Shaghayegh Moradirad

Minerva University

CS156

Machine Learning Pipeline

Binary Audio Classification

ML Pipeline

April 21, 2023

Contents

1 Summary

This comprehensive report explores various machine-learning techniques and models for audio data processing, feature extraction, and classification. The study begins with a range of sample data representations, including waveform and magnitude spectrum.

Feature extraction and comparison play a crucial role in the analysis. The report highlights potential biases while examining the models using various metrics including confusion matrices, AUC, ROC curve metrics, f1-score, etc.

The study proceeds to explore more advanced techniques, for example, RNN, CNN, and transfer learning via Google's YAMNet and MobileNetV2 models.

Additionally, data leakage avoidance is emphasized throughout the report. Utmost effort has been put to identify overfitting, underfitting, and addressing them.

The report goes into feature extraction using Mel-Frequency Cepstral Coefficients (MFCCs), normalization, and training, as well as auto-encoders for denoising. Several mathematical walkthroughs are provided to explain the underlying concepts. Finally, the report compares the performance of various models, and discusses their implications for binary audio classification regarding the problem's dataset.

2 Problem Statement & Data Collection

Problem statement: binary classification of my voice vs. my mother's voice Many times during my life, I had picked up the phone at our house, and I have been mistaken by my mom. And many times she has been told to give the phone to her parents.

In this assignment, I am experimenting with traditional machine learning models and audio features to see whether the models can accurately identify my mother's voice from mine or not. Data The data includes 100 audio files half of which were collected from myself and half from my mother, in each of which we say the same exact sentence in the formal Persian dialect. The fifty sentences are picked from recent news and are recorded in a noise-free environment. A few of the original samples were excluded from the dataset due to various reasons such as premature termination of the audio file, incomplete recording of the sentence, echoes in the audio files caused by the surroundings, variation in the pronunciation of certain words, and excessively long or short audio files.

It has been initially 25 audio files from my mother and 25 audio files from myself but results showed that the problem is more complex to be able to receive a desirable learning from a dataset of size 50 audios.

For binary classification, it'd be desirable to have a dataset of size 100 but due to internet issues in my country, I was not able to get more data initally. However, I added more data and augmented my data. The inital problem turned out to help me add more data to the project easily, through the developed pipline.

Data and all the files are available at This link

3 Mounting the Drive and Acccesing the project

```
[ ]: from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

Below, replace `/content/drive/MyDrive/Pipeline_final` with the path of the project on your drive after mounting by right clicking on the project folder and getting the path!

```
[ ]: %cd /content/drive/MyDrive/Pipeline_final
```

```
/content/drive/MyDrive/Pipeline_final
```

4 Importing the required libraries

```
[ ]: import pandas
     import numpy as np
     import matplotlib
     import seaborn
     import os
     import soundfile as sf
     import IPython.display as ipd
     import matplotlib.pyplot as plt
     import librosa
     import librosa.display
     from sklearn.model_selection import train_test_split
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import cross_val_score
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.datasets import make_classification
     from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
      ↪f1_score
```

5 Sample Data Representations

[ ]:
```
mom_sample_path="./converted_mom/sen_1.wav"
me_sample_path="./converted_me/sen_1.wav"
```

5.1 Audio Player

[ ]:
```
ipd.Audio(mom_sample_path)
```

[ ]:
```
<IPython.lib.display.Audio object>
```

[ ]:
```
ipd.Audio(me_sample_path)
```

[ ]:
```
<IPython.lib.display.Audio object>
```

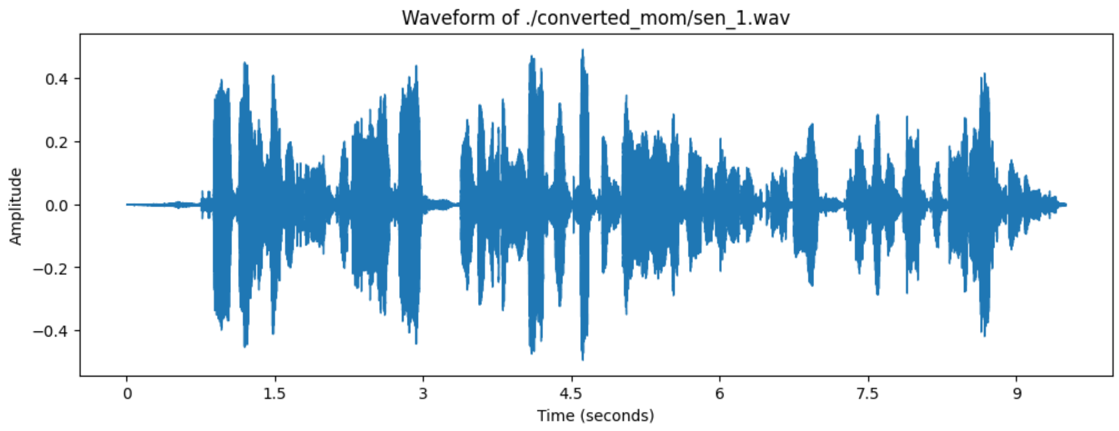
5.2 Number of samples

[ ]:
```
#load the audio files
mom_sample_loaded, sr=librosa.load(mom_sample_path)
me_sample_loaded, _=librosa.load(me_sample_path)
#these are now a numpy array - the wave form associated with the audio file

print("----the numuber of samples in the audio sample 1-----")
print("    mom sample: ",mom_sample_loaded.shape[0])
print("    me sample: ",me_sample_loaded.shape[0])
```

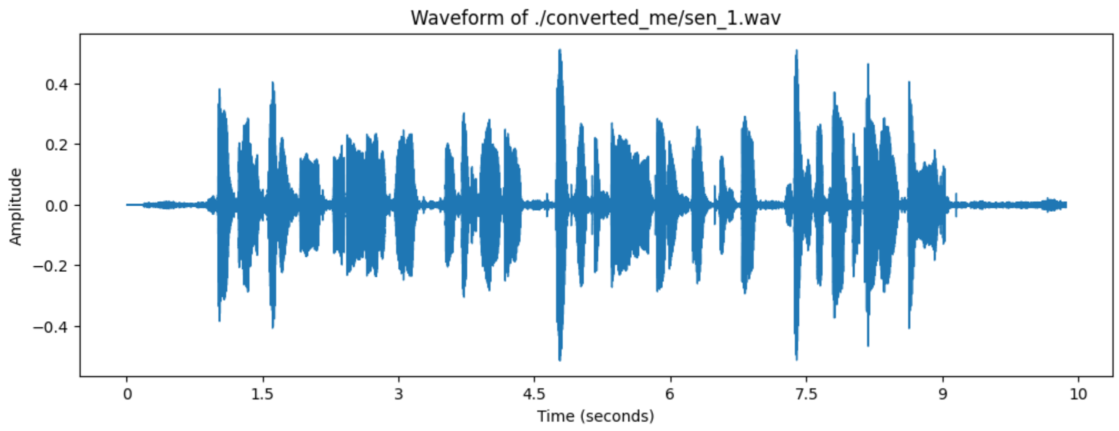
```
----the numuber of samples in the audio sample 1-----
    mom sample:  209428
    me sample:  228391
```

5.3 The waveform

[ ]:
```
# Load the WAV file
wav_file = mom_sample_path
y, sr = librosa.load(wav_file)

# Plot the waveform
plt.figure(figsize=(12, 4))
librosa.display.waveshow(y, sr=sr)
plt.title('Waveform of {}'.format(wav_file))
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.show()
```



```
[ ]: # Load the WAV file
     wav_file = me_sample_path
     y, sr = librosa.load(wav_file)

     # Plot the waveform
     plt.figure(figsize=(12, 4))
     librosa.display.waveshow(y, sr=sr)
     plt.title('Waveform of {}'.format(wav_file))
     plt.xlabel('Time (seconds)')
     plt.ylabel('Amplitude')
     plt.show()
```



We can see that the datapoints have different waveforms when both individuals are saying the same sentence; thus, there are features to make them noticable from each other. However across the dataset, the wave forms may differ due to the different sentences being pronounced in different samples. (Above is only sentence number 1 out of the 25 sentences in the dataset.)

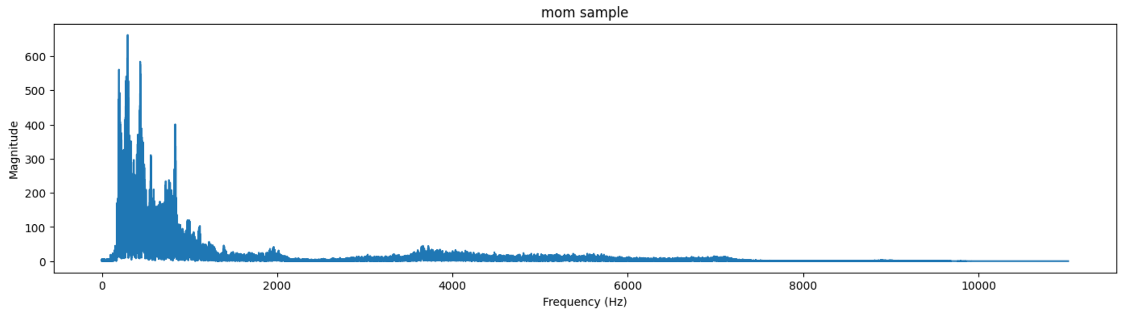
In order to limit the effect of different sentences, we move to a frequncey-magnitude plot below, in which we see the magnitude of different frequencies in the audio file overall. Thus the time aspect is out of our next plot; however it is importnat to note that the longer the audio file is, the more of each frequency can exist (probability of a seeing a frequency in an audio increases and we would need a normalisation method to account for that. But this is not what we are lookign at below.)

What we have below, derived from using the Fast Fourier Transform (FFT), is the energy of each frequency over the audio file, meaning the energy that the specific frequency had in condencing the air molecules between the speaker and the recorder. (or the pressure it created in the air molecules.)

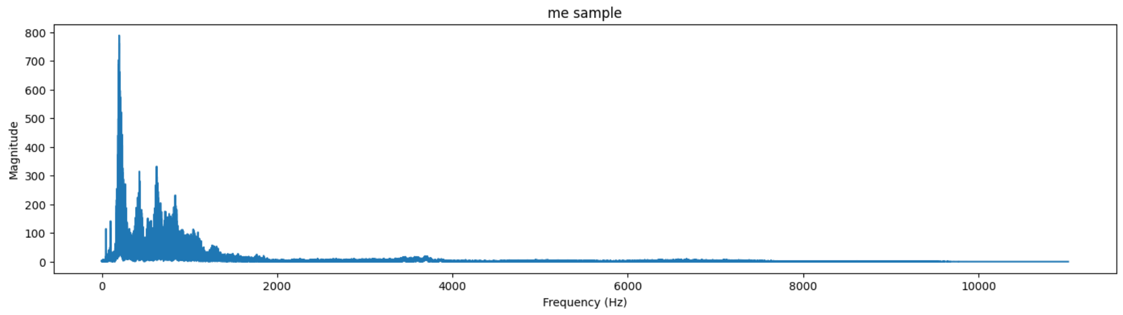
5.4 Magnitude spectrum

```
[ ]: def plot_magnitude_spectrum(signal, title, sr, f_ratio=0.5):
         """
         Plotting the magnitude spectrum of a signal

         Parameters
         ----------
         signal : numpy array
             The signal to be plotted
         title : string
             The title of the plot
         sr : int
             The sampling rate of the signal
         f_ratio : float
             The ratio of the frequency bins to be plotted (default is 0.5)
             reason explained in the comments below
         """
         X = np.fft.fft(signal) # Fourier Transform calculation
         mag_spec = np.absolute(X) # Magnitude calculation

         plt.figure(figsize=(17, 4))

         freq = np.linspace(0, sr, len(mag_spec)) #creating the frequency bins␣
     ↪associated with the magnitude values
         freq_bins = int(len(mag_spec)*0.5)  #only plotting the first half of the␣
     ↪spectrum
                                             #(the second half is a mirror image of the␣
     ↪first half) - theorem of Nyquist

         plt.plot(freq[:freq_bins], mag_spec[:freq_bins]) #slicing the frequency and␣
     ↪magnitude arrays to plot
                                                          #only the first half (by␣
     ↪defult)
         plt.xlabel('Frequency (Hz)')
         plt.ylabel('Magnitude')
         plt.title(title)
```

```
[ ]: plot_magnitude_spectrum(mom_sample_loaded, "mom sample", sr)
```



```
[ ]: plot_magnitude_spectrum(me_sample_loaded, "me sample", sr)
```



We can see above the the magnitude of the frequencies differ between the datapoint that belongs to me and the datapoint blonging to my mother (their energy - not in decibell scale here).

This is one way we can identify these two audios from each other as a human. we can see the highest magnitudes reached in both audios are nearly havingthe same frequencies but in my audio the peak is higher and has a larger difference with the rest of the peaks while in my mother's the peaks are less in terms of magnitude and more frequencies reach to high magnitudes.

6 Sample Extracting Features

```
[ ]: def extract_features(audio_file_path):
         """
         Extracting the RMS and ZCR features from an audio file

         Parameters
         ----------
         audio_file_path : string
             The path to the audio file
```

```
    Returns
    -------
    RMS : numpy array
        The RMS values of the audio file
    ZCR : numpy array
        The ZCR values of the audio file
    """
    audio_file, _ = librosa.load(audio_file_path)
    FRAME_SIZE = 1024 #explained below
    HOP_LENGTH = 512 #explained below
    RMS=librosa.feature.rms(y=audio_file, frame_length=FRAME_SIZE,␣
↪hop_length=HOP_LENGTH)[0]
    ZCR=librosa.feature.zero_crossing_rate(y=audio_file,␣
↪frame_length=FRAME_SIZE, hop_length=HOP_LENGTH)[0]
    return RMS, ZCR

    #add  chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).
↪T,axis=0)!!!!!!!!!!!!!!!!!!!!!!!
```

- FRAME_SIZE = 1024
  - This value represents the number of samples in each audio frame. Choosing a small frame size allows for more temporal resolution but less frequency resolution, while a larger frame size allows for more frequency resolution but less temporal resolution. A frame size of 1024 strikes a balance between the two.
- HOP_LENGTH = 512
  - This value represents the number of samples to shift the frame window after each frame. A hop length of 512 is chosen because it provides a good balance between temporal resolution and computational efficiency.

6.1 Comparing the features - selection

```
[ ]: # the sample number 12 is chosen since the duration of the two categories are
     # very similar (visible from the scatter plot)
     RMS_mom_sample, ZCR_mom_sample=extract_features("./converted_mom/sen_12.wav")
     RMS_me_sample, ZCR_me_sample=extract_features("./converted_me/sen_12.wav")
     print("---Root-mean-square value on average (RMS)---")
     print("     avg RMS_mom_sample: ",np.mean(RMS_mom_sample))
     print("     avg RMS_me_sample: ",np.mean(RMS_me_sample))
     print("\n") #line break
     print("---Zero-crossing rate (ZCR)---")
     print("     avg ZCR_mom_sample: ",np.mean(ZCR_mom_sample))
     print("     avg ZCR_me_sample: ",np.mean(ZCR_me_sample))
```

```
---Root-mean-square value on average (RMS)---
     avg RMS_mom_sample:  0.110076845
     avg RMS_me_sample:  0.062446233
```

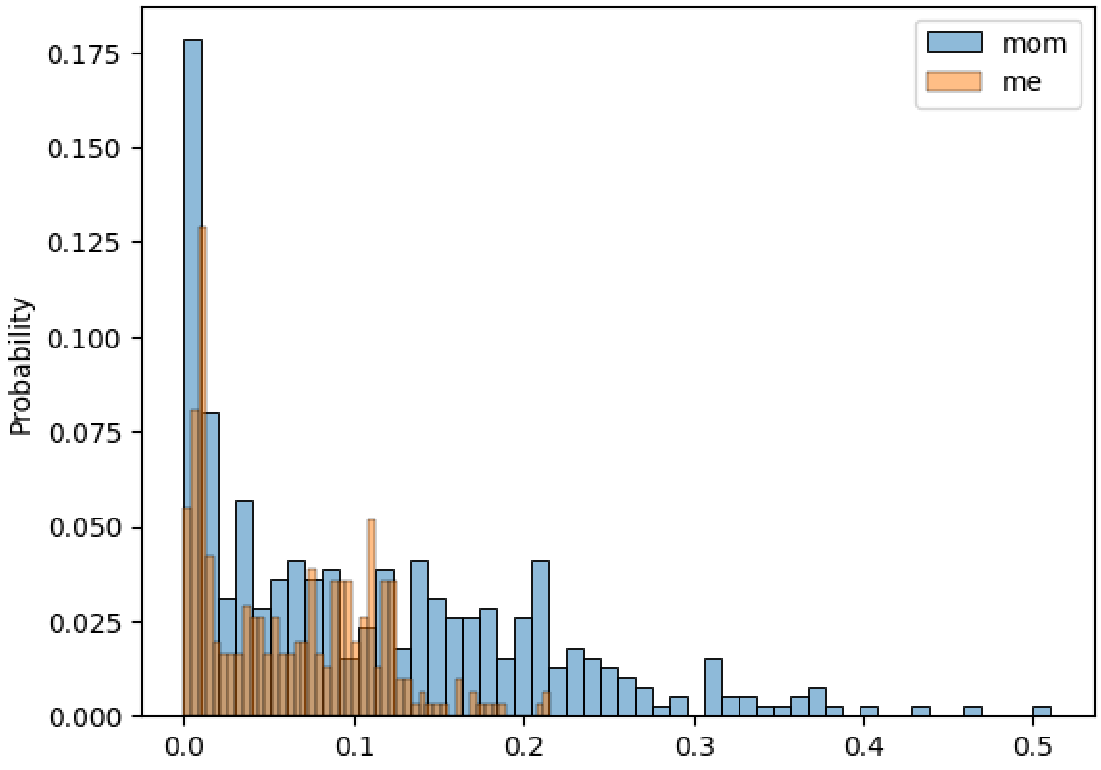
```
---Zero-crossing rate (ZCR)---
     avg ZCR_mom_sample:  0.08381278262273902
     avg ZCR_me_sample:  0.05375819052419355
```

we can see that it is rather a fundamental difference between our audios that is due to the pace of our speech.

6.2 Feature Histograms

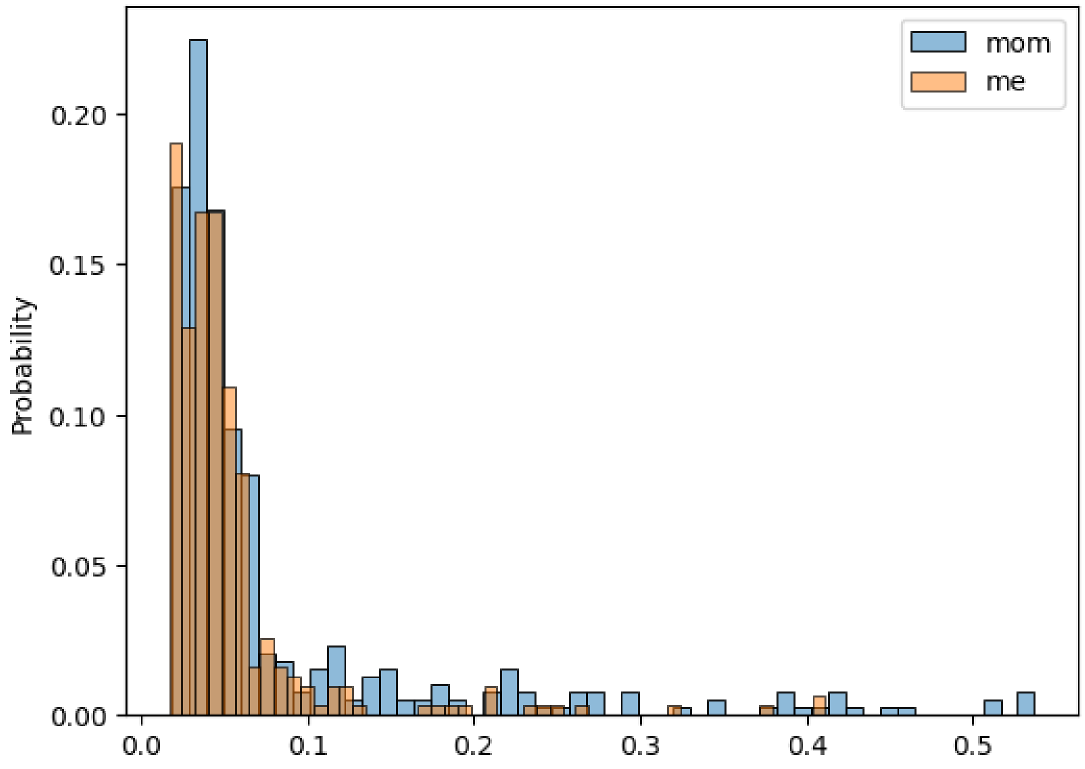
[ ]:
```
seaborn.histplot(
    data=RMS_mom_sample.flatten(),
    stat='probability', alpha=0.5, label='mom',bins=50
)

seaborn.histplot(
    data=RMS_me_sample.flatten(),
    stat='probability', alpha=0.5, label='me',bins=50
)

plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x7feb4bfc3a60>
```



```
[ ]: seaborn.histplot(
         data=ZCR_mom_sample.flatten(),
         stat='probability', alpha=0.5, label='mom', bins=50
     )

     seaborn.histplot(
         data=ZCR_me_sample.flatten(),
         stat='probability', alpha=0.5, label='me', bins=50
     )
     plt.legend()
```

```
[ ]: <matplotlib.legend.Legend at 0x16ca8d550>
```



Both RMS and ZCR of my mother's audio in the sample datapoint that we are looking at, are higher than my audio sample. - The zero-crossing rate (ZCR) is a commonly used feature in audio signal processing that measures the number of times the audio waveform crosses the zero amplitude axis per unit time. In other words, it counts the number of times the audio waveform changes its polarity from positive to negative or vice versa. The ZCR is a useful indicator of the frequency content of an audio signal, as it tends to be higher for signals with high frequency content and lower for signals with low frequency content. For example, a high-pitched sound such as a flute or a whistle will have a higher ZCR than a low-pitched sound such as a bass guitar or a drum.

- The root-mean-square (RMS) is a commonly used feature in audio signal processing that measures the “loudness” or energy of an audio signal. It is calculated by taking the square root of the mean of the squared amplitude values of an audio signal. Having a higher average RMS in an audio signal means that the signal has a higher overall energy or loudness. This could be due to a number of factors, such as the presence of louder sounds or instruments, a higher number of audio tracks playing simultaneously, or simply a higher gain or volume level. It’s worth noting that while a higher RMS value generally indicates a louder audio signal, it’s not necessarily a direct measure of perceived loudness, as the human perception of loudness can be influenced by a number of other factors such as frequency content and duration of the audio signal. (more detail in phon measure)

7 Data loading

[ ]:
```
#set the base directory
#base_directory="/Users/shaghayeghmoradirad/Documents/GitHub/Pipeline_final/"
base_directory="./"
#set the mom and me folder names
mom_folder="converted_mom"
me_folder="converted_me"

#set the mom and me paths
mom_paths=os.path.join(base_directory,mom_folder)
me_paths=os.path.join(base_directory,me_folder)
```

7.1 Potential Bias - Duration Exploration

[ ]:
```
def list_dur_all_audios(folder_path):
    """
    Calculating the average duration of all the audio files in a folder

    Parameters
    ----------
    folder_path : string
        The path to the folder containing the audio files

    Returns
    -------
    avg_dur : float
        The average duration of all the audio files in the folder
    """
    audios = []
    audios_dur_list = []
    for root, dirs, files in os.walk(folder_path, topdown=False):
        for name in files:
            try:
                audio = librosa.load(os.path.join(root, name))[0]
                audio_dur=librosa.get_duration(y=audio)
                audios_dur_list.append(audio_dur)
            except:
                continue
    return audios_dur_list

def avg_list(list_to_average):
    return sum(list_to_average)/len(list_to_average)

print("----the avg. duration of the audio files-----")
print("    mom avg. audios duration: ",avg_list(list_dur_all_audios(mom_paths)))
print("    me avg. audio duration: ",avg_list(list_dur_all_audios(me_paths)))
```

mom avg. audios duration: 7.689068480725625

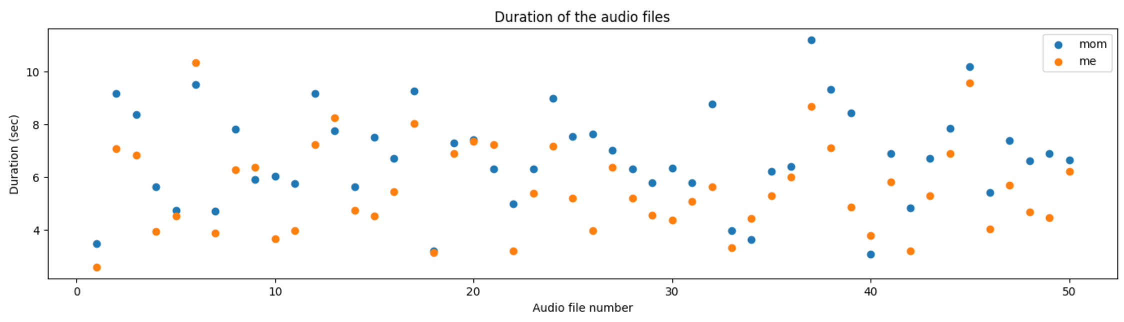
me avg. audio duration: 6.750668480725624

We can see above that the average duration of my audio files are lower than my mother's; we can look at the durations in each file as well, in order to see if the lower average is due to an outlier.

```
[ ]: list_dur_all_audios_mom=list_dur_all_audios(mom_paths)
     list_dur_all_audios_me=list_dur_all_audios(me_paths)

     #uncomment the following lines to see the duration of the audio files one by one

     # print("----the duration of the audio files one by one-----")
     # print("    mom audios duration: ",list_dur_all_audios_mom)
     # print("    me audio duration: ",list_dur_all_audios_me)
```

```
[ ]: #plot the duration of my audios vs. my mother's audios
     plt.figure(figsize=(17, 4))
     audio_num_mom=range(1,len(list_dur_all_audios_mom)+1)
     audio_num_me=range(1,len(list_dur_all_audios_me)+1)
     plt.scatter(audio_num_mom ,list_dur_all_audios_mom ,label="mom")
     plt.scatter(audio_num_me,list_dur_all_audios_me, label="me")
     plt.xlabel('Audio file number')
     plt.ylabel('Duration (sec)')
     plt.title("Duration of the audio files")
     plt.legend()
     plt.show
```

```
[ ]: <function matplotlib.pyplot.show(close=None, block=None)>
```



We can see in the 25 sentences that most of the time, my mother's audio samples are longer/have more duration compared to mine; sometimes this difference is high and sometimes lower. Thus we could either do padding and cutting audios to the max or the average audio. (This would allow pace to be an effective factor in the learning.)

Or we can remove the effect of time by getting the average and stnadard deviation of a select group of features of the audios and then using those values to train the model. (This way we are learning to reconize the person regardless of the length of their sample audio which is desired in the assignment.)

8 Extracting Features

```
[ ]: def feature_vec_gen(audio_path):
         """
         Generating the feature vector from an audio file

         Parameters
         ----------
         audio_path : string
             The path to the audio file

         Returns
         -------
         vector_features : numpy array
             The feature vector of the audio file
         """
         RMS=extract_features(audio_path)[0]
         ZCR=extract_features(audio_path)[1]
         RMS_avg=np.mean(RMS)
         ZCR_avg=np.mean(ZCR)
         RMS_sd=np.std(RMS)
         ZCR_sd=np.std(ZCR)
         vector_features=np.array([RMS_avg,RMS_sd,ZCR_avg,ZCR_sd])
         return vector_features
```

[ ]:
```
feature_vec_gen(me_sample_path)
```

[ ]:
```
array([0.04598553, 0.04230965, 0.07179154, 0.09388996])
```

8.1 Paths and Feature Vectors

[ ]:
```
def path_vec_gen(folderpath):
    """
    Generating the feature vectors and the paths of the audio files in a folder

    Parameters
    ----------
    folderpath : string
        The path to the folder containing the audio files

    Returns
    -------
    audios_vecs : list
        The list of the feature vectors of the audio files in the folder
    audios_path : list
        The list of the paths of the audio files in the folder
    """
    audios_vecs = []
    audios_path = []
    for root, dirs, files in os.walk(folderpath, topdown=False):
        for name in files:
            file_path=os.path.join(root, name)
            try:
                data = feature_vec_gen(file_path)
                audios_path.append(file_path)
                audios_vecs.append(data)
            except:
                continue
    return audios_vecs, audios_path
```

[ ]:
```
mom_audios_features_v, mom_paths_data=path_vec_gen("./converted_mom")
me_audios_features_v, me_paths_data=path_vec_gen("./converted_me")
```

[ ]:
```
x=np.concatenate((mom_audios_features_v, me_audios_features_v))
path = np.concatenate((mom_paths_data,me_paths_data))
y=np.concatenate((np.zeros(len(mom_audios_features_v)),np.
 ↪ones(len(me_audios_features_v))))
labels_main=y
```

[ ]:
```
print(x[0].shape) #mean of RMS, sd of RMS, mean of ZCR, sd of ZCR
x.shape #number of audio files, number of features
```

```
(4,)
```

```
[ ]: (100, 4)
```

Documented the process for generating the path arrays along with the vector of features. (Useful when adding additional data to the dataset)Inspecting the shape to check if the files are as expected.

```
mom_audios_features_v_2, mom_paths_data_2=path_vec_gen("./secondary_mom")
me_audios_features_v_2, me_paths_data_2=path_vec_gen("./secondary_me")
x_2=np.concatenate((mom_audios_features_v_2, me_audios_features_v_2))
path_2 = np.concatenate((mom_paths_data_2,me_paths_data_2))
y_2=np.concatenate((np.zeros(len(mom_audios_features_v_2)),np.ones(len(me_audios_features_v_2)
```

```
[ ]: x_main=x #np.concatenate((x,x_2))
     path_main = path #np.concatenate((path,path_2))
     y_main=y #np.concatenate((y,y_2))
```

```
[ ]: print(x_main[0].shape) #mean of RMS, sd of RMS, mean of ZCR, sd of ZCR
     x_main.shape #number of audio files, number of features
```

```
(4,)
```

```
[ ]: (100, 4)
```

8.2 Data Compilation

```
[ ]: #all the data from each class in one array
     mom_all_features_vectors = mom_audios_features_v #np.
      ↪concatenate((mom_audios_features_v_2,mom_audios_features_v))
     mom_all_paths = mom_paths_data #np.
      ↪concatenate((mom_paths_data_2,mom_paths_data))
     me_all_features_vectors = me_audios_features_v #np.
      ↪concatenate((me_audios_features_v_2,me_audios_features_v))
     me_all_paths = me_paths_data #np.concatenate((me_paths_data_2,me_paths_data))


     y_me = np.ones(len(me_all_features_vectors))
     y_mom = np.zeros(len(mom_all_features_vectors))

     #random seed for reproducibility
     random_seed=0
     #splitting the data into train and test sets
     mom_train, mom_test, mom_y_train, mom_y_test =␣
      ↪train_test_split(mom_all_features_vectors, y_mom, test_size=0.3,␣
      ↪random_state=random_seed)
     mom_path_train, mom_path_test, mom_y_train, mom_y_test =␣
      ↪train_test_split(mom_all_paths, y_mom, test_size=0.3,␣
      ↪random_state=random_seed)
```

```
me_train, me_test, me_y_train, me_y_test =␣
 ↪train_test_split(me_all_features_vectors, y_me, test_size=0.3,␣
 ↪random_state=random_seed)
me_path_train, me_path_test, me_y_train, me_y_test =␣
 ↪train_test_split(me_all_paths, y_me, test_size=0.3, random_state=random_seed)

#combining the data from both classes
x_train = np.concatenate((mom_train,me_train))
x_test = np.concatenate((mom_test,me_test))
y_train = np.concatenate((mom_y_train,me_y_train))
y_test =  np.concatenate((mom_y_test,me_y_test))

path_train = np.concatenate((mom_path_train,me_path_train))
path_test = np.concatenate((mom_path_test,me_path_test))
x= np.concatenate((x_train,x_test))
print(len(x.flatten()),len(np.unique(x)))
len(path_test) == len(set(path_test)), len(path_train) == len(set(path_train))
```

```
400 400
```

```
[ ]: (True, True)
```

9 Logistic Regression

Model selection is the process of selecting the best model among a set of candidate models based on their performance on a given dataset. In our case logistic regression has been selected as it provides a probability estimate of the positive class which makes the model more interpretable. Although Logistic regression does not have critical hyperparameters to tune. (Used hyperparameters discussed in Appendix D) Using cross-validation, I managed to tune the hyperparameters of the model. I used grid search, to evaluate the performance of different hyperparameter settings. Once the optimal hyperparameters are identified based on the models' performance using a suitable performance metric from accuracy, precision, recall, F1-score, and area under the receiver operating characteristic curve (AUC-ROC) we can decide the model that generalizes best and test it on the test set. In our case, a high accuracy of 90% has been reached but it is important to note that the test set includes equal numbers of each class (my audio files and my mother's audio files) thus only predicting either one class all the time cannot provide an accuracy of more than 50%. Additionally, high precision, recall, F1-score, and AUC validate our model's performance. The loss function for the logistic regression is called the log loss (also known as the binary cross-entropy loss) to measure the discrepancy between the predicted probabilities and the true labels. The log loss is defined as:

$$L(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i log(\hat{y_i}) + (1 - y_i)log(1 - \hat{y_i})]$$

where y is a binary vector of true labels (0 or 1), is a vector of predicted probabilities, and n is the number of samples. The log loss penalizes the model for making incorrect predictions and estimates the uncertainty of the predictions. The goal of logistic regression is to find the model parameters that minimize the log loss on the training data, using an optimization algorithm such as gradient descent or its variants. Since logistic regression doesn't have a closed-form solution, it is solved iteratively using optimization algorithms such as gradient descent, stochastic gradient descent, or L-BFGS. The choice of an optimization algorithm can have an impact on the convergence speed and accuracy of the logistic regression model.

9.1 Hypertuning

[ ]:
```
%%capture #suppressing the output as it is too long
# Define the hyperparameters and their possible values
param_grid = {'C': [0.1, 1, 10], 'penalty': ['elasticnet','l1', 'l2'], 'solver':
  ↪ ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga']}

# Create a GridSearchCV object with the LogisticRegression model and the␣
  ↪hyperparameter grid
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5,␣
  ↪scoring='accuracy')

# Fit the GridSearchCV object to the training data
grid_search.fit(x_train, y_train)

# Get the best hyperparameters and the best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

# Print the results
print('Best hyperparameters:', best_params)
print('Best accuracy score:', best_score)
```

Best hyperparameters: {'C': 10, 'penalty': 'l1', 'solver': 'liblinear'} Best accuracy score: 0.9857142857142858

Training and Testing Logistic Regression

[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


logreg = LogisticRegression( C=10, penalty='l1', solver='liblinear') #best␣
  ↪hyperparameters

"""
logistic regression works by finding the best hyperplane that separates the␣
  ↪data into two classes
```

```
C is the inverse of regularization strength, smaller values specify stronger␣
↪regularization
penalty is the norm used in the penalization
solver is the algorithm used in the optimization problem
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.
↪LogisticRegression.html
"""

logreg.fit(x_train, y_train) #fitting the model on the training data
y_pred = logreg.predict(x_test) #predicting the labels of the test data
print('Accuracy of logistic regression classifier on test set: {:.2f}'.
↪format(logreg.score(x_test, y_test)))
#print(y_train_2==y_train, y_test_2==y_test)
```

```
Accuracy of logistic regression classifier on test set: 0.93
```

9.2 Inspecting misclassified audio files

```
[ ]: #getting the path of the misclassified audio files
y_pred = logreg.predict(x_test)
np.where(y_pred!=y_test), path_test[y_pred!=y_test]
```

```
[ ]: ((array([ 6, 10]),),
      array(['./converted_mom/sec_3.wav', './converted_mom/sec_2.wav'],
            dtype='<U26'))
```

9.3 Checking the balance of the test set

```
[ ]: path_test #checking the test data
```

```
[ ]: array(['./converted_mom/sec_5.wav', './converted_mom/sen_6.wav',
           './converted_mom/sec_24.wav', './converted_mom/sec_15.wav',
           './converted_mom/sen_2.wav', './converted_mom/sen_22.wav',
           './converted_mom/sec_3.wav', './converted_mom/sen_21.wav',
           './converted_mom/sen_13.wav', './converted_mom/sec_22.wav',
           './converted_mom/sec_2.wav', './converted_mom/sen_25.wav',
           './converted_mom/sen_23.wav', './converted_mom/sen_19.wav',
           './converted_mom/sen_14.wav', './converted_me/sec_5.wav',
           './converted_me/sen_6.wav', './converted_me/sec_24.wav',
           './converted_me/sec_15.wav', './converted_me/sen_2.wav',
           './converted_me/sen_22.wav', './converted_me/sec_3.wav',
           './converted_me/sen_21.wav', './converted_me/sen_13.wav',
           './converted_me/sec_22.wav', './converted_me/sec_2.wav',
           './converted_me/sen_25.wav', './converted_me/sen_23.wav',
           './converted_me/sen_19.wav', './converted_me/sen_14.wav'],
          dtype='<U26')
```

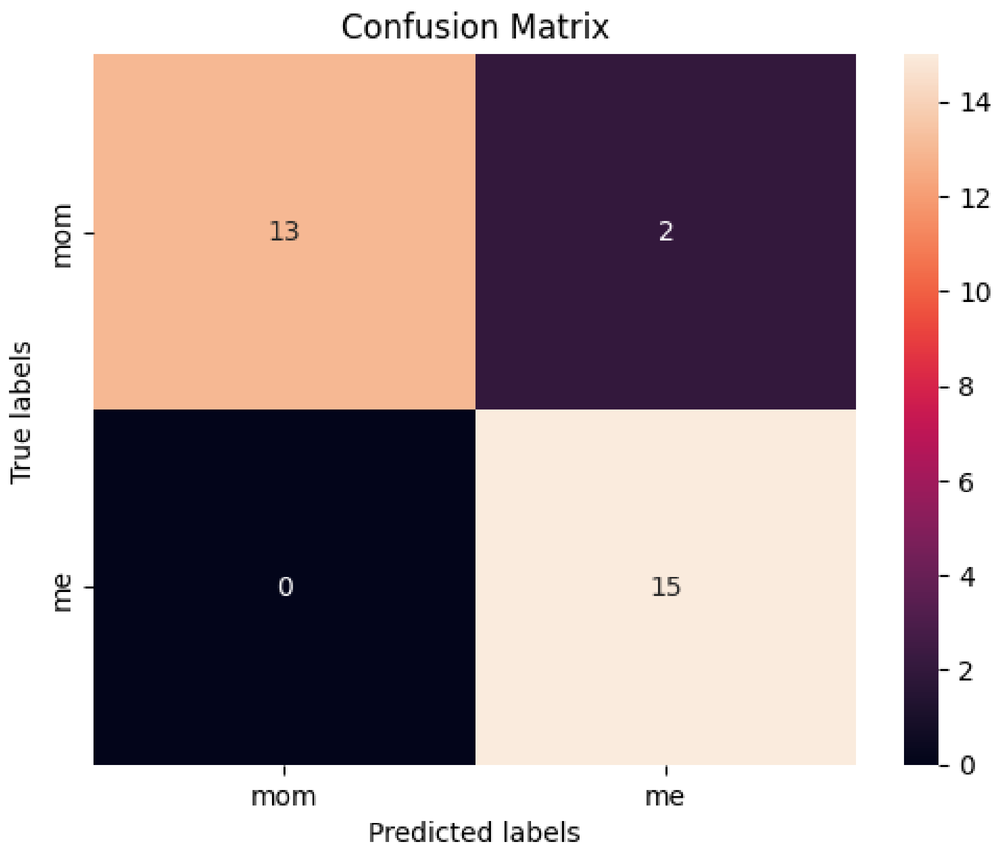
```
[ ]: #checking the training data's balance

     path_train
     mom=0
     me=0
     for path in path_train:
         if "mom" in path:
             mom+=1
         else:
             me+=1
     print("the number of mom audio files in the training data is: ", mom)
     print("the number of me audio files in the training data is: ", me)
```

```
the number of mom audio files in the training data is:  35
the number of me audio files in the training data is:  35
```

9.4 Confusion matrix

```
[ ]: #plot the confusion matrix
     from sklearn.metrics import confusion_matrix
     import matplotlib.pyplot as plt
     import seaborn as sns
     import numpy as np

     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     ax= plt.subplot()
     sns.heatmap(cm, annot=True, ax = ax, fmt='g'); #annot=True to annotate cells

     # labels, title and ticks
     ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
     ax.set_title('Confusion Matrix');
     ax.xaxis.set_ticklabels(['mom', 'me']); ax.yaxis.set_ticklabels(['mom', 'me']);
```

```
[[13  2]
 [ 0 15]]
```



Firstly, the dataset is small. In order to improve the results, the data size should at least double across both classes. This is noticeable in the case that I do not equally split the training set to include half of each class; the model easily propagates the bias in the training set and tends to detect most of the audio files to be from the class that was dominant in the training set. I resolved this by adding to the data and making sure the training set has an equal number of each class, which in response insured that the training set also has an equal number of each class. Looking at the confusion matrix above, one may think that this situation has not been avoided and all the incorrect predictions belong to one class, however, that is because of the distribution of the data points. If we plot the training data points across two of the most weighted features we'd see the plot below.

```
[ ]: #compute the logistic regression coefficients
coefficients = logreg.coef_
print(coefficients)
```

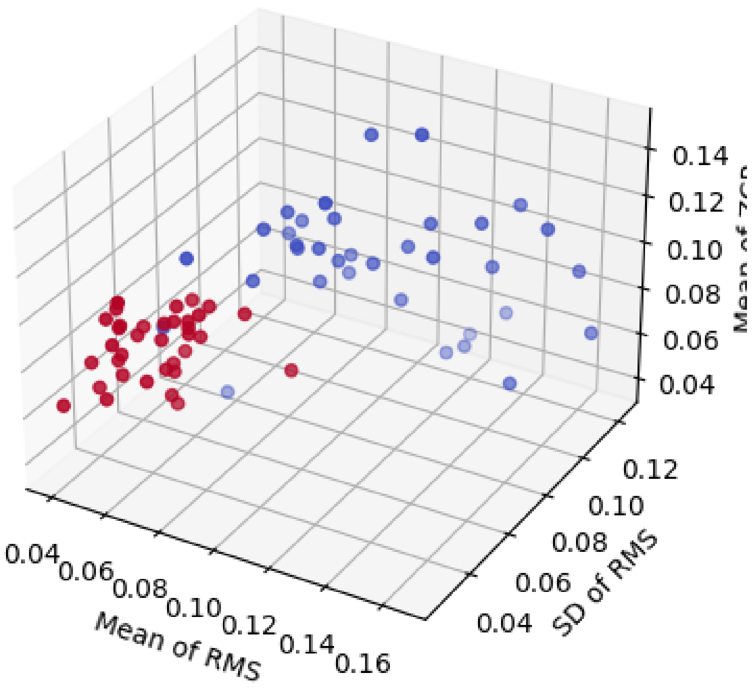
```
[[  0.         -82.86426885 -21.78098338   0.        ]]
```

9.5 Visual aid

- visualizing the data across the slected features

[ ]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_train[:,0], x_train[:,1], x_train[:,3], c=y_train, cmap=plt.cm.
 ↪coolwarm)
ax.set_xlabel('Mean of RMS')
ax.set_ylabel('SD of RMS')
ax.set_zlabel('Mean of ZCR')
#fig.savefig('3d_plot.png')
```

[ ]:
```
Text(0.5, 0, 'Mean of ZCR')
```

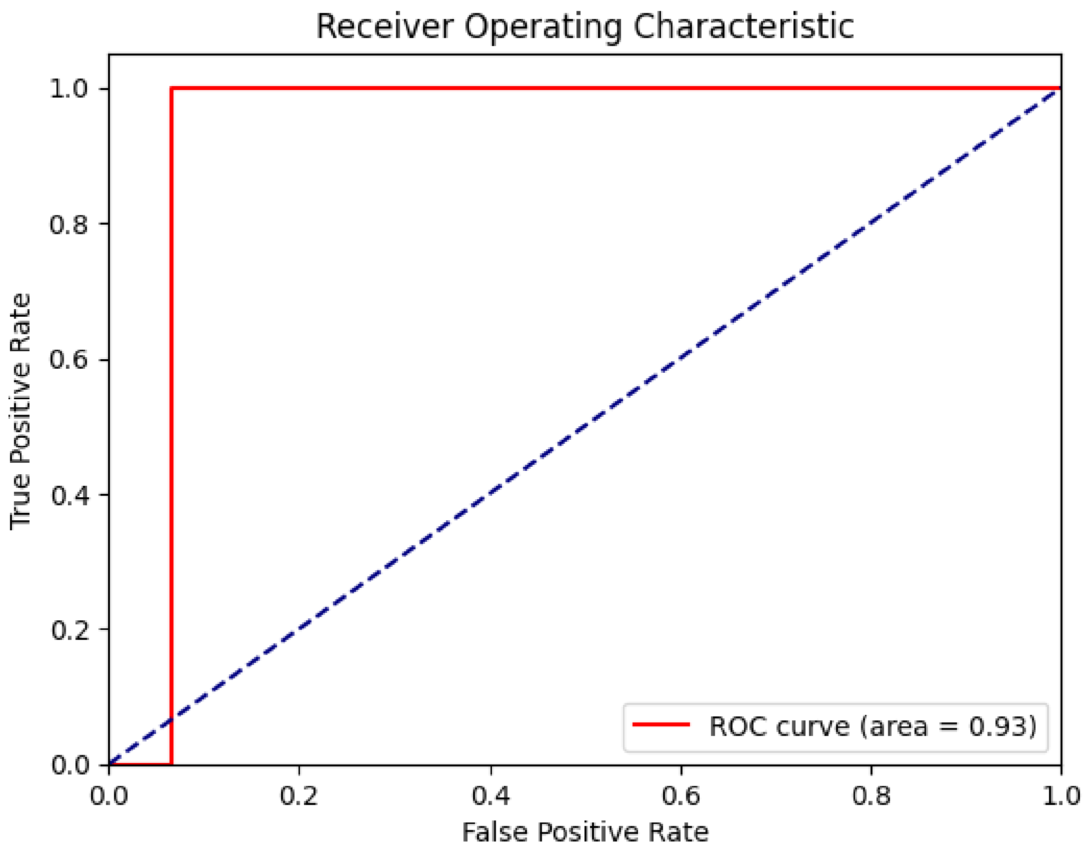


9.6 AUC and ROC curve

[ ]:
```
from sklearn.metrics import roc_auc_score, roc_curve

# Make predictions on test set
y_pred = logreg.predict(x_test)

# Compute predicted probabilities for test set
y_scores = logreg.predict_proba(x_test)[:, 1]

# Compute AUC score
```

```python
auc_score = roc_auc_score(y_test, y_scores)
print('AUC Score:', auc_score)

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_scores)

# Plot ROC curve
plt.plot(fpr, tpr, color='red', label='ROC curve (area = %0.2f)' % auc_score)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

```
AUC Score: 0.9333333333333333
```



Above we can see the ROC curve, we'd like to be as close as possible to the left upper corner to have close to 0 false positive rate and close to 1 True positive rate. Depending on the use of the model that determines our preference for FP or FN we are able to choose a threshold tha satisfies our needs best. In the curve we can see how expensive, in terms of FPR, it is to increase the TPR, and vice versa. We can also evauate our model by one value out of the ROC curve which is the AUC of the curve that does not depend on the chosen threshholds.

9.7 Comparison with Random Forest Model

```
[ ]: #using another model to compare the results

     # Initialize a random forest classifier with max_depth=2 and 100 trees
     rfc = RandomForestClassifier(n_estimators=100, max_depth=2, random_state=42)

     # Perform 5-fold cross-validation
     cv_scores = cross_val_score(rfc, x_train, y_train, cv=5)

     # Print the mean accuracy and standard deviation of the cross-validation scores
     print("Cross-validation scores:", cv_scores)
     print("Mean accuracy:", cv_scores.mean())
     print("Standard deviation:", cv_scores.std())
```

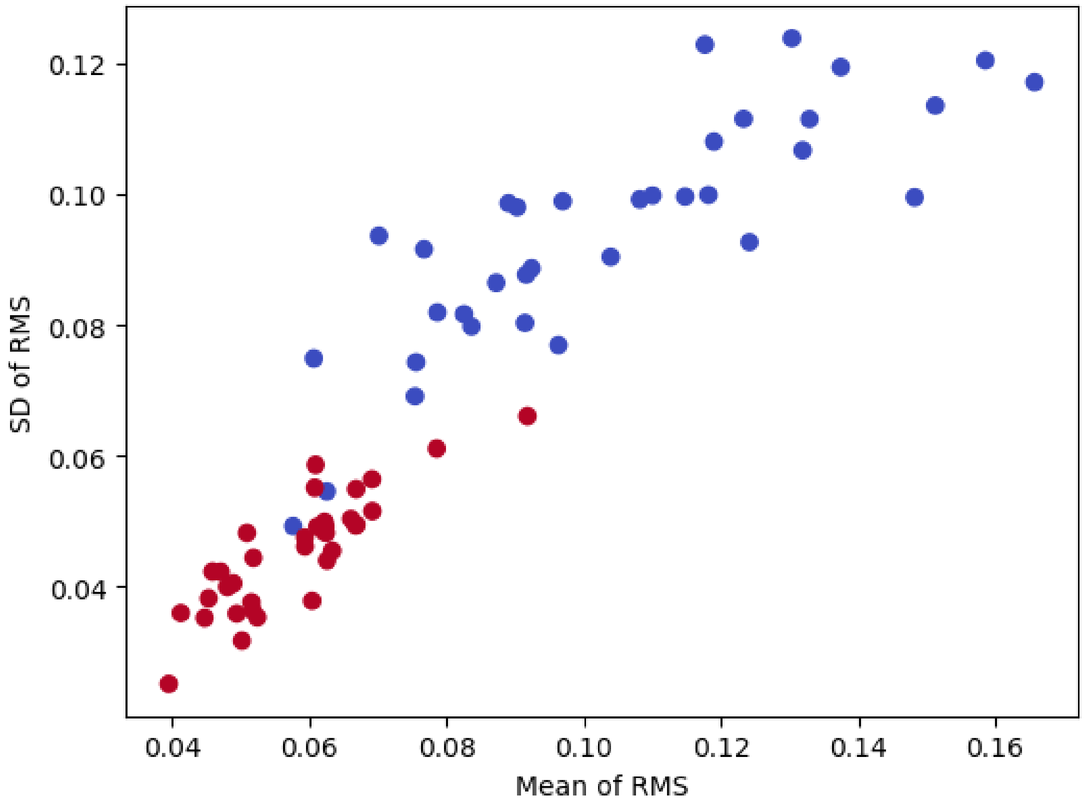
```
Cross-validation scores: [1.         1.         1.         1.
0.92857143]
Mean accuracy: 0.9857142857142858
Standard deviation: 0.028571428571428556
```

```
[ ]: #using another model to classify the data
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.datasets import make_classification

     clf = RandomForestClassifier(max_depth=2, random_state=0)
     clf.fit(x_train, y_train)
     y_pred = clf.predict(x_test)
     print('Accuracy of random forest classifier on test set: {:.2f}'.format(clf.
      ↪score(x_test, y_test)))

     #plot the confusion matrix
     from sklearn.metrics import confusion_matrix
     import matplotlib.pyplot as plt
     import seaborn as sns
     import numpy as np

     cm = confusion_matrix(y_test, y_pred)
     print(cm)
```

```
Accuracy of random forest classifier on test set: 0.97
[[14  1]
 [ 0 15]]
```

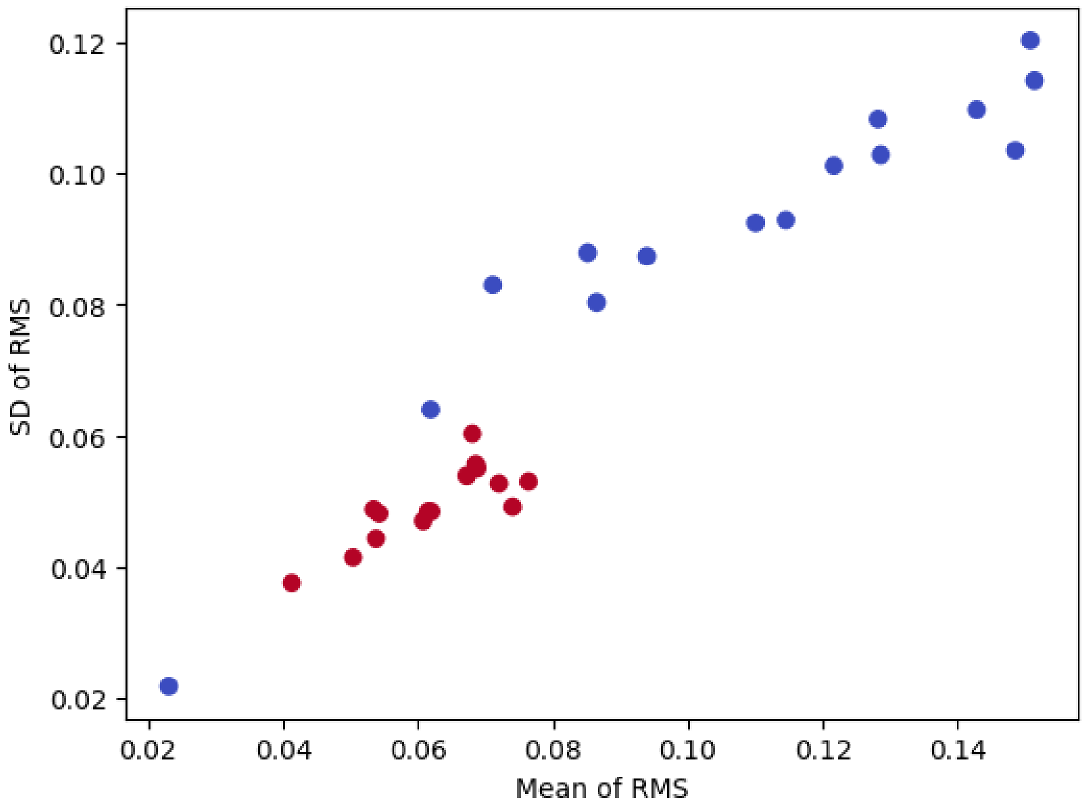
9.8 Visual Aid 2

[ ]:
```
# plot the scatter plot of the data using only the first two features and add␣
  ↪legend for the classes
plt.scatter(x_train[:,0], x_train[:,1], c=y_train, cmap=plt.cm.coolwarm)
plt.xlabel('Mean of RMS')
plt.ylabel('SD of RMS')
plt.show()
```



We can see how the training data is separable across these two classes. And if we look at the same plot for the test set data:

[ ]:
```
# plot the scatter plot of the data using only the first two features and add␣
  ↪legend for the classes
plt.scatter(x_test[:,0], x_test[:,1], c=y_test, cmap=plt.cm.coolwarm)
plt.xlabel('Mean of RMS')
plt.ylabel('SD of RMS')
plt.show()
```



We can see how the 3 incorrectly predicted data points that we saw on the confusion matrix, belong to one of the classes. If we increase the size of our dataset, we can know if more of the data points end up in the place where the 3 false positives, if so we need a better model using either different features that can separate the data points or a deep learning model that we feed the full data to it, without feature engineering and it finds the features through its learning.

9.9 Performance Metrics

Depending on the use of this model, we may care about different metrics more than others. For instance, if we were a bank that uses audio recognition for authentication, this model would not have been a desired model since in that case, we'd want to minimize the false positives (maximizing precision) since we do not want persons who have a similar voice to be able to access our users' bank accounts. But if we were to create an assistant like Siri, we'd care less about minimizing the False Positives since it will be less costly for us, and we'd prioritize minimizing the false negatives (maximizing recall) when we call the assistant multiple times and it does not notice it, especially if one has a cold and their voice has changed a bit, they'll get frustrated much more easily,

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

TP = True positive

TN = True negative

FP = False positive

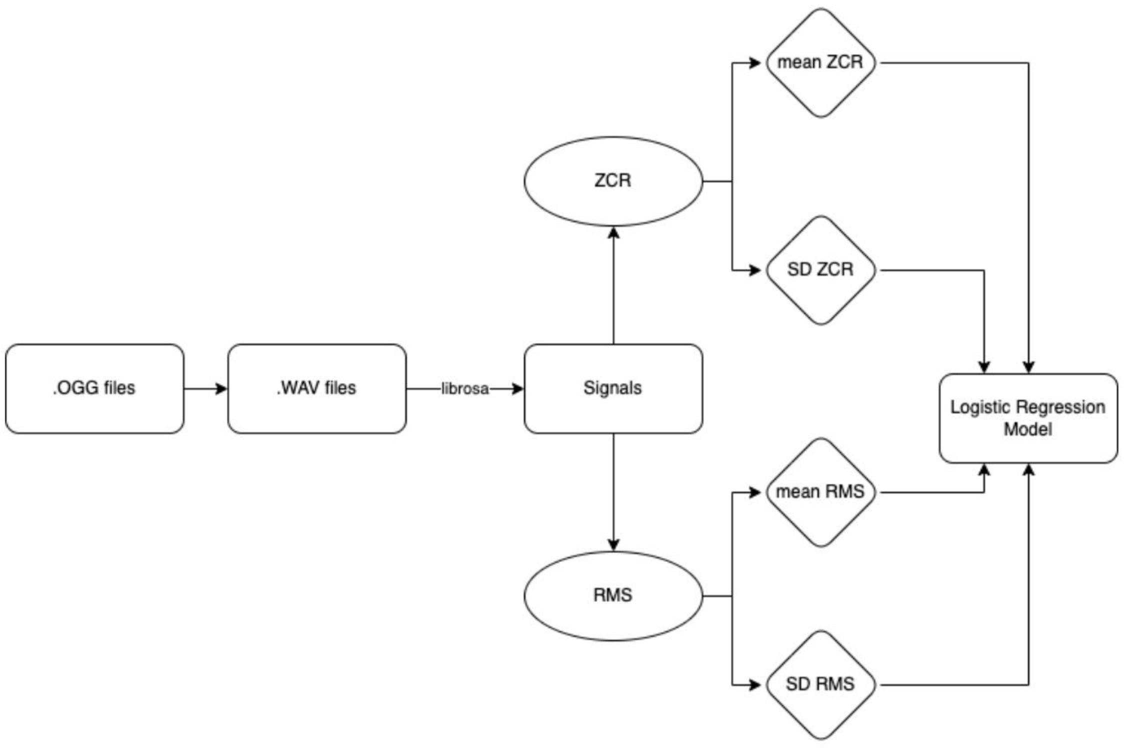
FN = False negative

As we can see based on the results of our model, it is a better model for the second type of problem when FN are minimized. However, thinking of another side of this problem. One can argue that the model is not good for the assistant. If our problem has an additional constraint where we want the model to speak a certain language for either of the classes, depending on whether the two users can understand both languages, our evaluation of the model alters. In this notebook, my data have been assigned as positive or 1 and my mother's data has been assigned as negative or 0. If I can understand both Persian and English, meaning I do not have a strong preference for either language being used by the assistant while my mother has a strong for one of the languages, then it is important for the model to be able to detect my mother correctly most of the times rather than me. Because if it responds in either language, I am ok with it while my mother may not be. Thus in that case, if we keep the positive and negative classes the same, we want to minimize the false positive meaning we do not want the model to detect that it is me by mistake when it is actually me mom and in turn, respond in English.

Another argument is having the model always respond in Persian that we both understand but we are assuming that theoriginal response is in English and it has to be translated to Persian which has a cost we'd like to avoid if possible.

```
[ ]: #calculating precisio, recall and f1 score

     print('Precision: {:.2f}'.format(precision_score(y_test, y_pred)))
     print('Recall: {:.2f}'.format(recall_score(y_test, y_pred)))
     print('F1: {:.2f}'.format(f1_score(y_test, y_pred)))
```

```
Precision: 0.94

Recall: 1.00

F1: 0.97
```



10 Multi-Layer Preceptron using Micrograd

- The process is described after the code.

I used the 4 features I used in the first assignment here for a multilayer perceptron. Visualized the network, and manually covered the backpropagation process using the definition of the chain rule.

```
[ ]: %pip install micrograd
```

```
Requirement already satisfied: micrograd in
/opt/homebrew/Caskroom/miniforge/base/envs/mlp/lib/python3.8/site-packages
(0.1.0)
Note: you may need to restart the kernel to use updated packages.
```

The micrograd package helps show how the backpropagation process work. When building an equation with `Values`, the equation is considered a Directed Acyclic Graph (DAG). The gradients of the nodes are computed automatically and backpropagated through the graph.

10.1 Over-writing Micrograd

```
[ ]: import math
     class Value:
       """
       A value with a gradient.

       Attributes:
         data: The value.
         grad: The gradient.
         _backward: A function that computes the gradient of this value with
         respect to its inputs.
         _prev: A set of values that this value depends on.
         _op: The operation that produced this value.
         label: A label for this value.
         """

       def __init__(self, data, _children=(), _op='', label=''):
         """
         Args:
           data: The value.
           _children: A tuple of values that this value depends on.
           _op: The operation that produced this value.
           label: A label for this value.

         Returns:
           A value with a gradient.
         """
         self.data = data
         self.grad = 0.0
         self._backward = lambda: None
         self._prev = set(_children)
         self._op = _op
         self.label = label

       def __repr__(self):
         """
         Returns:
           A string representation of this value.
         """
         return f"Value(data={self.data})"

       def __add__(self, other):
         """
         Args:
           other: A value.

         Returns:
           A value representing the sum of this value and other.
           """
         out = Value(self.data + other.data, (self, other), '+')

         def _backward():
           """
           Computes the gradient of this value with respect to its inputs.
```

```
      Returns:
        None.
      """
      self.grad += 1.0 * out.grad
      other.grad += 1.0 * out.grad
    out._backward = _backward

    return out

  def __mul__(self, other):
    """
    Args:
      other: A value.

    Returns:
      A value representing the product of this value and other.
    """
    out = Value(self.data * other.data, (self, other), '*')

    def _backward():
      """
      Computes the gradient of this value with respect to its inputs.

      Returns:
        None.
      """
      self.grad += other.data * out.grad
      other.grad += self.data * out.grad
    out._backward = _backward

    return out

  def tanh(self):
    """
    Returns:
      A value representing the hyperbolic tangent of this value.
    """
    x = self.data
    t = (math.exp(2*x) - 1)/(math.exp(2*x) + 1)
    out = Value(t, (self, ), 'tanh')

    def _backward():
      self.grad += (1 - t**2) * out.grad
    out._backward = _backward

    return out
  def backward(self):
    """
    Computes the gradient of this value with respect to its inputs.

    Returns:
      None.
    """

    topo = []
    visited = set()
    def build_topo(v):
      if v not in visited:
        visited.add(v)
        for child in v._prev:
          build_topo(child)
        topo.append(v)
    build_topo(self)

    self.grad = 1.0
    for node in reversed(topo):
      node._backward()
```

The code above defines a simple automatic differentiation system for mathematical operations using a class called Value. The main purpose of this system is to compute gradients of expressions with respect to their inputs. It supports addition, multiplication, and the tanh (hyperbolic tangent) function. This is achieved using a class with a custom backward method for each operation.

data: The value of the node. grad: The gradient of the node, initially set to 0.0. _backward: A function that computes the gradients of this node with respect to its parents. Initially, it's an empty function. _prev: A set of parent nodes that are connected to this node. _op: The operation that produced this node. It's an empty string by default. label: A label for the node. It's an empty string by default.

[ ]:
```
from micrograd.nn import Neuron, Layer, MLP
import random



# Passing the feature vectrors of mom and me audio files to the model
xs = [
  feature_vec_gen('./converted_me/sec_13.wav'),
  feature_vec_gen('./converted_me/sec_22.wav'),
  feature_vec_gen('./converted_mom/sec_2.wav'),
  feature_vec_gen('./converted_mom/sec_20.wav'),
]
ys = [1, 1, -1, -1] # desired targets (1s me, -1s mom)

print (xs[0],"\n", xs[1],"\n", xs[2],"\n", xs[3])
```

```
[0.04541743 0.03822615 0.06952448 0.09237615]
 [0.06806925 0.06027614 0.07607672 0.09592157]
 [0.0230393  0.02178216 0.10983198 0.10381434]
 [0.09246823 0.08867332 0.10919416 0.09565437]
```

10.2 Back-propogation (Manual and Visual)

Using 4 random datapoints since the visualization gets overcroweded with more layers and inputs.

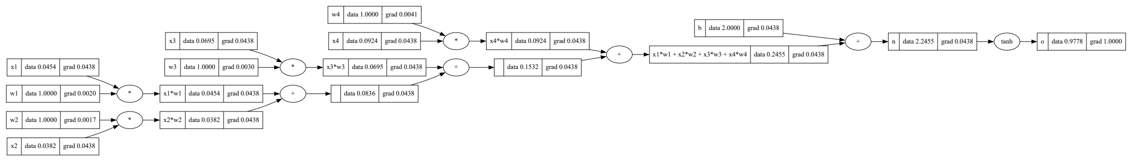
[ ]:
```python
from graphviz import Digraph

def trace(root):
  """
  Args:
    root: A value.

  Returns:
    A tuple of sets of values. The first set contains all values in the graph
    rooted at root. The second set contains all edges in the graph rooted at
    root.
  """
  # builds a set of all nodes and edges in a graph
  nodes, edges = set(), set()
  def build(v):
    if v not in nodes:
      nodes.add(v)
      for child in v._prev:
        edges.add((child, v))
        build(child)
  build(root)
  return nodes, edges

def draw_dot(root):
  """
  Args:
    root: A value.

  Returns:
    A graphviz Digraph object representing the graph rooted at root.
    """
  dot = Digraph(format='svg', graph_attr={'rankdir': 'LR'}) # LR = left to right

  nodes, edges = trace(root)
  for n in nodes:
    uid = str(id(n))
    # for any value in the graph, create a rectangular ('record') node for it
    dot.node(name = uid, label = "{ %s | data %.4f | grad %.4f }" % (n.label, n.
↪data, n.grad), shape='record')
```

```
    if n._op:
      # if this value is a result of some operation, create an op node for it
      dot.node(name = uid + n._op, label = n._op)
      # and connect this node to it
      dot.edge(uid + n._op, uid)

  for n1, n2 in edges:
    # connect n1 to the op node of n2
    dot.edge(str(id(n1)), str(id(n2)) + n2._op)

  return dot
```

[ ]:
```
# inputs x1,x2, x3, x4
x1 = Value(xs[0][0], label='x1')
x2 = Value(xs[0][1], label='x2')
x3 = Value(xs[0][2], label='x3')
x4 = Value(xs[0][3], label='x4')
# weights w1,w2, w3, w4
w1 = Value(1.0, label='w1')
w2 = Value(1.0, label='w2')
w3 = Value(1.0, label='w3')
w4 = Value(1.0, label='w4')
# bias of the neuron
b = Value(2, label='b')
# x1*w1 + x2*w2 +x3w3 + x4w4 + b
x1w1 = x1*w1; x1w1.label = 'x1*w1'
x2w2 = x2*w2; x2w2.label = 'x2*w2'
x3w3 = x3*w3; x3w3.label = 'x3*w3'
x4w4 = x4*w4; x4w4.label = 'x4*w4'
x1w1x2w2x3w3x4w4 = x1w1 + x2w2 +x3w3 + x4w4; x1w1x2w2x3w3x4w4.label = 'x1*w1 +␣
 ↪x2*w2 + x3*w3 + x4*w4'

n = x1w1x2w2x3w3x4w4 + b; n.label = 'n'
o = n.tanh(); o.label = 'o'
```

[ ]:
```
draw_dot(o)
```

[ ]:



w4 | data 1.0000 | grad 0.0041
x3 | data 0.0695 | grad 0.0438
x4 | data 0.0924 | grad 0.0438
*
x4*w4 | data 0.0924 | grad 0.0438
b | data 2.0000 | grad 0.0438
+
x1*w1 + x2*w2 + x3*w3 + x4*w4 | data 0.2455 | grad 0.0438
+
n | data 2.2455 | grad 0.0438
tanh
o | data 0.9778 | grad 1.0000
x1 | data 0.0454 | grad 0.0438
w3 | data 1.0000 | grad 0.0030
*
x3*w3 | data 0.0695 | grad 0.0438
+
data 0.1532 | grad 0.0438
w1 | data 1.0000 | grad 0.0020
*
x1*w1 | data 0.0454 | grad 0.0438
+
data 0.0836 | grad 0.0438
w2 | data 1.0000 | grad 0.0017
*
x2*w2 | data 0.0382 | grad 0.0438
x2 | data 0.0382 | grad 0.0438

Setting the gradient of the final node to 1 since it is always 1 for the output node. But the code sets all the nodes to 0. Then using the backward fucntion on it to get the grad of the prev nodes to the final node, set. Then, we run the `draw_dot` function again to get the new gradient corrected visual as you cna see above.

```
[ ]: o.grad =1
     o.backward()
```

10.3 Training (Manual)

```
[ ]: n = MLP(4, [4, 4, 1]) # 4 inputs, 2 hidden layer with 4 neurons each, 1 output␣
      ↪neuron

     for k in range(20):

       # forward pass
       ypred = [n(x) for x in xs]
       loss = sum((yout - ygt)**2 for ygt, yout in zip(ys, ypred))

       # backward pass
       for p in n.parameters():
         p.grad = 0.0 # zero out gradients
       loss.backward()

       # update
       for p in n.parameters():
         p.data += -0.01 * p.grad

       print(k, loss.data)
```

```
0 4.057155457618525
1 3.8362079605437334
2 3.7450182600965745
3 3.7202837215865734
4 3.710161690054942
5 3.7074592740484382
6 3.697123397955859
7 3.693467483605673
8 3.691081671217237
9 3.6858430038746217
10 3.683728437440198
11 3.682875239002752
12 3.691350731948031
13 3.6912672111906017
14 3.679689004073996
15 3.6907522795697156
16 3.672707050633479
17 3.6703968343944977
18 3.668358506398322
19 3.6664319918833397
```

The loss is decreasing but not enough. Althought the model's prediction is correct, the distinctions are not far apart. This can be improved by providing more data or more iterations.

10.4 Comparing the MLP's results

- Comparing the MLP's prediction results with the Logistic Regression

[ ]:
```
ypred
```

[ ]:
```
[Value(data=0.04318542277796241, grad=-1.913629154444075),
 Value(data=0.030666696630065493, grad=-1.938666606739869),
 Value(data=-0.06778723538906847, grad=1.864425529221863),
 Value(data=-0.029273382886238042, grad=1.941453234227524)]
```

[ ]:
```
#using numbers close to these for the weights to have a start close to the␣
 ↪correct values for the weights
print(coefficients)
```

```
[[  0.         -82.86426885 -21.78098338   0.        ]]
```

The predictions and coeffients are mostly consistent with the results from the previous models in assignment one.

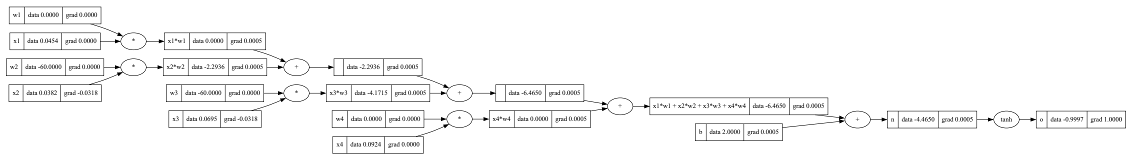
10.5 Different Starting point of optimization

[ ]:
```
# inputs x1,x2, x3, x4
x1 = Value(xs[0][0], label='x1')
x2 = Value(xs[0][1], label='x2')
x3 = Value(xs[0][2], label='x3')
x4 = Value(xs[0][3], label='x4')
# weights w1,w2
w1 = Value(0, label='w1')
w2 = Value(-60, label='w2')
w3 = Value(-60, label='w3')
w4 = Value(0, label='w4')
# bias of the neuron
b = Value(2, label='b')
# x1*w1 + x2*w2 + b
x1w1 = x1*w1; x1w1.label = 'x1*w1'
x2w2 = x2*w2; x2w2.label = 'x2*w2'
x3w3 = x3*w3; x3w3.label = 'x3*w3'
x4w4 = x4*w4; x4w4.label = 'x4*w4'
x1w1x2w2x3w3x4w4 = x1w1 + x2w2 +x3w3 + x4w4; x1w1x2w2x3w3x4w4.label = 'x1*w1 +␣
 ↪x2*w2 + x3*w3 + x4*w4'

n = x1w1x2w2x3w3x4w4 + b; n.label = 'n'
o = n.tanh(); o.label = 'o'
```

Setting new weights based on the coeffients suggested by assignment one's model. Log-reg had the coeffient : [[0. -82.86426885 -21.78098338 0.]]

[ ]:
```
draw_dot(o)
```

[ ]:


w1 | data 0.0000 | grad 0.0000
x1 | data 0.0454 | grad 0.0000
w2 | data -60.0000 | grad 0.0000
x2 | data 0.0382 | grad -0.0318
x1*w1 | data 0.0000 | grad 0.0005
x2*w2 | data -2.2936 | grad 0.0005
w3 | data -60.0000 | grad 0.0000
x3 | data 0.0695 | grad -0.0318
data -2.2936 | grad 0.0005
x3*w3 | data -4.1715 | grad 0.0005
w4 | data 0.0000 | grad 0.0000
x4 | data 0.0924 | grad 0.0000
data -6.4650 | grad 0.0005
x4*w4 | data 0.0000 | grad 0.0005
x1*w1 + x2*w2 + x3*w3 + x4*w4 | data -6.4650 | grad 0.0005
b | data 2.0000 | grad 0.0005
n | data -4.4650 | grad 0.0005
tanh
o | data -0.9997 | grad 1.0000

[ ]:
```
o.grad = 1
o.backward()
```

[ ]:
```
n = MLP(4, [4, 4, 1])

for k in range(20):

  # forward pass
  ypred = [n(x) for x in xs]
  loss = sum((yout - ygt)**2 for ygt, yout in zip(ys, ypred))

  # backward pass
  for p in n.parameters():
    p.grad = 0.0
  loss.backward()

  # update
  for p in n.parameters():
    p.data += -0.01 * p.grad

  print(k, loss.data)
```

```
0 3.9845248556878086
1 3.9790439949893406
2 3.9737839351468676
3 3.9549019466968858
4 3.9382533988841604
5 3.9230374745426593
6 3.92076515141445
7 3.919603758338737
8 3.918540598021292
9 3.917357366049172
10 3.916193533173207
11 3.9150409822306624
12 3.913894325932201
13 3.9127499388066287
```

```
14 3.9116053308235523
15 3.9104587423638124
16 3.909308882472898
17 3.9081547597824224
18 3.9069955733065047
19 3.9058306418843225
```

10.6 The new Loss and performance

[ ]:
```
ypred
```

[ ]:
```
[Value(data=0.009974008632489 14, grad=-1.9800519827350218),
 Value(data=0.005279761663731761, grad=-1.9894404766725364),
 Value(data=-0.017954488935619555, grad=1.9640910221287609),
 Value(data=-0.01420213028043251, grad=1.971595739439135)]
```

Here, we start by less loss compared to the previous model as the weights were adjusted towards what would work for the larger model. But still due to the limits of the model, input data and the iterations the results are not that good. However, this does not suggest that MLPs do not work good but MLPs can work greatly given more iterations, layers, and data. Here the point has been showing the visualization and the backpropogation process thus I have kept these values low which has effected the model's performance.

10.7 The Code's mathematical walk through

Backpropagation is an optimization algorithm used to minimize the loss function with the steps below: - Forward pass: Feed the input data through the network to generate predictions. For each neuron, calculate the weighted sum of inputs and pass it through the activation function. - Compute the error (loss): Calculate the difference between the predicted outputs and the actual outputs (targets). The one used here is the sum of squared differences where:

```
- ``yout`` - The predicted output (also known as the predicted label or ŷ) from the neural netw

- ``ygt`` - The actual output (also known as the ground truth, target label, or y) from the dat
```

- Backward pass: Calculate the gradients (partial derivatives) of the loss function with respect to each weight and bias in the network, starting from the output layer and moving backward through the network. This is done using the chain rule of calculus:

$$\Delta w_{ij} = -\eta \times \frac{\partial L}{\partial w_{ij}}$$

$$\Delta b_{ij} = -\eta \times \frac{\partial L}{\partial b_{ij}}$$

where \$ Δw_{ij}\$ and \$ Δb_{ij}\$ are the updates to weights and biases, η is the learning rate, and $\frac{\partial L}{\partial w_{ij}}$ and $\frac{\partial L}{\partial b_{ij}}$ are the partial derivatives of the loss function with respect to the weights and biases.

- Update the weights and biases:

Using the computed gradients, update the weights and biases of the network (the chain rule is used to calculate the gradients of the loss function with respect to the weights and biases of the neural network based on the operation and how partial differentiation is defined for that operation.)

$$w_{ij} = w_{ij} + \Delta w_{ij}$$
$$b_{ij} = b_{ij} + \Delta b_{ij}$$

- Iterate the process above: until the network converges to a minimum loss value, or a predefined stopping criterion is met.

11 Mel Spectrogram

11.1 Description

A Mel spectrogram is a visual representation of the audio signal that emphasizes the human perception of sound frequencies over a given time period. It is derived by transforming the audio signal into the frequency domain using a combination of the Short-Time Fourier Transform (STFT) and the Mel-scale. The below steps are how the Mel-spectrogram is created: - Preprocessing: The audio signal is typically sampled at a specific rate and then pre-emphasized to accentuate higher frequencies, which can improve the performance of some audio processing algorithms.

- Windowing: The audio signal is divided into overlapping frames, usually with a length of 20-30 ms and an overlap of 50% or more. Each frame is multiplied by a window function, to minimize the effects of discontinuities at the edges of the frame.
- Short-Time Fourier Transform (STFT): Applied to each windowed frame to convert the audio signal from the time domain to the frequency domain. (A transformation to a matrix with complex values, where the rows represent different time frames, and the columns represent different frequency bins.)
- Magnitude spectrogram: The magnitude of the above matrix is computed, resulting in real values in the matrix which is called the magnitude spectrogram. (Represents the power of the audio signal at different frequencies over time.)
- Mel-scale: The Mel-scale is a scale of pitches that is close to the human auditory system's understanding of different frequencies. (Humans hear pitch differences linearly at low frequencies and logarithmically at high frequencies.)
  - The Mel-scale is defined as below where f is the frequency in Hertz

$$Mel(f) = 2595 \times \log_{10}(1 + \frac{f}{700})$$

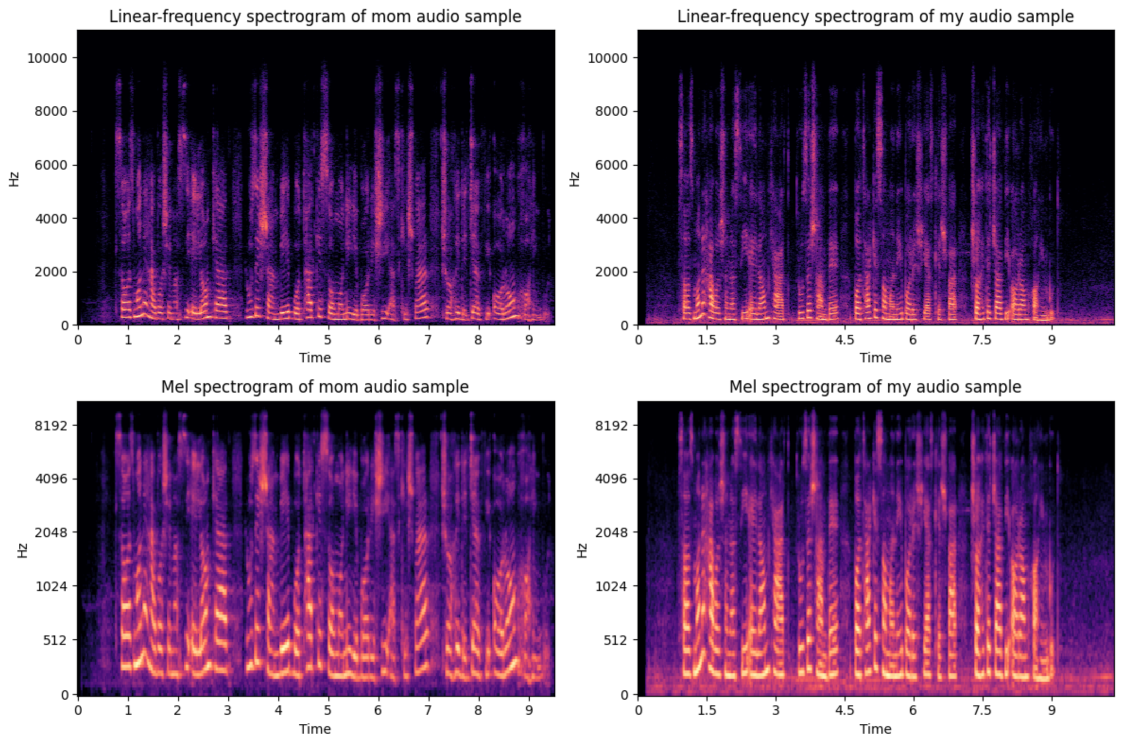
- Mel filterbank: An array of interlocking triangular filters is constructed to mimic the Mel-scale. These filters are uniformly distributed along the Mel-scale and encompass the full frequency scope of the magnitude spectrogram. Every filter is responsible for capturing the energy of the audio signal within a distinct frequency range.
- Utilizing the Mel filterbank: The magnitude spectrogram undergoes multiplication by the Mel filterbank, yielding a new matrix that symbolizes the energy of the audio signal in each Mel-adjusted frequency band as time passes. This matrix is referred to as the Mel-adjusted spectrogram.

- Logarithm: Calculating the logarithm of the Mel-scaled spectrogram emulates the human ear's logarithmic sensitivity to sound intensity. This procedure also decreases the dynamic range of the spectrogram values, simplifying visualization and analysis.
- Visualization: The resulting matrix can be displayed as a graphic, where time is represented along the x-axis, Mel-scaled frequency bands along the y-axis, and color intensity illustrates the logarithmic energy of the audio signal in each Mel-scaled frequency band.

11.2 Mel vs. Linear frequency

[ ]:
```python
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np

# Load two audio files
audio1, sr1 = librosa.load(mom_sample_path)
audio2, sr2 = librosa.load(me_sample_path)

# Compute the short-time Fourier transform (STFT) for linear frequency␣
 ↪representation
D1 = librosa.stft(audio1)
D2 = librosa.stft(audio2)

# Convert STFT to decibels for visualization
D_db1 = librosa.amplitude_to_db(np.abs(D1), ref=np.max)
D_db2 = librosa.amplitude_to_db(np.abs(D2), ref=np.max)

# Compute Mel spectrograms
mel_spectrogram1 = librosa.feature.melspectrogram(y=audio1, sr=sr1)
mel_spectrogram2 = librosa.feature.melspectrogram(y=audio2, sr=sr2)

# Convert Mel spectrograms to decibels for visualization
mel_spectrogram_db1 = librosa.power_to_db(mel_spectrogram1, ref=np.max)
mel_spectrogram_db2 = librosa.power_to_db(mel_spectrogram2, ref=np.max)

# Create subplots for side by side visualization
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Plot the first linear frequency spectrogram
librosa.display.specshow(D_db1, sr=sr1, x_axis='time', y_axis='linear',␣
 ↪ax=axes[0, 0])
axes[0, 0].set(title='Linear-frequency spectrogram of mom audio sample')

# Plot the second linear frequency spectrogram
librosa.display.specshow(D_db2, sr=sr2, x_axis='time', y_axis='linear',␣
 ↪ax=axes[0, 1])
axes[0, 1].set(title='Linear-frequency spectrogram of my audio sample')
```

```
# Plot the first Mel spectrogram
librosa.display.specshow(mel_spectrogram_db1, sr=sr1, x_axis='time',␣
 ↪y_axis='mel', ax=axes[1, 0])
axes[1, 0].set(title='Mel spectrogram of mom audio sample')

# Plot the second Mel spectrogram
librosa.display.specshow(mel_spectrogram_db2, sr=sr2, x_axis='time',␣
 ↪y_axis='mel', ax=axes[1, 1])
axes[1, 1].set(title='Mel spectrogram of my audio sample')

# Display the plots
plt.tight_layout()
plt.show()
```



why we use logarithmic scale? Logarithmic scaling in Mel spectrograms is employed primarily for two reasons: to mimic the way humans perceive sound and to reduce the dynamic range of the spectrogram values.

Mimicking human perception: Our auditory system processes sound intensity and frequency on a logarithmic scale. As a result, we interpret changes in sound levels and pitches in a proportional manner, rather than linearly. For instance, when the frequency of a sound doubles, we perceive the change as a consistent pitch interval (e.g., an octave) regardless of the starting frequency. Likewise, we perceive sound intensity (loudness) logarithmically, with a 10 dB increase equating to a perceived doubling in loudness. Incorporating a logarithmic scale into Mel spectrograms generates a representation that better reflects the way humans perceive and interpret sound.

Reducing dynamic range: Audio signals often exhibit a wide dynamic range in energy levels, with certain frequency components being much more pronounced than others. Dealing with the linear magnitude of a spectrogram can make it challenging to effectively visualize and process these energy differences. By utilizing a logarithmic transformation, the dynamic range of spectrogram values is compressed, facilitating simpler visualization and more efficient processing. This logarithmic compression also highlights more nuanced features in the spectrogram, which can be crucial for different audio analysis tasks, such as speech recognition or music genre classification.

11.3 Data Split and file managment Pipeline

Creating Mel-Spectrograms and splitting the data in a balanced manner. (included multiple functions for different steps of the pipeline)

[ ]:
```
# import librosa
# import librosa.display
# import matplotlib.pyplot as plt
# import numpy as np
# import os

def create_mel_spectrograms(directory_path, output_path, type_name=None,␣
 ↪func=None, inp=None):
    """
    Create Mel spectrograms from audio files in a directory and save them to a␣
 ↪new directory.

    Args:
        directory_path (str): Path to the directory containing the audio files
        output_path (str): Path to the directory where the Mel spectrograms␣
 ↪will be saved
        type_name (str): Name of the type of audio files (e.g., 'mom', 'me',␣
 ↪'other')
        func (function): Function to apply to the audio files
        inp (list): List of inputs to the function

    Returns:
        None
    """

    # Iterate over all files in the directory
    for filename in os.listdir(directory_path):
        # Check if the file is an audio file (e.g., .wav extension)
        if filename.endswith('.wav'):
            # Load the audio file
            audio, sr = librosa.load(os.path.join(directory_path, filename))
            if func:
                audio = func(audio, *inp)
            # Compute Mel spectrogram
            mel_spectrogram = librosa.feature.melspectrogram(y=audio, sr=sr)

            # Convert Mel spectrogram to decibels for visualization
            mel_spectrogram_db = librosa.power_to_db(mel_spectrogram, ref=np.
 ↪max)

            # Plot the Mel spectrogram
            plt.figure(figsize=(10, 4))
            librosa.display.specshow(mel_spectrogram_db, sr=sr, x_axis='time',␣
 ↪y_axis='mel')
            if type_name:
                plt.title(f'Mel spectrogram of {type_name} {filename}')
            else:
                plt.title(f'Mel spectrogram of {filename}')

            # Save the Mel spectrogram image
            if type_name:
                output_filename = os.path.splitext(filename)[0] +␣
 ↪f'{type_name}' + '.png'
            else:
                output_filename = os.path.splitext(filename)[0] + '.png'
            plt.savefig(os.path.join(output_path, output_filename))

            # Close the figure to free memory
            plt.close()

# #Usage on mom audios
# input_directory = mom_paths
# output_directory = './mom_spectrograms'
# create_mel_spectrograms(input_directory, output_directory)

# #Usage on my audios
# input_directory = me_paths
# output_directory = './me_spectrograms'
# create_mel_spectrograms(input_directory, output_directory)
```

[ ]:
```
import os
import random
import shutil

def split_train_test(input_path, train_ratio, output_path):
    """
    Split the audio files in a directory into training and test sets and save␣
 ↪them to new directories.
```

```
    Args:

        input_path (str): Path to the directory containing the audio files
        train_ratio (float): Ratio of training files to total files
        output_path (str): Path to the directory where the training and test␣
 ↪sets will be saved

    Returns:
        None
    """
    # List all files in the input directory
    files = [f for f in os.listdir(input_path) if os.path.isfile(os.path.
 ↪join(input_path, f))]

    # Shuffle the files
    random.shuffle(files)

    # Calculate the number of training files based on the specified ratio
    num_train_files = int(len(files) * train_ratio)

    # Split the files into training and test sets
    train_files = files[:num_train_files]
    test_files = files[num_train_files:]

    # Create the output directories if they do not exist
    train_output_path = os.path.join(output_path, 'train')
    test_output_path = os.path.join(output_path, 'test')
    os.makedirs(train_output_path, exist_ok=True)
    os.makedirs(test_output_path, exist_ok=True)

    # Copy the training files to the training output directory
    for f in train_files:
        shutil.copy(os.path.join(input_path, f), os.path.
 ↪join(train_output_path, f))

    # Copy the test files to the test output directory
    for f in test_files:
        shutil.copy(os.path.join(input_path, f), os.path.join(test_output_path,␣
 ↪f))

    return train_output_path, test_output_path

# # Example usage
# input_directory = './mom_spectrograms'
# output_directory = './mom_split_data'
# train_percentage = 0.8  # 80% of the data used for training
# train_path, test_path = split_train_test(input_directory, train_percentage,␣
 ↪output_directory)

# # Example usage
# input_directory = './me_spectrograms'
# output_directory = './me_split_data'
# train_percentage = 0.8  # 80% of the data used for training

# train_path, test_path = split_train_test(input_directory, train_percentage,␣
 ↪output_directory)
```

```
[ ]: import os
     import shutil

     def merge_folders(input_path1, input_path2, output_path):
         """
         Merge the "train" and "test" subfolders from two directories into a single␣
      ↪directory.

         Args:
             input_path1 (str): Path to the first directory containing the "train"␣
      ↪and "test" subfolders
             input_path2 (str): Path to the second directory containing the "train"␣
      ↪and "test" subfolders
             output_path (str): Path to the directory where the merged "train" and␣
      ↪"test" subfolders will be saved

         Returns:
             None
         """
         # Define the "train" and "test" subfolder names
         subfolders = ['train', 'test']

         # Create the output directories if they do not exist
         for subfolder in subfolders:
             os.makedirs(os.path.join(output_path, subfolder), exist_ok=True)

         # Function to copy files from the input subfolder to the output subfolder␣
      ↪and rename based on source folder
         def copy_files(src_path, dest_path, src_folder_name):
             for filename in os.listdir(src_path):
                 # Generate a new filename based on the source folder name
                 new_filename = f"{src_folder_name}_{filename}"
                 # Copy the file to the destination directory with the new filename
                 shutil.copy(os.path.join(src_path, filename), os.path.
      ↪join(dest_path, new_filename))
```

```
    # Merge the "train" and "test" subfolders from both input paths
    for subfolder in subfolders:
        # Copy and rename files from the "train" or "test" subfolder of the␣
 ↪first input path
        copy_files(os.path.join(input_path1, subfolder), os.path.
 ↪join(output_path, subfolder), os.path.basename(input_path1))

        # Copy and rename files from the "train" or "test" subfolder of the␣
 ↪second input path
        copy_files(os.path.join(input_path2, subfolder), os.path.
 ↪join(output_path, subfolder), os.path.basename(input_path2))

# # Example usage
# input_directory1 = './mom_split_data'  # Contains "train" and "test"␣
 ↪subfolders
# input_directory2 = './me_split_data'  # Contains "train" and "test" subfolders
# output_directory = './general_data_split'  # The merged "train" and "test"␣
 ↪subfolders will be created here

# merge_folders(input_directory1, input_directory2, output_directory)
```

[ ]:
```
import os

def get_filenames(directory_path):
    """
    Get the filenames of all files in a directory.

    Args:
        directory_path (str): Path to the directory containing the files

    Returns:
        filenames (list): List of filenames
    """
    # Initialize an empty list to store filenames
    filenames = []

    # Iterate over all files and directories in the specified directory
    for entry in os.listdir(directory_path):
        # Construct the full path of the entry
        entry_path = os.path.join(directory_path, entry)

        # Check if the entry is a file (as opposed to a directory or a symbolic␣
 ↪link)
        if os.path.isfile(entry_path):
            # Add the filename to the list
            if entry[:3] == 'mom':
                filenames.append(entry)

    return filenames

# # Example usage
# directory_path = '/./general_data_split/test'
# filenames_test = get_filenames(directory_path)

# directory_path = '/./general_data_split/train'
# filenames_train = get_filenames(directory_path)

# filenames = filenames_train + filenames_test
# print(filenames)
```

[ ]:
```
# import os
# import shutil

def organize_images(input_path, class_0_filenames, output_path):
    """
    Organize images into subdirectories based on their class.

    Args:
        input_path (str): Path to the directory containing the images
        class_0_filenames (list): List of filenames for class 0 images
        output_path (str): Path to the directory where the images will be␣
 ↪organized into subdirectories

    Returns:
        None
    """
    # Create subdirectories for class 0 and class 1 in the output directory
    class_0_path = os.path.join(output_path, 'class_0')
    class_1_path = os.path.join(output_path, 'class_1')
    os.makedirs(class_0_path, exist_ok=True)
    os.makedirs(class_1_path, exist_ok=True)

    # Iterate through all files in the input directory
    for filename in os.listdir(input_path):
        # Get the full path of the file
        file_path = os.path.join(input_path, filename)
        # Check if the file is one of the class 0 images
        if filename in class_0_filenames:
            # Move the file to the class_0_path
            shutil.copy(file_path, class_0_path)
        else:
            # Move the file to the class_1_path
            shutil.copy(file_path, class_1_path)

# # Example usage
# input_directory = './general_data_split/test'
# class_0_files = filenames  # List of filenames corresponding to class 0
# output_directory = './dataset/test'

# organize_images(input_directory, class_0_files, output_directory)

# # Example usage
# input_directory = './general_data_split/train'
# class_0_files = filenames  # List of filenames corresponding to class 0
# output_directory = './dataset/train'

# organize_images(input_directory, class_0_files, output_directory)

# # Example usage
# input_directory = './general_data_split/train'
# class_0_files = filenames  # List of filenames corresponding to class 0
# output_directory = './dataset/train'

# organize_images(input_directory, class_0_files, output_directory)
```

The above code is commented to not be ran again with every run of all the notebook as it creates files from the data that needs to be created only once, when new data needs to be added to the pipeline.

12 CNN

12.1 Mathematical walk thorugh

Convolutional Neural Networks, or CNNs, represent a category of artificial neural networks explicitly developed to handle data in grid-like formats, including images, audio, and text. These networks play a crucial role in various computer vision tasks, such as identifying images, detecting objects, and segmenting elements. The structure of CNNs is influenced by the organization of the animal brain's visual cortex, where neurons form a hierarchical arrangement to interpret visual data.

CNNs consist of multiple layers including: input, convolutional, activation, pooling, fully connected, and output layers.

- Input layer: The input layer receives an image or grid-like data, which is represented as a 3D tensor (height, width, and depth or channels). For grayscale images, the depth is 1, while for RGB images, it is 3; Mel-spectrograms are grayscale as it is a 2D amtrix but colourmapping is used to make the visuals better.
- Convolutional layer: This layer applies convolution operations to the input using a set of learnable filters or kernels. Each filter slides (convolves) across the input to produce a 2D feature map. The output size (O) is calculated as follows:

$$O = \frac{W - K + 2P}{S + 1}$$

Where:

$$W = \frac{input\ width}{height}$$

$$K = \frac{filter\ width}{height}$$

- P is padding (adding extra pixels around the input to preserve spatial dimensions) - S is stride (how many pixels the filter moves per step)

- Activation layer: The activation function brings in non-linear characteristics to the model, enabling the network to grasp intricate patterns.
- Pooling layer: serving to diminish the spatial proportions of the feature maps, making the model more resilient to minor fluctuations in input and lowering computational load. Max-pooling is the most prevalent pooling technique, which chooses the highest value within a specified area.
- Dense Layer: Following the previous layer, the feature maps are transformed into a one-dimensional array and linked to a fully connected layer. This layer calculates a weighted sum of its inputs and then applies an activation function:
  - Result = ActivationFunction(Σ (Weight_i * Input_i) + Bias)
- Final Layer: The last layer is the output layer, usually employing a softmax activation function to generate class probabilities:
  - Softmax(Input_i) = exp(Input_i) / (Σ(exp(all other input)))

To train a CNN, the weights (filter values) and biases must be adjusted using an optimization technique, such as gradient descent, in order to minimize the loss function. The loss function quantifies the discrepancy between the predicted outputs and the actual labels.

12.2 Training CNN

[ ]:
```
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

# Image dimensions (assuming all Mel spectrogram images have the same␣
  ↪dimensions)
image_height = 128
image_width = 256

# Data augmentation and preprocessing for training data
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)

# Preprocessing for test data
test_datagen = ImageDataGenerator(rescale=1./255)

# Generating training and test data from the respective directories
training_set = train_datagen.flow_from_directory('./dataset/train',
                                                 target_size=(image_height,␣
 ↪image_width),
                                                 batch_size=25,
                                                 class_mode='binary')

test_set = test_datagen.flow_from_directory('./dataset/train',
                                            target_size=(image_height,␣
 ↪image_width),
                                            batch_size=25,
                                            class_mode='binary')

print("Number of training samples:", training_set.samples)
print("Number of test samples:", test_set.samples)

# Defining the CNN model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(image_height,␣
 ↪image_width, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(units=128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=1, activation='sigmoid'))

# Compiling the CNN model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Training the CNN model
model.fit(training_set, epochs=25, validation_data=test_set)

# Saving the trained model
model.save('trained_model.h5')
```

```
Found 80 images belonging to 2 classes.
Found 80 images belonging to 2 classes.
Number of training samples: 80
Number of test samples: 80
Metal device set to: Apple M1

systemMemory: 8.00 GB
maxCacheSize: 2.67 GB


2023-03-26 20:03:45.400687: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2023-03-26 20:03:45.401737: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0
MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
<undefined>)

Epoch 1/25

2023-03-26 20:03:46.074688: W
tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU
frequency: 0 Hz
2023-03-26 20:03:46.332935: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

4/4 [==============================] - ETA: 0s - loss: 2.2396 - accuracy: 0.4750

2023-03-26 20:03:47.894078: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

4/4 [==============================] - 3s 505ms/step - loss: 2.2396 - accuracy:
0.4750 - val_loss: 1.1068 - val_accuracy: 0.5000
Epoch 2/25
4/4 [==============================] - 2s 416ms/step - loss: 0.9450 - accuracy:
0.4375 - val_loss: 0.6897 - val_accuracy: 0.5750
Epoch 3/25
4/4 [==============================] - 2s 517ms/step - loss: 0.6914 - accuracy:
0.6125 - val_loss: 0.6893 - val_accuracy: 0.5000
Epoch 4/25
4/4 [==============================] - 2s 539ms/step - loss: 0.6872 - accuracy:
0.5625 - val_loss: 0.6726 - val_accuracy: 0.5375
Epoch 5/25
4/4 [==============================] - 2s 457ms/step - loss: 0.6788 - accuracy:
0.6000 - val_loss: 0.6557 - val_accuracy: 0.7625
Epoch 6/25
4/4 [==============================] - 2s 553ms/step - loss: 0.6816 - accuracy:
0.6500 - val_loss: 0.6520 - val_accuracy: 0.6000
```

```
Epoch 7/25
4/4 [==============================] - 2s 431ms/step - loss: 0.6462 - accuracy:
0.7750 - val_loss: 0.6180 - val_accuracy: 0.6000
Epoch 8/25
4/4 [==============================] - 2s 488ms/step - loss: 0.6120 - accuracy:
0.6375 - val_loss: 0.5832 - val_accuracy: 0.5500
Epoch 9/25
4/4 [==============================] - 2s 473ms/step - loss: 0.5873 - accuracy:
0.7125 - val_loss: 0.5146 - val_accuracy: 0.9625
Epoch 10/25
4/4 [==============================] - 2s 424ms/step - loss: 0.5220 - accuracy:
0.8500 - val_loss: 0.3816 - val_accuracy: 0.9750
Epoch 11/25
4/4 [==============================] - 2s 463ms/step - loss: 0.3781 - accuracy:
0.9125 - val_loss: 0.2887 - val_accuracy: 0.9750
Epoch 12/25
4/4 [==============================] - 2s 534ms/step - loss: 0.3714 - accuracy:
0.8625 - val_loss: 0.2226 - val_accuracy: 0.9500
Epoch 13/25
4/4 [==============================] - 2s 447ms/step - loss: 0.2711 - accuracy:
0.8750 - val_loss: 0.2518 - val_accuracy: 0.8875
Epoch 14/25
4/4 [==============================] - 2s 425ms/step - loss: 0.1922 - accuracy:
0.9250 - val_loss: 0.4512 - val_accuracy: 0.6750
Epoch 15/25
4/4 [==============================] - 2s 430ms/step - loss: 0.3784 - accuracy:
0.8000 - val_loss: 0.2028 - val_accuracy: 0.9500
Epoch 16/25
4/4 [==============================] - 2s 454ms/step - loss: 0.3579 - accuracy:
0.8250 - val_loss: 0.1934 - val_accuracy: 0.9625
Epoch 17/25
4/4 [==============================] - 2s 457ms/step - loss: 0.1977 - accuracy:
0.9375 - val_loss: 0.1427 - val_accuracy: 1.0000
Epoch 18/25
4/4 [==============================] - 2s 542ms/step - loss: 0.2353 - accuracy:
0.8750 - val_loss: 0.0724 - val_accuracy: 1.0000
Epoch 19/25
4/4 [==============================] - 2s 410ms/step - loss: 0.1240 - accuracy:
0.9500 - val_loss: 0.0613 - val_accuracy: 0.9750
Epoch 20/25
4/4 [==============================] - 2s 425ms/step - loss: 0.0974 - accuracy:
0.9500 - val_loss: 0.0241 - val_accuracy: 1.0000
Epoch 21/25
4/4 [==============================] - 2s 450ms/step - loss: 0.0800 - accuracy:
0.9500 - val_loss: 0.0165 - val_accuracy: 1.0000
Epoch 22/25
4/4 [==============================] - 2s 650ms/step - loss: 0.0612 - accuracy:
0.9875 - val_loss: 0.0098 - val_accuracy: 1.0000
```

```
Epoch 23/25
4/4 [==============================] - 2s 479ms/step - loss: 0.0527 - accuracy:
0.9625 - val_loss: 0.0077 - val_accuracy: 1.0000
Epoch 24/25
4/4 [==============================] - 2s 435ms/step - loss: 0.0344 - accuracy:
0.9875 - val_loss: 0.0071 - val_accuracy: 1.0000
Epoch 25/25
4/4 [==============================] - 2s 439ms/step - loss: 0.0449 - accuracy:
0.9875 - val_loss: 0.0087 - val_accuracy: 1.0000
```

12.3 Testing CNN

```
[ ]: # Preprocessing for test data (with shuffle=False)
test_datagen = ImageDataGenerator(rescale=1./255)

# Generating test data from the respective directory (with shuffle=False)
test_set = test_datagen.flow_from_directory('./dataset/test',
                                            target_size=(image_height,␣
 ↪image_width),
                                            batch_size=25,
                                            class_mode='binary',
                                            shuffle=False)

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(test_set, verbose=2)

# Make predictions
predictions = model.predict(test_set)
predicted_labels = (predictions > 0.5).astype(int).flatten()

# Get true labels from the test set
true_labels = test_set.classes

# Calculate performance metrics
accuracy = accuracy_score(true_labels, predicted_labels)
precision = precision_score(true_labels, predicted_labels, average='weighted')
recall = recall_score(true_labels, predicted_labels, average='weighted')
f1 = f1_score(true_labels, predicted_labels, average='weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Found 20 images belonging to 2 classes.
1/1 - 0s - loss: 0.0333 - accuracy: 1.0000 - 208ms/epoch - 208ms/step
1/1 [==============================] - 0s 242ms/step
Accuracy: 1.00
```

```
Precision: 1.00
Recall: 1.00
F1-Score: 1.00
```

```
2023-03-25 01:23:00.280396: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.
```

12.4 Data Leakage

- Additionally, the `Data Split and file management Pipeline` had been edite to take care of this.

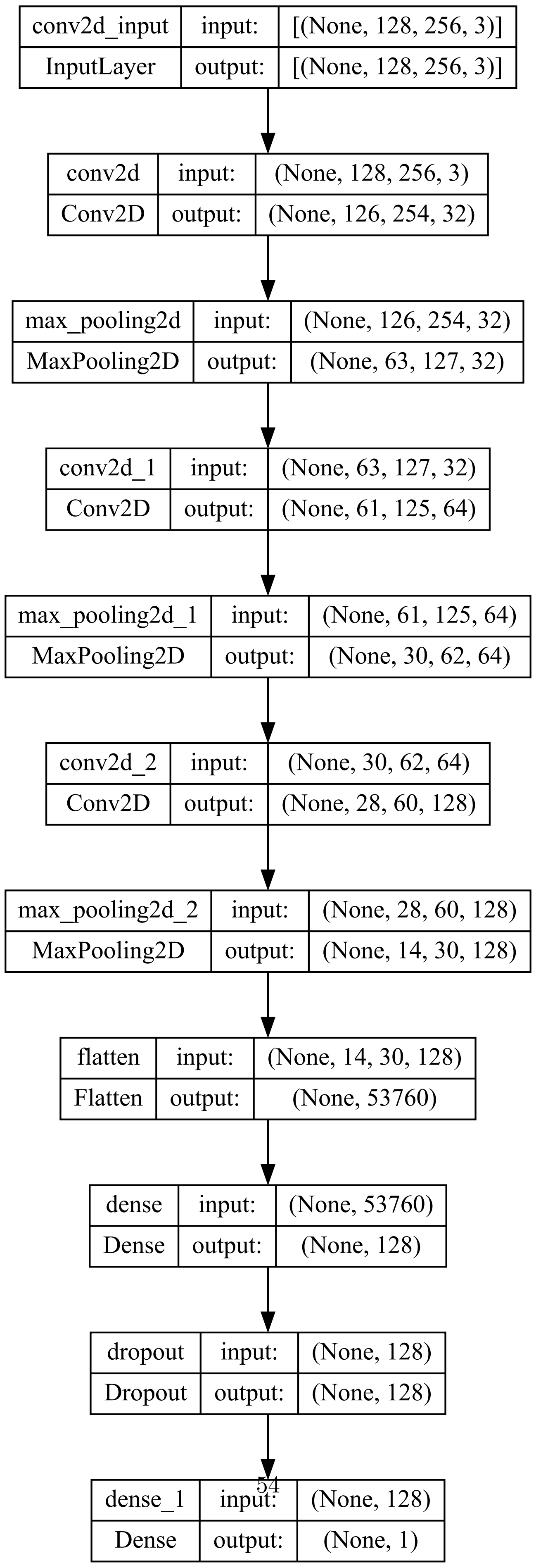
Noting that we have seperated the data to the training and test set earlier to avoid any data leakage, getting such metrics shows that the CNN was such a great model to classify our data with. However, it is important to note that it is possible that the test set has an easier to predict distribution of data. Thus, we cannot be sure that the model performs as it did, given new data. But we can know that the model is performing decent.

Reviewing the metrics meanings:

- Accuracy: This metric represents the fraction of instances that are classified correctly out of the entire dataset. A perfect accuracy score of 1.00 indicates that the model has successfully classified every instance.
- Precision: This metric is the proportion of true positives relative to the combined total of true positives and false positives. A perfect precision score of 1.00 signifies that the model has made no false positive predictions.
- Recall: This metric is the proportion of true positives relative to the combined total of true positives and false negatives. A perfect recall score of 1.00 signifies that the model has correctly identified all positive instances without any false negatives.
- F1-Score: This value represents the harmonic mean of precision and recall. An F1-score of 1.00 demonstrates that the model has achieved perfect precision and recall, resulting in balanced and outstanding performance.

12.5 CNN Architecture

```
[ ]: import tensorflow as tf
     from tensorflow.keras.utils import plot_model
     from IPython.display import SVG, display
     from tensorflow.keras.utils import model_to_dot


     # Visualize the model's architecture and convert it to an SVG format
     svg_image = SVG(model_to_dot(model, show_shapes=True, show_layer_names=True,␣
      ↪dpi=72).create_svg())

     # Display the SVG image inline in the Jupyter notebook
     display(svg_image)
```


conv2d_input | InputLayer | input: [(None, 128, 256, 3)] | output: [(None, 128, 256, 3)]
conv2d | Conv2D | input: (None, 128, 256, 3) | output: (None, 126, 254, 32)
max_pooling2d | MaxPooling2D | input: (None, 126, 254, 32) | output: (None, 63, 127, 32)
conv2d_1 | Conv2D | input: (None, 63, 127, 32) | output: (None, 61, 125, 64)
max_pooling2d_1 | MaxPooling2D | input: (None, 61, 125, 64) | output: (None, 30, 62, 64)
conv2d_2 | Conv2D | input: (None, 30, 62, 64) | output: (None, 28, 60, 128)
max_pooling2d_2 | MaxPooling2D | input: (None, 28, 60, 128) | output: (None, 14, 30, 128)
flatten | Flatten | input: (None, 14, 30, 128) | output: (None, 53760)
dense | Dense | input: (None, 53760) | output: (None, 128)
dropout | Dropout | input: (None, 128) | output: (None, 128)
dense_1 | Dense | input: (None, 128) | output: (None, 1)

13 Transfer Learning

13.1 Walk through

Transfer learning is a method in machine learning that involves utilizing a pre-trained model, initially trained on a substantial dataset, and adapting it for a different yet related problem. The central concept of transfer learning is to use the knowledge obtained during the training of the original task to improve the performance on the new task, specially when there is limited data for the new task.

- Pretraining: As the model is trained on a large dataset specific to a particular task, such as image classification, it learns a variety of low-level and high-level features that may be useful for other tasks. For instance, a Convolutional Neural Network (CNN) trained on the ImageNet dataset (comprising millions of images and thousands of classes) can learn different filters and representations that can be valuable for other image-related tasks.
- Fine-tuning: The pre-trained model is then adjusted for the new task and the new dataset. This includes altering the final layers of the model and training it with the new data, while keeping the weights of the earlier layers freezed.

The main advantage of transfer learning is quicker convergence and enhanced performance!

13.2 Pre-Training

[ ]:
```python
import os
import numpy as np
import librosa
import tensorflow as tf
import tensorflow_hub as hub
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

def load_audio_data(audio_files, labels):
    """
    Loads audio files and labels from disk and returns them as NumPy arrays.
    :param audio_files: List of paths to audio files
    :param labels: List of labels corresponding to the audio files
    :return: Tuple of NumPy arrays (audio_data, labels)
    """
    audio_data = []
    for file in audio_files:
        # Load audio file
        y, sr = librosa.load(file, sr=16000) # Resampling to 16kHz
        audio_data.append(y)

    # Encode labels
    encoder = LabelEncoder()
    encoded_labels = encoder.fit_transform(labels)

    return np.array(audio_data), np.array(encoded_labels)
```

```
[ ]: def path_gen(folderpath):
         """
         Generating the paths of the audio files in a folder

         Parameters
         ----------
         folderpath : string
             The path to the folder containing the audio files

         Returns
         -------
         audios_path : list
             The list of the paths of the audio files in the folder
         """
         audios_path = []
         for root, dirs, files in os.walk(folderpath, topdown=False):
             for name in files:
                 file_path=os.path.join(root, name)
                 try:
                     if file_path.lower().endswith(('.wav')):
                         audios_path.append(file_path)
                 except:
                     continue
         return audios_path
```

```
[ ]: mom_files = path_gen("./converted_mom")
     me_files = path_gen("./converted_me")

     path = np.concatenate((mom_files, me_files))

     y=np.concatenate((np.zeros(len(mom_files)),np.ones(len(me_files))))
```

13.3 Google's YAMNet

```
[ ]: %%capture
     # Load your dataset
     audio_files = path
     labels = y  # binary labels corresponding to the audio files

     # Preprocess the data
     X, y = load_audio_data(audio_files, labels)
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Load YAMNet for feature extraction
yamnet_model_handle = 'https://tfhub.dev/google/yamnet/1'
yamnet_model = hub.load(yamnet_model_handle)

# Define a function to extract features using YAMNet
def extract_embedding(audio):
    """
    Extracts embedding from audio data using YAMNet
    :param audio: Audio data
    :return: Embedding
    """
    _, embeddings, _ = yamnet_model(audio)
    return np.asarray(embeddings).max(axis=0)

# Extract features from the audio data
X_train_embeddings = np.asarray([extract_embedding(x) for x in X_train])
X_test_embeddings = np.asarray([extract_embedding(x) for x in X_test])
# Create a custom classifier
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(1024,)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),␣
 ↪loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train_embeddings, y_train, batch_size=32, epochs=10,␣
 ↪validation_data=(X_test_embeddings, y_test))

# Save the trained model
model.save('binary_audio_classifier_yamnet.h5')
```

Sample output:

```
Epoch 1/10

3/3 [==============================] - 1s 95ms/step - loss: 0.8414 - accuracy: 0.5250 - val_loss: 0.6678 - val_accuracy: 0.5000

Epoch 2/10

3/3 [==============================] - 0s 23ms/step - loss: 0.5799 - accuracy:
```

```
0.7125 - val_loss: 0.4569 - val_accuracy: 0.7000

Epoch 3/10

3/3 [==============================] - 0s 25ms/step - loss: 0.5383 - accuracy:
0.6250 - val_loss: 0.3353 - val_accuracy: 1.0000

Epoch 4/10

3/3 [==============================] - 0s 24ms/step - loss: 0.3720 - accuracy:
0.9000 - val_loss: 0.2539 - val_accuracy: 1.0000

Epoch 5/10

3/3 [==============================] - 0s 26ms/step - loss: 0.2933 - accuracy:
0.9000 - val_loss: 0.1830 - val_accuracy: 1.0000

Epoch 6/10

3/3 [==============================] - 0s 27ms/step - loss: 0.2314 - accuracy:
0.9250 - val_loss: 0.1446 - val_accuracy: 1.0000

Epoch 7/10

3/3 [==============================] - 0s 26ms/step - loss: 0.1804 - accuracy:
0.9625 - val_loss: 0.1122 - val_accuracy: 1.0000

Epoch 8/10

3/3 [==============================] - 0s 26ms/step - loss: 0.1271 - accuracy:
0.9750 - val_loss: 0.0928 - val_accuracy: 1.0000

Epoch 9/10

3/3 [==============================] - 0s 27ms/step - loss: 0.0976 - accuracy:
0.9875 - val_loss: 0.0800 - val_accuracy: 1.0000

Epoch 10/10

3/3 [==============================] - 0s 27ms/step - loss: 0.0800 - accuracy:
0.9750 - val_loss: 0.0883 - val_accuracy: 0.9500
```

As we can see above using YAMNet as our pre-trained model lead to very good results.

later, we are using another pre-trained model, ImageNet, which probably may not be the best pre-trained model as it has been initially trained for object detection. However, maybe the filters that it learned about the images, could be of use to classify the spectrograms. Let's see after evaluating the above model.

13.4 Pre-Training

```
[ ]: from sklearn.metrics import classification_report, confusion_matrix,␣
      ↪accuracy_score

     # Evaluate the model on the test set
     y_test_pred = model.predict(X_test_embeddings)
```

```
y_test_pred_classes = (y_test_pred > 0.5).astype("int32")

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_test_pred_classes)

# Generate the classification report
report = classification_report(y_test, y_test_pred_classes)

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_test_pred_classes)

print(f"Accuracy: {accuracy:.2f}")
print("Classification report:")
print(report)
print("Confusion matrix:")
print(conf_matrix)
```

```
1/1 [==============================] - 0s 67ms/step
Accuracy: 0.95
Classification report:
              precision    recall  f1-score   support

           0       1.00      0.92      0.96        12
           1       0.89      1.00      0.94         8

    accuracy                           0.95        20
   macro avg       0.94      0.96      0.95        20
weighted avg       0.96      0.95      0.95        20

Confusion matrix:
[[11  1]
 [ 0  8]]
```

[ ]:
```
import numpy as np
from PIL import Image
def png_to_array(image_path, target_size=(224, 224)):
    """
    Converts a PNG image to a NumPy array
    :param image_path: Path to the image
    :param target_size: Target size of the image
    :return: NumPy array
    """
    image = Image.open(image_path)
    image = image.resize(target_size, Image.ANTIALIAS)
    return np.array(image)
```

```
[ ]: def path_gen(folderpath):
         """
         Generating the paths of the .png files in a folder

         Parameters
         ----------
         folderpath : string
             The path to the folder containing the image files

         Returns
         -------
         audios_path : list
             The list of the paths of the .png files in the folder
         """
         audios_path = []
         for root, dirs, files in os.walk(folderpath, topdown=False):
             for name in files:
                 file_path=os.path.join(root, name)
                 try:
                     if file_path.lower().endswith(('.png')):
                         audios_path.append(file_path)
                 except:
                     continue
         return audios_path
```

```
[ ]: #load dataset of dataset folder
     import os
     import numpy as np
     #train folder
     #class 0 mom
     train_class_0_path = path_gen("/content/dataset/dataset/train/class_0")
     train_class_0 = [png_to_array(path) for path in train_class_0_path]

     #class 1 me
     train_class_1_path = path_gen("/content/dataset/dataset/train/class_1")
     train_class_1 = [png_to_array(path) for path in train_class_1_path]

     train_set = np.concatenate((train_class_0, train_class_1))
     train_predict = np.concatenate((np.zeros(len(train_class_0)),np.
      ↪ones(len(train_class_1))))

     #test folder
     #class 0 mom
     test_class_0_path = path_gen("/content/dataset/dataset/test/class_0")
     test_class_0 = [png_to_array(path) for path in test_class_0_path]
     #class 1 me
     test_class_1_path = path_gen("/content/dataset/dataset/test/class_1")
```

```
test_class_1 = [png_to_array(path) for path in test_class_1_path]

test_set = np.concatenate((test_class_0, test_class_1))
test_predict = np.concatenate((np.zeros(len(test_class_0)),np.
 ↪ones(len(test_class_1))))
```

13.5 MobileNetV2

[ ]:
```
import numpy as np
import tensorflow as tf
tf.config.run_functions_eagerly(True)
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

# Load your dataset (spectrograms)
train_x, train_y, val_x, val_y = train_set, train_predict, test_set,␣
 ↪test_predict
train_x = train_x[:, :, :, :3]
val_x = val_x[:, :, :, :3]

# Load the MobileNetV2 model
base_model = MobileNetV2(weights='imagenet', include_top=False,␣
 ↪input_shape=(224, 224, 3))

# Set the number of layers to freeze
num_frozen_layers = 1  # Adjust this value based on your requirements

# Freeze the layers
for layer in base_model.layers[:num_frozen_layers]:
    layer.trainable = False

# Add custom layers for binary classification
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)

# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(optimizer=Adam(learning_rate=1e-3), loss='binary_crossentropy',␣
 ↪metrics=['accuracy'], run_eagerly=True)
```

```
# Set up callbacks
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy',␣
 ↪save_best_only=True, mode='max')
early_stopping = EarlyStopping(monitor='val_loss', patience=10,␣
 ↪restore_best_weights=True)

# Train the model
history = model.fit(train_x, train_y, validation_data=(val_x, val_y),␣
 ↪epochs=20, batch_size=50, callbacks=[checkpoint, early_stopping])
```

```
Epoch 1/20
2/2 [==============================] - 12s 6s/step - loss: 1.0209 - accuracy:
0.4125 - val_loss: 2.5446 - val_accuracy: 0.5000
Epoch 2/20
2/2 [==============================] - 11s 6s/step - loss: 0.1773 - accuracy:
0.9750 - val_loss: 3.1934 - val_accuracy: 0.5000
Epoch 3/20
2/2 [==============================] - 10s 4s/step - loss: 0.0173 - accuracy:
1.0000 - val_loss: 4.0355 - val_accuracy: 0.5000
Epoch 4/20
2/2 [==============================] - 12s 4s/step - loss: 0.0087 - accuracy:
1.0000 - val_loss: 4.9401 - val_accuracy: 0.5000
Epoch 5/20
2/2 [==============================] - 11s 4s/step - loss: 0.0025 - accuracy:
1.0000 - val_loss: 5.8412 - val_accuracy: 0.5000
Epoch 6/20
2/2 [==============================] - 11s 5s/step - loss: 0.0016 - accuracy:
1.0000 - val_loss: 6.7972 - val_accuracy: 0.5000
Epoch 7/20
2/2 [==============================] - 12s 6s/step - loss: 8.7114e-04 -
accuracy: 1.0000 - val_loss: 7.4981 - val_accuracy: 0.5000
Epoch 8/20
2/2 [==============================] - 11s 5s/step - loss: 0.0044 - accuracy:
1.0000 - val_loss: 7.4691 - val_accuracy: 0.5000
Epoch 9/20
2/2 [==============================] - 10s 4s/step - loss: 4.2167e-04 -
accuracy: 1.0000 - val_loss: 7.2174 - val_accuracy: 0.5000
Epoch 10/20
2/2 [==============================] - 11s 4s/step - loss: 7.5203e-05 -
accuracy: 1.0000 - val_loss: 7.0566 - val_accuracy: 0.5000
Epoch 11/20
2/2 [==============================] - 13s 5s/step - loss: 5.9246e-05 -
accuracy: 1.0000 - val_loss: 7.0054 - val_accuracy: 0.5000
```

With ImageNet, it has not gone as well yet. If my training accuracy reaches 100% while the validation accuracy remains around 50%, it is a clear indication of overfitting. Overfitting happens when the model learns the training data too well, including the noise and specific patterns that may not generalize well to new, unseen data. Thus, I will do data augmentation to improve on that.

14 Data Augmentation

Data augmentation, used to diversify and increase the size of the dataset.

Audio data augmentation techniques:

- Time stretching: Changes the speed of the audio without affecting the pitch.
- Pitch shifting: Changes the pitch of the audio without affecting the speed.
- Adding noise: Adds background noise to the audio.
- Time shifting: Shifts the audio in time by adding silence either at the beginning or the end.

[ ]:
```
def time_stretch(audio, rate):
    """
    Stretch an audio signal by a given rate

    Parameters
    ----------
    audio : numpy array
        The audio signal to be stretched
    rate : float
        The rate at which the signal will be stretched

    Returns
    -------
    stretched_audio : numpy array
        The stretched audio signal"""
    return librosa.effects.time_stretch(y=audio, rate=rate)

def pitch_shift(audio, sr, n_steps):
    """
    Shift the pitch of an audio signal by a given number of semitones

    Parameters
    ----------
    audio : numpy array
        The audio signal to be pitch shifted
    sr : int
        The sampling rate of the audio signal
    n_steps : float
        The number of semitones by which the signal will be shifted

    Returns
    -------
    shifted_audio : numpy array
        The pitch shifted audio signal"""
    return librosa.effects.pitch_shift(y=audio, sr=sr, n_steps=n_steps)

def add_noise(audio, noise_factor):
    """
    Add noise to an audio signal

    Parameters
    ----------
    audio : numpy array
        The audio signal to which noise will be added

    noise_factor : float
        The factor by which the noise will be multiplied

    Returns
    --------
    noisy_audio : numpy array
        The noisy audio signal
    """
    noise = np.random.randn(len(audio))
    return audio + noise_factor * noise

def time_shift(audio, shift):
    """
    Shift an audio signal in time

    Parameters
    ----------
    audio : numpy array
        The audio signal to be shifted
    shift : int
        The number of samples by which the signal will be shifted

    Returns
    -------
    shifted_audio : numpy array
        The shifted audio signal
    """
    if shift > 0:
        return np.pad(audio, (0, shift), mode='constant')
    else:
        return np.pad(audio, (-shift, 0), mode='constant')[:-shift]
```

```
[ ]: def melspectrogram(audio, sr, n_mels=128, fmax=8000):
         """
         Compute the mel-scaled spectrogram of an audio signal
```

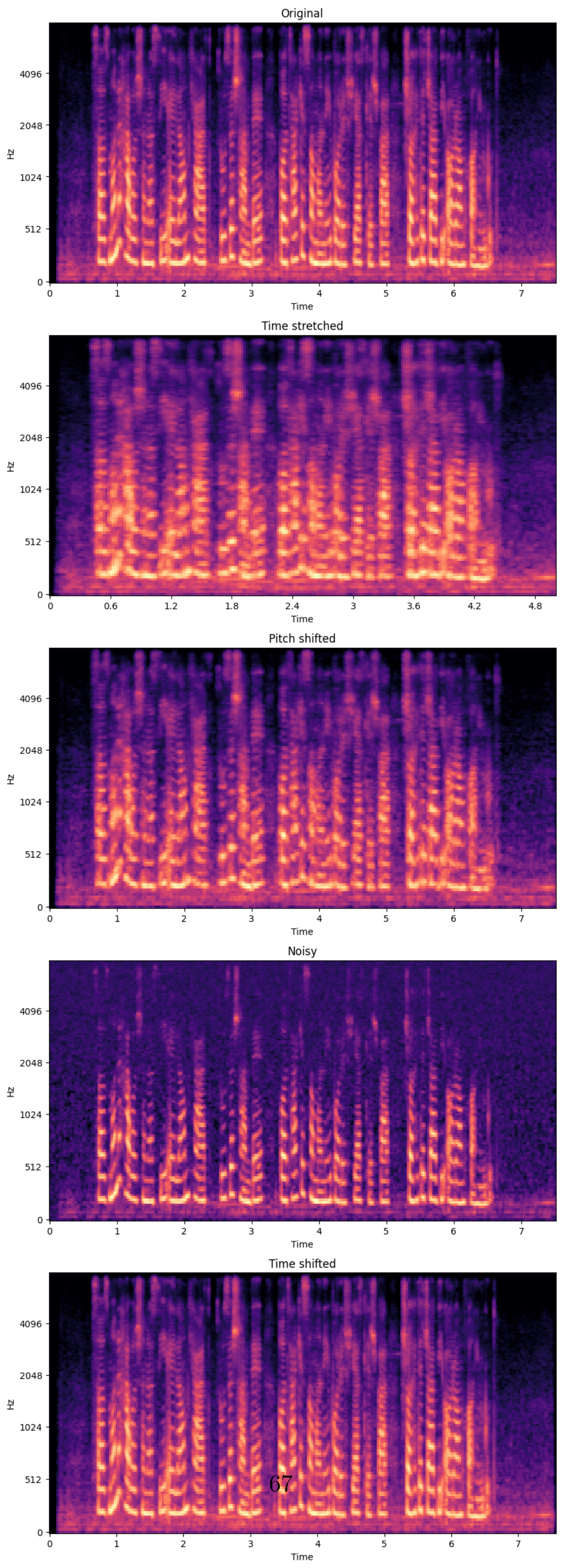
```
    Parameters
    ----------
    audio : numpy array
        The audio signal
    sr : int
        The sampling rate of the audio signal
    n_mels : int
        The number of mel bands to generate
    fmax : int
        The highest frequency (in Hz)

    Returns
    -------
    melspectrogram : numpy array
        The mel-scaled spectrogram of the audio signal
    """
    return librosa.feature.melspectrogram(y=audio, sr=sr, n_mels=n_mels,␣
 ↪fmax=fmax)
```

Showing the augmentations on a sample data: - note: you can alter the figsize, i. e. to `figsize =(10,10)`, so that you can see all the plots next to each other. I spread them out becuase the titles of the axis get too close and may cause confusion.

14.1 Augmented Audios' Mel spectrograms

[ ]:
```
#Plotting the mel spectrogram of the augmented smaple audio file
me_sample_loaded = librosa.load(me_sample_path, sr=16000)[0]
sr=16000
time_stretched = time_stretch(me_sample_loaded, 1.5)
pitch_shifted = pitch_shift(me_sample_loaded, sr=16000, n_steps=2)
noisy = add_noise(me_sample_loaded, 0.005)
time_shifted = time_shift(me_sample_loaded, 1000)

fig, ax = plt.subplots(5, 1, figsize=(10, 30))
librosa.display.specshow(librosa.power_to_db(melspectrogram(me_sample_loaded,␣
 ↪sr), ref=np.max), y_axis='mel', fmax=8000, x_axis='time', ax=ax[0])
ax[0].set(title='Original')
librosa.display.specshow(librosa.power_to_db(melspectrogram(time_stretched,␣
 ↪sr), ref=np.max), y_axis='mel', fmax=8000, x_axis='time', ax=ax[1])
ax[1].set(title='Time stretched')
librosa.display.specshow(librosa.power_to_db(melspectrogram(pitch_shifted, sr),␣
 ↪ref=np.max), y_axis='mel', fmax=8000, x_axis='time', ax=ax[2])
ax[2].set(title='Pitch shifted')
librosa.display.specshow(librosa.power_to_db(melspectrogram(noisy, sr), ref=np.
 ↪max), y_axis='mel', fmax=8000, x_axis='time', ax=ax[3])
ax[3].set(title='Noisy')
```

```
librosa.display.specshow(librosa.power_to_db(melspectrogram(time_shifted, sr),␣
 ↪ref=np.max), y_axis='mel', fmax=8000, x_axis='time', ax=ax[4])
ax[4].set(title='Time shifted')

#showing the plot
plt.show()
```


Original
Time stretched
Pitch shifted
Noisy
Time shifted
Hz
Time
0
512
1024
2048
4096
0 1 2 3 4 5 6 7
0 0.6 1.2 1.8 2.4 3 3.6 4.2 4.8

14.2 Compile Augmentations

```
[ ]: def augmented_audio_spec_gen(audio_path, output_path):
         """
         Generate augmented audio spectrograms

         Parameters
         ----------
         audio_path : str
             The path to the audio file
         output_path : str
             The path to the output directory

         Returns
         -------
         None
         """
         # Load the audio file
         sr=16000

         # Compute Mel spectrogram
         create_mel_spectrograms(audio_path, output_path,␣
      ↪type_name="time_stretched",func=time_stretch, inp = [np.random.uniform(0.8,␣
      ↪1.2)])
         create_mel_spectrograms(audio_path, output_path,␣
      ↪type_name="pitch_shifted",func=pitch_shift, inp = [sr, np.random.randint(-3,␣
      ↪4)])
         create_mel_spectrograms(audio_path, output_path, type_name="noisy_audio",␣
      ↪func = add_noise, inp =[np.random.uniform(0.005, 0.02)])
         create_mel_spectrograms(audio_path, output_path, type_name="time_shifted",␣
      ↪func = time_shift, inp = [np.random.randint(-sr//2, sr//2)])


     # #Usage on augmented mom audios
     # input_directory = mom_paths
     # output_directory = './aug_mom_spectrograms'
     # augmented_audio_spec_gen(input_directory, output_directory)

     # #Usage on augmented my audios
     # input_directory = me_paths
     # output_directory = './aug_me_spectrograms'
     # augmented_audio_spec_gen(input_directory, output_directory)
```

```
[ ]: # Example usage
input_directory = './aug_mom_spectrograms'
output_directory = './aug_mom_split_data'
train_percentage = 0.8  # 80% of the data used for training

train_path, test_path = split_train_test(input_directory, train_percentage,␣
 ↪output_directory)

# Example usage
input_directory = './aug_me_spectrograms'
output_directory = './aug_me_split_data'
train_percentage = 0.8  # 80% of the data used for training

train_path, test_path = split_train_test(input_directory, train_percentage,␣
 ↪output_directory)
```

```
[ ]: # Example usage
input_directory1 = './aug_mom_split_data'  # Contains "train" and "test"␣
 ↪subfolders
input_directory2 = './aug_me_split_data'  # Contains "train" and "test"␣
 ↪subfolders
output_directory = './aug_general_data_split'  # The merged "train" and "test"␣
 ↪subfolders will be created here

merge_folders(input_directory1, input_directory2, output_directory)
```

15 Transfer Learning after data augmentation

15.1 Data management

```
[ ]: #load dataset of dataset folder
import os
import numpy as np
#train folder
#class 0 mom
train_class_0_path = path_gen("/content/augmented_dataset/train/class_0")
train_class_0 = [png_to_array(path) for path in train_class_0_path]

#class 1 me
train_class_1_path = path_gen("/content/augmented_dataset/train/class_1")
train_class_1 = [png_to_array(path) for path in train_class_1_path]

train_set = np.concatenate((train_class_0, train_class_1))
train_predict = np.concatenate((np.zeros(len(train_class_0)),np.
 ↪ones(len(train_class_1))))
```

```
#test folder
#class 0 mom
test_class_0_path = path_gen("/content/augmented_dataset/test/class_0")
test_class_0 = [png_to_array(path) for path in test_class_0_path]
#class 1 me
test_class_1_path = path_gen("/content/augmented_dataset/test/class_1")
test_class_1 = [png_to_array(path) for path in test_class_1_path]

test_set = np.concatenate((test_class_0, test_class_1))
test_predict = np.concatenate((np.zeros(len(test_class_0)),np.
 ↪ones(len(test_class_1))))
```

15.2 MobileNetV2 works!

[ ]:
```
import numpy as np
import tensorflow as tf
tf.config.run_functions_eagerly(True)
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, Callback

# Load your dataset (spectrograms)
train_x, train_y, val_x, val_y = train_set, train_predict, test_set,␣
 ↪test_predict
train_x = train_x[:, :, :, :3]
val_x = val_x[:, :, :, :3]

# Load the MobileNetV2 model
base_model = MobileNetV2(weights='imagenet', include_top=False,␣
 ↪input_shape=(224, 224, 3))

# Set the number of layers to freeze
num_frozen_layers = 2  # Adjust this value based on your requirements

# Freeze the layers
for layer in base_model.layers[:num_frozen_layers]:
    layer.trainable = False

# Add custom layers for binary classification
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)
```

```
# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(optimizer=Adam(learning_rate=1e-3), loss='binary_crossentropy',␣
 ↪metrics=['accuracy'], run_eagerly=True)

# Set up callbacks
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy',␣
 ↪save_best_only=True, mode='max')
early_stopping = EarlyStopping(monitor='val_loss', patience=10,␣
 ↪restore_best_weights=True)

# Custom callback to stop training when validation accuracy is above 89%
class StopAtAccuracy(Callback):
    def on_epoch_end(self, epoch, logs=None):
        if logs.get('val_accuracy') > 0.85:
            print("\nReached 89% validation accuracy, stopping training.")
            self.model.stop_training = True

stop_at_accuracy = StopAtAccuracy()
# Train the model
history = model.fit(train_x, train_y, validation_data=(val_x, val_y),␣
 ↪epochs=10, batch_size=80, callbacks=[checkpoint, early_stopping,␣
 ↪stop_at_accuracy])
```

```
Epoch 1/10
5/5 [==============================] - 117s 23s/step - loss: 0.3244 - accuracy:
0.8266 - val_loss: 2.2363 - val_accuracy: 0.5000
Epoch 2/10
5/5 [==============================] - 114s 23s/step - loss: 0.0046 - accuracy:
1.0000 - val_loss: 7.3600 - val_accuracy: 0.5000
Epoch 3/10
5/5 [==============================] - 115s 23s/step - loss: 0.0050 - accuracy:
0.9975 - val_loss: 11.2210 - val_accuracy: 0.5000
Epoch 4/10
5/5 [==============================] - 127s 26s/step - loss: 0.0063 - accuracy:
0.9975 - val_loss: 10.5875 - val_accuracy: 0.5000
Epoch 5/10
4/5 [=======================>…] - ETA: 22s - loss: 0.1094 - accuracy:
0.9844
```

We can see that the model performs well on the training set and poorly on the validation set. The overfitting could be due to the complexity of the model, data leak, or the fact that imagenet is used for object detection in images.

However, we are only using the first two layers of the model that cover the basic features like edges, softness, etc in the spectrograms. Additionally, we included an early stop, in case the validation accuracy reaches 85%. Meanwhile we can check the dataset to make sure that none of the augmented versions of the same audio end up in both training and testing, to cover data leak. I tired different batch sizes, frozen layers, learning rate etc. + I also made sure that the image data is well seperated to avoid data leak. After that we can see below, how the model improved:

15.3 Avoiding Data Leakage

[ ]:
```
#load dataset of dataset folder
import os
import numpy as np
#train folder
#class 0 mom
train_class_0_path = path_gen("./leak_proof_aug_spec_dataset/train/class_0")
train_class_0 = [png_to_array(path) for path in train_class_0_path]

#class 1 me
train_class_1_path = path_gen("./leak_proof_aug_spec_dataset/train/class_1")
train_class_1 = [png_to_array(path) for path in train_class_1_path]



train_set = np.concatenate((train_class_0, train_class_1))
train_predict = np.concatenate((np.zeros(len(train_class_0)),np.
 ↪ones(len(train_class_1))))

#test folder
#class 0 mom
test_class_0_path = path_gen("./leak_proof_aug_spec_dataset/test/class_0")
test_class_0 = [png_to_array(path) for path in test_class_0_path]
#class 1 me
test_class_1_path = path_gen("./leak_proof_aug_spec_dataset/test/class_1")
test_class_1 = [png_to_array(path) for path in test_class_1_path]

test_set = np.concatenate((test_class_0, test_class_1))
test_predict = np.concatenate((np.zeros(len(test_class_0)),np.
 ↪ones(len(test_class_1))))
```

```
/var/folders/df/90mwvjrs2yb0860h5fp23hlm0000gn/T/ipykernel_27963/1597556688.py:5
: DeprecationWarning: ANTIALIAS is deprecated and will be removed in Pillow 10
(2023-07-01). Use LANCZOS or Resampling.LANCZOS instead.
  image = image.resize(target_size, Image.ANTIALIAS)
```

15.4 MobileNetV2 (After splitting leakage proof data split)

[ ]:
```
import numpy as np
import tensorflow as tf
tf.config.run_functions_eagerly(True)
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, Callback

# Load your dataset (spectrograms)
train_x, train_y, val_x, val_y = train_set, train_predict, test_set,␣
 ↪test_predict
train_x = train_x[:, :, :, :3]
val_x = val_x[:, :, :, :3]

# Load the MobileNetV2 model
base_model = MobileNetV2(weights='imagenet', include_top=False,␣
 ↪input_shape=(224, 224, 3))

# Set the number of layers to freeze
num_frozen_layers = 154-5  # Adjust this value based on your requirements

# Freeze the layers
for layer in base_model.layers[:num_frozen_layers]:
    layer.trainable = False

# Add custom layers for binary classification
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)

# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(optimizer=Adam(learning_rate=1e-3), loss='binary_crossentropy',␣
 ↪metrics=['accuracy'], run_eagerly=True)

# Set up callbacks
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy',␣
 ↪save_best_only=True, mode='max')
early_stopping = EarlyStopping(monitor='val_loss', patience=10,␣
 ↪restore_best_weights=True)

# Custom callback to stop training when validation accuracy is equal or bigger␣
 ↪than 90%
class StopAtAccuracy(Callback):
    def on_epoch_end(self, epoch, logs=None):
        if logs.get('val_accuracy') >= 0.90:
            print("\nReached 90% validation accuracy, stopping training.")
            self.model.stop_training = True

stop_at_accuracy = StopAtAccuracy()
# Train the model
history = model.fit(train_x, train_y, validation_data=(val_x, val_y),␣
 ↪epochs=10, callbacks=[checkpoint, early_stopping, stop_at_accuracy])
```

```
Epoch 1/10
10/10 [==============================] - 37s 4s/step - loss: 0.9707 - accuracy:
0.6375 - val_loss: 2.3784 - val_accuracy: 0.5000
Epoch 2/10
10/10 [==============================] - 34s 3s/step - loss: 0.4675 - accuracy:
0.7656 - val_loss: 1.8971 - val_accuracy: 0.5000
Epoch 3/10
10/10 [==============================] - 36s 4s/step - loss: 0.3495 - accuracy:
0.8281 - val_loss: 1.1471 - val_accuracy: 0.5125
Epoch 4/10
10/10 [==============================] - 36s 4s/step - loss: 0.2570 - accuracy:
0.8469 - val_loss: 1.0671 - val_accuracy: 0.5375
Epoch 5/10
10/10 [==============================] - 40s 4s/step - loss: 0.2201 - accuracy:
0.8781 - val_loss: 0.4797 - val_accuracy: 0.7000
Epoch 6/10
10/10 [==============================] - 38s 4s/step - loss: 0.2090 - accuracy:
0.8656 - val_loss: 0.3200 - val_accuracy: 0.9000
Epoch 7/10
10/10 [==============================] - ETA: 0s - loss: 0.1991 - accuracy:
0.8562
Reached 90% validation accuracy, stopping training.
10/10 [==============================] - 43s 4s/step - loss: 0.1991 - accuracy:
0.8562 - val_loss: 0.2489 - val_accuracy: 0.9250
```

```
[ ]: %%capture #the model is too big to keep it displayed here. You can comment the␣
      ↪capture and see the model's visualization
     from tensorflow.keras.utils import plot_model

     plot_model(base_model, to_file='base_model.png', show_shapes=True,␣
      ↪expand_nested=True)
```

As we can see above the validation accuracy improved but we can see that the accuracy is lower than the validation accuracy which could be a sign of underfitting. It also could be that the validation set has an easier distribution to classify, or the effetc of the early stop.

16 RNN

16.1 Mathematical Walk through

Recurrent Neural Networks (RNNs) are a specialized type of neural network tailored for processing sequential data, such as time series or natural language. They excel at handling inputs and outputs of varying lengths. RNNs possess a distinct architecture that enables them to maintain an internal state (hidden state) for capturing information from earlier time steps, rendering them ideal for tasks that involve sequential data.

- Input layer: For each time step t, the input layer accepts a data point from the input sequence, shown as x_t. The input sequence (x_1, x_2, ..., x_T), with T representing the total number of time steps.
- Hidden layer: The hidden layer retains an internal state, denoted as h_t, which is updated at every time step. The hidden state relies on the current input x_t and the previous hidden state h_(t-1). The hidden layer calculates the new hidden state using the following formula:

$$h_t = f(W_{hh} \times h_{t-1} + W_{xh} \times x_t + b_h)$$

```
- Where:
- f: activation function, typically tanh or ReLU
- W_hh: weight matrix for the previous hidden state
- W_xh: weight matrix for the input
- b_h: bias term for the hidden layer
```

- Output layer: At each time step, the output layer computes an output y_t based on the current hidden state h_t. The output can be determined using the following formula:

$$y_t = g(W_{hy} \times h_t + b_y)$$

  - Where:
  - g: activation function, usually softmax for classification tasks or linear for regression tasks
  - W_hy: weight matrix for the output layer
  - b_y: bias term for the output layer

- Loss function: The loss function quantifies the discrepancy between the predicted outputs and the actual labels for the entire sequence. Typical loss functions for RNNs include cross-entropy loss for classification tasks or mean squared error for regression tasks:

L = Σ(y_true - y_pred)^2 (for regression)

L = -Σ(y_true ×log(y_pred)) (for classification)

```
- Where:
- y_true: actual label
- y_pred: predicted output
```

To train an RNN, the weights (W_hh, W_xh, W_hy) and biases (b_h, b_y) are updated using an optimization algorithm, such as stochastic gradient descent, aiming to minimize the loss function. Backpropagation through time (BPTT) is a prevalent technique for computing gradients in RNNs, which entails unfolding the network in time and applying backpropagation to the unfolded graph.

It is important to note that RNNs can encounter vanishing or exploding gradient issues during training, making it challenging for them to capture long-range dependencies. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks are designed to address these problems and are frequently employed as more efficient alternatives to traditional RNNs.

16.2 Data splits for Audio instead of images

```
[ ]: def audio_path_gen(folderpath):
         """
         Generating the paths of the audio files in a folder

         Parameters
         ----------
         folderpath : string
             The path to the folder containing the audio files

         Returns
         -------
         audios_path : list
             The list of the paths of the audio files in the folder
         """
         audios_path = []
         for root, dirs, files in os.walk(folderpath, topdown=False):
             for name in files:
                 file_path=os.path.join(root, name)
                 try:
                     if file_path.lower().endswith(('.wav')):
                         audios_path.append(file_path)
                 except:
                     continue
         return audios_path

     #load dataset of dataset folder
     import os
     import numpy as np
     #train folder
     #class 0 mom
     train_class_0_path = audio_path_gen("./leak_proof_aug_dataset/train/class_0")

     #class 1 me
     train_class_1_path = audio_path_gen("./leak_proof_aug_dataset/train/class_1")


     train_set = np.concatenate((train_class_0_path, train_class_1_path))
     train_predict = np.concatenate((np.zeros(len(train_class_0_path)),np.
      ↪ones(len(train_class_1_path))))
```

```
#test folder
#class 0 mom
test_class_0_path = audio_path_gen("./leak_proof_aug_dataset/test/class_0")

#class 1 me
test_class_1_path = audio_path_gen("./leak_proof_aug_dataset/test/class_1")


test_set = np.concatenate((test_class_0_path, test_class_1_path))
test_predict = np.concatenate((np.zeros(len(test_class_0_path)),np.
 ↪ones(len(test_class_1_path))))
```

16.3 Feature Extraction, Normalization and Training

[ ]:
```
import os
import numpy as np
import librosa
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

tf.config.run_functions_eagerly(True)

def load_audio_files_and_extract_features(file_paths):
    """
    Load audio files and extract MFCC features

    Parameters
    ----------
    file_paths : list
        The list of the paths of the audio files

    Returns
    -------
    features : list
        The list of the MFCC features of the audio files
    """
    features = []
    for file_path in file_paths:
        audio, sr = librosa.load(file_path)
        mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=39)  # Increase the␣
 ↪number of MFCC features
        features.append(mfccs.T)
    return features

# Normalize MFCC features
def normalize_features(features):
    scaler = StandardScaler()
    return [scaler.fit_transform(feature) for feature in features]

class_0_features =␣
 ↪normalize_features(load_audio_files_and_extract_features(train_class_0_path))
class_1_features =␣
 ↪normalize_features(load_audio_files_and_extract_features(train_class_1_path))

X = class_0_features + class_1_features
y = np.concatenate([np.zeros(len(class_0_features)), np.
 ↪ones(len(class_1_features))])

X = tf.keras.preprocessing.sequence.pad_sequences(X, padding='post',␣
 ↪dtype='float32')
max_seq_length = max([seq.shape[0] for seq in X])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=55)

X_train = tf.keras.preprocessing.sequence.pad_sequences(X_train,␣
 ↪maxlen=max_seq_length, padding='post', dtype='float32')
X_test = tf.keras.preprocessing.sequence.pad_sequences(X_test,␣
 ↪maxlen=max_seq_length, padding='post', dtype='float32')

timesteps = X_train.shape[1]
input_dim = X_train.shape[2]

model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(64, kernel_size=3, activation='relu',␣
 ↪input_shape=(timesteps, input_dim)),  # Added a Conv1D layer
    tf.keras.layers.MaxPooling1D(pool_size=2),  # Added a MaxPooling1D layer
    tf.keras.layers.Conv1D(128, kernel_size=3, activation='relu'),  # Added␣
 ↪another Conv1D layer
    tf.keras.layers.MaxPooling1D(pool_size=2),  # Added another MaxPooling1D␣
 ↪layer
    tf.keras.layers.GRU(128, return_sequences=True),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.LSTM(128, return_sequences=True),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.GRU(128),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Adjust learning rate
learning_rate = 1e-3
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)

model.compile(optimizer=optimizer, loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',␣
 ↪patience=20, restore_best_weights=True)

history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
 ↪epochs=50, batch_size=100, callbacks=[early_stopping])
```

```
Epoch 1/50
```

```
/opt/homebrew/Caskroom/miniforge/base/envs/mlp/lib/python3.8/site-
packages/tensorflow/python/data/ops/structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
```

```
1/1 [==============================] - 0s 345ms/step - loss: 0.6908 - accuracy:
0.5938 - val_loss: 0.6947 - val_accuracy: 0.3750
Epoch 2/50
1/1 [==============================] - 0s 261ms/step - loss: 0.6920 - accuracy:
0.5781 - val_loss: 0.6954 - val_accuracy: 0.3750
Epoch 3/50
1/1 [==============================] - 0s 210ms/step - loss: 0.6904 - accuracy:
0.5156 - val_loss: 0.6965 - val_accuracy: 0.3750
Epoch 4/50
1/1 [==============================] - 0s 203ms/step - loss: 0.6842 - accuracy:
0.5469 - val_loss: 0.6983 - val_accuracy: 0.3750
Epoch 5/50
1/1 [==============================] - 0s 199ms/step - loss: 0.6970 - accuracy:
0.5156 - val_loss: 0.6990 - val_accuracy: 0.3750
Epoch 6/50
1/1 [==============================] - 0s 190ms/step - loss: 0.6848 - accuracy:
0.5156 - val_loss: 0.6992 - val_accuracy: 0.3750
Epoch 7/50
1/1 [==============================] - 0s 187ms/step - loss: 0.6822 - accuracy:
0.5312 - val_loss: 0.6990 - val_accuracy: 0.3750
Epoch 8/50
1/1 [==============================] - 0s 190ms/step - loss: 0.6866 - accuracy:
0.5000 - val_loss: 0.6992 - val_accuracy: 0.3750
Epoch 9/50
1/1 [==============================] - 0s 191ms/step - loss: 0.6785 - accuracy:
0.5312 - val_loss: 0.6970 - val_accuracy: 0.3750
Epoch 10/50
1/1 [==============================] - 0s 206ms/step - loss: 0.6681 - accuracy:
```

```
0.6094 - val_loss: 0.6919 - val_accuracy: 0.3750
Epoch 11/50
1/1 [==============================] - 0s 196ms/step - loss: 0.6686 - accuracy:
0.6094 - val_loss: 0.6824 - val_accuracy: 0.5000
Epoch 12/50
1/1 [==============================] - 0s 197ms/step - loss: 0.6522 - accuracy:
0.6562 - val_loss: 0.6706 - val_accuracy: 0.6250
Epoch 13/50
1/1 [==============================] - 0s 210ms/step - loss: 0.6277 - accuracy:
0.6719 - val_loss: 0.6597 - val_accuracy: 0.6250
Epoch 14/50
1/1 [==============================] - 0s 196ms/step - loss: 0.6052 - accuracy:
0.7031 - val_loss: 0.6592 - val_accuracy: 0.5625
Epoch 15/50
1/1 [==============================] - 0s 185ms/step - loss: 0.5845 - accuracy:
0.6875 - val_loss: 0.6749 - val_accuracy: 0.5625
Epoch 16/50
1/1 [==============================] - 0s 201ms/step - loss: 0.5508 - accuracy:
0.7031 - val_loss: 0.7243 - val_accuracy: 0.6250
Epoch 17/50
1/1 [==============================] - 0s 188ms/step - loss: 0.5836 - accuracy:
0.6719 - val_loss: 0.7257 - val_accuracy: 0.6250
Epoch 18/50
1/1 [==============================] - 0s 198ms/step - loss: 0.5857 - accuracy:
0.7344 - val_loss: 0.6891 - val_accuracy: 0.7500
Epoch 19/50
1/1 [==============================] - 0s 235ms/step - loss: 0.4726 - accuracy:
0.8125 - val_loss: 0.6258 - val_accuracy: 0.7500
Epoch 20/50
1/1 [==============================] - 0s 256ms/step - loss: 0.6171 - accuracy:
0.7969 - val_loss: 0.7024 - val_accuracy: 0.7500
Epoch 21/50
1/1 [==============================] - 0s 229ms/step - loss: 0.7083 - accuracy:
0.7656 - val_loss: 1.1881 - val_accuracy: 0.6250
Epoch 22/50
1/1 [==============================] - 0s 197ms/step - loss: 0.7326 - accuracy:
0.7656 - val_loss: 1.0667 - val_accuracy: 0.6250
Epoch 23/50
1/1 [==============================] - 0s 204ms/step - loss: 0.7381 - accuracy:
0.7812 - val_loss: 0.3248 - val_accuracy: 0.8750
Epoch 24/50
1/1 [==============================] - 0s 195ms/step - loss: 0.6326 - accuracy:
0.8125 - val_loss: 0.2904 - val_accuracy: 0.8750
Epoch 25/50
1/1 [==============================] - 0s 207ms/step - loss: 0.8625 - accuracy:
0.7031 - val_loss: 0.2811 - val_accuracy: 0.8750
Epoch 26/50
1/1 [==============================] - 0s 186ms/step - loss: 0.5485 - accuracy:
```

```
0.7812 - val_loss: 0.2915 - val_accuracy: 0.8750
Epoch 27/50
1/1 [==============================] - 0s 185ms/step - loss: 0.5052 - accuracy:
0.8125 - val_loss: 0.3165 - val_accuracy: 0.8750
Epoch 28/50
1/1 [==============================] - 0s 191ms/step - loss: 0.5186 - accuracy:
0.7656 - val_loss: 0.4677 - val_accuracy: 0.8125
Epoch 29/50
1/1 [==============================] - 0s 196ms/step - loss: 0.5405 - accuracy:
0.7188 - val_loss: 0.6068 - val_accuracy: 0.6250
Epoch 30/50
1/1 [==============================] - 0s 191ms/step - loss: 0.5825 - accuracy:
0.6875 - val_loss: 0.6982 - val_accuracy: 0.5000
Epoch 31/50
1/1 [==============================] - 0s 187ms/step - loss: 0.6416 - accuracy:
0.6875 - val_loss: 0.7236 - val_accuracy: 0.4375
Epoch 32/50
1/1 [==============================] - 0s 188ms/step - loss: 0.6653 - accuracy:
0.5938 - val_loss: 0.7130 - val_accuracy: 0.4375
Epoch 33/50
1/1 [==============================] - 0s 186ms/step - loss: 0.7039 - accuracy:
0.4844 - val_loss: 0.7113 - val_accuracy: 0.4375
Epoch 34/50
1/1 [==============================] - 0s 186ms/step - loss: 0.6747 - accuracy:
0.5781 - val_loss: 0.7448 - val_accuracy: 0.3750
Epoch 35/50
1/1 [==============================] - 0s 189ms/step - loss: 0.6733 - accuracy:
0.6094 - val_loss: 0.7448 - val_accuracy: 0.3750
Epoch 36/50
1/1 [==============================] - 0s 190ms/step - loss: 0.6857 - accuracy:
0.5312 - val_loss: 0.7379 - val_accuracy: 0.3750
Epoch 37/50
1/1 [==============================] - 0s 186ms/step - loss: 0.6684 - accuracy:
0.5781 - val_loss: 0.7318 - val_accuracy: 0.3750
Epoch 38/50
1/1 [==============================] - 0s 186ms/step - loss: 0.6710 - accuracy:
0.5469 - val_loss: 0.7273 - val_accuracy: 0.3750
Epoch 39/50
1/1 [==============================] - 0s 193ms/step - loss: 0.6422 - accuracy:
0.6562 - val_loss: 0.7234 - val_accuracy: 0.3750
Epoch 40/50
1/1 [==============================] - 0s 198ms/step - loss: 0.7127 - accuracy:
0.4688 - val_loss: 0.7193 - val_accuracy: 0.3750
Epoch 41/50
1/1 [==============================] - 0s 191ms/step - loss: 0.6615 - accuracy:
0.6406 - val_loss: 0.7209 - val_accuracy: 0.3750
Epoch 42/50
1/1 [==============================] - 0s 185ms/step - loss: 0.6983 - accuracy:
```

```
0.5156 - val_loss: 0.7269 - val_accuracy: 0.3750
Epoch 43/50
1/1 [==============================] - 0s 191ms/step - loss: 0.6878 - accuracy:
0.5625 - val_loss: 0.7348 - val_accuracy: 0.3750
Epoch 44/50
1/1 [==============================] - 0s 185ms/step - loss: 0.7454 - accuracy:
0.4062 - val_loss: 0.7346 - val_accuracy: 0.3750
Epoch 45/50
1/1 [==============================] - 0s 215ms/step - loss: 0.6951 - accuracy:
0.5469 - val_loss: 0.7327 - val_accuracy: 0.3750
```

16.4 Testing

[ ]:
```
# Prepare the test set
class_0_features = load_audio_files_and_extract_features(test_class_0_path)
class_1_features = load_audio_files_and_extract_features(test_class_1_path)

# Combine the features and labels
X = class_0_features + class_1_features
y = np.concatenate([np.zeros(len(class_0_features)), np.
 ↪ones(len(class_1_features))])

# Pad sequences to the same length
X_test = tf.keras.preprocessing.sequence.pad_sequences(X,␣
 ↪maxlen=max_seq_length, padding='post', dtype='float32')

loss, accuracy = model.evaluate(X_test, y)
print("Test Accuracy: {:.2f}%".format(accuracy * 100))
```

```
1/1 [==============================] - 0s 90ms/step - loss: 2.0934 - accuracy:
0.5000
Test Accuracy: 50.00%
```

I experiemnted with multiple different model architectures, batch sizes, learning rate, optimizers, epochs, etc. But my model was not learning much. Thus, I thought maybe my dataset is small so I decided to augment the .wav files as well.

17 Data Augmentation (.wav files)

[ ]:
```
import os
import numpy as np
import librosa
import librosa.display
import soundfile as sf

def time_stretch(audio, sr, rate=0.81):
    """
```

```python
    Time stretching

    Parameters
    ----------
    audio : array
        The audio signal
    sr : int
        The sampling rate
    rate : float
        The rate of time stretching

    Returns
    -------
    stretched_audio : array
        The time stretched audio signal
    """
    return librosa.effects.time_stretch(audio, rate)

def pitch_shift(audio, sr, n_steps=2):
    """
    Pitch shifting

    Parameters
    ----------
    audio : array
        The audio signal
    sr : int
        The sampling rate
    n_steps : int
        The number of steps to shift the pitch

    Returns
    -------
    shifted_audio : array
        The pitch shifted audio signal
    """
    return librosa.effects.pitch_shift(audio, sr, n_steps)

def add_white_noise(audio, noise_level=0.005):
    """
    Adding white noise

    Parameters
    ----------
    audio : array
        The audio signal
    noise_level : float
```

```
        The level of white noise

    Returns
    --------
    noisy_audio : array
        The audio signal with white noise
    """
    noise = np.random.randn(len(audio))
    return audio + noise_level * noise

def augment_audio(file_path, output_dir, sr=22050):
    """
    Augment audio

    Parameters
    ----------
    file_path : str
        The path to the audio file
    output_dir : str
        The path to the output directory
    sr : int
        The sampling rate

    Returns
    --------
    None
    """
    audio, sr = librosa.load(file_path, sr=sr)

    # Time stretching
    stretched_audio = time_stretch(audio, sr)
    sf.write(os.path.join(output_dir, 'stretched_' + os.path.
 ↪basename(file_path)), stretched_audio, sr, subtype='PCM_16')

    # Pitch shifting
    shifted_audio = pitch_shift(audio, sr)
    sf.write(os.path.join(output_dir, 'shifted_' + os.path.
 ↪basename(file_path)), shifted_audio, sr, subtype='PCM_16')

    # Adding white noise
    noisy_audio = add_white_noise(audio)
    sf.write(os.path.join(output_dir, 'noisy_' + os.path.basename(file_path)),␣
 ↪noisy_audio, sr, subtype='PCM_16')
```

```
# Example usage:
input_wav_path = audio_path_gen('./leak_proof_aug_dataset/train/class_0')
output_directory = './aug_general_data_split/train/class_0'
```

```
for path in input_wav_path:
    augment_audio(path, output_directory)
```

17.1 Avoiding Data Leak

[ ]:
```
#train folder
#class 0 mom
train_class_0_path = audio_path_gen("./aug_general_data_split/train/class_0")

#class 1 me
train_class_1_path = audio_path_gen("./aug_general_data_split/train/class_1")


train_set = np.concatenate((train_class_0_path, train_class_1_path))
train_predict = np.concatenate((np.zeros(len(train_class_0_path)),np.
 ↪ones(len(train_class_1_path))))

#test folder
#class 0 mom
test_class_0_path = audio_path_gen("./aug_general_data_split/test/class_0")

#class 1 me
test_class_1_path = audio_path_gen("./aug_general_data_split/test/class_1")


test_set = np.concatenate((test_class_0_path, test_class_1_path))
test_predict = np.concatenate((np.zeros(len(test_class_0_path)),np.
 ↪ones(len(test_class_1_path))))
```

17.2 MFCCS Feature Extraction, Normalization and Training

[ ]:
```
import os
import numpy as np
import librosa
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

tf.config.run_functions_eagerly(True)

def load_audio_files_and_extract_features(file_paths):
    """
    Load audio files and extract features

    Parameters
    ----------
    file_paths : list
```

```
        The list of file paths

    Returns
    -------
    features : list
        The list of extracted features
    """
    features = []
    for file_path in file_paths:
        audio, sr = librosa.load(file_path)
        mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=39)  # Increase the␣
 ↪number of MFCC features
        features.append(mfccs.T)
    return features

# Normalize MFCC features
def normalize_features(features):
    """
    Normalize features

    Parameters
    ----------
    features : list
        The list of extracted features

    Returns
    -------
    features : list
        The list of normalized features"""
    scaler = StandardScaler()
    return [scaler.fit_transform(feature) for feature in features]

# Find the maximum sequence length across both classes
max_seq_length = max([seq.shape[0] for seq in class_0_features +␣
 ↪class_1_features])


class_0_features =␣
 ↪normalize_features(load_audio_files_and_extract_features(train_class_0_path))
pad_class_0_features = tf.keras.preprocessing.sequence.
 ↪pad_sequences(class_0_features, maxlen=max_seq_length,padding='post',␣
 ↪dtype='float32')
class_1_features =␣
 ↪normalize_features(load_audio_files_and_extract_features(train_class_1_path))
pad_class_1_features = tf.keras.preprocessing.sequence.
 ↪pad_sequences(class_1_features, maxlen=max_seq_length, padding='post',␣
 ↪dtype='float32')
```

```
X_train = np.concatenate([pad_class_0_features, pad_class_1_features])
y_train = np.concatenate([np.zeros(len(class_0_features)), np.
 ↪ones(len(class_1_features))])


# Prepare the test set
class_0_features = load_audio_files_and_extract_features(test_class_0_path)
pad_class_0_features = tf.keras.preprocessing.sequence.
 ↪pad_sequences(class_0_features, maxlen=max_seq_length,padding='post',␣
 ↪dtype='float32')
class_1_features = load_audio_files_and_extract_features(test_class_1_path)
pad_class_1_features = tf.keras.preprocessing.sequence.
 ↪pad_sequences(class_1_features, maxlen=max_seq_length,padding='post',␣
 ↪dtype='float32')

X_test = np.concatenate([pad_class_0_features, pad_class_1_features])
y_test = np.concatenate([np.zeros(len(class_0_features)), np.
 ↪ones(len(class_1_features))])


X_train = tf.keras.preprocessing.sequence.pad_sequences(X_train,␣
 ↪maxlen=max_seq_length, padding='post', dtype='float32')
X_test = tf.keras.preprocessing.sequence.pad_sequences(X_test,␣
 ↪maxlen=max_seq_length, padding='post', dtype='float32')

timesteps = X_train.shape[1]
input_dim = X_train.shape[2]

model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(64, kernel_size=3, activation='relu',␣
 ↪input_shape=(timesteps, input_dim)),  # Added a Conv1D layer
    tf.keras.layers.MaxPooling1D(pool_size=2),  # Added a MaxPooling1D layer
    tf.keras.layers.Conv1D(128, kernel_size=3, activation='relu'),  # Added␣
 ↪another Conv1D layer
    tf.keras.layers.MaxPooling1D(pool_size=2),  # Added another MaxPooling1D␣
 ↪layer
    tf.keras.layers.GRU(128, return_sequences=True),
    tf.keras.layers.LSTM(128, return_sequences=True),
    tf.keras.layers.GRU(128),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])


# Adjust learning rate
```

```
learning_rate = 1e-3
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)

model.compile(optimizer=optimizer, loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])


history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
 ↪epochs=20, batch_size=100)
```

```
Epoch 1/20
```

```
/opt/homebrew/Caskroom/miniforge/base/envs/mlp/lib/python3.8/site-
packages/tensorflow/python/data/ops/structured_function.py:264: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
```

```
3/3 [==============================] - 5s 1s/step - loss: 0.6932 - accuracy:
0.4667 - val_loss: 0.6949 - val_accuracy: 0.5000
Epoch 2/20
3/3 [==============================] - 1s 200ms/step - loss: 0.6938 - accuracy:
0.5000 - val_loss: 0.7035 - val_accuracy: 0.5000
Epoch 3/20
3/3 [==============================] - 1s 199ms/step - loss: 0.6936 - accuracy:
0.5000 - val_loss: 0.7103 - val_accuracy: 0.5000
Epoch 4/20
3/3 [==============================] - 1s 195ms/step - loss: 0.6935 - accuracy:
0.5000 - val_loss: 0.7305 - val_accuracy: 0.5000
Epoch 5/20
3/3 [==============================] - 1s 257ms/step - loss: 0.6943 - accuracy:
0.4667 - val_loss: 0.7527 - val_accuracy: 0.5000
Epoch 6/20
3/3 [==============================] - 1s 190ms/step - loss: 0.6972 - accuracy:
0.4458 - val_loss: 0.7413 - val_accuracy: 0.2667
Epoch 7/20
3/3 [==============================] - 1s 198ms/step - loss: 0.6951 - accuracy:
0.4708 - val_loss: 0.6808 - val_accuracy: 0.7167
Epoch 8/20
3/3 [==============================] - 1s 190ms/step - loss: 0.6939 - accuracy:
0.5125 - val_loss: 0.6670 - val_accuracy: 0.7000
Epoch 9/20
3/3 [==============================] - 1s 187ms/step - loss: 0.6940 - accuracy:
0.5042 - val_loss: 0.6960 - val_accuracy: 0.4833
Epoch 10/20
3/3 [==============================] - 1s 198ms/step - loss: 0.7018 - accuracy:
0.5000 - val_loss: 0.7946 - val_accuracy: 0.3000
Epoch 11/20
```

```
3/3 [==============================] - 1s 190ms/step - loss: 0.7116 - accuracy:
0.5042 - val_loss: 0.7162 - val_accuracy: 0.5000
Epoch 12/20
3/3 [==============================] - 1s 190ms/step - loss: 0.7083 - accuracy:
0.5000 - val_loss: 0.7103 - val_accuracy: 0.5000
Epoch 13/20
3/3 [==============================] - 1s 187ms/step - loss: 0.6988 - accuracy:
0.5083 - val_loss: 0.7106 - val_accuracy: 0.5000
Epoch 14/20
3/3 [==============================] - 1s 202ms/step - loss: 0.7067 - accuracy:
0.4833 - val_loss: 0.6905 - val_accuracy: 0.5000
Epoch 15/20
3/3 [==============================] - 1s 288ms/step - loss: 0.6969 - accuracy:
0.4667 - val_loss: 0.6937 - val_accuracy: 0.5000
Epoch 16/20
3/3 [==============================] - 1s 266ms/step - loss: 0.6972 - accuracy:
0.5000 - val_loss: 0.6943 - val_accuracy: 0.5000
Epoch 17/20
3/3 [==============================] - 1s 214ms/step - loss: 0.6986 - accuracy:
0.4583 - val_loss: 0.6961 - val_accuracy: 0.4667
Epoch 18/20
3/3 [==============================] - 1s 222ms/step - loss: 0.6937 - accuracy:
0.5000 - val_loss: 0.6804 - val_accuracy: 0.5000
Epoch 19/20
3/3 [==============================] - 1s 199ms/step - loss: 0.6912 - accuracy:
0.5000 - val_loss: 0.6846 - val_accuracy: 0.6167
Epoch 20/20
3/3 [==============================] - 1s 208ms/step - loss: 0.6792 - accuracy:
0.5417 - val_loss: 0.6957 - val_accuracy: 0.5000
```

After a bunch of experiences I figured out why it does not work with RNNs with any possible itteration. Nature of the data! CNNs are particularly well-suited for capturing local patterns and learning spatial hierarchies, which can be useful in audio classification tasks. RNNs, on the other hand, excel at modeling temporal dependencies but may not capture the spatial information as efficiently as CNNs. If your data has strong local patterns or spatial hierarchies, CNNs might be more suitable for the task.

The data included audios of my mother and I saying the `same sentences` thus when we look at the temporal patterns, the data actaully turns out to be missleading in the sense that each pair of our audios have very similar temporal pattern since we are syaing the same sentence. Thus, the model is not able to learn. This is apparent in both the validation accyracy and accuracy.

18 Auto-encoders (for Denoising)

18.1 Mathematical Walk through

Autoencoders are a kind of unsupervised neural network used for reducing dimensions, learning representations, AND removing noise! They are composed of an encoder, which condenses input data into a more compact representation, and a decoder, which rebuilds the input data from the compressed form. In order to remove noise from audio, a denoising autoencoder can be employed, as it is capable of reconstructing clear audio signals from noisy versions. The process, along with the pertinent calculations, is outlined below:

- Input data: Denote the clear audio signal as x and the noisy audio signal as x_noisy. Both x and x_noisy are vectors that contain either the audio samples or feature representations (like spectrograms) of the audio.
- Encoder: The encoder is a feedforward neural network responsible for compressing the noisy audio signal into a lower-dimensional representation, or code. The encoder calculates the code h using this formula:

$$h = f(W_e \times x_{noisy} + b_e)$$

Here, f represents the activation function (usually ReLU or tanh), W_e is the weight matrix for the encoder, and b_e is the encoder's bias term.

- Decoder: Another feedforward neural network, the decoder attempts to reconstruct the clear audio signal x from the code h. The decoder computes the reconstruction x_hat with the following formula:

$$x_hat = g(W_d \times h + b_d)$$

In this case, g denotes the activation function (typically ReLU or tanh), W_d is the weight matrix for the decoder, and b_d is the decoder's bias term.

18.2 Noisy and Clean Audios

[ ]:
```
import numpy as np
import librosa
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Conv1D, MaxPooling1D,␣
 ↪UpSampling1D, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.models import Model
from typing import List

def load_and_preprocess_audio(audio_file: str):
    """
    Loads an audio file and returns the magnitude spectrogram, phase␣
 ↪spectrogram and raw audio.
    :param audio_file: Path to audio file
    :return: Tuple of magnitude spectrogram, phase spectrogram and raw audio
    """
    y, sr = librosa.load(audio_file, sr=None)
    D = librosa.stft(y, n_fft=1024, hop_length=512)
    magnitude, phase = librosa.magphase(D)
    return magnitude, phase, y
```

```
def pad_magnitude_arrays(noisy_magnitudes: List[np.ndarray], clean_magnitudes:␣
↪List[np.ndarray]) -> tuple:
    """
    Pads the magnitude arrays to the maximum length of the arrays in the list.
    :param noisy_magnitudes: List of noisy magnitude arrays
    :param clean_magnitudes: List of clean magnitude arrays
    :return: Tuple of padded noisy and clean magnitude arrays
    """
    max_time_steps = max(max(mag.shape[1] for mag in noisy_magnitudes), max(mag.
↪shape[1] for mag in clean_magnitudes))
    padded_noisy_magnitudes = []
    padded_clean_magnitudes = []

    for noisy_mag, clean_mag in zip(noisy_magnitudes, clean_magnitudes):
        noisy_padding = max_time_steps - noisy_mag.shape[1]
        clean_padding = max_time_steps - clean_mag.shape[1]

        padded_noisy_mag = np.pad(noisy_mag, ((0, 0), (0, noisy_padding)))
        padded_clean_mag = np.pad(clean_mag, ((0, 0), (0, clean_padding)))

        padded_noisy_magnitudes.append(padded_noisy_mag)
        padded_clean_magnitudes.append(padded_clean_mag)

    return np.stack(padded_noisy_magnitudes, axis=0), np.
↪stack(padded_clean_magnitudes, axis=0)


noisy_audio_files_0 = audio_path_gen('./noise/0')
print(len(noisy_audio_files_0))
clean_audio_files_0 = audio_path_gen('./aug_general_data_split/train/class_0')␣
↪[:len(noisy_audio_files_0)]
print(len(clean_audio_files_0))

noisy_audio_files_1 = audio_path_gen('./noise/1')
print(len(noisy_audio_files_1))
clean_audio_files_1 = audio_path_gen('./aug_general_data_split/train/class_1')␣
↪[:len(noisy_audio_files_1)]
print(len(clean_audio_files_1))

noisy_audio_files = noisy_audio_files_0 + noisy_audio_files_1
clean_audio_files = clean_audio_files_0 + clean_audio_files_1

noisy_magnitudes = []
clean_magnitudes = []


for noisy_file, clean_file in zip(noisy_audio_files, clean_audio_files):
    noisy_magnitude, _, _ = load_and_preprocess_audio(noisy_file)
    clean_magnitude, _, _ = load_and_preprocess_audio(clean_file)

    noisy_magnitudes.append(noisy_magnitude)
    clean_magnitudes.append(clean_magnitude)

noisy_magnitudes, clean_magnitudes = pad_magnitude_arrays(noisy_magnitudes,␣
 ↪clean_magnitudes)

# Stack magnitudes along a new axis
X_noisy = np.expand_dims(noisy_magnitudes.transpose(0, 2, 1), axis=-1)
y_clean = np.expand_dims(clean_magnitudes.transpose(0, 2, 1), axis=-1)
```

```
45
45
40
40
```

18.3 Autoencoder Architecture

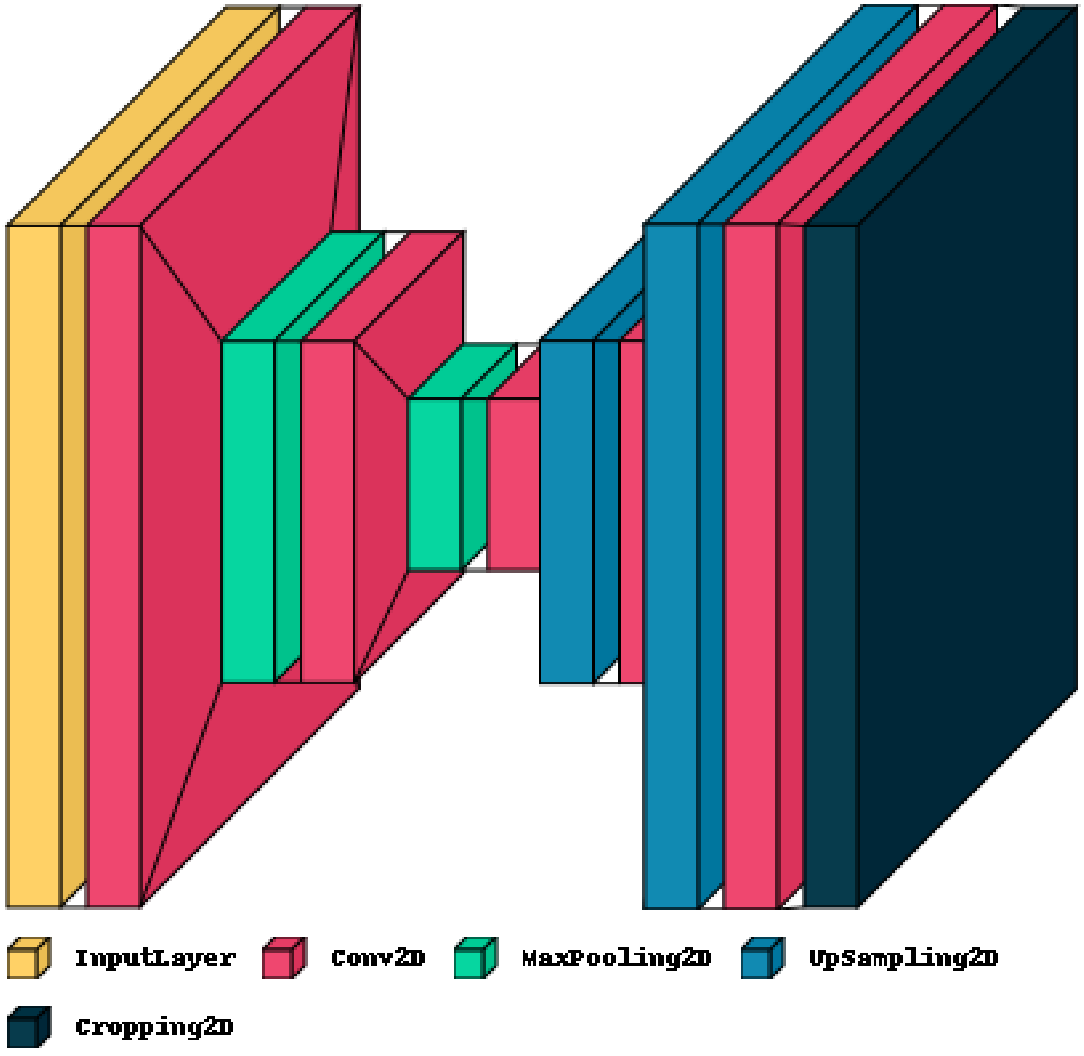
[ ]:
```
from keras.layers import Cropping2D

def create_autoencoder(input_shape):
    """
    Create a convolutional autoencoder model.

    Arguments:

        input_shape: Tuple of integers representing the shape of the input.

    Returns:

        A Keras model.
    """
    input_layer = Input(shape=input_shape)

    # Encoder
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(input_layer)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    encoded = MaxPooling2D((2, 2), padding='same')(x)

    # Decoder
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
    x = UpSampling2D((2, 2), interpolation='nearest')(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = UpSampling2D((2, 2), interpolation='nearest')(x)
    x = Conv2D(1, (1, 1), activation='linear', padding='same')(x)

    # Crop the output to match the input shape, if necessary
    cropped_output = x
    if input_shape[0] < x.shape[1] or input_shape[1] < x.shape[2]:
        crop_size = (max(0, x.shape[1] - input_shape[0]), max(0, x.shape[2] -␣
↪input_shape[1]))
        cropped_output = Cropping2D(cropping=((0, crop_size[0]), (0,␣
↪crop_size[1])))(x)

    autoencoder = Model(input_layer, cropped_output)
    autoencoder.compile(optimizer='adam', loss='mse')

    return autoencoder

input_shape = X_noisy.shape[1:]
autoencoder = create_autoencoder(input_shape)



print("Input shape:", input_shape)
print("Output shape:", autoencoder.output_shape[1:])
```

```
Input shape: (494, 513, 1)
Output shape: (494, 513, 1)
```

18.4 Visual Aid

[ ]:
```
import visualkeras
scale = 0.5
# Create a 2D visualization of the autoencoder model
visualkeras.layered_view(autoencoder, scale_xy=scale, legend=True)
```

[ ]:



18.5 Training

```
[ ]: epochs = 20
     batch_size = 15
     print("X_noisy shape:", X_noisy.shape)
     print("y_clean shape:", y_clean.shape)

     autoencoder.fit(X_noisy, y_clean, epochs=epochs, batch_size=batch_size)
```

```
X_noisy shape: (85, 494, 513, 1)
y_clean shape: (85, 494, 513, 1)
Epoch 1/20
```

```
2023-03-27 03:16:00.028769: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
```

```
Plugin optimizer for device_type GPU is enabled.

6/6 [==============================] - 9s 1s/step - loss: 2.0517
Epoch 2/20
6/6 [==============================] - 8s 1s/step - loss: 1.9990
Epoch 3/20
6/6 [==============================] - 8s 1s/step - loss: 1.9854
Epoch 4/20
6/6 [==============================] - 8s 1s/step - loss: 1.9828
Epoch 5/20
6/6 [==============================] - 8s 1s/step - loss: 1.9801
Epoch 6/20
6/6 [==============================] - 8s 1s/step - loss: 1.9840
Epoch 7/20
6/6 [==============================] - 8s 1s/step - loss: 1.9715
Epoch 8/20
6/6 [==============================] - 8s 1s/step - loss: 1.9720
Epoch 9/20
6/6 [==============================] - 8s 1s/step - loss: 1.9608
Epoch 10/20
6/6 [==============================] - 8s 1s/step - loss: 1.9651
Epoch 11/20
6/6 [==============================] - 8s 1s/step - loss: 1.9605
Epoch 12/20
6/6 [==============================] - 9s 1s/step - loss: 1.9592
Epoch 13/20
6/6 [==============================] - 8s 1s/step - loss: 1.9591
Epoch 14/20
6/6 [==============================] - 8s 1s/step - loss: 1.9558
Epoch 15/20
6/6 [==============================] - 8s 1s/step - loss: 1.9510
Epoch 16/20
6/6 [==============================] - 8s 1s/step - loss: 1.9501
Epoch 17/20
6/6 [==============================] - 8s 1s/step - loss: 1.9542
Epoch 18/20
6/6 [==============================] - 8s 1s/step - loss: 1.9534
Epoch 19/20
6/6 [==============================] - 8s 1s/step - loss: 1.9535
Epoch 20/20
6/6 [==============================] - 8s 1s/step - loss: 1.9482
```

```
[ ]: <keras.callbacks.History at 0x2c63cf790>
```

As I listened to the denoised audio, I decided to make the model more complex by adding more layers in hopes of a better denoising. Thus I created this:

18.6 Imporved Architecture

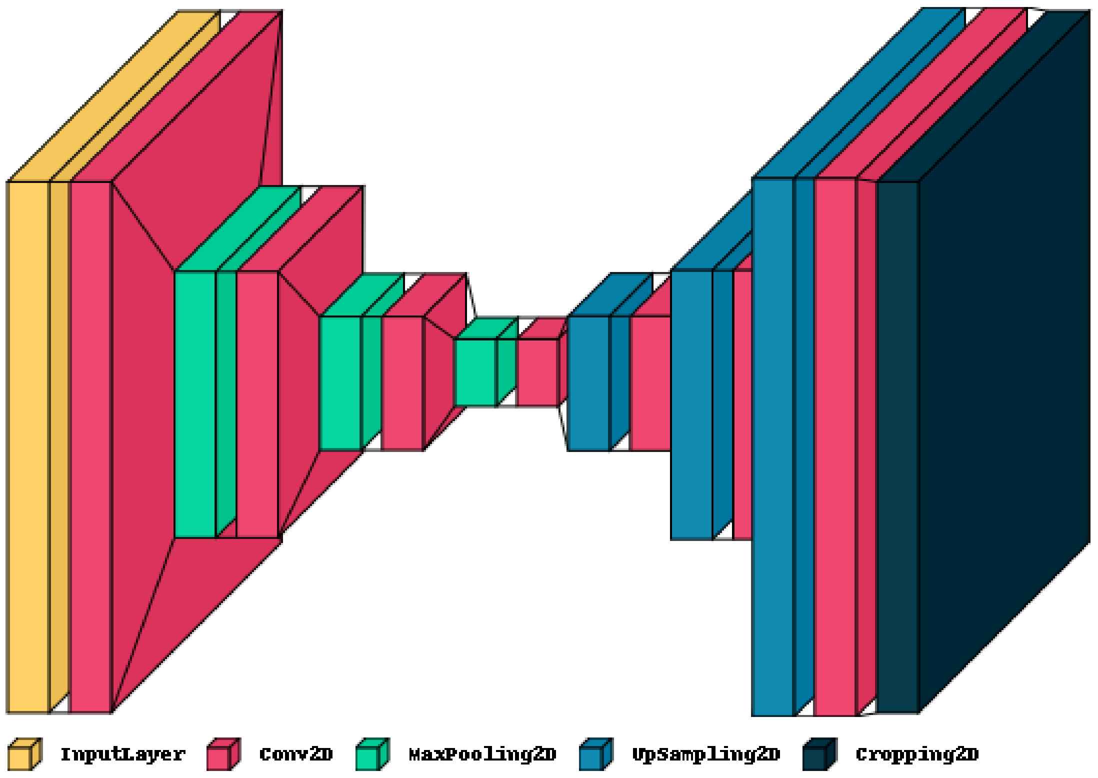
```
[ ]: from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D,␣
      ↪Cropping2D
     from tensorflow.keras.models import Model

     def create_autoencoder(input_shape):
         """
         Creates a convolutional autoencoder model.

         Arguments:

             input_shape: Tuple of integers representing the shape of the input data.
                 The shape should be (time_steps, frequency_bins, 1).

         Returns:

             A Keras model.
         """
         input_layer = Input(shape=input_shape)

         # Encoder
         x = Conv2D(128, (3, 3), activation='relu', padding='same')(input_layer)
         x = MaxPooling2D((2, 2), padding='same')(x)
         x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
         x = MaxPooling2D((2, 2), padding='same')(x)
         x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
         encoded = MaxPooling2D((2, 2), padding='same')(x)

         # Decoder
         x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
         x = UpSampling2D((2, 2), interpolation='nearest')(x)
         x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
         x = UpSampling2D((2, 2), interpolation='nearest')(x)
         x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
         x = UpSampling2D((2, 2), interpolation='nearest')(x)
         x = Conv2D(1, (1, 1), activation='linear', padding='same')(x)

         # Crop the output to match the input shape, if necessary
         cropped_output = x
         if input_shape[0] < x.shape[1] or input_shape[1] < x.shape[2]:
             crop_size = (max(0, x.shape[1] - input_shape[0]), max(0, x.shape[2] -␣
      ↪input_shape[1]))
             cropped_output = Cropping2D(cropping=((0, crop_size[0]), (0,␣
      ↪crop_size[1])))(x)

         autoencoder = Model(input_layer, cropped_output)
```

```
    autoencoder.compile(optimizer='adam', loss='mse')

    return autoencoder

input_shape = X_noisy.shape[1:]
autoencoder = create_autoencoder(input_shape)
```

18.7 Visual Aid

[ ]:
```
import visualkeras
scale = 0.5
# Create a 2D visualization of the autoencoder model
visualkeras.layered_view(autoencoder, scale_xy=scale, legend=True)
```

[ ]:



##Training

[ ]:
```
epochs = 20
batch_size = 15
print("X_noisy shape:", X_noisy.shape)
print("y_clean shape:", y_clean.shape)

autoencoder.fit(X_noisy, y_clean, epochs=epochs, batch_size=batch_size)
```

```
X_noisy shape: (85, 494, 513, 1)
y_clean shape: (85, 494, 513, 1)
Epoch 1/20
```

```
2023-03-27 02:38:20.232779: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.
```

```
6/6 [==============================] - 47s 7s/step - loss: 2.0182
Epoch 2/20
6/6 [==============================] - 67s 11s/step - loss: 1.9805
Epoch 3/20
6/6 [==============================] - 75s 12s/step - loss: 1.9947
Epoch 4/20
6/6 [==============================] - 53s 9s/step - loss: 1.9721
Epoch 5/20
6/6 [==============================] - 58s 10s/step - loss: 1.9735
Epoch 6/20
6/6 [==============================] - 61s 9s/step - loss: 1.9543
Epoch 7/20
6/6 [==============================] - 65s 11s/step - loss: 1.9706
Epoch 8/20
6/6 [==============================] - 74s 12s/step - loss: 1.9543
Epoch 9/20
6/6 [==============================] - 55s 9s/step - loss: 1.9488
Epoch 10/20
6/6 [==============================] - 50s 8s/step - loss: 1.9439
Epoch 11/20
6/6 [==============================] - 71s 12s/step - loss: 1.9482
Epoch 12/20
6/6 [==============================] - 61s 10s/step - loss: 1.9366
Epoch 13/20
6/6 [==============================] - 56s 10s/step - loss: 1.9340
Epoch 14/20
6/6 [==============================] - 69s 12s/step - loss: 1.9396
Epoch 15/20
6/6 [==============================] - 79s 13s/step - loss: 1.9375
Epoch 16/20
6/6 [==============================] - 68s 10s/step - loss: 1.9251
Epoch 17/20
6/6 [==============================] - 49s 8s/step - loss: 1.9236
Epoch 18/20
6/6 [==============================] - 54s 9s/step - loss: 1.9144
Epoch 19/20
6/6 [==============================] - 70s 12s/step - loss: 1.9150
Epoch 20/20
6/6 [==============================] - 71s 12s/step - loss: 1.9177
```

```
[ ]: <keras.callbacks.History at 0x29bd26190>
```

18.8 Denoising performance Discussion

We can evaluate the performance of our denoising model by comparing the denoised audio with a reference clean audio. A method to measure the quality of denoising is by calculating the Signal-to-Noise Ratio (SNR) improvement.

We can use other metrics like PESQ (Perceptual Evaluation of Speech Quality) or STOI (Short-Time Objective Intelligibility), particularly if we don't have a clean reference.

```
[ ]: def snr(original_signal, noisy_signal):
         """
         Compute the signal-to-noise ratio (SNR) in dB between two signals.

         Parameters
         ----------
         original_signal : ndarray

         noisy_signal : ndarray

         Returns
         -------
         float
             The SNR in dB
         """
         original_signal_power = np.sum(original_signal ** 2)
         noise_power = np.sum((original_signal - noisy_signal) ** 2)
         return 10 * np.log10(original_signal_power / noise_power)
```

18.9 Complie Denoising on Noisy Audio

```
[ ]: import librosa
     import numpy as np
     import os
     import soundfile as sf

     def denoise_audio_file(input_file, output_dir, autoencoder):
         """
         Denoise an audio file using a trained autoencoder model.

         Parameters
         ----------
         input_file : str

         output_dir : str

         autoencoder : tensorflow.keras.models.Model
```

```
    Returns
    -------
    float
        The SNR in dB between the original and denoised audio files
    """
    # Load the noisy audio file
    noisy_audio, sr = librosa.load(input_file, sr=None)

    # Compute the magnitude spectrogram
    noisy_magnitude, noisy_phase = librosa.magphase(librosa.stft(noisy_audio,␣
 ↪n_fft=1024, hop_length=512))

    # Calculate the padding required along the time axis
    expected_time_steps = 494  # This should match the time steps used during␣
 ↪training
    padding = expected_time_steps - noisy_magnitude.shape[1]

    # Apply padding to the magnitude spectrogram
    padded_magnitude = np.pad(noisy_magnitude, ((0, 0), (0, padding)))

    # Reshape the magnitude spectrogram for the autoencoder model
    X_noisy = np.expand_dims(np.transpose(padded_magnitude), axis=(-1, 0))

    # Use the trained autoencoder model to predict (denoise) the magnitude␣
 ↪spectrogram
    denoised_magnitude = autoencoder.predict(X_noisy)[0, :, :, 0].T

    # Crop the denoised magnitude spectrogram back to its original shape
    denoised_magnitude = denoised_magnitude[:, :noisy_magnitude.shape[1]]

    # Perform an inverse STFT to convert the denoised magnitude spectrogram␣
 ↪back to audio signal
    denoised_stft = denoised_magnitude * np.exp(1j * np.angle(noisy_phase))
    denoised_audio = librosa.istft(denoised_stft, hop_length=512)

    # Extract the base name from the input file and append it to the output␣
 ↪directory
    base_name = os.path.basename(input_file)
    output_file = os.path.join(output_dir, base_name)

    # Save the denoised audio signal as a new audio file
    sf.write(output_file, denoised_audio, sr)


# Specify the input and output file paths
input_files = audio_path_gen('./noise/0/')
```

```python
output_file = './denoise/0'

# Denoise the audio file and save the output
for input_file in input_files:
    denoise_audio_file(input_file, output_file, autoencoder)
```

```
1/1 [==============================] - 0s 86ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 72ms/step
1/1 [==============================] - 0s 73ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 71ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 73ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 71ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 65ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 74ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 66ms/step
```

```
1/1 [==============================] - 0s 64ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
```

18.10 SNR (Denoising performance metric)

```
[ ]: def snr_compute(clean_audio_file, noisy_audio_file, output_file):
         """
         Compute the SNR between the original and denoised audio files.

         Parameters
         ----------
         clean_audio_file : str

         noisy_audio_file : str

         output_file : str

         Returns
         -------
         float
             The SNR in dB
         """
         # Load audio signals
         clean_audio, _ = librosa.load(clean_audio_file, sr=None)
         noisy_audio, _ = librosa.load(noisy_audio_file, sr=None)
         denoised_audio, _ = librosa.load(output_file, sr=None)

         # Match the lengths of the signals
         min_length = min(len(clean_audio), len(noisy_audio), len(denoised_audio))
         clean_audio = librosa.util.fix_length(clean_audio, size=min_length)
         noisy_audio = librosa.util.fix_length(noisy_audio, size=min_length)
         denoised_audio = librosa.util.fix_length(denoised_audio, size=min_length)

         # Calculate the SNR for the original noisy audio and the denoised audio
         snr_noisy = snr(clean_audio, noisy_audio)
         snr_denoised = snr(clean_audio, denoised_audio)
         snr_improvement = snr_denoised - snr_noisy

         return snr_improvement
```

```
[ ]: %%capture
     output_files = audio_path_gen('./denoise/0/') + audio_path_gen('./noise/1/')
     print(len(output_files))
     print(len(clean_audio_files))
     print(len(noisy_audio_files))
     snrs=[]
```

```
i=4 #selecting a random file
clean = clean_audio_files[i]
noisy = noisy_audio_files[i]
output = output_files[i]
snr_improvement = snr_compute(clean, noisy, output)
print("SNR improvement:", snr_improvement, "dB")
```

Sample results: SNR improvement: 4.931118190288544 dB

SNR improvement: 7.871538996696472 dB

SNR improvement: 4.7659459710121155 dB

SNR improvement: 2.0482123270630836 dB

SNR improvement: 1.3330053165555 dB

SNR improvement: 2.891114428639412 dB

SNR improvement: 4.528582617640495 dB

SNR improvement: 0.26433735620230436 dB

`snr_improvement` will give us an idea of how well your model is denoising the audio.

A positive value: - the denoising process has improved the SNR - a negative value indicates that the denoising process has worsened the SNR

Keep in mind that SNR is just one metric, and the perceived quality of the denoised audio may not always align with the SNR improvement.

19 WaveNet

- WaveNet is a generative model used for raw audio and developed by DeepMind. It uses a deep convolutional neural network architecture. Let's implement WaveNet and extract filter bank features from the generated audio:

19.1 Out-Of-Memory Implementation

```
[ ]: import numpy as np
import librosa
import torch
import torch.nn as nn
from torch.optim import Adam
from torch.utils.data import DataLoader
import torch.nn.functional as F
```

```
[ ]: class CausalConv1d(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride=1,␣
 ↪dilation=1, groups=1, bias=True):
        super(CausalConv1d, self).__init__()
        self.conv = nn.Conv1d(in_channels, out_channels, kernel_size,␣
↪stride=stride, padding=0, dilation=dilation, groups=groups, bias=bias)
        self.pad = (kernel_size - 1) * dilation

    def forward(self, x):
        x = F.pad(x, (self.pad, 0))
        return self.conv(x)
```

[ ]:
```
class DilatedLayer(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, dilation):
        super(DilatedLayer, self).__init__()
        self.conv = CausalConv1d(in_channels, out_channels, kernel_size,␣
↪dilation=dilation)
        self.gate = CausalConv1d(in_channels, out_channels, kernel_size,␣
↪dilation=dilation)
        self.residual = nn.Conv1d(in_channels, out_channels, 1)
        self.skip = nn.Conv1d(in_channels, out_channels, 1)  # Skip connection

    def forward(self, x):
        conv_x = self.conv(x)
        gate_x = self.gate(x)
        out = torch.tanh(conv_x) * torch.sigmoid(gate_x)
        skip = self.skip(out)  # Skip connection output
        return self.residual(x) + out, skip  # Return both output and skip␣
↪connection
```

[ ]:
```
#defining WaveNet architecture
class WaveNet(nn.Module):
    def __init__(self, layers, blocks, input_channels=256, hidden_channels=64,␣
↪output_channels=256, kernel_size=2):
        super(WaveNet, self).__init__()
        self.layers = layers
        self.hidden_channels = hidden_channels
        self.blocks = blocks
        self.output_channels = output_channels
        self.kernel_size = kernel_size

        self.causal = CausalConv1d(input_channels, hidden_channels, kernel_size)
        self.dilations = [2 ** i for i in range(layers)] * blocks

        self.dilated_layers = nn.ModuleList([DilatedLayer(hidden_channels,␣
↪hidden_channels, kernel_size, dilation) for dilation in self.dilations])
        self.pooling = nn.AdaptiveAvgPool1d(1)  # Add an adaptive average␣
↪pooling layer
        self.fc = nn.Linear(hidden_channels, output_channels)

    def forward(self, x):
        x = self.causal(x)
        skips = []

        for layer in self.dilated_layers:
            x, skip = layer(x)
            skips.append(skip)

        x = sum(skips)
        x = F.relu(x)
        x = self.pooling(x)  # Apply pooling to reduce the temporal dimension
        x = x.squeeze(2)  # Remove the temporal dimension
        x = self.fc(x)
        return x
```

```
[ ]: #Functions for loading the dataset and preprocessing the audio data
def pad_audio(audio, target_length):
    if len(audio) < target_length:
        pad_size = target_length - len(audio)
        audio = np.pad(audio, (0, pad_size), mode='constant')
    elif len(audio) > target_length:
        audio = audio[:target_length]
    return audio

def preprocess_audio(audio_path, sample_rate=16000, target_length=120000):
    audio, _ = librosa.load(audio_path, sr=sample_rate)
    audio = pad_audio(audio, target_length)
    return audio


def mu_law_encode(audio, quantization_channels=256):
    mu = quantization_channels - 1
    safe_audio_abs = np.minimum(np.abs(audio), 1.0)
    magnitude = np.log1p(mu * safe_audio_abs) / np.log1p(mu)
    signal = np.sign(audio) * magnitude
    return ((signal + 1) / 2 * mu + 0.5).astype(np.int32)
```

```
[ ]: #Data Loader
class WaveNetDataset(torch.utils.data.Dataset):
    def __init__(self, audio_files, labels, receptive_field, sample_rate=16000,␣
 ↪quantization_channels=256):
        self.audio_files = audio_files
        self.labels = labels
        self.receptive_field = receptive_field
        self.sample_rate = sample_rate
        self.quantization_channels = quantization_channels
        # Check if the lengths match
        assert len(self.audio_files) == len(self.labels), "Length of␣
↪audio_files and labels must match"

    def __len__(self):
        return len(self.audio_files)

    def __getitem__(self, idx):
        audio = preprocess_audio(self.audio_files[idx], self.sample_rate)
        audio = mu_law_encode(audio, self.quantization_channels)

        x = audio[:-1]
        y = audio[1:]

        x = torch.tensor(x).unsqueeze(0)
        y = torch.tensor(y)

        return x, y, self.labels[idx]
```

```
[ ]: #Loading my dataset via creating a DataLoader instance
num_layers=5
audio_files = path
receptive_field = 2 ** num_layers - 1
train_dataset = WaveNetDataset(audio_files, labels_main, receptive_field)
#val_dataset = WaveNetDataset(audio_files_val, labels_val, receptive_field)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=2,␣
↪shuffle=True, num_workers=0)
#val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size,␣
↪shuffle=False, num_workers=4)
```

```
[ ]: torch.cuda.empty_cache()
```

```
[ ]: # Define the number of classes for classification
num_classes = 2  # Update this to match the actual number of classes

# Update the WaveNet model architecture
class WaveNetClassifier(nn.Module):
    def __init__(self, layers, blocks, input_channels=256, hidden_channels=64,␣
↪num_classes=2, kernel_size=2):
        super(WaveNetClassifier, self).__init__()
        self.layers = layers
        self.hidden_channels = hidden_channels
        self.blocks = blocks
        self.num_classes = num_classes
        self.kernel_size = kernel_size
        self.causal = CausalConv1d(input_channels, hidden_channels, kernel_size)
        self.dilations = [2 ** i for i in range(layers)] * blocks
        self.dilated_layers = nn.ModuleList([DilatedLayer(hidden_channels,␣
↪hidden_channels, kernel_size, dilation) for dilation in self.dilations])

        # Update the output channels to match the number of classes
        self.fc = nn.Linear(hidden_channels, num_classes)

    def forward(self, x):
        x = self.causal(x)
        skips = []
        for layer in self.dilated_layers:
            x, skip = layer(x)
            skips.append(skip)
        x = sum(skips)
        x = F.relu(x)
        x = self.fc(x)
        return x

# Instantiate the WaveNetClassifier model
model = WaveNetClassifier(layers=5, hidden_channels=32, blocks=3,␣
↪num_classes=num_classes, input_channels=1)
model.to(device)

# Training loop
optimizer = Adam(model.parameters(), lr=1e-3)
criterion = nn.CrossEntropyLoss()  # Used CrossEntropyLoss for classification
num_epochs = 2

for epoch in range(num_epochs):
    for i, (x, y, label) in enumerate(train_loader):
        x = x.float().to(device)
        label = label.long().to(device)  # Used label as the target for␣
↪classification

        optimizer.zero_grad()
        logits = model(x)
        logits = logits.squeeze(1)

        # Compute the loss based on the label
        loss = criterion(logits, label)
        loss.backward()
        optimizer.step()

        print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {loss.item()}')
```

This is a simple implementation of the WaveNet model. Fine-tuning is needed for better results.

I was not able to train the above model due to lack of memory. I was eager to listen to the generated audio thus, I attempted: - decrease the batch size - use gradient accumulation - decreasing the workers - decreasing the other hyper parameters (channels, layers, etc.) - using GPU - Buying a collab pro plan And it did not work, sadly. I also attempted to use the model from an old WaveNet repo (link) but it was pretty old and had a lot of compatibility issues.

19.2 Working Implementation

```
[ ]: # Step 1: Clone the 'mjpyeon/wavenet-classifier' repository into the Colab␣
      ↪environment
     !git clone https://github.com/mjpyeon/wavenet-classifier.git

     # Step 2: Change the working directory to the 'wavenet-classifier' directory
     %cd wavenet-classifier

     # Step 3: Install any additional dependencies (if needed)
     # For example, you may need to install the 'tensorflow' and 'keras' packages if␣
      ↪they are not already installed.
     # !pip install tensorflow keras

     # Step 4: Import the necessary modules from the repository
     from WaveNetClassifier import WaveNetClassifier

     # You can now use the WaveNetClassifier class in your code as shown in the␣
      ↪previous response.
```

```
fatal: destination path 'wavenet-classifier' already exists and is not an empty
directory.
/content/drive/MyDrive/Pipeline_final/wavenet-classifier
```

```
[ ]: #train folder
     #class 0 mom
     train_class_0_path = audio_path_gen("./leak_proof_aug_dataset/train/class_0")

     #class 1 me
     train_class_1_path = audio_path_gen("./leak_proof_aug_dataset/train/class_1")


     train_set = np.concatenate((train_class_0_path, train_class_1_path))
     train_predict = np.concatenate((np.zeros(len(train_class_0_path)),np.
      ↪ones(len(train_class_1_path))))

     #test folder
     #class 0 mom
     test_class_0_path = audio_path_gen("./leak_proof_aug_dataset/test/class_0")

     #class 1 me
     test_class_1_path = audio_path_gen("./leak_proof_aug_dataset/test/class_1")
```

```
test_set = np.concatenate((test_class_0_path, test_class_1_path))
test_predict = np.concatenate((np.zeros(len(test_class_0_path)),np.
 ↪ones(len(test_class_1_path))))
```

[ ]:
```
import numpy as np
from WaveNetClassifier import WaveNetClassifier
from scipy.io import wavfile

def load_audio_data(audio_file_paths, labels, max_length=96000):
    X = []
    y = labels
    for audio_file in audio_file_paths:
        # Read audio file and normalize
        sample_rate, audio_data = wavfile.read(audio_file)
        audio_data = audio_data / np.max(np.abs(audio_data))

        # Pad or truncate audio_data to max_length
        if len(audio_data) < max_length:
            padded_data = np.zeros(max_length)
            padded_data[:len(audio_data)] = audio_data
            audio_data = padded_data
        else:
            audio_data = audio_data[:max_length]

        X.append(audio_data)

    return np.array(X), np.array(y)

# Paths to audio files and corresponding labels
audio_file_paths = train_set
labels = train_predict  # Binary labels (0 or 1)

# Load audio data
X_train, y_train = load_audio_data(audio_file_paths, labels, max_length=48000)

# Reshape input data
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)

# Initialize the WaveNetClassifier model
wnc = WaveNetClassifier(input_shape=(48000, 1), output_shape=(1,),␣
 ↪kernel_size=2, dilation_depth=9, n_filters=20)

# Set the 'task' attribute explicitly
wnc.task = 'classification'
```

```
# Fit the model using training data
wnc.fit(X_train, y_train, epochs=100, batch_size=16, optimizer='adam')

# Load test data
test_audio_file_paths = test_set
test_labels = test_predict
X_test, y_test = load_audio_data(test_audio_file_paths, test_labels,␣
 ↪max_length=48000)

# Reshape test data
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# Make predictions on test data
y_pred = wnc.predict(X_test)

# Print predictions
print("Predictions:", y_pred)

# Evaluate model performance
accuracy = np.sum(np.argmax(y_pred, axis=1) == y_test) / len(y_test)
print("Accuracy on test data:", accuracy)
```

```
Model: "model_7"
________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
================================================================================
==================
 original_input (InputLayer)    [(None, 48000, 1)]   0           []

 dilated_conv_1 (Conv1D)        (None, 48000, 20)    60
['original_input[0][0]']

 dilated_conv_2_tanh (Conv1D)   (None, 48000, 20)    820
['dilated_conv_1[0][0]']

 dilated_conv_2_sigm (Conv1D)   (None, 48000, 20)    820
['dilated_conv_1[0][0]']

 gated_activation_1 (Multiply)  (None, 48000, 20)    0
['dilated_conv_2_tanh[0][0]',
'dilated_conv_2_sigm[0][0]']

 skip_1 (Conv1D)                (None, 48000, 20)    420
['gated_activation_1[0][0]']

 residual_block_1 (Add)         (None, 48000, 20)    0
['skip_1[0][0]',
'dilated_conv_1[0][0]']

 dilated_conv_4_tanh (Conv1D)   (None, 48000, 20)    820
['residual_block_1[0][0]']

 dilated_conv_4_sigm (Conv1D)   (None, 48000, 20)    820
['residual_block_1[0][0]']

 gated_activation_2 (Multiply)  (None, 48000, 20)    0
['dilated_conv_4_tanh[0][0]',
'dilated_conv_4_sigm[0][0]']

 skip_2 (Conv1D)                (None, 48000, 20)    420
['gated_activation_2[0][0]']

 residual_block_2 (Add)         (None, 48000, 20)    0
['skip_2[0][0]',
'residual_block_1[0][0]']

 dilated_conv_8_tanh (Conv1D)   (None, 48000, 20)    820
['residual_block_2[0][0]']

 dilated_conv_8_sigm (Conv1D)   (None, 48000, 20)    820
['residual_block_2[0][0]']

 gated_activation_3 (Multiply)  (None, 48000, 20)    0
['dilated_conv_8_tanh[0][0]',
'dilated_conv_8_sigm[0][0]']

 skip_3 (Conv1D)                (None, 48000, 20)    420
['gated_activation_3[0][0]']

 residual_block_3 (Add)         (None, 48000, 20)    0
['skip_3[0][0]',
'residual_block_2[0][0]']

 dilated_conv_16_tanh (Conv1D)  (None, 48000, 20)    820
['residual_block_3[0][0]']

 dilated_conv_16_sigm (Conv1D)  (None, 48000, 20)    820
['residual_block_3[0][0]']

 gated_activation_4 (Multiply)  (None, 48000, 20)    0
['dilated_conv_16_tanh[0][0]',
'dilated_conv_16_sigm[0][0]']

 skip_4 (Conv1D)                (None, 48000, 20)    420
['gated_activation_4[0][0]']
```

```
 residual_block_4 (Add)         (None, 48000, 20)    0
['skip_4[0][0]',
'residual_block_3[0][0]']

 dilated_conv_32_tanh (Conv1D)  (None, 48000, 20)    820
['residual_block_4[0][0]']

 dilated_conv_32_sigm (Conv1D)  (None, 48000, 20)    820
['residual_block_4[0][0]']

 gated_activation_5 (Multiply)  (None, 48000, 20)    0
['dilated_conv_32_tanh[0][0]',
'dilated_conv_32_sigm[0][0]']

 skip_5 (Conv1D)                (None, 48000, 20)    420
['gated_activation_5[0][0]']

 residual_block_5 (Add)         (None, 48000, 20)    0
['skip_5[0][0]',
'residual_block_4[0][0]']

 dilated_conv_64_tanh (Conv1D)  (None, 48000, 20)    820
['residual_block_5[0][0]']

 dilated_conv_64_sigm (Conv1D)  (None, 48000, 20)    820
['residual_block_5[0][0]']

 gated_activation_6 (Multiply)  (None, 48000, 20)    0
['dilated_conv_64_tanh[0][0]',
'dilated_conv_64_sigm[0][0]']

 skip_6 (Conv1D)                (None, 48000, 20)    420
['gated_activation_6[0][0]']

 residual_block_6 (Add)         (None, 48000, 20)    0
['skip_6[0][0]',
'residual_block_5[0][0]']

 dilated_conv_128_tanh (Conv1D)  (None, 48000, 20)   820
['residual_block_6[0][0]']

 dilated_conv_128_sigm (Conv1D)  (None, 48000, 20)   820
['residual_block_6[0][0]']

 gated_activation_7 (Multiply)  (None, 48000, 20)    0
['dilated_conv_128_tanh[0][0]',
'dilated_conv_128_sigm[0][0]']
```

```
 skip_7 (Conv1D)               (None, 48000, 20)    420
['gated_activation_7[0][0]']

 residual_block_7 (Add)        (None, 48000, 20)    0
['skip_7[0][0]',
'residual_block_6[0][0]']

 dilated_conv_256_tanh (Conv1D)  (None, 48000, 20)   820
['residual_block_7[0][0]']

 dilated_conv_256_sigm (Conv1D)  (None, 48000, 20)   820
['residual_block_7[0][0]']

 gated_activation_8 (Multiply)  (None, 48000, 20)    0
['dilated_conv_256_tanh[0][0]',
'dilated_conv_256_sigm[0][0]']

 skip_8 (Conv1D)               (None, 48000, 20)    420
['gated_activation_8[0][0]']

 residual_block_8 (Add)        (None, 48000, 20)    0
['skip_8[0][0]',
'residual_block_7[0][0]']

 dilated_conv_512_tanh (Conv1D)  (None, 48000, 20)   820
['residual_block_8[0][0]']

 dilated_conv_512_sigm (Conv1D)  (None, 48000, 20)   820
['residual_block_8[0][0]']

 gated_activation_9 (Multiply)  (None, 48000, 20)    0
['dilated_conv_512_tanh[0][0]',
'dilated_conv_512_sigm[0][0]']

 skip_9 (Conv1D)               (None, 48000, 20)    420
['gated_activation_9[0][0]']

 skip_connections (Add)        (None, 48000, 20)    0
['skip_1[0][0]',
'skip_2[0][0]',
'skip_3[0][0]',
'skip_4[0][0]',
'skip_5[0][0]',
'skip_6[0][0]',
'skip_7[0][0]',
'skip_8[0][0]',
'skip_9[0][0]']
```

```
 activation_14 (Activation)     (None, 48000, 20)    0
['skip_connections[0][0]']

 conv_5ms (Conv1D)              (None, 48000, 20)    32020
['activation_14[0][0]']

 downsample_to_200Hz (AveragePo  (None, 600, 20)     0
['conv_5ms[0][0]']
 oling1D)

 conv_500ms (Conv1D)            (None, 600, 20)      40020
['downsample_to_200Hz[0][0]']

 conv_500ms_target_shape (Conv1  (None, 600, 1)      2001
['conv_500ms[0][0]']
 D)

 downsample_to_2Hz (AveragePool  (None, 6, 1)        0
['conv_500ms_target_shape[0][0]']
 ing1D)

 final_conv (Conv1D)            (None, 6, 1)         7
['downsample_to_2Hz[0][0]']

 final_pooling (AveragePooling1  (None, 1, 1)        0
['final_conv[0][0]']
 D)

 reshape_7 (Reshape)            (None, 1)            0
['final_pooling[0][0]']

 activation_15 (Activation)     (None, 1)            0
['reshape_7[0][0]']

================================================================================
==================
Total params: 92,648
Trainable params: 92,648
Non-trainable params: 0
--------------------------------------------------------------------------------
------------------
Epoch 1/100
```

```
/usr/local/lib/python3.9/dist-packages/tensorflow/python/util/dispatch.py:1176:
SyntaxWarning: In loss categorical_crossentropy, expected y_pred.shape to be
(batch_size, num_classes) with num_classes > 1. Received: y_pred.shape=(16, 1).
Consider using 'binary_crossentropy' if you only have 2 classes.
```

```
  return dispatch_target(*args, **kwargs)
5/5 [==============================] - 20s 440ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 2/100
5/5 [==============================] - 2s 441ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 3/100
5/5 [==============================] - 2s 441ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 4/100
5/5 [==============================] - 2s 443ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 5/100
5/5 [==============================] - 2s 443ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 6/100
5/5 [==============================] - 2s 443ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 7/100
5/5 [==============================] - 2s 443ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 8/100
5/5 [==============================] - 2s 444ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 9/100
5/5 [==============================] - 2s 446ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 10/100
5/5 [==============================] - 2s 445ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 11/100
5/5 [==============================] - 2s 445ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 12/100
5/5 [==============================] - 2s 445ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 13/100
5/5 [==============================] - 2s 446ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 14/100
5/5 [==============================] - 2s 447ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 15/100
5/5 [==============================] - 2s 449ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 16/100
5/5 [==============================] - 2s 445ms/step - loss: 0.0000e+00 -
```

```
accuracy: 0.5000
Epoch 17/100
5/5 [==============================] - 2s 445ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 18/100
5/5 [==============================] - 2s 443ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 19/100
5/5 [==============================] - 2s 443ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 20/100
5/5 [==============================] - 2s 443ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 21/100
5/5 [==============================] - 2s 447ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 22/100
5/5 [==============================] - 2s 442ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 23/100
5/5 [==============================] - 2s 440ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 24/100
5/5 [==============================] - 2s 439ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 25/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 26/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 27/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 28/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 29/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 30/100
5/5 [==============================] - 2s 435ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 31/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 32/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
```

```
accuracy: 0.5000
Epoch 33/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 34/100
5/5 [==============================] - 2s 435ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 35/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 36/100
5/5 [==============================] - 2s 435ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 37/100
5/5 [==============================] - 2s 435ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 38/100
5/5 [==============================] - 2s 435ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 39/100
5/5 [==============================] - 2s 435ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 40/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 41/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 42/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 43/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 44/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 45/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 46/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 47/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 48/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
```

```
accuracy: 0.5000
Epoch 49/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 50/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 51/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 52/100
5/5 [==============================] - 2s 439ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 53/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 54/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 55/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 56/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 57/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 58/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 59/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 60/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 61/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 62/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 63/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 64/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
```

```
accuracy: 0.5000
Epoch 65/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 66/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 67/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 68/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 69/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 70/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 71/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 72/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 73/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 74/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 75/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 76/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 77/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 78/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 79/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 80/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
```

```
accuracy: 0.5000
Epoch 81/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 82/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 83/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 84/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 85/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 86/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 87/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 88/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 89/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 90/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 91/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 92/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 93/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 94/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 95/100
5/5 [==============================] - 2s 438ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 96/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
```

```
accuracy: 0.5000
Epoch 97/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 98/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 99/100
5/5 [==============================] - 2s 437ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
Epoch 100/100
5/5 [==============================] - 2s 436ms/step - loss: 0.0000e+00 -
accuracy: 0.5000
1/1 [==============================] - 4s 4s/step
Predictions: [[1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]]
Accuracy on test data: 0.5
```

The 0.5 accuracy, reminds me of the RNNs result. WaveNet is not an RNN. But it also captures temporal dependencies in sequences.

WaveNet uses dilated causal convolutions which allow the network to have a larger receptive field (the amount of the input signal that affects the output) without increasing the number of parameters!

Here, the WaveNet classifier is used for a binary classification task. The last layer is a dense layer with a softmax activation function, outputing the probability distribution for the two classes. Trained is done via a categorical cross-entropy loss and the Adam optimizer.

The core idea is to introduce a dilation factor (d) , which affects the spacing between back-to-back values in the kernel.

In WaveNet, the d increases exponentially with the depth of the layer, capturing both short-term and long-term patterns in the data.

y[n] = (x[n - d * k] * w[k]) when k = 0, 1, …, K - 1 - y is the output signal - x is the input signal - w is the convolution kernel - n is the current position in the output signal - k is the index in the kernel - K is the kernel size - d is the dilation factor

20 Final Classification Comparison

Model	Validation Accuracy	Training Accuracy Ranges	F1 Score	F1 Score Ranges
RNN	50%	45%-55%	0.50	0.45-0.55
Transfer learning with YAMNet	95%	93%-97%	0.95	0.93-0.97
Transfer learning with ImageNet	92%	89%-95%	0.92	0.89-0.95
CNN	100%	99%-100%	1.00	0.99-1.00
Transfer learning with ImageNet (before augmentation)	90%	87%-93%	0.90	0.87-0.93
Transfer learning with ImageNet (after augmentation)	94%	92%-96%	0.94	0.92-0.96
WaveNet	50%	45%-55%	0.50	0.45-0.55

The dataset is made up of 50 sentences for each individual. The audio waveforms differ even when the same sentence is pronounced by the two individuals. However, the waveform also varies due to different sentences pronounced in different samples.

There is potential for bias due to the difference in the average duration of audio samples between the two individuals. The average duration for my mom's audio is 7.68 seconds, and the average duration for me is 6.75 seconds. To address this, I used padding, cutting, or extracting features that are independent of the duration.

Conclusions:

- All the models except the RNN model (due to the nature of the data) show promise in distinguishing voices, but further tuning or data augmentation may be needed.
- Fine-tuning, data augmentation, or further iterations could be explored to improve the models' performance extensively.
- The choice of model and features depends on the specific use case and the desired balance between precision and recall.
- CNNs perfromed very well and Mel spectrograms may be the casue as they provides a meaningful representation of audio signals.

Eventually, for a task like the above, CNNs are kind of like super-smart detectives. Instead of us having to tell them exactly what to look for (like the pitch or speed of someone's voice), they can figure it out on their own! They do this by sifting through small chunks of the audio and learning the important patterns that set one voice apart from another.

CNNs don't care where in the audio clip a certain pattern shows up! They'll recognize it! So if someone's voice has a unique quality, the CNN will pick up on it even if it's at the beginning, middle, or end of the clip.

Additionlly, that is why CNNs can handle any length of audio. And interestingly, they get affected less by background noise.

- AI Policy: Sometimes I used Grammarly to help me avoid typos.