ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ **รหัสวิชา 04-624-201**
มหาวิทยาลัยเทคโนโลยีราชมงคลธัญบุรี **ชื่อวิชา** Machine Learning
ชื่อ นายสิทธิกร บุญณะ รหัสนักศึกษา 116630462039-4 กลุ่ม 1

ใบงานที่ 2 Neural Network
และการประยุกต์ใช้งาน NN

วัตถุประสงค์

1. เพื่อให้นักศึกษาเข้าใจหลักการทำงานของ Neural Network (NN) และแนวทางการประยุกต์ใช้ในงานจริงทั้งด้านการจำแนกและการพยากรณ์
2. เพื่อให้นักศึกษาสามารถพัฒนาผังการ ทำงาน ประมวลผล และแสดงผลลัพธ์ เพื่อพัฒนาระบบต้นแบบร่วมกับ NN ในการจำแนกและการพยากรณ์ได้อย่างถูกต้อง

LAB1: NN Basic for classify digits

Objective:

- Apply NN to classify digits.

Contents:

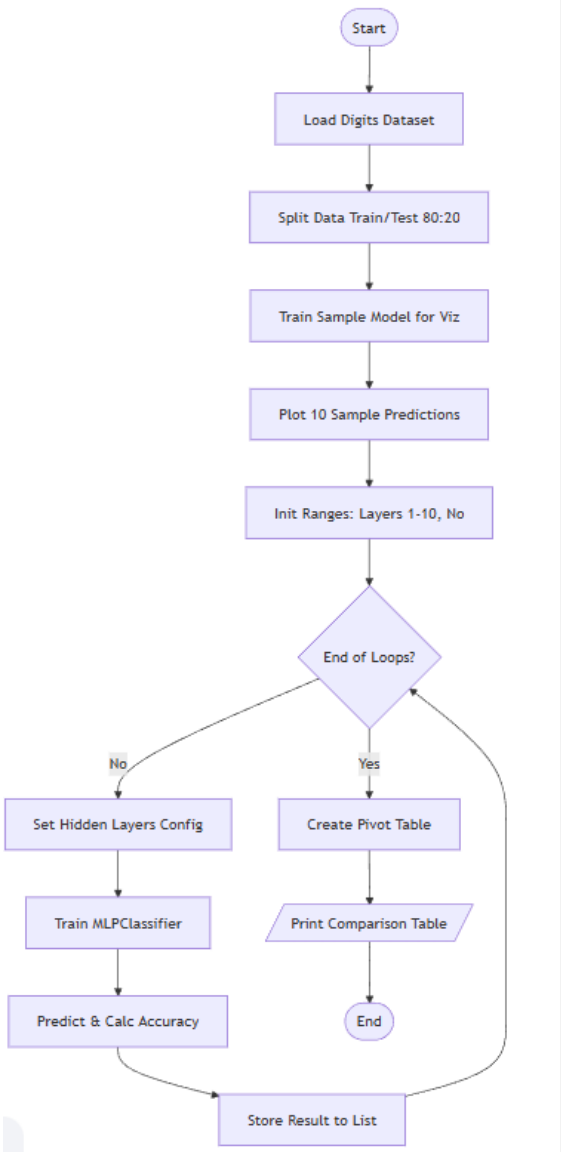
- Load the digits dataset.
- Split the data into training and testing sets.
- Train NN models.
- Evaluate each model using accuracy.

Output:

- Display sample predictions
- Compare the results of different network sizes in a table (1–10 layers with 10–1000 nodes per layer)

Code:	Flowchart:
import numpy as np import numpy as np import pandas as pd import matplotlib.pyplot as plt from sklearn.datasets import load_digits from sklearn.model_selection import train_test_split	

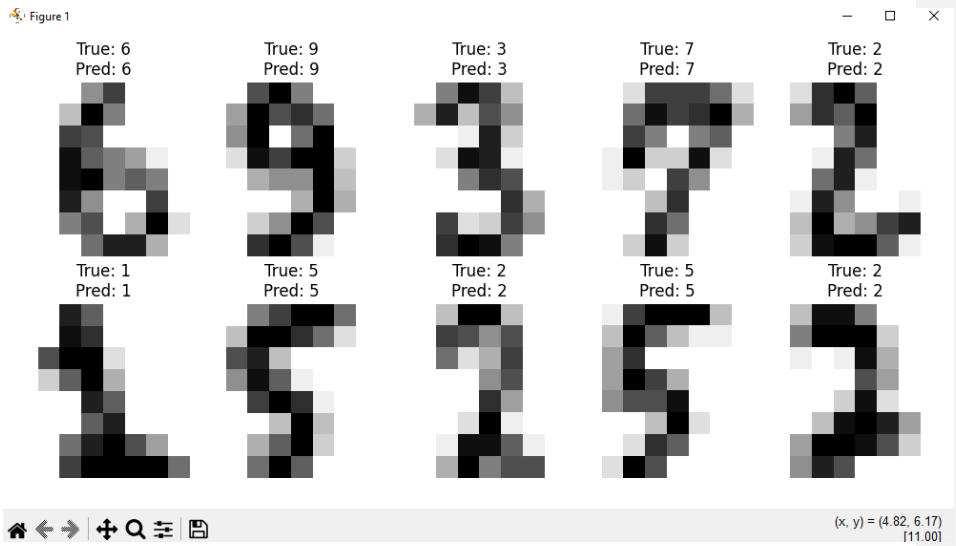
```
from sklearn.neural_network import
MLPClassifier
from sklearn.metrics import accuracy_score
import warnings

warnings.filterwarnings('ignore')

def main():
    print("Loading digits dataset...")
    digits = load_digits()
    X = digits.data
    y = digits.target

    X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)

    print(f"Data split: {X_train.shape[0]} training
samples, {X_test.shape[0]} testing samples.")

    print("\nTraining a sample model for
visualization...")
    sample_mlp =
MLPClassifier(hidden_layer_sizes=(100,),
max_iter=500, random_state=42)
    sample_mlp.fit(X_train, y_train)

    y_pred_sample =
sample_mlp.predict(X_test)
    sample_accuracy = accuracy_score(y_test,
y_pred_sample)
    print(f"Sample Model Accuracy:
{sample_accuracy:.4f}")

    print("Displaying sample predictions...")
    fig, axes = plt.subplots(2, 5, figsize=(10, 5))
```



```
    for i, ax in enumerate(axes.flat):
        ax.imshow(X_test[i].reshape(8, 8),
cmap='binary')
        ax.set_title(f"True: {y_test[i]}\nPred:
{y_pred_sample[i]}")
        ax.axis('off')
    plt.tight_layout()
    plt.show()

    print("\nComparing different network sizes
(this may take a while)...")

    layers_range = range(1, 11)
    nodes_range = [10, 50, 100, 500, 1000]

    results = []

    for layers in layers_range:
        for nodes in nodes_range:
            hidden_layers = tuple([nodes] * layers)

            mlp =
MLPClassifier(hidden_layer_sizes=hidden_layers,
max_iter=200, random_state=42)
            mlp.fit(X_train, y_train)

            y_pred = mlp.predict(X_test)
            acc = accuracy_score(y_test, y_pred)

            print(f"Layers: {layers}, Nodes/Layer:
{nodes} -> Accuracy: {acc:.4f}")

            results.append({
                'Hidden Layers': layers,
                'Nodes per Layer': nodes,
                'Accuracy': acc
```

```
        })

    results_df = pd.DataFrame(results)

    pivot_table = results_df.pivot(index='Hidden Layers', columns='Nodes per Layer', values='Accuracy')

    print("\nComparison Results (Accuracy):")
    print(pivot_table)

if __name__ == "__main__":
    main()
```

Result:



ให้ข้อคิดเห็น [ชบ1]: AI
จำแนกตัวเลขเขียนด้วยมือ (Handwritten Digits Classification) โดยใช้ **Neural Network (MLP)** มีขั้นตอนหลักดังนี้:

1.**เตรียมข้อมูล (Data Preparation):**
(1)
•ใช้ชุดข้อมูล **Digits** (รูปภาพ 8x8 พิกเซล ของตัวเลข 0-9)
•แบ่งข้อมูลเป็น **Train 80%** และ **Test 20%**

1.**ตรวจสอบเบื้องต้น (Visualization):**
(1)
(1)สร้างโมเดลตัวอย่างเพื่อทดสอบและแสดงผลเป็นรูปภาพ 10 รูป
(2)เปรียบเทียบค่าจริง (**True**) กับค่าที่ทำนาย (**Pred**) เพื่อยืนยันว่าโมเดลทำงานถูกต้อง

1.**การทดลองเปรียบเทียบโครงสร้าง (Experiment):**
(1)
(1)ทำการทดลองซ้ำๆ (Loop) โดยปรับเปลี่ยนขนาดของ Network 2 ตัวแปร:
(2)
•**จำนวนชั้น (Hidden Layers):** ตั้งแต่ 1 ถึง 10 ชั้น
•**จำนวนโหนด (Nodes):** 10, 50, 100, 500, และ 1000 โหนด

LEB 2: NN on the Face recognition.

Objective:

- Apply NN to Face recognition.

Contents:

- Load face recognition library.
- Split the data into training and testing sets.
- Train NN models.
- Evaluate each model using accuracy.

Output:

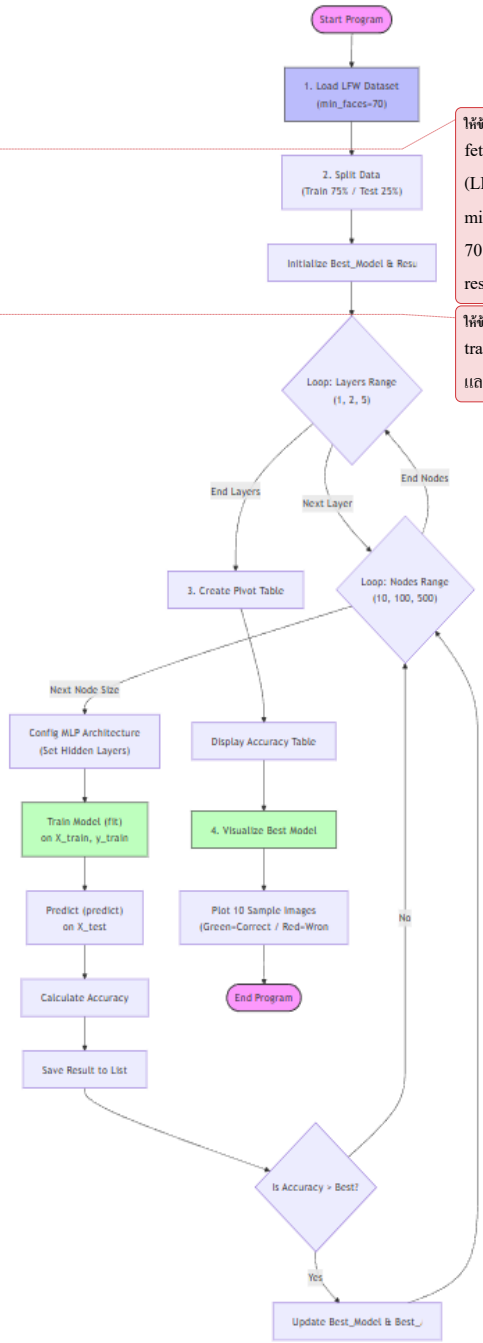
- Display sample predictions
- Compare the results of different network sizes in a table (1–10 layers with 10–1000 nodes per layer)

Code:	Flowchart:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import
train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import warnings
 	
warnings.filterwarnings('ignore')
 	
def main():
  print("Loading Face Dataset (LFW)...")
  lfw_people =
fetch_lfw_people(min_faces_per_person=70,
resize=0.4)
 	
  n_samples, h, w = lfw_people.images.shape
  X = lfw_people.data
  y = lfw_people.target
  target_names = lfw_people.target_names	

ให้ข้อคิดเห็น [ตบ2]: AI
ปิด Warning เพื่อความสะอาดของ Output

```
    n_classes = target_names.shape[0]

    print(f"Dataset Info: {n_samples} samples, {n_classes} classes, Image size: {h}x{w}")

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=42
    )

    print("\nComparing different network sizes...")

    layers_range = [1, 2, 5]
    nodes_range = [10, 100, 500]

    results = []

    best_model = None
    best_acc = 0

    for layers in layers_range:
        for nodes in nodes_range:
            hidden_layers = tuple([nodes] * layers)
            print(f" -> Training: {layers} Layers x {nodes} Nodes...")
            mlp = MLPClassifier(hidden_layer_sizes=hidden_layers, max_iter=200, random_state=42)
            mlp.fit(X_train, y_train)
            y_pred = mlp.predict(X_test)
            acc = accuracy_score(y_test, y_pred)
            results.append({
                'Layers': layers,
                'Nodes/Layer': nodes,
                'Accuracy': acc
            })
            if acc > best_acc:
```



ให้ข้อคิดเห็น **[ตบ3]:** AI
fetch_lfw_people: คำสั่งนี้จะไปโหลดรูปภาพใบหน้าคนดัง (LFW Dataset) มาจาก Server ของ Scikit-learn
min_faces_per_person=70: เลือกโหลดเฉพาะคนที่มีรูปมากกว่า 70 รูป เพื่อให้โมเดลมีตัวอย่างสอนเยอะพอจนจำแนกได้
resize=0.4: ย่อรูปให้เล็กลง เพื่อให้เทรนไวขึ้น

ให้ข้อคิดเห็น **[ตบ4]:** AI
train_test_split: แบ่งข้อมูลเป็น 2 กอง คือ Train (สำหรับสอน) และ Test (สำหรับสอบ) สัดส่วน 75:25 (เพราะ test_size=0.25)

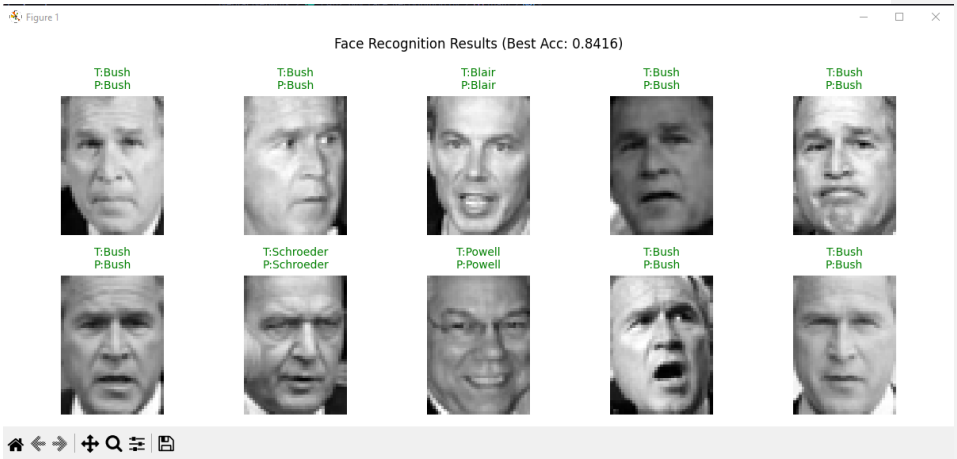
```
                best_acc = acc
                best_model = mlp
    results_df = pd.DataFrame(results)
    pivot_table = results_df.pivot(index='Layers',
columns='Nodes/Layer', values='Accuracy')
    print("\n" + "="*40)
    print(" Comparison Results (Accuracy) ")
    print("="*40)
    print(pivot_table)
    print("="*40)
    if best_model is not None:
        print("\nDisplaying sample predictions form
Best Model...")
        y_pred_best = best_model.predict(X_test)
        plt.figure(figsize=(12, 5))
        for i in range(10):
            plt.subplot(2, 5, i + 1)
            plt.imshow(X_test[i].reshape(h, w),
cmap='gray')
            true_name =
target_names[y_test[i]].split()[-1]
            pred_name =
target_names[y_pred_best[i]].split()[-1]
            color = 'green' if y_pred_best[i] ==
y_test[i] else 'red'
            plt.title(f"T:{true_name}\nP:{pred_name}",
color=color, fontsize=10)
            plt.axis('off')
        plt.suptitle(f"Face Recognition Results (Best
Acc: {best_acc:.4f})")
        plt.tight_layout()
        plt.show()
    else:
        print("No model trained.")
if __name__ == "__main__":
    main()
```

ให้ข้อคิดเห็น [ตบ5]: AI Compare Network Sizes (ตามโจทย์ 1-10 Layers, 10-1000 Nodes)
กำหนดช่วงที่จะทดสอบ (ลดจำนวนลงเล็กน้อยเพื่อให้รันเสร็จไวขึ้นสำหรับการทดสอบ)
Loop Compare Network Sizes: ส่วนนี้คือหัวใจสำคัญ
layers_range: เก็บจำนวนชั้น Hidden Layer ที่จะทดสอบ (เช่น 1, 2, 5 ชั้น)
nodes_range: เก็บจำนวน Node ต่อชั้น (เช่น 10, 100, 500 Nodes)
โปรแกรมจะวนลูปจับคู่ "ชั้น x โหนด" ทุกแบบ แล้วสร้าง MLPClassifier มาเทรน

ให้ข้อคิดเห็น [ตบ6]: Display Results Table

ให้ข้อคิดเห็น [ตบ7]: AI ตอนจบจะเอาผล Accuracy ทั้งหมดมาตีตารางให้ดูง่ายๆ ว่าโครงสร้างแบบไหนฉลาดที่สุด
Visualization: เอารูปภาพจาก Test set 10 รูปมาให้ best_model ทายดู
ถ้าชื่อสี เขียว แปลว่าทายถูก
ถ้าชื่อสี แดง แปลว่าทายผิด

Result:




```
Loading Face Dataset (LFW)... (อาจใช้เวลาดาวน์โหลดสักครู่ในครั้งแรก)
Dataset Info: 1288 samples, 7 classes, Image size: 50x37

Comparing different network sizes... (Process might take time)
 -> Training: 1 Layers x 10 Nodes...
 -> Training: 1 Layers x 100 Nodes...
 -> Training: 1 Layers x 500 Nodes...
 -> Training: 2 Layers x 10 Nodes...
 -> Training: 2 Layers x 100 Nodes...
 -> Training: 2 Layers x 500 Nodes...
 -> Training: 5 Layers x 10 Nodes...
 -> Training: 5 Layers x 100 Nodes...
 -> Training: 5 Layers x 500 Nodes...

========================================
 Comparison Results (Accuracy)
========================================
Nodes/Layer        10        100       500
Layers
1            0.736025  0.841615  0.841615
2            0.655280  0.841615  0.804348
5            0.652174  0.782609  0.742236
========================================

Displaying sample predictions form Best Model...
```

LEB 3: NN on Iris.csv. Classification.

Link: https://www.kaggle.com/datasets/saurabh00007/iriscsv

Objective:

- Apply NN to Iris.csv.

Contents:

- Load the iris.csv data from directory dataset.
- Split the data into training and testing sets.
- Train NN models using 5 layers (1 layer: 50 Node)
- Evaluate each model using accuracy.

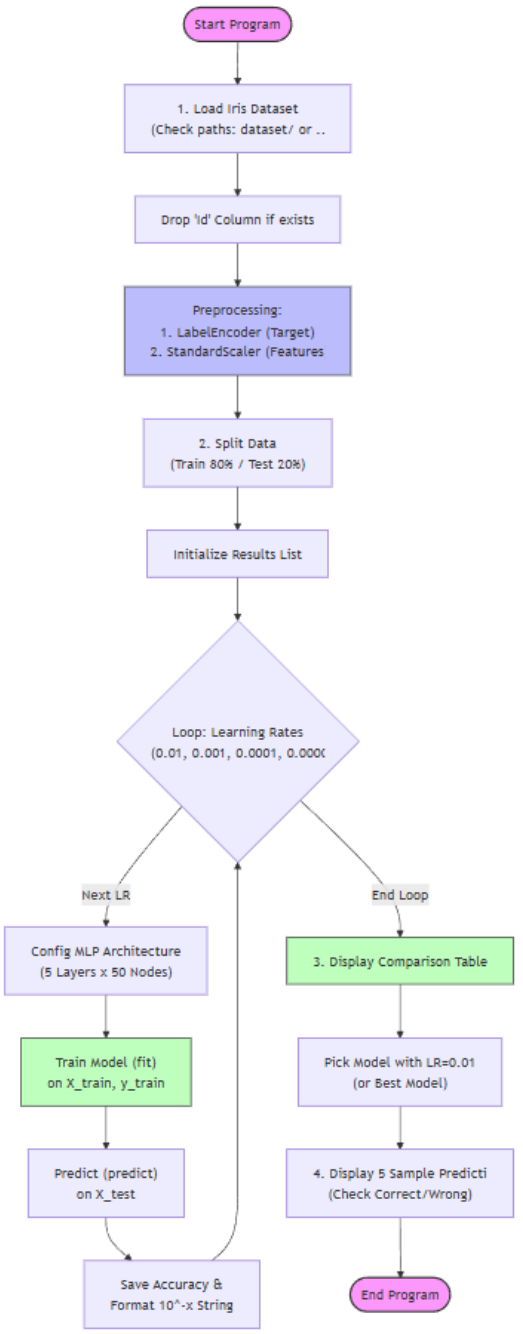
Output:

- Display sample predictions.
- Compare the results of different learning rates in a table (10^{-2}, to 10^{-5})

Code:	Flowchart:

import pandas as pd
import numpy as np
from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import LabelEncoder,
StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import warnings
import os
warnings.filterwarnings('ignore')
def main():
  print("Loading Iris dataset...")
  possible_paths = ['dataset/Iris.csv',
'../dataset/Iris.csv']
  dataset_path = None
  for path in possible_paths:
  if os.path.exists(path):
  dataset_path = path
  break
  if dataset_path is None:	

```
        print("Error: ไม่พบไฟล์ 'dataset/Iris.csv'")
        print("กรุณาสร้างโฟลเดอร์ 'dataset' และใส่ไฟล์ Iris.csv ลงไป หรือแก้ไข path ในโค้ด")

        return
    df = pd.read_csv(dataset_path)
    if 'Id' in df.columns:
        df = df.drop('Id', axis=1)
    X = df.iloc[:, :-1]
    y = df.iloc[:, -1]
    le = LabelEncoder()
    y = le.fit_transform(y)
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )
    learning_rates = [0.01, 0.001, 0.0001, 0.00001]
    hidden_layers_config = (50, 50, 50, 50, 50)
    results = []
    models = {}
    print(f"\nTraining NN with 5 Layers {hidden_layers_config}...")
    print(f"{'Learning Rate':<15} | {'Accuracy':<10}")
    print("-" * 30)
    for lr in learning_rates:
        mlp = MLPClassifier(
            hidden_layer_sizes=hidden_layers_config,
            learning_rate_init=lr,
            max_iter=1000,
            random_state=42
        )
        mlp.fit(X_train, y_train)
        y_pred = mlp.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
```



```
        results.append({
            'Learning Rate': lr,
            'LR Scientific': f"10^{int(np.log10(lr))}",
            'Accuracy': acc
        })
        models[lr] = mlp
        print(f"{lr:<15} | {acc:.4f}")
    results_df = pd.DataFrame(results)
    results_df = results_df[['LR Scientific', 'Learning
Rate', 'Accuracy']]

    print("\n" + "="*40)
    print(" Comparison Results (Learning Rate) ")
    print("="*40)
    print(results_df)
    print("="*40)
    best_lr = 0.01
    best_model = models[best_lr]
    print(f"\nSample Predictions (using model with
LR={best_lr}):")
    sample_indices =
np.random.choice(len(X_test), 5, replace=False)

 X_sample = X_test[sample_indices]
    y_sample_true = y_test[sample_indices

y_sample_pred = best_model.predict(X_sample)
    y_true_names =
le.inverse_transform(y_sample_true)

 y_pred_names =
le.inverse_transform(y_sample_pred)
    for i in range(5):
        status = "✅ Correct" if y_sample_true[i]
== y_sample_pred[i] else "❌ Wrong"
```

```
        print(f"Sample {i+1}:
True=[{y_true_names[i]:<15}] vs
Pred=[{y_pred_names[i]:<15}] -> {status}")
if __name__ == "__main__":
    main()
```

Result:

```
Loading Iris dataset...

Training NN with 5 Layers (50, 50, 50, 50, 50)...
Learning Rate    | Accuracy
------------------------------
0.01             | 0.9667
0.001            | 1.0000
0.0001           | 1.0000
1e-05            | 0.7000

========================================
 Comparison Results (Learning Rate)
========================================
  LR Scientific  Learning Rate  Accuracy
0         10^-2        0.01000  0.966667
1         10^-3        0.00100  1.000000
2         10^-4        0.00010  1.000000
3         10^-5        0.00001  0.700000
========================================

Sample Predictions (using model with LR=0.01):
Sample 1: True=[Iris-setosa    ] vs Pred=[Iris-setosa    ] -> ✅ Correct
Sample 2: True=[Iris-setosa    ] vs Pred=[Iris-setosa    ] -> ✅ Correct
Sample 3: True=[Iris-setosa    ] vs Pred=[Iris-setosa    ] -> ✅ Correct
Sample 4: True=[Iris-virginica ] vs Pred=[Iris-virginica ] -> ✅ Correct
Sample 5: True=[Iris-setosa    ] vs Pred=[Iris-setosa    ] -> ✅ Correct
```

ให้ข้อคิดเห็น **[ตบ8]: AI**

ส่วนที่ 1: การนำเข้าเครื่องมือ (Import Libraries)

ส่วนบนสุดของโค้ดคือการเรียกใช้ Library ที่จำเป็นสำหรับงาน Data Science และ Machine Learning:

- **pandas**: ใช้จัดการข้อมูลในรูปแบบตาราง (DataFrame) และอ่านไฟล์ CSV
- **numpy**: ใช้สำหรับการคำนวณทางคณิตศาสตร์และจัดการ Array
- **sklearn (Scikit-learn)**: เป็นพระเอกหลักที่ใช้สำหรับ:
  -
  - แบ่งข้อมูล (train_test_split)
  - เตรียมข้อมูล (LabelEncoder, StandardScaler)
  - สร้างโมเดล Neural Network (MLPClassifier)
  - วัดผล (accuracy_score)

ส่วนที่ 2: การโหลดไฟล์ข้อมูล (Load Data with Path Check)

โค้ดส่วนนี้ออกแบบมาให้ **Robust (แข็งแรง)** ป้องกัน Error เรื่องหาไฟล์ไม่เจอ:

1. **possible_paths**: กำหนดรายชื่อที่อยู่ไฟล์ที่เป็นไปได้ (เช่น dataset/Iris.csv หรือ ../dataset/Iris.csv)
2. **for path in possible_paths**: วนลูปเช็คว่าไฟล์อยู่ที่ไหน ถ้าเจอ (os.path.exists) ให้จำ Path นั้นไว้แล้วหยุดหา
3. **Error Handling**: ถ้าหาไม่เจอเลย จะแจ้งเตือนให้ผู้ใช้สร้างโฟลเดอร์หรือแก้ Path ก่อนจบโปรแกรม

ส่วนที่ 3: การเตรียมข้อมูล (Preprocessing)

เมื่อโหลดข้อมูลมาแล้ว ต้องทำความสะอาดและแปลงร่างก่อนส่งให้ AI เรียนรู้:

1. **Drop ID**: ลบคอลัมน์ Id ทิ้ง เพราะเป็นแค่เลขรหัส ไม่ช่วยในการแยกแยะสายพันธุ์ดอกไม้
2. **แยก X, y**:
   (1)
   - X (Features): ข้อมูลนำเข้า (ความกว้าง/ยาวของกลีบ)
   - y (Target): คำตอบที่ต้องการทำนาย (ชื่อสายพันธุ์)
1. **Label Encoding**: แปลงชื่อสายพันธุ์ที่เป็นตัวหนังสือ (เช่น "Iris-setosa") ให้เป็นตัวเลข (0, 1, 2) เพื่อให้คอมพิวเตอร์เข้าใจ
2. **Standard Scaler**: สำคัญมากสำหรับ Neural Network คือการปรับค่าข้อมูลทุกคอลัมน์ให้อยู่ในสเกลเดียวกัน (Mean=0, Std=1) เพื่อให้โมเดลเรียนรู้ได้เร็วและแม่นยำ

ส่วนที่ 4: การแบ่งข้อมูล (Train/Test Split)

Python
X_train, X_test, y_train, y_test = train_test_split(...)

แบ่งข้อมูลออกเป็น 2 กอง:

- **Train (80%)**: เอาไว้สอนโมเดล (ให้เรียนรู้)
- **Test (20%)**: เอาไว้สอบวัดผล (ห้ามแอบดูเฉลย)
- random_state=42: ล็อคผลการสุ่มให้เหมือนเดิมทุกครั้งที่รัน (เพื่อให้ผลการทดลองนิ่ง)

ส่วนที่ 5: การทดลองและเทรนโมเดล (Experiment Loop)

...

LEB 4: NN on Microscopic Fungi Classification

Link: https://www.kaggle.com/datasets/anshtanwar/microscopic-fungi-images

Objective:

- Apply NN to Fungi classify.

Contents:

- Load the Fungi dataset from directory (Link).
- Split the data into training and testing sets.
- Train NN models using 5 layers (1 layer: 100 Node)
- Evaluate each model using accuracy.

Output:

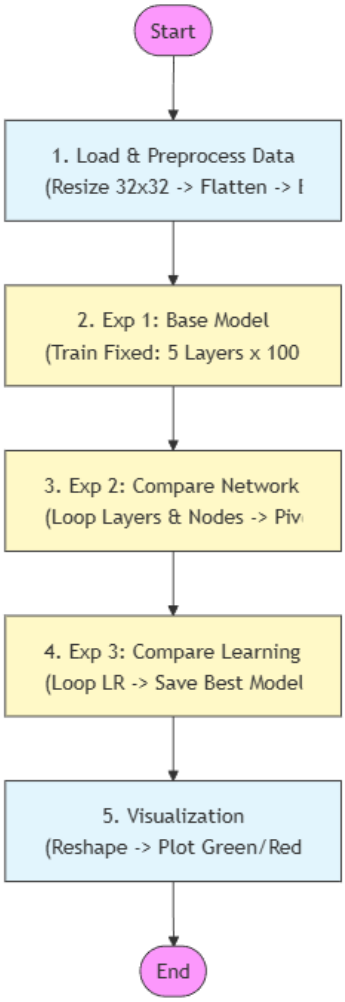
- Classification accuracy on the test set.
- Compare the results of different network sizes in a table (1–10 layers with 10–1000 nodes per layer)
- Compare the results of different learning rates in a table (10^{-2}, to 10^{-5})

Code or Algorithms:	Flowchart:

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from PIL import Image
import warnings
warnings.filterwarnings('ignore')
 	
def load_images_from_folder(folder, image_size=(32, 32)):
  images = []
  labels = []
 	
  if not os.path.exists(folder):
  print(f"Error: Folder not found: {folder}")	

ให้ข้อคิดเห็น **[ตบ9]:** *ปิด Warning เพื่อความสะอาดของ Output*

```
        return np.array(images), np.array(labels)

    print(f"Scanning folder: {folder}")

    # วนลูปตามโฟลเดอร์ของแต่ละคลาส (class_name)
    for filename in os.listdir(folder):
        class_path = os.path.join(folder, filename)

        # ถ้าเป็นโฟลเดอร์ (คือ Class)
        if os.path.isdir(class_path):
            class_name = filename
            # อ่านรูปภาพใน Class นั้นๆ
            for img_name in os.listdir(class_path):
                img_path = os.path.join(class_path,
img_name)
                try:
                    with Image.open(img_path) as img:
                        # Resize รูปให้เล็กลงเพื่อลดจำนวน
Features (เช่น 32x32 = 1024 pixels)
                        img =
img.resize(image_size).convert('RGB')
                        # แปลงเป็น numpy array และ Normalize
(0-1)
                        img_array = np.array(img).flatten() /
255.0

                        images.append(img_array)
                        labels.append(class_name)
                except Exception as e:
                    pass
        # กรณี dataset ไม่มี subfolder (รูปกองรวมกัน) - ข้ามไป
ก่อน หรือเขียน logic เพิ่ม

    return np.array(images), np.array(labels)

def main():
```



```
    print("="*50)
    print(" LAB 4: NN on Microscopic Fungi Classification")
    print("="*50)

    # ----------------------------------------------------------
    # 1. Load Dataset
    # ----------------------------------------------------------
    train_dir = "../dataset/fungi/train"
    test_dir = "../dataset/fungi/test"

    # Fallback paths
    if not os.path.exists(train_dir): train_dir =
"dataset/fungi/train"
    if not os.path.exists(test_dir): test_dir =
"dataset/fungi/test"

    print("\nLoading Training Data...")
    X_train, y_train_labels =
load_images_from_folder(train_dir, image_size=(32, 32))

    print("\nLoading Testing Data...")
    X_test, y_test_labels =
load_images_from_folder(test_dir, image_size=(32, 32))

    if len(X_train) == 0:
        print("Error: No training data found. Please check
dataset path.")
        return

    print(f"\nData Loaded:")
    print(f"Train: {X_train.shape} samples")
    print(f"Test:  {X_test.shape} samples")

    # Encode Labels
    le = LabelEncoder()
    y_train = le.fit_transform(y_train_labels)
```

```
    y_test = le.transform(y_test_labels)

    classes = le.classes_
    print(f"Classes: {classes}")

    # ----------------------------------------------------------
    # 2. Experiment 1: 5 Layers (100 Nodes each)
    # ----------------------------------------------------------
    print("\n" + "-"*50)
    print(" Experiment 1: 5 Layers x 100 Nodes")
    print("-" * 50)

    mlp_base = MLPClassifier(hidden_layer_sizes=(100,
100, 100, 100, 100), max_iter=200, random_state=42)
    mlp_base.fit(X_train, y_train)

    acc_base = mlp_base.score(X_test, y_test)
    print(f"Base Model Accuracy: {acc_base:.4f}")

    # ----------------------------------------------------------
    # 3. Experiment 2: Compare Network Sizes
    # ----------------------------------------------------------
    print("\n" + "-"*50)
    print(" Experiment 2: Compare Network Sizes (Layers &
Nodes)")
    print("-" * 50)

    layers_test = [1, 3, 5]
    nodes_test = [10, 50, 100]

    results_size = []

    for l in layers_test:
        for n in nodes_test:
            hidden_layers = tuple([n] * l)
            print(f"Training: {l} Layers, {n} Nodes...")
```

```
            mlp =
MLPClassifier(hidden_layer_sizes=hidden_layers,
max_iter=100, random_state=42)
            mlp.fit(X_train, y_train)
            acc = mlp.score(X_test, y_test)

            results_size.append({'Layers': l, 'Nodes': n,
'Accuracy': acc})

    df_size = pd.DataFrame(results_size)
    pivot_size = df_size.pivot(index='Layers',
columns='Nodes', values='Accuracy')
    print("\nNetwork Size Comparison (Accuracy):")
    print(pivot_size)

    # ----------------------------------------------------------
    # 4. Experiment 3: Compare Learning Rates
    # ----------------------------------------------------------
    print("\n" + "-"*50)
    print(" Experiment 3: Compare Learning Rates (10^-2
to 10^-5)")
    print("-" * 50)

    learning_rates = [0.01, 0.001, 0.0001, 0.00001]
    results_lr = []

    best_model = None
    best_acc = 0

    # ใช้โครงสร้างเดิม (5 Layers, 100 Nodes) เพื่อทดสอบ LR
    fixed_structure = (100, 100, 100, 100, 100)

    for lr in learning_rates:
        print(f"Training LR: {lr}...")
```

```
        mlp =
MLPClassifier(hidden_layer_sizes=fixed_structure,
learning_rate_init=lr, max_iter=200, random_state=42)
        mlp.fit(X_train, y_train)
        acc = mlp.score(X_test, y_test)

        results_lr.append({
            'Learning Rate': lr,
            'LR Scientific': f"10^{int(np.log10(lr))}",
            'Accuracy': acc
        })

        if acc >= best_acc:
            best_acc = acc
            best_model = mlp

    df_lr = pd.DataFrame(results_lr)[['LR Scientific',
'Learning Rate', 'Accuracy']]
    print("\nLearning Rate Comparison:")
    print(df_lr)

    # ----------------------------------------------------------
    # 5. Sample Predictions
    # ----------------------------------------------------------
    print("\n" + "-"*50)
    print(" Sample Predictions (Best Model)")
    print("-" * 50)

    # สุ่มรูปมาแสดง 10 รูป
    indices = np.random.choice(len(X_test), 10,
replace=False)

    plt.figure(figsize=(15, 6))
    for i, idx in enumerate(indices):
        image = X_test[idx].reshape(32, 32, 3) # Reshape
กลับเป็นรูปภาพ (32x32x3)
```

```
        true_label = le.inverse_transform([y_test[idx]])[0]
        pred_label = le.inverse_transform([best_model.predict([X_test[idx]])])[0]

        color = 'green' if true_label == pred_label else 'red'

        plt.subplot(2, 5, i+1)
        plt.imshow(image)
        plt.title(f"True: {true_label}\nPred: {pred_label}", color=color)
        plt.axis('off')

    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```

ให้ข้อคิดเห็น **[ตบ10]:** AI

ช่วงโค้ด: ตั้งแต่ import ... จนจบฟังก์ชัน load_images_from_folder

- **สิ่งที่ทำ:**
  -
    - นำเข้า Library ที่จำเป็น (NumPy, Pandas, Sklearn, PIL, Matplotlib)
    - สร้างฟังก์ชัน load_images_from_folder: เปรียบเสมือน "คนงาน" ที่เดินเข้าไปในโฟลเดอร์
  - **Resize:** ย่อรูปเหลือ 32x32 pixel (เพื่อลดภาระการคำนวณ)
  - **Flatten:** แปลงรูปภาพที่เป็นแผ่นสี่เหลี่ยม ให้กลายเป็น **"เส้นตรงยาวๆ" (Vector)** เพราะ MLPClassifier รับข้อมูลแบบเส้นตรงเท่านั้น
  - **Normalize:** หารด้วย 255.0 เพื่อปรับค่าสีจาก 0-255 ให้เหลือช่วง 0-1 (ช่วยให้ AI เรียนรู้ไวขึ้น)

2. ส่วนโหลดข้อมูลและแปลง

ช่วงโค้ด: เริ่มต้น def main(): ... ถึง print(f"Classes: {classes}")

- **สิ่งที่ทำ:**
  -
    - กำหนด Path ของโฟลเดอร์รูปภาพ (มีระบบกัน Error ถ้าหาไฟล์ไม่เจอ)
    - เรียกใช้ฟังก์ชันจากข้อ 1 มาโหลดข้อมูลใส่ตัวแปร X_train (รูปภาพ) และ y_train_labels (ชื่อคลาส)
  - **Label Encoder:** แปลงชื่อเชื้อรา (เช่น "H1", "H2") ให้เป็นตัวเลข (0, 1) เพื่อให้คอมพิวเตอร์นำไปคำนวณต่อได้

ผมจะแบ่งคำอธิบายโดยจับคู่กับ **"ช่วงบรรทัดของโค้ด"** ให้เห็นภาพชัดเจนนะครับ คุณสามารถนำหัวข้อเหล่านี้ไปเขียนลงในรายงานหรือสไลด์นำเสนอได้เลยครับ

1. ส่วนเตรียมการและฟังก์ชันจัดการรูปภาพ (บรรทัดที่ 1 - 42)

ช่วงโค้ด: ตั้งแต่ import ... จนจบฟังก์ชัน load_images_from_folder

- **สิ่งที่ทำ:**
  -
    - นำเข้า Library ที่จำเป็น (NumPy, Pandas, Sklearn, PIL, Matplotlib)
    - สร้างฟังก์ชัน load_images_from_folder: เปรียบเสมือน "คนงาน" ที่เดินเข้าไปในโฟลเดอร์
  - **Resize:** ย่อรูปเหลือ 32x32 pixel (เพื่อลดภาระการคำนวณ)
  - **Flatten:** แปลงรูปภาพที่เป็นแผ่นสี่เหลี่ยม ให้กลายเป็น **"เส้นตรงยาวๆ" (Vector)** เพราะ MLPClassifier รับข้อมูลแบบเส้นตรงเท่านั้น
  - **Normalize:** หารด้วย 255.0 เพื่อปรับค่าสีจาก 0-255 ให้เหลือช่วง 0-1 (ช่วยให้ AI เรียนรู้ไวขึ้น)

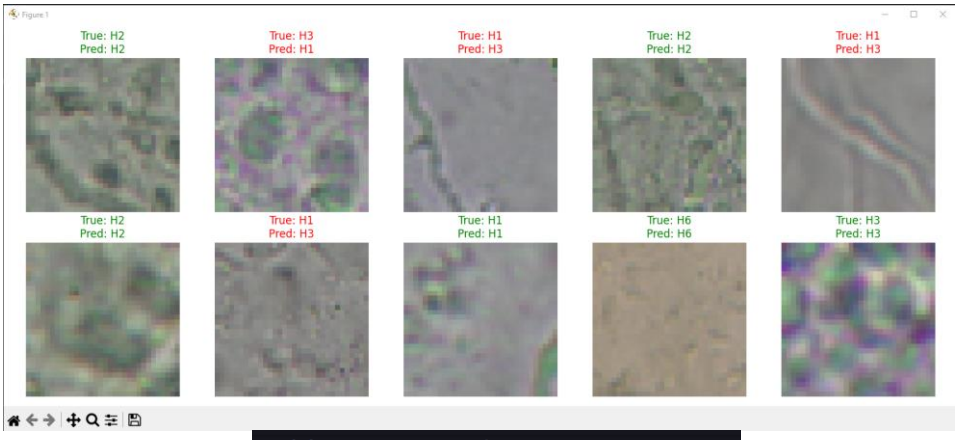
2. ส่วนโหลดข้อมูลและแปลงฉลาก (บรรทัดที่ 44 - 83)

ช่วงโค้ด: เริ่มต้น def main(): ... ถึง print(f"Classes: {classes}")

- **สิ่งที่ทำ:**
  -

...

Result:




```
Training: 1 Layers, 10 Nodes...
------------------------------------------------
Training: 1 Layers, 10 Nodes...
Training: 1 Layers, 10 Nodes...
Training: 1 Layers, 50 Nodes...
Training: 1 Layers, 50 Nodes...
Training: 1 Layers, 100 Nodes...
Training: 3 Layers, 10 Nodes...
Training: 1 Layers, 100 Nodes...
Training: 3 Layers, 10 Nodes...
Training: 3 Layers, 50 Nodes...
Training: 3 Layers, 10 Nodes...
Training: 3 Layers, 50 Nodes...
Training: 3 Layers, 50 Nodes...
Training: 3 Layers, 100 Nodes...
Training: 3 Layers, 100 Nodes...
Training: 5 Layers, 10 Nodes...
Training: 5 Layers, 10 Nodes...
Training: 5 Layers, 50 Nodes...
Training: 5 Layers, 100 Nodes...

Network Size Comparison (Accuracy):
Nodes        10        50        100
Layers
1       0.526608  0.550998  0.424612
3       0.576497  0.494457  0.573171
5       0.497783  0.533259  0.552106

------------------------------------------------
 Experiment 3: Compare Learning Rates (10^-2 to 10^-5)
------------------------------------------------
Training LR: 0.01...
Training LR: 0.001...
Training LR: 0.0001...
Training LR: 1e-05...

Learning Rate Comparison:
  LR Scientific  Learning Rate  Accuracy
0         10^-2        0.01000  0.431264
1         10^-3        0.00100  0.566519
2         10^-4        0.00010  0.525499
3         10^-5        0.00001  0.552106

------------------------------------------------
 Sample Predictions (Best Model)
------------------------------------------------
```

LEB 5: NN for Blood Cells

Link: https://www.kaggle.com/datasets/unclesamulus/blood-cells-image-dataset

Objective:

- Apply NN to Blood Cells classify.

Contents:

- Load the Blood Cells dataset from directory (Link).
- Split the data into training and testing sets.
- Train NN models
- Evaluate each model using accuracy

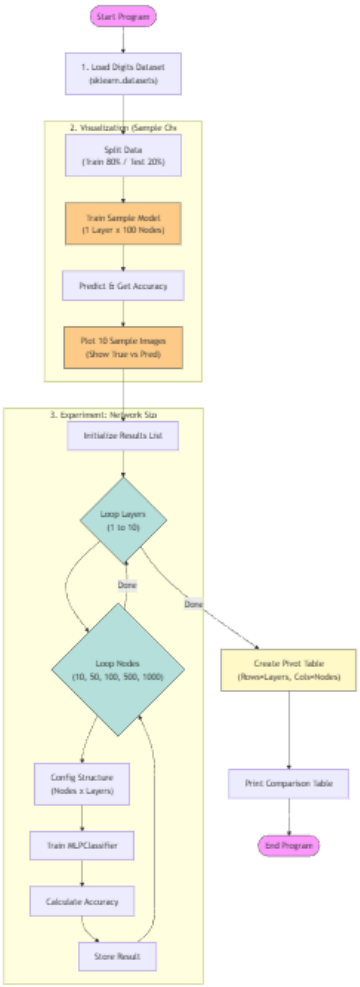
Output:

- Classification accuracy on the test set.
- Compare the results of different network sizes in a table (1–10 layers with 10–1000 nodes per layer)
- Compare the results of different learning rates in a table (10^{-2}, to 10^{-5})

Code or Algorithms:	Flowchart:

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from PIL import Image
import warnings
 	
warnings.filterwarnings('ignore')
 	
def load_data(root_folder, image_size=(64, 64),
max_samples_per_class=None):
  images = []
  labels = []
 	
  if not os.path.exists(root_folder):
  print(f"Error: Folder not found: {root_folder}")	

```
        return np.array(images), np.array(labels)

    print(f"Scanning folder: {root_folder}")

    classes = [d for d in os.listdir(root_folder) if
os.path.isdir(os.path.join(root_folder, d))]
    print(f"Found {len(classes)} Classes: {classes}")

    for class_name in classes:
        class_path = os.path.join(root_folder, class_name)
        count = 0
        for img_name in os.listdir(class_path):
            if max_samples_per_class and count >=
max_samples_per_class:
                break

            img_path = os.path.join(class_path, img_name)
            try:
                with Image.open(img_path) as img:
                    img = img.resize(image_size).convert('RGB')
                    img_array = np.array(img).flatten() / 255.0

                    images.append(img_array)
                    labels.append(class_name)
                    count += 1
            except Exception as e:
                pass
        print(f" -> Loaded {count} images from {class_name}")

    return np.array(images), np.array(labels)

def main():
    print("="*50)
    print(" LAB 5: NN on Blood Cells Classification")
    print("="*50)
```



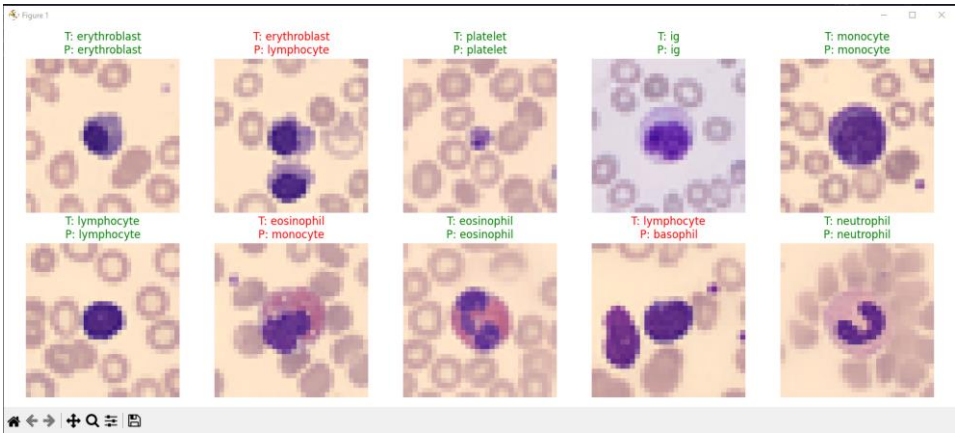
```
    dataset_path = "../dataset/bloodcells"
    if not os.path.exists(dataset_path):
        dataset_path = "dataset/bloodcells"

    X, y_labels = load_data(dataset_path, image_size=(32,
32), max_samples_per_class=200)

    if len(X) == 0:
        print("Error: No data loaded.")
        return

    le = LabelEncoder()
    y = le.fit_transform(y_labels)

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    print(f"\nFinal Split: {X_train.shape[0]} Train,
{X_test.shape[0]} Test samples")

    print("\n" + "="*50)
    print(" Experiment 1: Compare Network Sizes")
    print("=" * 50)

    layers_test = [1, 2, 5]
    nodes_test = [10, 100, 500]

    results_size = []

    for l in layers_test:
        for n in nodes_test:
            hidden_layers = tuple([n] * l)
            print(f"Training: {l} Layers, {n} Nodes...")
```

```
            mlp =
MLPClassifier(hidden_layer_sizes=hidden_layers,
max_iter=200, random_state=42)
            mlp.fit(X_train, y_train)
            acc = mlp.score(X_test, y_test)

            results_size.append({'Layers': l, 'Nodes': n,
'Accuracy': acc})

    df_size = pd.DataFrame(results_size)
    pivot_size = df_size.pivot(index='Layers',
columns='Nodes', values='Accuracy')
    print("\nNetwork Size Results:")
    print(pivot_size)

    print("\n" + "="*50)
    print(" Experiment 2: Compare Learning Rates")
    print("=" * 50)

    learning_rates = [0.01, 0.001, 0.0001, 0.00001]

    fixed_struct = (100, 100)

    results_lr = []
    best_model = None
    best_acc = 0

    for lr in learning_rates:
        print(f"Training LR: {lr}...")
        mlp = MLPClassifier(hidden_layer_sizes=fixed_struct,
learning_rate_init=lr, max_iter=300, random_state=42)
        mlp.fit(X_train, y_train)
        acc = mlp.score(X_test, y_test)

        results_lr.append({
            'Learning Rate': lr,
```

```
            'LR Sci': f"10^{int(np.log10(lr))}",
            'Accuracy': acc
        })

        if acc >= best_acc:
            best_acc = acc
            best_model = mlp

    df_lr = pd.DataFrame(results_lr)[['LR Sci', 'Learning Rate',
'Accuracy']]
    print("\nLearning Rate Results:")
    print(df_lr)

    print("\nDisplaying samples from Best Model...")

    indices = np.random.choice(len(X_test), 10,
replace=False)
    plt.figure(figsize=(15, 6))

    for i, idx in enumerate(indices):
        img_show = X_test[idx].reshape(32, 32, 3)

        true_label = le.inverse_transform([y_test[idx]])[0]
        pred_label =
le.inverse_transform([best_model.predict([X_test[idx]])])[0]

        color = 'green' if true_label == pred_label else 'red'

        plt.subplot(2, 5, i+1)
        plt.imshow(img_show)
        plt.title(f"T: {true_label}\nP: {pred_label}",
color=color)
        plt.axis('off')
    plt.tight_layout()
    plt.show()
if __name__ == "__main__":
```

main()

Result:




```
Network Size Results:
Network Size Results:
Network Size Results:
Nodes        10        100       500
Layers
1       0.109375  0.753125  0.737500
2       0.112500  0.740625  0.715625
2       0.112500  0.740625  0.715625
5       0.540625  0.718750  0.712500

================================================
 Experiment 2: Compare Learning Rates
================================================
Training LR: 0.01...
Training LR: 0.001...
Training LR: 0.01...
Training LR: 0.001...
Training LR: 0.001...
Training LR: 0.0001...
Training LR: 1e-05...

Learning Rate Results:
Learning Rate Results:
  LR Sci  Learning Rate  Accuracy
0  10^-2        0.01000  0.656250
1  10^-3        0.00100  0.740625
2  10^-4        0.00010  0.743750
3  10^-5        0.00001  0.640625

Displaying samples from Best Model...
```

ให้ข้อคิดเห็น **[ตบ11]: AI**

1. ฟังก์ชันโหลดและเตรียมรูปภาพ (load_data)

ส่วนนี้ทำหน้าที่เป็น "คนเตรียมวัตถุดิบ" ก่อนส่งเข้าครัว (Neural Network):

- **Scanning:** สแกนโฟลเดอร์เพื่อหาชื่อคลาสอัตโนมัติ (เช่น Eosinophil, Lymphocyte)
- **Image Processing:**
- **Resize:** ย่อรูปเหลือ 32x32 พิกเซล (กำหนดใน main) เพื่อให้คอมพิวเตอร์ประมวลผลไหว
- **Flatten:** แปลงรูปภาพ 2 มิติ ให้เป็น **เส้นตรง (Vector)** เพราะ MLP รับข้อมูลแบบเส้นตรงเท่านั้น
- **Normalization:** หารค่าสีด้วย 255.0 เพื่อปรับค่าให้อยู่ช่วง 0-1 (ช่วยให้ AI เรียนรู้ไวขึ้น)

2. การเตรียมข้อมูลหลัก (Main Data Setup)

- **Label Encoding:** แปลงชื่อเซลล์เม็ดเลือด (Text) ให้เป็นตัวเลข (0, 1, 2, ...) เพื่อให้คำนวณได้
- **Train/Test Split:** แบ่งข้อมูลเป็น 2 กอง (80% ไว้สอน, 20% ไว้สอบ)

3. การทดลองที่ 1: หาขนาดสมองที่เหมาะสม (Network Size Experiment)

- **Grid Search:** วนลูปทดสอบจับคู่โครงสร้างต่างๆ
-
- จำนวนชั้น (Layers): 1, 2, 5 ชั้น
- จำนวนเซลล์สมอง (Nodes): 10, 100, 500 ตัว
- **Pivot Table:** สรุปผลออกมาเป็นตารางเพื่อดูว่าโครงสร้างไหนให้ความแม่นยำ (Accuracy) สูงสุด

4. การทดลองที่ 2: หาอัตราการเรียนรู้ (Learning Rate Experiment)

- ใช้โครงสร้างคงที่ (2 ชั้น, 100 โหนด) แล้วเปลี่ยนความเร็วในการเรียนรู้ (0.01 ถึง 0.00001)
- **Best Model Tracking:** โค้ดจะคอยจำว่าโมเดลตัวไหนเก่งที่สุด (best_model) เพื่อเอาไว้ใช้โชว์ผลงานในตอนท้าย

LEB 6: NN with Smoothed COVID-19 Data.

LINK: https://www.kaggle.com/datasets/hosammhmdali/covid-19-dataset

Objective:

- Apply NN to Smoothed COVID-19 Data (.CSV) (THA).

Contents:

- Load the smoothed COVID-19 dataset from a CSV file (provided link).
- Preprocess the data and split it into training and testing sets.
- Construct and train NN models with 10 hidden layers (1 layers: 100 nodes).
- Evaluate the performance of each NN model using the accuracy metric..

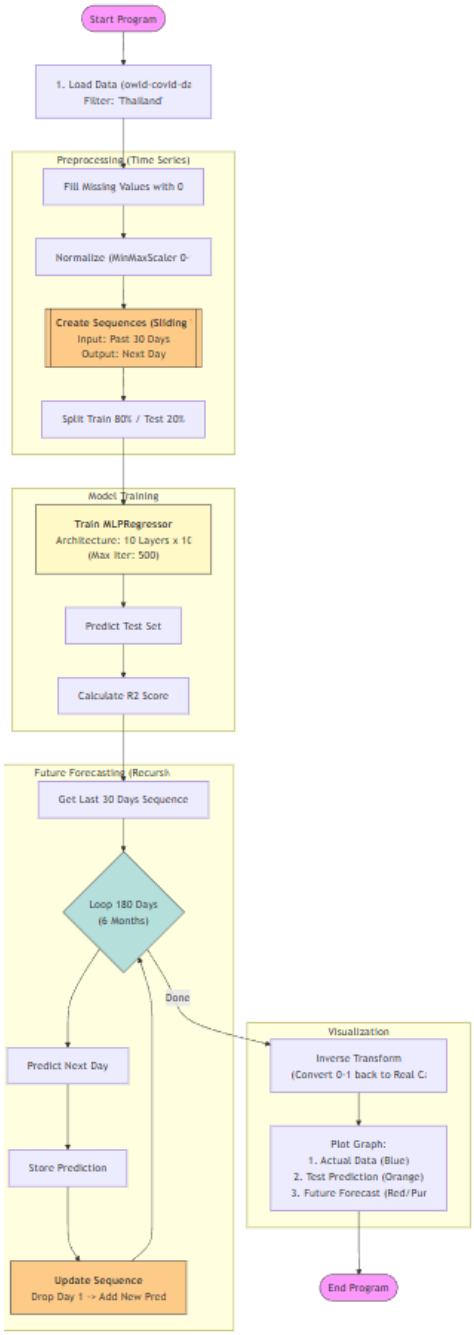
Output:

- Accuracy score of each NN for comparison across different layers and nodes.
- Time-series plots comparing: Actual, Smoothed Trend, and NN Forecast for 3-month and 6-month forecasting horizons, similar to the provided example figure.

Code or Algorithms:	**Flowchart:**

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score
import os
 	
def create_sequences(data, seq_length):
  xs = []
  ys = []
  for i in range(len(data)-seq_length-1):
  x = data[i:(i+seq_length)]
  y = data[i+seq_length]
  xs.append(x)
  ys.append(y)
  return np.array(xs), np.array(ys)
 	
def main():
  print("="*50)
  print(" LAB 6: NN on Smoothed COVID-19 Data (Forecasting)")
  print("="*50)
  # 1. Load Data
  dataset_path = "../dataset/covid/owid-covid-data.csv"
  if not os.path.exists(dataset_path):
  dataset_path = "dataset/covid/owid-covid-data.csv"
  print(f"Loading dataset from: {dataset_path}")
  try:
  df = pd.read_csv(dataset_path)
  except FileNotFoundError:
  print("Error: Dataset not found.")	

```
        return
    # 2. Filter Data (Thailand, Smoothed Cases)
    country = 'Thailand'
    target_col = 'new_cases_smoothed'
    print(f"Filtering data for: {country}")
    df_thai = df[df['location'] == country].copy()
    df_thai['date'] = pd.to_datetime(df_thai['date'])
    df_thai = df_thai.sort_values('date')
    # Handle Missing Values
    df_thai[target_col] = df_thai[target_col].fillna(0)
    data = df_thai[target_col].values.reshape(-1, 1)
    dates = df_thai['date'].values
    print(f"Total Records: {len(data)}")
    # 3. Preprocessing
    scaler = MinMaxScaler()
    data_normalized = scaler.fit_transform(data)
    # Create Sequences
    seq_length = 30 # Look back 30 days to predict next day
    X, y = create_sequences(data_normalized, seq_length)

    # Split Train/Test (80/20)
    train_size = int(len(X) * 0.8)
    X_train, X_test = X[:train_size], X[train_size:]
    y_train, y_test = y[:train_size], y[train_size:]
    # Flatten input for MLP (samples, features) -> (samples, seq_length)
    X_train = X_train.reshape(X_train.shape[0], -1)
    X_test = X_test.reshape(X_test.shape[0], -1)
    print(f"Training Samples: {X_train.shape[0]}")
    print(f"Testing Samples: {X_test.shape[0]}")
    # 4. Train NN Model
    # 10 Hidden Layers, 100 Nodes each
    hidden_layers = tuple([100] * 10)
```



```
    print(f"Training MLPRegressor with Architecture: {hidden_layers}")
    mlp = MLPRegressor(hidden_layer_sizes=hidden_layers,
                       activation='relu',
                       solver='adam',
                       max_iter=500,
                       random_state=42)

    mlp.fit(X_train, y_train.ravel())
    # 5. Evaluate
    y_pred_test = mlp.predict(X_test)
    y_pred_test_inv = scaler.inverse_transform(y_pred_test.reshape(-1, 1))
    y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))
    score = r2_score(y_test_inv, y_pred_test_inv)
    print(f"Model R2 Score (Accuracy): {score:.4f}")

    # 6. Forecast Future
    # We need the last sequence from the entire data to start forecasting future
    last_sequence = data_normalized[-seq_length:].reshape(1, -1)
    future_days_3m = 90
    future_days_6m = 180
    forecast_3m = []
    curr_seq = last_sequence.copy()
    print(f"Forecasting next {future_days_6m} days...")
    for i in range(future_days_6m):
        next_pred = mlp.predict(curr_seq)[0]
        if i < future_days_3m:
            forecast_3m.append(next_pred)
        # Update sequence: remove first element, add new prediction
```

```
        curr_seq = np.append(curr_seq[:, 1:],
[[next_pred]], axis=1)
    forecast_6m = forecast_3m + [curr_seq[0, -1]] #
Continuation (simplified logic for list storage)
    # Actually, let's re-run loop or just store all 180
    # Re-doing clean forecast list
    forecast_full = []
    curr_seq = last_sequence.copy()
    for i in range(future_days_6m):
        next_pred = mlp.predict(curr_seq)[0]
        forecast_full.append(next_pred)
        curr_seq = np.append(curr_seq[:, 1:],
[[next_pred]], axis=1)
    forecast_full_inv =
scaler.inverse_transform(np.array(forecast_full).reshape(-
1, 1))
    # Generate Dates for Forecast
    last_date = dates[-1]
    future_dates = pd.date_range(start=last_date +
pd.Timedelta(days=1), periods=future_days_6m)
    # 7. Plot Results
    plt.figure(figsize=(12, 6))

    # Plot Actual Data (Historical)
    plt.plot(dates, data, label='Actual / Smoothed
Trend', color='blue', alpha=0.6)
    # Plot Test Predictions (to see model fit)
    # Align dates for test predictions
    test_dates = dates[train_size+seq_length+1 :
train_size+seq_length+1+len(y_test)]
    # Note: alignment can be tricky with sequences.
    # Index of y_test[0] corresponds to original data
index: train_size + seq_length
    test_date_indices = range(train_size + seq_length,
train_size + seq_length + len(y_test))
    test_dates_plot = dates[test_date_indices]
```

```
    plt.plot(test_dates_plot, y_pred_test_inv, label='Model Test Prediction', color='orange', linestyle='--')
    # Plot 3-Month Forecast
    plt.plot(future_dates[:90], forecast_full_inv[:90], label='3-Month Forecast', color='red', linewidth=2)
    # Plot 6-Month Forecast (Extension)
    plt.plot(future_dates[90:], forecast_full_inv[90:], label='6-Month Forecast', color='purple', linestyle=':', linewidth=2)
    plt.title(f'COVID-19 Forecasting (Thailand) - R2 Score: {score:.4f}')
    plt.xlabel('Date')
    plt.ylabel('New Cases Smoothed')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```

ให้ข้อคิดเห็น **[ตบ12]:** AI

1. การเตรียมฟังก์ชันช่วย (create_sequences)

ส่วนนี้สำคัญที่สุดสำหรับโจทย์ Time Series (อนุกรมเวลา)

- **หน้าที่:** แปลงข้อมูลตัวเลขเรียงกันยาวๆ ให้เป็นรูปแบบ **Input (อดีต) -> Output (อนาคต)**
- **วิธีการ (Sliding Window):** ใช้หน้าต่างขนาด 30 วัน (seq_length) เลื่อนไปทีละวัน เพื่อสร้างชุดข้อมูล เช่น:
  - Input: ข้อมูลวันที่ 1-30 -> Target: วันที่ 31
  - Input: ข้อมูลวันที่ 2-31 -> Target: วันที่ 32

2. การโหลดและกรองข้อมูล (Load & Filter Data)

- โหลดไฟล์ CSV owid-covid-data.csv
- **Filter:** เลือกเฉพาะประเทศไทย (Thailand) และเลือกคอลัมน์ new_cases_smoothed (ผู้ติดเชื้อรายใหม่แบบเฉลี่ย) ซึ่งจะช่วยลดความผันผวนรายวัน ทำให้โมเดลเรียนรู้แนวโน้มได้ดีกว่า
- **Cleaning:** เติมค่าว่าง (NaN) ด้วย 0 เพื่อป้องกัน Error

3. การเตรียมข้อมูลก่อนเข้าโมเดล (Preprocessing)

- **Normalization:** ใช้ MinMaxScaler ปรับข้อมูลทั้งหมดให้อยู่ในช่วง 0 ถึง 1 เพื่อให้ Neural Network คำนวณได้เร็วและแม่นยำขึ้น
- **Sequence Creation:** เรียกฟังก์ชันจากข้อ 1 มาแบ่งข้อมูลเป็นชุดๆ (ดูย้อนหลัง 30 วัน)
- **Split Data:** แบ่งข้อมูลเป็น Train (80%) และ Test (20%) เพื่อใช้สอนและวัดผลโมเดล

4. การสร้างและเทรนโมเดล (Model Training)

- **Deep Neural Network:** สร้างโมเดล MLPRegressor ที่มีความลึกมาก (Deep Learning)
- **Structure:** ใช้ Hidden Layers ถึง **10 ชั้น** ชั้นละ **100 โหนด** (tuple([100] * 10)) เพื่อให้โมเดลมีความซับซ้อนพอที่จะจับ Pattern ของการระบาดได้
- **Training:** สั่ง fit เพื่อให้โมเดลเรียนรู้จากข้อมูล Train

5. การวัดผล (Evaluation)

- นำโมเดลไปทำนายข้อมูลชุด Test
- **Inverse Transform:** แปลงค่าผลลัพธ์จาก 0-1 กลับมาเป็นจำนวนคนติดเชื้อจริง
- **R2 Score:** คำนวณค่าความแม่นยำ (ยิ่งเข้าใกล้ 1.0 ยิ่งดี) เพื่อดูว่าโมเดลทำนายแนวโน้มได้ถูกต้องแค่ไหน

6. การพยากรณ์อนาคต (Future Forecasting)

ส่วนนี้ใช้เทคนิค **Recursive Forecasting (ทำนายแบบวนซ้ำ)**:

- เริ่มต้นด้วยข้อมูล 30 วันสุดท้ายที่มีอยู่จริง
- **Loop 180 รอบ (6 เดือน):**
  1. ให้โมเดลทำนายวันถัดไป
  2. เก็บผลลัพธ์ไว้
  3. **Update Input:** ตัดข้อมูลวันแรกทิ้ง แล้วเอาผลที่เพิ่งทำนายได้ไส่ต่อท้าย เพื่อใช้ทำนายวันถัดไป

Result:

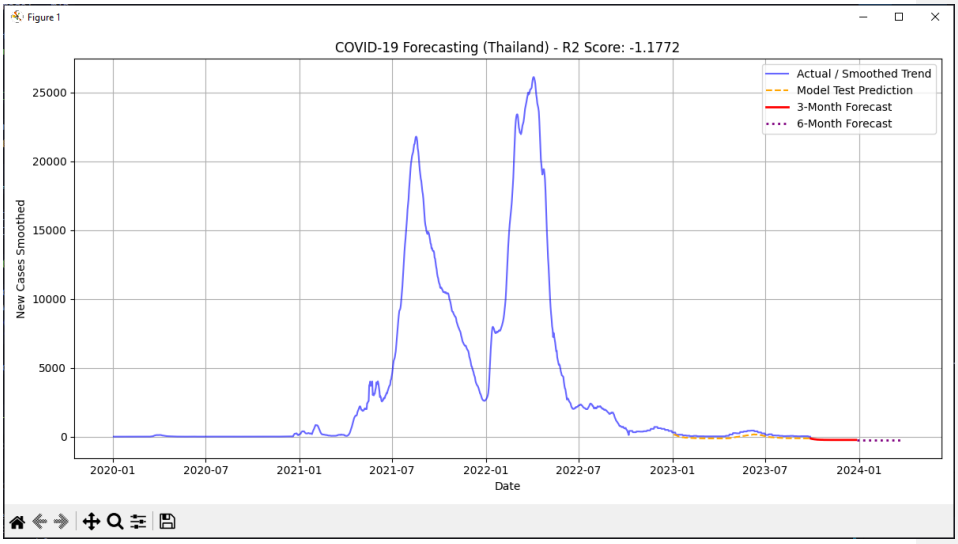


Figure 1
COVID-19 Forecasting (Thailand) - R2 Score: -1.1772
Actual / Smoothed Trend
Model Test Prediction
3-Month Forecast
6-Month Forecast
New Cases Smoothed
25000
20000
15000
10000
5000
0
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
Date