IoT

Práctica de Protocolos de Comunicación

Prisco García-Consuegra Martín
Laura García Perrín

Grado en Ciencia de Datos e Inteligencia Artificial

Universidad Politécnica de Madrid
Diciembre de 2023

1. Comandos de voz y adquisición de las muestras

Como primer paso deberemos seleccionar una lista de comandos de voz sencillos, que queremos que reconozca nuestro sistema. El número de comandos será pequeño y siempre menor de unos 10. Cada uno de estos consistirá en la pronunciación de una o dos palabras a lo sumo, de forma que, la duración de las grabaciones quede acotada en un tiempo de unos 500 ms. En concreto, los comandos de voz que elegimos para la práctica son:

Comando de Voz	Descripción
Up	Indica dirección «arriba»
Down	Indica dirección «abajo»
Stop	Detiene una acción o comando en curso

Figura 1: Comandos de voz elegidos

Se ha grabado un total de 100 muestras de los comandos de voz, verificándose que todos ellos son de calidad.

2. Diseño y desarrollo de la red neuronal convolucional para el reconocimiento de comandos de voz

El desarrollo de la red neuronal convolucional (CNN) se realizará empleando la herramienta TensorFlow Keras. La red está diseñada para reconocer comandos de voz y satisfacer los requisitos especificados en el enunciado. A continuación, se muestra un esquema general de los pasos que se siguen para la construcción de dicha red:

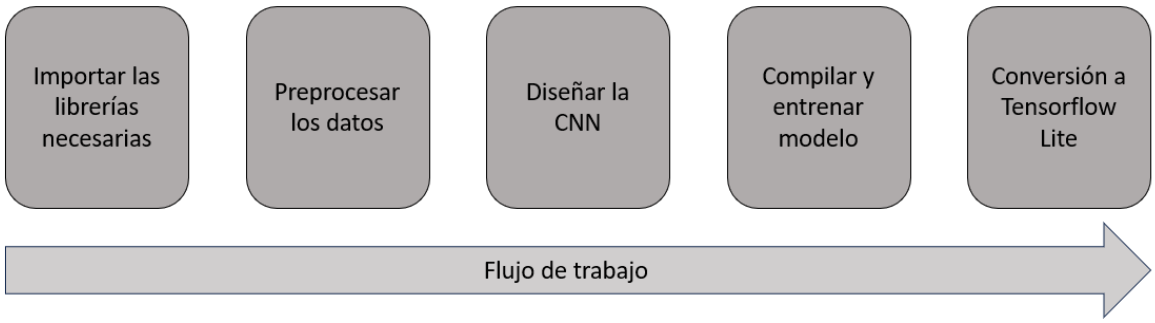

Figura 2: Esquema general de los pasos necesarios para construir la CNN

Para cumplir con los pasos necesarios en este apartado, se implementan una serie de métodos Python, que son los siguientes:

Función	**Descripción**
`calcular_espectrograma`	Calcula el espectrograma de una señal de audio utilizando una ventana de Hamming y la transformada de Fourier a corto plazo.
`cargar_y_procesar_audios`	Carga archivos de audio desde una carpeta y aplica la función `calcular_espectrograma` a cada uno para obtener sus espectrogramas.
`aplicar_kmeans`	Realiza la clasificación no supervisada de los espectrogramas utilizando el algoritmo KMeans.
`create_model`	Crea un modelo de red neuronal convolucional 1D para la clasificación de los datos de audio.
`model.fit`	Entrena el modelo de red neuronal con 75 épocas y un tamaño del *batch* de 32.
`representative_dataset_gen`	Genera un conjunto de datos representativo necesario para la conversión del modelo a TensorFlow Lite.
`convert_to_tflite`	Convierte el modelo entrenado a un formato TensorFlow Lite con optimizaciones y cuantificación.

Cuadro 1: Resumen de las fFunciones implementadas en el *script* de Python

Finalmente, el modelo obtiene un *accuracy* del 93 % aproximadamente en el conjunto de entrenamiento, lo que quiere decir que es mejorable ya que probablemente exista sobreajuste (*overfitting*). Ello podría deberse a consideraciones de diseño o bien al número de muestras de voz con las que se trabaja para entrenar la red neuronal.

3. Implementación del sistema completo en el dispositivo M5 Stick C Plus

Partiendo del esqueleto de proyecto Arduino proporcionado por el profesor de la asignatura, procederemos a completar la implementación. En concreto, se deben satisfacer una serie de pasos.

1. Primero volcamos el contenido del fichero que contiene nuestro modelo de red neuronal reducido con el comando ***xxd -i modelo_reducido.tflite*** > ***recvoz_red_nn.h*** y así obtenemos un vector que se incorpora dentro del fichero *recvoz_red_nn.h.*

2. Luego modificamos los valores definidos en el fichero *red_neuronal.h* con los empleados en nuestra red. En nuestro caso son:

   - N_TRAMAS_ESPECTROGRAMA_EXTENDIDO: **80**
   - N_TRAMAS_ESPECTROGRAMA: **63**
   - N_PUNTOS_FRECUENCIA_ESPECTROGRAMA: **129**
   - N_CLASES_RED_NEURONAL: **3**

3. A continuación aplicamos una ventana de Hamming de 256 muestras a las diferentes tramas de audio y siempre como paso previo a la ejecución de la FFT. En código se han añadido las siguientes líneas:

   *float hamming = 0.54 - 0.46 * cosf(2 * PI * i / VENTANA_MUESTRAS_FFT);*
   *fft_real->input[i] = NORMALIZACION_AUDIO * (float)bufferAudioPrevio[i] * hamming;*

   Tras las lecturas, la *fft_real* es multiplicada por la ventana de hamming que hemos añadido, quedando así la FFT lista para ser ejecutada.

4. Por último implementamos la operativa de la función Loop empleando para ello las funciones ya definidas en el código para la red neuronal (*red_neuronal.h*):

   - Cargamos como entrada la red neuronal del contenido de los espectrogramas, que paralelamente se van calculando y actualizando en la tarea *ObtenerEspectrograma*.
   - Invocamos la red neuronal para obtener las probabilidades de salida por clase.
   - Definimos la escritura a través de la terminal serie y también por pantalla del dispositivo las probabilidades anteriores.
   - Definimos y gestionamos unos umbrales asociados a dichas estimaciones para determinar cuándo se considera que el sistema ha reconocido uno de los comandos de voz.
   - Escribimos a través de la terminal serie y también por pantalla las decisiones de reconocimiento basados en los criterios establecidos previamente.

4. Output de la Aplicación

A continuación mostramos una fotografía correspondiente al funcionamiento de la aplicación en el M5-Stick C Plus.

Figura 3: M5-Stick C Plus

La salida del funcionamiento correcto de la aplicación visualizada en la terminal serie no ha podido conseguirse debido a la incompatibilidad de nuestro modelo de alta precisión con la capacidad de memoria que es capaz de manejar en cuanto a tensores el programa preestablecido.