Universidad Europea del Atlántico

“Inteligencia Artificial”

Proyecto modelo de reconocimiento de voz

Omar Hugo Alexander Barrios Catun

Santander 16/01/2024

Clase **SpecAugment**, que aplica enmascaramiento en el tiempo y en la frecuencia a espectrogramas de audio.

__init__(self, rate, policy=3, freq_mask=15, time_mask=35): Inicializa el módulo SpecAugment con la tasa, política, parámetro de enmascaramiento de frecuencia y parámetro de enmascaramiento de tiempo especificados.

forward(self, x): Pasa la entrada x a través de la función de política especificada.

policy1(self, x): Aplica la aumentación de espectrograma con un solo conjunto de máscaras a la entrada x basándose en una comprobación de probabilidad.

policy2(self, x): Aplica la aumentación de espectrograma con dos conjuntos de máscaras a la entrada x basándose en una comprobación de probabilidad.

policy3(self, x): Aplica ya sea la política 1 o la política 2 a la entrada x basándose en una comprobación de probabilidad.

```
class SpecAugment(nn.Module):

    Codeium: Refactor | Explain | Generate Docstring | X
    def __init__(self, rate, policy=3, freq_mask=15, time_mask=35): ···

    Codeium: Refactor | Explain | Generate Docstring | X
    def forward(self, x): ···

    Codeium: Refactor | Explain | Generate Docstring | X
    def policy1(self, x): ···

    Codeium: Refactor | Explain | Generate Docstring | X
    def policy2(self, x): ···

    Codeium: Refactor | Explain | Generate Docstring | X
    def policy3(self, x): ···
```

Clase **LogMelSpec** es una subclase de nn.Module y representa una transformación de espectrograma logarítmico Mel.

__init__(self, sample_rate=8000, n_mels=128, win_length=160, hop_length=80): Inicializa la clase y establece los parámetros para la transformación del espectrograma Mel.

forward(self, x): Toma un tensor de entrada x, aplica la transformación del espectrograma Mel a este, toma el logaritmo del tensor transformado y devuelve el resultado.

```
class LogMelSpec(nn.Module):

    Codeium: Refactor | Explain | Generate Docstring | ×
    def __init__(self, sample_rate=8000, n_mels=128, win_length=160, hop_length=80):

    Codeium: Refactor | Explain | Generate Docstring | ×
    def forward(self, x): ···
```

La clase **Data** es una subclase de torch.utils.data.Dataset y representa un conjunto de datos para el procesamiento de audio.

__init__(self, json_path, sample_rate, n_feats, specaug_rate, specaug_policy, time_mask, freq_mask, valid=False, shuffle=True, text_to_int=True, log_ex=True): Inicializa el conjunto de datos cargando un archivo JSON, configurando las transformaciones de audio según si el conjunto de datos es para validación o no, y estableciendo varios parámetros para el procesamiento de audio.

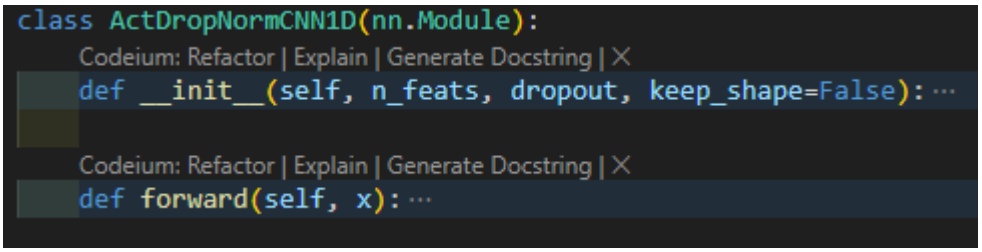
__len__(self): Devuelve la longitud del conjunto de datos. __getitem__(self, idx): Devuelve una muestra del conjunto de datos en el índice dado. Carga el archivo de audio, aplica transformaciones de audio y prepara el espectrograma y la etiqueta para el entrenamiento.

describe(self): Devuelve un resumen del conjunto de datos. Avísame si necesitas ayuda con algo más.

```
class Data(torch.utils.data.Dataset):

    # this makes it easier to be ovveride in argparse    "ovveride": Unknown word.
    parameters = {···

    Codeium: Refactor | Explain | Generate Docstring | ×
    def __init__(self, json_path, sample_rate, n_feats, specaug_rate, specaug_policy,


    Codeium: Refactor | Explain | Generate Docstring | ×
    def __len__(self): ···

    Codeium: Refactor | Explain | Generate Docstring | ×
    def __getitem__(self, idx): ···

    Codeium: Refactor | Explain | Generate Docstring | ×
    def describe(self): ···
```

Esta clase es un modelo de red neuronal convolucional (CNN) 1D con dropout y normalización de capa.
__init__(self, n_feats, dropout, keep_shape=False): Inicializa el modelo y establece la tasa de dropout, la normalización de capa y si se debe mantener la forma de la entrada.

forward(self, x): Realiza el pase hacia adelante del modelo. Transpone la entrada, aplica dropout, normalización de capa y la función de activación GELU, y luego devuelve la salida con la forma opcionalmente restaurada.

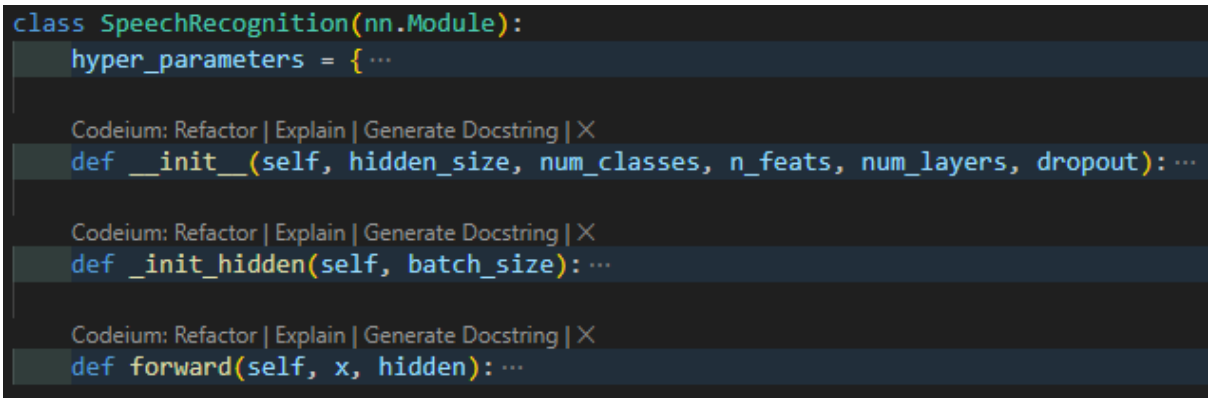


Esta definición de clase representa un modelo de reconocimiento de voz.

__init__(self, hidden_size, num_classes, n_feats, num_layers, dropout): Inicializa el modelo con los hiperparámetros dados y define las capas del modelo.

_init_hidden(self, batch_size): Inicializa el estado oculto de la capa LSTM.

forward(self, x, hidden): Realiza el pase hacia adelante del modelo. Aplica las capas de CNN y densas a la entrada, la pasa a través de la capa LSTM, aplica dropout y normalización de capa, y finalmente aplica una transformación lineal para obtener los logits de salida.



Esta clase define una utilidad de procesamiento de texto.

- El método **__init__** inicializa los diccionarios char_map e index_map mediante el análisis de una representación de cadena del mapa de caracteres.

- El método **text_to_int_sequence** toma un texto como entrada y lo convierte en una secuencia de enteros utilizando char_map.

- El método **int_to_text_sequence** toma una lista de etiquetas enteras como entrada y la convierte en una secuencia de texto utilizando index_map.

```
class TextProcess:
    Codeium: Refactor | Explain | Generate Docstring | X
    def __init__(self): ···

    Codeium: Refactor | Explain | X
    def text_to_int_sequence(self, text): ···

    Codeium: Refactor | Explain | X
    def int_to_text_sequence(self, labels): ···
```

Esta clase, **Listener**, se utiliza para capturar datos de audio de un flujo y agregarlos indefinidamente a una cola.

- Método **init**: Este método inicializa la clase y configura el flujo de audio, permitiendo que la clase esté lista para capturar y procesar datos de audio.

- Método **listen**: El método listen se encarga de leer datos del flujo de audio y agregarlos a la cola designada. Este paso asegura que los datos de audio entrantes se añadan continuamente a la cola para su procesamiento o utilización.

- Método **run**: El método run ejecuta el método listen en un hilo separado. Este enfoque permite que el proceso de escucha se ejecute de manera concurrente, garantizando una operación continua de captura de audio desde el flujo y su adición a la cola especificada.

```
You, hace 2 minutos | 2 authors (You and others) | Codeium: Explain
class Listener:

    Codeium: Refactor | Explain | X
    def __init__(self, sample_rate=8000, record_seconds=2): ···

    Codeium: Refactor | Explain | X
    def listen(self, queue): ···

    Codeium: Refactor | Explain | X
    def run(self, queue): ···
```

La clase **SpeechRecognitionEngine** es responsable de reconocimiento de voz.

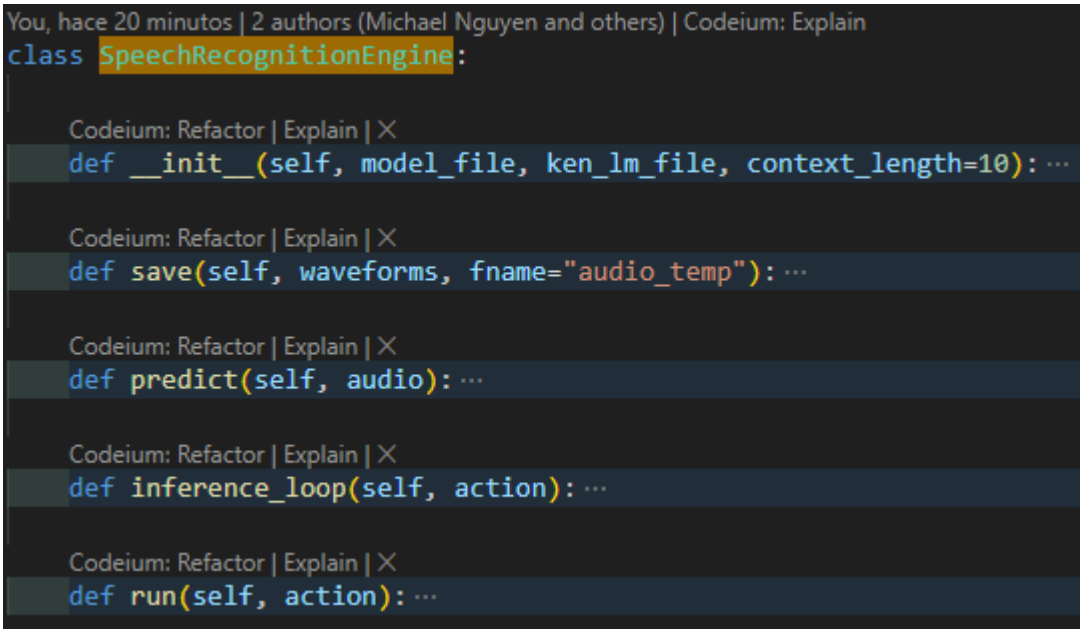
init(self, model_file, ken_lm_file, context_length=10): Inicializa la instancia de la clase con el archivo de modelo, archivo ken_lm y la longitud de contexto proporcionados.

save(self, waveforms, fname="audio_temp"): Guarda las formas de onda dadas en un archivo

predict(self, audio): Predice la salida para el audio proporcionado.

inference_loop(self, action): Ejecuta un bucle de inferencia que procesa continuamente datos de audio.

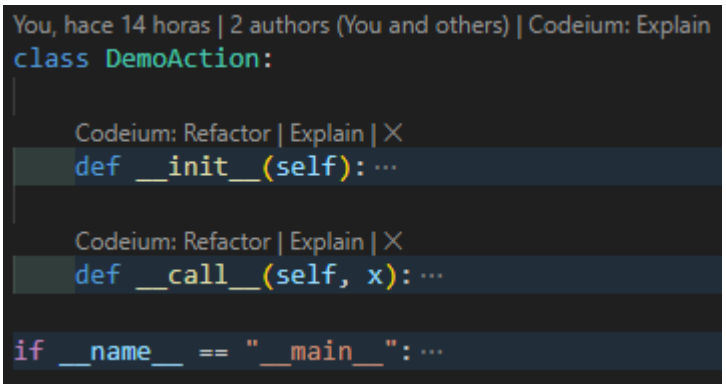
run(self, action): Ejecuta la acción al ejecutar el escucha y comenzar un nuevo hilo para el bucle de inferencia.



Clase **DemoAction**.

init(self): Inicializa una nueva instancia de la clase, estableciendo asr_results y current_beam como cadenas vacías.

call(self, x): El método de llamada para el objeto. Toma una tupla x que contiene resultados y current_context_length, establece current_beam como resultados, concatena asr_results y resultados en transcript, imprime transcript y actualiza asr_results si current_context_length es mayor que 10.



La clase **CTCBeamDecoder** se utiliza para la decodificación de búsqueda de haz en el reconocimiento de voz.

__init__(self, beam_size=100, blank_id=labels.index('_'), kenlm_path=None): Inicializa la clase BeamSearchWithLM. Configura el decodificador con el tamaño de haz, ID en blanco y la ruta del modelo de lenguaje KenLM especificados.

__call__(self, output): Llama al método decode del decodificador para generar resultados de búsqueda de haz para el tensor de salida proporcionado. Devuelve la representación de cadena convertida del mejor resultado de búsqueda de haz.

convert_to_string(self, tokens, vocab, seq_len): Convierte una lista de tokens en una cadena utilizando un vocabulario dado. Devuelve la representación de cadena de los tokens hasta la longitud de secuencia especificada.