IMPORTING DATASET

from google.colab import drive

drive.mount('/content/drive')

# Navigate to the directory containing the zip file

%cd /content/drive/MyDrive/dataset

# Extract the contents of the zip file

import zipfile

zip\_file\_path = 'faceimagedataset.zip'

extract\_folder\_path = '/content/drive/MyDrive/dataset/extracted1'  # Change to your desired extraction path

with zipfile.ZipFile(zip\_file\_path, 'r') as zip\_ref:

    zip\_ref.extractall(extract\_folder\_path)

# Navigate to the extracted folder

%cd /content/drive/MyDrive/dataset/extracted1

import os

# Replace this with your actual dataset path

dataset\_path = '/content/drive/MyDrive/dataset/extracted1'

# List contents of the dataset path

for root, dirs, files in os.walk(dataset\_path):

    print(f'Root: {root}')

    print(f'Directories: {dirs}')

    print(f'Files: {files}')

    print('------------------')

from PIL import Image

# imports the Image module from the Python Imaging Library (PIL)

import os

# Define the path to your main dataset folder

dataset\_path = '/content/drive/MyDrive/dataset/extracted1/imagedataset'

# Define the path to the preprocessed dataset folder

preprocessed\_path = '/content/drive/MyDrive/preprocessed\_dataset'

# Function to resize images and convert them to grayscale

def preprocess\_images(input\_folder, output\_folder, target\_size=(128, 128)):

    os.makedirs(output\_folder, exist\_ok=True)

    for family\_member in os.listdir(input\_folder):

        family\_member\_path = os.path.join(input\_folder, family\_member)

        output\_member\_folder = os.path.join(output\_folder, family\_member)

        # Ensure it's a directory

        if os.path.isdir(family\_member\_path):

            os.makedirs(output\_member\_folder, exist\_ok=True)

            for filename in os.listdir(family\_member\_path):

                img\_path = os.path.join(family\_member\_path, filename)

                output\_path = os.path.join(output\_member\_folder, filename)

                # Open the image and convert it to grayscale

                img = Image.open(img\_path).convert('L')

                # Resize the image to the target size

                img\_resized = img.resize(target\_size)

                # Save the preprocessed image

                img\_resized.save(output\_path)

# Preprocess images in the main dataset folder

preprocess\_images(dataset\_path, preprocessed\_path)

import tensorflow as tf

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define constants

IMG\_HEIGHT = 128

IMG\_WIDTH = 128

batch\_size = 15

epochs = 20

num\_classes = 3  # Number of family members

# Load preprocessed dataset

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

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

    '/content/drive/MyDrive/preprocessed\_dataset',

    target\_size=(IMG\_HEIGHT, IMG\_WIDTH),

    batch\_size=batch\_size,

    class\_mode='categorical'

)

# Model Architecture

model = Sequential([

    Conv2D(32, (3, 3), activation='relu', input\_shape=(IMG\_HEIGHT, IMG\_WIDTH, 3)),

    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),

    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu'),

    MaxPooling2D((2, 2)),

    # added to extract features from the images

    Flatten(),

    Dense(512, activation='relu'),

    # regularization to prevent overfitting

    Dropout(0.5),

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

])

# history\_cnn = model.fit(

#     train\_generator,

#     steps\_per\_epoch=train\_generator.samples // 16,

#     epochs=20,

#     validation\_data=validation\_generator,

#     validation\_steps=validation\_generator.samples // 16,

#     callbacks=[model\_checkpoint]

# )

# Compile the model

model.compile(optimizer='adam',

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

# model is compiled with an optimizer, a loss function, and evaluation metrics -for multi-class classification

# Train the model

model.fit(train\_generator,

          epochs=epochs)

# Save the model

model.save('/content/drive/MyDrive/dataset/extracted1/family\_member\_model.h5')

import tensorflow as tf

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.applications import VGG16

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define constants

IMG\_HEIGHT = 128

IMG\_WIDTH = 128

batch\_size = 15

epochs = 20

num\_classes = 3  # Number of family members

# Load pre-trained VGG16 model (excluding the top layer)

vgg\_base = VGG16(weights='imagenet', include\_top=False, input\_shape=(IMG\_HEIGHT, IMG\_WIDTH, 3))

# Freeze the convolutional base

for layer in vgg\_base.layers:

    layer.trainable = False

# Model Architecture

model = Sequential([

    vgg\_base,

    Flatten(),

    Dense(512, activation='relu'),

    Dropout(0.5),

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

])

# Compile the model

model.compile(optimizer=Adam(),

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

# Data Augmentation and Preprocessing

train\_datagen = ImageDataGenerator(

    rescale=1./255,

    rotation\_range=20,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    shear\_range=0.2,

    zoom\_range=0.2,

    horizontal\_flip=True,

    validation\_split=0.2  # Splitting training data into training and validation

)

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

    '/content/drive/MyDrive/preprocessed\_dataset',

    target\_size=(IMG\_HEIGHT, IMG\_WIDTH),

    batch\_size=batch\_size,

    class\_mode='categorical',

    subset='training'

)

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

    '/content/drive/MyDrive/preprocessed\_dataset',

    target\_size=(IMG\_HEIGHT, IMG\_WIDTH),

    batch\_size=batch\_size,

    class\_mode='categorical',

    subset='validation'

)

# Train the model

model.fit(train\_generator,

          epochs=epochs,

          validation\_data=validation\_generator)

# Save the model

model.save('/content/drive/MyDrive/dataset/extracted1/vgg\_family\_member\_model.h5')

import tensorflow as tf

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.applications import ResNet50

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define constants

IMG\_HEIGHT = 128

IMG\_WIDTH = 128

batch\_size = 32

epochs = 20

num\_classes = 3  # Number of family members

# Load pre-trained ResNet50 model (excluding the top layer)

resnet\_base = ResNet50(weights='imagenet', include\_top=False, input\_shape=(IMG\_HEIGHT, IMG\_WIDTH, 3))

# Freeze the convolutional base

for layer in resnet\_base.layers:

    layer.trainable = False

# Model Architecture

model = Sequential([

    resnet\_base,

    Flatten(),

    Dense(512, activation='relu'),

    Dropout(0.5),

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

])

# Compile the model

model.compile(optimizer=Adam(),

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

# Data Augmentation and Preprocessing

train\_datagen = ImageDataGenerator(

    rescale=1./255,

    rotation\_range=20,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    shear\_range=0.2,

    zoom\_range=0.2,

    horizontal\_flip=True,

    validation\_split=0.2  # Splitting training data into training and validation

)

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

    '/content/drive/MyDrive/preprocessed\_dataset',

    target\_size=(IMG\_HEIGHT, IMG\_WIDTH),

    batch\_size=batch\_size,

    class\_mode='categorical',

    subset='training'

)

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

    '/content/drive/MyDrive/preprocessed\_dataset',

    target\_size=(IMG\_HEIGHT, IMG\_WIDTH),

    batch\_size=batch\_size,

    class\_mode='categorical',

    subset='validation'

)

# Train the model

model.fit(train\_generator,

          epochs=epochs,

          validation\_data=validation\_generator)

# Save the model

model.save('/content/drive/MyDrive/dataset/extracted1/resnet50\_family\_member\_model.h5')

pip install tensorflow

import tensorflow as tf

tf.get\_logger().setLevel('ERROR')

import tensorflow as tf

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing import image

import numpy as np

import matplotlib.pyplot as plt

# Load the pre-trained model

model = load\_model('/content/drive/MyDrive/dataset/extracted1/family\_member\_model.h5')

# Define class indices

class\_indices = {'Gokul': 0, 'Dineshkumar': 1, 'Hemanth': 2}

# Function to preprocess input image

def preprocess\_image(img\_path):

    img = image.load\_img(img\_path, target\_size=(128, 128))

    img\_array = image.img\_to\_array(img)

    img\_array = np.expand\_dims(img\_array, axis=0)

    img\_array /= 255.0  # Normalize the image

    return img, img\_array

# Function to make prediction

def predict\_class(img\_path):

    \_, img\_array = preprocess\_image(img\_path)

    predictions = model.predict(img\_array)

    predicted\_class\_index = np.argmax(predictions)

    predicted\_class = list(class\_indices.keys())[predicted\_class\_index]

    return predicted\_class

# Path to the test image

img\_path = '/content/drive/MyDrive/dataset/testimage/test\_img5.jpg'

predicted\_class = predict\_class(img\_path)

# Load and display the input image with improved clarity

input\_img, \_ = preprocess\_image(img\_path)

plt.figure(figsize=(6, 6), dpi=100)  # Set figure size and DPI for improved clarity

plt.imshow(input\_img)

plt.axis('off')

plt.title(f'Predicted Class: {predicted\_class}')

plt.show()

# Display prediction result

if predicted\_class in ['Dineshkumar', 'Hemanth','Gokul']:

    print(f'Access granted! Welcome Home, {predicted\_class}!')

else:

    print('Access denied. Sorry, you are not recognized as a family member.')

from google.colab import drive

drive.mount('/content/drive')

import cv2

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.models import load\_model

def preprocess\_image(img):

    img = cv2.resize(img, (128, 128))

    img\_array = np.expand\_dims(img, axis=0)

    img\_array = img\_array.astype('float32') / 255.0  # Normalize the image

    return img\_array

def predict\_class(img):

    img\_array = preprocess\_image(img)

    predictions = model.predict(img\_array)

    predicted\_class\_index = np.argmax(predictions)

    predicted\_class = list(class\_indices.keys())[predicted\_class\_index]

    return predicted\_class

def show\_video\_frames(video\_path, total\_frames):

    # Open the video file

    video = cv2.VideoCapture(video\_path)

    # Get total number of frames in the video

    total\_frames\_video = int(video.get(cv2.CAP\_PROP\_FRAME\_COUNT))

    # Calculate frame skip based on the total number of frames

    frame\_skip = max(total\_frames\_video // total\_frames, 1)

    # Initialize variables

    frame\_count = 0

    fps = int(video.get(cv2.CAP\_PROP\_FPS))

    # Read and process frames

    while True:

        # Read a frame from the video

        ret, frame = video.read()

        # Check if frame was successfully read

        if not ret:

            break

        # Display the frame if frame\_count is multiple of frame\_skip

        if frame\_count % frame\_skip == 0:

            # Predict on frame

            predicted\_class = predict\_class(frame)

            # Format output

            output = ""

            if predicted\_class in ['Dineshkumar', 'Hemanth','Gokul']:

                output = f'Access granted! Welcome Home, {predicted\_class}!'

            else:

                output = 'Access denied. Sorry, you are not recognized as a family member.'

            # Display formatted output

            print(output)

            # Display frame with prediction

            plt.imshow(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

            plt.axis('off')

            plt.title(output)

            plt.show()

        # Increment frame count

        frame\_count += 1

    # Release the video object

    video.release()

# Load the pre-trained model

model = load\_model('/content/drive/MyDrive/dataset/extracted1/family\_member\_model.h5')

# Define class indices

class\_indices = {'Hemanth': 0, 'Gokul': 1, 'Dineshkumar': 2}

# Path to the video file in Google Drive

video\_path = '/content/drive/MyDrive/dataset/testimage/test\_vid4.mp4'

# Get total seconds of the video

video = cv2.VideoCapture(video\_path)

total\_seconds = int(video.get(cv2.CAP\_PROP\_FRAME\_COUNT)) // int(video.get(cv2.CAP\_PROP\_FPS))

video.release()

# Show frames equal to the total seconds of the video

show\_video\_frames(video\_path, total\_seconds)

from google.colab import drive

drive.mount('/content/drive')

import cv2

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.models import load\_model

def preprocess\_image(img):

    img = cv2.resize(img, (128, 128))

    img\_array = np.expand\_dims(img, axis=0)

    img\_array = img\_array.astype('float32') / 255.0  # Normalize the image

    return img\_array

def predict\_class(img):

    img\_array = preprocess\_image(img)

    predictions = model.predict(img\_array)

    predicted\_class\_index = np.argmax(predictions)

    predicted\_class = list(class\_indices.keys())[predicted\_class\_index]

    return predicted\_class

def show\_video\_frames(video\_path, total\_frames):

    # Open the video file

    video = cv2.VideoCapture(video\_path)

    # Get total number of frames in the video

    total\_frames\_video = int(video.get(cv2.CAP\_PROP\_FRAME\_COUNT))

    # Calculate frame skip based on the total number of frames

    frame\_skip = max(total\_frames\_video // total\_frames, 1)

    # Initialize variables

    frame\_count = 0

    fps = int(video.get(cv2.CAP\_PROP\_FPS))

    # Read and process frames

    while True:

        # Read a frame from the video

        ret, frame = video.read()

        # Check if frame was successfully read

        if not ret:

            break

        # Display the frame if frame\_count is multiple of frame\_skip

        if frame\_count % frame\_skip == 0:

            # Predict on frame

            predicted\_class = predict\_class(frame)

            # Format output

            output = ""

            if predicted\_class in ['Dineshkumar', 'Hemanth']:

                output = f'Access granted! Welcome Home, {predicted\_class}!'

            else:

                output = 'Access denied. Sorry, you are not recognized as a family member.'

            # Display formatted output

            print(output)

            # Display frame with prediction

            plt.imshow(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

            plt.axis('off')

            plt.title(output)

            plt.show()

        # Increment frame count

        frame\_count += 1

    # Release the video object

    video.release()

# Load the pre-trained model

model = load\_model('/content/drive/MyDrive/dataset/extracted1/family\_member\_model.h5')

# Define class indices

class\_indices = {'Hemanth': 0, 'Gokul': 1, 'Dineshkumar': 2}

# Path to the video file in Google Drive

video\_path = '/content/drive/MyDrive/dataset/testimage/test\_vid3.mp4'

# Get total seconds of the video

video = cv2.VideoCapture(video\_path)

total\_seconds = int(video.get(cv2.CAP\_PROP\_FRAME\_COUNT)) // int(video.get(cv2.CAP\_PROP\_FPS))

video.release()

# Show frames equal to the total seconds of the video

show\_video\_frames(video\_path, total\_seconds)

from google.colab import drive

drive.mount('/content/drive')

import cv2

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.models import load\_model

def preprocess\_image(img):

    img = cv2.resize(img, (128, 128))

    img\_array = np.expand\_dims(img, axis=0)

    img\_array = img\_array.astype('float32') / 255.0  # Normalize the image

    return img\_array

def predict\_class(img):

    img\_array = preprocess\_image(img)

    predictions = model.predict(img\_array)

    predicted\_class\_index = np.argmax(predictions)

    predicted\_class = list(class\_indices.keys())[predicted\_class\_index]

    return predicted\_class

def show\_video\_frames(video\_path, total\_frames):

    # Open the video file

    video = cv2.VideoCapture(video\_path)

    # Get total number of frames in the video

    total\_frames\_video = int(video.get(cv2.CAP\_PROP\_FRAME\_COUNT))

    # Calculate frame skip based on the total number of frames

    frame\_skip = max(total\_frames\_video // total\_frames, 1)

    # Initialize variables

    frame\_count = 0

    fps = int(video.get(cv2.CAP\_PROP\_FPS))

    # Read and process frames

    while True:

        # Read a frame from the video

        ret, frame = video.read()

        # Check if frame was successfully read

        if not ret:

            break

        # Display the frame if frame\_count is multiple of frame\_skip

        if frame\_count % frame\_skip == 0:

            # Predict on frame

            predicted\_class = predict\_class(frame)

            # Format output

            output = ""

            if predicted\_class in ['Dineshkumar', 'Hemanth','Gokul']:

                output = f'Access granted! Welcome Home, {predicted\_class}!'

            else:

                output = 'Access denied. Sorry, you are not recognized as a family member.'

            # Display formatted output

            print(output)

            # Display frame with prediction

            plt.imshow(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

            plt.axis('off')

            plt.title(output)

            plt.show()

        # Increment frame count

        frame\_count += 1

    # Release the video object

    video.release()

# Load the pre-trained model

model = load\_model('/content/drive/MyDrive/dataset/extracted1/family\_member\_model.h5')

# Define class indices

class\_indices = {'Hemanth': 0, 'Gokul': 1, 'Dineshkumar': 2}

# Path to the video file in Google Drive

video\_path = '/content/drive/MyDrive/dataset/testimage/test\_vid5.mp4'

# Get total seconds of the video

video = cv2.VideoCapture(video\_path)

total\_seconds = int(video.get(cv2.CAP\_PROP\_FRAME\_COUNT)) // int(video.get(cv2.CAP\_PROP\_FPS))

video.release()

# Show frames equal to the total seconds of the video

show\_video\_frames(video\_path, total\_seconds)