Training Manual on

Application of Machine Learning Technique in Agriculture

Compiled and Edited by

Hasan Md. Hamidur Rahman

Course Director, Computer and GIS Unit

Hasan Mahmud

Senior System Analyst, Computer and GIS Unit

Al-Helal

Programmer, Computer and GIS Unit

Computer & GIS Unit
Bangladesh Agricultural Research Council

Published by

Computer and GIS Unit, BARC, Farmgate, Dhaka 1215, Bangladesh.

Date of Publication

April 2026

Contributors:

1. Mr. Hasan Md. Hamidur Rahman, Director, Computer and GIS Unit, BARC
2. Dr. Md. Zulfiker Mahmud, Professor, Department of CSE, Jagannath University
3. Dr. Md Golam Mahboob, Chief Scientific Officer, Forestry Unit, BARC
4. Dr. Foyez Ahmed Prodhan, Associate Professor, Gazipur Agricultural University (GAU) Gazipur, Bangladesh
5. Mr. Hasan Mahmud, Senior System Analyst, Computer and GIS Unit, BARC
6. Mrs. Farzana Akter, Assistant Professor, Department of IoT and Robotics Engineering(IRE), University of Frontier Technology, Bangladesh
7. Mr. Al-Helal, Programmer (Com & GIS), BARC
8. Mr. Aditya Rajbongshi, Assistant Professor, University of Frontier Technology, Bangladesh

Designed by

Mohammad Nazmul Islam, Graphics Designer, BARC

Computer & GIS Unit
Bangladesh Agricultural Research Council

Table of Content

Machine Learning: Concepts, Techniques, and Applications

Farzana Akter
Assistant Professor, Department of IoT and Robotics engineering
University of Frontier Technology, Bangladesh

Why Machine Learning in Agriculture?

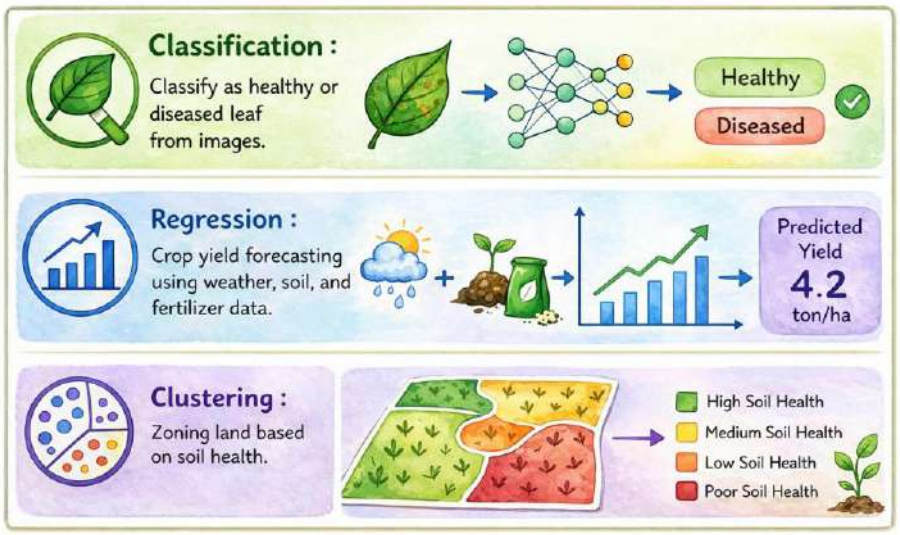
- Challenges:
  - Feeding a rapidly growing population
  - Limited agricultural land
  - Unpredictable climate and weather patterns
  - Declining labor availability
- Agriculture is shifting from experience-based practices to **data-driven decision-making**
- Role of ML: **prediction**, **automation**, **optimization**

Why Machine Learning in Agriculture?

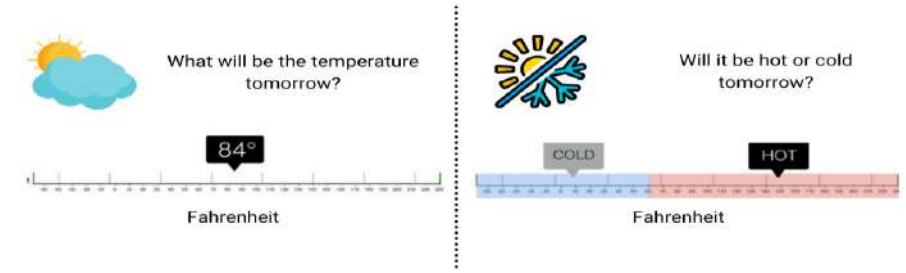
- **Precision agriculture**: Targeted use of water, fertilizer, pesticides using IoT, satellites, and sensors
- **Data-driven decisions**: Analyze historical & real-time data for prediction (e.g. disease outbreaks)
- **Automation**: AI & robotics for labor-intensive tasks like harvesting, weeding, sorting
- **Sustainability**: Minimize chemical usage, water consumption, and pollution.

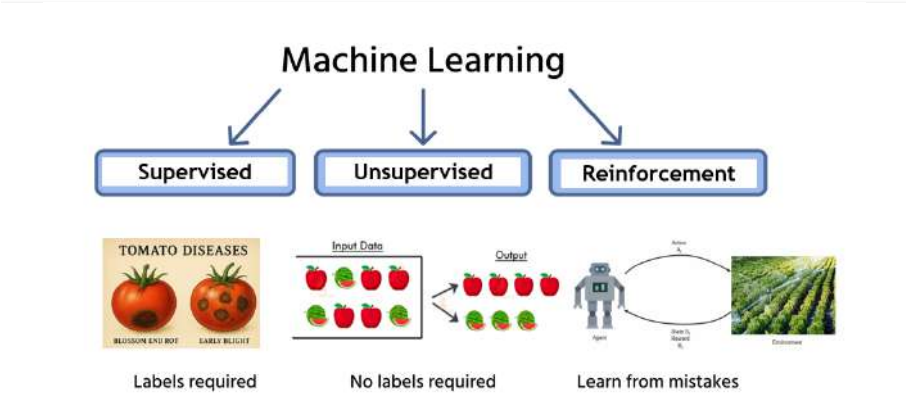
Core Concept of Machine Learning

- **Definition**: Learning patterns from data to make decisions.
- **Basic workflow**:
  - ***Data → Model → Prediction → Decision***
- **Decisions(Tasks)**
  - **Classification** : Disease identification from leaf images.
  - **Regression** : Crop yield forecasting using weather, soil, and fertilizer data.
  - **Clustering** : Zoning land based on soil health.

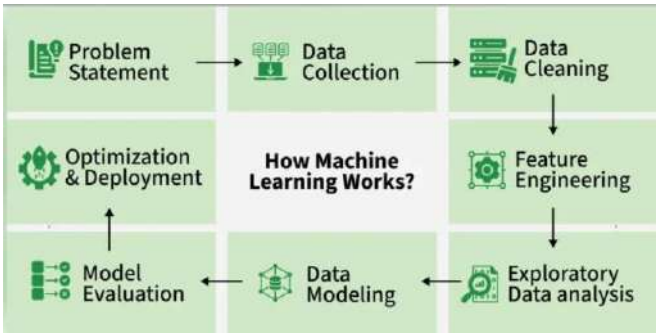


Classification or Regression?





Core Concept of Machine Learning



Data

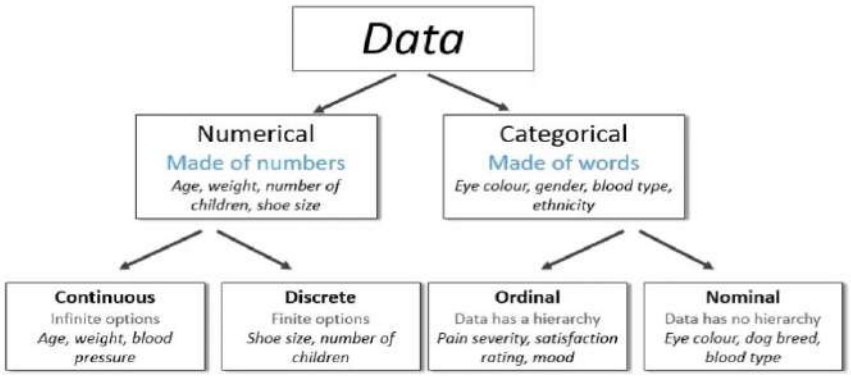
Agricultural datasets are:

- Multi-source (heterogeneous)
- Multi-scale (field → regional → global)
- Multi-modal (numeric, image, text, time-series)

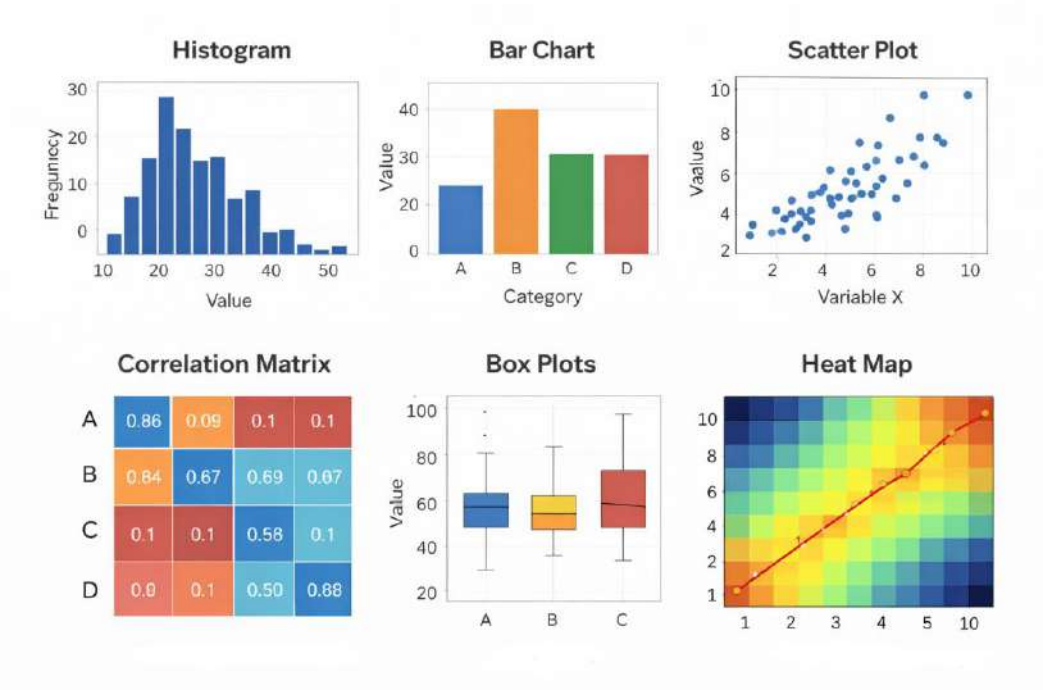
- Data can be broadly classified into three categories based on its structure: Structured, Semi-Structured, and Unstructured.

Structured

Dear Aysha,
I am sending this email to illustrate an example of semi structured data.
Regards,
Farzana



Exploratory Data Analysis (EDA)



Data Preprocessing

- **Exploring Your Data**
  - Understanding dataset structure
  - Data types (numerical, categorical, time-series, image)
  - Data visualization basics
  - Identifying patterns and anomalies
- **B. Data Preparation**
  - Data cleaning
  - Handling missing values
  - Normalization and scaling
  - Encoding categorical variables

Why Is Data Preprocessing Important?

- **Data Quality**
  - Ensuring the accuracy, consistency, and completeness of data.
  - Identifying duplicate records, inconsistent values, and data entry errors.
  - No quality data, no quality mining results!
    - Quality decisions must be based on quality data
    - Example: In customer records, duplicate email addresses are removed to avoid redundant marketing campaigns.
- Data preparation, cleaning, and transformation comprises the majority of the work in a data mining application (90%).

Data Cleaning

Day	Temp	Rainfall	Soil_pH	Crop	Fertilizer	Yield (Kg)
1	48	10	6.8	Rice	Urea	500
2	35	NULL	6.0	Rice	Urea	520
3	NULL	12	NULL	Wheat	NULL	510
4	33	15	6.5		Urea	-100
5	40	14	6.5	Rice	Urea	530

Missing Value Handling

- Techniques Used:
  - Remove Rows/Columns
    → If too many missing values
  - Mean / Median (Numerical)
    → Temperature, Rainfall
  - Mode (Categorical)
    → Crop Type, Fertilizer
  - Forward / Backward Fill *(Time-series)*
    → Use previous/next day data

Think of scaling like adjusting all musical instruments to the same pitch before a concert, if one plays off-key, the whole melody sounds wrong.

Normalization & Scaling

- Without scaling:
  - The model may assign far more importance to Yield (kg) because its numbers are larger.
  - Gradient-based algorithms like linear regression, logistic regression, and neural networks will struggle to converge efficiently.
- Feature scaling means adjusting the range of feature values so that they're on a similar scale.
- There are two main types:
  - Normalization (Min-Max Scaling): Rescales data to a specific range (commonly [0,1]).
  - Standardization (Z-score scaling): Centers data around the mean with unit variance.
- Scenario: Different feature ranges (common in agricultural datasets)

Feature	Range
Rainfall	0–300 mm
Temperature	20–40 °C
Soil Moisture	0–1

$$X' = \frac{X - Xmin}{Xmax - Xmin}$$

After Normalization (0–1):

Crop	Yield (kg)	Rainfall (mm)	Temperature (°C)
Rice	300	100	30
Maize	500	200	25
Wheat	700	300	35

Crop	Yield	Rainfall	Temperature
Rice	0.00	0.00	0.50
Maize	0.50	0.50	0.00
Wheat	1.00	1.00	1.00

Normalization & Scaling

Min-Max Scaling (Normalization)

- Min-Max scaling transforms each feature into a range between 0 and 1.

When to Use

- When you know your data has a fixed upper and lower bound.
- Works best when your data follows a uniform distribution.
- Common for neural networks because they perform better when inputs are small and bounded.

When Not to Use

- Not ideal if your data contains outliers, they can compress the range of most data points.

StandardScaler (Z-score Normalization)

It transforms data so that:

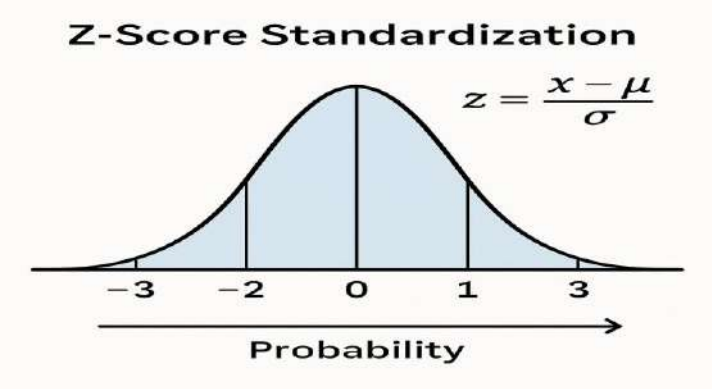
- Mean = 0
- Standard deviation = 1

In simple terms, it measures how far a value is from the mean in units of standard deviation.
When to Use

- For distance-based or gradient-based models
  e.g., SVM, KNN, Logistic Regression — prevents features with larger ranges from dominating
- When data is approximately normally distributed
  Z-score works best when data follows a Gaussian-like distribution

When NOT to Use

- When data has strong outliers or is highly skewed
  Mean and standard deviation get distorted
- For tree-based models
  e.g., Decision Tree, Random Forest — scaling is usually unnecessary



- **What Changes?**
  - Before (MinMaxScaler) → values between 0 and 1
  - Now (StandardScaler) → values centered around 0 (can be negative or positive)
- Mean of each column → 0
- Values:
  - Below mean → negative
  - Equal to mean → 0
  - Above mean → positive

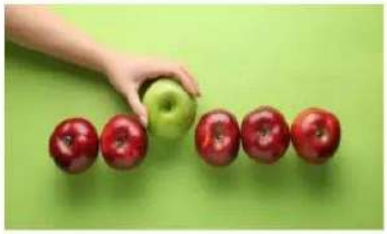


Fig 1

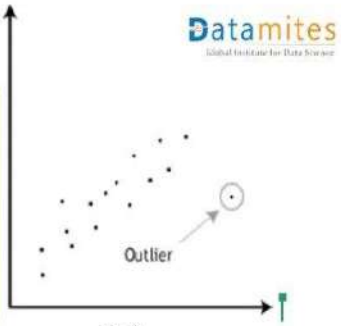


Fig 2

Types of Agricultural Data for Feature Extraction

- 🌦 Weather data (rainfall, temperature, humidity)
- 🌱 Soil data (pH, moisture, nitrogen content)
- 🛰 Remote sensing data (satellite/drone images)
- 📡 IoT sensor data (real-time field monitoring)
- 🌾 Crop yield & farming records

Common Feature Extraction Techniques

Statistical Feature Extraction

- Used for sensor and tabular data:
  - Mean, median, variance
  - Standard deviation, Min/max Scaling

Image-Based Feature Extraction (Crop & Disease Detection)

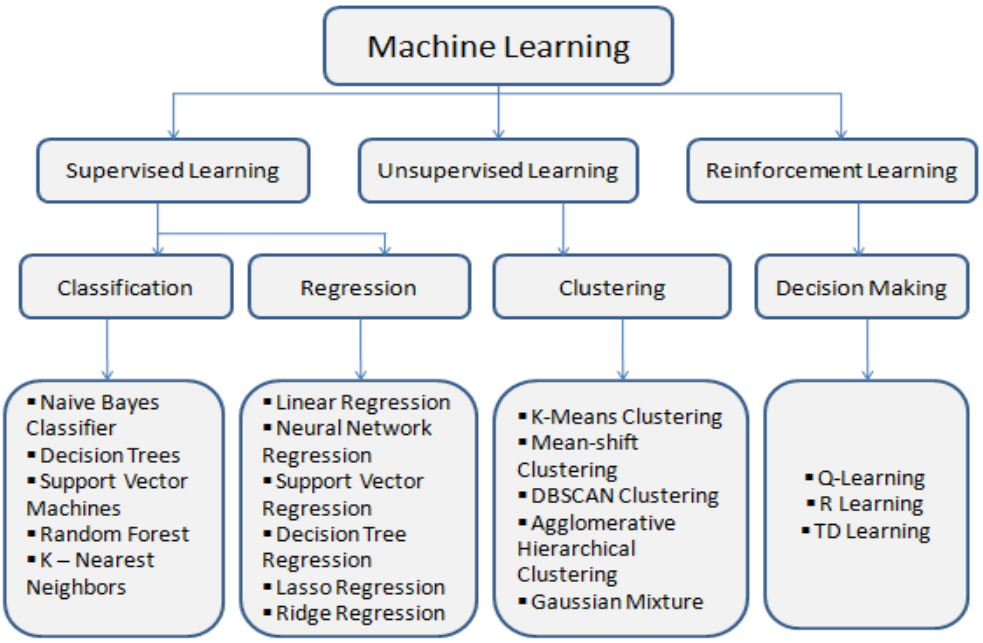
- Used in plant disease and remote sensing:
  - Texture features, Color features, Shape features (leaf area, lesion size)

Text-Based Feature Extraction

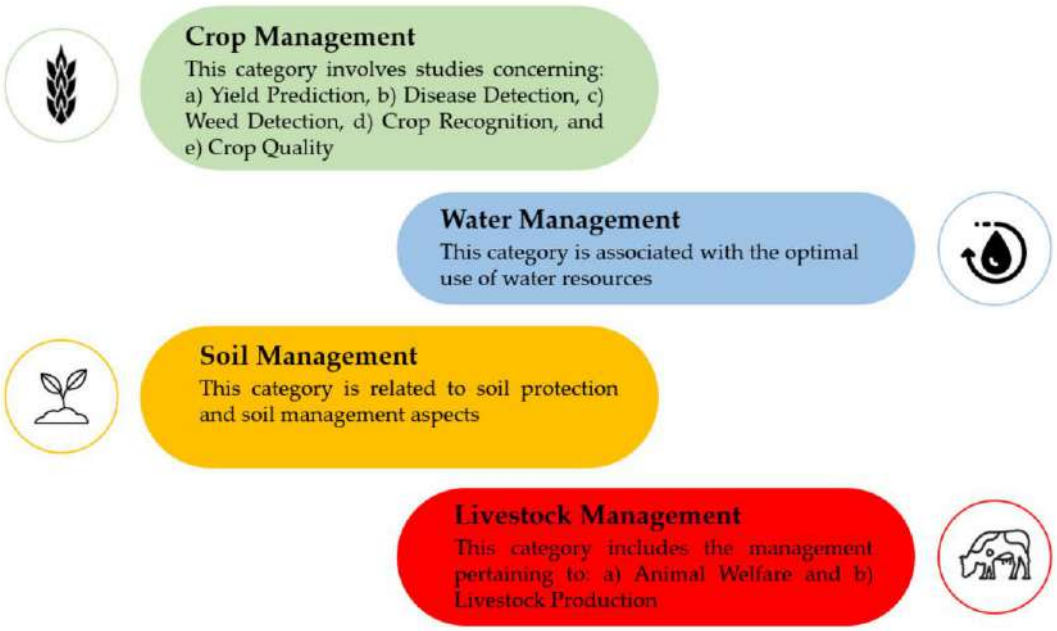
- Word Embeddings

Dimensionality Reduction Techniques

- Used for large-scale datasets:
  - Principal Component Analysis (PCA)



Evaluation Metrics		
Regression	**Classification**	**Clustering**
R^2	Accuracy	Silhouette Score
Adjusted R^2	Precision	Davies-Bouldin Index
MAE	Recall	Calinski-Harabasz Index
MSE	F1-Score	
RMSE	ROC-AUC	



Challenges and Research Scope

- **Challenges:**
  - Limited availability of high-quality agricultural data
  - Poor internet connectivity in rural areas
  - High cost of IoT devices and sensors
  - Lack of technical awareness among farmers
- **Future Scope:**
  - AI-powered drones for monitoring
  - Autonomous farming machinery
  - Smart greenhouse systems
  - Climate-resilient agriculture
  - Integration of robotics and IoT

Recent trends in AI in Agriculture

- Generative AI & Chatbots
- Precision Agriculture
- Autonomous Robots & Tractors
- Drones & Remote Sensing
- Advanced Weather Forecasting
- Disease Identification Apps
- Smart Soil Analysis

Basic Python Tutorial

Mr. Hasan Mahmud
Senior System Analyst (Com & GIS), BARC
Email: hasan.mahmud@barc.gov.bd

Python is a high-level, interpreted, general-purpose programming language known for its simplicity, readability, and versatility. It was created by Guido van Rossum and first released in 1991. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Python is often described as a programming language that is:

- Beginner-friendly
- Open-source and free
- Readable and clean in syntax
- Extremely versatile

Python Basics

A variable is a name that refers to a value stored in memory. Think of it as a labeled box that holds information you want to use later.

A data type in Python (or any programming language) defines what kind of value a variable can hold and what operations can be performed on that value.

Variable Assignment In Python, variables are created the moment you assign a value using the = (assignment) operator.

Data types and Assignments

Data Type	Description	Example
int	Integer numbers	`x = 10`
float	Floating-point (decimal) numbers	`pi = 3.14`
bool	Boolean values	`flag = True`
str	Text (string)	`name = "Alice"`
list	Ordered, mutable collection	`nums = [1, 2, 3]`
tuple	Ordered, immutable collection	`coords = (10.0, 20.0)`
dictionary	Unordered key-value collection	`{"wheat": 30, "rice": 40, "maize": 35}`
range	Sequence of numbers	`r = range(5)`

Code (Operators)

```
# Operators
a = 10
b = 3
print("Initial Value: a = ", a)
print("Initial Value: b =", b)
# Arithmetic Operators
print("Arithmetic Operators:")
print("a + b =", a + b)
print("a - b =", a - b)
print("a * b =", a * b)
print("a / b =", a / b)
print("a // b =", a // b)
print("a % b =", a % b)
print("a ** b =", a ** b)

# Comparison Operators
print("\nComparison Operators:")
print("a == b:", a == b)
print("a != b:", a != b)
print("a < b:", a < b)
print("a > b:", a > b)
print("a <= b:", a <= b)
print("a >= b:", a >= b)

# Assignment Operators
print("\nAssignment Operators:")
x = 5
print("\nInitial Value: x = ", x)
x += 2
print("x += 2:", x)
x -= 1
```

```
print("x -= 1:", x)
x *= 3
print("x *= 3:", x)
x /= 2
print("x /= 2:", x)
x %= 4
print("x %= 4:", x)

# Logical Operators
print("\nLogical Operators:")
x = True
y = False
print(f"\nInitial Value: x = {x} and y = {y}")
print("x and y:", x and y)
print("x or y:", x or y)
print("not x:", not x)
```

Code Output (Operators)

```
Initial Value: a =  10
Initial Value: b = 3
Arithmetic Operators:
a + b = 13
a - b = 7
a * b = 30
a / b = 3.3333333333333335
a // b = 3
a % b = 1
a ** b = 1000
```

Comparison Operators:

```
a == b: False
a != b: True
a < b: False
```

a > b: True

a <= b: False

a >= b: True

Assignment Operators:

Initial Value: x = 5

x += 2: 7

x -= 1: 6

x *= 3: 18

x /= 2: 9.0

x %= 4: 1.0

Logical Operators:

Initial Value: x = True and y = False

x and y: False

x or y: True

not x: False

Why Casting is Needed (Code)

```
# Cannot Add !!
x = "123"
y = x+1   #  Error (Quiz-Why?)
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-2-4504a196927f> in <cell line: 0>()
      1 # Cannot Add !!
      2 x = "123"
----> 3 y = x+1   #  Error (Quiz-Why?)

TypeError: can only concatenate str (not "int") to str
```

Casting (Code)

```
# Casting (2 possible scenarios of addition whichever we want)
x = "123"
```

```
y1 = int(x)+1
print(f"Type of y1: {type(y1)}, value of y1: {y1}")
y2 = x+"1"
print(f"Type of y2: {type(y2)}, value of y2: {y2}")
```

Casting (Output)

```
Type of y1: <class 'int'>, value of y1: 124
Type of y2: <class 'str'>, value of y2: 1231
```

Rules for Naming Variables

1. Must start with a letter (a–z, A–Z) or an underscore (_).
2. Can contain letters, numbers, and underscores.
3. Case-sensitive (Name and name are different).
4. Cannot use Python keywords (like if, while, class, etc.).

Keywords are reserved words that have special meaning. You cannot use them as variable names, function names, or identifiers. They are used to define the syntax and structure of Python code.

Loop

A **loop** is a logical structure in programming that allows a set of instructions to be executed **repeatedly** until a specific condition is met or no longer valid. It automates repetitive tasks without the need to write the same instructions multiple times.

Importance of Loops

- **Efficiency**
  Loops let you perform repetitive actions without rewriting code, saving time and reducing clutter.

- **Automation**
  In real-world tasks like processing multiple files, calculating statistics, or managing sensors in agriculture, loops help automate work that involves repetition.

- **Flexibility**
  You can easily adjust how many times an action happens, based on data or changing conditions.

- **Scalability**
  Loops allow your program to handle larger datasets or longer tasks without extra effort — it works the same whether there are 5 or 5,000 items.

- **Error Reduction**
  Repeating actions through loops helps reduce the chance of human error compared to copying and pasting code multiple times.
- **Logical Flow**
  Loops support logical operations like checking conditions or performing tasks at regular intervals (e.g., monitoring soil moisture every hour).

for Loop

A for loop is used when you know in advance how many times you want to repeat a task. It is typically used to iterate over a sequence like a list, range of numbers, or items in a collection.

Example:

A farmer wants to inspect 10 plants in a row — one by one. Since the number is fixed (10), a for loop is the most suitable.

while Loop

A while loop is used when the number of repetitions is not known beforehand. The loop continues as long as a certain condition remains true.

Example:

A sensor monitors soil moisture and continues irrigation until the moisture level is sufficient. Since the number of times it runs is not fixed, a while loop fits better

Code (for Loop – Simple)

```
farms = [
    ("Farm A", "wheat", 35),
    ("Farm B", "rice", 15),
    ("Farm C", "maize", 45),
    ("Farm D", "wheat", 25),
]
print("Irrigation Decision System\n")
print(f"Type of farms: {type(farms)}")
print(farms)
for farm in farms:
  print(f"\nType of farm: {type(farm)}")
  print(farm)
```

Code Output (for Loop – Simple)

Irrigation Decision System

Type of farms: <class 'list'>
[('Farm A', 'wheat', 35), ('Farm B', 'rice', 15), ('Farm C', 'maize', 45), ('Farm D', 'wheat', 25)]

Type of farm: <class 'tuple'>
('Farm A', 'wheat', 35)

Type of farm: <class 'tuple'>
('Farm B', 'rice', 15)

Type of farm: <class 'tuple'>
('Farm C', 'maize', 45)

Type of farm: <class 'tuple'>
('Farm D', 'wheat', 25)

Code (for Loop – Advanced)

```
farms = [
    ("Farm A", "wheat", 35),
    ("Farm B", "rice", 15),
    ("Farm C", "maize", 45),
#   ("Farm D", "rice", -1),  # Error in Data. Check the output by uncommenting this line
    ("Farm E", "wheat", 25),
]

print("Irrigation Decision System\n")

# Loop through farm data
for farm in farms:
    name, crop, moisture = farm
```

```
    # Check for faulty moisture value
    if moisture < 0:
        print(f"Error in data for {name}. Stopping analysis.")
        break  # Stop loop if error in data found

    # Skip farms growing crops not monitored
    if crop not in ["wheat", "rice"]:
        print(f"Skipping {name}: crop '{crop}' not monitored.")
        continue  # Skip this farm

    # Provide irrigation advice
    print(f"\nAnalyzing {name} ({crop}):")
    if moisture < 20:
        print("  ➤ Soil too dry. Irrigation needed.")
    elif moisture <= 40:
        print("  ➤ Moderate moisture. Monitor closely.")
    else:
        print("  ➤ Sufficient moisture. No action needed.")
print("\n Farm loop completed.\n")
```

Code Output (for Loop – Advanced)

```
Irrigation Decision System

Analyzing Farm A (wheat):
  ➤ Moderate moisture. Monitor closely.

Analyzing Farm B (rice):
  ➤ Soil too dry. Irrigation needed.
Skipping Farm C: crop 'maize' not monitored.

Analyzing Farm E (wheat):
  ➤ Moderate moisture. Monitor closely.
```

To use it in our code, we must first import it.

Concept	Description	Example
Library	Collection of tools	pandas
Package	Organized directory of modules	pandas.io, matplotlib.pyplot
Module	A single .py file	math, datetime

Code (Libraries)

```
!pip install geopandas

import geopandas as gpd
```

Code Output (Libraries)

```
Collecting geopandas
  Downloading geopandas-1.0.1-py3-none-any.whl.metadata (2.2 kB)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.0.2)
```

Code (File Writing)

```
total_yield_by_district.to_csv('/content/drive/MyDrive/output.csv', index=False)
with open('/content/drive/MyDrive/output.csv', 'a') as f:
    f.write('\n\n')

average_yield_by_crop.to_csv('/content/drive/MyDrive/output.csv', mode='a', index=False)
```

Object Oriented Programming (OOP)

1. Class Definition

A class is a blueprint for creating objects.

It defines attributes (variables) and methods (functions) that the objects created from the class will have.

◈ **init** is a constructor method that runs when an object is created.

◈ self refers to the instance of the class.

2. Object Creation

An object is an instance of a class. It contains state (attributes) and behavior (methods).

3. Method Calling

Methods of the class are called using the object.

Code (class)

```
# Class definition

class Person:
    # Constructor method (called when object is created)
    def __init__(self, name, age):
        self.name = name  # instance variable
        self.age = age

    # Instance method
    def introduce(self):
        print(f"Hi, I'm {self.name} and I'm {self.age} years old.")
    # Another method
    def birthday(self):
        self.age += 1
        print(f"Happy Birthday, {self.name}! You are now {self.age}.")
```

4 Important Concepts of OOP

1. Encapsulation

Bundles data (attributes) and methods that operate on the data into one unit (a class).

Protects internal object state by using access modifiers (private, public, etc. in some languages).

2. Inheritance

A class (child) can inherit attributes and methods from another class (parent).

Promotes code reusability.

3. Polymorphism

Allows methods to have different behaviors based on the object that is calling them.

Two types: method overriding (in child class) and method overloading (same method name with different parameters, depending on the language).

4. Abstraction

Hides complex implementation details and shows only the essential features of the object.

Often achieved using abstract classes or interfaces (in languages like Java, C++).

Python for Data Analysis – Beginner's Guide with Detailed Explanations

Mr. Al-Helal
Programmer (Com. & GIS)
Bangladesh Agricultural Research Council
Email: al.helal@barc.gov.bd

Loops (`for`, `while`)

Loops are fundamental constructs in programming that enable the execution of a block of code repeatedly. They help improve code efficiency by eliminating the need for repetitive statements.

For Loop

A `for` loop is typically used when the number of iterations is known in advance. It iterates over a sequence such as a list, tuple, string, or range.

While Loop

A `while` loop executes a block of code as long as a specified condition remains true. It is generally used when the number of iterations is not predetermined and depends on dynamic conditions.

Common Use Cases

- Iterating through elements in a dataset or collection
- Repeating operations until a specific condition is met
- Automating repetitive tasks in data processing and manipulation

Example

```
# For loop example
for i in range(1, 6):
    print(f"{i} squared is {i ** 2}")
# While loop example
x = 1
while x <= 5:
    print("Counting:", x)
    x += 1
```

📘 Functions in Python

1. Introduction

Functions are one of the core building blocks of Python programming. A function is a **named, reusable block of code** that performs a specific task. Instead of repeating the same logic multiple times, functions allow developers to write the logic once and reuse it throughout the program.

Functions improve **modularity, readability, maintainability, and scalability** of code.

2. Syntax of a Function

```
def function_name(parameters):
    # function body
    return output
```

3. Types of Functions

3.1 Built-in Functions

These are predefined functions provided by Python.

Examples:

```
print()
len()
sum()
max()
min()
```

3.2 User-defined Functions

Functions created by the programmer to solve specific problems.

4. Function Components

4.1 Parameters

Inputs passed to a function.

4.2 Return Value

Output produced by a function using `return`.

4.3 Docstring

A description inside triple quotes explaining function behavior.

5. Examples with Explanation

5.1 Basic Function

```
def greet(name):
    return f"Hello, {name}!"
print(greet("Al-Helal"))
```

Explanation:

- Input: `name`
- Output: greeting string
- Purpose: Simple reusable greeting generator

5.2 Function with Multiple Parameters

```
def add_numbers(a, b):
    return a + b

result = add_numbers(10, 20)
print("Sum:", result)
```

5.3 Function with Default Parameter

```
def power(base, exponent=2):
    return base ** exponent
print(power(5))     # default exponent = 2
print(power(5, 3))  # custom exponent
```

5.4 Function Returning Multiple Values

```
def stats(data):
    return min(data), max(data), sum(data) / len(data)
minimum, maximum, average = stats([10, 20, 30, 40])
print(minimum, maximum, average)
```

5.5 Function with Conditional Logic

```
def check_even_odd(number):
    if number % 2 == 0:
        return "Even"
```

```
    else:
        return "Odd"
print(check_even_odd(7))
```

6. Advanced Critical Functions (Real-World Use)

6.1 Data Cleaning Function (Preprocessing)

```
def clean_text(text):
    return text.strip().lower()
```

6.2 Safe Division Function (Error Handling)

```
def safe_divide(a, b):
    if b == 0:
        return None
    return a / b
```

6.3 Dataset Summary Function (EDA)

```
def dataset_summary(data):
    return {
        "count": len(data),
        "min": min(data),
        "max": max(data),
        "mean": sum(data) / len(data)
    }
```

6.4 Train-Test Split Function (ML Concept)

```
def train_test_split(data, ratio=0.8):
    index = int(len(data) * ratio)
    return data[:index], data[index:]
```

6.5 Classification Threshold Function

```
def classify(probability, threshold=0.5):
    return 1 if probability >= threshold else 0
```

Data Structures in Python

Python provides several built-in data structures used to efficiently store, organize, and manipulate data. These structures form the foundation of programming and data processing.

1. List

A **list** is an ordered and mutable collection of elements. It allows duplicate values and supports dynamic modification.

Key Features:

- Ordered
- Mutable (modifiable)
- Allows duplicates

Example:

```
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")
print(fruits)
```

2. Dictionary

A **dictionary** stores data in key-value pairs, enabling fast and efficient lookup.

Key Features:

- Unordered (Python 3.6+ maintains insertion order)
- Mutable
- Keys must be unique

Example:

```
person = {"name": "Helal", "age": 35}

print(person["name"])
print(person.get("name"))
```

3. Tuple

A **tuple** is an ordered but immutable collection of elements.

Key Features:

- Ordered
- Immutable (cannot be changed)
- Allows duplicates

Example:

```
coordinates = (10, 20)
print(coordinates)
```

4. Set

A **set** is an unordered collection of unique elements.

Key Features:

- Unordered
- No duplicates allowed
- Mutable (but elements must be immutable)

Example:

```
unique_numbers = set([1, 2, 2, 3, 3])
print(unique_numbers)
```

Summary of Data Structures

Structure	Ordered	Mutable	Duplicates
List	Yes	Yes	Yes
Dictionary	Yes*	Yes	Keys No
Tuple	Yes	No	Yes
Set	No	Yes	No

📂 File Handling in Python

1. Writing to a File

Creates or overwrites a file with new content.

```
with open("data.txt", "w") as f:
    f.write("Python is fun.")
```

2. Reading from a File

Reads content from an existing file.

```
with open("data.txt", "r") as f:
    content = f.read()
    print(content)
```

📌 Use Cases

- Data logging
- Dataset storage
- Configuration files
- Data preprocessing in ML

🔢 NumPy (Numerical Python)

NumPy is a core library for numerical computation in Python. It provides high-performance support for arrays and mathematical operations.

Key Features:

- Efficient multi-dimensional arrays
- Mathematical and statistical functions
- Foundation for scientific computing libraries

Example:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print("Mean:", np.mean(arr))
print("Standard Deviation:", np.std(arr))
```

2D Array Example:

```
matrix = np.array([[1, 2], [3, 4]])
print(matrix)
```

Pandas (Data Analysis Library)

Pandas is a powerful library used for data manipulation and analysis, especially for tabular (Excel-like) data.

Key Features:

- DataFrame structure (rows & columns)
- Data filtering and grouping
- Data cleaning and transformation
- CSV/Excel file handling

Example:

```
import pandas as pd

df = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Cathy'],
    'Age': [25, 30, 22]
})

print(df.head())
```

Filtering Data:

```
print(df[df['Age'] > 23])
```

7. Data Visualization

Data visualization is the process of representing data in graphical form to identify patterns, trends, and relationships more effectively.

It is a crucial step in **exploratory data analysis (EDA)**.

Key Libraries

1. Matplotlib

- Core Python plotting library
- Provides full control over plots

2. Seaborn

- Built on top of Matplotlib
- Used for statistical and attractive visualizations

Common Plot Types

- Line chart → trends over time
- Histogram → distribution of data
- Scatter plot → relationship between variables

Example

```
import matplotlib.pyplot as plt

import seaborn as sns
```

Line Chart

```
plt.plot([1, 2, 3], [2, 4, 6])

plt.title("Simple Line Chart")
```

```
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

Histogram

```
data = [10, 20, 20, 30, 30, 30]

sns.histplot(data, bins=3)
plt.title("Histogram")
plt.show()
```

8. Data Cleaning

Data cleaning is the process of identifying and correcting or removing inaccurate, incomplete, or inconsistent data.

It is one of the most important steps in data preprocessing.

Common Data Cleaning Operations

- Handling missing values
- Filling missing data
- Removing invalid entries
- Renaming columns

Example

```
import pandas as pd
data = {'Name': ['A', 'B', None], 'Age': [25, None, 30]}
df = pd.DataFrame(data)
```

Remove Missing Values

```
print(df.dropna())
```

Fill Missing Values

```
df_filled = df.fillna(0)
print(df_filled)
```

Rename Column

```
df.rename(columns={'Name': 'Full Name'}, inplace=True)
print(df)
```

9. Basic Statistics

Basic statistics help summarize and understand data using numerical measures.

📌 Common Statistical Measures

- Mean (average)
- Minimum value
- Maximum value
- Standard deviation
- Descriptive summary

Example

```
print("Max Age:", df['Age'].max())

print("Min Age:", df['Age'].min())

print("Mean Age:", df['Age'].mean())
```

Full Summary

```
print(df.describe())
```

10. Basic Machine Learning

Machine Learning enables systems to learn patterns from data and make predictions without explicit programming.

📌 Machine Learning Workflow

1. Load dataset
2. Split data into training and testing sets
3. Train model
4. Make predictions
5. Evaluate performance

Example (Classification with Random Forest)

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report
```

Load Dataset

```
iris = load_iris()

X = iris.data

y = iris.target
```

Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Train Model

```
model = RandomForestClassifier()

model.fit(X_train, y_train)
```

Prediction & Evaluation

```
y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
```

Agricultural & Geospatial Data as ML Features

Dr. Foyez Ahmed Prodhan
Associate Professor
Department of Agricultural Extension and Rural Development
Gazipur Agricultural University (GAU)
Email: foyez@gau.edu.bd

🌍 Agricultural & Geospatial Data as ML Features

🔑 1. Basic Concept

What is Geospatial Data?

Geospatial data is any data that has a **location (latitude, longitude)** associated with it.

What is Agricultural Data?

Data related to farming, such as:

- Crop yield
- Soil properties
- Weather conditions
- Irrigation patterns

🗺️ 2. Types of Geospatial Data Used in Agriculture

🌐 A. Raster Data (Continuous Surface Data)

- Grid of pixels
- Each pixel has a value

Examples:

- NDVI (vegetation index)
- Rainfall maps
- Temperature maps
- Elevation (DEM)

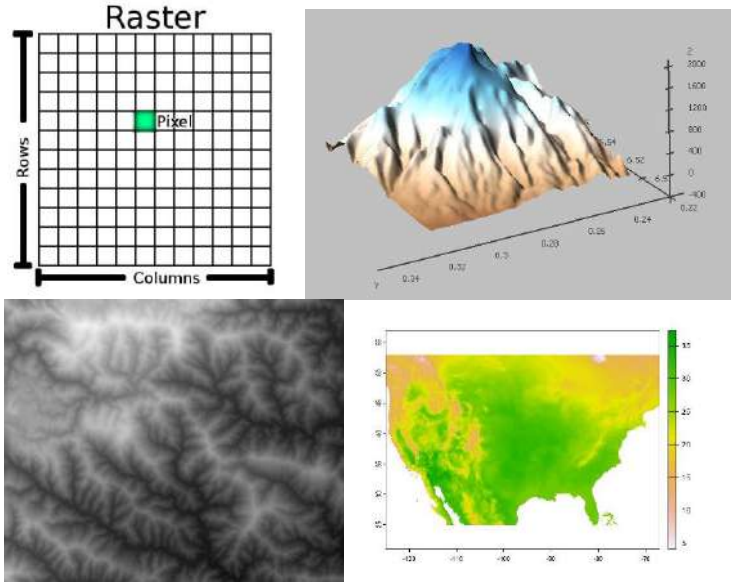


Figure. Example of raster data

🔑 B. Vector Data (Discrete Features)

- Points, lines, polygons

Examples:

- Farm boundaries
- Soil zones
- Irrigation networks
- Sampling points

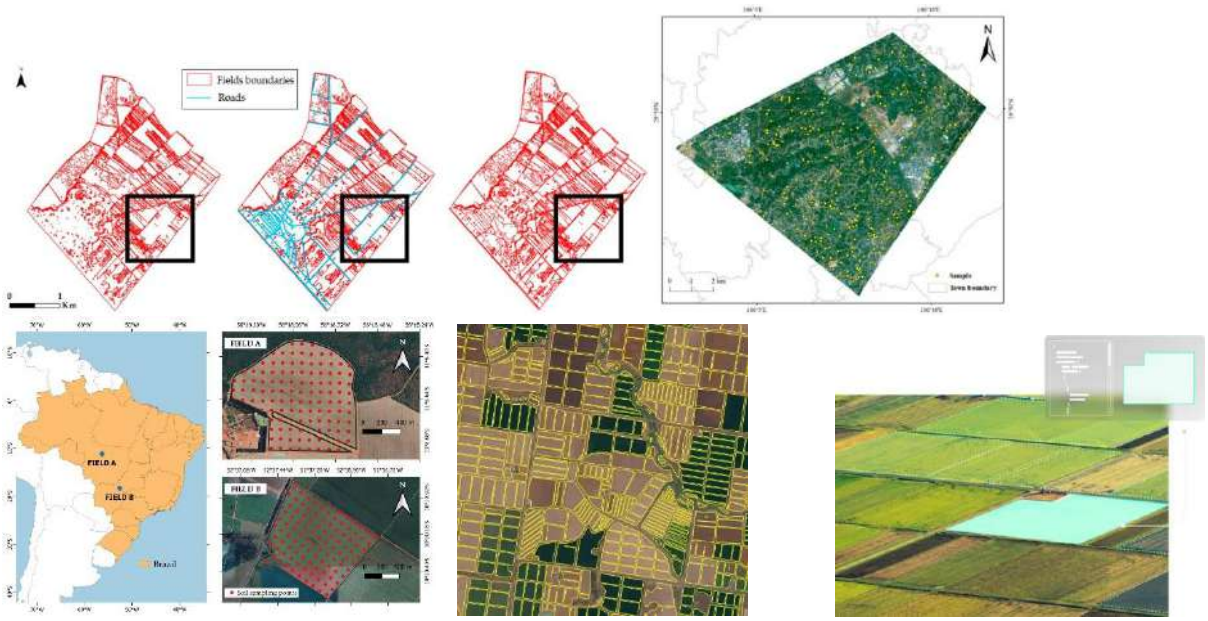


Figure. Example oof vector Data

Basic Concept of Machine Learning

1. Definition

Machine Learning (ML) is a branch of **Artificial Intelligence** that enables computers to **learn from data and improve their performance without being explicitly programmed**.

In simple terms:

Machine Learning allows a system to automatically learn patterns from data and make predictions or decisions.

2. Core Idea

Traditional programming:

- You give **rules + data → output**

Machine Learning:

- You give **data + output → system learns rules**

👉 The system discovers relationships between inputs and outputs.

3. Key Components of Machine Learning

3.1 Data

Data is the foundation of ML.

Types:

- Numerical (e.g., temperature, rainfall)
- Categorical (e.g., soil type, crop type)
- Images (e.g., satellite images)
- Time series (e.g., weather data over time)

3.2 Features
Features are the **input variables** used by the model.
Example:
- Rainfall
- Soil pH
- NDVI
- Temperature

3.3 Target (Label)
The output the model tries to predict.
Examples:
- Crop yield
- Crop type
- Disease presence

3.4 Model
A model is a mathematical function that maps:
Inputs (features) → Output (prediction)

3.5 Training
Training is the process where the model learns patterns from data.

3.6 Prediction
After training, the model can predict outcomes for new, unseen data.

4. How Machine Learning Works
Step-by-step process:
1. Collect data
2. Prepare and clean data
3. Select features
4. Train model
5. Evaluate model
6. Use model for prediction

5. Types of Machine Learning

5.1 Supervised Learning
- Uses labeled data (input + correct output)

Examples:
- Predict crop yield
- Classify crop type

Algorithms:
- Linear Regression
- Decision Trees
- Random Forest

5.2 Unsupervised Learning
- No labeled output
- Finds hidden patterns

Examples:
- Group similar farms
- Identify soil zones

Algorithms:

- Clustering (e.g., K-means)

5.3 Reinforcement Learning

- Learns through trial and error
- Uses rewards and penalties

Examples:

- Smart irrigation systems
- Autonomous farming machines

Turning Geospatial Data into ML Features

Step-by-Step Process

📌 Step 1: Define Target Variable

Example:

- Crop yield (tons/hectare)
- Crop type (classification)
- Disease presence (yes/no)

📌 Step 2: Select Input Layers

Data Type	Example
Soil	pH, texture
Climate	rainfall, temperature
Vegetation	NDVI
Terrain	elevation, slope

📌 Step 3: Extract Features at Locations

👉 For each location (point/pixel):

- Extract raster values
- Attach vector attributes

📊 Resulting Dataset

Lat	Lon	NDVI	Rainfall	Soil_pH	Elevation	Yield

This becomes a **training dataset for ML**.

Common agricultural and geospatial data sources used as features

Now let us look at the main categories of data that become ML features.

A. Soil data

Soil is one of the most important factors affecting agricultural productivity.

Common soil variables

- Soil texture (sand, silt, clay)
- Soil pH
- Organic matter
- Nitrogen content
- Phosphorus content

- Potassium content
- Cation exchange capacity
- Salinity
- Soil depth
- Soil moisture

From raw geospatial data to ML-ready features

Raw geospatial data usually cannot be fed directly into a machine learning model. It must first be transformed into a structured format.
The most common ML-ready form is a **table**, where:

- each row = one spatial observation
- each column = one feature
- one column = target variable

Field_ID	Latitude	Longitude	Soil_pH	Rainfall	NDVI	Elevation	Slope	Yield
F1	24.91	91.86	6.7	1320	0.74	42	3.1	4.8
F2	24.95	91.80	5.9	1280	0.61	58	6.8	3.9
F3	24.88	91.83	6.4	1355	0.79	38	2.0	5.2

This table can be used in:

- Linear Regression
- Random Forest
- XGBoost
- Neural Networks
- SVM
- Logistic Regression

Feature engineering from agricultural and geospatial data

Feature engineering means creating better predictors from raw data. This is one of the most important steps in machine learning.

A. Soil feature engineering
Raw soil data may include many measurements, but not all are directly useful.
Possible engineered features

- soil fertility score
- nutrient ratio
- soil texture category
- normalized pH
- moisture stress category

Example
Raw:

- pH = 5.6
- organic matter = 1.1
- nitrogen = 0.08

Engineered:

- fertility_index = weighted combination of pH, N, organic matter
- acidic_soil = 1

Creating spatial training datasets

A spatial training dataset is a machine learning dataset in which each sample is tied to a real-world location. This is the foundation of geospatial ML.

Steps to create a spatial training dataset

Step 1: Define the target

Decide what you want to predict.

Examples:

- yield per field
- crop type
- disease occurrence
- irrigation requirement
- soil class

Step 2: Collect ground truth data

Ground truth means actual observed values from the field.

Examples:

- measured crop yield
- farmer survey data
- soil sample results
- crop labels from field visits
- disease observations

Without reliable ground truth, supervised ML cannot be trained properly.

Step 3: Identify sample locations

Each observation needs a location.

Examples:

- field centroid
- soil sample point
- farm polygon
- grid cell
- village coordinate

Step 4: Gather spatial layers

Collect relevant geospatial layers such as:

- soil map
- climate raster
- DEM
- NDVI images
- irrigation map
- land cover map

Step 5: Extract features

Use GIS operations to attach values from these layers to each sample location.

Examples:

- extract rainfall from climate raster
- get soil polygon class
- calculate mean NDVI per field
- derive slope from DEM

Step 6: Clean and transform data
This includes:

- handling missing values
- removing duplicates
- normalizing variables
- encoding categorical fields
- aligning units and coordinate systems

Step 7: Build the final dataset
The dataset should now have:

- sample ID
- location
- features
- target label

Example: crop yield prediction
Let us walk through a full example.
Problem
Predict rice yield for farms in a region.
Ground truth
For 500 farms, we know:

- farm boundary
- actual yield in tons/hectare

Geospatial layers

- Soil map
- Rainfall raster
- Temperature raster
- NDVI time series
- Elevation raster

Features extracted

- soil pH
- soil texture
- rainfall during season
- average temperature
- NDVI at peak season
- elevation
- slope

Resulting table

Farm	Soil_pH	Soil_Texture	Rainfall	Temp	NDVI	Elevation	Slope	Yield
A	**6.3**	**Loam**	**1400**	**27.8**	**0.80**	**35**	**1.2**	**5.6**
B	**5.7**	**Clay**	**1220**	**29.1**	**0.67**	**50**	**4.5**	**4.3**

hen a model such as Random Forest is trained to learn the relationship between these features and yield.
Later, if we have the same features for new farms, the model can predict their yield.

Important preprocessing issues

Using agricultural and geospatial data in ML is powerful, but it has challenges.

A. Spatial resolution mismatch
Different datasets have different pixel sizes.
Example:

- NDVI at 10 m
- rainfall at 1 km
- soil map at 250 m

These do not align automatically.
Solution

- resample layers
- aggregate to common scale
- choose a unit of analysis such as field or grid cell

B. Coordinate reference system mismatch
Datasets may use different map projections.
Problem
If projections differ, layers may not overlap correctly.
Solution
Reproject everything to a common coordinate reference system.

C. Missing data
Common causes:

- cloud cover in satellite images
- missing weather records
- incomplete soil surveys

Solutions

- interpolation
- imputation
- masking
- using seasonal composites

D. Temporal mismatch
Agricultural data changes over time.
Example:

- soil map from 2021
- rainfall from 2025
- yield data from 2024
- NDVI from 2023

This can cause errors if layers do not correspond to the same season or year.
Solution
Try to align features with the time period relevant to the target.

Spatial autocorrelation and why it matters

This is a very important concept in spatial ML.
What is spatial autocorrelation?
It means nearby locations tend to be more similar than distant locations.
For example:

- neighboring farms may have similar soil
- nearby pixels may have similar vegetation
- adjacent areas may have similar rainfall

Why this matters for ML
If training and test samples are very close to each other, the model may seem highly accurate, but only because nearby points are similar.
This can lead to overly optimistic evaluation.
Better approach
Use:

- spatial train-test split
- block cross-validation
- region-based validation

This tests whether the model can generalize to new places, not just nearby places.

How machine learning models use these features
After building the spatial dataset, standard ML algorithms can be applied.
For regression
Predict continuous values:

- crop yield
- soil moisture
- biomass
- water requirement

Models:

- Linear Regression
- Random Forest Regressor
- Gradient Boosting
- XGBoost
- Neural Networks

For classification
Predict categories:

- crop type
- disease class
- land suitability class
- irrigated vs non-irrigated

Models:

- Logistic Regression
- Decision Trees
- Random Forest
- SVM
- XGBoost
- Deep Learning

Why feature engineering is often more important than model complexity
A very advanced model with poor features may perform badly.
A simpler model with strong agricultural and geospatial features may perform very well.

For example:

- A simple Random Forest with good rainfall, NDVI, slope, and soil features can outperform a deep model trained on weak or poorly prepared inputs.

This is why domain understanding matters.
To build strong features, you need to understand:

- agronomy
- GIS
- remote sensing

- climate effects
- crop growth stages

Example of Python code for RF Machine learning model

```
import os

import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
from pyproj import CRS
# Define data folder
dataFolder = r"D:\ML_Training\Data_05"
# Load data
ID = pd.read_csv(os.path.join(dataFolder, "GP_ID.csv"))
gps = pd.read_csv(os.path.join(dataFolder, "GP_GPS.csv"))
data = pd.read_csv(os.path.join(dataFolder, "GP_SOC_data.csv"))
# Preview
print(ID.head())
print(gps.head())
print(data.head())
# Merge dataframes by ID
df_01 = pd.merge(ID, gps, on="ID")
df = pd.merge(df_01, data, on="ID")

df.head()
# Check column names
print(df.columns)
# Replace these with your actual longitude and latitude column names
lon_col = df.columns[5]   # equivalent to R column 6
lat_col = df.columns[6]   # equivalent to R column 7

print("Longitude column:", lon_col)
print("Latitude column:", lat_col)
# Replace these with your actual longitude and latitude column names
lon_col = df.columns[5]   # equivalent to R column 6
lat_col = df.columns[6]   # equivalent to R column 7

print("Longitude column:", lon_col)
print("Latitude column:", lat_col)
# Create GeoDataFrame from longitude and latitude
gdf = gpd.GeoDataFrame(
    df,
    geometry=gpd.points_from_xy(df[lon_col], df[lat_col]),
    crs="EPSG:4326"
)
print(gdf.crs)
gdf.head()
# Check CRS
print(gdf.crs)
# Load state shapefile
state = gpd.read_file(os.path.join(dataFolder, "GP_STATE.shp"))
# Check shapefile CRS
print(state.crs)
state.head()
# Reproject point data to match state shapefile CRS
```

```
gdf_proj = gdf.to_crs(state.crs)

# Check projected CRS
print(gdf_proj.crs)
# Save projected GeoDataFrame as shapefile
output_shp = os.path.join(dataFolder, "GP_Data_PROJ.shp")
gdf_proj.to_file(output_shp)

print("Projected shapefile saved to:")
print(output_shp)
import matplotlib.pyplot as plt
# Plot original and projected data
fig, axes = plt.subplots(1, 2, figsize=(14, 6))
# Left: WGS84 points with state boundary reprojected to WGS84 for display
state_wgs84 = state.to_crs("EPSG:4326")
state_wgs84.plot(ax=axes[0], facecolor="none", edgecolor="black")
gdf.plot(ax=axes[0], markersize=5)
axes[0].set_title("WGS 1984")

# Right: projected points with original state boundary
state.plot(ax=axes[1], facecolor="none", edgecolor="black")
gdf_proj.plot(ax=axes[1], markersize=5)
axes[1].set_title("Albers Equal Area Conic")

plt.tight_layout()
plt.show()
import glob
import numpy as np
import rasterio
from rasterio.transform import xy
from rasterio.windows import Window
from rasterstats import point_query
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import pearsonr
from scipy.cluster.hierarchy import linkage, leaves_list
from scipy.spatial.distance import squareform
from sklearn.model_selection import train_test_split
# User inputs
# ------------------------------------------------------------
dataFolder = "D:/ML_Training/DATA_05/RASTER"   # change this
dataFolder1 = "D:/ML_Training/DATA_05"   # change this
spdf_path = os.path.join(dataFolder1, "GP_Data_PROJ.shp")  # change this
albers_crs = None  # e.g. "EPSG:5070" if needed
# ------------------------------------------------------------
# 1) Convert SPDF.PROJ to a dataframe and rename columns
# ------------------------------------------------------------
point_gdf = gpd.read_file(spdf_path)

# If needed, project to Albers
if albers_crs is not None:
    point_gdf = point_gdf.to_crs(albers_crs)
point_df = point_gdf.copy()
# Add x and y columns from geometry
point_df["x"] = point_df.geometry.x
```

```
point_df["y"] = point_df.geometry
# If you specifically want to rename the 9th and 10th columns like in R:
# cols = list(point_df.columns)
# cols[8] = "x"
# cols[9] = "y"
# point_df.columns = col
# 2) Create raster list and stack-like metadata
# ------------------------------------------------------------
raster_dir = dataFolder
glist = glob.glob(os.path.join(raster_dir, "*.tif"))
# Open rasters
rasters = {os.path.splitext(os.path.basename(fp))[0]: rasterio.open(fp) for fp in glist}

# Optional quick plot of the first raster
first_name = list(rasters.keys())[0]
first_src = rasters[first_name]

plt.figure(figsize=(8, 6))
plt.imshow(first_src.read(1), cmap="terrain")
plt.title(first_name)
plt.colorbar()
plt.tight_layout()
plt.show()
# 3) Extract raster values to point locations
# Equivalent to raster::extract(s, SPDF.PROJ, df=TRUE, method="simple")
# ------------------------------------------------------------
point_coords = [(geom.x, geom.y) for geom in point_gdf.geometry]

vals_dict = {}
for name, src in rasters.items():
    vals = point_query(point_gdf, src.name, interpolate="nearest")
    vals_dict[name] = vals

vals_df = pd.DataFrame(vals_dict)

# Similar to cbind(point.df, vals)
point_vals = pd.concat([point_df.reset_index(drop=True), vals_df.reset_index(drop=True)],
axis=1)
dataFolder = r"D:\ML_Training\Data_05"
# ------------------------------------------------------------
# 4) Load ID files and merge NLCD / FRG descriptions
# ------------------------------------------------------------
nlcd_id = pd.read_csv(os.path.join(dataFolder, "NLCD_ID.csv"))
frg_id = pd.read_csv(os.path.join(dataFolder, "FRG_ID.csv"))

mf_01 = point_vals.merge(nlcd_id, on="NLCD", how="inner")
mf_02 = mf_01.merge(frg_id, on="FRG", how="inner")
# Delete column 3 (extra ID) like R: mf.02 <- mf.02[, -3]
# R is 1-based; column 3 means Python index 2
#mf_02 = mf_02.drop(columns=mf_02.columns[1])
# Re-arrange columns
mf = mf_02[
    [
        "ID", "STATE_ID", "STATE", "FIPS", "COUNTY", "Longitude", "Latitude",
        "x", "y", "SOC", "ELEV", "Aspect", "Slope", "TPI", "K_Factor",
        "MAP", "MAT", "NDVI", "Slit_Clay", "NLCD", "FRG", "NLCD_DES", "FRG_DES"
    ]
```

```
].copy()

print(mf.head())
print(mf_02.columns.tolist())
print(mf_01.columns.tolist())
# Write CSV
mf.to_csv(os.path.join(dataFolder, "GP_all_data11.csv"), index=False)
# ------------------------------------------------------------
# 1. Set paths (FIXED)
# ------------------------------------------------------------
raster_dir = r"D:\ML_Training\DATA_05\RASTER"
dataFolder = r"D:\ML_Training\DATA_05"   # for saving output

out_path = os.path.join(dataFolder, "GP_prediction_grid_data1.csv")
# 3. Load rasters
# ------------------------------------------------------------
rasters = {
    os.path.splitext(os.path.basename(fp))[0]: rasterio.open(fp)
    for fp in glist
}

print("Loaded rasters:", list(rasters.keys()))
# ------------------------------------------------------------
# 4. Create prediction grid from ELEV raster
# ------------------------------------------------------------
dem_path = os.path.join(raster_dir, "ELEV.tif")

print("DEM path:", dem_path)
print("Exists:", os.path.exists(dem_path))

if not os.path.exists(dem_path):
    raise ValueError("❌ ELEV.tif not found!")
with rasterio.open(dem_path) as dem:
    band1 = dem.read(1)

    rows, cols = np.where(~np.isnan(band1))

    xs, ys = rasterio.transform.xy(dem.transform, rows, cols)

    grid_point_df = pd.DataFrame({"x": xs, "y": ys})

    dem_crs = dem.crs

print("Grid points created:", len(grid_point_df))
# ------------------------------------------------------------
# 5. Convert to GeoDataFrame
# ------------------------------------------------------------
grid_gdf = gpd.GeoDataFrame(
    grid_point_df,
    geometry=gpd.points_from_xy(grid_point_df["x"], grid_point_df["y"]),
    crs=dem_crs
)

print("GeoDataFrame ready")

# ------------------------------------------------------------
# 6. Extract raster values
# ------------------------------------------------------------
```

```
grid_vals_dict = {}

for name, src in rasters.items():
    vals = point_query(grid_gdf, src.name, interpolate="nearest")
    grid_vals_dict[name] = vals
    print(f"{name}: {len(vals)} values extracted")

df_grid = pd.DataFrame(grid_vals_dict)

print("df_grid shape:", df_grid.shape)

# ------------------------------------------------------------
# 7. Combine coordinates + raster values
# ------------------------------------------------------------
grid = pd.concat(
    [grid_gdf.drop(columns="geometry").reset_index(drop=True),
     df_grid.reset_index(drop=True)],
    axis=1
)

print("Grid shape:", grid.shape)
print(grid.head())

# ------------------------------------------------------------
# 8. Handle missing values (IMPORTANT)
# ------------------------------------------------------------
print("\nMissing values:")
print(grid.isna().sum())

# Drop rows where raster values are missing
grid_out = grid.dropna(subset=list(rasters.keys()))

print("After dropping NaNs:", grid_out.shape)

# ------------------------------------------------------------
# 9. Save CSV
# ------------------------------------------------------------
grid_out.to_csv(out_path, index=False)

print("\n✅ File saved to:")
print(out_path)
print("Exists:", os.path.exists(out_path))
# ------------------------------------------------------------
# 11. Correlation of SOC with environmental variables
# ------------------------------------------------------------
corr_cols = ["SOC", "ELEV", "Aspect", "Slope", "TPI", "K_Factor", "MAP", "MAT", "NDVI",
"Slit_Clay"]
df_cor = mf[corr_cols].copy()

print("\nCorrelation input:")
print(df_cor.head())
# Correlation matrix
R = df_cor.corr(method="pearson")

# P-value matrix
P = pd.DataFrame(np.ones((len(corr_cols), len(corr_cols))), index=corr_cols,
```

```
columns=corr_cols)

for i, c1 in enumerate(corr_cols):
    for j, c2 in enumerate(corr_cols):
        if i == j:
            P.iloc[i, j] = 0.0
        else:
            r, p = pearsonr(df_cor[c1], df_cor[c2])
            P.iloc[i, j] = p

print("\nCorrelation matrix:")
print(R)

print("\nP-value matrix:")
print(P)
# Plot with insignificant correlations blanked
mask = P > 0.05

plt.figure(figsize=(10, 8))
sns.heatmap(
    R,
    mask=mask,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    vmin=-1,
    vmax=1,
    square=True,
    cbar_kws={"label": "Correlation"}
)
plt.title("Correlation matrix")
plt.tight_layout()
plt.show()
# ----------------------------------------------------------------
# 12. Clustered correlation plot (similar to order='hclust')
# ----------------------------------------------------------------
dist = 1 - np.abs(R)
link = linkage(squareform(dist.values, checks=False), method="average")
order = leaves_list(link)

R_ord = R.iloc[order, order]
P_ord = P.iloc[order, order]
mask_ord = P_ord > 0.05

plt.figure(figsize=(10, 8))
sns.heatmap(
    R_ord,
    mask=mask_ord,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    vmin=-1,
    vmax=1,
    square=True,
```

```
    cbar_kws={"label": "Correlation"}
)
plt.title("Clustered correlation matrix")
plt.tight_layout()
plt.show()
# ------------------------------------------------------------
# 13. Variability of SOC in relation to NLCD (box-jitter plot)
# ------------------------------------------------------------
df_cat = mf[["SOC", "NLCD_DES", "FRG_DES"]].copy()
print("\nCategorical plot data:")
print(df_cat.head())

plt.figure(figsize=(10, 8))

# Boxplot
sns.boxplot(
    data=df_cat,
    x="SOC",
    y="NLCD_DES",
    color="white",
    fliersize=0,
    linewidth=1
)

# Jittered points
sns.stripplot(
    data=df_cat,
    x="SOC",
    y="NLCD_DES",
    hue="SOC",
    palette="Spectral",
    jitter=0.15,
    size=3,
    dodge=False
)

plt.xlabel("SOC")
plt.ylabel("")
plt.title("Variability of SOC in relation to NLCD")
plt.legend(title="SOC (mg C/g)", bbox_to_anchor=(1.02, 1), loc="upper left")
plt.tight_layout()
plt.show()
# ------------------------------------------------------------
# 14. Stratified train/test split by NLCD and FRG
# ------------------------------------------------------------
mf["NLCD"] = mf["NLCD"].astype("category")
mf["FRG"] = mf["FRG"].astype("category")

tr_prop = 0.80
seed = 101

train_parts = []
test_parts = []
```

```
for (nlcd, frg), group in mf.groupby(["NLCD", "FRG"], observed=True):
    if len(group) == 1:
        train_group = group.copy()
        test_group = group.iloc[0:0].copy()
    else:
        train_group, test_group = train_test_split(
            group,
            train_size=tr_prop,
            random_state=seed
        )

    train_parts.append(train_group)
    test_parts.append(test_group)

train = pd.concat(train_parts).reset_index(drop=True)
test = pd.concat(test_parts).reset_index(drop=True)

train.to_csv(os.path.join(dataFolder, "train_data.csv"), index=False)
test.to_csv(os.path.join(dataFolder, "test_data.csv"), index=False)

print("\nTrain shape:", train.shape)
print("Test shape:", test.shape)
```

Exploratory Data Analysis (EDA): RMSE, Accuracy, Confusion Matrix

Mr. Al-Helal
Programmer (Com & GIS)
Bangladesh Agricultural Research Council
Email: al.helal@barc.gov.bd

Understanding the Confusion Matrix in Machine Learning

Machine learning models are increasingly used in various applications to classify data into different categories. However evaluating the performance of these models is crucial to ensure their accuracy and reliability. One essential tool in this evaluation process is the confusion matrix. In this article we will work on confusion matrix, its significance in machine learning and how it can be used to improve the performance of classification models.

Understanding Confusion Matrix

A **confusion matrix** is a simple table that shows how well a classification model is performing by comparing its predictions to the actual results. It breaks down the predictions into four categories: correct predictions for both classes (**true positives** and **true negatives**) and incorrect predictions (**false positives** and **false negatives**). This helps you understand where the model is making mistakes, so you can improve it.

The matrix displays the number of instances produced by the model on the test data.

- **True Positive (TP):** The model correctly predicted a positive outcome (the actual outcome was positive).
- **True Negative (TN):** The model correctly predicted a negative outcome (the actual outcome was negative).
- **False Positive (FP):** The model incorrectly predicted a positive outcome (the actual outcome was negative). Also known as a Type I error.
- **False Negative (FN):** The model incorrectly predicted a negative outcome (the actual outcome was positive). Also known as a Type II error.

A **confusion matrix** helps you see how well a model is working by showing correct and incorrect predictions. It also helps calculate key measures like **accuracy**, **precision**, and **recall**, which give a better idea of performance, especially when the data is imbalanced.

Metrics based on Confusion Matrix Data

1. Accuracy

Accuracy measures how often the model's predictions are correct overall. It gives a general idea of how well the model is performing. However, accuracy can be misleading, especially with imbalanced datasets where one class dominates. For example, a model that predicts the majority class correctly most of the time might have high accuracy but still fail to capture important details about other classes.

$$\textbf{Accuracy} = \frac{\textbf{TP} + \textbf{TN}}{\textbf{TP} + \textbf{TN} + \textbf{FP} + \textbf{FN}}$$

2. Precision

Precision focuses on the quality of the model's positive predictions. It tells us how many of the instances predicted as positive are actually positive. Precision is important in situations where false positives need to be minimized, such as detecting spam emails or fraud.

$$\textbf{Precision} = \frac{\textbf{TP}}{\textbf{TP} + \textbf{FP}}$$

3. Recall

Recall measures how well the model identifies all actual positive cases. It shows the proportion of true positives detected out of all the actual positive instances. High recall is essential when missing positive cases has significant consequences, such as in medical diagnoses.

$$\textbf{Recall} = \frac{\textbf{TP}}{\textbf{TP} + \textbf{FN}}$$

4. F1-Score

F1-score combines precision and recall into a single metric to balance their trade-off. It provides a better sense of a model's overall performance, particularly for imbalanced datasets. The F1 score is helpful when both false positives and false negatives are important, though it assumes precision and recall are equally significant, which might not always align with the use case.

$$\textbf{F1} - \textbf{Score} = \frac{\textbf{2} \times \textbf{Precision} \times \textbf{Recall}}{\textbf{Precision} \ + \ \textbf{Recall}}$$

5. Specificity

Specificity is another important metric in the evaluation of classification models, particularly in binary classification. It measures the ability of a model to correctly identify negative instances. Specificity is also known as the True Negative Rate. Formula is given by:
Specificity=TN/TN+FP

6. Type 1 and Type 2 error

- **Type 1 error**
- A **Type 1 Error** occurs when the model incorrectly predicts a positive instance, but the actual instance is negative. This is also known as a **false positive**. Type 1 Errors affect the **precision** of a model, which measures the accuracy of positive predictions.

$$\textbf{Type 1 Error} = \frac{\textbf{FP}}{\textbf{FP} + \textbf{TN}}$$

- **Type 2 error**
  A **Type 2 Error** occurs when the model fails to predict a positive instance, even though it

is actually positive. This is also known as a **false negative**. Type 2 Errors impact the **recall** of a model, which measures how well the model identifies all actual positive cases.

$$\textbf{Type 2 Error} = \frac{\textbf{FN}}{\textbf{TP} + \textbf{FN}}$$

Example: A diagnostic test is used to detect a particular disease in patients.

- **Type 1 Error (False Positive):** This occurs when the test predicts a patient has the disease (positive result), but the patient is actually healthy (negative case).
- **Type 2 Error (False Negative):** This occurs when the test predicts the patient is healthy (negative result), but the patient actually has the disease (positive case).

Confusion Matrix For binary classification

A 2X2 Confusion matrix is shown below for the image recognition having a Dog image or Not Dog image:

	Predicted	**Predicted**
Actual	True Positive (TP)	False Negative (FN)
Actual	False Positive (FP)	True Negative (TN)

- **True Positive (TP):** It is the total counts having both predicted and actual values are Dog.
- **True Negative (TN):** It is the total counts having both predicted and actual values are Not Dog.
- **False Positive (FP):** It is the total counts having prediction is Dog while actually Not Dog.
- **False Negative (FN):** It is the total counts having prediction is Not Dog while actually, it is Dog.

Example: Confusion Matrix for Dog Image Recognition with Numbers

Index	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Actual	Dog	Dog	Dog	Not Dog	Dog	Not Dog	Dog	Dog	Not Dog	Not Dog
Predicted	Dog	Not Dog	Dog	Not Dog	Dog	Dog	Dog	Dog	Not Dog	Not Dog

Index	1	2	3	4	5	6	7	8	9	10
Result	TP	FN	TP	TN	TP	FP	TP	TP	TN	TN

- Actual Dog Counts = 6
- Actual Not Dog Counts = 4
- True Positive Counts = 5
- False Positive Counts = 1
- True Negative Counts = 3
- False Negative Counts = 1

		Predicted	
		Dog	**Not Dog**
Actual	**Dog**	True Positive (TP =5)	False Negative (FN =1)
	Not Dog	False Positive (FP=1)	True Negative (TN=3)

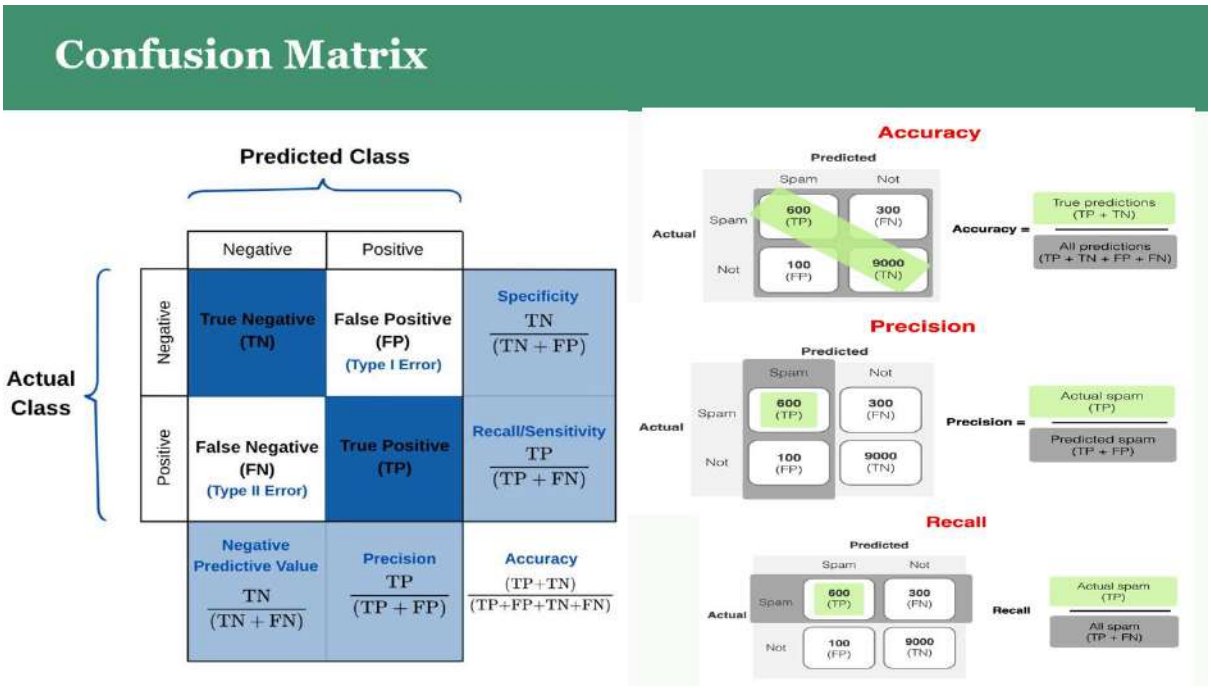
Example: Confusion Matrix for Image Classification (Cat, Dog, Horse)

	Predicted Cat	**Predicted Dog**	**Predicted Horse**
Actual Cat	True Positive (TP)	False Negative (FN)	False Negative (FN)
Actual Dog	False Negative (FN)	True Positive (TP)	False Negative (FN)
Actual Horse	False Negative (FN)	False Negative (FN)	True Positive (TP)

- The definitions of all the terms (TP, TN, FP and FN) are the same as described in the previous example.

Example with Numbers:
Let’s consider the scenario where the model processed 30 images:

	Predicted Cat	Predicted Dog	Predicted Horse
Actual Cat	8	1	1
Actual Dog	2	10	0
Actual Horse	0	2	8



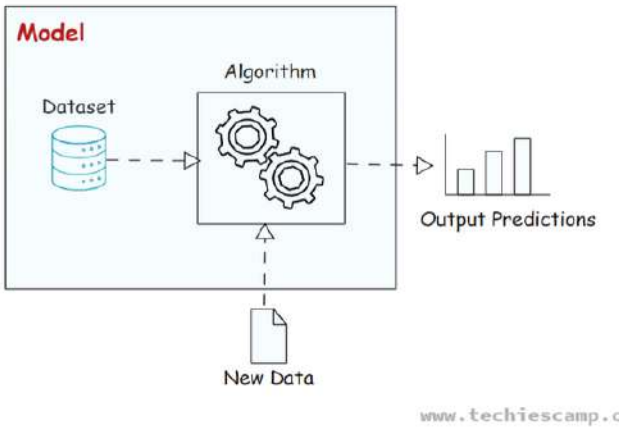
Supervised Learning – Classification Algorithms

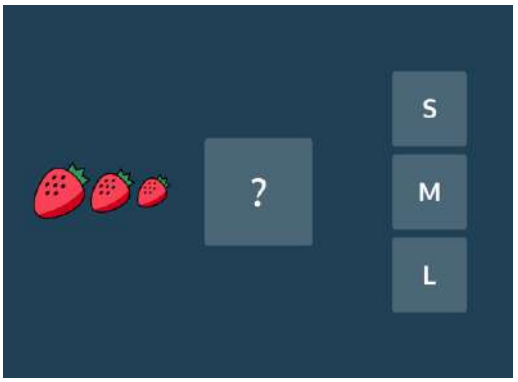
Mr. Aditya Rajbongshi
Assistant Professor, University of Frontier Technology, Bangladesh

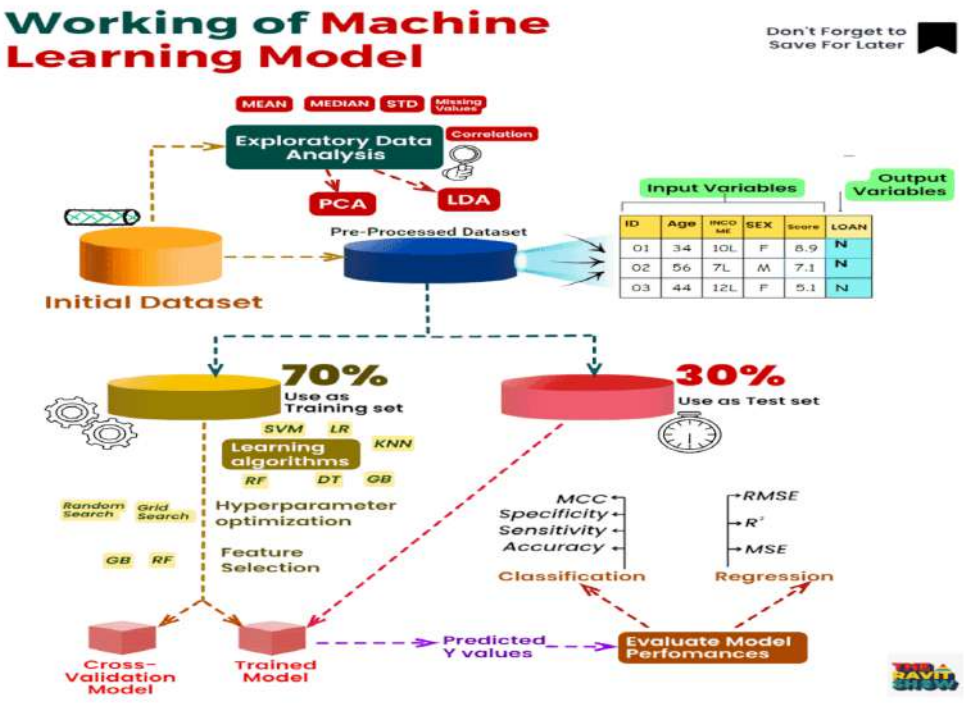
Image Classification using Machine Learning Classifier

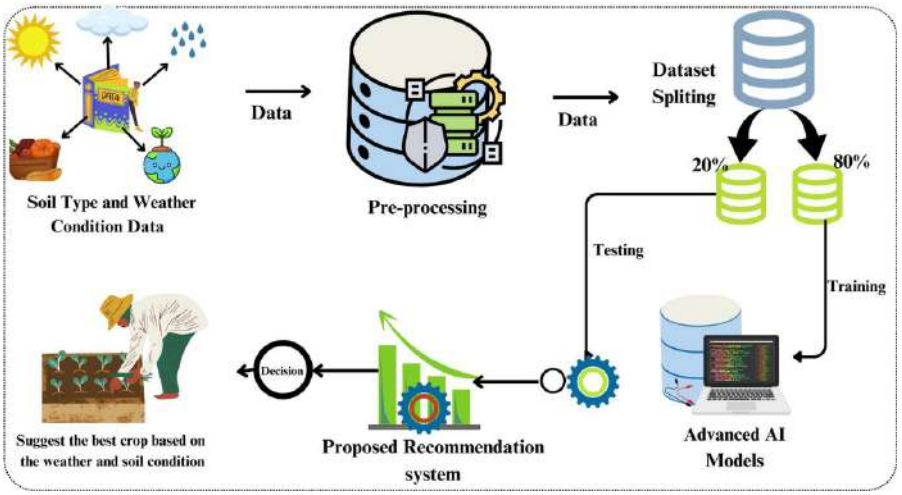
Model in ML ?

- In Machine Learning, a Classifier is a tool or function that helps the Computer make similar decisions.

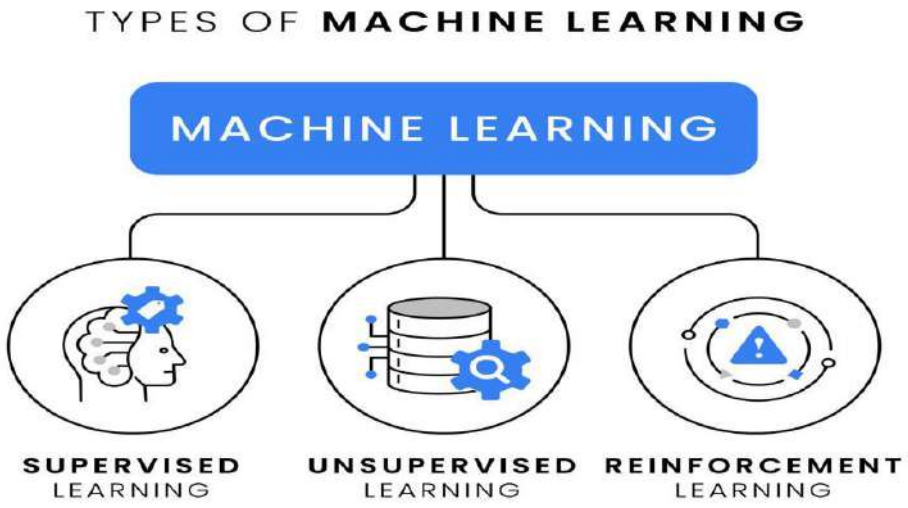


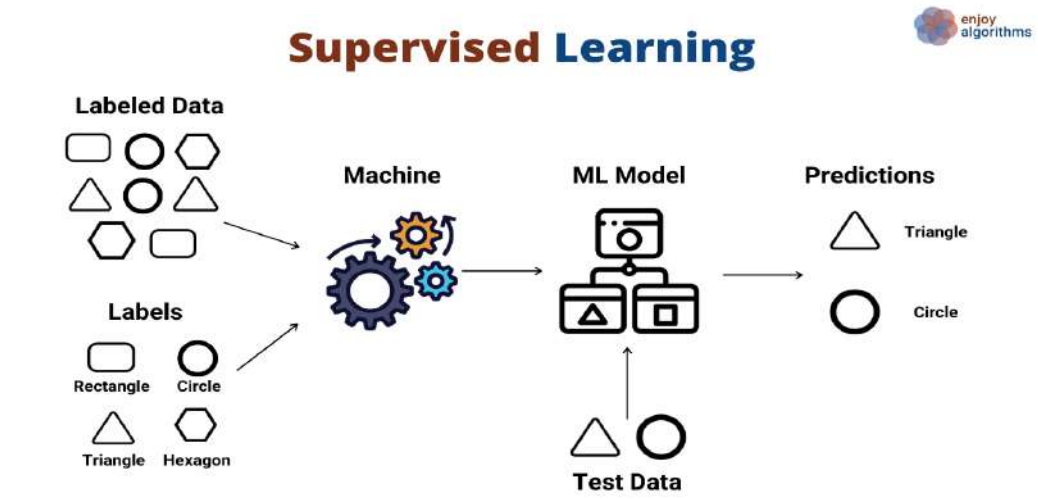


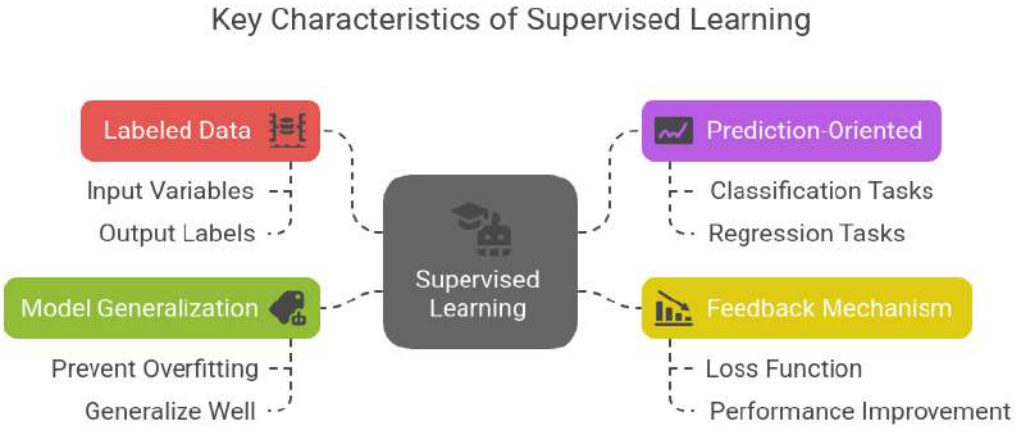



Soil Type and Weather Condition Data
Data
Pre-processing
Data
Dataset Spliting
20%
80%
Testing
Training
Advanced AI Models
Proposed Recommendation system
Decision
Suggest the best crop based on the weather and soil condition


Top 15 Datasets in Agriculture

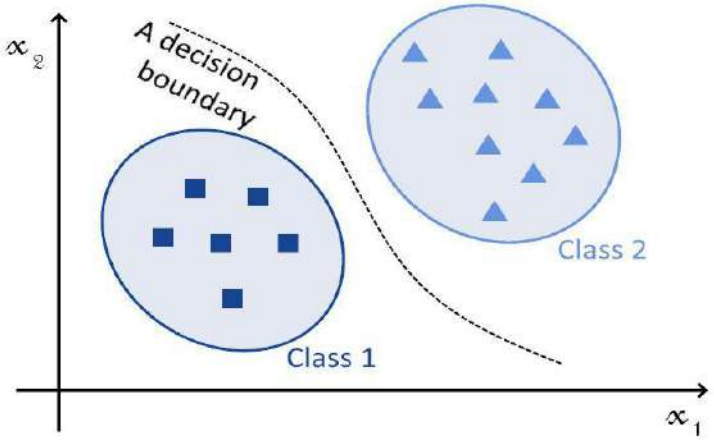

TYPES OF MACHINE LEARNING
MACHINE LEARNING
SUPERVISED LEARNING
UNSUPERVISED LEARNING
REINFORCEMENT LEARNING



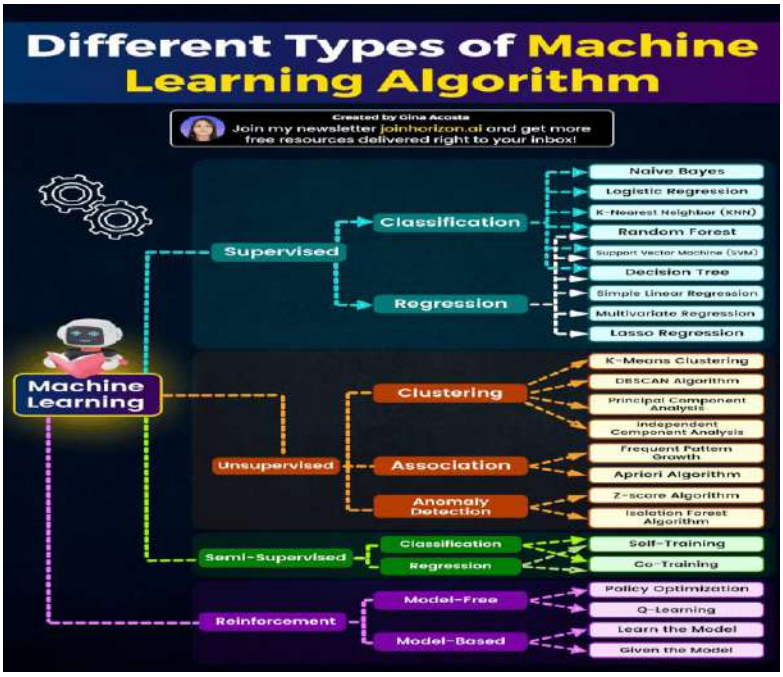


What is Classification Algorithm?

A type of supervised learning
Predicts categories (labels)
Output is discrete (e.g., Yes/No)



Different Types of Machine Learning Algorithm



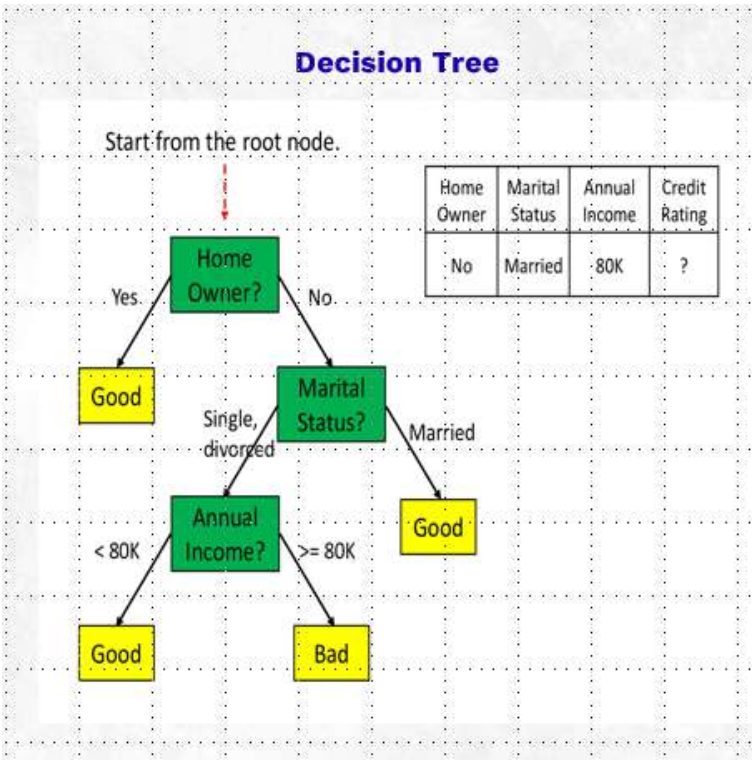
What is Decision Tree?

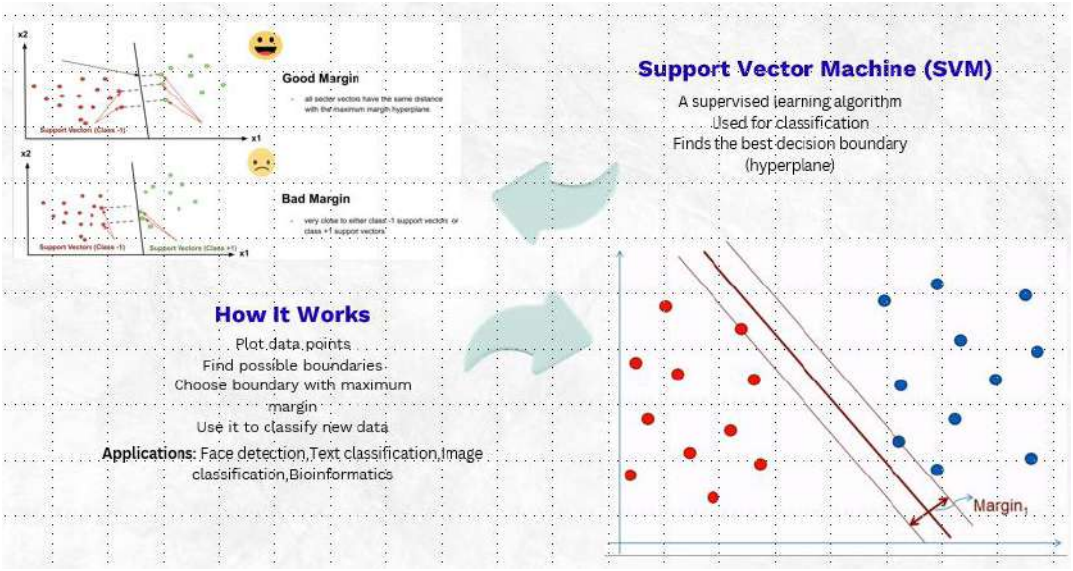
A supervised learning algorithm
Uses a tree-like structure
Makes decisions using if-else rules

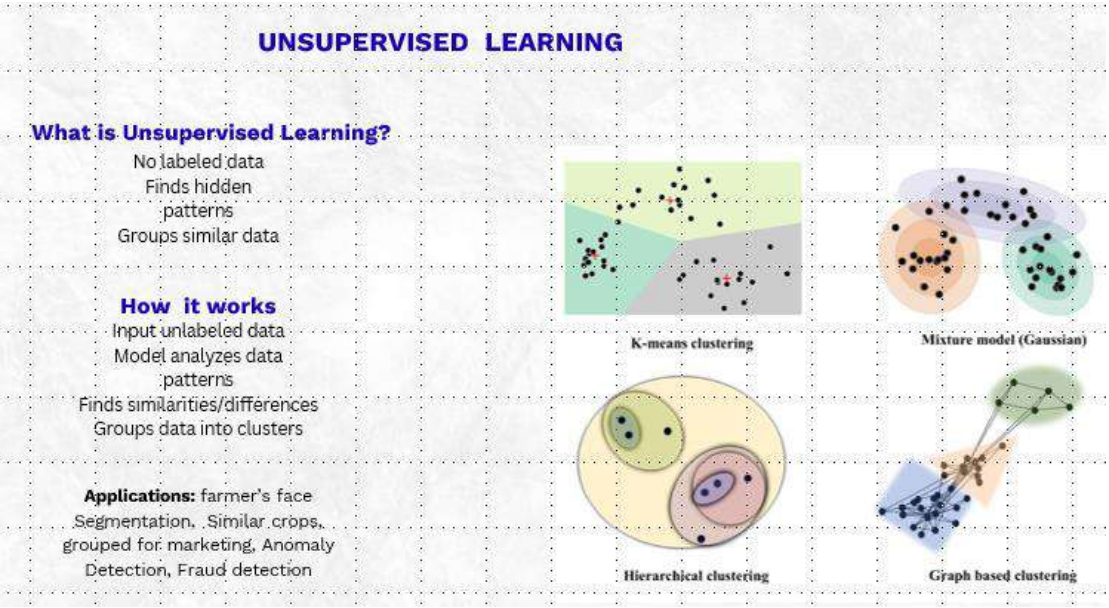
How It Works?

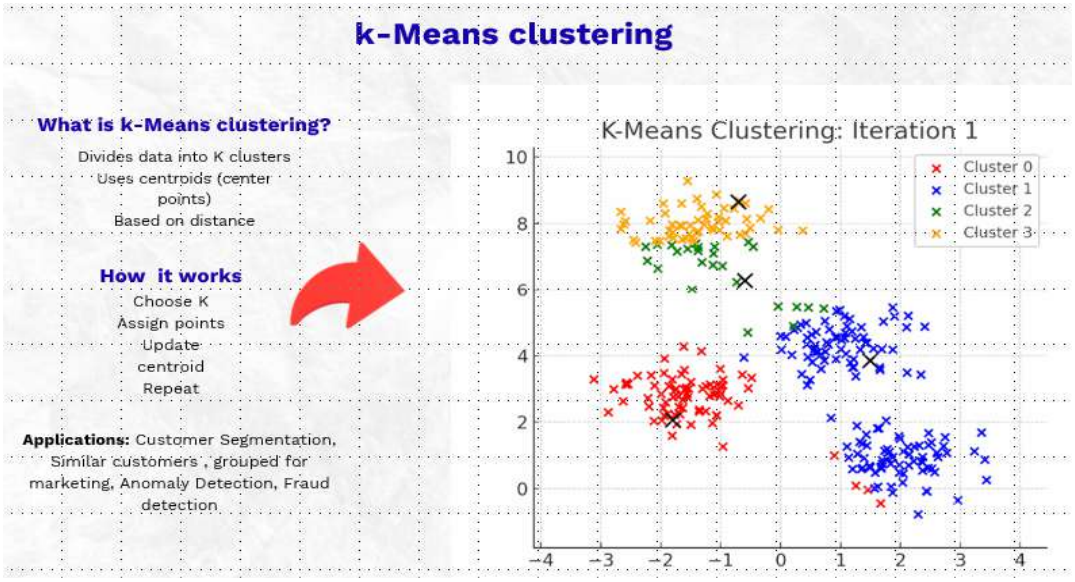
Select best feature (e.g., Age)
Split data into branches
Repeat splitting for each branch
Stop when final decision is reached

Applications:
Crop Diseases recognition, Medical diagnosis (Disease prediction), Spam email detection, Customer decision analysis




Good Margin
Bad Margin
Support Vector Machine (SVM)
A supervised learning algorithm
Used for classification
Finds the best decision boundary
(hyperplane)
How It Works
Plot data points
Find possible boundaries
Choose boundary with maximum margin
Use it to classify new data
Applications: Face detection,Text classification,Image classification,Bioinformatics
Margin


UNSUPERVISED LEARNING
What is Unsupervised Learning?
No labeled data
Finds hidden patterns
Groups similar data
How it works
Input unlabeled data
Model analyzes data patterns
Finds similarities/differences
Groups data into clusters
Applications: farmer's face Segmentation, Similar crops, grouped for marketing, Anomaly Detection, Fraud detection
K-means clustering
Mixture model (Gaussian)
Hierarchical clustering
Graph based clustering


k-Means clustering
What is k-Means clustering?
Divides data into K clusters
Uses centroids (center points)
Based on distance
How it works
Choose K
Assign points
Update centroid
Repeat
Applications: Customer Segmentation, Similar customers , grouped for marketing, Anomaly Detection, Fraud detection
K-Means Clustering: Iteration 1
Cluster 0
Cluster 1
Cluster 2
Cluster 3

HIERAERCHICAL CLUSTERING

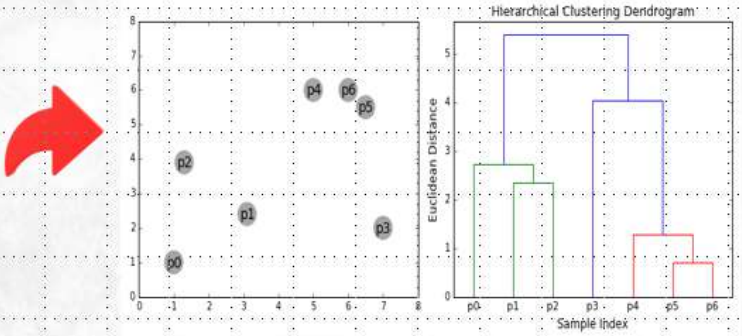
What is Hierarchical Clustering?

Creates a tree structure (dendrogram)
No need to define K initially

How it works?

Merge closest clusters (bottom-up)
Or split clusters (top-down)

Applications: Customer Segmentation, Similar customers, grouped for marketing, Anomaly Detection, Fraud detection



ENSEMBLE LEARNING

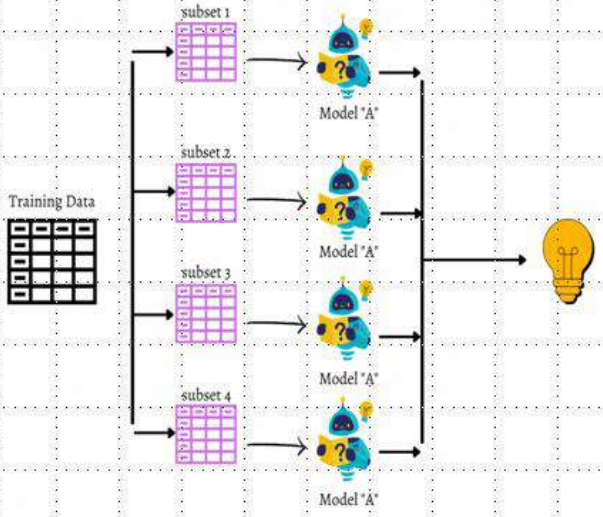
What is Ensemble Learning?

Combines multiple models
Improves accuracy
Reduces overfitting

How it works?

Take random samples of data
Train multiple decision trees
Each tree gives prediction
Combine results (voting)

Applications:
Agriculture prediction
Medical diagnosis,
Recommendation systems,



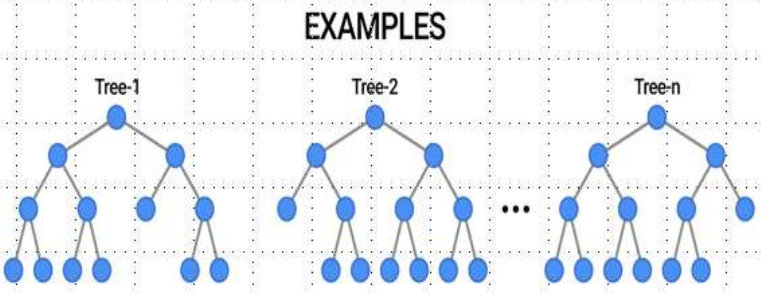
RANDOM FOREST

What is Ensemble Learning?

An ensemble learning algorithm
Uses multiple Decision Trees
Combines results for better accuracy

How it works?

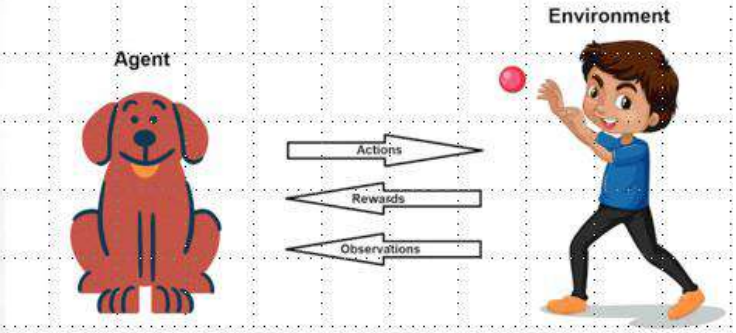
Take random samples from dataset Train multiple decision trees Each tree makes a prediction
Final output = majority voting



Reinforcement Learning

What is Reinforcement Learning?

Reinforcement learning is a type of machine learning where an agent learns by interacting with an environment and receiving rewards or penalties



COMPARISON

Supervised Learning Algorithms	Unsupervised Learning Algorithms	Reinforcement Learning Algorithms
• Data provided is labeled data, with specified output values.	• Data provided is unlabeled data, the output is not specified, machine makes its own prediction.	• The machine learns from its environment using rewards & errors.
• Used to solve Regression and classification problems.	• Used to solve Association and Clustering problems.	• Used to solve Reward based problems.
• Labeled data is used.	• Unlabeled data is used.	• No predefined data is used.
• External Supervision.	• No Supervision.	• No Supervision.
• Solve problems by mapping labeled input to known output	• Solves problems by understanding patterns and discovering output.	• Follows Trail and Error problem solving Approach.

Introduction to Neural Networks (ANN)

Dr. Md. Zulfiker Mahmud
Professor, Department of CSE, Jagannath University
Email: zulfiker@cse.jnu.ac.bd

Deep Learning & AI
Artificial Neural Networks (ANNs)

$$y_1 = \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b) = \sigma(\sum_i w_i \cdot x_i + b)$$

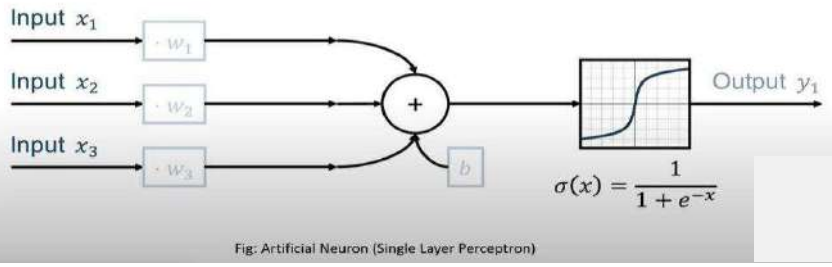


Fig: Artificial Neuron (Single Layer Perceptron)

Deep Learning & AI
Neuron in Human Brain

A human neuron is an incredibly complex biological cell with numerous structures like dendrites, an axon, the soma (cell body), and synaptic terminals. The human brain is estimated to contain approximately **86 billion** neurons.

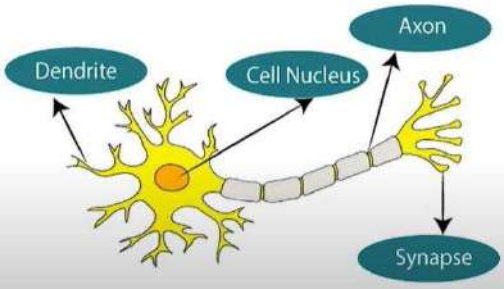


Fig: A Neuron

Deep Learning & AI

Artificial Neural Networks (ANNs)

- An artificial neuron is a simple computational model that includes input weights, an a
  an output.
- It processes information using mathematical functions, taking numeric inputs, multiplying them by weights, summing them up, adding a bias, and passing the result through an activation function.

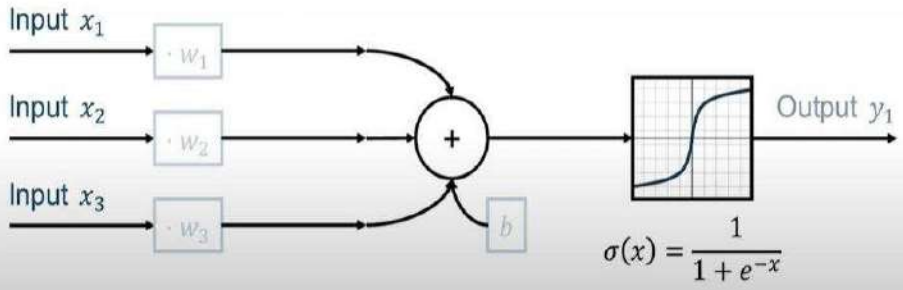
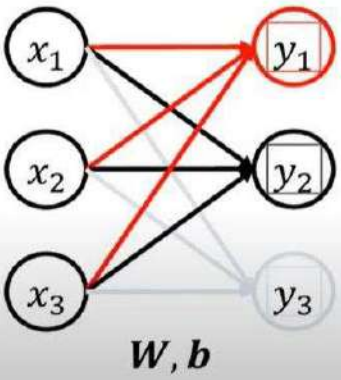


Fig: Artificial Neuron (Single Layer Perceptron)

Deep Learning & AI

Artificial Neural Networks (ANNs)

We can combine multiple perceptrons to create a layer.



We can thus rewrite the three computations as:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Or in a more simplified form:

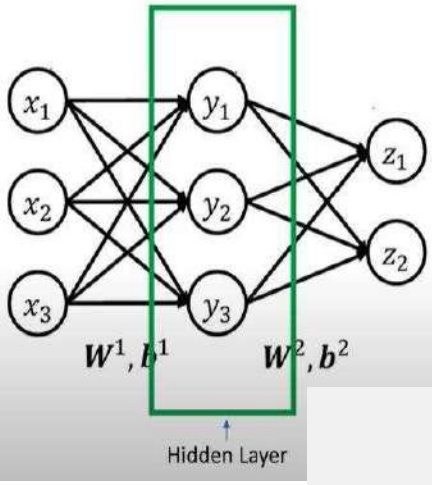
$$y = \sigma(W \cdot x + b)$$

Deep Learning & AI

Artificial Neural Networks (ANNs)

We call "hidden layer" any layer in between the input and the output layers.

- ❖ X: Input Layer
- ❖ Y: Hidden Layer
- ❖ Z: Output Layer



Deep Learning & AI

Artificial Neural Networks (ANNs)

We can chain multiple layers, with each output being the input of the next:

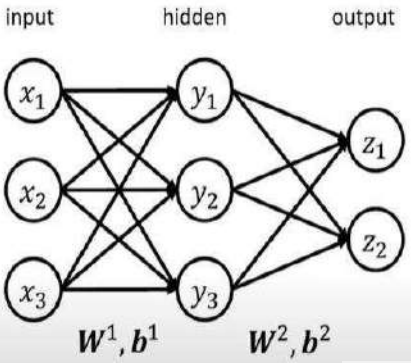
$$y = \sigma(W^1 \cdot x + b^1)$$

$$z = \sigma(W^2 \cdot y + b^2)$$

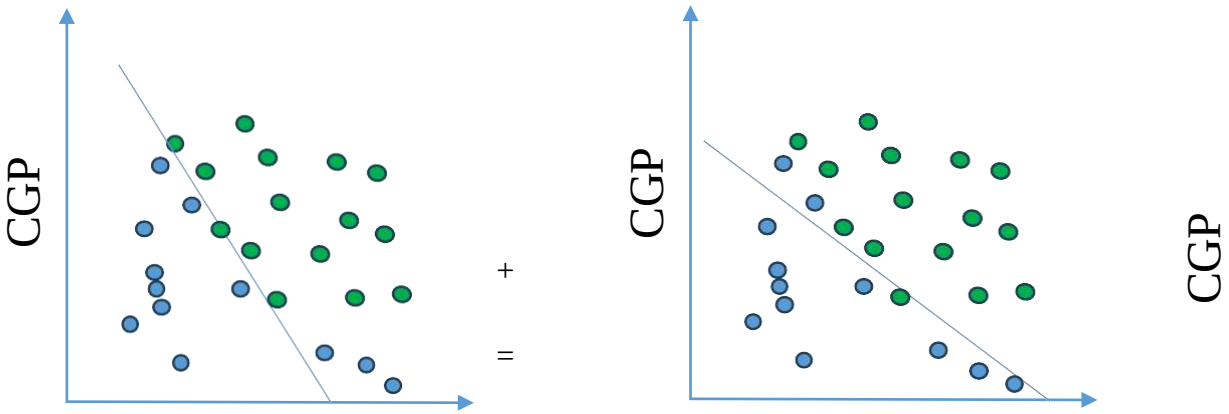
Combining these leads us to:

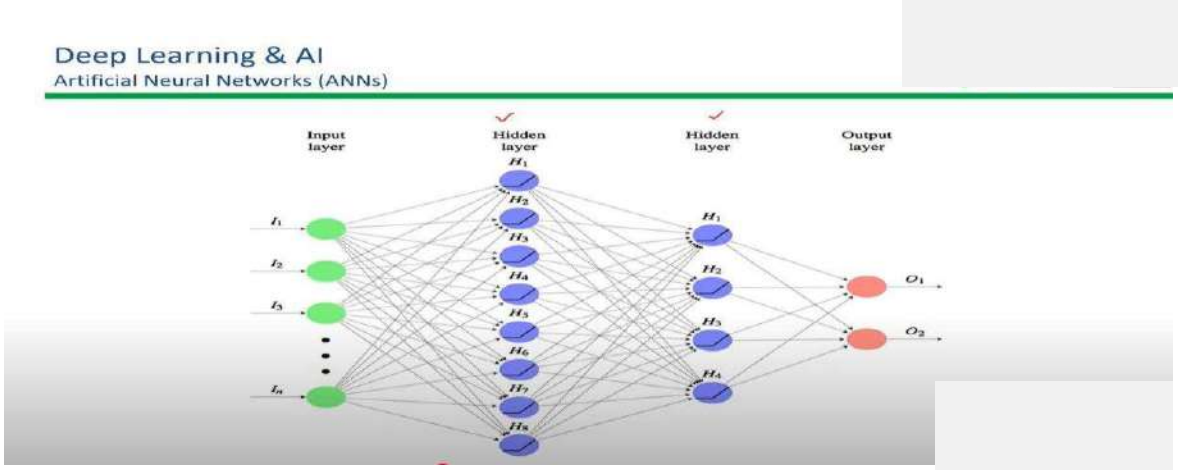
$$z = \sigma(W^2 \cdot \underbrace{\sigma(W^1 \cdot x + b^1)}_{y} + b^2)$$

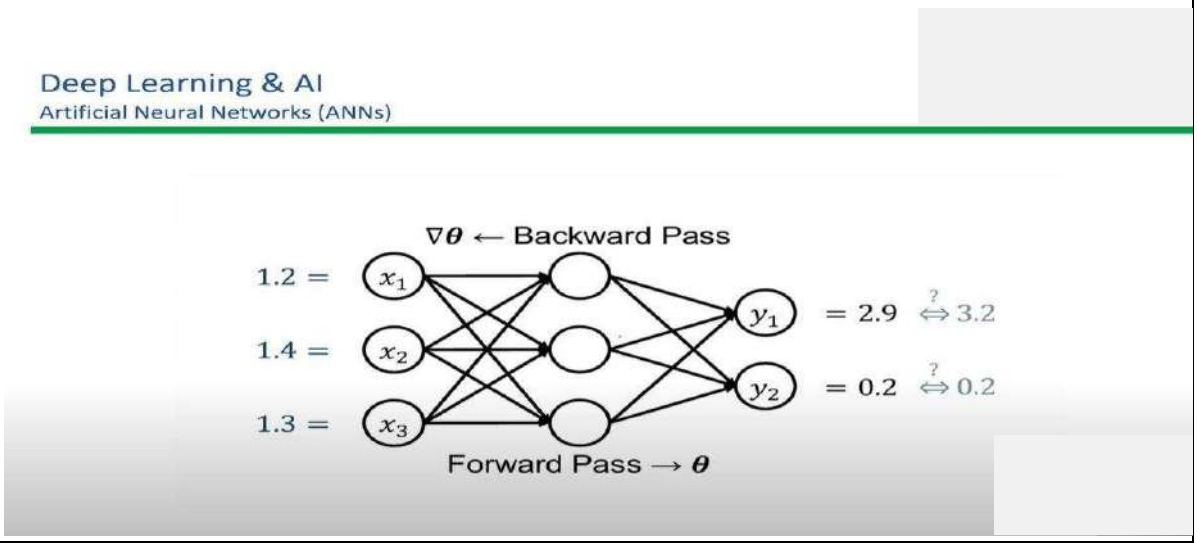
- Each layer has its own set of parameters (weights W^i and bias b^i)
- The underlying computation is a matrix multiplication described by $y^{i+1} = \sigma(W^i \cdot y^i + b^i)$



How ANN can capture non-linear pattern?







Forward Pass:

- The input values ($x_1 = 1.2, x_2 = 1.4, x_3 = 1.3$) are fed into the network.
- These inputs are processed by the hidden layer(s) through a series of weighted sums followed by activation functions (not shown in detail in the image).
- The resulting values are then passed to the output layer, producing the outputs ($y_1 = 2.9$ and $y_2 = 0.2$).

The forward pass is essentially the computation that happens when the network makes predictions. It starts from the input layer and propagates through the hidden layers and finally to the output layer.

Deep Learning & AI
Artificial Neural Networks (ANNs)

Backward Pass:

- This phase is about learning from errors and updating the weights (θ) in the network.
- The true target values for the outputs are compared with the predicted values from the forward pass (3.2 for y_1 and 0.2 for y_2).
- The differences between the predicted values and true values ($2.9 - 3.2 = -0.3$ for y_1 and $0.2 - 0.2 = 0$ for y_2) represent the errors.
- These errors are then used to calculate the gradients of the loss function with respect to the weights ($\nabla\theta$).

The backward pass involves calculating the gradient of the loss function with respect to each weight in the network (this gradient tells us how to change the weights to decrease the error). Then, the weights are updated typically using an optimization algorithm like gradient descent.

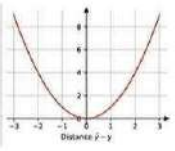
Deep Learning & AI
Artificial Neural Networks (ANNs)

The loss function is a comparison metric between the predicted outputs $\hat{y}_i$ and the expected outputs y_i.

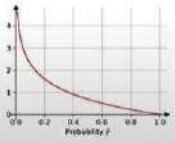
The choice of the loss function usually depends on the type of problem:

- For regression, a common metric is the mean squared error
- For classification, a common metric is the cross entropy

$$MSE(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$



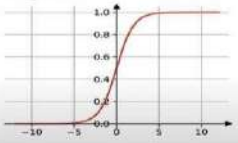
$$CE(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i} y_i \log(\hat{y}_i)$$



Activation Functions

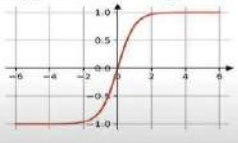
It is a mathematical function that takes the **weighted sum of the inputs** and **bias** as input and then generates an output, typically used to **add non-linearity** to the model.

Sigmoid



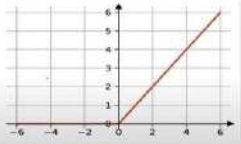
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Hyperbolic Tangent



$$\sigma(x) = \tanh(x)$$

Rectified Linear Unit



$$\sigma(x) = \max(x, 0)$$

Activation Functions

Why are activation functions important?

- **Non-Linearity:** Without non-linear activation functions, a neural network would essentially become a linear regression model, incapable of handling the complexities of most real-world data, which often are non-linear.
- **Control of Signal Flow:** Activation functions decide whether a neuron should be activated or not, effectively allowing the network to make complex decisions and learn from the data.
- **Learning Complex Patterns:** The introduction of non-linear activation functions allows the neural network to learn complex patterns in the data by stacking layers of neurons, each possibly using different activation functions.
- **Backpropagation:** Activation functions that are differentiable play a vital role in backpropagation, the training process for neural networks. The derivative of the activation function is used to update the weights of the neurons.

Activation Functions

Common types of activation functions:

- **Sigmoid or Logistic Function:** It squashes the input values into a range between **0 and 1**, which can represent probabilities (Binary class).
- **Tanh Function:** It scales the input values between **-1 and 1**, which is a zero-centered range, often leading to better training performance for multi-layer networks.
- **ReLU (Rectified Linear Unit):** It provides a linear response for positive inputs and **zero for negative inputs**, which allows for faster training and addresses the vanishing gradient problem to some extent.
- **Softmax:** Often used in the output layer for multi-class classification problems. The output of the softmax function for each class is in the range (0,1), and the **sum** of all the output probabilities for **all classes will be 1.**

Activation Functions

ReLU (Rectified Linear Unit):

- **Formula:** $\mathrm{ReLU}(x) = \max(0, x)$
- **Description:** ReLU is widely used because it helps mitigate the vanishing gradient problem, allowing models to learn faster and more effectively. It is simple and computationally efficient.

Activation Functions

ReLU (Rectified Linear Unit):

- **Formula:** $\mathrm{ReLU}(x) = \max(0, x)$
- **Description:** ReLU is widely used because it helps mitigate the vanishing gradient problem, allowing models to learn faster and more effectively. It is simple and computationally efficient.

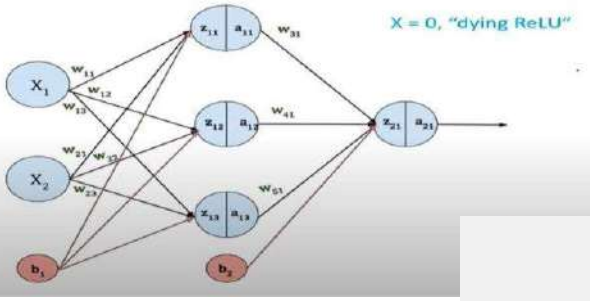
Variants: Due to some of the **limitations** of the standard ReLU, several variants have been proposed:

- **Leaky ReLU:** It allows a small, non-zero, constant gradient ϵ when the unit is not active and x is less than zero ($f(x) = \epsilon x$ for $x < 0$).
- **Parametric ReLU (PReLU):** It generalizes the leaky ReLU by allowing the gradient during the non-active phase to be learned during training.
- **Exponential Linear Unit (ELU):** It also allows for a small gradient when x is negative, but instead being linear, the negative part is an exponential decay.

Activation Functions

This means that the output is the input itself if the input is greater than zero, and zero if the input is zero or negative. Let's compute the ReLU activation for the following input values: $-3, -1, 0, 2, 4$.

1. **For** $x = -3$:
$f(-3) = \max(0, -3) = 0$
2. **For** $x = -1$:
$f(-1) = \max(0, -1) = 0$
3. **For** $x = 0$:
$f(0) = \max(0, 0) = 0$
4. **For** $x = 2$:
$f(2) = \max(0, 2) = 2$
5. **For** $x = 4$:
$f(4) = \max(0, 4) = 4$



Activation Functions

Leaky ReLU Function:

- **Function:** $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$
  - Here, α is a small positive coefficient, such as 0.01.
- **Output:** Positive inputs yield positive outputs (just like ReLU), but negative inputs result in small negative outputs instead of zeros because of the multiplication by the small coefficient α.

For example, if $\alpha = 0.01$ and $x = -5$, then:

- $f(x) = 0.01 \times -5 = -0.05$

Activation Functions

Key Differences

- Output Range:
  - **ReLU**: [0, ∞)
  - **Leaky ReLU**: (-∞, ∞), although negative values are very small
- Gradient:
  - **ReLU**: The gradient is 1 for positive inputs and 0 for negative inputs.
  - **Leaky ReLU**: The gradient is 1 for positive inputs and α (a small value) for negative inputs, which helps to keep information flowing through the network even when the input is negative.

Activation Functions

Tanh (Hyperbolic Tangent):

- **Formula**: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- **Description:** Tanh squashes the outputs to the range between -1 and 1, which can be more desirable than ReLU in certain cases, particularly if the model needs to handle negative outputs naturally.

- **Usage in Neural Networks:** It was particularly popular in the hidden layers of traditional neural networks and is still widely used in the **gates of LSTM** (Long Short-Term Memory) and **GRU** (Gated Recurrent Unit) cells in recurrent neural networks, where the regulation of information flow benefits from the symmetric properties of the function.
- **Disadvantages:** Vanishing Gradients Problem & More Computational Complexity

Activation Functions

Swish:

- **Formula:** $\text{Swish}(x) = x \cdot \sigma(x)$ → $f(x) = x \cdot \frac{1}{1+e^{-x}}$
- **Description**: A newer activation function that combines aspects of ReLU and sigmoid functions. It has been found to sometimes outperform ReLU in deeper networks.

Swish allows small negative values when the input is negative, it can maintain activation dynamics across a broader range of values, potentially capturing more complex patterns in the data.

Swish is computationally more expensive than 'ReLU' due to the calculation of the sigmoid function. This might be a consideration when designing systems where computational efficiency is critical.

Activation Functions

ELU (Exponential Linear Unit):

- **Formula:** $\mathrm{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$
- **Description:** ELU is similar to ReLU but includes a nonzero gradient for negative values, which helps reduce the vanishing gradient effect, leading to better performance and faster learning.

ELU: Outputs a smooth, non-linear response for negative inputs, with the output curving towards $-\alpha$ as x decreases.

Leaky ReLU: Provides a simple, linear response for negative inputs, proportionally scaled down by α.

Activation Functions

Sigmoid: The sigmoid function, also known as the logistic function, is a classical activation function used in neural networks, particularly in the output layer of binary classification models.

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

Disadvantages: Vanishing Gradient Problem, Non-zero-centered Output, Computational Complexity is High.

Activation Functions

Softmax Function: The Softmax function is defined as follows for a vector **z** of raw class scores from the final layer of a model:

$$\mathrm{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

where:

- z_i is the score for class i,
- e^{z_i} is the exponential function applied to z_i,
- the denominator is the sum of exponential scores for all classes in the vector $\mathbf{z}$.

Activation Functions

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Let's assume-
- Score for 1st Class: 2.0
- Score for 2nd Class: 1.0
- Score for 3rd Class: 0.5

- $e^{2.0} \approx 7.389$
- $e^{1.0} \approx 2.718$
- $e^{0.5} \approx 1.649$

Sum of exponentiated scores: $7.389 + 2.718 + 1.649 \approx 11.756$

Now, computing the probabilities:

- **Probability for Class 1:** $\frac{7.389}{11.756} \approx 0.628$
- **Probability for Class 2:** $\frac{2.718}{11.756} \approx 0.231$
- **Probability for Class 3:** $\frac{1.649}{11.756} \approx 0.140$

These probabilities sum to approximately 1.0, as expected.

Activation Functions

In regression problems where the goal is to predict a **continuous output** value, the activation function used in the output layer is typically a **linear activation function** or no activation function at all. The equation for the linear activation function, commonly used in the output layer of regression problems, is simply:

$$f(x) = x$$

Properties:

- **Input**: x (input value)
- **Output**: x (output is the same as the input)

This signifies that the output of the activation function is directly equal to the input value (x). It doesn't introduce any modification to the weighted sum of the inputs.

Activation Functions

Activation Function	Formula	Typical Usage
ReLU	$\text{ReLU}(x) = \max(0, x)$	Hidden layers (common for general purposes)
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	Output layer (binary classification)
Tanh	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	Hidden layers (range between -1 and 1)
Softmax	$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$	Output layer (multi-class classification)
Leaky ReLU	$\text{LeakyReLU}(x) = \max(\alpha x, x)$	Hidden layers (variation of ReLU)

Neural Networks

Epoch refers to one **complete pass** through the **entire training dataset.** During an epoch, the network will see every sample in the dataset once, allowing it to learn from the data by adjusting its weights based on the loss gradient.

- **Forward Pass:** Each input sample from the dataset is passed through the network to generate a prediction.
- **Loss Calculation:** The prediction is compared to the true value using a loss function, which quantifies the difference or error.
- **Backpropagation:** The gradient of the loss function is computed concerning each weight in the network. This involves applying the chain rule from calculus to find out how changes in weights affect the output error.
- **Weight Update:** The weights are then updated using an optimization algorithm, typically gradient descent or one of its variants (like Adam or RMSprop). The optimization algorithm adjusts the weights in a direction that minimizes the loss.
- **Repeat for All Batches:** Steps 1 to 4 are repeated for each batch in the training dataset until the entire been processed.

Neural Networks

If you train the neural network for 10 epochs, with each epoch updating the weights 10 times (as per your setup of 1000 samples and a batch size of 100), the total number of weight updates would be:

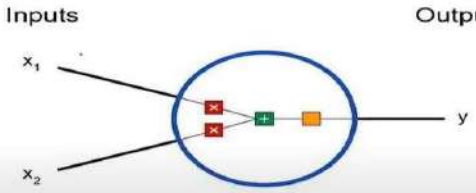
Total Updates = Updates per Epoch × Number of Epochs = 10×10 = 100 Times.

Thus, the weights would be updated a total of 100 times over the course of 10 epochs.

How Does an Artificial Neuron Learn?

Neural Networks

First, we must talk about neurons, the basic unit of a neural network. A neuron takes inputs, does some math with them and produces one output. Here's what a 2-input neuron looks like:



In neural network training, the weights are typically initialized randomly and then adjusted durir process through backpropagation. Backpropagation is an iterative algorithm that updates the we network based on the error between the predicted output and the actual output. By minimizing network can learn to make more accurate predictions.

Batch Size:

The **epochs=10** means that the neural network will train on the entire training dataset for **10 iterations**. During each epoch, the neural network will update its weights multiple times using backpropagation and stochastic gradient descent (or other optimization algorithms) until it has seen all the training examples. The number of weight updates during an epoch depends on the **batch_size**, which is another hyperparameter that determines how many samples are used to update the weights in each iteration.

For example: if we have a training dataset of 1000 samples and set the batch size to 100, the neural network will update its weights 10 times during an epoch (since 1000/100 = 10). During each weight update, the neural network will calculate the gradient of the loss function concerning the weights and use this gradient to adjust the weights in the direction that reduces the loss.

After 10 epochs, the neural network will have updated its weights 10 times on the entire training datase
hopefully learned to make accurate predictions on new data.

- Assume we have a 2-input neuron that uses the sigmoid activation function and has the following parameters:
  - w = [0, 1]
  - b = 0.5
- Now, let's give the neuron an input of x = [2, 3]. We'll use the dot product to write things more concisely:
  - (w.x) + b = ((w1*x1) + (w2*x2)) + b
    - = (0*2 + 1*3 + 0.5)
    - = 3.5

So, y = f ((x1*w1)+(x2*w2)+b)
=f(3.5)
~0.97

Note: $f(z) = \frac{1}{1+e^{-z}}$

$$f(3.5) = \frac{1}{1+e^{-3.5}} = \frac{1}{1+e^{-3.5}} \approx \frac{1}{1+0.0302} \approx \frac{1}{1.0302} \approx 0.9707$$

The neuron outputs given the inputs x=[2,3]. This process of passing inputs forward to get output is known as feedforward. That's it!

- Assume we have a 2-input neuron that uses the sigmoid activation function and has the following parameters:
  - w = [0, 1]
  - b = 0.5
- Now, let's give the neuron an input of x = [2, 3]. We'll use the dot product to write things more concisely:
  - (w.x) + b = ((w1*x1) + (w2*x2)) + b
    - = (0*2 + 1*3 + 0.5)
    - = 3.5

So, y = f ((x1*w1)+(x2*w2)+b) #sigmoid
=f(3.5)
~0.97
~1

The neuron outputs given the inputs x=[2,3]. This process of passing inputs forward to get output is known as feedforward. That's it!

Image/Data Classification using RNN & CNN

Dr. Md. Zulfiker Mahmud
Professor, Department of CSE, Jagannath University
Email: zulfiker@cse.jnu.ac.bd

Convolution Neural Network (CNN)

Computer Vision Fundamentals

Computer vision is a field of artificial intelligence that enables computers and systems to derive meaningful information from **digital images**, **videos**, and **other visual inputs**, and to act or make recommendations based on that information.

The key aspects of computer vision, summarized as main points:

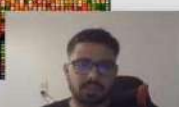
- **Image Recognition:** Identifying objects, people, and other elements within images.
- **Object Detection:** Recognizing and locating objects within an image using bounding boxes or other markers.
- **Image Segmentation:** Dividing an image into parts to simplify the analysis, often used in applications like medical imaging.
- **Pattern Recognition:** Recognizing patterns in visual data, such as shapes or movements.
- **Scene Reconstruction:** Reconstructing a 3D scene from images, used in augmented reality and robotics.
- **Video Tracking:** Tracking objects or individuals across a video sequence.
- **Image Restoration:** Restoring or enhancing the quality of degraded images.

Image Concepts

Image: In deep learning and computer vision, an image is a **digital representation of visual information**—like a photograph or a frame from a video—formatted in a way that can be processed by algorithms. The total number of pixels that represent the image is 1024x768 = 7,86,432. From **black at the lowest value (0)** to **white** at the **highest value (255)**, with shades of gray in between proportional to the value.

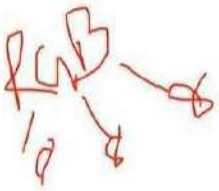
Image Size: 1024×768

Image Size: 1024×768



Pixel Concepts

A pixel, short for "picture element," is the **smallest unit of information in an image** or display. Each pixel represents a single point in the image. Pixels are arranged in a **grid pattern**. When viewed together at the proper distance, they form the complete image that our eyes perceive. An image with a resolution of 1920x1080 has 2,073,600 pixels.



24-bit Color (True Color):

- **Usage**: Most commonly used in JPEG images, web graphics, television screens, computer monitors, and smartphones.
- **Advantages**: Offers a good balance between color variety and file size, providing sufficient color depth for most practical applications without consuming as much storage or bandwidth as higher bit depths.

The term "True Color" is used to describe a color depth that allows for a representation of 24-bit color information. This configuration provides a total of 16,777,216 color variations. In True Color, each color component (Red, Green, and Blue) is allocated 8 bits, which allows each component 256 different intensity levels, leading to 256×256×256=16,777,216 possible colors.

Pixel Concepts

A pixel, short for "picture element," is the **smallest unit of information in an image** or display. Each pixel represents a single point in the image. Pixels are arranged in a **grid pattern**. When viewed together at the proper distance, they form the complete image that our eyes perceive. An image with a resolution of 1920x1080 has 2,073,600 pixels.

8-bit Grayscale:

For tasks where color information **might not add significant value**, converting images to **8-bit grayscale** can reduce computational requirements and streamline the learning process. This is common in tasks like text recognition, certain types of medical imaging, or when training efficiency is a priority.

Pixel Concepts

Color Representation:

- **8-bit Grayscale**: These images contain only shades of gray, meaning that each pixel carries information only about intensity or luminance, without any color data. The values range from 0 (black) to 255 (white), providing a total of 256 different shades of gray.

- **24-bit True Color**: True Color images use three color channels (Red, Green, and Blue), with each channel represented by 8 bits. This setup allows each channel to display 256 different intensity levels, combining to offer over 16 million possible colors (256 x 256 x 256).

Data Complexity and Size:

- **8-bit Grayscale**: In an 8-bit grayscale image, there is only one channel. These images are simpler and require less storage space and bandwidth because they consist of a single channel. An 8-bit grayscale image uses one byte per pixel.

- **24-bit True Color**: In a 24-bit color image, there is a total of three channels. These images are more c requiring three times the data for each pixel compared to grayscale images (since there are three cha using one byte). This makes them larger and more resource-intensive to process and store.

Pixel Concepts

1-bit (Monochrome):

- **Value Range:** 0-1
- **Colors:** Black and White (2 colors)

2-bit color:

- **Value Range:** 0-3
- **Colors:** 4 different colors or shades

3-bit color:

- **Value Range:** 0-7
- **Colors:** 8 different colors or shades

4-bit color:

- **Value Range:** 0-15
- **Colors:** 16 different colors or shades (often seen in early computer graphics)

Pixel Concepts

48-bit color (used in professional photography):

- **Value Range per Channel:** 0-65535
- **Total Colors:** This effectively allows trillions of different color nuances (important in high-end photography and printing).

64-bit color (HDR and advanced graphics):

- **Value Range per Channel:** 0-65535 for RGB and alpha
- **Total Colors:** This provides deep color with high dynamic range capabilities.

Working with Videos

- **Frame:** A "frame" in video and animation refers to a single still image in a sequence of images that constitute a video or animated content. When multiple frames are displayed rapidly, they create the illusion of motion.
- **Frame Rate:** It is measured in frames per second (FPS) and indicates the frequency at which these frames are displayed or projected.

Standard Frame Rates:

- **24 FPS:** Traditionally used in cinema. It provides a film-like motion blur that feels natural due to its closeness to how the human eye perceives movement.
- **30 FPS:** Common in television broadcasting in the NTSC (National Television System Committee) format. It offers slightly smoother motion compared to 24 FPS.
- **25 FPS:** Standard for television in countries using the PAL (Phase Alternating Line) and SECAM (Sequential Couleur Avec Memoire) formats.
- **60 FPS and higher:** Used in high-definition TV, live sports broadcasting, video games, and we streaming to produce very smooth motion effects.

Working with Videos

1. Object Detection and Tracking:

- **Moderate to High FPS**: For tasks like real-time object tracking in video feeds or sports analytics, a higher frame rate (30-60 FPS) can be beneficial. It provides more temporal information and smoother transitions, which can improve the accuracy and robustness of tracking algorithms.

2. Action Recognition:

- **Variable FPS**: The necessary frame rate can vary depending on the speed of the actions being analyzed. Faster actions might require higher frame rates to capture relevant motion details effectively. Typically, 30 FPS is a good starting point, but more dynamic scenes might benefit from up to 60 FPS.

Working with Videos

3. Behavior Analysis:

- **Lower to Moderate FPS**: For general behavior analysis, such as monitoring crowd movements or analyzing consumer behavior in stores, lower frame rates (15-30 FPS) might be sufficient. These applications often prioritize broader trends over split-second details.

4. Surveillance:

- **Lower FPS Sufficient**: In surveillance, especially in scenarios with less dynamic scenes, lower frame rates (5-15 FPS) are often adequate. This helps save on storage and processing while still capturing enough temporal information for activity detection and person identification.

5. Medical Imaging and Analysis:

- **Specific FPS Requirements**: Certain medical applications, such as analyzing heart rate or other quick physiological changes, may require very specific frame rates defined by the speed of the physiological events.

Convolutional Neural Network (CNN)

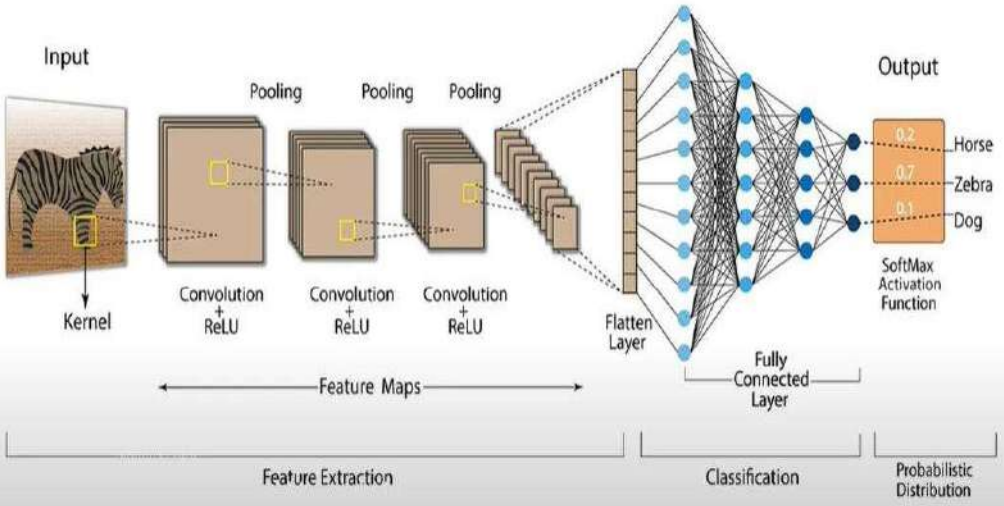


Fig: CNN Architecture

Convolutional Neural Network (CNN)

- **Input:** The process begins with an input image, which in this case is a **zebra.**
- **Padding**: Before applying the convolution operation, padding might be added around the border of the input image. Padding is used to **preserve the original size** of the image after convolution.
  - **Zero Padding (most common)**: Adds zeros around the border of the image.
  - **Reflective or Replication Padding**: Copies the edge values or reflects them around the border.
- **Convolution Layer:** To create feature maps, small matrices called **kernels** or filters are applied to the input image. These filters detect features such as edges and textures by sliding across the image. Each application of the filter results in a new feature map. (Next Slide)
- **ReLU Activation:** After convolution, the feature maps are passed through a Rectified Linear Unit (ReLU) activation function to introduce non-linearity, making the network capable of learning more complex patterns. It does this by turning all negative values to zero.
- **Pooling Layer:** After the ReLU activation, a pooling layer (typically max pooling) simplifies the output by reducing its dimensions but retaining the most important information. It selects the maximum value from a group of pixels in a feature map.

Convolutional Neural Network (CNN)

- **Flatten Layer:** After several convolution and pooling layers, high-level reasoning in the neural network occurs. The data is flattened into a single vector to prepare for the fully connected layer, which needs a flat input. (Next Slide)
- **Fully Connected Layer**: This layer is a deep neural network that uses the flattened data from the previous layers. Every neuron in this layer is connected to all the activations in the previous layer, and it's where the actual classification process begins to take shape.
- **Output Layer with SoftMax Function**: The last step in the process involves the SoftMax activation function, which is applied in the output layer. This function converts the output into a probability distribution, representing the likelihood of the input image belonging to one of the predefined classes (like horse, zebra, or dog).
- **Classification Output:** The output is a probabilistic distribution where each class (type of animal in this case) is assigned a probability score, indicating the confidence of the network in predicting each class.

More About Convolution Layer in CNN

Suppose, your input image has **three** channels (such as an RGB image), and you use a single filter, you **do not** get three separate feature maps for each channel. Instead, you get a **single feature map** from the convolution operation. This is how it works:

- **Multi-channel Filter:** Each filter in a convolution layer is three-dimensional when working with multi-channel images like RGB images. The depth of the filter matches the number of channels in the input image. So, for an RGB image, each filter has three layers—one for each channel (Red, Green, Blue).
- **Convolution Operation:** During the convolution operation, each layer of the filter is applied to the corresponding channel of the image. This means the first layer of the filter processes the Red channel, the second layer processes the Green channel, and the third layer processes the Blue channel.
- **Summation:** The results of these three convolutions (one per channel) are then summed up to produce a single output value. This summation results in a single feature map per filter, even though the filter is applied to multiple channels.

Convolutional Neural Network (Part-02)

Topics: Convolution & ReLU

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} + w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} + w_{31}x_{31} + w_{32}x_{32} + w_{33}x_{33}$$

$$O = \frac{M-F+2P}{S} + 1$$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

Input Image (5*5)

$*$

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

Kernel or Filter (3*3)

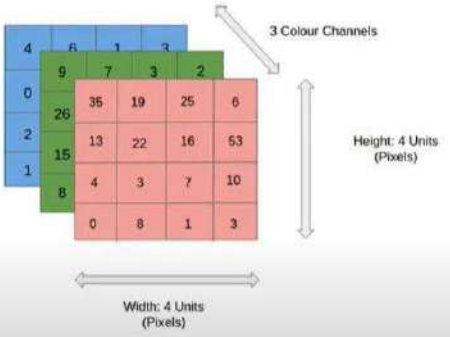
$=$

y_{11}		

Feature Map (3*3)

Input Layer

Fig: RGB Image

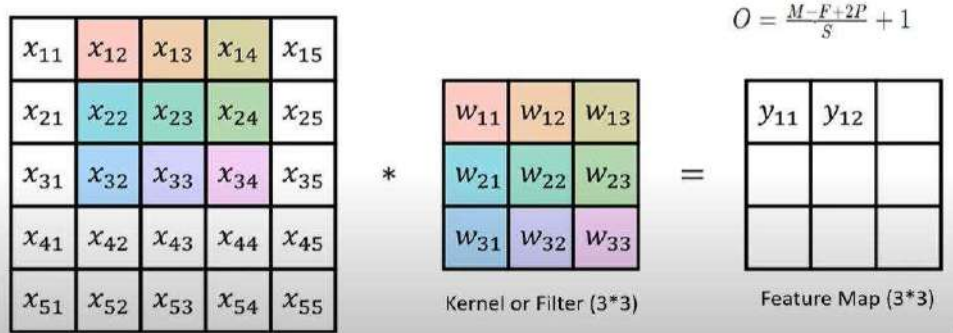


Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{13}x_{14} + w_{21}x_{22} + w_{22}x_{23} + w_{23}x_{24} + w_{31}x_{32} + w_{32}x_{33} + w_{33}x_{34}$$

$$O = \frac{M-F+2P}{S} + 1$$



Input Image (5*5)

Kernel or Filter (3*3)

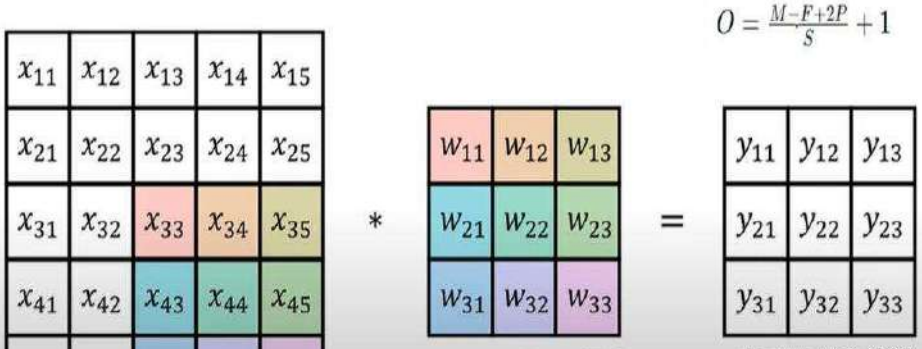
Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{33} = w_{11}x_{33} + w_{12}x_{34} + w_{13}x_{35} + w_{21}x_{43} + w_{22}x_{44} + w_{23}x_{45} + w_{31}x_{53} + w_{32}x_{54} + w_{33}x_{55}$$

$$O = \frac{M-F+2P}{S} + 1$$

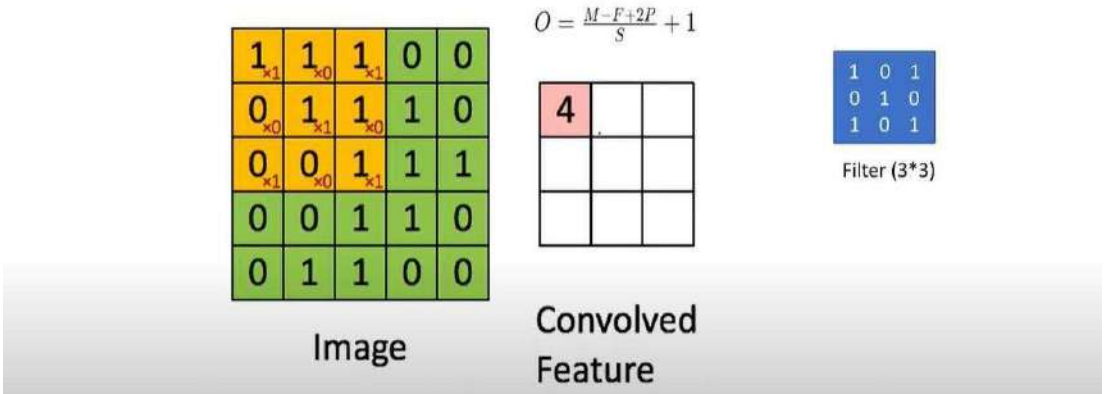


Input Image (5*5)

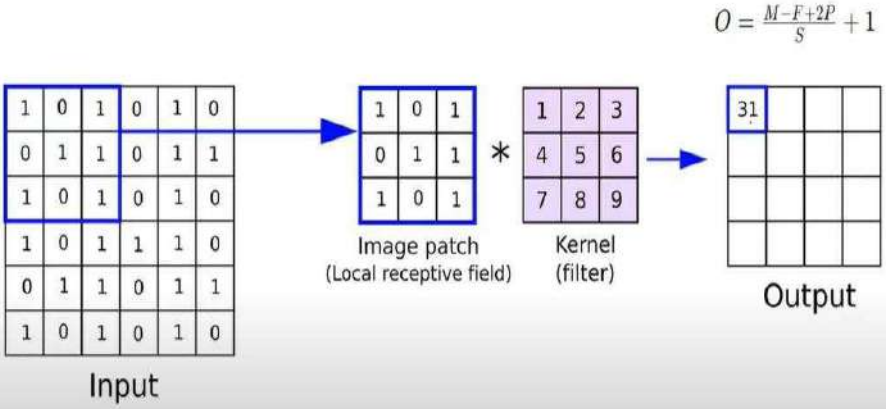
Kernel or Filter (3*3)

Feature Map (3*3)

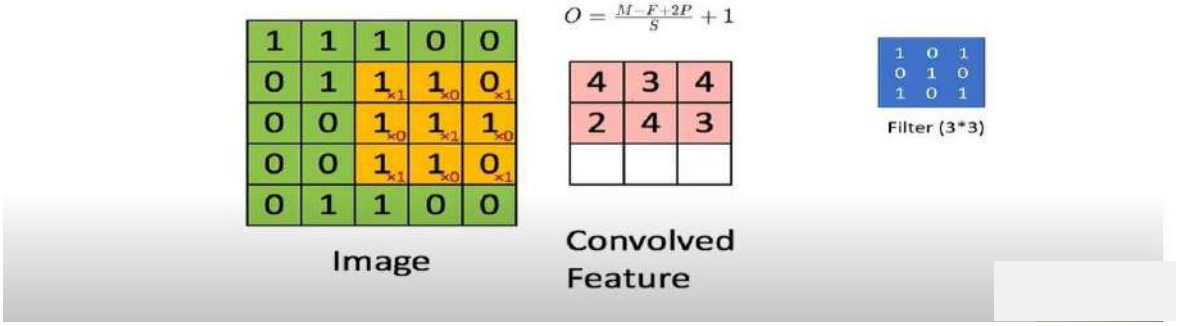
Convolutional Neural Network (CNN)



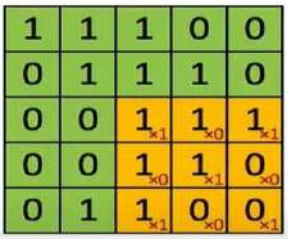
Convolutional Neural Network (CNN)



Convolutional Neural Network (CNN)



Convolutional Neural Network (CNN)



Image

$$O = \frac{M-F+2P}{S} + 1$$

4	3	4
2	4	3
2	3	4

Convolved Feature

1	0	1
0	1	0
1	0	1

Filter (3*3)

Convolutional Neural Network (CNN)

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

$$O = \frac{M-F+2P}{S} + 1$$

114	328	-26	470	158
53	266	-61		

Fig: Zero-Padding in CNN

Convolutional Neural Network (CNN)

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

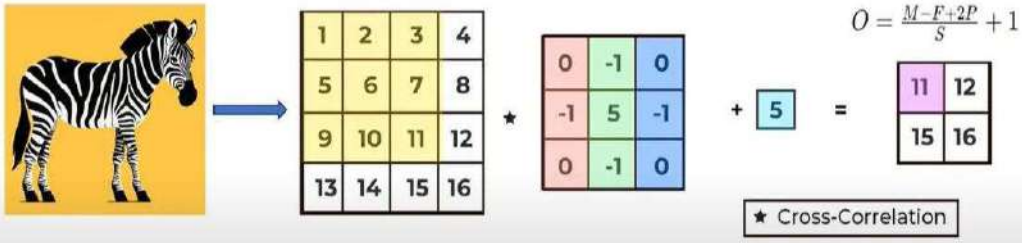
0	-1	0
-1	5	-1
0	-1	0

$$O = \frac{M-F+2P}{S} + 1$$

114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344

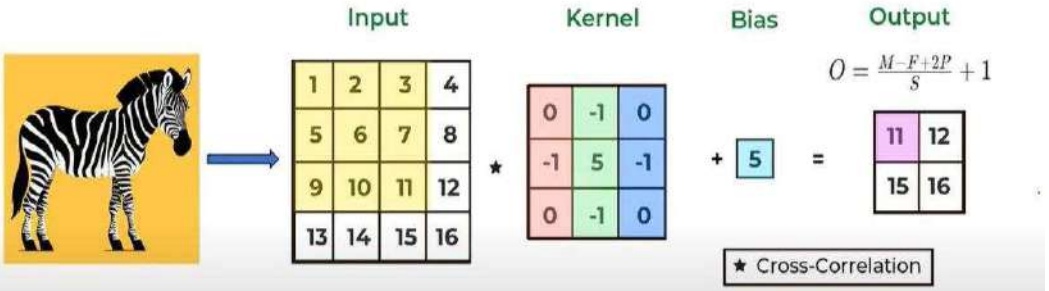
Fig: Zero-Padding in CNN

Convolutional Neural Network (CNN)



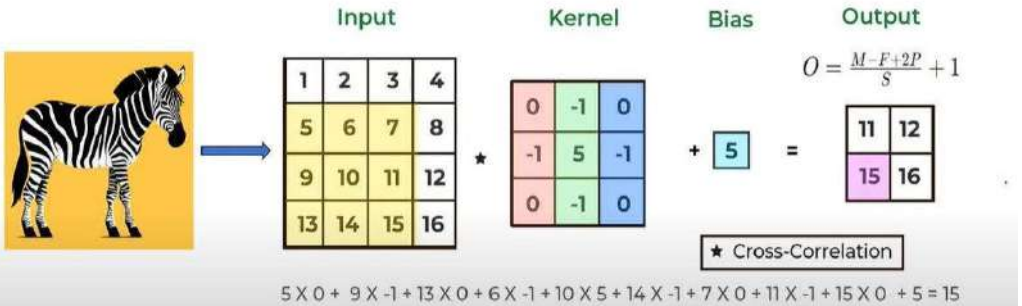
1 X 0 + 5 X -1 + 9 X 0 + 2 X -1 + 6 X 5 + 10 X -1 + 3 X 0 + 7 X -1 + 11 X 0 + 5 = 11

Convolutional Neural Network (CNN)

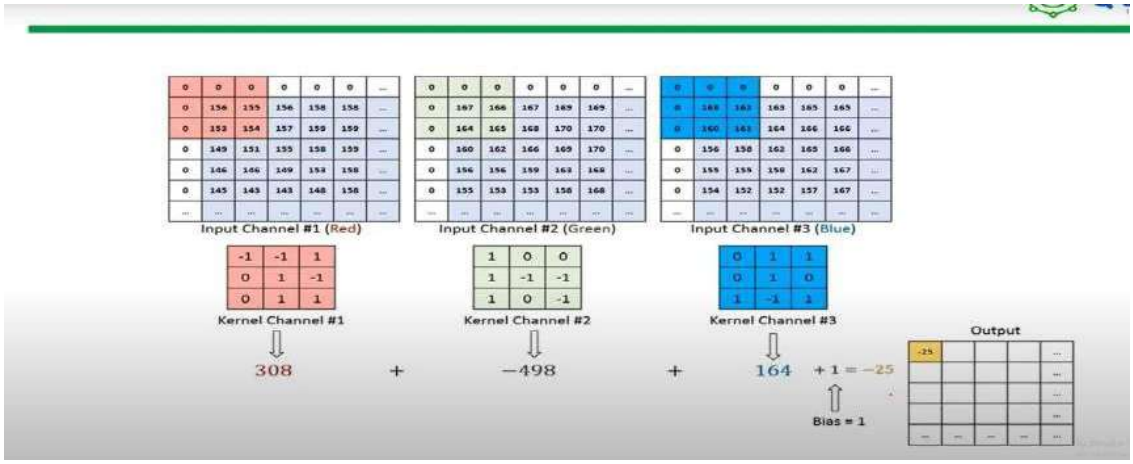


1 X 0 + 5 X -1 + 9 X 0 + 2 X -1 + 6 X 5 + 10 X -1 + 3 X 0 + 7 X -1 + 11 X 0 + 5 = 11

Convolutional Neural Network (CNN)



5 X 0 + 9 X -1 + 13 X 0 + 6 X -1 + 10 X 5 + 14 X -1 + 7 X 0 + 11 X -1 + 15 X 0 + 5 = 15



- **Prepare Input:** The input matrix includes pixel values or feature values arranged in a grid.
- **Position the Kernel:** The kernel (or filter) is a smaller matrix that you slide over the input matrix. Start at the **top-left** corner of the input and move the kernel over every valid position.
- **Perform Element-wise Multiplication:** For each position of the kernel, multiply each element of the kernel with the corresponding element of the input matrix that it covers.
- **Sum the Results:** After multiplying, sum up all the products obtained in the previous step to get a single output.
- **Adding Bias:** Add a bias term to the scalar value from the previous step. The bias is a single value that is learned during the training of the network. Adding a bias can help the model learn better by adjusting the output.
- **Generate Output Matrix:** The scalar result from each position of the kernel (after adding the bias) forms one element of the output matrix (or feature map).
- **Repeat for Multiple Filters:** If the CNN uses multiple filters, repeat steps 2-7 for each filter to produce multiple output matrices. Each filter can detect different features in the input, such as edges, textures, or other patterns.

Calculating Output Dimensions:

To find the dimensions of the output feature map (width and height), you can use the following formulas:

1. **Output Height (OH):**

$$OH = \left\lfloor \frac{H + 2P - F}{S} + 1 \right\rfloor$$

2. **Output Width (OW):**

$$OW = \left\lfloor \frac{W + 2P - F}{S} + 1 \right\rfloor$$

3. **Output Depth (OD):**
   - The output depth is equal to the number of filters K used in the convolutional layer.

Here,
W: The width of the input volume.
H: The height of the input volume.
D: The depth of the input volume (number of input channels).
K: The number of filters.
F: The spatial size (height and width) of the filter/kernel.
P: The amount of padding applied to the width and height of the input.
S: The stride, which is the step size the filter moves across the input.

Suppose you have an input volume of size 32×32×3 (width x height x depth), and you apply a convolutional layer with the following parameters:

- Filter size: 5×5
- Padding: 1 (applied to all sides)
- Stride: 1
- Number of filters: 10

Using the formulas, you would calculate the output dimensions as follows:

- **Output Height:**

$$OH = \left\lfloor \frac{32 + 2 \times 1 - 5}{1} + 1 \right\rfloor = 29$$

- **Output Height:**

$$OH = \left\lfloor \frac{32 + 2 \times 1 - 5}{1} + 1 \right\rfloor = 29$$

- **Output Width:**

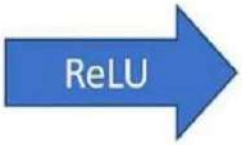
$$OW = \left\lfloor \frac{32 + 2 \times 1 - 5}{1} + 1 \right\rfloor = 29$$

- **Output Depth**: 10 (since there are 10 filters)

So, the output feature map would have dimensions 29×29×10.

1	14	-9	4
-2	-20	10	6
-3	3	11	1
2	54	-2	80

Fig: Feature Map (**Before** ReLU)



1	14	0	4
0	0	10	6
0	3	11	1
2	54	0	80

Fig: Feature Map (**After** ReLU)

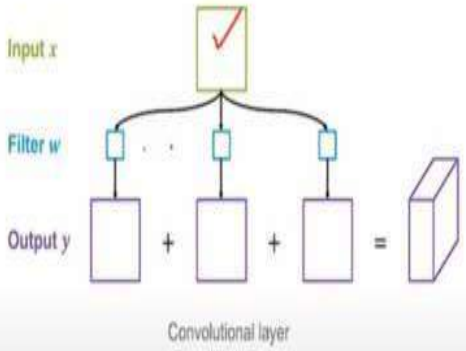
In a convolutional neural network (CNN), the pooling layer plays a crucial role in **reducing** the spatial dimensions (width and height) of the input volume for the subsequent layers.

The most common types of pooling are:

- **Max Pooling:** Outputs the maximum value from each patch of the feature map.
- **Average Pooling:** Outputs the average value from each patch.

In a convolutional layer, multiple filters can be used in parallel, which result in multiple convolutional operations in parallel on the same input.

The different output are concatenated to create a multi-channel feature map.



CNN: Pooling Layer

In a convolutional neural network (CNN), the pooling layer plays a crucial role in **reducing** the spatial dimensions (width and height) of the input volume for the subsequent layers.

Pooling layers in convolutional neural networks (CNNs) are important for three main reasons:

- **Efficiency:** They reduce the spatial dimensions of feature maps, lowering computational costs and the number of parameters.
- **Feature Invariance:** Pooling provides robustness to minor variations in the input, helping the network to recognize features regardless of their position.
- **Generalization:** By simplifying the features, pooling helps prevent overfitting, enhancing the model's ability to perform well on new, unseen data.

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

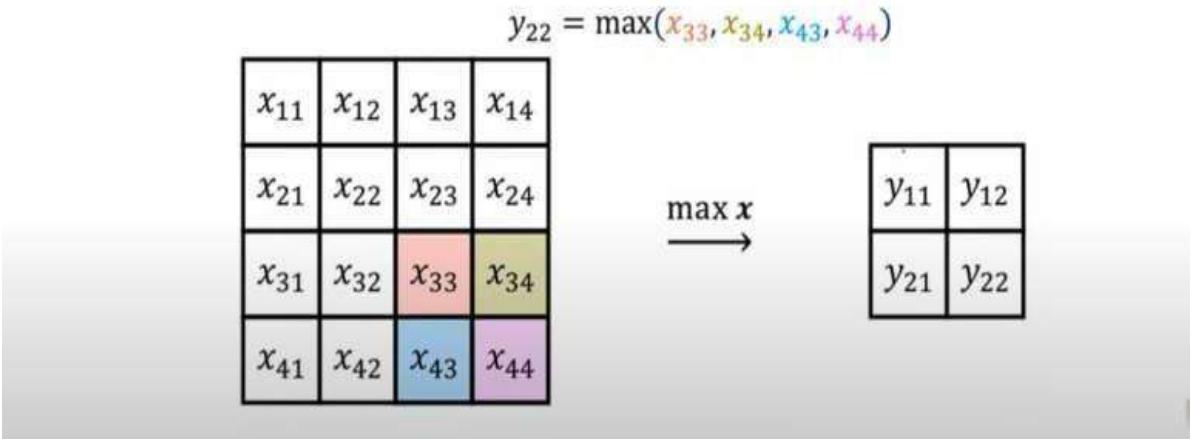
$$y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$$

x_{11}	x_{12}	x_{13}	x_{14}
x_{21}	x_{22}	x_{23}	x_{24}
x_{31}	x_{32}	x_{33}	x_{34}
x_{41}	x_{42}	x_{43}	x_{44}

$\xrightarrow{\max x}$

y_{11}	

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.



The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

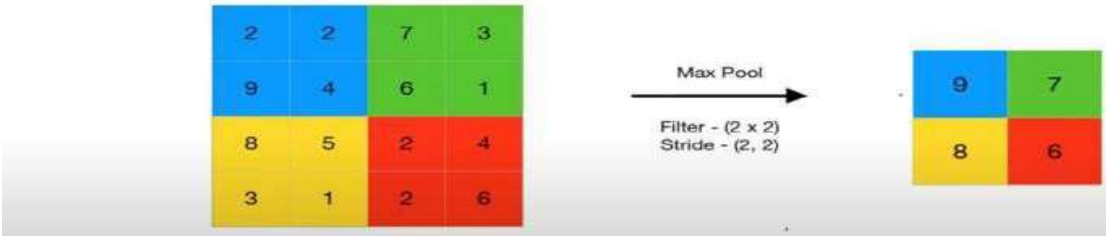


Fig: Max Pooling in CNN

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

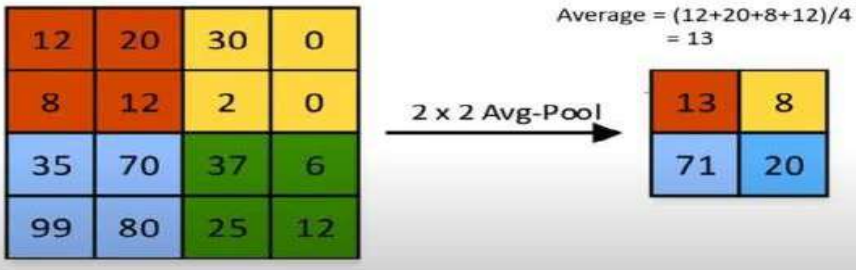


Fig: Average Pooling in CNN

More About Flatten Layer in CNN

Suppose, you will provide a **20x20** 2D data array to a **flatten layer**, it will convert this 2D array into a 1D vector as 400 individual elements.

How does it work when you use 100 neurons for a 400-element 1D input vector?

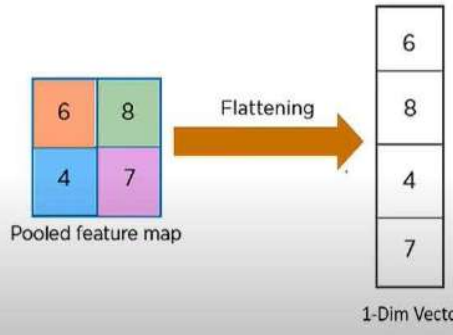
- **All Connections:** Each of the neurons is connected to every element of the input vector. For example, 100 neurons would each connect to all 400 elements from the input.
- **Weighted Sum and Bias:** Every connection has a weight. Each neuron computes a sum of the input elements multiplied by these weights, plus a bias term. This integrates information from all 400 points.

Mathematically: $\sum_{j=1}^{400} w_{ij} \cdot x_j + b_i$, where b_i is the bias for neuron i.

- **Activation Function:** This sum is then processed through an activation function, adding non-linearity and allowing the network to learn complex patterns.
- **The output of the Layer:** The output is a new vector, the size of which equals the number of neurons (for example 100). This output represents a condensed form of the input, capturing essential features.

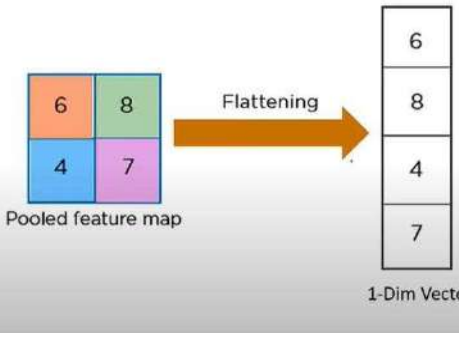
CNN: Flatten Layer

The flatten layer typically appears after the convolutional and pooling layers in convolutional neural network (CNN) architectures. The main **responsibility** of the flatten layer in a convolutional neural network (CNN) is to **reshape the multi-dimensional output** from the preceding convolutional or pooling layers into a **single, one-dimensional vector**.



CNN: Flatten Layer

The flatten layer typically appears after the convolutional and pooling layers in convolutional neural network (CNN) architectures. The main **responsibility** of the flatten layer in a convolutional neural network (CNN) is to **reshape the multi-dimensional output** from the preceding convolutional or pooling layers into a **single, one-dimensional vector**.



CNN: Dense Layer

Question: If a convolutional neural network has a flatten layer that receives a **10x10 dimensional feature map**, resulting in 100 flattened values, how are these values processed in the subsequent fully connected layer? Specifically, does each neuron in the next layer receive all 100 values, or does each value get assigned to a different neuron?

Answer: In a fully connected layer that follows a flatten layer in a convolutional neural network, **each neuron in the fully connected layer receives all 100 values** from the flattened 10x10 dimensional feature map. The fully connected layer operates such that each neuron is connected to every single input from the previous layer (in this case, the 100 flattened values).

CNN: Dimensions

Output dimensions describe the **size of the data tensor that results from a layer** within a CNN. This typically includes:

- **Width** and **Height**: These dimensions can change based on the type of layer (convolutional, pooling), the kernel size, the stride, and the padding used.
- **Depth** (or Channels): This dimension often changes in convolutional layers depending on the number of filters used. It stays the same through pooling layers unless pooling is done separately across channels.

Question: Consider the input tensor of dimensions 10×10×10 where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What are the dimensions of the output tensor after each operation? Report the name of the operation and the output dimensions as width x height x channels. There are 6 operations in total.

1. 3×3 Convolution (40 channels) with stride 1 and padding 1

- **Kernel Size:** 3×3
- **Stride:** 1
- **Padding:** 1
- **Input Dimensions:** $10 \times 10 \times 10$

$$\text{Output Width} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Height} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Channels} = 40$$

Dimension: 10 x 10 x 40

2. ReLU Activation

- Does not change dimensions.
- **Output Dimensions:** $10 \times 10 \times 40$

3. 3×3 Max Pooling with stride 1 and padding 1

- **Kernel Size:** 3×3
- **Stride:** 1
- **Padding:** 1

$$\text{Output Width} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Height} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Channels} = 40 \text{ (Channel dimension remains unchanged in pooling)}$$

Dimension: 10 x 10 x 40

4. 3×3 Convolution (20 channels) with stride 1 and padding 1

- **Kernel Size:** 3×3
- **Stride:** 1
- **Padding:** 1
- **Input Dimensions:** $10 \times 10 \times 40$

$$\text{Output Width} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Height} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Channels} = 20$$

Dimension: 10 x 10 x 20

5. ReLU Activation

- Does not change dimensions.
- **Output Dimensions:** $10 \times 10 \times 20$

6. 2×2 Max Pooling with stride 2 and padding 1

- **Kernel Size:** 2×2
- **Stride:** 2
- **Padding:** 1

$$\text{Output Width} = \left\lfloor \frac{10-2+2\times1}{2} \right\rfloor + 1 = 6$$
$$\text{Output Height} = \left\lfloor \frac{10-2+2\times1}{2} \right\rfloor + 1 = 6$$
$$\text{Output Channels} = 20 \text{ (Channel dimension remains unchanged in pooling)}$$

Dimension: 6 x 6 x 20

$$\text{Output Dimension} = \left\lfloor \frac{\text{Input Dimension} - \text{Kernel Size} + 2 \times \text{Padding}}{\text{Stride}} \right\rfloor + 1$$

Dimensions After Each Operation:

1. **After 3×3 Convolution**: $10 \times 10 \times 40$
2. **After ReLU Activation**: $10 \times 10 \times 40$
3. **After 3×3 Max Pooling**: $10 \times 10 \times 40$
4. **After 3×3 Convolution**: $10 \times 10 \times 20$
5. **After ReLU Activation**: $10 \times 10 \times 20$
6. **After 2×2 Max Pooling**: $6 \times 6 \times 20$

CNN: Counting the Parameters

Parameters reflect the model's learning capacity and complexity. More parameters can mean a more powerful model, but also one that is more prone to overfitting and is computationally more expensive to train and run.

Note: In typical Convolutional Neural Network (CNN) architectures, the **convolutional layers** are primarily responsible for adding parameters, which are the learnable weights and biases of the model.

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. **3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.**
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What is the total parameter count?

$$\text{Number of Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels} + 1) \times \text{Output Channels}$$

First 3x3 Convolution (40 channels):

- **Kernel Height**: 3
- **Kernel Width**: 3
- **Input Channels**: 10 (initial input channels)
- **Output Channels**: 40

$$\text{Parameters} = (3 \times 3 \times 10 + 1) \times 40 = (90 + 1) \times 40 = 3640$$

$$\text{Number of Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels} + 1) \times \text{Output Channels}$$

Second 3x3 Convolution (20 channels):

- The input to this layer is the output of the first max pooling layer, which maintains 40 channels.
- **Output Channels**: 20

$$\text{Parameters} = (3 \times 3 \times 40 + 1) \times 20 = (360 + 1) \times 20 = 7220$$

Total Parameters

Now, to find the total number of parameters in the CNN:

$$\text{Total Parameters} = 3640(\text{First Conv}) + 7220(\text{Second Conv}) = 10860$$

Thus, the CNN has a total of 10,860 parameters, solely from the convolutional layers as pooling layers and ReLU activations do not add any learnable parameters.

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What is the total parameter count?

If you are adding another convolutional layer with 10 output channels following the previous convolutional layer that had 20 output channels, you would again use the formula to calculate the parameters. This layer also uses 3x3 kernels:

Parameters for the Third Convolutional Layer:

- **Kernel Size**: 3×3
- **Input Channels**: 20 (output of the second convolution layer)
- **Output Channels**: 10
- **Bias**: 1 bias per output channel (10 in total)

Formula and Calculation:

$$\text{Number of Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels} + 1) \times \text{Output Channels}$$
$$\text{Number of Parameters} = (3 \times 3 \times 20 + 1) \times 10$$
$$\text{Number of Parameters} = (180 + 1) \times 10$$
$$\text{Number of Parameters} = 181 \times 10$$
$$\text{Number of Parameters} = 1810$$

Computer Vision

Computer vision is a field of artificial intelligence (AI) and computer science that focuses on enabling computers to interpret and understand **visual information** from the world. By processing and analyzing **images and videos**, computer vision systems aim to automate tasks that the human visual system can do.

Here are the main fields of application for computer vision:

1. Autonomous Vehicles
2. Medical Imaging
3. Surveillance and Security
4. Retail and E-commerce
5. Robotics
6. Augmented Reality (AR) and Virtual Reality (VR)
7. Agriculture
8. Manufacturing

Working Area	Purpose and Main Goal	Common Methods
Image Classification	Assign a label to an entire image.	CNNs (ResNet, VGG, etc.)
Object Detection	Identify and locate objects within an image.	Faster R-CNN, YOLO, SSD
Semantic Segmentation	Classify each pixel in the image into a category.	FCN, DeepLab, PSPNet
Instance Segmentation	Identify and describe each object instance in the image.	Mask R-CNN, SOLO
Panoptic Segmentation	Unified segmentation of both background and foreground objects.	Panoptic FPN, Panoptic-DeepLab
Image Generation and Synthesis	Generate new images from scratch or transform images.	GANs, VAEs
Image Reconstruction	Restore images from degraded or incomplete forms.	Super-resolution, De DnCNN

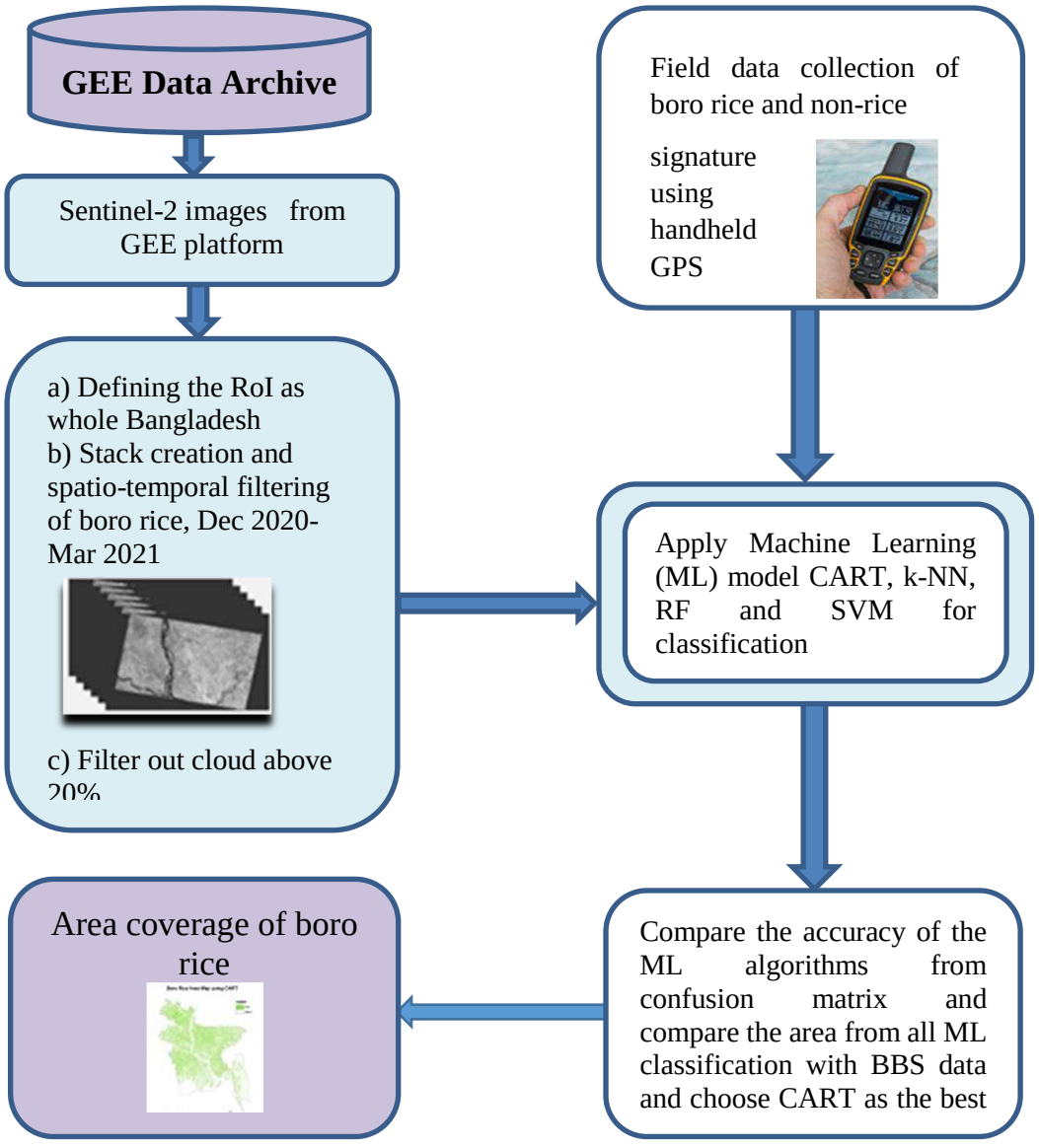
Working Area	Purpose and Main Goal	Common Methods
Image Retrieval	Find and retrieve images from a database based on a query.	Content-based image retrieval (CBIR), deep learning-based retrieval
Pose Estimation	Detect the pose or orientation of objects, often humans.	OpenPose, PoseNet
Action Recognition	Recognize and classify actions in videos or sequences of images.	3D CNNs, Two-Stream Networks
3D Computer Vision	Understand and interpret 3D structures from images or video.	Stereo vision, depth estimation, 3D reconstruction
Optical Character Recognition (OCR)	Recognize and extract text from images.	Tesseract, deep learning-based OCR
Visual Tracking	Track objects as they move across frames in a video.	KLT tracker, Siamese networks

Machine Learning-Based Estimation of Boro Rice Cultivated Area using Google Earth Engine (GEE)

Hasan Md. Hamidur Rahman, Director (Computer & GIS Unit)
Bangladesh Agricultural Research Council
Email: h.rahman@barc.gov.bd

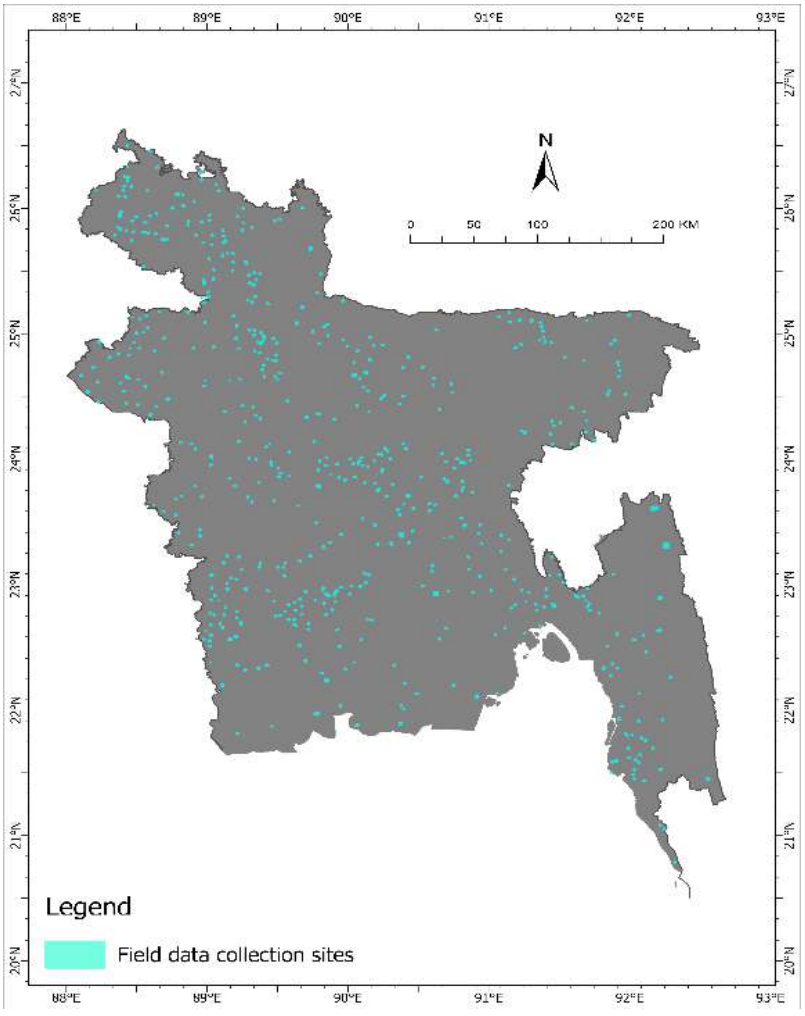
Steps for Boro rice area estimation

Flowchart of Boro rice area delineation:

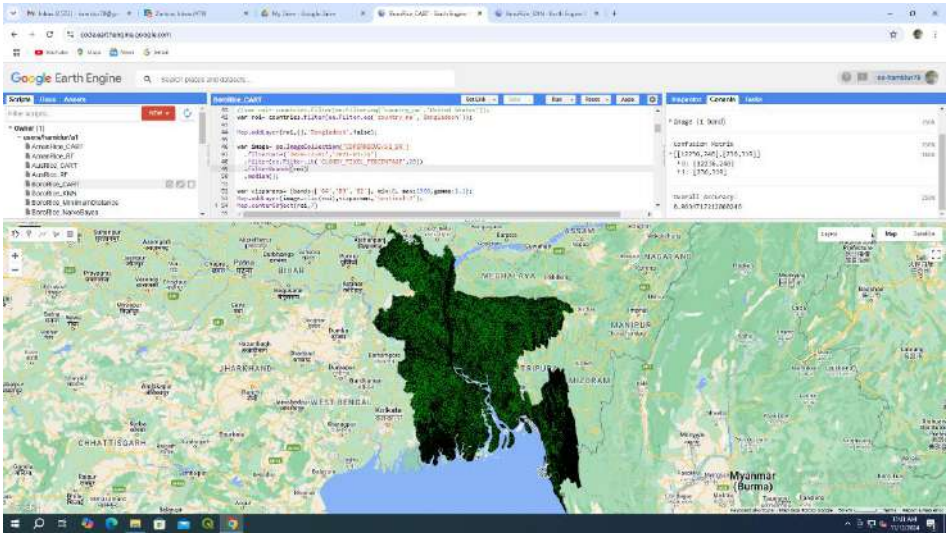


Field/ Signature data collection and processing

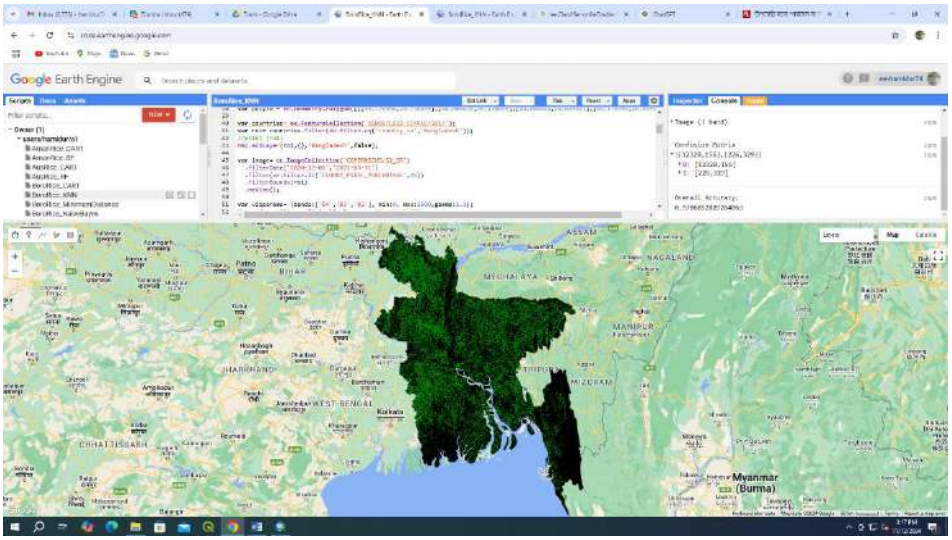
1. Field Data Collection through GPS device
2. The field data collected during boro seasons are compiled into an Excel sheet. The validated data is then saved as a CSV file, which serves as the attribute file.
3. Simultaneously, the GPX file from the handheld GPS devices is downloaded and imported into Google Earth Pro to verify the survey locations.
4. Polygons are drawn using the GPS points and digital photos for most of the fields.
5. The file is exported in KML format and opened in QGIS, where it is converted into a shapefile.
6. The shapefile is then joined with the attribute file (CSV) to merge the land types and other parameters.
7. Non-rice land types are generalized into a single class called 'non-rice'. In the shapefile, there are two columns named 'id' and 'feature_name', where the value '0' represents 'non-rice' and '1' represents ‘rice’ features.
8. The generalized shapefile is imported into the Google Earth Engine (GEE) asset, where it is used as the signature data in the machine learning (ML) model.



Screen shot of boro rice classified GEE raster data (CART)



Screen shot of boro rice classified GEE raster data (k-NN)



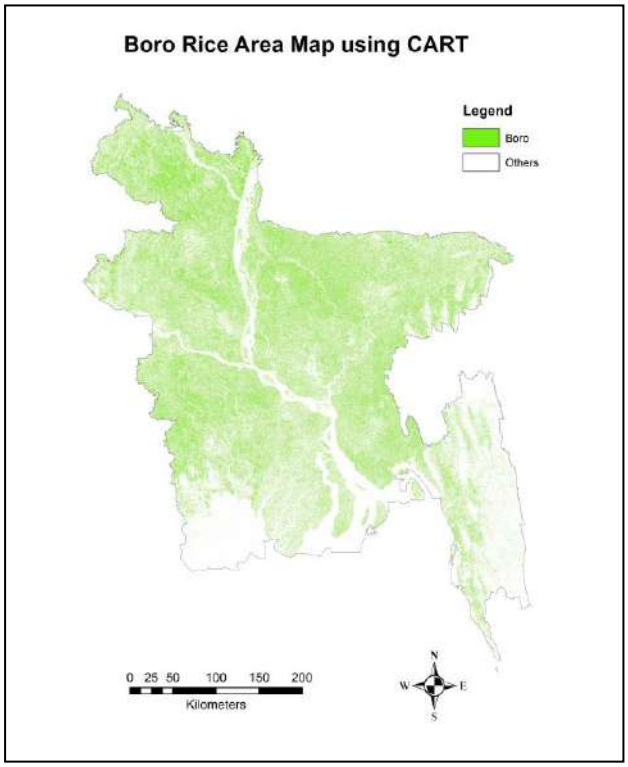
Steps to process data in ArcGIS

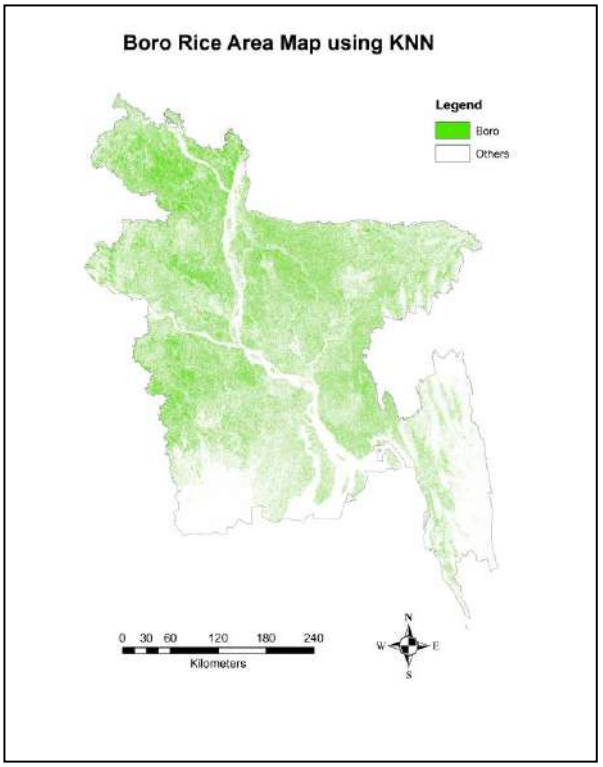
1. Boro Classified GEE Raster Data
2. Boro Reclassified Mosaiced Raster Data
3. Resampled Raster Data
4. Boro Final Area Result.

ArcGIS Tool for Raster data processing:

- Arc Tool Box-> Spatial Analysis Tools->Reclass->Reclassify
- Arc Tool Box-> Data Management Tools->Raster->Raster Dataset->Mosaic to New Raster
- Arc Tool Box-> Data Management Tools->Raster->Raster Processing->Resample

Boro rice area map





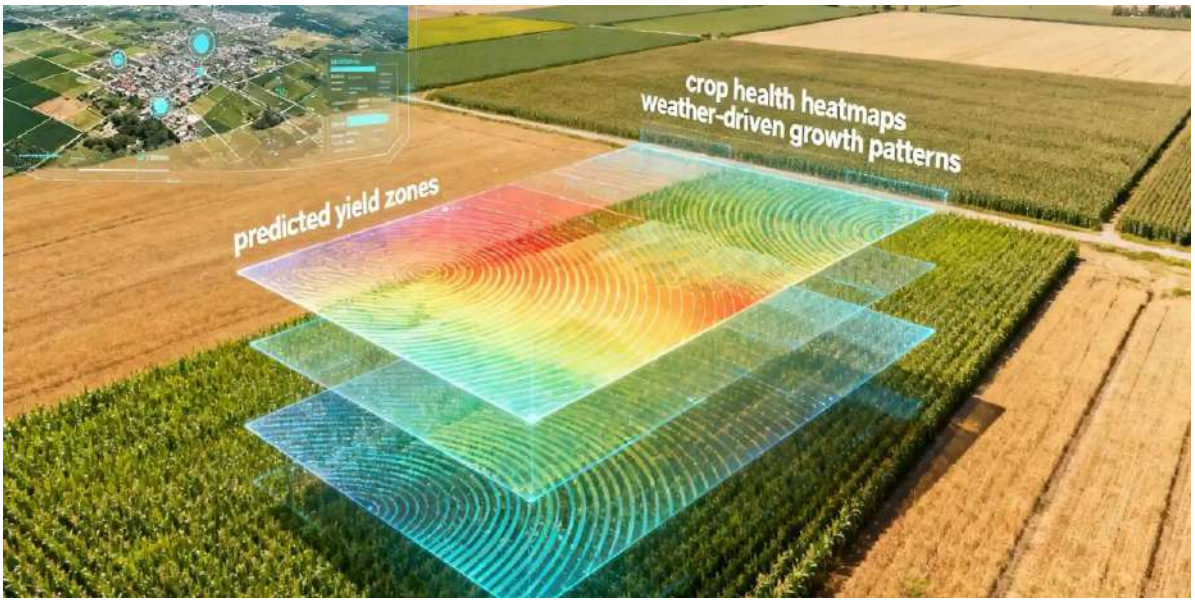
Boro rice area comparison

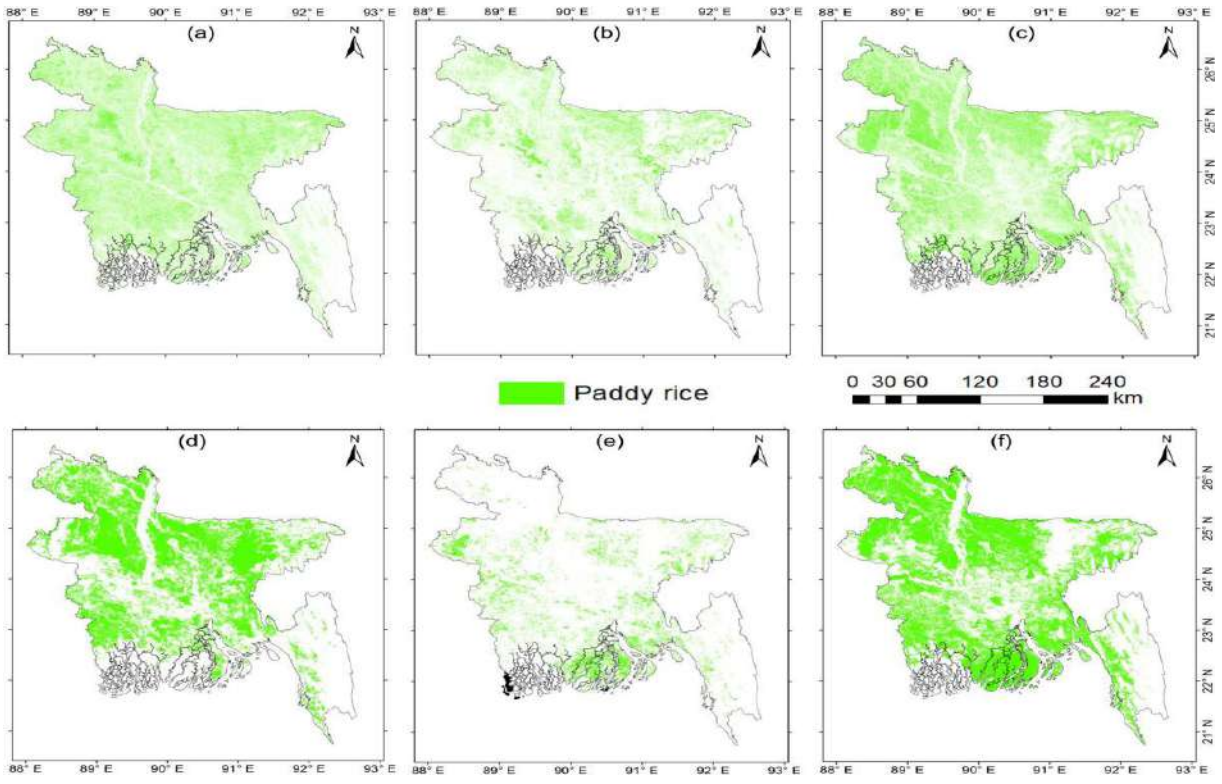
Sl.	Classification Algorithm	Pixel count (10m x 10m)	Area in Sq. Meter	Area in Sq. km	Area in Ha	Classification Accuracy of Algorithm (%)	Area Accuracy with respect to BBS (%)
1	**CART**	419957945	41995794500	41996	**4,199,579**	**96**	**88**
2	**k-NN**	384058603	38405860300	38406	**3,840,586**	97	**80**
				BBS Statistics 2020-21:	**4,786,621**	-	-

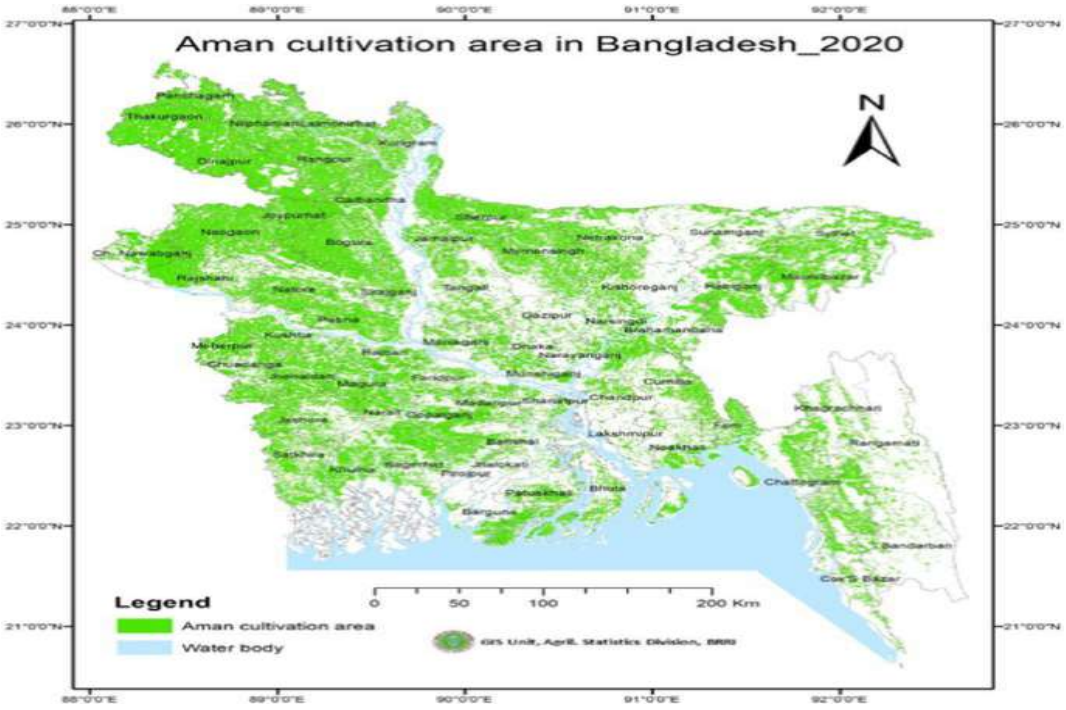
Future of AI & ML in Agriculture (Bangladesh)

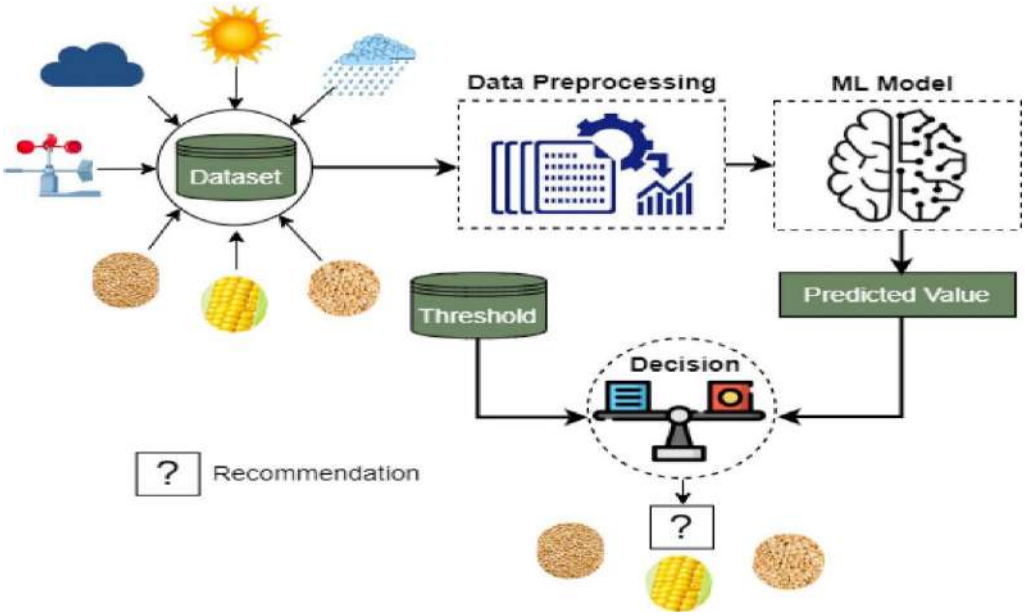
Dr. Md. Golam Mahboob
Chief Scientific Officer (Forestry Unit)
Bangladesh Agricultural Research Council
Email: golam.mahboob@barc.gov.bd

1. Smart Crop Prediction & Yield Optimization









What will happen:

- AI models will predict:
  - Best crop for a district
  - Expected yield
  - Risk of failure

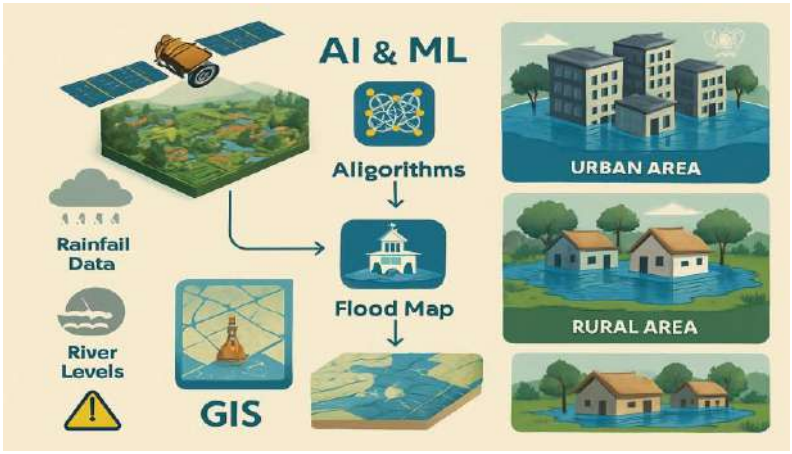
Bangladesh Context:

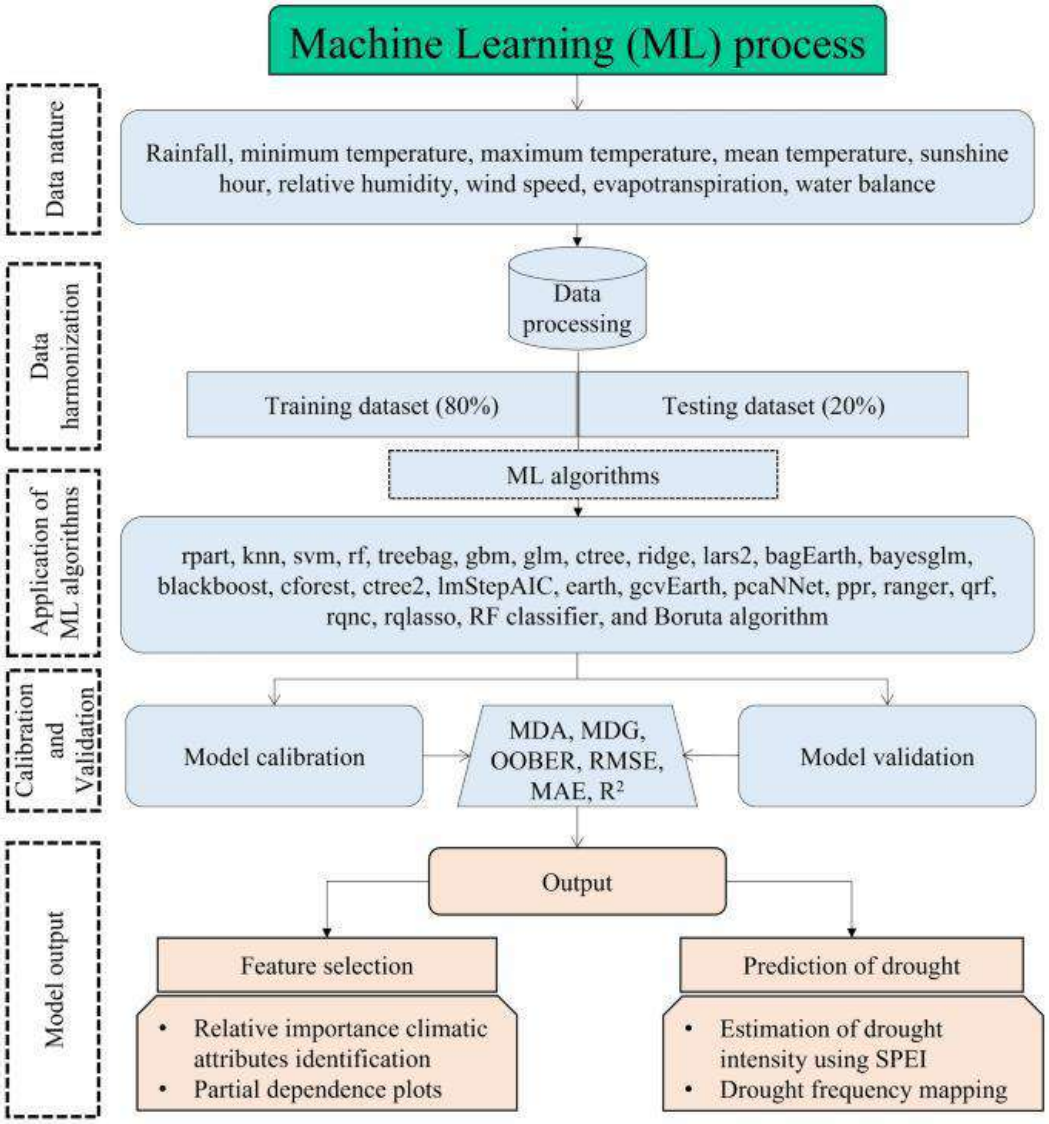
- With your PET + climate dataset (1961–2022), this is already possible
- BARC, DAE can build **district-wise crop suitability maps**

☞ Example:

- Rajshahi → Wheat (low rainfall)
- Sylhet → Rice (high rainfall)

🌦 2. Climate-Smart Agriculture





Future Applications:

- Flood prediction (using rainfall + river data)
- Drought early warning
- Seasonal crop advisory

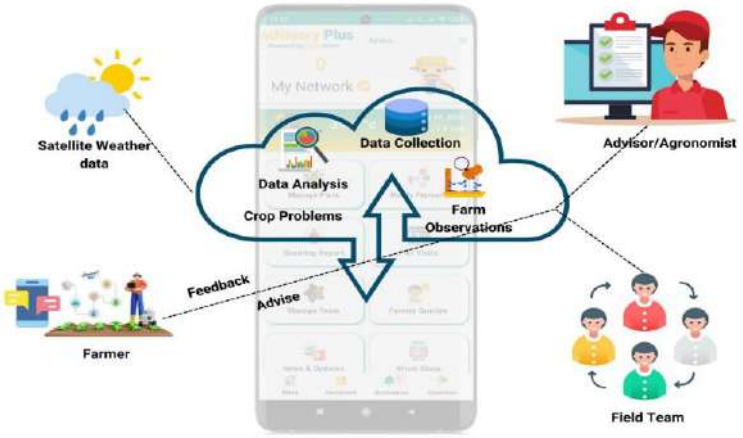
Why Important:

Bangladesh faces:

- Floods
- Cyclones
- Salinity intrusion

☞ AI can reduce crop loss significantly.

📱 3. AI-powered Apps



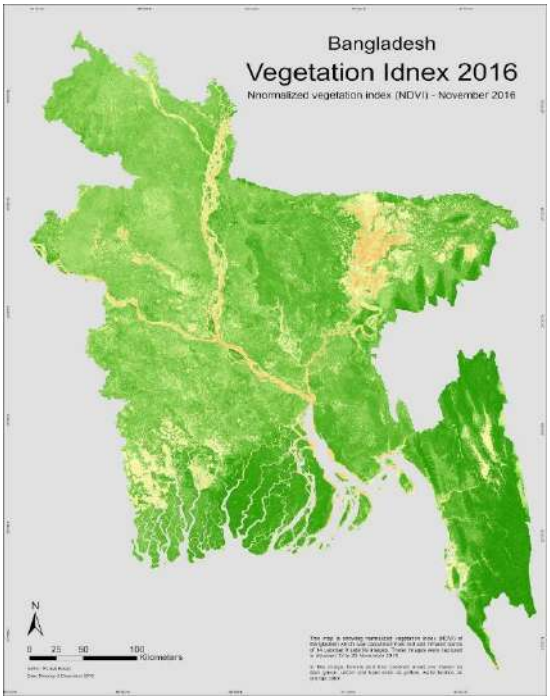
What will evolve:

- Farmers take a photo → AI detects disease
- Voice-based advisory in Bangla
- Personalized recommendations

Example Features:

- "When should I irrigate?"
- "Which fertilizer to use?"
- "Why is my leaf yellow?"

🛰️ 4. GIS + Remote Sensing



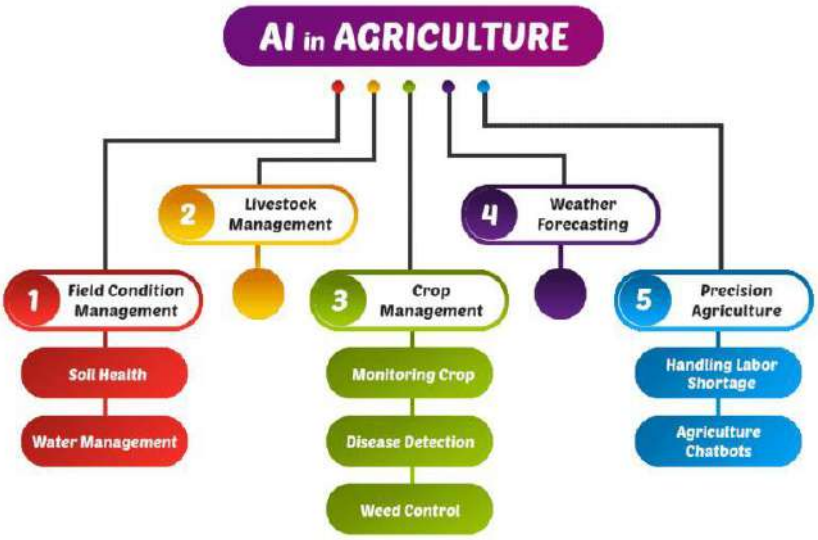
Future but slower in Bangladesh:

- Smart irrigation systems
- Drone spraying
- Sensor-based farming (IoT + ML)

Challenge:

- Small land size
- Cost constraints

6. Data-Driven Policy Making



Light Mode

Government Use:

- Predict national crop production
- Import/export planning
- Food security analysis

Conclusion

The integration of Machine Learning (ML) techniques in agriculture holds immense potential to transform the way we produce, manage, and forecast crop outcomes. This five-day training has introduced participants to essential ML concepts, data analysis tools, and real-world agricultural applications using Python and R programming. From foundational knowledge in data science and exploratory data analysis to hands-on experience with classification, regression, clustering, and ensemble learning methods such as Random Forests, participants have gained practical skills necessary for implementing intelligent decision-making systems in agriculture. The inclusion of neural networks and Google Earth Engine (GEE) has further empowered participants to explore advanced solutions such as image-based crop health detection and large-scale land monitoring. It is hoped that this training will not only strengthen participants' technical capabilities but also inspire them to apply these tools to address agricultural challenges in Bangladesh. By leveraging data-driven approaches, we can enhance crop productivity, ensure sustainability, and support policy-making that benefits farmers and the agricultural ecosystem.

The journey does not end here—continuous learning, experimentation, and collaboration will be key to driving meaningful change in the field of agricultural informatics.