Name : Abdul Rehman Section :B

Course : AIES Roll No : CT-22052

==================================================================

**What is an activation function?**

* An activation function determines whether a neuron should be activated or not by processing the input through a mathematical function. It introduces non-linearity into the network, allowing it to learn complex patterns. Common examples include ReLU, Sigmoid, and Tanh.

**What is underfitting and overfitting in neural networks?**

* **Underfitting**: The model is too simple to capture the underlying pattern of the data, leading to poor performance on both training and test data.
* **Overfitting**: The model learns the details and noise in the training data too well, resulting in great performance on training data but poor generalization to new data.

**What are common techniques to prevent overfitting?**

* **Regularization** (L1/L2), **dropout**, **early stopping**, **data augmentation**, and **cross-validation**.

**What is a loss function, and why is it important?**

* A loss function measures the difference between the model's predictions and the actual values. It is used to guide the optimization process, helping the model minimize the error during training.

**What is the purpose of a learning rate in neural networks?**

* The learning rate controls the size of the steps the model takes during optimization (gradient descent). A proper learning rate ensures the model converges efficiently without overshooting the optimal solution.

**Describe Types of Neural Networks:**

* **Feedforward Neural Networks (FNN)**: Simple networks where information moves in one direction from input to output.
* **Convolutional Neural Networks (CNN)**: Primarily used for image and video recognition, utilizing convolutional layers to detect spatial hierarchies.
* **Recurrent Neural Networks (RNN)**: Designed for sequential data, where connections between neurons form cycles, making them suitable for tasks like time series forecasting and language modeling.
* **Generative Adversarial Networks (GANs)**: Consist of two networks, a generator and a discriminator, competing to generate realistic data.
* **Long Short-Term Memory Networks (LSTMs)**: A type of RNN designed to capture long-range dependencies in sequential data.

# FaceRecognition.py

# Face recognition using TensorFlow and a public dataset (LFW subset)

# Trains a simple classifier on extracted features for up to 20 epochs

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import numpy as np

import os

import matplotlib.pyplot as plt

def prepare\_data(data\_dir, img\_size=(160, 160), batch\_size=32):

    # Use ImageDataGenerator for loading and augmenting images

    datagen = ImageDataGenerator(

        rescale=1./255,

        validation\_split=0.2,

        horizontal\_flip=True,

        zoom\_range=0.2,

        rotation\_range=20

    )

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

        data\_dir,

        target\_size=img\_size,

        batch\_size=batch\_size,

        class\_mode='categorical',

        subset='training',

        shuffle=True

    )

    val\_generator = datagen.flow\_from\_directory(

        data\_dir,

        target\_size=img\_size,

        batch\_size=batch\_size,

        class\_mode='categorical',

        subset='validation',

        shuffle=True

    )

    return train\_generator, val\_generator

def build\_model(num\_classes, input\_shape=(160, 160, 3)):

    # Use MobileNetV2 as feature extractor

    base\_model = MobileNetV2(input\_shape=input\_shape, include\_top=False, weights='imagenet')

    base\_model.trainable = False  # Freeze base model

    model = models.Sequential([

        base\_model,

        layers.GlobalAveragePooling2D(),

        layers.Dense(128, activation='relu'),

        layers.Dropout(0.5),

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

    ])

    model.compile(optimizer='adam',

                  loss='categorical\_crossentropy',

                  metrics=['accuracy'])

    return model

def plot\_training(history):

    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)

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

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

    plt.legend()

    plt.title('Accuracy')

    plt.subplot(1, 2, 2)

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

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

    plt.legend()

    plt.title('Loss')

    plt.show()

def main():

    # Please download the LFW dataset manually or use a small subset of face images organized in folders by person name.

    # For example, organize images in 'data/lfw' directory with subfolders for each person.

    data\_dir = 'data/lfw'  # Change this path to your dataset location

    if not os.path.exists(data\_dir):

        print(f"Dataset directory '{data\_dir}' not found. Please download and prepare the dataset.")

        return

    train\_gen, val\_gen = prepare\_data(data\_dir)

    num\_classes = train\_gen.num\_classes

    print(f"Number of classes: {num\_classes}")

    model = build\_model(num\_classes)

    history = model.fit(

        train\_gen,

        validation\_data=val\_gen,

        epochs=20

    )

    plot\_training(history)

    # Save the trained model

    model.save('face\_recognition\_model.h5')

    print("Model saved as face\_recognition\_model.h5")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

