eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Análisis de herramientas de Reconocimiento Biométrico e implementación de un modelo simple de Reconocimiento del Habla

Autor

Julen Elexpuru Ortiz

Director

Basilio Sierra

Tutor en empresa

Asier Ayestaran

informatika fakultatea
facultad de informática

Ibermática

2018

Agradecimientos

Quisiera aprovechar esta oportunidad para agradecer en primer lugar a todos los profesores que han hecho que llegue hasta aquí, especialmente aquellos que han tenido la suficiente paciencia para sacar de mi lo mejor que pudiera ofrecer en cada circunstancia.

Igualmente agradecer a familia y compañeros de estudios que mediante su esfuerzo y apoyo (económico, académico y moral) en las dificultades han conseguido finalmente hacerme prosperar a lo largo de estos largos años de carrera.

Finalmente agradecer también a mis responsables y compañeros de trabajo por haberme dado nada mas que facilidades, experiencia y buenos consejos para este y tantos otros proyectos.

Resumen

Este proyecto consiste en la mejora de una herramienta web en proceso de desarrollo en la empresa en la que se ha realizado el trabajo mediante la inclusión de elementos basados en la Inteligencia Artificial como pueden ser las Tecnologías de Reconocimiento Facial, el Reconocimiento de Voz y el Procesamiento del Lenguaje Natural (FRT, ASR y NLP respectivamente por sus siglas en Inglés).

Para ello, se han evaluado las diferentes soluciones comerciales que había disponibles y se ha utilizado la que más se adecuaba a las necesidades del proyecto. Adicionalmente y de cara a optimizar el proceso, se ha implementado un programa de reconocimiento de voz propio que permita reconocer una palabra de activación del servicio.

Índice general

Agradecimientos I

Resumen III

Índice general V

Índice de figuras IX

Índice de tablas XI

1. Introducción 1

1.1. Contexto 1

1.2. Estado del arte 2

1.2.1. Reconocimiento Facial 2

1.2.2. Reconocimiento de Voz 2

1.3. Alternativas comerciales y Propuesta 3

2. Gestión del Proyecto 5

2.1. Alcance del Proyecto 5

2.1.1. Alcance Empresarial 5

2.1.2. Alcance Académico 6

2.1.3. Exclusiones 7

2.2. Adquisición de Conocimientos 7
2.2.1. Aprendizaje teórico de diferentes fundamentos: 7
2.2.2. Lenguajes de programación: 7
2.2.3. APIs y Librerías 8
2.3. Herramientas utilizadas 8
2.3.1. Lenguajes de programación y librerías 8
2.3.2. Herramientas de Terceros 9
2.3.3. Keras 11
2.3.4. Control de versiones y Copias de Seguridad 12
2.3.5. Gestión del tiempo 12
2.4. Planificación 12
2.5. Análisis de Riesgos 12
2.6. Análisis de Factibilidad 13
2.7. Dedicación 13

3. Conceptos y Tecnologías **15**
3.1. Tratamiento de señales de sonido 15
3.1.1. Particularidades de la voz humana 15
3.1.2. La voz humana como señal auditiva 16
3.1.3. Coeficientes Cepstrales de las Frecuencias de Mel 18
3.2. Redes Neuronales 20
3.2.1. ¿Que es una Neurona?(Antecedentes biológicos) 20
3.2.2. Estructura y funcionamiento de las redes neuronales 22
3.2.3. Tipos de redes neuronales 27
3.3. Reconocimiento de Voz 32
3.3.1. Enfoques planteados hasta ahora 32

4. Desarrollo **37**

4.1. Reconocimiento Facial 37

4.2. Reconocimiento del Habla 40

4.3. Reconocimiento de Palabras Clave 43

5. Análisis del proyecto **47**

5.1. Complicaciones 47

5.2. Resultados 48

6. Conclusiones **51**

6.1. Conclusiones generales del proyecto 51

6.2. Lineas futuras 52

Anexos

A. Experimentación **57**

A.1. Tipo de red 58

A.2. Muestras 59

A.3. Tipo de activación 60

A.4. Tipo de error y optimizadores 60

A.5. Modificación de parámetros 61

B. Código **65**

B.1. Recorder 65

B.2. Reconocimiento 70

B.3. Main 75

B.4. Menus 82

Bibliografía **85**

Índice de figuras

3.1. Aparato fonador humano 16

3.2. Espectrograma en escala de Mel de la palabra *comando*. A mayor intensidad en una frecuencia, el color se vuelve más vivo. 17

3.3. Vocales del español clasificadas según sus dos primeros formantes [Bradlow 1995] 18

3.4. Proceso de obtención de los MFCC 20

3.5. Esquema de una neurona biológica 21

3.6. Topología típica de una red neuronal 22

3.7. Funcionamiento una neurona artificial [J.M. Gutierrez, Introducción a las Redes Neuronales, Universidad de Cantabria] 23

3.8. Función de paso 24

3.9. Función sigmoidal 25

3.10. Función de la tangente hiperbólica 25

3.11. Función softplus(verde) frente a la ReLU (azul) 26

3.12. Topología de una red feed-forward densa 27

3.13. Ejemplo de capa de neuronas para aprendizaje competitivo 29

3.14. Red convolución para el tratamiento de una matriz de datos (ej. una imagen) 30

3.15. Capa recurrente desenrollada para una secuencia con relación temporal . . 31

3.16. Detalle de la composición de una célula LSTM 32

3.17. Estructura de un modelo GMM-HMM 33

3.18. Estructura de un modelo DNN-HMM 35

4.1. Pantalla de login tras las modificaciones 38

4.2. Reconocimiento facial en curso 40

4.3. Prototipo del reconocimiento de voz y obtención de la intención asociada a la frase reconocida 41

4.4. Aplicación para el desarrollo de modelos de lenguaje de LUIS 41

4.5. Panel de pruebas de reconocimiento de voz. El botón del lateral izquierdo con el micrófono es lo único visible en la versión final. 42

4.6. Espectrograma generado por un hombre diciendo çomando" 44

5.1. Resultados obtenidos al generar un modelo con las muestras y parámetros actuales 48

Índice de tablas

2.1. Tabla de dedicaciones (en horas). 14

3.1. Región principal de los formantes vocálicos del español 18

A.1. Conjunto de valores obtenido mediante sesiones de entrenamiento con los parámetros especificados 62

1. CAPÍTULO

Introducción

1.1. Contexto

Se dispone de una herramienta web de gestión de maquinaria industrial que, de cara a hacer más intuitivo y ágil el mantenimiento de las diferentes máquinas en las que trabaja un operario, permite mediante un modelo 3D la visualización y modificación de los diferentes componentes y piezas de la máquina. Se da la situación de que el operario se encontrara muchas veces con la imposibilidad de utilizar las manos para manipular la herramienta debido a las tareas que esté realizando en ese momento.

Para facilitar estas labores se plantea:

- Que el dispositivo reconozca al operario, sin necesidad de que este tenga que introducir ninguna información, mediante reconocimiento facial.

- Que la herramienta responda a las órdenes habladas del operario mediante reconocimiento del lenguaje para realizar tareas básicas sin que este tenga la necesidad de interrumpir lo que esté haciendo con las manos.

1.2. Estado del arte

1.2.1. Reconocimiento Facial

Como se describe detalladamente en [1], el problema del reconocimiento facial ha suscitado mucho interés desde los comienzos de las diferentes tecnologías de Visión por Computador (CV) a finales de los años 60 [2]. Estas primeras técnicas se basaban sobre todo en la distancia entre diferentes puntos distintivos en las caras (las zonas alrededor de los ojos, la boca, etc). Si bien a finales de los 90 empezaron a verse estudios importantes sobre el tema con la utilización de Eigen-faces[3] o Fisher-faces [4] [5] y otra serie de técnicas más avanzadas que permitían un reconocimiento facial totalmente automatizado, no fue hasta las últimas décadas cuando se empezaron a desarrollar diferentes protocolos estandarizados de pruebas (FERET en el 98 o el XMV2TS en el 99) y el posterior auge de diferentes softwares que actualmente se comercializan.

Una de las vías de investigación más prometedoras actualmente es la del reconocimiento de modelos faciales en 3D generados a partir de imágenes 2D. Actualmente la mayoría de técnicas de reconocimiento facial desarrolladas y aceptadas rondan el 90-99 % de precisión [6] en los bancos de pruebas habituales.

Aplicaciones comunes de esta tecnología se pueden encontrar hoy en día en los diferentes sistemas de vigilancia y seguridad, redes sociales e incluso aplicaciones de Realidad Aumentada.

1.2.2. Reconocimiento de Voz

También conocido como Automatic Speech Recognition (ASR) en la literatura inglesa y siendo también utilizados Computer Speech Recognition (CSR) o Speech to Text (STT, si bien esto se refiere a una aplicación común pero más concreta), este problema multidisciplinar fue planteado a partir de mediados de siglo, pero limitado sobre todo por la capacidad de los computadores de la época, no empezó a desarrollarse hasta las 3 últimas décadas. Como se puede leer en [7] la primera aproximación a la resolución de este problema se da al plantearla como un problema de clasificación estadística dado un diccionario cerrado de palabras [8]. Es entre los 80-90 cuando empiezan a surgir dos conceptos interesantes como son el modelo acústico y el modelo de lenguaje ([9] y [10], por ejemplo entre otros muchos). En la siguiente década también fue popular el uso de

Modelos Ocultos de Markov (HMM) para plantear diversas soluciones[10] [11] [12], modelo que tomaría especial relevancia por sus resultados y se mantendría como base para diversas variantes hasta principios del siglo siguiente. El uso de de los HMM para ASR se distinguía principalmente por ser dependientes o independientes del contexto. A raíz de lo comentado anteriormente y para paliar alguna limitación inherente a los HMM comenzó a extenderse el uso de Redes Neuronales (ARN), inicialmente en modelos mixtos Neto95[13] y posteriormente en modelos que hicieran uso exclusivo de estas. Una revisión de referencia muy citada en este aspecto es [14] publicada en el MIT en el 89. A partir de este momento y debido a la fuerte apuesta por el Deep Learning para la resolución de esta clase de problemas, surgen una enorme variedad de métodos diferentes. Uno de los mas conocidos y utilizado para trabajar con datos que impliquen series temporales hoy en dia es el CTC [15]. Este algoritmo ha obtenido cierta popularidad en el reconocimiento de patrones (no sólo para ASR, sino también para reconocimiento de escritura).

Como es evidente, las aplicaciones de esta tecnología, en conjunto con el procesamiento del lenguaje natural, abarcan todos los múltiples asistentes implementados para móviles y ordenadores los últimos años, así como otro tipo de soluciones de seguridad o el subtitulado en tiempo real, por ejemplo.

1.3. Alternativas comerciales y Propuesta

Actualmente, existen múltiples soluciones comerciales ofertadas para resolver los diferentes problemas mencionados anteriormente. El modelo habitual de consumo de estos servicios suele ser el de Cloud Computing (CC), esto es, el uso de las herramientas que el proveedor proporciona mediante la llamada a diferentes direcciones de Internet. Esto se debe a que generalmente, el uso a gran escala de este tipo de herramientas suele conllevar una capacidad de cómputo considerable. Delegar la adquisición y mantenimiento del hardware para realizar todo esto supone normalmente una reducción drástica de los costes materiales y de implantación. Además, generalmente los proveedores ya ofertan APIs propias como solución a diferentes necesidades que pueda necesitar el cliente (gestión, monitorización, cómputo, ML e IA . . .). En este caso concreto se pueden encontrar tres grandes proveedores de servicios de CC:

- Amazon, propietario de Amazon Web Services (AWS)[1]. Para el tipo de tareas a rea-

[1] https://aws.amazon.com/es/

lizar en concreto, dispone de los servicios Rekognition (imagen), Transcribe (STT) y Comprehend.

- Google, propietario de Google Cloud[2]. Este dispone de Cloud Visión, Speech Cloud y Cloud Natural Language para estas tareas.
- Microsoft, propietario de Azure, tiene Face, Speech y LUIS (Language Understanding) para proveer estos servicios en su catalogo de Cognitive Services[3].

Teniendo en cuenta un análisis previo a mi incorporación realizado por la empresa y otro análisis posterior sobre el potencial de las diferentes alternativas y su aplicación para cumplir los objetivos comentados, se decidió que se utilizarían los servicios que provee Azure.

[2] https://cloud.google.com/products/
[3] https://azure.microsoft.com/es-es/services/cognitive-services/

2. CAPÍTULO

Gestión del Proyecto

2.1. Alcance del Proyecto

El alcance global del proyecto podría clasificarse en dos secciones diferenciadas: el Alcance Empresarial y el Alcance Académico.

2.1.1. Alcance Empresarial

El Alcance Empresarial representa los compromisos adquiridos para con la empresa durante la realización del proyecto y los objetivos a cumplir al finalizar este.

Objetivos:

- Análisis de las diferentes alternativas de servicios Cloud para ofrecer servicios de Reconocimiento Facial y Reconocimiento de Voz.
- Evaluar el rendimiento y las funcionalidades que ofrece cada uno de estos de cara a adaptarlo a la herramienta.
- Prototipado de herramientas que usen estos servicios.
- Integración de un prototipo funcional dentro de la herramienta existente.

- Traspaso de conocimientos adquiridos de estas herramientas para su posterior mejora y mantenimiento.

Objetivos Adicionales:

- Realizar pruebas con los prototipos desarrollados para diferentes localizaciones (Español, Inglés, Alemán. . .)
- Implementación de mejoras en la herramienta (Funcionalidades gráficas, modificaciones en la interfaz...)

Exclusiones

En principio no se ha definido ninguna exclusión.

2.1.2. Alcance Académico

El Alcance Académico engloba todos los objetivos a cumplir de cara a la presentación de un proyecto satisfactorio y los objetivos de aprendizaje durante la realización del mismo:

Objetivos:

- Aprendizaje de diferentes tecnologías y campos, a saber:
  - Redes Neuronales
  - Tratamiento de señales de audio
  - Procesamiento del lenguaje natural
- Creación de un complemento para mejorar la herramienta que implique el desarrollo de un algoritmo propio
- Redacción de una memoria del proyecto que sintetice y detalle los pormenores de este
- Preparación adecuada de la defensa del proyecto

Objetivos Adicionales:

- Ampliar la implementación propia para abarcar otras posibilidades (reconocimiento de diferentes palabras)
- Desarrollo de modelos acústicos (filtrado de ruido y mejora de la señal de entrada)

2.1.3. Exclusiones

No se ha definido ninguna exclusión inicialmente

2.2. Adquisición de Conocimientos

2.2.1. Aprendizaje teórico de diferentes fundamentos:

- Redes Neuronales: Diferentes artículos , tesis y tutoriales
- Tratamiento de señal de Audio: Buscar información sobre los rudimentos de la señal de Audio:
  - Espectro de frecuencias y Variación de las frecuencias en la voz humana
  - Características de los Coeficientes Cepstrales de Mel (MFCC)

2.2.2. Lenguajes de programación:

- .NET: Se tenían conocimientos rudimentarios previos de este lenguaje.
- JavaScript: El grueso de la implementación realizada es en este lenguaje.
  - Three.js
- Python: Aprendizaje de los rudimentos del lenguaje.
  - scipy

2.2.3. APIs y Librerías

- Azure
  - Face
  - Speech
  - LUIS
- TensorFlow
- Keras

2.3. Herramientas utilizadas

Como se indicaba en la sección anterior, ha sido necesario familiarizarse con el uso de diferentes herramientas y lenguajes de programación para poder llevar a buen puerto la implementación del proyecto. A continuación procederé a comentar algunas de ellas.

2.3.1. Lenguajes de programación y librerías

En cuanto a los lenguajes de programación,el núcleo la herramienta a mejorar estaba implementado en .NET (siguiendo el modelo MVC) con una fuerte carga de ficheros JS para realizar las funciones asociadas a las diferentes operaciones disponibles de la interfaz. También ha sido necesario familiarizarse con el consumo de servicios web de terceros para gestionar las llamadas al proveedor Azure. Adicionalmente y para la implementación de algunas funciones en concreto (cambios de visibilidad, selección, operaciones de rotación, etc...) también ha sido necesario la comprensión del funcionamiento de la librería de gráficos Three.js.

Además, se barajaron diferentes lenguajes de programación con potentes librerías matemáticas (R, Matlab...) para la implementación de las funciones de tratamiento de señal y redes neuronales. Siendo multiparadigma y con un enfoque más tradicionalmente programático, se decidió utilizar Python para estas tareas, con especial mención al uso de la librería scipy.

También se busco una librería con funciones de más alto nivel para la programación de redes neuronales, siendo TensorFlow y Keras las mejores opciones al estar ambas programadas en Python. Al tener un rendimiento similar para la resolución de estos problemas pero estar orientada a simplificar los primeros contactos con este tipo de modelos, se opto por la utilización de esta última.

2.3.2. Herramientas de Terceros

Azure

Azure es una plataforma perteneciente a Microsoft que ofrece múltiples servicios web alojados físicamente en centros de cómputo comúnmente conocidos como DataCenters. Dentro de este modelo de negocio, conocido como Cloud Computing, dispone de una variedad de servicios de almacenamiento, cómputo, desarrollo, herramientas propias...

Para este proyecto se ha hecho uso de las APIs de reconocimiento facial, de voz y procesamiento del lenguaje natural (Face, Speech y LUIS respectivamente) de las que dispone la plataforma. A continuación se hará una descripción detallada del funcionamiento a grandes rasgos y las herramientas y recursos que ofrecen cada una de ellas:

Face

Face es una API proporcionada por Azure para llevar a cabo diferentes tareas de reconocimiento facial. La herramienta actualmente es capaz de obtener hasta 64 caras por imagen, delimitadas por una suerte de BoundingBox, obteniendo además por cada una de ellas si se desea diferentes atributos que proporcionan información acerca del sexo, la edad, postura, vello facial, estado de ánimo...

Además ofrece diferentes funciones de tratamiento de estas, entre las que se incluyen:

- Comprobación de caras: Dada una pareja de caras, intenta deducir si pertenecen a la misma persona o no. El objetivo de esta función es ser usada para operaciones de verificación de identidad. Esta función es la base de todas las demás.
- Búsqueda de caras similares: Dada una cara, teniendo ya un conjunto de caras almacenadas (más adelante se precisará esto) e indicando un umbral, devuelve un subconjunto de la muestra inicial que contiene las caras cuya similitud a la original sea mayor que el umbral establecido.

- Agrupación de caras: Dado un conjunto de caras, realiza una clasificación por similitud de estas en diferentes grupos. Estos conjuntos se almacenan en una suerte de objeto sobre el que se pueden realizar varias operaciones (llamado Face Group).
- Identificación de caras: Dada una cara y teniendo de antemano una base de datos de caras, organizadas en diferentes grupos asociados cada uno a una persona, intenta identificar la cara entrante como una de las personas almacenadas en la base de datos. Esta función es una versión más avanzada de la verificación básica, que además permite gestionar el resultado de manera mucho más eficaz.
- Almacenamiento de caras: A raíz de las necesidades de las funciones comentadas anteriormente, la plataforma dispone de un sistema de almacenamiento y gestión de las diferentes caras que se utilizan. Provee tanto de objetos como funciones propias para facilitar estas tareas.

Speech

Para realizar diversas tareas relacionadas con el uso de la voz Azure proporciona la API de Speech, la cual dispone de diferentes herramientas para facilitar el desarrollo de aplicaciones con estas funciones. Actualmente, esta API aún está en desarrollo, por lo que está sujeta a cambios y algunos complementos no están disponibles. Actualmente puede realizar las siguientes operaciones:

- Voz a texto (STT): Al recibir una entrada de voz, convierte esta a texto con un margen de error mínimo en ambientes sin demasiado ruido o interferencias. Esto puede realizarse en diferentes idiomas, para cada uno de los cuales ya disponen de un modelo de lenguaje general entrenado para realizar el contraste e identificación de las palabras recibidas. A pesar de estar todavía en desarrollo, también dan la posibilidad de crear un modelo de lenguaje personalizado (pensado para procesar palabras concretas de contextos específicos como la medicina o las TI, por ejemplo) y un modelo acústico personalizado (pensado para adaptar el funcionamiento a espacios que planteen un entorno acústico complicado, como una fábrica o un estadio). El resultado obtenido por esta herramienta se puede integrar también con LUIS para extraer las intenciones del hablante o procesar el texto.
- Síntesis de Voz: Dado un texto, crea un audio de una voz similar a la humana que permite variaciones de sexo, idioma e incluso algunos acentos y dialectos.

- Traducción simultánea de Voz: permite la traducción en tiempo casi real de una entrada de voz, pudiendo devolver la salida como formato de audio o texto.

LUIS

Language Understanding (LUIS) es una IA de aprendizaje automático conversacional capaz de extraer a partir de las diferentes frases del usuario tanto el significado global como diferente información detallada. Esto puede integrarse en múltiples aplicaciones, como chatbots, reconocimiento oral y escrito o análisis automático en redes sociales o páginas de opinión.

La herramienta LUIS se basa en el uso de modelos de lenguaje, que pueden ser los predefinidos por el sistema o modelos personalizados definidos por el usuario. Estos se especifican mediante el uso combinado de intents (intenciones) y entities (entidades) y puede mejorar su funcionamiento y precisión mediante diferentes métodos de aprendizaje activo como la definición de listas de frases comunes, patrones y la revisión de las frases recibidas en el punto de acceso de la aplicación.

Todo esto puede realizarse mediante importación de texto plano correctamente formateado o usando la herramienta web que provee para creación y testeo del modelo.

2.3.3. Keras

Keras es una API de alto nivel que corre por encima de las tres principales librerías de redes neuronales existentes (TensorFlow, CNTK y Theano) y esta pensada para el aprendizaje y prototipado rápido. Permite la creación de redes convolucionales, recurrentes y combinaciones de estas, además de ser compatible para su ejecución tanto en la CPU como la GPU.

Otras ventajas de las que dispone es el hecho de ser una librería completamente integrada en Python, facilitando la inclusión de estas herramientas en cualquier otra función del código, y el planteamiento de cada una de las capas que pueden formar una red como módulos separados que puedan acoplarse fácilmente y ser extendidos con diferentes parámetros a necesidad.

2.3.4. Control de versiones y Copias de Seguridad

Como herramienta principal de control de versiones y copias de seguridad se ha utilizado la herramienta estandarizada en el lugar de trabajo, Microsoft Team Foundations, que además tiene la ventaja de esta integrada en VisualStudio, IDE usado para el desarrollo del proyecto. Adicionalmente, el código Python generado por mi se ha subido también a GitHub y se han almacenado copias tanto en el equipo corporativo como en el personal (en dos discos duros diferentes).

2.3.5. Gestión del tiempo

Para la gestión de los plazos del proyecto y el seguimiento de las tareas realizadas, se ha recurrido a la herramienta Tom's Planner. Esta permite la gestión de los hilos de las diferentes tareas de cada apartado y el seguimiento de su estado de manera visual e intuitiva.

2.4. Planificación

La planificación de el proyecto ha sido dinámica(si bien se ha mantenido bastante ajustada a lo establecido inicialmente), al ir adaptándose está a nuevos objetivos que se han ido presentando.

Tal y como se ha mencionado anteriormente, el seguimiento de la planificación se ha realizado con la herramienta Tom's Planner. Se puede consultar **aquí**.

2.5. Análisis de Riesgos

Como en todo proyecto existen riesgos posibles que pueden dificultar la realización del mismo si no se previenen. Si bien no hay un factor de riesgo demasiado elevado, están contemplados los siguientes riesgos:

- Vacaciones: Se contempla el tomar libres 2 días a finales de Julio, en principio el resto del periodo del Verano se computará como días laborables. Esto hace que el riesgo previsto sea mínimo.

- Enfermedad: Teniendo antecedentes previos, las bajas por enfermedad pueden suponer un riesgo de menor a moderado en función de la gravedad de la misma y el periodo de recuperación necesario. El riesgo asignado a este imprevisto es inicialmente bajo.

- Fallo tecnológico crítico: Puede darse el caso de problemas en el equipo en el que se desarrolle el proyecto o alguno de sus sistemas de almacenamiento. En principio y tomando las precauciones necesarias (comentadas en la sección de Copias de Seguridad) para paliarlo, no debería suponer un mayor inconveniente.

- Cambió de versión de las herramientas usadas: Se menciona este riesgo porque ya se ha detectado una vez durante el desarrollo del proyecto. Dado que tanto la versión previa como la nueva están documentadas con mayor o menor detalle por el fabricante, no debería haber mayor problema en adaptar lo implementado, pero puede suponer un funcionamiento incorrecto hasta que se detecta. Se asigna a esta circunstancia un factor de riesgo moderado

2.6. Análisis de Factibilidad

Teniendo en cuenta los diferentes factores detallados anteriormente y asumiendo los riesgos mencionados como algo que se puede afrontar en el tiempo y con los recursos disponibles, todo indica que en principio este proyecto es viable. Se dispone además de un amplio margen para ampliar el plazo en caso de que surgieran dificultades de cualquier tipo.

2.7. Dedicación

La tabla 2.1 refleja la dedicación aproximada(en horas) a cada una de las diferentes tareas principales detalladas anteriormente. Esto se corresponde en gran medida con la planificación realizada, sin suponer una desviación relevante de la estimación inicial:

Tarea	**Estimación**	**Dedicación**
Adquisición de Conocimientos	100	125
Implementación	75	60
Experimentación y Pruebas	30	40
Documentación	75	60
Reuniones	20	10
Total	300	295

Tabla 2.1: Tabla de dedicaciones (en horas).

3. CAPÍTULO

Conceptos y Tecnologías

3.1. Tratamiento de señales de sonido

3.1.1. Particularidades de la voz humana

A nivel anatómico, la voz humana esta formada por el conjunto de sonidos que puede emitir el aparato fonador humano, ilustrado en la Figura 3.1. Este abarca, a grandes rasgos, los órganos de respiración (cuyo componente principal son los pulmones) los órganos de fonación (principalmente la laringe) y las diferentes cavidades superiores que junto a la lengua y otros elementos de la boca componen los órganos de articulación.

Dentro de la laringe, se encuentran las cuerdas vocales, cuya abertura se denomina glotis. Cuando las cuerdas vocales se encuentran separadas, el aire pasa libremente y no produce sonido.Cuando esta se cierra en diferentes medidas, la glotis produce diferentes sonidos en función de factores como el tamaño de esta, la tensión aplicada y la velocidad por la que el aire circula por esta (siendo la frecuencia de vibración de las membranas proporcional a esta). Debido a esto, por ejemplo, al ser normalmente la laringe masculina de mayor tamaño que la femenina, los hombres tienden a tener la voz más grave. Así mismo, la emisión de sonidos más agudos requiere de mayor esfuerzo por parte del hablante[16].

El sonido generado por este aparato es un sonido complejo, es decir, esta compuesto por varias ondas simultaneas. La combinación de estas genera lo que se conoce como *timbre*. El timbre, a su vez, es la combinación de otros componentes: el espectro, la envolvente

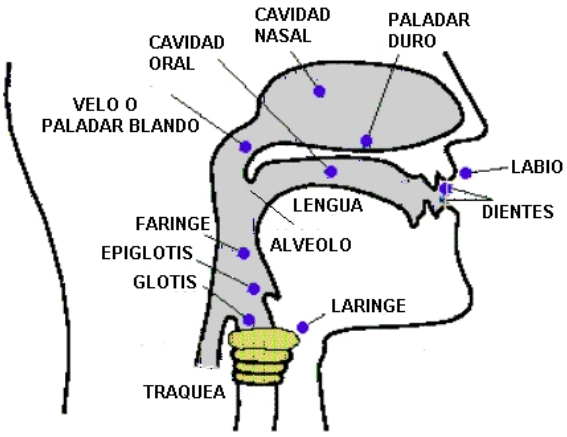


Figura 3.1: Aparato fonador humano

dinámica (o variación en el tiempo) y los formantes. Se hablara de todo esto en detalle en el apartado siguiente.

En definitiva, las posibles combinaciones de todos estos factores es lo que, en términos acústicos, aporta la riqueza y diferenciación que encontramos en las distintas voces humanas.

3.1.2. La voz humana como señal auditiva

Como se ha introducido en el apartado anterior, el sonido que produce la voz humana puede ser modulado e interpretado como cualquier otra señal acústica. Hay una cantidad considerable de conceptos asociados a la medición de esta señal[17], por lo que vamos a centrarnos sobre todo en aquellos que utilizaremos para el siguiente trabajo.

Dos conceptos fundamentales presentes en toda señal de audio son la energía y la frecuencia, que van tomando diferentes valores en cada momento concreto de la señal.

La energía (medida en dB) puede definirse como la intensidad de la señal durante lapso de tiempo, obtenida a partir del sumatorio al cuadrado de la amplitud de todas las frecuencias presentes en ese lapso.

Para una señal continua se definiría más formalmente como:

$$E = \int_{t_1}^{t_2} |x(t)|^2 dt$$

En cambio, para una muestra discreta (para el lapso mencionado anteriormente):

$$E = \sum_{n=m}^{N-1} x(m)^2$$

Siendo N el numero de muestras de la señal.

La frecuencia en cambio, se mide en Hz y representa la rapidez de la oscilación de la onda por unidad de tiempo (normalmente, segundos). Indica la cantidad de veces que un ciclo de la señal se repite en un segundo.

La relación entre ambas se establece mediante el espectro de sonido, representado por un espectrograma como el mostrado en la Figura 3.2, que muestra la distribución de la energía en cada frecuencia a lo largo de toda la función.

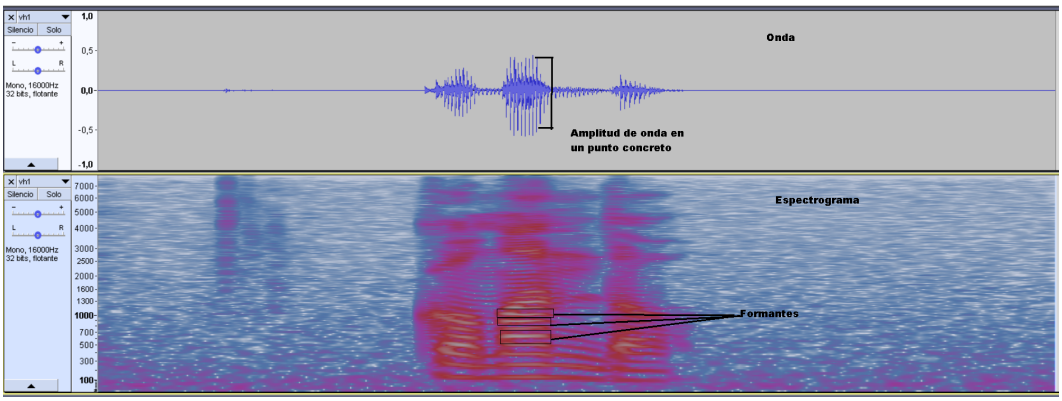


Figura 3.2: Espectrograma en escala de Mel de la palabra *comando*. A mayor intensidad en una frecuencia, el color se vuelve más vivo.

Otra característica interesante a evaluar en la señal de voz humana son los formantes[1]. Los formantes son los picos que se generan en varias frecuencias en un momento concreto. Analogamente a lo que en la musica seria un acorde formado por varias notas (ej. C Mayor: do, mi,sol) podríamos decir que cada uno de los fonemas que la voz humana puede emitir se configura con la combinación de los diferentes formantes. Se denominan en orden ascendente de menor a mayor frecuencia, siendo F_1 el de menor frecuencia, F_2 el siguiente y así sucesivamente.

Anatómicamente, estos formantes están asociados a los órganos articuladores y las cavidades superiores, siendo F1 la apertura de la boca, F2 la forma de la lengua, F3 la posición de esta... No hay un numero máximo de formantes establecido, pero en principio suele bastar con el uso de los dos primeros para categorizar cualquier vocal en la mayor parte de los idiomas(en la Figura 3.3 podemos observar esta representación en Español),

[1] https://es.wikipedia.org/wiki/Formante

pero si el idioma tiene muchas vocales podrían llegar a usarse hasta seis. En la musica suele denominarse al cuarto y quinto formante "formantes del canto"[18].

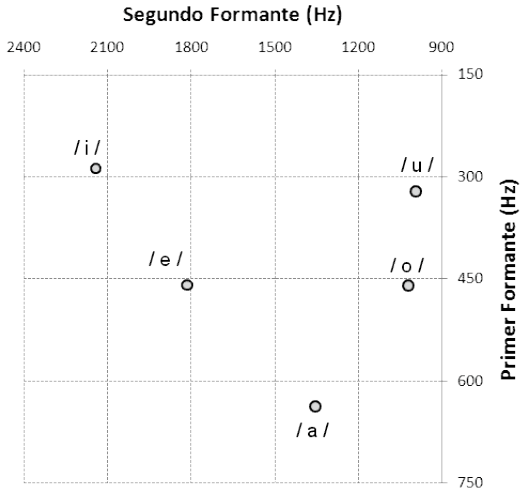


Figura 3.3: Vocales del español clasificadas según sus dos primeros formantes [Bradlow 1995]

Estos formantes pueden variar según el timbre de cada persona, además de otros factores a mayor escala como el sexo del hablante. Por tener una referencia, en la Tabla 3.1 podemos encontrar los valores aproximados que tienen las vocales en español[19]:

Vocal	Rango(Hz)
/u/	200-400
/o/	400-600
/a/	800-1200
/e/	400-600 y 2200-2600
/i/	200-400 y 3000-3500

Tabla 3.1: Región principal de los formantes vocálicos del español

Si comparamos estos valores con los presentados en la Figura 3.2 podemos apreciar una de manera visible una relación entre estos valores y los análisis realizados sobre las señales utilizadas.

3.1.3. Coeficientes Cepstrales de las Frecuencias de Mel

Por norma general, independientemente del método que se utilice para generar el modelo de reconocimiento de voz (ya sea mediante HMMs, ARNs...), las señales de audio que

se utilizan como entrada del modelo requieren un procesado previo. Este proceso se suele realizar subdividiendo cada muestra en *ventanas*, usadas para acotar cada uno de los fragmentos en los que se divide la señal, garantizar la continuidad de estos y resaltar la información más relevante contenida en ese espacio de tiempo mediante lo que se conoce como una función de enventanado (siendo las más conocidas Hann, Hamming y las variantes de Blackmann). Una forma simple de la cual podría introducirse cada audio en el sistema de clasificacion implementado seria interpretando la señal como un vector que represente el numero de muestras (a menor tamaño de ventana[2], más muestras), cada una de las cuales seria otro vector que contendría las frecuencias presentes en ese lapso y los valores de esta. Esto se podría apreciar de forma gráfica muy cómodamente mediante un espectrograma, como se ha comentado anteriormente.

Se puede intuir que esto viene a ser una discretización de una señal continua, pero ateniéndonos al teorema de muestreo de Nyquist-Shannon[3] sabemos que siempre que obtengamos al menos el doble de muestras que el ancho de banda que queramos obtener, la reconstrucción de la señal a raíz de los valores discretizados debería ser exacta. Como dato, tal y como se mencionaba en el punto anterior, sabiendo que no son necesarios todos los formantes para obtener información relevante de una muestra de voz y conociendo el intervalo de frecuencias en el que suelen presentarse, se suele considerar suficiente con un muestreo a 16kHz que proporcione muestras a 8kHz para interpretar la voz humana.

Sin embargo, introducir la señal de este modo puede no ser una opción muy viable, ya que la muestra sin ningún tipo de filtro es muy susceptible a otros factores como el ruido ambiente o distorsión sujeta al método de grabación. Para paliar estas circunstancias, una de las herramientas más utilizadas son los Coeficientes Cepstrales de las Frecuencias de Mel (más conocidos comúnmente como MFCC por su siglas en ingles)[4]. Estos toman como referencia la escala de Mel, la cual es una escala no lineal cuyas bandas de frecuencia están situadas logísticamente y cuyo punto de referencia esta situado en los 1000 Hz, equivalentes a 1000 mels en esta escala. Esta manera de espaciar el sonido se aproxima más que la lineal a la manera en la que un humano percibe el sonido.

A raíz de esto en los 80 se definió lo que se conocerían como los MFCC[20], un conjunto de coeficientes para la representación del habla basados en la percepción auditiva humana. Estos suponen otra forma de representar la señal inicial de modo que amplifique los componentes relevantes para el reconocimiento de voz mientras que intenta minimizar el

[2] https://es.wikipedia.org/wiki/Ventana_(función)
[3] https://es.wikipedia.org/wiki/Teorema_de_muestreo_de_Nyquist-Shannon
[4] https://es.wikipedia.org/wiki/MFCC

impacto de otros como el ruido de fondo, volumen etc. El proceso para la obtención de estos se puede ver representado en la Figura 3.4.

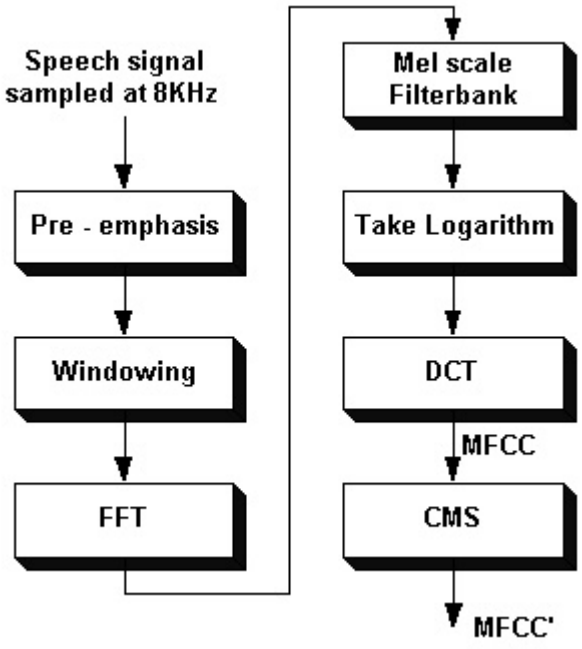


Figura 3.4: Proceso de obtención de los MFCC

El output obtenido tras realizar este proceso es un conjunto de cepstrums (típicamente unos 12, aunque pueden obtenerse más en función de para que se necesitan) que como se ha comentado anteriormente, representan una entrada mucho más refinada para introducir en los modelos de reconocimiento de voz desarrollados.

3.2. Redes Neuronales

Se ha hablado bastante de las redes neuronales y se seguirá hablando de ellas a lo largo de este documento, por lo que conviene hacer un alto para intentar explicar qué son, que características tienen, como funcionan y que tipos de redes neuronales hay.

3.2.1. ¿Que es una Neurona?(Antecedentes biológicos)

Si bien, por motivos que expondremos más adelante, hay una diferencia significativa entre el modelo biológico y el modelo artificial de neuronas y redes neuronales, no se puede negar que hubo una inspiración en la forma de operar de éstas a la hora de idear el modelo.

Una neurona biológica esta formada por diversos componentes que garantizan su sustento y supervivencia (núcleo, citoplasma...) y otros que utiliza para comunicarse con otras neuronas: los axones, formados por varias dendritas, que a su vez se componen de diferentes sinapsis.Todo esto puede apreciarse en la Figura 3.5. Existen diferentes tipos de sinapsis: actuadoras e inhibidoras. Cuando una neurona recibe una cierta cantidad de estímulos que sobrepasan un umbral de activación la neurona se dispara, propagando la señal al resto de neuronas a las que esta conectadas. Se estima que el cerebro humano se compone de unas 10^{11} neuronas con unas 10^4 conexiones cada una. El proceso de aprendizaje de estas neuronas implica que, a lo largo del tiempo, cuantas más señales recibe un axón más se fortalece. Esto se conoce como la teoría del aprendizaje Hebbiano [21].

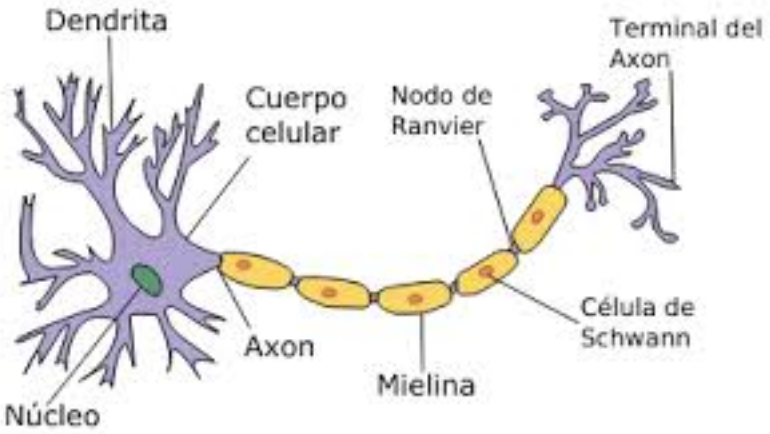


Figura 3.5: Esquema de una neurona biológica

Hablemos ahora de las redes neuronales artificiales. Estas están formadas por unidades o nodos (también conocidas como células o neuronas) que se conectan a otras a través de arcos dirigidos (modelando la conexión axón/dendrita). A su vez cada uno de estos arcos representa la salida de la unidad que sirve de entrada para otras unidades de la red. Cada uno de estos arcos tiene asociado un peso que indica la fuerza y el signo de la conexión (simulando la sinapsis). El comportamiento del conjunto de la red estaría por lo tanto determinado por la topología de la red, los pesos establecidos y la función de activación de las neuronas (análoga al umbral de activación biológico).

Y en gran medida, esta es toda la similitud que tienen ambos modelos. Hay que tener en cuenta que se trata de un modelo formal que puede ajustarse en mayor o menor medida al modelo biológico... o desviarse radicalmente, como ya se verá más adelante. En cualquier caso, ambos modelos comparten ciertas características[22]:

- El conocimiento se adquiere a través de un proceso de aprendizaje
- Las conexiones interneuronales ponderadas mediante pesos son las que definen la forma de procesar la información.

3.2.2. Estructura y funcionamiento de las redes neuronales

A partir de lo anteriormente definido, las redes neuronales suelen tener una estructura típica que consiste en agrupar diferentes capas de neuronas. Las neuronas que conforman cada capa realizan todos los cálculos en paralelo antes de pasar la salida a la siguiente capa. Tradicionalmente la red se compone de una capa de entrada, una capa de salida y un numero indeterminado de capas ocultas, en una estructura similar a la que puede observarse en la Figura 3.6. Cada capa contiene células del mismo tipo, pero no todas las capas tienen porqué ser iguales en número de células ni tipo de éstas.

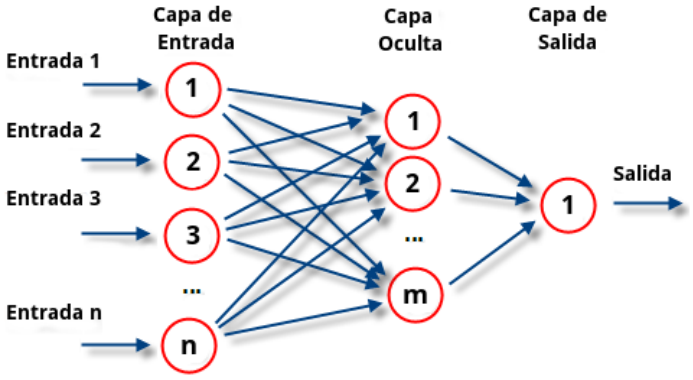


Figura 3.6: Topología típica de una red neuronal

Adicionalmente, el aprendizaje o proceso mediante el cual la red obtiene nueva información también puede ser de diferentes tipos: supervisado, no supervisado o por refuerzo. Otro dato a tener en cuenta es que la función de activación definida para cada conjunto de neuronas puede variar también, ajustándose a alguna de uso generalista (como la sigmoidal, tanh, relu...) o diseñada específicamente para el uso de una capa. Como puede observarse, son muchos los factores que determinan el funcionamiento de la red, y las posibles combinaciones de ellos hace que sea inabarcable la exposición de todas las alternativas. Por eso en los puntos siguientes se procederá a explicar algunos de los tipos más comunes en función de su topología o tipo de neuronas. Por ahora nos centraremos en detallar un poco más como afecta cada uno de los factores mencionados.

Neurona

Una neurona esta formada por tres elementos: el conjunto de nodos de entrada, la función de activación de la neurona y el conjunto de nodos de salida (A las que siguiendo la nomenclatura de la figura 3.7 denominaremos (X,f,Y) respectivamente). La función de activación f genera el valor de salida mediante los pesos w que le llegan de X y lo propaga hacia los nodos de Y. Podemos ver una representación gráfica de esto como una función de combinación lineal en la figura 3.7.

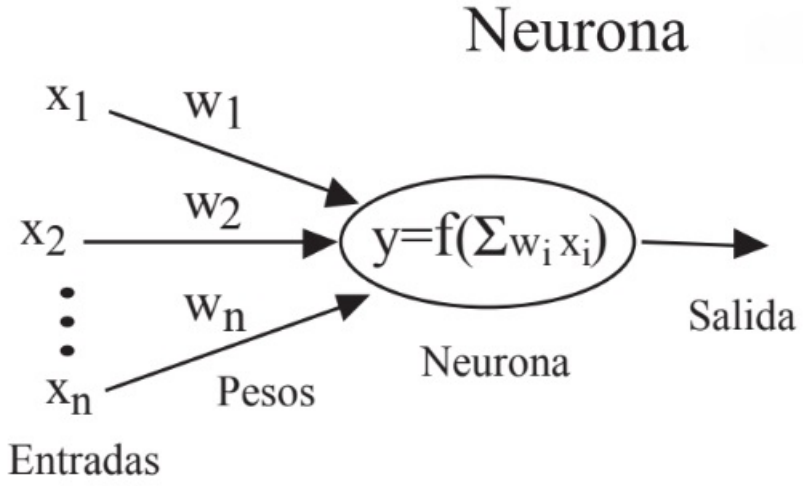


Figura 3.7: Funcionamiento una neurona artificial [J.M. Gutierrez, Introducción a las Redes Neuronales, Universidad de Cantabria]

Aprendizaje

Se define como aprendizaje supervisado aquel en el que se suministra a la red una muestra formada por el par (X,Y), donde X es el conjunto de los valores que toma la muestra e Y el conjunto de valores a inferir a partir de la muestra proporcionada. Esto podría expresarse como:

$$\triangle w_{ij} = \alpha x_i [y_j - \hat{y}_j]$$

Es decir, se ajusta el valor del peso w entre las dos unidades en proporción a la diferencia entre los valores deseado y calculado en cada una de las unidades de la capa de salida, siendo α la constante definida como tasa de aprendizaje.

Por otro lado en el aprendizaje no supervisado en cambio, al no tener un *objetivo*, simplemente la red aprenderá a adaptarse a los valores anteriores para generar nuevos valores. Es decir w_{ij} es la correlación entre el valor de las unidades x_i y x_j:

$$\triangle w_{ij} = x_i x_j$$

Tras la realización de todo el proceso de aprendizaje, suele hacerse uso de métodos como el Sum Square Error(SSE), Root Mean Square Error(RMSE) o el máximo error para calcular la calidad del modelo.

Funciones de activación

La función de activación es lo que determina el valor que propaga la neurona como input a la siguiente capa. Si bien la función puede ser una definida para la ocasión, lo habitual es encontrarse con alguna de estas:

- **Lineal:** $f(x) = x$
- **Funciones de paso:**Dan un valor de salida binaria en función de si la entrada supera un umbral dado o no

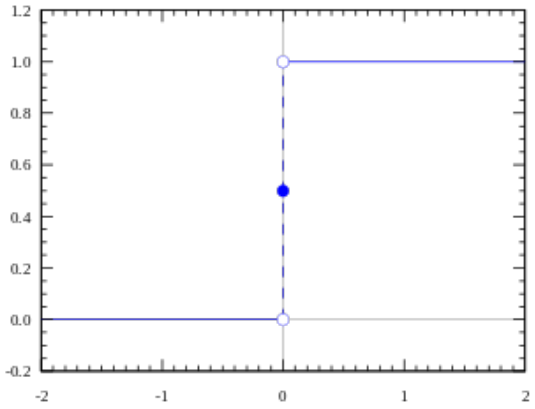


Figura 3.8: Función de paso

- **Funciones sigmoidales:**Funciones monótonas acotadas que dan una salida gradual no lineal

- Función logística de 0 a 1:

$$f_c(x) = \frac{1}{1 + e^{(}-cx)}$$

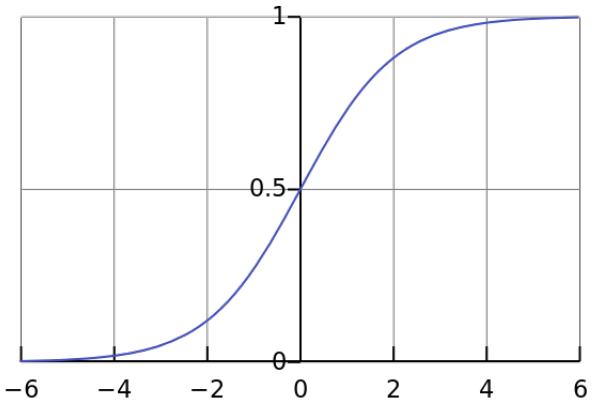


Figura 3.9: Función sigmoidal

- Función tangente hiperbólica de -1 a 1

$$f_c(x) = tanh(cx) = \frac{2}{1 + e^{(}-2x)}$$

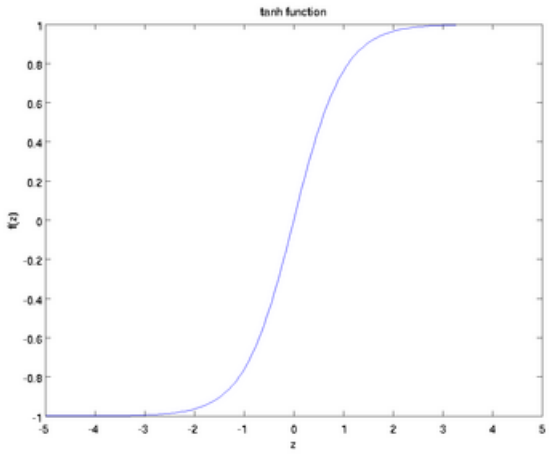


Figura 3.10: Función de la tangente hiperbólica

- **Rectificadores:**
  - ReLU:

$$f(x) = max(0, x)$$

  - Softplus: Aproximación suave no linear de la ReLU

$$f(x) = ln(1 + e^x)$$

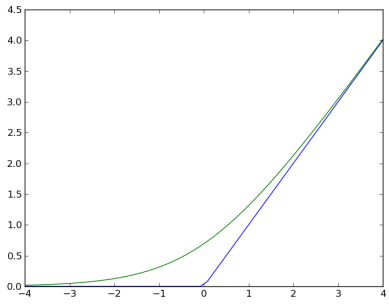


Figura 3.11: Función softplus(verde) frente a la ReLU (azul)

Debido a que muchas veces las redes a usarse van a ser multicapa o usen *backpropagation*, el uso de las funciones no-lineales suele estar más extendido(ya que si son lineales, al acumular capas el resultado final no es más que una combinación lineal de todas las salidas y seria equivalente a poner una sola capa con esa combinación). Sin embargo todas estas funciones se clasifican entre las conocidas como *gradient descendant*, es decir basadas en el cambio de gradiente, y por tanto al acumular múltiples capas están sujetas al problema de la desaparición del gradiente (o *vanishing gradient*) o su opuesto el problema del gradiente explosivo (*exploding gradient*). Sin entrar en mayores detalles, el vanishing gradient implica que la derivada de la función seria una constante, excluyendo el valor de la entrada y perdiendo de esta manera la noción del cambio a cada paso, reduciendo el gradiente de manera constante. El gradiente representa la velocidad a la que se actualizan los pesos. Esto implica que llegado cierto punto (por ejemplo, al propagarse muchas veces a través de múltiples capas) el gradiente seria tan pequeño que prácticamente inhabilitaría el futuro aprendizaje de la red a partir de ese punto. [5]

[5] Un articulo de divulgación interesante al respecto: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

3.2.3. Tipos de redes neuronales

Una vez más nos encontramos con que la variedad de modelos de redes neuronales que se han ido generando para resolver diferentes tipos de problemas ha derivado en un extenso compendio de estas, difícil de catalogar de manera simple. Por lo tanto, para ofrecer un plano general, intentaremos explicar las principales topologías de redes:

Propagación hacia adelante

También conocidas como redes *feed-forward*, este es el modelo tradicional y en principio el más simple. También se conoce como *perceptron*[23] cuando solo tiene una capa oculta.

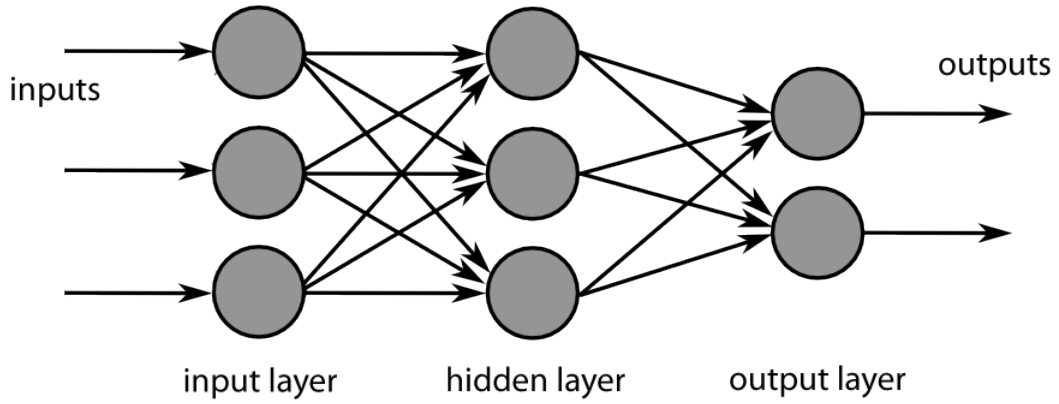


Figura 3.12: Topología de una red feed-forward densa

Una estructura del tipo de la que tenemos en la Figura 3.12 es totalmente dirigida, desde la capa de entrada hacia la de salida pasando por las capas ocultas sin realizar ningún tipo de ciclo. Las conexiones, si bien en la misma dirección, pueden estar arbitrariamente establecidas. En cualquier caso lo normal es encontrar lo que se denomina como redes densas, formadas por neuronas que establecen conexiones con todas las neuronas de la capa inmediatamente posterior

Propagación hacia atrás

El modelo de *backpropagation* o retropropagacion se vale del aprendizaje supervisado para, dado un conjunto de muestras y un conjunto de resultados a obtener a partir de ese

conjunto, obtener no solo el resultado de la red sino el error relativo al resultado esperado tras realizar el entrenamiento.

El proceso comienza como cualquier algoritmo de feed-forward, propagándose capa tras capa hasta llegar a la salida. En este momento, se compara el resultado obtenido contra el resultado a obtener y se obtiene el error global de la salida. Llegados a este punto, el algoritmo empieza a hacer el camino inverso siguiendo el mismo patrón que el entrenamiento calculando por cada salida de cada neurona de las capas ocultas el correspondiente delta (la diferencia entre el valor objetivo y el valor que toma esa salida). Con estos deltas, se obtiene el gradiente del peso de la conexión al multiplicarlos por la entrada a la neurona que se esta evaluando. Tras realizar esto, se resta un porcentaje del nuevo gradiente, en lo que se conoce como "tasa de aprendizaje". El valor de este ratio representa el contraste entre velocidad de aprendizaje/precisión, siendo mayor la primera cuanto más elevado sea y viceversa. El signo del gradiente indica si el error varia directa o inversamente en función del peso. Por lo tanto, el peso debe actualizarse al revés del gradiente. Esto da nombre a la categoría en la que se encaja éste junto a otros métodos de descenso de gradiente.

Propagación lateral

Este es un modelo poco habitual, pero implica un enfoque mixto entre el feed-forward y el backpropagation que puede generar un cierto aspecto de recurrencia parcial. Puede usarse en modelos que utilicen lo que se conoce como aprendizaje competitivo, en el cual las neuronas de cada capa ”compiten” por el derecho a propagar su resultado, en un esquema similar al que se puede observar en la Figura 3.13. Esto requiere una mayor concreción de la función de activación de la neurona. Parece ser que este tipo de redes y aprendizaje ofrece unos resultados bastante optimizados para la búsqueda de clusters o mapas auto-organizados de Kohonen.

Redes Convolucionales

El proceso de convolución por definición consiste en la aplicación de algo que distorsione la muestra inicial para obtener una versión de esa muestra que nos permita sacar conclusiones o características concretas. Matemáticamente, una convolución consiste en, dadas unas funciones f y g, utilizarlas para generar una tercera función superponiendo f y una versión trasladada de g. Por ejemplo, en imágenes se traduce en aplicación de

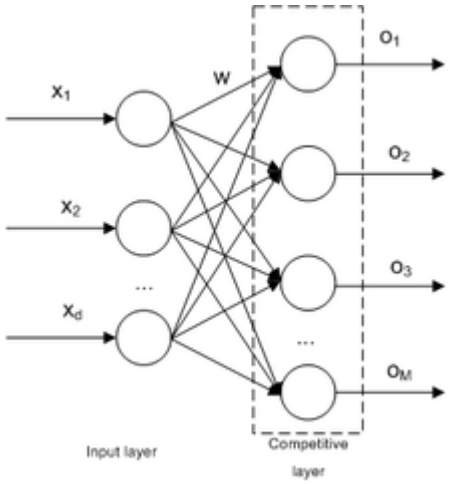


Figura 3.13: Ejemplo de capa de neuronas para aprendizaje competitivo

mascaras que filtran la imagen de entrada de diferentes formas distorsión mediante ruido Gaussiano, obtener "bordes"mediante Laplace del Gaussiano....

Estas redes pueden sacar partido tanto del aprendizaje supervisado para obtener relaciones entrada-salida como de la retropropagacion para evitar problemas de desaparición o explosividad del gradiente.

La estructura de una red convolución consiste en la sucesiva aplicación de capas de pooling, convolución o densas (Estas ultimas funcionando básicamente como un perceptron multicapa), de una manera similar a la que podemos observar en la Figura 3.14. Cada convolución procesa lo que se conoce como un campo receptivo. Esto en el procesado de una imagen podría entenderse como que cada capa de convolución procesaría una de las tres dimensiones de la sección de la imagen que este evaluando (alto, ancho y profundidad). Las capas de pooling o agrupamiento, por su parte combinan la salida de un conjunto de neuronas como entrada única para las neuronas de la capa siguiente, haciendo que el tamaño de la muestra a procesar por la siguiente capa sea inferior al de la anterior. Esto puede ser aplicado de manera local o global por cada capa en función de la estructura de la red.

Por su forma de operar es común ver que se utilicen para problemas de detección o clasificación de características, especialmente en imágenes.

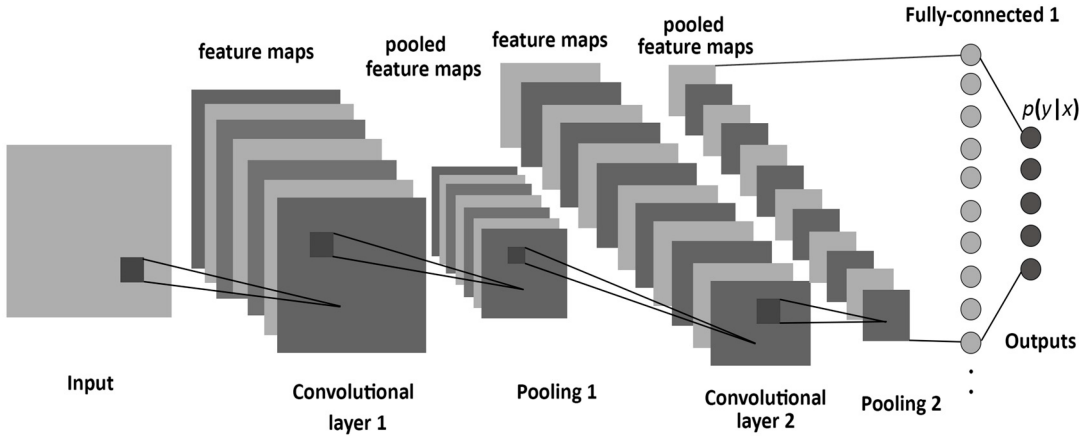


Figura 3.14: Red convolución para el tratamiento de una matriz de datos (ej. una imagen)

Redes Recurrentes

Cuando comentábamos la inspiración biológica de las redes neuronales mencionábamos como los modelos de redes artificiales eran en cierto modo una simplificación de los biológicos debido a que estos últimos tenían una interconexión mucho más masiva. Lo más próximo a este tipo de enfoque lo encontramos en las redes neuronales recurrentes.

Estas redes, a diferencia de las feedforward, no solo propagan la señal a la capa siguiente, sino que pueden hacerlo entre el resto de las neuronas de la misma capa e incluso la propia neurona mediante retroalimentación. Esto posibilita que las neuronas tengan en cierto modo ”memoria” y que el valor de la neurona pueda ser condicionado por los valores de la neurona anterior y posterior de la misma capa, aparte de si misma, obteniendo una simulación de comportamiento dinámico. Cuando tiene este tipo de comportamiento, se dice que además de recurrente es bidireccional. El potencial de estas redes por lo tanto reside en aquellas entradas que supongan una secuencia sobre la cual haya que hacer predicciones, ya sea que sigue un patrón concreto, series que tienen algún tipo de relación temporal o que tenemos información incompleta que queremos completar. Debido a que estas redes son menos intuitivas, un método habitual para representarlas es el diseño de una neurona tipo y aplicarle un desenrollado (o *unfold*) que represente una sección de la red, tal y como se muestra a continuacion en la Figura 3.15.

Este tipo de redes neuronales son, tanto por estructura como por necesidad de cómputo, mucho más complejas que las mencionadas hasta ahora, pero a diferencia de las otras, permiten una aproximación a resolución de problemas como los mencionados en tiempo

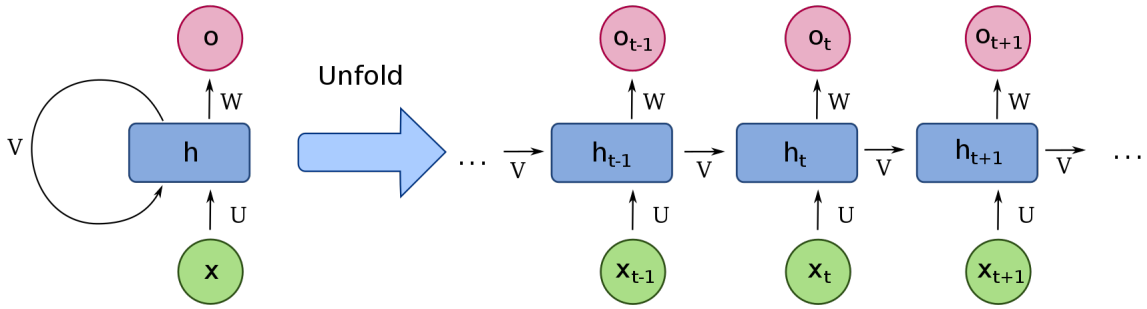


Figura 3.15: Capa recurrente desenrollada para una secuencia con relación temporal

real[6].

LSTM

Debido a su forma de funcionar, las redes recurrentes muchas veces toman estructuras mucho más intrincadas de cara a afrontar un tipo concreto de problemas. Son muchas las alternativas que se han propuesto, pero una de las más versátiles y extendidas fue la neurona Long Short-Term Memory (LSTM) ideada a finales de los 90 por Hochreiter-Schmidhuber[24].

Entre las bondades a grandes rasgos de esta variante figura el hecho de que controla los problemas de gradiente explosivo o su desaparición, a diferencia de una *full-recurrent*. También pueden acumularse sucesivas capas de este tipo de células para refinar los resultados y son de las que más partido sacan de la bidireccionalidad. Además gracias a la inclusión de lo que define como "puertas de olvido", permite almacenar en cada célula información a mucho más largo plazo que las recurrentes normales, ya que discrimina qué información se deja de lado y cual se mantiene entre diferentes iteraciones sobre la misma célula. Podemos observar la estructura de una celula de este tipo en la Figura 3.16.

Esta estructura ”interna” a la célula abrió la puerta a otras variantes de neuronas modificadas como la Gated Recurrent Units (GRU). Además el uso de este tipo de neuronas han derivado en algoritmos que actualmente son *state of the art* en diferentes problemas como el del reconocimiento de patrones[15].

[6]Un ejemplo de predicción de trayectorias http://www.paper.edu.cn/scholar/showpdf/NUT2cN2INTD0YxeQh

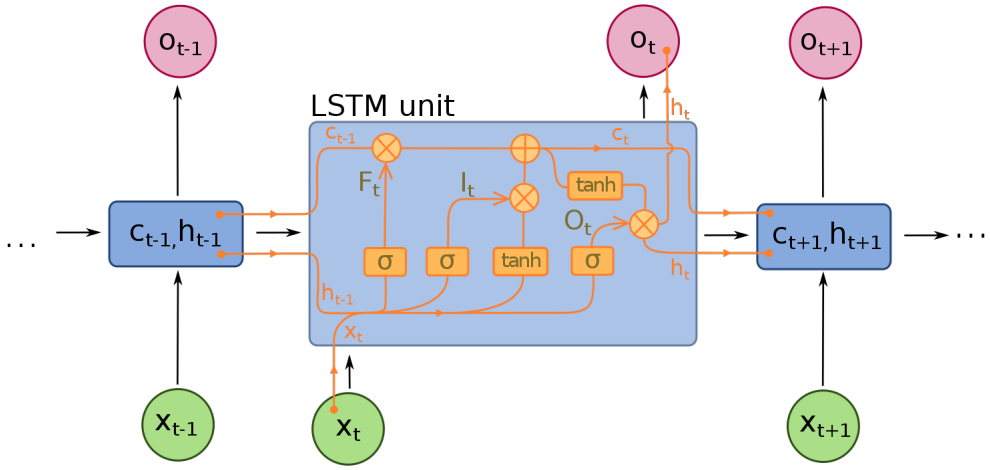


Figura 3.16: Detalle de la composición de una célula LSTM

3.3. Reconocimiento de Voz

3.3.1. Enfoques planteados hasta ahora

Tal y como se comentaba en el punto 1.2.2, junto con las diferentes tecnologías que han ido surgiendo, son varios los métodos que se han utilizado para encarar el problema del reconocimiento del habla. Nos apoyaremos en material generado para el seminario sobre DeepLearning(DL) del año pasado de la UPC [7] para explicarlo a grandes rasgos de manera más visual.

GMM-HMM

Los modelos ocultos de Markov fueron como ya se ha comentado en la sección del Estado del Arte la base de todo algoritmo de reconocimiento de voz del siglo XX. Pero para que el modelo pudiera interpretar las transiciones entre los diferentes estados era necesario un pre-proceso inicial, no solo del audio mediante extracción de características representativas (cepstrales, normalización del tracto vocal...), sino de las propias características de cara a asociar los valores de estas con diferentes elementos de un modelo oculto de Markov. Por ejemplo mediante el GMM obtendríamos los diferentes fonemas y mediante el HMM las asociaciones más probables de estos para formar diferentes palabras (básica-

[7] https://telecombcn-dl.github.io/2017-dlsl/

mente, lo que se conoce como el modelo acústico y el modelo del lenguaje). Vease Fig. 3.17.

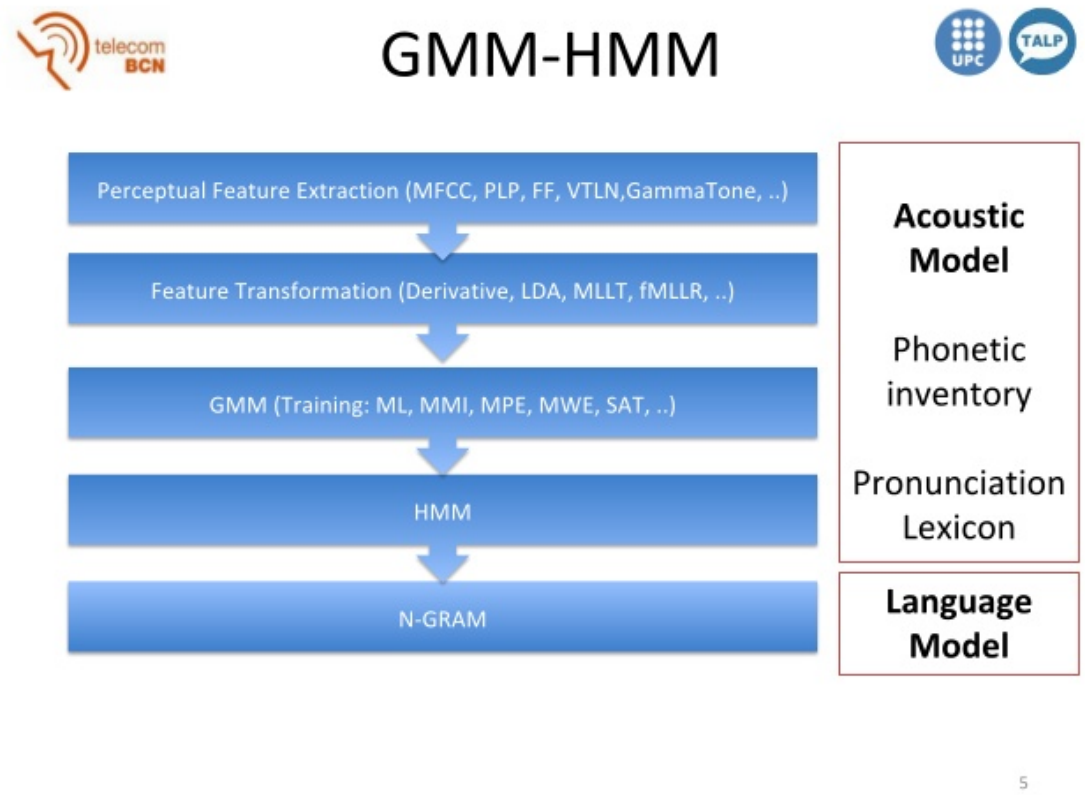
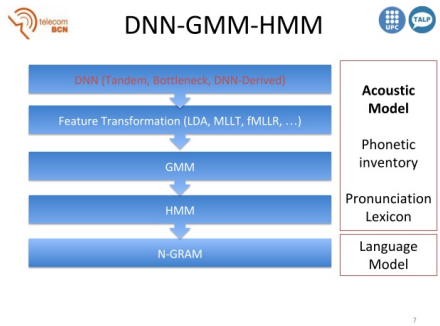


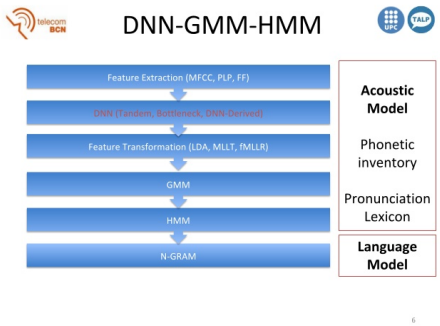
Figura 3.17: Estructura de un modelo GMM-HMM

DNN-GMM-HMM

Con la introducción de las redes neuronales en las diferentes soluciones de este problema, se aplicaron varios posibles enfoques.

Mediante la utilización de redes neuronales profundas (DNN), podríamos obtener bien un preproceso de la señal de audio mediante el reconocimiento de patrones en la propia señal o bien el uso de técnicas de extracción características sobre las que luego buscar similitudes, todo esto como refuerzo del modelo GMM-HMM mencionado antes.





DNN-GMM

Finalmente y siguiendo con la linea de aplicar DNNs para el reconocimiento de voz, se llego a la conclusión de que la optimización de la propia red (mediante la adición de capas, ampliación de la cantidad de células en éstas, refinado de la función de activación...) podía no sólo mejorar el rendimiento del sistema sino incluso realizar varias de las tareas en una sola pasada. Esto supuso un cambio de mentalidad importante: el entrenamiento previo (la definición de fonemas explícitamente que había que intentar ”encajar” en ciertos rangos o cualquier otra información conocida que se añadía antes de clasificar las muestras conocidas) perdía relevancia frente a la mejora que suponía el hecho de entrenar redes más eficaces gracias a la masificación de la cantidad de muestras. Esto es especialmente relevante para la creación del modelo acústico, que aun supone la carga de trabajo más pesada del proceso.

A falta de las mejoras de hardware por llegar y el uso de redes más complejas (como el uso de neuronas LSTM para procesar la información secuencial paliando el problema de la desaparición del gradiente) era común ver que modelos basados en DNNs todavía se apoyaran en los HMM para la creación del modelo de lenguaje.

A rasgos generales, la introducción de las DNNs en los sistemas de reconocimiento supuso que estos barajaran mucha más información, tanto de muestras como parámetros a evaluar. También permitían una calibración más fina del modelo basada en datos empíricos mediante la prueba de diferentes tamaños o funciones de activación. También permiten una computación más ”rápida” de estos modelos, si bien los modelos *state of the art* aun requieren de semanas de entrenamiento para alcanzar precisiones superiores al 90 %[8].

[8] https://ai.google/research/pubs/pub46687

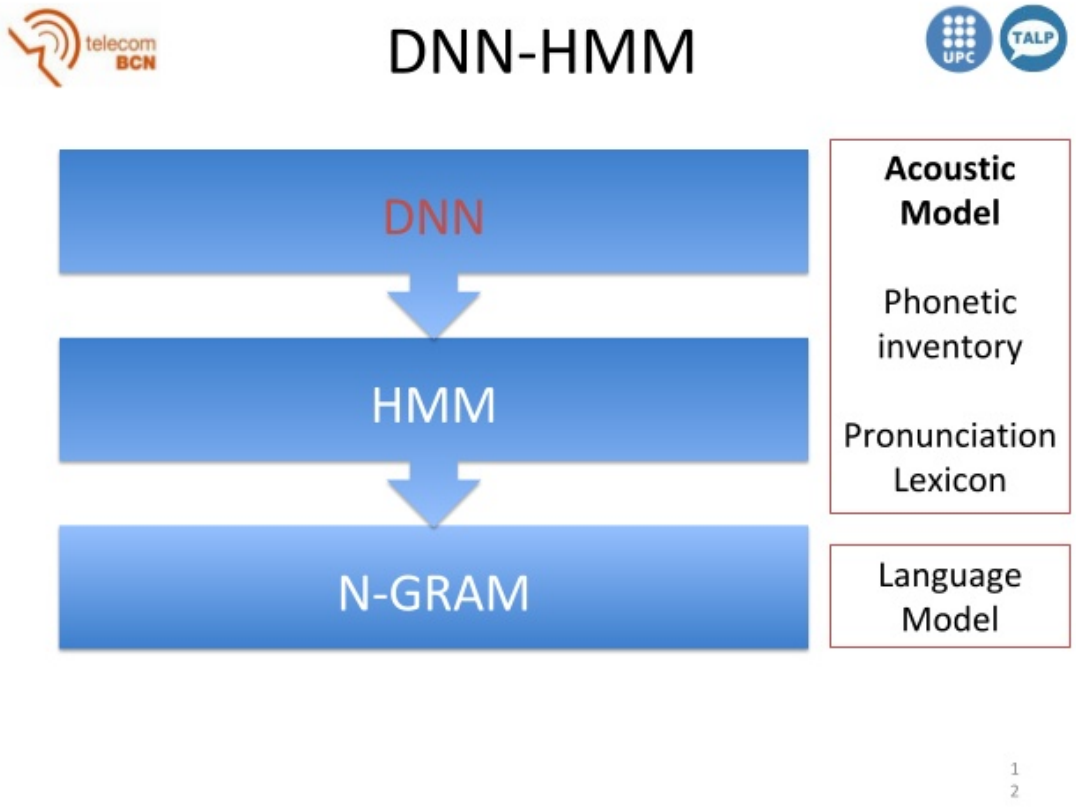


Figura 3.18: Estructura de un modelo DNN-HMM

4. CAPÍTULO

Desarrollo

Tal y como se mencionaba en el apartado 2.1.1, se han realizado varios trabajos mutuamente independientes para el desarrollo de este proyecto. Para una mayor diferenciación de las distintas soluciones implementadas, dividiremos este apartado en tres secciones principales: Reconocimiento Facial, Reconocimiento del Habla y Reconocimiento de Palabras Clave.

4.1. Reconocimiento Facial

Ya se comentaba en la introducción que uno de los objetivos del proyecto era la implementación de un sistema de reconocimiento facial para hacer el proceso de *login* en la herramienta. Partíamos de un estado en el cual el usuario debía introducir sus credenciales para poder realizar el login, con una interfaz ya diseñada para tales efectos.

La única modificación realizada en esta interfaz consiste en la inclusión de un botón que habilite el reconocimiento facial (Fig. 4.1) y un *canvas* oculto que nos permitirá más adelante realizar el seguimiento del reconocimiento facial que el usuario realice, mediante una presentación en tiempo real del stream que recibe la cámara.

Para el reconocimiento facial se ha utilizado la API de Face[1] que ofrece Azure como modulo de su paquete de herramientas de Visión. Se ha implementado tanto en C# como

[1]Funciones, parámetros y respuestas:https://westus.dev.cognitive.microsoft.com/docs/services/563879b61984550e40cbbe8d/operations/563879b61984550f30395236

Figura 4.1: Pantalla de login tras las modificaciones

en JavaScript. Para la gestión de los recursos necesarios se ha utilizado la implementación inicial en consola, ya que suponía una ventaja no tener que desarrollar una interfaz para las diferentes operaciones. Sin embargo, de cara a la integración en la herramienta, la segunda opción presentaba más prestaciones y facilidades.

En cualquiera de los dos casos, se consume un servicio REST para cada operación, siendo para ambas implementaciones el mismo *endpoint* con exactamente los mismos parámetros. Una vez gestionada la captura de la cara del usuario, el programa realiza una petición xhttp como la siguiente:

```
function Recognize() {
    return new Promise(
        function (resolve, reject) {
            //image at actualImage as jpeg
            document.getElementById("actualImage").toBlob(function (blob) {
                var url = uriDetect + "?" + $.param(paramDet);

                var xhttp = new XMLHttpRequest();
                xhttp.onreadystatechange = function ()
                {
                    if (this.readyState == 4 && this.status == 200)
                    {
                        if (xhttp.response != "[]") {
                            paramIdent.faceIds = [JSON.parse(xhttp.response)[0].faceId];
```

```
15                          resolve();
16                      }
17                      else {
18                          reject();
19                      }
20                  }
21                  else if (this.readyState == 4 && this.status != 200) {
22                      reject(xhttp.statusText);
23                  }
24              };
25
26              xhttp.open("POST", url, true);
27              xhttp.setRequestHeader("Content-type", "application/octet-stream");
28              xhttp.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);
29              xhttp.send(blob);
30
31          }, "image/jpeg");
32      });
```

Al realizarse correctamente, devuelve un JSON con los parámetros correspondientes a la imagen (en este caso el código asociado a las características obtenidas de la imagen enviada, llamado faceID en el contexto de la API). Con esos parámetros se realiza otra petición para confirmar si esa imagen se corresponde a alguien incluido en la base de datos y obtener su información. Finalmente con los datos almacenados sobre esa persona en la base de datos se realizaría el login con la función ya implementada para ello.

Como medida adicional, para asegurar que el login se haga correctamente a pesar de factores externos(cambios bruscos de iluminación, perdida parcial de la imagen por algún objeto...) a la hora de tomar la captura que se utiliza para la petición inicial, se hace una captura y procesado de esta una vez cada 2 segundos durante 10 segundos. Si tras este plazo no se hubiera podido realizar el login, se cierra el proceso de login por reconocimiento facial con un mensaje de error. En la figura 4.2 podemos observar como se realiza todo este proceso.

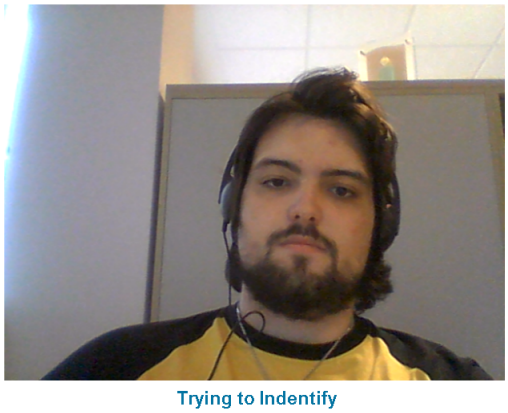


Figura 4.2: Reconocimiento facial en curso

4.2. Reconocimiento del Habla

En la implantación del sistema de Reconocimiento del Habla en la herramienta han sido necesarias dos herramientas diferentes: por un lado la API de Speech y por el otro LUIS (Language Understanding). Ambas, al igual que el punto anterior, son servicios REST y también fueron implementadas en C# y JS. Sin embargo esta vez se comprobó que una interfaz básica facilitaba el uso intuitivo de la herramienta y ambas daban diferentes tipos de problemas sobre los que se hablara en el apartado de Complicaciones. Por un lado, se implemento un prototipo básico y funcional para la transcripción de voz a texto (Fig 4.3) en el que se podía, entre otras cosas, probar el sistema con diferentes idiomas y cambiar el modo de entrada: Interactivo, mediante el clásico Start/Stop; Conversacional, que se mantiene activo y analiza las frases a medida que se le van diciendo o Dictado, similar al interactivo pero que realiza la transcripción prácticamente en tiempo real hasta que haya una pausa.

Una vez realizado todo esto, para poder extraer información relevante de las diferentes frases introducidas se procedió a la creación del modelo de lenguaje en LUIS. Para esto se facilita una aplicación online (Fig. 4.4) que permite la introducción de nuevas entradas

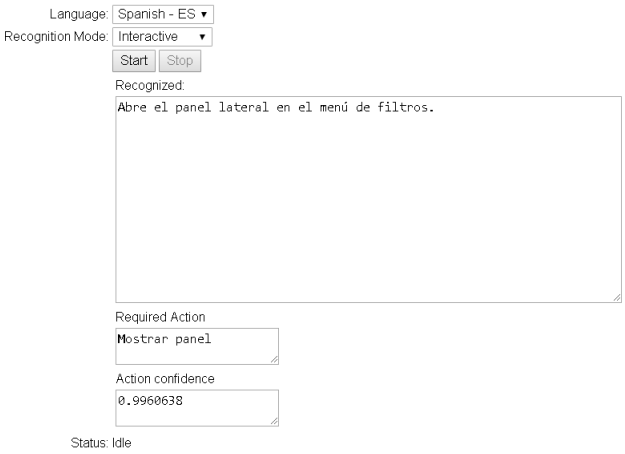


Figura 4.3: Prototipo del reconocimiento de voz y obtención de la intención asociada a la frase reconocida

y herramientas para clasificarlas e incluso opciones de revisión de las peticiones recibidas para realizar aprendizaje correctivo. Por cada acción a reconocer (llamada *Intent* en este contexto), se introducen múltiples *utterances* (enunciaciones) sobre las cuales hay que clasificar las diferentes entidades que aparezcan (las cuales pueden ser simples, jerárquicas, una serie de valores asociados a un estado...). Tras realizar un entrenamiento suficiente, empezando a realizar esto de manera automática al introducir nuevas utterances y ya estaba listo para ser usado en la aplicación.

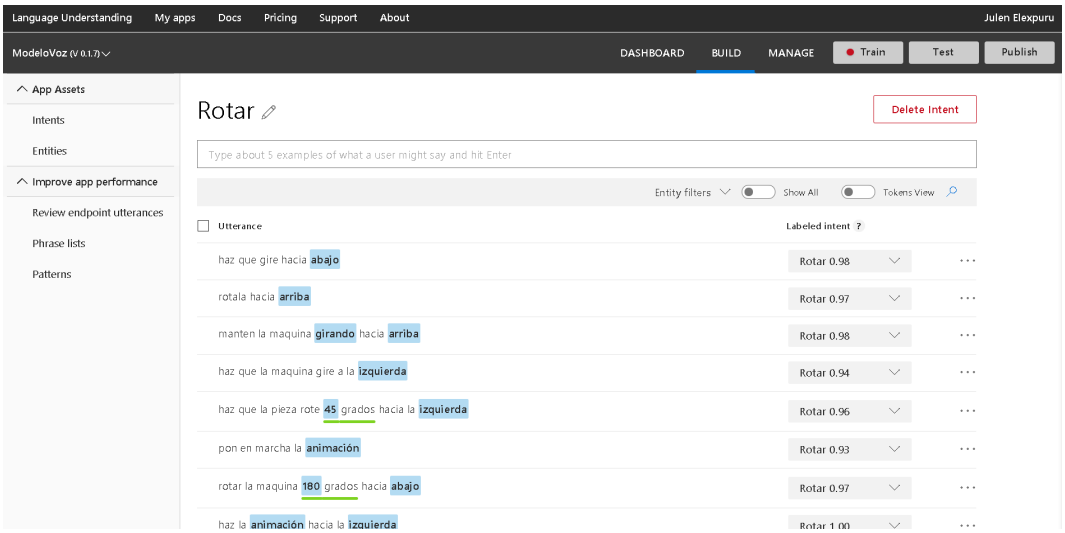


Figura 4.4: Aplicación para el desarrollo de modelos de lenguaje de LUIS

Implementado el proceso de llamada a estos servicios (una vez más mediante peticiones xhttp), el funcionamiento del sistema consiste en ejecutar primero el reconocedor para la obtención de lo dicho por el usuario para posteriormente enviarlo a la aplicación de procesado del lenguaje, la cual nos devuelve varios datos como la intención más probable (junto con su probabilidad y la de que sea cualquier otra), las entidades que contiene la frase indicada y el tipo de estas, todo esto en un JSON de un formato similar a este:

```
{
  "query": "abre el panel lateral en el menú de filtros",
  "topScoringIntent": {
    "intent": "Mostrar panel",
    "score": 0.9960638
  },
  "entities": [
    {
      "entity": "filtros",
      "type": "panel",
      "startIndex": 36,
      "endIndex": 43,
      "resolution": {
        "value": "Filtros"
      }
    },
  ]
}
```

...permitiendo así procesar las diferentes variables obtenidas y sus valores para realizar cualquier acción deseada.

Este prototipo se introdujo casi con el formato previamente mostrado en la herramienta sobre la que debía operar, si bien una vez realizadas todas pruebas pertinentes se oculto por motivos estéticos, habilitando la funcionalidad conversacional mediante un simple botón (Fig 4.5).

Figura 4.5: Panel de pruebas de reconocimiento de voz. El botón del lateral izquierdo con el micrófono es lo único visible en la versión final.

4.3. Reconocimiento de Palabras Clave

El proceso de creación de un modelo de Reconocimiento de Palabras Clave fue arduo principalmente por la necesidad de conocer con que se estaba trabajando y por lo tanto la implementación de varias funciones auxiliares para poder observar los patrones que seguían las señales, espectrogramas y diferentes coeficientes generados por estos.

Inicialmente se procedió al desarrollo de una herramienta de grabación de muestras sobre las que poder trabajar. Para ello se tomaron como referencia los parámetros observados en el apartado 3.1 que indicaban que, para poder realizar una buena extracción de coeficientes para el Reconocimiento del Habla, es recomendable que la muestra se tome a 16kHz (Obteniendo un ancho de banda de 8kHz) y necesario que este grabada en mono, es decir, que este formada por una sola señal de audio. Para representarlo numéricamente se considero suficiente el tamaño int16. La codificación se realizo en WAV al ser este uno de los formatos de más fácil manejo, en este caso tratable directamente desde Python.

Al margen de lo que es la grabación en si se diseño un sistema de gestión automático de la muestras para facilitar su posterior uso y clasificación. Para ello simplemente se ha tenido en cuenta si la muestra es valida (contiene comando o no) y el sexo del hablante (por la diferencia de frecuencias que se suele presentar entre estos), representando así cada archivo como una combinación de v/n y h/m (ejemplo, nh1 representa la primera muestra no valida grabada por un hombre).

Con el proceso de grabación ya solventado, se procedió a implementar una opción que generase el espectrograma de las diferentes muestras. Esto permitía realizar una evaluación visual de las diferentes muestras que se estaban generando para evitar así la introducción de ruido que de simple oídas no se detectara. Además, ateniéndonos a la Figura 4.6, podemos confirmar que el ancho de banda elegido es en principio suficiente para captar toda la información relevante necesaria.

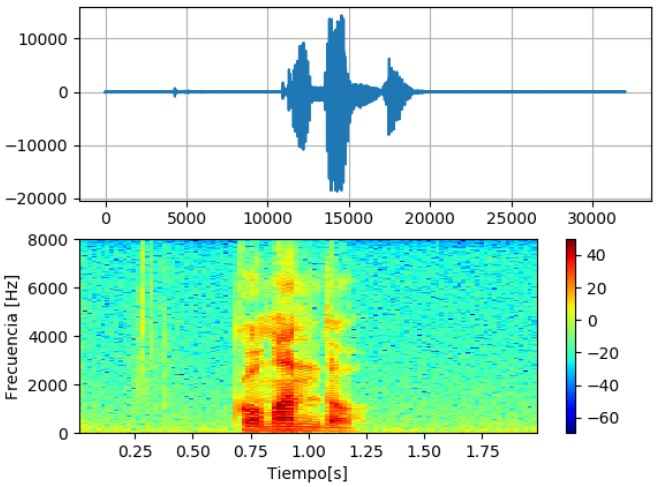


Figura 4.6: Espectrograma generado por un hombre diciendo çomando"

Inicialmente se considero la introducción de la información contenida en el espectrograma directamente al clasificador, pero tras observar que los resultados obtenidos eran insuficientes, se procedió a la extracción de los MFCC.

Para la extracción de estos, se utilizo la librería python_speech_features[2] que mediante la función mfcc permite generar los coeficientes cepstrales y almacenarlos en un *numpy.array*. Para gestionar estos diferentes arrays más cómodamente se realizo un volcado de los mismos a archivos de texto. Al extraer estos coeficientes, tal y como detalla la documentación de la librería, se pueden especificar varios parámetros diferentes de cara a optimizar los coeficientes obtenidos para el problema a resolver. En este caso, los parámetros por defecto ofrecían unos resultados bastante buenos, así que la única modificación que se le realizo fue especificar la función de enventanado como Hamming, ya que por defecto no utiliza ventana. Aplicar esta ventana ayuda a resaltar la parte central de cada fragmento de la muestra sobre el que se este trabajando.

Con los coeficientes ya obtenidos, el único paso restante era la introducción de estos al modelo encargado de realizar la clasificación. Para ello se han probado una serie de entre todas las combinaciones posibles, siendo las que mejor han funcionado hasta ahora las redes multicapa LSTM, lo cual es lógico, ya que están pensadas para trabajar sobre audios, vídeos, o cualquier otro tipo de reconocimiento de patrones. En base a esta estructura se han probado diferentes modificaciones mediante la inclusión de capas de Dropout o

[2] https://python-speech-features.readthedocs.io/en/latest/

ajustando los diferentes parámetros de entrenamiento (ver el apartado Experimentación del Apéndice).

5. CAPÍTULO

Análisis del proyecto

Este capitulo pretende ofrecer un análisis a posterior del proyecto una vez realizado, con animo de hacer una autoevaluación del desarrollo de este y comprobar cuanto se ha desviado de su cometido inicial.

5.1. Complicaciones

Si bien gracias a una correcta planificación inicial y una buena estimación de los tiempos requeridos para cada sección de el proyecto no ha habido mayores incidentes si se podría considerar que ha habido algún contratiempo menor:

- Por un lado, dado que es un tema muy extenso y que se puede abordar desde múltiples enfoques, si que hubo una ligera perdida de tiempo al documentar el estado del arte y decidir cual de las opciones se adaptaba más al problema a solventar. Hay que considerar también que como el objetivo era encontrar mejoras a aplicar a lo que ya estaba desarrollando había una cierta incertidumbre inicial sobre cual era la mejor opción sobre la que centrarse (reconocimiento facial o de voz).
- Relacionado con el punto anterior, ya en la fase de implementación al observar que los resultados iniciales se estancaban a partir de cierta precisión, hubo que replantear el preproceso que se estaba realizando y la clasificación de los nuevos parámetros, lo que lastro considerablemente la etapa final del proyecto.

- Además, en cuanto a los servicios utilizados para reconocimiento facial y de voz, en función de la red en la que se este trabajando la versión de consola puede dar problemas de conectividad. En cambio, al usar la versión JS, se utiliza una clase llamada Recognizer incluida en la API que al parecer tira del estándar getUserMedia, que tras las ultimas actualizaciones de los navegadores en verano necesita de conexiones https o directamente no funciona en iOS si no se usa Safari.

5.2. Resultados

Con el proyecto ya finalizado y una versión estable del modelo generada, se han alcanzado precisiones que oscilan entre el 75 y el 85 %(En la Figura 5.1 se muestra una captura de un modelo generado en este rango). Esta variación se debe principalmente a la naturaleza estocástica de los modelos basados en redes neuronales. Esto no llega a los niveles de precisión del *state of art* actual (situado aproximadamente entre el 90/95 % de precisión)[1] pero es una aproximación razonablemente buena teniendo en cuenta el tiempo de desarrollo.

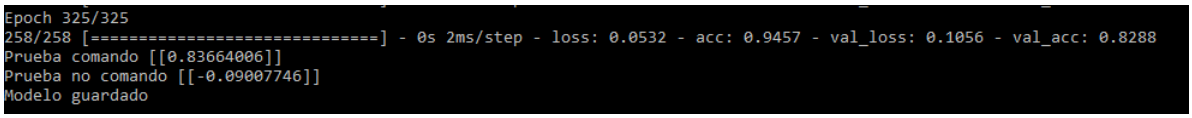


Figura 5.1: Resultados obtenidos al generar un modelo con las muestras y parámetros actuales

Para alcanzar estos resultados, como se ha comentado en otros apartados, se han utilizado muestras procesadas para que duren exactamente 2 segundos conteniendo cada una de ellas una palabra. Han sido grabadas a 16kHz en un único canal. Para la extracción de características se ha usado un tamaño de ventana de 0.02ms espaciadas entre ellas por 0.01ms usando una ventana Hamming de 400 puntos, obteniendo con todo esto 13 cepstrales.

Las muestras obtenidas para entrenar el modelo provienen tanto de hombres como mujeres de entre 20 y 40 años. En total se han utilizado 369 muestras, equivalentes a 738 segundos de audio. De estas muestras, 73 suponen muestras validas de la palabra comando (38 proporcionadas por hombres y 35 por mujeres) y 296 muestras no validas (147

[1] Sin ir más lejos, un articulo reciente con esos resultados: `https://irjet.net/archives/V5/i9/IRJET-V5I9169.pdf`

pertenecientes a hombres y 149 a mujeres). Estos valores se multiplican por 6 al realizar la ampliación de muestras mediante variación de los coeficientes (ver apartado Experimentación).

Si bien la muestra es bastante homogénea en cuanto al sexo se refiere, si que se ha detectado una menor precisión en el reconocimiento de la palabra comando en mujeres que en hombres. Esta variación oscila entre un 5 y un 15 % en la precisión tras validar los resultados, siendo un comportamiento claramente a mejorar. Por otro lado, en la versión realizada haciendo uso de muestras ampliadas esta diferencia se reduce a un 5 % prácticamente constante, si bien a costa de algo de precisión en general(65/70 %).

6. CAPÍTULO

Conclusiones

El objetivo de este capitulo es realizar una síntesis y hacer una evaluación de los datos y resultados presentados.

6.1. Conclusiones generales del proyecto

A pesar de que había bastantes objetivos por cumplir, tanto en lo referente a las obligaciones adquiridas con la empresa como en lo que al desarrollo del proyecto académico se refiere, se han podido cumplir todos ellos de manera satisfactoria, incluyendo algunos objetivos adicionales (como la corrección y mejora de elementos relacionados con three.js).

Lamentablemente no ha habido ocasión de profundizar en el uso de otros modelos de reconocimiento del lenguaje, ya que se ha primado mejorar la eficacia de lo que se había realizado frente a la elaboración de modelos más complejos. Si bien quizás no se ha conseguido un rendimiento *state of the art*, lo considero un resultado bastante bueno teniendo en cuenta que el punto de partida era un conocimiento prácticamente nulo de muchas de las tecnologías usadas y el tratamiento de señales. Hay que considerar también la complejidad de obtener una cantidad de muestras considerable en función del problema a resolver dentro del reconocimiento del habla (por ejemplo, es mucho más fácil obtener material en otros idiomas). También el hecho de generar un dataset efectivo, más allá de simplemente añadir la palabra correcta, requiere también un análisis y estudio detallado de cuanta información aportan las nuevas muestras.

Lo realmente enriquecedor de este proyecto ha sido el que haya planteado la posibilidad de afrontar un problema real combinando todas las herramientas disponibles, siendo estas tan diferentes como pueden ser las tecnologías web, los gráficos por computador o los principios del procesamiento del lenguaje natural. El extenso proceso de documentación que ha habido que realizar para poder tener una base mínima sobre la que poder operar en el tema ha sido también una motivación para interesarse por temas que hasta ahora no me hubiera planteado.

Es resaltable también que el proceso de planificación y gestión del tiempo que se ha llevado ha permitido realizar el proyecto en un plazo bastante ajustado sin tener que forzar los ritmos de este.

Finalmente y de cara a la empresa, cabe comentar que aunque la optimización propuesta pueda parecer simple, su implantación supondrá un ahorro de costes significativo en el servicio contratado (se ha calculado que aproximadamente un 80 % de las llamadas necesarias) lo que aporta más competitividad a la solución desarrollada.

6.2. Lineas futuras

Dado que, como se comentaba en el apartado anterior, aun quedaría mucho trabajo por hacer para alcanzar una precisión que pudiera considerarse competitiva con los resultados obtenidos por el *state of the art* actual (que rondan el 95 % de precisión) una linea futura obvia seria la exploración de nuevos modelos de reconocimiento del lenguaje basados en otras arquitecturas más allá de las *isolated words* como podrían ser el procesamiento de fonemas en una secuencia de audio (algoritmos como el CTC) o cualquier otro algoritmo de clasificación por secuencias temporales (TSC). También podrían considerarse otras aplicaciones más allá del simple reconocimiento de estas secuencias de audio como el procesamiento del lenguaje en ellas (para extracción de estados de animo, contexto, significado...).

Otra linea de investigación a tener en cuenta seria el uso de otros clasificadores aparte de las redes neuronales (como los HMM o los Support Vector Machines) y las diferentes combinaciones entre todos ellos, si bien ahora mismo los primeros se consideran algo desfasados en el contexto de ASR. También podría intentar desarrollarse un clasificador más adaptado al contexto del problema, pero eso requeriría un esfuerzo significativo.

Además, el uso de otras técnicas para procesar las señales de audio podría ser interesante, ya que es un campo que abarca muchas posibilidades y ayudaría a mejorar la calidad

de las muestras utilizada para cualquiera de las opciones anteriores o incluso resaltar la información relevante contenida en ellas mediante la aplicación de diferentes filtros.

Otro tema sobre el que se podría investigar mucho más porque se ha tratado muy superficialmente en este proyecto podría ser desarrollar un reconocimiento facial más avanzado mediante el uso de técnicas para reconocimiento 3D, que prevendrían muchas de las carencias que tiene el reconocimiento 2D habitual, usado por el servicio actual.

Anexos

A. ANEXO

Experimentación

En este apartado se dará algo más de detalle sobre los diferentes comportamientos que se han podido apreciar durante el entrenamiento del modelo de Reconocimiento de Palabras Clave.

Antes de proceder a desgranar los detalles de las diferentes pruebas que se han realizado, se aclararan de manera concisa algunos términos habituales en la implementación de redes neuronales que se verán a continuación:

- **Hidden-size:** También conocido como tamaño de la capa oculta o simplemente tamaño de la capa. Indica la cantidad de neuronas de la que dispone cada capa.
- **Epoch:** Un epoch equivale a una iteración completa del modelo a través de los datos proporcionados como entrada y salida. Es decir, el numero de epochs equivale a el numero de veces que se entrena el modelo.
- **Loss:** O función de perdida. Es la función que calcula la perdida, que viene a suponer la diferencia entre los valores originales y los predichos. Se puede utilizar cualquier función, pero las habituales suelen ser MSE, SSE o en caso de ser un problema con múltiples clasificaciones posibles, Cathegorical crossentropy.
- **Learning rate:** O tasa de aprendizaje. Cada modelo suele tener asociada una tasa de aprendizaje (o parámetro similar) predeterminada. Sin embargo, es uno de los parámetros más modificados, ya que influye directamente en la eficacia del modelo generado (a costa de mayor tiempo y coste de computo). A menor learning rate,

más tarda un modelo en generarse, pero con cambios menos radicales entre epochs, lo que aporta un aprendizaje ”más seguro”.

- **Decay rate:** Asociado con el punto anterior, la tasa de decay representa cuanta capacidad de aprendizaje pierde el modelo a lo largo del tiempo (generalmente, medido en epochs). Esto permite entre otras cosas, prevenir el overfitting a base de hacer que un modelo ”muy entrenado” deje de modificar gran parte de la información que ya tiene.

- **Momentum:** El momentum es un parámetro que permite ajustar la velocidad a la que cambia el gradiente cuando se acerca hacia un resultado valido de acuerdo a la función objetivo. En resumen, aumentar el momentum supone dar mayor valor a los pesos que más influyan a la hora de obtener un buen resultado en el clasificador.

- **Dropout:** Las capas de Dropout están definidas en la mayor parte de APIs de redes neuronales para permitir que en cada epoch, de manera aleatoria (en una tasa que se establece entre 0 y 1), descarte el resultado obtenido. Aplicar este tipo de capas suele se una buena idea cuando se dan casos de overfitting o con estructuras de redes neuronales que sean propensas a este.

Habiendo aclarado este conjunto de terminos, podemos proceder a relatar algunos de los cambios que se han planteado o han sido probados:

A.1. Tipo de red

En cuanto a los tipos de redes utilizados, inicialmente se probo con redes Densas, que rápidamente demostraron no ser aptas para este tipo de problema sin una adaptación más exhaustiva de la entrada. A continuación se barajo el uso de las redes Convolucionales y posteriormente el de las LSTMs. Las primeras daban resultados aceptables, pero exigían una mayor cantidad de muestras y especificar más parámetros que las segundas para dar resultados mínimamente parecidos. Solo con la aplicación de 4 capas LSTM con los valores por defecto el clasificador empezó a rondar el 60 % de precisión. Adicionalmente, se probaron por curiosidad las GRUs, que vienen a ser algo similar a las LSTM pero con menos parámetros. El resultado era algo similar al de las LSTM aunque ligeramente más lento. Finalmente, mencionar que para la ejecución final de los entrenamientos se ha hecho uso de las variantes CuDNN(librerías que proporciona NVIDIA para procesado de

DNN en tarjetas gráficas) de estas dos ultimas capas para agilizar el proceso, suponiendo entre un 60 y un 80 % menos de tiempo para este.

Una vez decidido usar capas LSTM, se probaron diferentes números de capas. Con una o 2 capas los resultados obtenidos eran bastante pobres. A partir de 3 capas empezaron a obtenerse resultados razonables pero ya con 5 o más parecía no notarse diferencia de precisión pero si un aumento del coste de computo. Por lo tanto al principio se opto por 3 capas LSTM y, al aumentar el numero de muestras, se opto por añadir una 4 ya que se observo que los resultados si seguían mejorando.

Adicionalmente, para la versión con muestras ampliadas, se observo que era mucho más propensa al overfitting. Para paliar esto se hizo uso de capas de Dropout con un índice de 0.2 (es decir, cada epoch tenia un 20 % de posibilidades de ser descartado) tras cada Capa de LSTM.

A parte, para poder procesar un resultado único de las capas anteriormente mencionadas, es necesario incluir una ultima capa densa (con una sola neurona basta) para ”sintetizar” lo que llega de las neuronas de la capa anterior. Es a esta capa a la que se aplica la función de activación que nos interesa, en las anteriores se ha dejado la predeterminada (en el caso de las LSTM, la tangente hiperbólica).

A.2. Muestras

Podríamos decir que a lo largo de las diferentes fases de implementación de la red neuronal, si nos atenemos al numero de muestras, ha habido 3 situaciones: la etapa inicial, con pocas muestras y poca variedad de ellas (siendo principalmente grabaciones mías); la etapa intermedia, en la que se realizo una modificación de las muestras disponibles para generar muestras sintéticas y finalmente la etapa final, en la que se hizo una recopilación de muestras lo más variada posible en cuanto a genero y personas, con tal de reducir la dependencia del sistema a la voz original del entrenamiento y mejorar su funcionamiento en general.

Durante la primera fase simplemente se realizaron tanteos de diferentes tipos de redes como ya se ha mencionado. Pero se llego a un punto en el cual no se conseguía ningún tipo de mejoría tras alcanzar una precisión concreta, ni haciendo uso de lo que se mencionara más abajo ni aumentando la precisión de los adiós (grabándolos con frecuencias mayores o probando diferentes tamaños y formatos de ventana). Al observar que las recomendaciones habituales para este tipo de circunstancias solían tener relación con el

numero de muestras pero habiendo dificultad para conseguirlas de manera inmediata, se procedió a buscar métodos de data aumentativo con los que modificar las muestras actuales para generar muestras sintéticas. La mejor opción entre las que se encontró resulto ser la perturbación de los cepstrales con coeficientes mínimos (entre 0.9 y 1.1). Generando 5 muestras sintéticas por cada muestra original, se dispuso de un dataset lo suficientemente amplio para poder seguir realizando pruebas mientras se obtenían muestras reales suficientes para poder trabajar sobre ellas. Cuando se llego a la fase 3, se disponía de una cantidad de muestras significativamente mayor que al principio. Aunque inferior a la cantidad de muestras sintéticas, han ofrecido resultados mejores que estas, probablemente por ser más coherentes que las generadas de manera aleatoria.

A.3. Tipo de activación

A falta de mayor información o tiempo para probar todas las combinaciones posibles y descartando rotundamente la alternativa de diseñar una función especifica para el problema, se decidió realizar pruebas con las tres funciones más comunes, a saber: la sigmoidea (y su variante dura), la tangente y la relu.

Mientras que las 3 primeras daban resultados dentro de lo normal (siendo la tangente la que sobresalía) la relu con sus valores por defecto daba resultados difíciles de interpretar, por lo que se descarto a mediados de la implementación.

A.4. Tipo de error y optimizadores

De manera similar al apartado anterior, se opto por utilizar para medir el error varias de las opciones más comunes: binary-crossentropy y mse. Se podría haber usado otras variaciones de la raíz del error, pero la mse es una de las más usadas y que mejores resultados aporta a nivel general. En este caso concreto, al ser la base del problema una decisión de aceptación o rechazo, se supuso que binary aportaría mejores resultados. Con pocas muestras ambos daban resultados mediocres, pero al aumentar el numero de muestras los resultados de usar mse suponían una mejora significativa frente a lo anterior.

En cuanto a los optimizadores nos encontramos una vez más en la misma tesitura y se barajaron sobre todo los usos de: sgd (el descenso de gradiente clásico) y las variantes ada (de gradiente adaptativo): adagrad, adadelta, adam... Al final todas son relativamente

parecidas, con más o menos parámetros con los que jugar con el gradiente. En cuanto a eficiencia se refiere, si es cierto que adam era el que permitía alcanzar buenos resultados más rápidamente (con menos epochs y coste). Sin embargo tanto con los falsos positivos como con los falsos negativos se obtenían valores extremos y en las pruebas reales no daba la sensación de tener la precisión que indicaba. Por lo tanto, con intención de tener más control sobre las operaciones que se realizaban y siendo ya las redes LSTM bastante eficientes a la hora de trabajar con el gradiente, se decidió usar el sgd como optimizador.

A.5. Modificación de parámetros

Una vez teniendo como referencia el apartado anterior, empezó un proceso basado completamente en la prueba de diferentes combinaciones de los parámetros del optimizador: la tasa de aprendizaje, la decadencia de esta y el momentum.

La tasa de aprendizaje por defecto del sgd es 0.01, el momentum 0 y el decay 0. partiendo de este punto, a partir de los 100 epochs más o menos, la red empezaba a tener un overfitting considerable, por lo que se empezó a experimentar con los diferentes valores de estos parámetros resultando en algo aproximado a la tabla a continuación.

Para interpretar esta tabla hay que tener en cuenta un par de cosas. La primera y mas importante es que por muchas veces que se ejecute el método para crear un modelo, al ser estos de naturaleza estocástica, no siempre va a devolver los mismos resultados. Sin embargo, cuanto mejor sea el modelo, mas se parecerán unos a otros, por lo que a efectos de hacernos una idea general vamos a suponer que estos datos son una representación suficientemente bien aproximada.

Por otro lado y debido al anterior, podría haberse representado el overfit teniendo en cuenta el epoch en el cual la validación empezara a torcerse o en el punto en el que las perdidas del entrenamiento y la validación comenzaran a divergir (cuanto mas cerca estén de converger, mas estables son los resultados del modelo entrenado) y considerando lo que le sobrase desde ese punto. Sin embargo, a efectos de hacerlo mas intuitivo se ha preferido realizar una batería de pruebas después de cada entrenamiento y observar en función de los resultados devueltos si devuelve resultados coherentes o no (cuanto mas overfitting, menos coherentes y viceversa). La batería de pruebas ha consistido en:

- Varias veces la palabras comando (cuantos mas positivos, mejor)

Epochs	LR	M	Precisión	Validación	Overfit?
100	0.01	0	0.9380	0.6372	C
	0.01	0.1	0.9225	0.7928	C
	0.01	0.2	0.9884	0.8378	C
	0.01	0.3	0.9884	0.7297	B
200	0.01	0	1	0.8649	C
	0.01	0.1	1	0.81	C
	0.01	0.2	1	0.8018	C
	0.01	0.3	0.9961	0.7027	C
100	0.001	0	0.8256	0.964	D
	0.001	0.1	0.8295	0.964	D
	0.001	0.2	0.8295	0.964	D
	0.001	0.3	0.8295	0.964	D
200	0.001	0	0.8295	0.964	D
	0.001	0.1	0.8450	0.8018	C
	0.001	0.2	0.8798	0.9640	D
	0.001	0.3	0.8566	0.9369	C
300	0.001	0	0.8295	0.964	C
	0.001	0.1	0.8992	0.8288	B
	0.001	0.2	0.8905	0.8069	B
	0.001	0.3	0.9031	0.7477	A

Tabla A.1: Conjunto de valores obtenido mediante sesiones de entrenamiento con los parámetros especificados

- Varias palabras que en ningún caso deberían confundirse (cuantos mas positivos, peor)
- Varias palabras que en algún caso podrían confundirse (cuantos mas positivos, peor, pero no tanto como el punto anterior)

En función de eso se les ha asignado cuatro etiquetas:

- A : Los positivos son mayoritariamente ciertos, los falsos positivos escasos y exclusivamente en palabras difíciles
- B : Los positivos son mayoritariamente ciertos, los falsos positivos ocurren y en palabras no necesariamente difíciles
- C : Los positivos se dan con bastante aleatoriedad.
- D : Prácticamente siempre positivo o negativo debido a demasiado underfitting o overfitting.

Se han excluido mas epochs con lr=0.01 debido a que las muestras actuales ya entraban en overfitting. Asi mismo, el numero de epochs necesario para alcanzar un resultado valido con lr=0.0001 era tan elevado, que no se ha tenido en consideración para esta experimentación. Por otro lado, se ha inferido de los experimentos que los mejores resultados se obtienen cuando la precisión de entrenamiento es superior a la de validación. Sin embargo, parece que si la de entrenamiento no es lo suficientemente alta (superior a 0.95), la de validación es relativamente irrelevante ya que no es coherente con la batería de pruebas.

Un referente que se ha demostrado relativamente bueno para calcular la precisión real en los modelos que daban resultados aceptables ha sido multiplicar ambas precisiones, ajustándose esa cifra entre el 75 y el 85 %.

En cualquier caso, todos estos números son meramente orientativos, ya que buscando otras combinaciones o cambiando otros parámetros como los mencionados en puntos anteriores de esta sección, podrían darse resultados radicalmente diferentes. Sin ir mas lejos, si bien se han usado esos numeros para mantener una estructura en las pruebas, los mejores resultados se han obtenido con 325/350 epochs con un lr=0.001 y un momentum=0.3.

B. ANEXO

Código

B.1. Recorder

```
# coding: utf-8
import shutil
from pathlib import Path
import numpy as np
# np.set_printoptions(threshold='inf')
import sounddevice as sd
import scipy
from scipy.io import wavfile
from scipy import signal
import python_speech_features as psf
import soundfile as sf
import pydub
import ffmpeg
import matplotlib.pyplot as plt
import random

#Area de trabajo
area="muestras"
```

```
#Zona de volcado
zona="muestras2"


def grabar(path):

    print("Grabando")
    duration = 2
    if(area=="muestras"):
        fs = 16000
    else:
        fs = 44100
    rec = sd.rec(duration * fs, samplerate=fs,channels=1, dtype='int16')
    sd.wait()

    return rec, fs, path

def guardar_audio(path,frecuencia,grab):
    path=path+".wav"
    wavfile.write(path,frecuencia,grab)
    melFeatures(path)
    print("Muestra guardada")

def espectrograma(wav):
    sample_rate, samples = wavfile.read(wav)
    frequencies, times, spectrogram =
    signal.spectrogram(samples, sample_rate, nperseg=320, noverlap=16)
    dBS = 10 * np.log10(spectrogram)  # convert to dB

    plt.subplot(2,1,1)
    plt.grid(True)
    plt.plot(samples)
```

```
    plt.subplot(2,1,2)
    plt.pcolormesh(times, frequencies, dBS,cmap='jet')
    #plt.imshow(dBS,aspect='auto',origin='lower',cmap='rainbow')
    plt.ylabel('Frecuencia [Hz]')
    plt.xlabel('Tiempo[s]')
    plt.colorbar()
    plt.show()

def melFeatures(wav):
    sample_rate, samples = wavfile.read(wav)
    if(area=="muestras"):
        features= psf.mfcc(samples,sample_rate,
        winfunc= lambda x:np.hamming(400))#,winlen=0.02,winstep=0.003,,numcep=13)
        #features=psf.mfcc(samples,sample_rate)
    else:
        features= psf.mfcc(samples, sample_rate,nfft=2048)
    dir= wav.split(".")[0]
    np.savetxt(dir,features)

def instaMel(grab,fs):
    if(area=="muestras"):
        return psf.mfcc(grab,fs,
        winfunc= lambda x:np.hamming(400))#,winlen=0.02,winstep=0.003,,numcep=13)
    else:
        return psf.mfcc(grab, fs,nfft=2048)

def gestor_muestras(path):
    file= open(path)
    lines= file.readlines()
    vh=int (lines[2])
    vm=int (lines[4])
    nh=int (lines[7])
    nm=int (lines[9])
    t=int(lines[11])
```

```
    file.close()

    return vh,vm,nh,nm,t

def mod_gestor(vh,vm,nh,nm,t,path):
    file=open(path,"w")
    file.write("Valida\nHombre\n"+str (vh)+"\nMujer\n"+str (vm)+
    "\nNo valida\nHombre\n"+str (nh)+"\nMujer\n"+str (nm)+"\nTest\n"+str (t))
    file.close()

def mod_muestra(path):

    vh,vm,nh,nm,t =gestor_muestras(zona+"/gestor_muestras.txt");
    s=wavfile.read(area+path +".wav")

    #obtener el numero independientemente del formato
    if not(len(path.split("vh"))<2):
        format=path.split("vh")[0]
        case="vh"
        n=int(path.split("vh")[1])
    elif not(len(path.split("vm"))<2):
        format=path.split("vm")[0]
        case="vm"
        n=int(path.split("vm")[1])
    elif not(len(path.split("nh"))<2):
        format=path.split("nh")[0]
        case="nh"
        n=int(path.split("nh")[1])
    elif not(len(path.split("nm"))<2):
        format=path.split("nm")[0]
        case="nm"
        n=int(path.split("nm")[1])

    index=n+(5*(n-1))
```

```
    shutil.copyfile(area+"/"+ format +case + str(n),
    zona+"/"+ format +case +str(index))

    if (area=="muestras"):
        m=instaMel(s[1],16000)
    else:
        m=instaMel(s[1],44100)

    for i in range(index+1,index+6):
        rand=random.uniform(0.90,1.1)
        mm=m*rand
        np.savetxt(zona+"/"+format +case +str(i),mm)
    if(case=="vh"):
        mod_gestor(vh+6,vm,nh,nm,t,zona+"/gestor_muestras.txt")
    elif(case=="vm"):
        mod_gestor(vh,vm+6,nh,nm,t,zona+"/gestor_muestras.txt")
    elif(case=="nh"):
        mod_gestor(vh,vm,nh+6,nm,t,zona+"/gestor_muestras.txt")
    if(case=="nm"):
        mod_gestor(vh,vm,nh,nm+6,t,zona+"/gestor_muestras.txt")

def procesar_audios(file):

    audio= pydub.AudioSegment.from_wav(file)

    audio_chunks = pydub.silence.split_on_silence(audio,
    # must be silent for at least(ms)
    min_silence_len=300,

    # consider it silent if quieter than dBFS
    silence_thresh=-30
    )

    for i, chunk in enumerate(audio_chunks):
```

```
        if(len(chunk)<2000):
            chunk2=rellenar_audio(chunk);
            out_file = "audios/procesados/chunk{0}.wav".format(i)
            print ("exporting " + out_file)
            chunk2.export(out_file, format="wav")

def rellenar_audio(chunk):
    silence=2000-len(chunk)
    silence_audio= pydub.AudioSegment.silent(duration=(silence/2),frame_rate=16000)
    return silence_audio+chunk+silence_audio
```

B.2. Reconocimiento

```
import os
from pathlib import Path

import keras
from keras.models import Sequential
from keras.layers import Dense, Activation,Bidirectional,LSTM,
CuDNNLSTM,GRU,Dropout

import sounddevice as sd
import soundfile as sf

import numpy as np
# np.set_printoptions(threshold='inf')
import scipy
from scipy.io import wavfile
from scipy import signal
import python_speech_features as psf
import matplotlib.pyplot as plt
from pandas import DataFrame
```

```
from recorder import gestor_muestras as gest


def obtener_muestra(path):
    sample= np.loadtxt(path)

    return sample

def crear_modelo():

    ################################################## Opcion LSTM

    #Modificar para probar distintos modelos
    hidden_size=100
    epochs=325
    learning=0.001
    decay_r=learning /epochs
    moment=0.3
    activation="tanh"

    if(activation=="tanh"):
        valida=1
        novalida=0
    elif(activation=="sigmoid"):
        valida=1
        novalida=0

    print("Creando Modelo...")
    model= Sequential()
    #model.add(Bidirectional(CuDNNLSTM(hidden_size,input_shape=[199,13],return_se
    model.add(CuDNNLSTM(hidden_size,input_shape=[199,13],return_sequences=True))
    #model.add(GRU(hidden_size,input_shape=[199,13],return_sequences=True))
    #model.add(LSTM(hidden_size,input_shape=[199,13],return_sequences=True))
    #model.add(Dropout(0.2))
```

```
#model.add(LSTM(hidden_size,input_shape=[199,13],return_sequences=True))
model.add(CuDNNLSTM(hidden_size,input_shape=[199,13],return_sequences=True))
#model.add(Dropout(0.2))
#model.add(LSTM(hidden_size,input_shape=[199,13],return_sequences=True))
model.add(CuDNNLSTM(hidden_size,input_shape=[199,13],return_sequences=True))
#model.add(Dropout(0.2))
#model.add(Bidirectional(CuDNNLSTM(hidden_size,input_shape=[199,13])))
model.add(CuDNNLSTM(hidden_size,input_shape=[199,13]))
#model.add(LSTM(hidden_size,input_shape=[199,13]))
#model.add(Dropout(0.2))
model.add(Dense(1,activation=activation))

opt=keras.optimizers.sgd(lr=learning,momentum=moment,decay=decay_r)
#opt=keras.optimizers.adam(lr=0.0001)
model.compile(opt,
              loss='mse',
              metrics=['accuracy'])

##########################################################training samples
print("Obteniendo muestras disponibles")
grupo="muestras/"
inputs=[]
results=[]

vh,vm,nh,nm,_=gest(grupo+"gestor_muestras.txt")

for i in range(vh+1):
    if(i>0):
        inputs.append(obtener_muestra(grupo+"validas/vh"+str(i)))
        #if(i==((i*6)+1) or i==1):
        results.append(valida)  #Ponderacion de las muestras reales
        #else:
        #    results.append(0.75)    #Ponderacion de las muestras artificiales
for i in range(vm+1):
```

```
        if(i>0):
            inputs.append(obtener_muestra(grupo+"validas/vm"+str(i)))
            #if(i==((i*6)+1) or i==1):
            results.append(valida)  #Ponderacion de las muestras reales
            #else:
            #    results.append(0.75)    #Ponderacion de las muestras artificiale
    for i in range(nh+1):
        if(i>0):
            inputs.append(obtener_muestra(grupo+"novalidas/nh"+str(i)))
            results.append(novalida)
    for i in range(nm+1):
        if(i>0):
            inputs.append(obtener_muestra(grupo+"novalidas/nm"+str(i)))
            results.append(novalida)

    ##########################################################training
    print("Entrenando el modelo")
    inputs=np.array(inputs)
    results=np.array(results)
    history=model.fit(inputs,results,validation_split=0.3,epochs=epochs)

    ##########################################################validation


    ps=obtener_muestra("muestras/test/test1")
    pn=obtener_muestra("muestras/test/test2")
    print("Prueba comando", model.predict(np.expand_dims(ps,axis=0)))
    print("Prueba no comando", model.predict(np.expand_dims(pn,axis=0)))

    # stored history
    train = DataFrame()
    val = DataFrame()
    train[str(i)] = history.history['loss']
    val[str(i)] = history.history['val_loss']
```

```
    # plot train and validation loss across multiple runs
    plt.plot(train, color='blue', label='train')
    plt.plot(val, color='orange', label='validation')
    plt.title('model train vs validation loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper right')
    plt.show()

    model.save("modelo_comando")
    print("Modelo guardado")
    #tfjs.converters.save_keras_model(model, "modelotfjs")
    return model;

def entrenar_modelo(path,correct):
    print("Procediendo a entrenar modelo...")

    if (not Path("modelo_comando").is_file()):
        print("No existe modelo")
        crear_modelo()
        entrenar_modelo(path)
    else:
        model=keras.models.load_model("modelo_comando");

        while not (Path("muestras/"+path).is_file()):
            print("El archivo indicado no existe, introduzca uno valido")
            path=input()
        train=obtener_muestra("muestras/"+path)
        model.train_on_batch(np.expand_dims(train,axis=0),[correct])
        model.save("modelo_comando")
        print("Modelo entrenado")

def obtener_prediccion(path):
```

```
    model=keras.models.load_model("modelo_comando")
    print("Procesando prediccion...")

    candidate= obtener_muestra("muestras/"+path)
    result= model.predict(np.expand_dims(candidate,axis=0))

    if (result[0] >=0.6) :
        print("COMANDO ", result[0])
    else:
        print("NO COMANDO ", result[0])

def insta_pred(mel):
    model=keras.models.load_model("modelo_comando")
    print("Procesando prediccion...")

    result= model.predict(np.expand_dims(mel,axis=0))
    if (result[0] >=0.6) :
        print("COMANDO ", result[0])
    else:
        print("NO COMANDO ", result[0])

    return model
```

B.3. Main

```
import os
from pathlib import Path

import reconocimiento as recon
import recorder as rec
import menus as ui
import re
```

```
import keras
import numpy as np

done=False
model=None

#Area de trabajo
area="muestras"
#area="muestras2"
#area="muestras44"
#area="muestras442"

def pred(model,sample):
    print("Procesando prediccion...")

    result= model.predict(np.expand_dims(sample,axis=0))
    if result[0] >=0.6 :
        print("COMANDO ", result[0])
    else:
        print("NO COMANDO ", result[0])


print()
print("Herramienta de gestion del modelo ASR")
opt= ui.Menu()

while (not done):
     ### Nueva Muestra
    if (opt==1):
        grab,fs,_=rec.grabar(None) ###Grabacion

        ###Datos de la muestra
        print("¿Es una muestra valida?(s/n)")
        vn=input()
```

```
        while not re.match("[sSnN]$",vn):
            print("Introduzca una respuesta valida (s/n)")
            vn=input()
        print("¿Es una muestra de hombre o mujer?(h/m)")
        hm=input()
        while not re.match("[hHmM]$",hm):
            print("Introduzca una respuesta valida (h/m)")
            hm=input()

        ###comprobacion de muestras existentes para establecer id de la nueva
        vh,vm,nh,nm,t =rec.gestor_muestras(area+"/gestor_muestras.txt");
        if(re.match("[sS]",vn) and re.match("[hH]",hm)):
            path=area+"/validas/vh"+str(vh+1)
            rec.mod_gestor(vh+1,vm,nh,nm,t,area+"/gestor_muestras.txt")
        elif(re.match("[sS]",vn) and re.match("[mM]",hm)):
            path=area+"/validas/vm"+str(vm+1)
            rec.mod_gestor(vh,vm+1,nh,nm,t,area+"/gestor_muestras.txt")
        elif(re.match("[nN]",vn) and re.match("[hH]",hm)):
            path=area+"/novalidas/nh"+str(nh+1)
            rec.mod_gestor(vh,vm,nh+1,nm,t,area+"/gestor_muestras.txt")
        elif(re.match("[nN]",vn) and re.match("[mM]",hm)):
            path=area+"/novalidas/nm"+str(nm+1)
            rec.mod_gestor(vh,vm,nh,nm+1,t,area+"/gestor_muestras.txt")

        rec.guardar_audio(path,fs,grab)
        opt= ui.Menu()

    ### Crear Modelo
    elif(opt==2):
        model=recon.crear_modelo()
        opt= ui.Menu()

    #### Cargar modelo
    elif(opt==3):
```

```
        print("¿Quiere cargar el ultimo modelo disponible?")
        sn=input()
        while not re.match("[sSnN]$",sn):
            print("Introduzca una respuesta valida (s/n)")
            sn=input()
        if(re.match("[sS]",sn)):
            print("Cargando modelo")
            model=keras.models.load_model("modelo_comando")
            print("Modelo cargado: modelo_comando")
        else:
            print("Introduzca el nombre de modelo que desee")
            md=input()
            while not (Path("modelosFuncionales/"+md).is_file()):
                print("Introduzca un modelo valido")
                md=input()
            print("Cargando modelo")
            model=keras.models.load_model("modelosFuncionales/"+md)
            print("Modelo cargado: "+md)

        opt= ui.Menu()

    ### Realizar Prediccion
    elif(opt==4):
        print("Introduzca una muestra valida")

        muestra=input()
        while not (Path("muestras/"+muestra).is_file()):
            print("El archivo indicado no existe, introduzca uno valido")
            muestra=input()
        if(model==None):
            recon.obtener_prediccion(muestra)
        else:
            candidate=recon.obtener_muestra("muestras/"+muestra)
            pred(model,candidate)
```

```
        opt= ui.Menu()

    ### Ejecutar Prediccion
    elif(opt==5):
        print("Reconocimiento de una palabra")

        if(model==None):
            grab,fs,_=rec.grabar(None)
            print("Procesando")
            sample=rec.instaMel(grab,fs)
            model=recon.insta_pred(sample)
        else:
            grab,fs,_=rec.grabar(None)
            print("Procesando")
            sample=rec.instaMel(grab,fs)
            pred(model,sample)
        opt= ui.Menu()

    ###Opcion oculta: Ampliar muestras
    elif(opt==8):
        print("¿Seguro que quiere ampliar las muestras actuales?")
        yn=input()
        while not re.match("[sSnN]$",yn):
           print("Introduzca una respuesta valida (s/n)")
           yn=input()
        if(re.match("[nN]$",yn)):
            done=True
            break

        print("Procediendo a multiplicar muestras existentes")

        vh,vm,nh,nm,t =rec.gestor_muestras(area+"/gestor_muestras.txt");
        for i in range(vh+1):
            if(i>0):
```

```
                rec.mod_muestra("/validas/vh"+str(i))
        for i in range(vm+1):
            if(i>0):
                rec.mod_muestra("/validas/vm"+str(i))
        for i in range(nh+1):
            if(i>0):
                rec.mod_muestra("/novalidas/nh"+str(i))
        for i in range(nm+1):
            if(i>0):
                rec.mod_muestra("/novalidas/nm"+str(i))

        print("Muestras artificiales generadas correctamente")
        opt= ui.Menu()

    ###Opcion oculta: Muestras test
    elif(opt==9):
        grab,fs,_=rec.grabar(None) ###Grabacion

        ###comprobacion de muestras existentes para establecer id de la nueva
        vh,vm,nh,nm,t =rec.gestor_muestras("muestras/gestor_muestras.txt");
        path="muestras/test/test"+str(t+1)
        rec.mod_gestor(vh,vm,nh,nm,t+1,"muestras/gestor_muestras.txt")

        rec.guardar_audio(path,fs,grab)
        opt= ui.Menu()

    ###Opcion oculta: Procesar audios
    elif(opt=='p'):
        print("Introduzca un audio valido")

        muestra=input()
        while not (Path("audios/procesar/"+muestra+".wav").is_file()):
            print("El archivo indicado no existe, introduzca uno valido")
            muestra=input()
```

```
        rec.procesar_audios("audios/procesar/"+muestra+".wav")

        opt= ui.Menu()

    ###Opcion oculta: Cepstrales de los audios externos
    elif(opt=='c'):
        print("Obteniendo cepstrales de las muestras actuales")

        vh,vm,nh,nm,t=rec.gestor_muestras(area+"/gestor_muestras.txt")

        for i in range(vh+1):
            if(i>0):
                rec.melFeatures(area+"/validas/vh"+str(i)+".wav")
        for i in range(vm+1):
            if(i>0):
                rec.melFeatures(area+"/validas/vm"+str(i)+".wav")
        for i in range(nh+1):
            if(i>0):
                rec.melFeatures(area+"/novalidas/nh"+str(i)+".wav")
        for i in range(nm+1):
            if(i>0):
                rec.melFeatures(area+"/novalidas/nm"+str(i)+".wav")
        for i in range(t+1):
            if(i>0):
                rec.melFeatures(area+"/test/test"+str(i)+".wav")

        print("Cepstrales obtenidos")
        opt=ui.Menu()

    #####Opcion oculta: espectrogramas
    elif(opt=='e'):
        print("Obteniendo espectrograma")
```

```
        rec.espectrograma("muestras/validas/vm1.wav")

        print("Espectrograma obtenido")
        opt=ui.Menu()

    #####Opcion oculta: resumen
    elif(opt=='s'):
        print("\n\nResumen del modelo actual:")

        model.summary()

        opt=ui.Menu()

    elif(opt==0):
        done=True
```

B.4. Menus

```
import keyboard

def Menu():
    done=False
    print()
    print()
    print("Menu de opciones principales:")
    print("\t1)Grabar nueva muestra")
    print("\t2)Crear modelo")
    print("\t3)Cargar modelo")
    print("\t4)Predecir muestra")
    print("\t5)Ejecutar programa")
    print("\t0)Salir")

    while(not done):
```

```
k=keyboard.read_key()

if(k=='1'):
    done=True
    return 1
elif(k=='2'):
    done=True
    return 2
elif(k=='3'):
    done=True
    return 3
elif(k=='4'):
    done=True
    return 4
elif(k=='5'):
    done=True
    return 5
elif(k=='8'):
    done=True
    return 8
elif(k=='9'):
    done=True
    return 9
elif(k=='p'):
    return 'p'
    done=True
elif(k=='c'):
    return 'c'
    done=True
elif(k=='e'):
    return 'e'
    done=True
elif(k=='s'):
    return 's'
```

```
            done=True
        elif(k=='0'):
            done=True
            return 0
```

Bibliografía

[1] W. Zhao, R. Chellappa, R. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM Comp. Surv. Vol. 36 No.4*, pp. 399–458, 2003. https://www.cs.ucf.edu/~dcm/Teaching/COT4810-Spring2011/Literature/FaceRecognition.pdf.

[2] W. Bledsoe, "The model method in facial recognition," *Tech. rep. PRI:15*, 1964.

[3] L. Sirovich and M. Kirby, "Low-dimensional procedure for the characterization of human face," *J. Opt. Soc. Am*, 1987. http://www.face-rec.org/interesting-papers/General/ld.pdf.

[4] P. Belhumeur, J. Espanha, and D. Kriegman, "Eigenfaces vs. fisherfaces: Recognition using class specific linear projection," *IEEE Trans. Patt. Anal. Mach. Intell. 19*, 1997. http://www.dtic.mil/dtic/tr/fulltext/u2/1015508.pdf.

[5] K. Etemad and R. Chellappa, "Discriminantanalysis for recognition of human face images," *J. Opt. Soc. Am. A 14*, 1997. http://www.face-rec.org/Algorithms/LDA/discriminant-analysis-for-recognition.pdf.

[6] I. Masi, Y. Wu, T. Hassner, and P. Natarajan, "Deep face recognition: A survey," 2018. http://sibgrapi.sid.inpe.br/col/sid.inpe.br/sibgrapi/2018/09.10.22.41/doc/PID5564503.pdf.

[7] E. T. M. Gori, "A survey of hybrid ann/hmm models for automatic speech recognition," *Elseiver Science Neurocomputing 37*, pp. 91–126, 2001. http://www.informatik.uni-ulm.de/ni/staff/FSchwenker/lit/papers/2001-Trentin-NC.pdf.

[8] R. O. Duda, P. E. Hart, and D. G. Stork, "Pattern classification and scene analysis," 1973. http://web.mit.edu/~cocosci/Papers/PatternClassificationBayes.PDF.

[9] P. F. Brown, "The acoustic-modeling problem in automatic speech recognition," 1987. http://www.dtic.mil/dtic/tr/fulltext/u2/a188529.pdf.

[10] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Procedings of the IEEE. Vol. 77 No. 2*, 1989. http://www.robots.ox.ac.uk:5000/~vgg/rg/papers/hmm.pdf.

[11] L. R. Rabiner, "An introduction to hidden markov models," *IEEE ASSP Mag 77*, 1986. http://www.stat.rice.edu/~yu/stat670/rabinerASSPmag.pdf.

[12] F. Johansen and M. Johnsen, "Global optimisation of hybrid hmm-architectures using global discriminative training," *ICSLP Vol 1 Yokohama*, 1994. https://www.isca-speech.org/archive/archive_papers/icslp_1994/i94_0239.pdf.

[13] D. Kershaw, T. Robinson, and M. Hochberg, "Context-dependant classes in a hybrid recurrent network-hmm speech recognition system," 1995. http://papers.nips.cc/paper/1039-context-dependent-classes-in-a-hybrid-recurrent-network-hmm-speech-recognit pdf.

[14] R. P. Lippmann, "Review of neural networks for speech recognition," *MIT Press*, 1989. https://www.mitpressjournals.org/doi/abs/10.1162/neco.1989.1.1.1.

[15] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," *ICML '06 Proceedings of the 23rd international conference on Machine learning*, pp. 369–376, 2006. http://www.cs.toronto.edu/~graves/icml_2006.pdf.

[16] F. Miyara, "La voz humana," *Universidad Nacional de Rosario*, 2001. https://sites.google.com/site/andreslucero2/lavozhumana.pdf.

[17] L. M. B. Tobón, "Caracterización de los indicadores acústicos de la voz de los estudiantes del programa licenciatura en música de la universidad de caldas," *El artista: revista de investigaciones en música y artes plásticas*, pp. 46–64, 2008. https://dialnet.unirioja.es/servlet/articulo?codigo=3091482.

[18] J. Sundberg, "The acoustics of the singing voice," *Scientific American*, pp. 82–91, 1977. http://www.music.mcgill.ca/~gary/courses/papers/Sundberg_SingingVoice_ScientificAmerican_1977.pdf.

[19] E. M. Celdrán, "En torno a las vocales del español: Análisis y reconocimiento," *Estudios de fonética experimental*, pp. 195–218, 1995. https://www.raco.cat/index.php/EFE/article/viewFile/144415/256847.

[20] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *Haskins Lab. Status Report on Speech Research*, 1980. http://www.dtic.mil/dtic/tr/fulltext/u2/a085320.pdf#page=201.

[21] D. Hebb, "The organization of behavior: A neurophysiological approach," *Wiley*, 1949.

[22] S. Haykin, "Neural networks: A comprehensive foundation," *Macmillan*, p. 2, 1994.

[23] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review Vol. 65 Nº 6*, 1958.

[24] J. S. Sepp Hochreiter, "Long short-term memory," *Neural Computation Vol. 9 Nº8*, pp. 1735–1780, 1997.