Diagnosis of lungs condition

The goal of this project was to diagnose patients' lung conditions, specifically identifying wheezes and crackles, using a machine learning algorithm.

To give you some context, we were provided with two main types of data: audio files that contained recordings of patients' respiration cycles, and text files that included timestamps indicating when each respiration cycle started and ended. The text files also contained labels that told us whether the patient was experiencing wheezes or crackles.

The first step in the project was to convert the audio data into a numerical format for analysis. For this, I used the **librosa** library, which is excellent for audio processing.

Once I had the numerical data, the next task was to fragment the audio file according to the respiration cycle timestamps. This involved partitioning the audio array based on the starting and ending points provided in the text files.

One challenge I faced was that different patients had varying lengths of respiration cycles. To address this, I utilized **Mel-frequency cepstral coefficients** (MFCCs). This technique allowed me to extract relevant features from the audio data while standardizing the length of the respiration cycles, so they were consistent across the dataset.

After obtaining the features, I converted this data into equal-sized images using **OpenCV**. I labeled these images to indicate whether they represented the presence or absence of wheezes and crackles. This labeling was crucial for training our machine learning model.

Next, I trained a **Convolutional Neural Network (CNN)** on these images, using the labels as target variables. After training and testing the model, I was pleased to achieve approximately **80% accuracy**. This result indicated a promising level of performance for diagnosing lung conditions using this approach.

In conclusion, this project highlighted the potential of machine learning in medical diagnostics, and I believe there's a lot of scope for future work, such as expanding the dataset or enhancing the model's accuracy through various techniques.

```python
# Step 1: Setup - Connect to Google Drive and Install Necessary Libraries
from google.colab import drive
drive.mount('/content/drive')  # Access Google Drive content in Colab

# Install libraries for audio and image processing
```

```
!pip install librosa Pillow opencv-python

# Step 2: Import Required Libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import librosa
import librosa.display
import cv2
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten,
Dropout, LeakyReLU, Activation
from tensorflow.keras.callbacks import ModelCheckpoint
from datetime import datetime

# Enable inline plotting for Colab
%matplotlib inline

# Step 3: Define Directories
directory = '/content/drive/MyDrive/mosaicps1/datafile'  # Define path to
audio and data files

# Step 4: Load Filenames in Directory
# Get and sort all filenames for use in processing
filename_array = []
for filename in os.scandir(directory):
   if filename.is_file():
       print(filename.path)
       filename_array.append(filename.path)
filename_array.sort()  # Ensure files are sorted by name

# Step 5: Additional Feature Extraction for MFCC
def func(extra_feature_data, mfc):
   # Adjust MFCC based on additional features extracted from filename
   mfc += ('Tc' == extra_feature_data[2]) * 10
   mfc += ('Al' == extra_feature_data[2]) * 20
   mfc += ('Ar' == extra_feature_data[2]) * 30
   mfc += ('Pl' == extra_feature_data[2]) * 40
   mfc += ('Pr' == extra_feature_data[2]) * 50
   mfc += ('Ll' == extra_feature_data[2]) * 60
   return mfc

# Step 6: Extract Features from Audio Files and Save Spectrograms
extracted_features = []  # Store features extracted from audio segments
for i in range(0, len(filename_array), 2):
   # Extract additional features from filename if necessary
```

```
    extra_features_data = filename_array[i + 1].split('_')
    # Load CSV data for timestamps and classifications
    df = pd.read_csv(filename_array[i], delimiter="\t", header=None)
    # Load the audio data with librosa
    data, sample_rate = librosa.load(filename_array[i + 1])

    # Extract segments based on CSV file timestamps
    for j in range(df.shape[0]):
        voice_segment = data[int((df.iloc[j, 0] * sample_rate) /
20):int((df.iloc[j, 1] * sample_rate) / 20)]
        mfccs = librosa.feature.mfcc(y=voice_segment, sr=sample_rate,
n_mfcc=50)
        mfccs = librosa.amplitude_to_db(mfccs)

        # Plot and save spectrogram as an image file
        plt.figure(figsize=(5, 5))
        librosa.display.specshow(mfccs, sr=sample_rate, x_axis='time',
y_axis='hz')
        plt.colorbar()
        plt.tight_layout()
        plt.close()  # Close plot after saving to save memory

        # Generate filename for saving the image
        image_filename =
f"/content/drive/MyDrive/mosaicps1/images/{filename_array[i +
1].split('datafile/')[1]}_{j}1.png"
        plt.savefig(image_filename, dpi=180)

        # Store filename and features for future use
        extracted_features.append([image_filename, df.iloc[j, 2], df.iloc[j,
3]])

# Step 7: Convert Extracted Features into DataFrame for Model Input
extracted_features_df = pd.DataFrame(extracted_features, columns=['feature',
'crackles', 'wheezes'])
features = np.array(extracted_features_df['feature'].tolist())
crackles = np.array(extracted_features_df['crackles'].tolist())
wheezes = np.array(extracted_features_df['wheezes'].tolist())

# Step 8: Data Preprocessing Function
def data_preprocessing(data_path, width=180, height=180):
    data = []
    for image in data_path:
        img = cv2.imread(image, 1).astype(np.float64)
        resized_img = cv2.resize(img, (width, height)).reshape(width * height
* 3)
        extra_features_data = image.split('_')
        processed_img = func(extra_features_data, resized_img) / 316.0  #
```

```
Normalize
       data.append(processed_img)
   return np.array(data)

# Preprocess feature data for model training
features, crackles = data_preprocessing(features), np.array(crackles)

# Step 9: Split Data for Training and Testing
from sklearn.model_selection import train_test_split
features_train1, features_test1, crackles_train, crackles_test =
train_test_split(features, crackles, test_size=0.2, random_state=42,
shuffle=True)
features_train2, features_test2, wheezes_train, wheezes_test =
train_test_split(features, wheezes, test_size=0.2, random_state=42,
shuffle=True)

# Reshape data to match model input
features_train1 = features_train1.reshape(features_train1.shape[0], 180,
180, 3)
features_test1 = features_test1.reshape(features_test1.shape[0], 180, 180,
3)
features_train2 = features_train2.reshape(features_train2.shape[0], 180,
180, 3)
features_test2 = features_test2.reshape(features_test2.shape[0], 180, 180,
3)

# Step 10: Define and Compile Models
def create_model():
   model = Sequential([
       Conv2D(16, (3, 3), strides=(2, 2), padding='SAME', input_shape=(180,
180, 3)),
       LeakyReLU(alpha=0.1),
       MaxPool2D(padding='same'),
       Conv2D(16, (3, 3), padding='SAME'),
       Conv2D(16, (3, 3), padding='SAME'),
       LeakyReLU(alpha=0.1),
       MaxPool2D(padding='same'),
       Flatten(),
       Dense(4096, activation='relu'),
       Dense(500, activation='relu'),
       Dense(100, activation='relu'),
       Dense(1, activation='sigmoid')
   ])
   model.compile(loss='BinaryCrossentropy', metrics=['accuracy'],
optimizer='adam')
   return model

# Create and train models
```

```
model1 = create_model()
model2 = create_model()

# Step 11: Train Model 1 for Crackles
checkpointer1 =
ModelCheckpoint(filepath='/content/drive/MyDrive/mosaicps1/saved_models/audi
o_classification1.keras', save_best_only=True, verbose=1)
model1.fit(features_train1, crackles_train, batch_size=32, epochs=30,
validation_data=(features_test1, crackles_test), callbacks=[checkpointer1],
verbose=1)

# Evaluate model
test_accuracy1 = model1.evaluate(features_test1, crackles_test, verbose=0)
print(f"Crackles Model Test Accuracy: {test_accuracy1[1]}")

# Step 12: Train Model 2 for Wheezes
checkpointer2 =
ModelCheckpoint(filepath='/content/drive/MyDrive/mosaicps1/saved_models/audi
o_classification2.keras', save_best_only=True, verbose=1)
model2.fit(features_train2, wheezes_train, batch_size=32, epochs=30,
validation_data=(features_test2, wheezes_test), callbacks=[checkpointer2],
verbose=1)

# Evaluate model
test_accuracy2 = model2.evaluate(features_test2, wheezes_test, verbose=0)
print(f"Wheezes Model Test Accuracy: {test_accuracy2[1]}")


##################################

Testing

# Import necessary libraries
from google.colab import drive
import os
import pandas as pd
import numpy as np
import librosa
import librosa.display
import matplotlib.pyplot as plt
import cv2
from keras.models import load_model

# Mount Google Drive
def mount_drive():
   drive.mount('/content/drive')

# Load pre-trained models
```

```python
def load_models(model_paths):
   return [load_model(path) for path in model_paths]

# Function to modify features based on extra feature data
def modify_features(extra_feature_data, mfc):
   feature_map = {
       'Tc': 10, 'Al': 20, 'Ar': 30,
       'Pl': 40, 'Pr': 50, 'Ll': 60
   }
   mfc += feature_map.get(extra_feature_data[2], 0)
   return mfc

# Data preprocessing function for images
def data_preprocessing(data_path, width=180, height=180):
   data = []

   for image in data_path:
       img = cv2.imread(image, 1)
       extra_features_data = image.split('_')

       img = img.astype(np.float64)
       normalized_img = img
       new_img = cv2.resize(normalized_img, (width, height))
       new_img = new_img.reshape(width * height * 3)
       new_img = modify_features(extra_features_data, new_img)
       new_img /= 316.0
       data.append(new_img)

   return np.array(data)

# Function to collect test filenames
def collect_test_filenames(directory):
   return sorted([file.path for file in os.scandir(directory) if
file.is_file()])

# Function to extract features and generate spectrograms
def extract_features(test_filename_arr, output_directory):
   extracted_features = []

   for i in range(0, len(test_filename_arr), 2):
       df = pd.read_csv(test_filename_arr[i], delimiter="\t", header=None)
       data, sample_rate = librosa.load(test_filename_arr[i + 1])

       for j in range(df.shape[0]):
           start_idx = int((df[0][j] * 441000) / 20)
           end_idx = int((df[1][j] * 441000) / 20)
           test_voice = data[start_idx:end_idx]
```

```python
            # Extract MFCCs and convert to dB
            mfccs = librosa.feature.mfcc(y=test_voice, sr=sample_rate,
n_mfcc=50)
            mfccs_db = librosa.amplitude_to_db(mfccs)

            # Display and save the spectrogram
            plt.figure(figsize=(1, 1))
            librosa.display.specshow(mfccs_db, sr=sample_rate, x_axis='time',
y_axis='hz')
            test_filename1 =
f"{output_directory}{os.path.basename(test_filename_arr[i + 1])}_{j}_1.png"
            plt.savefig(test_filename1, dpi=180)
            plt.close()

            # Append extracted features
            extracted_features.append([test_filename1, df[2][j], df[3][j]])

    return pd.DataFrame(extracted_features, columns=['feature', 'crackles',
'wheezes'])

# Function to predict and calculate accuracy
def predict_and_evaluate(models, X, y_crackles, y_wheezes):
    correct, total = 0, 0

    for j, train in enumerate(X):
        train = np.expand_dims(train, axis=0)

        # Predict for crackles and wheezes
        predicted_label1 = models[0].predict(train)
        predicted_label2 = models[1].predict(train)

        total += 2
        if np.round(predicted_label1[0][0]) == y_crackles.iloc[j]:
            correct += 1
        if np.round(predicted_label2[0][0]) == y_wheezes.iloc[j]:
            correct += 1

        # Display predictions
        print(f'Predicted - Crackles: {np.round(predicted_label1[0][0])},
Wheezes: {np.round(predicted_label2[0][0])}')
        print(f'Reality - Crackles: {y_crackles.iloc[j]}, Wheezes:
{y_wheezes.iloc[j]}')

    # Final accuracy calculation
    final_accuracy = (correct * 100) / total
    print(f'Final accuracy on test data: {final_accuracy:.2f}%')

# Main function to run the complete process
```

```python
def main():
    # Step 1: Mount Google Drive
    mount_drive()

    # Step 2: Load models
    model_paths = [

'/content/drive/MyDrive/mosaicps1/saved_models/audio_classification1.keras',

'/content/drive/MyDrive/mosaicps1/saved_models/audio_classification2.keras'
    ]
    models = load_models(model_paths)

    # Step 3: Define directories
    test_directory = '/content/drive/MyDrive/mosaicps1/testfile/'
    output_directory = '/content/drive/MyDrive/mosaicps1/testimages/'

    # Step 4: Collect test filenames
    test_filename_arr = collect_test_filenames(test_directory)

    # Step 5: Extract features
    extracted_features_df = extract_features(test_filename_arr,
output_directory)

    # Prepare data for predictions
    X = np.array(extracted_features_df['feature'].tolist())
    y_crackles = extracted_features_df['crackles']
    y_wheezes = extracted_features_df['wheezes']
    X = data_preprocessing(X)

    # Reshape and normalize input data
    X = X.reshape(X.shape[0], 180, 180, 3).astype('float32')

    # Step 6: Predict and evaluate
    predict_and_evaluate(models, X, y_crackles, y_wheezes)

# Run the main function
if __name__ == "__main__":
    main()
```

Coin Bazaar

Project Explanation: CoinBazaar - Stock Crypto Trading App

Overview: The **CoinBazaar** project is a **crypto trading platform** that allows users to trade cryptocurrencies in real-time. The app was developed as part of an event organized by the Electronics Engineering Society, and the objective was to build a fully-functional stock trading application with a focus on a **user-friendly experience** and **real-time trading**.

Problem Statement: The goal of the project was to create an app that allows users to:

1. **Trade cryptocurrencies** (buy/sell).
2. **Monitor stock prices** in real-time.
3. Ensure the app is **responsive** for both **mobile** and **laptop** users.
4. Provide a seamless user experience to improve the trading process.

Role in the Project: I was responsible for the **front-end development** of the app. My primary tasks involved:

- Designing and building the **user interface (UI)** in ReactJS.
- Integrating the front-end with the back-end system, ensuring smooth data flow and interactivity.
- Ensuring that the app is **100% responsive** on both mobile and desktop devices.
- Enhancing the user experience through **UI/UX design** and various performance optimizations.

Technologies Used:

- **Front-End:** ReactJS, Bootstrap, CSS, and other frameworks to ensure the app's responsiveness and smooth interactivity.
- **Back-End:** Node.js for server-side development, handling API requests and ensuring data consistency.
- **Database:** MongoDB to store user data and transaction history.

- **Deployment:** The app was deployed using **Vercel** to ensure easy scalability and quick updates.

Features Implemented:

1. **Real-Time Trading:** The app allows users to monitor live cryptocurrency prices and execute buy/sell orders in real-time. This involved integrating third-party APIs that provide up-to-the-minute price updates.
2. **Real-Time Stock Data:** Users can view live market prices and recent trends for various cryptocurrencies. I integrated live data feeds to provide seamless updates within the app.
3. **User Registration and Login:** A simple yet secure authentication system was implemented for users to create accounts, log in, and track their trading activity.
4. **Transaction History:** Users can view a log of their previous trades, including time, price, and volume of transactions.
5. **Responsive Design:** The app was designed to be fully responsive, so users can trade seamlessly whether they are using a mobile device or a desktop computer.

Challenges Faced and Solutions:

1. **Real-time Data Handling:**
   - **Challenge:** Real-time data updates and the smooth refresh of stock prices can be tricky, especially when dealing with high-frequency trading data.
   - **Solution:** I integrated **WebSocket** connections and API polling to ensure that price data was updated in real-time without performance issues.
2. **Ensuring Responsiveness:**
   - **Challenge:** Making the application fully responsive across devices was another challenge. We wanted to ensure that the user experience was consistent on both mobile and desktop.
   - **Solution:** We used **Bootstrap** for grid layout management and custom media queries to ensure the app adjusted dynamically to various screen sizes.
3. **Back-End Integration:**
   - **Challenge:** Ensuring smooth data flow between the front-end and back-end, especially while managing large datasets (e.g., transaction histories) and real-time price updates.
   - **Solution:** I worked closely with the back-end team to ensure data was efficiently fetched using API endpoints and stored correctly in the **MongoDB** database.

Outcome:

- The project was completed within a **5-day hackathon** timeline, where we built the entire application from scratch.
- Our team **won third place** among over 50 teams in the competition, which was a great validation of our hard work and ability to deliver under pressure.

Key Takeaways:

- This project gave me hands-on experience in full-stack development, including **front-end technologies** (ReactJS, CSS) and **back-end integration** (Node.js, MongoDB).
- I learned how to design **scalable and responsive applications** and handle challenges related to real-time data processing and system performance.
- The project also taught me how to **collaborate effectively** in a team, communicate ideas clearly, and iterate on feedback during the hackathon to improve the product.