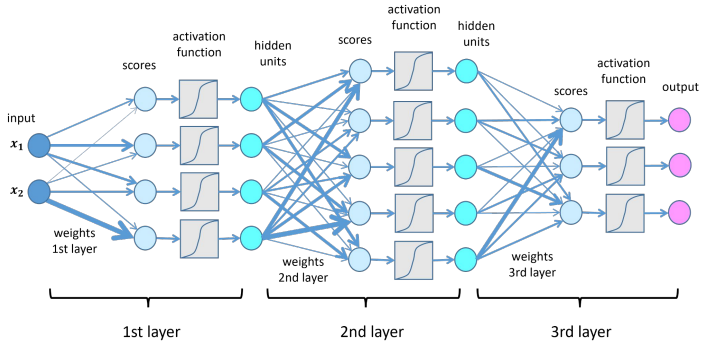
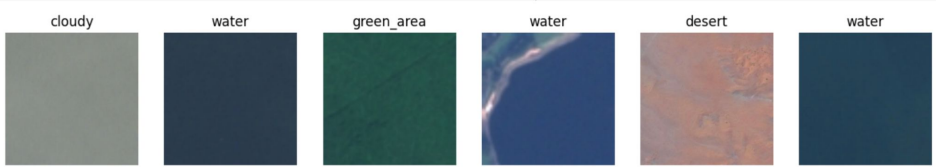
Basic AI in Computer Vision for Satellite Classification

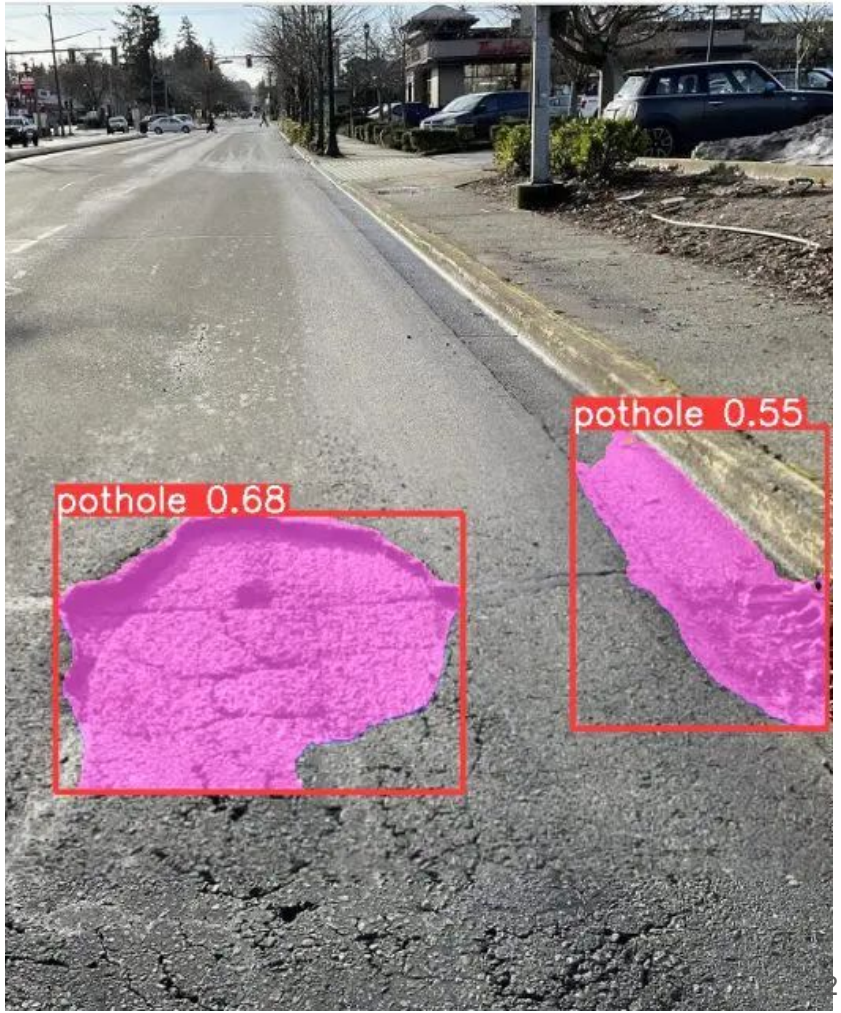
Teerapong Panboonyuen, Ph.D.
teerapong.pa@chula.ac.th

Outlines

- Basic AI
- Lab: Titanic Machine Learning
- Lab: Satellite Classification
- Lab: Image Segmentation

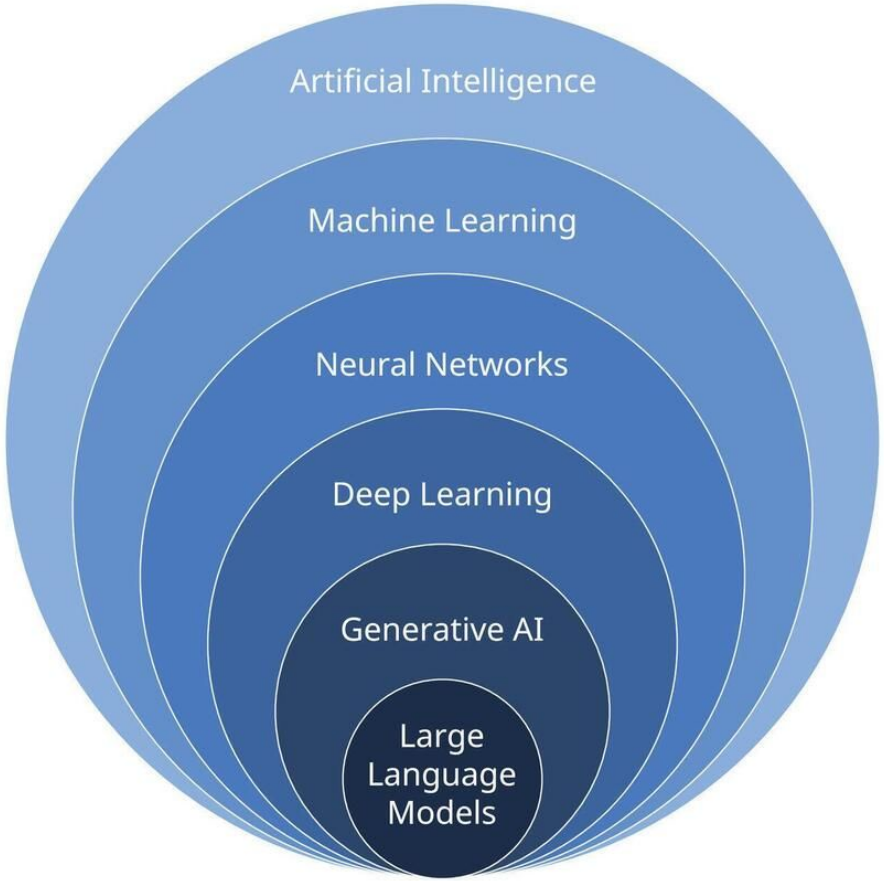


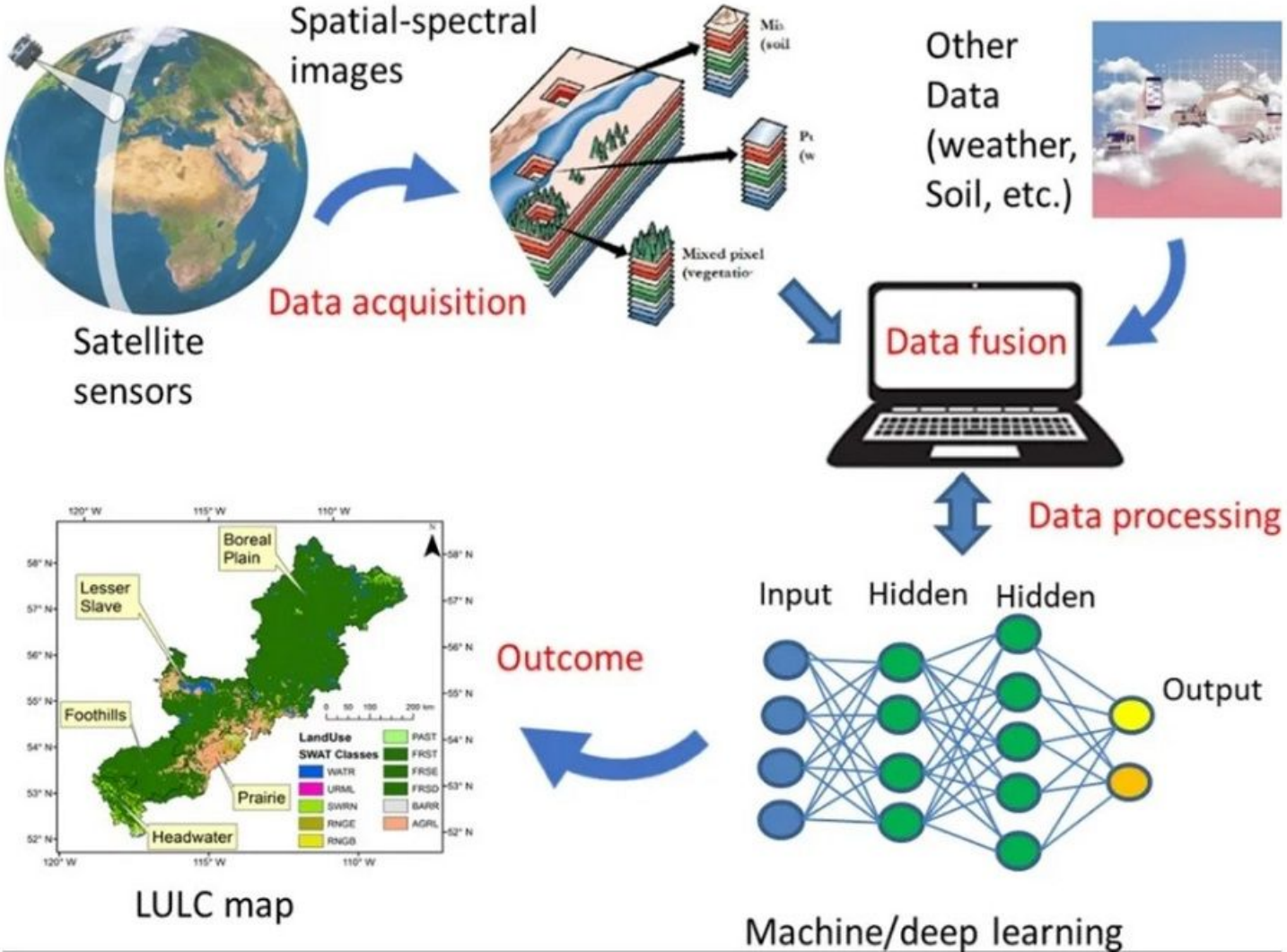




Overview of AI in Computer Vision

- **Artificial Intelligence (AI)** enables machines to interpret and understand visual data.
- **Computer Vision (CV)** is the field that allows computers to extract, analyze, and process visual information from the world.


Artificial Intelligence
Machine Learning
Neural Networks
Deep Learning
Generative AI
Large Language Models


Spatial-spectral images
Other Data (weather, Soil, etc.)
Satellite sensors
Data acquisition
Data fusion
Data processing
Input
Hidden
Hidden
Output
Outcome
Boreal Plain
Lesser Slave
Foothills
Prairie
Headwater
LULC map
Machine/deep learning


LLM
LARGE LANGUAGE MODEL

Business and Legal

- Purchasing agents
- Compensation specialists
- Management analysts
- Market research analysts
- Marketing specialists
- Lawyers and paralegals

Finance

- Insurance underwriters
- Budget analysts
- Accountants and auditors
- Personal financial advisors
- Credit professionals
- Financial analysts
- Tax preparers

Social Sciences

- Geographers
- Epidemiologists
- Survey researchers
- Political scientists
- Sociologists
- Economists

Writing and Editing

- Writers and authors
- Reporters and correspondents
- Technical writers
- Interpreters and translators
- Editors

STEM

- Programmers and software developers
- Web developers
- Some types of engineers
- Data scientists
- Physicists
- Medical scientists
- Operations research analysts

Sales

- Insurance sales agents
- Advertising sales agents
- Travel agents
- Securities, commodities and financial sellers
- Telemarketers

Office and Administrative Support

- Procurement clerks
- Credit authorizers, checkers and clerks
- Cargo and freight agents
- Statistical assistants
- Loan interviewers and clerks
- Billing and posting clerks

Other

- Postsecondary teachers
- Public relations specialists
- Interior designers

Aspect	AI (Artificial Intelligence)	GenAI (Generative AI)
Purpose	AI performs tasks that require human-like intelligence, such as problem-solving, decision-making, and analysis. **Example:** AI makes computers do smart things like recognizing cats in photos, understanding what you say to a voice assistant, and deciding what to show in your social media feed.	GenAI focuses on generating creative content and mimicking human-like output, like text, music, and art. **Example:** GenAI helps computers make art, music, and text that looks like it was made by humans, even though it's created by machines.
Learning	AI learns from existing data and uses it for predictions and tasks. **Example:** AI learns from lots of data, like thousands of cat images, to get better at recognizing cats. It uses this learning to improve its cat-spotting skill and similar tasks.	GenAI learns to generate content by analyzing patterns in existing data and creating new content based on those patterns. **Example:** GenAI learns to create by studying examples of human creations. For instance, it might learn from many beautiful paintings to make its own artwork.
Business Application	In business, AI is used for things like predicting customer preferences (e.g., what products you might like to buy), automating routine tasks (e.g., data entry or answering common customer questions), and fraud detection (e.g., spotting suspicious credit card activity).	GenAI has applications in creative industries like advertising and marketing (e.g., creating compelling advertisements), video game design (e.g., generating game environments and characters), and content creation (e.g., generating social media posts, art, and music). It can also be used for personalized product recommendations and even deepfake technology for the entertainment industry.
Popular tools	- **TensorFlow:** A powerful machine learning framework. - **PyTorch:** A deep learning framework known for its flexibility. - **Scikit-learn:** A popular library for machine learning and data analysis.	- **OpenAI's GPT-3**: A language model known for generating human-like text. - **NVIDIA's StyleGAN2:** Used for creating realistic images and artwork. - **DeepDream:** A tool that turns photos into psychedelic, artistic images.
Ethical Concerns	Raises ethical concerns related to privacy, bias, job displacement, and algorithm transparency. **Example:** AI can sometimes invade your privacy, show you biased information, or take over jobs in certain industries. It's like a robot butler that might accidentally spill your secrets.	Raises ethical concerns, particularly in the context of deep fakes, content manipulation, and potential misinformation issues. **Example:** GenAI raises concerns about deepfake videos where people's faces are put on others' bodies in videos, which can be misleading or even used for deceptive purposes. It can create content that blurs the line between real and fake.

What is AI?

ANI vs. AGI vs. ASI

Artificial narrow intelligence (ANI)

Designed to perform specific tasks

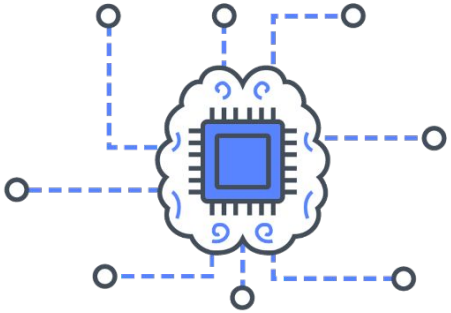
Artificial general intelligence (AGI)

Can behave in a human-like way across all tasks

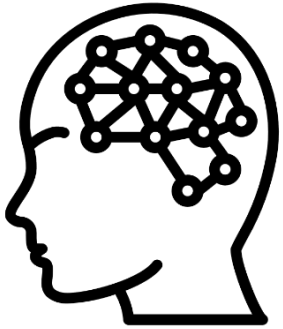
Artificial super intelligence (ASI)

Smarter than humans—the stuff of sci-fi

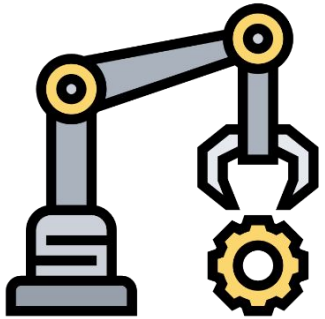
_zapier



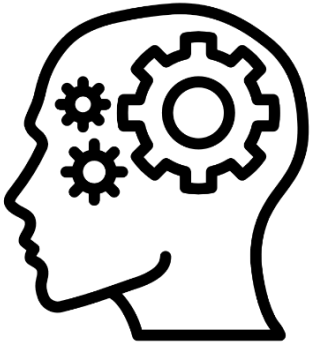
Machine Learning



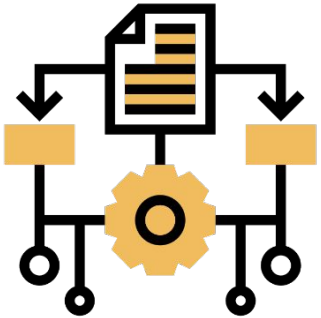
Neural Networks



Robotics

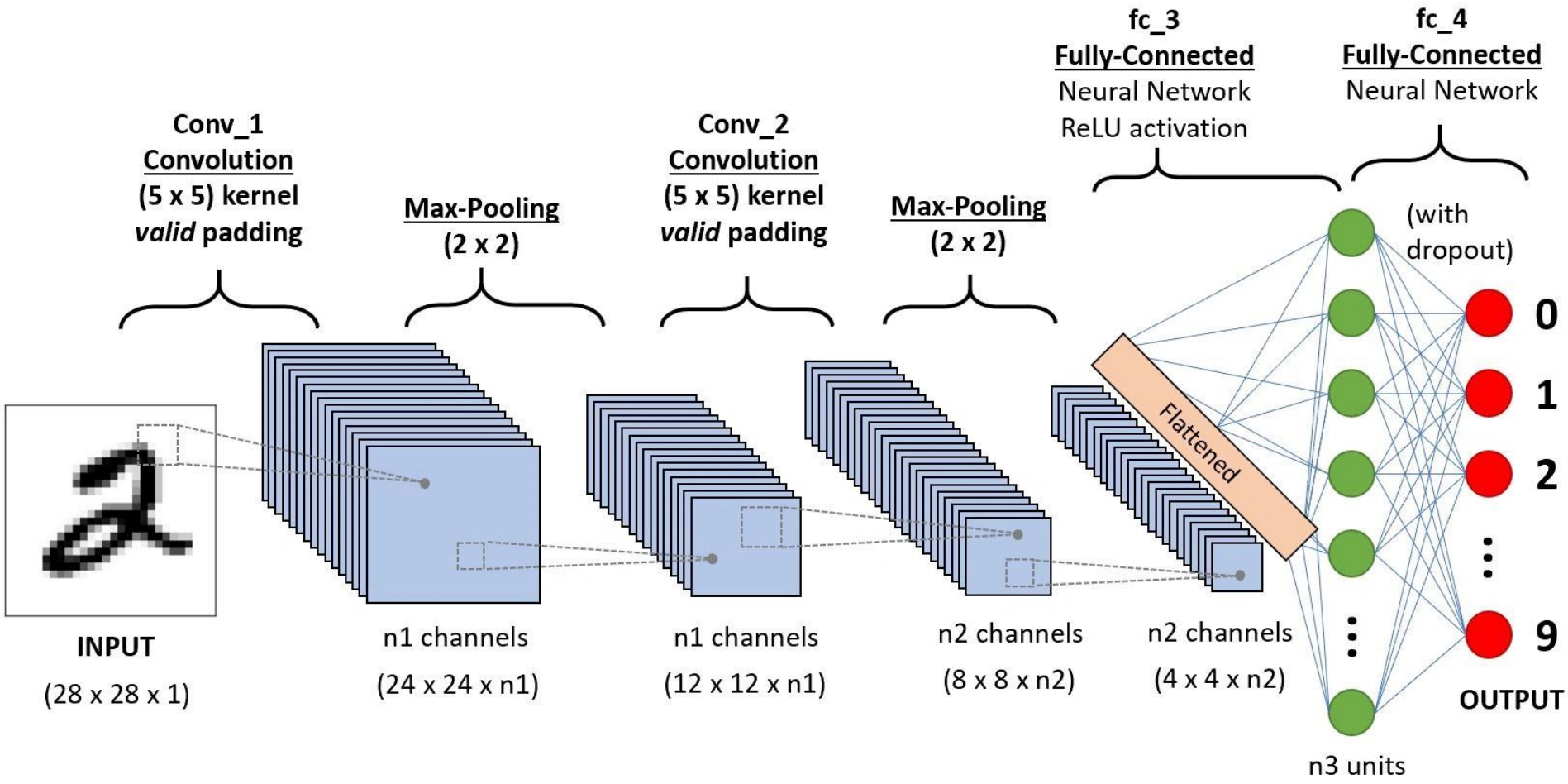


Expert Systems

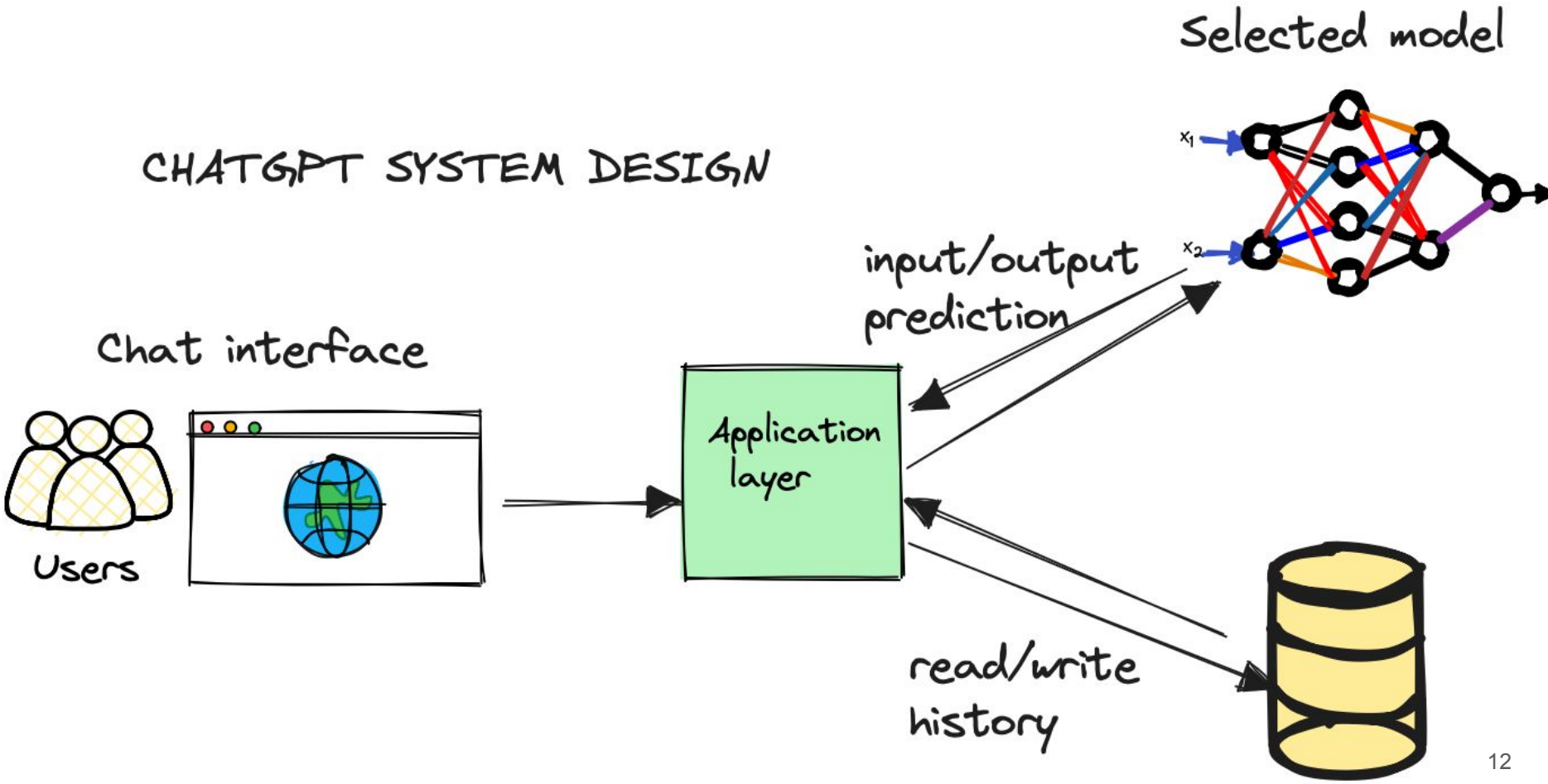


Fuzzy Logic

Natural Language Processing


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

CHATGPT SYSTEM DESIGN



Satellite Image Classification

- **Purpose:** Automatically classify satellite images into categories (e.g., urban, forest, water).
- **Basic Steps:**

    1. **Image Preprocessing:** Resizing, normalization.
    2. **Feature Extraction:** Convolutional layers.
    3. **Classification Layer:** Softmax for output probabilities.

Key Components in Basic AI Models for Satellite CV

1. **Convolutional Layers (Conv)**
    - Detect features like edges, textures, patterns in satellite imagery.
    - Convolution filters slide over the image to extract relevant features.
2. **Pooling Layers**
    - **Max Pooling:** Reduces spatial dimensions, retaining important features.
    - **Average Pooling:** Reduces dimensions and retains feature map's average information.
3. **Softmax Activation**
    - **Classification:** Converts the output into probability values, for multi-class classification tasks.

Machine Learning Basics

- **Types:**
    - Supervised (Regression, Classification)
    - Unsupervised (Clustering, Dimensionality Reduction)
    - Reinforcement Learning

Train-Test Split

- **Why?** Prevents overfitting, evaluates performance.
- **Common Ratios:** 80-20, 70-30
- **Example (Python):**

python Copy code

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Random Forest

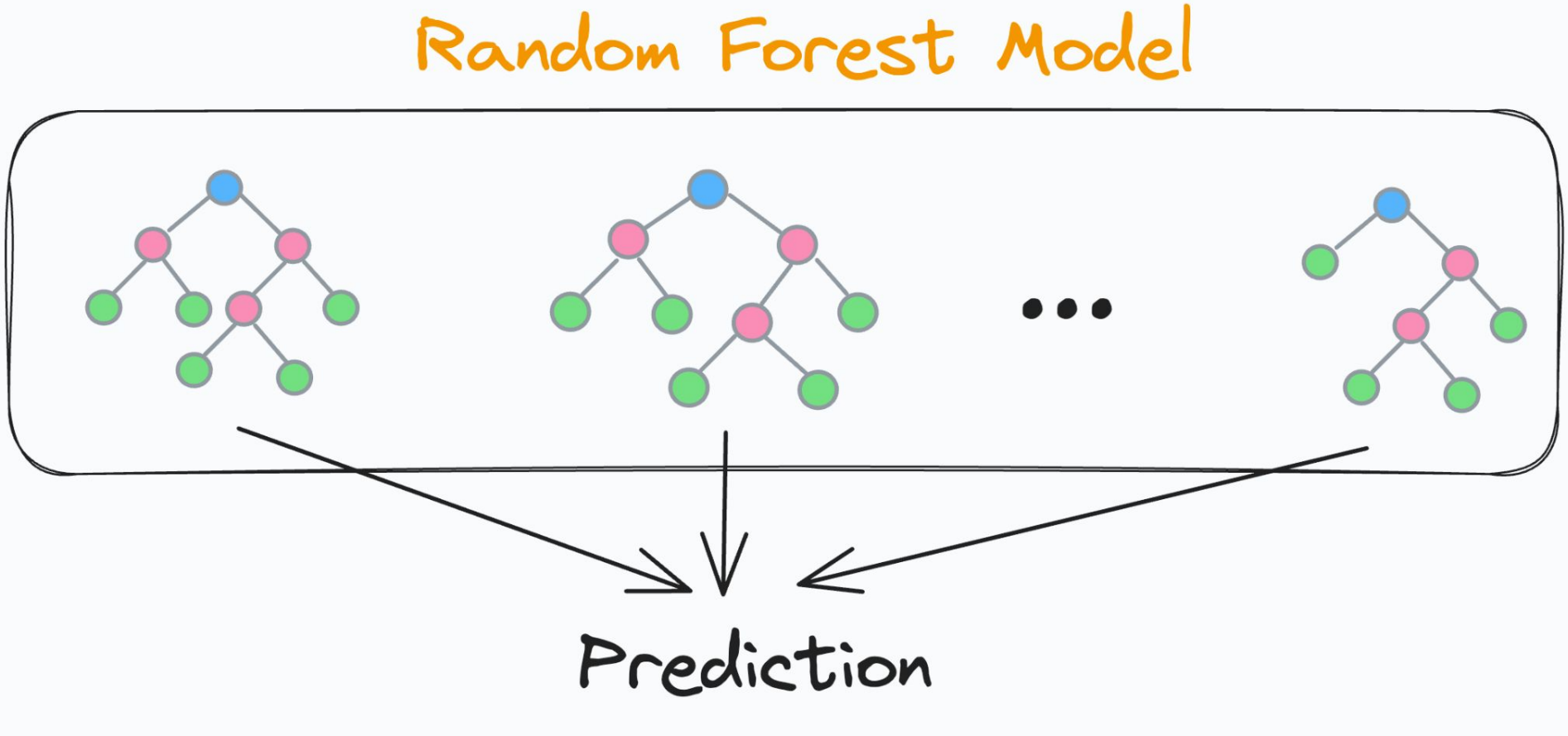
- **Ensemble of Decision Trees**
- **Reduces overfitting**
- **Example (Python):**

python

Copy code

```python
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

https://blog.dailydoseofds.com/p/your-random-forest-is-underperforming

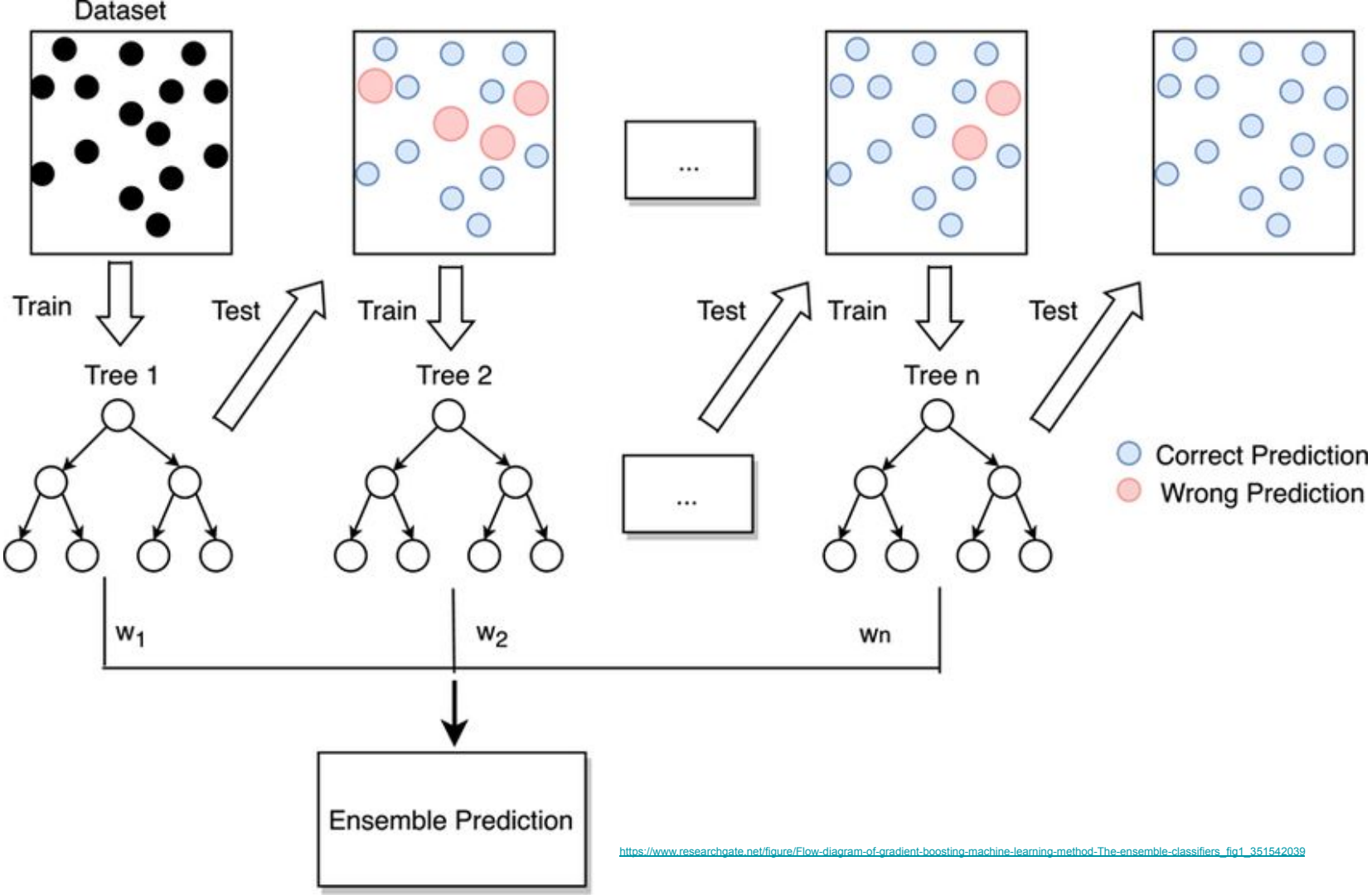


Gradient Boosting

- **Sequential tree improvement**
- **Popular Variants:** XGBoost, LightGBM, CatBoost
- **Example (Python):**

python Copy code

```python
from sklearn.ensemble import GradientBoostingClassifier
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1)
clf.fit(X_train, y_train)
```

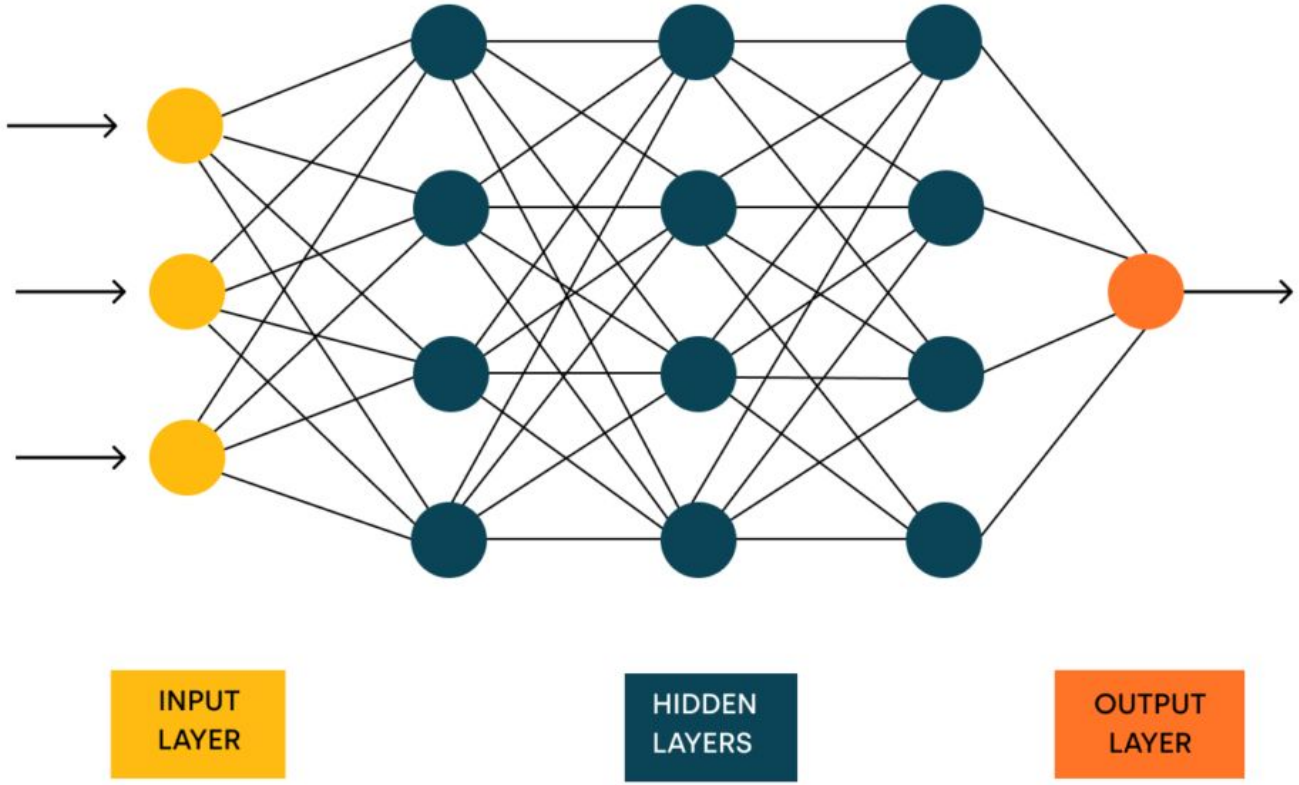


https://www.researchgate.net/figure/Flow-diagram-of-gradient-boosting-machine-learning-method-The-ensemble-classifiers_fig1_351542039

Neural Networks

- **Input → Hidden Layers → Output**
- **Example (PyTorch):**

```python
import torch.nn as nn
model = nn.Sequential(nn.Linear(10, 50), nn.ReLU(), nn.Linear(50, 1), nn.Sigmoid()
```


INPUT LAYER
HIDDEN LAYERS
OUTPUT LAYER

Confusion Matrix

Example Data:

Actual / Predicted	Positive	Negative
Positive (P)	TP = 56	FN = 18
Negative (N)	FP = 15	TN = 90

Python Code:

python

Copy code

```python
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

Metrics

- **Accuracy:**

$$\frac{TP + TN}{TP + FP + TN + FN} = \frac{56 + 90}{56 + 15 + 90 + 18} = 0.83$$

- **Precision:**

$$\frac{TP}{TP + FP} = \frac{56}{56 + 15} = 0.79$$

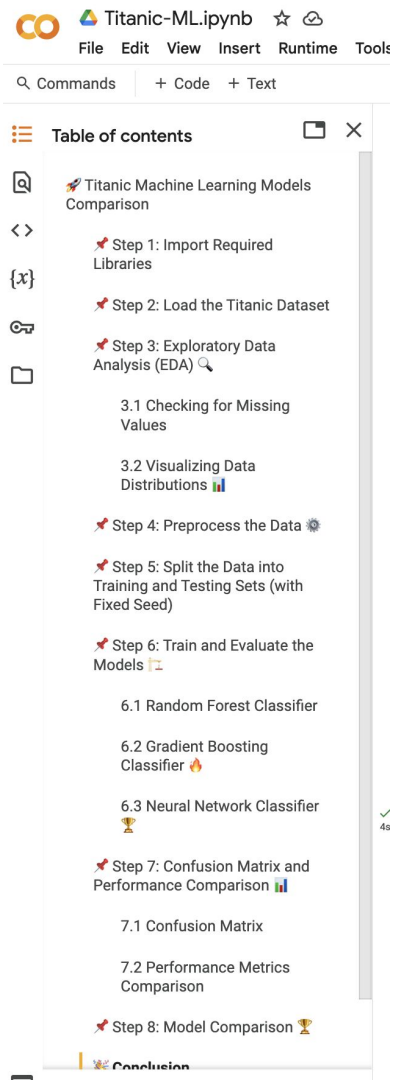
- **Recall:**

$$\frac{TP}{TP + FN} = \frac{56}{56 + 18} = 0.76$$

- **F1-Score:**

$$2 \times \frac{0.79 \times 0.76}{0.79 + 0.76} = 0.77$$

Titanic Machine Learning



🚀 Titanic Machine Learning Models Comparison

By Kao Panboonyuen

This Colab notebook will guide you through:

- ✅ Loading the Titanic dataset
- ✅ Exploring and analyzing the data (EDA)
- ✅ Splitting the data into training and testing sets
- ✅ Training 3 machine learning models:
  - 🏗️ **Random Forest Classifier**
  - 🔥 **Gradient Boosting Classifier**
  - 🏆 **Neural Network Classifier**
- ✅ Evaluating model performance using:
  - 📊 **Confusion Matrix**
  - 📉 **Precision, Recall, F1-Score**
- ✅ Comparing the performance of each model to find the best one

📌 Step 1: Import Required Libraries

To get started, we'll need several libraries for data manipulation, visualization, and machine learning tasks.

4s

```
# Import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, classification_report, f1_score, precision_score, recall_score
```

📌 Step 2: Load the Titanic Dataset

Now, let's load the Titanic dataset from the provided GitHub URL.

```
# Load Titanic dataset from the provided link
url = 'https://github.com/kaopanboonyuen/OCSB-AI/blob/main/dataset/titanic-dataset.csv?raw=true'
data = pd.read_csv(url)

# Display the first few rows of the dataset
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Next steps: Generate code with data | View recommended plots | New interactive sheet

📌 Step 3: Exploratory Data Analysis (EDA) 🔍

Before diving into the models, it's important to understand our data. We will look for missing values, visualize some basic distributions, and explore the relationships between different features.

3.1 Checking for Missing Values

We should start by checking for any missing values in the dataset.

0s

```
# Check for missing values
data.isnull().sum()
```

	0
PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

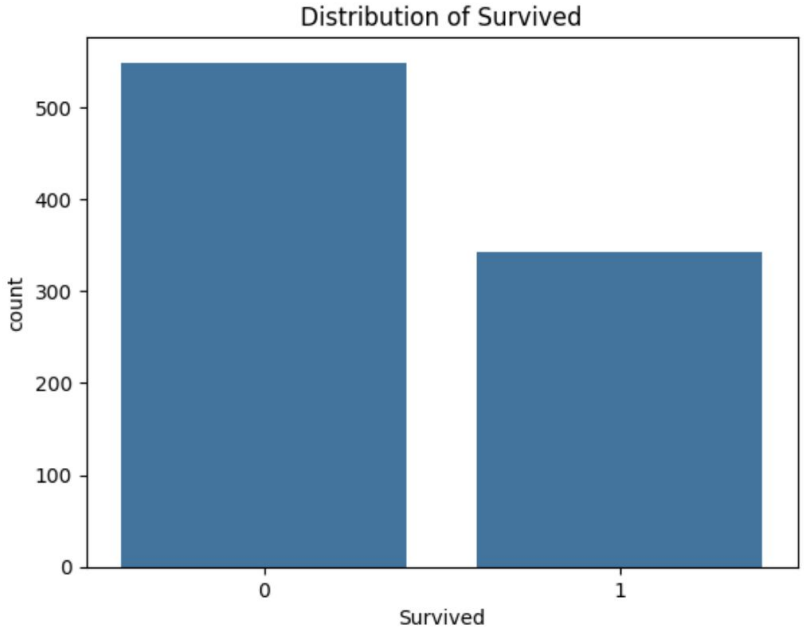
dtype: int64

3.2 Visualizing Data Distributions

Let's visualize the distribution of the "Survived" column (target variable) and some other features.

```
# Visualize the target variable "Survived"
sns.countplot(x='Survived', data=data)
plt.title('Distribution of Survived')
plt.show()

# Visualize the distribution of age
sns.histplot(data['Age'].dropna(), kde=True)
plt.title('Age Distribution')
plt.show()
```



📌 Step 4: Preprocess the Data ⚙️

We need to handle missing values and convert categorical features into numerical ones. We will also split the dataset into features and labels.

0s

```
# Handle missing values (impute or drop)
data['Age'].fillna(data['Age'].mean(), inplace=True)
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

# Convert categorical features (Sex and Embarked) to numeric
data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})
data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)

# Features and target variable
X = data.drop(['Survived', 'Name', 'Ticket', 'Cabin', 'PassengerId'], axis=1)
y = data['Survived']
```

📌 Step 5: Split the Data into Training and Testing Sets (with Fixed Seed)

Now, let's split the data into training and testing sets, ensuring we can reproduce the results by setting a random seed.

✓ 0s

```
[6]  # Split the data with a fixed seed for reproducibility
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

📌 Step 6: Train and Evaluate the Models 🏗️

We will now train three different machine learning models: Random Forest, Gradient Boosting, and Neural Network.

6.1 Random Forest Classifier

```python
# Train Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

# Evaluate the model
rf_pred = rf_model.predict(X_test)
print("Random Forest Classifier - Classification Report:\n", classification_report(y_test, rf_pred, digits=4))
```

```
Random Forest Classifier - Classification Report:
              precision    recall  f1-score   support

           0     0.8333    0.8571    0.8451       105
           1     0.7887    0.7568    0.7724        74

    accuracy                         0.8156       179
   macro avg     0.8110    0.8069    0.8087       179
weighted avg     0.8149    0.8156    0.8150       179
```

6.2 Gradient Boosting Classifier 🔥

```python
# Train Gradient Boosting Classifier
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X_train, y_train)

# Evaluate the model
gb_pred = gb_model.predict(X_test)
print("Gradient Boosting Classifier - Classification Report:\n", classification_report(y_test, gb_pred, digits=4))
```

```
Gradient Boosting Classifier - Classification Report:
              precision    recall  f1-score   support

           0     0.8103    0.8952    0.8507       105
           1     0.8254    0.7027    0.7591        74

    accuracy                         0.8156       179
   macro avg     0.8179    0.7990    0.8049       179
weighted avg     0.8166    0.8156    0.8128       179
```

6.3 Neural Network Classifier 🏆

```
[10] # Train Neural Network Classifier (MLP)
     nn_model = MLPClassifier(random_state=42)
     nn_model.fit(X_train, y_train)

     # Evaluate the model
     nn_pred = nn_model.predict(X_test)
     print("Neural Network Classifier - Classification Report:\n", classification_report(y_test, nn_pred, digits=4))
```

1s

```
Neural Network Classifier - Classification Report:
              precision    recall  f1-score   support

           0     0.8148    0.8381    0.8263       105
           1     0.7606    0.7297    0.7448        74

    accuracy                         0.7933       179
   macro avg     0.7877    0.7839    0.7856       179
weighted avg     0.7924    0.7933    0.7926       179
```

📌 Step 7: Confusion Matrix and Performance Comparison 📊

Let's create confusion matrices and compare the F1-score, Precision, and Recall for each model.

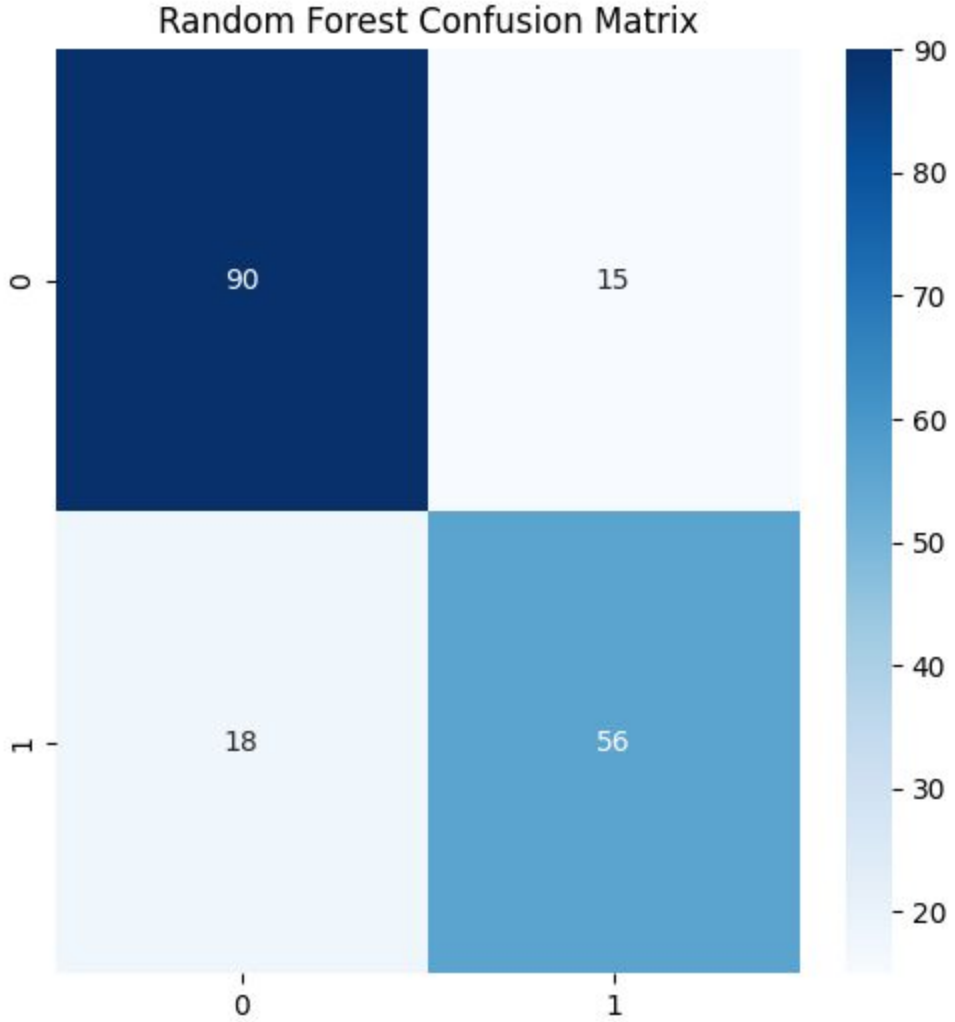
7.1 Confusion Matrix

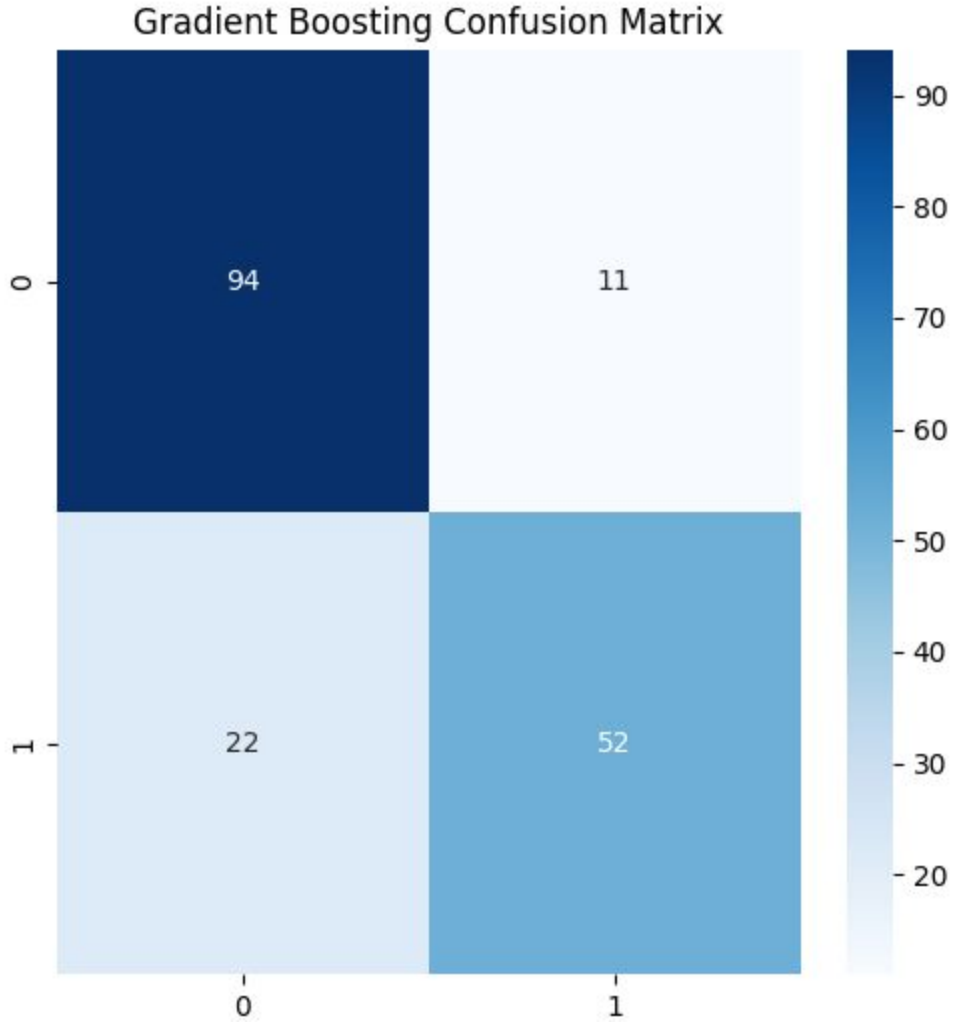
✓ 1s

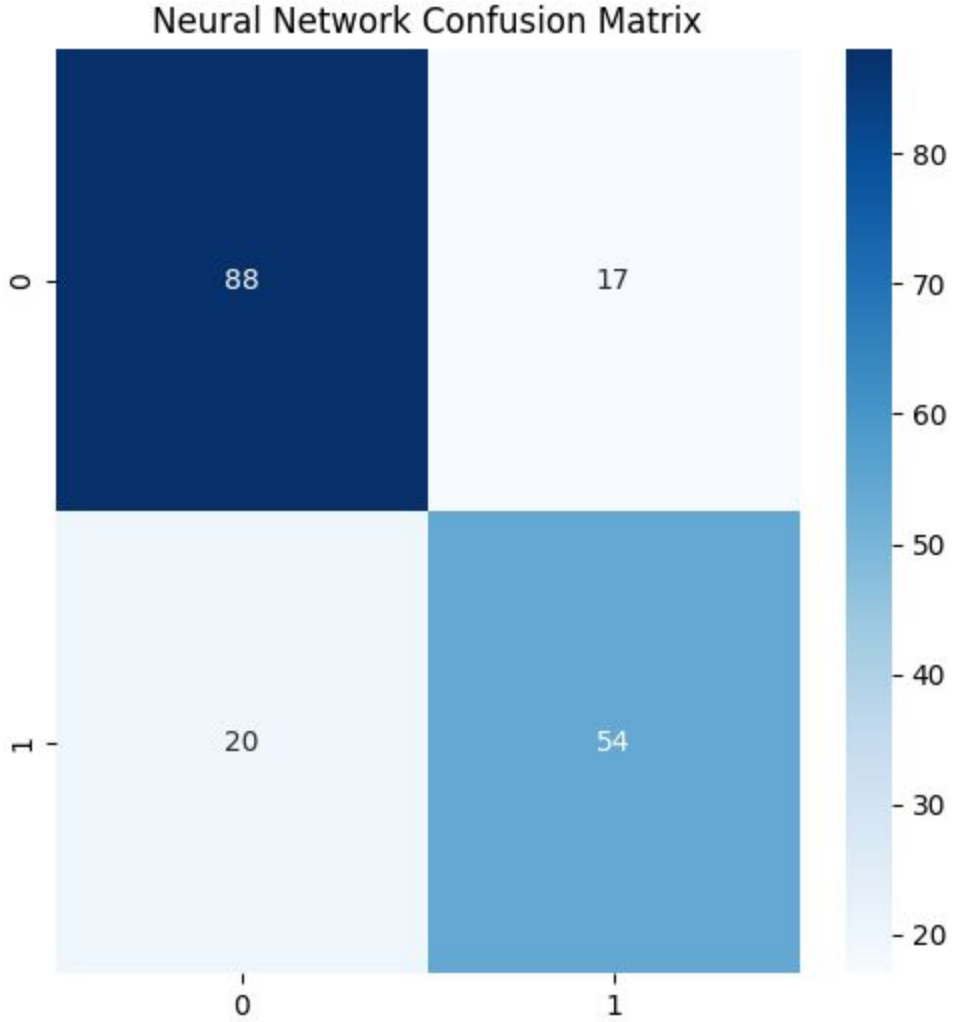
```
# Confusion Matrix for Random Forest
plt.figure(figsize=(6, 6))
sns.heatmap(confusion_matrix(y_test, rf_pred), annot=True, fmt='d', cmap='Blues')
plt.title("Random Forest Confusion Matrix")
plt.show()

# Confusion Matrix for Gradient Boosting
plt.figure(figsize=(6, 6))
sns.heatmap(confusion_matrix(y_test, gb_pred), annot=True, fmt='d', cmap='Blues')
plt.title("Gradient Boosting Confusion Matrix")
plt.show()

# Confusion Matrix for Neural Network
plt.figure(figsize=(6, 6))
sns.heatmap(confusion_matrix(y_test, nn_pred), annot=True, fmt='d', cmap='Blues')
plt.title("Neural Network Confusion Matrix")
plt.show()
```


Random Forest Confusion Matrix
90
15
18
56
0
1
0
1
90
80
70
60
50
40
30
20


Gradient Boosting Confusion Matrix
94
11
22
52
0
1
0
1
90
80
70
60
50
40
30
20


Neural Network Confusion Matrix
88
17
20
54
0
1
0
1
80
70
60
50
40
30
20

7.2 Performance Metrics Comparison

```
f1_rf = f1_score(y_test, rf_pred)
f1_gb = f1_score(y_test, gb_pred)
f1_nn = f1_score(y_test, nn_pred)

print(f"Random Forest F1-Score: {f1_rf:.4f}")
print(f"Gradient Boosting F1-Score: {f1_gb:.4f}")
print(f"Neural Network F1-Score: {f1_nn:.4f}")

# Precision and Recall Comparison
precision_rf = precision_score(y_test, rf_pred)
precision_gb = precision_score(y_test, gb_pred)
precision_nn = precision_score(y_test, nn_pred)

recall_rf = recall_score(y_test, rf_pred)
recall_gb = recall_score(y_test, gb_pred)
recall_nn = recall_score(y_test, nn_pred)

print(f"Random Forest Precision: {precision_rf:.4f}, Recall: {recall_rf:.4f}")
print(f"Gradient Boosting Precision: {precision_gb:.4f}, Recall: {recall_gb:.4f}")
print(f"Neural Network Precision: {precision_nn:.4f}, Recall: {recall_nn:.4f}")
```

```
Random Forest F1-Score: 0.7724
Gradient Boosting F1-Score: 0.7591
Neural Network F1-Score: 0.7448
Random Forest Precision: 0.7887, Recall: 0.7568
Gradient Boosting Precision: 0.8254, Recall: 0.7027
Neural Network Precision: 0.7606, Recall: 0.7297
```

📌 Step 8: Model Comparison 🏆

Finally, we will compare the performance of the three models to determine which one performed the best.

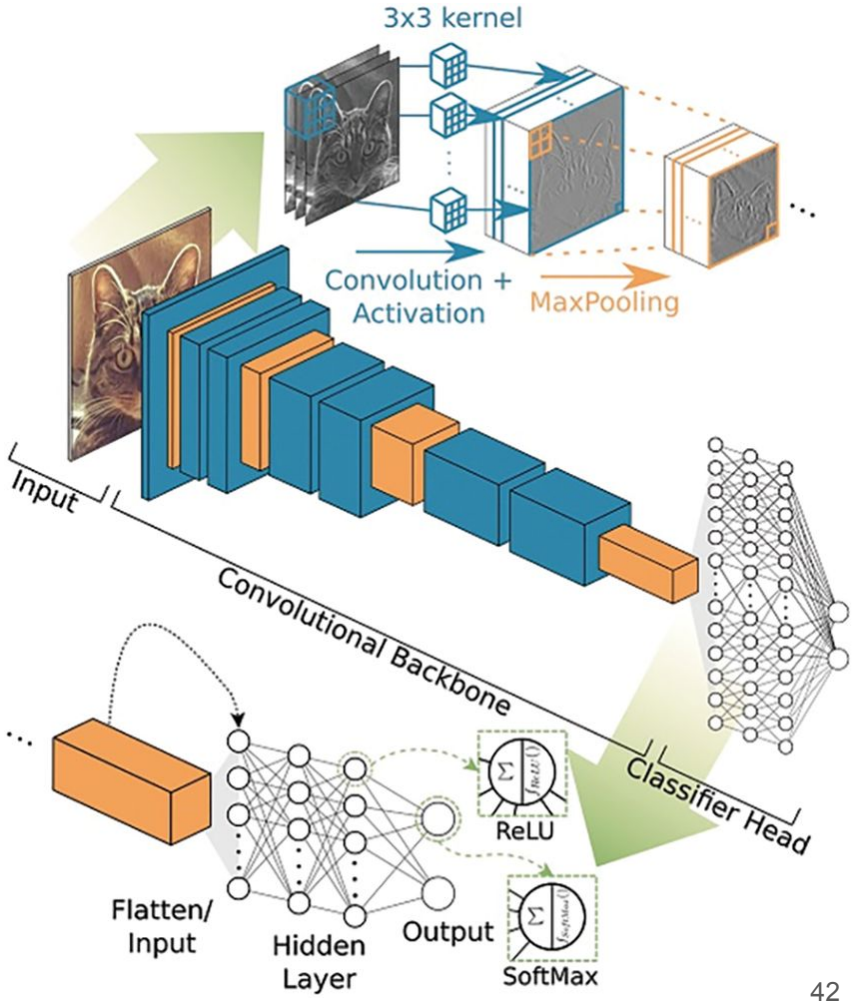
- Based on **F1-Score**, **Precision**, and **Recall**, we will decide the winning model.

🎉 Conclusion

In this notebook, we have:

- Loaded and preprocessed the Titanic dataset.
- Performed exploratory data analysis (EDA).
- Built and evaluated three different machine learning models: Random Forest, Gradient Boosting, and Neural Networks.
- Compared the performance of each model using confusion matrices and classification reports.

Satellite Classification

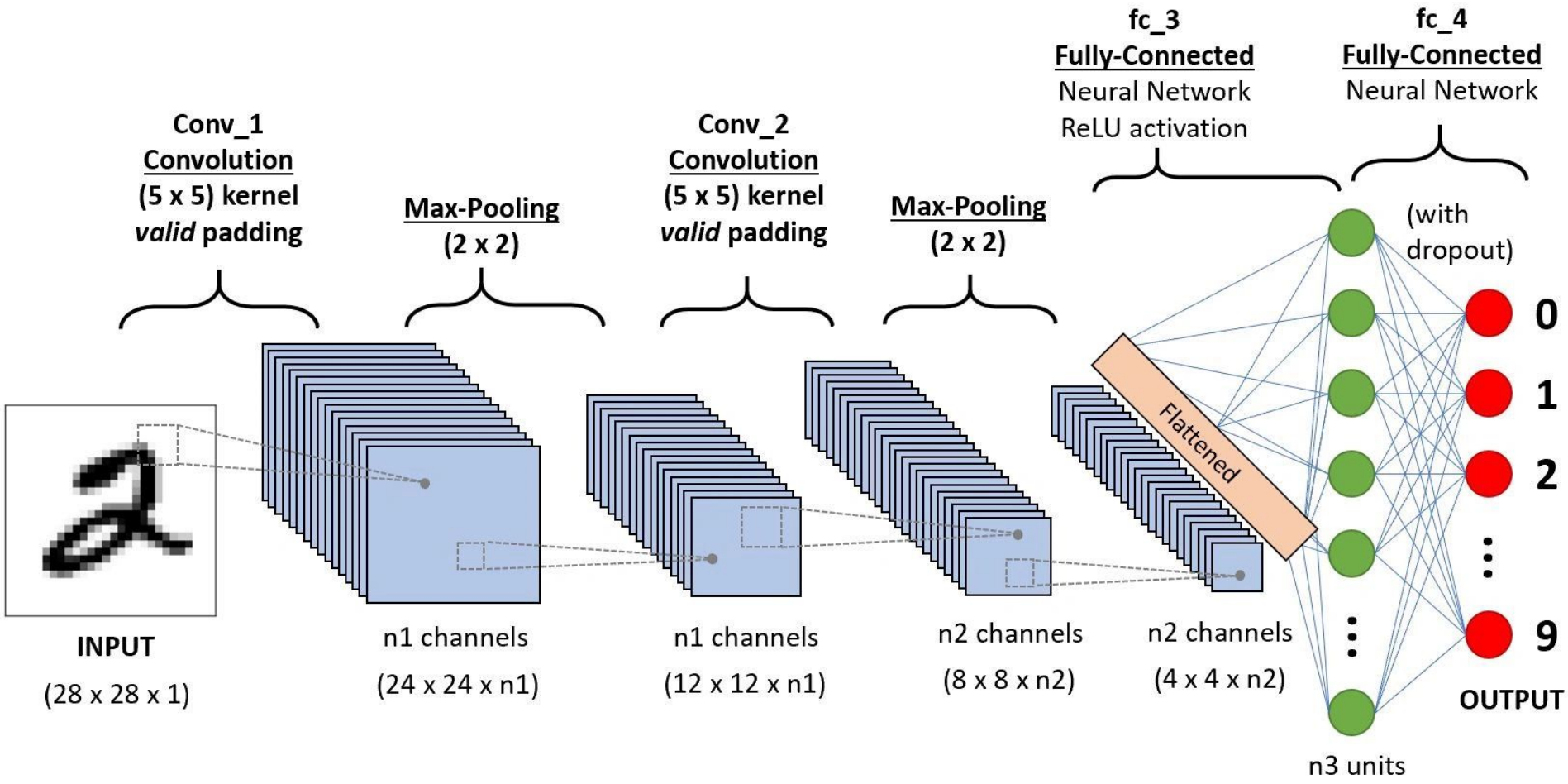


Basic Vision Network Architecture for Satellite Classification

1. **Input Image:** Satellite image of a specific region.
2. **Convolution Layer:** Extracts spatial features.
3. **Pooling Layer:** Reduces feature map size.
4. **Attention Layer:** Focuses on key areas.
5. **Fully Connected Layers + Softmax:** Final classification.

Convolutional Neural Networks (CNNs)

- **Uses:** Image classification, object detection, segmentation
- **Layers:** Convolution → Activation → Pooling → Fully Connected


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

Convolution Layer

Formula to Compute Output Size:

$$O = \frac{(I - K + 2P)}{S} + 1$$

Where:

- I = Input size (Height/Width)
- K = Kernel size
- P = Padding
- S = Stride
- O = Output size

Example Calculation:

Input: 32×32 image, **Kernel:** 3×3, **Stride:** 1, **Padding:** 1

$$O = \frac{(32 - 3 + 2(1))}{1} + 1 = \frac{(32 - 3 + 2)}{1} + 1 = 32$$

💡 **Same size output when** $P = \frac{(K-1)}{2}$

Stride & Padding

- **Stride:** How far the filter moves
  - **Stride = 1** → Small movement, detailed features
  - **Stride = 2** → Faster, lower resolution
- **Padding:** Adds zeros around input
  - **Valid (No Padding, P=0)** → Output shrinks
  - **Same (P=1 for 3×3 filter)** → Keeps size same

Pooling Layer (Downsampling)

Max Pooling (Example 2×2, Stride 2)

- Takes the **maximum** value in each 2×2 region
- **Reduces size, retains important features**

Formula for Output:

$$O = \frac{(I - K)}{S} + 1$$

Example: **Input 32×32, Pooling 2×2, Stride 2**

$$O = \frac{(32 - 2)}{2} + 1 = 16$$

Fully Connected Layer

- **Flattens features** → Passes to Dense Layers
- **Example:**
  - Output from CNN: **512 values**
  - Dense layer: **256 neurons**
  - Weights: **512×256**
  - **Output:** 512×256 dot product + bias

Activation Functions

- **ReLU (Rectified Linear Unit):** $f(x) = \max(0, x)$
  - Introduces non-linearity
- **Softmax (for multi-class classification)**

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

  - Converts logits into probabilities

Dropout (Regularization)

- **Prevents overfitting** by randomly setting neurons to **0** during training
- Example: **Dropout(0.5) → 50% neurons inactive**

Python Example (CNN in PyTorch)

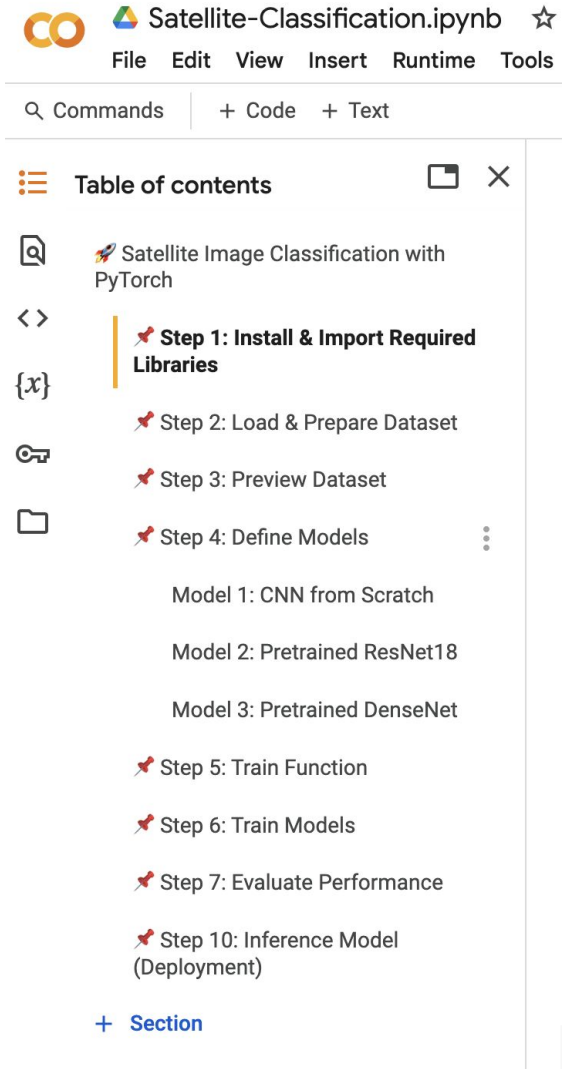
python

Copy code

```python
import torch.nn as nn

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(16 * 16 * 16, 128)  # Flatten
        self.fc2 = nn.Linear(128, 10)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = self.pool(nn.ReLU()(self.conv1(x)))  # Conv → ReLU → Pool
        x = x.view(-1, 16 * 16 * 16)  # Flatten
        x = self.dropout(nn.ReLU()(self.fc1(x)))  # Dense → ReLU → Dropout
        x = self.fc2(x)  # Output Layer
        return nn.Softmax(dim=1)(x)
```



🚀 Satellite Image Classification with PyTorch

By Kao Panboonyuen

This Colab notebook will guide you through:

- ✅ Preparing and loading the dataset
- ✅ Exploring images and labels
- ✅ Performing exploratory data analysis (EDA)
- ✅ Splitting the dataset into Train/Val/Test (with fixed seed)
- ✅ Training 3 models:
  - 🏗️ Simple CNN (from scratch)
  - 🔥 ResNet18 (Pretrained)
  - 🏆 DenseNet (Pretrained)
- ✅ Saving & loading model weights
- ✅ Evaluating performance with accuracy, confusion matrix, precision, recall, F1-score
- ✅ Visualizing correct & incorrect predictions for error analysis

📌 Step 1: Install & Import Required Libraries

```
!pip install torch==2.0.0+cu117 torchvision torchaudio --index-url https://download.pytorch.org/whl/cu117
```

```
Looking in indexes: https://download.pytorch.org/whl/cu117
Collecting torch==2.0.0+cu117
  Downloading https://download.pytorch.org/whl/cu117/torch-2.0.0%2Bcu117-cp311-cp311-linux_x86_64.whl (1843.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 GB 590.5 kB/s eta 0:00:00
Requirement already satisfied: torchvision in /usr/local/lib/python3.11/dist-packages (0.15.1+cpu)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.11/dist-packages (2.0.1+cpu)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch==2.0.0+cu117) (3.17.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from torch==2.0.0+cu117) (4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.11/dist-packages (from torch==2.0.0+cu117) (1.13.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch==2.0.0+cu117) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch==2.0.0+cu117) (3.1.6)
Collecting triton==2.0.0 (from torch==2.0.0+cu117)
  Downloading https://download.pytorch.org/whl/triton-2.0.0-1-cp311-cp311-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (63.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.3/63.3 MB 11.4 MB/s eta 0:00:00
Requirement already satisfied: cmake in /usr/local/lib/python3.11/dist-packages (from triton==2.0.0->torch==2.0.0+cu117) (3.31.6)
Collecting lit (from triton==2.0.0->torch==2.0.0+cu117)
  Downloading https://download.pytorch.org/whl/lit-15.0.7.tar.gz (132 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 132.3/132.3 kB 992.4 kB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from torchvision) (1.26.4)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from torchvision) (2.32.3)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.11/dist-packages (from torchvision) (11.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch==2.0.0+cu117) (3.0.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->torchvision) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->torchvision) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->torchvision) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->torchvision) (2025.1.31)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy->torch==2.0.0+cu117) (1.3.0)
Building wheels for collected packages: lit
  Building wheel for lit (setup.py) ... done
  Created wheel for lit: filename=lit-15.0.7-py3-none-any.whl size=89991 sha256=79b94491731a90bf7c10cac2b5f153a3d567c5fb4ea35849125108bfd18db2
  Stored in directory: /root/.cache/pip/wheels/fc/5d/45/34fe9945d5e45e261134e72284395be36c2d4828af38e2b0fe
Successfully built lit
Installing collected packages: lit, triton, torch
  Attempting uninstall: triton
    Found existing installation: triton 3.2.0
```

```
print(torch.cuda.is_available())
print(torch.cuda.device_count())
print(torch.cuda.get_device_name(0))
```

```
True
1
Tesla T4
```


```
# Download dataset
!wget https://github.com/kaopanboonyuen/OCSB-AI/raw/main/dataset/satellite-dataset.zip -O satellite-dataset.zip
!unzip satellite-dataset.zip -d dataset
```

📌 Step 2: Load & Prepare Dataset

```
[ ] # Define dataset path
    dataset_path = '/content/dataset/satellite-dataset'

    # Define image transformations
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
    ])
```

```
[ ] # Load dataset
    dataset = datasets.ImageFolder(root=dataset_path, transform=transform)
    classes = dataset.classes  # Get class names
    print("Dataset classes:", classes)
```

```
Dataset classes: ['cloudy', 'desert', 'green_area', 'water']
```

```
[ ] # Split dataset (70% train, 15% validation, 15% test)
    train_size = int(0.7 * len(dataset))
    val_size = int(0.15 * len(dataset))
    test_size = len(dataset) - train_size - val_size

    train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size, test_size])
```

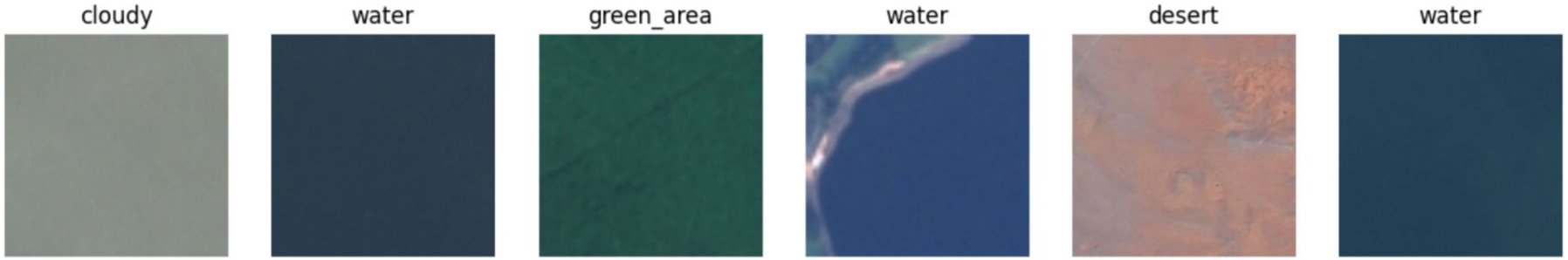
```
[ ] # DataLoader (Batch Size = 32)
    batch_size = 32
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

📌 Step 3: Preview Dataset

```python
# Visualize a few images
import matplotlib.pyplot as plt
import random

def show_images(dataset, num=6):
    fig, axes = plt.subplots(1, num, figsize=(15, 5))
    indices = random.sample(range(len(dataset)), num)  # Select random indices
    for i, idx in enumerate(indices):
        img, label = dataset[idx]
        axes[i].imshow(img.permute(1, 2, 0))
        axes[i].set_title(classes[label])
        axes[i].axis('off')
    plt.show()

show_images(train_dataset)
```



📌 Step 4: Define Models

Model 1: CNN from Scratch

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 56 * 56, 128)
        self.fc2 = nn.Linear(128, len(classes))

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Model 2: Pretrained ResNet18

```
[ ]  model_resnet = models.resnet18(pretrained=True)
     for param in model_resnet.parameters():
         param.requires_grad = False
     model_resnet.fc = nn.Linear(model_resnet.fc.in_features, len(classes))
```

Model 3: Pretrained DenseNet

```
[ ] model_densenet = models.densenet121(pretrained=True)
    for param in model_densenet.parameters():
        param.requires_grad = False
    model_densenet.classifier = nn.Linear(model_densenet.classifier.in_features, len(classes))
```

📌 Step 5: Train Function

```python
def train_model(model, train_loader, val_loader, epochs=10, learning_rate=0.001):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    train_losses, val_losses, train_accs, val_accs = [], [], [], []

    for epoch in range(epochs):
        model.train()
        running_loss, correct, total = 0.0, 0, 0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        train_losses.append(running_loss / len(train_loader))
        train_accs.append(correct / total)

        print(f"Epoch {epoch+1}: Train Loss = {train_losses[-1]:.4f}, Train Acc = {train_accs[-1]:.4f}")

    print("Training Complete!")
    return model, train_losses, train_accs
```

📌 Step 6: Train Models

```
# Train Simple CNN
model1 = SimpleCNN()
train_model(model1, train_loader, val_loader)
```

```
Epoch 1: Train Loss = 0.4911, Train Acc = 0.7962
Epoch 2: Train Loss = 0.3213, Train Acc = 0.8741
Epoch 3: Train Loss = 0.2923, Train Acc = 0.8894
Epoch 4: Train Loss = 0.3016, Train Acc = 0.8724
Epoch 5: Train Loss = 0.2634, Train Acc = 0.9021
Epoch 6: Train Loss = 0.2333, Train Acc = 0.9051
Epoch 7: Train Loss = 0.2446, Train Acc = 0.9005
Epoch 8: Train Loss = 0.2561, Train Acc = 0.8972
Epoch 9: Train Loss = 0.2088, Train Acc = 0.9203
Epoch 10: Train Loss = 0.2214, Train Acc = 0.9094
Training Complete!
(SimpleCNN(
   (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
   (fc1): Linear(in_features=200704, out_features=128, bias=True)
   (fc2): Linear(in_features=128, out_features=4, bias=True)
 ),
 [0.4910834494737848,
```

```
[ ] # Train ResNet
    train_model(model_resnet, train_loader, val_loader)
```

```
Epoch 1: Train Loss = 0.3927, Train Acc = 0.9008
Epoch 2: Train Loss = 0.1196, Train Acc = 0.9769
Epoch 3: Train Loss = 0.0885, Train Acc = 0.9779
Epoch 4: Train Loss = 0.0770, Train Acc = 0.9802
Epoch 5: Train Loss = 0.0557, Train Acc = 0.9886
Epoch 6: Train Loss = 0.0601, Train Acc = 0.9827
Epoch 7: Train Loss = 0.0581, Train Acc = 0.9843
Epoch 8: Train Loss = 0.0477, Train Acc = 0.9876
Epoch 9: Train Loss = 0.0390, Train Acc = 0.9901
Epoch 10: Train Loss = 0.0305, Train Acc = 0.9931
Training Complete!
(ResNet(
   (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
   (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (relu): ReLU(inplace=True)
   (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
   (layer1): Sequential(
     (0): BasicBlock(
       (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
       (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (relu): ReLU(inplace=True)
       (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
       (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     )
     (1): BasicBlock(
       (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
       (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (relu): ReLU(inplace=True)
       (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
       (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     )
   )
   (layer2): Sequential(
```

```
# Train DenseNet
train_model(model_densenet, train_loader, val_loader)
```

```
Epoch 1: Train Loss = 0.3673, Train Acc = 0.9066
Epoch 2: Train Loss = 0.1080, Train Acc = 0.9784
Epoch 3: Train Loss = 0.0728, Train Acc = 0.9827
Epoch 4: Train Loss = 0.0628, Train Acc = 0.9833
Epoch 5: Train Loss = 0.0544, Train Acc = 0.9863
Epoch 6: Train Loss = 0.0429, Train Acc = 0.9891
Epoch 7: Train Loss = 0.0393, Train Acc = 0.9909
Epoch 8: Train Loss = 0.0408, Train Acc = 0.9901
Epoch 9: Train Loss = 0.0343, Train Acc = 0.9911
Epoch 10: Train Loss = 0.0316, Train Acc = 0.9919
Training Complete!
(DenseNet(
   (features): Sequential(
     (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
     (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (relu0): ReLU(inplace=True)
     (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
     (denseblock1): _DenseBlock(
       (denselayer1): _DenseLayer(
         (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (relu1): ReLU(inplace=True)
         (conv1): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
         (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (relu2): ReLU(inplace=True)
         (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
       )
       (denselayer2): _DenseLayer(
         (norm1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (relu1): ReLU(inplace=True)
         (conv1): Conv2d(96, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
         (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (relu2): ReLU(inplace=True)
```

📌 Step 7: Evaluate Performance

```
[ ]  import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.metrics import confusion_matrix

     # Function to plot confusion matrix
     def plot_confusion_matrix(true_labels, predictions, classes):
         cm = confusion_matrix(true_labels, predictions)
         plt.figure(figsize=(8, 6))
         sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
         plt.title('Confusion Matrix')
         plt.xlabel('Predicted')
         plt.ylabel('True')
         plt.show()
```

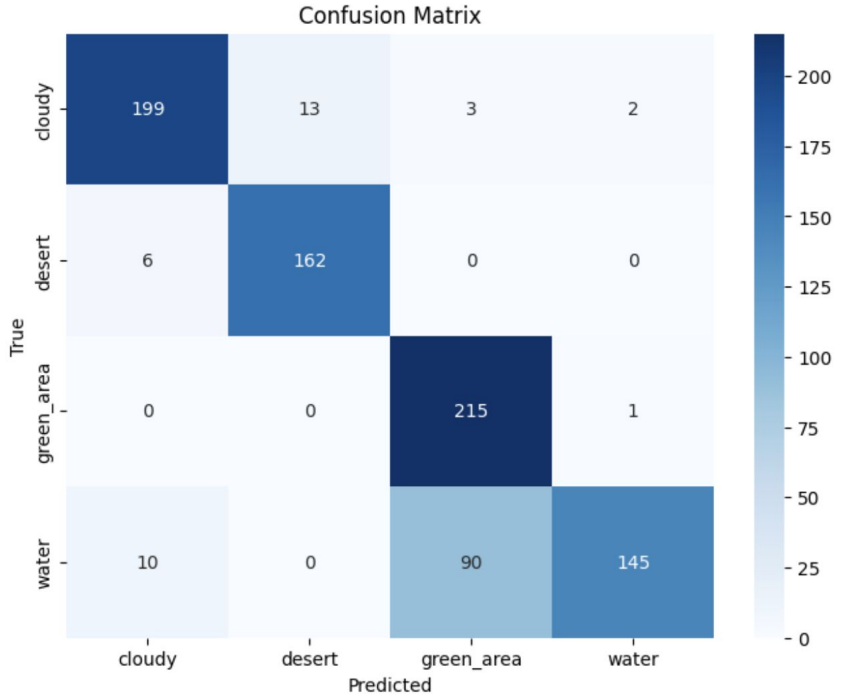
```
     from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report

     def evaluate_model(model, test_loader, classes):
         model.eval()
         device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
         model.to(device)
         predictions, true_labels = [], []
         with torch.no_grad():
             for inputs, labels in test_loader:
                 inputs, labels = inputs.to(device), labels.to(device)
                 outputs = model(inputs)
                 _, predicted = torch.max(outputs, 1)
                 predictions.extend(predicted.cpu().numpy())
                 true_labels.extend(labels.cpu().numpy())

         # Calculate performance metrics
         acc = accuracy_score(true_labels, predictions)
         precision = precision_score(true_labels, predictions, average='weighted')
         recall = recall_score(true_labels, predictions, average='weighted')
         f1 = f1_score(true_labels, predictions, average='weighted')
```

```
evaluate_model(model1, test_loader, classes)
evaluate_model(model_resnet, test_loader, classes)
evaluate_model(model_densenet, test_loader, classes)
```

	precision	recall	f1-score	support
cloudy	0.9256	0.9171	0.9213	217
desert	0.9257	0.9643	0.9446	168
green_area	0.6981	0.9954	0.8206	216
water	0.9797	0.5918	0.7379	245
accuracy			0.8522	846
macro avg	0.8823	0.8671	0.8561	846
weighted avg	0.8832	0.8522	0.8471	846



📌 Step 9: Save, Load, and Inference Model Weights

```
[ ] # Save the trained model (DenseNet)
    torch.save(model_densenet.state_dict(), 'DenseNet_model.pth')
    print("Model DenseNet saved to DenseNet_model.pth")
```

```
Model DenseNet saved to DenseNet_model.pth
```

```
[ ] import torch
    import torchvision.models as models
    import torch.nn as nn

    # Load the pre-trained DenseNet model structure
    model_densenet = models.densenet121(weights=models.DenseNet121_Weights.IMAGENET1K_V1)

    # Modify the final classifier layer to match your number of classes
    model_densenet.classifier = nn.Linear(model_densenet.classifier.in_features, len(classes))

    # Load the saved model weights
    model_densenet.load_state_dict(torch.load('DenseNet_model.pth'))
    print("Model DenseNet loaded from DenseNet_model.pth")

    # Set the model to evaluation mode (important for inference)
    model_densenet.eval()

    # Now you can perform inference
```

```
Model DenseNet loaded from DenseNet_model.pth
DenseNet(
  (features): Sequential(
```

📌 Step 10: Inference Model (Deployment)

```python
import torch
from torchvision import transforms
from PIL import Image
import matplotlib.pyplot as plt

# Function to perform inference
def predict_image(image_path, model, classes, transform):
    # Load image and convert to RGB (in case it has 4 channels like RGBA)
    img = Image.open(image_path).convert('RGB')

    # Apply the transformations (resize and tensor conversion)
    img = transform(img).unsqueeze(0)  # Add batch dimension

    # Send the image to the same device as the model
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    img = img.to(device)

    # Perform inference
    with torch.no_grad():
        output = model(img)
        _, predicted_class = torch.max(output, 1)

    # Display the predicted class
    print(f"Predicted class: {classes[predicted_class]}")

    # Show the image
    img_show = img.squeeze().cpu().numpy().transpose((1, 2, 0))  # Convert tensor to numpy for plotting
    plt.imshow(img_show)
    plt.title(f"Predicted: {classes[predicted_class]}")
    plt.axis('off')
    plt.show()

# Example image path (replace with your image file)
image_path = '/content/dataset/satellite-dataset/cloudy/train_10021.jpg'

# Apply the same transformations used during training
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# Run inference
predict_image(image_path, model_densenet, classes, transform)
```

```
Predicted class: cloudy
```

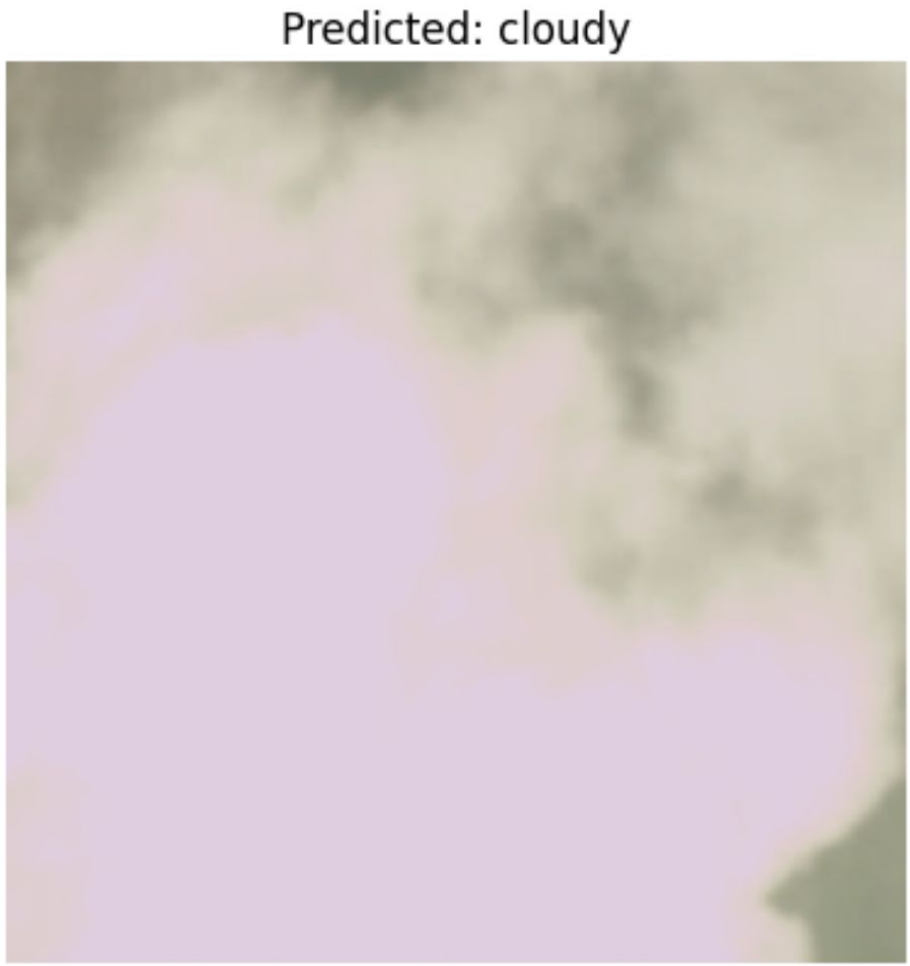
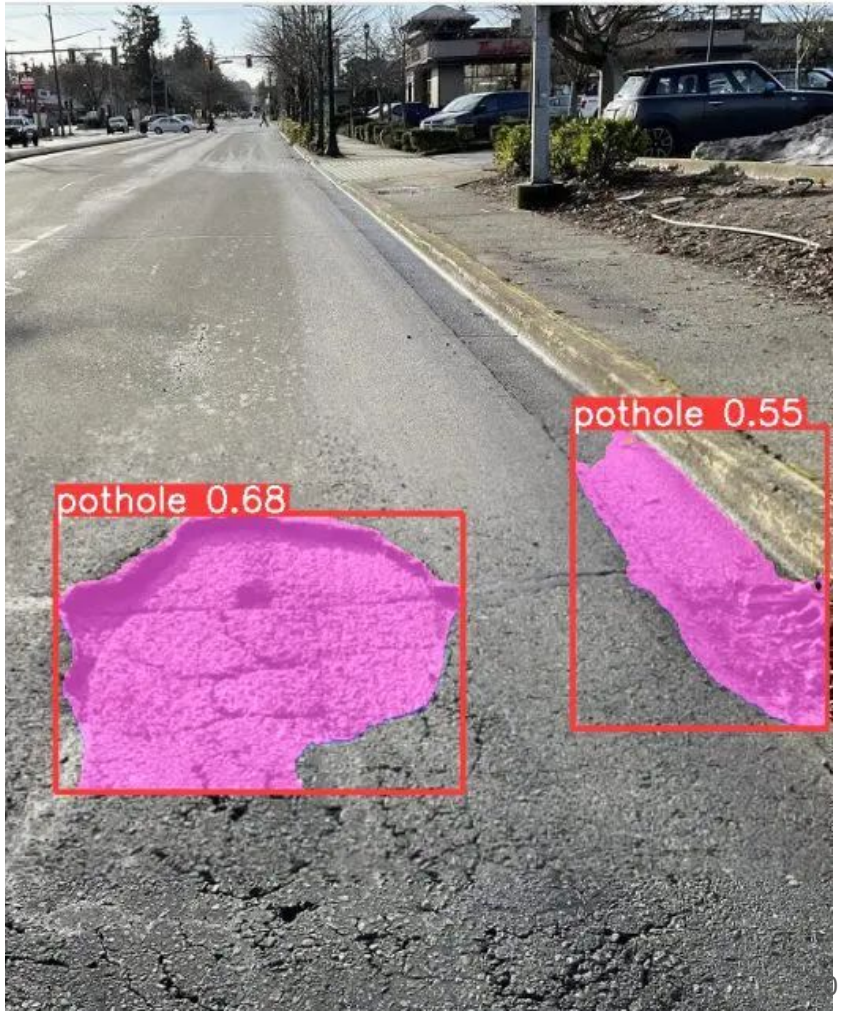


Image Segmentation



Pothole-Segmentation.ipynb

File Edit View Insert Runtime Too

Commands + Code + Text

Table of contents

🚀 Pothole Segmentation with YOLOv12 (Small Model)

By Kao Panboonyuen

This Colab notebook will guide you through:

- ✅ Preparing and loading the dataset
- ✅ Exploring images and labels
- ✅ Performing exploratory data analysis (EDA)
- ✅ Training YOLOv12n (small model) for segmentation
- ✅ Evaluating performance with accuracy, confusion matrix, precision, recall, F1-score
- ✅ Performing inference and error analysis

📌 Step 1: Install & Import Required Libraries

In this step, we'll install the necessary libraries for training YOLOv12. You'll need to install the YOLOv12 package, and PyTorch to enable GPU acceleration.

2m

```
!pip install torch==2.0.0+cu117 torchvision torchaudio --index-url https://download.pytorch.org/whl/cu117
!pip install ultralytics
```

```
    Uninstalling torch-2.6.0+cu124:
      Successfully uninstalled torch-2.6.0+cu124
  Attempting uninstall: torchvision
    Found existing installation: torchvision 0.21.0+cu124
    Uninstalling torchvision-0.21.0+cu124:
      Successfully uninstalled torchvision-0.21.0+cu124
  Attempting uninstall: torchaudio
    Found existing installation: torchaudio 2.6.0+cu124
    Uninstalling torchaudio-2.6.0+cu124:
      Successfully uninstalled torchaudio-2.6.0+cu124
Successfully installed lit-15.0.7 torch-2.0.0+cu117 torchaudio-2.0.1+cu117 torchvision-0.15.1+cu117 triton-2.0.0
Collecting ultralytics
  Downloading ultralytics-8.3.90-py3-none-any.whl.metadata (35 kB)
```

✓ 0s

```
[5]  print(torch.cuda.is_available())
     print(torch.cuda.device_count())
     print(torch.cuda.get_device_name(0))
```

```
True
1
Tesla T4
```

📌 Step 2: Download and Prepare the Dataset

Next, we will download the dataset from the link provided and extract it.

```
[6]  !wget https://github.com/kaopanboonyuen/OCSB-AI/blob/main/dataset/pothole_dataset.zip?raw=true -O pothole_dataset.zip
     with zipfile.ZipFile('pothole_dataset.zip', 'r') as zip_ref:
         zip_ref.extractall('/content/')
```

```
--2025-03-15 09:06:33--  https://github.com/kaopanboonyuen/OCSB-AI/blob/main/dataset/pothole_dataset.zip?raw=true
Resolving github.com (github.com)... 20.205.243.166
Connecting to github.com (github.com)|20.205.243.166|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github.com/kaopanboonyuen/OCSB-AI/raw/refs/heads/main/dataset/pothole_dataset.zip [following]
--2025-03-15 09:06:34--  https://github.com/kaopanboonyuen/OCSB-AI/raw/refs/heads/main/dataset/pothole_dataset.zip
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/kaopanboonyuen/OCSB-AI/refs/heads/main/dataset/pothole_dataset.zip [following]
--2025-03-15 09:06:34--  https://raw.githubusercontent.com/kaopanboonyuen/OCSB-AI/refs/heads/main/dataset/pothole_dataset.zip
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 56836200 (54M) [application/zip]
Saving to: 'pothole_dataset.zip'

pothole_dataset.zip 100%[===================>]  54.20M  --.-KB/s    in 0.1s

2025-03-15 09:06:41 (369 MB/s) - 'pothole_dataset.zip' saved [56836200/56836200]
```

📌 Step 3: Organize Data in YOLOv12 Format

YOLOv12 requires the data to be organized in a specific format. We'll make sure that the labels and images are correctly set up.

0s

```
[7]  os.makedirs('/content/dataset', exist_ok=True)
     os.makedirs('/content/dataset/train', exist_ok=True)
     os.makedirs('/content/dataset/valid', exist_ok=True)

     shutil.move('/content/pothole_dataset/train/images', '/content/dataset/train/images')
     shutil.move('/content/pothole_dataset/train/labels', '/content/dataset/train/labels')
     shutil.move('/content/pothole_dataset/valid/images', '/content/dataset/valid/images')
     shutil.move('/content/pothole_dataset/valid/labels', '/content/dataset/valid/labels')
```

```
'/content/dataset/valid/labels'
```

📌 Step 4: Preview Dataset with Random Image Samples

```python
import random
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Path to the dataset
train_images_dir = '/content/dataset/train/images'
train_labels_dir = '/content/dataset/train/labels'

# Get a random image file
random_image_file = random.choice(os.listdir(train_images_dir))
random_image_path = os.path.join(train_images_dir, random_image_file)

# Load the image
image = cv2.imread(random_image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Load the corresponding label
label_file = random_image_file.replace('.jpg', '.txt')  # Assuming .jpg format for images
label_path = os.path.join(train_labels_dir, label_file)

# Read the label file (assume it's in YOLO format or polygon format)
with open(label_path, 'r') as file:
    labels = file.readlines()

# Plot the image and the segmentation label
plt.imshow(image_rgb)

for label in labels:
    parts = label.strip().split()

    # If the label has more than 5 values, it's likely a polygon format
    if len(parts) > 5:
        class_id = int(parts[0])  # Get the class ID (not used for visualization here)
        # Parse the polygon points from the label
        polygon_points = list(map(float, parts[1:]))  # All but the first are coordinates
```

Sample Image and Segmentation Polygon

📌 Step 5: Configure the Dataset YAML File

For YOLOv12 to know how to process the dataset, we need to create a `data.yaml` configuration file. This file specifies where the dataset is located and defines the class names.

```
[9]  data_yaml = """
     train: /content/dataset/train/images
     val: /content/dataset/valid/images

     nc: 1
     names: ['Pothole']
     """
     with open("/content/dataset/data.yaml", "w") as f:
         f.write(data_yaml)
```

📌 Step 6: Load the YOLOv12 Model

Now, load the YOLOv12 small model (YOLOv12n) from the ultralytics package.

```
[10] # Download the pretrained YOLOv12 model (if available)
     from ultralytics import YOLO

     # Load the pretrained model (adjust the URL or model name if necessary)
     model = YOLO("yolov10n.pt")  # Replace this with the correct path or model
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov10n.pt to 'yolov10n.pt'...
100%|██████████| 5.59M/5.59M [00:00<00:00, 298MB/s]
```

📌 Step 7: Train the Model

Now that we have everything ready, we can start training the model. We will fine-tune the pre-trained YOLOv12 small model on the pothole dataset.

```
[11] model.train(data='/content/dataset/data.yaml', epochs=50, imgsz=640, batch=16, device=0)  # Change device to your GPU
```

3m

```
Ultralytics 8.3.90 🚀 Python-3.11.11 torch-2.0.0+cu117 CUDA:0 (Tesla T4, 15095MiB)
engine/trainer: task=detect, mode=train, model=yolov10n.pt, data=/content/dataset/data.yaml, epochs=50, time=None, patience=100,
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100%|██████████| 755k/755k [00:00<00:00, 115MB/s]
Overriding model.yaml nc=80 with nc=1

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1      9856  ultralytics.nn.modules.block.SCDown          [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1     36096  ultralytics.nn.modules.block.SCDown          [128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 5]
 10                  -1  1    249728  ultralytics.nn.modules.block.PSA             [256, 256]
 11                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 12             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
```

📌 Step 7: Train the Model

Now that we have everything ready, we can start training the model. We will fine-tune the pre-trained YOLOv12 small model on the pothole dataset.

```
model.train(data='/content/dataset/data.yaml', epochs=50, imgsz=640, batch=16, device=0)  # Change device to your GPU
```

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/50      2.95G      2.793      6.665      2.809         47        640: 100%|██████████| 45/45 [00:16<00:00,  2.65it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:01<00:00,  1.09it

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/50      3.41G       2.99      5.688      2.946         75        640: 100%|██████████| 45/45 [00:20<00:00,  2.14it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:01<00:00,  1.67it

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       3/50      3.41G      3.052      4.935      2.908         86        640: 100%|██████████| 45/45 [00:13<00:00,  3.24it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:00<00:00,  2.65it

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       4/50      3.41G      3.112      4.581      2.979         81        640: 100%|██████████| 45/45 [00:14<00:00,  3.21it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:00<00:00,  3.00it

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       5/50      3.41G      3.056      4.181      2.911         71        640: 100%|██████████| 45/45 [00:14<00:00,  3.19it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:00<00:00,  3.37it
```

📌 Step 8: Evaluate the Model

Once training is complete, evaluate the model's performance on the validation dataset.

```
[12] results = model.val()  # Evaluate on validation set
```

```
Ultralytics 8.3.90 🚀 Python-3.11.11 torch-2.0.0+cu117 CUDA:0 (Tesla T4, 15095MiB)
YOLOv10n summary (fused): 125 layers, 2,694,806 parameters, 0 gradients, 8.2 GFLOPs
val: Scanning /content/dataset/valid/labels.cache... 60 images, 0 backgrounds, 0 corrupt: 100%|██████████| 60/60 [00:00<?, ?
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:02<00:00,
                   all         60        201      0.666      0.672      0.686      0.429
Speed: 10.0ms preprocess, 10.9ms inference, 0.0ms loss, 0.2ms postprocess per image
Results saved to runs/detect/train2
```

📌 Step 9: Inference and Error Analysis

After evaluation, it's time to run inference on some images and analyze the errors. Let's run inference on a few images and visualize the predictions.

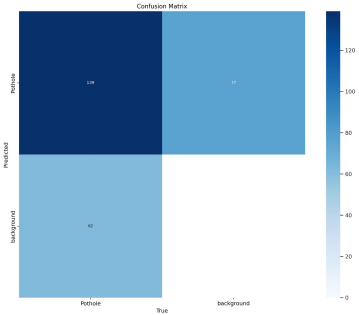
```python
# Perform inference on validation images
results = model.predict('/content/dataset/valid/images', save=True)

# Show inference results for a specific image
image_path = '/content/dataset/valid/images/sample_image.jpg'

# Check if the image exists
if os.path.exists(image_path):
    # Read the image
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB for matplotlib
    plt.imshow(img_rgb)
    plt.axis('off')  # Hide axes
    plt.show()

    # Get the inference results for this image
    # This will display the prediction (boxes, labels, etc.)
    prediction_results = results.pandas().xywh  # Get results in xywh format
    print(prediction_results)  # Print predictions
else:
    print(f"Image not found at {image_path}")
```

```
image 1/60 /content/dataset/valid/images/pic-114-_jpg.rf.a0f30e06b3b96d7879d5f55a7012433c.jpg: 640x640
image 2/60 /content/dataset/valid/images/pic-116-_jpg.rf.ddcb9d0e0bcf2a1a5096c4f04e6b7f9e.jpg: 640x640
image 3/60 /content/dataset/valid/images/pic-123-_jpg.rf.385ae3fbcdabda81f72ddf11f6a4b93d.jpg: 640x640
```



📌 Step 10: Confusion Matrix and Precision-Recall-F1 Analysis

Evaluate the predictions using metrics like Precision, Recall, and F1-Score:

0s

```
from IPython.display import Image, display

# Path to the confusion matrix image
confusion_matrix_path = '/content/runs/detect/train/confusion_matrix.png'

# Display the confusion matrix image
display(Image(filename=confusion_matrix_path))
```

📌 Step 11: Save and Load Model

Finally, save your trained model and load it again when needed.

✓ 0s

```
[19] # Save the model
     model.save('/content/yolov12n_pothole.pt')

     # Load the model
     model = YOLO('/content/yolov12n_pothole.pt')
```

Let's execute some code.