Libraries:

os: Interacts with the operating system.

matplotlib.pyplot: Creates visualizations like graphs.

numpy: Handles numerical computations and arrays.

Librosa: This library is used for both audio and video analysis.

IPython.display: Displays audio in Jupyter/IPython environments.

tensorflow: TensorFlow library is used for machine learning and deep neural networks.

from tensorflow.keras.models import Sequential: Imports the Keras Sequential model, allowing the creation of neural network models layer by layer.

from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPooling2D, Dropout, BatchNormalization: Imports specific layer types (e.g., Conv2D, Dense, Flatten, MaxPooling2D, Dropout, BatchNormalization) used to build neural network architectures.

Sequential(): Initializes a sequential neural network model in Keras, allowing the creation of models layer by layer.

Conv2D(): Creates a 2D convolutional layer used for spatial convolution over images.

Dense(): Adds a fully connected (dense) layer to the neural network model.

Flatten(): Flattens the input, converting multi-dimensional data into a one-dimensional array.

MaxPooling2D(): Performs max pooling operation for reducing the spatial dimensions of the input volume.

Dropout(): Applies dropout regularization to the input, randomly setting a fraction of input units to zero during training to prevent overfitting. Means deactivation of neurons.

BatchNormalization(): Normalizes and scales inputs, aiding faster convergence and better performance in deep neural networks.

```python
def preprocess(file_path, label, sample_rate=16000, max_length=48000, target_shape=(128, 128)):
    # Load the audio file using librosa
    audio_data, sr = librosa.load(file_path, sr=sample_rate, mono=True)

    # Truncate or pad the audio to a specific length
    if len(audio_data) < max_length:
        audio_data = np.pad(audio_data, (0, max_length - len(audio_data)), 'constant')
    else:
        audio_data = audio_data[:max_length]

    # Compute the spectrogram using librosa and resize it
    spectrogram = librosa.feature.melspectrogram(y=audio_data, sr=sample_rate, n_fft=512,
                                                 hop_length=256, win_length=320)
    spectrogram = librosa.power_to_db(spectrogram, ref=np.max)

    # Resize the spectrogram
    spectrogram = np.resize(spectrogram, target_shape)  # Resizing to target_shape

    # Expand dimensions to match the TensorFlow output format
    spectrogram = np.expand_dims(spectrogram, axis=2)

    return spectrogram, label
```

file_path: Path to the audio file.

label: A label associated with the audio file (e.g., class label for classification tasks).

sample_rate: Desired sampling rate for loading the audio (default set to 16000 Hz).

max_length: Maximum length of audio data after truncation or padding (default set to 48000 samples).

target_shape: Desired shape of the spectrogram.

Librosa: loads the audio data which gives the result of the audio wave as well as the sampling rate.

Adjusting Audio Length: Truncates or pads the audio data to a maximum length ('max_length') specified. Padding adds zeros to make shorter audio reach the specified length, while truncating cuts longer audio to fit the specified length.

Computing Spectrogram: Computes the mel spectrogram of the processed audio using Librosa. Adjusts the spectrogram to decibel scale using 'librosa.power_to_db'.

Resizing Spectrogram: Resizes the spectrogram to a specified 'target_shape' for further preprocessing.

Formatting for TensorFlow: Adapts the spectrogram's shape to match the expected format for TensorFlow models like CNN by adding an extra dimension.

```python
# Paths to clean and noisy directories
clean_dir = os.path.join(directory, 'clean_trainset_28spk_wav')
noisy_dir = os.path.join(directory, 'noisy_trainset_28spk_wav')
```

```python
# List all files in the directories with corresponding labels
clean_files = [(os.path.join(clean_dir, filename), 1) for filename in os.listdir(clean_dir) if filename.endswith('.wav')]
noisy_files = [(os.path.join(noisy_dir, filename), 0) for filename in os.listdir(noisy_dir) if filename.endswith('.wav')]
```

Along with the file paths, labels are also added. 1 for Clean files and 0 for Noisy files.

```
# Merge clean and noisy files into a single list
all_files = clean_files + noisy_files
```

Combined both Clean and Noisy Files

```
# Preprocess data and assign labels
preprocessed_data = []
for file_path, label in all_files:
    spectrogram, label = preprocess(file_path, label)
    preprocessed_data.append((spectrogram, label))
```

Converting all the .wav files into the spectogram values of array with the respective label and appended it in the empty list named preprocessed_data.

```
# Separate the spectrograms and labels
spectrograms, labels = zip(*preprocessed_data)
```

unzipping and seperating both the spectograms and labels.

.\

```
# Convert preprocessed data to NumPy arrays
spectrograms = np.array(spectrograms)
labels = np.array(labels)
```

```
X=spectrograms
y=labels
```

Creating X and y variables for Model Building.

```
# Define the CNN model
model = Sequential()

# Convolutional layers
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=X.shape[1:]))  # Assuming X.shape[1:] provides the input shape
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Flatten layer to transition from convolutional to dense layers
model.add(Flatten())

# Dense layers
model.add(Dense(128, activation='relu',kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))  # Adding dropout for regularization
model.add(Dense(1, activation='sigmoid'))  # Output layer with sigmoid activation for binary classification

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Print model summary
model.summary()
```

For Avoiding the overfitting problem, we have done the regularization and also Batchnormalization.

model.compile(): Configures the model for training.

optimizer='adam': Uses the Adam optimization algorithm.

loss='binary_crossentropy': Appropriate for binary classification tasks.

metrics=['accuracy']: Evaluates model performance based on accuracy during training.

```
# Train the model
history = model.fit(X, y, epochs=5, batch_size=64, validation_split=0.2)
```

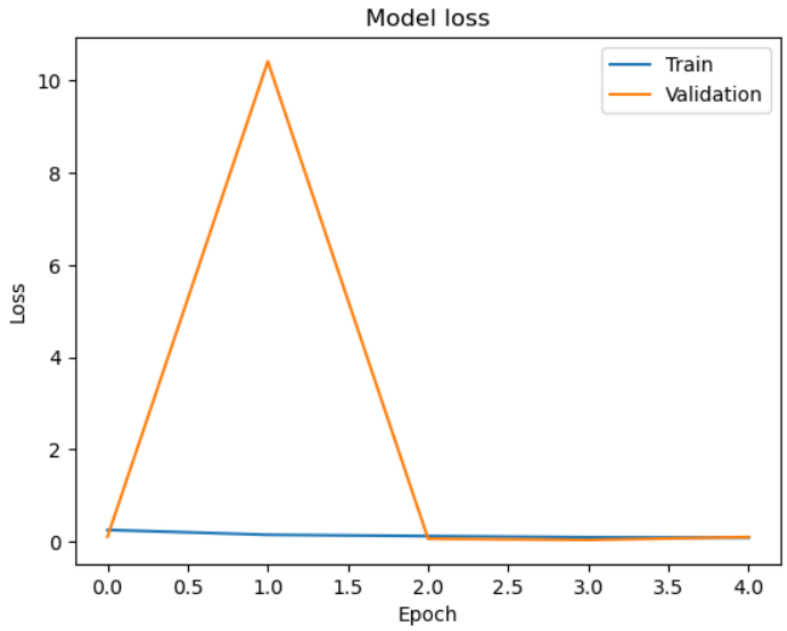
```
Epoch 1/5
WARNING:tensorflow:From C:\Users\sujit\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTe
nsorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\sujit\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.execut
ing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

271/271 [==============================] - 49s 177ms/step - loss: 0.2572 - accuracy: 0.9703 - val_loss: 0.1115 - val_accuracy:
1.0000
Epoch 2/5
271/271 [==============================] - 48s 176ms/step - loss: 0.1523 - accuracy: 0.9818 - val_loss: 10.4160 - val_accuracy:
0.1939
Epoch 3/5
271/271 [==============================] - 48s 177ms/step - loss: 0.1241 - accuracy: 0.9844 - val_loss: 0.0661 - val_accuracy:
1.0000
Epoch 4/5
271/271 [==============================] - 48s 177ms/step - loss: 0.0978 - accuracy: 0.9848 - val_loss: 0.0400 - val_accuracy:
1.0000
Epoch 5/5
271/271 [==============================] - 48s 178ms/step - loss: 0.0883 - accuracy: 0.9881 - val_loss: 0.1029 - val_accuracy:
0.9734
```

We have trained the model into the history variable.

batch_size=64: Determines the number of samples processed before updating the model's weights.

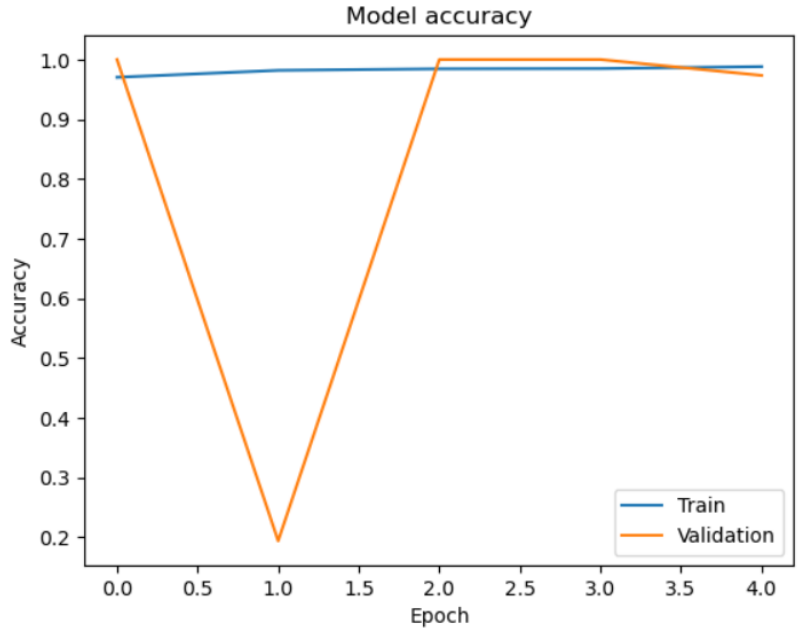
validation_split=0.2: Splits a portion (20%) of the data as a validation set to monitor model performance during training.

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```



This figure represents the Model Loss between the train and validation data. From graph we can say that there is no change in the train and validation loss.

```
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])A
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()
```



This figure represents the Model Accuracy between the train and validation data. From graph we can say that there is a negligible variation between them.

```
def preprocess_test(file_path, sample_rate=16000, max_length=48000, target_shape=(128, 128)):
    audio_data, sr = librosa.load(file_path, sr=sample_rate, mono=True)
    if len(audio_data) < max_length:
        audio_data = np.pad(audio_data, (0, max_length - len(audio_data)), 'constant')
    else:
        audio_data = audio_data[:max_length]
    spectrogram = librosa.feature.melspectrogram(y=audio_data, sr=sample_rate, n_fft=512,
                                                 hop_length=256, win_length=320)
    spectrogram = librosa.power_to_db(spectrogram, ref=np.max)
    spectrogram = np.resize(spectrogram, target_shape)
    spectrogram = np.expand_dims(spectrogram, axis=2)
    return spectrogram

# File path of the test audio
test_audio_path = r"D:\Noisy_Speech_Database\clean_testset_wav\p232_005.wav"

preprocessed_test_audio = preprocess_test(test_audio_path)
```

Testing the Audio Files using the Model

```
# Predict on the preprocessed test audio
predictions = model.predict(np.array([preprocessed_test_audio]))

if predictions >= 0.5:
    prediction = 1
else:
    prediction = 0

# Display the prediction result
print("Prediction:", prediction)
```

```
1/1 [==============================] - 0s 24ms/step
Prediction: 1
```

In the above code we are converting the audio file into the spectogram and giving as input to the model that we trained and checking the output.

If the probability is greater than 0.5 then the value is taken as 1 i.e., Clean file else 0 i.e., Noisy file.