**HTML**

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="x-ua-compatible" content="ie=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>TumorTracker - Home</title>

<link href="{{url\_for('static', filename='css/style.css')}}" rel="stylesheet" type="text/css">

<link href="{{url\_for('static', filename='img/logos/tumorTracker fave logo.png')}}" rel="icon">

</head>

<body>

*<!-- Nav Bar -->*

    <nav class="nav-bar">

        <a href="#">

            <img src="{{url\_for('static', filename='img/logos/tumorTracker logo.png')}}" class="nav-logo" alt="TumorTracker Logo">

        </a>

        <ul class="nav-options">

            <li>

                <a href="#" class="nav-links">About</a>

            </li>

        </ul>

    </nav>

*<!-- End of Nav Bar -->*

*<!-- Hero Section -->*

    <section class="hero">

        <div class="content-card">

            <div class="content-card-main">

                <div class="card-content-left">

                    <img src="{{url\_for('static', filename='img/vector img/undraw\_medical\_research\_qg4d.png')}}" class="hero-vector" alt="Meical Care Vector">

                </div>

                <div class="card-content-right">

                    <h3>Get Started</h3>

                    <h4>Let's diagnose you</h4>

*<!-- Form -->*

                    <form class="card-form" method="post" enctype="multipart/form-data">

                        <p>Below, select a clear image of the skin lesion you want to analyze and then click the "Upload" button.</p>

                        <input class="select-image" id="select-image" type="file" name="file" accept="image/\*">

                        <input type="submit" class="submit-btn" value="Upload">

                    </form>

*<!-- End of Form -->*

                </div>

            </div>

            <div id="result"></div>

            <div class="copyrights">

                <p>By using our software, you are agreeing to our <a href="#">Terms and Conditions</a></p>

                <div class="copyrighted">

                    &copy;Copyright 2024. All Rights Reserved. <br>

                    <p>Powered by <strong>Binary Solutions</strong>.</p>

                </div>

            </div>

        </div>

    </section>

*<!-- End of Hero -->*

    <srcipt src="{{url\_for('static', filename='js/script.js')}}"></script>

</body>

**END OF HTML**

**PYTHON**

**Train\_model.py:**

import os

os.environ['TF\_ENABLE\_ONEDNN\_OPTS'] = '0' *#for floating point representation issue with pc*

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.callbacks import EarlyStopping

*# Definition of model*

def create\_model():

    model = Sequential([

        Conv2D(16, (3, 3), *activation*='relu', *input\_shape*=(224, 224, 3)),

        MaxPooling2D(2, 2),

        Conv2D(32, (3, 3), *activation*='relu'),

        MaxPooling2D(2, 2),

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

        MaxPooling2D(2, 2),

        Flatten(),

        Dense(64, *activation*='relu'),

        Dropout(0.5),

        Dense(1, *activation*='sigmoid')

    ])

    model.compile(*optimizer*='adam',

*loss*='binary\_crossentropy',

*metrics*=['accuracy'])

    return model

*#Dataset paths*

BATCH\_SIZE=32

DATASET\_PATH = 'data/melanoma'

TRAIN\_DIR=os.path.join(DATASET\_PATH, 'train')

TEST\_DIR=os.path.join(DATASET\_PATH, 'test')

*# Data augmentation*

train\_datagen=ImageDataGenerator(

*rescale*=1./255,

*rotation\_range*=20,

*width\_shift\_range*=0.2,

*height\_shift\_range*=0.2,

*zoom\_range*=0.2,

*horizontal\_flip*=True,

*vertical\_fip*=True,

*fill\_mode*='nearest'

)

test\_datagen=ImageDataGenerator(*rescale*=1./255)

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

    TRAIN\_DIR,

*target\_size*=(224, 224),

*batch\_size*=BATCH\_SIZE,

*class\_mode*='binary'

)

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

    TEST\_DIR,

*target\_size*=(224, 224),

*batch\_size*=BATCH\_SIZE,

*class\_mode*='binary'

)

*# Calculate steps*

TOTAL\_TRAIN\_IMAGES=1000 *#500\*2 classes*

TOTAL\_TEST\_IMAGES=1000 *#500\*2 classes*

steps\_per\_epoch = TOTAL\_TRAIN\_IMAGES // BATCH\_SIZE

validation\_steps = TOTAL\_TEST\_IMAGES // BATCH\_SIZE

*# Print dataset information*

print(f"Total training images: {TOTAL\_TRAIN\_IMAGES}")

print(f"Total test images: {TOTAL\_TEST\_IMAGES}")

print(f"Batch size: {BATCH\_SIZE}")

print(f"Steps per epoch: {steps\_per\_epoch}")

print(f"Validation steps: {validation\_steps}")

print(train\_generator.class\_indices)

*#Create and train model*

model=create\_model()

*# history=model.fit(*

*#     train\_generator,*

*#     steps\_per\_epoch=steps\_per\_epoch,*

*#     epochs=10,*

*#     validation\_data=test\_generator,*

*#     validation\_steps=validation\_steps*

*# )*

early\_stopping = EarlyStopping(

*monitor*='val\_loss',

*patience*=3,

*restore\_best\_weights*=True

)

*#Training model to explicitly capture history*

try:

    history=model.fit(

        train\_generator,

*steps\_per\_epoch*=steps\_per\_epoch,

*epochs*=30,

*validation\_data*=test\_generator,

*validation\_steps*=validation\_steps,

*verbose*=1,

*callbacks*=[early\_stopping]

    )

*# Saving of model and weights seperately for bwetter compatibility*

    model.save('models/melanoma\_model.h5', *save\_fomrat*='h5')

    model.save\_weights('models/melanoma\_model.h5')

*# Save model architecture as JSON*

    model\_json=model.to\_json()

    with open("models/model\_architecture.json", "w") as json\_file:

        json\_file.write(model\_json)

    print("\nTraining completed and model saved!")

*# Print final mertics with error handling*

    try:

        final\_accuracy = history.history['accuracy'][-1]

        final\_val\_accuracy = history.history['val\_accuracy'][-1]

        final\_loss = history.histor['loss'][-1]

        final\_val\_loss = model.history.history['val\_loss'][-1]

        print(f"\nFinal training accuracy: {final\_accuracy:.4f}")

        print(f"\nFinal vaidation accuracy: {final\_val\_accuracy:.4f}")

        print(f"\nFinal training loss: {final\_loss:.4f}")

        print(f"\nFinal vaidation loss: {final\_val\_loss:.4f}")

    except Exception as e:

        print("\nCould not print final metrics, but model training completed.")

        evaluation = model.evaluate(test\_generator, *steps*=validation\_steps)

        print(f"\nFinal evaluation metrics:")

        print(f"\Loss: {evaluation[0]:.4f}")

        print(f"\Accuracy: {evaluation[1]:.4f}")

except Exception as e:

    print(f"An error has occured: {str(e)}")

**app.py:**

import os

os.environ['TF\_ENABLE\_ONEDNN\_OPTS'] = '0' *#for floating point representation issue with pc*

import tensorflow as tf

from tensorflow.keras.models import load\_model

import tensorflow.keras.layers as layers

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

from flask import Flask, request, render\_template

import numpy as np

from PIL import Image

import io

*# Create a Flask instance*

app=Flask(\_\_name\_\_)

*# Create the exact same model architecture as in train\_model*

def create\_model():

    model = tf.keras.Sequential([

        Conv2D(16, (3, 3), *activation*='relu', *input\_shape*=(224, 224, 3)),

        MaxPooling2D(2, 2),

        Conv2D(32, (3, 3), *activation*='relu'),

        MaxPooling2D(2, 2),

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

        MaxPooling2D(2, 2),

        Flatten(),

        Dense(64, *activation*='relu'),

        Dropout(0.5),

        Dense(1, *activation*='sigmoid')

    ])

    model.compile(*optimizer*='adam',

*loss*='binary\_crossentropy',

*metrics*=['accuracy'])

    return model

*# Create model and load weights*

try:

    model=create\_model()

    model.load\_weights('models/melanoma\_model.h5')

    print("Model loaded successfully!")

except Exception as e:

    print(f"Error loading model:{e}")

*# Add Flask Routes*

@app.route('/', *methods*=['GET'])

def home():

    return render\_template('index.html')

melanoma\_info = {

    'positive':{

        'description': 'Potential melanoma detected',

        'recommendations':[

            'Consult a dematologist immediately',

            'Take photos of the lesion for documentatio npurposes',

            'Do not wait or delay seeking professional medical attention',

            'Prepare a list of any symptoms or changes you\'ve noticed'

        ]

    },

    'negative':{

        'description': 'No melanoma indicators detected',

        'recommendations':[

            'Continue regular skin self-examinations',

            'Use sun protection',

            'Monitor any changes in your skin',

            'Schedule routine skin check-ups with your doctor'

        ]

    },

}

@app.route('/', *methods*=['POST'])

def predict():

    try:

*# Get the image from the POST request*

        file=request.files['file']

*# Convert to PIL image*

        img=Image.open(io.BytesIO(file.read()))

*# Resize to match model input size*

        img=img.resize((224,224))

*# Convert to array and preprocess*

        img\_array=tf.keras.preprocessing.image.img\_to\_array(img)

*# Normalize*

        img\_array=img\_array/255.0

*# Add batch dimensions*

        img\_array=tf.expland\_dims(img\_array, 0)

*# Make Prediction*

        prediction=model.predict(img\_array)

        result='positive' if prediction[0][0]>0.5 else 'negative'

        confidence=float(prediction[0][0]) if result=='positive' else float(1-prediction[0][0])

        return render\_template('index.html',

*prediction*=result,

*confidence*=f"{confidence\*100:.2f}%",

*info*=melanoma\_info[result])

    except Exception as e:

        return render\_template('index.html', *error*=str(e))

if \_\_name\_\_=='\_\_main\_\_':

    app.run(*debug*=True)

**END OF PYTHON**

**JAVASCRIPT**

*// image upload*

document.getElementById('select-image').addEventListener('change', async(*e*)=>{

    const file = *e*.target.files[0];

    const formData = new FormData();

    formData.append('image', file);

    try{

        const response = await fetch('/predict', {

            method:'POST',

            body:formData

        });

        const result = await response.json();

        if (result.error){

            displayError(result.error);

        }

        else{

            displayResult(result);

        }

    }

    catch (error){

        displayError(error.message);

    }

});

function displayResult(*result*){

    const resultDiv = document.getElementById('result');

    resultDiv.innerHTML=`

        <div id="result ${*result*.result}">

            <h3>Detection Result:</div>

            <div class="confidence">

                Confidence: ${*result*.confidence}

            </div>

            <p>${*result*.info.description}</p>

            <div class="recommendations">

                <h4>Recommenndations:</h4>

                <ul>

                    ${*result*.info,recommendations.map(*rec*=>`<li>${*rec*}</li>`).join('')}

                </ul>

            </div>

        </div>

    `;

}

function displayError(*error*){

    const resultDiv=document.getElementById('result');

    resultDiv.innerHTML=`<div class="error">Error: ${*error*}</div>`;

}

**END OF JAVASCRIPT**