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

ใบงานที่ 2 Convolutional Neural Network และการประยุกต์ใช้งาน CNN

วัตถุประสงค์

1. เพื่อให้นักศึกษาเข้าใจหลักการทำงานของ Convolutional Neural Network (CNN) และแนวทางการประยุกต์ใช้ในงานจริงทั้งด้านการจำแนกและการพยากรณ์
2. เพื่อให้นักศึกษาสามารถพัฒนาผังการทำงาน ประมวลผล และแสดงผลลัพธ์ เพื่อพัฒนาระบบต้นแบบร่วมกับ CNN ในการจำแนกและการพยากรณ์ได้อย่างถูกต้อง

LAB1: CNN Basic for classify digits

Objective:

- Apply CNN to classify digits.

Contents:

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

Output:

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

Code:

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import os
physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) > 0:
```

```
    tf.config.experimental.set_memory_growth(physical_devices[0], True)
def load_data():
    print("Loading MNIST Dataset...")
    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
    x_train = x_train.reshape((-1, 28, 28, 1)).astype('float32') / 255.0
    x_test = x_test.reshape((-1, 28, 28, 1)).astype('float32') / 255.0
    y_train = tf.keras.utils.to_categorical(y_train, 10)
    y_test = tf.keras.utils.to_categorical(y_test, 10)
    x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.1,
random_state=42)
    return (x_train, y_train), (x_val, y_val), (x_test, y_test)
def build_cnn_model(num_layers, num_filters, input_shape=(28, 28, 1)):
    model = models.Sequential()
    for i in range(num_layers):
        if i == 0:
            model.add(layers.Conv2D(num_filters, (3, 3), activation='relu', padding='same',
input_shape=input_shape))
        else:
            model.add(layers.Conv2D(num_filters, (3, 3), activation='relu', padding='same'))
        if i % 2 == 0:
             model.add(layers.MaxPooling2D((2, 2), padding='same'))
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(10, activation='softmax')) # Output 10 digits
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

def main():
    (x_train, y_train), (x_val, y_val), (x_test, y_test) = load_data()
    layers_to_test = [1, 2, 3]
    filters_to_test = [32, 64]
    results = []
    print(f"\nStarting Experiment...")
    print(f"Layers to test: {layers_to_test}")
    print(f"Filters (Nodes) to test: {filters_to_test}")
    best_model = None
    best_acc = 0
    for l in layers_to_test:
        for f in filters_to_test:
            print(f"\n--- Training CNN: {l} Layers, {f} Filters ---")
            try:
                model = build_cnn_model(num_layers=l, num_filters=f)
                history = model.fit(x_train, y_train, epochs=3, batch_size=64,
validation_data=(x_val, y_val), verbose=1)
                test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
                print(f"Result: Accuracy = {test_acc:.4f}")
                results.append({
                    'Conv Layers': l,
                    'Filters/Nodes': f,
                    'Test Accuracy': test_acc
```

```
                })
                if test_acc > best_acc:
                    best_acc = test_acc
                    best_model = model
            except Exception as e:
                print(f"Error training {l} layers, {f} filters: {e}")
    print("\n" + "="*50)
    print(" Experiment Results: CNN Architecture Comparison")
    print("="*50)
    df_results = pd.DataFrame(results)

    if not df_results.empty:
        pivot_table = df_results.pivot(index='Conv Layers', columns='Filters/Nodes',
values='Test Accuracy')
        print(pivot_table)
    else:
        print("No results to display.")
    print("="*50)
    if best_model:
        print("\nDisplaying Sample Predictions from Best Model...")
        predictions = best_model.predict(x_test[:10])
        predicted_labels = np.argmax(predictions, axis=1)
        true_labels = np.argmax(y_test[:10], axis=1)
        plt.figure(figsize=(15, 3))
        for i in range(10):
            plt.subplot(1, 10, i+1)
            plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
            color = 'green' if predicted_labels[i] == true_labels[i] else 'red'
            plt.title(f"P:{predicted_labels[i]}\nT:{true_labels[i]}", color=color)
            plt.axis('off')
        plt.show()
if __name__ == "__main__":
    main()
```

ให้ข้อคิดเห็น **[ชบ1]:** ใช้ AI

ส่วนที่ 1: การตั้งค่าและเรียกใช้ไลบรารี (Setup & GPU Config) เริ่มต้นด้วยการนำเข้าไลบรารีที่จำเป็นสำหรับการทำ Deep Learning ได้แก่ TensorFlow และ Keras รวมถึงเครื่องมือจัดการข้อมูลอย่าง NumPy และ Pandas พร้อมทั้งตัววาดกราฟ Matplotlib ในส่วนนี้ยังมีการเขียนคำสั่งตรวจสอบฮาร์ดแวร์เพื่อเช็คว่าเครื่องมี GPU หรือไม่ หากพบ GPU ระบบจะตั้งค่า set_memory_growth เป็น True เพื่อบริหารจัดการหน่วยความจำการ์ดจอให้ขยายตัวตามการใช้งานจริง ป้องกันปัญหาโปรแกรมจอง Ram การ์ดจอจนเต็มเครื่อง

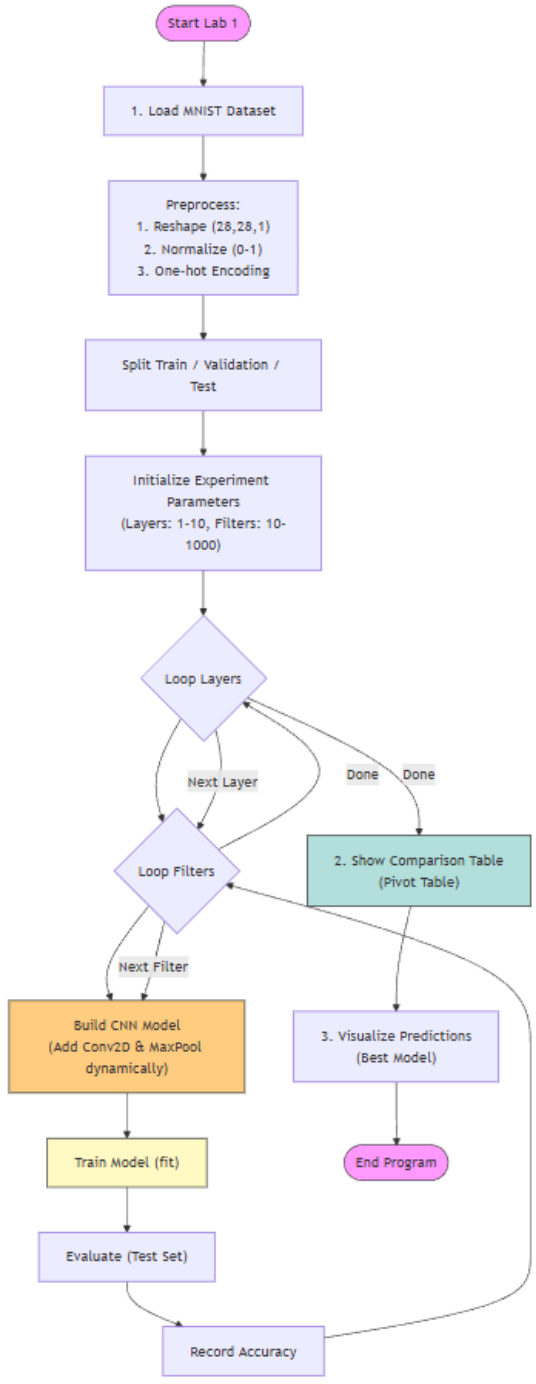
ส่วนที่ 2: การเตรียมข้อมูล (Data Preprocessing) ฟังก์ชัน load_data ทำหน้าที่โหลดชุดข้อมูลตัวเลขเขียนมือ (MNIST) เข้าสู่ระบบ จากนั้นจะเข้าสู่กระบวนการปรับปรุงข้อมูล (Preprocessing) เพื่อให้เหมาะกับโครงสร้าง CNN เริ่มจากการเปลี่ยนมิติของภาพ (Reshape) ให้เป็น 3 มิติ (28x28x1), การปรับค่าสี (Normalize) จากช่วง 0-255 ให้เหลือช่วง 0-1 เพื่อช่วยให้โมเดลคำนวณได้เร็วขึ้น, และการแปลงข้อมูลเฉลย (Labels) ให้เป็นรูปแบบ One-hot Encoding รวมถึงมีการแบ่งข้อมูลส่วนหนึ่งออกมาเป็น Validation Set เพื่อใช้ตรวจสอบความแม่นยำระหว่างการฝึกสอน

ส่วนที่ 3: การสร้างโครงสร้างโมเดล (Model Architecture) ฟังก์ชัน build_cnn_model ถูกออกแบบมาให้สร้างโมเดลแบบ Dynamic คือสามารถปรับเปลี่ยนความลึกและความซับซ้อนได้ตามพารามิเตอร์ที่ส่งเข้ามา โดยใช้ลูป for เพื่อสร้างชั้น Convolutional Layers ตามจำนวนที่ต้องการ และมีการใส่เงื่อนไขให้เพิ่มชั้น Max Pooling ทุกๆ 2 เลเยอร์เพื่อช่วยลดขนาดภาพและคัดกรองฟีเจอร์ที่สำคัญ ปิดท้ายด้วยการ Flatten ข้อมูลและส่งเข้าสู่ชั้น Dense เพื่อประมวลผลและจำแนกตัวเลขทั้ง 10 ตัวด้วยฟังก์ชัน Softmax

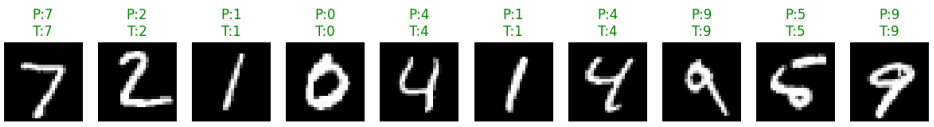
ส่วนที่ 4: การทดลองและฝึกสอน (Training Experiment) ในฟังก์ชัน main จะเป็นการจำลองการทดลอง (Experiment Loop) โดยกำหนดชุดตัวแปรที่จะทดสอบ ได้แก่ จำนวนชั้นของเลเยอร์ (Layers) และจำนวนฟิลเตอร์ (Filters) โปรแกรมจะทำงานแบบ Grid Search คือวนลูปสร้างโมเดลตามค่าคอนฟิกเหล่านี้ทีละคู่ แล้วทำการเทรนโมเดล (Training) เป็นเวลา 3 Epochs จากนั้นจึงวัดผล (Evaluation) กับชุดข้อมูลทดสอบเพื่อหาค่าความแม่นยำ (Accuracy) และเก็บข้อมูลโมเดลที่ให้ผลลัพธ์ดีที่สุด (Best Model) เอาไว้ใช้งานต่อ

ส่วนที่ 5: การสรุปผลและแสดงภาพ (Reporting & Visualization) ส่วนสุดท้ายเป็นการนำผลลัพธ์จากการทดลองทั้งหมดมาสรุปให้อยู่ในรูปแบบตาราง Pivot Table โดยใช้ Pandas เพื่อเปรียบเทียบว่าการปรับ Layer และ Filter ส่งผลต่อความแม่นยำอย่างไร และนำโมเดลที่ดีที่สุดมาทดสอบพยากรณ์ภาพจริงจำนวน 10 ภาพ โดยใช้ Matplotlib วาดภาพผลลัพธ์พร้อมระบุสี ...

Flowchart:


Start Lab 1
1. Load MNIST Dataset
Preprocess:
1. Reshape (28,28,1)
2. Normalize (0-1)
3. One-hot Encoding
Split Train / Validation / Test
Initialize Experiment Parameters (Layers: 1-10, Filters: 10-1000)
Loop Layers
Next Layer
Done
Done
Loop Filters
2. Show Comparison Table (Pivot Table)
Next Filter
Build CNN Model (Add Conv2D & MaxPool dynamically)
3. Visualize Predictions (Best Model)
Train Model (fit)
End Program
Evaluate (Test Set)
Record Accuracy

Result:



LEB 2: CNN on the Face recognition.

Objective:

- Apply CNN to Face recognition.

Contents:

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

Output:

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

Code:

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.metrics import accuracy_score
import os
physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)
def load_data():
    print("Loading LFW Faces Dataset...")
    try:
```

```
        lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
        n_samples, h, w = lfw_people.images.shape
        X = lfw_people.images
        y = lfw_people.target
        target_names = lfw_people.target_names
        n_classes = target_names.shape[0]
        print(f"Dataset Stats: {n_samples} samples, Image Size: {h}x{w}, Classes:
{n_classes}")
        print(f"Classes: {target_names}")
        X = X.reshape(n_samples, h, w, 1)
        if np.max(X) > 1.0:
            X = X / 255.0
        x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)

        return (x_train, y_train), (x_test, y_test), (h, w, 1), target_names,
n_classes
    except Exception as e:
        print(f"Error loading LFW dataset: {e}")
        return None, None, None, None, None
def build_cnn_model(num_layers, num_filters, input_shape, num_classes):
    model = models.Sequential()
    for i in range(num_layers):
        if i == 0:
            model.add(layers.Conv2D(num_filters, (3, 3), activation='relu',
padding='same', input_shape=input_shape))
        else:
            model.add(layers.Conv2D(num_filters, (3, 3), activation='relu',
padding='same'))         if i % 2 == 0:
             model.add(layers.MaxPooling2D((2, 2), padding='same'))
             model.add(layers.Dropout(0.25))
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(num_classes, activation='softmax'))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
def main():
    (x_train, y_train), (x_test, y_test), input_shape, target_names, n_classes =
load_data()
    if x_train is None:
        return

    layers_to_test = [1, 2, 3]
    filters_to_test = [32, 64]
    results = []
    print(f"\nStarting Experiment on Face Recognition...")
    best_model = None
    best_acc = 0
    for l in layers_to_test:
```

```
        for f in filters_to_test:
            print(f"\n--- Training CNN: {l} Layers, {f} Filters ---")
            try:
                model = build_cnn_model(l, f, input_shape, n_classes)
                history = model.fit(x_train, y_train, epochs=10, batch_size=32,
validation_data=(x_test, y_test), verbose=1)
                test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
                print(f"Result: Accuracy = {test_acc:.4f}")
                results.append({
                    'Conv Layers': l,
                    'Filters/Nodes': f,
                    'Test Accuracy': test_acc
                })
                if test_acc > best_acc:
                    best_acc = test_acc
                    best_model = model
            except Exception as e:
                print(f"Error training {l} layers, {f} filters: {e}")
    print("\n" + "="*50)
    print(" Experiment Results: CNN Face Recognition")
    print("="*50)
    df_results = pd.DataFrame(results)
    if not df_results.empty:
        pivot_table = df_results.pivot(index='Conv Layers', columns='Filters/Nodes',
values='Test Accuracy')

        print(pivot_table)
    print("="*50)

    if best_model:
        print("\nDisplaying Sample Predictions from Best Model...")
        predictions = best_model.predict(x_test[:5])
        predicted_ids = np.argmax(predictions, axis=1)
        true_ids = np.argmax(y_test[:5], axis=1)
        plt.figure(figsize=(15, 4))
        for i in range(5):
            plt.subplot(1, 5, i+1)
            plt.imshow(x_test[i].reshape(input_shape[0], input_shape[1]),
cmap='gray')

            pred_name = target_names[predicted_ids[i]].split()[-1] # Last name
            true_name = target_names[true_ids[i]].split()[-1]
            color = 'green' if predicted_ids[i] == true_ids[i] else 'red'
            plt.title(f"P:{pred_name}\nT:{true_name}", color=color)
            plt.axis('off')
        plt.show()

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

ให้ข้อคิดเห็น **[ตบ2]: ใช้ AI**

ส่วนที่ 1: การตั้งค่าและเรียกใช้ไลบรารี (Setup & GPU Config) ในส่วนแรกจะเป็นการนำเข้าไลบรารีที่จำเป็น ได้แก่ TensorFlow/Keras สำหรับสร้างโมเดล Deep Learning, Scikit-learn สำหรับโหลดชุดข้อมูล LFW และจัดการข้อมูล, Pandas สำหรับทำตารางสรุปผล, และ Matplotlib สำหรับแสดงภาพใบหน้า นอกจากนี้ยังมีการตรวจสอบและตั้งค่า GPU (set_memory_growth) เพื่อให้บริหารจัดการทรัพยากรเครื่องได้อย่างมีประสิทธิภาพขณะรันโมเดล

ส่วนที่ 2: การโหลดและเตรียมข้อมูลใบหน้า (Data Loading & Preprocessing) ฟังก์ชัน load_data ทำหน้าที่ดึงข้อมูลใบหน้าบุคคลที่มีชื่อเสียง (Labeled Faces in the Wild - LFW) ผ่านคำสั่ง fetch_lfw_people โดยกำหนดให้เลือกเฉพาะบุคคลที่มีรูปอย่างน้อย 70 รูปเพื่อให้โมเดลเรียนรู้ได้เพียงพอ และย่อขนาดภาพลง (resize=0.4) เพื่อให้เทรนได้เร็วขึ้น จากนั้นทำการเตรียมข้อมูล 4 ขั้นตอน:

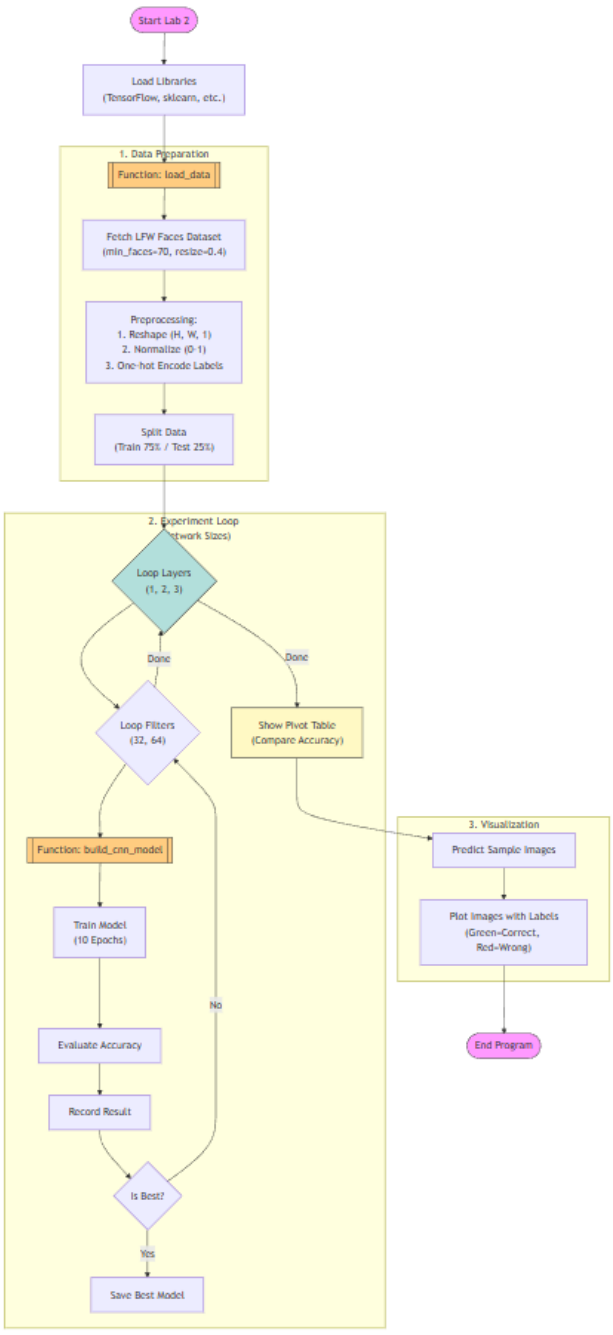
1. **Reshape:** เปลี่ยนมิติภาพให้เป็น 3 มิติ (ความสูง, ความกว้าง, 1) เพื่อระบุว่าเป็นภาพขาวดำ
2. **Normalize:** ปรับค่าสีให้อยู่ในช่วง 0-1
3. **One-hot Encoding:** แปลงชื่อคน (Target Labels) ให้เป็นรหัสเวกเตอร์
4. **Data Splitting:** แบ่งข้อมูลเป็นชุด Train และ Test ในอัตราส่วน 75:25

ส่วนที่ 3: การสร้างโครงสร้างโมเดล CNN (Model Architecture) ฟังก์ชัน build_cnn_model ถูกออกแบบมาให้สร้างโมเดลแบบ Dynamic ตามพารามิเตอร์จำนวนชั้น (Layers) และจำนวนตัวกรอง (Filters) ที่รับเข้ามา โดยมีจุดเด่นคือการใส่ **Dropout Layer** (อัตรา 0.25 และ 0.5) เข้าไปในโครงสร้างโมเดลเพื่อช่วยลดปัญหา Overfitting ซึ่งมักเกิดขึ้นได้ง่ายกับการจดจำใบหน้าที่มีจำนวนข้อมูลไม่มากนัก ส่วนโครงสร้างหลักยังคงประกอบด้วย Conv2D สำหรับสกัดฟีเจอร์, MaxPooling2D สำหรับย่อขนาด, และ Dense Layer สำหรับการจำแนกบุคคล

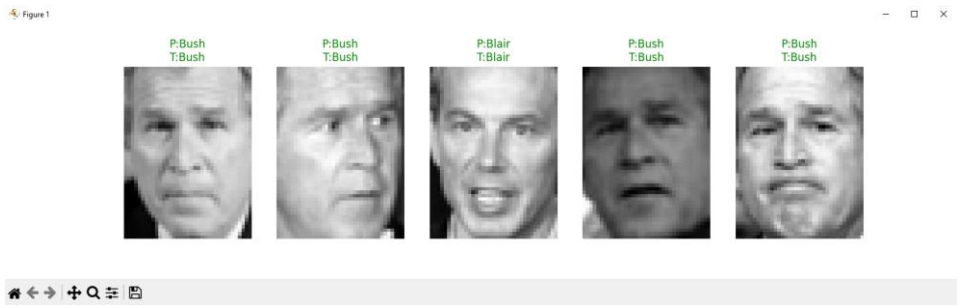
ส่วนที่ 4: การทดลองและฝึกสอน (Training Experiment) ในฟังก์ชัน main จะเป็นการจำลองการทดลองเพื่อหาโครงสร้างที่เหมาะสมที่สุด โดยกำหนดตัวแปรทดสอบจำนวนชั้นเลเยอร์ (1, 2, 3 ชั้น) และจำนวนฟิลเตอร์ (32, 64) โปรแกรมจะวนลูปสร้างโมเดลและทำการเทรน (Training) จำนวน 10 Epochs เนื่องจากงานจดจำใบหน้ามีความซับซ้อนกว่าตัวเลข จึงต้องใช้รอบการเทรนที่มากกว่า Lab ก่อนหน้า จากนั้นจะวัดผลความแม่นยำ (Accuracy) กับชุดข้อมูลทดสอบและเก็บค่าโมเดลที่ดีที่สุดไว้

ส่วนที่ 5: การสรุปผลและแสดงภาพ (Reporting & Visualization) ส่วนสุดท้ายจะแสดงผลลัพธ์การเปรียบเทียบความแม่นยำในรูปแบบตาราง Pivot Table ผ่าน Pandas เพื่อให้เห็นแนวโน้มว่าการเพิ่มความลึกหรือความกว้างของโมเดลส่งผล ...

Flowchart:


Start Lab 2
Load Libraries
(TensorFlow, sklearn, etc.)
1. Data Preparation
Function: load_data
Fetch LFW Faces Dataset
(min_faces=70, resize=0.4)
Preprocessing:
1. Reshape (H, W, 1)
2. Normalize (0-1)
3. One-hot Encode Labels
Split Data
(Train 75% / Test 25%)
2. Experiment Loop
Loop Layers
(1, 2, 3)
Done
Done
Loop Filters
(32, 64)
Show Pivot Table
(Compare Accuracy)
Function: build_cnn_model
Train Model
(10 Epochs)
Evaluate Accuracy
Record Result
Is Best?
No
Yes
Save Best Model
3. Visualization
Predict Sample Images
Plot Images with Labels
(Green=Correct,
Red=Wrong)
End Program

Result:



LEB 3: CNN on Iris.csv. Classification.

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

Objective:

- Apply CNN to Iris.csv.

Contents:

- Load the iris.csv data from directory dataset.
- Split the data into training and testing sets.
- Train CNN models.
- Evaluate each model using accuracy.

Output:

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

Code:

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```python
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score
import os

# ตั้งค่า GPU
physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)

def load_data(file_path):
    print(f"Loading Dataset from {file_path}...")
    try:
        df = pd.read_csv(file_path)

        # แยก Features และ Labels
        # สมมติว่า Column สุดท้ายคือ Label (Species)
        X = df.iloc[:, 1:-1].values # ตัด Id (ถ้ามี) และ Label
        y = df.iloc[:, -1].values

        # ถ้ารูปแบบไฟล์ Iris มาตรฐานมักจะมี Id เป็น column แรก หรือไม่มีเลย
        # ลองเช็คว่า column แรกเป็นตัวเลขเรียงกันไหม ถ้าใช่ให้ตัดออก
        if 'Id' in df.columns or 'id' in df.columns:
            X = df.iloc[:, 1:-1].values
        else:
            X = df.iloc[:, :-1].values

        # Encode Label
        le = LabelEncoder()
        y = le.fit_transform(y)
        num_classes = len(np.unique(y))

        # Scale Data
        scaler = StandardScaler()
        X = scaler.fit_transform(X)

        # Reshape for Conv1D: (Samples, Features, 1)
        # เรามี 4 Features (SepalLength, SepalWidth, PetalLength, PetalWidth)
        X = X.reshape(X.shape[0], X.shape[1], 1)

        # One-hot encoding
        y = tf.keras.utils.to_categorical(y, num_classes)

        # Split
        x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

        print(f"Data Shape: {X.shape}, Classes: {num_classes}")
        return (x_train, y_train), (x_test, y_test), num_classes, le

    except Exception as e:
        print(f"Error loading Iris dataset: {e}")
        return None, None, None, None
```

```
def build_cnn1d_model(num_layers, num_filters, input_shape, num_classes,
learning_rate=0.001):
    model = models.Sequential()

    # 1. Add Conv1D Layers
    for i in range(num_layers):
        if i == 0:
            model.add(layers.Conv1D(num_filters, kernel_size=2, activation='relu',
padding='same', input_shape=input_shape))
        else:
            model.add(layers.Conv1D(num_filters, kernel_size=2, activation='relu',
padding='same'))

        # MaxPool 1D (Optional, Iris features are few so maybe skip or use size
1/2)
        # เนื่องจาก Feature น้อย (4) ถ้า pool มากไปจะเหลือ 0 ทำให้ error
        # ใส่ Pool แค่ layer แรกๆ หรือไม่ใส่เลยสำหรับข้อมูลน้อยๆ
        if i == 0 and input_shape[0] > 2:
            model.add(layers.MaxPooling1D(pool_size=2, padding='same'))

    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))

    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)

    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

def main():
    # Path to Iris.csv
    file_path = os.path.join(os.path.dirname(__file__), '..', 'dataset',
'Iris.csv')

    if not os.path.exists(file_path):
        print(f"File not found: {file_path}")
        # ลองหาแบบ relative path ปกติ
        file_path = 'dataset/Iris.csv'
        if not os.path.exists(file_path):
            print(f"File strictly not found at {file_path}. Please check path.")
            return

    # 1. Load Data
    (x_train, y_train), (x_test, y_test), num_classes, le = load_data(file_path)

    if x_train is None: return

    input_shape = (x_train.shape[1], 1)
```

```
    # --- Experiment 1: Learning Rates ---
    print("\n" + "="*50)
    print(" Experiment 1: Different Learning Rates")
    print(" (Fixed Architecture: 2 Layers, 32 Filters)")
    print("="*50)
    lrs_to_test = [0.01, 0.001, 0.0001, 0.00001] # 10^-2 to 10^-5
    results_lr = []
    for lr in lrs_to_test:
        print(f"Testing Learning Rate: {lr}")
        model = build_cnn1d_model(num_layers=2, num_filters=32,
input_shape=input_shape, num_classes=num_classes, learning_rate=lr)
        model.fit(x_train, y_train, epochs=50, batch_size=16, verbose=0) # Silent
training
        loss, acc = model.evaluate(x_test, y_test, verbose=0)
        print(f" -> Accuracy: {acc:.4f}")
        results_lr.append({'Learning Rate': lr, 'Accuracy': acc})

    df_lr = pd.DataFrame(results_lr)
    print("\nResults Table (Learning Rate):")
    print(df_lr)

    # --- Experiment 2: Network Sizes ---
    print("\n" + "="*50)
    print(" Experiment 2: Network Sizes (Layers & Nodes)")
    print(" (Fixed Learning Rate: 0.001)")
    print("="*50)

    layers_to_test = [1, 2, 3] # สามารถเพิ่มถึง 10 ได้
    filters_to_test = [16, 32, 64, 128] # สามารถเพิ่มถึง 1000 ได้
    results_size = []

    for l in layers_to_test:
        for f in filters_to_test:
            print(f"Testing: {l} Layers, {f} Filters...")
            try:
                model = build_cnn1d_model(num_layers=l, num_filters=f,
input_shape=input_shape, num_classes=num_classes)
                model.fit(x_train, y_train, epochs=30, batch_size=16, verbose=0)
                loss, acc = model.evaluate(x_test, y_test, verbose=0)
                results_size.append({'Layers': l, 'Filters': f, 'Accuracy': acc})
            except Exception as e:
                print(f"Error: {e}")

    df_size = pd.DataFrame(results_size)
    pivot_size = df_size.pivot(index='Layers', columns='Filters',
values='Accuracy')
    print("\nResults Table (Network Sizes):")
    print(pivot_size)
    print("="*50)

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

ให้ข้อคิดเห็น **[ตบ3]:** ใช้ AI

ส่วนที่ 1: การตั้งค่าและเรียกใช้ไลบรารี (Setup & GPU Config) เริ่มต้นด้วยการนำเข้าไลบรารีสำหรับ Deep Learning และ Data Science ได้แก่ TensorFlow, Keras, NumPy, Pandas, Matplotlib และ Scikit-learn ในส่วนนี้มีการเขียนคำสั่งตรวจสอบฮาร์ดแวร์เพื่อเช็คว่าเครื่องมี GPU หรือไม่ หากพบระบบจะตั้งค่า set_memory_growth เป็น True เพื่อบริหารจัดการหน่วยความจำการ์ดจอให้ขยายตัวตามการใช้งานจริง ป้องกันไม่ให้โปรแกรมจองทรัพยากรจนหมดตั้งแต่เริ่มทำงาน

ส่วนที่ 2: การเตรียมข้อมูล (Data Preprocessing) ฟังก์ชัน load_data ทำหน้าที่โหลดไฟล์ CSV ของชุดข้อมูลดอกไอริส (Iris Dataset) เข้ามาและทำการคัดแยกข้อมูล โดยตัดคอลัมน์ที่ไม่จำเป็นอย่าง Id ออก แยกส่วนที่เป็นคุณลักษณะ (Features) และคำตอบ (Labels) จากนั้นทำการแปลงข้อมูลคำตอบที่เป็นข้อความให้เป็นตัวเลขด้วย LabelEncoder และปรับมาตรฐานข้อมูล (Scaling) ด้วย StandardScaler เพื่อให้ตัวเลขอยู่ในระนาบเดียวกัน จุดสำคัญที่สุดในส่วนนี้คือการแปลงมิติข้อมูล (Reshape) จากตาราง 2 มิติ ให้เป็น 3 มิติ (Samples, Features, 1) เพื่อให้สามารถป้อนเข้าสู่เลเยอร์ Conv1D ของ CNN ได้ พร้อมทั้งทำ One-hot Encoding และแบ่งข้อมูลเป็นชุด Train/Test ในอัตราส่วน 80:20

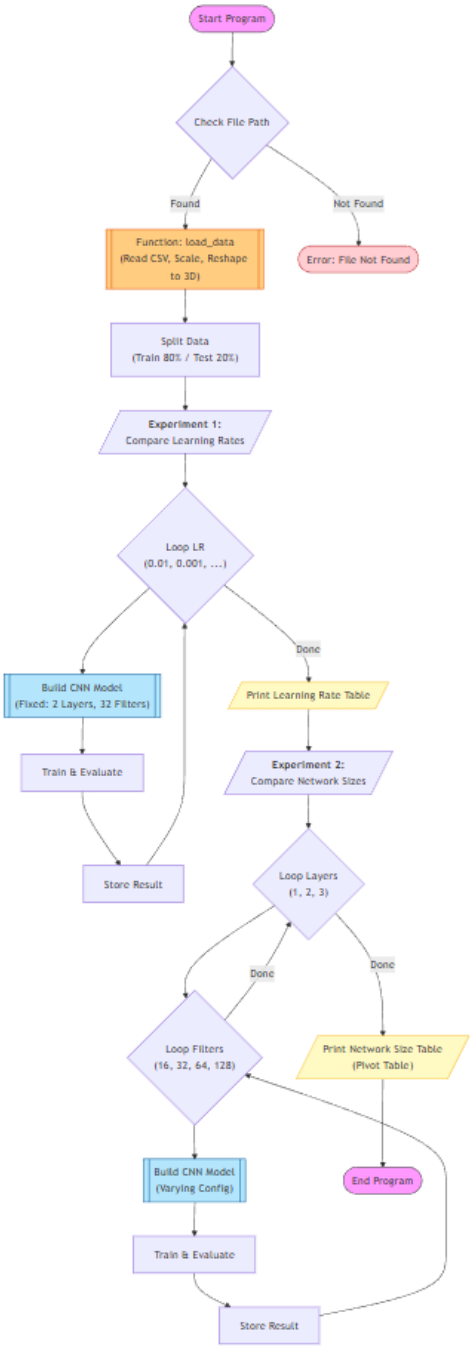
ส่วนที่ 3: การสร้างโครงสร้างโมเดล (Model Architecture) ฟังก์ชัน build_cnn1d_model ถูกออกแบบมาให้สร้างโมเดล CNN แบบ 1 มิติ (1D-CNN) ที่มีความยืดหยุ่นสูง โดยใช้ลูป for สร้างชั้น Conv1D ตามจำนวนเลเยอร์ที่กำหนด และมีการใส่เงื่อนไขพิเศษสำหรับ Pooling Layer คือจะใส่ก็ต่อเมื่อข้อมูลขาเข้ามีขนาดใหญ่พอเท่านั้น (ป้องกัน Error หากข้อมูลถูกย่อจนเหลือ 0 เพราะ Iris มี Feature น้อย) จากนั้นส่งข้อมูลเข้าสู่ชั้น Flatten และ Dense Layer ตามลำดับ โดยใช้ฟังก์ชันกระตุ้นแบบ ReLU ในชั้นซ่อน และ Softmax ในชั้นสุดท้ายเพื่อจำแนกสายพันธุ์ดอกไม้ 3 ประเภท

ส่วนที่ 4: การทดลองและฝึกสอน (Training Experiment) ในฟังก์ชัน main จะแบ่งการทดลองออกเป็น 2 ส่วนหลัก:

1. **การทดลองที่ 1 (Learning Rate):** ทดสอบปรับค่าอัตราการเรียนรู้ (Learning Rate) ตั้งแต่ 0.01 ถึง 0.00001 โดยใช้โครงสร้างโมเดลคงที่ (2 Layers, 32 Filters) เพื่อหาค่าที่ทำให้โมเดลเรียนรู้ได้ดีที่สุด
2. **การทดลองที่ 2 (Network Size):** ทดสอบปรับขนาดโครงสร้างโมเดลแบบ Grid Search โดยแปรผันจำนวนชั้น (1-3 Layers) และจำนวนฟิลเตอร์ (16-128 Filters) เพื่อเปรียบเทียบประสิทธิภาพความแม่นยำของแต่ละโครงสร้าง

ส่วนที่ 5: การสรุปผล (Result Reporting) ส่วนสุดท้ายเป็นการรวบรวมค่าความแม่นยำ (Accuracy) ที่ได้จากการทดลองทั้งสองส่วนมาแสดงผล โดยใช้ Pandas DataFrame ในการจัดรูปแบบข้อมูล การทดลองเรื่อง Learning Rate จะแสดงเป็นตารางรายการปกติ ส่วนการทดลองเรื่อง Network Size จะแสดงผลในรูปแบบ ...

Flowchart:


Start Program
Check File Path
Found
Not Found
Function: load_data
(Read CSV, Scale, Reshape to 3D)
Error: File Not Found
Split Data
(Train 80% / Test 20%)
Experiment 1:
Compare Learning Rates
Loop LR
(0.01, 0.001, ...)
Done
Build CNN Model
(Fixed: 2 Layers, 32 Filters)
Print Learning Rate Table
Train & Evaluate
Experiment 2:
Compare Network Sizes
Store Result
Loop Layers
(1, 2, 3)
Done
Done
Loop Filters
(16, 32, 64, 128)
Print Network Size Table
(Pivot Table)
Build CNN Model
(Varying Config)
End Program
Train & Evaluate
Store Result

Result:

```
ning/CNN/LAB3_CNN_Iris.py"
ut_dim` argument to a layer. When using Sequential models, prefer using an
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Testing: 1 Layers, 32 Filters...
Testing: 1 Layers, 64 Filters...
Testing: 1 Layers, 128 Filters...
Testing: 2 Layers, 16 Filters...
Testing: 2 Layers, 32 Filters...
Testing: 2 Layers, 64 Filters...
Testing: 2 Layers, 128 Filters...
Testing: 3 Layers, 16 Filters...
Testing: 3 Layers, 32 Filters...
Testing: 3 Layers, 64 Filters...
Testing: 3 Layers, 128 Filters...

Results Table (Network Sizes):
Filters       16   32        64   128
Layers
1        0.966667  1.0  0.966667  1.0
2        1.000000  1.0  1.000000  1.0
3        1.000000  1.0  1.000000  1.0
================================================
```

LEB 4: CNN on Microscopic Fungi Classification

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

Objective:

- Apply CNN to Fungi classify.

Contents:

- Load the Fungi dataset from directory.
- Split the data into training and testing sets.
- Train CNN 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 Convolutional 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:

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import pathlib

# ตั้งค่า GPU
physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)

def get_dataset_paths():
    # หา path ของ dataset
    base_dir = os.path.dirname(__file__)
    dataset_dir = os.path.join(base_dir, '..', 'dataset', 'fungi')

    # Check
    if not os.path.exists(dataset_dir):
        # Fallback path if running from root
        dataset_dir = 'dataset/fungi'

    train_dir = os.path.join(dataset_dir, 'train')
    valid_dir = os.path.join(dataset_dir, 'valid')
    test_dir = os.path.join(dataset_dir, 'test')

    return train_dir, valid_dir, test_dir

def load_data(img_size=(150, 150), batch_size=32):
    train_dir, valid_dir, test_dir = get_dataset_paths()

    print(f"Loading data from: {train_dir}")

    if not os.path.exists(train_dir):
        print(f"Error: Directory not found!")
        return None, None, None, None

    # Load using image_dataset_from_directory
    train_ds = tf.keras.utils.image_dataset_from_directory(
        train_dir,
        seed=123,
        image_size=img_size,
        batch_size=batch_size,
        label_mode='categorical'
    )

    val_ds = tf.keras.utils.image_dataset_from_directory(
        valid_dir,
        seed=123,
```

```
        image_size=img_size,
        batch_size=batch_size,
        label_mode='categorical'
    )

    test_ds = tf.keras.utils.image_dataset_from_directory(
        test_dir,
        seed=123,
        image_size=img_size,
        batch_size=batch_size,
        label_mode='categorical'
    )

    class_names = train_ds.class_names
    num_classes = len(class_names)
    print(f"Found {num_classes} classes: {class_names}")

    # Optimize datasets
    AUTOTUNE = tf.data.AUTOTUNE
    train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
    val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
    test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)

    return train_ds, val_ds, test_ds, class_names

def build_cnn_model(num_layers, num_filters, input_shape, num_classes,
learning_rate=0.001):
    model = models.Sequential()

    # Rescaling Layer
    model.add(layers.Rescaling(1./255, input_shape=input_shape))

    # Conv Layers
    for i in range(num_layers):
        if i == 0:
            model.add(layers.Conv2D(num_filters, (3, 3), activation='relu',
padding='same'))
        else:
            model.add(layers.Conv2D(num_filters, (3, 3), activation='relu',
padding='same'))

        if i % 2 == 0:
            model.add(layers.MaxPooling2D((2, 2)))

    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))

    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)

    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
```

```
                  metrics=['accuracy'])
    return model

def main():
    IMG_SIZE = (150, 150)
    BATCH_SIZE = 32

    train_ds, val_ds, test_ds, class_names = load_data(IMG_SIZE, BATCH_SIZE)

    if train_ds is None: return

    input_shape = IMG_SIZE + (3,)
    num_classes = len(class_names)

    # --- Experiment 1: Learning Rates ---
    print("\n" + "="*50)
    print(" Experiment 1: Different Learning Rates")
    print(" (Fixed Architecture: 2 Layers, 32 Filters)")
    print("="*50)

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

    for lr in lrs_to_test:
        print(f"Testing Learning Rate: {lr}")
        model = build_cnn_model(num_layers=2, num_filters=32,
input_shape=input_shape, num_classes=num_classes, learning_rate=lr)

        # Train (Short epochs for demo)
        model.fit(train_ds, validation_data=val_ds, epochs=3, verbose=1)

        loss, acc = model.evaluate(test_ds, verbose=0)
        print(f" -> Accuracy: {acc:.4f}")
        results_lr.append({'Learning Rate': lr, 'Accuracy': acc})

    df_lr = pd.DataFrame(results_lr)
    print("\nResults Table (Learning Rate):")
    print(df_lr)

    # --- Experiment 2: Network Sizes ---
    print("\n" + "="*50)
    print(" Experiment 2: Network Sizes (Layers & Nodes)")
    print(" (Fixed Learning Rate: 0.001)")
    print("="*50)

    layers_to_test = [1, 2, 3]
    filters_to_test = [32, 64]
    results_size = []

    best_model = None
    best_acc = 0

```

```
    for l in layers_to_test:
        for f in filters_to_test:
            print(f"\nTesting: {l} Layers, {f} Filters...")
            try:
                model = build_cnn_model(num_layers=l, num_filters=f,
input_shape=input_shape, num_classes=num_classes)
                model.fit(train_ds, validation_data=val_ds, epochs=3, verbose=1)

                loss, acc = model.evaluate(test_ds, verbose=0)
                results_size.append({'Layers': l, 'Filters': f, 'Accuracy': acc})

                if acc > best_acc:
                    best_acc = acc
                    best_model = model

            except Exception as e:
                print(f"Error: {e}")

    df_size = pd.DataFrame(results_size)
    if not df_size.empty:
        pivot_size = df_size.pivot(index='Layers', columns='Filters',
values='Accuracy')
        print("\nResults Table (Network Sizes):")
        print(pivot_size)
    print("="*50)

    # Output Predictions
    if best_model:
        print("\nSample Predictions from Best Model:")
        plt.figure(figsize=(15, 5))
        for images, labels in test_ds.take(1):
            predictions = best_model.predict(images)
            for i in range(5):
                ax = plt.subplot(1, 5, i + 1)
                plt.imshow(images[i].numpy().astype("uint8"))

                pred_label = class_names[np.argmax(predictions[i])]
                true_label = class_names[np.argmax(labels[i])]

                color = 'green' if pred_label == true_label else 'red'
                plt.title(f"P:{pred_label}\nT:{true_label}", color=color)
                plt.axis("off")
        plt.show()

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

ให้ข้อคิดเห็น **[ตบ4]:** ใช้ AI

ส่วนที่ 1: การตั้งค่าและเรียกใช้ไลบรารี (Setup & GPU Config) เริ่มต้นด้วยการนำเข้าไลบรารีที่จำเป็นสำหรับ Deep Learning และการจัดการไฟล์ ได้แก่ TensorFlow, Keras, NumPy, Pandas, Matplotlib และ OS (สำหรับการจัดการ Path ไฟล์) โดยมีการตรวจสอบฮาร์ดแวร์เพื่อดูว่ามี GPU หรือไม่ หากพบระบบจะตั้งค่า set_memory_growth เป็น True เพื่อบริหารจัดการหน่วยความจำการ์ดจอให้ขยายตัวตามการใช้งานจริง ป้องกันปัญหา Out of Memory (OOM) เมื่อต้องประมวลผลรูปภาพจำนวนมาก

ส่วนที่ 2: การโหลดและเตรียมข้อมูลภาพ (Data Loading from Directory) ฟังก์ชัน load_data และ get_dataset_paths ทำหน้าที่ค้นหาโฟลเดอร์เก็บรูปภาพ (Fungi Dataset) และโหลดรูปภาพเข้าสู่ระบบโดยตรงจากฮาร์ดดิสก์ผ่านคำสั่ง image_dataset_from_directory ซึ่งเป็นวิธีมาตรฐานสำหรับการจัดการชุดข้อมูลรูปภาพขนาดใหญ่ ข้อมูลจะถูกโหลดตามโครงสร้างโฟลเดอร์ (Train, Valid, Test) และปรับขนาดภาพ (Resize) ให้เป็น 150x150 พิกเซล พร้อมทั้งทำ One-hot Encoding ให้กับฉลากคำตอบ (Labels) โดยอัตโนมัติ นอกจากนี้ยังมีการใช้เทคนิค prefetch, cache, และ shuffle (AUTOTUNE) เพื่อเพิ่มประสิทธิภาพการส่งข้อมูลเข้าสู่ GPU ทำให้การเทรนโมเดลรวดเร็วขึ้น

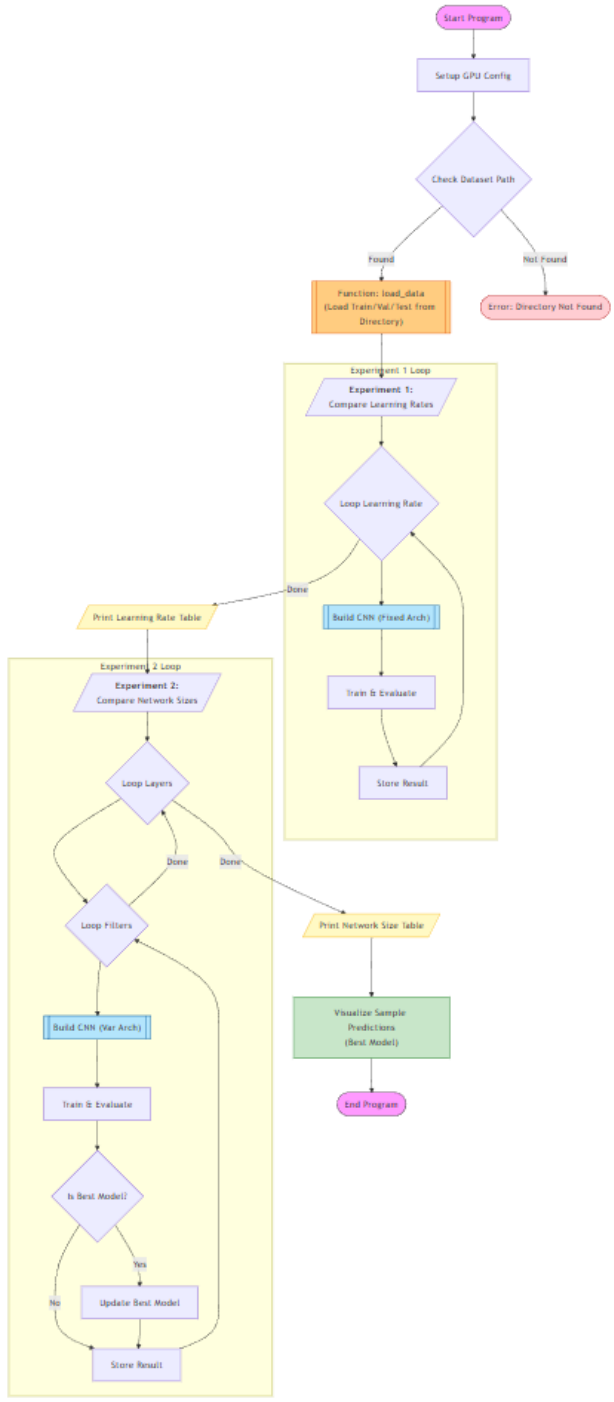
ส่วนที่ 3: การสร้างโครงสร้างโมเดล CNN (Model Architecture) ฟังก์ชัน build_cnn_model เริ่มต้นด้วยการใส่ชั้น **Rescaling** (1./255) เป็นด่านแรกเพื่อปรับค่าสีของพิกเซลจาก 0-255 ให้อยู่ในช่วง 0-1 ซึ่งช่วยให้โมเดลคำนวณได้แม่นยำขึ้น จากนั้นสร้างเลเยอร์แบบ Dynamic ตามจำนวนชั้นที่กำหนด โดยใช้ Conv2D สำหรับสกัดฟีเจอร์ และมีการใส่ MaxPooling2D ทุกๆ 2 ชั้นเพื่อลดขนาดมิติของภาพลง ปิดท้ายด้วยการ Flatten ข้อมูลส่งเข้าสู่ Dense Layer และจบด้วย Softmax เพื่อจำแนกประเภทของเห็ดรา

ส่วนที่ 4: การทดลองและฝึกสอน (Training Experiment) ในฟังก์ชัน main มีการออกแบบการทดลองออกเป็น 2 ส่วน:

1. **การทดลองที่ 1 (Learning Rate):** ทดสอบค่า Learning Rate ที่แตกต่างกัน 4 ระดับ (0.01 - 0.00001) โดยใช้โครงสร้างโมเดลคงที่ เพื่อหาอัตราการเรียนรู้ที่เหมาะสมที่สุดที่ไม่ทำให้กราฟการเรียนรู้แกว่งหรือช้าเกินไป
2. **การทดลองที่ 2 (Network Size):** ทดสอบปรับขนาดโครงสร้างโมเดลโดยแปรผันจำนวนชั้น (1-3 Layers) และจำนวนฟิลเตอร์ (32-64 Filters) เพื่อเปรียบเทียบหาโครงสร้างที่ให้ความแม่นยำสูงสุด

ส่วนที่ 5: การสรุปผลและแสดงภาพ (Reporting & Visualization) ส่วนสุดท้ายเป็นการรวบรวมผลลัพธ์ความแม่นยำ (Accuracy) จากทุกการทดลองมาแสดงในรูปแบบตาราง Pivot Table ผ่าน Pandas เพื่อเปรียบเทียบประสิทธิภาพได้ง่าย และนำ ...

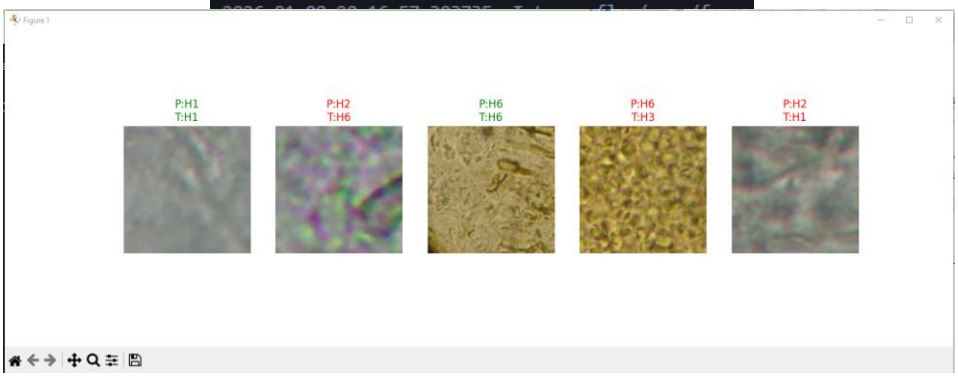
Flowchart :


Start Program
Setup GPU Config
Check Dataset Path
Found
Not Found
Function: load_data
(Load Train/Val./Test from
Directory)
Error: Directory Not Found
Experiment 1 Loop
Experiment 1:
Compare Learning Rates
Loop Learning Rate
Done
Print Learning Rate Table
Build CNN (Fixed Arch)
Train & Evaluate
Store Result
Experiment 2 Loop
Experiment 2:
Compare Network Sizes
Loop Layers
Done
Done
Loop Filters
Print Network Size Table
Build CNN (Var Arch)
Visualize Sample
Predictions
(Best Model)
Train & Evaluate
End Program
Is Best Model?
Yes
No
Update Best Model
Store Result

Result:

```
Results Table (Network Sizes):
Filters        32        64
Layers
1         0.544346  0.611973
2         0.606430  0.552106
3         0.594235  0.503326
================================================

Sample Predictions from Best Model:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 306ms/step
```



LEB 5: CNN for Blood Cells

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

Objective:

- Apply CNN to Blood Cells classify.

Contents:

- Load the Blood Cells dataset from directory.
- Split the data into training and testing sets.
- Train CNN 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 Convolutional 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:

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os

# ตั้งค่า GPU
physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)

def get_dataset_path():
    base_dir = os.path.dirname(__file__)
    dataset_dir = os.path.join(base_dir, '..', 'dataset', 'bloodcells')

    if not os.path.exists(dataset_dir):
        # Fallback
        dataset_dir = 'dataset/bloodcells'

    return dataset_dir

def load_data(img_size=(150, 150), batch_size=32):
    dataset_dir = get_dataset_path()
    print(f"Loading data from: {dataset_dir}")

    if not os.path.exists(dataset_dir):
        print(f"Error: Directory not found!")
        return None, None, None

    # Load and split using validation_split
    # Subset 'training' -> Train set
    train_ds = tf.keras.utils.image_dataset_from_directory(
        dataset_dir,
        validation_split=0.2, # 20% for test
        subset="training",
        seed=123,
        image_size=img_size,
        batch_size=batch_size,
        label_mode='categorical'
    )

    # Subset 'validation' -> Test set (in this context)
    test_ds = tf.keras.utils.image_dataset_from_directory(
        dataset_dir,
        validation_split=0.2,
        subset="validation",
        seed=123,
        image_size=img_size,
```

```
        batch_size=batch_size,
        label_mode='categorical'
    )
    class_names = train_ds.class_names
    num_classes = len(class_names)
    print(f"Found {num_classes} classes: {class_names}")
    AUTOTUNE = tf.data.AUTOTUNE
    train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
    test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
    return train_ds, test_ds, class_names
def build_cnn_model(num_layers, num_filters, input_shape, num_classes,
learning_rate=0.001):
    model = models.Sequential()
    model.add(layers.Rescaling(1./255, input_shape=input_shape))

    for i in range(num_layers):
        if i == 0:
            model.add(layers.Conv2D(num_filters, (3, 3), activation='relu',
padding='same'))
        else:
            model.add(layers.Conv2D(num_filters, (3, 3), activation='relu',
padding='same'))
        if i % 2 == 0:
            model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
def main():
    IMG_SIZE = (150, 150)
    BATCH_SIZE = 32
    train_ds, test_ds, class_names = load_data(IMG_SIZE, BATCH_SIZE)
    if train_ds is None: return
    input_shape = IMG_SIZE + (3,)
    num_classes = len(class_names)
    print("\n" + "="*50)
    print(" Experiment 1: Different Learning Rates")
    print(" (Fixed Architecture: 2 Layers, 32 Filters)")
    print("="*50)
    lrs_to_test = [0.01, 0.001, 0.0001, 0.00001]
    results_lr = []
    for lr in lrs_to_test:
        print(f"Testing Learning Rate: {lr}")
        model = build_cnn_model(num_layers=2, num_filters=32,
input_shape=input_shape, num_classes=num_classes, learning_rate=lr)
        mode = model.fit(train_ds, epochs=3, verbose=1) # 3 epochs for speed
        loss, acc = model.evaluate(test_ds, verbose=0)
        print(f" -> Accuracy: {acc:.4f}")
```

```
            results_lr.append({'Learning Rate': lr, 'Accuracy': acc})
        df_lr = pd.DataFrame(results_lr)
        print("\nResults Table (Learning Rate):")
        print(df_lr)
        print("\n" + "="*50)
        print(" Experiment 2: Network Sizes (Layers & Nodes)")
        print(" (Fixed Learning Rate: 0.001)")
        print("="*50)
        layers_to_test = [1, 2, 3]
        filters_to_test = [32, 64]
        results_size = []
        best_model = None
        best_acc = 0
        for l in layers_to_test:
            for f in filters_to_test:
                print(f"\nTesting: {l} Layers, {f} Filters...")
                try:
                    model = build_cnn_model(num_layers=l, num_filters=f,
input_shape=input_shape, num_classes=num_classes)
                    model.fit(train_ds, epochs=3, verbose=1)
                    loss, acc = model.evaluate(test_ds, verbose=0)
                    results_size.append({'Layers': l, 'Filters': f, 'Accuracy': acc})
                    if acc > best_acc:
                        best_acc = acc
                        best_model = model
                except Exception as e:
                    print(f"Error: {e}")
        df_size = pd.DataFrame(results_size)
        if not df_size.empty:
            pivot_size = df_size.pivot(index='Layers', columns='Filters',
values='Accuracy')
            print("\nResults Table (Network Sizes):")
            print(pivot_size)
        print("="*50)
        if best_model:
            print("\nSample Predictions from Best Model:")
            plt.figure(figsize=(15, 5))
            for images, labels in test_ds.take(1):
                predictions = best_model.predict(images)
                for i in range(5):
                    ax = plt.subplot(1, 5, i + 1)
                    plt.imshow(images[i].numpy().astype("uint8"))
                    pred_label = class_names[np.argmax(predictions[i])]
                    true_label = class_names[np.argmax(labels[i])]
                    color = 'green' if pred_label == true_label else 'red'
                    plt.title(f"P:{pred_label}\nT:{true_label}", color=color)
                    plt.axis("off")
            plt.show()
if __name__ == "__main__":
    main()
```

ให้ข้อคิดเห็น **[ตย5]:** ใช้ AI

ส่วนที่ 1: การตั้งค่าและเรียกใช้ไลบรารี (Setup & GPU Config) เริ่มต้นด้วยการนำเข้าไลบรารีที่จำเป็น ได้แก่ TensorFlow, Keras, NumPy, Pandas, Matplotlib และ OS สำหรับจัดการไฟล์ ในส่วนนี้มีการเขียนคำสั่งตรวจสอบฮาร์ดแวร์เพื่อเช็คว่าเครื่องมี GPU หรือไม่ หากพบระบบจะตั้งค่า set_memory_growth เป็น True เพื่อบริหารจัดการหน่วยความจำการ์ดจอให้ขยายตัวตามการใช้งานจริง ซึ่งสำคัญมากสำหรับการประมวลผลข้อมูลภาพทางการแพทย์ที่มีความละเอียด

ส่วนที่ 2: การโหลดและเตรียมข้อมูลภาพ (Data Loading & Splitting) ฟังก์ชัน load_data ถูกออกแบบมาให้โหลดภาพจากโฟลเดอร์โดยใช้คำสั่ง image_dataset_from_directory ซึ่งมีความสามารถพิเศษในการแบ่งข้อมูลเป็นชุด Train และ Validation ได้ในตัว โดยกำหนด validation_split=0.2 เพื่อแบ่งข้อมูล 80% สำหรับการฝึกสอน (Training) และ 20% สำหรับการทดสอบ (Validation/Testing) โดยอัตโนมัติ นอกจากนี้ยังมีการใช้เทคนิค AUTOTUNE (Cache, Shuffle, Prefetch) เพื่อเพิ่มความเร็วในการส่งข้อมูลเข้าสู่โมเดลขณะเทรน

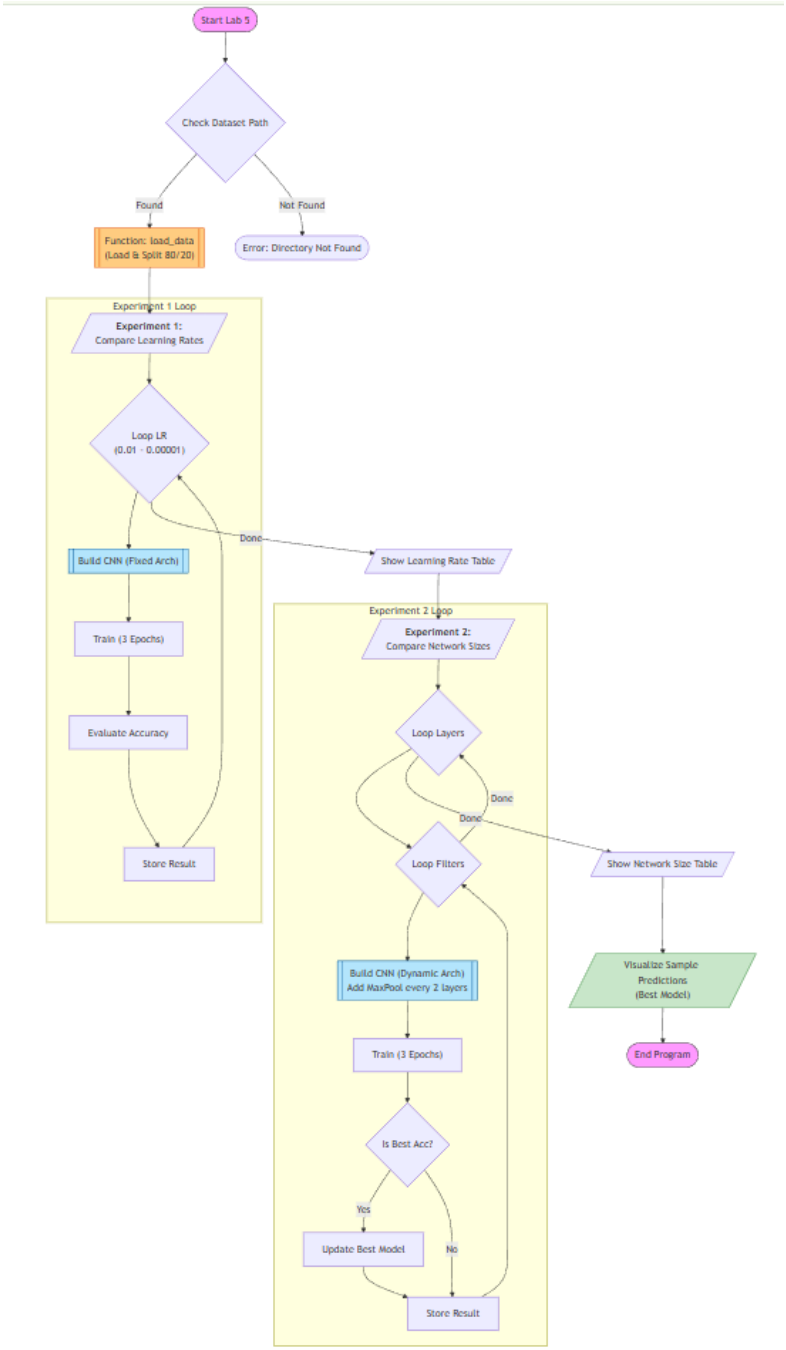
ส่วนที่ 3: การสร้างโครงสร้างโมเดล CNN (Model Architecture) ฟังก์ชัน build_cnn_model เริ่มต้นด้วยการปรับมาตรฐานข้อมูลภาพ (Normalization) ผ่านชั้น Rescaling(1./255) เพื่อแปลงค่าสีให้อยู่ในช่วง 0-1 จากนั้นสร้างโครงสร้างโมเดลแบบ Dynamic ตามพารามิเตอร์ที่กำหนด โดยใช้ Convolutional Layer เพื่อสกัดฟีเจอร์ของเซลล์เม็ดเลือด และมีการใส่ MaxPooling ทุกๆ 2 เลเยอร์เพื่อลดขนาดภาพลงอย่างเหมาะสม ปิดท้ายด้วยการ Flatten ข้อมูลส่งเข้าสู่ Dense Layer และใช้ Softmax Activation เพื่อจำแนกประเภทของเซลล์เม็ดเลือด 4 ชนิด (Eosinophil, Lymphocyte, Monocyte, Neutrophil)

ส่วนที่ 4: การทดลองและฝึกสอน (Training Experiment) ในฟังก์ชัน main ได้แบ่งการทดลองออกเป็น 2 ส่วนหลักเพื่อศึกษาปัจจัยที่ส่งผลต่อประสิทธิภาพโมเดล:

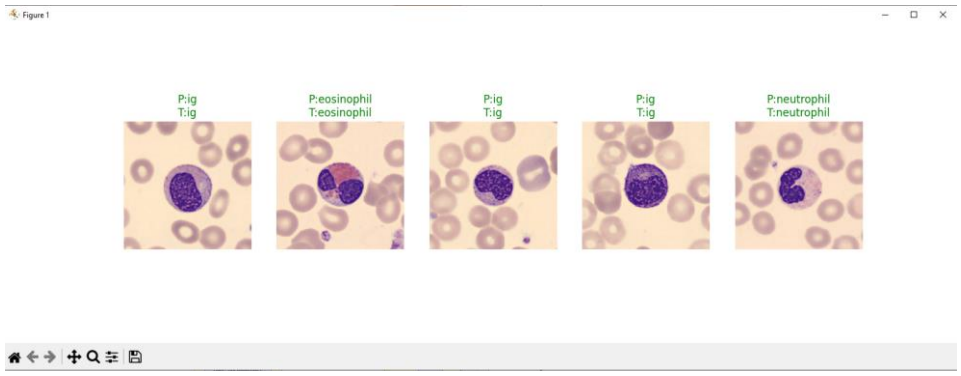
1. **การทดลองที่ 1 (Learning Rate):** ทดสอบค่า Learning Rate ตั้งแต่ 0.01 ถึง 0.00001 โดยใช้โครงสร้างโมเดลคงที่ (2 Layers, 32 Filters) เพื่อหาอัตราการเรียนรู้ที่ทำให้โมเดลลู่เข้าสู่ค่าความแม่นยำสูงสุดได้ดีที่สุด
2. **การทดลองที่ 2 (Network Size):** ทดสอบปรับเปลี่ยนความซับซ้อนของโมเดลโดยแปรผันจำนวนชั้น (1-3 Layers) และจำนวนฟิลเตอร์ (32-64 Filters) เพื่อเปรียบเทียบหาโครงสร้างที่เหมาะสมที่สุดสำหรับโจทย์นี้

ส่วนที่ 5: การสรุปผลและแสดงภาพ (Reporting & Visualization) ส่วนสุดท้ายเป็นการรวบรวมผลลัพธ์ความแม่นยำ (Accuracy) มาแสดงในรูปแบบตารางเปรียบเทียบ โดยใช้ Pandas DataFrame และ Pivot Table เพื่อให้เห็นภาพรวมประสิทธิภาพของแต่ละการตั้งค่าได้ชัดเจน นอกจากนี้ยังนำโมเดลที่ให้ค่าความ...

Flowchart :


Start Lab 5
Check Dataset Path
Found
Not Found
Function: load_data
(Load & Split 80/20)
Error: Directory Not Found
Experiment 1 Loop
Experiment 1:
Compare Learning Rates
Loop LR
(0.01 - 0.00001)
Done
Build CNN (Fixed Arch)
Train (3 Epochs)
Evaluate Accuracy
Store Result
Show Learning Rate Table
Experiment 2 Loop
Experiment 2:
Compare Network Sizes
Loop Layers
Done
Done
Loop Filters
Show Network Size Table
Build CNN (Dynamic Arch)
Add MaxPool every 2 layers
Visualize Sample
Predictions
(Best Model)
Train (3 Epochs)
End Program
Is Best Acc?
Yes
No
Update Best Model
Store Result

Result:




```
Results Table (Network Sizes):
Filters        32        64
Layers
1        0.651551  0.881510
2        0.893212  0.850205
3        0.926858  0.897893
================================================

Sample Predictions from Best Model:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 212ms/step
```

LEB 6: CNN with Smoothed COVID-19 Data.

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

Objective:

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

Contents:

- Load the smoothed COVID-19 dataset from a CSV file.
- Preprocess the data and split it into training and testing sets.
- Construct and train CNN models (1-10 convolutional layers with 10–1000 nodes per layer).
- Evaluate the performance of each CNN model using the accuracy metric.

Output:

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

Code or Algorithms:

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
import os

# ตั้งค่า GPU
physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)

def load_data():
    base_dir = os.path.dirname(__file__)
    file_path = os.path.join(base_dir, '..', 'dataset', 'covid', 'owid-covid-
data.csv')

    if not os.path.exists(file_path):
        # Fallback
        file_path = 'dataset/covid/owid-covid-data.csv'
        if not os.path.exists(file_path):
            print(f"Error: File not found at {file_path}")
            return None

    print(f"Loading data from {file_path}...")
    df = pd.read_csv(file_path)

    # 1. Filter for Thailand
    df_tha = df[df['iso_code'] == 'THA'].copy()

    # 2. Select target: new_cases_smoothed
    # Fill NaN with 0 for early days
    df_tha['new_cases_smoothed'] = df_tha['new_cases_smoothed'].fillna(0)

    # Sort by date
    df_tha['date'] = pd.to_datetime(df_tha['date'])
    df_tha = df_tha.sort_values('date')

    data = df_tha['new_cases_smoothed'].values.reshape(-1, 1)
    dates = df_tha['date'].values

    print(f"Loaded Thailand data: {len(data)} days.")
    return data, dates

def create_sequences(data, lookback_window=30):
    X, y = [], []
    for i in range(len(data) - lookback_window):
        X.append(data[i:i+lookback_window])
```

```
        y.append(data[i+lookback_window])
    return np.array(X), np.array(y)

def build_cnn1d_regression(num_layers, num_nodes, input_shape):
    model = models.Sequential()

    for i in range(num_layers):
        if i == 0:
            model.add(layers.Conv1D(num_nodes, kernel_size=3, activation='relu',
padding='same', input_shape=input_shape))
        else:
            model.add(layers.Conv1D(num_nodes, kernel_size=3, activation='relu',
padding='same'))

        if i % 2 == 0:
            model.add(layers.MaxPooling1D(pool_size=2, padding='same'))

    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1, activation='linear')) # Regression

    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

def calculate_accuracy(y_true, y_pred):
    # Custom "Accuracy" for regression: 1 - (MAE / Mean)
    mae = mean_absolute_error(y_true, y_pred)
    mean_val = np.mean(y_true)
    if mean_val == 0: return 0
    accuracy = (1 - (mae / mean_val)) * 100
    return max(0, accuracy)

def main():
    # 1. Load Data
    data, dates = load_data()
    if data is None: return

    # 2. Preprocess
    scaler = MinMaxScaler(feature_range=(0, 1))
    data_scaled = scaler.fit_transform(data)

    LOOKBACK = 30
    X, y = create_sequences(data_scaled, LOOKBACK)

    # Split Train/Test (Time series split, no shuffle)
    split_size = int(len(X) * 0.8)
    X_train, y_train = X[:split_size], y[:split_size]
    X_test, y_test = X[split_size:], y[split_size:]

    print(f"Train size: {len(X_train)}, Test size: {len(X_test)}")

    input_shape = (LOOKBACK, 1)
```

```
    # --- Experiment: Network Sizes ---
    print("\n" + "="*50)
    print(" Experiment: Network Sizes (Layers & Nodes)")
    print("="*50)

    layers_to_test = [1, 2, 3]
    nodes_to_test = [32, 64, 128]
    results = []

    best_model = None
    best_acc = 0

    for l in layers_to_test:
        for n in nodes_to_test:
            print(f"Testing: {l} Layers, {n} Nodes...")
            model = build_cnn1d_regression(l, n, input_shape)

            model.fit(X_train, y_train, epochs=20, batch_size=32, verbose=0)

            loss, mae = model.evaluate(X_test, y_test, verbose=0)

            # Predict for accuracy calculation
            pred_scaled = model.predict(X_test, verbose=0)
            pred = scaler.inverse_transform(pred_scaled)
            actual = scaler.inverse_transform(y_test)

            acc = calculate_accuracy(actual, pred)

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

            if acc > best_acc:
                best_acc = acc
                best_model = model

    df_results = pd.DataFrame(results)
    pivot_table = df_results.pivot(index='Layers', columns='Nodes', values='Accuracy
(%)')
    print("\nResults Table (Forecast Accuracy %):")
    print(pivot_table)
    print("="*50)

    # --- Forecasting (3 & 6 Months) ---
    if best_model:
        print("\nGenerating Forecasts...")

        # Helper to forecast n steps
        def forecast_future(model, last_sequence, n_steps):
            future_forecast = []
            curr_seq = last_sequence.copy()

            for _ in range(n_steps):
```

```
                # Predict next point
                # curr_seq shape (LOOKBACK, 1) -> (1, LOOKBACK, 1)
                next_pred = model.predict(curr_seq.reshape(1, LOOKBACK, 1),
verbose=0)[0, 0]
                future_forecast.append(next_pred)

                # Update sequence: remove first, add pred
                curr_seq = np.roll(curr_seq, -1)
                curr_seq[-1] = next_pred

            return np.array(future_forecast).reshape(-1, 1)

        # Use the very last data sequence to predict into unknown future
        last_seq = data_scaled[-LOOKBACK:]

        # 3 Months ~ 90 Days
        forecast_3m = forecast_future(best_model, last_seq, 90)
        forecast_3m = scaler.inverse_transform(forecast_3m)

        # 6 Months ~ 180 Days
        forecast_6m = forecast_future(best_model, last_seq, 180)
        forecast_6m = scaler.inverse_transform(forecast_6m)

        # Plotting
        plt.figure(figsize=(12, 6))

        # Plot only last 365 days of actual data for clarity
        plt.plot(dates[-365:], scaler.inverse_transform(data_scaled)[-365:],
label='Actual (Smoothed)', color='blue')

        # Generate future dates
        last_date = dates[-1]
        dates_3m = pd.date_range(start=last_date, periods=91)[1:]
        dates_6m = pd.date_range(start=last_date, periods=181)[1:]

        plt.plot(dates_6m, forecast_6m, label='6-Month Forecast', color='red',
linestyle='--')
        plt.plot(dates_3m, forecast_3m, label='3-Month Forecast', color='orange',
linestyle='--')

        plt.title('COVID-19 Thailand: Smoothed Cases vs CNN Forecast')
        plt.xlabel('Date')
        plt.ylabel('New Cases Smoothed')
        plt.legend()
        plt.grid(True)
        plt.show()

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

ให้ข้อคิดเห็น **[๓6]:** ใช้ AI

ส่วนที่ 1: การตั้งค่าและเรียกใช้ไลบรารี (Setup & GPU Config) เริ่มต้นด้วยการนำเข้าไลบรารีสำคัญสำหรับงาน Deep Learning และ Time Series ได้แก่ TensorFlow, Keras, NumPy, Pandas, Matplotlib และ Scikit-learn (สำหรับ MinMaxScaler และ Metrics) ในส่วนนี้มีการเขียนคำสั่งตรวจสอบฮาร์ดแวร์เพื่อเช็ค GPU และตั้งค่า set_memory_growth เพื่อให้ระบบบริหารจัดการหน่วยความจำการ์ดจออย่างมีประสิทธิภาพ

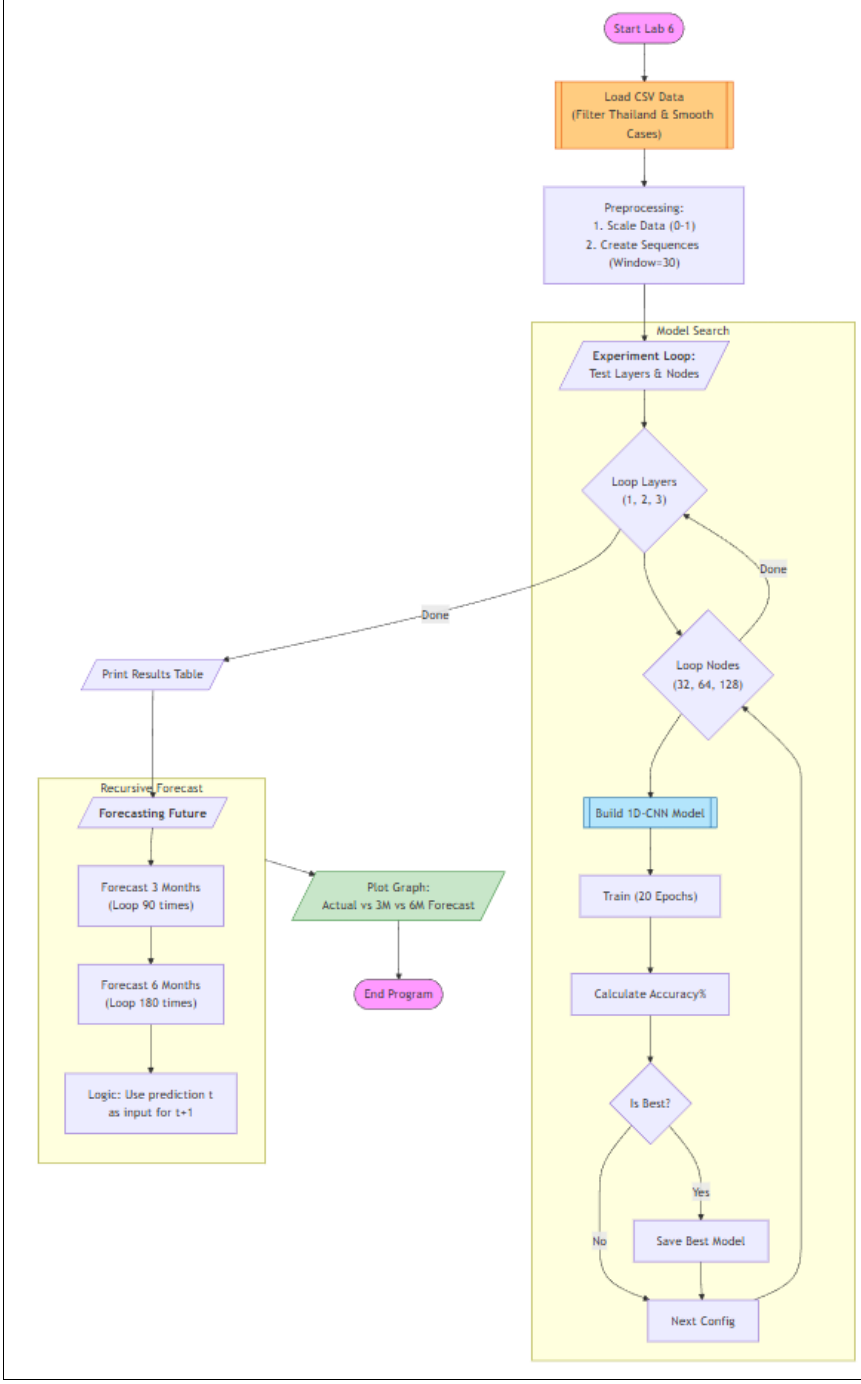
ส่วนที่ 2: การโหลดและเตรียมข้อมูล (Data Loading & Preprocessing) ฟังก์ชัน load_data ทำหน้าที่โหลดไฟล์ CSV ข้อมูล COVID-19 ทั่วโลกและคัดกรองเฉพาะข้อมูลของประเทศไทย (Thailand) จากนั้นเลือกใช้คอลัมน์ new_cases_smoothed (ยอดผู้ติดเชื้อรายวันที่ปรับเรียบแล้ว) เป็นเป้าหมายในการพยากรณ์ โดยมีการเติมค่าสูญย์ (Fill NA) ในช่วงแรกที่ยังไม่มีการระบาดและเรียงลำดับข้อมูลตามเวลา ฟังก์ชัน create_sequences ทำหน้าที่แปลงข้อมูลอนุกรมเวลาให้เป็นรูปแบบ Sliding Window โดยกำหนด lookback_window เท่ากับ 30 วัน เพื่อให้โมเดลใช้ข้อมูล 30 วันย้อนหลังในการทำนายยอดผู้ติดเชื้อในวันถัดไป นอกจากนี้ยังมีการปรับสเกลข้อมูล (Scaling) ให้อยู่ในช่วง 0-1 ด้วย MinMaxScaler เพื่อให้โมเดลเรียนรู้ได้ดีขึ้น

ส่วนที่ 3: การสร้างโครงสร้างโมเดล CNN (Model Architecture) ฟังก์ชัน build_cnn1d_regression สร้างโมเดลแบบ 1D-CNN สำหรับงานถดถอย (Regression) โดยเฉพาะ โครงสร้างประกอบด้วยชั้น Conv1D เพื่อสกัดรูปแบบแนวโน้มจากข้อมูลอนุกรมเวลา และ MaxPooling1D เพื่อลดความซับซ้อนของข้อมูล ปิดท้ายด้วย Flatten และ Dense Layer โดยชั้นสุดท้ายมีเพียง 1 Node และใช้ Activation แบบ Linear เพื่อให้โมเดลสามารถพยากรณ์ค่าตัวเลขต่อเนื่อง (ยอดผู้ติดเชื้อ) ได้โดยตรง

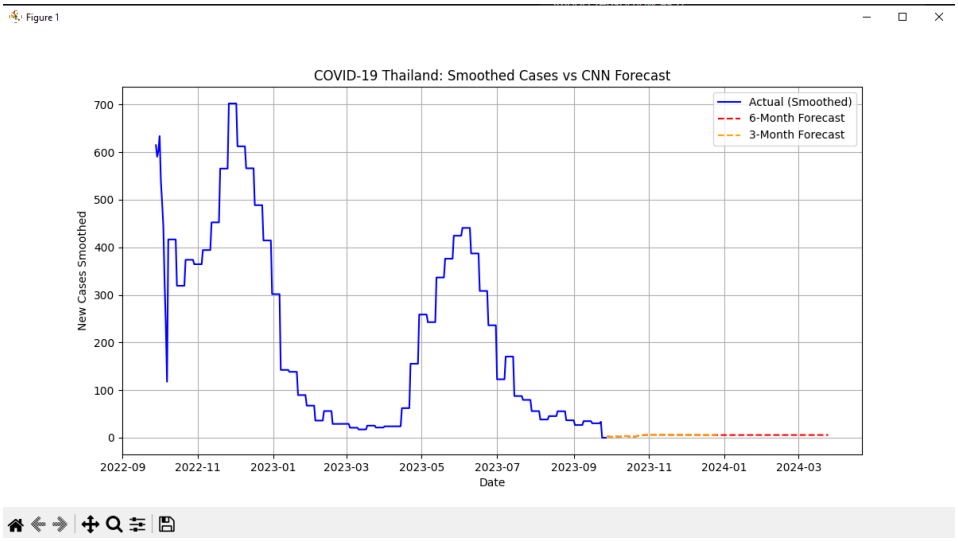
ส่วนที่ 4: การทดลองและฝึกสอน (Training Experiment) ในฟังก์ชัน main ได้ออกแบบการทดลองเพื่อค้นหาโครงสร้างโมเดลที่ดีที่สุด โดยแปรผันจำนวนชั้น (1-3 Layers) และจำนวน Node (32-128 Nodes) โมเดลแต่ละรูปแบบจะถูกเทรนด้วยข้อมูล 80% แรกของช่วงเวลา และทดสอบกับข้อมูล 20% หลัง (Time Series Split) โดยวัดผลด้วยค่าความแม่นยำที่คำนวณจาก Mean Absolute Error (MAE) เทียบกับค่าเฉลี่ยจริง

ส่วนที่ 5: การพยากรณ์และแสดงผล (Forecasting & Visualization) ส่วนสุดท้ายนำโมเดลที่แม่นยำที่สุด (Best Model) มาทำการพยากรณ์ล่วงหน้าแบบ Recursive Forecasting คือการนำค่าที่ทำนายได้ป้อนกลับเป็น Input สำหรับวันถัดไป เพื่อสร้างเส้นกราฟพยากรณ์ล่วงหน้า 3 เดือน (90 วัน) และ 6 เดือน (180 วัน) ผลลัพธ์จะถูกแสดงเป็นกราฟเปรียบเทียบระหว่างข้อมูลจริง (เส้นสีน้ำเงิน) และข้อมูลพยากรณ์ (เส้นประสีส้มและแดง) เพื่อให้เห็นแนวโน้มการระบาดในอนาคตตามมุมมองของ AI

Flowchart:


Start Lab 6
Load CSV Data (Filter Thailand & Smooth Cases)
Preprocessing: 1. Scale Data (0-1) 2. Create Sequences (Window=30)
Model Search
Experiment Loop: Test Layers & Nodes
Loop Layers (1, 2, 3)
Done
Loop Nodes (32, 64, 128)
Done
Build 1D-CNN Model
Train (20 Epochs)
Calculate Accuracy%
Is Best?
Yes
Save Best Model
No
Next Config
Print Results Table
Recursive Forecast
Forecasting Future
Forecast 3 Months (Loop 90 times)
Forecast 6 Months (Loop 180 times)
Logic: Use prediction t as input for t+1
Plot Graph: Actual vs 3M vs 6M Forecast
End Program

Result:




```
Results Table (Forecast Accuracy %):
Nodes         32         64         128
Layers
1       53.312452  40.811127  78.873960
2       59.053725  39.263064  84.545526
3       63.808139  54.916774  53.767442
================================================
```