import numpy as np

import os

import cv2

from sklearn.model\_selection import train\_test\_split

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from keras.utils import to\_categorical

from keras.callbacks import ModelCheckpoint

from sklearn.metrics import accuracy\_score

# Function to load and preprocess the dataset

def load\_dataset(dataset\_path):

data = []

labels = []

classes = sorted(os.listdir(dataset\_path))

for i, class\_name in enumerate(classes):

class\_path = os.path.join(dataset\_path, class\_name)

for image\_name in os.listdir(class\_path):

image\_path = os.path.join(class\_path, image\_name)

image = cv2.imread(image\_path)

if image is not None:

# Resize image to a fixed size (e.g., 64x64)

image = cv2.resize(image, (64, 64))

# Normalize pixel values to [0, 1]

image = image / 255.0

data.append(image)

labels.append(i)

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

# Load dataset

dataset\_path = 'path/to/dataset'

data, labels = load\_dataset(dataset\_path)

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, labels, test\_size=0.2, random\_state=42)

# Convert labels to one-hot encoding

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

# Define CNN architecture

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(64, 64, 3)))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Flatten())

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

# Compile model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Define callbacks (e.g., model checkpoint)

checkpoint = ModelCheckpoint('best\_model.h5', monitor='val\_accuracy', mode='max', verbose=1, save\_best\_only=True)

# Train model

history = model.fit(X\_train, y\_train, epochs=20, batch\_size=64, validation\_split=0.2, callbacks=[checkpoint])

# Evaluate model on test set

loss, accuracy = model.evaluate(X\_test, y\_test)

print("Test Accuracy:", accuracy)

# Make predictions

predictions = model.predict(X\_test)

predicted\_labels = np.argmax(predictions, axis=1)

true\_labels = np.argmax(y\_test, axis=1)

test\_accuracy = accuracy\_score(true\_labels, predicted\_labels)

print("Test Accuracy (from predictions):", test\_accuracy)