UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Detección de palabras clave en señales de voz: una comparativa de modelos de deep learning

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Valls Lozano, Oscar

Tutor/a: Naranjo Ornedo, Valeriana

Cotutor/a externo: PRITISH, CHANDNA

CURSO ACADÉMICO: 2021/2022

Resumen

Los modelos de Keyword Spotting han sido ampliamente adoptados como solución para la ejecución de comandos simples de voz. Desde que se empezaron a popularizar los asistentes de voz como Siri, Alexa o Google Assistant, en los dispositivos móviles, y más recientemente en el ámbito de la domótica con dispositivos como Google Home, Apple HomePod o Amazon Echo, la detección de palabras clave se ha empleado para la activación del dispositivo mediante una palabra clave como "Oye Siri" para que el dispositivo reconozca comandos más complejos empleando modelos de lenguaje. La clave de la adopción de dichos modelos es su bajo consumo de recursos para que puedan funcionar durante largos periodos de tiempo, ejecutándose en el dispositivo y sin drenar su batería. A lo largo de su historia, se han ido proponiendo distintas arquitecturas que realizan dicha tarea. En este trabajo se propone una comparativa entre los modelos más recientes que forman parte del estado del arte para la evaluación tanto de su rendimiento como clasificadores como de su latencia y eficiencia para el posible entrenamiento, con un vocabulario específico, e implementación en tiempo real del modelo con mejores prestaciones.

Resum

Els models de detecció de paraules clau han estat àmpliament adoptats com a solució per a la execució de comandaments simples de veu. Des que els assistents de veu com Siri, Alexa o Google Assistant es van començar a popularitzar als dispositius mòbils, i més recentment a l'àmbit de la domòtica amb dispositius com Google Home, Apple HomePod o Amazon Echo, la detecció de paraules clau s'ha empleat per a la activació del dispositiu mitjançant una paraula clau com "Oye Siri" per fer que el dispositiu comence a reconéixer comandaments més complexos amb models de llenguatge. La clau de l'adopció d'aquests tipus de models és el seu baix consum de recursos per que puga funcionar durant llargs períodes de temps executant-se als dispositius i sense drenar la seua bateria. Al llarg d'aquest treball es proposa la comparació entre els models que formen part de l'estat de l'art per a evaluar, tant les seues prestacions com a classificadors, com la seua latència i eficiència amb l'objectiu d'un posible entrenament amb un vocabulari específic i una implementació en temps real del model amb les millors prestacions.

Abstract

Keyword Spotting (KWS) models, have been widely adopted as a solution for simple voice commands execution. Since voice assistants such as Siri, Alexa or Google Assistant started to gain popularity on mobile devices, and more recently in the home automation field with devices like Google Home, Apple HomePod or Amazon Echo; KWS has been used as the activation method for the device with a keyword such as "Hey Siri" to start recognising more complex commands using language models. The key to understand the adoption of these models is their low resources consumption and the possibility to make them work for long periods of time, being executed on device without draining the battery. In this paper, a comparison of the state-of-the-art models is proposed to evaluate both their performance as classificators and their latency, to possibly train the best model with a specific vocabulary and real-time implementation.

A mis padres.

Índice general

Capítulo 1

Introducción a la detección de palabras clave

1.1. Definición del problema

La interacción con dispositivos mediante la voz es hoy en día algo habitual, pudiendo manejar con ella desde nuestro dispositivo móvil, hasta una instalación domótica o el sistema de infoentretenimiento de un automóvil. La tarea de reconocer y analizar el discurso a partir de una señal de audio que contiene voz hablada se define como *Automatic Speech Recognition (ASR)* o reconocimiento automático del discurso, y se sustenta en dos tipos de algoritmos principales: los basados en modelos de lenguaje y los de detección de palabras clave. Los primeros se encargan de reconocer cada palabra del discurso, teniendo como posibles casos de uso la generación automática de subtítulos en vídeos, o sistemas de asistente virtual como Siri, Google Assistant, etc. Los modelos de lenguaje, por la complejidad de sus tareas, son modelos computacionalmente costosos, pensados para ser ejecutados en servicios en la nube. Por ello, pueden resultar ineficientes a la hora de realizar tareas más sencillas, que no requieran comprender la totalidad del discurso que se está recibiendo, como la detección de un comando como "Oye Siri". Ahí es donde toman relevancia los modelos de detección de palabras clave (DPC), cuyo paradigma de funcionamiento es completamente distinto. Un modelo DPC se caracteriza por ser más liviano, por ello, puede reconocer la señal de audio en tiempo real, detectando aquellas palabras que conoce y descartando el resto de ellas. Todo esto con un consumo computacional muy bajo, y por ello se convierte en una buena solución para desempeñar funciones en la que el modelo tenga que estar "escuchando" durante largos pe-

ríodos de tiempo pero que el discurso detectado pueda no ser relevante. Además, están pensados para poder ejecutarse localmente en dispositivos móviles, en los cuales también ha de primar el ahorro en el consumo de la batería. La considerable adopción de este tipo de modelos en tiempos recientes es debido principalmente al cambio de paradigma que se produjo en este campo cuando se introdujeron los modelos basados en redes neuronales y aprendizaje profundo. Estos aportan una considerable mejora de la eficacia en la realización de sus tareas respecto a los modelos que existían anteriormente. Y debido a que los modelos basados en aprendizaje profundo tienen mucho más que ofrecer, las posibles aplicaciones que puede tener la detección de palabras clave son mucho más avanzadas de lo que fueron hace un tiempo.

1.2. Arquitectura de un modelo de DPC

Un modelo de detección de palabras clave es un clasificador. Este realiza la tarea de reconocer si la palabra que tiene a su entrada es conocida o no. Además, si detecta una palabra conocida, también decidirá de qué palabra del vocabulario conocido se trata.

En el caso de un clasificador basado en una red neuronal, lo que nos refleja la decisión tomada por el algoritmo son las neuronas de la capa de salida. El valor de salida de cada neurona de la última capa será el valor de confianza de cada categoría del clasificador.

1.2.1. Entrada

Como se puede observar en la figura 1.1, la información de entrada al algoritmo será, naturalmente, una señal de audio, la cual será preprocesada y se le extraerán las características necesarias para introducirlas en la red neuronal y que esta realice la predicción.

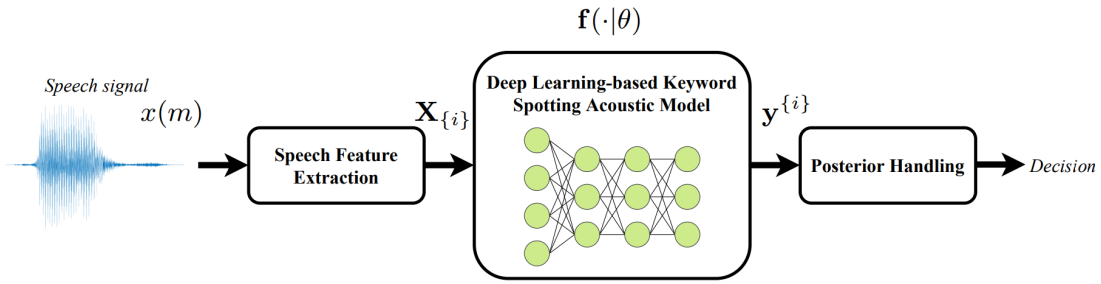

Figura 1.1: Diagrama básico de un modelo de DPC basado en deep learning [1]

1.2.2. Preprocesado y extracción de características

La señal de audio tiene que ser preprocesada en la práctica totalidad de los algoritmos con el fin de extraer valores que permitan identificar patrones en la misma, dichos valores se llaman características. En la mayoría de casos se emplean los MFCCs del audio como características. Los MFCC *(Mel Frequency Cepstral Coefficients)* son coeficientes que representan el audio en el dominio temporal y frecuencial, y cuyo cálculo está fundamentado en la percepción humana del

sonido. Estos están dispuestos en una matriz cuyo eje horizontal es el dominio del tiempo y el vertical el frecuencial, de forma que un valor alto en una coordenada (t, f) representa contenido espectral en la frecuencia f en el instante t. De esta forma, se convierte el audio, que es una señal unidimensional, en una matriz de dos dimensiones que puede ser interpretada como una imagen, como la que se observa en la figura 1.2.

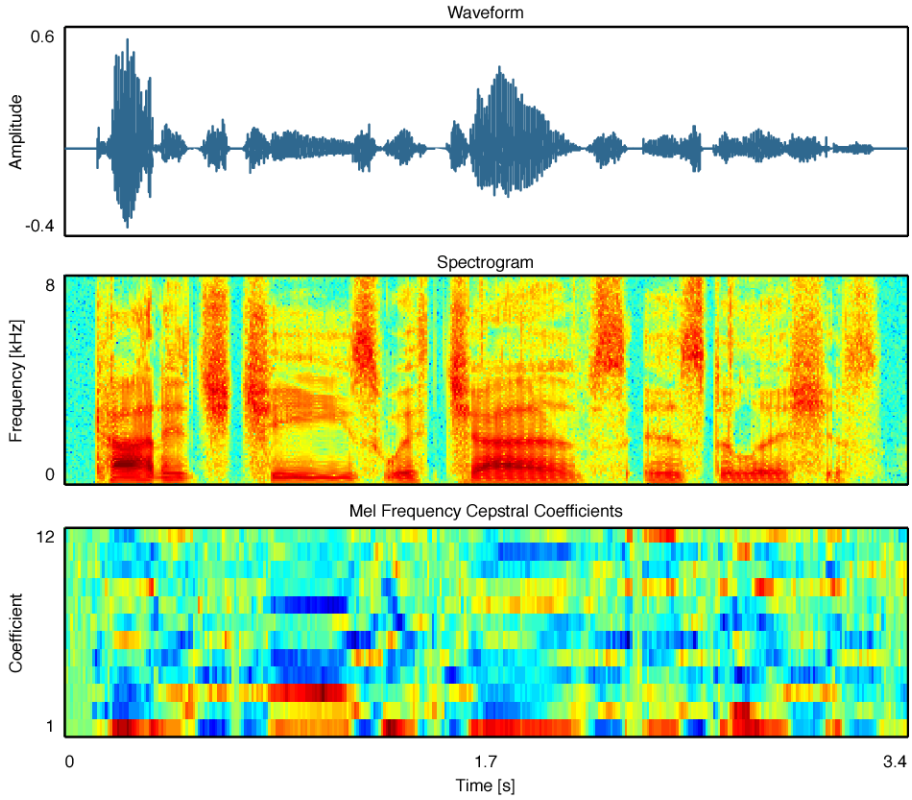

Figura 1.2: Representación gráfica de los MFCC [2]

Los MFCC se calculan de la siguiente forma:

1. Enventanado temporal de la señal
2. Transformada de Fourier discreta de la señal enventanada
3. Aplicar el banco de filtros de Mel a la DFT. Estos son una serie de filtros paso banda que replican la definición en frecuencia que tiene el oído humano, captando mejor las variaciones pequeñas de frecuencia en bajas frecuencias y al contrario en alta frecuencia. Por ello, y como se observa en la figura 1.3, los filtros con frecuencias centrales bajas tienen bandas

más estrecha y conforme aumenta la frecuencia central se va aumentando el ancho de la banda de paso.

4. Se toma el logaritmo de las energías de cada frecuencia de Mel.

5. Se aplica la transformada discreta del coseno a estos logaritmos.

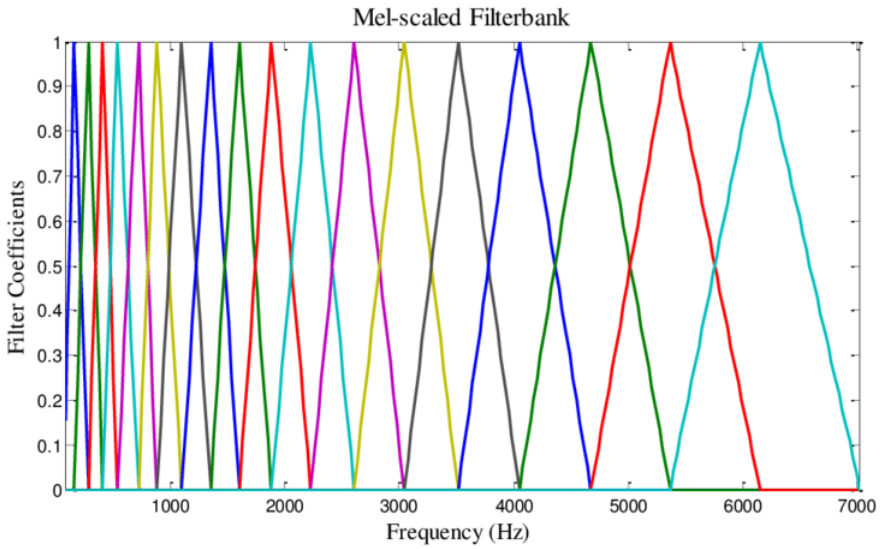

Figura 1.3: Esquema del banco de filtros de Mel [3]

1.2.3. Salida

En cuanto a la salida del algoritmo, cada neurona de la capa de salida va asociada a una de las palabras que se incluyen en el vocabulario, además de otra neurona que se activará cuando la palabra detectada sea desconocida y una última que se activará cuando no se detecte discurso en el audio recibido por la red.

Como se observa en la figura 1.4 la red asigna una puntuación entre [0,1] a cada neurona, que a su vez referencia a cada categoría, la máxima puntuación de la capa de salida se tomará como la decisión que ha tomado el clasificador y contra más próxima sea esta a 1, más confianza tendrá el clasificador de que la predicción es correcta.

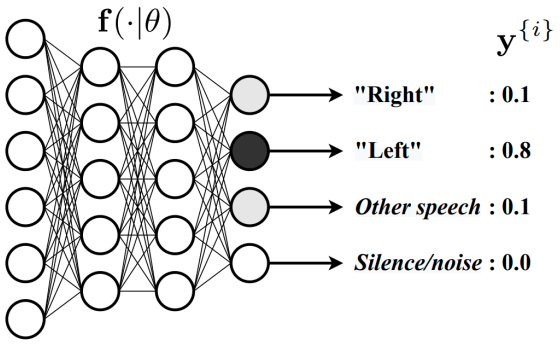

Figura 1.4: Ejemplo de salidas de un modelo de DPC basado en deep learning [1]

1.2.4. Implementación en tiempo real

Una red neuronal funciona correctamente cuando siempre recibe la información con el mismo formato. En el caso de un clasificador de audio, el audio tendrá que tener el mismo número de muestras, por ello todos los audios que reciba tendrán que durar lo mismo y tener la misma frecuencia de muestreo, tanto en el proceso de entrenamiento como en inferencia. Los algoritmos de DPC están pensados para funcionar en tiempo real, por lo que el audio que recibirá no es finito. Por ello, esta señal de audio se tendrá que dividir en ventanas temporales, las cuales se adaptarán al formato de la entrada del algoritmo. El clasificador realizará una inferencia para cada ventana temporal. A la salida del clasificador se tendrá un stream de las puntuaciones que salen de la última capa de la red. Cada frame de este stream de salida está asociado a una ventana temporal de audio de entrada. El método de enventanado es crucial para el correcto funcionamiento del clasificador, puesto que este sólo conoce la información contenida en cada ventana que recibe. Por ello, el objetivo en este proceso es aumentar al máximo las probabilidades de que una palabra clave quede dentro de una única ventana y no se divida en varias, ya que si esto ocurre, el clasificador recibirá una fracción de la palabra en cada ventana, y el algoritmo no entiende el contexto del resto de ventanas. Por ello, el tamaño de ventana tendrá que ser lo suficientemente grande como para que quepa la palabra más larga del vocabulario, teniendo en cuenta que una ventana grande introducirá latencia y se daría la posibilidad de que dos palabras entrasen en una misma ventana. Por otro lado, como se puede ver en la figura 1.5, es recomendable utilizar solapamiento para que sea menos probable que una palabra quede separada en dos ventanas distintas.

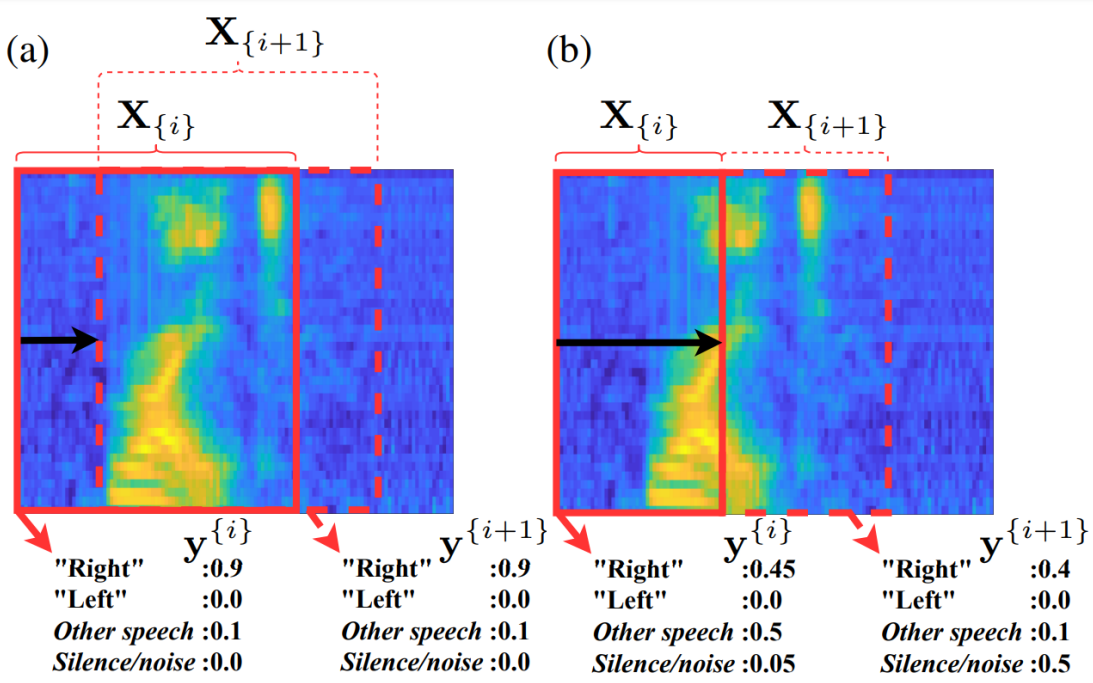

Figura 1.5: Funcionamiento del modelo en tiempo real con y sin solapamiento [1]

Capítulo 2

Estado del arte

Los investigadores en las tecnologías del reconocimiento del discurso vienen ya trabajando bastante tiempo en algoritmos de detección de palabras clave. Las primeras arquitecturas propuestas estaba basada en el uso de sistemas de reconocimiento continuo del discurso con vocabulario amplio *(large vocabulary continuous speech recognition, LVCSR)* [4]. Una de sus principales ventajas fue la flexibilidad que estos tenían cuando se cambiaban las palabras del vocabulario; no obstante, estos modelos tenían una complejidad computacional que los hacía inservibles para las funciones que desempeñan los algoritmos de DPC hoy en día. La alternativa que surgió posteriormente a estos fueron los algoritmos basados en los modelos ocultos de Markov (MOM)[5]. Estos algoritmos ya fueron algo más livianos y eficientes, aun así, requerían de decodificación Viterbi, la cual podía hacer que el MOM fuese más demandante a nivel computacional dependiendo de su topología.

En el año 2014 se presentó el primer modelo de DPC basado en redes neuronales. Este tipo de modelos supusieron una gran mejora en el campo, porque posibilitan tener un modelo que obtenga buenos resultados con baja complejidad computacional. Por ello, los modelos basados en redes neuronales se han adaptado como el estándar y son los que forman parte del estado del arte. Por ello, en este capítulo se realizará una breve descripción del funcionamiento básico de las redes neuronales para posteriormente pasar a describir las arquitecturas que forman parte del estado del arte en el ámbito de la detección de palabras clave.

2.1. Introducción a las redes neuronales

Como se ha dicho, las redes neuronales son el tipo de algoritmo sobre el que se basan los principales modelos del estado del arte en en ámbito de la detección de palabras clave. Antes de presentar y describir dichos modelos, en esta sección se presentarán los conceptos clave que permiten entender cualquier red neuronal y sus principios de funcionamiento.

Las redes neuronales son un tipo de algoritmos compuestos por unidades básicas de cálculo, llamadas neuronas. Estas se interconectan entre ellas formando capas, las cuales forman una red. Este tipo de algoritmos se incluyen en el campo del *machine learning*, por lo que su principal característica es que son capaces de desempeñar tareas gracias al aprendizaje adquirido en un proceso de entrenamiento.

2.1.1. La neurona

La neurona artificial tiene una estructura parecida a la neurona cerebral. Esta tiene varios parámetros de entrada, los cuales son recibidos y procesados para obtener un resultado a la salida de la neurona, el cual se toma como entrada en otra neurona distinta. Los cálculos realizados en la

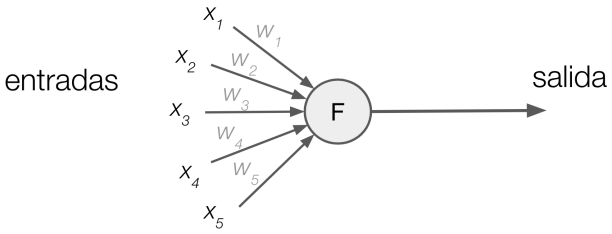

Figura 2.1: Estructura de una neurona

neurona son los siguientes:

1. Ponderación del valor entrada X_n con el peso W_n asignado a dicha entrada
2. Suma de todas las entradas ponderadas
3. Se aplica la función de activación al resultado de la suma ponderada de las entradas

Con lo cual, la fórmula final sería la siguiente:

$$y = f(\sum_{n=0}^{N}(X_n * W_n))$$

Siendo $f(x)$ la función de activación, la cual tendrá que cumplir el requisito de no ser lineal, de lo contrario, concatenar la salida de una neurona con la entrada de la siguiente sería equivalente a tener una neurona con las entradas de la primera y de la segunda. Las funciones de activación más frecuentes son la función ReLU, la Sigmoide y la Softmax. La forma que se tiene de definir el comportamiento esperado de la neurona es ajustando sus pesos de acuerdo con la salida esperada para una entrada dada. Más adelante se verá cómo se hace esto en el proceso de entrenamiento.

2.1.2. Perceptrón multicapa

Una vez visto el funcionamiento de la neurona, cabría plantearse cómo combinarlas para formar una red neuronal. Como ya se ha dicho, las neuronas se van concatenando de forma que la salida de unas es la entrada de las siguientes, pero a la vez las neuronas también se agrupan formando tal y como muestra la figura 2.2.

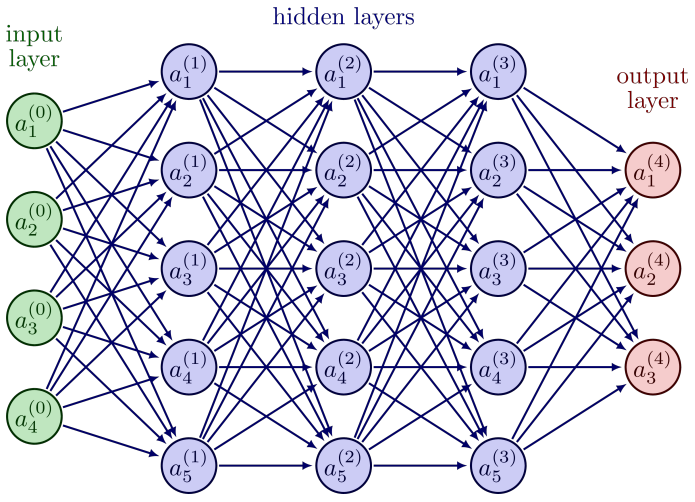

Figura 2.2: Estructura de una red neuronal

Donde el conjunto de las salidas de las neuronas de una capa son las entradas para cada una de las neuronas de la capa siguiente. Como se observa en la figura 2.2 se distinguen tres tipos de capas.

La de entrada, que tendrá tantas neuronas como características queramos introducir en la red. La capa de salida se tendrá que adecuar al tipo de problema que resuelva la red, y en el caso de un clasificador, esta capa tendrá tantas neuronas como clases tenga el problema de clasificación y a su vez, cada neurona corresponderá a una de las clases, por lo que la activación de cada neurona significará la detección de cada una de las clases conocidas. Por último las capas ocultas, que son las que agregan complejidad a la red y las que permiten reconocer y establecer patrones más complejos conforme se aumenta el número de capas ocultas y de neuronas por capa.

2.1.3. Entrenamiento de una red neuronal

El proceso de entrenamiento es aquel en el que la red neuronal ajusta los pesos de todas las conexiones entre sus neuronas para obtener las salidas deseadas ante las entradas que se le de a la red. Esto se hace mediante un mecanismo llamado *backpropagation*, que emplea el algoritmo del descenso por gradiente.

Lo primero necesario para el entrenamiento es una función de error, esta cuantifica lo bien que funciona la red y se puede entender como la diferencia que hay entre el resultado obtenido y el resultado esperado. Durante el proceso de entrenamiento, el objetivo será buscar un valor de los pesos para los cuales la función de error tiene un valor mínimo, de forma que el resultado obtenido es lo más similar posible al esperado. Esta búsqueda se realiza mediante el algoritmo de descenso por gradiente. El gradiente de una función se entiende como un vector que apunta hacia el máximo, y si la función de error se expresa en función de los pesos de la red, el gradiente de esta función de error nos dará la dirección en la que se tendrán que mover los pesos para llegar al máximo de la función de error. Si estos se mueven en la dirección contraria al gradiente, eso no necesariamente significa que se vaya en dirección hacia el mínimo, pero con un proceso de entrenamiento adecuado, finalmente se alcanzará el mínimo error y la red estará entrenada. Cuando se dice que los pesos ”se mueven”se hace referencia al mecanismo de *backpropagation*, que consiste en que se calcula la función de error en función de la salida obtenida y a partir de la función de error se modifican cada peso para minimizar el error, es decir, a partir de lo que se obtiene en la salida, se modifica el algoritmo para ir mejorándolo, de ahí el término *backpropagation*.

Dicho esto, se podría concluir con que el proceso de entrenamiento consiste simplemente en mostrar al modelo una cantidad suficiente de ejemplos de entrada y su correspondiente salida deseada. De esta forma el modelo es capaz de reconocer patrones en las entradas de forma que cuando se pase como entrada un ejemplo de una categoría conocida pero que no haya visto nunca, la red sea capaz de reconocerlo. La clave de un buen entrenamiento es que el modelo sea capaz de generalizar y no se especialice únicamente en los ejemplos que se han empleado para el entrenamiento. Por ello, siempre se reserva una cantidad de ejemplos llamado conjunto de validación, el cuál se empleará para verificar que el modelo los clasifica correctamente pero estos datos no contribuyen al entrenamiento, es decir, los pesos no se modifican si el modelo comete un error en un dato de validación. El objetivo de esto es que el porcentaje de acierto de los datos de entrenamiento y de validación sea parecido, de esta forma nos aseguraremos que el modelo es capaz de generalizar y no se está produciendo *overfitting*, que es el nombre que recibe este exceso de especialización de la red.

2.1.4. Limitaciones del perceptrón multicapa

Las redes neuronales han ido evolucionando y se han ido proponiendo distintos tipos de redes más complejas que mejoran su eficacia en otro tipo de problemas. El perceptrón multicapa, es la primera arquitectura de red neuronal que se propuso y la más elemental. Esta es altamente eficaz tratando información dispuesta en un vector de características de una dimensión. Por ello, cuando se intentó aplicar las redes neuronales al ámbito del reconocimiento de imágenes se tuvo que introducir un nuevo tipo de arquitectura que fuese capaz de reconocer o identificar patrones en información dispuesta en matrices de dos dimensiones, como es el caso de las imágenes. Esta arquitectura fue la red convolucional.

2.1.5. Redes neuronales convolucionales

La red neuronal convolucional introduce las capas convolucionales como mecanismo para el reconocimiento de patrones y extracción de características en disposiciones de información más extensa, especialmente en imágenes. Para ello utiliza la operación de la convolución. En una imagen,

la convolución en un punto es la suma ponderada por unos coeficientes de los valores de una región definida alrededor de ese punto. En una red convolucional, los coeficientes de la convolución se entrenan de forma que la red es capaz de extraer las características de una imagen por sí sola. En el caso de la detección de palabras clave, una posible imagen de la cual se podrían extraer características mediante este mecanismo, son los MFCC.

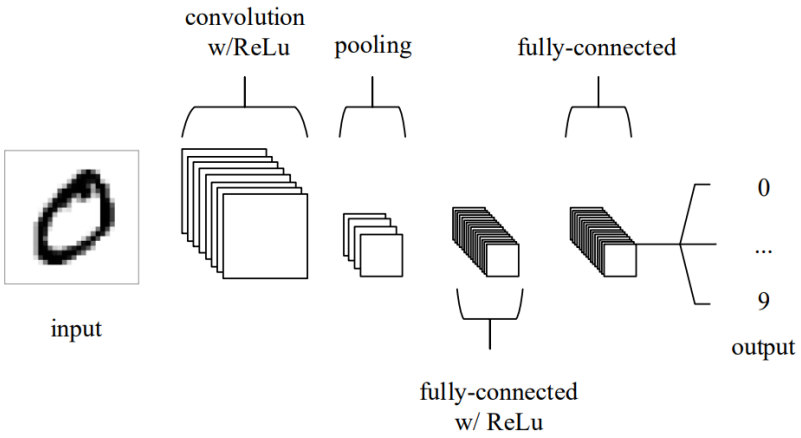

Figura 2.3: Estructura de una red neuronal convolucional [6]

Tal y como se observa en la figura 2.3, una red neuronal convolucional tiene tres tipos de capas:

1. Entrada

   Recibe los valores de los píxeles de la imagen de entrada.

2. Convolucional

   Se encarga de realizar la operación de convolución para realizar las transformaciones necesarias a la imagen. Puesto que la convolución es una operación lineal, al resultado de esta se le aplica la función de activación ReLu. De no aplicarse dicha función, el efecto de varias capas convolucionales concatenadas sería indiferente del de una única capa, por la linealidad de la convolución.

3. *Pooling*

   Esta capa se encarga de realizar un diezmado espacial de la imagen, es decir, reducir su resolución, con el fin de reducir también el número de parámetros de la red y aumentar

así su eficiencia, en la medida de lo posible y siempre que la complejidad de las imágenes tratadas lo permita.

4. *Fully-connected* Esta capa es la misma que encontramos en el perceptrón multicapa y se encarga de realizar la tarea de clasificación a partir de las características extraídas por las capas anteriores.

2.1.6. Modelos de detección de palabras clave

Los modelos de detección de palabras clave que forman parte del estado del arte alcanzan una complejidad mayor a la de una red neuronal convolucional, pero se fundamentan en la misma idea, y es que la red, al recibir el audio de entrada, sea capaz de extraer características del audio y reconocer aquello que caracterice a cada palabra. Los mecanismos que se emplean para ello son muy diversos, pero ahí es donde reside la complejidad y eficacia de los modelos más recientes.

2.2. *Keyword Transformer*

La arquitectura del *Transformer* [7], ha demostrado ser altamente eficaz en gran parte de los ámbitos de aplicación del deep learning, como el procesamiento natural del lenguaje, la clasificación de imágenes, el reconocimiento del discurso, detección de objetos, etc. La mejora principal que introducen los transformers son los mecanismos de atención, que permiten reconocer patrones de mayor complejidad que las redes recurrentes o convolucionales, que son las que se empleaban en este ámbito hasta el momento.

2.2.1. Mecanismos de atención

Los mecanismos de atención se introdujeron como mejora frente a la limitación que tenían las redes hora de modelizar secuencias y reconocer patrones en las mismas. Esta limitación consiste en que su capacidad de detectar relaciones entre componentes de una estructura de información, ya sea una imagen, una señal de audio o una frase escrita, depende en gran medida de la distancia que exista entre estos componentes. Esto puede suponer un problema, ya que puede darse el caso de que dos elementos que estén alejados guarden una relación, bien sea semántica, si nos referimos al ámbito del procesamiento del lenguaje, visual, o sonora, si nos referimos a imágenes o señales de audio. Los mecanismos de atención dan solución a esta problemática puesto que son capaces de conocer más contexto de la estructura de información que estén manejando.

El mecanismo de atención funciona generando un vector *query* y un vector *key* para cada elemento de la secuencia. Intuitivamente, podemos entender el vector *query* como la representación de cómo tiene que ser otro elemento para que el elemento al que hace referencia el vector le preste atención. Por otro lado, el vector *key* será una representación de las características del elemento al que hace referencia. Dicho esto, la atención es el producto escalar entre los vectores *query* y *key* de todos los elementos de la secuencia. De esta forma, la atención entre un *query* y un *key* parecidos, será cercana a 1, esto se puede entender como que lo que un elemento ”le interesaz las características de otro son afines, por lo que este le prestará atención. Cada elemento de la secuencia tendrá su propio vector de atención, el cual contendrá el grado de interés que generan en el cada uno de los

elementos que forman la secuencia.

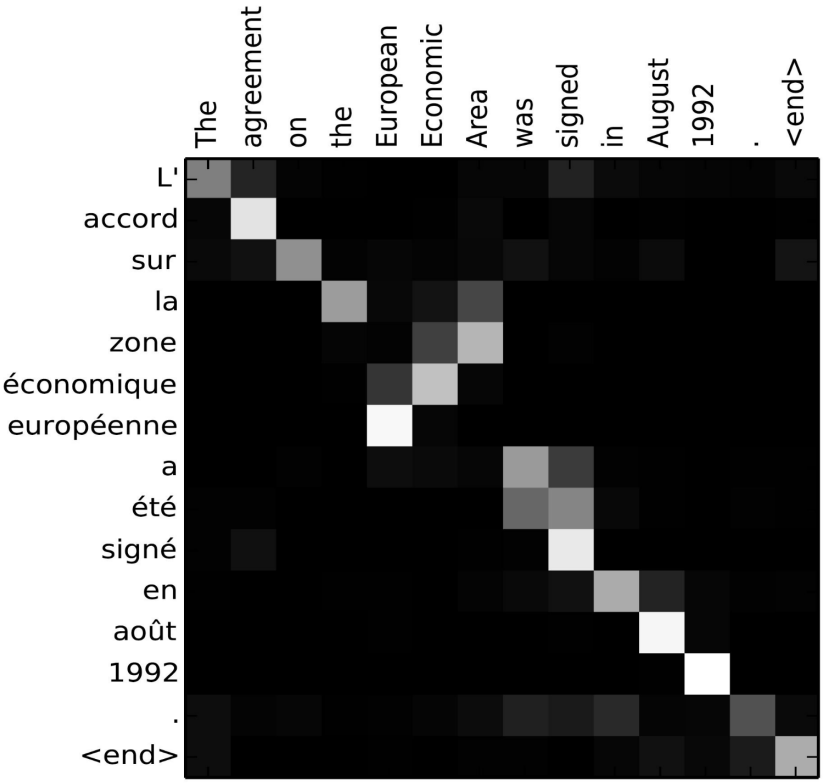

Figura 2.4: Ejemplo de una matriz de atención en la traducción de una frase [8]

En la figura 2.4 se puede ver un ejemplo de matriz de atención, donde cada fila o columna es el vector de atención del elemento de la secuencia. Los valores altos de atención se mantienen para las palabras que significan lo mismo y en los grupos de palabras que guardan una relación semántica. En definitiva, si se aplican los mecanismos de atención al procesamiento del lenguaje, la atención se puede entender intuitivamente como la atención que le prestaríamos los seres humanos al resto de palabras cuando estamos leyendo cada una de las palabras de una frase.

2.2.2. Arquitectura

La arquitectura del Keyword Transformer [9] parte de la un modelo basado en un transformer propuesto para el ámbito de la visión artificial llamado Vision Transformer[10], y lo adapta aplicando los mecanismos de atención en el dominio del tiempo a la hora de procesar una señal de audio, y

con ello obtiene una mayor eficacia a la hora de realizar la detección de palabras clave con alta eficiencia en cuanto a numero de parámetros y tamaño del modelo.

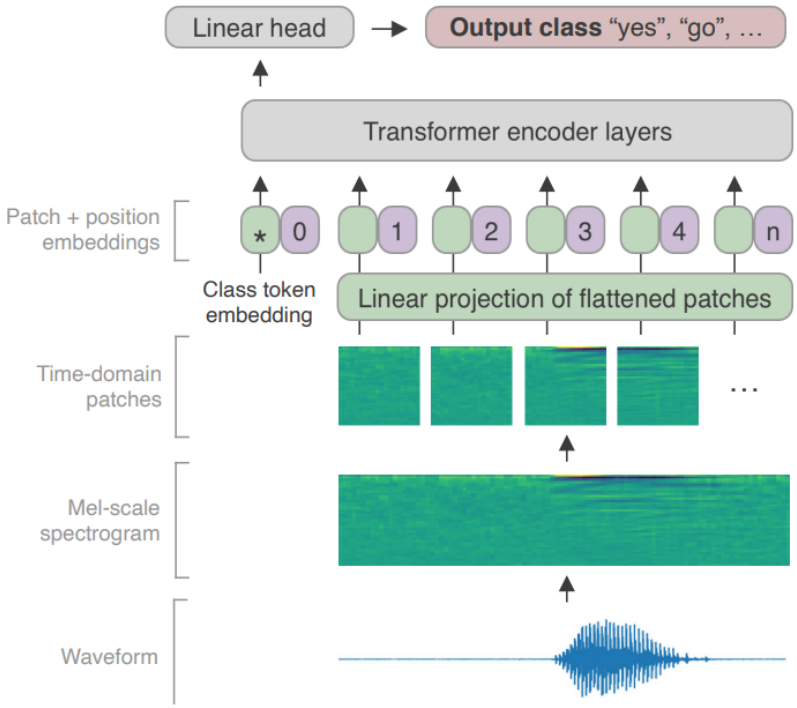

Figura 2.5: Arquitectura del Keyword Transformer [9]

En la figura 2.5 se puede observar la estructura de la arquitectura del keyword transformer. La señal de audio de entrada se preprocesa y se obtiene su espectrograma de Mel, que se divide en ventanas sin solapamiento en el dominio del tiempo. Cada ventana, o *patch*, se convierte en un vector de una dimensión, y junto con el *embedding* posicional que le corresponda, compone la entrada al Transformer encoder. El *embedding* posicional informa de la posición de un *patch* respecto al resto para proveer de contexto al Transformer y que pueda funcionar el mecanismo de atención ya que se puede establecer relaciones entre los *patches* temporales. Posteriormente, el *transformer encoder*, que tiene una estructura como la que se observa en la figura 2.6, realiza la extracción de características que entrarán al perceptrón multicapa que realizará la clasificación.

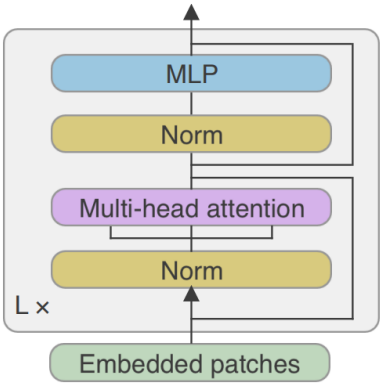

Figura 2.6: Estructura del transformer encoder del Keyword Transformer [9]

Como se observa en la tabla 2.1, extraída del artículo de referencia, las tres versiones de la arquitectura tienen distintas dimensiones, lo que resulta en distintos números de parámetros entrenables y por ello, distintas latencias, eficiencias y precisiones. El modelo KWT-1 tiene menor precisión pero es el que menor latencia tiene. En cambio, el modelo KWT-3 tiene mayor precisión pero a costa de una mayor latencia y el modelo KWT-2 se encuentra en un punto medio entre ambos.

Modelo	dim	mlp-dim	heads	capas	parámetros
KWT-1	64	256	1	12	607k
KWT-2	128	512	2	12	2,394k
KWT-3	192	768	3	12	5,361k

Tabla 2.1: Dimensiones de los modelos de la arquitectura Keyword Transformer

2.3. *Temporal convolution residual network*

2.3.1. ResNet

La arquitectura del ResNet surge ante una problemática que pueden presentar las redes neuronales durante su entrenamiento, y es que el algoritmo de descenso por gradiente pierde eficacia cuando las redes crecen en número de capas. Esto hace que cuando se tiene una arquitectura con muchas capas, el entrenamiento llegue a un punto en el que la red no aprende más ya que el descenso por gradiente no es capaz de modificar los pesos de la forma significativa. La solución que propone la ResNet ante esto es introducir el bloque que se muestra en la figura 2.7

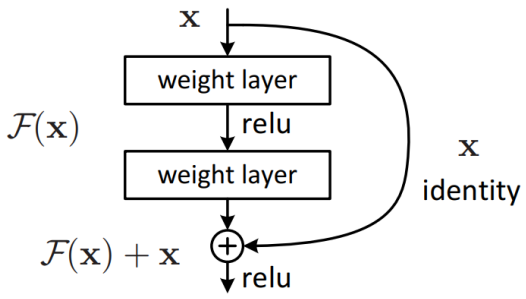

Figura 2.7: Bloque de red residual [11]

Este bloque no es más que una concatenación de capas, bien sean convolucionales o *fully connected*, pero con la particularidad de que a la salida de la última de capa se le suma la entrada de la primera de ellas. Con este tipo de conexiones entre las capas, ya se pueden entrenar redes con un gran número de capas sin peligro de que el entrenamiento no sea capaz de avanzar.

2.3.2. Arquitectura TC-ResNet

La arquitectura TC-ResNet [12] se propuso como una solución para la detección de palabras clave en dispositivos móviles. El artículo de referencia hace mención a un aumento de la velocidad del modelo de más de 300 veces respecto a un modelo de DPC del estado del arte. Esta arquitectura está basada en una red convolucional residual, es decir, bloques residuales formados por capas convolucionales, como se muestra en la figura 2.8.

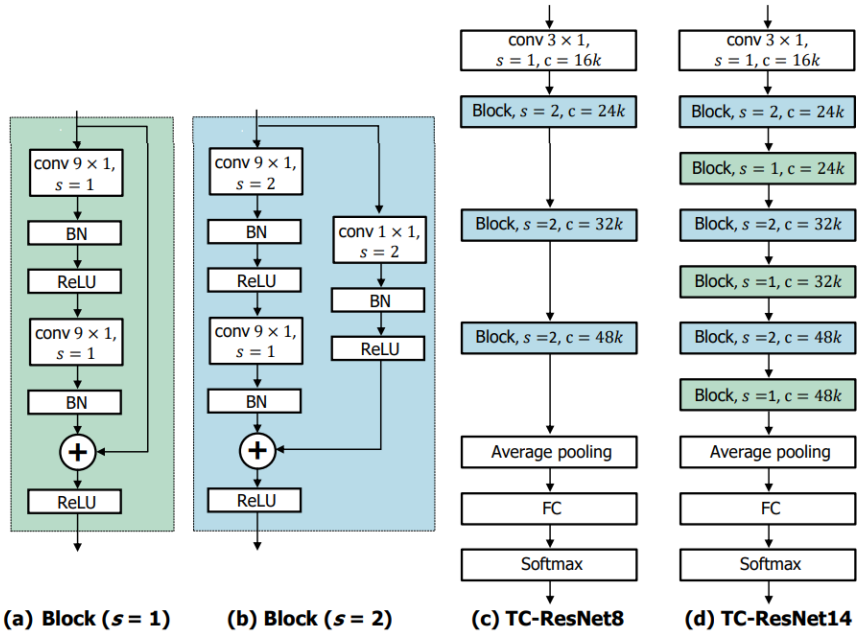

Figura 2.8: Arquitectura de TC-ResNet [12]

Además, en esta arquitectura se propone un mejora en la eficiencia de la arquitectura puesto que la convolución únicamente se realiza en el dominio del tiempo y esto tiene un menor coste computacional.

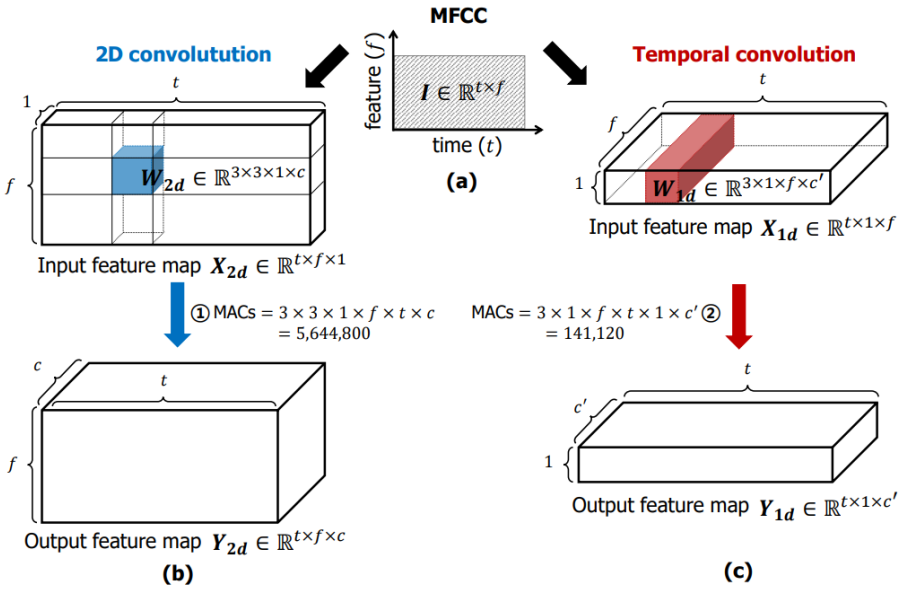

Figura 2.9: Diferencia ente convolución en 2D (b) y convolución temporal (c) [12]

En la figura 2.9 se puede observar las diferencias que existen entre la convolución en 2 dimensiones y 1 dimensión. Lógicamente, los resultados de ambas operaciones son distintos, pero la clave de esta arquitectura está en que, el resultado de la convolución de una dimensión, en este caso en el dominio del tiempo, es una buena representación de las características del audio y como ya se ha dicho, la convolución en una dimensión tiene un menor coste computacional y por ello, se calcula más rápido.

Los autores proponen distintos modelos con distintas dimensiones.

1. TC-ResNet8-1.0
2. TC-ResNet8-1.5
3. TC-ResNet14-1.0
4. TC-ResNet14-1.5

En este caso se proponen cuatro modelos con distintas combinaciones de números de bloques residuales y ancho de multiplicación. Los modelos TC-ResNet8 tienen 3 bloques residuales y 16,24,32,48 canales para cada capa. Los modelos TC-ResNet14 tienen el mismo número de canales para cada capa y 6 bloques residuales. El sufijo 1.5 hace referencia al ancho del multiplicador, que aumenta el número de canales de cada capa 1,5 veces, quedando la disposición de canales como 24, 36, 48, 72.

2.4. Xception1D

La arquitectura del Xception1D[13] se trata de una adaptación del modelo Xception [14]. El modelo original aprovecha las convoluciones separables para aumentar la eficiencia de esta operación, pero lo hace en matrices de 2 dimensiones. Lo que se propone en el modelo Xception1D es una adaptación de esta operación a vectores de 1 dimensión. En este caso no se propone ninguna variante con distintas dimensiones del modelo.

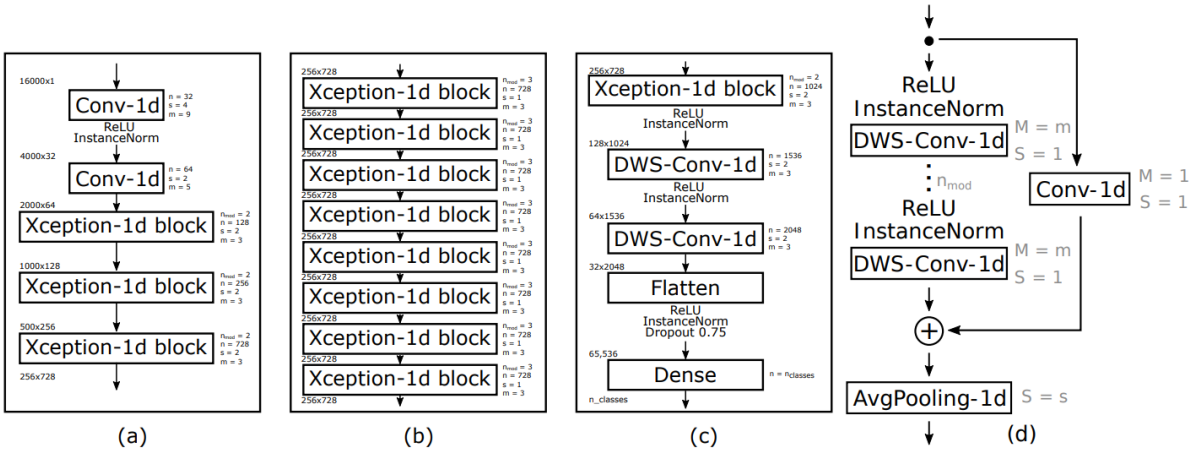

Figura 2.10: Arquitectura Xception-1D [13] donde n es el número de canales, s es el *stride* de las convoluciones, m es el tamaño de los filtros de convolución, nmod es el número de convoluciones y nclases es el número de salidas de la red. La arquitectura tiene 3 módulos principales: (a) es el módulo de entrada y se encarga de adaptar la señal de audio a una representación más sintetizada, (b) el módulo intermedio está encargado de aprender y reconocer la representación entregada por el módulo de entrada y (c) es el módulo de clasificación, que se encarga de realizar la predicción a partir de las características entregadas por el módulo intermedio. El bloque Xception-1d está representado en (d).

2.5. Wav2Keyword

Esta arquitectura se trata de un modelo de *transfer learning*, en el cual se emplea el *encoder* Wav2Vec 2.0 ya entrenado, que convierte una señal de audio en *embeddings* que representan el discurso de entrada. Dichos *embeddings* entran en una red, que es la que es entrenada, que realiza la clasificación a partir de los *embeddings* recibidos.

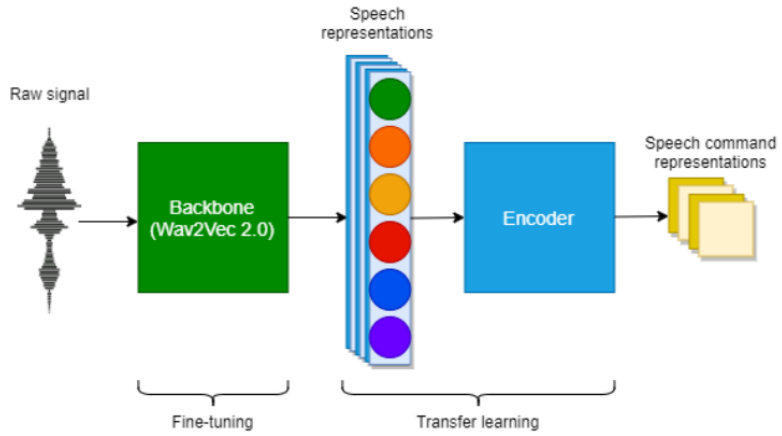

Figura 2.11: Esquema de la arquitectura Wav2KWS [15]

Capítulo 3

Objetivos

3.1. Motivación y contexto del estudio

El presente trabajo se realizó en un entorno profesional durante el período de prácticas en la empresa Voicemod, el fabricante de un software que cambia el sonido de la voz en tiempo real. Dentro de la empresa hay un departamento de investigación y desarrollo de tecnologías de audio, gran parte de las investigaciones que se llevan a cabo son sobre aplicaciones del *deep learning* procesado y análisis de la señal de audio. La motivación principal por parte de la empresa para llevar a cabo el proyecto es su voluntad de integrar funciones de interacción por voz en la aplicación. Por ello, estaba en marcha un proyecto investigando modelos basados en lenguaje, y por otro lado, se puso en marcha el proyecto de detección de palabras clave. Las directrices recibidas fueron que se quería saber cuales eran los modelos más recientes y ver cómo funcionaba cada uno para poder tomar una decisión a la hora de elegir un modelo en caso de que se quisiese emplear en la implementación de alguna función del software, para ello había que testear, naturalmente, al rendimiento en la clasificación pero también había que asegurarse de que los modelos pudiesen funcionar en tiempo real.
Dicho esto, el estudio realizado en este trabajo se trata de una comparativa entre los modelos de DPC descritos en el capítulo 2. La motivación para realizar dicha comparativa es la falta de información acerca del funcionamiento global de los algoritmos que proveen sus respectivos artículos de referencia. Esto es debido a que la finalidad de estos artículos es simplemente la de informar de

cuál es la aportación o la mejora que introduce el modelo presentado. Esta mejora o aportación en algún aspecto, no necesariamente tiene que implicar que el modelo cumpla otros requerimientos en otros ámbitos.

Es por ello que resulta interesante realizar tests a estos modelos, no sólo para probar cuál es su funcionamiento general, sino también para compararlos entre ellos y ver qué hace bien cada uno y poder tomar una decisión si en algún momento se quisiera implementar alguno de ellos en tiempo real para que desempeñe alguna tarea.

La intención a la hora de realizar estos test, por lo tanto, no es simular una situación real para realizar un test exhaustivo de cómo se comportarían los modelos bajo unas condiciones determinadas. El planteamiento es, más bien, el tratar de proveer a los modelos de igualdad de condiciones de test, para poder comparar los resultados de los modelos y poder ver dónde están las ventajas y desventajas de cada uno.

3.2. Objetivos generales y específicos

Una vez conocida la motivación y la razón por la que puede ser interesante realizar el estudio, se plantearán los objetivos generales y específicos, con el fin de poder tener una visión clara de cómo tendrán que plantearse los tests y pruebas a realizar para que estas puedan cumplir con dichos objetivos.

1. Objetivo general

    a) Evaluar los modelos en igualdad de condiciones para poder compararlos y juzgar objetivamente cuál sería el más adecuado si se plantease alguna aplicación concreta.

2. Objetivos específicos

    a) Entrenar todos los modelos para replicar los resultados de los artículos de referencia.

    b) Desarrollar el código de los tests de acuerdo con los requerimientos del estudio.

    c) Implementar el cálculo de métricas de rendimiento de los modelos para ser medidas durante las inferencias.

d) Tratar y procesar los datos obtenidos de los test para poder comparar los modelos de forma sencilla.

3.3. Planteamiento y condiciones de test

Una vez conocidos los objetivos del trabajo se planteará cómo serán los tests para que estos cumplan con los objetivos. En primer lugar, las evaluaciones se realizarán mediante un test de inferencia, es decir, el modelo ya entrenado hará sus predicciones sobre datos que no conoce. Esta sería la condición general de funcionamiento de cualquiera de los modelos, si en algún momento alguno de ellos se llevase a alguna aplicación real. Por otro lado, los datos de audio que se emplearán para realizar dichas tareas tendrán que ser los mismos para que exista igualdad de condiciones. Por lo tanto, todos los modelos se tendrán que entrenar con la misma base de datos. Además, este conjunto de datos tendrá que dividirse en conjunto de entrenamiento, evaluación y test. Estas divisiones también tendrán que ser iguales, en el sentido de que deberán contener las mismas muestras de audio de cada categoría, de esta forma, las estadísticas de clasificación que se obtengan serán comparables entre modelos y las diferencias que se vean serán producidas por el algoritmo y no por los datos empleados para entrenarlos ni para testearlo.

Por otro lado, si se quiere medir la velocidad de inferencia de los modelos, los tests deberán ser ejecutados siempre en el mismo dispositivo. Teniendo en cuenta que la mayoría de aplicaciones a las que van dirigidos este tipo de algoritmos, podríamos considerar que en la mayoría de los casos la ejecución de las inferencias se realizará en dispositivos móviles, por lo que el hardware empleado para el procesamiento de las mismas será una CPU, ya que la práctica totalidad de los dispositivos móviles carecen de una GPU dedicada a ejecutar inferencias en redes neuronales. Se podría dar el caso de algún modelo que funcione a una velocidad apta en una GPU pero no en una CPU. Para evitar esta situación se ejecutarán las inferencias en la CPU y de esa forma se obtiene una validación más fiable de que el modelo es apto para funcionar en tiempo real en un dispositivo móvil.

Capítulo 4

Metodología

En este capítulo se describirá en detalle todos los procedimientos llevados a cabo para cumplir con los objetivos planteados en el capítulo anterior y obtener los resultados deseados.

4.1. *Google Speech Commands Dataset*

Uno de los objetivos principales de este trabajo es realizar los tests en igualdad de condiciones para los modelos. Para ello, uno de los aspectos más relevantes a tener en cuenta es la base de datos de audios con el que se entrenen los modelos y se hagan los tests posteriormente. En este caso, se empleará la base de datos Google Speech Commands [16], propuesto por Google en el año 2018 y adoptado como el estándar para la realización de tareas de evaluación de modelos de detección de palabras clave.

La base de datos tiene dos versiones, ambas poseen las mismas categorías de palabras pero la versión más reciente (V2) contiene más audios de cada categoría. En este trabajo se empleará siempre la V2. Esta versión contiene un total de 105.829 grabaciones de 35 palabras clave distintas. En el artículo de referencia se propone un subconjunto de únicamente 10 palabras. Palabras:

- visual
- learn
- dog
- left*
- nine
- wow
- backward
- two
- happy
- go*

▪ up*	▪ tree	▪ right*	▪ house	▪ three
▪ bed	▪ seven	▪ eight	▪ marvin	▪ down*
▪ stop*	▪ on*	▪ no*	▪ sheila	▪ cat
▪ one	▪ four	▪ six	▪ five	▪ follow
▪ zero	▪ bird	▪ forward	▪ off*	▪ yes*

Nota: Las palabras marcadas con (*) pertenecen al subconjunto de 10 palabras.

Con estos dos conjuntos, se tienen dos modos de funcionamiento. El primero, y más básico, es aquel en el que el modelo se entrena empleando la base de datos completa para que éste diferencie entre las palabras de la base de datos. El otro modo, más similar a una situación de funcionamiento real, consiste en entrenar el modelo utilizando como palabras del vocabulario las 10 palabras del subconjunto y el resto de categorías etiquetarlas como desconocidas. De esta forma el modelo es capaz de diferenciar no solo las palabras clave del vocabulario entre ellas, sino que también es capaz de discriminar si la palabra que recibe es conocida o no. De esta forma, el modelo ha aprendido palabras desconocidas, que en una aplicación real de estos algoritmos, son las que más probabilidad de aparición tienen. Por ello, se podría considerar que el modelo de DPC hace una clasificación binaria y una clasificación multiclase a la vez. Por un lado detecta si una palabra recibida es conocida o no, y por otro, si la palabra es conocida, la clasifica entre las palabras del vocabulario.

Por otro lado, la base de datos se tendrá que dividir en subconjunto de entrenamiento, validación y test. Además, sería conveniente que estos subconjuntos contuviesen los mismos audios de cada categoría en el entrenamiento y test de todas las arquitecturas, para así mantener los datos lo más invariables posibles. Esto ha sido posible puesto que la base de datos, al ser ampliamente utilizada para este tipo de tareas, está integrada en las librerías *TensorFlow* y *Pytorch*, permitiendo estas crear una clase *Dataset* y mediante funciones incluidas en la librería, dividir la base de datos a conveniencia. Para este trabajo, la división se ha realizado en una proporción de un 80 % para entrenamiento, 10 % para validación y 10 % para test. De esta forma, el conjunto de test del sub-

conjunto de 10 palabras es de 4890 audios, 408 de los cuales son palabras desconocidas, y el de la base de datos completa es de 12106 audios, serán estos conjuntos los que se emplearán para realizar las inferencias para calcular las métricas de evaluación.

4.2. Proceso de entrenamiento

Todos los modelos se entrenarán y testearán en ambas situaciones, es decir, con las 35 palabras, que es un número más elevado de categorías pero sin palabras desconocidas, por lo que se testeará más a fondo la clasificación multiclase. Y por otro lado, con el subconjunto de 10 palabras, donde la clasificación multiclase no será tan exigente pero sí que se tendrá que realizar la clasificación binaria entre palabra conocida y no conocida. Además, como se ha presentado en el capítulo 2, los artículo de referencia de las arquitecturas *Keyword Transformer* y *TC-ResNet* proponen distintos modelos con distintas dimensiones y número de parámetros entrenables. Para evaluar estas arquitecturas más a fondo se entrenarán y testearán todos los modelos propuestos para cada una de las arquitecturas. En el proceso de entrenamiento el objetivo será replicar los resultados presentados en los artículos de referencia de los modelos realizando las mínimas modificaciones al código original, ya que en teoría, ese es el mejor resultado posible del entrenamiento que se puede obtener. El entorno que se empleará para entrenar los modelos será el servidor remoto de entrenamiento de la empresa Voicemod, equipado con una GPU *Nvidia RTX 3090* y una *Nvidia TITAN RTX*.

4.3. Métricas de evaluación

Capítulo 5

Resultados

En este capítulo se presentarán los resultados más relevantes obtenidos en los test de inferencia que nos permitan juzgar cómo funcionan los modelos y cual es su rendimiento en cada uno de los aspectos evaluados.

5.1. Modelos descartados

Tras realizar los tests se decidió descartar dos de los modelos candidatos presentados en el capítulo 2. Estos fueron el modelo Wav2KWS y el Xception1D, y la razón para ello es que no mostraron un rendimiento aceptable, por lo que no tenía sentido continuar calculando métricas o comparar estas con las de los otros dos modelos, puesto que hay una inferioridad considerable en prácticamente todos los aspectos.

5.1.1. Xception1D

Cuando se realizaron las primeras inferencias, se hizo notable la complejidad del modelo ya que el tiempo medio de cada inferencia era de unos 3 segundos. Este tiempo de inferencia es naturalmente incompatible con cualquier aplicación en tiempo real. Para descartar que la causa de este resultado fuese externa al modelo se implementó una función de python que nos permitiese conocer el

número de parámetros que tiene el modelo. Que el modelo tenga un número de parámetros elevado supone que el número de operaciones que se tienen que hacer por cada inferencia es elevado también, lo cual favorece que el tiempo de inferencia crezca. La función implementada devolvió un valor de 22,027,731 para el modelo de 10 palabras y de 23,600,619 para el de 35. Si tenemos en cuenta la referencia del número de parámetros que, según el artículo de referencia, tienen los modelos de una arquitectura como Keyword Transformer (Ver Tabla 2.1), observamos que el número de parámetros del Xception1D es significativamente más alto. Esto nos hace pensar que el elevado tiempo de inferencia se debe principalmente a la complejidad del modelo.

5.1.2. Wav2KWS

En el test de inferencia del modelo Wav2KWS se obtuvieron unos resultados de *accuracy* relativamente bajos, 85 % para el de 10 palabras y 81 % para el de 35 palabras. No son unos resultados malos, pero teniendo en cuenta la *accuracy* de los dos modelos restantes, no es relevante compararlos con Wav2KWS porque es un modelo con una *accuracy* considerablemente inferior.

5.2. Gráficos y análisis de resultados

En esta sección se mostrarán y revisarán los resultados más relevantes obtenidos a partir de los test de los modelos TC-ResNet y Keyword Transformer. Mediante gráficos que permitan comparar los valores de las métricas obtenidos por cada uno de los modelos se tratará de juzgar en qué aspectos un modelo es mejor que el resto y en qué posibles situaciones su funcionamiento supondría una ventaja.

5.2.1. Resultados de *accuracy* y latencia

Uno de los efectos más notables que se observó durante el análisis de los datos es el compromiso que existe entre la precisión del modelo y el tiempo que tarda en hacerse la inferencia. Esto tiene sentido ya que con un modelo de mayor complejidad, es capaz de alcanzar mejores resultados de *accuracy*, pero sacrificando el tiempo de inferencia ya que, al aumentar el número de parámetros del modelo, aumenta el número de operaciones realizadas por cada inferencia y con ello el tiempo que tarda en realizarse esta.

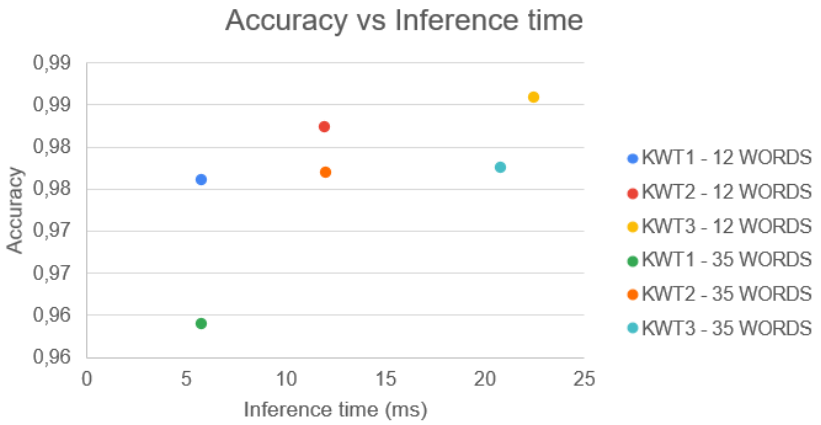

Figura 5.1: *Accuracy* y tiempo por inferencia de los modelos de *Keyword Transformer*

En el caso de la arquitectura *Keyword transformer* (ver figura 5.1), se observan valores de *accuracy* muy buenos. Alcanzando el modelo KWT-3 de 12 categorías un valor de accuracy por encima del 99 %. En este caso se ve claramente el efecto descrito anteriormente, según el cual los modelos con menor latencia tienen un accuracy relativamente menor. El modelo que mejor rinde en estos términos es el que se sitúe más cerca de la esquina superior izquierda del gráfico, es decir, mayor accuracy y menor tiempo de inferencia. Siguiendo este criterio, de los modelos de 12 categorías, los más óptimos son KWT-1 con menor latencia y KWT-2 con un poco más de accuracy pero también más latencia. No obstante, el modelo KWT-3 sigue resultando interesante por su excelente *accuracy*, ya que su latencia, si bien es la más alta de los tres modelos, puede ser aceptable en ciertas aplicaciones. Por otro lado, se puede observar que los puntos de un mismo modelo pero con distinto número de categorías están en la misma vertical, únicamente en el caso de KWT-1 y KWT-2, lo

cual nos indica que el hecho de agregar categorías de clasificación a los modelos no incrementa su complejidad puesto que su latencia se mantiene constante, salvo en el caso del KWT-3 en el cual sí que se observa un aumento de la latencia de unos 2ms.

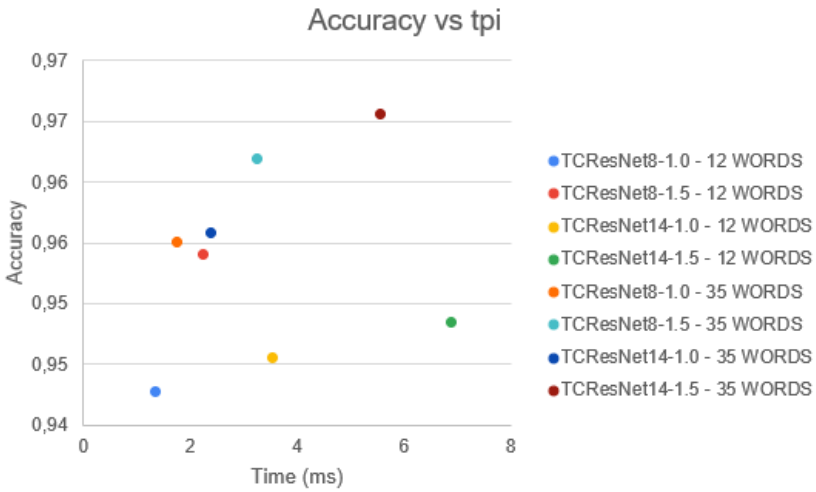

Figura 5.2: Accuracy y tiempo por inferencia de los modelos de TC-ResNet

En el caso de la arquitectura TC-ResNet (ver figura 5.2), se observa que son modelos, en general, más rápidos que el Keyword Transformer. Si se sigue el criterio empleado anteriormente, de buscar el modelo que se sitúe más próximo a la esquina superior izquierda, se puede concluir que, de los modelos de 12 categorías, el más optimizado es el TC-ResNet8-1.5, con un tiempo de inferencia de algo más de 2ms y una accuracy de casi el 96 %. Por otro lado, de los modelos de 35 palabras, el modelo óptimo también sería el TC-ResNet8-1.5, con una latencia de algo más de 3ms y un accuracy de más del 96 %.

En esta gráfica también se observa la tendencia según la cual los modelos con mayor accuracy tienen también mayor latencia, pero hay excepciones. En el caso de los modelos de 12 categorías, se observa que el modelo TC-ResNet8-1.5 tiene un accuracy por encima de la tendencia del resto de modelos. Por otro lado, en los modelos de 35 palabras, los modelos TC-ResNet8-1.5 y TC-ResNet14-1.0 tienen los puestos intercambiados, ya que el segundo, al ser más complejo, debería tener mayor accuracy y mayor latencia que el primero y no es así.

5.2.2. Tamaño del vocabulario y accuracy

En esta sección se analizará mas en detalle cómo afecta a los modelos en términos de accuracy, el hecho de aumentar el tamaño del vocabulario, es decir cuando se pasa de 10 palabras a 35.

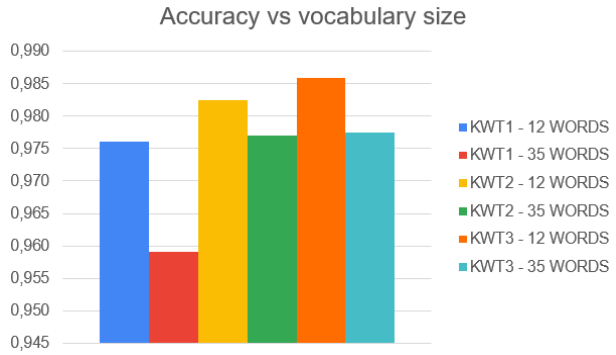

Figura 5.3: Accuracy y tamaño de vocabulario de los modelos de Keyword Transformer

Observando la gráfica que muestra los resultados del Keyword Transformer, se puede ver claramente que aumentar el tamaño del vocabulario penaliza negativamente la accuracy de los modelos. De las tres variantes, donde se observa este efecto de forma más pronunciada es en el KWT-1, en el cual la accuracy baja desde un 97,5 % hasta menos de un 96 %. Por el contrario, en el KWT-2 es donde se observa la mínima diferencia entre los dos tamaños de vocabulario.

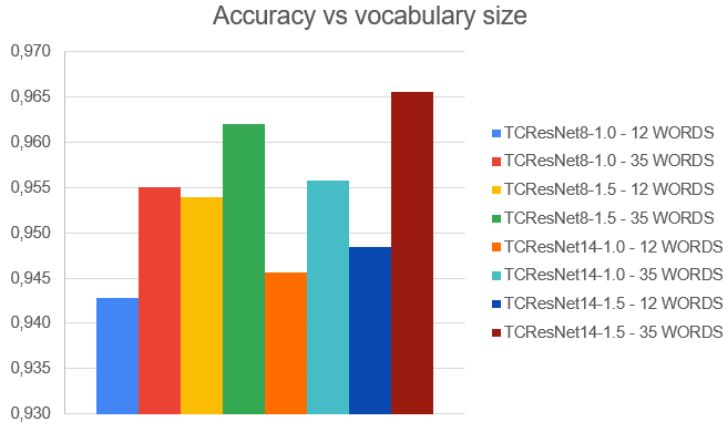

Figura 5.4: Accuracy y tamaño de vocabulario de los modelos de TC-ResNet

Por otro lado, tal y como se observa en la figura 5.4, la arquitectura TC-ResNet presenta el efecto contrario. Un mismo modelo tiene mayor *accuracy* funcionando con 35 palabras. Esta diferencia es máxima en el modelo TC-ResNet14-1.5 y mínima en el modelo TC-ResNet8-1.5.

5.2.3. Clasificación binaria

En esta sección se presentarán los resultados obtenidos relacionados con la clasificación binaria que hacen los modelos de 10 palabras. Recordemos que estos modelos han sido entrenados con palabras conocidas y desconocidas, y por ello los modelos también son capaces de diferenciar entre estos dos tipos de palabras. Los gráficos elaborados para evaluar los modelos son los siguientes.

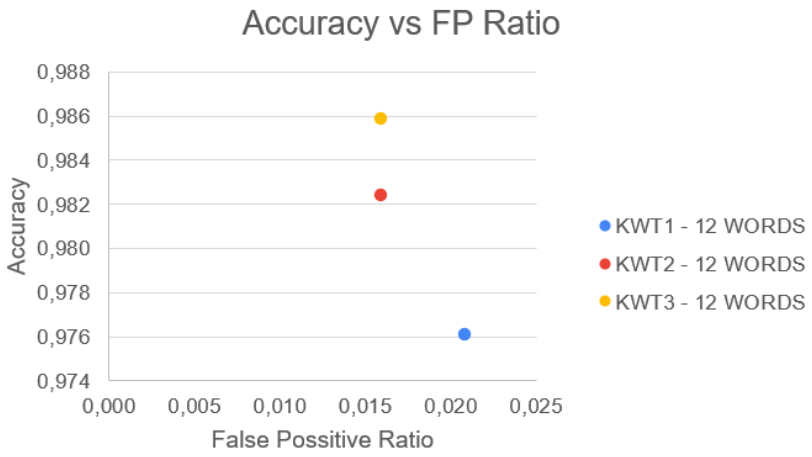

Figura 5.5: Accuracy y ratio de falsos positivos de KWT

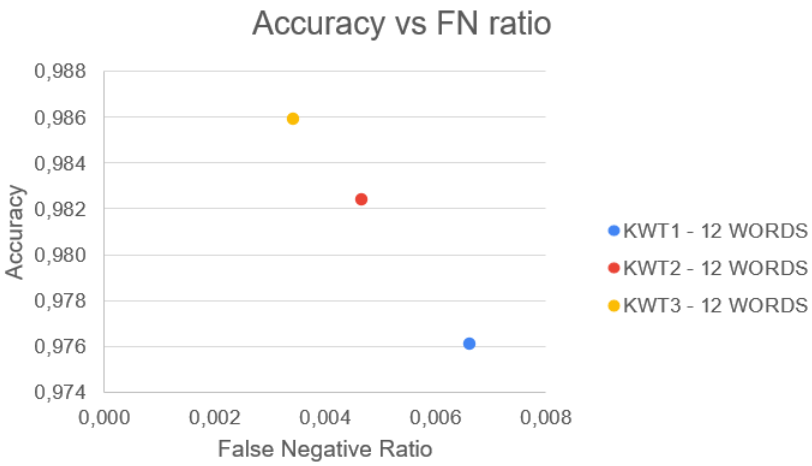

Figura 5.6: Accuracy y ratio de falsos negativos de KWT

Si se observa el orden de los valores de ratio de falso positivo y negativo, se puede apreciar que el keyword transformer tiende a cometer más falsos positivos5.5, es decir, suele reconocer palabras desconocidas como conocidas. No obstante, los valores de falsos positivos alcanzados en el test son más que aceptables. Por el contrario se ve que los valores de falsos negativos 5.6 son menores, siendo el caso peor (KWT-1) inferior al 1 %.

Se observa también que a menor valor de FPR y FNR, mayor accuracy, y resulta obvio, porque un falso positivo o falso negativo afecta negativamente al resultado de la accuracy. No obstante, se produce una excepción, y es que el modelo KWT-3 tiene el mismo valor de FPR que el KWT-2 pero una accuracy mayor, lo que quiere decir que el modelo no supone ninguna mejora en la tarea de clasificar palabras conocidas o desconocidas, pero las palabras conocidas sí que las diferencia mejor entre ellas.

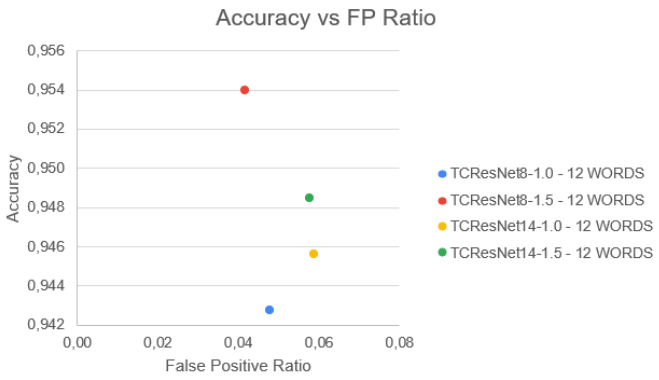

Figura 5.7: Accuracy y ratio de falsos positivos de TC-ResNet

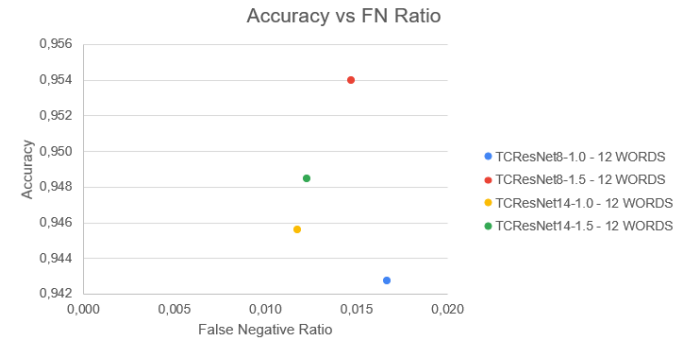

Figura 5.8: Accuracy y ratio de falsos negativos de TC-ResNet

En el caso de la arquitectura TC-ResNet, también se puede ver que comete más falsos positivos que falsos negativos, pero en este caso el valor del ratio de falsos positivos (FPR) (figura 5.7) es mayor que en el KWT, llegando en el peor de los casos a un 6 %. Ocurre lo mismo con el ratio de falsos negativos (FNR) (figura 5.8), el cual es menor que el ratio de falsos positivos, pero el KWT ha obtenido mejores resultados también en esta métrica. Respecto a la relación entre la *accuracy* y el FPR, existe la tendencia observada anteriormente, según la cual, este ratio mejora con la *accuracy*, aunque no es una tendencia lineal, y el modelo TC-ResNet8-1.0 se sale de esta tendencia puesto que, teniendo la *accuracy* más baja tiene el segundo FPR más bajo. Observando el gráfico del FNR (figura 5.8), se puede ver que los puntos guardan cierta simetría vertical con la gráfica del FPR. Esto quiere decir, que aquí no se cumple el hecho de que contra mayor *accuracy*, menor es el FNR. Aunque el modelo con peor resultado en este ratio es el que tiene también peor *accuracy*, este es la excepción de los cuatro, puesto que el que mejor FNR ha obtenido es el segundo con peor *accuracy*.

5.2.4. Rendimiento temporal

En esta sección se estudiarán los resultados obtenidos referentes al rendimiento temporal de los modelos con el fin de poder juzgar cómo se comportarían en este sentido en una situación real de funcionamiento en *streaming*.

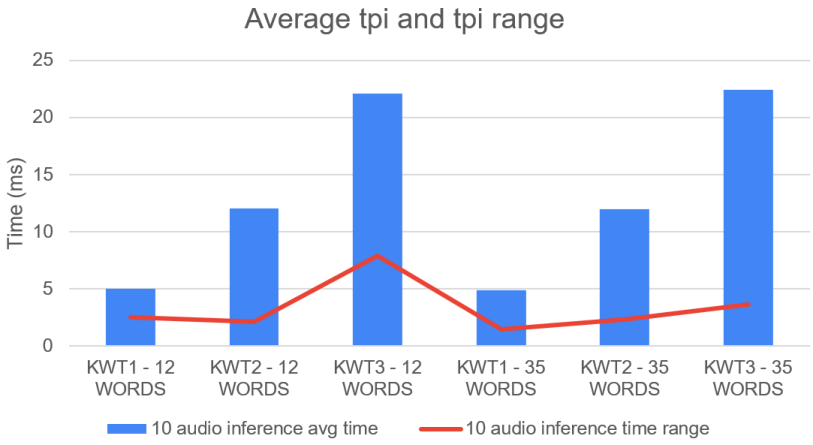

Figura 5.9: Métricas temporales de los modelos de KWT

En el gráfico de la figura 5.9 se pueden observar los resultados de la arquitectura de Keyword Transformer. Se puede ver claramente el efecto del aumento del tamaño y complejidad de los modelos en el tiempo de inferencia. Lo que muestran las barras azules es la media de tiempo por inferencia de 10 inferencias seleccionadas aleatoriamente de entre todas las realizadas durante el test. Por otro lado, la línea roja representa el valor del rango de los 10 tiempos sobre los que se ha calculado la media. Al ser los rangos relativamente bajos, con respecto al tiempo medio de inferencia, podemos tomar el valor medio obtenido como bueno. Que un modelo tenga un tiempo de inferencia alto tiene implicaciones directas sobre la posibilidad de que este funcione en tiempo real, en este caso, el modelo más lento es el KWT-3 con un tiempo medio de inferencia de 22ms, que es una latencia más que aceptable para el funcionamiento en streaming de un modelo de DPC.

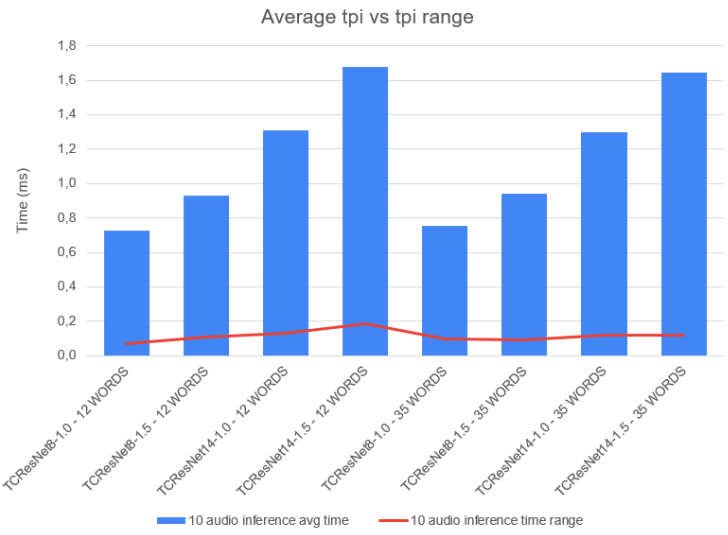

Figura 5.10: Métricas temporales de los modelos de TC-ResNet

Si se observa la gráfica de la figura 5.10, que muestra los resultados de los modelos de la arquitectura TC-ResNet, se puede ver de nuevo que los modelos de mayores dimensiones tienen mayor latencia. Pero en este caso los valores de rango son representativamente más bajos en relación al tiempo medio de inferencia, lo que nos hace pensar que la latencia es más invariante y por ello este modelo sería más estable funcionando en streaming. El resultado que más llama la atención es el tiempo medio de inferencia, y es que el modelo con mayor latencia ha obtenido un valor de algo más de 1.6ms que es incluso más rápido que el modelo más liviano de la arquitectura KWT.

Capítulo 6

Conclusiones y propuestas de trabajos futuros

Tras la revisión realizada de los datos, exponiendo en el capítulo anterior aquellos más relevantes, en este capítulo se sacarán algunas conclusiones en base a los datos obtenidos y se propondrán diversas líneas de trabajo que surgen a partir de este estudio y que podrían aportar resultados e información relevante en el ámbito de la detección de palabras clave.

6.1. Cumplimiento de objetivos

En primer lugar, se revisarán los objetivos planteados al inicio del presente trabajo con el fin de poder juzgar si, una vez realizado el estudio, se han cumplido.

1. **Evaluar los modelos en igualdad de condiciones**

   Gracias a el uso que se ha hecho del dataset Google Speech Commands para el entrenamiento y la inferencia de los modelos, creando los subconjuntos de entrenamiento, validación y test de la misma forma para todos los modelos; además de haber ejecutado las inferencias en las mismas condiciones de hardware, se podría concluir que ha habido igualdad en las condiciones de test para todos los modelos.

2. **Obtener información sobre el funcionamiento de los modelos en su conjunto**

   Los tres aspectos a los que hay que prestar atención para conocer la eficacia de un modelo de DPC son: la clasificación multiclase entre palabras del vocabulario, la clasificación binaria entre palabra conocida o desconocida y la eficiencia temporal. Las métricas obtenidas en las inferencias son las que permiten recabar toda esta información para luego ser analizada y poder saber de forma objetiva cómo rinden los distintos algoritmos los tres aspectos.

3. **Comparar modelos para poder juzgar objetivamente cuál sería el más adecuado si se plantease alguna aplicación concreta**

   A partir de los datos numéricos de las métricas obtenidos, se elaboraron distintos gráficos con el fin de poder evaluar de una forma más sencilla y visual cómo funciona cada modelo y saber cuales son las ventajas y desventajas de cada modelo.

6.2. Conclusiones sobre los modelos testeados

En esta sección se realizará un análisis resumido de los resultados obtenidos por cada arquitectura y se tratará de proponer distintos casos de uso para los cuales cada arquitectura sería la óptima, únicamente teniendo en cuenta las dos arquitecturas que han obtenido resultados aceptables, que son Keyword Transformer y TC-ResNet.

El primer modelo tiene una mejor accuracy, alcanzando la mejor de las variantes un máximo de un 99 %, pero algo más de latencia que el segundo, siendo la mínima latencia obtenida por el modelo KWT-1 de 5 ms. Además, obtuvo mejores resultados con un vocabulario más pequeño y en la clasificación binaria. Por ello, esta arquitectura resultaría óptima en aplicaciones de *wake-word detection* o ejecución de comandos por voz, donde la latencia y tiempo de respuesta del sistema no es tan relevante y lo que realmente mejora la calidad del servicio es el reconocimiento preciso de la palabra pronunciada.

Por otro lado, el modelo TC-ResNet ha mostrado un rendimiento temporal muy superior al del Keyword Transformer, con latencias entre 1 ms y 6 ms. No obstante, sus métricas de clasificación han resultado ser inferiores, aunque aceptables, en el mejor de los casos se ha obtenido un 96 %

de *accuracy*. Además, se ha visto que funciona mejor cuando el vocabulario incluye más palabras. El caso de uso óptimo de esta arquitectura sería aquel en el cual lo decisivo sea la baja latencia, la detección en tiempo real y una cantidad de palabras conocidas elevada. En ese sentido, es posible que la arquitectura TC-ResNet se pudiese emplear como un detector de palabras clave, que más allá de detectar comandos de voz, sería capaz de analizar un discurso más natural en tiempo real en busca de palabras conocidas.

6.3. Futuras lineas de trabajo

6.3.1. Estudio del tamaño del vocabulario óptimo

Como se ha observado en el capítulo anterior, la *accuracy* de los modelos variaba en función del tamaño del vocabulario, viendo que el Keyword Transformer funciona mejor con el subconjunto de 10 palabras y la arquitectura TC-ResNet funciona mejor con el vocabulario de 35 palabras. Para estudiar si esto es algo propio de los modelos y no es algo coyuntural, se podría plantear un estudio para averiguar si existe alguna relación clara entre el tamaño del vocabulario y las métricas de clasificación. Un posible planteaminento para realizar dicho estudio podría ser, tomando como hipótesis que el Keyword Transformer funciona mejor con vocabularios pequeños, entrenarlo con varios vocabularios cada vez más pequeños, para ver si existe alguna tendencia y el accuracy sigue mejorando conforme disminuye el número de palabras conocidas. De la misma forma, se podría entrenar con vocabularios de más de 30 palabras para ver si sigue empeorando. Con la arquitectura TC-ResNet el planteamiento sería el mismo, pero sabiendo que este modelo se comporta de forma inversa al Keyword Transformer y funciona mejor con vocabularios grandes, lo interesante sería ir aumentando el tamaño del vocabulario para ver si el valor del accuracy sigue mejorando y en ese caso, ver hasta donde puede llegar.

6.3.2. Testeo en tiempo real

Todos los test realizados en este trabajo han sido en modo *non-streaming*, es decir, los audios que se han empleado son finitos y únicamente contienen una palabra, además estos audios están

etiquetados, es decir que se conoce la palabra que contienen, para facilitar la obtención de las métricas de evaluación. No obstante, las condiciones de funcionamiento de un algoritmo de DPC en una aplicación real son muy distintas. Por ello, el próximo paso hacia poder implementar alguno de estos modelos en una aplicación real, sería testearlo modo *streaming*, es decir, en tiempo real. (Ver 1.2.4).

Para dicho estudio se emplearían audios de discurso natural, los cuales se podrían pasar por el modelo y testear la clasificación en tiempo real. El inconveniente de este test sería el cálculo de métricas de evaluación, ya que en modo *streaming*, cada inferencia se hace para un audio, sino que va ligada a un fragmento del mismo, el cual no tiene por qué contener una palabra completa etiquetable. Es por ello que el etiquetado debería tratarse más bien de una transcripción de los audios, en los que cada palabra de la transcripción contiene a su vez una marca temporal que informa de en qué momento se pronuncia dicha palabra dentro del audio. Gracias a esa marca temporal se podría relacionar una inferencia, o grupo de inferencias del bucle, a cada palabra para poder evaluar si ha sido detectada y clasificada correctamente.

6.3.3. Estudio de calidad de servicio

Gracias a los tests realizados en el presente trabajo se ha abierto la posibilidad de evaluar y comparar las distintas arquitecturas para poder juzgar cómo funciona cada una en los distintos aspectos relevantes a evaluar para considerar un correcto funcionamiento del modelo. No obstante, para continuar optimizando las arquitecturas y que el estado del arte avance en la dirección correcta; sería razonable estudiar cuáles son los requerimientos técnicos de un servicio de detección de palabras clave de cara al usuario final. Para ello, se podrían abandonar las métricas y realizar tests subjetivos para poder conocer qué es lo que enriquece un servicio de DPC. Poder conocer cuál es la latencia máxima que se podría considerar aceptable para un servicio de DPC, o la accuracy mínima; saber si los usuarios prefieren que un modelo cometa más falsos positivos o más falsos negativos. Esta información podría ayudar en la toma de decisiones a la hora desarrollar nuevas arquitecturas y elegir cuál emplear para según qué aplicación.

Bibliografía

[1] Iván López-Espejo y col. "Deep Spoken Keyword Spotting: An Overview". En: *IEEE Access* 10 (2022), págs. 4169-4199. DOI: 10.1109/ACCESS.2021.3139508.

[2] Stefan Scherer. "Analyzing the User's State in HCI: From Crisp Emotions to Conversational Dispositions". Tesis doct. Ene. de 2011.

[3] Yusnita mohd ali y col. "Analysis of Accent-Sensitive Words in Multi-Resolution Mel-Frequency Cepstral Coefficients for Classification of Accents in Malaysian English". En: *International Journal of Automotive and Mechanical Engineering* 7 (jun. de 2013), págs. 1053-1073. DOI: 10.15282/ijame.7.2012.21.0086.

[4] M. Weintraub. "Keyword-spotting using SRI's DECIPHER large-vocabulary speech-recognition system". En: *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 2. 1993, 463-466 vol.2. DOI: 10.1109/ICASSP.1993.319341.

[5] J.R. Rohlicek y col. "Continuous hidden Markov modeling for speaker-independent word spotting". En: *International Conference on Acoustics, Speech, and Signal Processing,* 1989, 627-630 vol.1. DOI: 10.1109/ICASSP.1989.266505.

[6] Keiron O'Shea y Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. DOI: 10.48550/ARXIV.1511.08458. URL: https://arxiv.org/abs/1511.08458.

[7] Ashish Vaswani y col. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[8] Dzmitry Bahdanau, Kyunghyun Cho y Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. DOI: 10.48550/ARXIV.1409.0473. URL: https://arxiv.org/abs/1409.0473.

[9] Axel Berg, Mark O'Connor y Miguel Tairum Cruz. "Keyword Transformer: A Self-Attention Model for Keyword Spotting". En: *Interspeech 2021*. ISCA, ago. de 2021. DOI: 10.21437/interspeech.2021-1286. URL: https://doi.org/10.21437%2Finterspeech.2021-1286.

[10] Alexey Dosovitskiy y col. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. DOI: 10.48550/ARXIV.2010.11929. URL: https://arxiv.org/abs/2010.11929.

[11] Kaiming He y col. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[12] Seungwoo Choi y col. *Temporal Convolution for Real-time Keyword Spotting on Mobile Devices*. 2019. DOI: 10.48550/ARXIV.1904.03814. URL: https://arxiv.org/abs/1904.03814.

[13] Iván Vallés-Pérez y col. *End-to-end Keyword Spotting using Xception-1d*. 2021. DOI: 10.48550/ARXIV.2110.07498. URL: https://arxiv.org/abs/2110.07498.

[14] François Chollet. *Xception: Deep Learning with Depthwise Separable Convolutions*. 2017. DOI: 10.1109/CVPR.2017.195.

[15] Deokjin Seo, Heung-Seon Oh y Yuchul Jung. *Wav2KWS: Transfer Learning From Speech Representations for Keyword Spotting*. 2021. DOI: 10.1109/ACCESS.2021.3078715.

[16] Pete Warden. *Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition*. 2018. DOI: 10.48550/ARXIV.1804.03209. URL: https://arxiv.org/abs/1804.03209.