```
# ==============================
# Skin Disease Classification CNN
# ==============================

# --- 1. Mount Google Drive (if running on Colab) ---
from google.colab import drive
drive.mount('/content/drive')

# --- 2. Import Libraries ---
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Flatten, Dense
from sklearn.metrics import confusion_matrix, precision_recall_fscore_support
import tensorflow as tf

# --- 3. Define Configurable Paths ---
BASE_DIR = "/content/drive/MyDrive/SkinDiseaseProject"
DATASET_INPUT = os.path.join(BASE_DIR, "dataset_raw")      # raw dataset
DATASET_OUTPUT = os.path.join(BASE_DIR, "dataset_split")    # split dataset
MODEL_DIR = os.path.join(BASE_DIR, "models")               # saved models
os.makedirs(MODEL_DIR, exist_ok=True)

# --- 4. Split Data into Train/Val/Test (only once) ---
import splitfolders
```

```
# Uncomment this if you need to split dataset again
# splitfolders.ratio(
#     DATASET_INPUT,
#     output=DATASET_OUTPUT,
#     seed=42,
#     ratio=(0.8, 0.1, 0.1)
# )

# --- 5. Data Generators ---
train_datagen = ImageDataGenerator(rescale=1/255)
valid_datagen = ImageDataGenerator(rescale=1/255)
test_datagen = ImageDataGenerator(rescale=1/255)

train_dataset = train_datagen.flow_from_directory(
    os.path.join(DATASET_OUTPUT, "train"),
    target_size=(200, 200),
    color_mode='grayscale',
    class_mode='categorical',
    batch_size=32
)

valid_dataset = valid_datagen.flow_from_directory(
    os.path.join(DATASET_OUTPUT, "val"),
    target_size=(200, 200),
    color_mode='grayscale',
    class_mode='categorical',
    batch_size=32
)
```

```
test_dataset = test_datagen.flow_from_directory(
    os.path.join(DATASET_OUTPUT, "test"),
    target_size=(200, 200),
    color_mode='grayscale',
    class_mode='categorical',
    batch_size=32,
    shuffle=False
)

# --- 6. Build CNN Model ---
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=train_dataset.image_shape),
    MaxPool2D(2),
    Conv2D(32, (3,3), activation='relu'), MaxPool2D(2),
    Conv2D(64, (3,3), activation='relu'), MaxPool2D(2),
    Conv2D(64, (3,3), activation='relu'), MaxPool2D(2),
    Conv2D(128, (3,3), activation='relu'), MaxPool2D(2),
    Conv2D(128, (3,3), activation='relu'), MaxPool2D(2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(len(train_dataset.class_indices), activation='softmax')
])

model.summary()

# --- 7. Compile Model ---
import keras
METRICS = [
    'accuracy',
```

```
    keras.metrics.Precision(name='precision'),
    keras.metrics.Recall(name='recall')
]

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=METRICS)

# --- 8. Train Model ---
history = model.fit(
    train_dataset,
    validation_data=valid_dataset,
    epochs=30
)

# --- 9. Plot Training History ---
fig, ax = plt.subplots(1, 4, figsize=(20, 3))
ax = ax.ravel()
for i, metric in enumerate(['precision', 'recall', 'accuracy', 'loss']):
    ax[i].plot(history.history[metric])
    ax[i].plot(history.history['val_' + metric])
    ax[i].set_title(metric)
    ax[i].set_xlabel('epochs')
    ax[i].legend(['train', 'val'])

# --- 10. Evaluate Model ---
diseases_labels = list(train_dataset.class_indices.keys())

def evaluate(actual, predictions):
```

```
    predicted_classes = np.argmax(predictions, axis=1)
    accuracy = (predicted_classes == actual).sum() / actual.shape[0]
    print(f'Accuracy: {accuracy:.2f}')
    precision, recall, f1, _ = precision_recall_fscore_support(actual, predicted_classes, average='macro')
    print(f'Precision: {precision:.2f}, Recall: {recall:.2f}, F1: {f1:.2f}')

    plt.figure(figsize=(12,10))
    sns.heatmap(confusion_matrix(actual, predicted_classes),
                annot=True, fmt='d', cmap="Blues",
                xticklabels=diseases_labels,
                yticklabels=diseases_labels)
    plt.title("Confusion Matrix")
    plt.show()

predictions = model.predict(test_dataset)
evaluate(test_dataset.classes, predictions)

# --- 11. Save Model ---
model.save(os.path.join(MODEL_DIR, "skin_disease_model.h5"))
print(f"Model saved at {MODEL_DIR}")

# --- 12. Single Image Prediction Helper ---
def predict_image(img_path):
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=(200,200),
color_mode='grayscale')
    img_array = tf.keras.preprocessing.image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, 0)

    preds = model.predict(img_array)
```

```
    predicted_class = np.argmax(preds[0])
    confidence = np.max(preds[0])

    print(f"Image: {os.path.basename(img_path)}")
    print(f"Predicted Class: {diseases_labels[predicted_class]}")
    print(f"Confidence: {confidence:.2f}")

# Example:
# predict_image("/content/drive/MyDrive/SkinDiseaseProject/sample.jpg")
```