Di | is the number of samples in the i-th child node.
- Var(D) is the variance of the target variable in node D
- y-bar is the mean of the target variable values in node D

Supervised Learning Algorithms: Decision Trees <DEMO>

DEMO: Session 1 – Decision Trees



Supervised Learning Algorithms: K-Nearest Neighbors

k-Nearest Neighbors (k-NN) is a **supervised** machine learning algorithm used for both **classification** and **regression** tasks. It's a simple and intuitive algorithm that relies on the idea of finding the "**nearest**" data points in the training set to a given test point and using those neighbors to make a prediction.

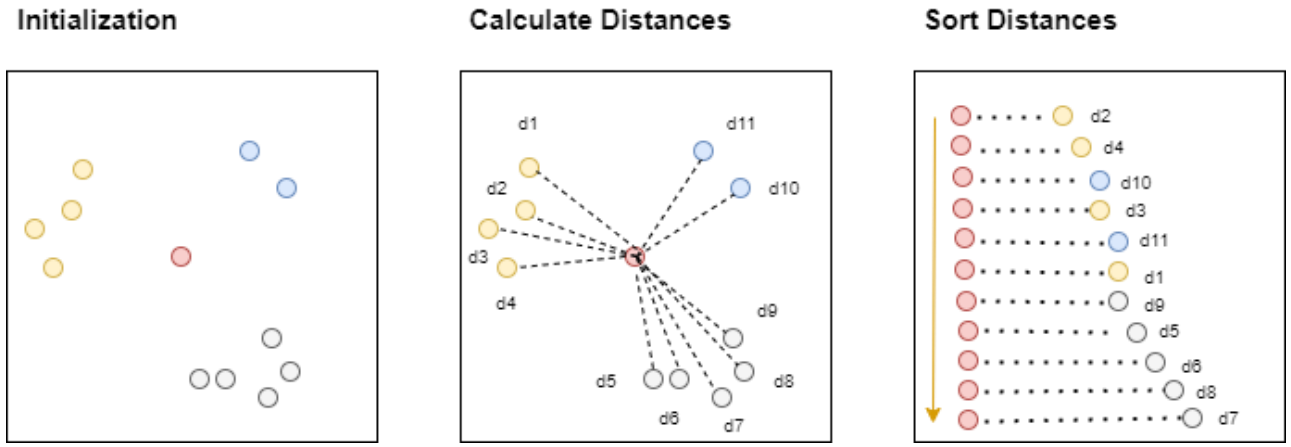


Fig. K-NN Algorithm Steps

Supervised Learning Algorithms: K-Nearest Neighbors

1. **Training Phase:**
   - Store the entire dataset in memory.
   - No explicit training is done, as k-NN is a lazy learner.
2. **Prediction Phase:**
   - When given a new, unseen data point, the algorithm finds the k data points in the training set that are closest (most similar) to the new point.
   - The "closeness" is typically determined by a distance metric like Euclidean distance, Manhattan distance, etc.
3. **Classification:**
   - In the classification task, the algorithm assigns a class label to the new point based on the majority class among its k nearest neighbors.
4. **Regression:**
   - In regression, the algorithm predicts a continuous value for the new point based on the average (or some other measure) of the target values of its k nearest neighbors.

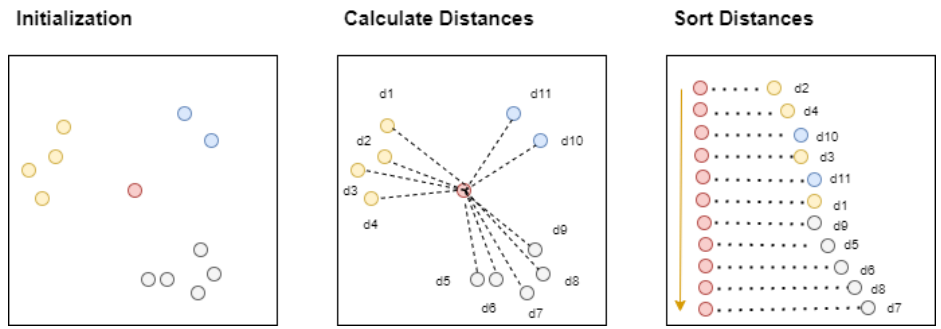


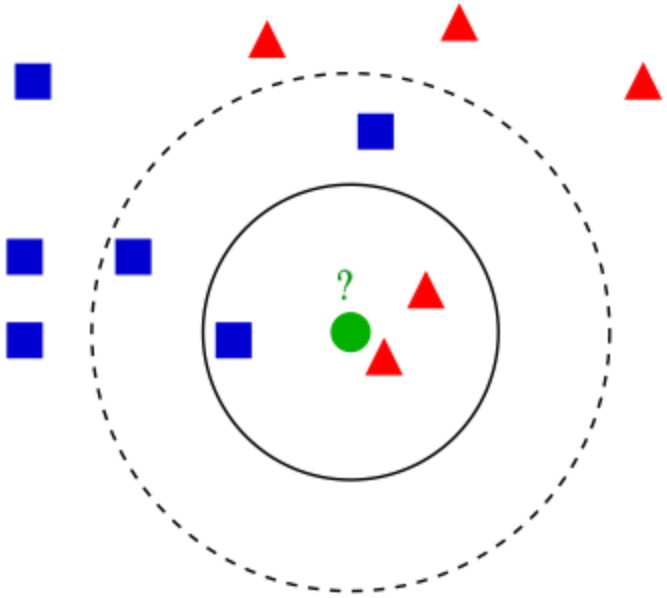
Fig. K-NN Algorithm Steps

Supervised Learning Algorithms: K-Nearest Neighbors

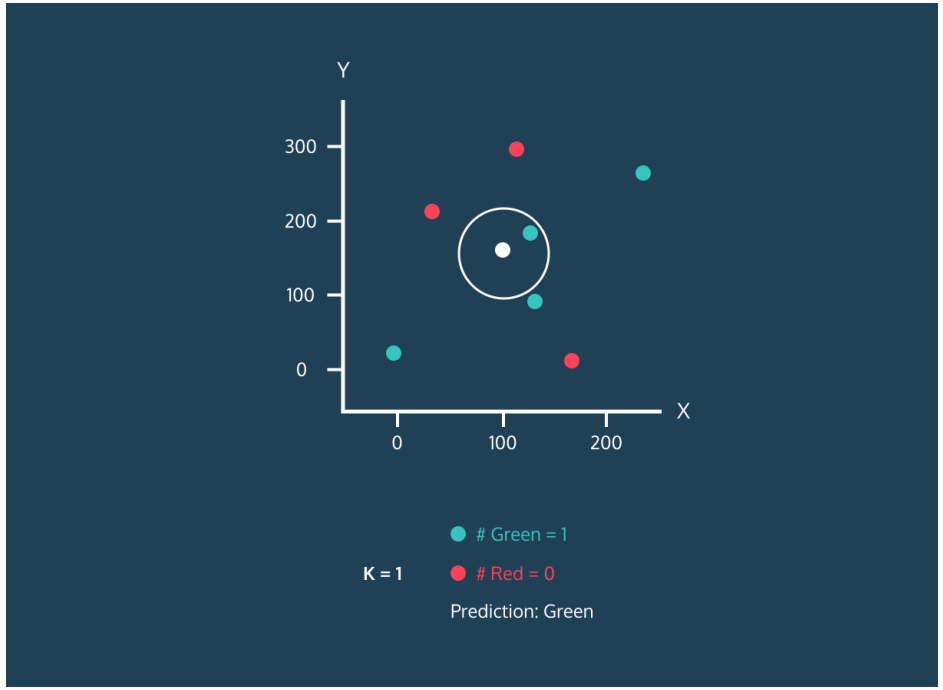
1. **Key Parameters:**
   - k: This is the number of neighbors to consider. It's a hyperparameter that you need to choose. A small k might lead to noise in the prediction, while a large k might smooth out the decision boundaries.
   - Distance Metric: This defines how the "closeness" of data points is calculated. Common options include Euclidean distance, Manhattan distance, Minkowski distance, etc.
2. **Advantages:**
   - Simple to understand and implement.
   - No explicit training phase, making it computationally efficient during training.
   - Can be used for both classification and regression tasks.
3. **Disadvantages:**
   - Can be computationally expensive during prediction, especially with large datasets.
   - Sensitive to the choice of distance metric and value of k.
   - Doesn't learn underlying patterns in the data, which can lead to suboptimal performance in some cases.
4. **Considerations:**
   - Scaling of features is crucial as k-NN is sensitive to the scale of the variables.
   - It's important to choose an appropriate value of k through techniques like cross-validation.

Supervised Learning Algorithms: K-Nearest Neighbors <Example>

Example of *k*-NN classification. The test sample (green dot) should be classified either to blue squares or to red triangles. If *k* = *3* (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If *k* = *5* (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle).



Supervised Learning Algorithms: K-Nearest Neighbors <DEMO>



DEMO: Session 1 - k-Nearest Neighbors (k-NN)

Supervised Learning Algorithms: Commonly Used Regression Algorithms

Regression Algorithms:

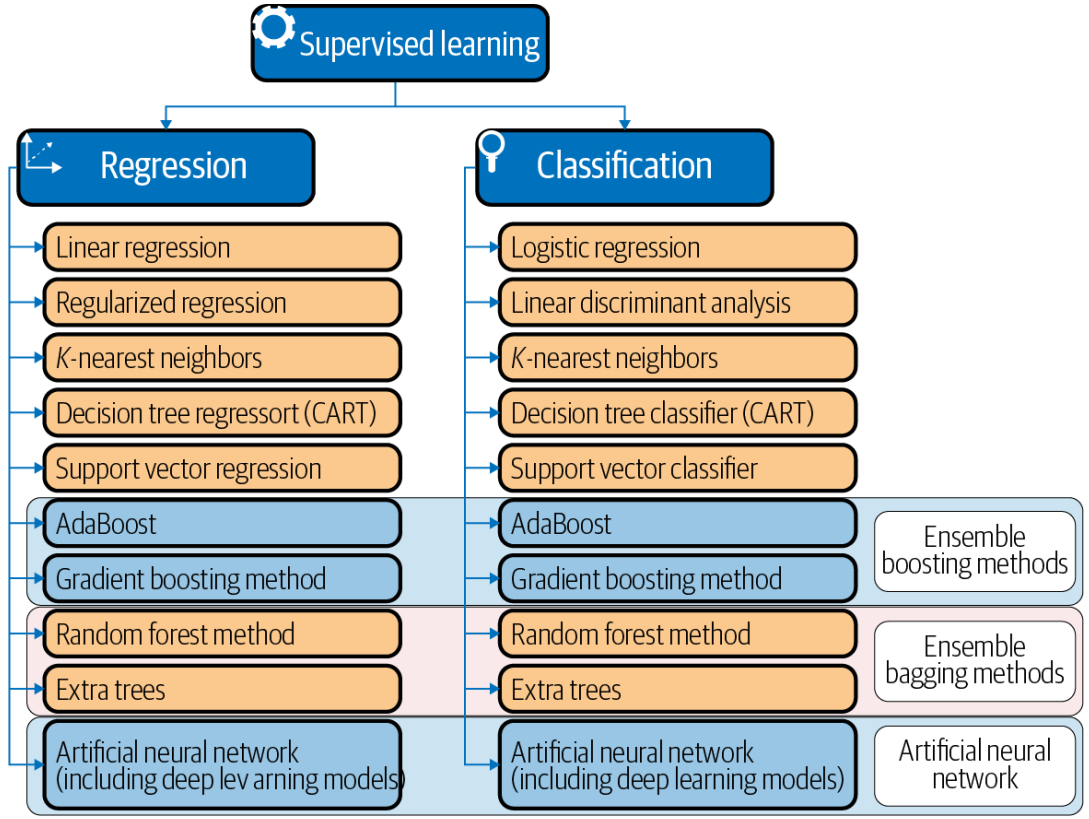
1. **Linear Regression**: Predicts a continuous output variable based on the input features by fitting a linear equation to the observed data.
2. **Ridge Regression (L2 regularization)**: Similar to linear regression but adds a penalty term to the coefficients to prevent overfitting.
3. **Lasso Regression (L1 regularization)**: Similar to ridge regression, but uses the absolute values of coefficients, which can lead to sparsity in the model.
4. **ElasticNet**: A combination of L1 and L2 regularization that balances between Ridge and Lasso regression.
5. **Decision Tree Regression**: Uses a decision tree to predict continuous values.
6. **Random Forest Regression**: Ensemble method that uses multiple decision trees to improve accuracy and control overfitting.
7. **Gradient Boosting Regression**: Builds an additive model in a forward stage-wise manner, optimizing for the residual errors.
8. **Support Vector Regression (SVR)**: Uses support vector machines to perform regression.

Supervised Learning Algorithms: Commonly Used Classification Algorithms

Classification Algorithms:

1. **Logistic Regression**: Used for binary classification problems, models the probability that a given instance belongs to a particular category.
2. **K-Nearest Neighbors (KNN)**: Classifies data points based on the majority class of their k-nearest neighbors.
3. **Support Vector Machines (SVM)**: Finds the hyperplane that best separates classes in a high-dimensional space.
4. **Naive Bayes**: Applies Bayes' theorem with the "naive" assumption that features are independent, commonly used for text classification.
5. **Decision Tree Classification**: Divides the data into subsets based on the value of features to make categorical predictions.
6. **Random Forest Classification**: Ensemble method that uses multiple decision trees for classification.
7. **Gradient Boosting Classification**: Builds an ensemble of weak learners (usually decision trees) in a forward stage-wise manner.
8. **Neural Networks (Deep Learning)**: Multi-layered networks of interconnected nodes used for complex classification tasks.
9. **XGBoost, LightGBM, CatBoost** (Gradient Boosting variations): Highly optimized implementations of gradient boosting algorithms.
10. **Adaboost**: Combines multiple weak learners to create a strong learner.

Supervised Learning Algorithms: Classification and Regression Algorithms



Source: www.oreilly.com

Supervised Learning Algorithms: Parametric vs. Nonparametric

There are two main types of machine learning algorithms: **parametric** and **nonparametric**.

Parametric Algorithms

Parametric algorithms are based on a mathematical model that defines the relationship between inputs and outputs. This makes them more restrictive than nonparametric algorithms, but it also makes them faster and easier to train. Parametric algorithms are most appropriate for problems where the input data is well-defined and predictable.

Examples of parametric algorithms: Linear regression, Logistic regression, and Neural networks.

Nonparametric Algorithms

Nonparametric algorithms are not based on a mathematical model; instead, they learn from the data itself. This makes them more flexible than parametric algorithms but also more computationally expensive. Nonparametric algorithms are most appropriate for problems where the input data is not well-defined or is too complex to be modeled using a parametric algorithm.

Examples of nonparametric algorithms: Decision trees, K-NN.

Supervised Learning: Evaluation Metrics
<Classification Metrics>

Question: When dealing with a classification problem, how can we **evaluate** its **effectiveness** and thus conduct a **comparative** study with other **classifiers**?

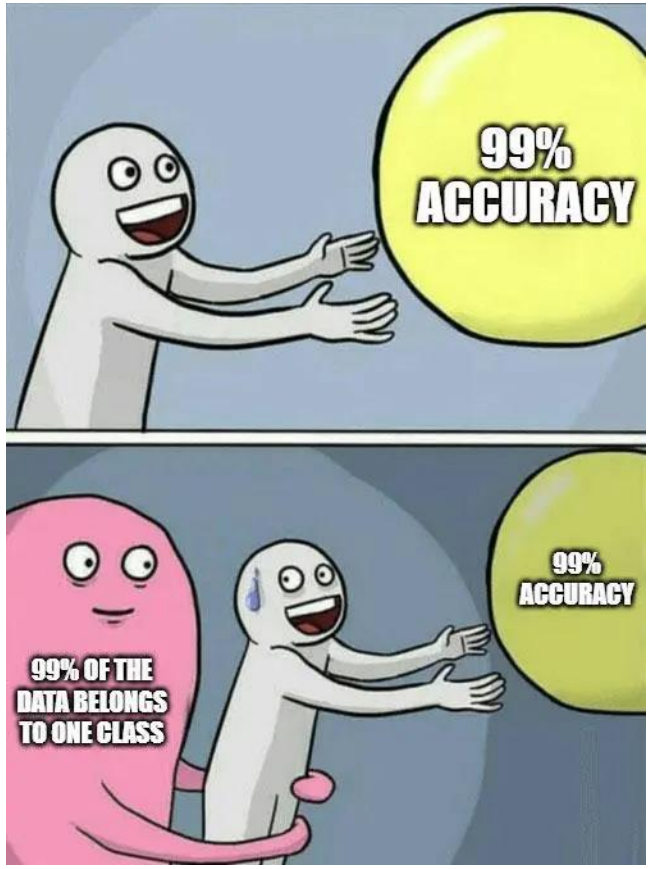
Answer: We calculate the correct classified data points and divide it by the total number of data points in the dataset.

$$accuracy = \frac{Correctly\ classified\ datapoints}{Total\ number\ of\ all\ datapoints}$$

Another Question: Is the accuracy metric enough? Are there potential problems with it?

Short Answer: No, It is not enough and Yes, There is potential problems

Supervised Learning: Evaluation Metrics
<Classification Metrics>



Supervised Learning: Evaluation Metrics <Classification Metrics>

Some potential **problems** with relying only on **accuracy**

Problem	Description	Examples
Imbalanced Datasets	In datasets where one class significantly outweighs the others (class imbalance), a model can achieve high accuracy by simply predicting the majority class most of the time.	In a medical dataset where only 5% of patients have a rare disease, a model that always predicts "no disease" would still achieve 95% accuracy.
Misleading in Rare Events	In applications where detecting rare events is crucial (e.g., fraud detection, disease diagnosis), accuracy can be misleading. A high accuracy score may not reflect the model's ability to correctly identify these critical cases. Accuracy may not reflect the model's ability to detect rare, but important, occurrences.	In fraud detection, a model that misses rare instances of fraud (false negatives) could have high accuracy but would be ineffective.

Supervised Learning: Evaluation Metrics <Classification Metrics>

Question: What is the solution? Are there other alternatives to Accuracy?

Answer: There are other metrics that can be used: Recall/Sensitivity, Precision, Specificity, F-Score, ROC-AUC Curve...etc. **WAIT!!!**

Before explaining those metrics, let's first understand the confusing "**Confusion Matrix**"

Hold on to your laptops and let's see what is the Confusion Matrix

Supervised Learning: Evaluation Metrics
<Classification Metrics>

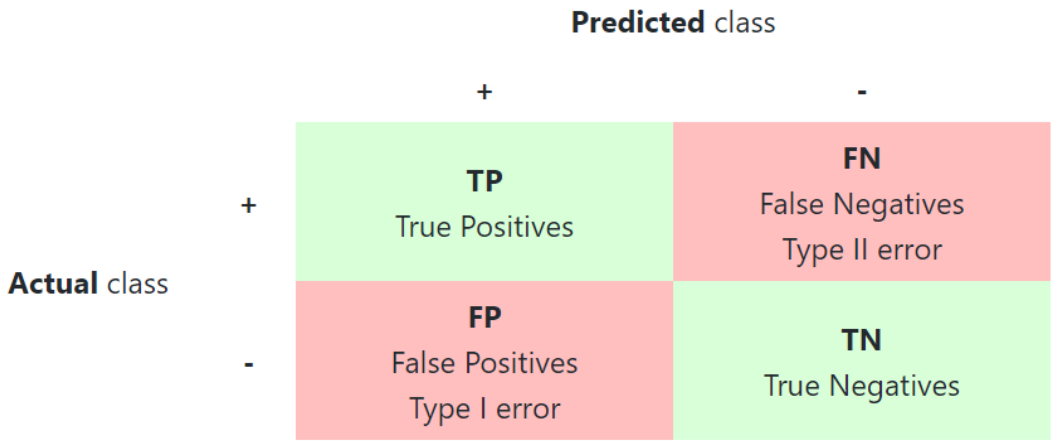


Fig. Confusion Matrix Layout

Metric	Formula	Interpretation
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Overall performance of model
Precision	$\frac{TP}{TP + FP}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{TP}{TP + FN}$	Coverage of actual positive sample
Specificity	$\frac{TN}{TN + FP}$	Coverage of actual negative sample
F1 score	$\frac{2TP}{2TP + FP + FN}$	Hybrid metric useful for unbalanced classes

Tab. Basic classification metrics

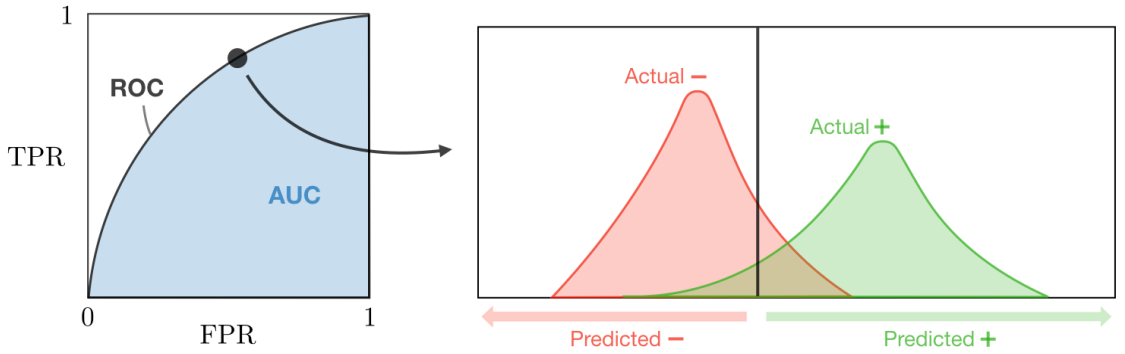
WANT SOME MORE METRICS? LET'S SEE ROC AND AUC CURVES

Supervised Learning: Evaluation Metrics
<Classification Metrics>

ROC: The receiver operating curve, also noted as ROC, is the plot of TPR versus FPR by varying the threshold. These metrics are summed up below:

Metric	Formula	Equivalent
True Positive Rate TPR	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$	Recall, sensitivity
False Positive Rate FPR	$\frac{\mathrm{FP}}{\mathrm{TN}+\mathrm{FP}}$	1-specificity

AUC: The area under the receiving operating curve, also noted AUC or AUROC, is the area below the ROC as shown in the following figure:



Source: https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-machine-learning-tips-and-tricks

Supervised Learning: Evaluation Metrics
<Classification Metrics>

Example Scenario:

Let's say we have a binary classification problem for a medical test that identifies a disease:
Out of 100 actual cases of the disease:
The test correctly identifies 80 as positive (True Positives).
The test incorrectly identifies 20 as negative (False Negatives).
Out of 100 non-cases (healthy individuals):
The test correctly identifies 90 as negative (True Negatives).
The test incorrectly identifies 10 as positive (False Positives).

Find the confusion matrix and compute the accuracy, recall, precision, and specificity!!

Supervised Learning: Evaluation Metrics
<Regression Metrics>

These regression metrics serve different purposes in evaluating the performance of regression models. **MAE**, **MSE**, and **RMSE** focus on the accuracy of predictions, while **R^2 (R-squared)** assesses the overall goodness of fit of the model. Depending on the specific characteristics of the problem, different metrics may be more appropriate for evaluation.

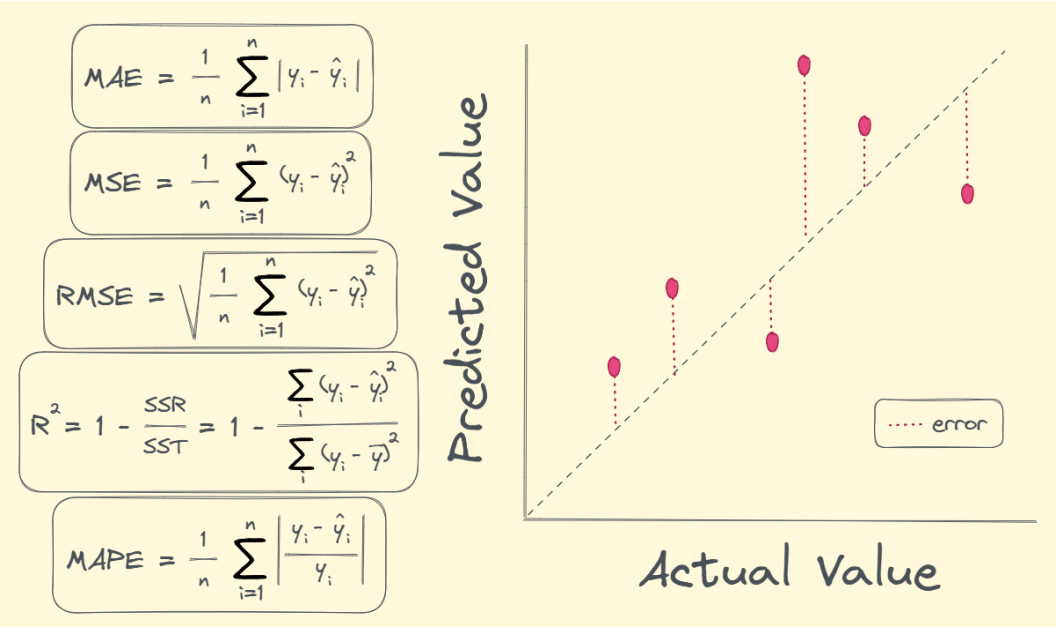


Fig. Regression evaluation metrics

Unsupervised Learning Algorithms

- This is called unsupervised learning because unlike supervised learning, there are no given labels/outputs and the machine itself finds the answers.
- **Motivation:** The goal of unsupervised learning is to find and discover hidden patterns in unlabeled data $\{x^{(1)},...,x^{(m)}\}$.
- **Example:** Clustering similar documents based on content/text.

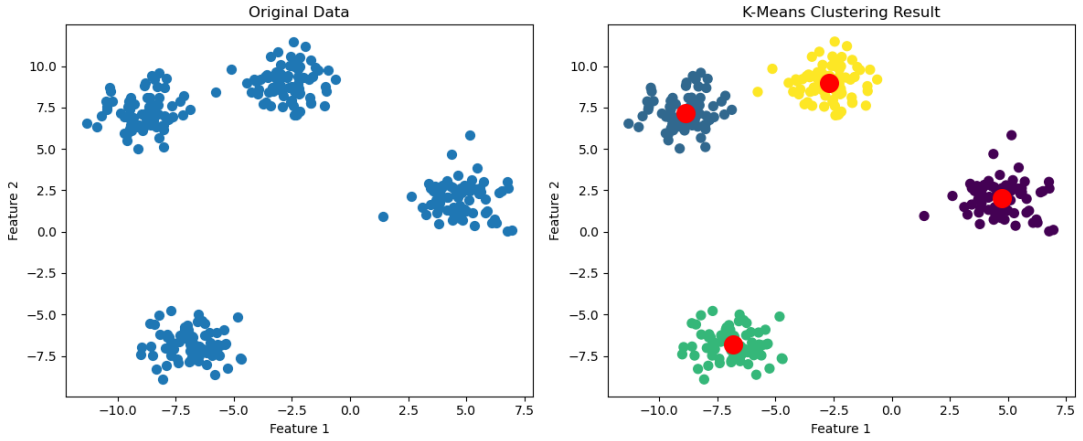


Fig. A simple illustration of the clustering technique

Unsupervised Learning Algorithms

- Types of unsupervised learning algorithms:
  - **Clustering:** Clustering aims to group similar data points together based on their features or attributes. The goal is to identify natural clusters in the data. **Examples:** Customer Segmentation: Grouping customers based on purchasing behavior. Image Segmentation: Dividing an image into regions with similar characteristics.
  - **Dimensionality Reduction:** Dimensionality reduction techniques aim to reduce the number of features or variables in the data while preserving important information.
  - **Association:** Association aims to discover relationships or patterns in data. It identifies sets of items that frequently occur together. **Examples**: Market Basket Analysis: Identifying which products tend to be bought together in a shopping basket. Recommender Systems: Suggesting products or content based on user behavior.

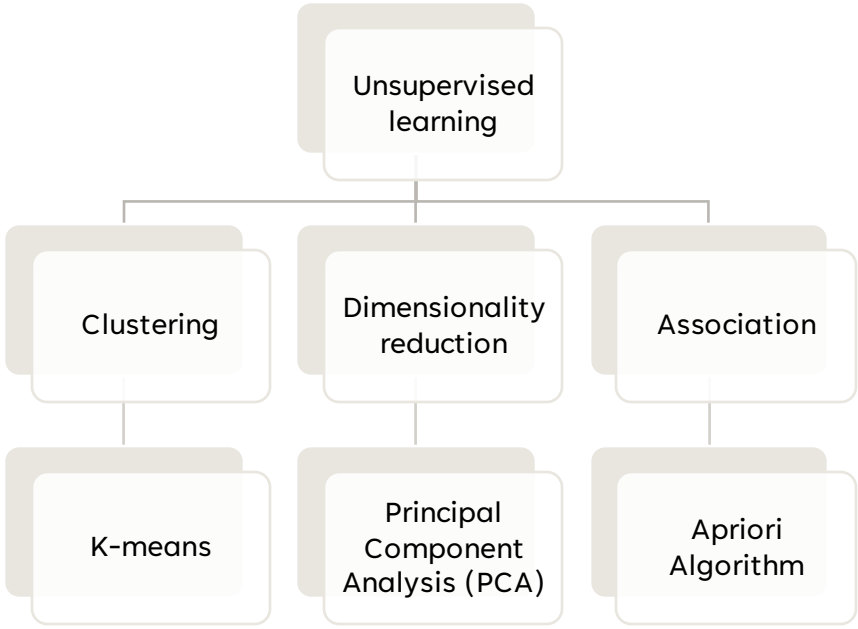


Fig. Some unsupervised learning algorithms

Unsupervised Learning Algorithms: Clustering

- Clustering in machine learning is a technique used to group similar data points together based on their features or attributes. The goal is to identify natural clusters in the data, where data points within a cluster are more similar to each other compared to points in different clusters.
- Clustering is an unsupervised learning technique, which means that the algorithm does not rely on labeled data. It learns from the inherent structure in the input data without any specific guidance.
- As the examples are unlabeled, clustering relies on unsupervised machine learning. If the examples are labeled, then clustering becomes classification.
- Clustering Use Cases:
  - market segmentation
  - social network analysis
  - search result grouping
  - medical imaging
  - image segmentation
  - anomaly detection

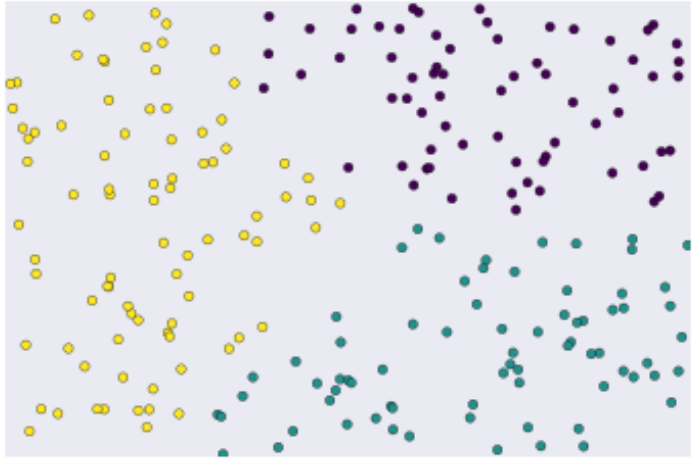
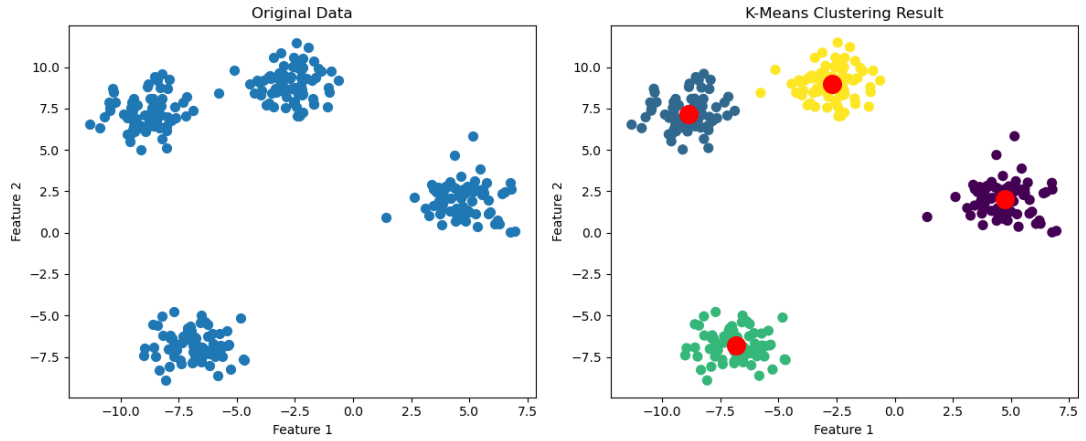


Fig. Unlabeled examples grouped into three clusters.

https://nicola17.github.io/tfjs-tsne-demo/

Unsupervised Learning Algorithms: Clustering using K-Means Algorithm

The K-Means algorithm is a popular clustering algorithm (centroid-based) used in machine learning. It aims to partition a dataset into K distinct, non-overlapping subsets (or clusters) based on the similarity of data points. The "K" in K-Means refers to the number of clusters that the algorithm aims to find.



Unsupervised Learning Algorithms: Clustering using K-Means Algorithm

How does the k-mean algorithm work?

We note $\mathbf{c^{(i)}}$ the cluster of data point **i** and $\boldsymbol{\mu_j}$ the center of cluster **j**.

Algorithm: After randomly initializing the cluster centroids $\mu_1, \mu_2, \ldots, \mu_k$. The k-means algorithm repeats the following step until convergence:

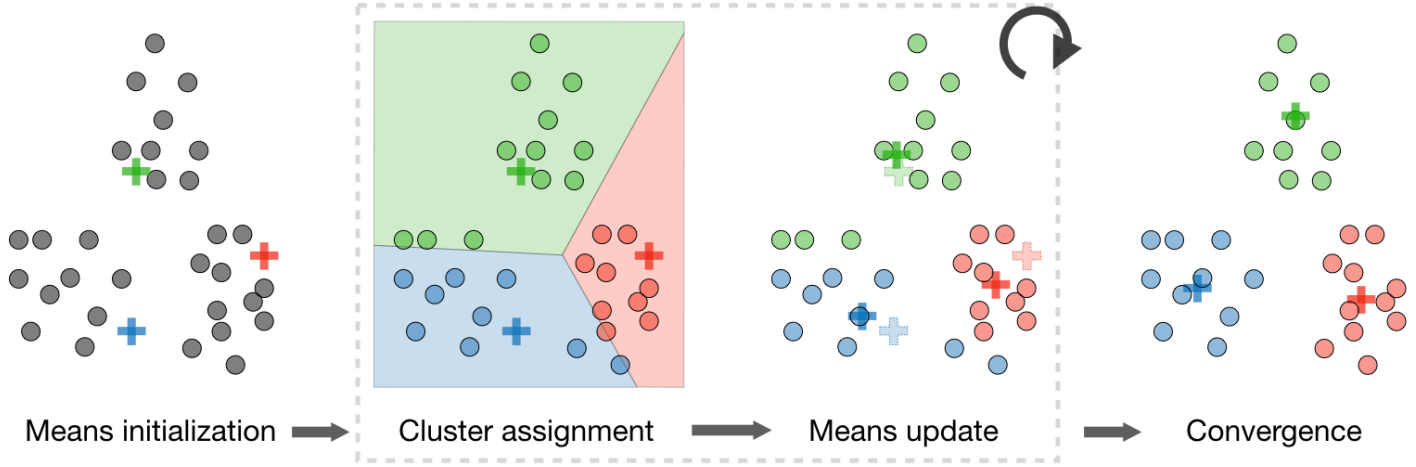
$$c^{(i)} = \arg\min_j ||x^{(i)} - \mu_j||^2$$

and

$$\mu_j = \frac{\sum_{i=1}^{m} 1_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^{m} 1_{\{c^{(i)}=j\}}}$$

Unsupervised Learning Algorithms: Clustering using K-Means Algorithm

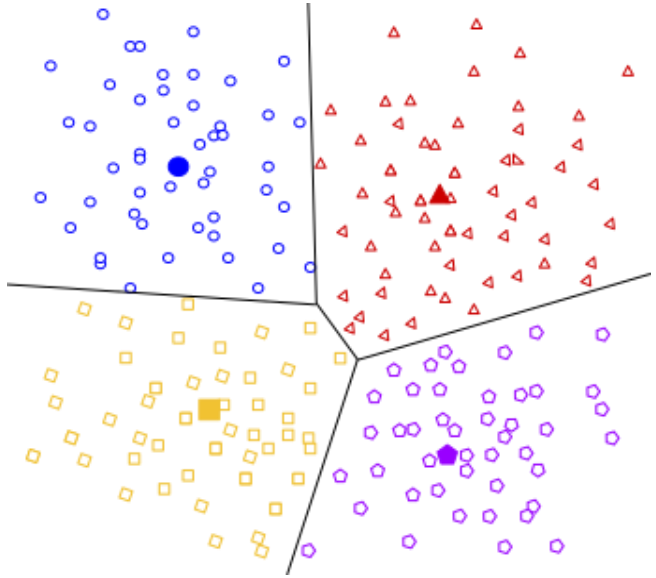
How does the k-mean algorithm work?



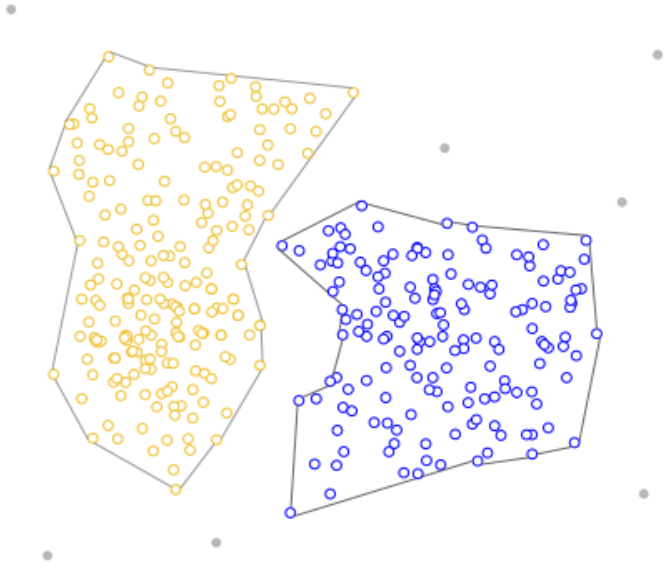
In order to see if the algorithm **converges**, we look at the **distortion function** defined as follows:

$$J(c,\mu) = \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

Unsupervised Learning Algorithms: Clustering Algorithms



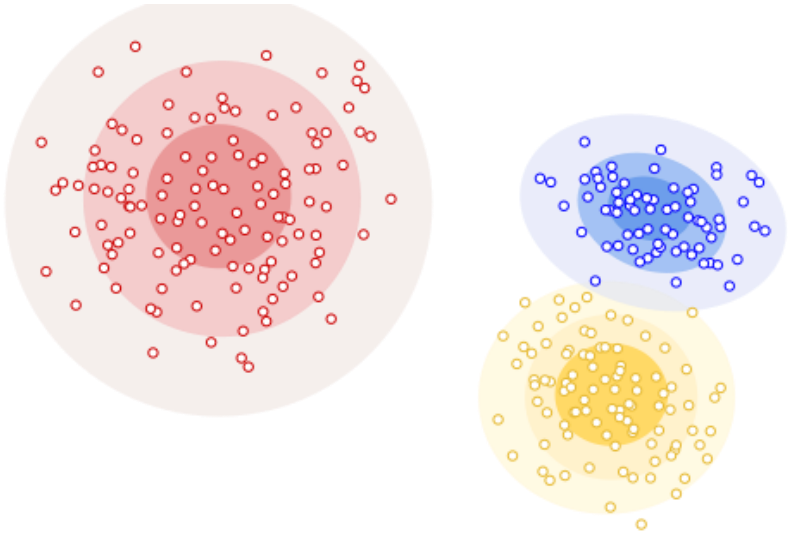
Centroid-based Clustering



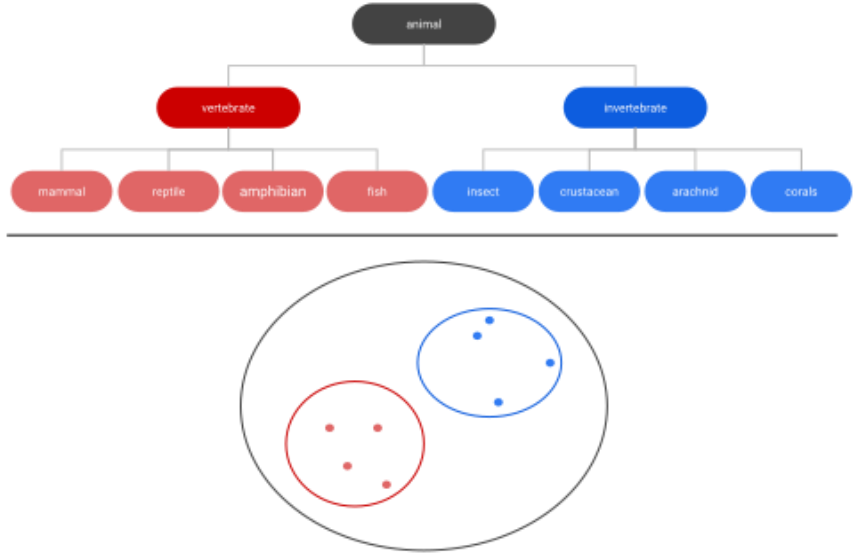
Density-based Clustering

For more details: A Comprehensive Survey of Clustering Algorithms

Unsupervised Learning Algorithms: Clustering Algorithms



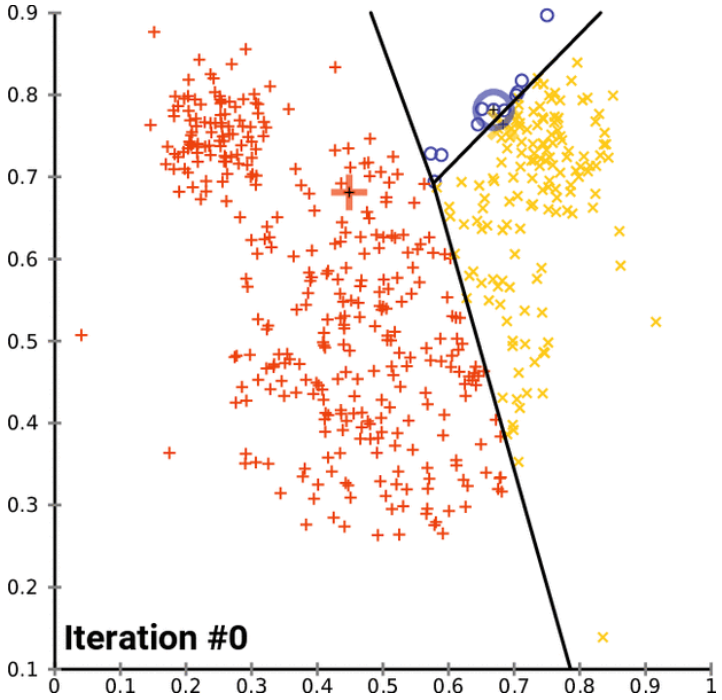
Distribution-based Clustering



Hierarchical Clustering

For more details: A Comprehensive Survey of Clustering Algorithms

Unsupervised Learning Algorithms: Clustering using K-Means – PYTHON DEMO



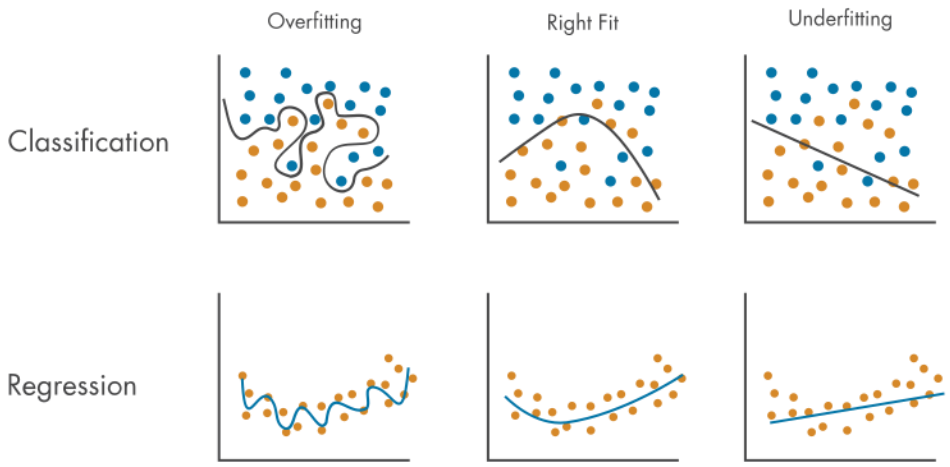
Source: Wikipedia

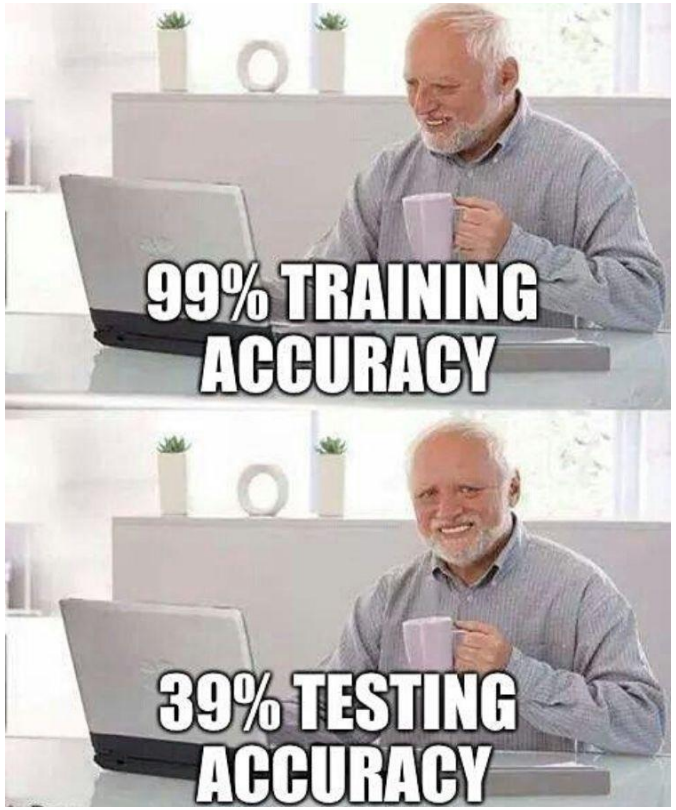
DEMO: Session 1 - Unsupervised Learning Clustering using K-Means Algorithm

Machine Learning: Important Concepts
<Overfitting and Underfitting>

Two **important** concepts:

- **Overfitting** and **underfitting**
- **Bias** and **variance** trade-off

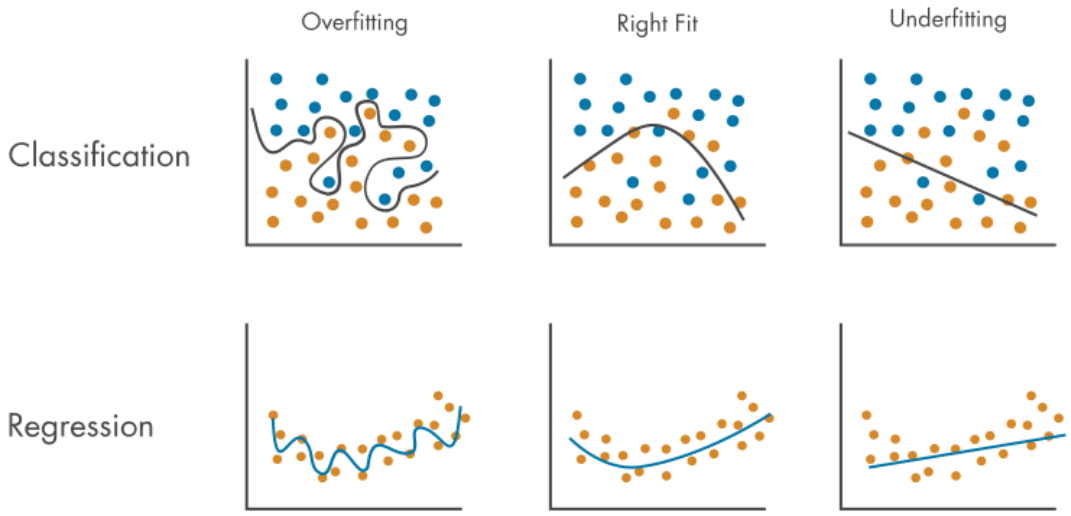




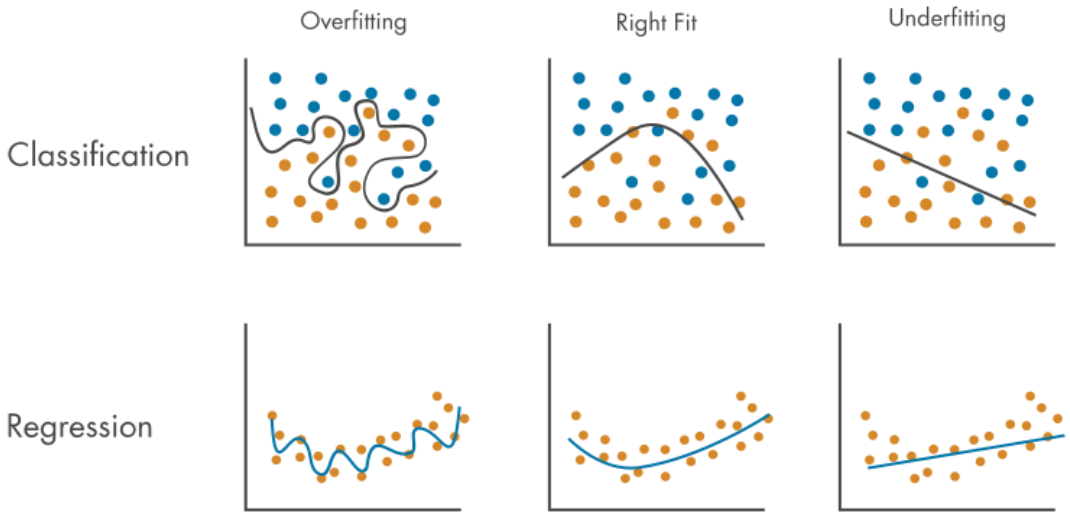
Machine Learning: Important Concepts
<Overfitting and Underfitting>

Underfitting and **overfitting** are two common problems in machine learning that occur when a model is not able to generalize well to new, unseen data.

- **Underfitting:** Underfitting occurs when a model is too simple to capture the underlying trend of the data. It performs poorly on both the training data and unseen data.
  - Causes of underfitting:
    - Using a very simple model (e.g., linear regression for a non-linear problem).
    - Not having enough features in the model.
    - Using too few training samples.

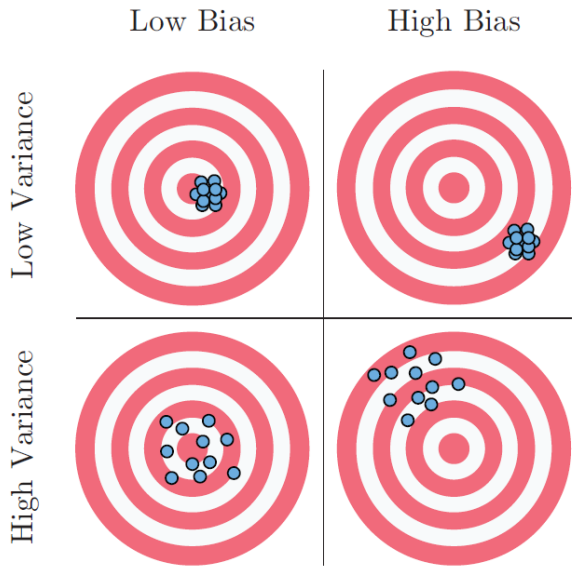


Machine Learning: Important Concepts
<Overfitting and Underfitting>

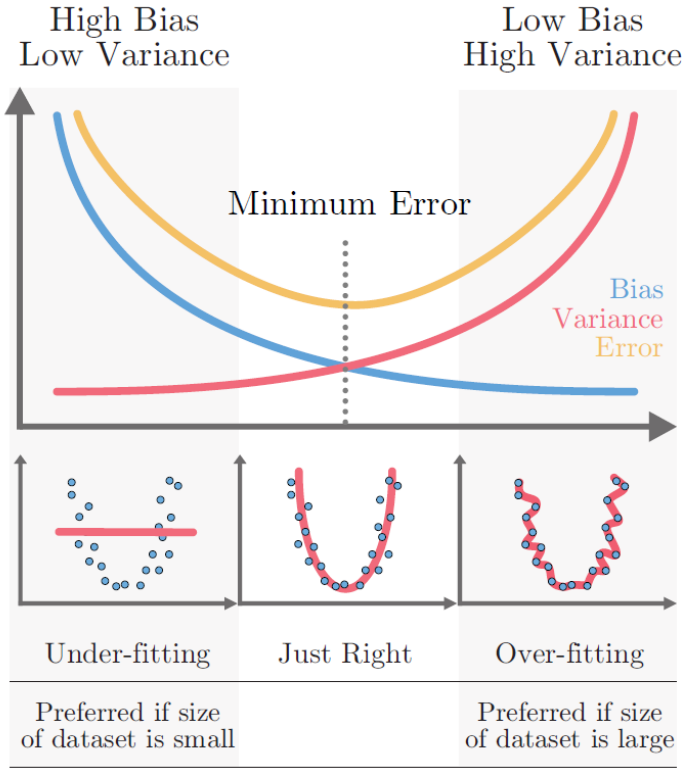


- **Overfitting:** Overfitting happens when a model learns the training data too well, capturing noise or random fluctuations that are not representative of the true underlying pattern. As a result, it performs well on the training data but poorly on unseen data.
  - Causes of underfitting:
    - Using a very complex model (e.g., a high-degree polynomial for a simple problem).
    - Having too many features relative to the number of training samples.

Machine Learning: Important Concepts
<Bias-Variance Tradeoff>



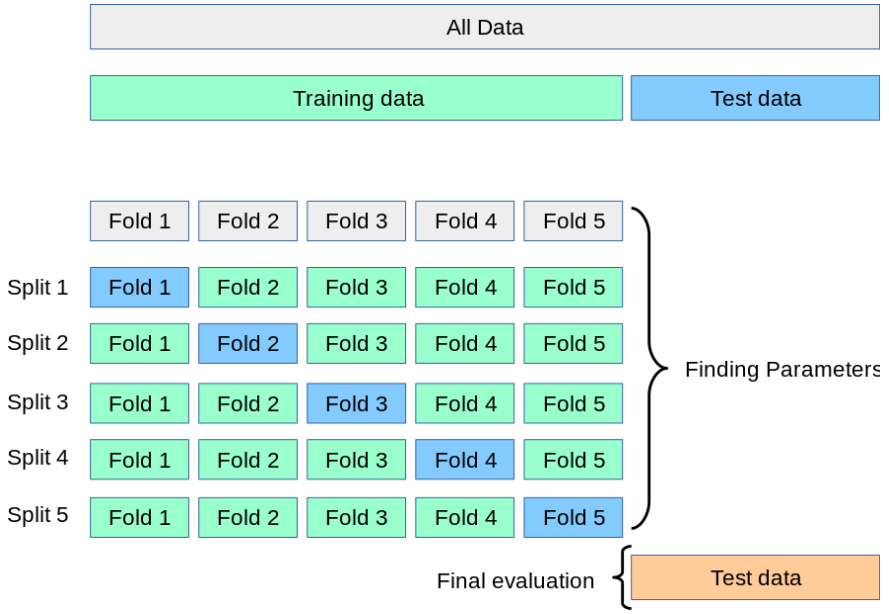
- **What is Bias:**
  - Error between average model prediction and ground truth
  - The bias of the estimated function tells us the capacity of the underlying model to predict the values
- **High Bias:** Overly-simplified model, Underfitting, and High error on both train and test sets
- **What is Variance?**
  - Average variability in the model prediction for the given dataset
  - The variance of the estimated function tells you how much the function can adjust to the change in the dataset
- **High Variance:** Overly-complex model, Overfitting, Low error on train data and high on test, Starts modelling the noise in the input.



Machine Learning: Important Concepts
<K-Fold Cross Validation>

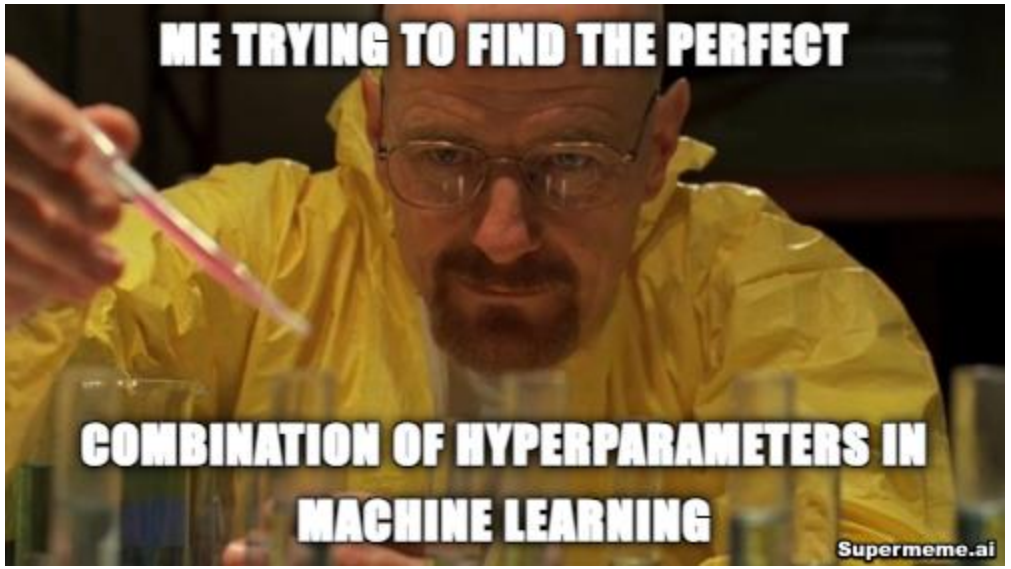
K-fold cross-validation: One of the techniques to evaluate machine learning models on limited data samples and one of the employed techniques to detect overfitting and how well a model will generalize to new, unseen data.

- **Idea:** The k-fold cross-validation (CV) refers to the **k** number of the folds or subset division of the dataset (The model is then trained and evaluated **k** times, using each fold as the testing set once and the remaining **k-1** folds as the training set). It is generally a technique used in machine learning to evaluate the performance of the models on the entire dataset, it ensures that every data point from the dataset has the chances of appearing in both training and testing sets, and thus it is one of the best approaches to evaluate the models' performance.



Machine Learning: Important Concepts
<Hyper-Parameters Optimization/Tuning>

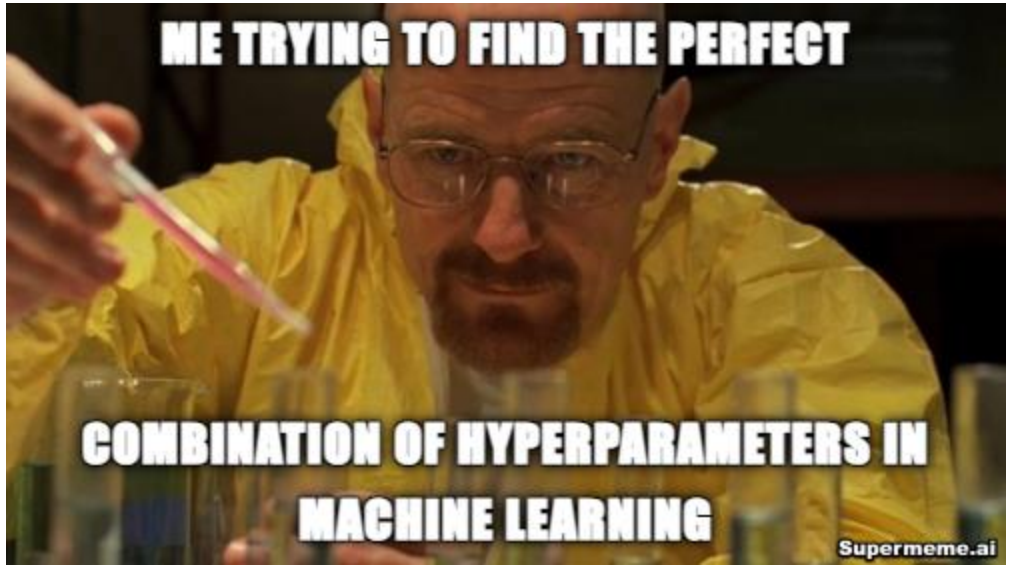
Problem: We have seen so far different machine learning algorithms; linear regression, logistic regression, decision tree, ...etc. In linear regression for example, we talked about something called "learning rate", in decision trees, we talked about the criterion (to measure the quality of a split) like : gini impurity and entropy.
Question: How to choose the best values for these hyperparameters



Machine Learning: Important Concepts
<Hyper-Parameters Optimization/Tuning>

Definition: A hyperparameter is a configuration setting for a machine-learning model that is not learned from the data. Instead, it is set prior to training and remains constant during the training process. These parameters are essential for controlling the learning process, but they are not learned from the data itself.

In contrast, the parameters of a machine learning model are the variables that the model learns from the training data. For example, in a linear regression model, the coefficients for each feature are learned parameters.



Machine Learning: Important Concepts
<Hyper-Parameters Optimization/Tuning>

Hyperparameters of some machine learning algorithms:

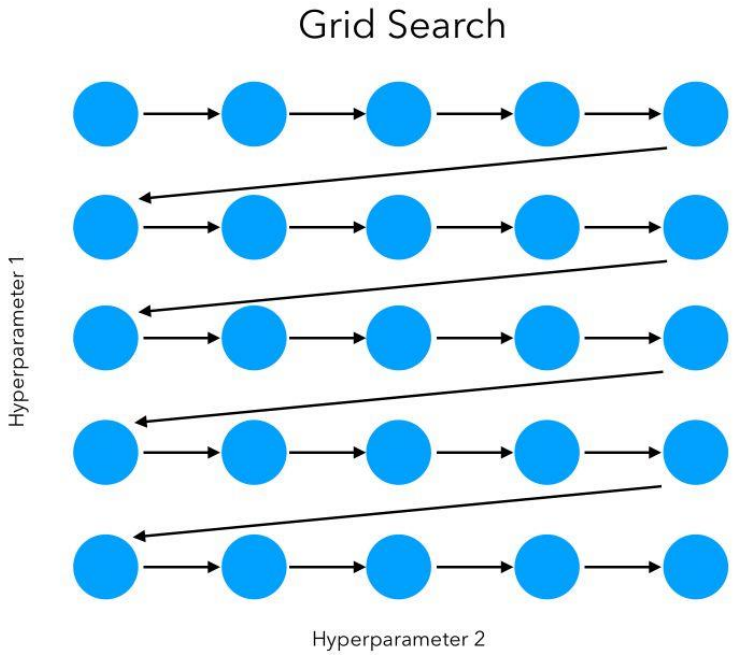
- **Linear regression:** Linear regression is not a complex algorithm; therefore, it does not have hyperparameters except the learning rate (alpha). However, there are some advanced regression algorithms that have more hyperparameters.
- **Logistic Regression:** Solver, penalty, and 'C' regularization strength.
- **Decision Trees:** Criterion, Maximum depth.
- **K-Nearest Neighbour:** Number of Neighbours 'K'.

class `sklearn.tree.`**`DecisionTreeClassifier`**(**, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0*) [source]

class `sklearn.linear_model.`**`LogisticRegression`**(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

Machine Learning: Important Concepts
<Hyper-Parameters Optimization/Tuning>

- **Hyperparameter optimization** is the process of **finding** the **optimal hyperparameters** for a machine-learning model in order to obtain the optimal performance of the model. **Hyperparameters** are parameters not learned from the data but the user sets them before training the model. **Hyperparameter optimization** is a crucial task in the process of building machine learning models because the performance of a machine learning model heavily depends on the chosen values for its **hyperparameters**. Through a systematic exploration of various **hyperparameter** combinations, we can identify the exact values that yield optimal performance for a given task or problem.
- **Several approaches** to hyperparameter optimization/tuning, including **manual tuning**, **grid search**, **random search**, and some other advanced techniques like **Bayesian optimization** and **genetic algorithms.**



Source: https://maelfabien.github.io/machinelearning/Explorium4/#what-is-hyperparameter-optimization

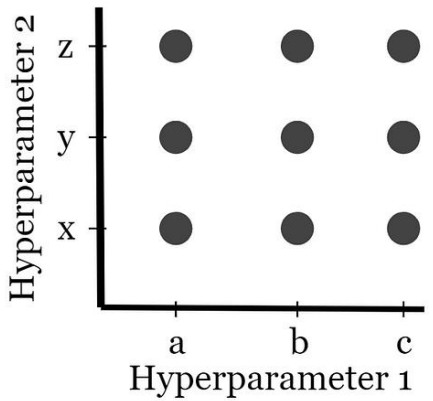
Machine Learning: Important Concepts
<Hyper-Parameters Optimization/Tuning>

- **Grid Search:** In grid search, a set of possible values for each hyperparameter is specified by the user, and the algorithm comprehensively evaluates all possible combinations of hyperparameters.
- **Random Search:** Random search randomly selects hyperparameter values from predefined ranges. It doesn't cover the entire search space like grid search but can be more efficient in practice, especially if some hyperparameters are less influential.
- **Note:** There are some libraries and tools available that can help with hyperparameter optimization/tuning, such as **'GridSearch'** and **'RandomSearch'** in scikit-learn library, **Optuna**, **Hyperopt**, and **Ray Tune**.

Grid Search

Pseudocode

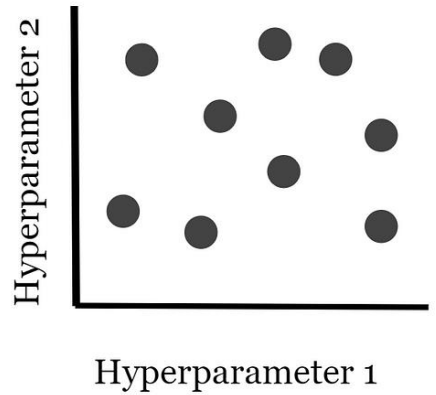
```
Hyperparameter_One = [a, b, c]
Hyperparameter_Two = [x, y, z]
```



Random Search

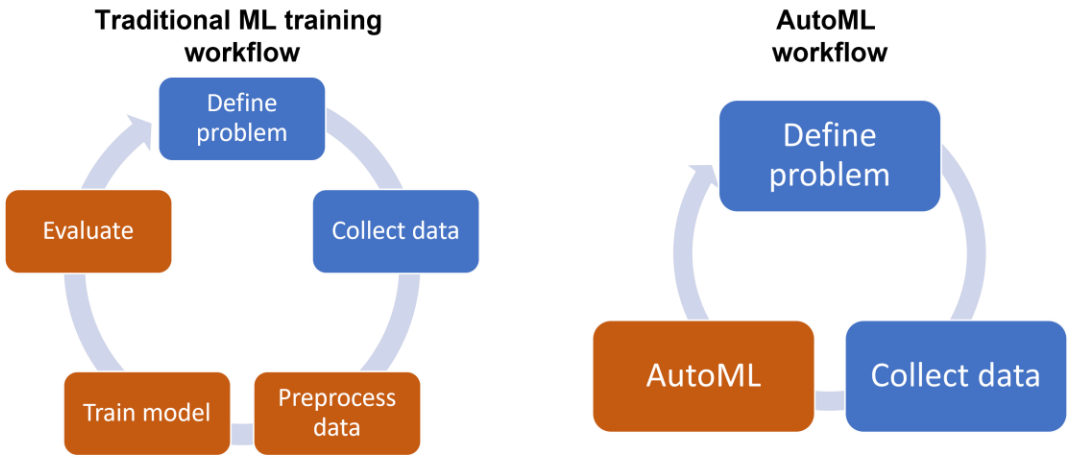
Pseudocode

```
Hyperparameter_One = random.num(range)
Hyperparameter_Two = random.num(range)
```



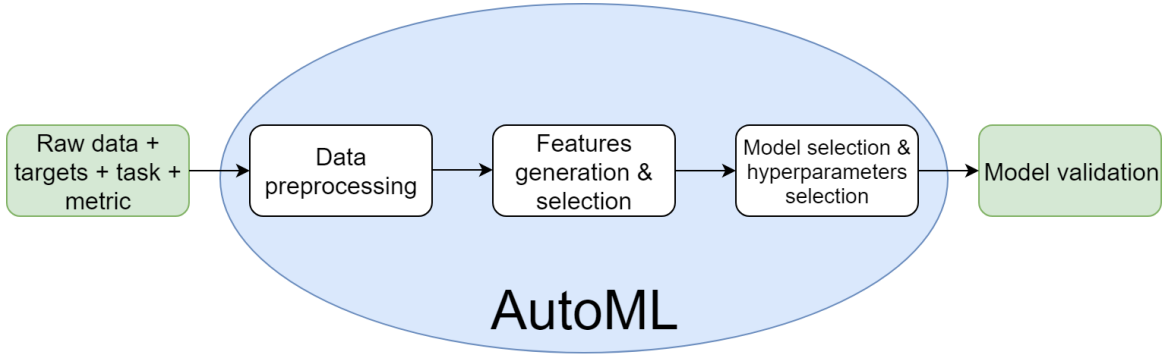
Automated Machine learning (AutoML)

Automated machine learning or simply AutoML, refers to the process of automating the end-to-end process of applying machine learning to real-world problems. It aims to make machine learning more accessible to individuals and organizations with limited expertise in data science and machine learning.

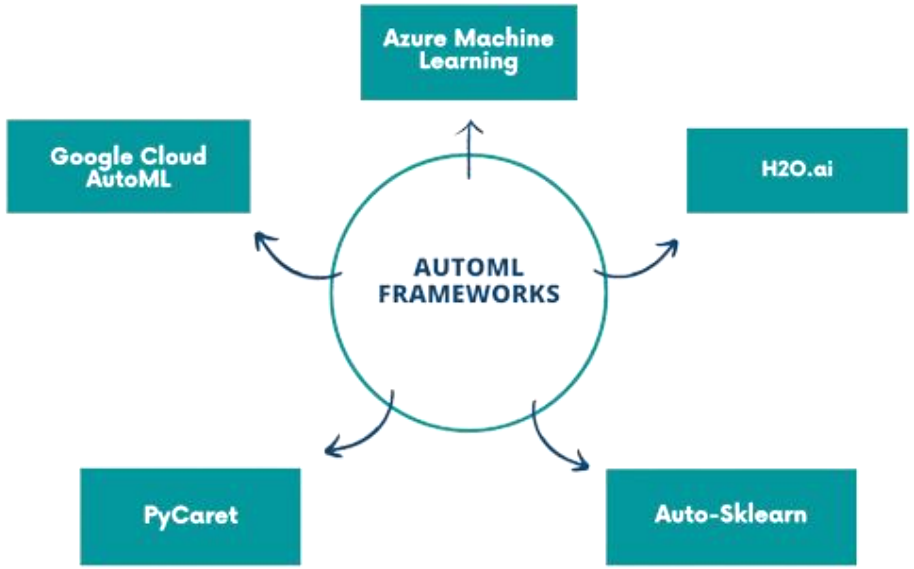


Source: learn.microsoft.com

Automated Machine learning (AutoML)
< Key Aspects >



Automated Machine learning (AutoML)
< Some Frameworks >



Automated Machine learning (AutoML)
<DEMO>

DEMO: Session 1 – Automated Machine Learning AutoML

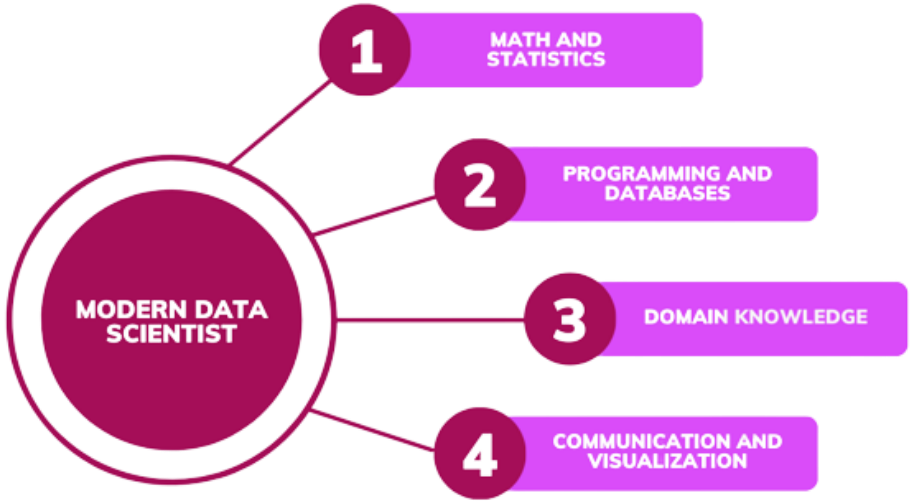
Automated Machine learning (AutoML)
< AutoML vs Data scientists >

Will AutoML replace data scientists and machine learning engineers?

Short Answer: No, but…

Automated Machine learning (AutoML)
< AutoML vs Data scientists >

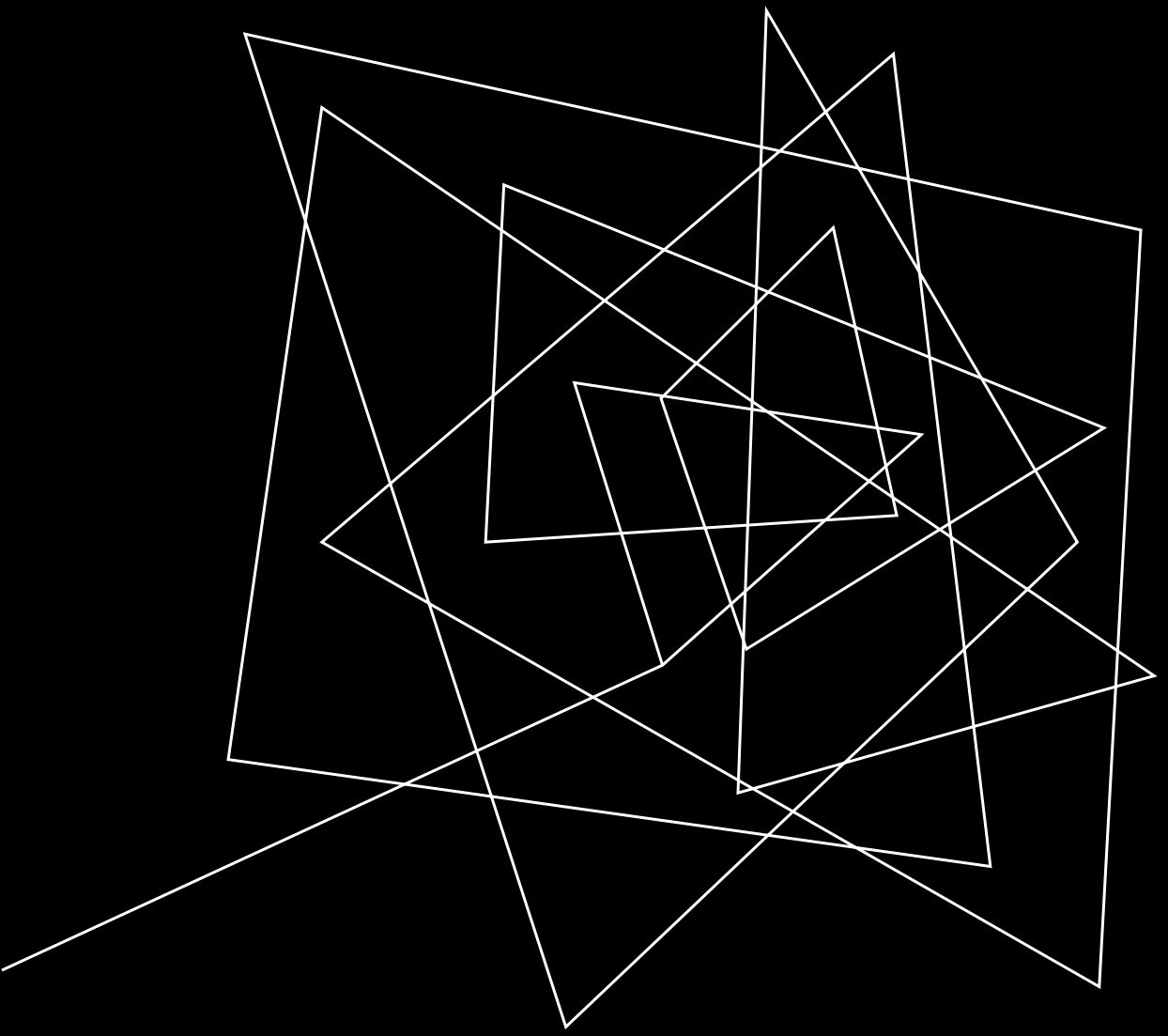
- **AutoML** will not replace data scientists.
- Data scientists must be specialized and educated much more than before.
- **AutoML** will boost productivity and make the work of a data scientist more interesting and more challenging.
- Domain knowledge will become more important.
- Scripting will become less important.



Classical Machine Learning: TO-DO Projects

Regression: House prices prediction

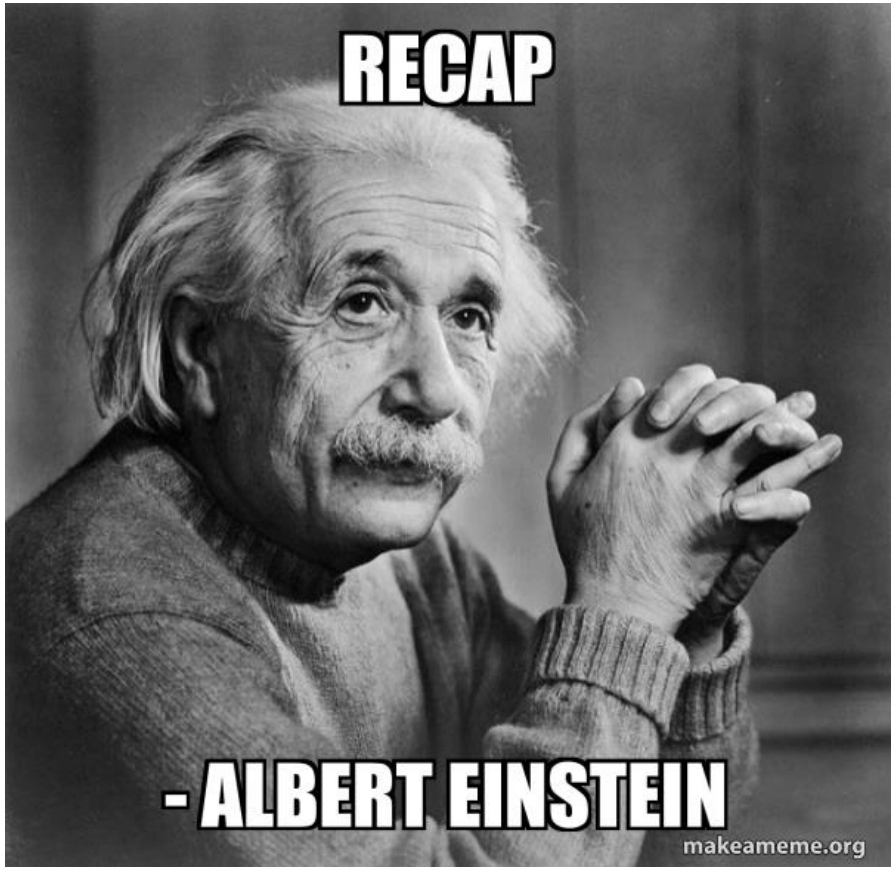
Classification: Predict survival on the Titanic



SESSION 2: DEEP LEARNING

Perceptron, Artificial Neural Networks, and Backpropagation

Session 1: Machine Learning
<A RECAP>




DEEP
LEARNING

Deep Learning: Introduction

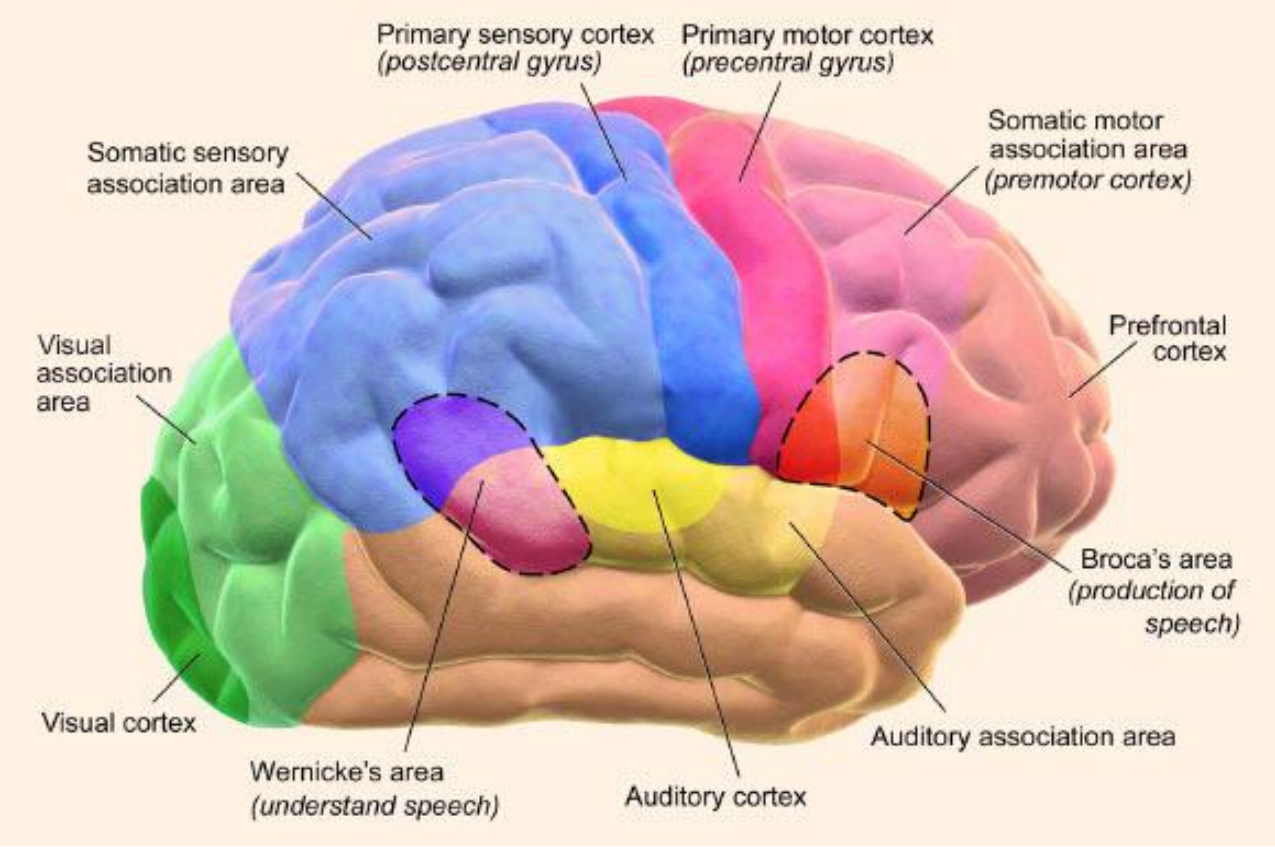
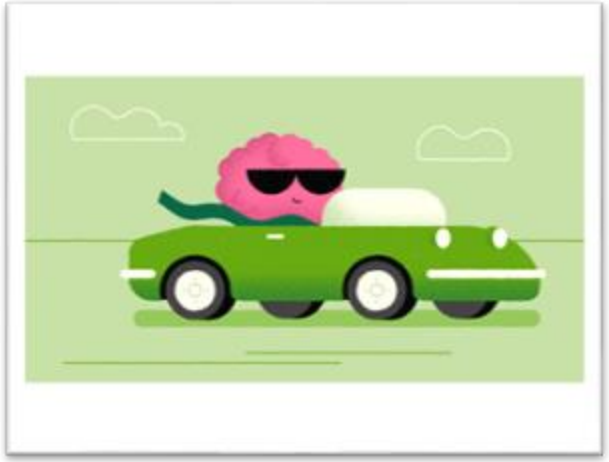


Fig. Human brain

Deep Learning: Introduction

The human brain is amazing:

1. Robust and powerful even with the existence of some problems (Neuroplasticity)
2. The ability to learn and to adapt with new and unfamiliar environments
3. The ability to deal with incomplete and noisy information
4. Parallel Processing/Computing



Deep Learning: Introduction

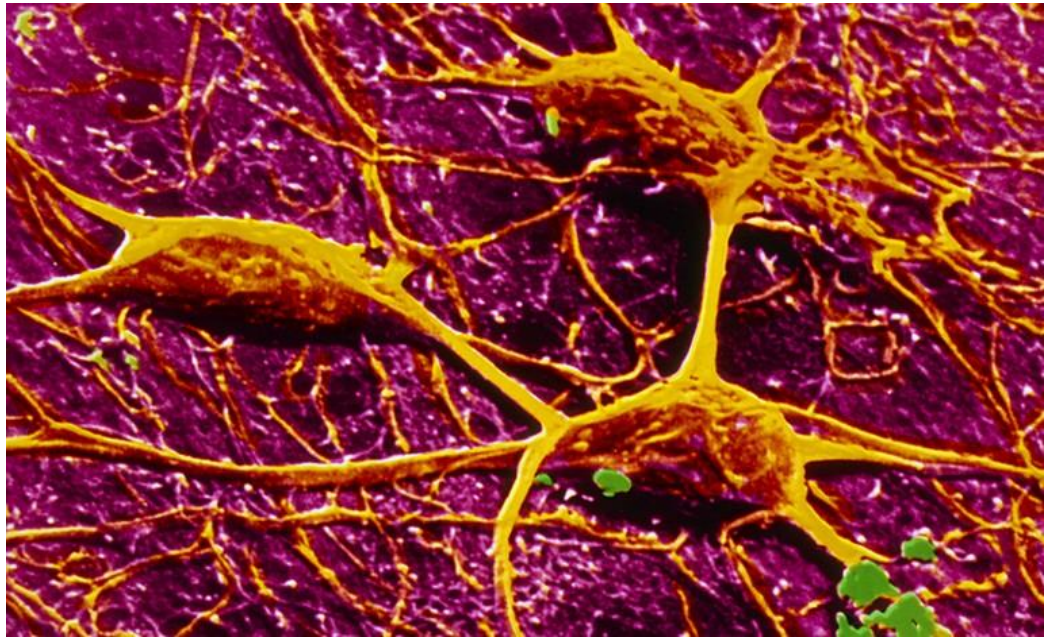


Fig. Neurons store and transmit information in the brain.
Credit: CNRI/SPL

Gif. Biological neurons

Deep Learning: Introduction

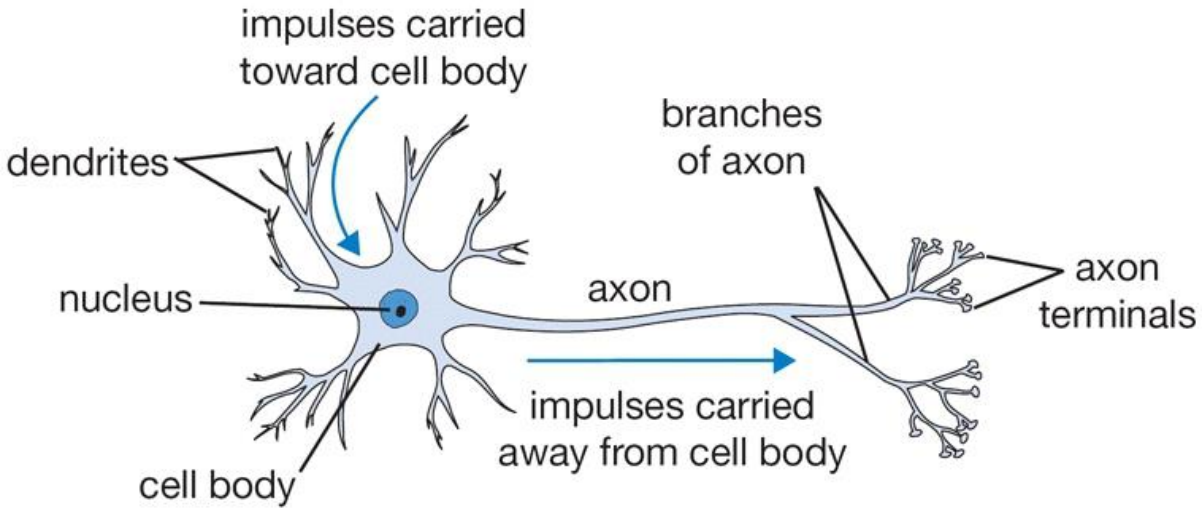


Fig. A cartoon drawing of a biological neuron

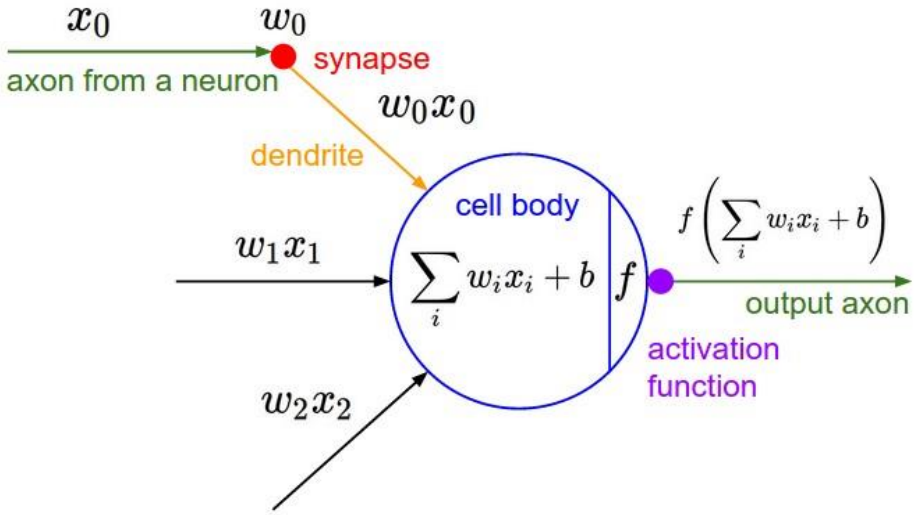


Fig. Mathematical model of the biological neuron

Deep Learning: Perceptron

Perceptron is a simplified model of a biological neuron proposed by **Rosenblatt** (1957-58)

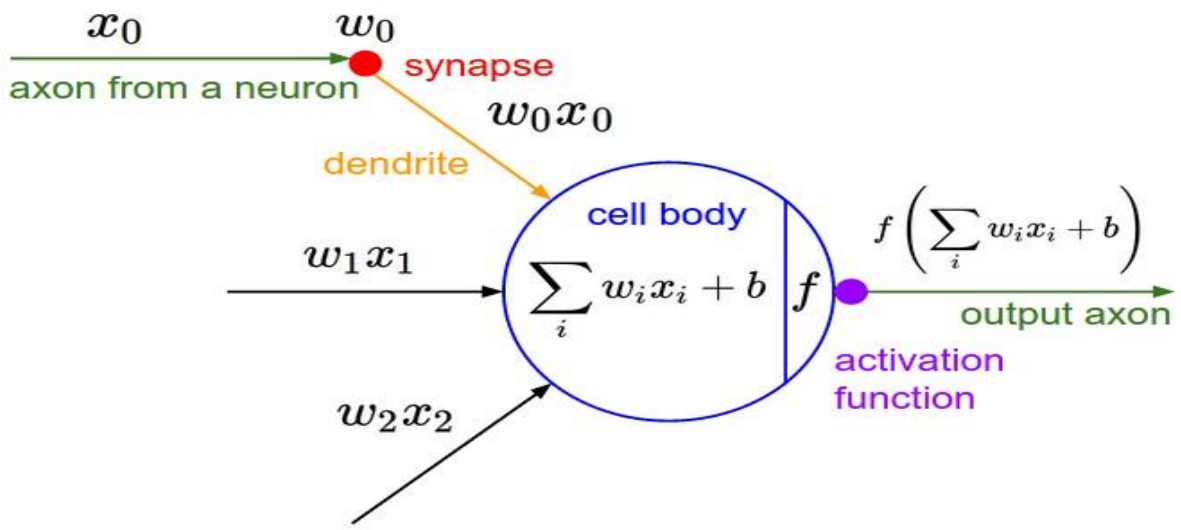


Fig. Mathematical model of the biological neuron (Perceptron)

Fig. Frank Rosenblatt

Deep Learning: Perceptron

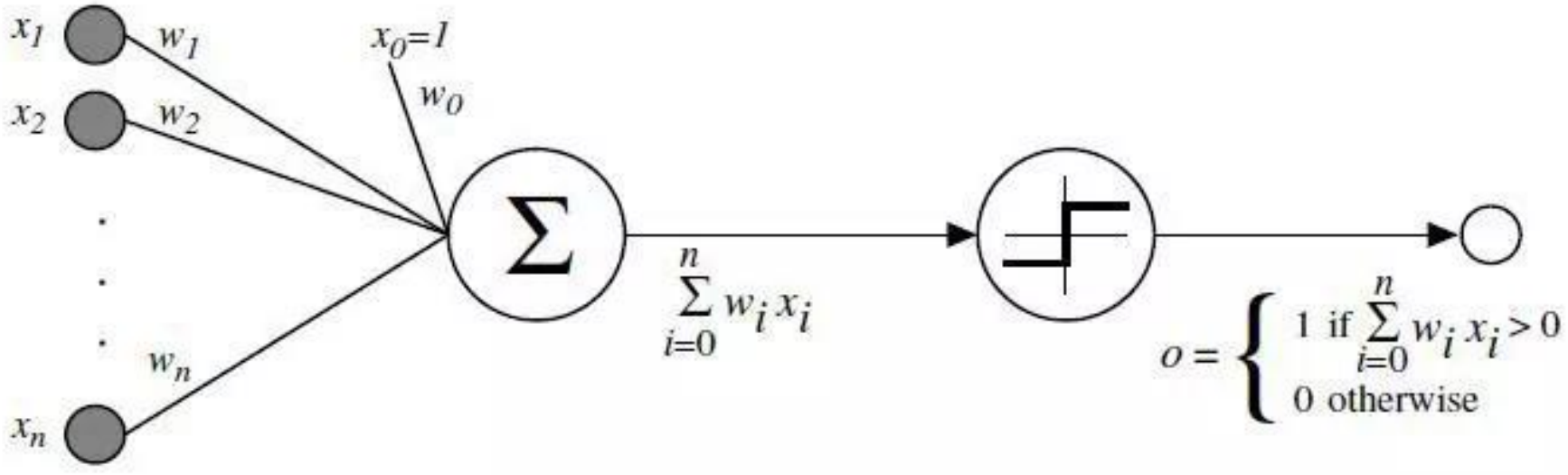


Fig. A diagram showing how the Perceptron works.

Deep Learning: Mathematical Model of The Perceptron

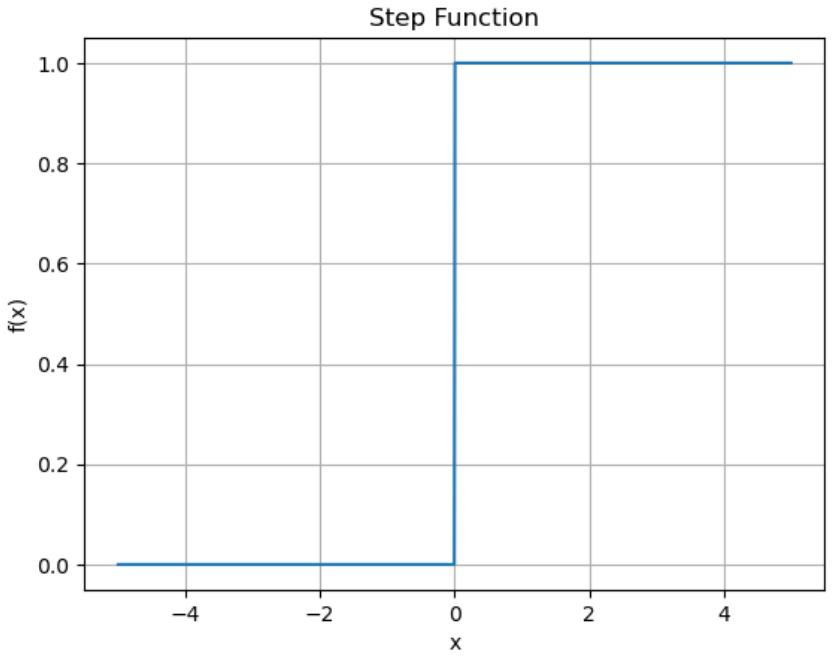
$$output = f(\sum_{i=1}^{n} w_i x_i + b)$$

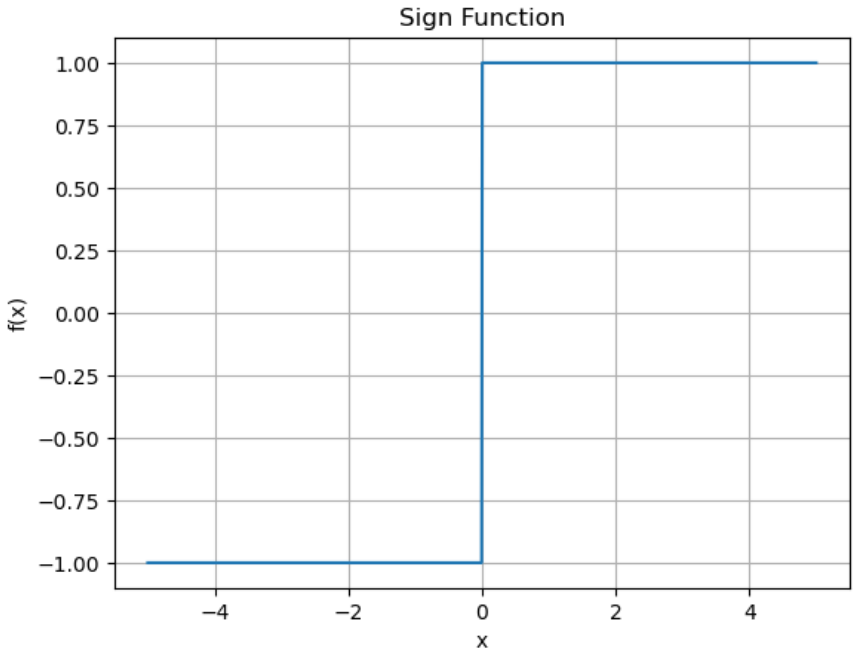
Activation function

Weights

Bias

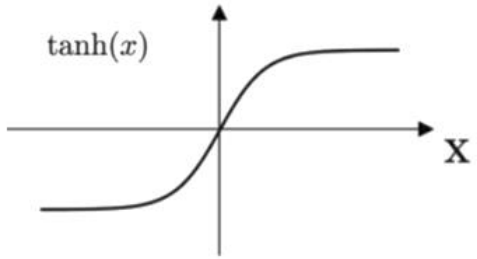
Perceptron: Activation Functions



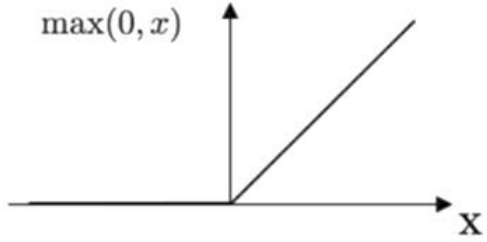


Perceptron: Activation Functions

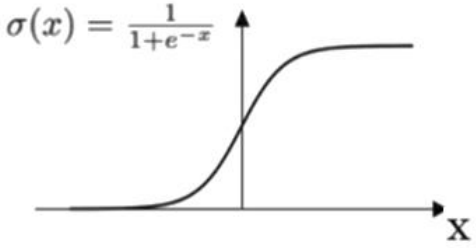
Tanh



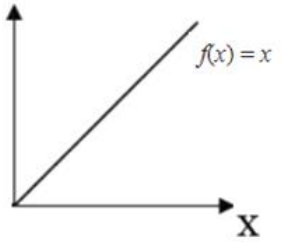
ReLU



Sigmoid

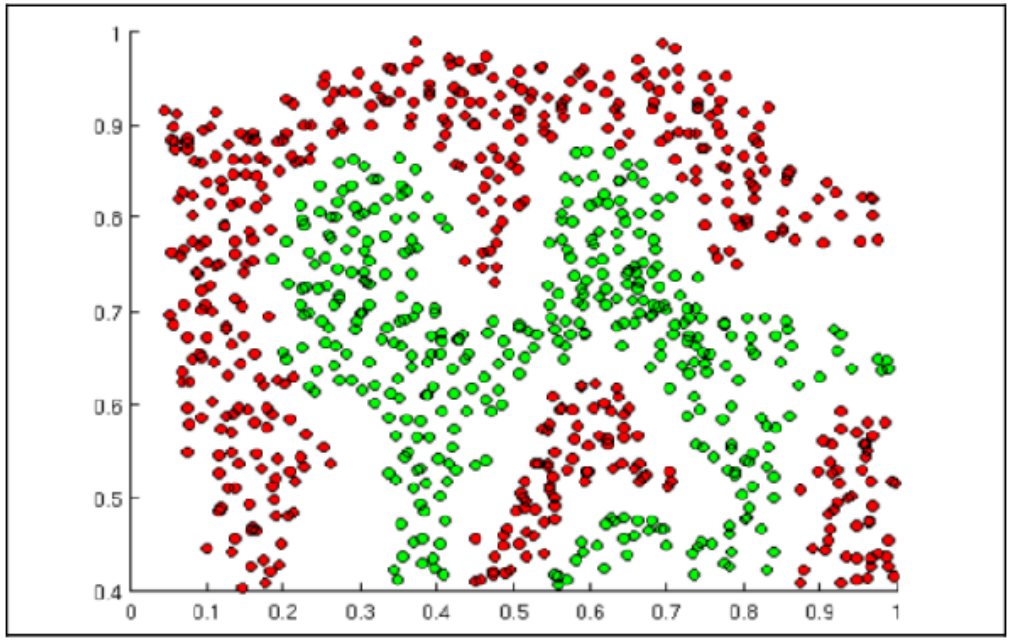


Linear



Perceptron: Importance of Activation Functions

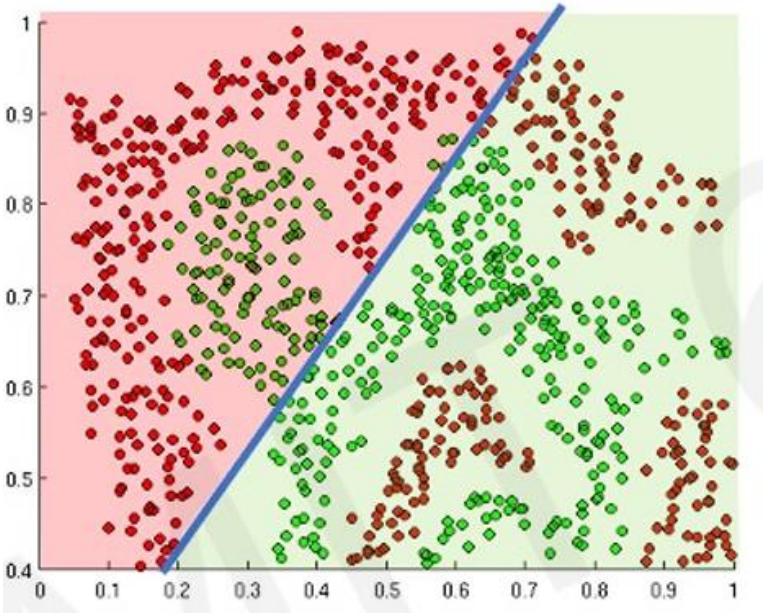
The importance of the activation functions is to introduce non-linearities into the network.



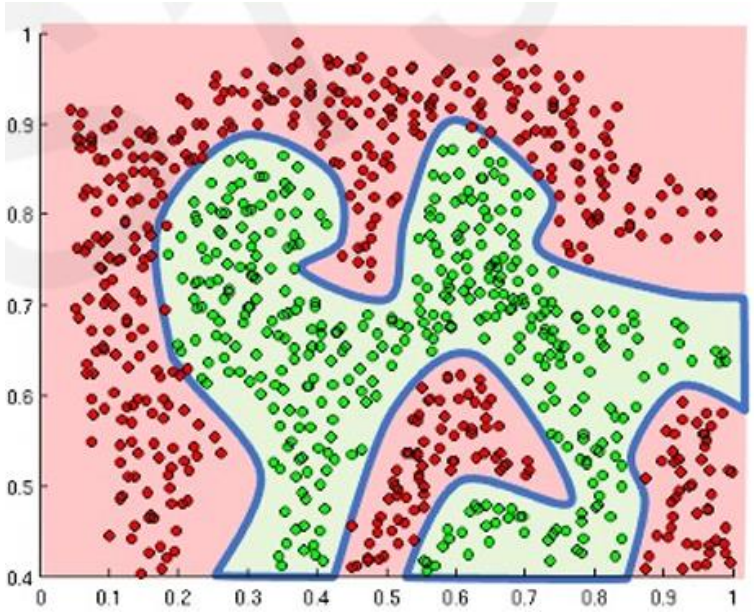
What if we wanted to build a neural network to distinguish green vs red points?

Perceptron: Importance of Activation Functions

The importance of the activation functions is to introduce non-linearities into the network.



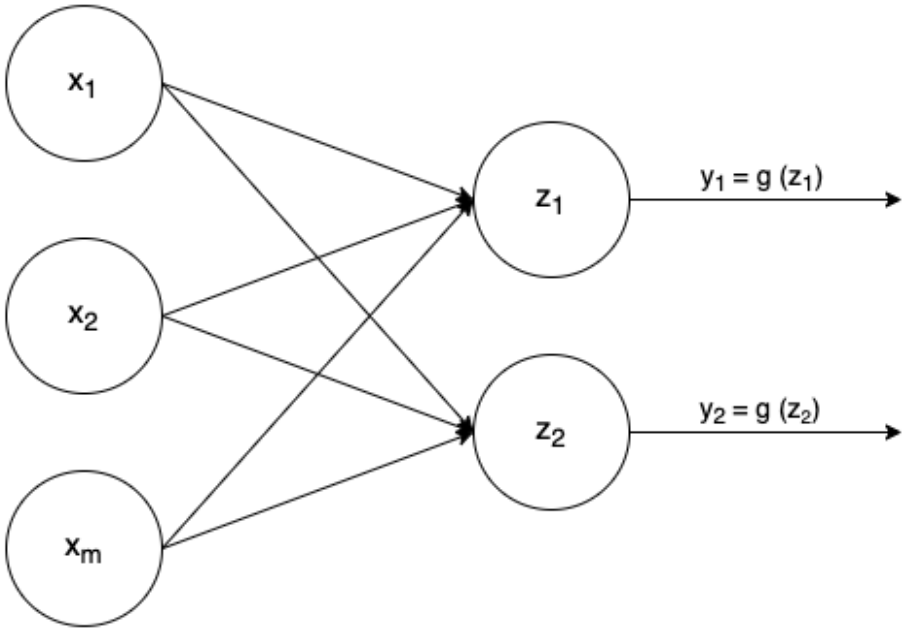
Linear activation functions produce linear decision boundaries no matter the network size



Non-linear activation functions allow to approximate complex function

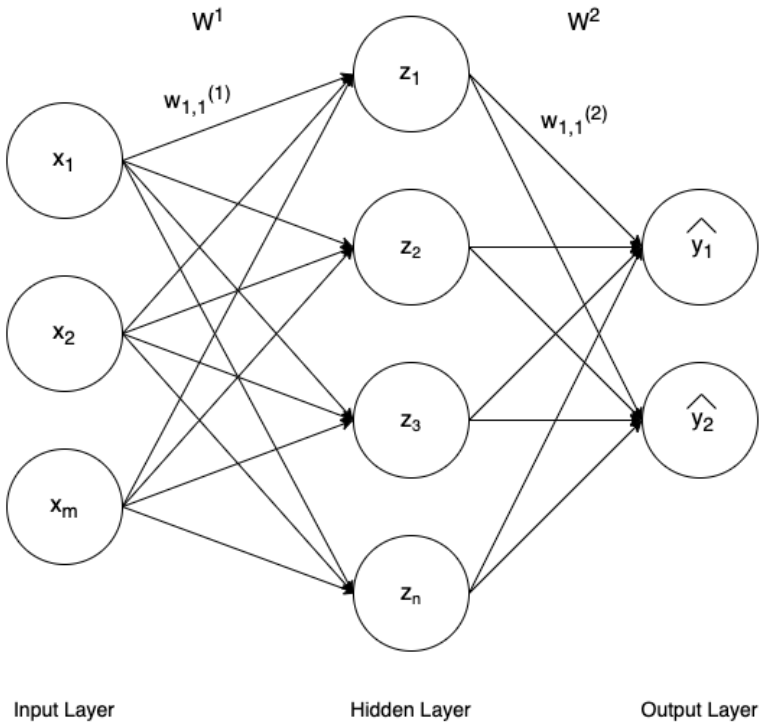
Perceptron: Multi-output Perceptron

- The **problem** of **XOR** logic function demonstrated the **limitations** of the **perceptron.**
- We can link two or more **perceptrons** with each other.
- Each perceptron is connected and linked with each input, it is called **fully connected layer** or **dense layer**.



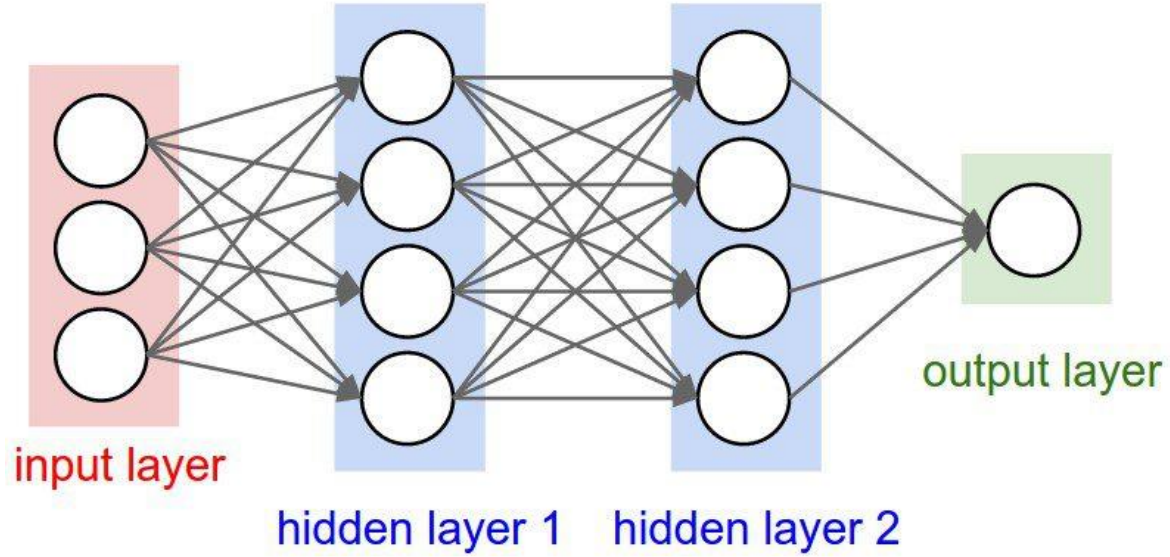
Neural Networks: Single Layer Neural Network

- Like in the human brain, the power and robustness of the biological neurons is when they are fully connected to each other.
- The figure in the right shows the simplest architecture of single layer neural network

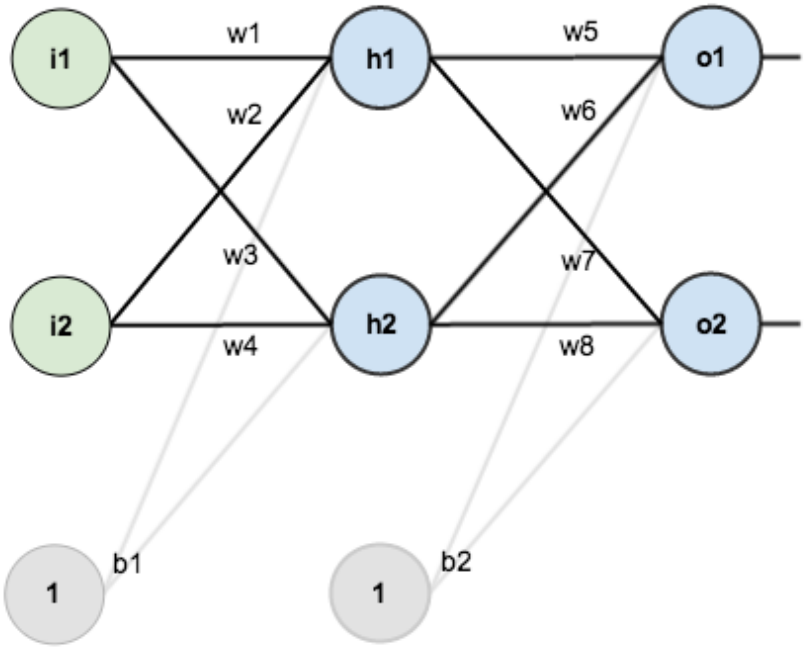


Neural Networks: Multi-Layer Neural Network

Like in the human brain, the power and robustness of the biological neurons is when they are fully connected to each other, it starts to become more and more complex by adding/stacking more and more hidden layers, at this level we are talking about **Multi-layer perceptron.** **(DEEP LEARNING)**



Deep Learning: Weights and Biases in a Neural Network

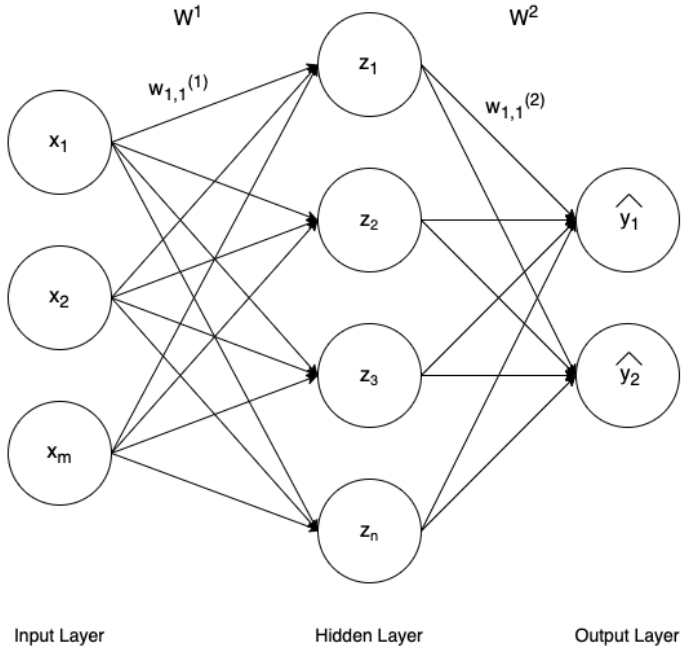


- The ultimate goal is to **find** the **optimal** values for the **weights** and **biases** that provide **good** predictions and thus **high accuracy**
- **Training** the artificial neural network to find the **weights** and **biases**

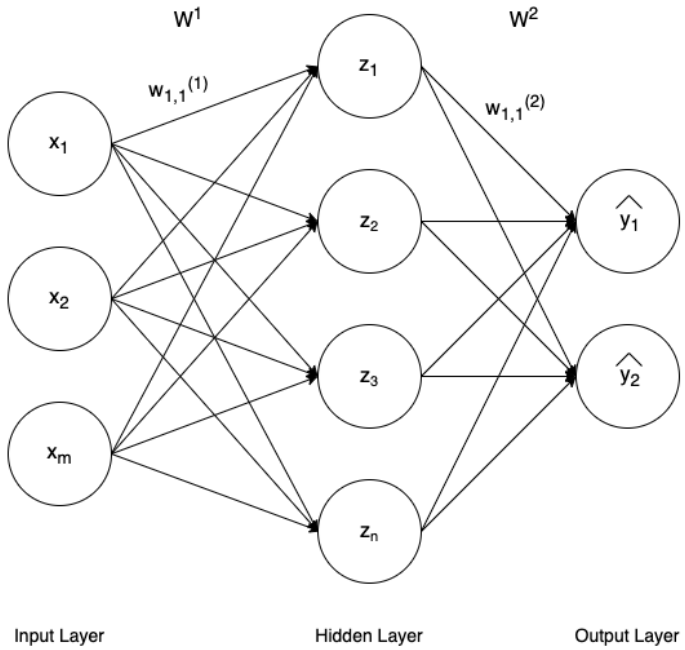
Deep Learning: How is a neural network "classically" trained?

How do humans learn?

Note: The term "classically" refers to one of the widely used techniques employed to train a neural network. There are many other "advanced " techniques



Deep Learning: How is a neural network "classically" trained?



- Before training a neural network, we have to define the prediction target (regression or classification) in order to determine the **loss/cost function** to be **minimized**

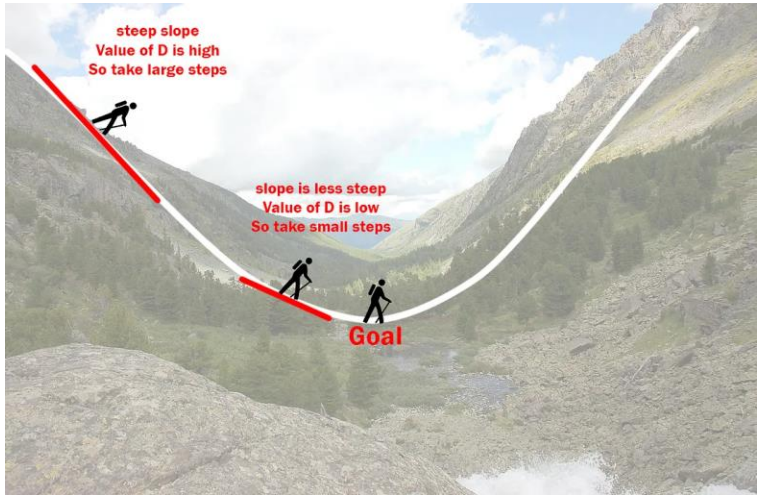
Deep Learning: How is a neural network "classically" trained? <Loss Optimization>

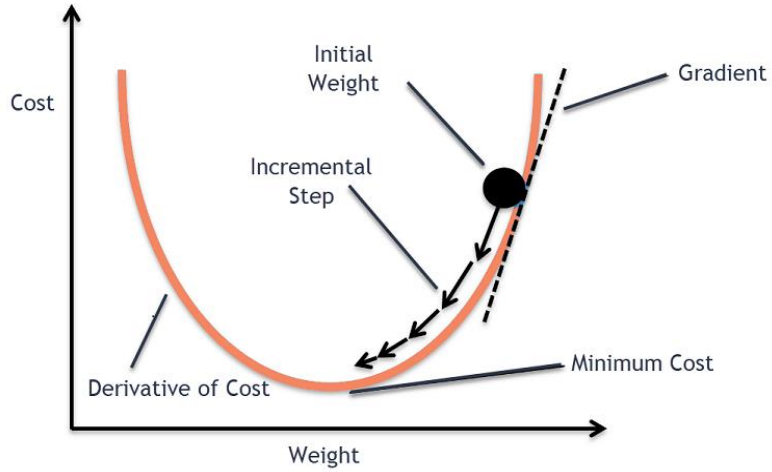
- Let's break this down, and let's suppose that we have a supervised learning task **T**, with inputs **X** and output **y.**
- Let **f** represent the neural network, **L** the loss function, and **W** the ensemble of weights and biases
- The **goal** is to find the parameters (weights and biases) that minimize the loss function **L.**

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

Deep Learning: Gradient Descent Is Back
<Loss Optimization>





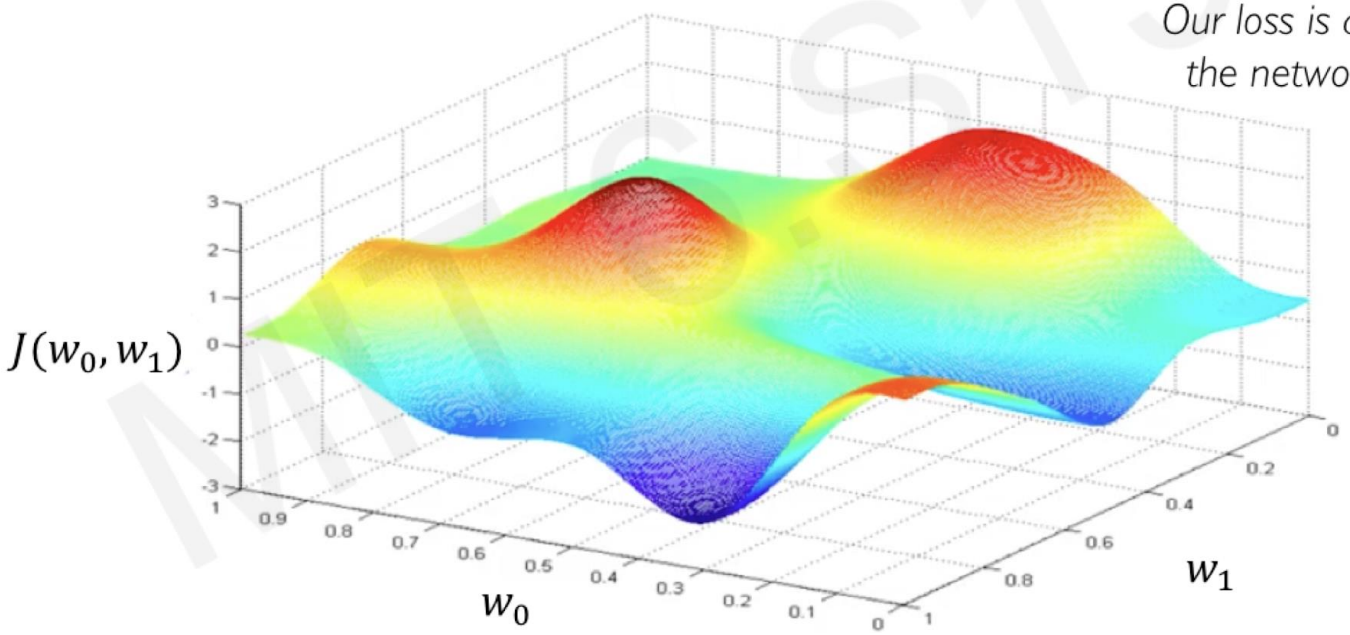
Note: Gradient descent is one of the optimization algorithms to find the parameters (weights and biases) that minimize the loss function (Local or Global Minimum)

Deep Learning: Gradient Descent <Loss Optimization>

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$

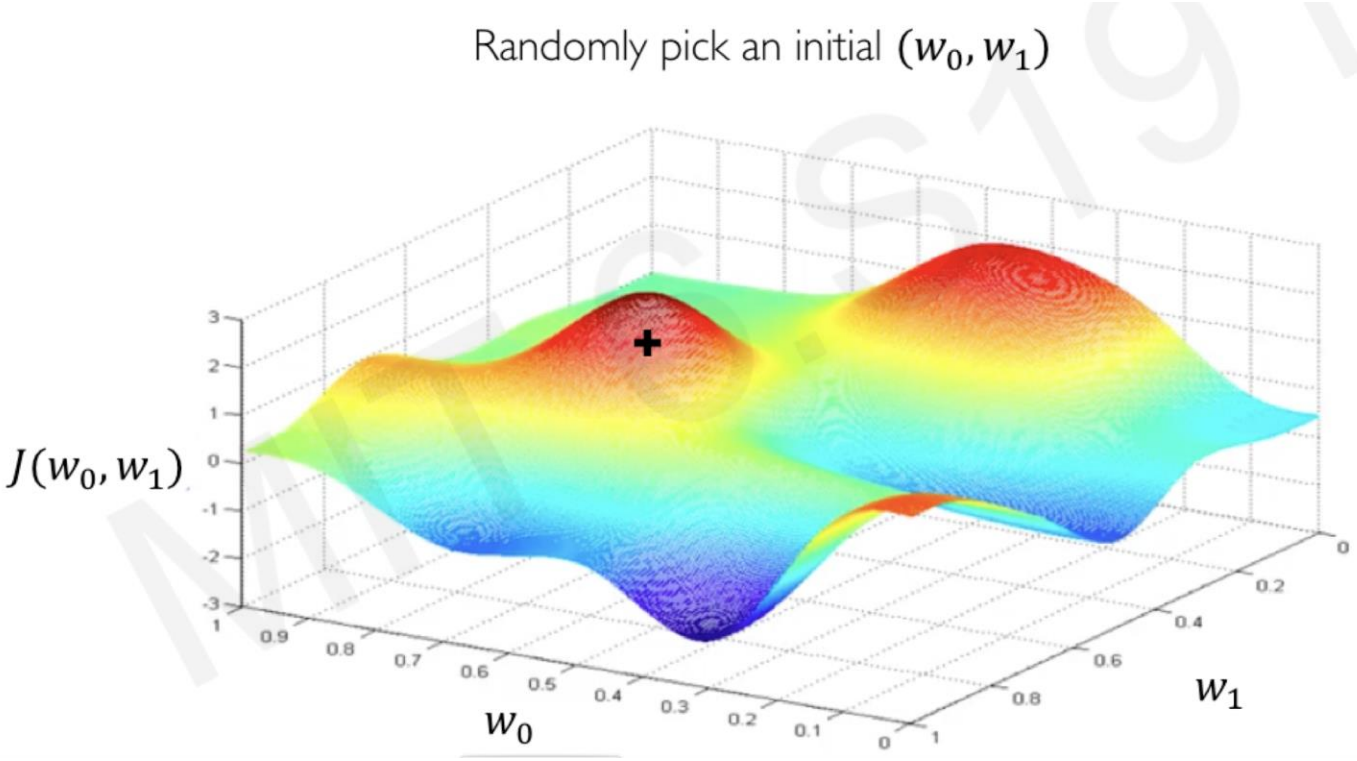
Remember:

Our loss is a function of the network weights!



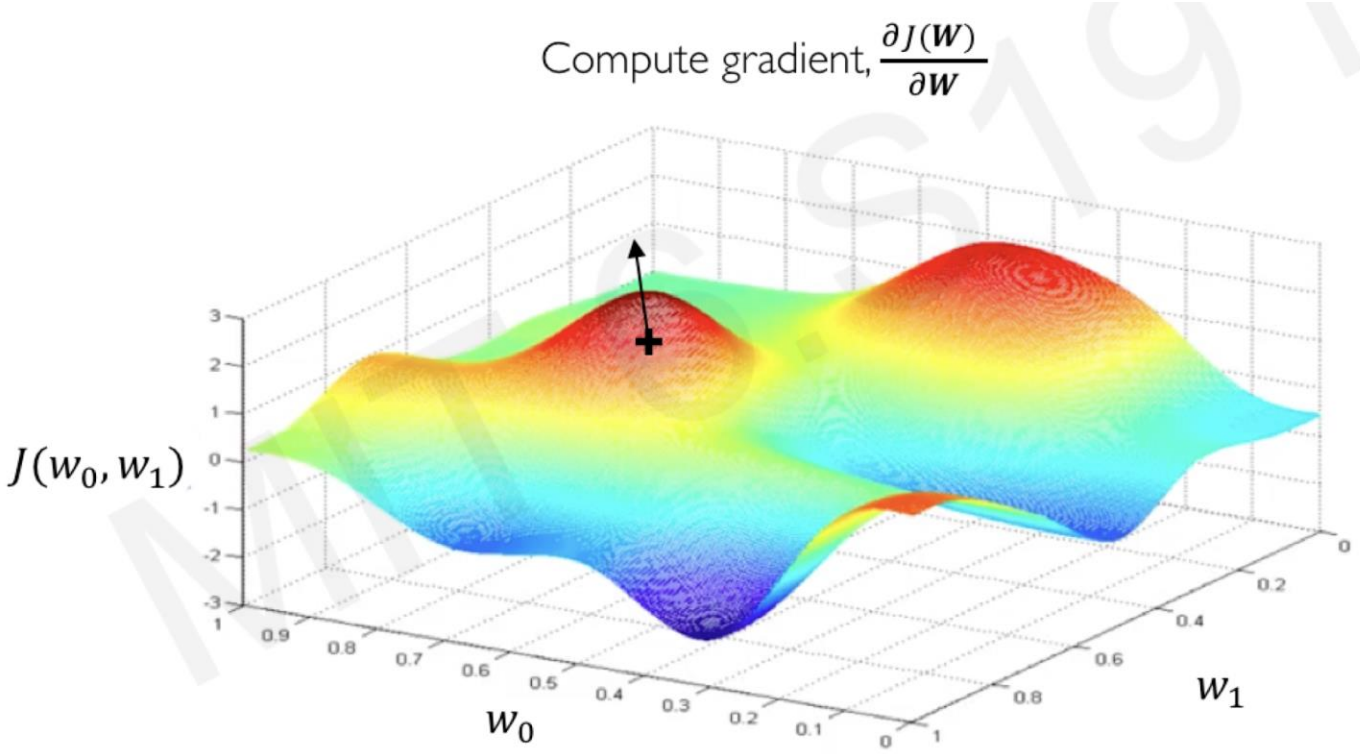
Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Deep Learning: Gradient Descent <Loss Optimization>



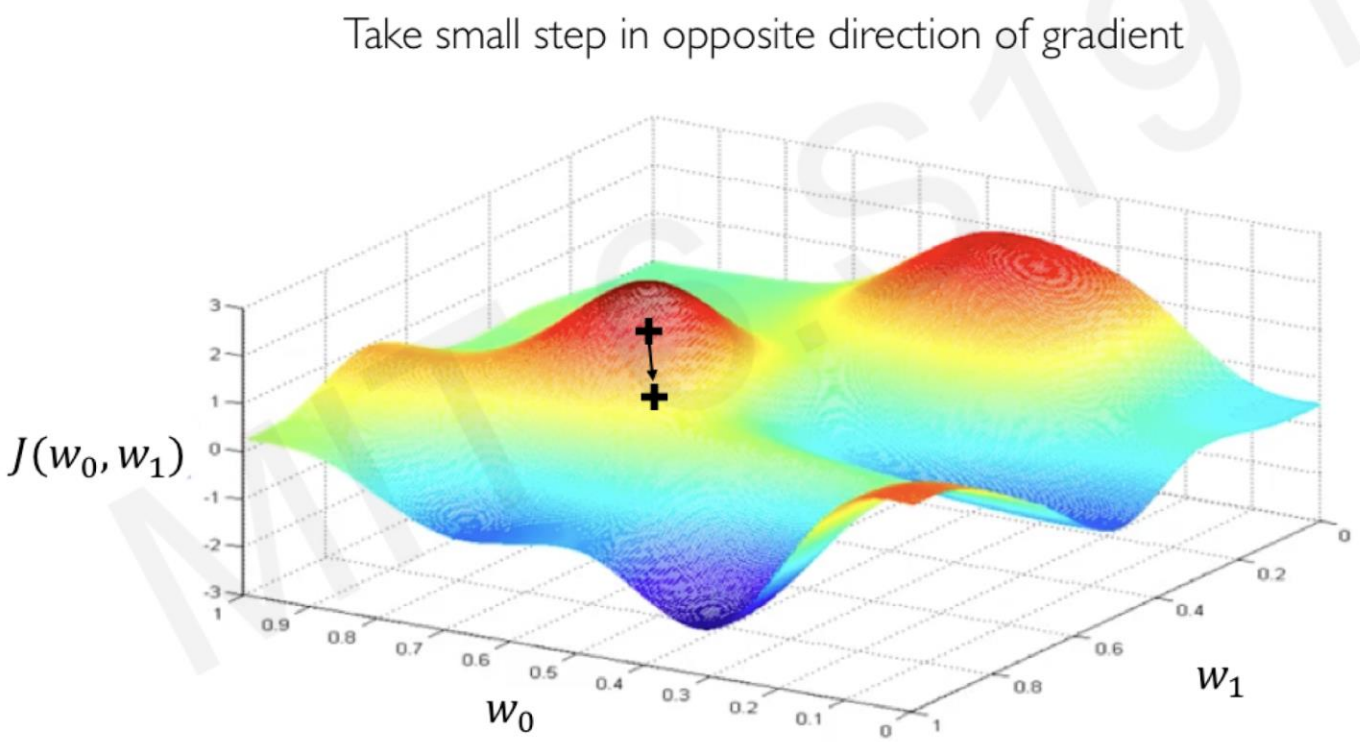
Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Deep Learning: Gradient Descent <Loss Optimization>



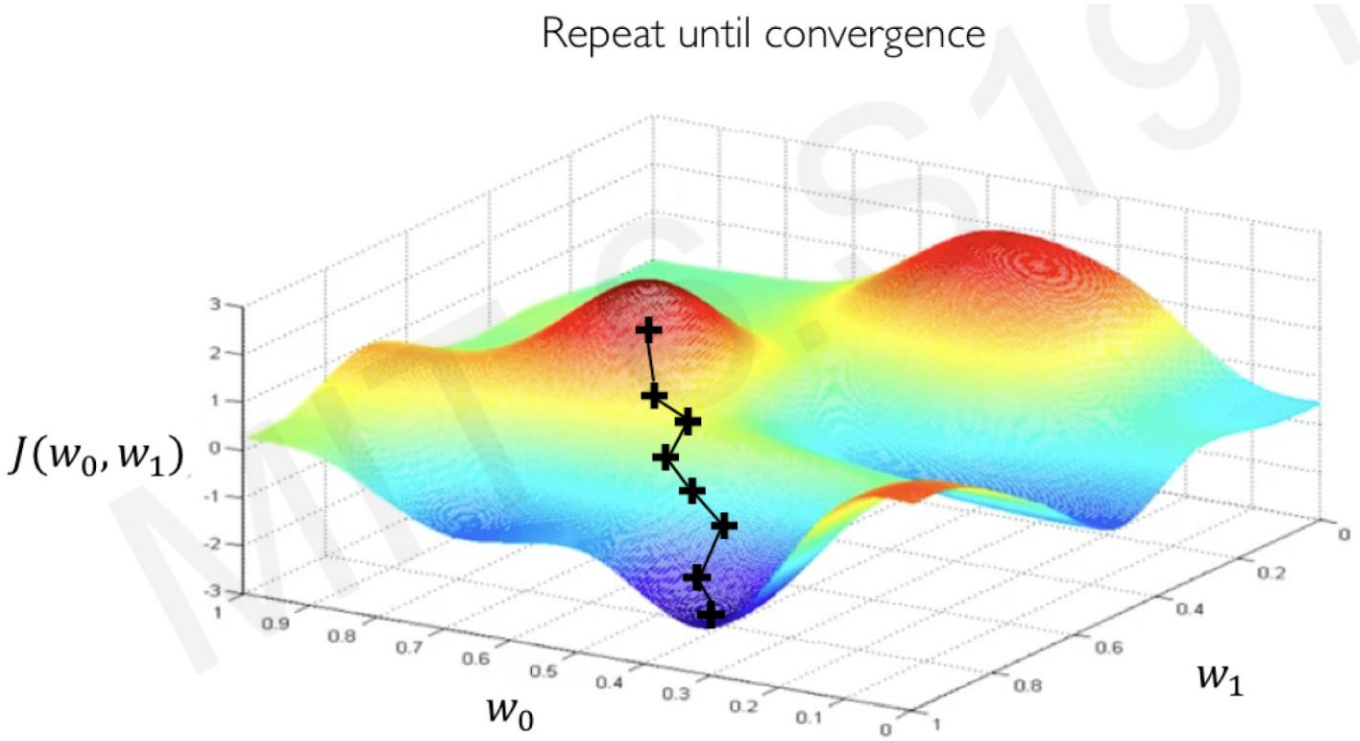
Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Deep Learning: Gradient Descent <Loss Optimization>



Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Deep Learning: Gradient Descent <Loss Optimization>



Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Deep Learning: Gradient Descent <Loss Optimization>

Gradient Descent

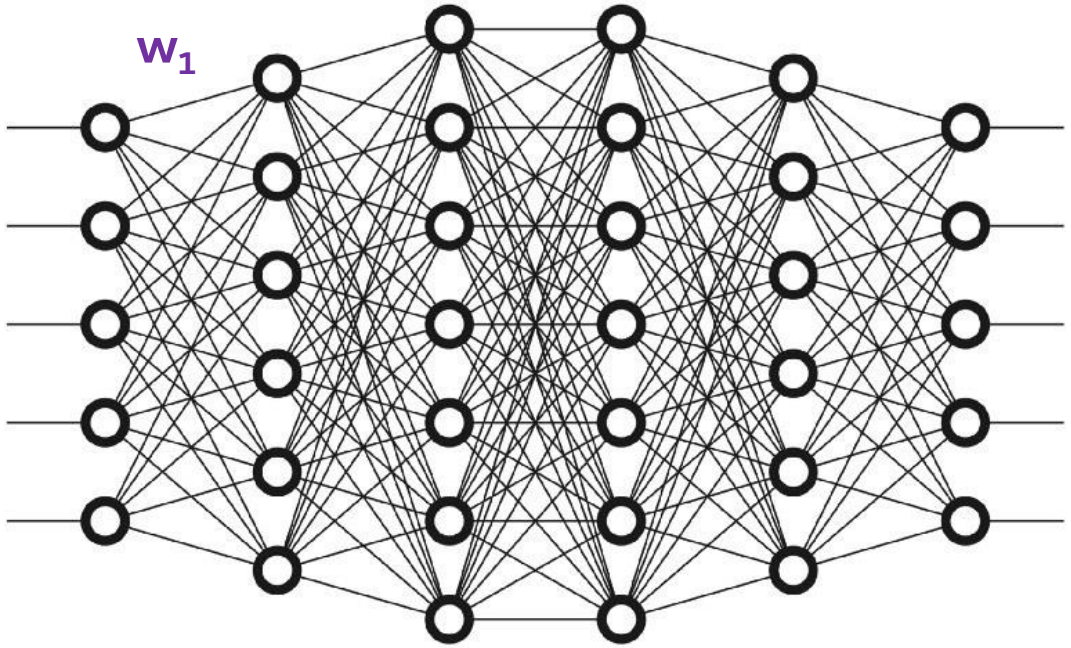
Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Problem: How to compute the gradient for a deep neural network?

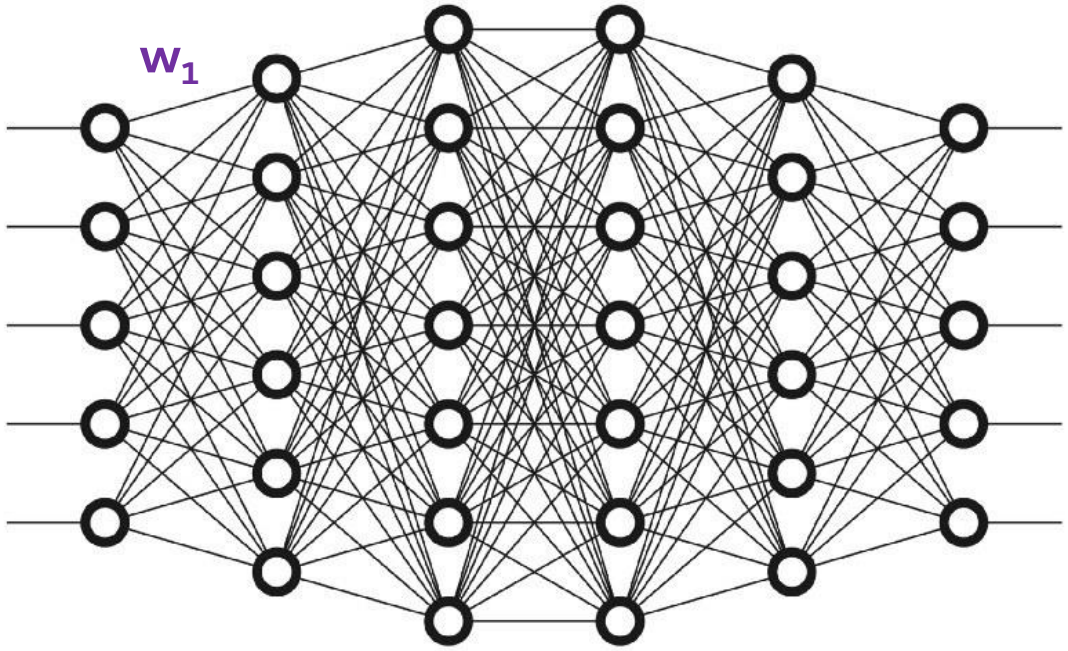
Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Deep Learning: Gradient Descent



$$\frac{\partial J(W)}{\partial w1} ?$$

Deep Learning: Backpropagation



Backpropagation is the solution

$$\frac{\partial J(W)}{\partial w1}?$$

Deep Learning: Backpropagation

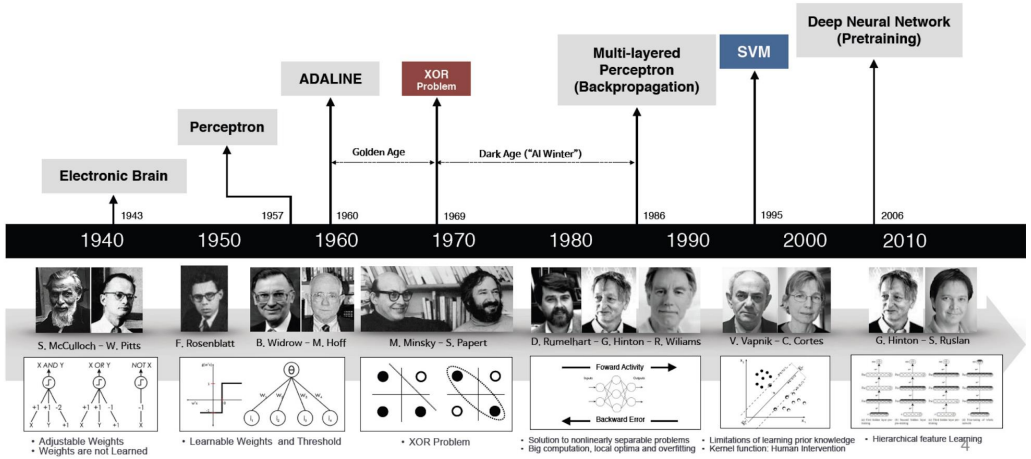
- **Backpropagation** is one of the reasons that made training deep neural network possible.
- **Backpropagation** is basically based on **chain rule.**

The Chain Rule If f and g are both differentiable and $F = f \circ g$ is the composite function defined by $F(x) = f(g(x))$, then F is differentiable and F' is given by the product

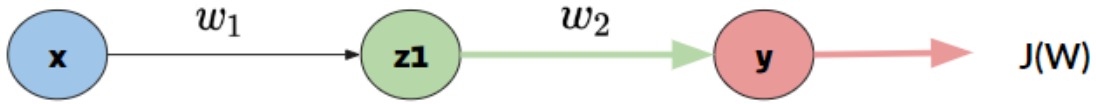
$$F'(x) = f'(g(x))g'(x)$$

In Leibniz notation, if $y = f(u)$ and $u = g(x)$ are both differentiable functions, then

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

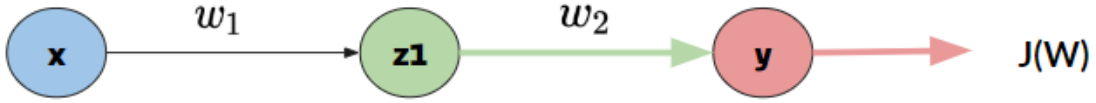


Deep Learning: Backpropagation <Example>



$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial y} * \frac{\partial y}{\partial w_2}$$

Deep Learning: Backpropagation <Example>



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial y} * \frac{\partial y}{\partial w_1}$$

Re-applying the chain rule

$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial y} * \frac{\partial y}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

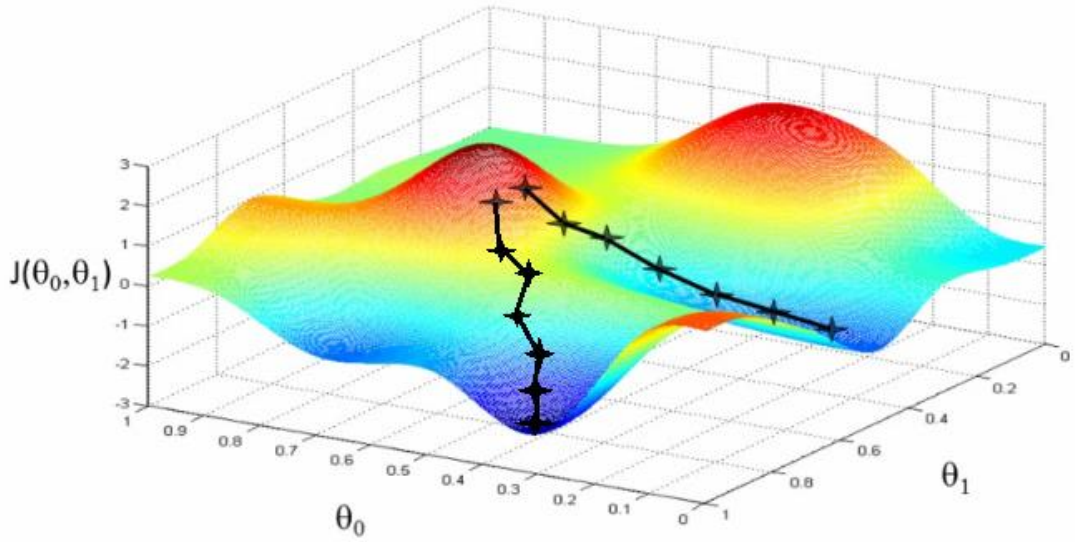
Deep Learning: What is the difference between Gradient Descent and Backpropagation?

- **Gradient descent** is an optimization algorithm for minimizing the loss of a predictive model with regard to a training dataset.
- **Back-propagation** is an automatic differentiation algorithm for calculating gradients for the weights in a neural network graph structure.
- **Gradient descent** and the **back-propagation of error** algorithms together are used to train neural network models.

	Backpropagation	Gradient Descent
Definition	An algorithm for calculating the gradients of the cost function	Optimization algorithm used to find the weights that minimize the cost function
Requirements	Differentiation via the chain rule	•Gradient via Backpropagation •Learning rate
Process	Propagating the error backwards and calculating the gradient of the error function with respect to the weights	Descending down the cost function until the minimum point and find the corresponding weights

Deep Learning: Gradient Descent Algorithms

- **Gradient Descent**
- Stochastic Gradient Descent **(SGD)**
- **Adam** (Adaptable moment estimation)
- Adaptive Gradient Algorithm (**AdaGrad**)
- Root Mean Square Propagation (**RMSProp**)

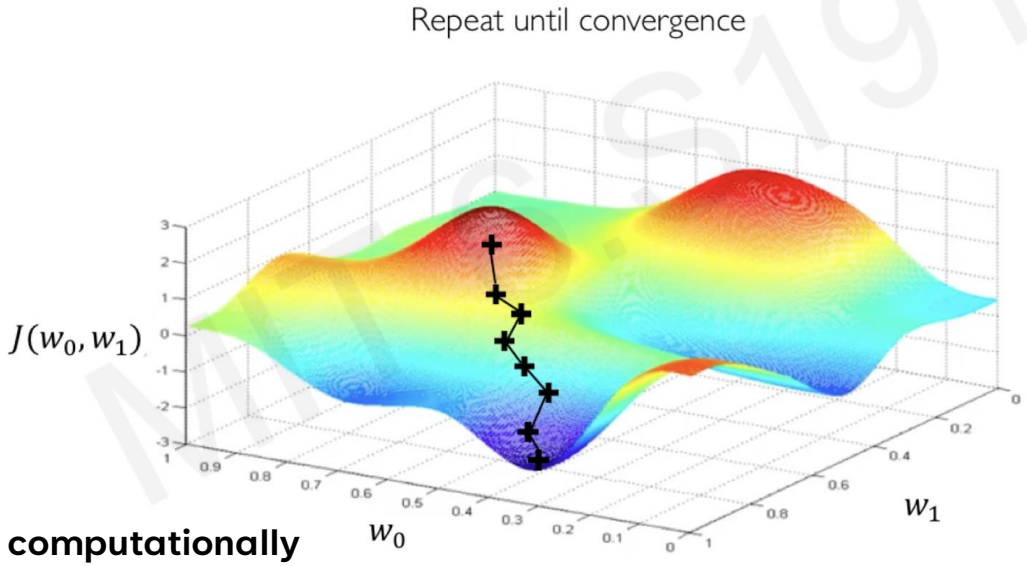


Deep Learning: Stochastic Mini-batch Gradient Descent

- **Problem:** Imagine you are building a deep learning model for image classification (Millions of images), you cannot feed all these images directly to the model and start training on all these images at once (computing the loss function), even if you have a solid/strong computer, this would be challenging if not impossible to train the model in that way.
- **Solution:** Feeding Mini batches of the initial dataset to the model.

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights



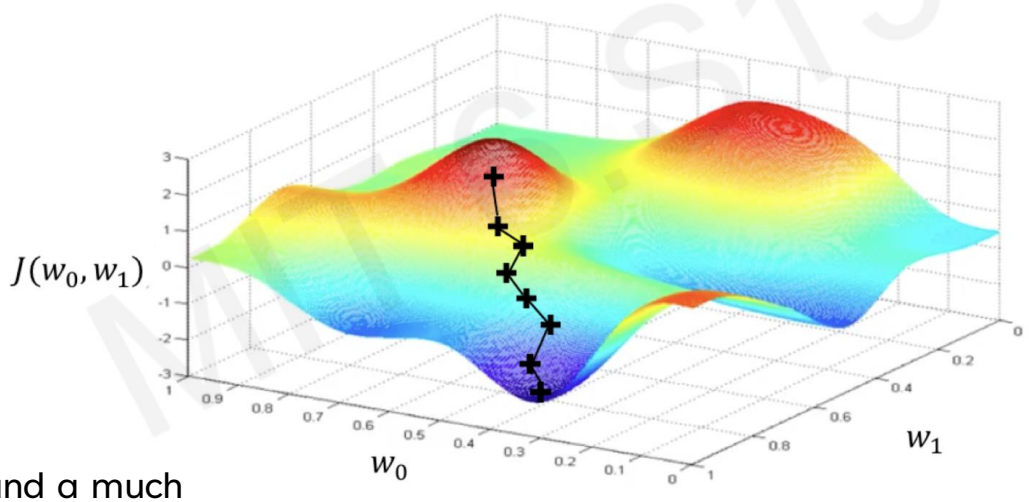
Can be very **computationally intensive** to compute!

Deep Learning: Stochastic Mini-batch Gradient Descent

Algorithm

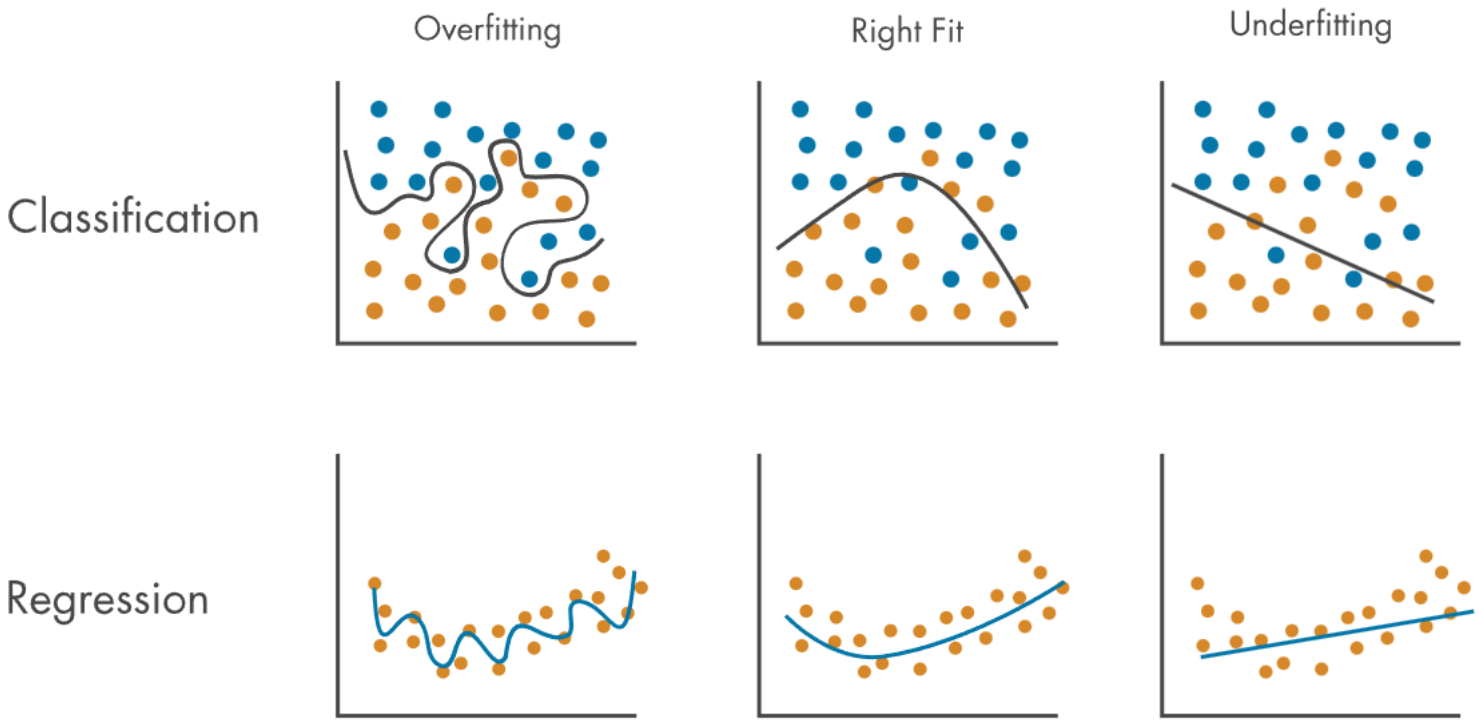
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

Repeat until convergence



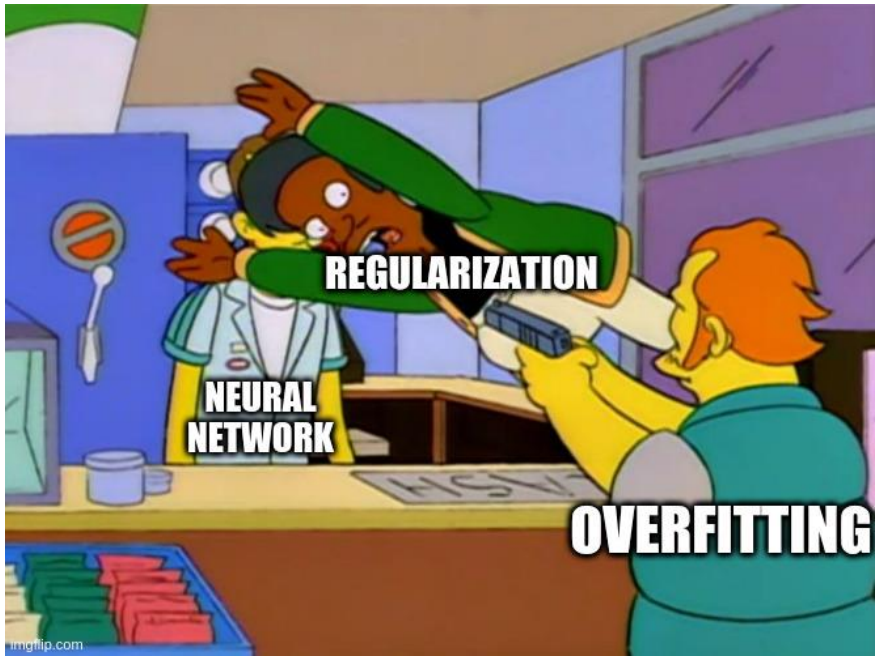
Fast to compute and a much better estimate of the true gradient!

Deep Learning: The Problem of Overfitting



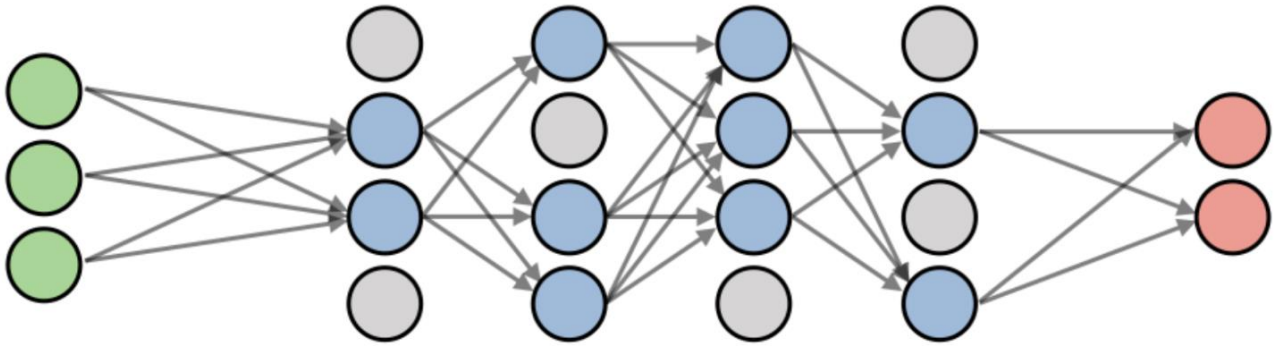
Deep Learning: Regularization

- **What is Regularization?**
  - Technique that constraints our optimization problem to discourage complex models
- **Why do we need it ?**
  - To improve generalization of our model on unseen data



Regularization: Dropout

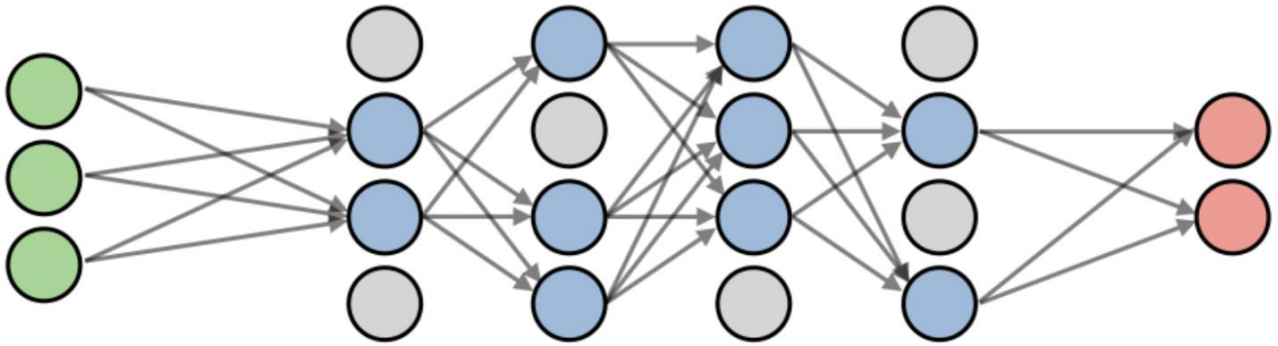
- Dropout is one of the simplest techniques used to regularize the models and prevent overfitting.
- The idea is simple, you set randomly some neurons(activations) to 0.



Regularization: Dropout

- During training, randomly set some activations to 0
  - Typically, 'drop' 50% of neurons/activations in a layer
  - Forces the network to not rely on any node

```
tf.keras.layers.Dropout(p=0.5)
```



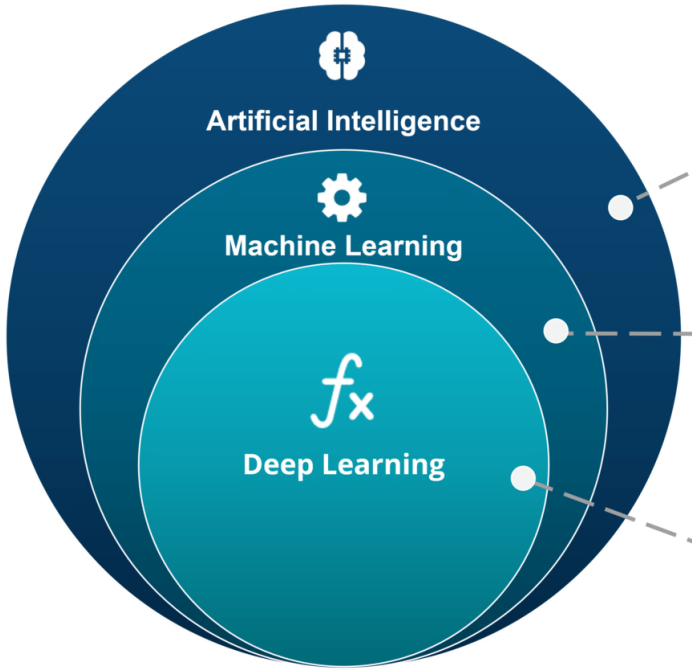
Deep Learning: Frameworks

Deep Learning: Perceptron and Artificial Neural Network <DEMO>

DEMO: Session 2 – Perceptron and Artificial Neural Networks

From Classical Machine Learning to Deep Learning

From Classical Machine Learning to Deep Learning

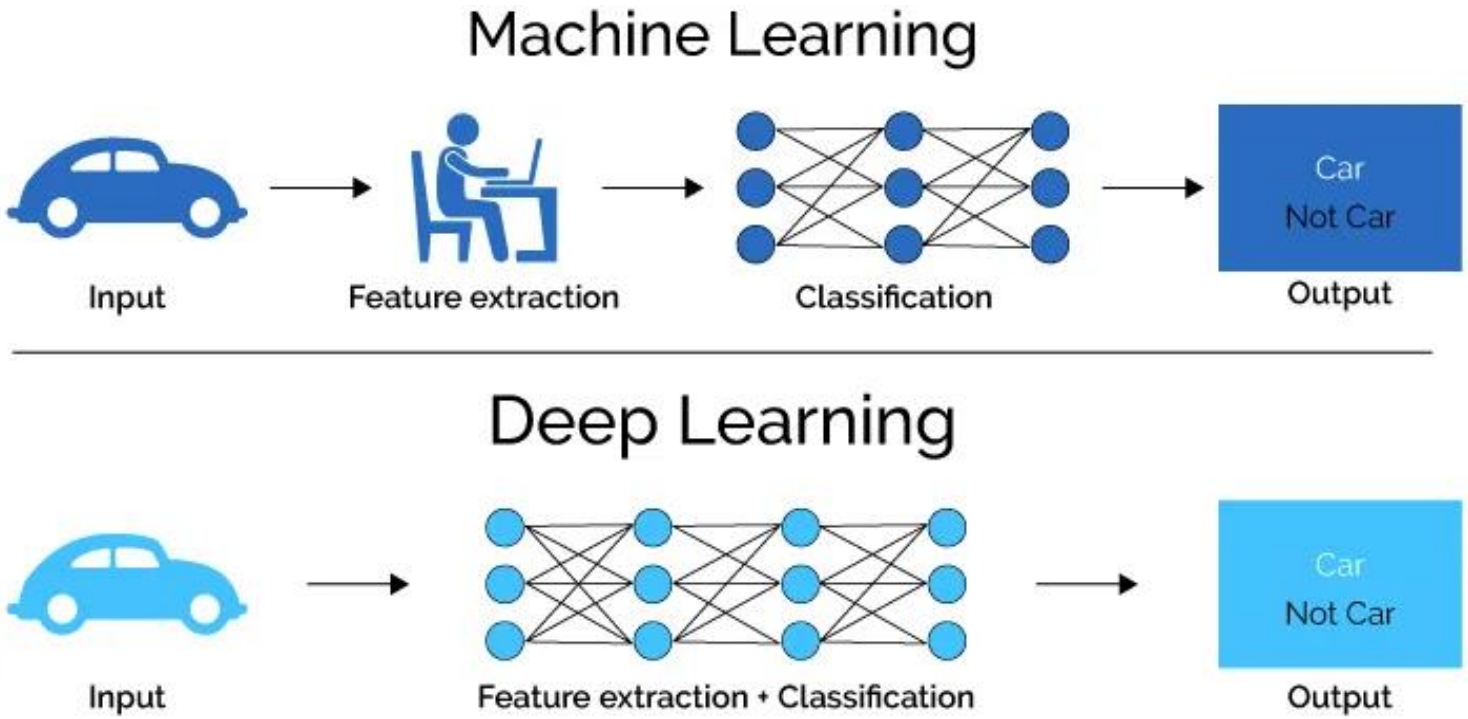


ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour

MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience

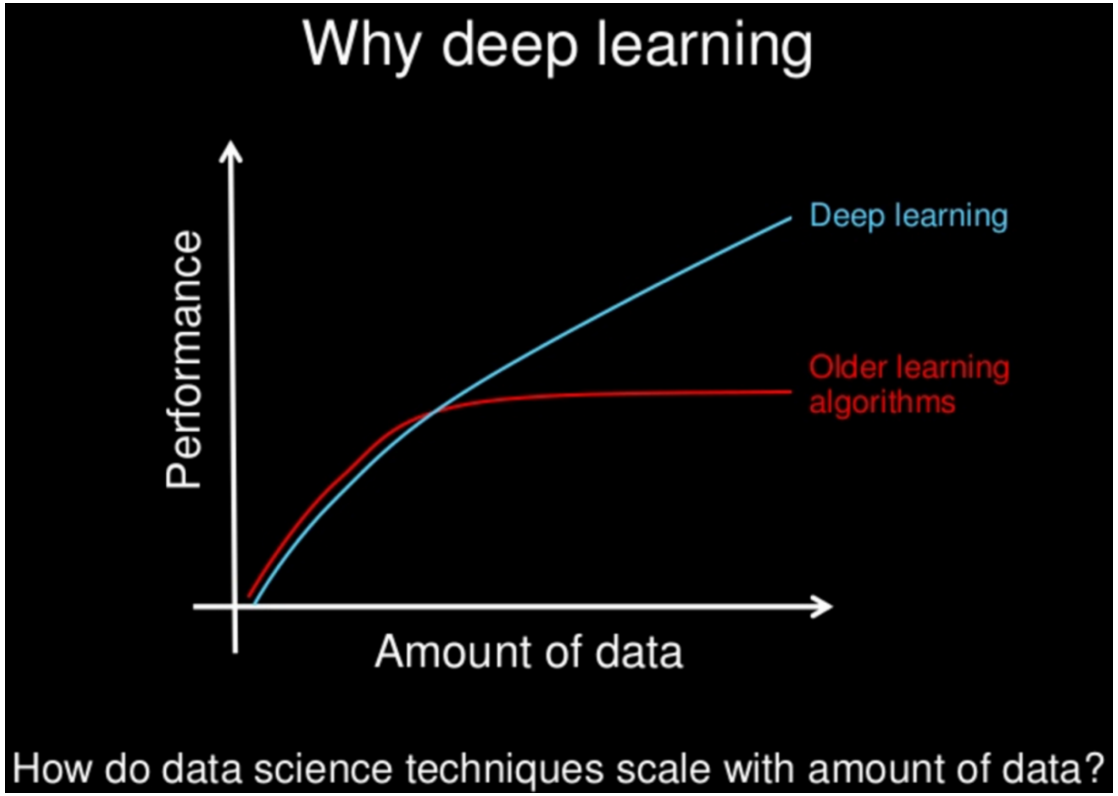
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

From Classical Machine Learning to Deep Learning



Feature extraction: Machine learning vs Deep learning

From Classical Machine Learning to Deep Learning



From Classical Machine Learning to Deep Learning

Aspect	Classical machine learning	Deep learning
Data size	Can perform well with smaller datasets.	Often requires large amounts of data for effective training.
Feature Engineering	Often requires domain knowledge for manual feature creation.	Automatically learns hierarchical features.
Computation	Computationally less intensive compared to deep learning.	Requires powerful hardware, like GPUs, for efficient training.
NLP + Computer vision + Audio/Speech recognition	Limited and less effective in these tasks	Highly effective and represents state-of-the-art for these tasks
Human-like Learning	Relies on human-engineered features and rules.	Can automatically learn complex patterns like humans.

Table. The comparative table between classical machine learning (ML) and deep learning (DL) based on various aspects

From Classical Machine Learning to Deep Learning: Examples and Case Studies

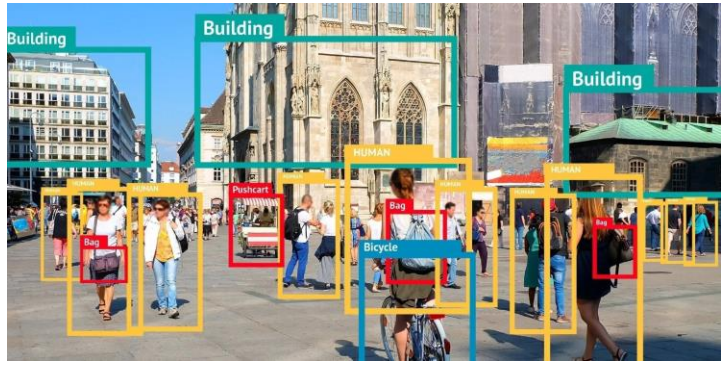
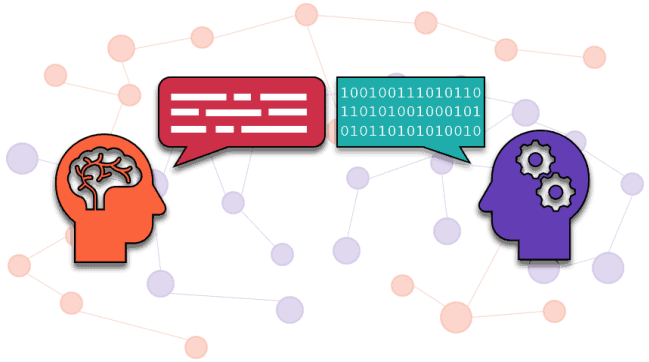


Image Recognition and Computer Vision

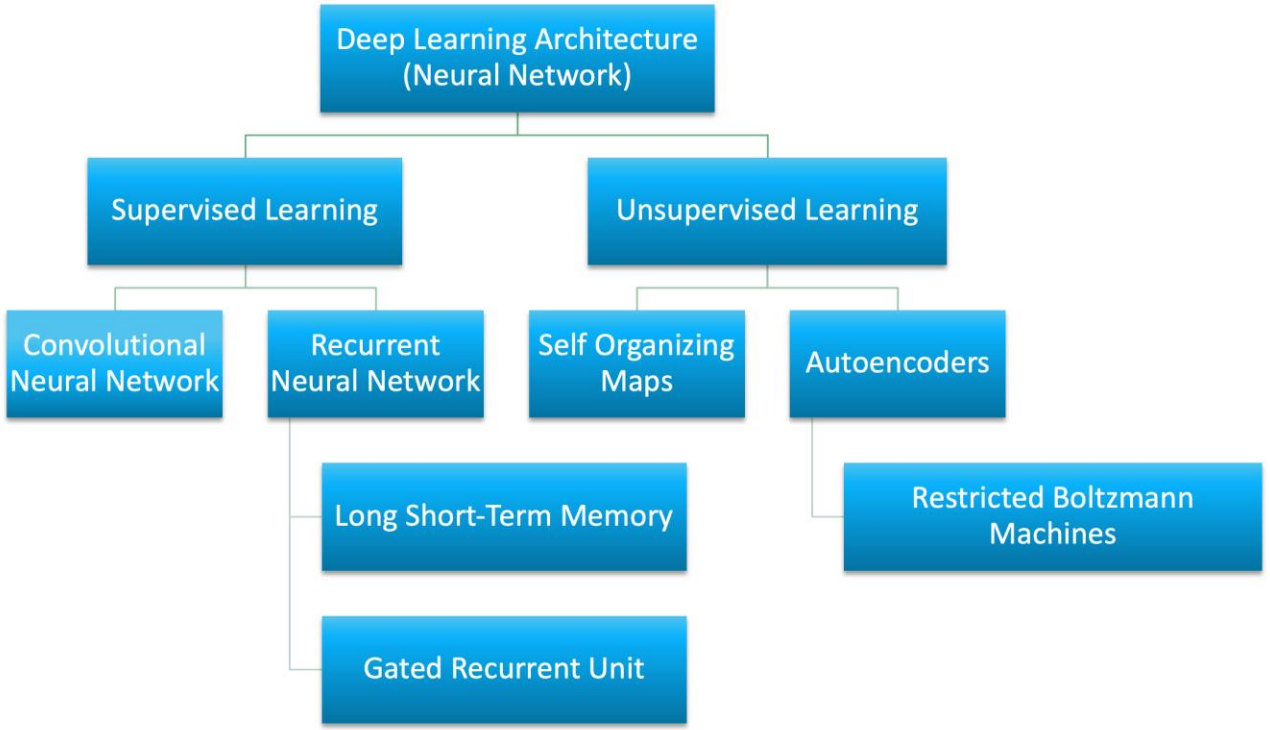


Natural Language Processing (NLP)

Speech Recognition

Healthcare (DeepMind's AlphaFold), Game Playing, Recommendation System, Robotics, ...etc.

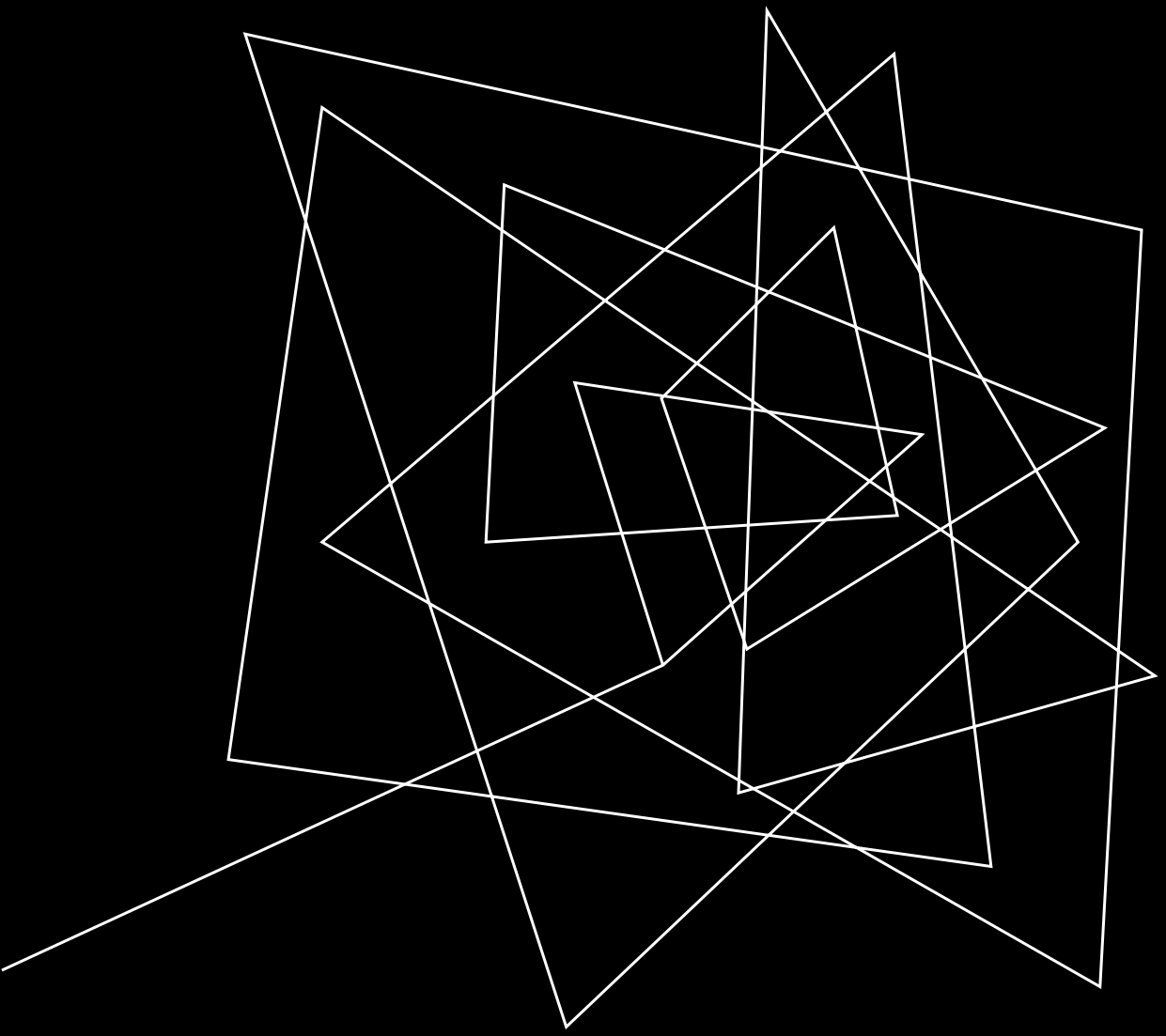
Deep Learning: Architectures



Source: https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/

Deep Learning: TO-DO Project

Classification using ANN: Heart Disease Prediction

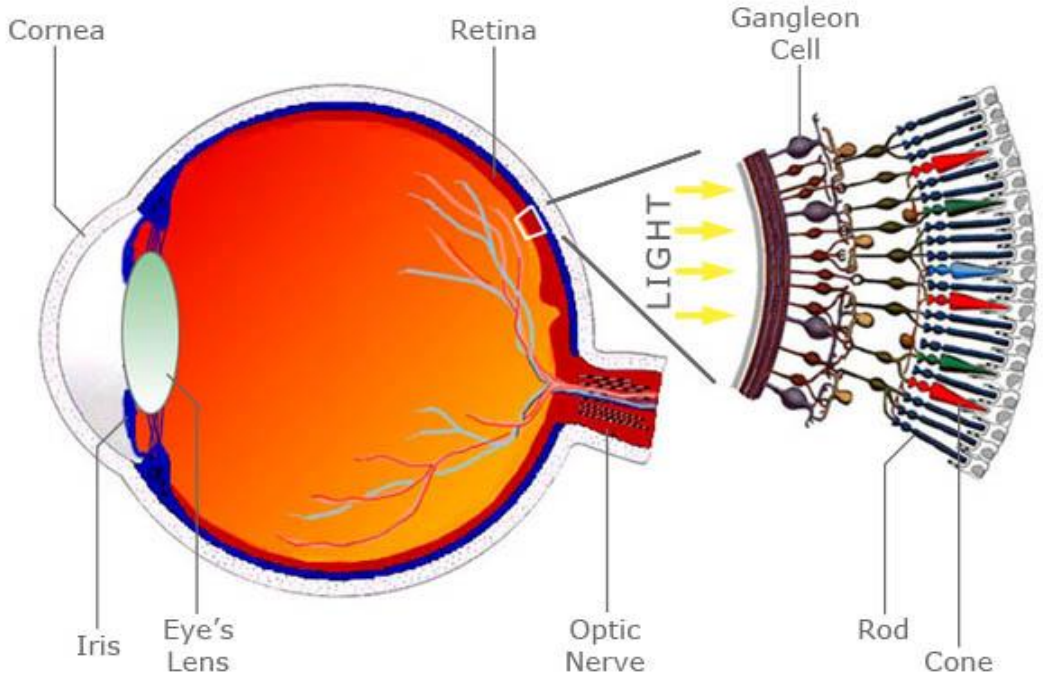


DEEP LEARNING FOR COMPUTER VISION

Convolution Neural Networks (CNNs), CNN Architectures, Transfer Learning, GANs, and Data Augmentation

Introduction: Computer Vision

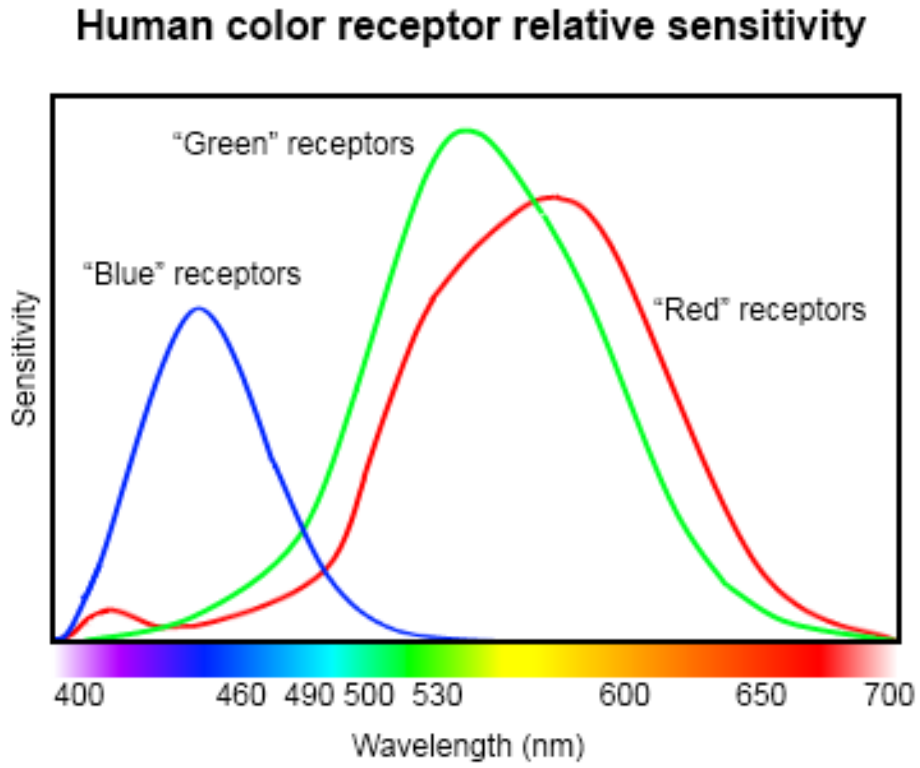
- The human eye (vision) is one of the most important parts that process the information visualized by the human (Information/Image Processing).
- **Idea:** In the human eye, there is what we call "**Cone Cell**", and these cells receive the light from the **Retina**
  - These "Cone" cells are sensitive to some **colors (electrical signals).**
  - These "**Cone**" cells are important because they contribute to the "Color perception/cognition"



Source: https://www.blueconemonochromacy.org/how-the-eye-functions/

Introduction: Computer Vision

- Generally, researchers found that the human being has a **Tri-color vision**.
- The cone cells are sensitive to **RED**, **GREEN**, and **BLUE**
- This can be seen as a projection on a 3-D space, each wavelength can be represented using 3 values (a ratio) **Red**, **Green**, and **Blue**



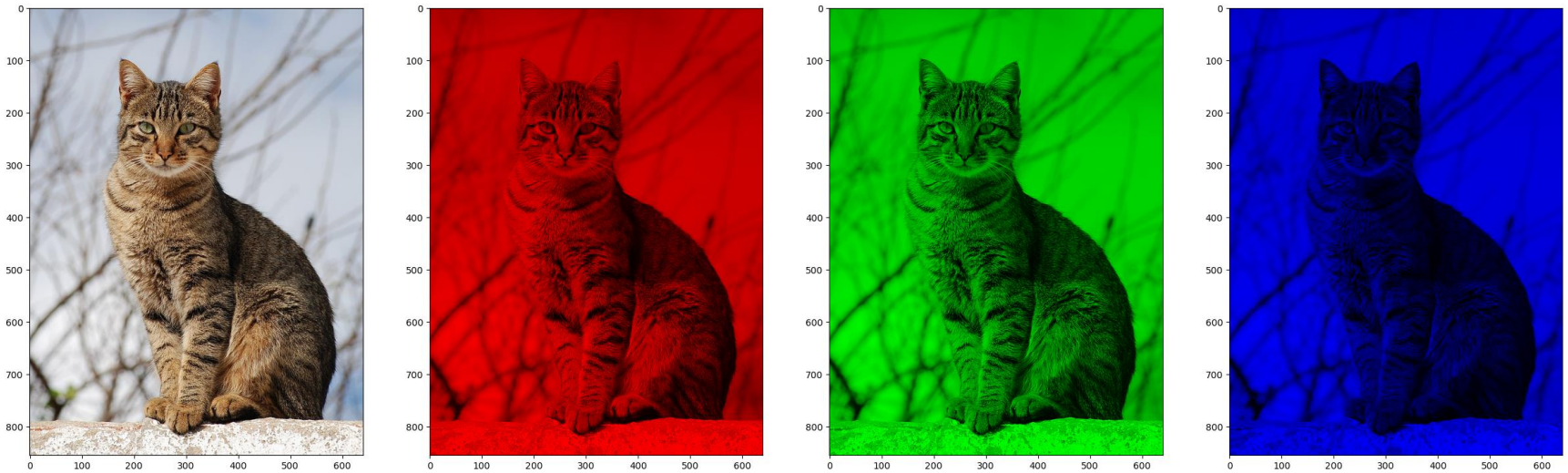
Introduction: Computer Vision

The computers use **RGB** representation to represent images. Each picture you see on your device is represented under a 3D matrix representing the values of **Red**, **Blue**, and **Green** for each pixel.

Source: https://fr.mathworks.com/help/images/image-types-in-the-toolbox.html
You can try it here: https://pixspy.com/

Introduction: Computer Vision

Question: How to extract **RGB** values from an image?



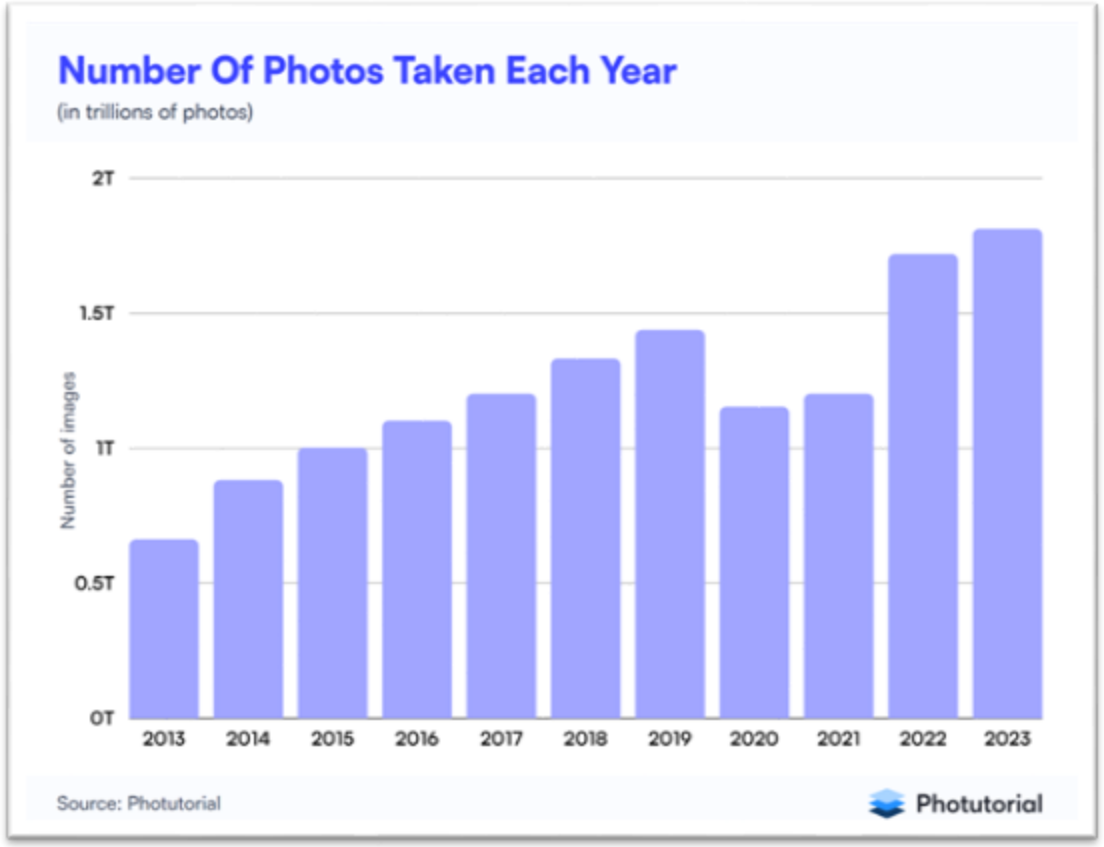
Demo: Session 3 – Computer Vision and CNN

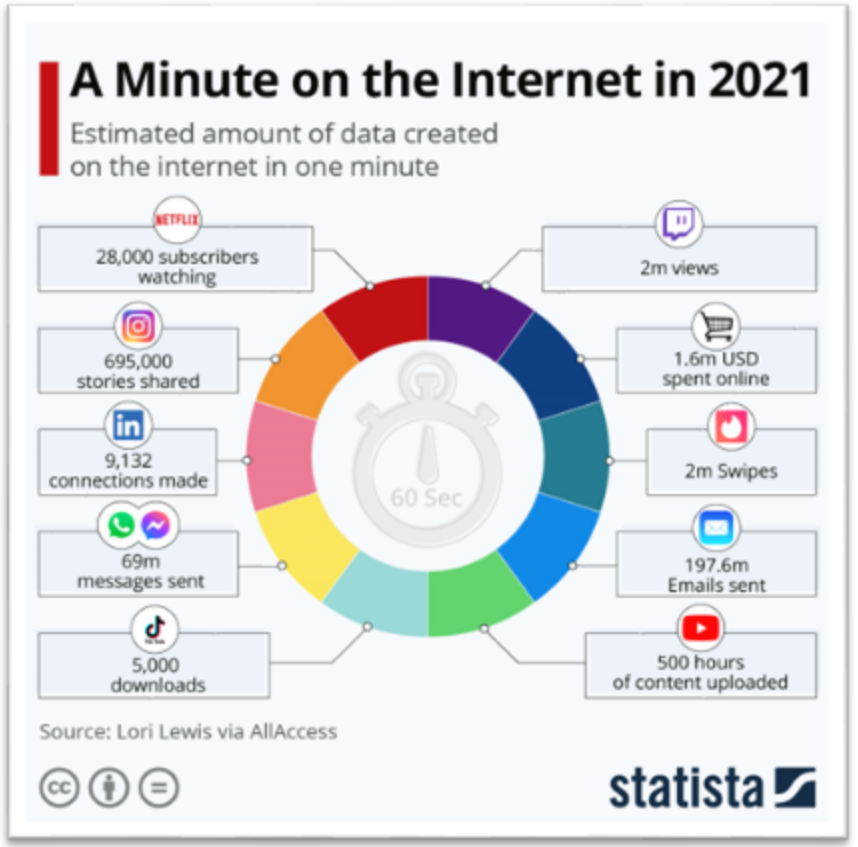
Introduction: Computer Vision
<DATA EXPLOSION>



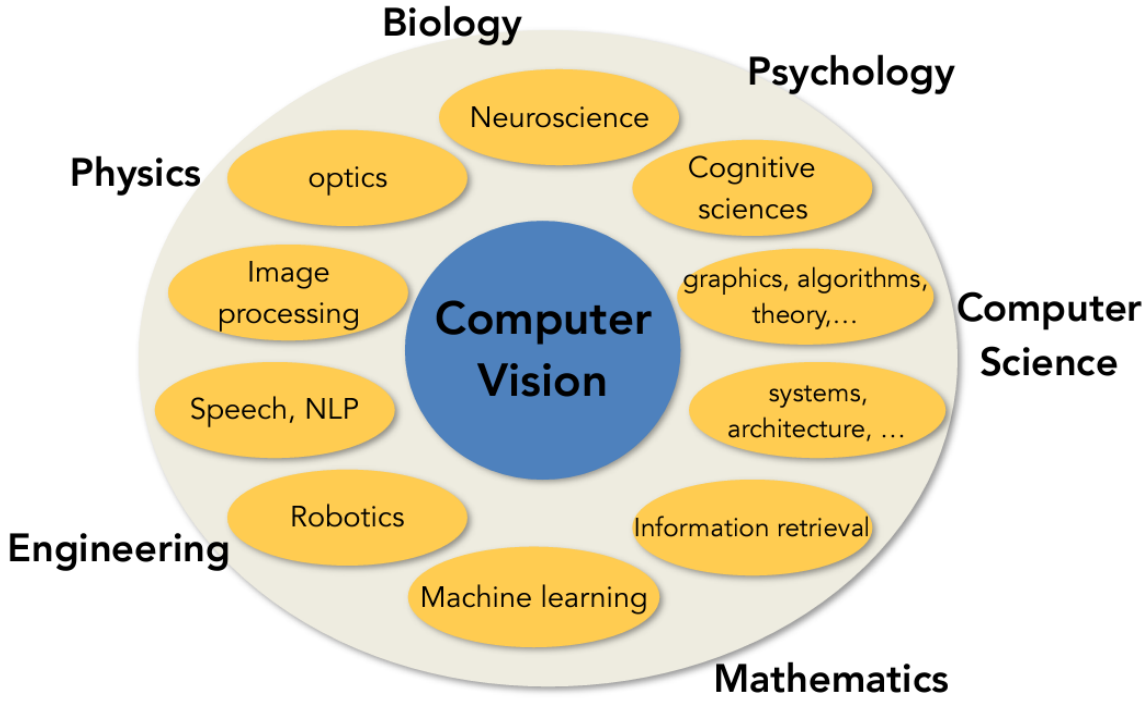



Introduction: Computer Vision
<DATA EXPLOSION>

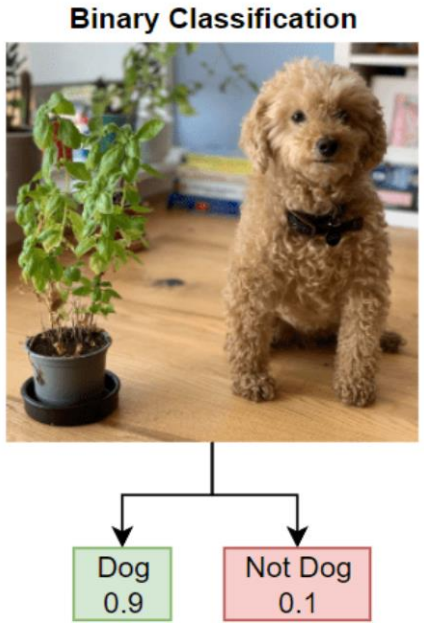


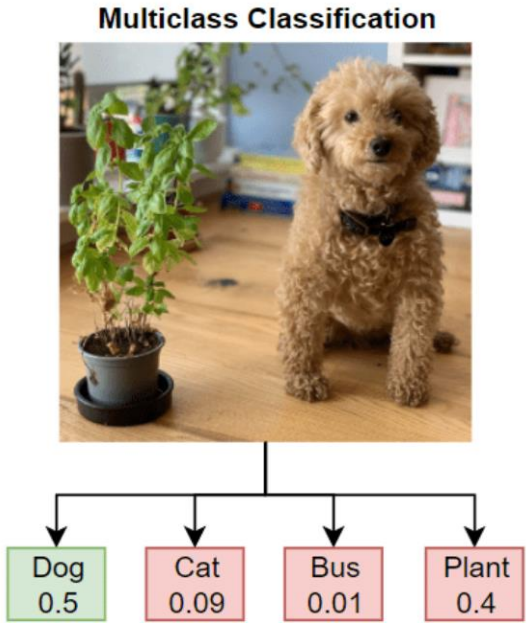


Introduction: Computer Vision <Interdisciplinary Field>



Computer Vision: Image Classification





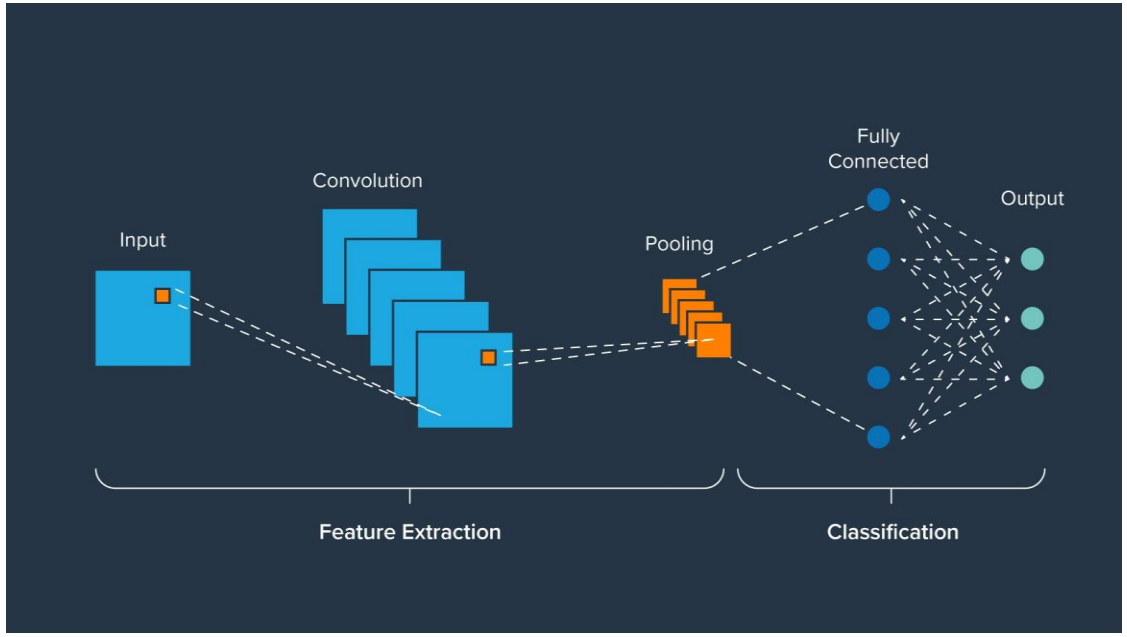


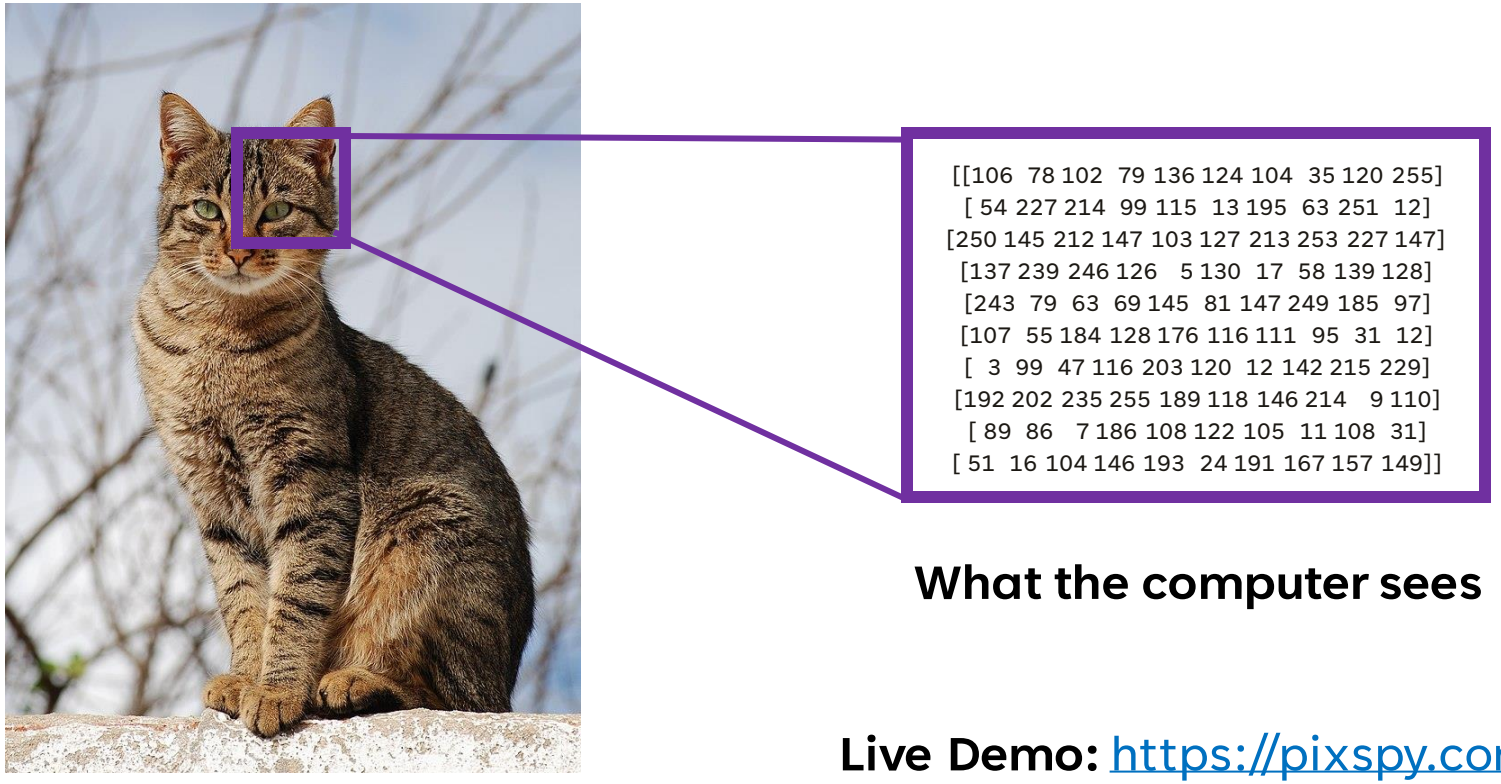
Image Classification: A Core Task in Computer Vision

Assume given set of discrete labels = {Dog, Cat, Truck, Plane, ...etc.}

→ **Cat**

Image Classification: A Core Task in Computer Vision

The Problem: Semantic Gap



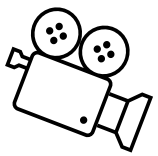
What the computer sees

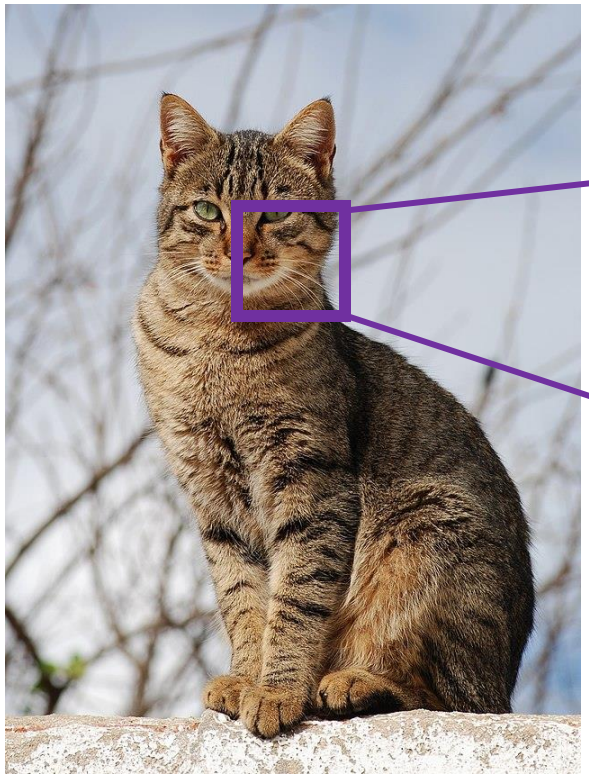
An image is just a big grid of numbers between [0, 255]: e.g., 800 * 600 * 3 (3 channels RGB)

Live Demo: https://pixspy.com/

Image Classification: A Core Task in Computer Vision

Challenges: Viewpoint variation





```
[[106  78 102  79 136 124 104  35 120 255]
 [ 54 227 214  99 115  13 195  63 251  12]
 [250 145 212 147 103 127 213 253 227 147]
 [137 239 246 126   5 130  17  58 139 128]
 [243  79  63  69 145  81 147 249 185  97]
 [107  55 184 128 176 116 111  95  31  12]
 [  3  99  47 116 203 120  12 142 215 229]
 [192 202 235 255 189 118 146 214   9 110]
 [ 89  86   7 186 108 122 105  11 108  31]
 [ 51  16 104 146 193  24 191 167 157 149]]
```

What the computer sees

All the pixels **change** when the **camera moves**. But somehow it is **still representing** the same object, "**Cat**". Our **algorithms** need to be **robust** and **adapted** to this.

Image Classification: A Core Task in Computer Vision

Other Challenges: Illumination and Deformation

Image Classification: A Core Task in Computer Vision

Other Challenges: Occlusion and Background clutter

Other Challenges: Interclass variation

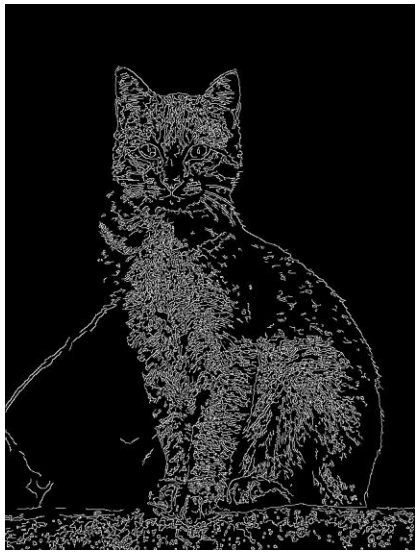
Image Classification: An Image Classifier?

```
def classify_image(image):
    # Some magic here?
    return class_label
```

In **traditional programming** techniques, there is **no obvious** way to hard code the **image classifier** algorithm.

Image Classification: Attempts have been made!

Find edges



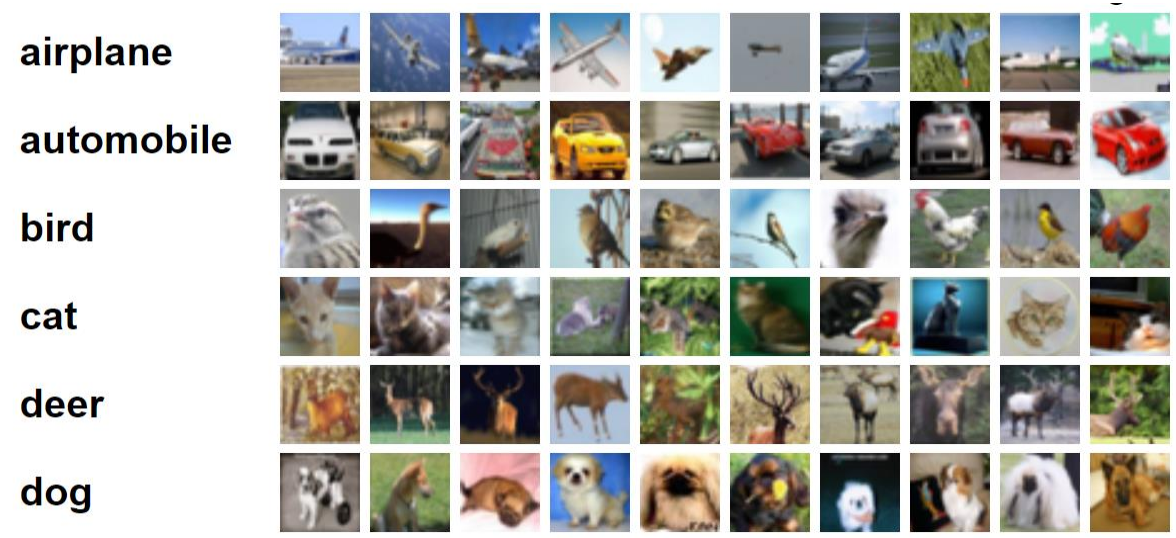
Find corners

Up, Down, Left, Right

John Canny, "A Computational Approach to Edge Detection", IEEE TPAMI 1986

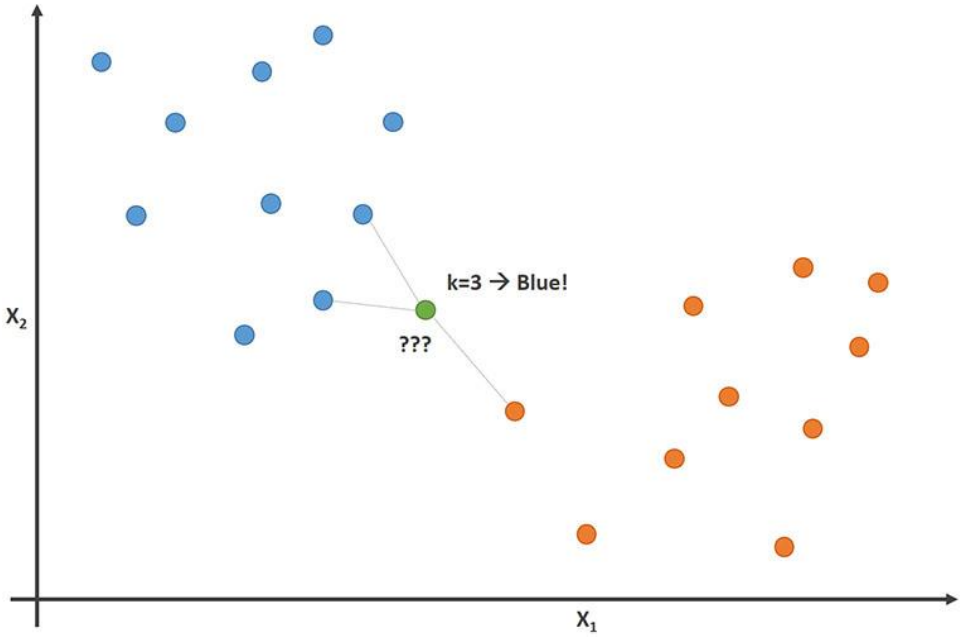
Image Classification: Data-Driven Approach

1. Collect a dataset of images and their corresponding labels
2. Use a machine learning algorithm to train a classifier
3. Test and evaluate the classifier on new (unseen) images



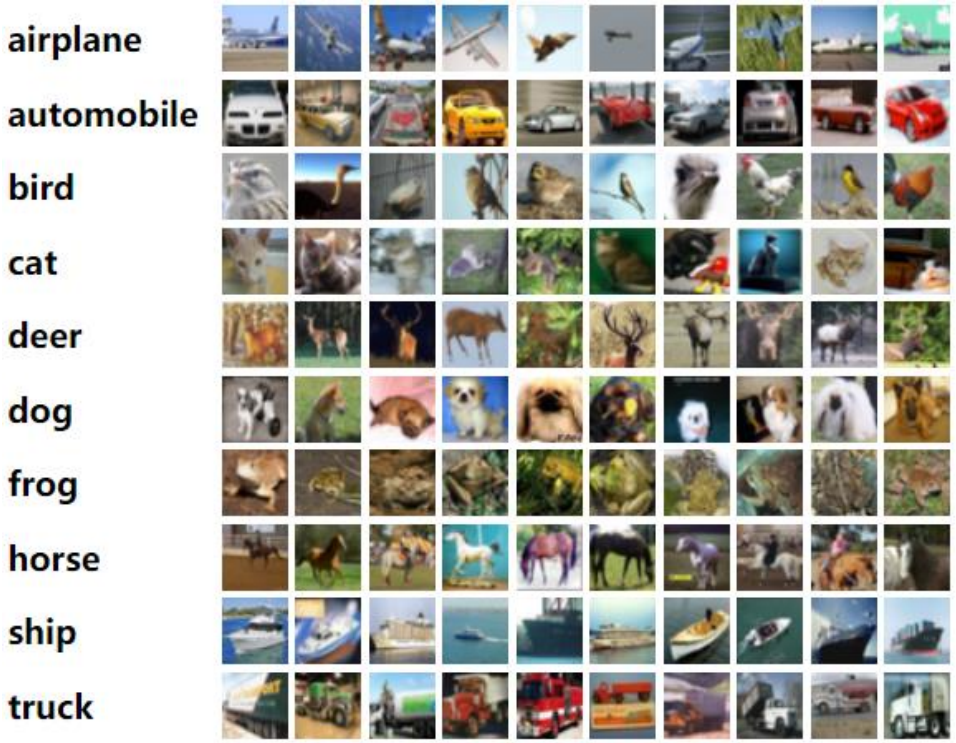
Example of training dataset

Image Classification: First Classifier <Nearest Neighbor>



The k-nearest neighbors' algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

Image Classification: First Classifier <Nearest Neighbor>

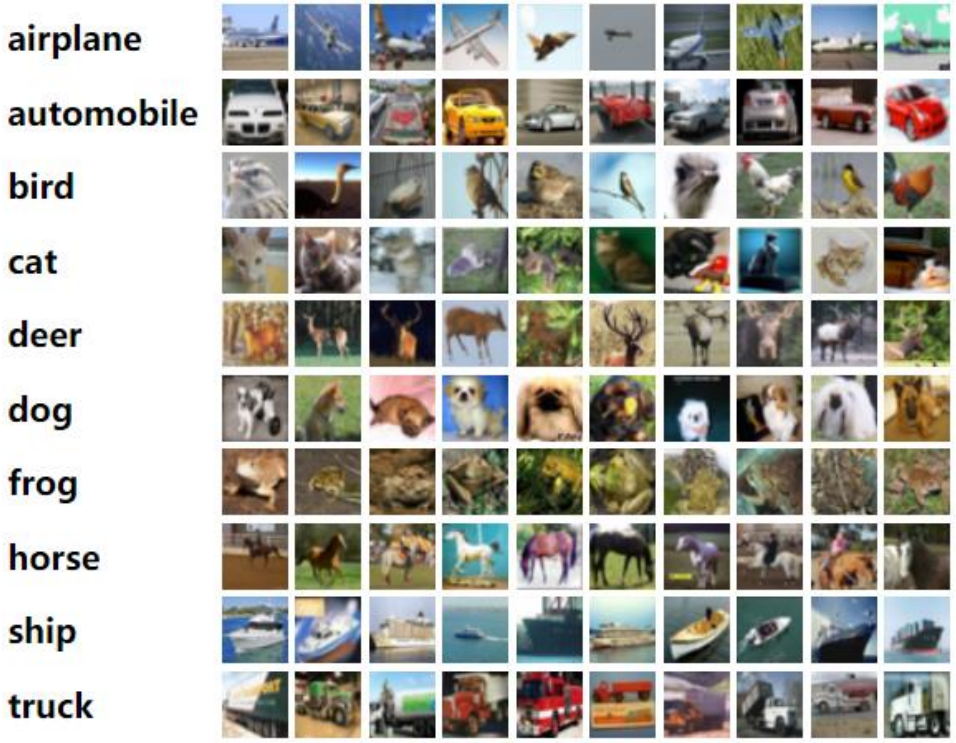


https://www.cs.toronto.edu/~kriz/cifar.html

Example of dataset: CIFAR-10

- The CIFAR-10 dataset (Canadian Institute For Advanced Research) is a collection of images that are commonly used to train machine learning and computer vision algorithms.
- 10 Classes
- 6000 images per class
- 50,000 Training images
- 10,000 Testing images
- 32x32 color images
- They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

Image Classification: First Classifier <Nearest Neighbor>



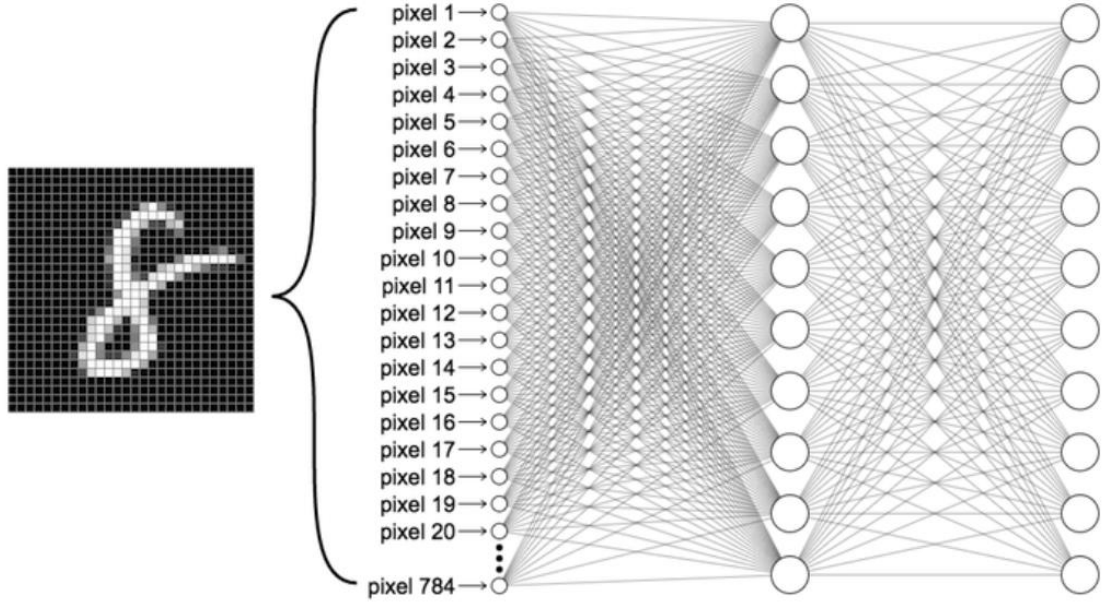
https://www.cs.toronto.edu/~kriz/cifar.html

Test images and nearest neighbors

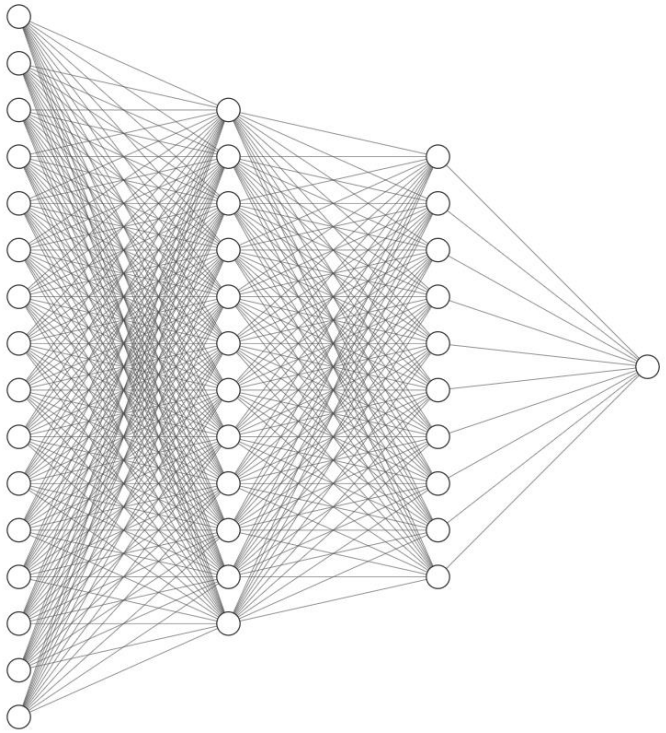
Computer Vision: Why The MLPs (or ANNs) are not Good for Images?

In the last session **(Session 2: Deep Learning)**, we used the MLP to classify the MNIST dataset, A reminder:

- Small images with 28*28 pixels
- The input layer has 784 input neurons
- Two hidden layers
- 10 output neurons
- **The total number of parameters: 235,146**



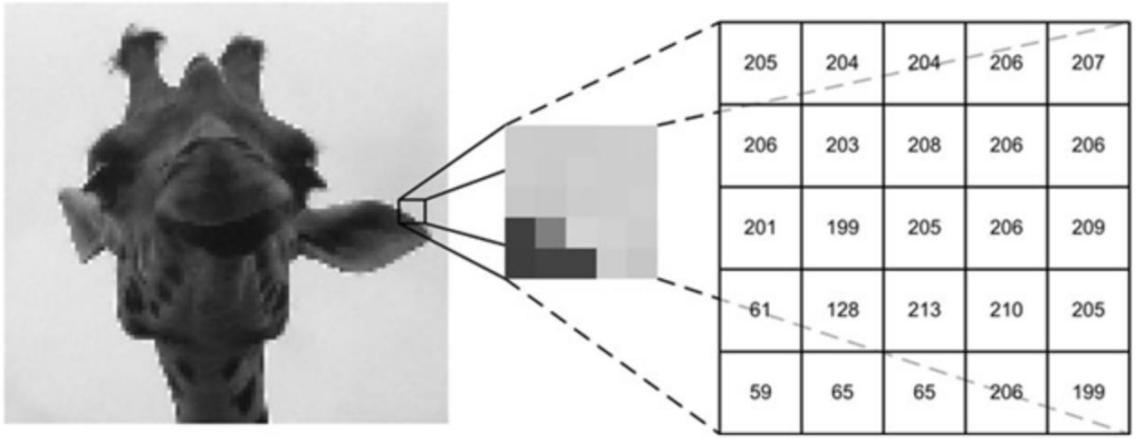
Computer Vision: Why The MLPs (or ANNs) are not Good for Images?

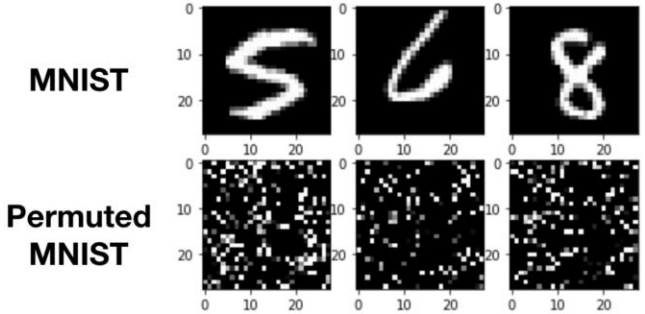


Now imagine, we want to build an MLP for RGB images.

- 855*640*3 Input layer
- 2 Hidden layers with 512 neurons each
- One output neuron
- **The total number of parameters: 840,762,881**

Computer Vision: Why The MLPs (or ANNs) are not Good for Images?



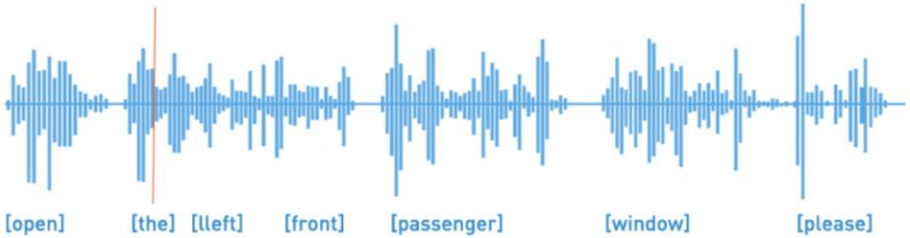


Another problem with MLPs for images, missing neighborhood information (Neighborhood and image structure)

Demo: Session 3 – Computer Vision and CNN

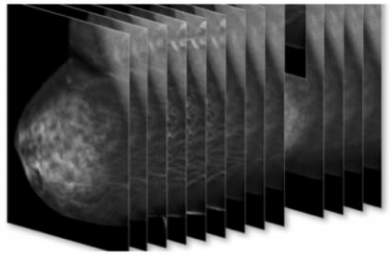
Computer Vision: The Importance of Signal Structure

1D

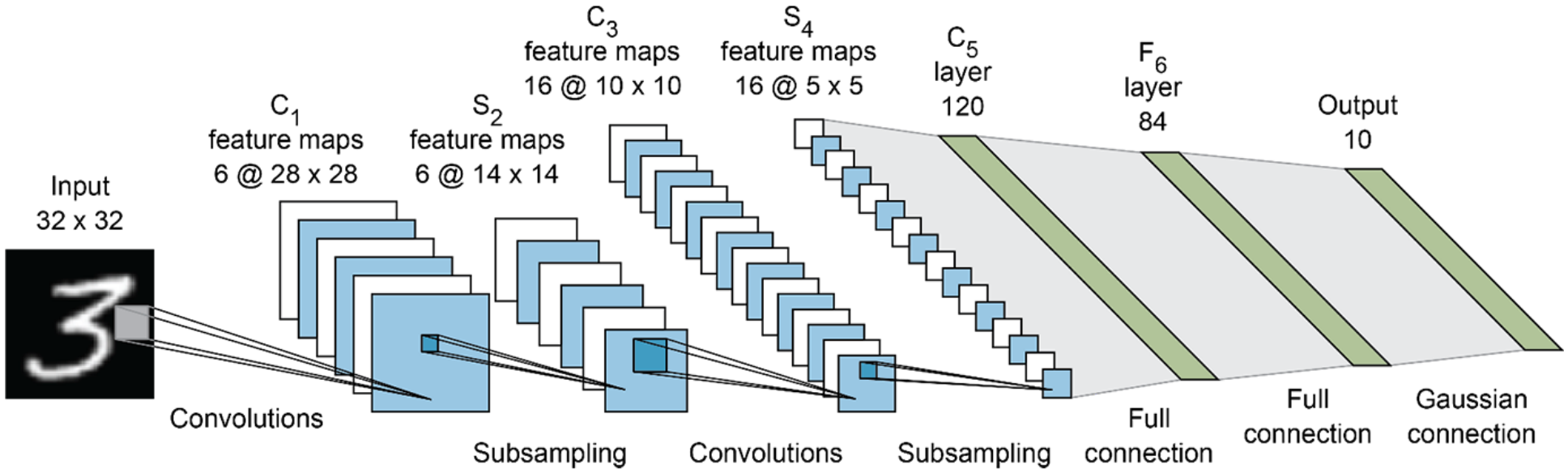


2D

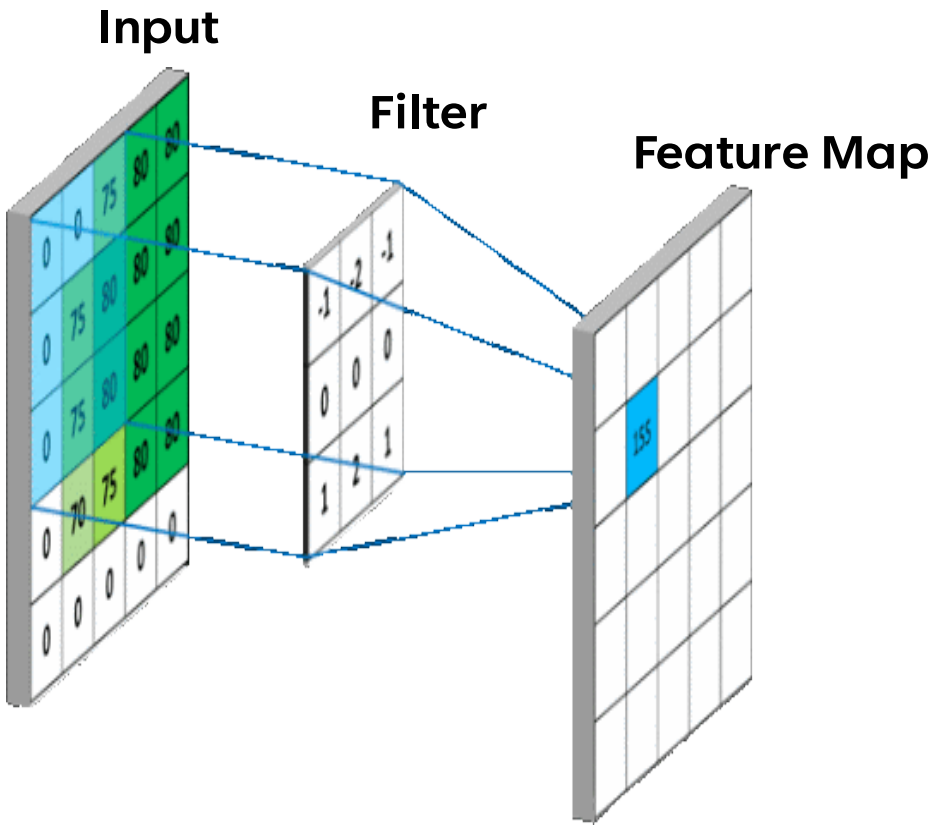
3D



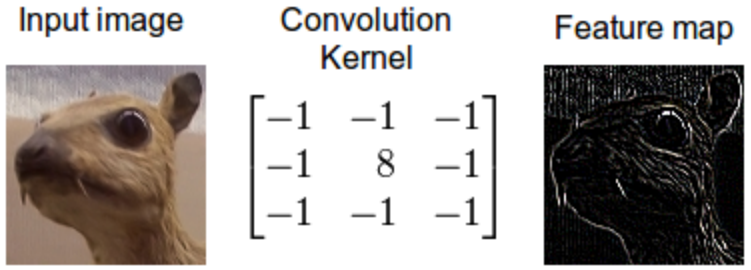
Convolutional Neural Networks (CNNs)



Convolutional Neural Networks (CNNs)



- The term convolution refers to **the mathematical combination of two functions to produce a third function**. It merges two sets of information. In the case of a CNN, the convolution is performed on the input data with the use of a filter or kernel (these terms are used interchangeably) to then produce a feature map.
- It is a very important technique to find patterns in images and image processing



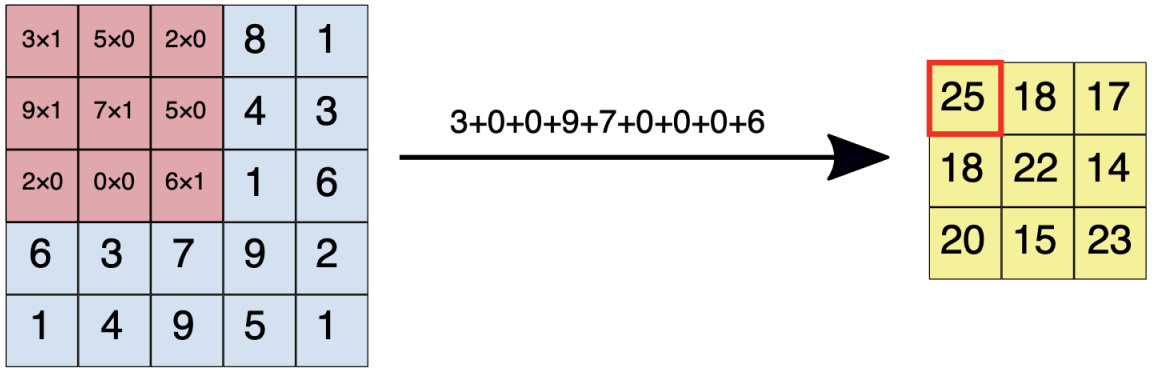
Convolutional Neural Networks (CNNs): Filter

Input

3	5	2	8	1
9	7	5	4	3
2	0	6	1	6
6	3	7	9	2
1	4	9	5	1

1	0	0
1	1	0
0	0	1

Filter/Kernel/Feature detector



Result: Feature map

Note: Feature map **output size** = [**size** of the **input** - **size** of the kernel/filter] + 1

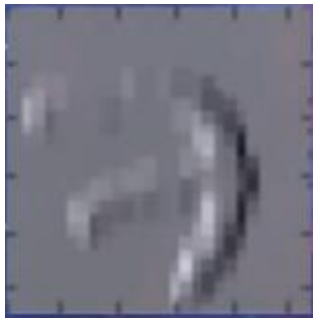
Convolutional Neural Networks (CNNs): Filter

Filter 1

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

Filter 2

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



Filter 3

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

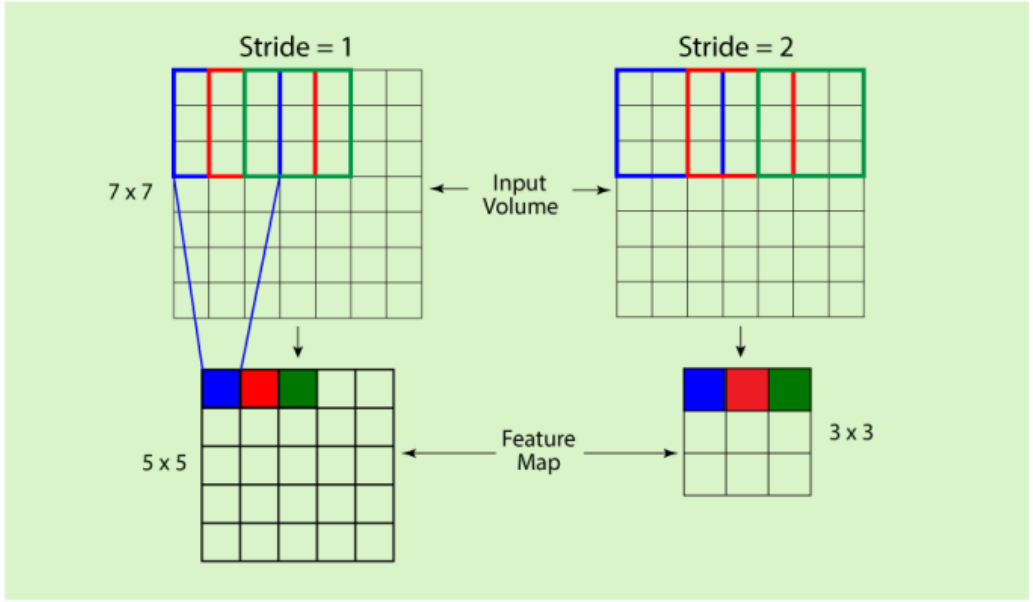
Filter 4

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

Source: [Convolutional Neural Networks (CNNs) explained]

Convolutional Neural Networks (CNNs): Stride

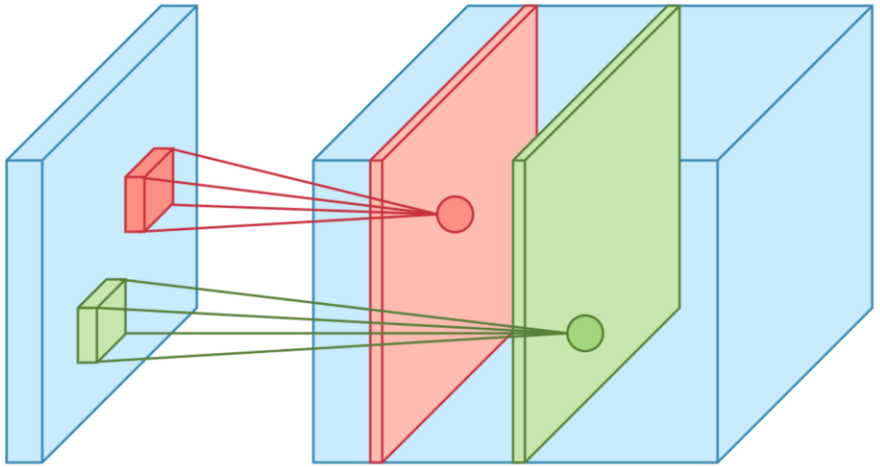
- The filter is moved across the image left to right, top to bottom, with a one-pixel column change on the horizontal movements, then a one-pixel row change on the vertical movements.
- The amount of movement between application of the filter to the input image is referred to as the **STRIDE.**



Note: Feature map **output size** = { [**size** of the **input** - **size** of the kernel/filter] / **STRIPE** } + 1

Convolutional Neural Networks (CNNs): ConvLayers

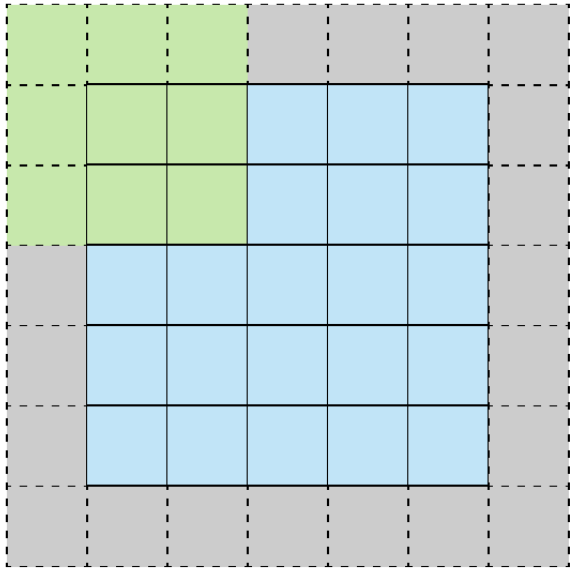
Conv Layer



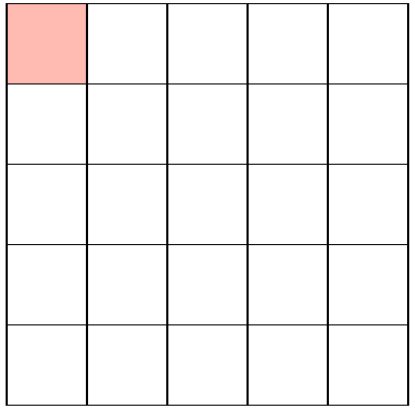
- Convolutional neural networks are a type of neural network in which we are trying to **find** the **filters** and **biases** that **minimize** the **loss function**.
- **Conv Layer** contains different filters/kernels with different lengths and widths.
- The filters are **computed** by the **CNN**, and that is what makes the **CNN** more **powerful**.

Convolutional Neural Networks (CNNs): Padding

Some employed techniques in **CNNs**: **Padding**



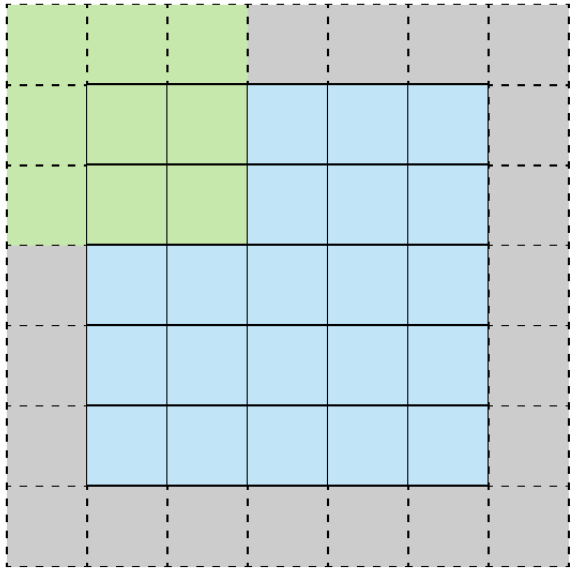
Stride 1 with Padding



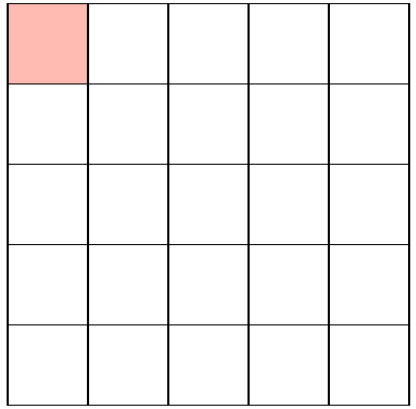
Feature Map

Convolutional Neural Networks (CNNs): Padding

- Padding is the best approach, where the number of pixels needed for the convolutional kernel to process the edge pixels are added onto the outside.
- Fixing the border effect problem with padding



Stride 1 with Padding

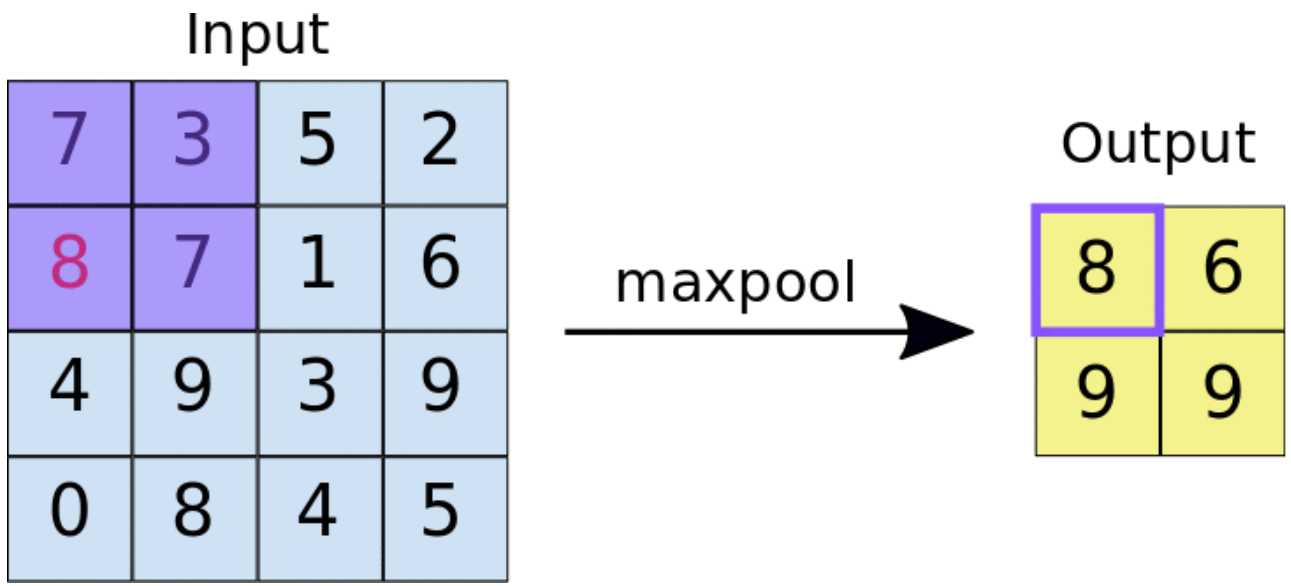


Feature Map

Convolutional Neural Networks (CNNs): Pooling

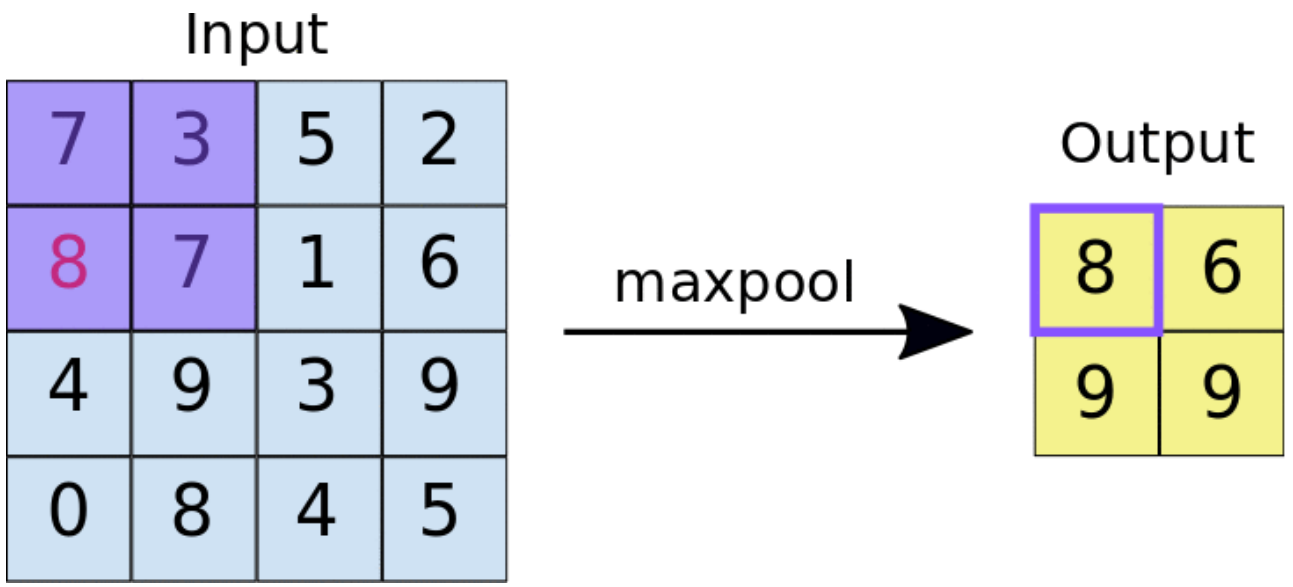
Some used techniques in **ConvNets**: **Pooling**

- **Pooling** is required to down sample the detection of features in feature maps.
- **Pooling layers** provide an approach to down sampling feature maps by summarizing the presence of features in patches of the feature map.
- Two common pooling methods are **average pooling** and **max pooling**

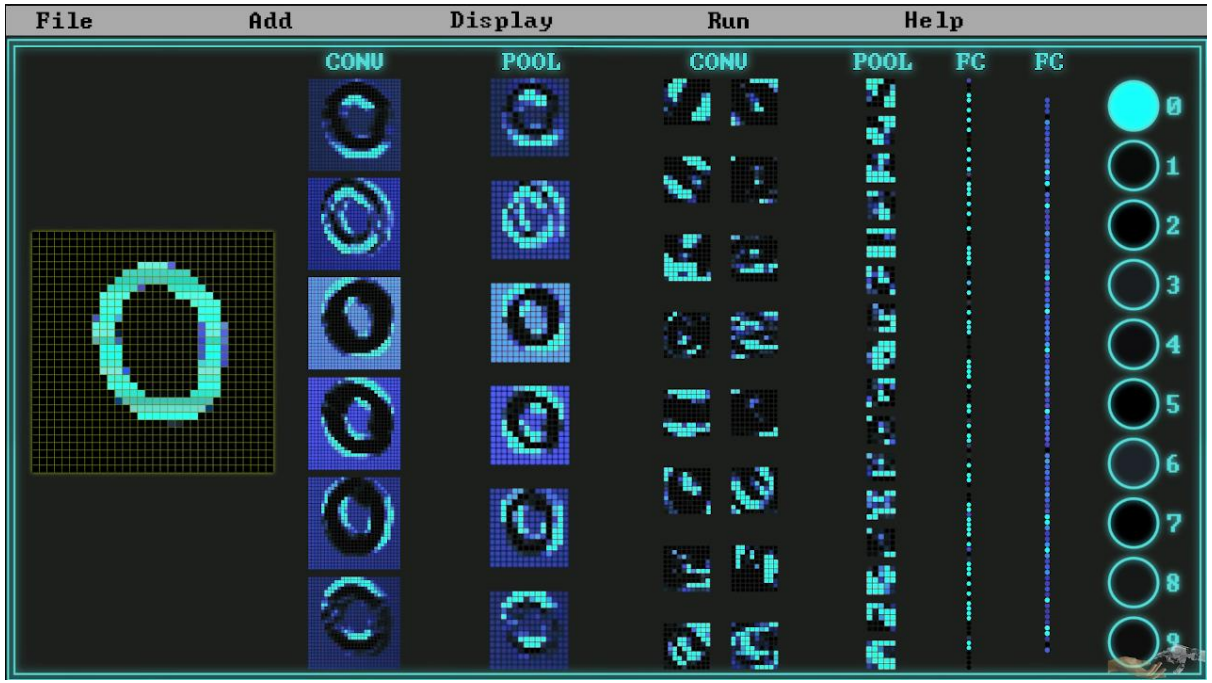


Convolutional Neural Networks (CNNs): MaxPooling

Some used techniques in **ConvNets**: **MaxPooling**

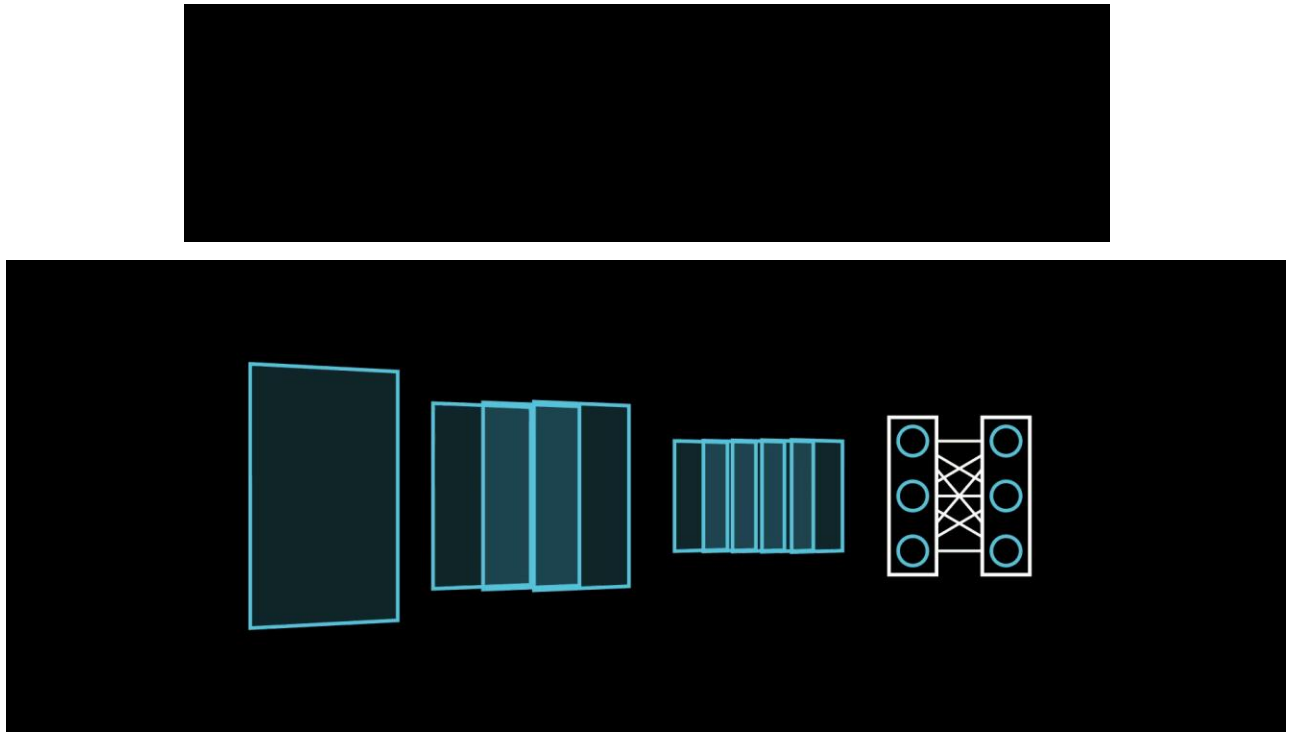


Convolutional Neural Networks (CNNs): CNN Visualized <DEMO>



[Convolutional Neural Networks Explained (CNN Visualized)]

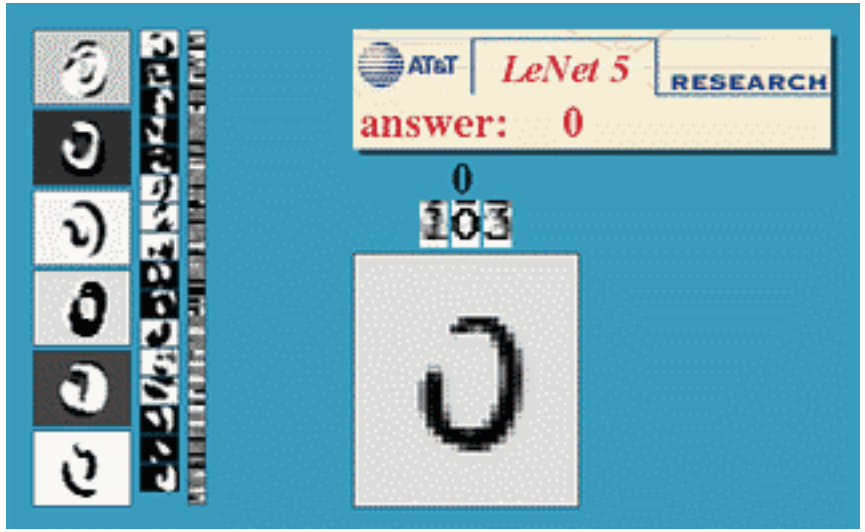
Convolutional Neural Networks (CNNs): CNN Visualization



ManimML Library Link: https://github.com/helblazer811/ManimML

Convolutional Neural Networks (CNNs): LeNet5

LeNet-5 is a convolutional neural network architecture designed for handwritten digit recognition. It was introduced by **Yann LeCun**, Léon Bottou, Yoshua Bengio, and Patrick Haffner in the paper titled "**Gradient-Based Learning Applied to Document Recognition**" published in **1998**.

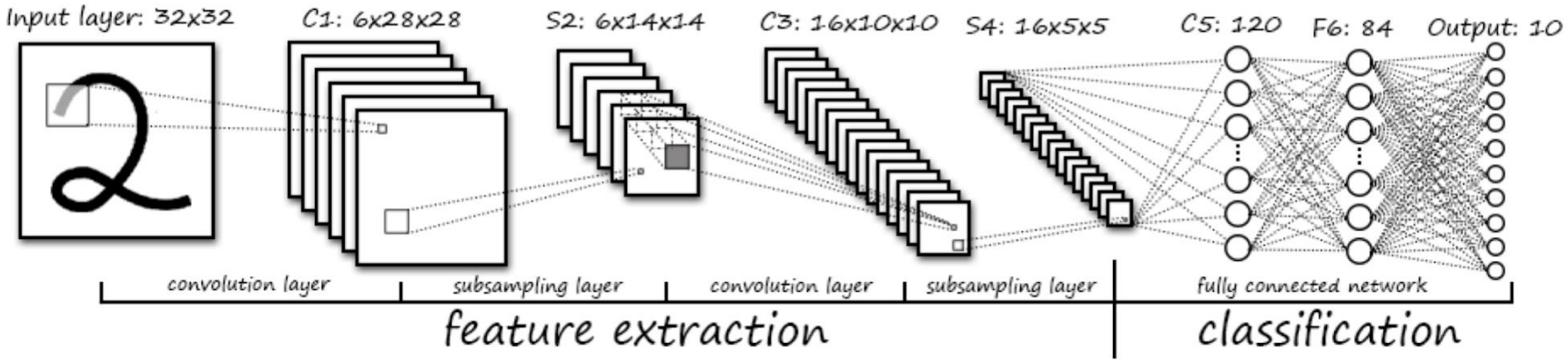


LeNet5 (1998)

Yann LeCun

Convolutional Neural Networks (CNNs): LeNet5

LeNet5 Architecture(1998)

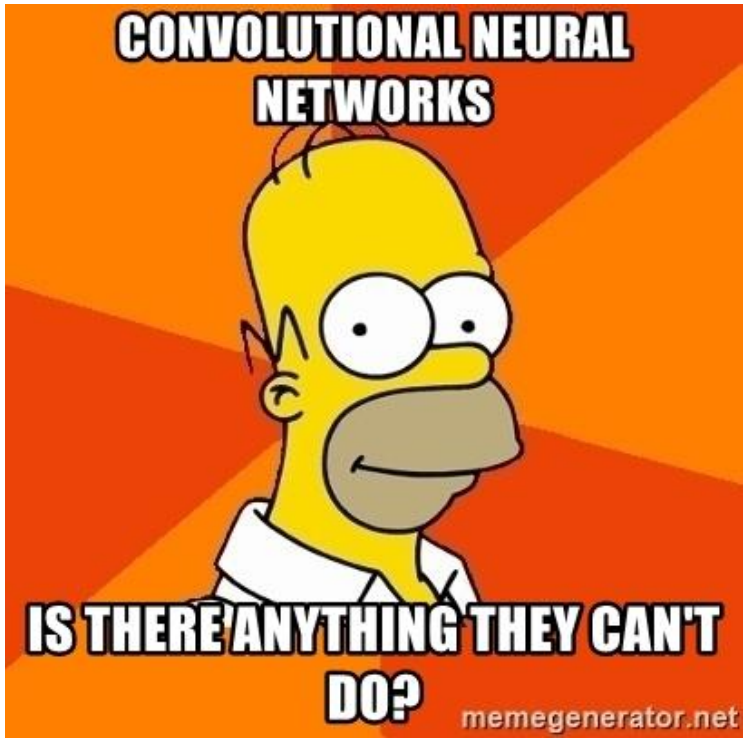


Convolutional Neural Networks (CNNs): LeNet5

LeNet5 Architecture(1998)

1. **Convolutional Layers**: LeNet-5 consists of seven layers, including two convolutional layers. The first convolutional layer applies six filters with a 5x5 receptive field, followed by a 2x2 max-pooling operation.
2. **Activation Functions**: In LeNet-5, the hyperbolic tangent function (tanh) is used as the activation function.
3. **Subsampling Layers**: Following the first convolutional layer, LeNet-5 has a subsampling layer (max-pooling) to reduce the spatial dimensions of the feature maps.
4. **Second Convolutional Layer**: The second convolutional layer applies sixteen 5x5 filters to the subsampled feature maps from the first convolutional layer. This is followed by another 2x2 max-pooling operation.
5. **Fully Connected Layers**: After the convolutional layers, there are three fully connected layers. The first fully connected layer has 120 units, the second has 84 units, and the final output layer has 10 units (corresponding to the 10 possible digits).
6. **Flatten Operation**: Between the convolutional and fully connected layers, there is a flatten operation to convert the 3D volume output from the convolutional layers into a 1D vector.
7. **Softmax Activation**: The output layer uses a softmax activation function, which turns the raw scores (logits) into class probabilities.

Convolutional Neural Networks (CNNs) Architectures: DEMO

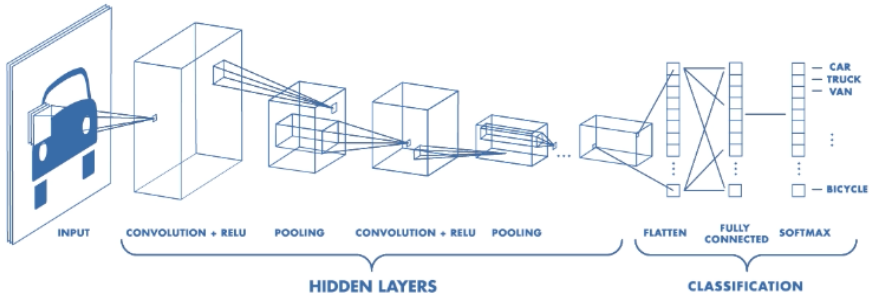


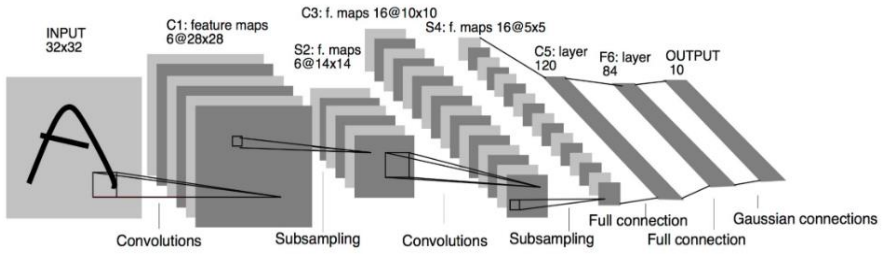
Demo: Session 3 – Computer Vision and CNN

Convolutional Neural Networks (CNNs) Architectures: Classic CNNs

Classic ConvNet Architecture:

- Input
- Conv blocks
  - Convolution + activation (ReLU)
  - Convolution + activation (ReLU)
  - ....
  - Maxpooling 2*2
- Output
  - Fully connected layers
  - Softmax



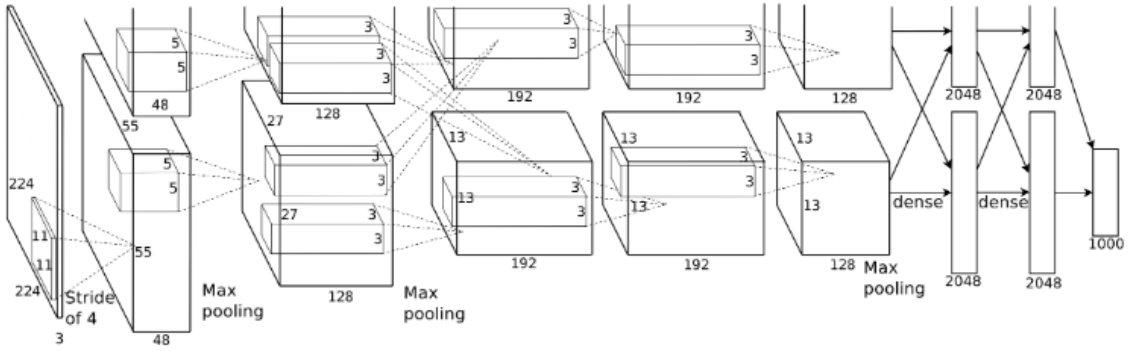


Convolutional Neural Networks (CNNs) Architectures: AlexNet

AlexNet is a deep convolutional neural network architecture designed for image classification tasks. It was developed by **Alex Krizhevsky**, **Ilya Sutskever**, and **Geoffrey Hinton** and **won** the **ImageNet** Large Scale Visual Recognition Challenge (ILSVRC) in **2012**.

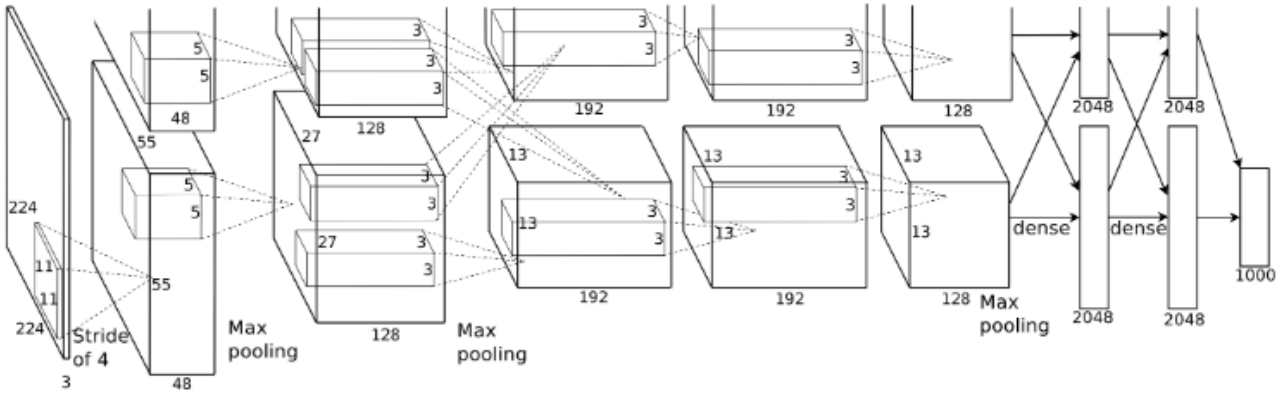
First conv layer: kernel 11x11x3x96 stride 4

- Kernel shape: (11,11,3,96)
- Output shape: (55,55,96)
- Number of parameters: 34,944
- Equivalent MLP parameters: 43.7 x 1e9



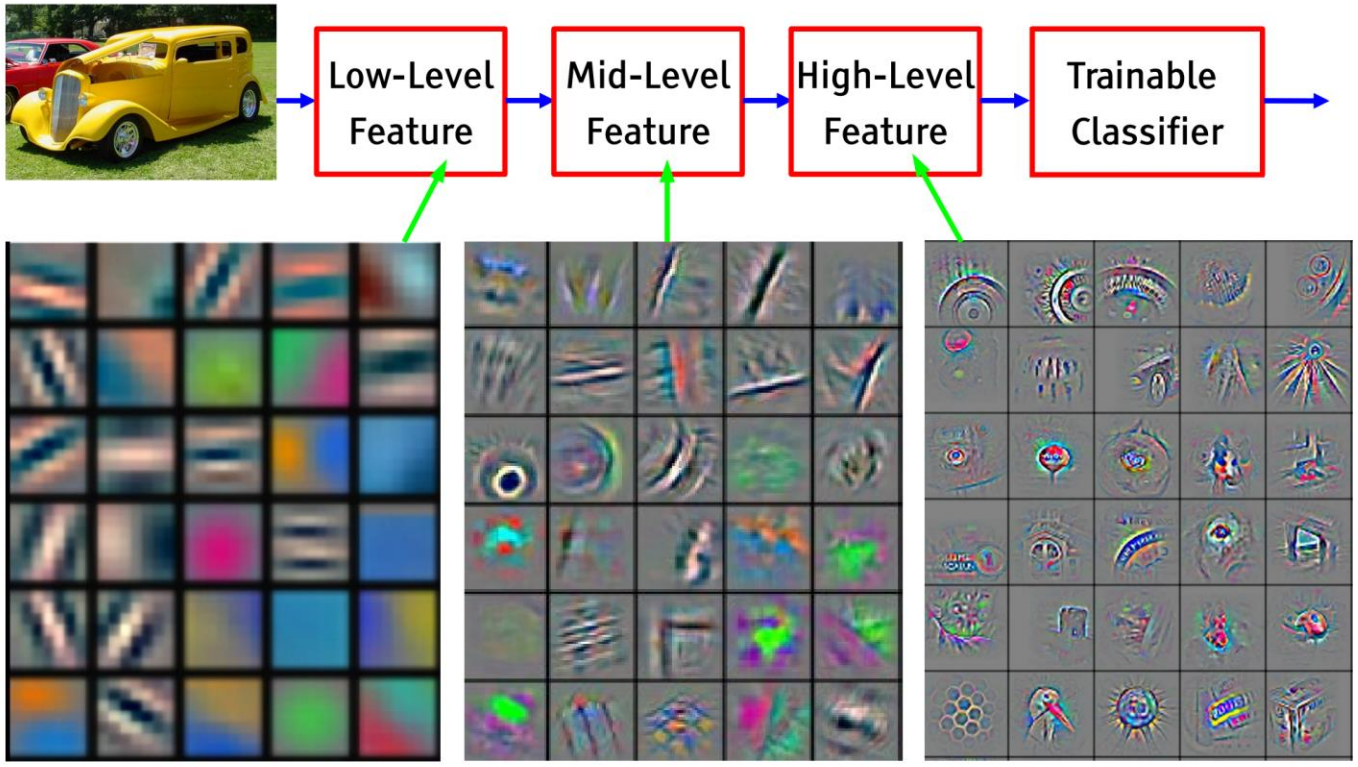
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012).

Convolutional Neural Networks (CNNs) Architectures: AlexNet




```
INPUT:     [227x227x3]
CONV1:     [55x55x96]   96 11x11 filters at stride 4, pad 0
MAX POOL1: [27x27x96]      3x3   filters at stride 2
CONV2:     [27x27x256] 256 5x5   filters at stride 1, pad 2
MAX POOL2: [13x13x256]     3x3   filters at stride 2
CONV3:     [13x13x384] 384 3x3   filters at stride 1, pad 1
CONV4:     [13x13x384] 384 3x3   filters at stride 1, pad 1
CONV5:     [13x13x256] 256 3x3   filters at stride 1, pad 1
MAX POOL3: [6x6x256]       3x3   filters at stride 2
FC6:       [4096]      4096 neurons
FC7:       [4096]      4096 neurons
FC8:       [1000]      1000 neurons (softmax logits)
```

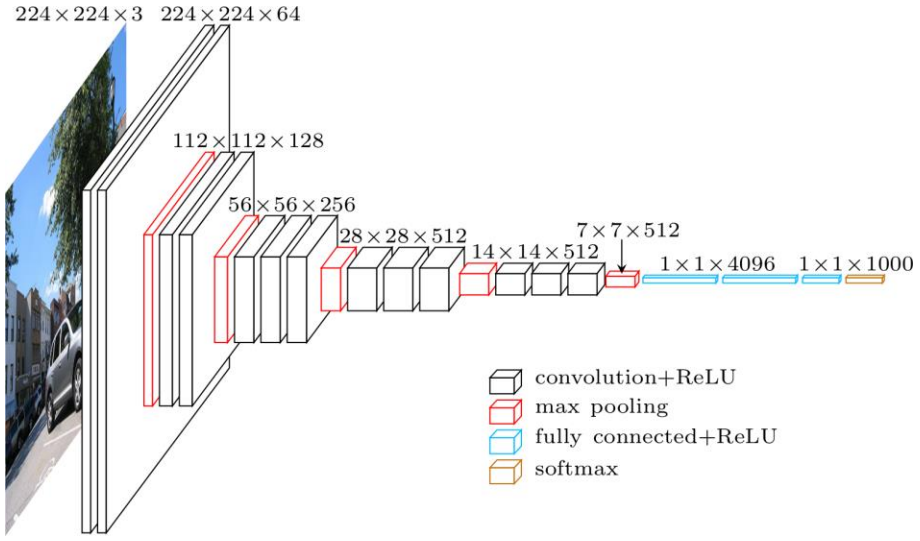
Convolutional Neural Networks (CNNs) Architectures: Hierarchical Representation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Convolutional Neural Networks (CNNs) Architectures: VGG-16

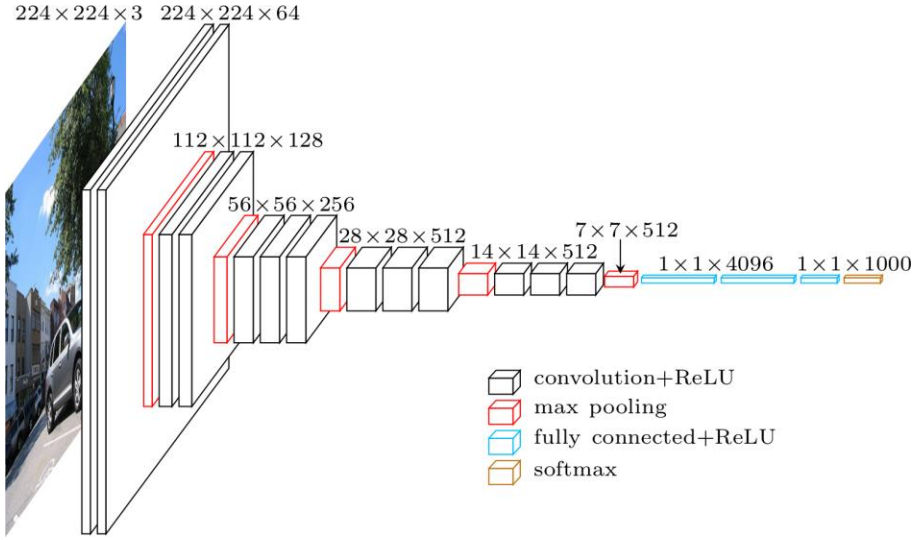
VGG-16 is a deep convolutional neural network architecture that was introduced by the **Visual Geometry Group (VGG)** at the University of Oxford. It was presented in the paper titled "**Very Deep Convolutional Networks for Large-Scale Image Recognition**" by **Karen Simonyan** and **Andrew Zisserman**. **VGG-16** was a runner-up in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in **2014**.



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

Convolutional Neural Networks (CNNs) Architectures: VGG-16

- **Architecture:** VGG-16 is characterized by its deep architecture. It has 16 weight layers, including 13 convolutional layers and 3 fully connected layers.
- **Convolutional Layers:** VGG-16 uses a series of small 3x3 convolutional filters with a stride of 1 and 'same' padding. This allows it to learn complex features by stacking multiple layers of small receptive fields.
- **Max-Pooling:** After every two convolutional layers, max-pooling with a 2x2 window and stride 2 is applied. This helps reduce the spatial dimensions of the feature maps while retaining important information.
- **ReLU Activation:** Like AlexNet, VGG-16 uses the Rectified Linear Unit (ReLU) activation function after each convolutional layer.
- **Fully Connected Layers:** The final layers of the network are fully connected. The architecture ends with two fully connected layers, each with 4,096 units, followed by an output layer with 1,000 units (corresponding to the 1,000 ImageNet classes).



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

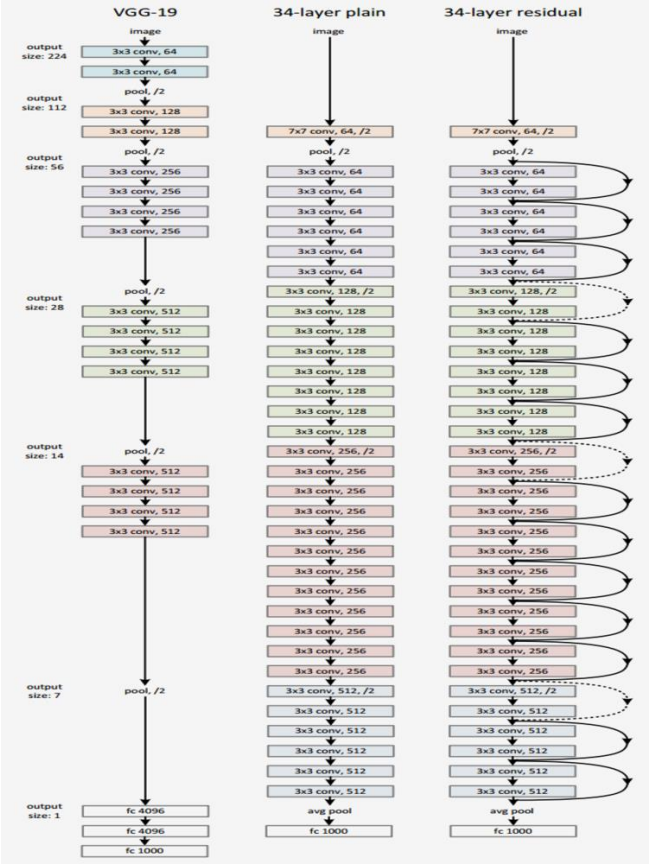
Convolutional Neural Networks (CNNs) Architectures: VGG-16

VGG-16 in Keras

```
model.add(Convolution2D(64, 3, 3, activation='relu',input_shape=(3,224,224)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1000, activation='softmax'))
```

Convolutional Neural Networks (CNNs) Architectures: ResNet

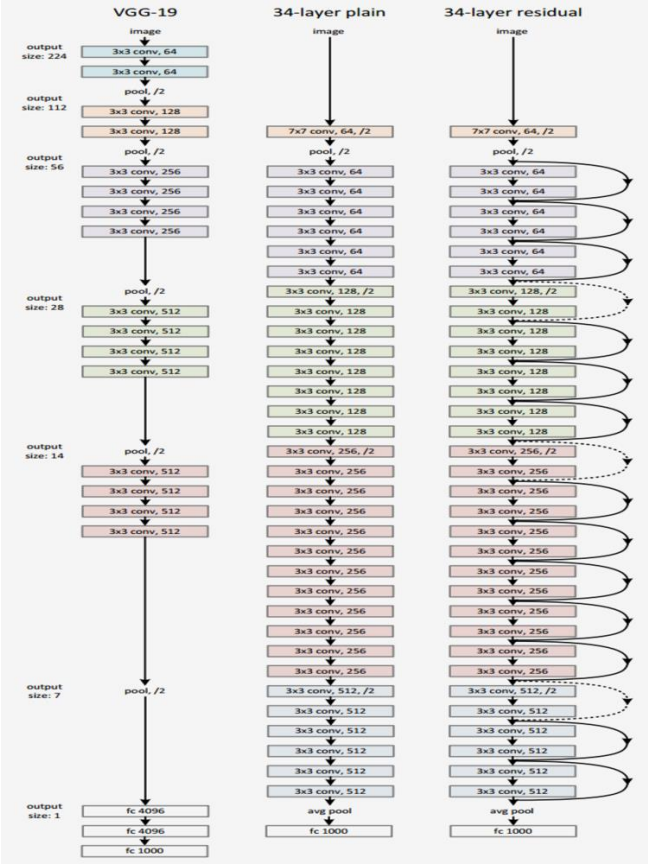
- ResNet, short for Residual Network, is a deep neural network architecture that was introduced in the paper titled "Deep Residual Learning for Image Recognition" by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, published in 2015. ResNet is known for its ability to train very deep networks effectively, overcoming the problem of vanishing gradients.



He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

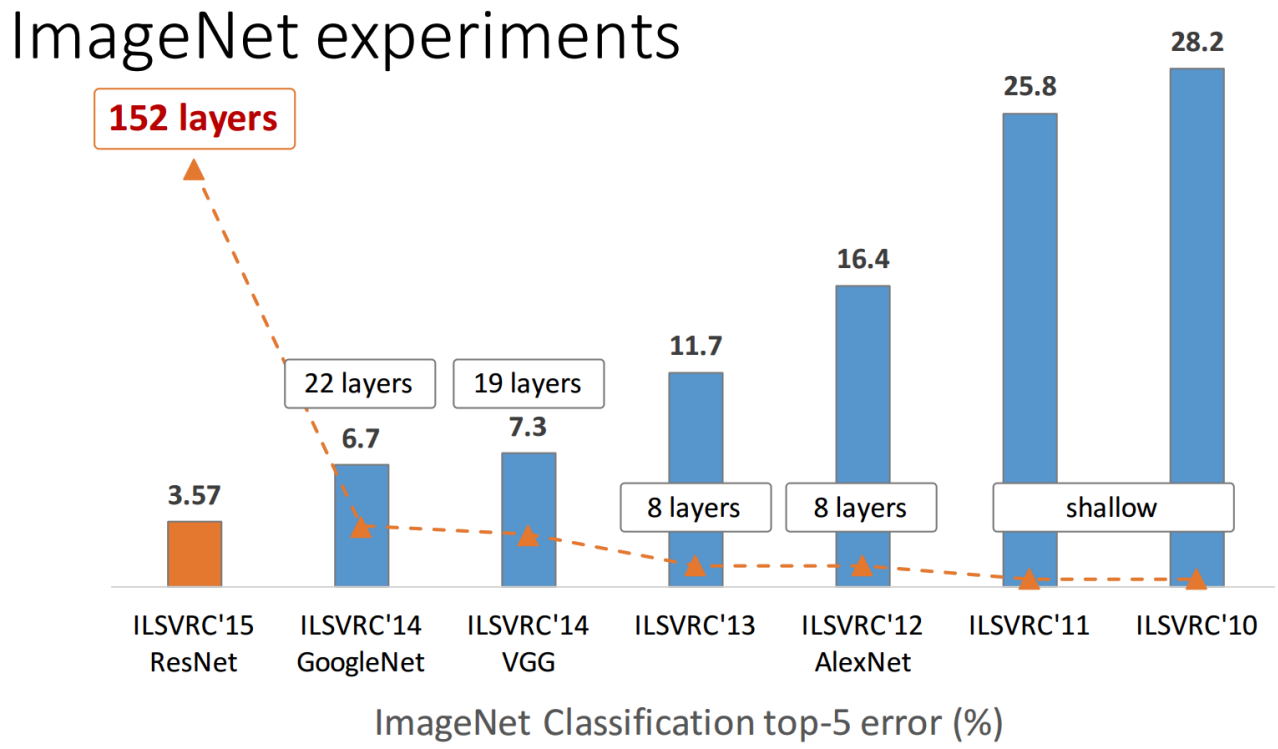
Convolutional Neural Networks (CNNs) Architectures: ResNet

- **ResNet50** Compared to **VGG**:
  - Superior accuracy in all vision tasks
    5.25% top-5 error vs 7.1%
  - Less parameters
    25M vs 138M
  - Computational complexity
    3.8B Flops vs 15.3B Flops
  - Fully Convolutional until the last layer



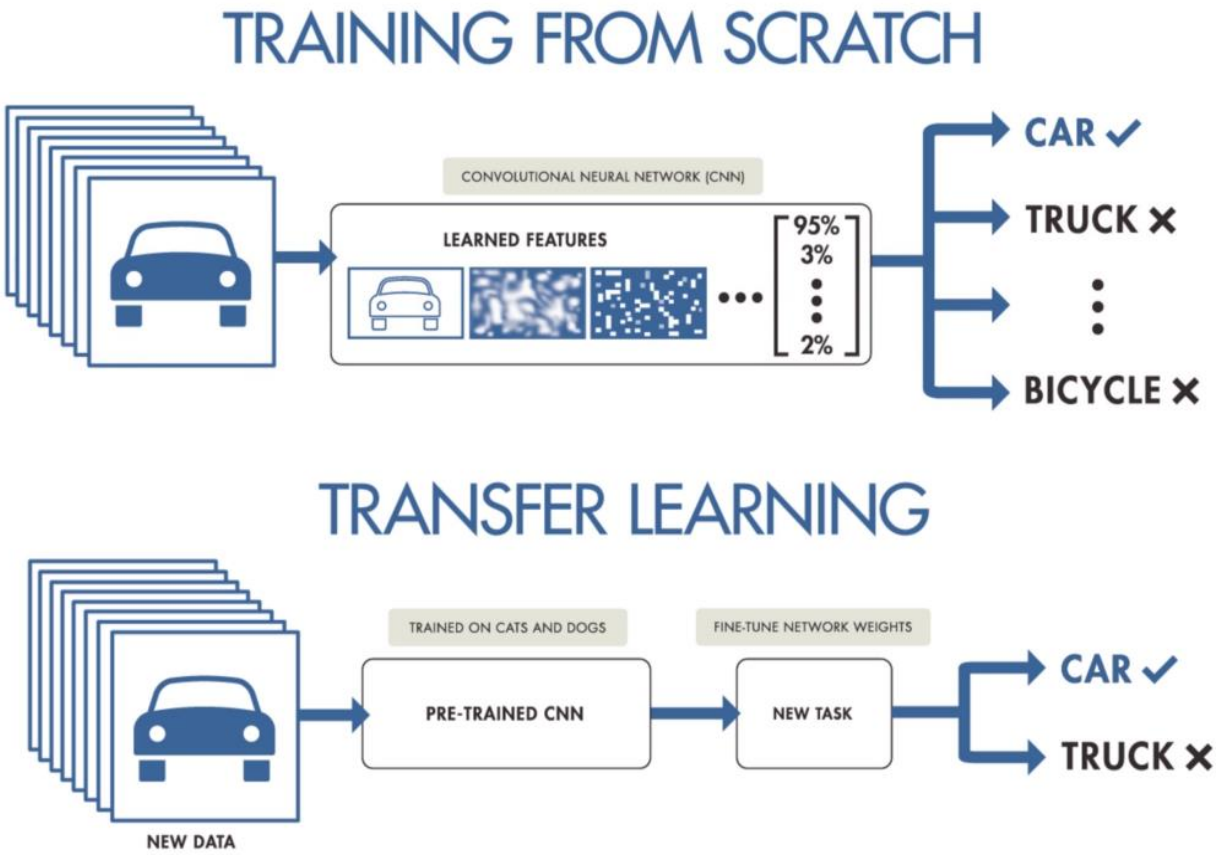
He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Convolutional Neural Networks (CNNs) Architectures: Deeper is Better



from Kaiming He slides "Deep residual learning for image recognition." ICML. 2016.

Transfer Learning and Pre-Trained Models

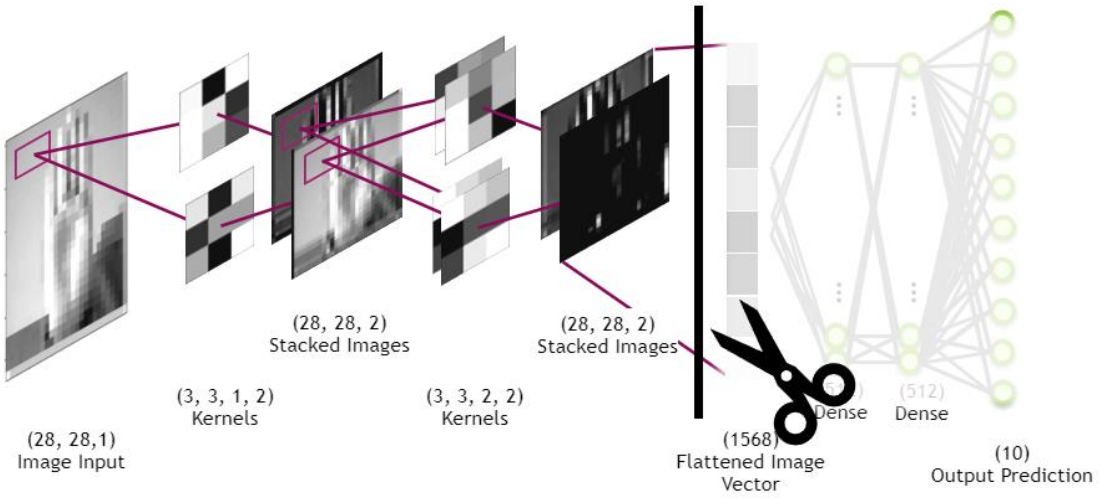


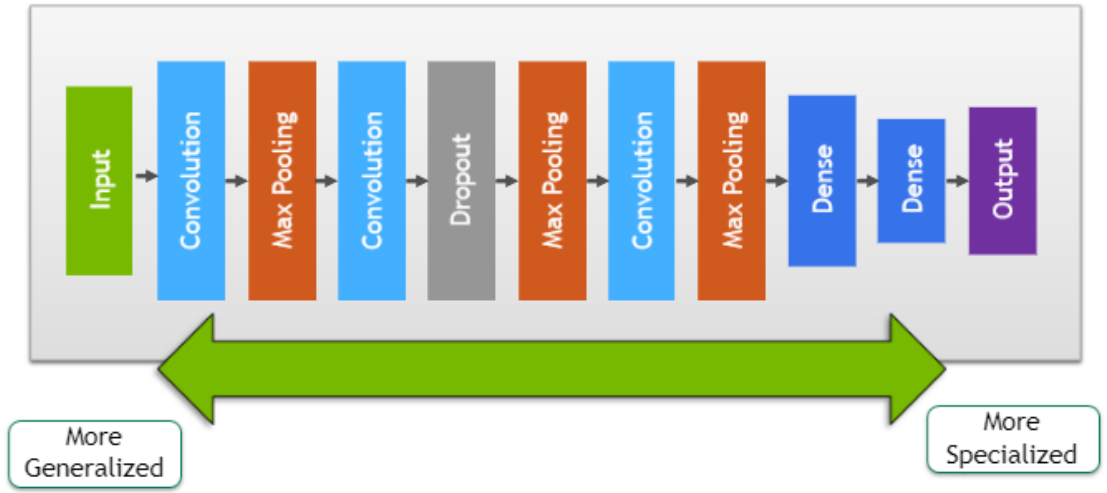
Transfer Learning and Pre-Trained Models

Problem: Training a model on **ImageNet** from scratch takes **days** or **weeks**.
Hint: Many models trained on ImageNet and their weights are publicly available!
Solution: Transfer learning

- Use pre-trained weights, remove last layers to compute representations of images
- Train a classification model from these features on a new classification task
- The network is used as a generic feature extractor
- Better than handcrafted feature extraction on natural images

Transfer Learning and Pre-Trained Models

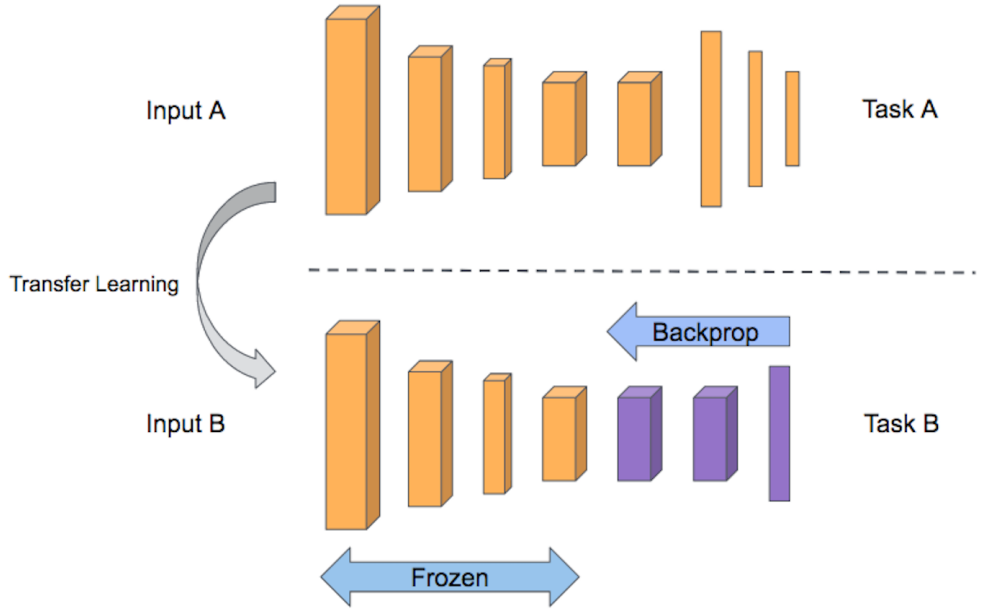




Transfer Learning and Pre-Trained Models

Transfer Learning (TL) from task A to task B can be very useful if:

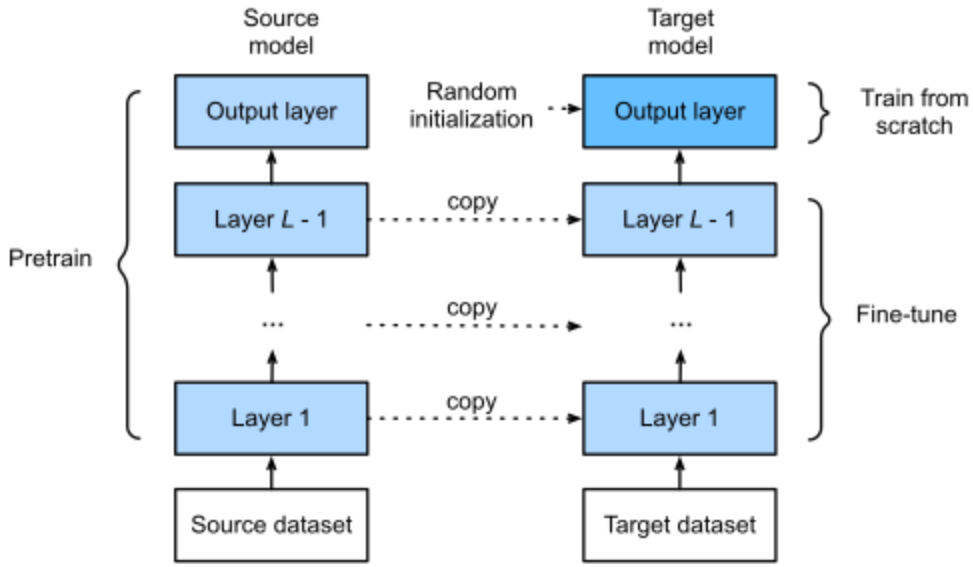
- Task A and B have the same input X.
- There is a lot more data for task A than task B.
- Low level features from task A could be very helpful for learning task B



Transfer Learning and Pre-Trained Models: Fine-Tuning

Fine-tuning:

- Retraining the (some) parameters of the network (given enough data)
- Truncate the last layer(s) of the pre-trained network
- Freeze the remaining layers weights
- Add a (linear) classifier on top and train it for a few epochs
- Then fine-tune the whole network or the few deepest layers
- Use a smaller learning rate when fine tuning



Source: http://d2l.ai/chapter_computer-vision/fine-tuning.html

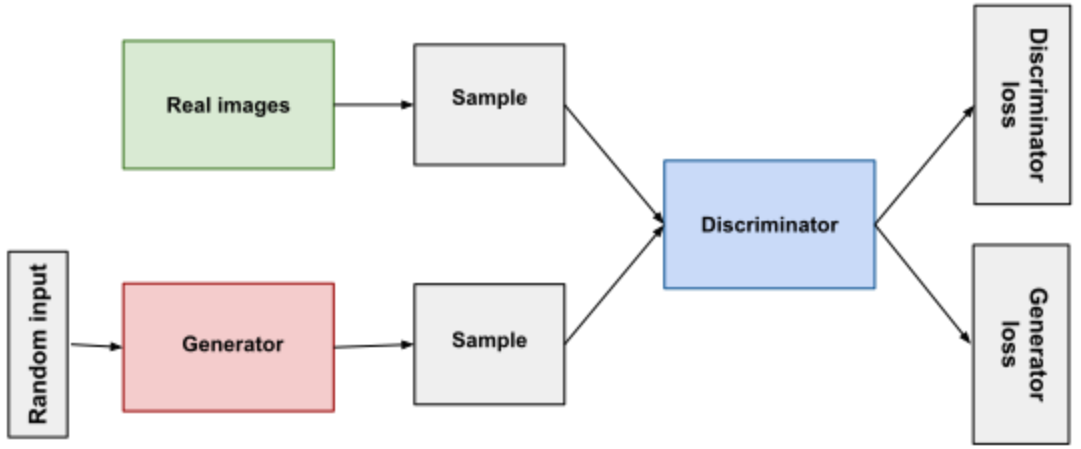
Transfer Learning and Pre-Trained Models: DEMO

Project: Session 3 – Computer Vision and CNN

Generative Adversarial Network (GAN)

Generative Adversarial Networks (GANs) are a class of machine learning models (Generative models) introduced by **Ian Goodfellow** and his colleagues in **2014**. **GANs** are a type of generative model, which means they are designed to generate new data that is similar to some existing dataset.
A GAN consists of two neural networks:

- **Generator**: This network generates data. It takes random noise as input and transforms it into data samples that are intended to be similar to real data. For example, in a GAN that generates images, the generator takes random noise as input and tries to produce images that look like real photographs.
- **Discriminator**: This network tries to distinguish between real data (from the actual dataset) and fake data generated by the generator. It is trained to assign high probabilities to real data and low probabilities to fake data.

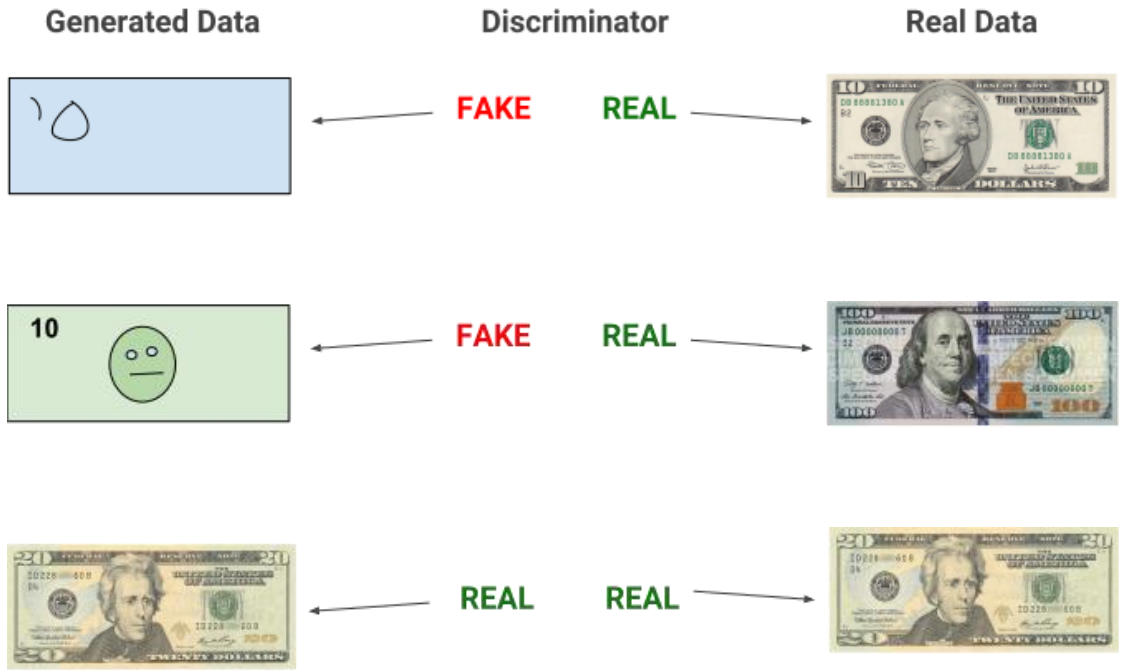


Source: https://developers.google.com/machine-learning/gan/gan_structure

Generative Adversarial Network (GAN): Overview of GAN Structure

A generative adversarial network (**GAN**) has **two parts**:

- The **generator** learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- The **discriminator** learns to distinguish the **generator's** fake data from real data. The **discriminator** penalizes the generator for producing implausible results.



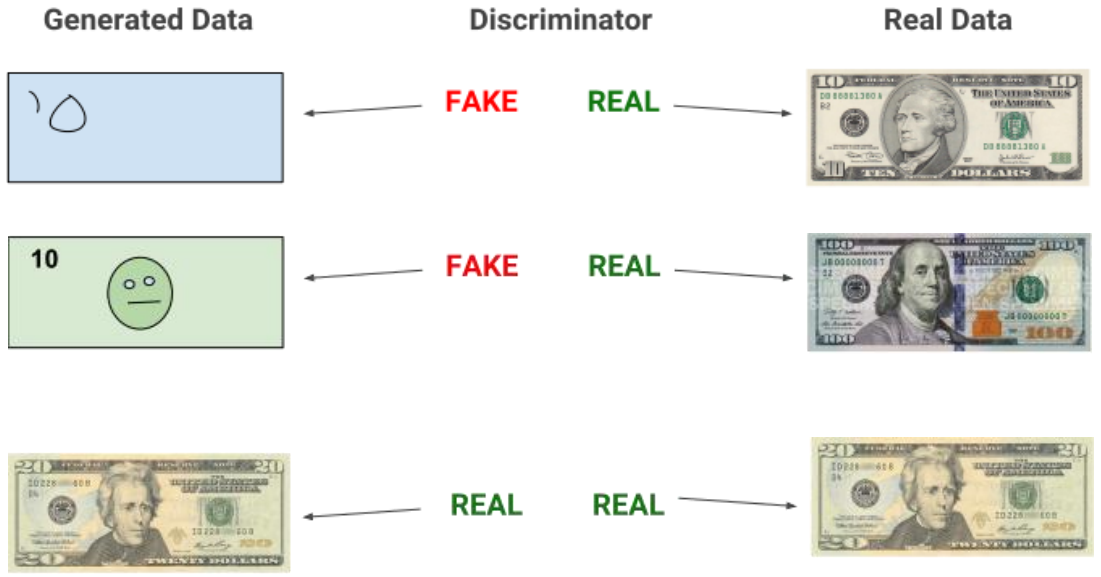
Source: https://developers.google.com/machine-learning/gan/gan_structure

Generative Adversarial Network (GAN): Overview of GAN Structure

The training process of a GAN involves a kind of adversarial game between these two networks:

1. Initially, the generator produces random samples, which are usually of low quality and unlikely to resemble the real data.
2. The discriminator is trained on a mixture of real and fake data, learning to distinguish between them.
3. The generator then tries to produce better fake samples to fool the discriminator.
4. The process continues iteratively, with both networks improving over time. The generator learns to generate more convincing data, while the discriminator becomes better at distinguishing real from fake.

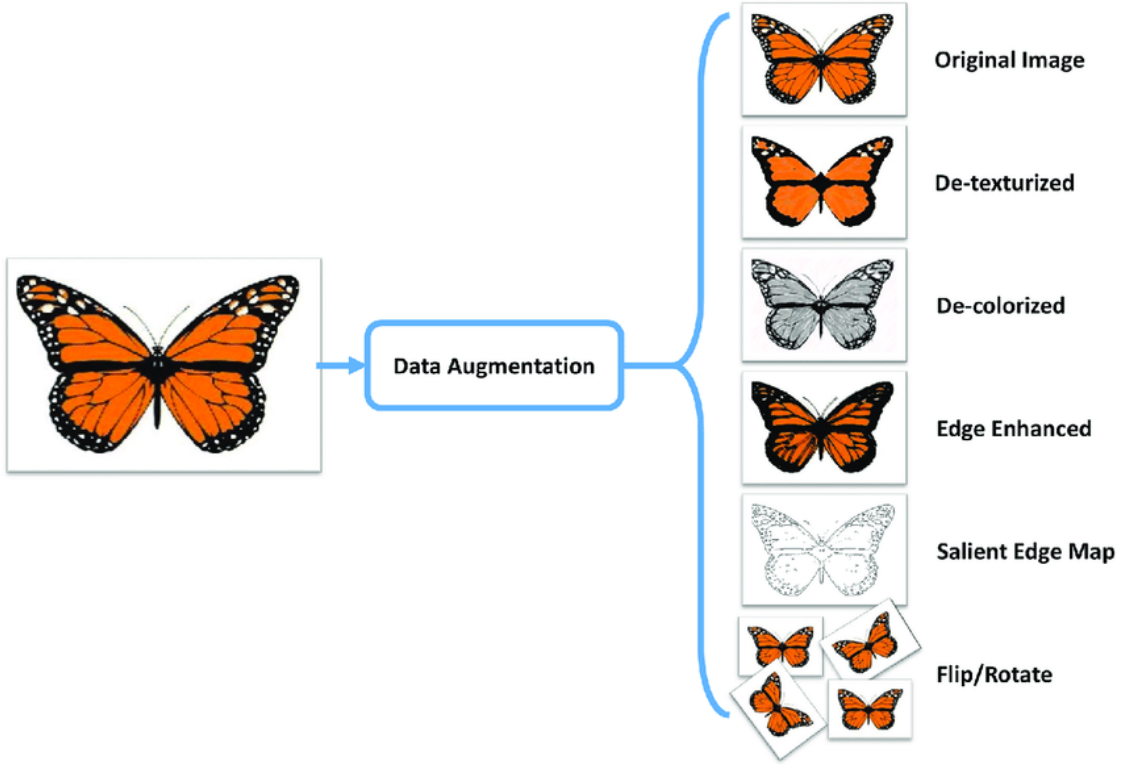
Ideally, this process leads to a point where the generator produces data that is indistinguishable from real data to the discriminator. At this point, the GAN has reached a stable state, and the generator is said to have learned the underlying distribution of the real data.



Source: https://developers.google.com/machine-learning/gan/gan_structure

Data Augmentation for Images

Definition: Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points.



Source: https://docs.smartly.ai/

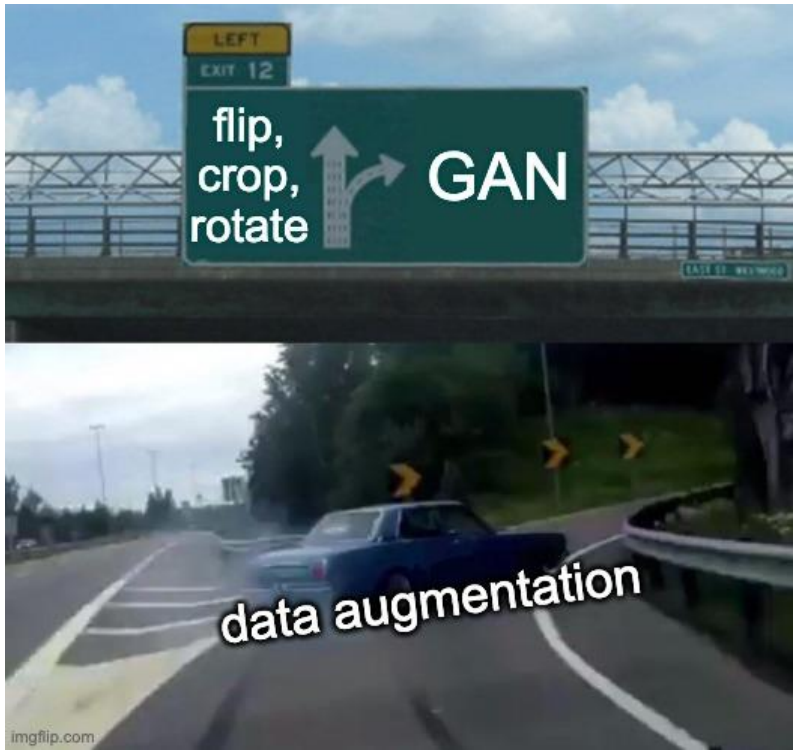
Data Augmentation: Augmented vs. Synthetic Data

Augmented data is driven from original data with some minor changes. In the case of image augmentation, we make geometric and color space transformations (flipping, resizing, cropping, brightness, contrast) to increase the size and diversity of the training set.

Synthetic data is generated artificially without using the original dataset. It often uses DNNs (Deep Neural Networks) and GANs (Generative Adversarial Networks) to generate synthetic data.

Note: the augmentation techniques are not limited to images. You can augment audio, video, text, and other types of data too.

Data Augmentation: Augmented vs. Synthetic Data



Source: https://cs236g.stanford.edu/memes

Data Augmentation: When to Use Data Augmentation?

- To prevent models from overfitting.
- The initial training set is too small.
- To improve the model accuracy.
- To Reduce the operational cost of labeling and cleaning the raw dataset.

Data Augmentation Techniques for Images

- **Geometric transformations:** randomly flip, crop, rotate, stretch, and zoom images. You need to be careful about applying multiple transformations on the same images, as this can reduce model performance.
- **Color space transformations:** randomly change RGB color channels, contrast, and brightness.
- **Kernel filters:** randomly change the sharpness or blurring of the image.
- **Random erasing:** delete some part of the initial image.
- **Mixing images:** blending and mixing multiple images.

Advanced Data Augmentation Techniques for Images

- **Generative adversarial networks (GANs):** used to generate new data points or images. It does not require existing data to generate synthetic data.
- **Neural Style Transfer:** a series of convolutional layers trained to deconstruct images and separate context and style.

Examples of Photorealistic GAN-Generated Faces.Taken from Progressive Growing of GANs for Improved Quality, Stability, and Variation, 2017.

Example of Neural Style Transfer

Data Augmentation: Applications in Healthcare

- **Acquiring** and **labeling** medical imaging datasets is **time-consuming** and **expensive**. You also need a subject matter expert to validate the dataset before performing data analysis. Using geometric and other transformations can help you train robust and accurate machine-learning models.
- For example, in the case of Pneumonia Classification, you can use random cropping, zooming, stretching, and color space transformation to improve the model performance. However, you need to be careful about certain augmentations as they can result in opposite results. For example, random rotation and reflection along the x-axis are not recommended for the X-ray imaging dataset.

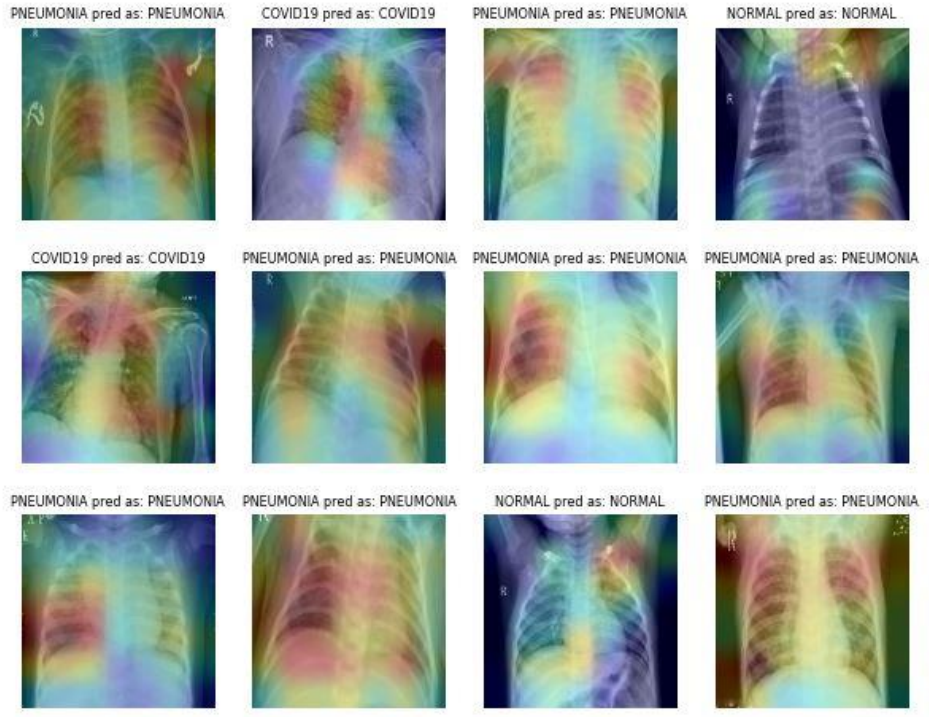


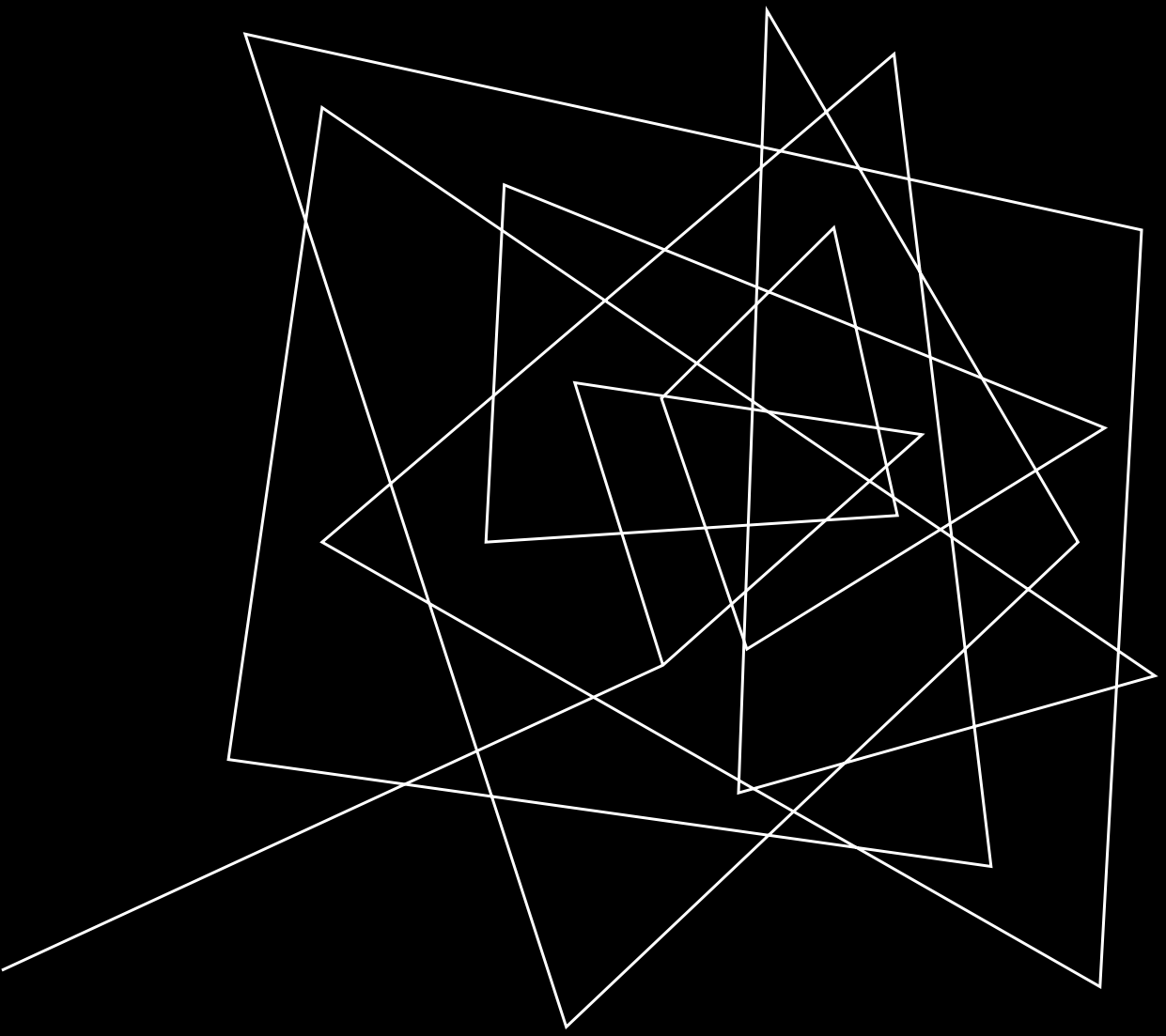
Image from ibrahimsobh.github.io | kaggle-COVID19-Classification

Data Augmentation: Keras Demo

```
from keras.preprocessing.image import ImageDataGenerator

image_gen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    channel_shift_range=9,
    fill_mode='nearest'
)

train_flow = image_gen.flow_from_directory(train_folder)
model.fit_generator(train_flow, train_flow.n)
```

Deep Learning for Computer Vision (CNNs): TO-DO Project

Project: Chest X-ray Image Classification for Disease Diagnosis | Chest X-ray (Covid-19 & Pneumonia)

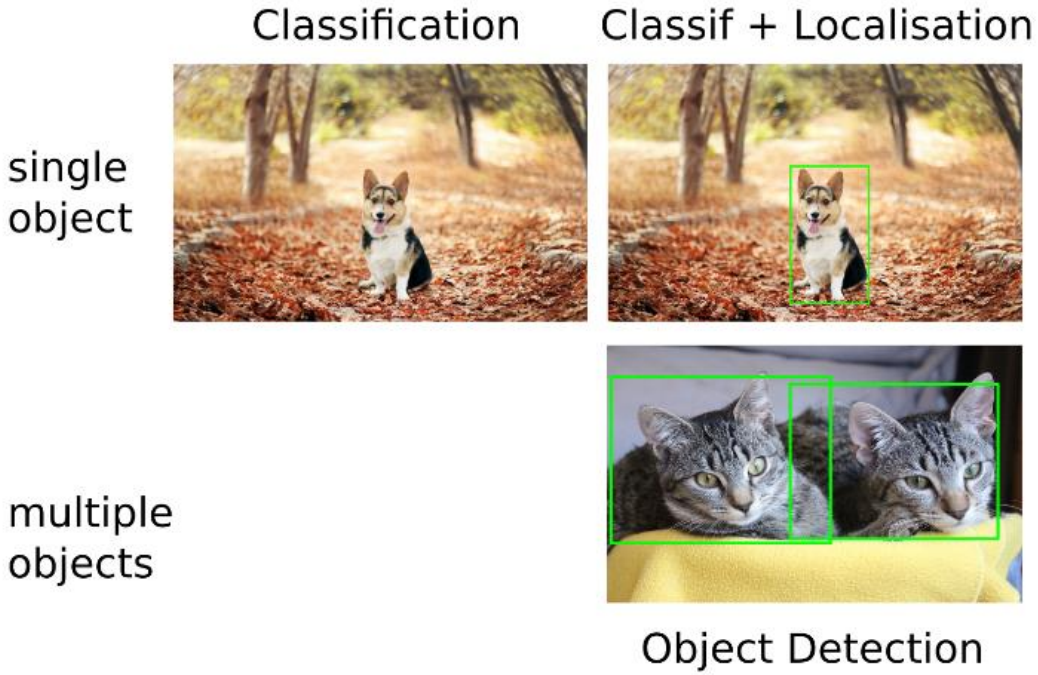


YOLO FOR OBJECT DETECTION

YOLO and its variants for object detection

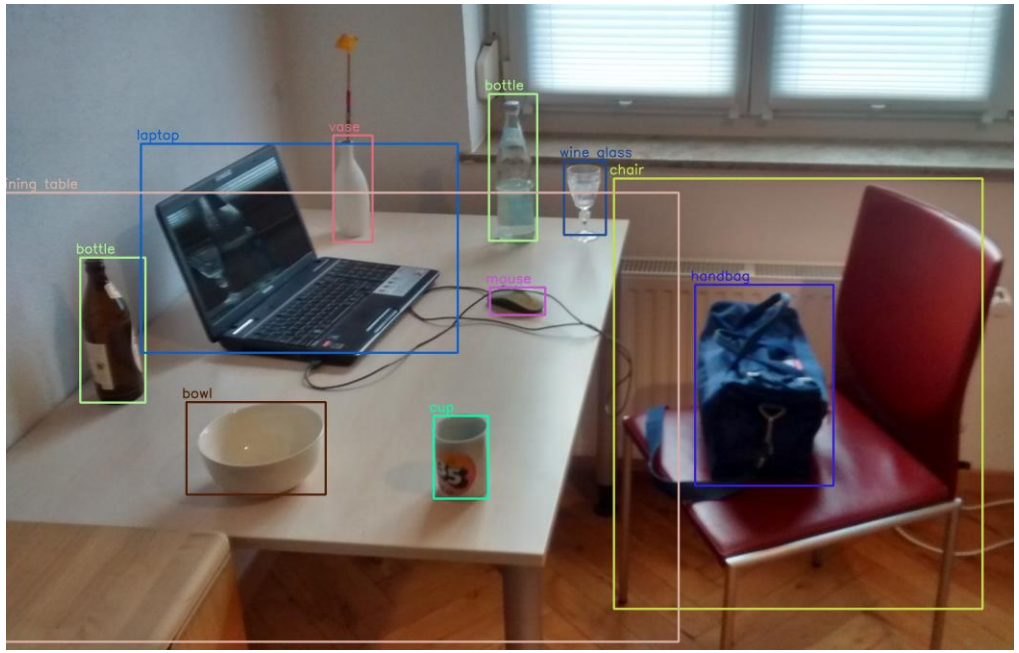
Introduction: Beyond Image Classification

- **Previous lecture:** Image classification using CNNs
- **Limitations**
  - Mostly on centered images
  - Only a single object per image
  - Not enough for many real-world vision tasks



Introduction: What is Object Detection?

- Object detection is a computer vision technique for locating and identifying instances of objects in **images** or **videos**



Introduction: Object Detection

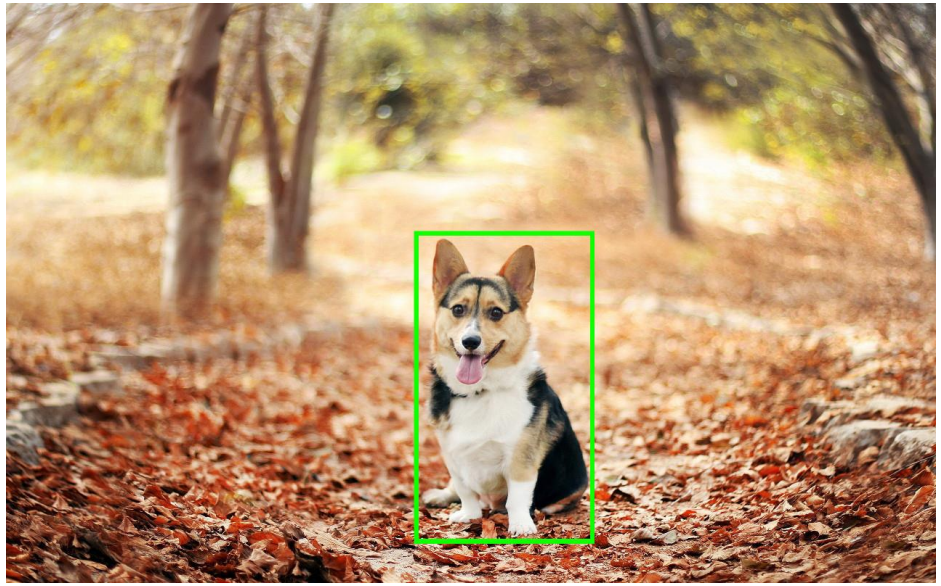
- **Object detection** is a technique used in computer vision for the identification and localization of objects within an image or a video.
- **Image Localization** is the process of identifying the correct location of one or multiple objects using bounding boxes, which correspond to rectangular shapes around the objects.

This process is sometimes confused with image classification or image recognition, which aims to predict the class of an image or an object within an image into one of the categories or classes.

Source: YOLO Object Detection Explained

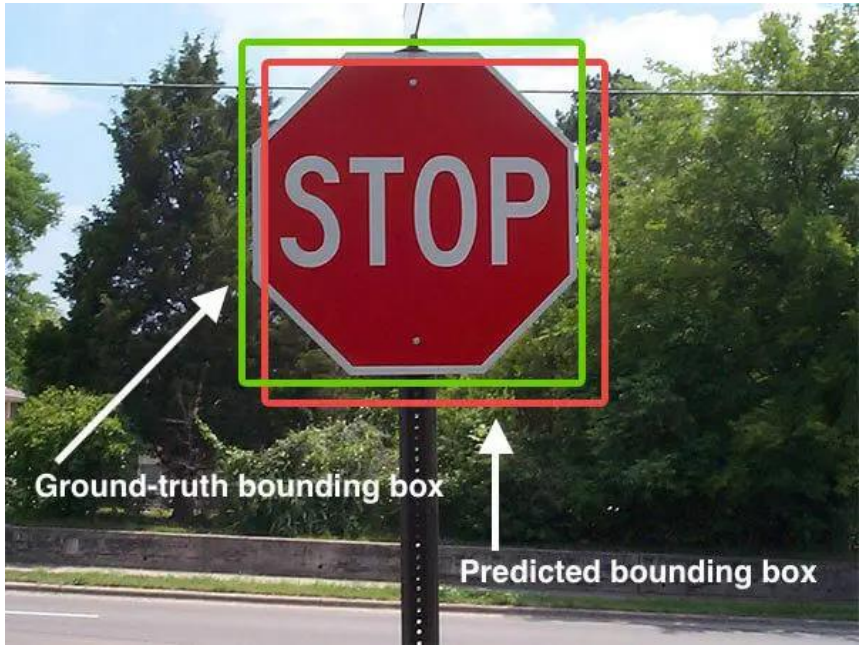
Introduction: Localization

- Single object per image
- Predict coordinates of a bounding box (x, y, w, h)
- Evaluate via Intersection over Union (IoU)

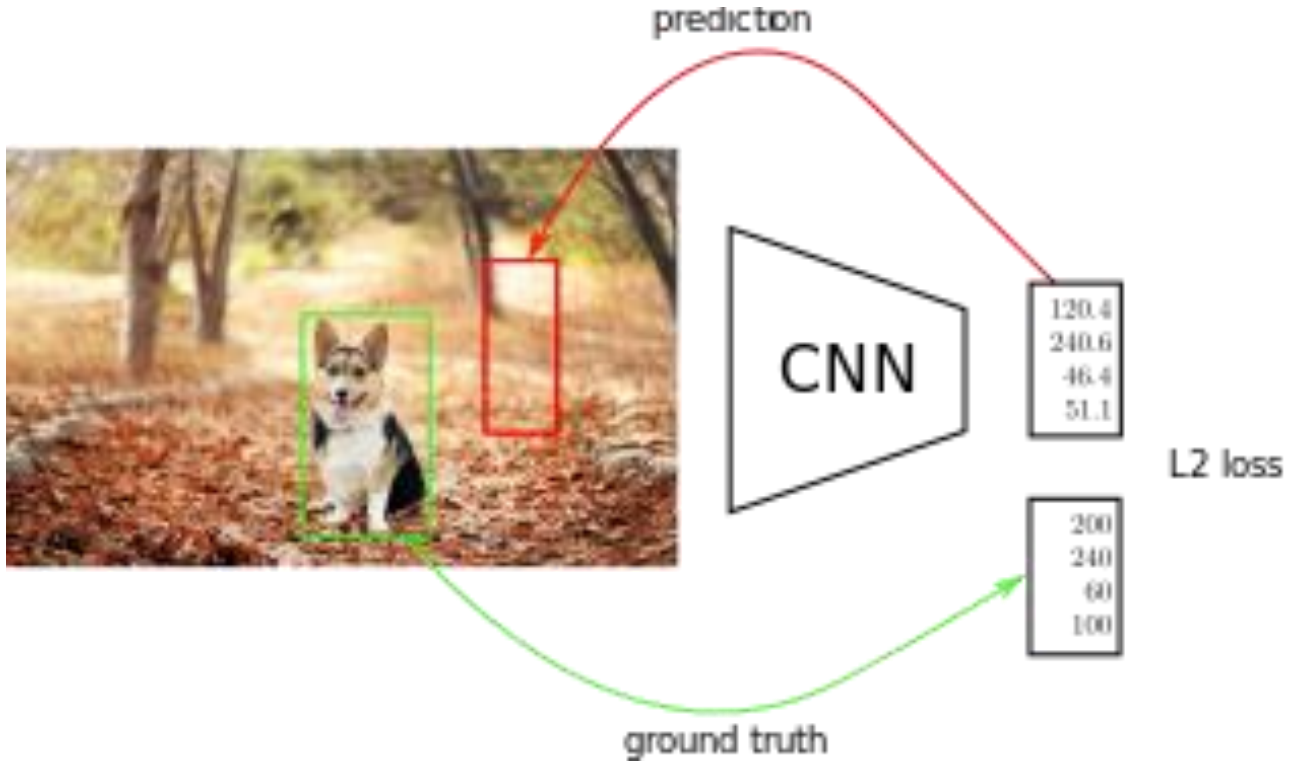


Introduction: Localization

Evaluate via Intersection over Union (IoU): Intersection over Union (IoU) is used to evaluate the performance of object detection by comparing the ground truth bounding box to the predicted bounding box.

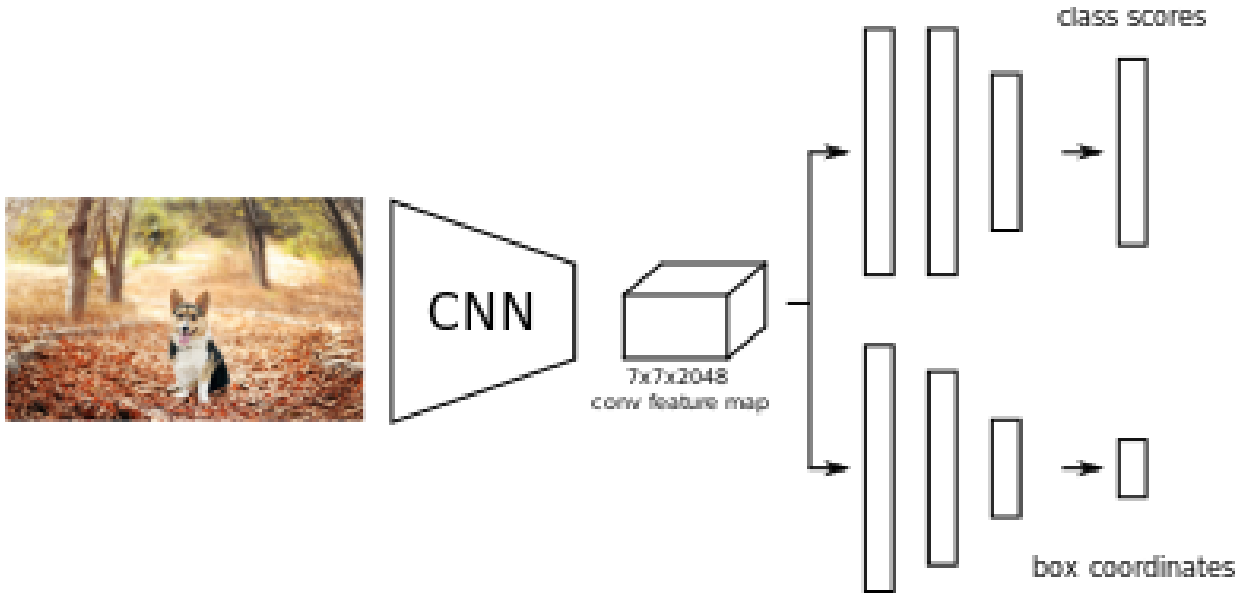


Introduction: Localization as Regression



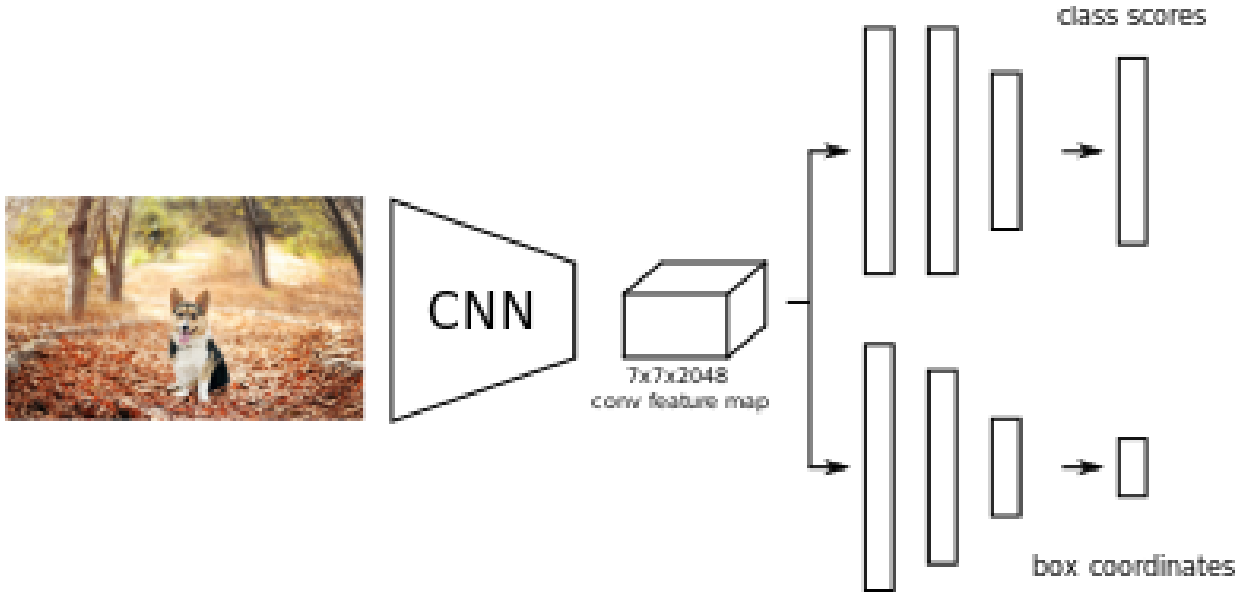
Introduction: Classification + Localization

- Use a pre-trained CNN on ImageNet (ex. ResNet)
- The "localization head" is trained separately with regression
- Possible end-to-end finetuning of both tasks
- At test time, use both heads



Introduction: Classification + Localization

- **C** classes, 4 output dimensions (1 box)
- Predict exactly N objects: predict (N×4) coordinates and (N×K) class scores



Introduction: Object Detection

Image Classification: Is this a cat or a dog?

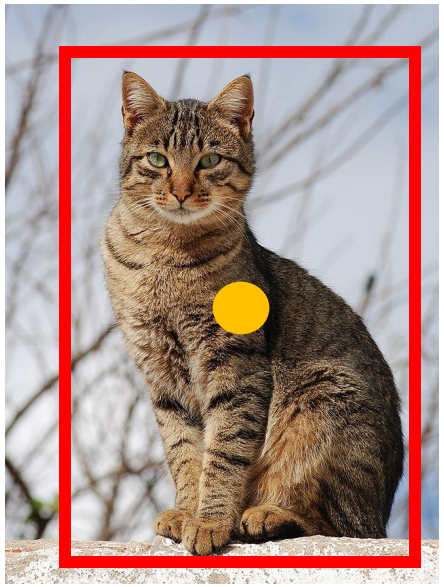
Output: Dog = 0 | Cat = 1

Object Detection: Where exactly is the cat in this image?

Output: Dog = 0 | Cat = 1 + Bounding Box (Red)

Introduction: Object Detection

Object Detection and Localization

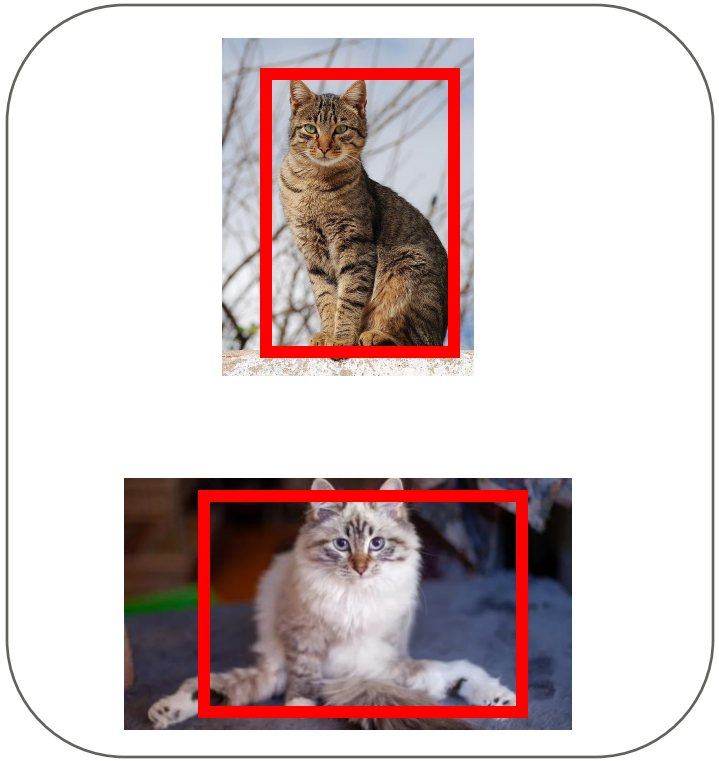


Output

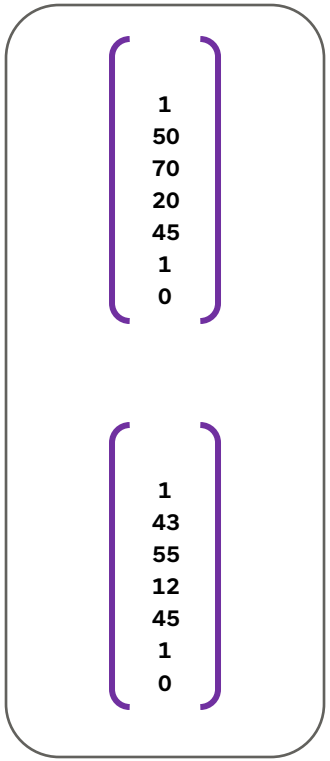
C_1 = Cat
C_2 = Dog

P_c	1
B_x	50
B_y	70
B_w	20
B_h	45
C_1	1
C_2	0

Introduction: Object Detection



X_train



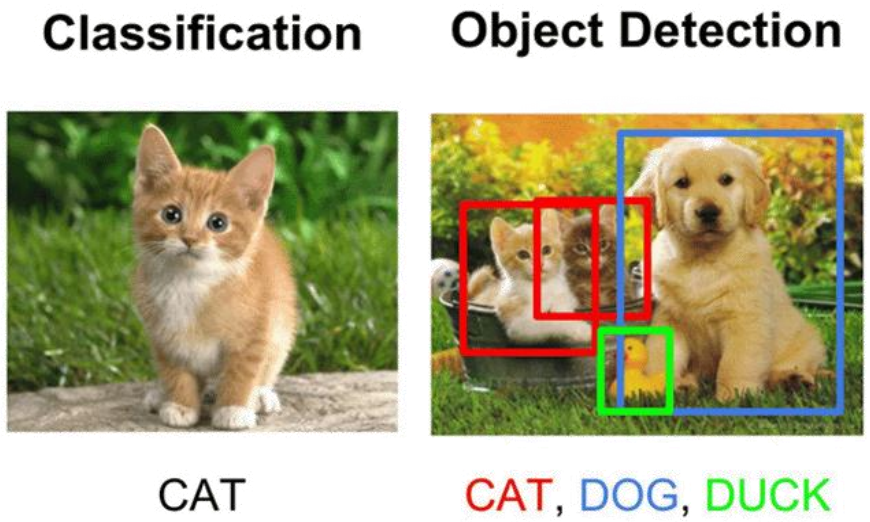
y_train

CNN

P_c
B_x
B_y
B_w
B_h
C_1
C_2

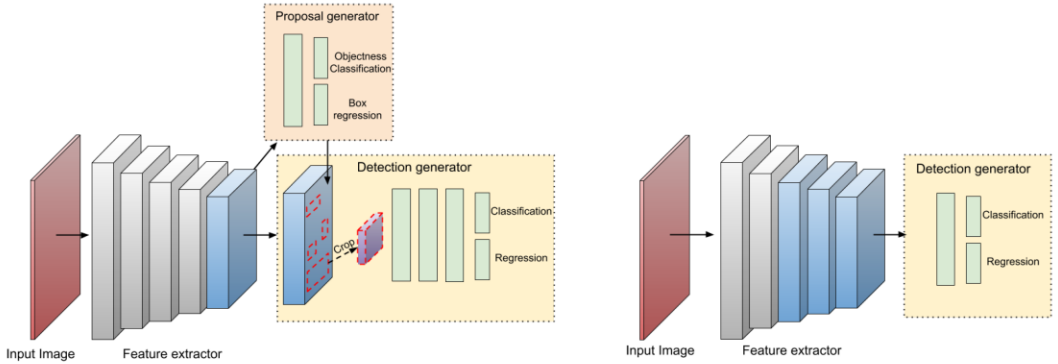
Introduction: Object Detection

- **We don't know in advance the number of objects in the image.**
  - Unlike image classification, where the goal is to assign a single label to an entire image, object detection is concerned with identifying multiple objects and their precise locations within an image. The number of objects in an image can vary, and we don't have prior information about how many there are.
- **Object detection** relies on **object proposal** and **object classification**
  - **Object detection** involves two main steps: proposing regions in the image where objects might be located (object proposal), and then classifying these proposed regions to determine what objects are present (object classification).
  - **Object proposal:** find regions of interest (**RoIs**) in the image
  - **Object classification:** classify the object in these regions
- **Two main families:**
  - Single-Stage: A grid in the image where each cell is a proposal (SSD, YOLO, RetinaNet)
  - Two-Stage: Region proposal then classification (Faster-RCNN)



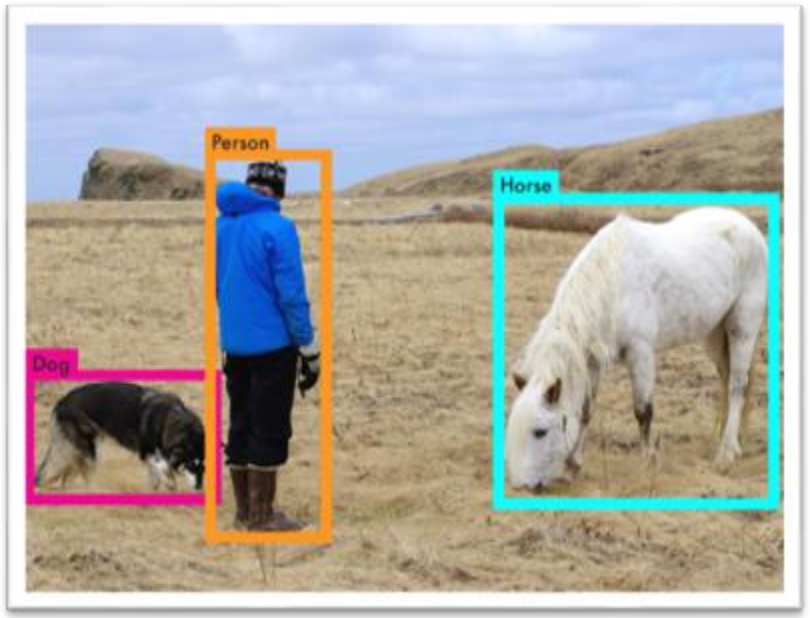
Introduction: Single-Stage vs. Two-Stage Object Detection

- **Single-Stage:**
  - In single-stage detectors, the process of proposing regions and classifying objects is done simultaneously in a single step. This means that a grid is applied to the entire image, and each cell in the grid is responsible for both proposing regions and classifying objects. Examples of single-stage detectors include SSD (Single Shot MultiBox Detector), YOLO (You Only Look Once), and RetinaNet.
- **Two-Stage:**
  - In two-stage detectors, the process is divided into two steps. The first stage involves generating region proposals, which are areas in the image likely to contain objects. These proposed regions are then passed to the second stage, which performs the actual object classification. An example of a two-stage detector is Faster R-CNN (Region-based Convolutional Neural Network).



(a) Two-stage Faster R-CNN (b) One-stage RetinaNet

YOLO: You Only Look Once



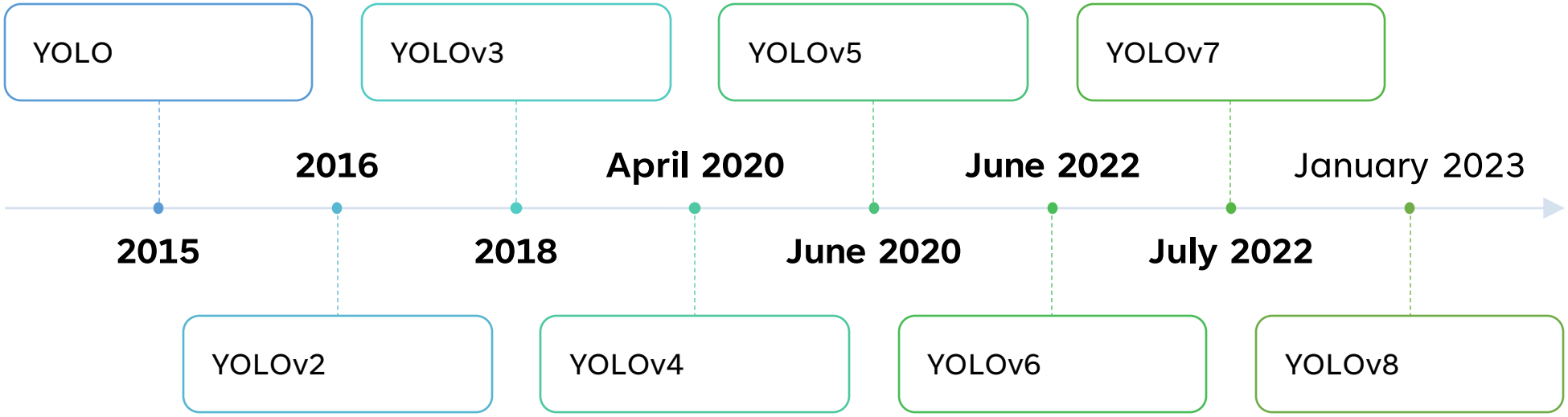
Paper Title: You Only Look Once: Unified, Real-Time Object Detection

YOLO: You Only Look Once

You Only Look Once (YOLO) is a **state-of-the-art**, **real-time** object detection algorithm introduced in **2015** by **Joseph Redmon**, **Santosh Divvala**, **Ross Girshick**, and **Ali Farhadi** in their famous research paper "**You Only Look Once: Unified, Real-Time Object Detection**".
The authors frame the object detection problem as a **regression** problem instead of a **classification** task by spatially separating bounding boxes and associating probabilities to each of the detected images using a single convolutional neural network (CNN).

YOLO for Object Detection: State-of-the-art

YOLO for Object Detection: State-of-the-art



Ultralytics YOLOv8:
[The State-of-the-Art YOLO Model]

YOLO for Object Detection: What Makes YOLO Popular for Object Detection?

Real-time Detection:

- YOLO is designed for real-time object detection. It can process images and provide object detections in a single pass, making it significantly faster than many other object detection algorithms.

Single-shot Detection:

- Unlike traditional object detection methods that involve multiple stages (like region proposal networks followed by classification and refinement), YOLO is a single-shot detector. This means it predicts bounding boxes and class probabilities directly from the entire image, which leads to faster inference times.

Efficiency and Speed:

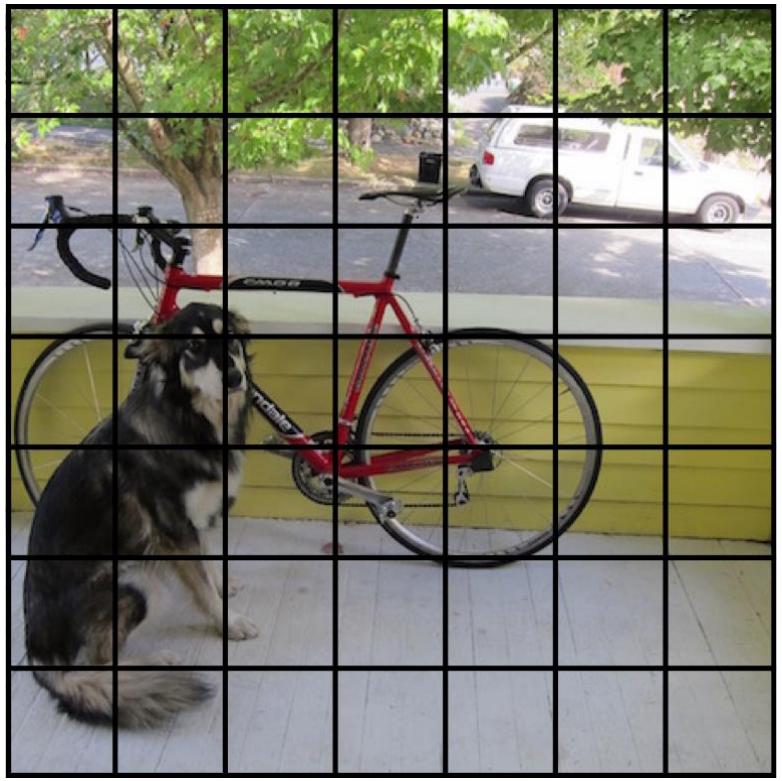
- YOLO's architecture is efficient and can be optimized for deployment on various hardware platforms, including CPUs, GPUs, and even specialized hardware like TPUs (Tensor Processing Units).
- YOLO is far beyond other state-of-the-art models in accuracy with very few background errors.

Better generalization:

- This is especially true for the new versions of YOLO. With those advancements, YOLO pushed a little further by providing a better generalization for new domains, which makes it great for applications relying on fast and robust object detection.
- For instance, the Automatic Detection of Melanoma with Yolo Deep Convolutional Neural Networks paper shows that the first version YOLOv1 has the lowest mean average precision for the automatic detection of melanoma disease, compared to YOLOv2 and YOLOv3.

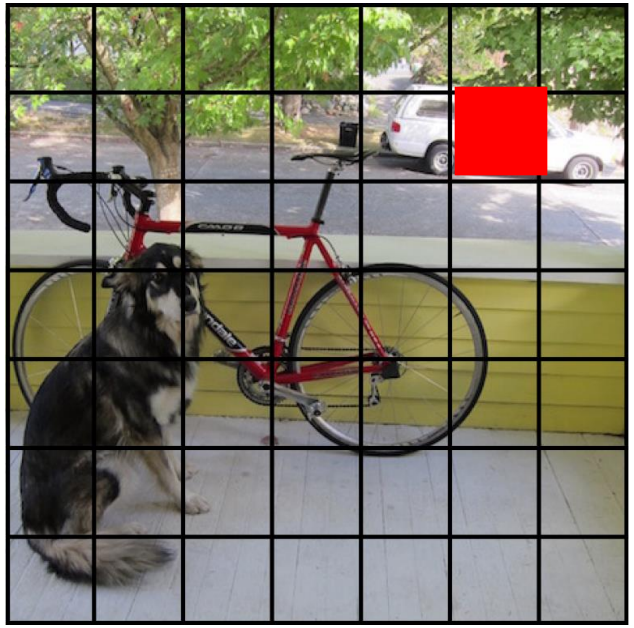
YOLO for Object Detection: Understanding YOLO

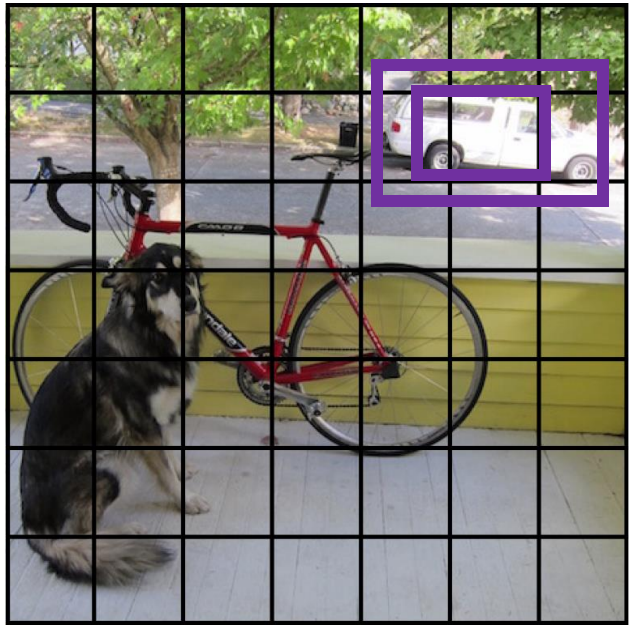
We split the image into an S*S grid (S = 7)



YOLO for Object Detection: Understanding YOLO

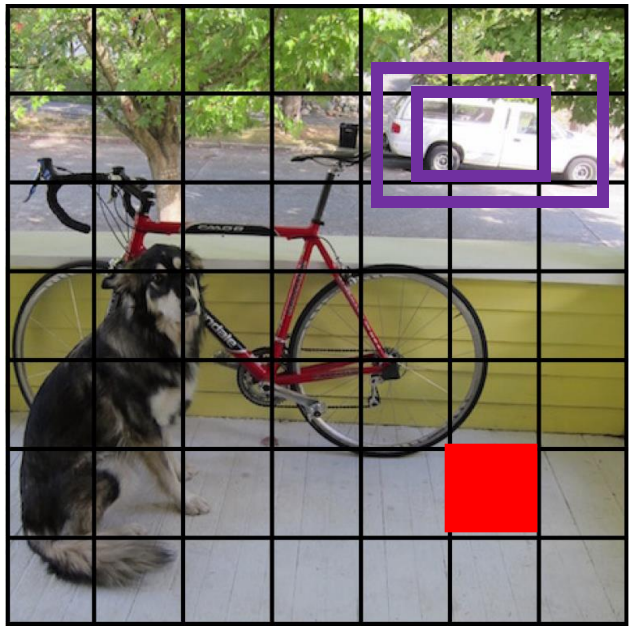
Each cell predicts boxes and confidences: P(object)

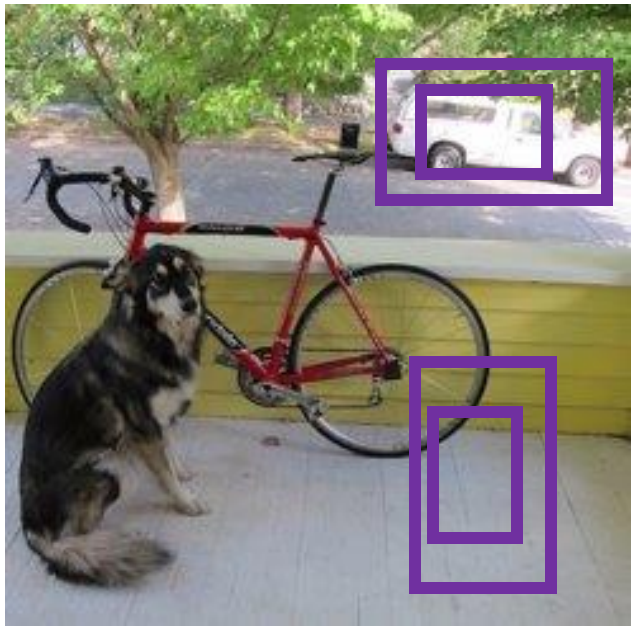




YOLO for Object Detection: Understanding YOLO

Each cell predicts boxes and confidences: P(object)





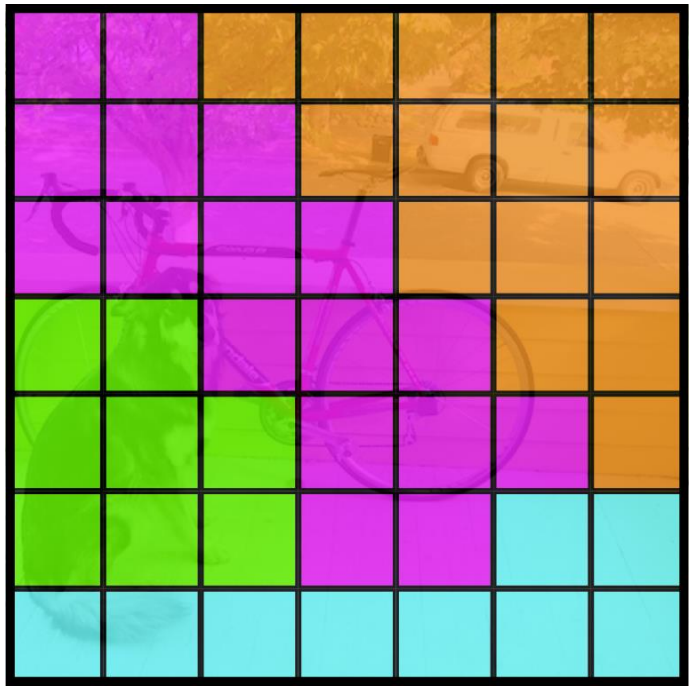
YOLO for Object Detection: Understanding YOLO

Each cell predicts boxes and confidences: P(object)

YOLO for Object Detection: Understanding YOLO

Each cell predicts a class probability

Bicycle



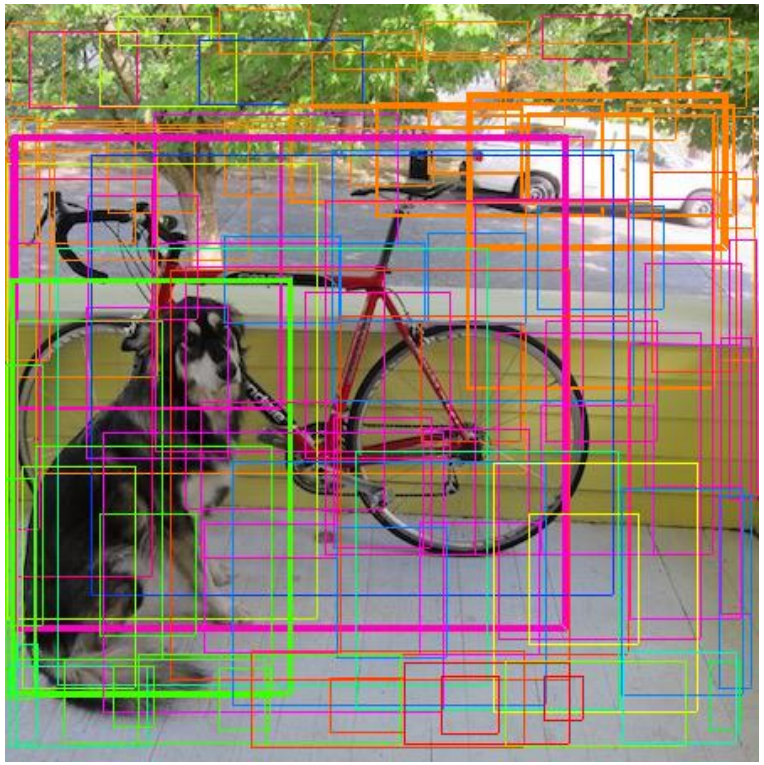
Car

Dog

Dining table

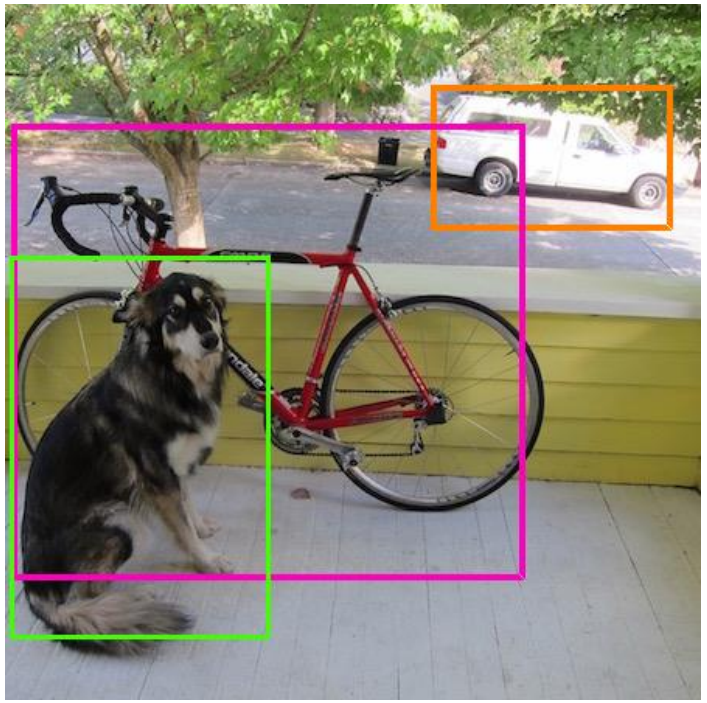
YOLO for Object Detection: Understanding YOLO

Then we combine the box and class predictions

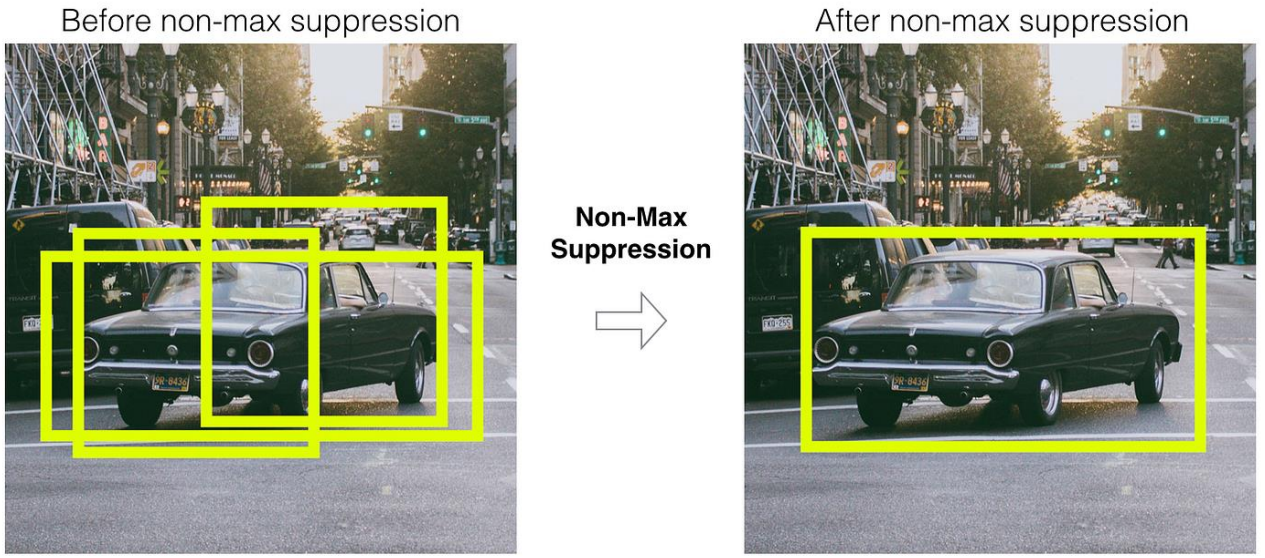


YOLO for Object Detection: Understanding YOLO

Finally, we do NMS and threshold detections



YOLO for Object Detection: Understanding YOLO



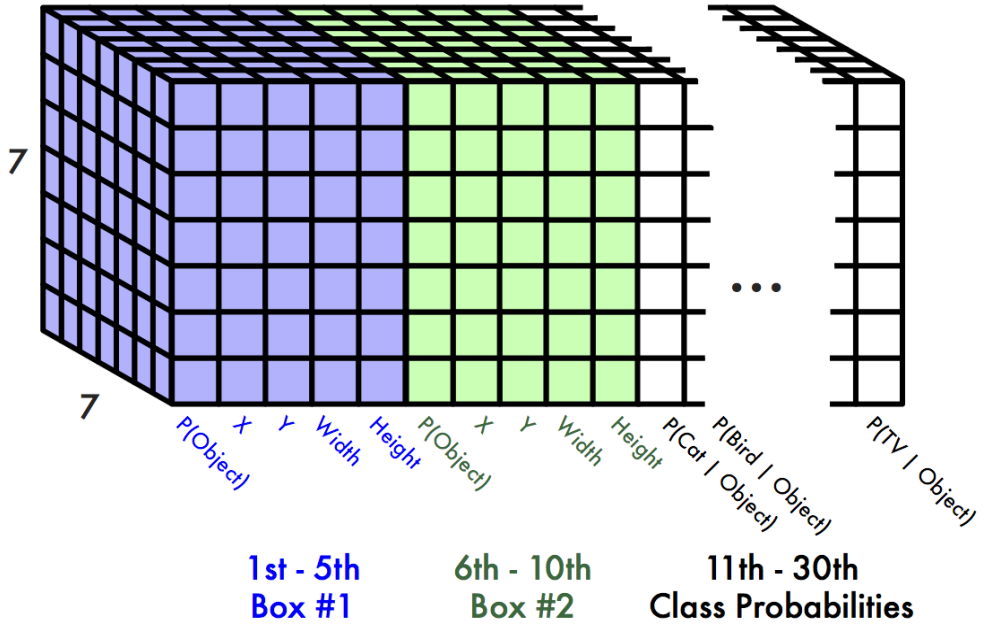
YOLO for Object Detection: Understanding YOLO

This parameterization fixes the output size

Each cell predicts:
- For each bounding box:
- 4 coordinates (x, y, w, h)
- 1 confidence value
- Some number of class probabilities

For Pascal VOC:
- 7x7 grid
- 2 bounding boxes / cell
- 20 classes

7 x 7 x (2 x 5 + 20) = 7 x 7 x 30 tensor = **1470 outputs**



YOLO for Object Detection: Understanding YOLO

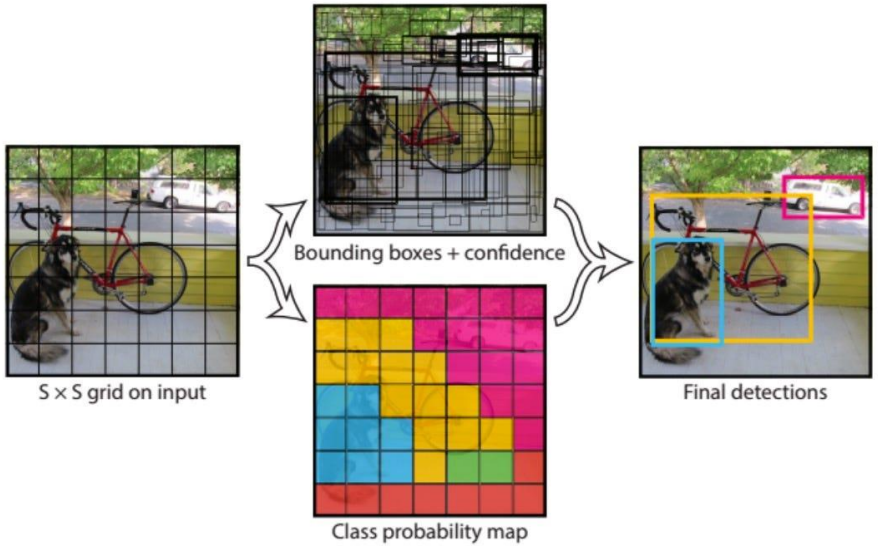
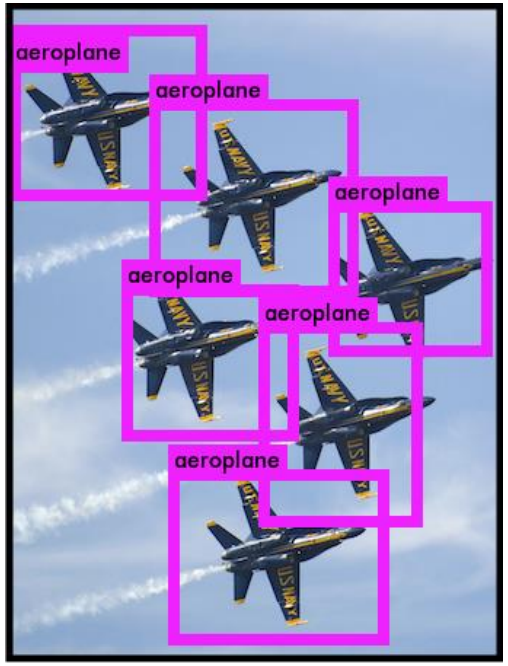
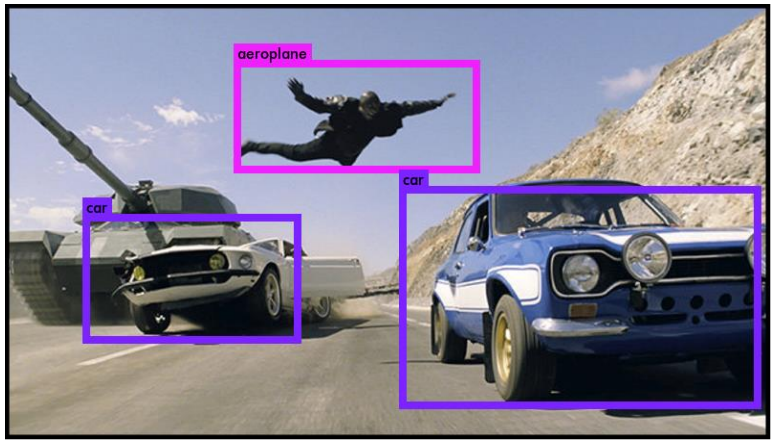
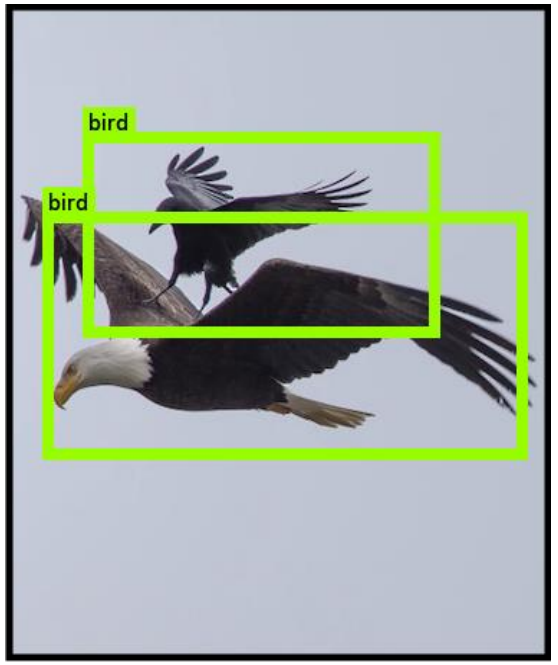


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

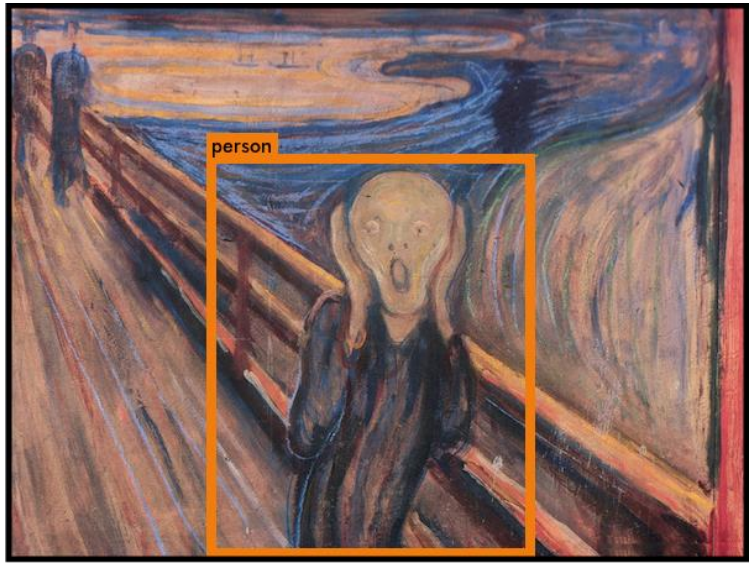
YOLO for Object Detection: YOLO works across a variety of natural images







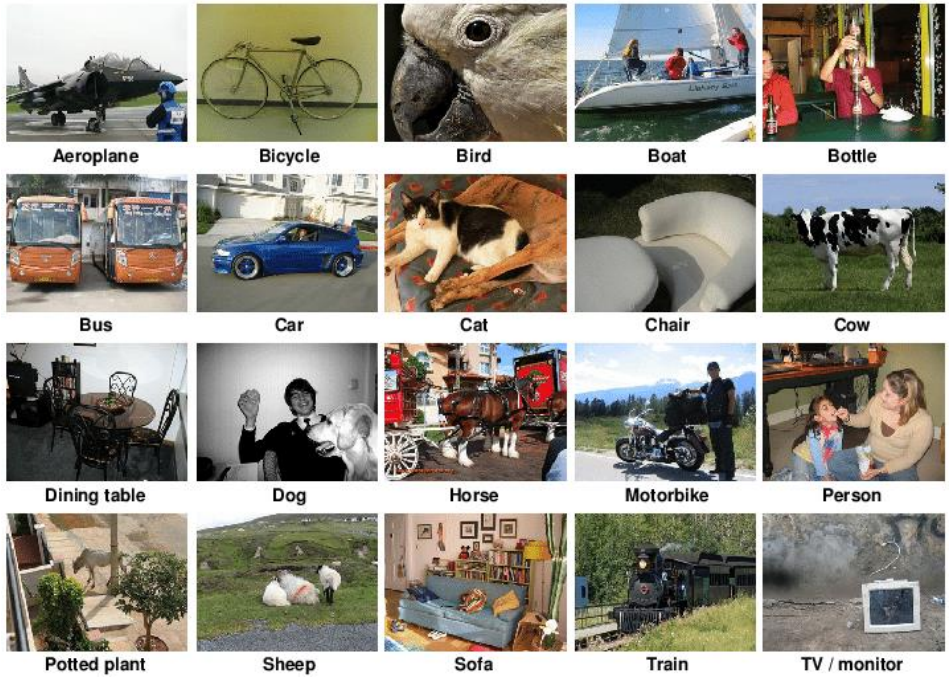
YOLO for Object Detection: It also generalizes well to new domains (like art)



YOLO for Object Detection: Understanding YOLO <Training Dataset>

Link to the paper: The PASCAL Visual Object Classes Challenge: A Retrospective

- **Pascal VOC detection dataset:**
  - The PASCAL VOC (Visual Object Classes) dataset is another widely used benchmark in computer vision, specifically for object detection, segmentation, and classification tasks. It was created to encourage the development and evaluation of algorithms for object detection and related tasks.
- 20 classes
- A total of 11500 images
- **Note:** Most of the recent papers use COCO instead of Pascal VOC



YOLO for Object Detection: Understanding YOLO <Training Dataset>

Link to the dataset: [COCO Dataset]

- **COCO Dataset:** The COCO dataset, which stands for Common Objects in Context, is a widely used benchmark in computer vision. It is designed for object recognition, segmentation, and captioning tasks. The dataset is curated by the Microsoft Research team and contains a large collection of images with objects in complex scenes.
- Over 80 object categories
- around 330,000 images

YOLO for Object Detection: Architecture and Implementation

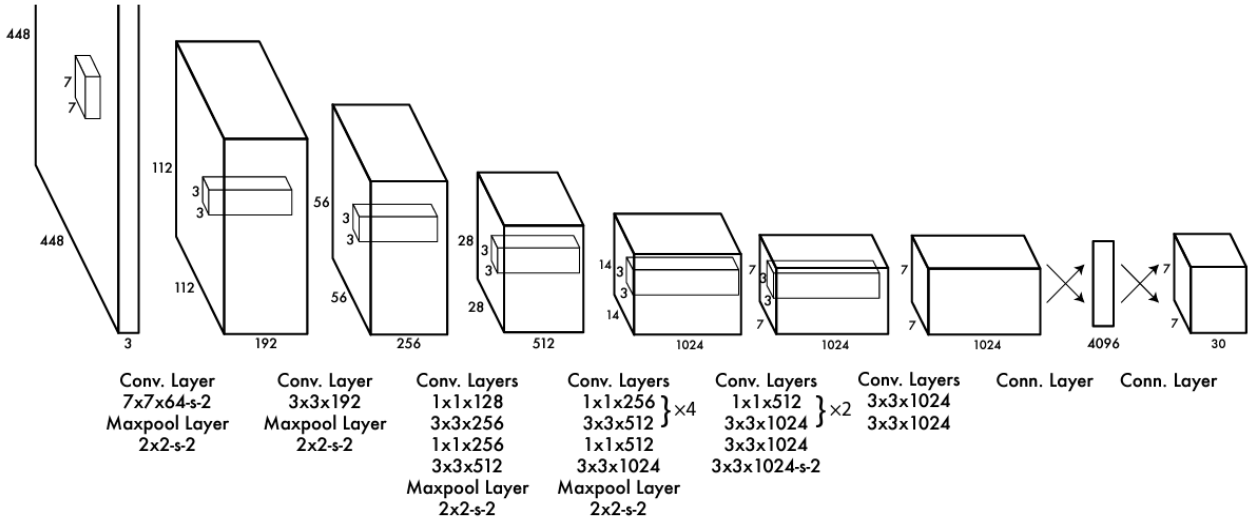
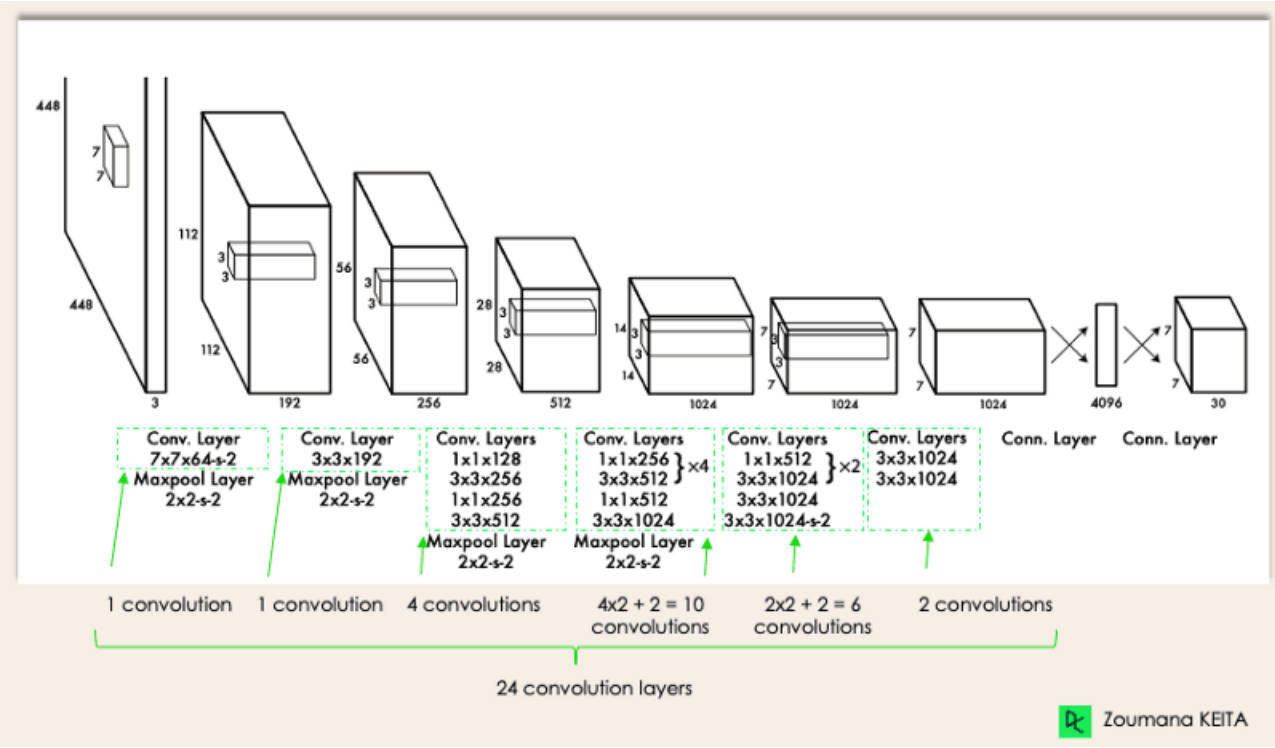


Fig. YOLO Architecture

Paper Link: [You only look once: Unified, real-time object detection]

YOLO for Object Detection: Architecture and Implementation



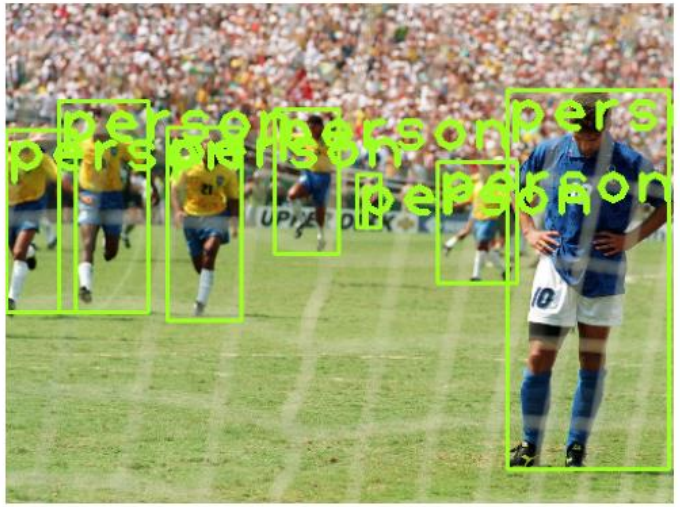
Paper Link: [You only look once: Unified, real-time object detection]

Computer Vision Libraries: OpenCV

- **OpenCV** is a great tool for image processing and performing computer vision tasks. It is an open-source library that can be used to perform tasks like face detection, objection tracking, landmark detection, and much more.
- **OpenCV** was originally developed by **Intel** in June 2000 and has since been maintained by a community of developers. It is written in C++ and has bindings for various programming languages including Python, Java, and MATLAB, making it accessible to a wide range of developers.

OpenCV and YOLOv3 for Object Detection

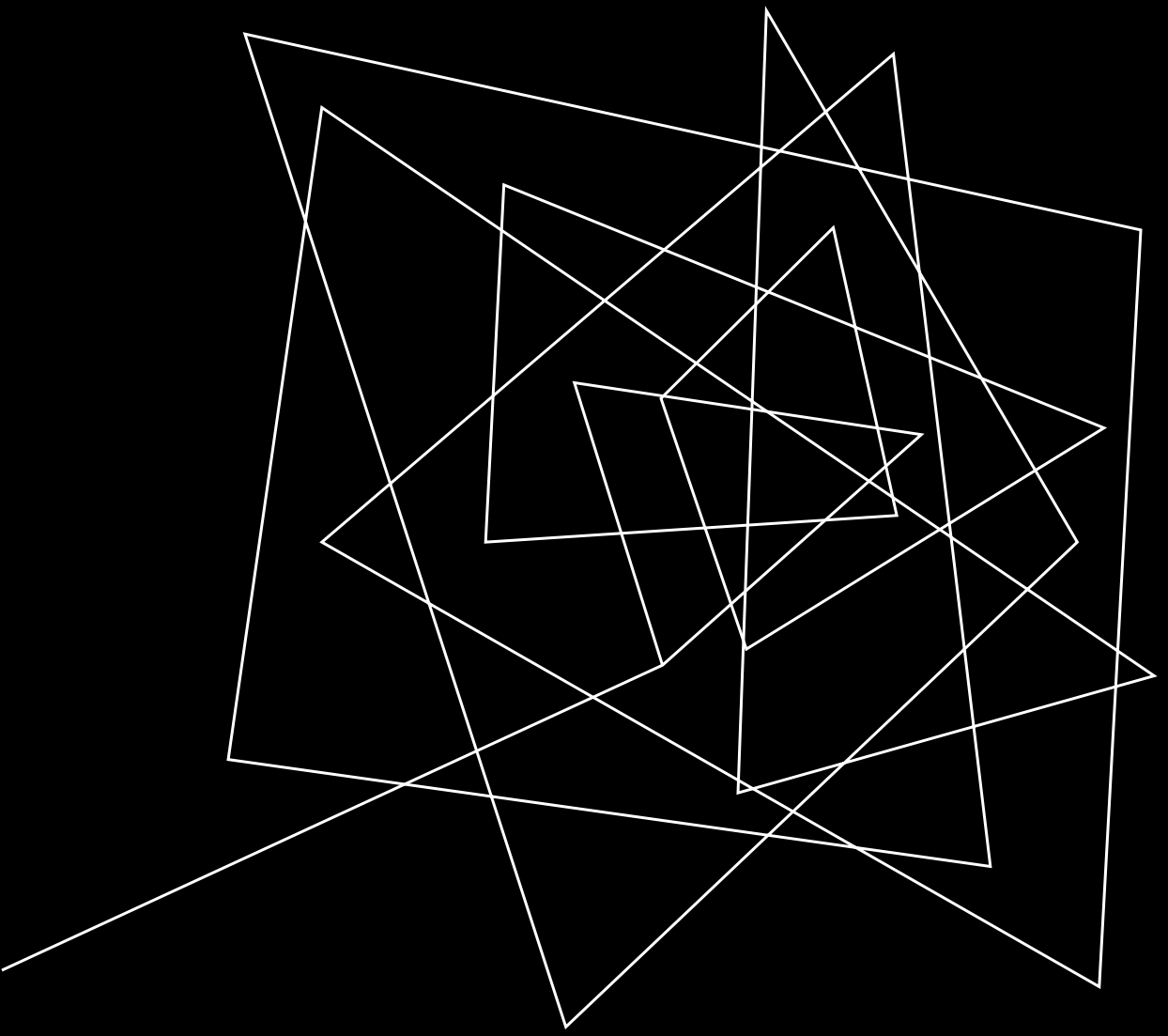
Demo: Session 4- YOLO for Object Detection



Tutorial: YOLO Object Detection

YOLO for Object Detection: TO-DO Project

Project: [Fruit and Vegetable Detection using YOLO]



U-NET FOR BIOMEDICAL IMAGE SEGMENTATION

U-Net and its variants

Introduction: So far Image Classification and Object Detection

This image is CC0 public domain

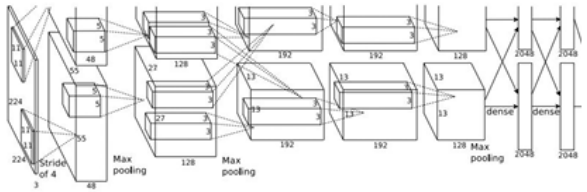
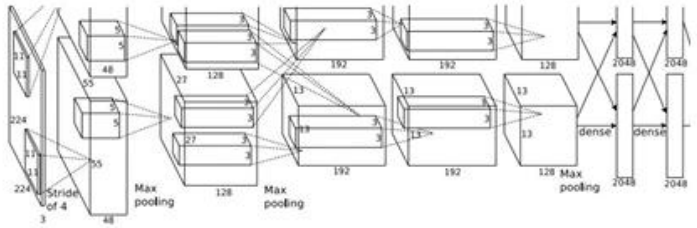


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector:
4096

Fully-Connected:
4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...



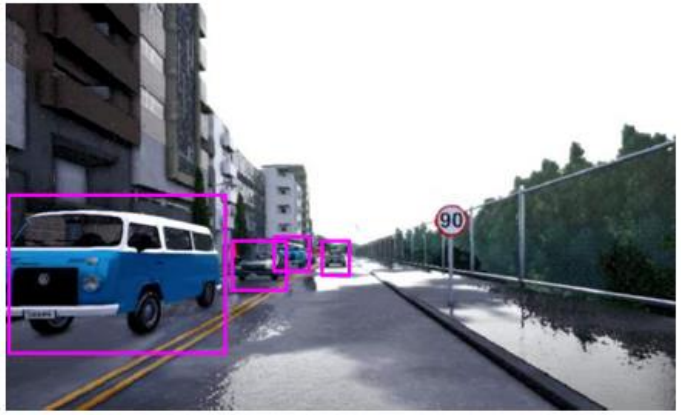
CAT: (x, y, w, h)

Introduction: Image Segmentation

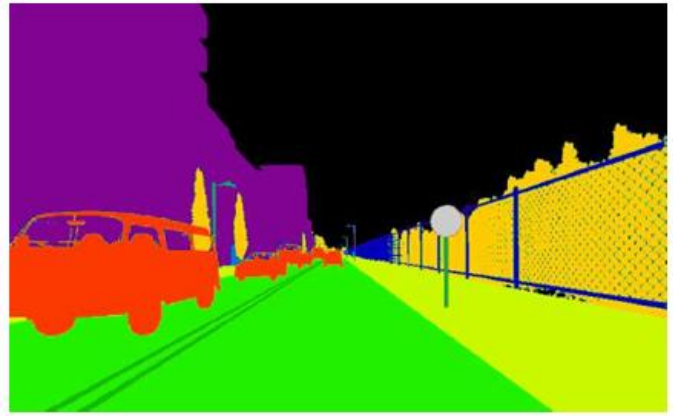
Definition: Image segmentation is a fundamental task in computer vision that involves dividing an image into multiple segments or regions, each of which corresponds to a meaningful object or part of the scene. The goal of image segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze.

Object Detection vs. Image Segmentation

Input image



Object detection



Semantic segmentation

Image Segmentation: Semantic Segmentation vs. Instance Segmentation

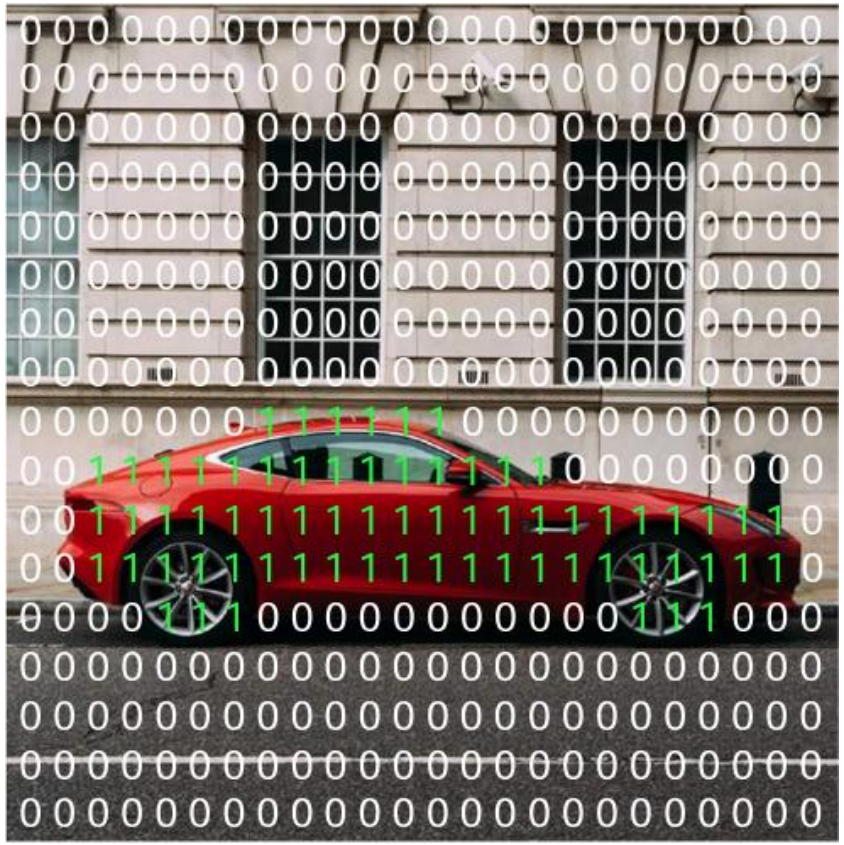
Semantic segmentation and **instance segmentation** are two different approaches to **segmenting objects** in an **image**, and they serve distinct purposes:

- **Semantic Segmentation:**
  - **Definition:** Semantic segmentation involves labeling each pixel in an image with a class label that corresponds to a specific category or object. It classifies each pixel into predefined categories, such as person, car, road, sky, etc.
  - **Output:** The output of semantic segmentation is a high-resolution map where each pixel is assigned a class label. Pixels belonging to the same class share similar visual characteristics.
  - **Example:** In an image containing a person, a car, and a road, semantic segmentation would label each pixel to indicate which class it belongs to (e.g., all pixels corresponding to the person would be labeled as "person").
  - **Use Case:** Semantic segmentation is widely used in tasks like scene understanding, object recognition, and image-to-text descriptions.

Image Segmentation: Semantic Segmentation vs. Instance Segmentation

- **Instance Segmentation:**
  - Definition: Instance segmentation goes a step further than semantic segmentation. In addition to labeling each pixel with a class, it also distinguishes between individual instances of the same class. This means that if there are multiple objects of the same class (e.g., multiple cars or multiple persons) in the image, they will each be assigned a unique label.
  - Output: The output of instance segmentation provides a unique label for each object instance. This allows for precise delineation of individual objects.
  - Example: In an image with multiple cars, instance segmentation would label each car separately, assigning a unique identifier to each one.
  - Use Case: Instance segmentation is particularly important in scenarios where it is necessary to differentiate between individual objects of the same class, such as in robotics, autonomous driving, and object counting.

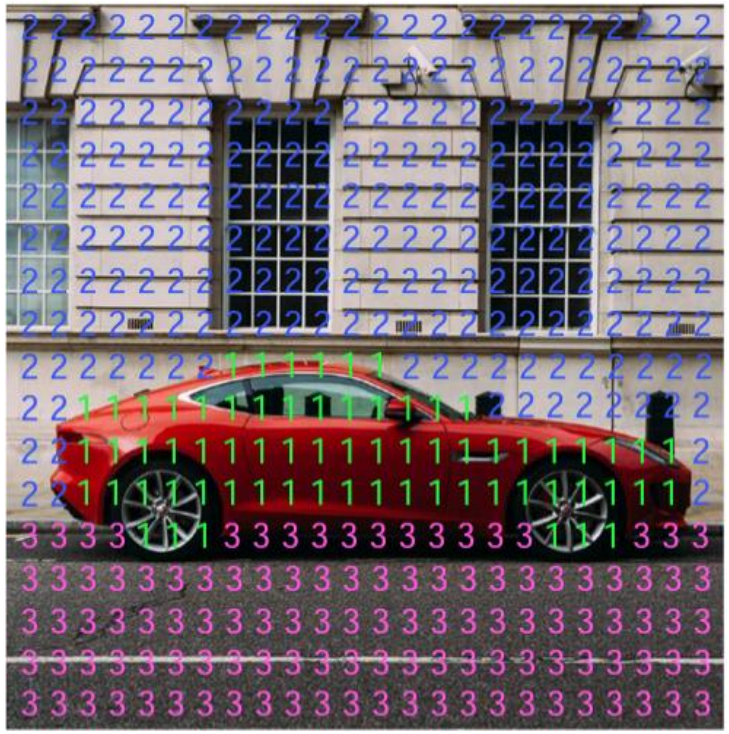
Per-pixel Class Labels



Class 0 = Not a car

Class 1 = Car

Per-pixel Class Labels



1. Car
2. Building
3. Road

Image Segmentation: Fully Convolutional Network (FCN)

A Fully Convolutional Network (FCN) is a type of neural network architecture designed for semantic segmentation tasks in computer vision. Semantic segmentation involves classifying each pixel in an image into specific categories, enabling the understanding of the scene at a pixel level. FCNs are particularly well-suited for tasks where spatial information is crucial, such as object detection and image segmentation.

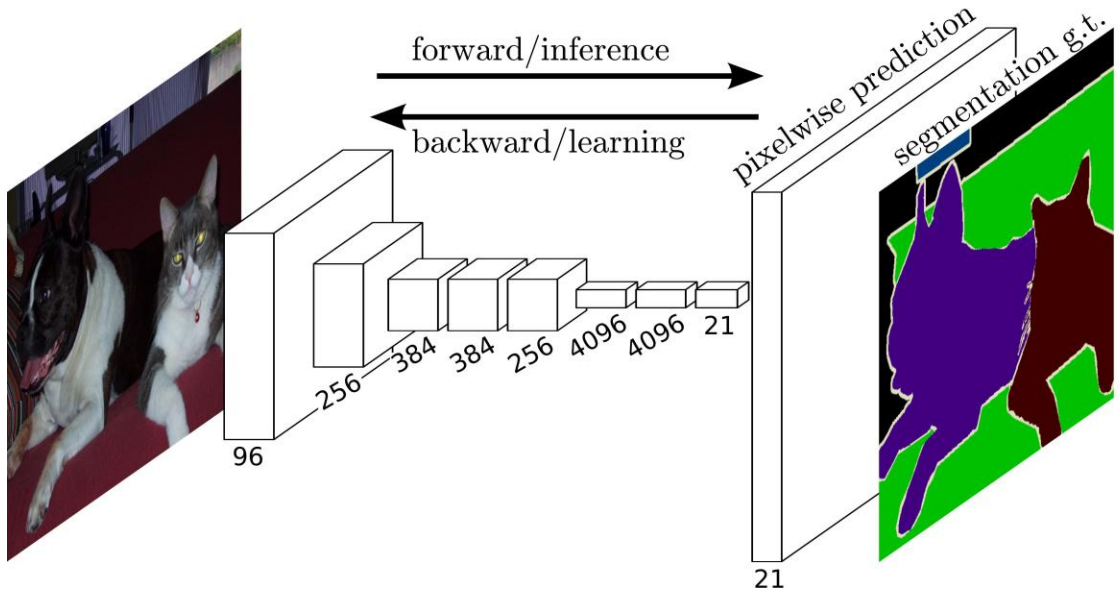
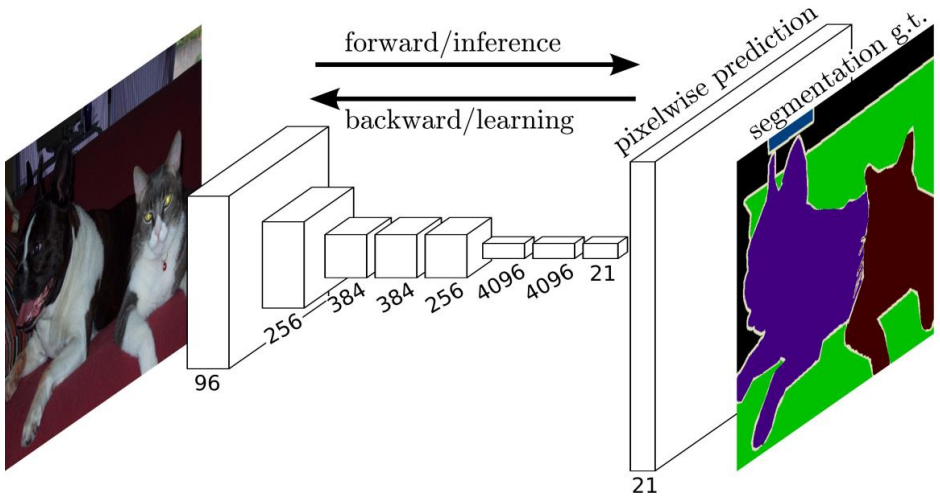


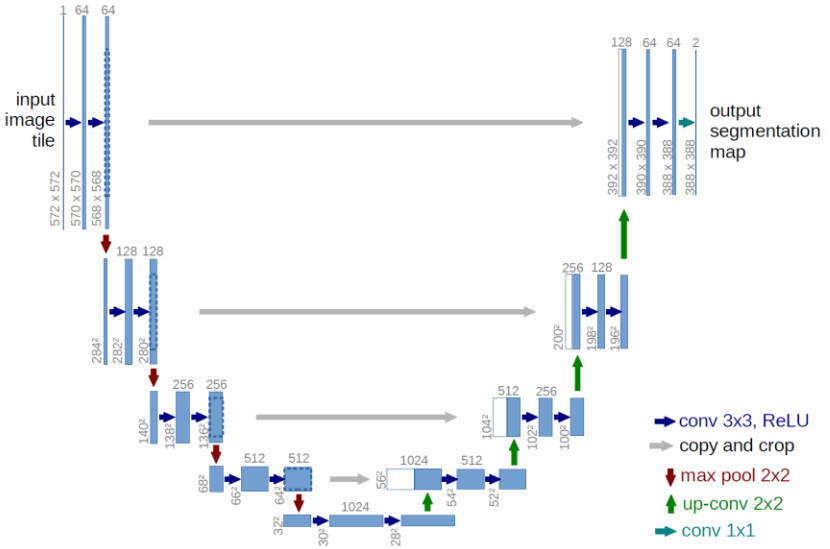
Image Segmentation: Fully Convolutional Network (FCN)

The original Fully Convolutional Network (FCN) architecture was introduced by Jonathan Long, Evan Shelhamer, and Trevor Darrell in the paper titled "Fully Convolutional Networks for Semantic Segmentation" in 2015. FCNs have since become a fundamental building block for many state-of-the-art models in semantic segmentation and related computer vision tasks. **Variants** of **FCNs**, such as **U-Net**, **SegNet**, and **Deeplab**, have been developed to address specific challenges and improve performance in different scenarios.

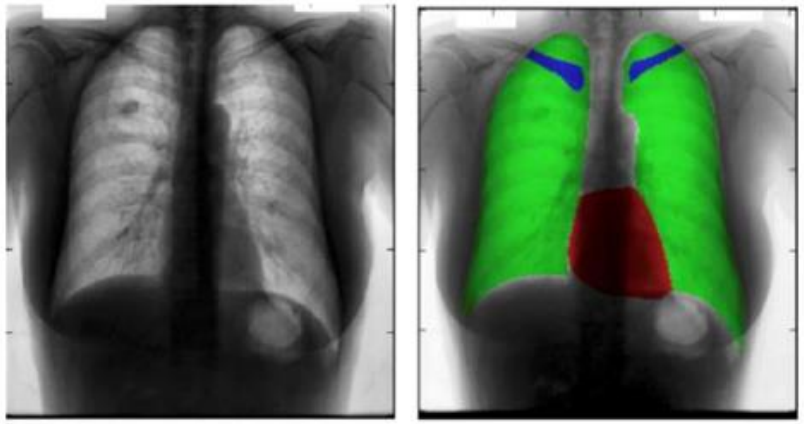


U-NET: Introduction

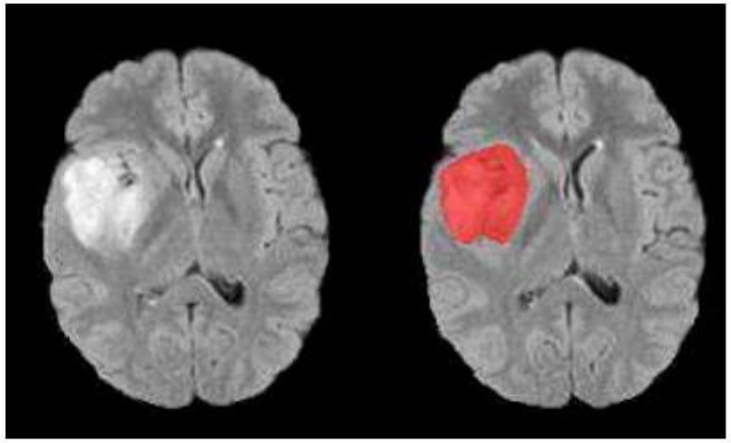
- **U-Net** is a popular **convolutional neural network architecture** designed for image segmentation tasks. It was introduced by **Olaf Ronneberger**, **Philipp Fischer**, and **Thomas Brox** in their 2015 paper titled "U-Net: Convolutional Networks for Biomedical Image Segmentation."
- The architecture of **U-Net** is characterized by its **U-shaped structure (Figure on the right)**, which consists of a **contracting path (left side)** and an **expansive path (right side).** This unique design allows the network to capture both local and global information, making it **highly effective for tasks like biomedical image segmentation.**
- **Fun Fact:** When the researchers wrote the original U-NET paper, they were thinking of the application of biomedical image segmentation. But these ideas turned out to be useful for many other computer vision semantic segmentation applications as well.



Motivations for U-NET



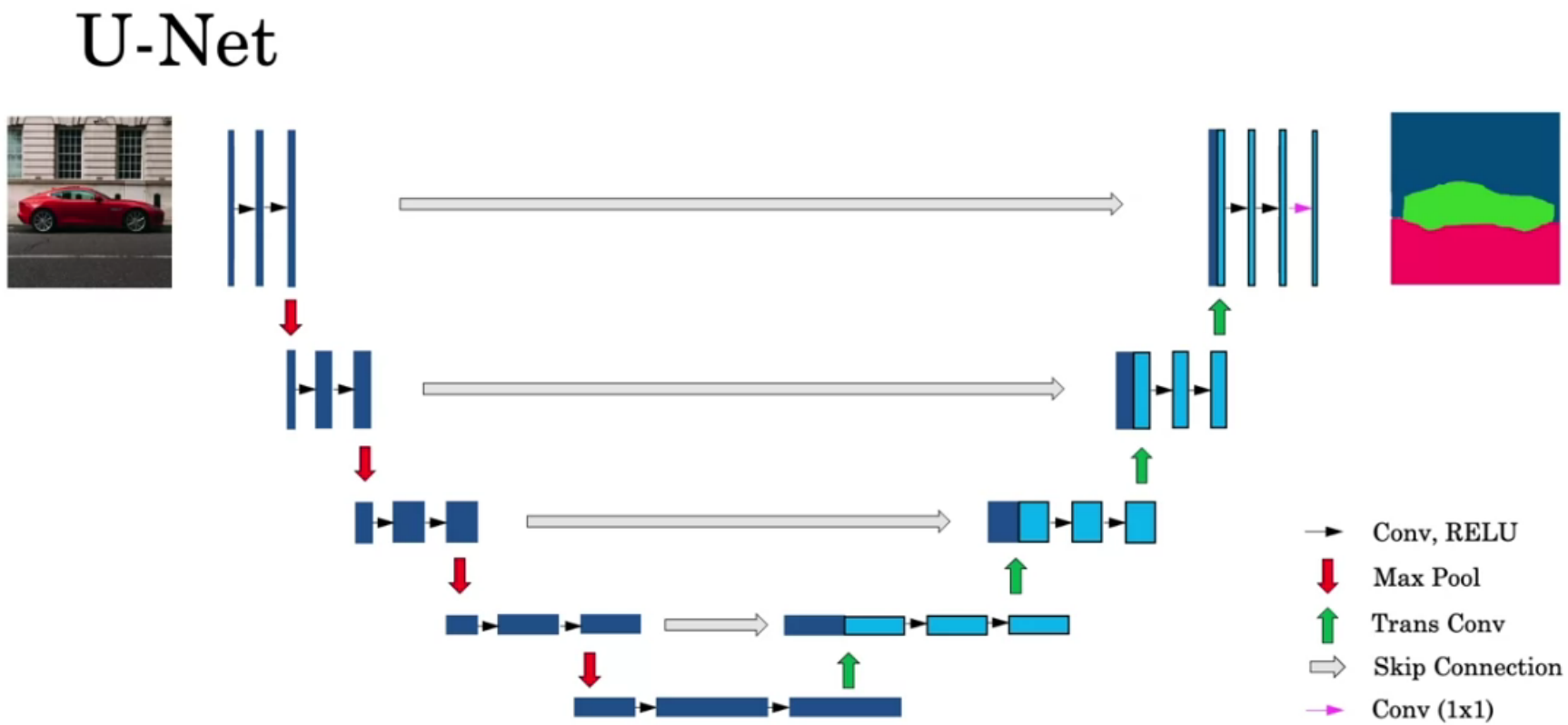
Chest X-Ray



Brain MRI

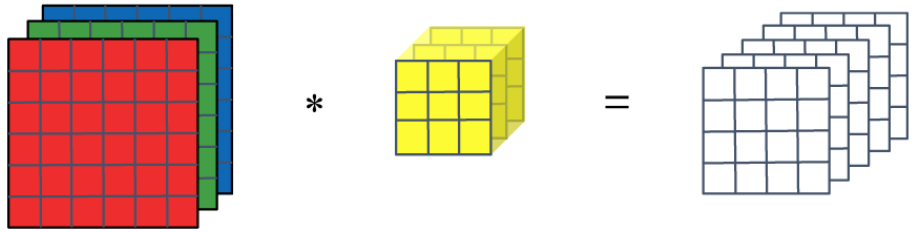
- [Novikov et al., 2017, Fully Convolutional Architectures for Multi-Class Segmentation in Chest Radiographs]
- [Dong et al., 2017, Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks]

U-NET: Architecture

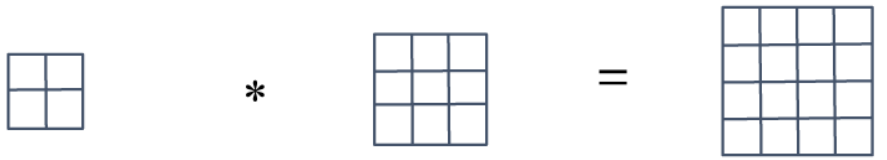


U-NET: Transpose Convolution

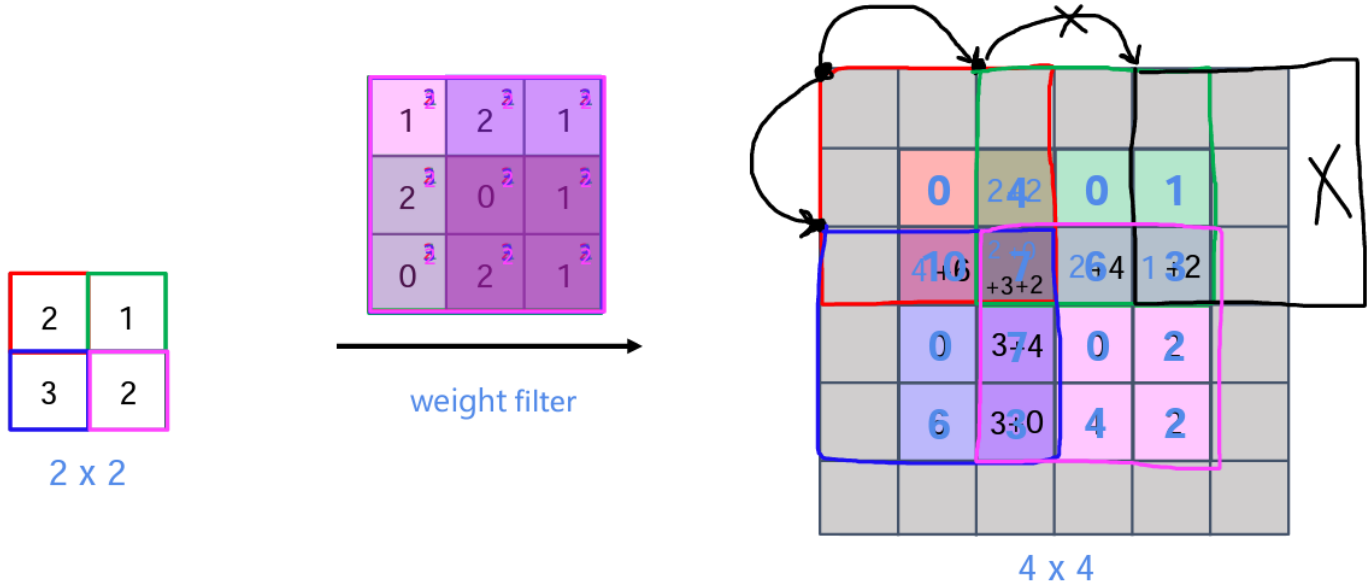
Normal Convolution



Transpose Convolution



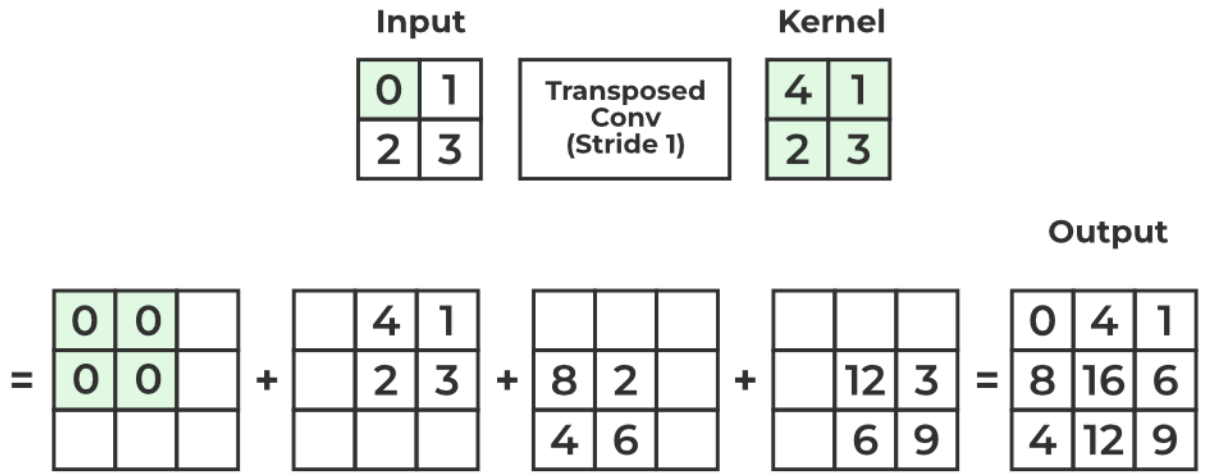
U-NET: Transpose Convolution



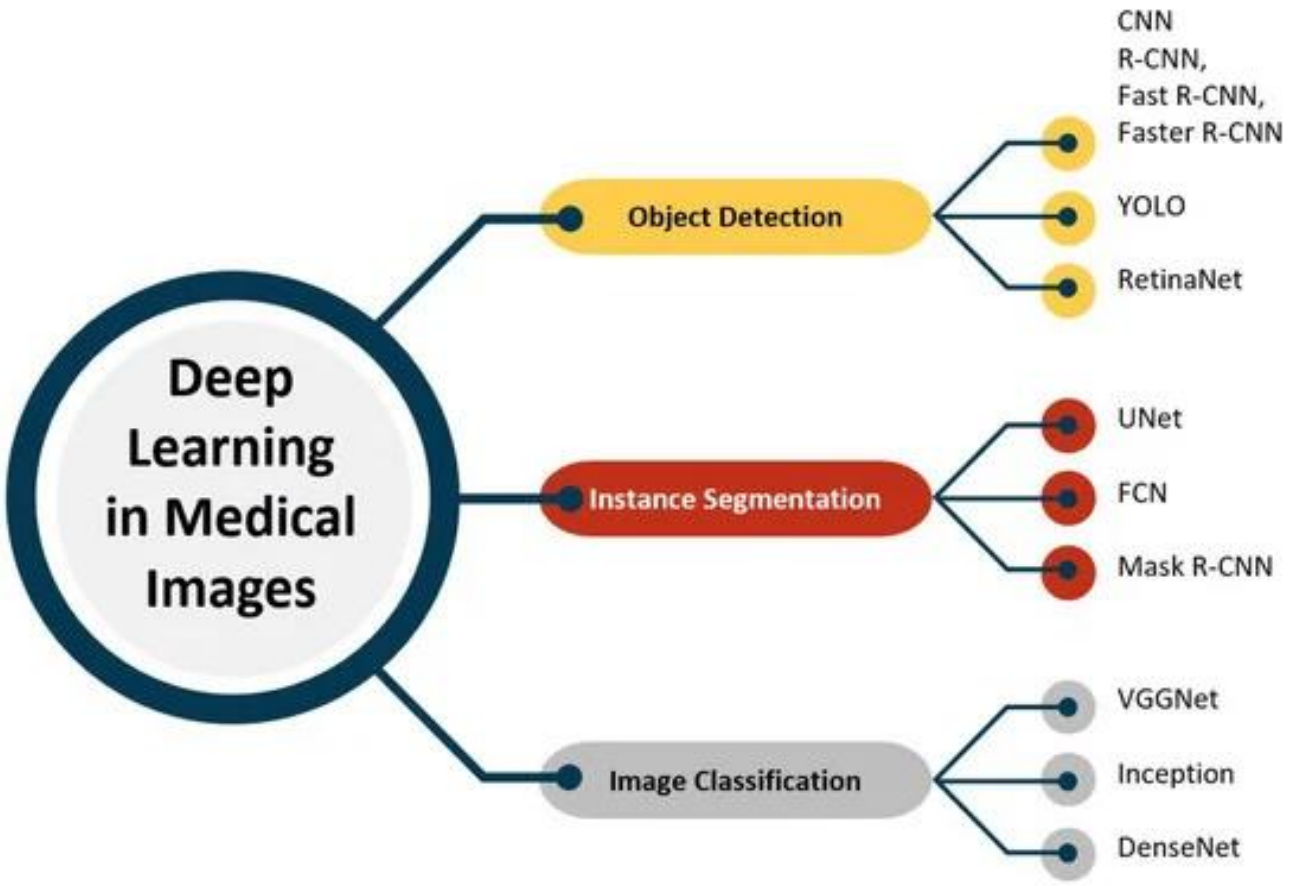
filter f x f = 3 x 3 padding p = 1 stride s = 2

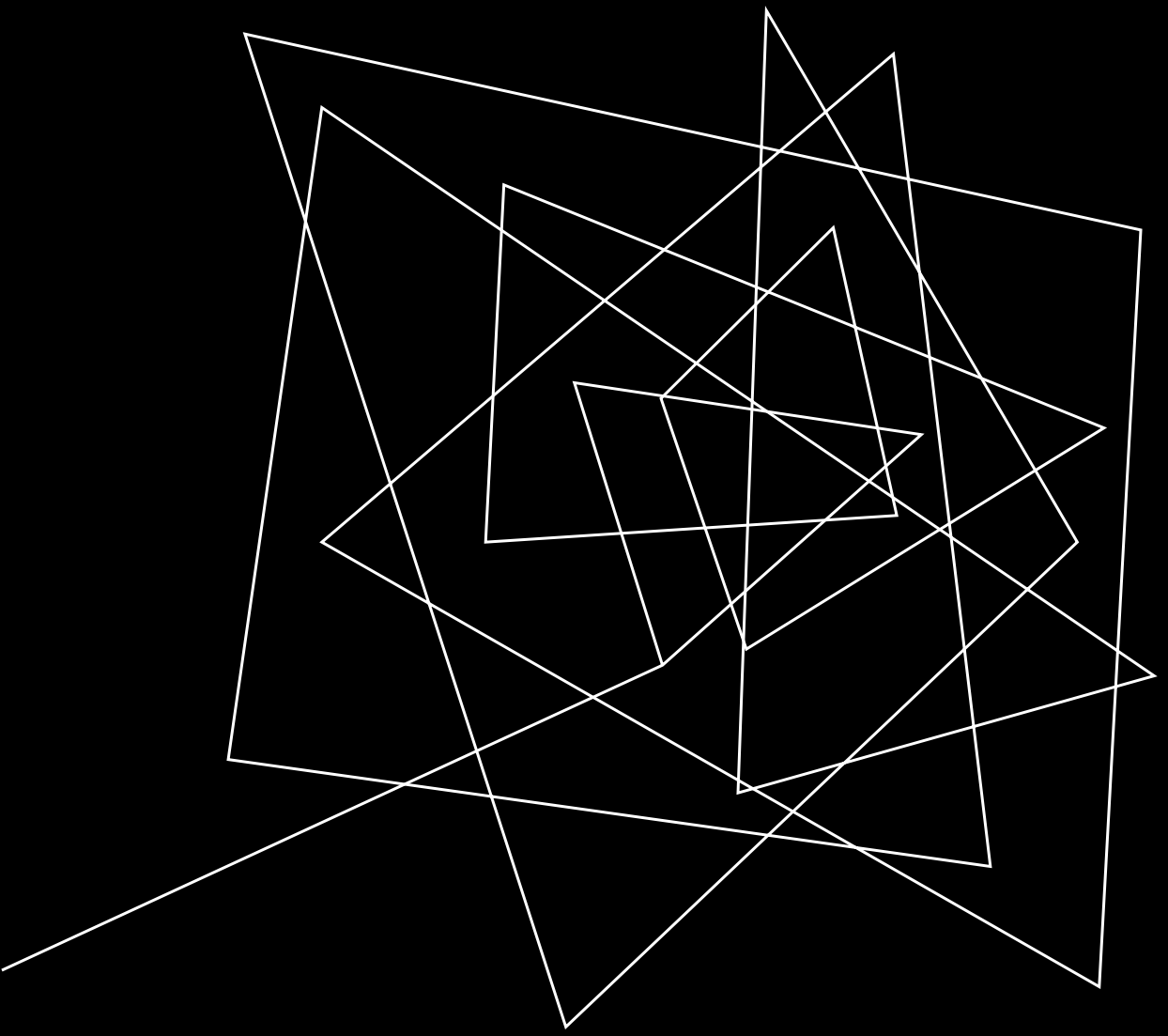
Source: Coursera

U-NET: Transpose Convolution



Deep Learning in Medical Images

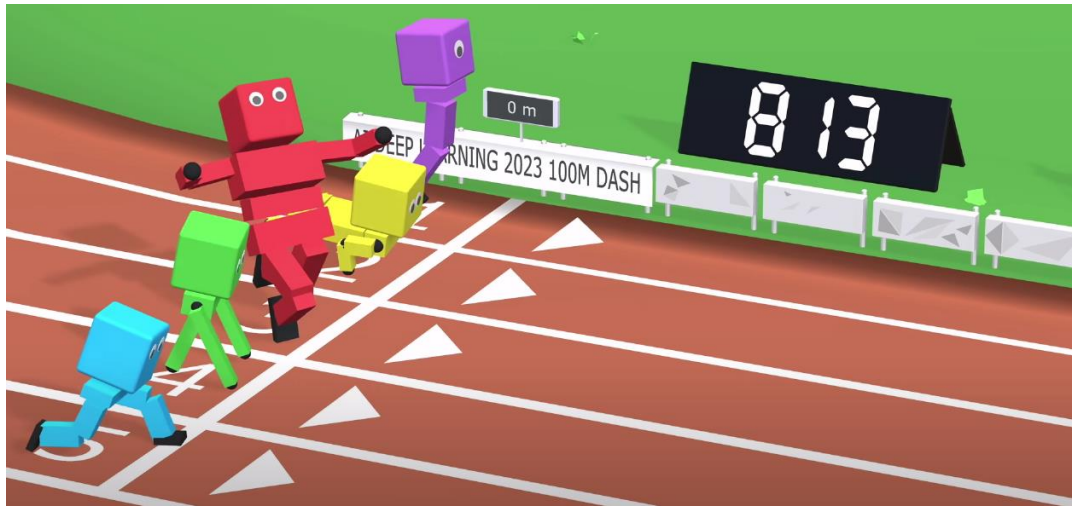




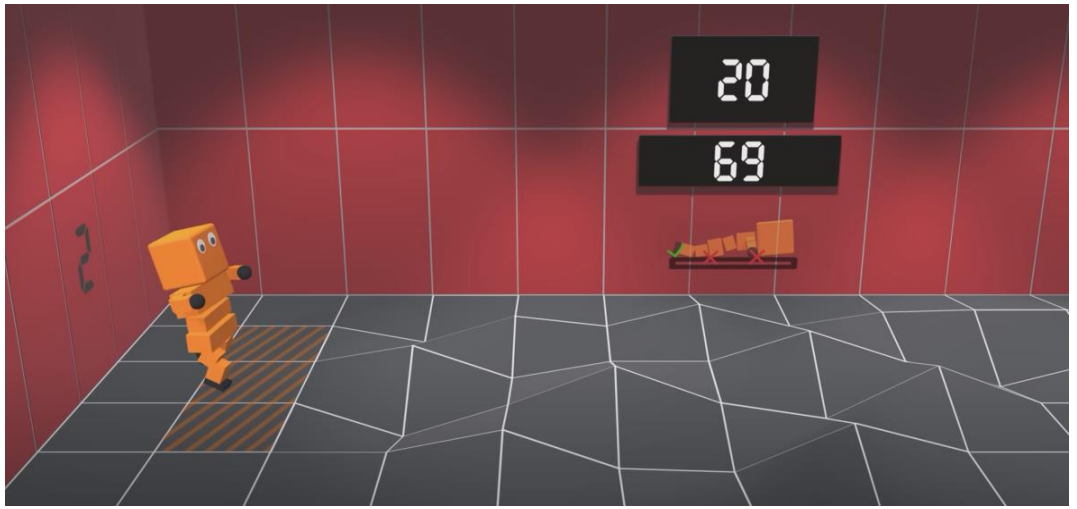
REINFORCEMENT LEARNING

Deep Reinforcement Learning

Reinforcement Learning



AI Olympics – 100m Race: https://www.youtube.com/watch?v=pJPdW8WWAso&ab_channel=AIWarehouse



AI Learns to Walk (deep reinforcement learning): https://www.youtube.com/watch?v=L_4BPjLBF4E&ab_channel=AIWarehouse

Reinforcement Learning

Reinforcement Learning focuses on teaching agents through trial and error

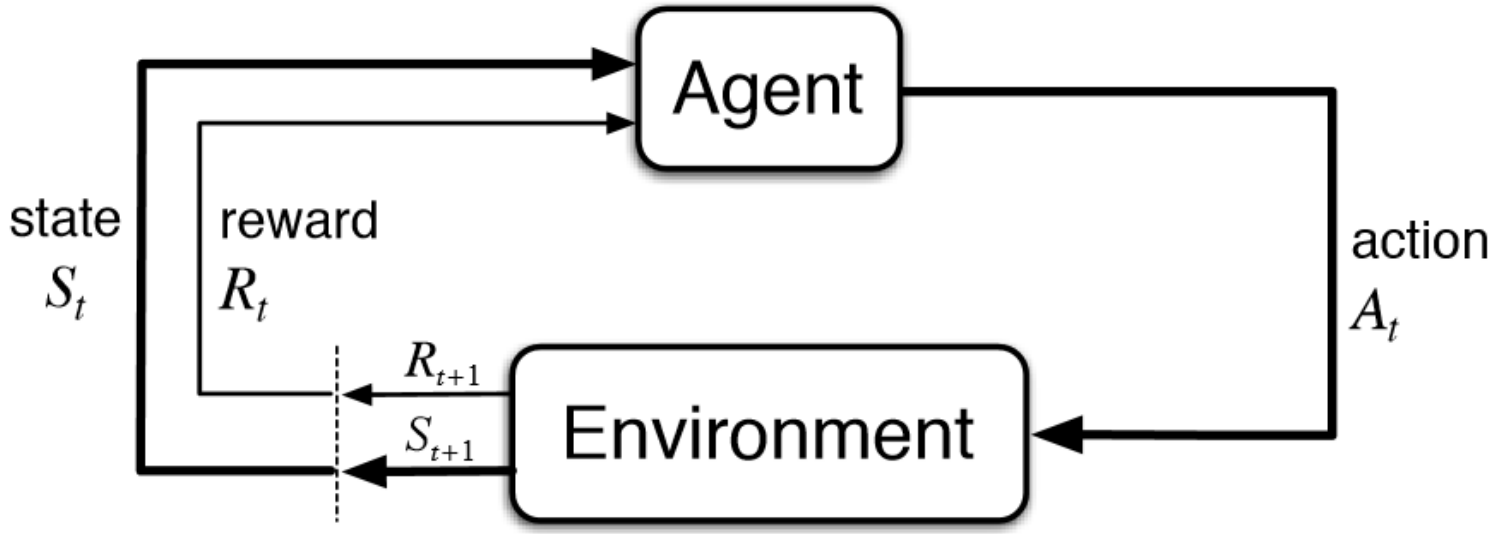


Fig. A simple illustration of the Reinforcement Learning Approach

Reinforcement Learning: Basic Concepts

There are four fundamental concepts that underpin most Reinforcement Learning (RL) projects. They are illustrated as follows:

- **Agent:** The actor operating within the environment, it is usually governed by a policy (a rule that decides what action to take).
- **Environment:** The world in which the agent can operate in.
- **Action:** The agent can do something withing the environment known as an action
- **Reward and observations:** In return, the agent receives a reward and a view of what the environment looks like after acting on it.

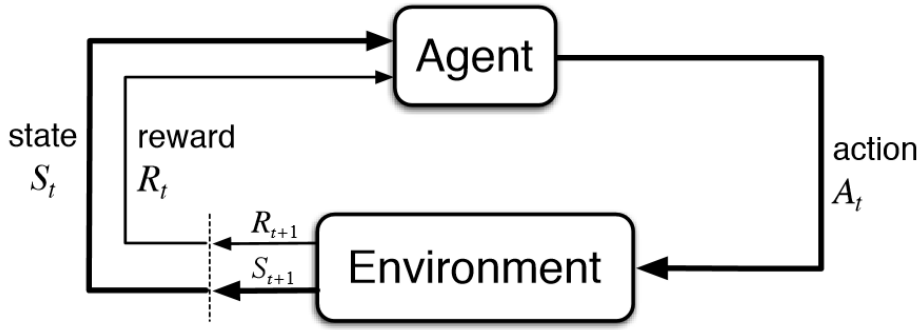
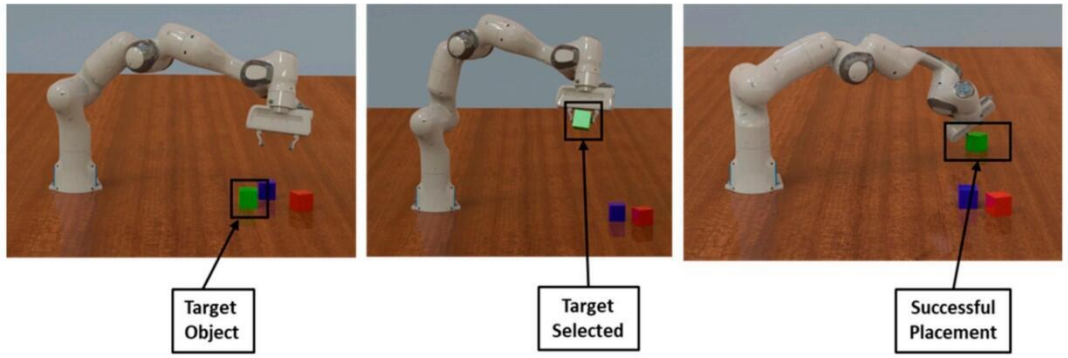


Fig. A simple illustration of the Reinforcement Learning Approach

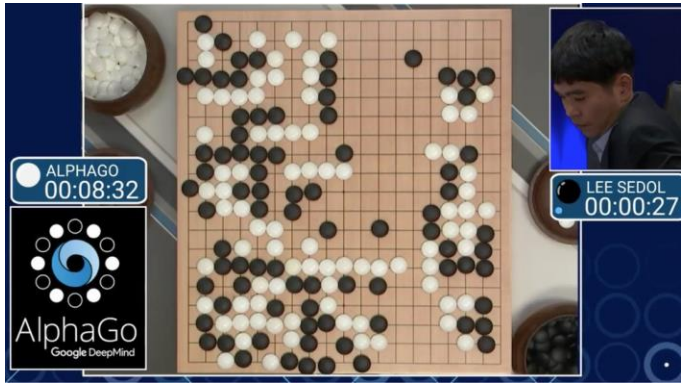
Reinforcement Learning: Applications

- **Game Playing**
  - **AlphaGo** and **AlphaZero**: DeepMind's AlphaGo program demonstrated groundbreaking performance in the ancient game of Go.
  - **OpenAI Five:** OpenAI's team of agents learned to play the complex game of Dota 2 at a professional level.
- **Robotics**
  - **Robotic Arm Control:** RL is used to train robotic arms to perform tasks like picking and placing objects, assembling components, and more.
  - **Autonomous Vehicles:** RL helps in training self-driving cars to navigate complex environments and make decisions on the road.
- **Healthcare**
  - **Drug Discovery:** RL is used to discover and optimize drug compounds by simulating chemical interactions.
  - **Personalized Treatment Plans:** RL models can recommend personalized treatment plans for patients based on their specific medical history and conditions.

Reinforcement Learning: Applications



Pick and Place: Reinforcement Learning for Robotic Control



Go Game: AlphaGo vs. Lee Sedol

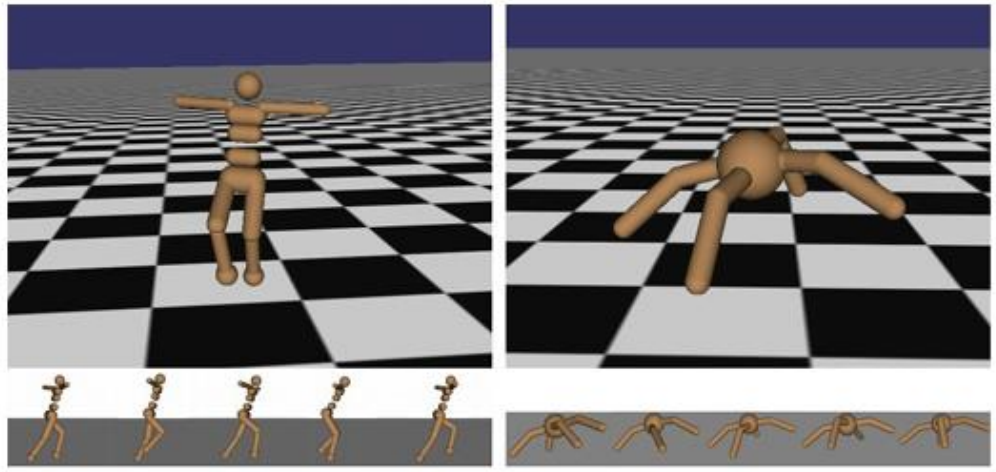
OpenAI Five Beats World Champion DOTA2 Team

Reinforcement Learning: Applications

- **Supply chain and logistics**
  - **Inventory Management:** RL can help optimize inventory levels to balance supply and demand, reducing carrying costs while avoiding stockouts.
  - **Routing and Scheduling:** RL can optimize routes for delivery vehicles or schedule tasks in a warehouse.
- **Game design and testing**
  - **Game AI:** RL is used to develop intelligent, adaptive non-player characters (NPCs) in video games.
  - **Automated Game Testing:** RL agents can be used to automatically test and evaluate different aspects of game performance.

Reinforcement Learning: Applications
<Robot Locomotion>

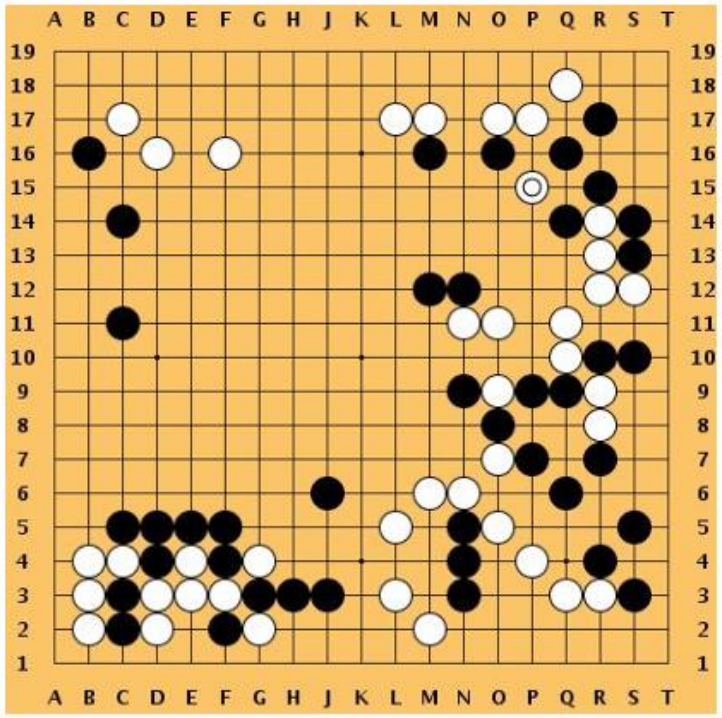
- **Objective:** Make the robot move forward
- **State:** Angle and position of the joints
- **Action:** Torques applied on joints
- **Reward:** 1 at each time step upright + forward movement



Reinforcement Learning: Applications
<Atari Games>

- **Objective:** Complete the game with the highest score
- **State:** Raw pixel inputs of the game state
- **Action:** Game controls e.g. Left, Right, Up, Down
- **Reward:** Score increase/decrease at each time step

Reinforcement Learning: Applications
<GO GAME>



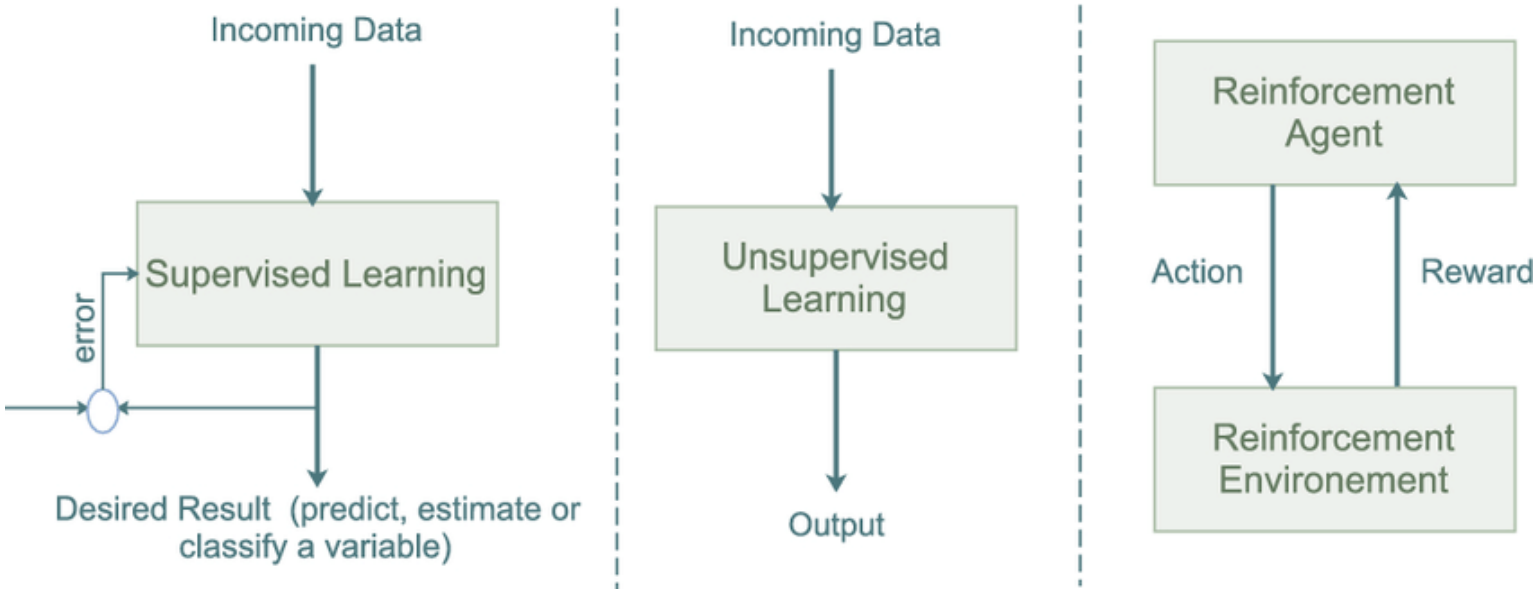
Objective: Win the game!
State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

Reinforcement Learning: Limitations and Considerations

Reinforcement Learning (RL) comes with several limitations and challenges that can make it a complex paradigm to apply in certain situations. Here are some of the key limitations:

- **Sample Inefficiency:** RL algorithms often require a large number of interactions with the environment to learn effective policies. This can be impractical or costly in real-world scenarios.
- **Exploration vs. Exploitation Trade-off:** Finding the right balance between exploring new actions and exploiting known good actions is a challenging problem, especially in environments with sparse rewards.
- **High-Dimensional State and Action Spaces:** Many real-world problems have high-dimensional state and action spaces, which can make learning and generalization difficult for traditional RL algorithms.
- **Multi-Agent Interactions:** In environments with multiple agents, the behavior of one agent may impact the learning of others, leading to complex and sometimes adversarial interactions.

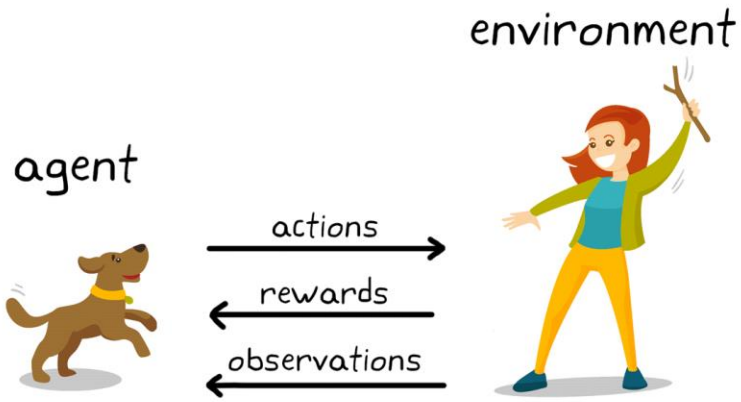
Reinforcement Learning: Comparison with Supervised and Unsupervised Learning



Reinforcement Learning: Main Steps

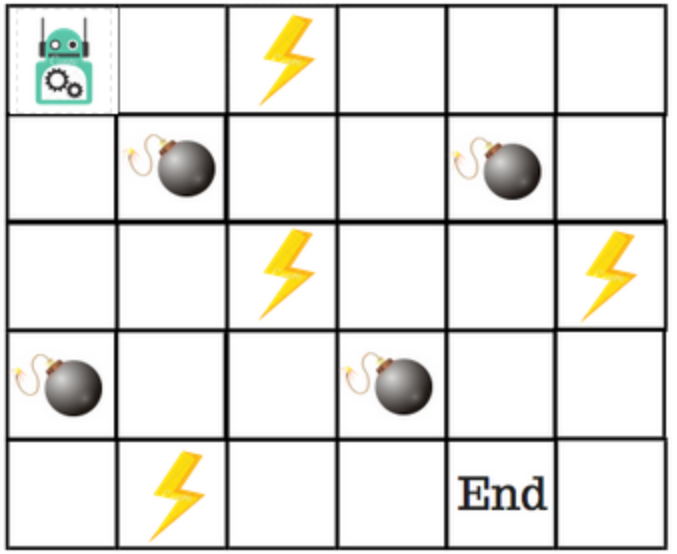
We can break down reinforcement learning into five simple steps:

1. The agent is at state zero in an environment.
2. It will take an action based on a specific strategy.
3. It will receive a reward or punishment based on that action.
4. By learning from previous moves and optimizing the strategy.
5. The process will repeat until an optimal strategy is found.



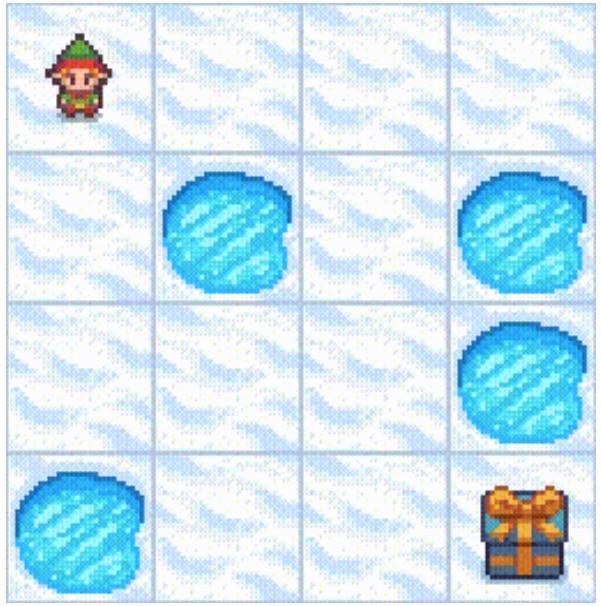
Reinforcement Learning: Q-Learning

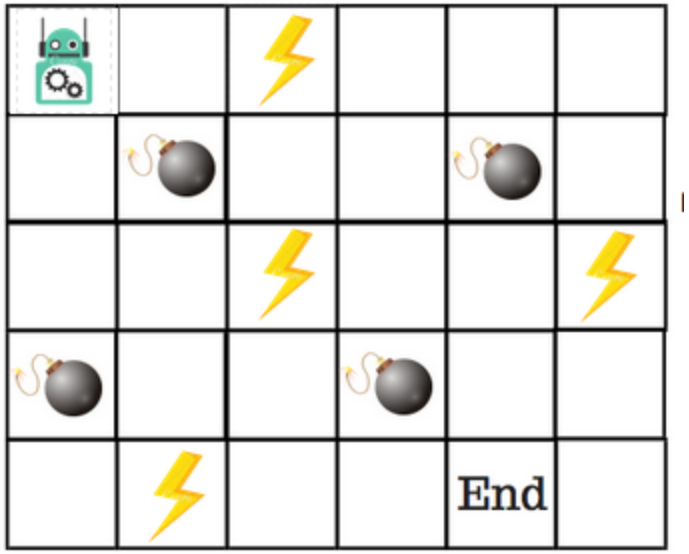
Definition: Q-Learning is a model-free reinforcement learning algorithm that allows an agent to learn a policy, or make decisions, without having a model of the environment. It's suitable for situations where the dynamics of the environment are not known in advance. The “Q” stands for quality. Quality represents how valuable the action is in maximizing future rewards.



Reinforcement Learning: How does Q-Learning Algorithm work?

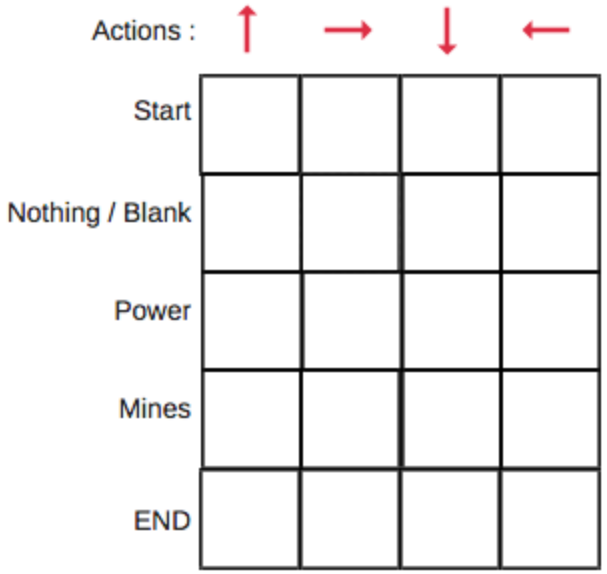
We will learn in detail how Q-learning works by using the example of a frozen lake. In this environment, the agent must cross the frozen lake from the start to the goal, without falling into the holes. The best strategy is to reach goals by taking the shortest path.

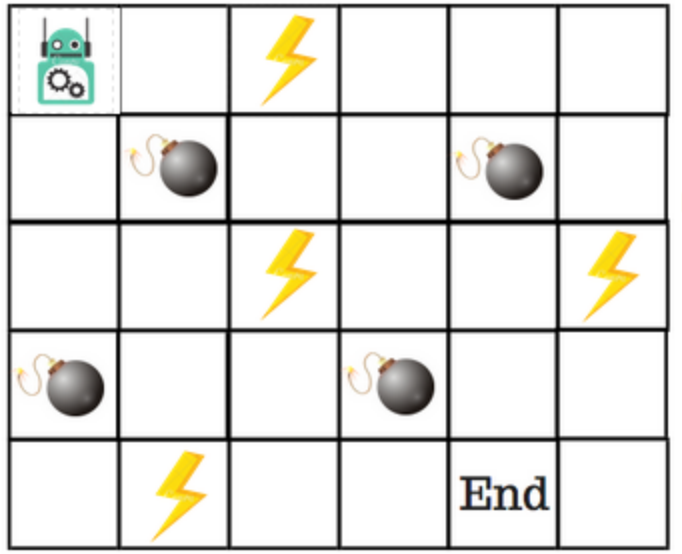




Q-Learning Algorithm: Q-Table

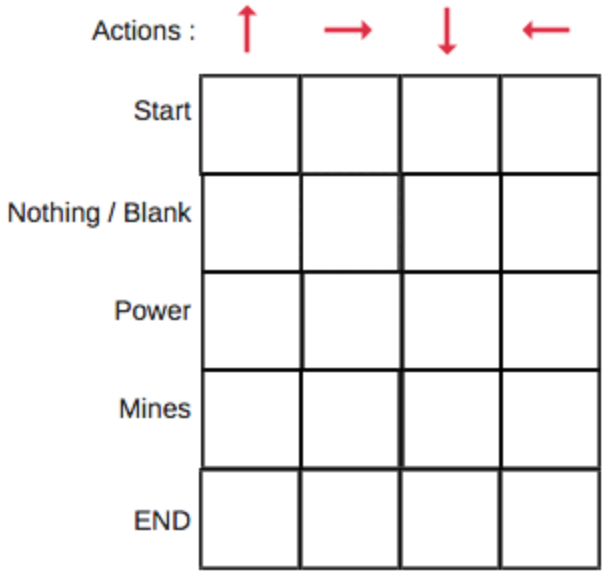
Q-Table is just a fancy name for a simple lookup table where we calculate the maximum expected future rewards for action at each state. Basically, this table will guide us to the best action at each state. Q-table is a data structure of sets of actions and states, and we use the Q-learning algorithm to update the values in the table. (In the Q-Table, the columns are the actions and the rows are the states.)

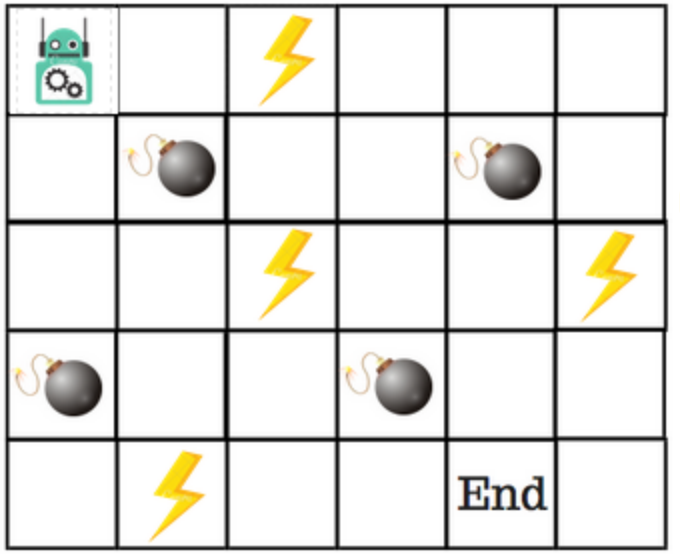




Q-Learning Algorithm: Q-Table

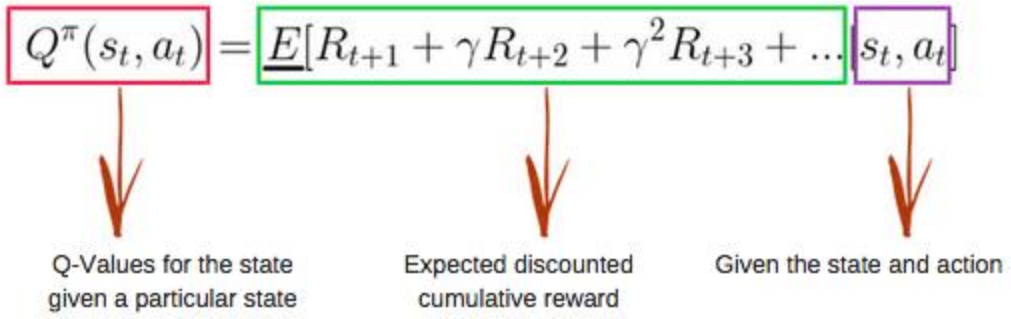
Each Q-table score will be the maximum expected future reward that the robot will get if it takes that action at that state. This is an iterative process, as we need to improve the Q-Table at each iteration.





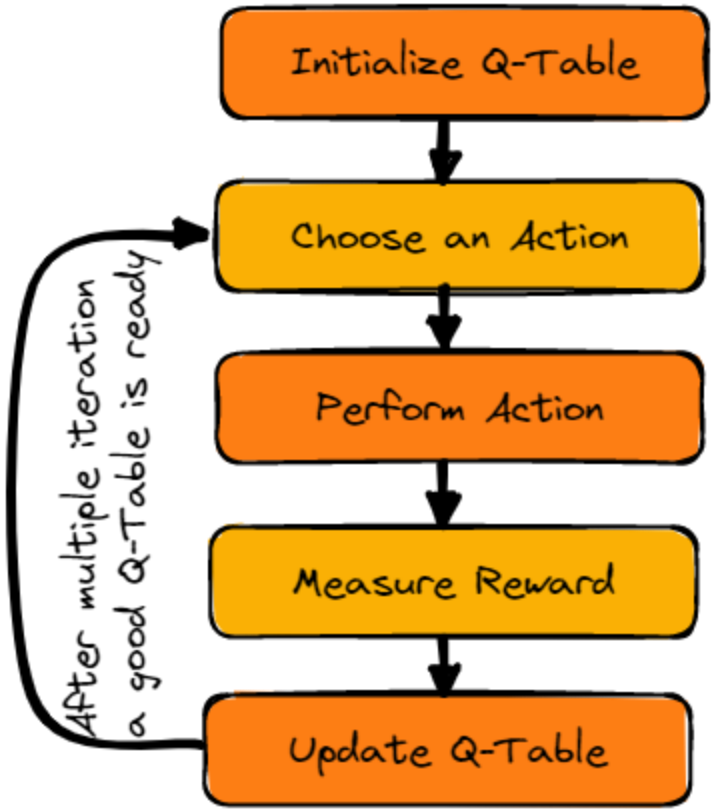
Q-Learning Algorithm: Q-Function

The Q-function uses the Bellman equation and takes state(s) and action(a) as input. The equation simplifies the state values and state-action value calculation.

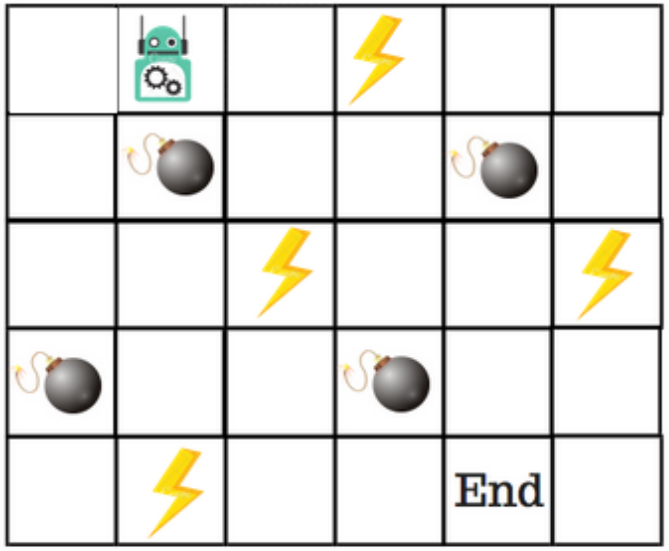


Using the above function, we get the values of Q for the cells in the table.
When we start, all the values in the Q-table are zeros.

Q-Learning Algorithm: Q-Learning Algorithm



Reinforcement Learning: DEMO



Actions :	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

Demo: [Session 6 - Reinforcement Learning]

REFERENCES

kaggle

Medium

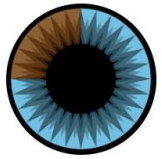
YouTube

coursera

Stanford University

Massachusetts Institute of Technology

datacamp



3Blue1Brown
@3blue1brown 5.49M subscribers 135 videos

أكاديمية الفهرية Al-Fihriya Academy
@al-fihriyaacademy6448 · 25.1K subscribers · 81 videos

StatQuest with Josh Starmer
@statquest 1.01M subscribers 264 videos

Your Feedback Matters!

https://forms.gle/T3co5xZK6pWTBCm67

THANK YOU

Idriss JAIRI

Email: Idriss.jairi@univ-lille.fr

LinkedIn: linkedin.com/in/jairiidriss/