```python
# -*- coding: utf-8 -*-
"""
Created on Sat Mar  8 12:42:33 2025

@author: CMP
"""

import streamlit as st
import sqlite3  # Import SQLite module
import os
import base64
import subprocess

# Function to convert a file to base64
def get_base64(bin_file):
    with open(bin_file, 'rb') as f:
        data = f.read()
    return base64.b64encode(data).decode()


# Function to set the background of the Streamlit app
def set_background(png_file):
    bin_str = get_base64(png_file)
    page_bg_img = f'''
    <style>
    .stApp {{
        background-image: url("{bin_str}");
        background-position: center;
        background-size: cover;
        font-family: "Times New Roman", serif;
    }}
    h1, h2, h3, p {{
        font-family: "Times New Roman", serif;
    }}
    </style>
    '''
    st.markdown(page_bg_img, unsafe_allow_html=True)


# Set the background image for the app
set_background('Background/1.png')


# Function to initialize the SQLite database
def init_db():
    conn = sqlite3.connect('users.db')  # Connect to the SQLite database
    cursor = conn.cursor()

    # Create the users table if it does not exist
    cursor.execute('''CREATE TABLE IF NOT EXISTS users (
                      username TEXT PRIMARY KEY,
                      password TEXT)''')

    conn.commit()
    conn.close()


# Function to render the home page
def home():
```

```
    """
    Renders the Home page.
    Greets and provides information about the app.
    """
    st.markdown("<h1 style='text-align: center;'>Thyroid Detection And Classification Using Dnn Based On Hybrid Meta-Heuristic And Lstm Technique</h1>", unsafe_allow_html=True)

    # Animation
    animation_path = "background/1.gif"  # Relative path to your GIF
    if os.path.exists(animation_path):  # Ensure the file exists
        animation_base64 = get_base64(animation_path)
        st.markdown(f"""
        <div style="display: flex; justify-content: center; margin-bottom: 20px;">
          <img src="{animation_base64}" alt="Fitness Animation" style="max-width: 400px;"/>
        </div>
        """, unsafe_allow_html=True)
    else:
        st.error(f"Error: Animation file not found at '{animation_path}'")

    st.markdown("<p style='text-align: center;'>Welcome to our Thyroid Detection And Classification Using Dnn Based On Hybrid Meta-Heuristic And Lstm Technique! </p>", unsafe_allow_html=True)


# Function to register a user
def signup():
    """
    Renders the sign-up page for the web application.
    Allows a new user to create an account by providing a username and password.
    Checks for existing username and password confirmation before adding the user.
    """
    st.markdown("<h2 style='text-align: center;'>Create a New Account</h2>", unsafe_allow_html=True)
    st.markdown("<p style='text-align: center;'>Please enter your details to create a new account:</p>", unsafe_allow_html=True)

    col1, col2, col3 = st.columns([1, 2, 1])

    with col2:
        username = st.text_input("Username")
        password = st.text_input("Password", type="password")
        confirm_password = st.text_input("Confirm Password", type="password")

        if st.button("Sign Up"):
            # Initialize the database if not done already
            init_db()

            # Check if username already exists
            conn = sqlite3.connect('users.db')
            cursor = conn.cursor()
            cursor.execute('SELECT * FROM users WHERE username = ?', (username,))
            user = cursor.fetchone()

            if user:
                st.error("Username already exists. Please choose a different username.")
            elif password != confirm_password:
                st.error("Passwords do not match. Please try again.")
            else:
                # Insert new user into the database
                cursor.execute('INSERT INTO users (username, password) VALUES (?, ?)', (username, password))
                conn.commit()
```

```
            st.success("You have successfully signed up!")
            st.info("Please go to the Login page to log in.")

        conn.close()


# Function to log in a user
def login():
    """
    Renders the login page for the web application.
    Allows an existing user to log in by providing their username and password.
    Checks for the existence of the username and matches the password.
    """
    st.markdown("<h2 style='text-align: center;'>Login Section</h2>", unsafe_allow_html=True)
    st.markdown("<p style='text-align: center;'>Please enter your credentials to log in:</p>", unsafe_allow_html=True)

    col1, col2, col3 = st.columns([1, 2, 1])

    with col2:
        username = st.text_input("Username")
        password = st.text_input("Password", type="password")

        if st.button("Login"):
            # Initialize the database if not done already
            init_db()

            # Verify the credentials
            conn = sqlite3.connect('users.db')
            cursor = conn.cursor()
            cursor.execute('SELECT * FROM users WHERE username = ? AND password = ?', (username, password))
            user = cursor.fetchone()

            if user:
                st.success(f"Welcome {username.title()}!")
                st.write("You have successfully logged in.")
                # Assuming app1.py is the app you want to launch after login
                subprocess.run(["streamlit", "run", "app1.py"])
            else:
                st.error("Incorrect username or password. Please try again.")

            conn.close()


# Main function to run the Streamlit app
def main():
    """
    Main function to run the Streamlit app.
    Provides navigation between the sign-up and login pages using a sidebar dropdown menu list.
    """
    # Sidebar navigation
    choice = st.sidebar.radio("Menu", ["Home", "Login", "Sign Up"], index=0)  # Default to 'Home'

    if choice == "Sign Up":
        signup()
    elif choice == "Login":
        login()
    elif choice == "Home":
        home()
```

```
if __name__ == "__main__":
    main()
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
from keras.models import load_model
import streamlit as st
from PIL import Image
import base64
from gtts import gTTS

import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
from keras.models import load_model
import streamlit as st
from PIL import Image
import base64
from gtts import gTTS

def get_base64(bin_file):
    with open(bin_file, 'rb') as f:
        data = f.read()
    return base64.b64encode(data).decode()


# Function to set the background of the Streamlit app
def set_background(png_file):
    bin_str = get_base64(png_file)
    page_bg_img = f'''
    <style>
    .stApp {{
        background-image: url("{bin_str}");
        background-position: center;
        background-size: cover;
        font-family: "Times New Roman", serif;
    }}
    h1, h2, h3, p {{
        font-family: "Times New Roman", serif;
    }}
    </style>
    '''
    st.markdown(page_bg_img, unsafe_allow_html=True)

set_background('Background/1.png')

# Streamlit app title
st.title("Project : Thyroid Detection And Classification Using Dnn Based On Hybrid Meta-Heuristic And Lstm Technique
")

# Load the model
model = load_model("model.h5")

# Define image dimensions and categories
WIDTH, HEIGHT = 65, 65
categories = ['thyroid_cancer','thyroid_ditis','thyroid_hyper','thyroid_hypo','thyroid_nodule','thyroid_normal']
```

```
# Function to load and preprocess the image
def load_and_preprocess_image(image):
    image = np.array(image)

    # Ensure the image has 3 channels (RGB)
    if image.ndim == 2:  # Grayscale image
        image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
    elif image.shape[2] == 4:  # RGBA image
        image = cv2.cvtColor(image, cv2.COLOR_RGBA2RGB)

    test_image = cv2.resize(image, (WIDTH, HEIGHT))
    test_data = np.array(test_image, dtype="float") / 255.0
    test_data = test_data.reshape([-1, WIDTH, HEIGHT, 3])
    return image, test_data

# Function to segment the image using thresholding
def segment_image(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, thresholded = cv2.threshold(gray, 128, 255, cv2.THRESH_BINARY)
    segmented_image = cv2.bitwise_and(image, image, mask=thresholded)
    return segmented_image

# Function for disease recommendation based on prediction
def disease_recommendation(prediction):
    recommendations = {
        'thyroid_cancer': "It seems like you might have thyroid cancer. It's highly recommended to see a doctor for further
diagnosis and treatment.",
        'thyroid_ditis': "You may have thyroiditis. Please consult a doctor for proper treatment.",
        'thyroid_hyper': "It appears that you have hyperthyroidism. It is important to follow up with a healthcare provider for
treatment options.",
        'thyroid_hypo': "Hypothyroidism is detected. Make sure to follow up with a healthcare provider for a treatment
plan.",
        'thyroid_nodule': "A thyroid nodule has been detected. It's recommended to have it checked by a medical
professional.",
        'thyroid_normal': "The thyroid appears normal. Maintain a healthy lifestyle and schedule regular checkups."
    }
    return recommendations.get(prediction, "Please consult a healthcare provider for further evaluation.")

# Streamlit interface
st.title("Image Classification Prediction")
st.write("Upload images to get the predictions.")

uploaded_files = st.file_uploader("Choose images...", type=["jpg", "jpeg", "png"], accept_multiple_files=True)

if uploaded_files:
    predictions_list = []  # Initialize an empty list to store predictions
    for uploaded_file in uploaded_files:
        if uploaded_file is not None:
            # Load image with PIL
            image = Image.open(uploaded_file)

            # Display the uploaded image
            st.image(image, caption="Uploaded Image", use_column_width=True)

            # Preprocess the image
            test_image_o, test_data = load_and_preprocess_image(image)

            # Make prediction
```

```
            pred = model.predict(test_data)
            predictions = np.argmax(pred, axis=1)  # return to label

            # Append prediction to the list
            prediction = categories[predictions[0]]
            predictions_list.append(prediction)

            # Display the prediction
            st.write(f'Prediction: {prediction}')

            # Provide disease-specific recommendation
            recommendation = disease_recommendation(prediction)
            st.write(f"Recommendation: {recommendation}")

            # Display the image with the prediction title
            fig = plt.figure()
            fig.patch.set_facecolor('xkcd:white')
            plt.title(prediction)
            plt.imshow(cv2.cvtColor(test_image_o, cv2.COLOR_BGR2RGB))
            plt.axis('off')  # Hide axes
            st.pyplot(fig)

            # Segment the image
            segmented_image = segment_image(test_image_o)

            # Display the segmented image
            fig = plt.figure()
            fig.patch.set_facecolor('xkcd:white')
            plt.title('Segmented Image')
            plt.imshow(cv2.cvtColor(segmented_image, cv2.COLOR_BGR2RGB))
            plt.axis('off')  # Hide axes
            st.pyplot(fig)

    # Display all predictions
    st.write("All Predictions:")
    st.write(predictions_list)

    # Convert predictions list to string and create audio
    predictions_text = ' '.join(predictions_list)
    language = 'en'
    speech = gTTS(text=predictions_text, lang=language, slow=False)
    speech.save("sample.mp3")
    audio_path = "sample.mp3"  # Replace with the path to your MP3 audio file

    st.audio(audio_path, format='audio/mp3')
else:
    st.write("Please upload image files.")

import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras import Input
from tensorflow.keras.layers import Dense, Dropout, Conv2D, BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.utils import to_categorical
```

```
from numpy import argmax
from sklearn.metrics import confusion_matrix, classification_report
from mlxtend.plotting import plot_confusion_matrix
import easygui

#==============================Input Data=====================================
path = 'Dataset/'

# categories
categories = ['thyroid_cancer','thyroid_ditis','thyroid_hyper','thyroid_hypo','thyroid_nodule','thyroid_normal']

# let's display some of the pictures
for category in categories:
    fig, _ = plt.subplots(2,2)
    fig.suptitle(category)
    fig.patch.set_facecolor('xkcd:white')
    for k, v in enumerate(os.listdir(path+category)[:4]):
        img = plt.imread(path+category+'/'+v)
        plt.subplot(2, 2, k+1)
        plt.axis('off')
        plt.imshow(img)
    #     cv2.imshow(' Image',img)
    #     cv2.waitKey(0)
    #     cv2.destroyAllWindows()
    # plt.show()

shape0 = []
shape1 = []

for category in categories:
    for files in os.listdir(path+category):
        shape0.append(plt.imread(path+category+'/'+ files).shape[0])
        shape1.append(plt.imread(path+category+'/'+ files).shape[1])
    print(category, ' => height min : ', min(shape0), 'width min : ', min(shape1))
    print(category, ' => height max : ', max(shape0), 'width max : ', max(shape1))
    shape0 = []
    shape1 = []

# initialize the data and labels
data = []
labels = []
imagePaths = []
HEIGHT = 65
WIDTH = 65
N_CHANNELS = 3

# grab the image paths and randomly shuffle them
for k, category in enumerate(categories):
    for f in os.listdir(path+category):
        imagePaths.append([path+category+'/'+f, k])

import random
random.shuffle(imagePaths)
print(imagePaths[:10])

# loop over the input images
for imagePath in imagePaths:
# load the image, resize the image to be HEIGHT * WIDTH pixels (ignoring aspect ratio) and store the image in the data
list
```

```
    image = cv2.imread(imagePath[0])
    image = cv2.resize(image, (WIDTH, HEIGHT))  # .flatten()
    data.append(image)

    # extract the class label from the image path and update the
    # labels list
    label = imagePath[1]
    labels.append(label)

# scale the raw pixel intensities to the range [0, 1]
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)

# Let's check everything is ok
fig, _ = plt.subplots(2,2)
fig.suptitle("Sample Input")
fig.patch.set_facecolor('xkcd:white')
for i in range(4):
    plt.subplot(2,2, i+1)
    plt.imshow(data[i])
    plt.axis('off')
#     cv2.imshow(' Image',data[1])
#     cv2.waitKey(0)
#     cv2.destroyAllWindows()
# #    plt.title(categories[labels[i]])
# plt.show()



# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.2, random_state=42)
# Preprocess class labels
trainY =to_categorical(trainY, 6)

print(trainX.shape)
print(testX.shape)
print(trainY.shape)
print(testY.shape)

#================================Classification================================
"'DENSENET121'"

def build_densenet():
    densenet = DenseNet121(weights='imagenet', include_top=False)

    input = Input(shape=(HEIGHT, WIDTH, N_CHANNELS))
    x = Conv2D(3, (3, 3), padding='same')(input)

    x = densenet(x)

    x = GlobalAveragePooling2D()(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
    x = Dense(256, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)

    # multi output
```

```
    output = Dense(6,activation = 'softmax', name='root')(x)


    # model
    model = Model(input,output)

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

    return model

model = build_densenet()

model.fit(trainX, trainY, batch_size=32, epochs=5, verbose=1)
import tensorflow as tf

tf.keras.models.save_model(model,'model.h5')

history=model.history.history
#Plotting the accuracy
train_loss = history['loss']
train_acc = history['accuracy']

    # Performance graph
plt.figure()
plt.plot(train_loss, label='Loss')
plt.plot(train_acc, label='Accuracy')
plt.title('Performance Plot')
plt.legend()
plt.show()

print("Accuracy of the CNN is:",model.evaluate(trainX,trainY)[1]*100, "%")
#============================Analytic Results================================
pred = model.predict(testX)
predictions = argmax(pred, axis=1)
print('Classification Report')
cr=classification_report(testY, predictions,target_names=categories)
print(cr)
print('Confusion Matrix')
cm = confusion_matrix(testY, predictions)
print(cm)
#Confusion Matrix Plot
plt.figure()
plot_confusion_matrix(cm,figsize=(15,15), class_names = categories,
                      show_normed = True);

plt.title( "Model confusion matrix")
plt.style.use("ggplot")
plt.show()
import tensorflow as tf
model = tf.keras.models.load_model('model.h5')
#================================Prediction=====================================
from tkinter import filedialog
test_data=[]
Image = filedialog.askopenfilename()
head_tail = os.path.split(Image)
fileNo=head_tail[1].split('.')
test_image_o = cv2.imread(Image)
test_image = cv2.resize(test_image_o, (WIDTH, HEIGHT))
```

```
#test_data.append(test_image)
# scale the raw pixel intensities to the range [0, 1]
test_data = np.array(test_image, dtype="float") / 255.0
test_data=test_data.reshape([-1,65, 65, 3])
pred = model.predict(test_data)
predictions = argmax(pred, axis=1) # return to label
print ('Prediction : '+categories[predictions[0]])
#Imersing into the plot
fig = plt.figure()
fig.patch.set_facecolor('xkcd:white')
plt.title(categories[predictions[0]])
plt.imshow(test_image_o)
#================================================================================



# -*- coding: utf-8 -*-
"""
Created on Sat Mar  8 12:37:05 2025

@author: CMP
"""

import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras import Input
from tensorflow.keras.layers import Dense, Dropout, Conv2D, BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import DenseNet121, VGG16, VGG19, MobileNetV3
from tensorflow.keras.utils import to_categorical
from numpy import argmax
from sklearn.metrics import confusion_matrix, classification_report
from mlxtend.plotting import plot_confusion_matrix
import easygui

#==============================Input Data=====================================
path = 'Dataset/'

# categories
categories = ['thyroid_cancer','thyroid_ditis','thyroid_hyper','thyroid_hypo','thyroid_nodule','thyroid_normal']

# Segmentation: Optimized Otsu's Method and Edge Detection
def optimized_otsu(image):
    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)
    _, thresh_image = cv2.threshold(blurred_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    return thresh_image

# Display sample images from each category
for category in categories:
    fig, _ = plt.subplots(2, 2)
    fig.suptitle(category)
    fig.patch.set_facecolor('xkcd:white')
    for k, v in enumerate(os.listdir(path+category)[:4]):
        img = plt.imread(path+category+'/'+v)
        plt.subplot(2, 2, k+1)
```

```
        plt.axis('off')
        plt.imshow(img)

#==============================Image Preprocessing====================================
shape0 = []
shape1 = []

for category in categories:
    for files in os.listdir(path+category):
        shape0.append(plt.imread(path+category+'/'+ files).shape[0])
        shape1.append(plt.imread(path+category+'/'+ files).shape[1])

# initialize the data and labels
data = []
labels = []
imagePaths = []
HEIGHT = 65
WIDTH = 65
N_CHANNELS = 3

# grab the image paths and randomly shuffle them
for k, category in enumerate(categories):
    for f in os.listdir(path+category):
        imagePaths.append([path+category+'/'+f, k])

import random
random.shuffle(imagePaths)

# loop over the input images
for imagePath in imagePaths:
    image = cv2.imread(imagePath[0])
    image = cv2.resize(image, (WIDTH, HEIGHT))
    data.append(image)

    label = imagePath[1]
    labels.append(label)

# scale the raw pixel intensities to the range [0, 1]
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)

# Partition the data into training and testing splits using 80% for training and 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.2, random_state=42)
trainY = to_categorical(trainY, 6)

#================================= Classification: Model Definition
==================================

def build_densenet():
    densenet = DenseNet121(weights='imagenet', include_top=False)
    input = Input(shape=(HEIGHT, WIDTH, N_CHANNELS))
    x = Conv2D(3, (3, 3), padding='same')(input)
    x = densenet(x)
    x = GlobalAveragePooling2D()(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
    x = Dense(256, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
```

```
    output = Dense(6, activation='softmax', name='root')(x)
    model = Model(input, output)
    model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])
    model.summary()
    return model

def build_vgg16():
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=(HEIGHT, WIDTH, N_CHANNELS))
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.5)(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(6, activation='softmax', name='root')(x)
    model = Model(inputs=base_model.input, outputs=output)
    model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])
    model.summary()
    return model

def build_vgg19():
    base_model = VGG19(weights='imagenet', include_top=False, input_shape=(HEIGHT, WIDTH, N_CHANNELS))
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.5)(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(6, activation='softmax', name='root')(x)
    model = Model(inputs=base_model.input, outputs=output)
    model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])
    model.summary()
    return model

def build_hybrid_densenet_mobilenet():
    base_model_1 = DenseNet121(weights='imagenet', include_top=False, input_shape=(HEIGHT, WIDTH,
N_CHANNELS))
    base_model_2 = MobileNetV3(weights='imagenet', include_top=False, input_shape=(HEIGHT, WIDTH,
N_CHANNELS))

    x1 = base_model_1.output
    x1 = GlobalAveragePooling2D()(x1)

    x2 = base_model_2.output
    x2 = GlobalAveragePooling2D()(x2)

    x = np.concatenate([x1, x2], axis=-1)
    x = Dropout(0.5)(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(6, activation='softmax', name='root')(x)

    model = Model(inputs=[base_model_1.input, base_model_2.input], outputs=output)
    model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])
    model.summary()
    return model

#================================ Model Training ==================================
model = build_densenet()  # You can switch to `build_vgg16()` or `build_hybrid_densenet_mobilenet()`

history = model.fit(trainX, trainY, batch_size=32, epochs=5, verbose=1)
```

```
# Save the model
import tensorflow as tf
tf.keras.models.save_model(model, 'model.h5')

# Performance Graphs
train_loss = history.history['loss']
train_acc = history.history['accuracy']
plt.figure()
plt.plot(train_loss, label='Loss')
plt.plot(train_acc, label='Accuracy')
plt.title('Performance Plot')
plt.legend()
plt.show()

print("Accuracy of the CNN is:", model.evaluate(trainX, trainY)[1] * 100, "%")

#================================ Analytic Results ==================================
# Test set prediction and evaluation
pred = model.predict(testX)
predictions = argmax(pred, axis=1)

print('Classification Report')
cr = classification_report(testY, predictions, target_names=categories)
print(cr)

print('Confusion Matrix')
cm = confusion_matrix(testY, predictions)
print(cm)

# Confusion Matrix Plot
plt.figure()
plot_confusion_matrix(cm, figsize=(15, 15), class_names=categories, show_normed=True)
plt.title("Model confusion matrix")
plt.style.use("ggplot")
plt.show()

#================================ Prediction on New Image ================================
# from tkinter import filedialog

# # Load a new image for prediction
# test_data = []
# Image = filedialog.askopenfilename()
# head_tail = os.path.split(Image)
# fileNo = head_tail[1].split('.')
# test_image_o = cv2.imread(Image)
# test_image = cv2.resize(test_image_o, (WIDTH, HEIGHT))

# # Normalize and prepare for prediction
# test_image_normalized = test_image / 255.0
# test_data = np.expand_dims(test_image_normalized, axis=0)

# # Predict
# pred = model.predict(test_data)
# predictions = argmax(pred, axis=1)
# print('Prediction: ' + categories[predictions[0]])

# # Display the image with prediction label
# fig = plt.figure()
```

```
# fig.patch.set_facecolor('xkcd:white')
# plt.title(categories[predictions[0]])
# plt.imshow(test_image_o)
# plt.show()
```