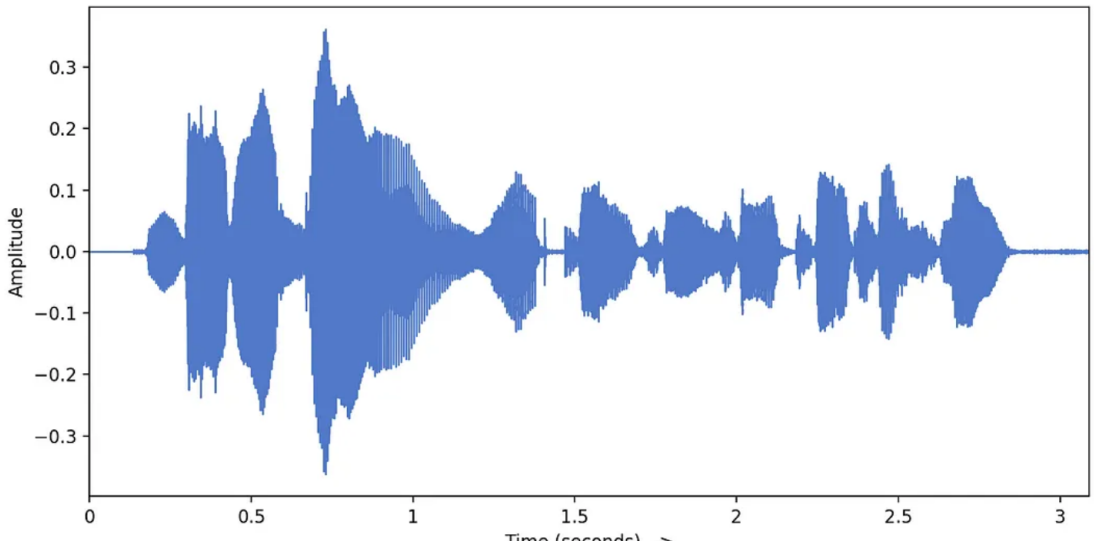
Report speaker recognition using CNN

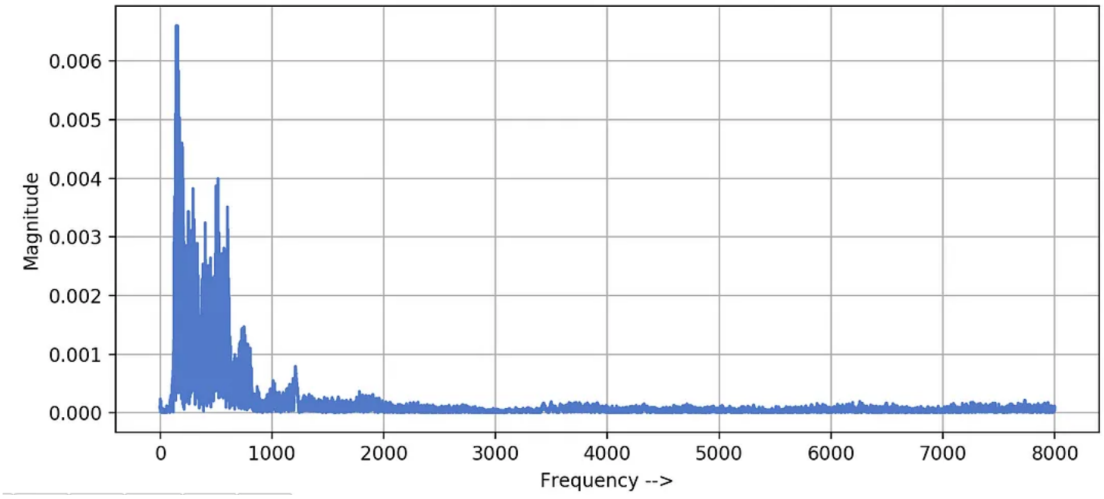
FFT:

Cơ sở lý thuyết:



Trực quan này được gọi là biểu diễn miền thời gian của một tín hiệu cụ thể. Điều này cho chúng ta thấy độ lớn (biên độ) của sóng âm thay đổi theo thời gian

Những biên độ này không cung cấp thông tin hữu ích, vì chúng chỉ nói về độ lớn của bản ghi âm thanh. Để hiểu rõ hơn về tín hiệu âm thanh, việc biến đổi nó thành miền tần số là cần thiết. Biểu diễn miền tần số của một tín hiệu cho chúng ta biết các tần số khác nhau có trong tín hiệu đó. Biến đổi Fourier là một khái niệm toán học có thể chuyển đổi một tín hiệu liên tục từ miền thời gian sang miền tần số



Triển khai:

FFT:

Chọn tỉ lệ lấy mẫu 16kHz

```
SAMPLING_RATE = 16000
```

```
def audio_to_fft(audio):
    # Since tf.signal.fft applies FFT on the innermost dimension,
    # we need to squeeze the dimensions and then expand them again
    # after FFT
    audio = tf.squeeze(audio, axis=-1)
    fft = tf.signal.fft(
        tf.cast(tf.complex(real=audio, imag=tf.zeros_like(audio)), tf.complex64)
    )
    fft = tf.expand_dims(fft, axis=-1)

    # Return the absolute value of the first half of the FFT
    # which represents the positive frequencies
    return tf.math.abs(fft[:, : (audio.shape[1] // 2), :])
```

1. **Nén Điều Kích**: `tf.squeeze(audio, axis=-1)` loại bỏ các kích thước có độ dài là 1 từ tensor `audio`, giúp chuẩn bị cho việc áp dụng FFT. Thông thường, tensor `audio` có thể có một kích thước "ảo" cuối cùng với độ dài là 1 (ví dụ, `[samples, 1]`), và bước này giúp chuyển nó thành `[samples]`, phù hợp để thực hiện FFT.

2. **Biến Đổi Fourier Nhanh (FFT)**: `tf.signal.fft(tf.cast(tf.complex(real=audio, imag=tf.zeros_like(audio)), tf.complex64))` thực hiện FFT trên tín hiệu âm thanh. Trước tiên, tín hiệu được chuyển đổi sang số phức với phần thực là `audio` và phần ảo là một tensor không có cùng kích thước với `audio`. Sau đó, áp dụng FFT trên tensor số phức này. FFT chuyển tín hiệu từ miền thời gian sang miền tần số, cho phép phân tích các thành phần tần số trong tín hiệu.
3. **Mở Rộng Kích Thước**: `fft = tf.expand_dims(fft, axis=-1)` thêm một kích thước vào cuối tensor `fft` để chuẩn bị cho các bước xử lý tiếp theo hoặc để giữ cho cấu trúc dữ liệu nhất quán với các phần khác của pipeline xử lý.
4. **Trích Xuất Tần Số Dương**: `tf.math.abs(fft[:, : (audio.shape[1] // 2), :])` lấy phần tuyệt đối của nửa đầu của FFT. Trong biến đổi Fourier, cả hai nửa của kết quả đều chứa thông tin về tần số, nhưng chúng là gương của nhau và thông tin tần số thực sự có thể được tìm thấy trong nửa đầu. Phần tuyệt đối được lấy để chuyển từ số phức sang giá trị thực, thể hiện biên độ của các thành phần tần số.

Kết quả cuối cùng là một tensor chứa biên độ của các thành phần tần số dương trong tín hiệu âm thanh, có thể được sử dụng để phân tích hoặc như là đặc trưng đầu vào cho các mô hình học máy.

Build Model:

Vì các data là các video 1 giây của mỗi người và tỉ lệ lấy mẫu là 16000/s rất lơn nên ta cần xây dựng một model với bộ lọc tăng dần để tăng cường phần trích xuất đặc trưng và độ sâu của mạng. Đồng thời dùng **Average Pooling để giảm dữ liệu đầu ra**

```python
def residual_block(x, filters, conv_num=3, activation="relu"):
    # Shortcut
    s = keras.layers.Conv1D(filters, 1, padding="same")(x)
    for i in range(conv_num - 1):
        x = keras.layers.Conv1D(filters, 3, padding="same")(x)
        x = keras.layers.Activation(activation)(x)
    x = keras.layers.Conv1D(filters, 3, padding="same")(x)
    x = keras.layers.Add()([x, s])
    x = keras.layers.Activation(activation)(x)
    return keras.layers.MaxPool1D(pool_size=2, strides=2)(x)


def build_model(input_shape, num_classes):
    inputs = keras.layers.Input(shape=input_shape, name="input")

    x = residual_block(inputs, 16, 2)
    x = residual_block(x, 32, 2)
    x = residual_block(x, 64, 3)
    x = residual_block(x, 128, 3)
    x = residual_block(x, 128, 3)

    x = keras.layers.AveragePooling1D(pool_size=3, strides=3)(x)
    x = keras.layers.Flatten()(x)
    x = keras.layers.Dense(256, activation="relu")(x)
    x = keras.layers.Dense(128, activation="relu")(x)

    outputs = keras.layers.Dense(num_classes, activation="softmax", name="output")(x)

    return keras.models.Model(inputs=inputs, outputs=outputs)


model = build_model((SAMPLING_RATE // 2, 1), len(class_names))
```

Kết quả:

```python
print(model.evaluate(valid_ds))
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 146ms/step - accuracy: 0.9446 - loss: 0.1606
[0.16722077131271362, 0.9438202381134033]
```

MFCC:

Cơ sở lý thuyết:

Nguyên lý hình thành tiếng nói: Không khí đi từ phổi, qua khí quản, lên vòm miệng. Ở vòm miệng, các rung động trong không khí được tổng hợp (cộng hưởng, triệt tiêu, ...) và TẠO THÀNH ÂM THANH thoát ra khỏi miệng

Âm, âm tiết, âm vị: Một từ có thể cấu tạo bởi một hoặc nhiều âm tiết. Mỗi âm tiết có thể có cách phát âm khác nhau, gọi là âm vị. VD chữ "**ough**" trong câu sau có tới 6 kiểu phát âm.

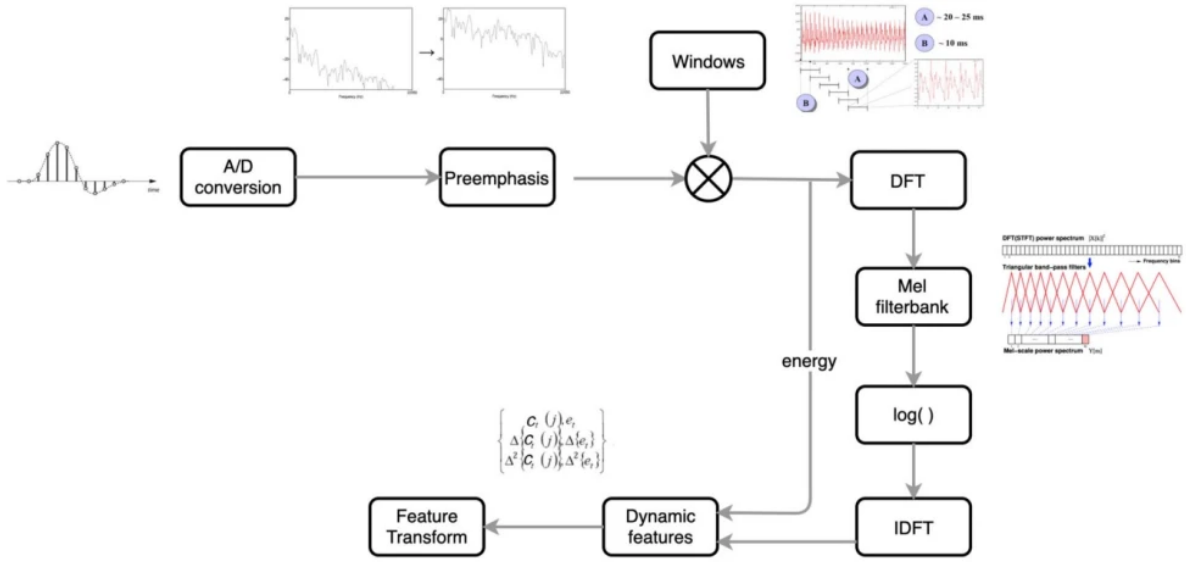
Tuy nhiên, cùng một âm vị nhưng mỗi người đọc ra lại khác nhau do giọng mỗi người mỗi khác. Âm là khi một người thực hiện hóa việc đọc ra âm tiết. Mỗi người có một cao độ riêng đặc trưng cho người đó, được gọi là F0. Mỗi âm tiết cũng có các tần số đặc trưng cho chúng, được gọi là F1, F2, F3 **(> F0)**.

Cơ chế hoạt động của tai: Tai người cấu tạo theo hình xoắn ốc. Trên thành tai có các sợi lông tơ dùng để cảm nhận âm thanh đi vào tai. Càng vào sâu trong tai, số lượng lông tơ tăng lên; sợi lông tơ mỏng đi. Do đó, từng vùng trong tai đảm nhiệm việc cảm nhận các sóng âm thanh khác nhau. Tai người **nhạy với âm thanh có tần số thấp, kém nhạy ở tần số cao.** Cơ chế máy móc thu âm thanh khác với cách con người cảm nhận âm thanh, do đó, cần có một cơ chế để mapping (Mel Filterbank).

Fourier Transform: Là một hàm số đối xứng, giúp chuyển một tín hiệu **điều hòa, khả tích** (ví dụ như âm thanh) từ miền thời gian về miền tần số. Điều này thực hiện được dựa trên lý thuyết chuỗi **Fourier**: Bất kỳ một hàm số khả tích trên miền điều hòa đều có thể biểu diễn dưới dạng tổng của các sóng **sin** (tương đương với biểu diễn được dưới dạng **tổng các miền tần số thành phần**).

Feature Extraction – MFCC cho xử lý tiếng nói

Quá trình từ âm thanh đầu với tới tạo ra các thuộc tính đặc trưng MFCC



MFCC là một thuật toán để trích xuất thông tin từ tín hiệu đầu vào. Ta xét bài toán đơn giản: Đầu vào là các tập audio, mỗi audio là một âm tiết. Tìm các feature tốt để phân biệt các âm vị này. MFCC chính là thuật toán giúp ta tìm các feature kia.

Notes: Bài này chỉ notes lại những ý chính quan trọng, đọc thêm các tài liệu để hiểu rõ hơn.

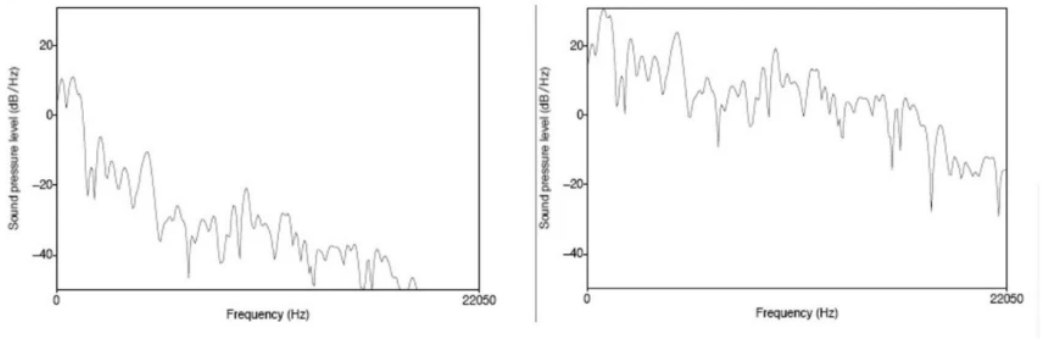
A/D conversion

Âm thanh là dạng tín hiệu liên tục, trong khi đó máy tính làm việc với các con số rời rạc. Ta cần lấy mẫu tại các khoảng thời gian cách đều nhau với 1 tần số lấy mẫu xác định (sample rate) để chuyển từ dạng tín hiệu liên tục về dạng rời rạc. VD sample_rate = 8000 ⟶ trong 1s lấy 8000 giá trị.

Preemphasis

Các âm ở tần số thấp có năng lượng cao; các âm ở tần số cao có mức năng lượng thấp. Tuy nhiên, do cấu trúc của thanh quản và các bộ phận phát âm nên ở tần số cao vẫn chứa những thông tin quan trọng về âm vị. Do đó, ta phải kích năng lượng ở các tần số cao này lên.

$$x'[t_d] = x[t_d] - \alpha x[t_d - 1] \qquad 0.95 < \alpha < 0.99$$



Windows

Thay vì biến đổi Fourier trên cả đoạn âm thanh dài, ta trượt 1 cửa sổ (windows) dọc theo tín hiệu để lấy ra các **frame** rồi mới áp dụng DFT trên từng frame này (DFT – Discrete Fourier Transform).

Do trong xử lý âm thanh, ngữ cảnh rất quan trọng do đó, mỗi windows cần được chia thành 3 states: nối với âm trước, âm của chính nó, nối với âm sau. Mỗi giây, trung bình một người nói khoảng 3 từ, mỗi từ có 4 âm, mỗi âm chia ra 3 states → cần phân biệt 36 states trong 1s. Vậy độ dài mỗi **windows=25ms** là hợp lý.

Các windows cũng cần overlap với nhau một khoảng 10ms để lưu trữ thông tin overlapping.

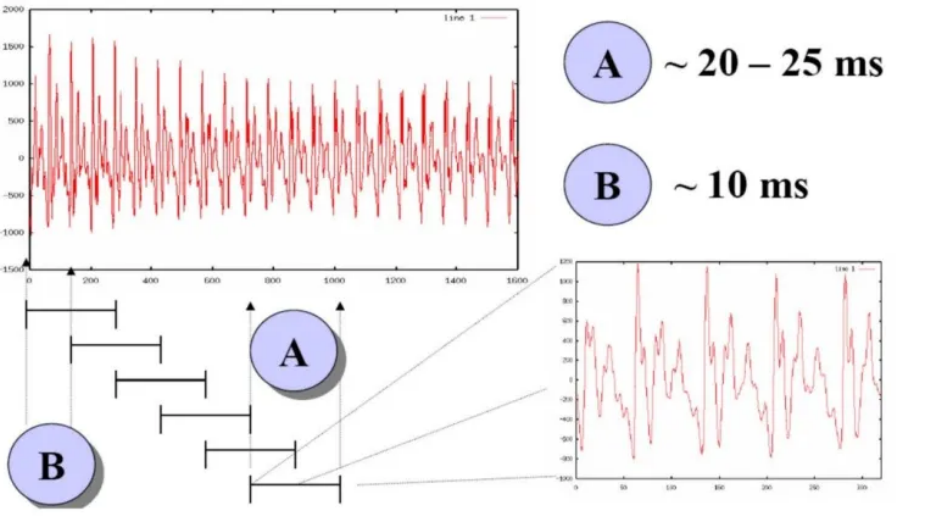
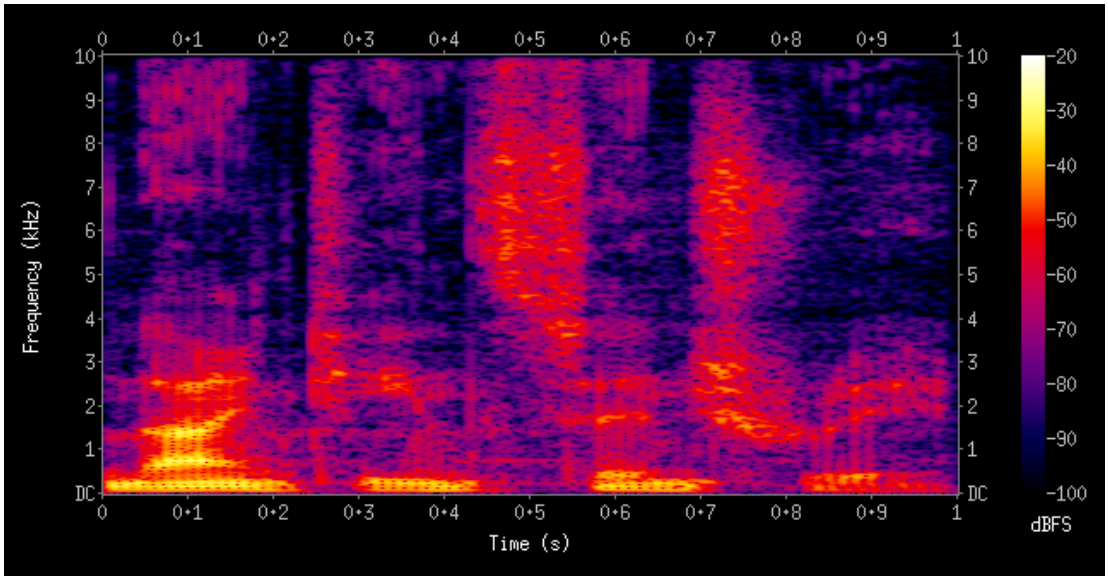


Image from Bryan Pellom

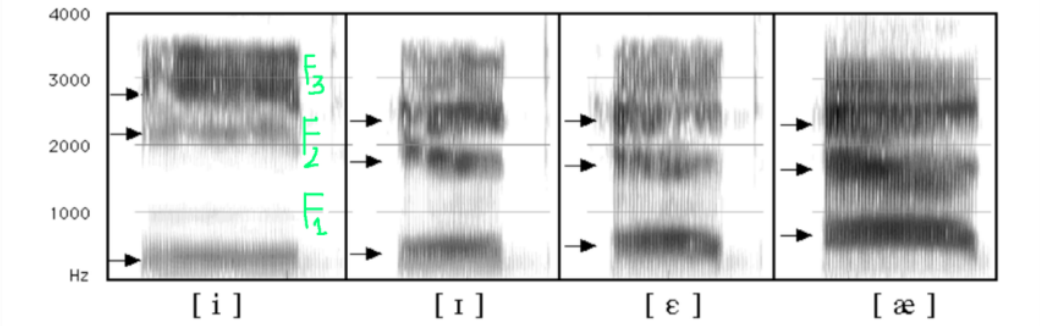
Ngoài ra, để lọc nhiễu, người ta cũng nhân tích chập các windows thu được với các hàm lọc nhiễu như Hamming window, Hanning window, ...

DFT

Trên từng frame, ta áp dụng DFT – Discrete Fourier Transform. Kết quả thu được là 1 spectrogram.



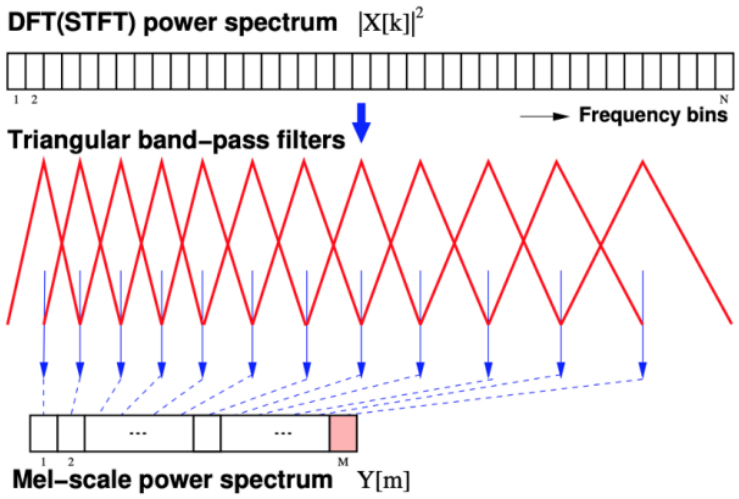
Hình dưới là các spectrogram của 4 nguyên âm. Quan sát spectrogram lần lượt từ dưới lên, người ta nhận thấy có 1 vài tần số đặc trưng gọi là các **formant**, gọi là các tần số F1, F2, F3 ... Các chuyên gia về ngữ âm học có thể dựa vào vị trí, thời gian, sự thay đổi các formant trên spectrogram để xác định đoạn âm thanh đó là của âm vị nào.



Tuy nhiên trong nhiều bài toán (đặc biệt là speech recognition), spectrogram không phải là sự lựa chọn hoàn hảo. Vì vậy ta cần thêm vài bước tính nữa để thu được dạng **MFCC**, tốt hơn, phổ biến hơn, hiệu quả hơn spectrogram.

Mel filterbank

Cách cảm nhận của tai người là phi tuyến tính, không giống các thiết bị đo. Tai người cảm nhận tốt ở các tần số thấp, kém nhạy cảm với các tần số cao. Ta cần 1 cơ chế mapping tương tự như vậy.



Trước hết, ta bình phương các giá trị trong spectrogram thu được **DFT power spectrum** (phổ công suất). Sau đó, ta áp dụng 1 tập các bộ lọc thông dải **Mel-scale filter** trên từng khoảng tần số (mỗi filter áp dụng trên 1 dải tần xác định). Giá trị output của từng filter là năng lượng dải tần số mà filter đó cover (bao phủ) được. Ta thu được **Mel-scale power spectrum**. Ngoài ra, các filter dùng cho dải tần thấp thường hẹp hơn các filter dùng cho dải tần cao.

log()

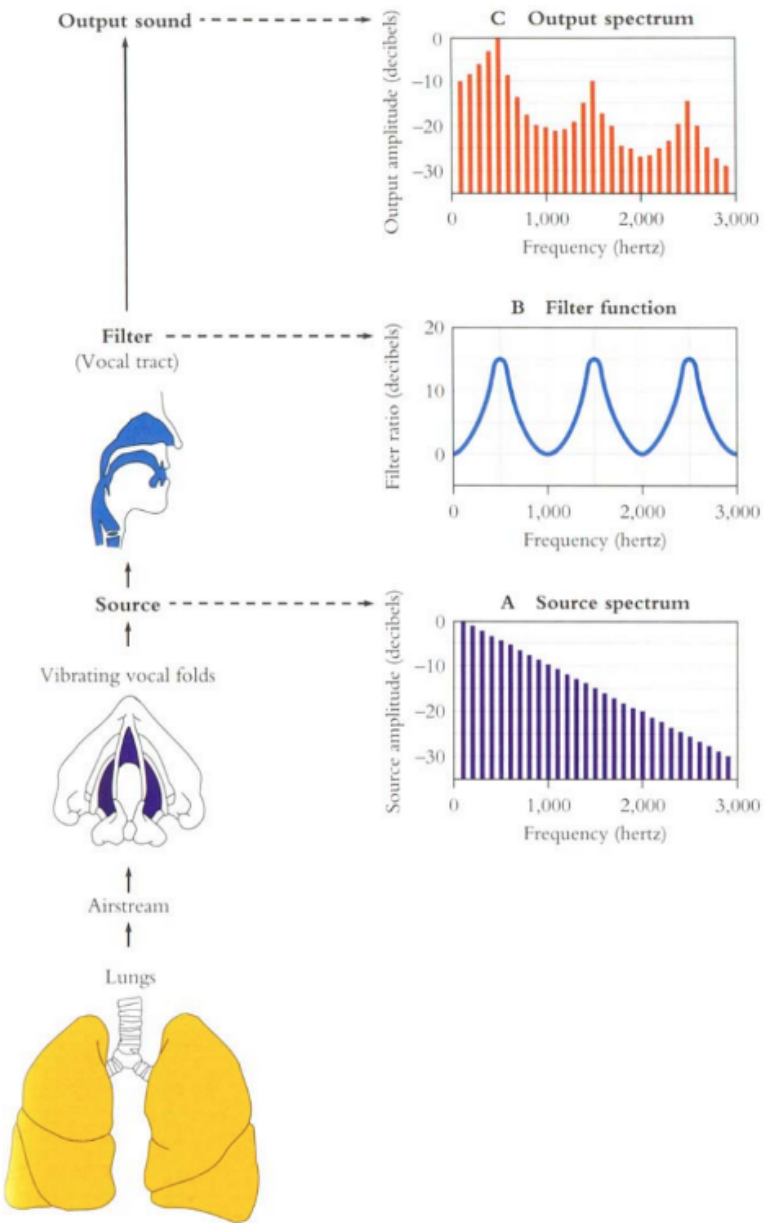
Mel filterbank trả về phổ công suất của âm thanh, hay còn gọi là phổ năng lượng. Thực tế rằng con người kém nhạy cảm trong sự thay đổi năng lượng ở các tần số cao, nhạy cảm hơn ở tần số thấp. Vì vậy ta sẽ tính log trên Mel-scale power spectrum. Điều này còn giúp giảm các biến thể âm thanh không đáng kể để nhận dạng giọng nói.

Chúng ta cần thực hiện hai việc nữa. Thứ nhất, cần loại bỏ F0 (thông tin về cao độ) vì ta không muốn thuật toán xử lý giọng nói (ví dụ ở đây là phân loại âm vị) của

mình phụ thuộc vào cao độ của từng người. Thứ hai, ta cần trích xuất các đặc trưng sau khi loại bỏ F0 để làm đặc trưng cho từng âm vị.

IDFT: Loại bỏ thông tin F0.

Như đã mô tả ở phần trước, giọng nói của chúng ta có tần số F0 – tần số cơ bản và các **formant** F1, F2, F3 ... Tần số F0 ở nam giới khoảng 125 Hz, ở nữ là 210 Hz, đặc trưng cho cao độ giọng nói ở từng người. Thông tin về cao độ này không giúp ích trong nhận dạng giọng nói, nên ta cần tìm cách để loại thông tin về F0 đi, giúp các mô hình nhận dạng không bị phụ thuộc vào cao độ giọng từng người. (F1, F2, F3 > F0)

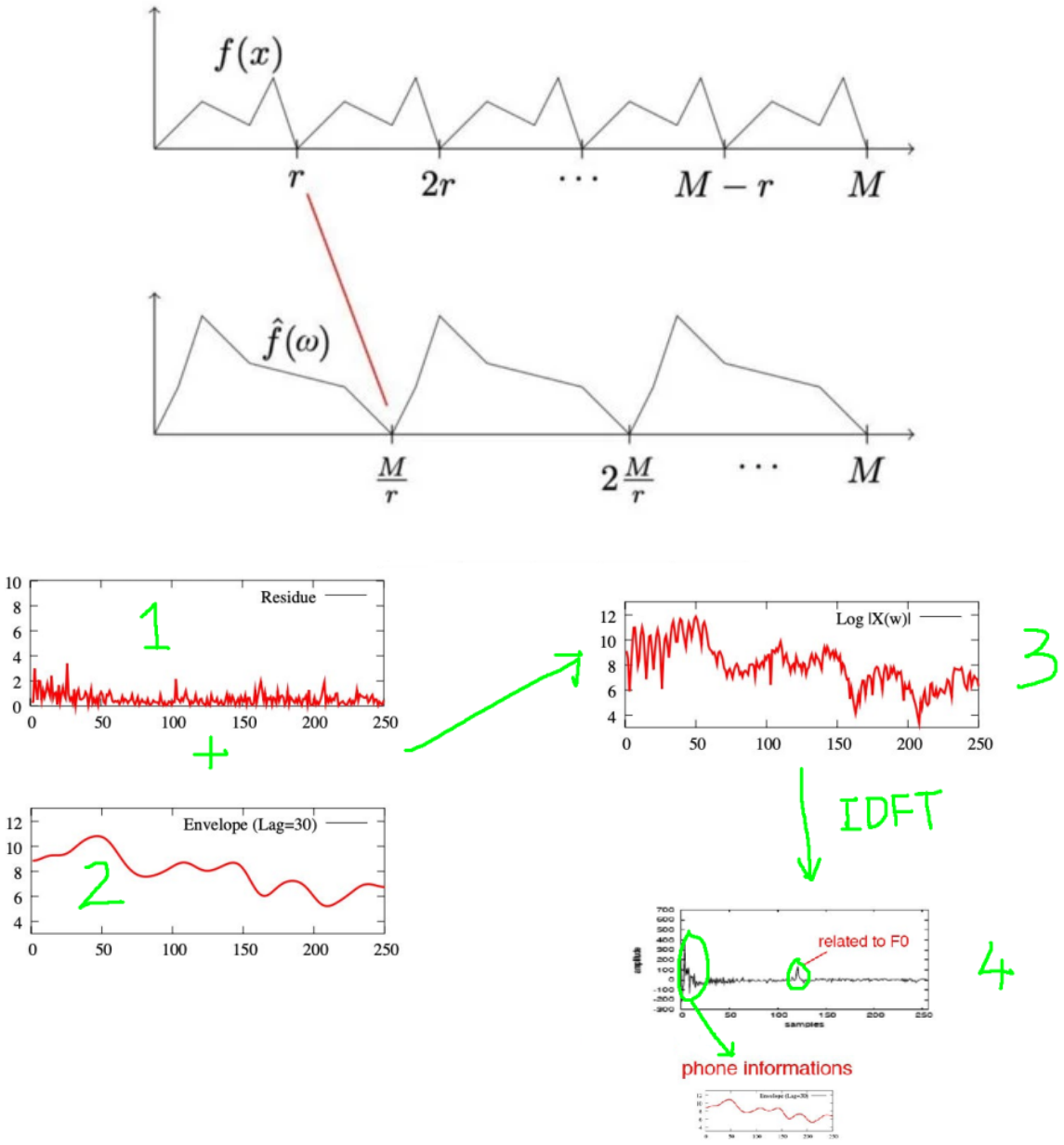


QUESTION: Why don’t we remove F0 in frequency domain???

Chúng ta chuyển qua miền thời gian để loại bỏ F0.

Nhớ lại rằng chu kỳ của các sóng trong miền tần số hay thời gian sẽ bị nghịch đảo sau khi biến đổi DFT.

F0 (pitch information) có chu kỳ ngắn nhất trong miền tần số do đó, khi chuyển qua miền thời gian, chu kỳ của nó là lớn nhất. Do đó, ta có thể dễ dàng tách biệt nó khỏi những thành phần còn lại F1, F2, ...



Trong hình này, tín hiệu chúng ta thu được là đồ thị 3, nhưng thông tin quan trọng chúng ta cần là phần 2, thông tin cần loại bỏ là phần 1. Để loại bỏ đi thông tin về F0, ta làm 1 bước biến đổi Fourier ngược (IDFT) về miền thời gian, ta thu được Cepstrum. Nếu để ý kỹ, ta sẽ nhận ra rằng tên gọi "**cepstrum**" thực ra là đảo ngược 4 chữ cái đầu của "**spetrum**".

Khi đó, với Cepstrum thu được, phần thông tin liên quan tới F0 và phần thông tin liên quan tới F1, F2, F3 ... nằm tách biệt nhau như 2 phần khoanh tròn trong hình 4. Ta chỉ đơn giản lấy thông tin trong đoạn đầu của cepstrum (phần được khoanh tròn to trong hình 4). Để tính MFCC, ta chỉ cần lấy 12 giá trị đầu tiên.

Phép biến đổi IDFT cũng tương đương với 1 phép biến đổi DCT **discrete cosine transformation**. DCT là 1 phép biến đổi trực giao. Về mặt toán học, phép biến đổi này tạo ra các **uncorrelated features**, có thể hiểu là các feature độc lập hoặc có độ tương quan kém với nhau. Trong các thuật toán Machine learning, **uncorrelated features** thường cho hiểu quả tốt hơn. Như vậy sau bước này, ta thu được 12 Cepstral features.

Dynamic features (delta)

Như vậy, mỗi frame ta đã extract ra được 12 Cepstral features làm 12 feature đầu tiên của MFCC. feature thứ 13 là năng lượng của frame đó, tính theo công thức bên dưới. Nó giúp ta phân biệt các âm.

$$Energy = \sum_{t=t_1}^{t_2} x^2[t]$$

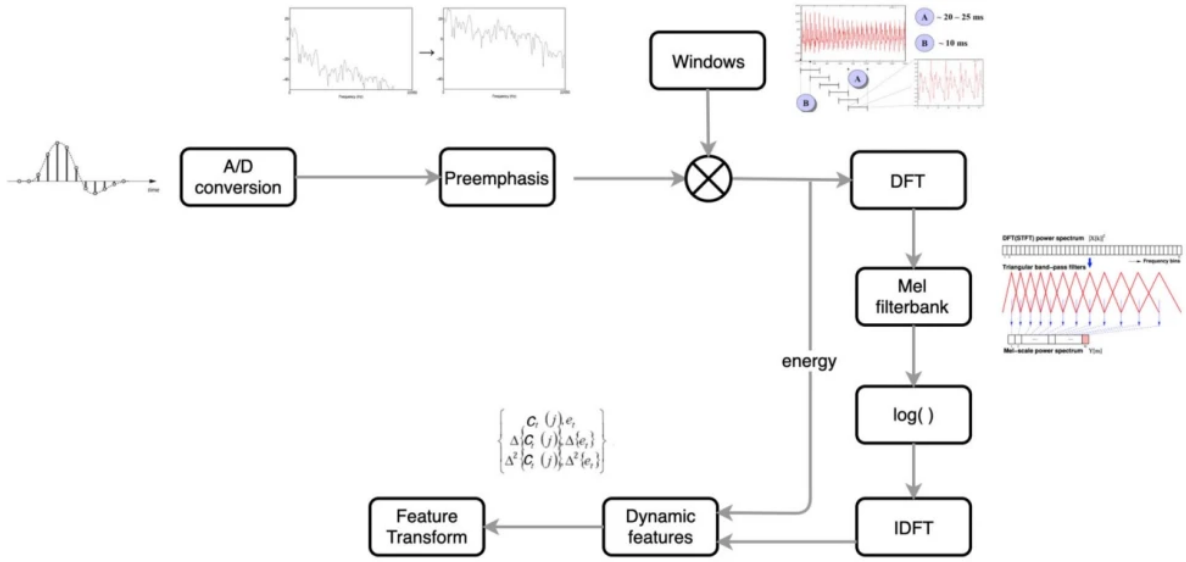
Trong nhận dạng tiếng nói, thông tin về bối cảnh và sự thay đổi rất quan trọng. VD tại những điểm mở đầu hoặc kết thúc ở nhiều phụ âm, sự thay đổi này rất rõ rệt, có thể nhận dạng các âm vị dựa vào sự thay đổi này. 13 hệ số tiếp theo chính là đạo hàm bậc 1 (theo thời gian) của 13 feature đầu tiên. Nó chứa thông tin về sự thay đổi từ frame thứ (t) đến frame (t+1). Công thức:

$$d(t) = \frac{c(t+1) - c(t-1)}{2}$$

Tương tự như vậy, 13 giá trị cuối của MFCC là sự thay đổi d(t) theo thời gian – đạo hàm của d(t), đồng thời là đạo hàm bậc 2 của c(t). Công thức:

$b(t) = 1/2[d(t+1) - d(t-1)]$

Vậy, từ 12 cepstral feature và power feature thứ 13, ta đạo hàm 2 lần và thu được 39 feature. Đây chính là MFCC feature. Cùng nhìn lại toàn bộ quá trình để tạo ra MFCC:



Triển khai:

Vì MFCC sau khi trích xuất đặc trưng sẽ đưa về độ dài nhỏ hơn nhiều so với FFT nên ta cần dùng một data mới có độ dài lớn hơn 1s, ở đây ta chọn 3s với tỉ lệ lấy mẫu mỗi giây là 16000 tức ta có 48000 mẫu

MFCC:

```python
def audio_to_mfcc(audio, sample_rate, num_mfccs=13):
    # Step 1: STFT
    print(tf.squeeze(audio, axis=-1))
    stfts = tf.signal.stft(tf.squeeze(audio, axis=-1),
                           frame_length=1440,
                           frame_step=480,
                           fft_length=1440)
    spectrograms = tf.abs(stfts)
    print("a: ",spectrograms.shape)

    # Step 2: Warp the linear-scale spectrograms into the mel-scale.
    num_spectrogram_bins = stfts.shape[-1]
    lower_edge_hertz, upper_edge_hertz, num_mel_bins = 80.0, 7600, 40
    linear_to_mel_weight_matrix = tf.signal.linear_to_mel_weight_matrix(
        num_mel_bins, num_spectrogram_bins, 16000, lower_edge_hertz,
        upper_edge_hertz)

    print("b: ",linear_to_mel_weight_matrix.shape[-1:])

    mel_spectrograms = tf.tensordot(
        spectrograms, linear_to_mel_weight_matrix, 1)
    mel_spectrograms.set_shape(spectrograms.shape[:-1].concatenate(
        linear_to_mel_weight_matrix.shape[-1:]))

    # Step 3: Compute a stabilized log to get log-magnitude mel-scale spectrograms.
    log_mel_spectrograms = tf.math.log(mel_spectrograms + 1e-6)

    # Step 4: Compute MFCCs from log_mel_spectrograms and take the first 13.
    mfccs = tf.signal.mfccs_from_log_mel_spectrograms(
        log_mel_spectrograms)[..., :tf.newaxis]

    print(mfccs)

    return mfccs
```

Bước 1: Tính STFT (Short-Time Fourier Transform)

- **Nén Điều Kích**: `tf.squeeze(audio, axis=-1)` loại bỏ kích thước là 1 ở chiều cuối cùng của tensor `audio`, chuyển tensor từ dạng `[samples, 1]` thành `[samples]`, chuẩn bị cho việc tính toán STFT.
- **Tính STFT**: `tf.signal.stft(...)` tính Short-Time Fourier Transform của tín hiệu, chuyển đổi tín hiệu từ miền thời gian sang miền tần số. `frame_length` và `frame_step` xác định kích thước của cửa sổ và bước dịch chuyển giữa các cửa

sổ liên tiếp, còn `fft_length` xác định độ dài của biến đổi Fourier nhanh, ảnh hưởng đến độ phân giải tần số.

- **Biểu Diễn Dưới Dạng Số Phức**: STFT trả về một tensor số phức, với phần thực đại diện cho biên độ và phần ảo đại diện cho pha.
- **Tính Giá Trị Tuyệt Đối**: `tf.abs(stfts)` tính giá trị tuyệt đối của kết quả STFT để lấy biên độ của tín hiệu, loại bỏ thông tin về pha.

Bước 2: Chuyển Đổi Sang Thang Mel

- **Xác Định Thông Số Thang Mel**: Xác định ranh giới tần số thấp nhất và cao nhất (80 Hz đến 7600 Hz trong trường hợp này), và số lượng "bins" trong thang Mel.
- **Tính Ma Trận Trọng Số Chuyển Đổi**: `tf.signal.linear_to_mel_weight_matrix(...)` tạo ma trận trọng số để chuyển đổi từ tần số tuyến tính sang thang Mel.
- **Áp Dụng Chuyển Đổi**: `tf.tensordot(spectrograms, linear_to_mel_weight_matrix, 1)` sử dụng phép nhân tensor để áp dụng ma trận trọng số chuyển đổi lên biểu đồ biên độ STFT, thu được biểu đồ Mel.

Bước 3: Tính Log-Magnitude của Mel-Spectrogram

- **Tính Log-Magnitude**: `tf.math.log(mel_spectrograms + 1e-6)` tính logarit của Mel-spectrogram để ổn định giá trị và giảm thiểu ảnh hưởng của các biến động biên độ nhỏ.

Bước 4: Tính MFCCs

- **Tính MFCCs**: `tf.signal.mfccs_from_log_mel_spectrograms(log_mel_spectrograms)[..., :num_mfccs]` tính MFCCs từ log-Mel-spectrograms. MFCCs là các hệ số mô tả hình dạng của phổ âm thanh trong thang Mel, và thường chỉ một số hệ số đầu tiên được sử dụng. Trong trường hợp này, chỉ lấy `num_mfccs` hệ số đầu tiên.

Build model:

```python
def build_model(input_shape, num_classes):
    inputs = keras.layers.Input(shape=input_shape, name="input")

    # Initial Convolutional Layers
    x = keras.layers.Conv2D(32, (3, 3), activation="relu", padding="same")(inputs)
    x = keras.layers.Conv2D(32, (3, 3), activation="relu", padding="same")(x)
    x = keras.layers.MaxPooling2D((2, 2))(x)
    x = keras.layers.BatchNormalization()(x)


    x = keras.layers.Conv2D(64, (3, 3), activation="relu", padding="same")(x)
    x = keras.layers.Conv2D(64, (3, 3), activation="relu", padding="same")(x)
    x = keras.layers.MaxPooling2D((2, 2))(x)
    x = keras.layers.BatchNormalization()(x)

    # Residual Blocks
    x = keras.layers.Conv2D(128, (3, 3), activation="relu", padding="same")(x)
    x = keras.layers.Conv2D(128, (3, 3), activation="relu", padding="same")(x)
    x = keras.layers.MaxPooling2D((2, 2))(x)
    x = keras.layers.GlobalAveragePooling2D()(x)

    # Flatten and Dense Layers
    x = keras.layers.Dense(256, activation="relu")(x)
    x = keras.layers.Dropout(0.5)(x)
    x = keras.layers.Dense(128, activation="relu")(x)
    x = keras.layers.Dropout(0.5)(x)

    outputs = keras.layers.Dense(num_classes, activation="softmax")(x)

    # Create model
    model = keras.models.Model(inputs=inputs, outputs=outputs)
    return model
```

Kết quả:

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.9220 - loss: 0.5622
[0.6534116268157959, 0.9142857193946838]
```

Với kết quả dựa trên 50 tập test thì kết quả ra 44/50 = 0.88 gần với 0.92 nên không bị overfitting

Thống kê

Tên model	tham số cửa sổ	accuracy	tập test	Tập valid
MFCC-40bin	(1440, 480)	92%	6/10	44/50
MFCC-40bin-phobangrong-98%	(480, 160)	98%	3/10	47/50
MFCC-13bin-2	(1440, 480)	90%	4/10	36/50
MFCC-13bin-phobangrong-72%	(480, 160)	72%	6/10	36/50
MFCC-40bin-hamming2	(1440, 480)	(98%)	6/10	47/50
FFT		94%		47/50

Đặt vấn đề:

tại sao MFCC tốt hơn FFT rất nhiều nhưng kết quả sau train lại thấp hơn so với FFT?