D:\DOWNLOADS\PRODIGY_ML_04.py

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import zipfile
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Define paths
zip_path = r"C:\Users\aspk1\Videos\leapGestRecog.zip"  # Change this to your zip file path
extract_path = r"C:\Users\aspk1\Videos\leapGestRecog"  # Change this to your extraction path

# Verify if the path is correct
if not os.path.exists(zip_path):
    raise FileNotFoundError(f"Cannot find the file at {zip_path}")

# Extract the dataset
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# Update the dataset path
dataset_path = os.path.join(extract_path, 'leapGestRecog')

```

```
# Verify the extracted directory structure
print(f"Dataset path: {dataset_path}")
if not os.path.exists(dataset_path):
    raise FileNotFoundError(f"Cannot find the directory at {dataset_path}")
else:
    print(f"Contents of the dataset directory: {os.listdir(dataset_path)}")

# Map the gestures to labels
gesture_labels = {
    '01_palm': 0,
    '02_l': 1,
    '03_fist': 2,
    '04_fist_moved': 3,
    '05_thumb': 4,
    '06_index': 5,
    '07_ok': 6,
    '08_palm_moved': 7,
    '09_c': 8,
    '10_down': 9
}

# Initialize lists to store images and labels
images = []
labels = []

# Load images and their corresponding labels
for sub_dir in os.listdir(dataset_path):
    sub_dir_path = os.path.join(dataset_path, sub_dir)
    if os.path.isdir(sub_dir_path):
        for gesture, label in gesture_labels.items():
            gesture_dir = os.path.join(sub_dir_path, gesture)
```

```
            print(f"Checking directory: {gesture_dir}")  # Debugging line
            if not os.path.exists(gesture_dir):
                print(f"Skipping missing directory: {gesture_dir}")
                continue
            for image_file in os.listdir(gesture_dir):
                image_path = os.path.join(gesture_dir, image_file)
                print(f"Loading image: {image_path}")  # Debugging line
                image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Read the image in grayscale
                if image is None:
                    print(f"Skipping corrupted image: {image_path}")
                    continue
                image = cv2.resize(image, (128, 128))  # Resize all images to a fixed size
                images.append(image)
                labels.append(label)

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Debugging information to check if images are loaded correctly
print(f"Total images loaded: {len(images)}")
if len(images) == 0:
    raise ValueError("No images were loaded. Please check the dataset path and structure.")

# Normalize images
```

```
images = images / 255.0

# One-hot encode labels
labels = to_categorical(labels,
num_classes=len(gesture_labels))

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images,
labels, test_size=0.2, random_state=42)

# Debugging information for the dataset split
print(f"Training set size: {X_train.shape[0]}")
print(f"Testing set size: {X_test.shape[0]}")

# Reshape images for the model
X_train = X_train.reshape(-1, 128, 128, 1)
X_test = X_test.reshape(-1, 128, 128, 1)

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
datagen.fit(X_train)

print("Data loaded and preprocessed successfully.")

# Step 2: Model Development

# Build the model
model = Sequential([
```

```
    Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 1)),
    MaxPooling2D((2, 2)),
    Dropout(0.2),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(gesture_labels), activation='softmax')
])

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

model.summary()

# Step 3: Model Training

# Train the model
history = model.fit(datagen.flow(X_train, y_train, batch_size=32), epochs=20, validation_data=(X_test, y_test))

# Step 4: Model Evaluation

# Evaluate the model
```

```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f'Test accuracy: {test_acc:.2f}')

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```