Heartbeat Audio Classification

Introduction:

Heart disease continues to be the world's greatest cause of mortality, accounting for more deaths each year than all other medical conditions [World Health Organization, 2017]. Because there is a severe scarcity of doctors in rural areas, 75% of primary care is given by unqualified practitioners, contributing to the high death rate from cardiovascular diseases (CVDs) in these areas. Although echocardiograms and electrocardiograms (ECGs) are useful instruments for keeping an eye on heart health, their high cost and need for specialized training prevent them from being widely used in environments with limited resources. Our goal is to develop a dependable, quick, and inexpensive solution that frontline healthcare professionals without training or anybody with internet connection may use.
The purpose of this approach is to identify people who require additional medical evaluation, particularly in areas where access to healthcare providers is restricted. It is possible to considerably lower the risk factors linked to these deaths by detecting CVDs early. In this study, we automate cardiac auscultation—the process of identifying irregularities in heart sounds. We introduce an automated heart sound categorization system that combines a deep convolutional neural network (CNN) with time-frequency heat map representations.

Our Project

This project aims to classify heartbeat audio recordings into two categories: `normal` and `abnormal`.

1. Normal: healthy heart sounds classified by hearing the pattern *lub-dub, lub-dub…*
2. Abnormal : heart sounds like murmur, extrasystole, extrahls or an artifact sound containing no heart sound

The dataset used consists of audio files, which are pre-processed and then used to train a Convolutional Neural Network (CNN) model for classification.

Using our source, we obtained 2 audios classified as normal, and 7 audios classified as abnormal in .m4a format, which were converted to .wav format and then later segmented into 60 audio files each.

Heartbeat_Classifier.ipynb

Dataset Preparation

This is our training notebook. The heart sound data is stored in two folders: `set_c` and `set_d`. Each audio file is labeled with a date prefix, which we map to either `normal` or `abnormal`. The audio files are loaded using the `librosa` library of python. Then we create a dataframe called `dataset` using `pandas`, containing the filenames (path) of the audio data, their labels and offset. The audio files with duration less than 3 seconds were skipped.

Exploratory Data Analysis (EDA)

Once the data frame was created, we performed exploratory data analysis on it. We also visualized the distribution, label counts and audio durations for `normal` and `abnormal` audio clips using pie and pie charts.

Train-Test Split

We split the dataset into training and testing sets, with the ratio of 80 % training data and 20% testing data.

Data visualizations and exploratory data analysis were also performed on the train and test data

Feature Extraction

We extract Mel-Frequency Cepstral Coefficients (MFCCs) as features for training the model by creating `extract_features()` function.

The `extract_features` function is applied to each audio file in the training and testing datasets. This function extracts Mel-frequency cepstral coefficients (MFCCs), which are a common feature used in audio processing to capture the power spectrum of sound. These features are stored in `x_train` and `x_test` lists and then converted to NumPy arrays for efficient numerical computations. We used the `tqdm` library for visual feedback on the progress of the process.

Label Encoding:

Then the `LabelEncoder` from Scikit-Learn is used to convert categorical labels (normal, abnormal) into numerical format and stored in `y_train` and `y_test`. This step is crucial for training the model, as neural networks require numerical input.

Then, the Class weights are computed to handle class imbalance using Scikit-Learn's `compute_class_weight` function and then converted into a dictionary format for use during model training. This helps in ensuring that the model does not become biased towards the majority class.

Data Reshaping and One-Hot Encoding:

The input feature arrays `x_train` and `x_test` are reshaped to add an additional dimension, making them compatible with the input requirements of Convolutional Neural Networks (CNNs), whereas, the labels `y_train` and `y_test` are converted to one-hot encoded format using Keras' `to_categorical` function. This step is necessary for multi-class classification tasks.

Model Creation and Training

The heart sound classification CNN model is built using Keras' Sequential API, consisting of 4 convolutional blocks, each consisting of 2D convolutional layers, a max-pooling layer with a pool size of 2x2, dropout layer with dropout rate 0.2 in three layers and 0.5 in last fourth layer. Then a global average pooling layer is added to reduce each feature map to a single number by taking the average. At last, a fully connected (dense) layer with the number of units equal to the number of classes (i.e., the number of unique labels) is added.

The model is then compiled with categorical cross-entropy function, an Adam optimizer initialized at learning rate = 0.001, and accuracy metric. The `fit` method is called to train the model on the training data (`x_train` and `y_train`) for 300 epochs with a batch size of 128.

- The validation data (`x_test` and `y_test`) is provided for evaluation during training.
- Class weights are passed to handle class imbalance.
- The `shuffle` parameter is set to `True` to shuffle the training data before each epoch.

We also used `%%time` magic command to measure the execution time of the training process.

Evaluation

We evaluate the model performance on the test set and visualize the results using classification report. We also visualize the training history, including loss and accuracy curves.

The trained model was saved for further use and application:

```
model_name = "heartbeat_classifier (normalised).h5"

model.save(model_name)
```

Predictions

The model's performance is evaluated on the test set to assess its accuracy and ability to generalize unseen data by making predictions on the test data (`x_test`). The `verbose=1` parameter ensures that the prediction progress is displayed.
Two empty lists, `y_true` and `y_pred`, are initialized to store the true labels and predicted labels, respectively. `for` loop is used to iterate over each prediction and its corresponding true label in the test set.

By converting the model's output probabilities to class labels, this code prepares the data for further evaluation, such as generating a classification report and computing performance metrics. The resulting `y_true` and `y_pred` lists can be used to assess the model's accuracy, precision, recall, and other relevant metrics.

The trained model can be used by loading it and to make predictions on new audio files.

Heartbeat_detect.ipynb

This is our detection/classification notebook. We imported the necessary libraries first. We then created a dataframe called `dataset` containing audio file names/paths, labels `normal` and `abnormal` and offset. We the split the dataset into train and test data and performed mfcc feature extraction on x_train and x_test and reshaped them, whereas, we did label encoding to convert categorical labels (normal, abnormal) into numerical format and stored in `y_train` and `y_test`.

Classification of a random heartbeat sound

A new heartbeat sound in `.mp4` format was converted to `.wav` using `moviepy` library `AudioFileClip` function. We then loaded our trained model using `load_model` from `keras` and performed the necessary feature extraction on our new heartbeat sound and then used the `model.predict` to predict and visualize the accuracy and loss score along with predicted label (whether it is `normal` or `abnormal` heartbeat sound) and `confidence`.

Gradio_heartbeat_detect.py

This is our `gradio` based user interface python file, used to create a user interface that loads data from audio files, uploaded through the interface or recorded using a microphone.

Input

The code loads the pre-trained model and performs feature extraction on the input audio data.

Output

In the output the interface displays the results in the form of predicted label class whether the audio file is `normal` heartbeat or `abnormal` or `no heartbeat is detected`.

The `interface.launch()` method starts the Gradio interface, and setting `share=True` in the brackets of launch, generates a shareable link that can be accessed by others.

Ngrok

In order to provide a shareable, secure and temporary HTTPS link for the Gradio interface, we use ngrok. Ngrok is a tool that creates a secure tunnel to your local machine, allowing you to expose a local server to the internet. It is useful for demo purposes or when you need to access your local server from a remote location.

After downloading and installing the ngrok from its main website, we make sure that the server is running. We then start ngrok by specifying the port on which your Gradio interface is running: ngrok http http://localhost:7860 (7860 is the port number here)

The command then provides us a list of information which also includes the temporary HTTPS link to access our Gradio interface over the internet and it can also be shared with others to access the interface securely.