The provided code is a machine learning pipeline for training a Convolutional Neural Network (CNN) model to classify hand gestures using the Leap Gesture Recognition dataset. Below is an explanation of the code:

**1. Importing Libraries**

import os

import cv2

import numpy as np

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.optimizers import Adam

import matplotlib.pyplot as plt

* **os**: Interacts with the operating system for handling file paths.
* **cv2**: OpenCV library used for computer vision tasks (though not used directly here).
* **numpy**: Used for array operations and handling data.
* **tensorflow and keras**: Deep learning framework used to build and train the CNN model.
* **matplotlib**: Used for plotting the training and validation accuracy/loss graphs.

**2. Setting Paths and Image Dimensions**

base\_data\_path = r"C:\Users\sathi\OneDrive\Desktop\WORKSPACE\Prodigy\TASK-4\leapGestRecog" # Path to the dataset

img\_width, img\_height = 128, 128 # Image dimensions for resizing

* base\_data\_path: Specifies the path to the directory where the leapGestRecog dataset is located.
* img\_width, img\_height: Defines the dimensions to which the images will be resized (128x128 pixels).

**3. Data Augmentation**

train\_datagen = ImageDataGenerator(

rescale=1./255,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True

)

validation\_datagen = ImageDataGenerator(rescale=1./255)

* **train\_datagen**: Augments the training images by rescaling pixel values to the range [0,1] and applying random transformations (shear, zoom, horizontal flip) to introduce diversity and help prevent overfitting.
* **validation\_datagen**: Only rescales the validation images (no augmentation) to standardize them.

**4. Loading Data Using ImageDataGenerator**

train\_generator = train\_datagen.flow\_from\_directory(

os.path.join(base\_data\_path, 'train'),

target\_size=(img\_width, img\_height),

batch\_size=32,

class\_mode='categorical'

)

validation\_generator = validation\_datagen.flow\_from\_directory(

os.path.join(base\_data\_path, 'validation'),

target\_size=(img\_width, img\_height),

batch\_size=32,

class\_mode='categorical'

)

* **train\_generator**: Loads the training images from the train directory. It automatically labels images based on their subfolder names (one subfolder per class of gesture) and applies the transformations defined in train\_datagen.
* **validation\_generator**: Loads the validation images from the validation directory and applies only rescaling.

**5. Building the CNN Model**

model = Sequential()

* **Sequential**: Initializes a linear stack of layers for the CNN.

**First Convolutional Layer**

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

model.add(MaxPooling2D(pool\_size=(2, 2)))

* **Conv2D**: Adds a 2D convolutional layer with 32 filters, each of size 3x3. ReLU activation function is applied.
* **MaxPooling2D**: Downsamples the image by taking the maximum value in each 2x2 block.

**Second Convolutional Layer**

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

model.add(MaxPooling2D(pool\_size=(2, 2)))

* Another convolutional layer with 64 filters and max pooling.

**Third Convolutional Layer**

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

model.add(MaxPooling2D(pool\_size=(2, 2)))

* Third convolutional layer with 128 filters and max pooling.

**Flatten Layer**

model.add(Flatten())

* **Flatten**: Flattens the 3D output of the previous convolutional layers into a 1D vector to feed it into the fully connected layers.

**Fully Connected Layers**

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(10, activation='softmax'))

* **Dense**: Fully connected layer with 128 units and ReLU activation.
* **Dropout**: Dropout regularization to reduce overfitting (50% of neurons are dropped during training).
* **Dense(10, activation='softmax')**: The final layer with 10 output units (for 10 gesture classes) and softmax activation to output probabilities.

**6. Compiling the Model**

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

* **Adam optimizer**: Optimizer used for training.
* **Loss function**: Categorical crossentropy (since it's a multi-class classification problem).
* **Metrics**: The accuracy is tracked during training.

**7. Training the Model**

history = model.fit(

train\_generator,

steps\_per\_epoch=train\_generator.samples // train\_generator.batch\_size,

epochs=10,

validation\_data=validation\_generator,

validation\_steps=validation\_generator.samples // validation\_generator.batch\_size

)

* **model.fit**: Starts training the model.
  + steps\_per\_epoch: Defines how many steps (batches) to process per epoch (based on the number of samples and batch size).
  + epochs=10: The number of times the entire dataset will be passed through the model.
  + validation\_steps: Number of validation batches to process at the end of each epoch.

**8. Saving the Model**

model.save("hand\_gesture\_model.h5")

* **model.save**: Saves the trained model in the H5 format.

**9. Plotting Training & Validation Accuracy and Loss**

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val\_accuracy'], label = 'val\_accuracy')

plt.title('Model accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.show()

plt.plot(history.history['loss'], label='loss')

plt.plot(history.history['val\_loss'], label = 'val\_loss')

plt.title('Model loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend(loc='upper right')

plt.show()

* **Matplotlib**: Plots graphs for training and validation accuracy, as well as loss, across all epochs. This helps visualize model performance and check for overfitting.

**Summary**

This code is a complete pipeline for training a deep learning model on hand gesture recognition, including data preprocessing, CNN architecture, model training, evaluation, and saving. It uses data augmentation to improve generalization, with a CNN architecture consisting of convolutional layers, pooling layers, and dense layers. After training, it plots the performance metrics to visualize the model's learning.