Facultad de Ciencias

Herramienta software para la detección de apraxias
(Apraxia detection software)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Santos Bringas Tejero

Director: Rafael Duque Medina

Co-Director: José Luis Montaña Arnaiz

Julio - 2017

Índice

Índice II

Índice de figuras IV

Índice de tablas VI

1. Introducción 4
1.1. Motivación 4
1.2. Contexto tecnológico 4
1.3. Objetivos del proyecto 5

2. Material y métodos 6
2.1. Tecnologías utilizadas 6
2.1.1. Python (versión 3.5.0) 6
2.1.2. Anaconda (versión 4.4.0) 6
2.1.3. Java (versión 1.8.0) 7
2.2. Librerías utilizadas 7
2.2.1. Keras (versión 2.0.2) 7
2.2.2. Theano (versión 0.9.0) 8
2.2.3. Numpy (versión 1.12.1) 8
2.2.4. Nvidia CUDA (versión 8.0) 9
2.2.5. Java Swing (versión 1.9.0) 9
2.2.6. VLC Media Player (versión 2.2.6) y vlcj (versión 3.8.0) . . 9
2.3. Herramientas utilizadas 10
2.3.1. Eclipse (versión neon ó 4.6.1) 10
2.3.2. WindowBuilder Editor (versión 1.9.0) 10
2.3.3. Tarjeta Gráfica Nvidia GTX 765M 10
2.4. Dataset utilizado 11
2.5. Metodología de trabajo 14

3. Diseño y construcción del sistema y la aplicación 16
3.1. Análisis de requisitos funcionales y no funcionales 16
3.1.1. Requisitos de alto nivel 16
3.1.2. Requisitos funcionales 16
3.1.3. Requisitos no funcionales 17
3.1.4. Definición de casos de uso 18
3.2. Desarrollo de la red: Explicación teórica e implementación 20
3.2.1. Las redes neuronales 20
3.2.2. Redes neuronales convolucionales: Clasificación de datos multidimensionales 24
3.2.3. Implementación del sistema de clasificación 26
3.3. Construccion de la app 31

4. Evaluación y pruebas **36**
4.1. Ejecuciónes de Keras 36
4.2. Pruebas de imagen 37
4.3. Pruebas de vídeo 38

5. Conclusiones y siguientes pasos **40**
5.1. Conclusiones del proyecto 40
5.2. Siguientes pasos a realizar 41

Índice de figuras

1. Python 6
2. Anaconda 6
3. Java 7
4. Keras 7
5. Theano 8
6. Numpy 8
7. CUDA 9
8. VLC 9
9. Eclipse 10
10. WindowBuilder 10
11. Clase 'a' 11
12. Clase 'b' 11
13. Clase 'c' 11
14. Clase 'd' 11
15. Clase 'e' 12
16. Clase 'f' 12
17. Clase 'g' 12
18. Clase 'h' 12
19. Clase 'i' 12
20. Clase 'j' 12
21. Clase 'boxing' 13
22. Clase 'handclapping' 13
23. Clase 'handwaving' 13
24. Clase 'jogging' 13
25. Clase 'running' 13
26. Clase 'walking' 13
27. Ciclo de vida iterativo 14
28. Ciclo de vida incremental 15
29. Red neuronal 21
30. Proceso de la capa convolucional 24
31. Modelo de red convolucional 25
32. Generador de imágenes 26
33. Carga de imagenes 26
34. Primeras capas de la red 27
35. Capas de clasificación de la red 27
36. Compilación del modelo 28
37. Entrenamiento de la red 28
38. Primeras capas de la red 29
39. Capas de clasificación de la red 30
40. Compilación del modelo para dos clases 30
41. Ventana principal de la aplicación 31
42. Lista de gestos disponibles 32
43. Ventana de selección de vídeo 33

44. Ventana principal con vídeo reproduciendo 34
45. Proceso de cálculo de la probabilidad 35
46. Muestra de una ejecución de Keras 36
47. Tasa de acierto en la prueba con imágenes 37
48. Tasa de acierto en la prueba con vídeos de varias clases 38
49. Tasa de acierto en la prueba con vídeos de dos clases 39

Índice de tablas

1. Requisitos funcionales 16
2. Caso de uso Cambiar gesto 18
3. Caso de uso Seleccionar vídeo 19
4. Caso de uso Clasificar vídeo 20

Agradecimientos

En primer lugar quiero agradecer a mi familia y sobre todo a mis padres y a mi hermana por apoyarme en todos estos años de carrera, animándome a continuar en las etapas malas y felicitándome por el esfuerzo invertido.

Agradecérselo a mi novia, la cual no solo me ha ayudado cuando lo necesitaba, sino que ha estado conmigo en los buenos y en los malos momentos.

A mis directores Rafael Duque y José Luis Montaña por ayudarme en la realización de este proyecto y por su atención y paciencia como profesores durante estos años. Agradecer también a todos los profesores que he tenido a lo largo de la carrera, que nos han transmitido su conocimiento tanto a mi como a mis compañeros.

Y por último a mis amigos de la universidad, los cuales han trabajado y reído conmigo estos cuatro años de clase, llevándome buenas amistades tras acabar esta etapa.

Resumen

El Alzheimer es una enfermedad neurológica que afecta a gran parte de la población. Esta enfermedad se caracteriza principalmente por provocar una pérdida de memoria en los afectados, aunque también provoca otros síntomas como la desorientación o cambios de humor. Por esto, es interesante tratar de detectar esta enfermedad en sus fases iniciales para poder tratar al paciente cuanto antes.

Uno de los síntomas de esta enfermedad, desde sus fases iniciales, es la apraxia, que produce una dificultad para realizar gestos voluntarios, los cuales una persona sin la enfermedad podría realizar sin muchos problemas. Es por esto que los médicos intentan encontrar apraxias en los pacientes para tratar de disgnosticar la enfermedad cuanto antes.

Para detectar apraxias los médicos proponen a los pacientes una serie de gestos que deberán imitar. Dependiendo de su ejecución y del tiempo que hayan empleado para hacerlo se decide si el gesto esta bien o mal. Tras hacer varios gestos se puede determinar si el paciente tiene principios de Alzheimer o no con mayor o menor probabilidad.

En algunos casos, no se sabe determinar si el gesto está bien o mal ejecutado. Es por ello que se quiere desarrollar una aplicación para ayudar a hacer esta clasificación, entre bueno o malo, para ayudar a resolver los casos más difíciles. Para ello, en este proyecto se ha propuesto grabar las acciones de los pacientes y después clasificarlos mediante una aplicación software que se creará con este fin.

Para hacer la predicción se utilizarán redes neuronales, más concretamente una variante de éstas, las redes neuronales convolucionales, que están diseñadas para tratar con datos multidimensionales como los vídeos de este problema. Con todo esto, se construirá una aplicación de escritorio para ayudar a encontrar apraxias en los pacientes.

Palabras clave: redes neuronales artificiales, redes neuronales convolucionales, enfermedad de Alzheimer, aplicación de escritorio.

Abstract

Alzheimer is a neurological disease that affects a large part of the people. This disease is distinguished for causing memory loss issues on the affected people, but it causes some other symptoms like disorientation or mood swings. Because of this, it is interesting to detect this disease in its initial stages to treat the patient properly.

One of the symptoms of this disease, since its initial stages, is the apraxia, that causes some difficulties on the realization of voluntary gestures that a healthy person could do without trouble. For all of this medics try to find apraxias in the patients to diagnose the disease as soon as possible.

In order to detect apraxias doctors give the patients some gestures that they must imitate. Depending of their execution and the time spent doing the gesture medics can determine if the gesture is correct or not. After several gestures it can be determined whether the patient can be affected by Alzheimer or not and how likely it is.

In some cases, it is difficult to determine whether a gesture is well or bad executed. Because of this it is required to design an application to help to make this classification, between good or bad execution, to help to resolve the hardest cases. For doing it, this project suggests to film the gestures made by the patients and then classified with the software application created with this proposal.

In order to do the prediction artificial neural networks will be used, more specifically a variant of this nets, the convolutional neural networks. This networks are designed to work with multidimensional data like the videos described before. With all of this, a desktop app will be created and will contribute to help medics to find apraxias in the patients.

Keywords: artificial neural networks, convolutional neural networks, Alzheimer disease, desktop app.

1. Introducción

En este apartado se explica el problema a tratar, así como la solución que se ha buscado al mismo y es la que se ha desarrollado durante la realización del proyecto. También se detallan los objetivos a cumplir en el proyecto.

1.1. Motivación

El Alzheimer es una enfermedad neurodegenerativa que afecta a una gran parte de la población en edad avanzada [1]. En España, el 6 % de personas mayores de 65 años están afectadas y el 40 % de las personas mayores de 85 años padecen esta enfermedad [2].

Los síntomas de la enfermedad son muy diversos: el más conocido y principal es la pérdida de memoria, aunque también está acompañado de problemas de orientación, cambios bruscos de humor, problemas de lenguaje o pérdida de contacto con la realidad entre otros [3].

El problema de esta enfermedad es que cuando aparecen los primeros síntomas claros puede ser tarde para paliarla y tratarla, necesitando el paciente de cuidados cada vez más exhaustivos. Es por ello que es necesario detectar esta enfermedad en etapas tempranas.

Uno de los primeros síntomas de la enfermedad es la apraxia, un déficit a la hora de realizar gestos voluntarios. Por esto, para diagnosticar la enfermedad se suelen buscar apraxias en los pacientes. Para ello, los/as médicos/as actualmente someten a los pacientes a ciertas pruebas motoras: mediante una serie de gestos se puede detectar la apraxia, dependiendo de si es correcta su ejecución y del tiempo requerido.

Este problema fue propuesto por doctores del Hospital de Valdecilla de Santander. En algunos casos estos gestos producen dudas, ya que no pueden decidir si un paciente ha hecho un gesto correctamente o no para someterlo a pruebas más avanzadas y costosas. Por ello, se quiere desarrollar una aplicación para poder ayudar a los/as médicos/as a decidir en los casos más difíciles.

1.2. Contexto tecnológico

Tras plantearse este problema, se decidió hacer una aplicación que, a partir de vídeos grabados a los pacientes, decidiera si es correcta la ejecución del gesto o no. Por ello se exploraron las posibles soluciones, destacando desde un principio la de crear un sistema inteligente para la toma de decisiones, en concreto utilizando una red neuronal artificial.

Una red neuronal artificial es un sistema inteligente que, como su propio nombre indica, intenta imitar una red de neuronas del cerebro humano [4]. Como un cerebro, las redes neuronales artificiales están formadas por neuronas, que están interconectadas unas con otras para formar la red.

Estas redes pueden "aprender" como si de un cerebro humano se tratase. Esto se consigue entrenando la red neuronal artificial, dándole ejemplos de entradas y asignándole una salida a cada una de estas, de manera que la red sepa

si ha acertado o si ha fallado y poder corregir su error. Con ejemplos variados se puede conseguir una buena tasa de acierto y, de esta manera, el sistema resulta útil y confiable.

Una red neuronal artificial básica está diseñada para recibir, trabajar y devolver datos numéricos, por lo que se necesita utilizar una variación de estas: las redes neuronales convolucionales. Una red convolucional es capaz de recibir entradas muy diferentes, incluyendo texto, imágenes, audio e incluso vídeo. La estructura de una red convolucional es muy similar a la de una red neuronal básica, teniendo las neuronas interconectadas y cada una de ellas recibiendo una serie de entradas y dando una salida. La particularidad de estas redes es que pueden trabajar con datos multidimensionales, como son los casos mencionados. Es por esto que resulta muy útil para resolver el problema planteado.

En los apartados siguientes se definirán más concretamente estos sistemas inteligentes, así como su funcionamiento y su utilización para resolver el caso concreto del problema a tratar.

Por último, cabe destacar que prácticamente cualquier persona posee un computador personal hoy en día, más concretamente un hospital o un/a doctor/a, que dispondrá de uno o más de uno para su utilización. Es por ello que se ha desarrollado una aplicación de escritorio sencilla para que pueda realizar los análisis el/ella mismo/a de una manera rápida y sencilla.

El programa recibirá los vídeos de los pacientes, grabados con una cámara o programa externos, y devolverá un resultado, que podrá ser bueno o malo dependiendo de la ejecución del gesto, así como de la probabilidad con la que el programa ha determinado que ese gesto estaba bien o mal. Esto ayudará a detectar apraxias en los pacientes, sobre todo para aquellos casos en los que haya dudas.

1.3. Objetivos del proyecto

Como resumen a estos dos últimos apartados, se indican a continuación los objetivos concretos de este proyecto:

- Crear un sistema inteligente que sepa decidir, con una alta tasa de aciertos, si un vídeo pertenece a una clase o a otra (por ejemplo a "gesto bien" o a "gesto mal").

- Entrenar este sistema con vídeos de varios gestos diferentes que permita obtener varios resultados de un mismo paciente. Con esto, aumentamos la confianza del sistema, ya que se realizan diferentes pruebas.

- Desarrollar una aplicación de escritorio usable y sencilla que permita a los/as médicos/as aprovechar estos sistemas y obtener un resultado que lo ayude a detectar si el paciente tiene una apraxia.

2. Material y métodos

En este apartado se definen los materiales utilizados, incluyendo tecnologías, herramientas y librerías; y los métodos seguidos para la realización del proyecto. También se especifica el motivo de su elección y de su uso concreto en el proyecto.

2.1. Tecnologías utilizadas

A continuación se definen las distintas tecnologías utilizadas para la realización del proyecto, incluyendo los diferentes de lenguajes de programación en los que se ha desarrollado el código.

2.1.1. Python (versión 3.5.0)

Python [5] (logo en la Figura 1) es un lenguaje de programación de alto nivel multiparadigma, interpretado, multiplataforma y que usa variables dinámicas. Es uno de los lenguajes de programación más utilizados [6]. Destaca por su amplia variedad de librerías creadas para algoritmos matemáticos e inteligencia artificial, como es el caso en el que esta basado el trabajo. Por ello, resultó relativamente sencillo encontrar una librería (Keras) que cumplía con los requisitos especificados.

Figura 1: Python
Logo de Python

Es por esto mismo que Python se ha utilizado para hacer los programas principales, los que crean la red neuronal, la entrenan y también el programa que hace la clasificación del gesto y el cálculo de la probabilidad con la red neuronal ya entrenada. Todo esto se realiza con diversas librerías de código abierto que hay disponibles, destacando en este proyecto Keras, como ya se ha comentado. En el próximo apartado se detallan más concretamente estas librerías.

2.1.2. Anaconda (versión 4.4.0)

Figura 2: Anaconda
Logo de Anaconda

Anaconda [7] (logo en la Figura 2) es un entorno de trabajo de código abierto construido sobre Python y que dispone de muchos paquetes y librerías distintos con los que integrar diversos programas. Este entorno de trabajo, como ya se ha dicho, dispone de diversos paquetes y librerías de código abierto y mantenidos por una comunidad, los cuales además integran muchos softwares diferentes desarrollados por empresas colaboradoras, como Intel, Microsoft o Nvidia.

Para el desarrollo de este proyecto resultó necesaria su instalación, preferible sobre la básica de Python, ya que integra mejor algunas librerías de las que se habla en el capítulo correspondiente, sobre todo de Nvidia CUDA. Además,

gracias a la herramienta 'pip' de Python, la instalación de la ultima versión de estos paquetes es muy sencilla.

2.1.3. Java (versión 1.8.0)

Java [8] (logo en la Figura 3) es un lenguaje de programación de alto nivel orientado a objetos, compilado, proveniente del lenguaje C. Según diversos estudios y encuestas, es el lenguaje de programación más utilizado [6] gracias sobre todo a la gran cantidad de dispositivos que lo ejecutan, desde muchos navegadores y aplicaciones de escritorio hasta dispositivos móviles, con el sistema operativo Android.

Figura 3: Java
Logo de Java

En este proyecto se ha utilizado para construir la aplicación de escritorio que será la que utilicen los usuarios finales, es decir, los hospitales o, concretamente, los/as médicos/as y enfermeros/as. Gracias al entorno de desarrollo Eclipse y a diversas librerías y herramientas resulta muy sencillo crear una aplicación que sea usable e intutitiva, además de ser capaz de ejecutarse en prácticamente cualquier computadora.

2.2. Librerías utilizadas

2.2.1. Keras (versión 2.0.2)

Figura 4: Keras
Logo de Keras

Keras [9] (logo en la Figura 4) es una API de desarrollo de código abierto desarrollada en Python y orientada al aprendizaje automático y 'deep learning'. Es una herramienta diseñada para crear redes neuronales, desde algunas sencillas para hacer primeras experimentaciones hasta otras más potentes que permitan realizar aplicaciones reales, como por ejemplo la detección de objetos, animales, personas, etcétera en imágenes. Además, permite la integración con otras librerías de 'deep learning' como Theano o TensorFlow.

Esta es la biblioteca principal con la que se ha trabajado, ya que provee muchas funciones para crear redes neuronales potentes y para su aprendizaje a partir de un dataset adecuado, con un cierto número de muestras y, tras realizar pruebas, se alcanzaba una buena tasa de acierto, por lo cual el sistema resulta confiable. Además, tras construir el sistema y entrenarlo, resulta muy fácil y rápido cargar el modelo creado y los pesos de la red neuronal entrenada para hacer nuevas predicciones.

Como punto negativo, si se construye una red neuronal muy compleja y que utiliza datos muy grandes, como puede ser el caso de los vídeos que se utilizan en este proyecto, el proceso de entrenamiento resulta muy largo, llegando a

tardar varias horas (se detallará más adelante). La librería viene preparada para mejorar esto, aprovechando las ventajas del paralelismo y de la potencia de las tarjetas gráficas para realizar operaciones, aumentando la eficiencia del proceso y reduciendo el tiempo de entrenamiento. Esto se consigue gracias al entorno que proporciona Anaconda y al software de paralelización Nvidia CUDA, que permite aprovechar las tarjetas creadas por la misma empresa, Nvidia.

2.2.2. Theano (versión 0.9.0)

Theano [10] (logo en la Figura 5) es una librería de código abierto de Python que permite hacer cálculos complejos con datos multidimensionales, generalmente en forma de arrays, de una manera sencilla y eficiente. Esta librería está integrada con muchas funciones de otra librería matemática Numpy, que se describe en el siguiente apartado. Además, también permite la paralelización y el uso de las GPU (tarjetas gráficas) para aumentar la velocidad de procesamiento.

Figura 5: Theano
Logo de Theano

En el proyecto se utiliza como base para Keras con la cual poder realizar las operaciones de cálculo necesarias para entrenar a la red neuronal creada. Esta librería puede ser sustituida por TensorFlow, con un funcionamiento semejante y obteniendo resultados similares.

2.2.3. Numpy (versión 1.12.1)

Esta librería de código abierto [11] (logo en la Figura 6) es una de las más conocidas y utilizadas en Python, ya que provee de muchas funciones matemáticas y gestión de arrays multidimensionales, por lo que resulta esencial para muchos proyectos.

Figura 6: Numpy
Logo de Numpy

Como puede resultar obvio, para este proyecto también es necesaria esta librería ya que es indispensable para el funcionamiento de Keras, Theano, CUDA y también para la gestión de los datos de las imágenes y los vídeos del propio proyecto.

2.2.4. Nvidia CUDA (versión 8.0)

Figura 7: CUDA
Logo de CUDA

Esta librería [12] (logo en la Figura 7) permite el aprovechamiento óptimo de la capacidad de las GPU o tarjétas gráficas de Nvidia. Gracias a esta librería se pueden paralelizar muchos de los cálculos que debería realizar la CPU, aprovechando el hardware del computador para optimizar las operaciones y reducir considerablemente el tiempo empleado.

Como ya se ha mencionado, esta librería aprovecha la tarjeta gráfica de la que dispone el ordenador portátil en el que se ha desarrollado el proyecto para acelerar los cálculos. Esta librería se integra con el entorno Anaconda, como ya se ha dicho, y es totalmente compatible con el resto de librerías anteriormente descritas.

2.2.5. Java Swing (versión 1.9.0)

Esta librería [13] desarrollada para Java es una biblioteca de funciones gráficas para crear interfaces gráficas de usuario (en adelante GUI, del inglés Graphical User Interface). Se integra perfectamente con el entorno de desarrollo Eclipse, estando instalada de inicio en algunas versiones del mismo. Esta librería proporciona una gran variedad de componentes gráficos, como botones, campos de texto, y algunos más complejos como la integración de vídeos. Además, estos componentes son personalizables y su utilización es muy sencilla.

La librería se ha utilizado en el proyecto para hacer la aplicación final que usaran los usuarios y que, como ya se dijo en la sección de Java, puede ejecutarse en casi cualquier computador.

2.2.6. VLC Media Player (versión 2.2.6) y vlcj (versión 3.8.0)

Figura 8: VLC
Logo de VLC

VLC [14] (logo en la Figura 8) es un programa de reproducción de contenido multimedia gratuito bastante conocido y utilizado, pero en este caso se habla de las librerías de funciones de las que dispone. Con la integración para Java que proporciona la otra librería, vlcj, se puede aprovechar este reproductor para integrarlo en el desarrollo de GUI y conseguir un resultado más completo.

En el proyecto se ha aprovechado para mostrar el vídeo seleccionado a la hora de hacer los análisis. Se ha escogido este reproductor ya que es gratuito y su instalación es bastante sencilla.

2.3. Herramientas utilizadas

2.3.1. Eclipse (versión neon ó 4.6.1)

Eclipse [15] (Logo en la Figura 9) es el entorno de desarrollo más utilizado ya que es gratuito y permite el desarrollo con varios compiladores de diferentes lenguajes de programación, como C, C++ o Java. Además, su interfaz es muy intuitiva, permite gestionar los proyectos de una manera sencilla, dispone de varios plug-ins instalados y permite la instalación de otros desarrollados por la comunidad y de bibliotecas de funciones que sean necesarias para los proyectos. También dispone de herramientas de desarrollo de software comunes de estos entornos como resaltado de errores, un modo debug paso a paso, etcétera.

Figura 9: Eclipse
Logo de Eclipse

Para trabajar con Java y crear la aplicación de escritorio se ha utilizado este entorno, ya que es un entorno agradable e intuitivo, con facilidad para corregir errores y desarrollar código. Además, integra la siguiente herramienta gráfica, WindowBuilder Editor, con lo que es más sencillo crear la GUI.

2.3.2. WindowBuilder Editor (versión 1.9.0)

Figura 10: WindowBuilder
Logo de WindowBuilder

Esta herramienta [16] (logo en la Figura 10) de diseño gráfico de GUI se integra perfectamente con el entorno de desarrollo Eclipse antes mencionado. Permite hacer una previsualización de la interfaz diseñada por código y añadir y mover componentes con el uso del ratón, por lo que es mucho más sencillo e intuitivo crear las aplicaciones, ya que se puede distribuir mejor la interfaz que por código.

Junto con Java Swing se ha utilizado para crear la interfaz de usuario, sirviendo esta herramienta como guía y para un mejor desarrollo gráfico del proyecto por sus características para ordenar los componentes y el funcionamiento de cada uno de ellos.

2.3.3. Tarjeta Gráfica Nvidia GTX 765M

Esta tarjeta gráfica, creada por la empresa Nvidia, es la que, como ya se ha dicho, ha sido utilizada para acelerar los cálculos del entrenamiento de la red neuronal. Gracias a las librerías mencionadas en los anteriores apartados se ha conseguido aprovechar este hardware para la realización del proyecto.

2.4. Dataset utilizado

Para realizar las pruebas correspondientes al desarrollo del proyecto se utilizaron dos dataset encontrados en dos universidades diferentes y que tienen como finalidad la investigación, como es el caso de este proyecto. Estos dos dataset son meramente de prueba, para comprobar el funcionamiento del software y de la confiabilidad que se puede alcanzar con el sistema, pues el dataset final no será ninguno de estos, sino uno proporcionado por el Hospital de Valdecilla de Santander, ya que será quien utilice este software finalmente.

Los dos dataset escogidos, con una breve explicación y una foto que define cada clase, son los siguientes:

- NUS hand posture dataset II [17]: Este dataset está extraido de la Universidad Nacional de Singapur o NUS. Este dataset tiene dos variantes, con ruido humano y sin él. Es un dataset de imágenes de manos de diferentes personas haciendo 10 algunas letras del alfabeto de sordomudos, no correspondiendo las letras asignadas a la letra gestualizada.

  En el dataset de las manos sin ruido humano hay disponibles 2000 imágenes de 160x120 píxeles divididas en partes iguales en, como ya se ha dicho, 10 clases diferentes. En las imágenes con ruido humano hay disponibles 750 imágenes de 320x240 píxeles divididas de la misma manera. En este último aparecen personas de fondo para hacer de 'ruido' y así comprobar que realmente el sistema es confiable en diferentes situaciones. Algunas imágenes de ejemplo, una por cada clase, son (Figuras 11 a 20):

Figura 11: Clase 'a'

Figura 12: Clase 'b'

Figura 13: Clase 'c'

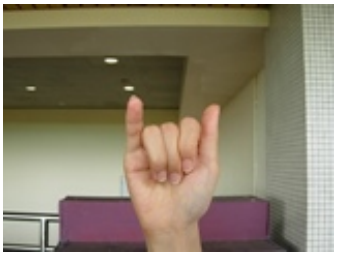

Figura 14: Clase 'd'

Figura 15: Clase 'e'

Figura 16: Clase 'f'

Figura 17: Clase 'g'

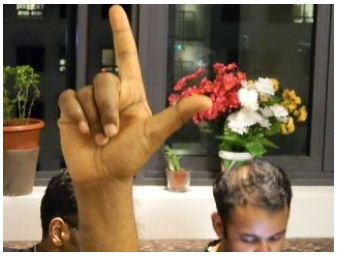

Figura 18: Clase 'h'

Figura 19: Clase 'i'

Figura 20: Clase 'j'

- Recognition of human actions, Action Database [18]: Este dataset está extraido de la Universidad de Estocolmo o SU. El dataset está compuesto de varios vídeos de personas realizando diversos ejercicios y están grabadas de cuerpo completo. Este dataset tiene 600 vídeos, todos ellos en blanco y negro, divididos igualmente en 6 clases diferentes. Algunos fotogramas de ejemplo, uno por cada clase, son (Figuras 21 a 26):

Figura 21: Clase 'boxing'

Figura 22: Clase 'handclapping'

Figura 23: Clase 'handwaving'

Figura 24: Clase 'jogging'

Figura 25: Clase 'running'

Figura 26: Clase 'walking'

2.5. Metodología de trabajo

Durante el desarrollo de este proyecto se ha utilizado una metodología iterativa e incremental [19], ya que era necesario tener una metodología ágil que permitiera desarrollar primeras versiones de una manera rápida para hacer pruebas y pivotar con los resultados, bien cambiando el software elegido inicialmente o bien continuar con esa base. Esto era necesario sobre todo para la primera versión, ya que no se conocía el funcionamiento ni los resultados alcanzables con el software mencionado. A continuación se describen los dos modelos utilizados en el proyecto.

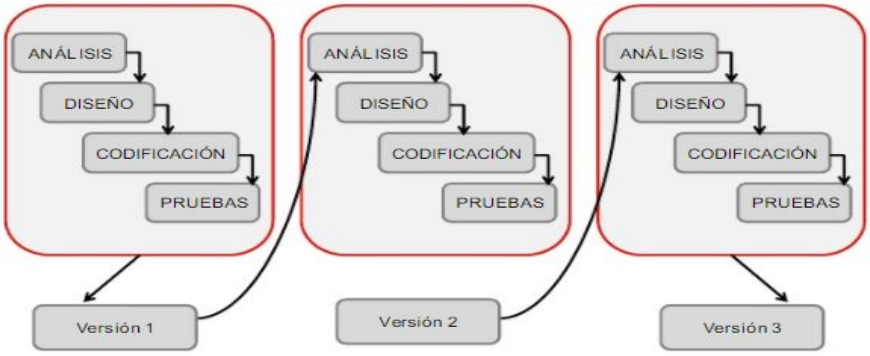

Figura 27: Ciclo de vida iterativo
Fuente: [19]

Como indica la Figura 27, la metodología iterativa tiene un ciclo de vida en cascada y, como su propio nombre indica, consiste en hacer varias iteraciones mejorando el sistema en cada una de ellas para cumplir los requisitos especificados por el cliente. Al final de cada iteración y tras las pruebas correspondientes se le muestra al cliente.

En este caso, se le mostraron a los médicos de Valdecilla los resultados obtenidos en cada iteración. Después de recibir más información por su parte, se consiguió avanzar progresivamente y corregir los fallos o imperfecciones que tenía cada versión.

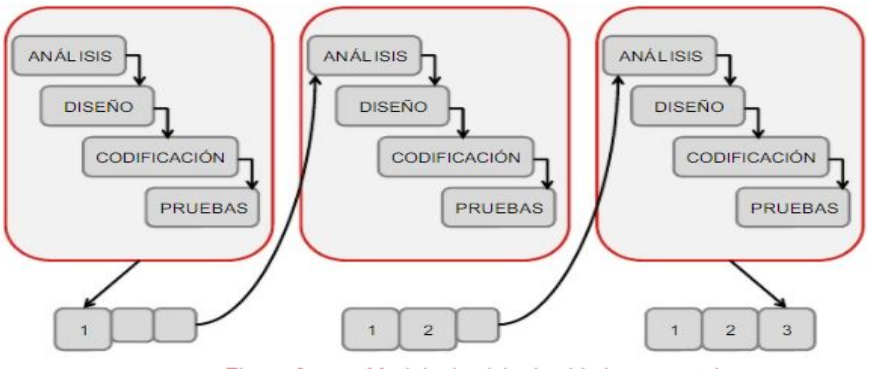

Figura 28: Ciclo de vida incremental
Fuente: [19]

La metodología incremental es similar a (o una evolución de) la iterativa, teniendo también varias iteraciones, como se aprecia en la Figura 28. En este caso, en cada iteración se incrementa, como su propio nombre indica, la funcionalidad de la versión anterior, añadiendo nuevas funcionalidades que cumplan nuevos requisitos. Cada versión es entregada al cliente para que pueda comprobar una muestra de lo que será el producto final, pudiendo modificar los requisitos o dando otros nuevos a partir de cada versión.

En este proyecto, los doctores de Valdecilla pudieron comprobar la eficacia de las dos muestras creadas, primero una con un dataset de imágenes y después con uno de vídeos. De esta manera se generaron nuevos requisitos y se modificaron algunos. Un ejemplo concreto de esto es: inicialmente, se pensaba que se iba a querer distinguir entre varios gestos, es decir, decir si el gesto realizado era el gesto A, el gesto B u otro de los disponibles. Tras alguna reunión con los doctores se dijo que la clasificación sería binaria, solo con dos clases, determinando si el gesto esta bien o mal y con varios gestos distintos.

Con estas dos metodologías de trabajo se pudo avanzar correctamente con el desarrollo del proyecto, incluyendo a los doctores de Valdecilla en el proceso para obtener su opinión en varias ocasiones. Con ello, se asegura que el producto final esté más cerca de lo que ellos esperan y así ayudarles a detectar apraxias en los pacientes, ayudando así al hospital, a los/as médicos/as y a los propios pacientes.

3. Diseño y construcción del sistema y la aplicación

En el siguiente apartado se detalla todo lo relacionado con la investigación previa, el análisis de requisitos y el desarrollo del propio proyecto. También se comentará cuales de los materiales mencionados se utilizaron en cada momento de las etapas del proyecto.

3.1. Análisis de requisitos funcionales y no funcionales

En esta sección se describen los requisitos que se definieron en las primeras fases del proyecto. Estos se definieron en diferentes reuniones con neurólogos de Valdecilla. Debido a que es un trabajo orientado más a investigación y a resolver un problema en concreto, los requisitos de la aplicación final son pocos comparados con los de otras aplicaciónes.

3.1.1. Requisitos de alto nivel

Los únicos usuarios, inicialmente, de la aplicación serán los/as médicos/as y auxiliares del Hospital de Valdecilla de Santander. La finalidad de la aplicación es que se de una 'puntuación' a un gesto. Otra manera de interpretar esto es que se clasifique el gesto como 'bueno' o 'malo' con una cierta probabilidad. Es por ello que el único requisito de alto nivel que tiene la aplicación es:

- Calcular puntuación de vídeo

Este requisito, no llegando a ser siquiera un requisito "completo" como tal, ya que solamente se define una parte de un requisito de alto nivel, pues estas puntuaciones no se guardarán ni editarán de ningún modo. El objetivo es que se puedan evaluar muchos vídeos de diferentes pacientes y que el usuario lo anote, bien en otro programa o bien en un documento para su posterior consulta.

3.1.2. Requisitos funcionales

A continuación, en la Tabla 1, se definen el único requisito funcional de la aplicación. Este requisito va relacionado con el requisito de alto nivel del apartado anterior.

Identificador	**Descripción**
RF01	El usuario puede introducir un vídeo de un gesto realizado por un paciente y obtener si el gesto está bien o mal y con qué probabilidad

Tabla 1: Requisitos funcionales

3.1.3. Requisitos no funcionales

A continuación se describen los requisitos no funcionales que debe de tener la aplicación para que su funcionamiento sea el esperado y satisfaga al usuario [20].

- **Adecuación funcional**

  La aplicación debe funcionar adecuadamente para cumplir los requisitos especificados en las reuniones con los neurólogos de Valdecilla. Para ello debe de retornar un resultado correcto con una buena tasa de acierto, ya que para ello se ha entrenado el sistema inteligente. Es posible que en algún caso aislado difiera del resultado correcto, pero este se aislará, con vídeos de otros gestos del mismo paciente. Además, debe de retornar el mismo resultado para el mismo vídeo, de lo contrario la aplicación no funcionará adecuadamente.

- **Eficiencia**

  Como ya se ha dicho, los usuarios deben poder analizar varios vídeos de los pacientes por lo que es necesario que el análisis no dure demasiado, no tomando mucho tiempo en analizar cada uno de ellos. De lo contrario, la aplicación sería pesada de utilizar.

- **Usabilidad**

  La aplicación debe de ser usable por cualquier tipo de usuario. Es por ello que debe de tener una interfaz sencilla y con la suficiente claridad para su uso, con un tiempo de aprendizaje corto permitiendo así su pronta incorporación a la rutina de diagnóstico de pacientes con Alzheimer.

- **Fiabilidad**

  Ligado al apartado de eficiencia, el sistema debe funcionar correctamente, estando preparado para los posibles fallos que puedan surgir y permitiendo al usuario continuar con la ejecución normal sin que le suponga mucho tiempo, como ya se comentó en el requisito mencionado.

- **Mantenibilidad**

  En un inicio, la aplicación contendrá varios sistemas entrenados, uno por cada uno de los gestos que realizarán los pacientes. Más adelante, es posible que algunos de estos gestos se modifiquen, se añadan o eliminen y/o se mejoren las redes neuronales, incrementando el tamaño del dataset utilizado para el entrenamiento de estas.

- **Portabilidad**

  La aplicación debe de ser capaz de ejecutarse en distintos computadores, ya que se va a utilizar para ayudar a hacer diagnósticos a pacientes en los hospitales. Por ello, debe de poder funcionar de igual manera en diversos computadores diferentes.

3.1.4. Definición de casos de uso

En este apartado se detalla el funcionamiento que debe tener la aplicación durante su ejecución y la interacción del usuario con ella. Los casos de uso definidos en las Tablas 2, 3 y 4 son los que se desglosan del requisito funcional del apartado anterior.

Nombre	Cambiar gesto
Actor principal	Usuario
Actores secundarios	
Descripción	El usuario cambia el gesto para calcular
Evento de activación	El usuario pulsa en la lista de gestos disponibles
Precondición	
Garantía si éxito	El sistema guarda el gesto seleccionado
Garantías mínimas	Los datos del sistema se mantienen consistentes
Escenario principal	1. El sistema muestra al usuario la lista completa de gestos disponibles. 2. El usuario selecciona un gesto de la lista. 3. El sistema vuelve a la ventana principal y guarda el gesto seleccionado.
Extensiones	
Comentarios	

Tabla 2: Caso de uso Cambiar gesto

Nombre	Seleccionar vídeo
Actor principal	Usuario
Actores secundarios	
Descripción	El usuario selecciona un vídeo
Evento de activación	El usuario pulsa el botón 'Seleccionar vídeo'
Precondición	
Garantía si éxito	El sistema almacena el vídeo para su análisis
Garantías mínimas	Los datos del sistema y del usuario se mantienen consistentes
Escenario principal	1. El sistema muestra al usuario el sistema de archivos en una nueva ventana para seleccionar un vídeo. 2. El usuario busca y selecciona el vídeo entre los archivos. 3. El usuario pulsa el botón 'Aceptar'. 4. El sistema vuelve a la ventana principal, guarda la dirección del vídeo e inicia la reproducción del mismo en bucle para que lo visualice el usuario.
Extensiones	2a. El usuario pulsa el botón 'Cancelar'. 1. El sistema vuelve a la ventana principal. 4a. El sistema no encuentra el vídeo en los archivos. 1. El sistema muestra un mensaje de error.
Comentarios	

Tabla 3: Caso de uso Seleccionar vídeo

Nombre	Clasificar vídeo
Actor principal	Usuario
Actores secundarios	
Descripción	El sistema clasifica el vídeo y calcula la probabilidad
Evento de activación	El usuario pulsa el botón 'Calcular probabilidad'
Precondición	Hay un vídeo seleccionado
Garantía si éxito	El sistema retorna la clasificación y la probabilidad
Garantías mínimas	Los datos del sistema y del usuario se mantienen consistentes
Escenario principal	1. El sistema muestra al usuario en una nueva ventana una barra de carga y los pasos del proceso de cálculo. 2. El sistema termina el cálculo de la clasificación del vídeo seleccionado y muestra al usuario el resultado. 3. El usuario pulsa el botón 'Aceptar'. 4. El sistema vuelve a la ventana principal.
Extensiones	
Comentarios	

Tabla 4: Caso de uso Clasificar vídeo

3.2. Desarrollo de la red: Explicación teórica e implementación

Inicialmente, se requirió de una primera investigación teórica para hacer el diseño del sistema inteligente que haga la clasificación y la evaluación del gesto. Es por ello que se analizaron las redes neuronales y sus variantes, conocer su funcionamiento y, de esta manera, aplicarlo a la resolución del problema a resolver.

3.2.1. Las redes neuronales

Las redes neuronales artificiales (en adelante ANN, del inglés "Artificial Neural Networks") reciben su nombre de la base de las mismas, el cerebro humano [4]. Son sistemas inteligentes que tienen un diseño que imita al de una red neuronal cerebral y son utilizadas para hacer desde cálculos simples, como regresiones sencillas, hasta operaciones más avanzadas, como clasificaciones o clustering con cientos de variables diferentes.

Estas redes artificiales, como las cerebrales, están formadas por neuronas interconectadas entre si. Estas neuronas son las que realizan los cálculos progresivamente, pasando los resultados de unas a otras.

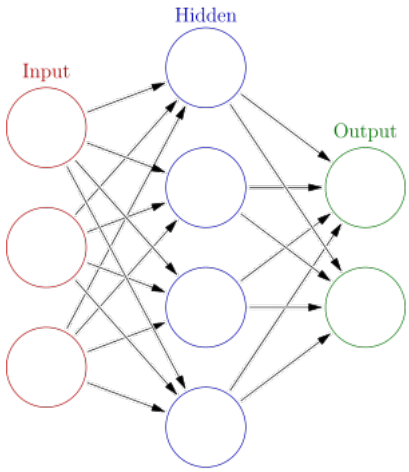

Figura 29: Red neuronal
Fuente Wikipedia [21]

Como se puede ver en la Figura 29, las neuronas están divididas y ordenadas en diferentes capas, siendo la salida de cada una de las neuronas (de izquierda a derecha) la entrada de las neuronas de la siguiente capa. El resultado de una neurona pasará a las neuronas de la capa siguiente que estén conectadas a ella. De este modo, los resultados se van propagando hasta el final, dando finalmente un resultado, que deberá ser interpretado y/o traducido.

Hay tres distintos tipos de capas:

- Capa de entrada o 'input layer': Es la capa de la red que contiene las neuronas que inician los cálculos, ya que son las que tienen los datos de entrada. Estas neuronas no tienen ninguna neurona previa conectada a ella, pues su entrada es la entrada de la propia ANN.

- Capas ocultas o 'hidden layer': Estas capas son todas las capas intermedias de la red. Como su nombre indica, son "desconocidas" para el exterior, ya que no se conoce su valor al ser las que realizan los cálculos intermedios y no tienen una salida directa de la ANN.

- Capa de salida o 'output layer': Esta capa es la final de la red. Es la capa que contiene las neuronas que dan el resultado de la red. Cada una de estas neuronas da un dato de salida de la ANN.

Cada neurona tiene asociadas dos funciones distintas: la función de entrada y la función de activación [22].

La función de entrada es aquella que modifica los valores de entrada de la neurona para operar con ellos. Cada capa de la red tiene asociada una matriz de números llamada matriz de pesos. Estos pesos afectan a cada una de las entradas, sumando, multiplicando, etc. Lo más común es que se multipliquen

las entradas por sus pesos correspondientes. De este modo, se le puede dar más importancia a unos valores que a otros. Además, suele haber también un valor adicional llamado umbral para cada una de las neuronas que se resta al resultado total de la operación anterior.

Con ello, formamos una matriz para cada capa con una serie de pesos y valores umbral distintos. Esto es, si tenemos i neuronas con j entradas cada una tendriamos una matriz de $i * (j + 1)$ pesos diferentes.

Todos estos valores forman la función de entrada para la neurona i, que tiene la siguiente forma:

$$F_{ent\,i} = \sum_{j=1}^{n} w_{ij} * x_j - u_j$$

con w_{ij} el peso j de la neurona i, x_j la entrada j y u_j el umbral.

La función de activación es la función que da la salida de la neurona, utilizando el resultado de la función de entrada. Hay varias funciones diferentes, siendo la más utilizada la función sigmoide, aunque hay muchas otras diferentes, como la lineal o la tangente hiperbólica. Además, cada neurona de una red puede tener una función de activación diferente [22].

El cálculo de la función de activación es:

$$F_{act\,i} = f(F_{ent\,i})$$

con f la función de activación seleccionada.

Otra variante de la función de activación muy utilizada son las neuronas todonada que dan una salida binaria, 1 o 0, dependiendo de si el valor de la función de entrada es mayor o menor que 0 respectivamente. Esto puede entenderse también como que la función de activación "activa" o "desactiva" la neurona.

La ANN creada puede ser muy diferente, pudiendo tener muchas o pocas conexiones, tener ciclos entre ellas y hasta volver a capas anteriores. Esto permite crear redes complejas que se adapten al problema que se intenta resolver.

La característica más llamativa de las ANN es su capacidad de aprendizaje. Esto es debido a la facilidad para variar la salida cambiando los pesos de la ANN. Para "enseñar" a una ANN es necesaria entrenarla con una muestra o 'dataset' que contenga datos que representen bien a la población real del problema a resolver.

El dataset se divide en dos partes: muestra de entrenamiento o 'train' y muestra de testeo o 'test'. La muestra de entrenamiento es la que hace el entrenamiento de la red, es decir, en el que se basará el aprendizaje de la red. La muestra de testeo se utiliza cuando la red está entrenada para determinar la fiabilidad de la misma.

Para ello existen dos técnicas o mecanismos de aprendizaje [23]:

- Aprendizaje supervisado: este mecanismo consiste en realizar el entrenamiento de la red controlándolo externamente. Esto se consigue observando la salida que se obtiene en cada caso, comprobando si es la esperada según la entrada que haya tenido. En el caso de que no coincida, se tratará de corregir el error modificando los pesos de la ANN.

- Aprendizaje no supervisado: este mecanismo, a diferencia del anterior, consiste en no dar ninguna información a la red durante su entrenamiento. A partir de los datos de entrenamiento tratará de extraer características comunes entre ellos y modificando los pesos de la ANN. Cuando se le presenta un nuevo dato a analizar, la red busca las similitudes que haya entre el dato y los que se le dieron en fase de entrenamiento.

Para comprobar cuánto se acerca a la solución óptima, la ANN calcula una función de pérdida o de coste, que tratará de minimizar para mejorar en las predicciones. En algunos casos, simplemente comprueba el número de errores cometidos y trata de minimizarlos; en otros casos calcula la distancia entre la salida esperada y el resultado. Esta función es la que ayuda a la ANN a aprender.

Mediante algoritmos más complejos que utilizan técnicas de estadística y optimización, se puede mejorar o evolucionar la red original y con ello obtener mejores resultados o predicciones con nuevos datos. Uno de los algoritmos de aprendizaje más utilizados es el de propagación hacia atrás, que calcula el error producido por cada neurona, desde las neuronas de la salida hacia la capa de entrada. De esta manera se puede saber la fracción de error introducido por cada neurona y así corregir sus pesos en base a esa fracción del error.

En este caso, se ha utilizado una técnica de aprendizaje supervisado, dándole una serie de datos con su respectiva clasificación. Keras realiza una serie de iteraciones, aplicando el algoritmo de propagación hacia atrás en cada una, y que tratan de mejorar los resultados de las iteraciones previas:

En primer lugar, hace un entrenamiento con los datos de entrenamiento, calculando la función pérdida a minimizar y la tasa de acierto. Después, utiliza los datos de entrenamiento y calcula la pérdida y la tasa de acierto. Con esto, vuelve a realizar otra iteración, tratando de minimizar las funciones perdida y, con ello, mejorar la tasa de acierto. Esto se repite tantas veces como se le indique a Keras o, como opción adicional, se le puede decir que pare cuando deje de aprender, es decir, cuando la función pérdida no mejore 'n' iteraciones seguidas.

Este proceso puede ser demasiado largo, dependiendo del tamaño del dataset, del tamaño de los datos de entrada y del número de iteraciones que se realicen. En el caso concreto de este proyecto, esto se puede ver claramente en las iteraciones que se han hecho. En el primer caso, al tratarse de imágenes de pequeño tamaño no requería demasiado tiempo, alcanzándose una buena tasa de acierto en unos minutos. En los casos sucesivos, al tratar con vídeos, que a su vez contienen diversos fotogramas, este tiempo creció considerablemente, tardando horas en completar el entrenamiento.

Tras realizar el entrenamiento, el modelo de la red, previamente diseñado, y los pesos de la misma se guardan en ficheros. Estos ficheros son los que utiliza la aplicación para hacer los cálculos. Como la ANN ya está creada y entrenada, el tiempo requerido es breve, ya que el computador solo necesita hacer los cálculos de una entrada una sola vez y no varias como cuando se entrenó.

3.2.2. Redes neuronales convolucionales: Clasificación de datos multidimensionales

Las redes neuronales convolucionales (en adelante CNN, del inglés "Convolutional Neural Networks") son una variante de las ANN. Se caracterizan por estar diseñadas para trabajar con datos multimensionales como datos en 2D tales como imágenes o audios, e incluso en 3D, tales como vídeos, entendidos como una sucesión de imágenes.

La estructura de las CNN es similar a las de las ANN, habiendo varias capas de neuronas interconectadas entre sí. En el caso de las CNN hay varias capas distintas, cada una de ellas teniendo una función en la red particularmente.

Las capas de una CNN, distinguiéndose por su funcionalidad, son [24]:

- Capa convolucional: esta capa, que da nombre al tipo de red, es la que se ocupa de hacer la extracción de las características de la entrada. A esta operación se la conoce como convolución. Esto lo consigue aplicando una serie de filtros o 'kernels', que son una serie de matrices que se aplican a la matriz de datos de entrada, obteniendo una matriz de datos resultante de la convolución.

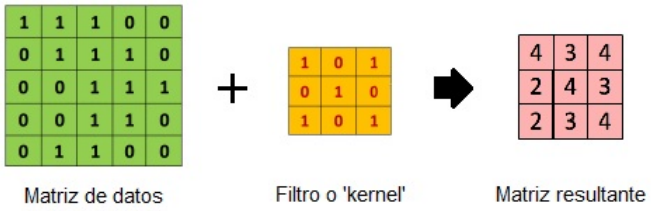

Figura 30: Proceso de la capa convolucional
Fuente: [24]

  En la Figura 30 se puede ver la matriz de datos y el filtro que se le aplica. Para obtener el resultado, se aplica el filtro a todas las zonas posibles. Por ejemplo, el filtro se aplica en la submatriz (0,0) a (2,2) para obtener el punto (0,0). Esto además hacer que la matriz resultante sea menor que la original, dependiendo del tamaño de la matriz de entrada y del filtro aplicado.

- Capa de "rectificación lineal" o de ReLU (del inglés Rectified Linear Unit): esta capa de la red es complementaria a la de convolución. Esta capa se utiliza para introducir una no-linearidad a la función de convolución, la cual dará una salida lineal respecto de los datos de entrada. Al introducir una no-linearidad se evita generar la misma solución para problemas similares.

  La capa de rectificación opera dato a dato, a diferencia de la capa convolucional que operaba sobre submatrices. Esta capa transforma los números

negativos en 0, dejando los positivos como están o, lo que es lo mismo, calcular la función máximo entre el dato y 0. Existen otras operaciones que también sirven para introducir no-linealidad al modelo, como la tangente o la sigmoide.

- Capa de agrupamiento o 'pooling': esta capa, como su propio nombre indica, hace un agrupamiento de todos los datos de la matriz que estén contiguos para reducir la matriz. Esto es similar a lo que se hace con la convolución que se opera con submatrices de los datos. En este caso, se opera por regiones, partiendo la matriz en submatrices independientes.

  Por lo general, la operación que se suele utilizar es la función máximo, que además es una función no-lineal. Otra función también utilizada es la función sumatorio, aunque en este caso es una función lineal.

- Capa de clasificación: esta capa es la última de la CNN. Es básicamente una ANN completamente conectada, con varias capas de una red básica. Es completamente conectada ya que todas las neuronas de una están conectadas a todas las de la capa anterior y a todas las de la capa siguiente. Esto es porque, al tratarse de imágenes o vídeos, en cada cálculo es necesario conocer los datos de todas las regiones de la imágen.

  Esta última capa o red es la que dará la salida a la CNN, haciendo los últimos cálculos. Además, al llegar a ella, los datos de entrada se habrán reducido tanto que será más sencillo y rápido tratarlos con una red neuronal sencilla y no con capas que requieren muchas operaciones.

Para formar una CNN se conectan diversas de estas capas de modo que se vaya reduciendo el dato de entrada, extrayendo las características de las imágenes, vídeos, etc.

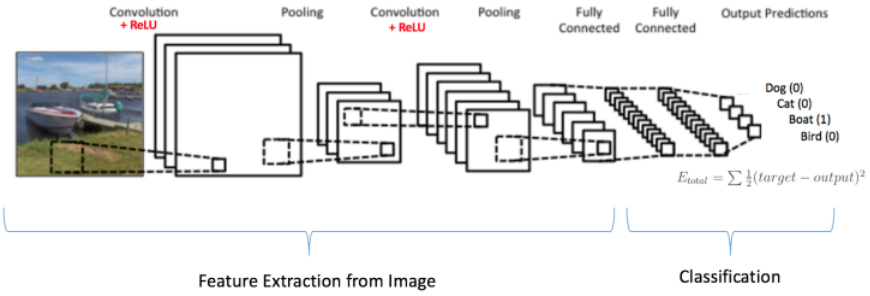

Figura 31: Modelo de red convolucional
Fuente: [24]

En la Figura 31 se puede ver una construcción de una CNN. Normalmente, se suelen utilizar las capas convolucional y de rectificación juntas para, como ya se ha dicho, corregir la linearidad de la capa de convolución. Después de introducir

unas pocas de estos pares de capas se suele introducir una de agrupamiento para reducir el tamaño y quedarse solo con los datos importantes. Tras realizar esto varias veces y habiendo reducido la entrada considerablemente, finalmente se introduce la capa de clasificación para obtener un resultado.

Para la realización de este proyecto se ha utilizado, como ya se ha mencionado, el modelo de CNN, ya que los datos a tratar son imágenes y vídeos. En el apartado siguiente se mostrará cómo se ha construido la red neuronal en cada uno de los casos.

3.2.3. Implementación del sistema de clasificación

Tras investigar el funcionamiento de las CNN se diseño y construyó el modelo de las redes que harán las clasificaciones con Keras. Primero de todo, para poder introducir las imágenes a la red, se tuvo que hacer un preprocesamiento.

```
train_datagen = ImageDataGenerator(
    rescale = 1. / 255,
    shear_range = 0,
    zoom_range = 0.2,
    horizontal_flip = True)
```

Figura 32: Generador de imágenes

Como se ve en la Figura 32, previamente las imágenes son tratadas. Las imágenes son una matriz de datos divididos en triplas con valores en $[0, 255]$ donde, los tres valores juntos, representan un color en el modelo RGB. Para simplificar los cálculos es conveniente tenerlos en $[0, 1]$, por lo que se divide cada dato de entrada entre 255. Además, se ha introducido una aleatoriedad, permitiendo un zoom de hasta un 20 % y que las imágenes se volteen horizontalmente.

```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size = (img_width, img_height),
    batch_size = batch_size,
    class_mode = 'categorical')
```

Figura 33: Carga de imagenes

En la Figura 33 se puede ver como se hace la carga de imágenes a partir del generador anterior. Esta función lo que hace es cargar las imágenes con su respectiva clase, que en este caso está en el nombre del directorio que la contiene. Además, carga cada imágen para que tenga unas dimensiones dadas y así que todas las imágenes sean iguales. A parte de esto, las imágenes se almacenan en pequeños grupos o 'batches' para que Keras trabaje con ellos de este modo más fácilmente, tratando de extraer características de estos conjuntamente.

Para el caso de los vídeos se hace algo similar aunque, debido a que Keras no permite introducir directamente vídeos, se tuvo que programar una función

propia. Esta función sirve para extraer los frames de cada imágen, agruparlos para que pertenezcan al mismo dato de entrada para la red y para realizar las operaciones de reescalado de las imágenes y del color.

1. Red para la clasificación de imágenes.

   En este caso se ha desarrollado una red sencilla, ya que las imágenes son pequeñas y se quería hacer una comprobación del funcionamiento de Keras.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Figura 34: Primeras capas de la red

   Como se ve en la Figura 34, se crea un modelo secuencial, que es el que se ha explicado anteriormente y donde no hay ciclos de ningún tipo. La primera capa, la capa de entrada, es la que restringe los datos a que tengan una cierta forma.

   En esta primera parte de la red, se han hecho cinco fases iguales, todas con una convolución de datos 2D, al tratarse de imágenes, luego una capa de rectificación lineal y después el 'pooling'. Como ya se ha dicho, esto se realiza varias veces para extraer las características de las imágenes al mismo tiempo que se reducen de tamaño.

```
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dense(len(clases)))
model.add(Activation('sigmoid'))
```

Figura 35: Capas de clasificación de la red

En la Figura 35 se puede ver la última parte de la red. Primero, para poder conectar la capa de 'pooling' hay que "aplanar" la matriz, esto es, convertirla en un vector o array unidimensional con la capa 'Flatten'. Después, se ha introducido una capa totalmente conectada o 'Dense'. Entre esta capa y la final se ha introducido una de rectificación para, como ya se explicó, mantener la no-linearidad del modelo. Finalmente, se vuelve a hacer lo mismo en la última capa, esta vez realizando la función sigmoide.

Con esto, el modelo ya queda construido, aunque quedan de hacer la compilación y el entrenamiento.

```
sgd = SGD(lr = 0.01, decay = 0.01/epochs, momentum = 0.9, nesterov = True)
model.compile(loss = 'categorical_crossentropy',
              optimizer = sgd,
              metrics = ['accuracy'])
```

Figura 36: Compilación del modelo

En la Figura 36 vemos como se ha compilado el modelo. En primer lugar, se ha creado una instancia de un optimizador, concretamente de uno de gradiente descendente, con varios parámetros que se fueron cambiando para comprobar el funcionamiento y aumentar o reducir la velocidad de aprendizaje.

En la compilación del modelo se le pasó a la función este optimizador, además de indicar la función para calcular las pérdidas y, adicionalmente, que en el proceso de ejecución se mostrase la tasa de acierto lograda en cada momento.

```
model.fit_generator(
    generator = train_generator,
    steps_per_epoch = nb_train_samples // batch_size,
    epochs = epochs,
    verbose = 1,
    callbacks = [logger],
    validation_data = validation_generator,
    validation_steps = nb_validation_samples // batch_size)
```

Figura 37: Entrenamiento de la red

Como se ve en la Figura 37, aquí es donde se realiza la operación que inicia el entrenamiento, la operación más costosa de todas ya que requiere de muchas operaciones e iteraciones. Se puede ver que se le han pasado varios parámetros a la función.

Los dos primeros son relativos a los datos de 'train', el primero es el generador de imágenes, que es el que carga las imágenes y las preprocesa; y el segundo es el número de procesamientos que hace por iteración, que es el módulo del número de datos entre el tamaño de los grupos hechos. Los

dos últimos parámetros son iguales que estos dos, siendo en este caso los referidos a los datos de 'test'. Otro parámetro importante es el de 'epochs' que indica el número de iteraciones distintas realiza el entrenamiento.

Los otros dos parámetros son adicionales: 'verbose' indica que se muestren los errores y avisos completamente; y por último, se le ha indicado que se guarde el log de la ejecución en un archivo, como se indica en el parámetro 'callbacks'.

2. Red para la clasificación de vídeos con varias clases.

   Esta es la segunda iteración del proyecto, haciendo un cambio de imágenes a vídeos. Como ya se ha comentado se tuvo que hacer un preprocesamiento de los vídeos para reducirlos a imágenes, por lo que tras esto el procedimiento a seguir para la construcción de la red fue similar al anterior, teniendo simplemente que introducir algunos cambios y hacer la red algo más grande para que sea capaz de contener más información.

```
model = Sequential()
model.add(TimeDistributed(Conv2D(32, (3,3),
    kernel_initializer='he_normal',
    activation='relu'), input_shape=input_shape))
model.add(TimeDistributed(Conv2D(32, (3,3),
    kernel_initializer='he_normal',
    activation='relu')))
model.add(TimeDistributed(MaxPooling2D()))
model.add(TimeDistributed(Conv2D(48, (3,3),
    kernel_initializer='he_normal',
    activation='relu')))
model.add(TimeDistributed(Conv2D(48, (3,3),
    kernel_initializer='he_normal',
    activation='relu')))
model.add(TimeDistributed(MaxPooling2D()))
model.add(TimeDistributed(Conv2D(64, (3,3),
    kernel_initializer='he_normal',
    activation='relu')))
model.add(TimeDistributed(Conv2D(64, (3,3),
    kernel_initializer='he_normal',
    activation='relu')))
model.add(TimeDistributed(MaxPooling2D()))
model.add(TimeDistributed(Conv2D(128, (3,3),
    kernel_initializer='he_normal',
    activation='relu')))
model.add(TimeDistributed(Conv2D(128, (3,3),
    kernel_initializer='he_normal',
    activation='relu')))
model.add(TimeDistributed(MaxPooling2D()))
```

Figura 38: Primeras capas de la red

En la Figura 38 vemos que, como en la primera red creada, se crea un modelo secuencial. También se van alternando capas de convolución con las

capas de rectificación para mantener la no-linearidad del modelo. En este caso, se han creado dos de estos pares antes de la capa de agrupamiento con el fin de almacenar más información de las imágenes.

En este caso, se ha tenido que utilizar otra clase adicional, la clase 'TimeDristributed' que permite introducir otra dimensión más al modelo, en este caso la del tiempo, como es el caso de los vídeos, que son datos en 2D (fotogramas) pero que varían con el tiempo.

```
model.add(TimeDistributed(Flatten()))
model.add(LSTM(256, return_sequences=True))
model.add(Flatten())
model.add(Dense(512))
model.add(Dropout(0.5))
model.add(Dense(len(clases), activation='softmax'))
```

Figura 39: Capas de clasificación de la red

Como se aprecia en la Figura 39, aquí se ha hecho como en el caso anterior, haciendo primero la transformación de los datos como matriz a vectores pero en este caso, al estar tratando con 3 dimensiones, la capa 'Flatten' devolverá vectores por cada secuencia del vídeo. Por ello, se introduce otra capa más, la capa LSTM, que agrupa varias secuencias de vectores en una sola, por lo que ya se pueden operar los datos con capas básicas.

Tras esto, se introducen dos capas totalmente conectadas y, entre ellas, una capa de descarte, que sirve para introducir 'ruido' en la red. Con ello reducimos problemas de sobreajuste, causado por una predicción buena sobre los datos de entrenamiento pero mala para nuevos datos.

Finalmente, la compilación y la iniciación del entrenamiento se hicieron de la misma manera que la primera red. Los cambios importantes se incluyeron en el diseño de la propia red.

3. Red para la clasificación de vídeos con dos clases.

   Esta red es muy similar a la anterior, utilizando el mismo modelo y solo cambiando que, en vez de varias clases, solamente hay que hacer una predicción binaria. Por ello, el único cambio que se hizo es:

```
model.compile(loss='binary_crossentropy', optimizer=sgd,
              metrics=['accuracy'])
```

Figura 40: Compilación del modelo para dos clases

Como se puede ver en la Figura 40, el cambio se ha hecho en la función pérdida, que resulta mejor calcularla con una específica para dos clases que con otra hecha para tener varias clases.

Todo esto crea e inicia el proceso de entrenamiento de la red neuronal creada. Los resultados de estos entrenamientos se verán en los apartados del capítulo 5 Evaluación y pruebas.

3.3. Construccion de la app

La aplicación se ha desarrollado en el entorno de trabajo Eclipse, utilizando Java como lenguaje de programación, Swing como librería adicional y Window-Builder Editor como herramienta de diseño de interfaces gráficas.

La aplicación creada es una aplicación sencilla que cubre la funcionalidad requerida. De este modo, los/as médicos/as no tendrán complicaciones a la hora de utilizar la aplicación.

La aplicación se ha diseñado para cumplir los requisitos especificados en la sección 3.1 Análisis de requisitos funcionales y no funcionales. Las distintas funcionalidades de la aplicación cumplen el requisito funcional y la misma se ha diseñado para que cumpla con los requisitos no funcionales especificados. Los casos de uso, al ser pocos, se traducen en pocas funcionalidades de la aplicación y tener pocas interacciones con el usuario.

A continuación se detallan las distintas funciones junto con imágenes que ejemplifican una ejecución normal.

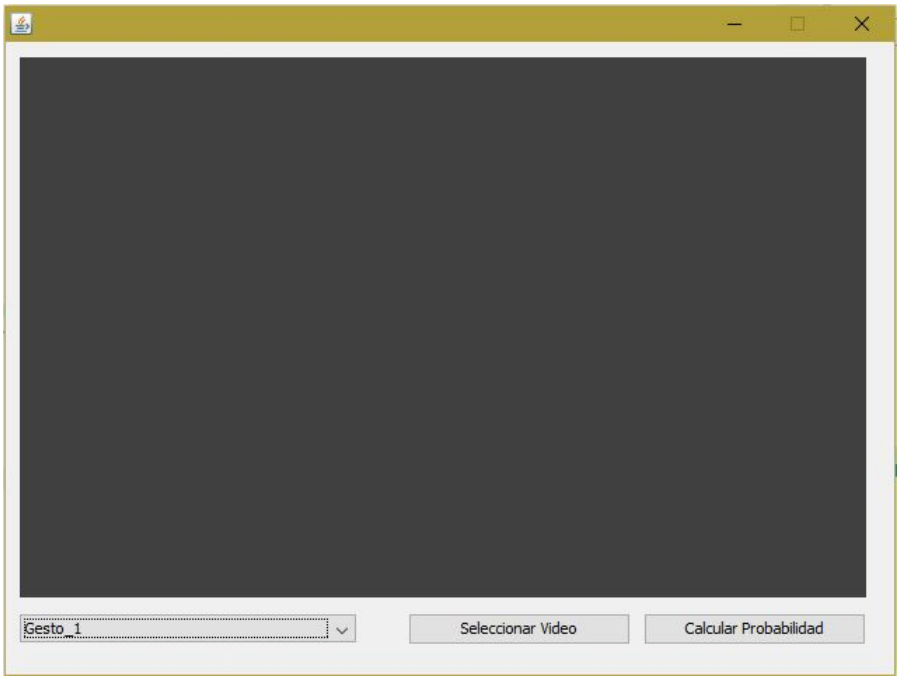

Figura 41: Ventana principal de la aplicación

En la Figura 41 se muestra la ventana principal justo cuando se abre la apli-

cación. En ella se muestran la lista desplegable de gestos disponibles, el botón de selección de vídeo (Seleccionar Vídeo) y el botón para hacer el cálculo (Calcular Probabilidad). Además, se muestra un panel en gris donde se reproducirán los vídeos seleccionados en cada momento.

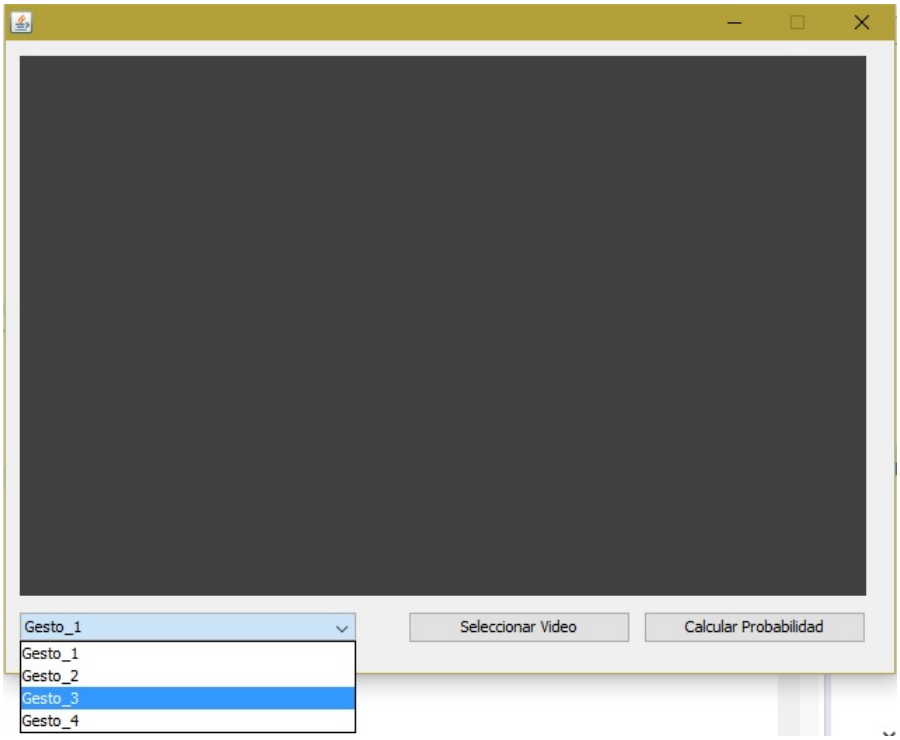

Figura 42: Lista de gestos disponibles

Como se ve en la Figura 42, si se selecciona la lista de gestos, ésta se despliega mostrando los gestos para los que se puede hacer la predicción. Esto cubre el caso de uso 'Cambiar gesto'. De momento, los nombres son provisionales, a la espera de que se genere una versión final con redes entrenadas con datos reales. Tras seleccionar un gesto, este será utilizado para hacer la predicción.

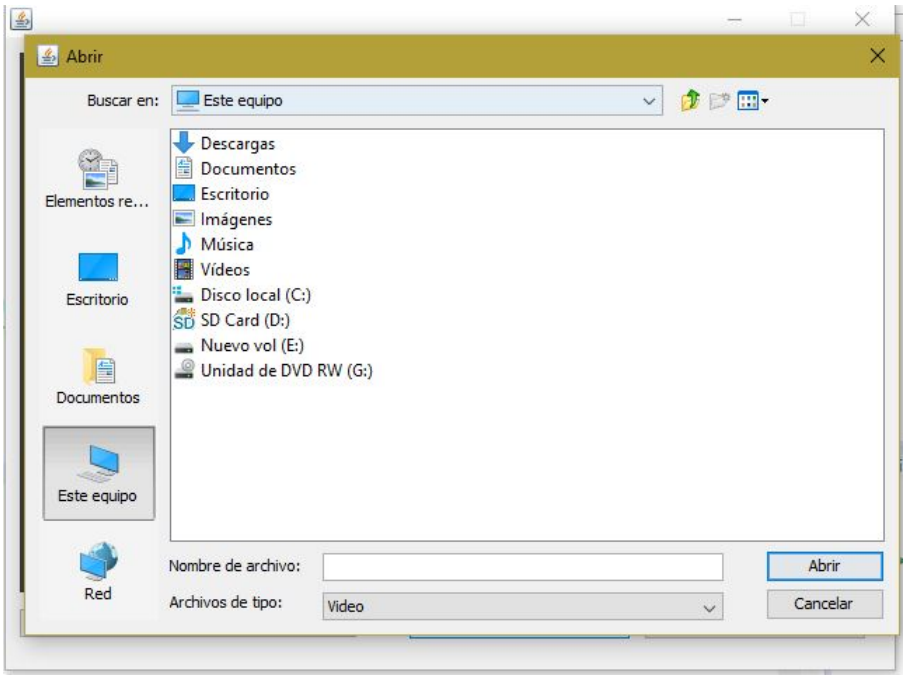

Figura 43: Ventana de selección de vídeo

Como se puede ver en la Figura 43, si se pulsa el botón 'Seleccionar Vídeo' se abrirá otra ventana, donde el usuario podrá seleccionar en su sistema un vídeo para analizar. Esto cubre el caso de uso 'Seleccionar vídeo'. Cuando se seleccione un vídeo, el usuario pulsará 'Aceptar' para completar la acción. En cualquier momento, puede cerrar la ventana o pulsar el botón 'Cancelar' para volver a la ventana principal sin terminar de seleccionar el vídeo.

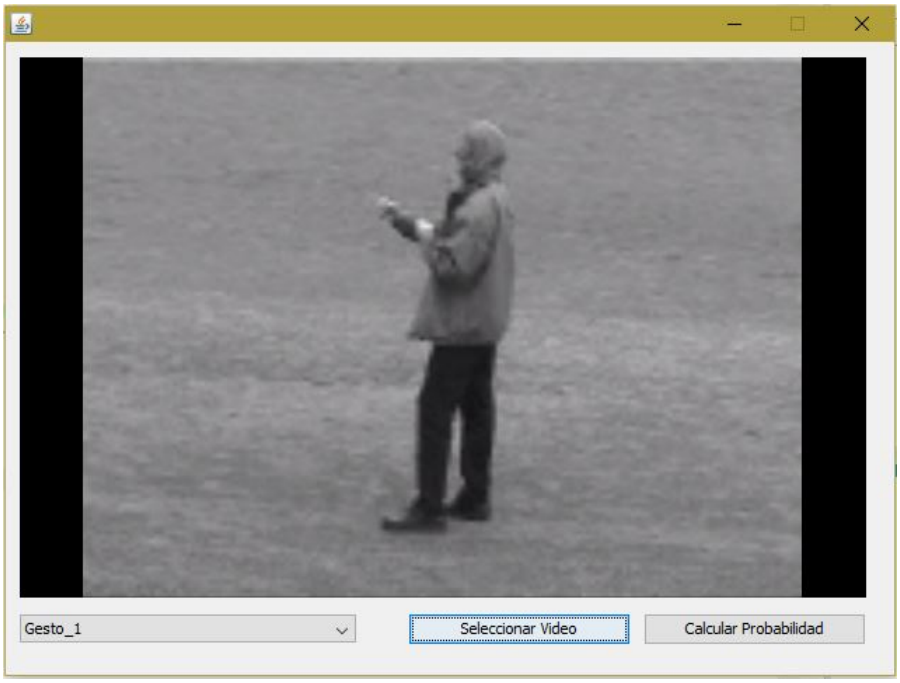

Figura 44: Ventana principal con vídeo reproduciendo

Cuando se selecciona el vídeo y se pulsa 'Aceptar' la ventana se cerrará, volviendo a la ventana principal. En ella, se reproducirá en el panel de la ventana el vídeo seleccionado en bucle, como se ve en la Figura 44. Si se selecciona otro vídeo el que estaba reproduciendo se cambiará por el nuevo.

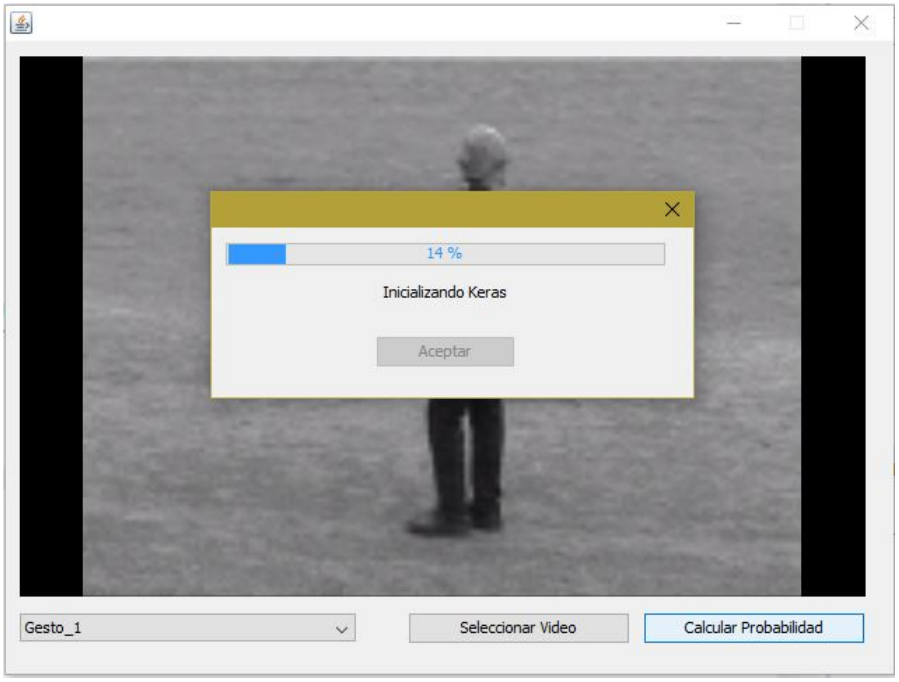

Figura 45: Proceso de cálculo de la probabilidad

Si se pulsa el botón 'Calcular probabilidad' aparecerá una ventana emergente, como se ve en la Figura 45. Esta ventana muestra el progreso de la operación de cálculo mediante una barra de carga y varios mensajes con información relativa al proceso. Al finalizar se mostrará al usuario el resultado de la operación: la clasificación, 'bueno' o 'malo', y la probabilidad de la predicción. Esto cubre el caso de uso 'Clasificar vídeo'.

Esta aplicación cumple con la funcionalidad requerida como se ha detallado. Esta funcionalidad podrá cambiarse en el caso que los usuarios lo requieran, modificando o añadiendo nuevas funciones.

4. Evaluación y pruebas

En este apartado se explica las pruebas que se han realizado con el software y tras entrenarlo con el dataset. También se explica brevemente los resultados obtenidos en las pruebas.

4.1. Ejecuciónes de Keras

Inicialmente, se probaron varias funciones de Keras para comprobar que su instalación era correcta y fuese fácil de hacer. Además, se construyó una pequeña red neuronal para comprobar que las ejecuciones se podían realizar sin ningún problema.

```
Epoch 20/50
66/66 [==============================] - 243s - loss: 0.6012 - acc: 0.7045 - val_loss: 0.5721 - val_acc: 0.8000
Epoch 21/50
66/66 [==============================] - 243s - loss: 0.6117 - acc: 0.7045 - val_loss: 0.6439 - val_acc: 0.7000
Epoch 22/50
66/66 [==============================] - 245s - loss: 0.5936 - acc: 0.6894 - val_loss: 0.5977 - val_acc: 0.7000
Epoch 23/50
66/66 [==============================] - 243s - loss: 0.5789 - acc: 0.7273 - val_loss: 0.6924 - val_acc: 0.6500
Epoch 24/50
66/66 [==============================] - 243s - loss: 0.5516 - acc: 0.7348 - val_loss: 0.5449 - val_acc: 0.8000
Epoch 25/50
66/66 [==============================] - 244s - loss: 0.5316 - acc: 0.7803 - val_loss: 0.5651 - val_acc: 0.8000
Epoch 26/50
66/66 [==============================] - 243s - loss: 0.5546 - acc: 0.7652 - val_loss: 0.5490 - val_acc: 0.8000
Epoch 27/50
66/66 [==============================] - 243s - loss: 0.5078 - acc: 0.7803 - val_loss: 0.5488 - val_acc: 0.7000
Epoch 28/50
66/66 [==============================] - 242s - loss: 0.5149 - acc: 0.8182 - val_loss: 0.5496 - val_acc: 0.8000
Epoch 29/50
66/66 [==============================] - 245s - loss: 0.5461 - acc: 0.7576 - val_loss: 0.5758 - val_acc: 0.7500
Epoch 30/50
66/66 [==============================] - 243s - loss: 0.5157 - acc: 0.7500 - val_loss: 0.6248 - val_acc: 0.6500
Epoch 31/50
66/66 [==============================] - 243s - loss: 0.5043 - acc: 0.7879 - val_loss: 0.4092 - val_acc: 0.9500
Epoch 32/50
66/66 [==============================] - 248s - loss: 0.4947 - acc: 0.7955 - val_loss: 0.5711 - val_acc: 0.7000
Epoch 33/50
66/66 [==============================] - 248s - loss: 0.5214 - acc: 0.7576 - val_loss: 0.4852 - val_acc: 0.8000
Epoch 34/50
66/66 [==============================] - 247s - loss: 0.4516 - acc: 0.8258 - val_loss: 0.5079 - val_acc: 0.8000
Epoch 35/50
66/66 [==============================] - 246s - loss: 0.4681 - acc: 0.8409 - val_loss: 0.5930 - val_acc: 0.6500
Epoch 36/50
66/66 [==============================] - 249s - loss: 0.4588 - acc: 0.8333 - val_loss: 0.4284 - val_acc: 0.9500
Epoch 37/50
66/66 [==============================] - 247s - loss: 0.4905 - acc: 0.8030 - val_loss: 0.5285 - val_acc: 0.8000
Epoch 38/50
66/66 [==============================] - 250s - loss: 0.4077 - acc: 0.8788 - val_loss: 0.5118 - val_acc: 0.9000
Epoch 39/50
66/66 [==============================] - 248s - loss: 0.4642 - acc: 0.7652 - val_loss: 0.6141 - val_acc: 0.7000
Epoch 40/50
66/66 [==============================] - 247s - loss: 0.4421 - acc: 0.8409 - val_loss: 0.5182 - val_acc: 0.8000
Epoch 41/50
66/66 [==============================] - 249s - loss: 0.4445 - acc: 0.8409 - val_loss: 0.2985 - val_acc: 1.0000
Epoch 42/50
66/66 [==============================] - 253s - loss: 0.4252 - acc: 0.8258 - val_loss: 0.5533 - val_acc: 0.8000
Epoch 43/50
66/66 [==============================] - 249s - loss: 0.4179 - acc: 0.8636 - val_loss: 0.5523 - val_acc: 0.7000
Epoch 44/50
66/66 [==============================] - 250s - loss: 0.4033 - acc: 0.8636 - val_loss: 0.4029 - val_acc: 0.9500
Epoch 45/50
66/66 [==============================] - 254s - loss: 0.3962 - acc: 0.8561 - val_loss: 0.4847 - val_acc: 0.7500
Epoch 46/50
66/66 [==============================] - 252s - loss: 0.4211 - acc: 0.8030 - val_loss: 0.3663 - val_acc: 0.9000
Epoch 47/50
66/66 [==============================] - 247s - loss: 0.4083 - acc: 0.8561 - val_loss: 0.5808 - val_acc: 0.6500
Epoch 48/50
66/66 [==============================] - 250s - loss: 0.3366 - acc: 0.9091 - val_loss: 0.3803 - val_acc: 0.9500
Epoch 49/50
66/66 [==============================] - 248s - loss: 0.3569 - acc: 0.8712 - val_loss: 0.3954 - val_acc: 0.8500
Epoch 50/50
66/66 [==============================] - 249s - loss: 0.3479 - acc: 0.9091 - val_loss: 0.3968 - val_acc: 0.7500
```

Figura 46: Muestra de una ejecución de Keras

En la Figura 46 se ve una ejecución normal de Keras. Ésta muestra por consola varias cosas: la iteración actual, el progreso, el tiempo que ha empleado en esa iteración, las pérdidas, la tasa de acierto en 'train' y las pérdidas y la tasa de acierto en 'test'. Como se ve en la imágen, el tiempo empleado en este proceso es muy alto, alrededor de los 4 minutos por cada iteración. Este ejemplo de ejecución es de una ejecución de las pruebas de vídeo.

4.2. Pruebas de imagen

En la primera iteración del proyecto se desarrolló la CNN para trabajar con imágenes y se probó con el dataset de imágenes descrito en la sección 2.4 Dataset utilizado [17].

Como se puede observar en las imágenes adjuntas, las imágenes son muy diferentes, con distintos fondos y distintas personas realizando los gestos. Algunos de estos gestos son similares a otros, como por ejemplo la clase 'a' y la clase 'd'. Esto incrementa la dificultad del problema para estas dos clases, ya que clasificará mal en las clases similares, pudiendo confundirlas.

Además, una parte de las imágenes tienen ruido humano para que se asemeje más a una imagen obtenida de una situación real, como será en el ambiente de un hospital. En el caso del proyecto, al grabarse a los pacientes frontalmente, el ruido humano lo causarán los propios pacientes, al aparecer detrás de los gestos que realizan con sus propias manos.

En este caso, se realizaron clasificaciones de las imágenes en 10 clases diferentes, lo cual es más complejo que el caso final en el que la clasificación será binaria.

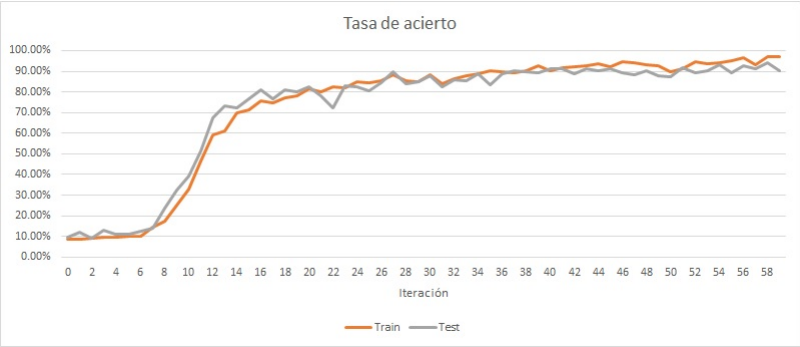

Figura 47: Tasa de acierto en la prueba con imágenes

Como se ve en la Figura 47, la tasa de acierto en 'Train' y en 'Test' aumenta desde el principio, alcanzándose una buena tasa de acierto rápidamente y manteniéndose en las diferentes iteraciones, llegando a incrementarse hasta superar el 90 % al final. Es por ello que la primera prueba, la clasificación con imágenes, dio resultados favorables y se comprobó que Keras era una buena elección para continuar trabajando con ello.

4.3. Pruebas de vídeo

A partir de la segunda iteración ya se construyó la CNN para trabajar con vídeos. Un vídeo no es sino una sucesión de imágenes que pasan cada cierto intervalo de tiempo por lo que esta red es una evolución de la anterior, añadiendo una 'tercera dimensión' que es el tiempo.

Para realizar estas pruebas se utilizó el dataset descrito en la sección 2.4 Dataset utilizado [18]. Este dataset, como se pueden ver en los fotogramas extraídos de algunos vídeos, están en blanco y negro y muestran a personas realizando gestos a cuerpo completo.

Como pasa en el caso anterior, hay algunas clases que son similares, como por ejemplo la clase 'jogging' y la clase 'running', por lo que, también como en el caso de las imágenes, el sistema confundirá estas clases más a menudo.

Para comprobar la capacidad de la CNN y de Keras se hizo una primera prueba de multiclasificación con las 6 clases disponibles.

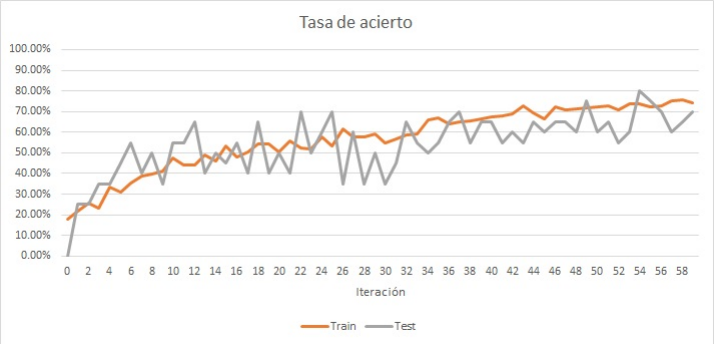

Figura 48: Tasa de acierto en la prueba con vídeos de varias clases

En la Figura 48 se ve que el progreso de la tasa de acierto es continuado en las sucesivas iteraciones. En el caso del 'Test' hay algunas zonas en las que decrementa ligeramente la tasa de acierto, pero finalmente se ve que se mantiene alta también al final, llegando a un 70 % en las iteraciones finales aproximadamente. Esta segunda prueba también es satisfactoria, ya que con 6 clases diferentes y tratando con vídeos se consigue una buena tasa de aciertos.

Para la segunda parte ya se probó el sistema con una clasificación binaria, que se aproxima más al caso final, donde habrá que clasificar un gesto como bueno o malo. En este caso, hay pocos vídeos para realizar el entrenamiento, habiendo 67 vídeos en cada clase.

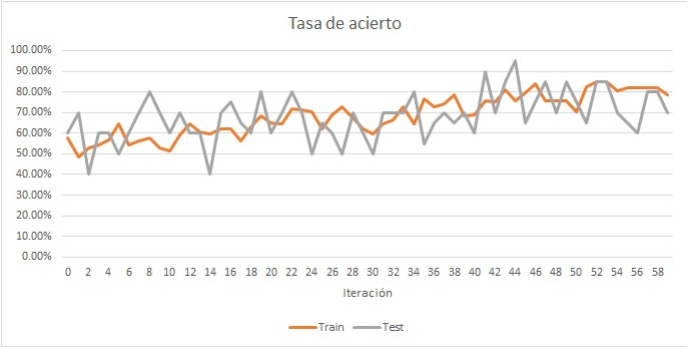

Figura 49: Tasa de acierto en la prueba con vídeos de dos clases

Como se ve en la Figura 49, la tasa de acierto se mantiene aproximadamente superior al 50 %, entorno al 60 % y el 70 %, aunque se puede ver que hay un incremento pequeño en la fase de 'Train', superando el 70 % al final, por lo que con alguna iteración más se alcanzaría una tasa de acierto incluso mejor.

Todas estas pruebas sirvieron para comprobar el funcionamiento de Keras y de este modo certificar que era un software válido para su utilización y para resolver el problema planteado. Estas pruebas demuestran que se puede construir con Keras una aplicación que ayude a los/as médicos/as a detectar apraxias en los pacientes.

5. Conclusiones y siguientes pasos

En este último apartado se realiza la valoración general del proyecto y se describen cómo se han cumplido los objetivos marcados inicialmente. Además, se describen los siguientes pasos a seguir para mejorar la aplicación creada.

5.1. Conclusiones del proyecto

Durante la realización de este proyecto se han desarrollado un sistema inteligente, utilizando redes neuronales convolucionales para ello; y una aplicación sencilla para aprovechar la capacidad de procesamiento de este tipo de redes. Esta red neuronal esta preparada para procesar vídeos, por lo que se podrá grabar a los pacientes y, de esta manera, ayudar a detectar apraxias en los gestos realizados por ellos.

Al inicio del proyecto se mencionaron varios objetivos a cumplir. Tras la realización del mismo, se vuelven a enumerar para comprobar si se han cumplido y en qué medida:

- Crear un sistema inteligente que sepa decidir, con una alta tasa de aciertos, si un vídeo pertenece a una clase o a otra (por ejemplo a "gesto bien" o a "gesto mal").

  Este objetivo se ha cumplido y con buena tasa de aciertos, ya que se ha desarrollado una red neuronal que acierta en hasta un 70 % de los casos. Esta probabilidad se podría incrementar tratando de cambiar algunos parámetros de la red o aumentando el número de capas.

- Entrenar este sistema con vídeos de varios gestos diferentes que permita obtener varios resultados de un mismo paciente. Con esto, aumentamos la confianza del sistema, ya que se realizan diferentes pruebas.

  Este objetivo se ha realizado parcialmente, ya que la red se ha entrenado con vídeos de un dataset. Solamente queda entrenar esta red con vídeos que proporcionen los/as médicos/as de los pacientes reales.

- Desarrollar una aplicación de escritorio usable y sencilla que permita a los/as médicos/as aprovechar estos sistemas y obtener un resultado que lo ayude a detectar si el paciente tiene una apraxia.

  Este objetivo se ha cumplido, ya que se ha creado una aplicación sencilla para que sea usable por cualquier usuario y, con ello, se pueda aprovechar toda la investigación y trabajo realizados.

Por ello, se puede decir que los resultados del proyecto son satisfactorios, quedando por hacer algunas pequeãs tareas para primero completar la funcionalidad de la aplicación, haciendo falta los vídeos de los casos reales, y probando la aplicación tras ello para pulir los posibles fallos que pudiera haber.

5.2. Siguientes pasos a realizar

La aplicación desarrollada es una muestra de la capacidad del modelo de CNN y de Keras para obtener predicciones con bastante tasa de acierto. Además, la propia aplicación es otra iteración más del proyecto, no siendo la final, necesitando de los datos de pacientes para hacer los últimos entrenamientos de la red y que la aplicación pueda utilizarse en un entorno real.

Los siguientes pasos, a partir de esta aplicación serán:

- Realizar entrenamientos con los datos de pacientes reales que capten los doctores de Valdecilla.
- Comprobar la tasa de acierto con estos gestos y así determinar si es útil para ayudar a detectar apraxias.
- Añadir más gestos diferentes en iteraciones posteriores y estudiando los gestos que son más útiles para detectar apraxias.

Con esto lo que se busca es optimizar el rendimiento de la aplicación para que así se facilite el trabajo de los doctores y mejorar la vida de los pacientes, ayudando a detectar el Alzheimer en las primeras fases. De esta forma el tratamiento puede comenzar antes, lo cual repercutirá positivamente en su salud.

Bibliografía

[1] A.M. Gil, *El Alzheimer*, ser. ¿Qué sabemos de...? Consejo Superior de Investigaciones Científicas, 2009.

[2] Tunstall, "Datos estadísticos en españa," [Accedido última vez: 30 de Junio, 2017]. [Online]. Disponible en: http://www.alzheimers-support.com/es/alzheimer-stats-spain.html

[3] Cruz Roja, "Alzheimer," [Última consulta: 30 de Junio, 2017]. [Online]. Disponible en: http://www.cruzroja.es/portal/page?_pageid=418,12392669&_dad=portal30&_schema=PORTAL30

[4] K. Gurney, *An Introduction to Neural Networks*. Taylor & Francis, 2003.

[5] "Python," [Última consulta: 27 de Junio, 2017]. [Online]. Disponible en: https://www.python.org/

[6] "Pypl popularity of programming language," [Última consulta: 27 de Junio, 2017]. [Online]. Disponible en: http://pypl.github.io/PYPL.html

[7] "Anaconda," [Última consulta: 27 de Junio, 2017]. [Online]. Disponible en: https://www.continuum.io/downloads

[8] "Java," [Última consulta: 27 de Junio, 2017]. [Online]. Disponible en: https://www.java.com/es/

[9] "Keras," [Última consulta: 27 de Junio, 2017]. [Online]. Disponible en: https://keras.io/

[10] "Theano," [Última consulta: 27 de Junio, 2017]. [Online]. Disponible en: http://deeplearning.net/software/theano/

[11] "Numpy," [Última consulta: 27 de Junio, 2017]. [Online]. Disponible en: http://www.numpy.org/

[12] "Nvidia cuda," [Última consulta: 27 de Junio, 2017]. [Online]. Disponible en: http://www.nvidia.es/object/cuda-parallel-computing-es.html

[13] "Java swing," [Última consulta: 27 de Junio, 2017]. [Online]. Disponible en: http://docs.oracle.com/javase/6/docs/technotes/guides/swing/

[14] "Vlc media player," [Última consulta: 27 de Junio, 2017]. [Online]. Disponible en: http://www.videolan.org/vlc/index.es.html

[15] "Eclipse," [Última consulta: 27 de Junio, 2017]. [Online]. Disponible en: https://eclipse.org/home/index.php

[16] "Windowbuilder editor," [Última consulta: 27 de Junio, 2017]. [Online]. Disponible en: https://eclipse.org/windowbuilder/

[17] National University of Singapore, "The nus hand posture datasets: Data set ii," [Última consulta: 29 de Marzo, 2017]. [Online]. Disponible en: https://www.ece.nus.edu.sg/stfpage/elepv/NUS-HandSet/

[18] School of Computer Science and Stockholm University Communication, "Recognition of human actions: Action database," [Última consulta: 29 de Abril, 2017]. [Online]. Disponible en: http://www.nada.kth.se/cvap/actions/

[19] Laboratorio Nacional de Calidad del Software de INTECO, *Ingeniería del Software: Metodologías y ciclos de vida.* Instituto Nacional de Tecnologías de la Comunicación, 2009.

[20] ISO Standards, "Systems and software engineering systems and software quality requirements and evaluation," [Última consulta: 1 de Julio, 2017]. [Online]. Disponible en: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

[21] "Red neuronal artificial," [Última consulta: 31 de Junio, 2017]. [Online]. Disponible en: https://es.wikipedia.org/wiki/Red_neuronal_artificial

[22] D. Jorge Matich, "Tema 8. redes neuronales," 2001, [Última consulta: 1 de Julio, 2017]. [Online]. Disponible en: https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf

[23] P. Larrañaga, I. Inza, and A. Moujahid, "Tema 8. redes neuronales," [Última consulta: 1 de Julio, 2017]. [Online]. Disponible en: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf

[24] U. Karn, "An intuitive explanation of convolutional neural networks," 2016, [Última consulta: 1 de Julio, 2017]. [Online]. Disponible en: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/