ESEIAAT

Tesis de Licenciatura

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

---

Estudio. Estimación del foco de atención visual mediante trayectorias orientadas.

Memoria

Licenciatura: Grado en Ingeniería de Sistemas Audiovisuales

Fecha de entrega: 10-06-2019

Estudiante: Villegas Santos, Albert

Director: Morros Rubio, Ramón

Codirector: López Palma, Manuel

RESUMEN

En este proyecto, aplicaremos los mecanismos y las técnicas que nos ofrecen las tecnologías de *Deep Learning,* para poder obtener el grado de atención que recibe un anuncio por parte de las personas, ya que se trata de la información más relevante y que más se usa en estudios y proyectos de márquetin, porque ese dato, a fin de cuentas, es el indicador del éxito de dicha campaña publicitaria. Por ello, la cuantificación de la atención recibida por una campaña es una de las mediciones más útiles y que más buscan los publicistas.

Para conseguir este objetivo, primero crearemos una base de datos, que contendrá un volumen de imágenes considerable de capturas tomadas a diferentes sujetos, con una cámara cenital, en las cuales aparecen las cabezas de diferentes personas con las cuales buscamos realizar unas medidas de la atención. Estas medidas de atención se llevarán a cabo mediante la estimación del ángulo en que miran las personas, y esto se podrá realizar gracias a la orientación de la cabeza del sujeto, que nos permite estimar la dirección donde se fijan las personas. Esta base de datos nos servirá para entrenar y probar un sistema *Deep Learning* creado por nosotros.

Una vez tengamos esa fuente de datos preparada, procederemos con el diseño de una red neuronal convolucional, a la cual entrenaremos y evaluaremos con el objetivo de obtener de ella los ángulos de orientación de las cabezas que aparecen en las imágenes captadas por la cámara cenital.

▪ ABSTRACT

In this project we will apply the mechanisms and techniques that *Deep Learning* technologies offer us, to obtain the attention degree that an advertisement receives from the people, as it is the most important and relevant information used in marketing studies and projects. After all, this result is the indicator of the success of the advertising campaign. That is why the quantification of the attention received by an advertising campaign is one of the most useful and seeked measurements by the advertisers.

To meet this goal, we will first create a database, which will contain a considerable volume of images of captures taken by different individuals, with a zenith camera, in which appear the heads of different people with whom we seek to make some measures of attention. These measures of attention will be carried out by estimating the angle at which people look, and this can be done thanks to the orientation of the head of the subject, which allows us to estimate the direction where people look. This database will help us to train and test a *Deep Learning* system created by us.

Once we have the data source prepared, we will proceed with the design of a convolutional neural network, that we will train and evaluate with the goal of obtaining from it, the angles of inclination and orientation of the heads that appear in the images captured by the zenith camera.

- **CONTENIDOS**

RESUMEN 2
ABSTRACT 3
CONTENIDOS 4
LISTA DE FIGURAS 6
LISTA DE TABLAS 7
GLOSARIO 8
1. INTRODUCCIÓN 9
1.1. Motivación 9
1.2. Objetivos 10
1.3. Alcance 10
1.4. Declaración de honor 11
1.5. Requerimientos 11
2. ESTADO DEL ARTE 12
3. MARCO TEÓRICO 14
3.1. Conceptos previos 14
3.1.1. Machine Learning 14
3.1.1.1. Conjunto de Train / Validation / Test 16
3.1.2. Deep Learning 16
3.1.3. Redes neuronales artificiales 17
3.2. Redes neuronales convolucionales 18
3.2.1. Capas de una CNN 18
3.2.1.1. Capa convolucional 19
3.2.1.2. Capa pooling 20
3.2.1.3. Capa fully-connected 21
3.2.2. Funciones de optimización de una CNN 21
3.2.2.1. RELU 21
3.2.2.2. Softmax 22
3.2.3. Funciones de activación de una CNN 22

3.2.3.1. ADAM 22
3.2.3.2. SGD 23
3.2.4. Funcionamiento de una CNN 23
4. METODOLOGIA 25
4.1. Base de datos 25
4.1.1. Creación 25
4.1.2. Limpieza 27
4.1.3. Normalización 28
4.2. Modelo de la CNN 29
4.3. Obtención de los datos del Dataset 31
4.4. Algoritmo de pérdidas de la CNN 32
4.5. Método general 33
5. EXPERIMENTOS 35
5.1. Entrenamiento 1: Overfitting 35
5.2. Entrenamiento 2: Dataset 35
6. RESULTADOS 37
6.1. Experimento 1 37
6.2. Experimento 2 38
7. PRESUPUESTO 40
8. CONCLUSIONES Y FUTURO TRABAJO 42
9. REFERENCIAS 44
10. ANEXO 46

- **LISTA DE FIGURAS**

Fig. 1 12
Fig. 2 16
Fig. 3 18
Fig. 4 19
Fig. 5 19
Fig. 6 20
Fig. 7 21
Fig. 8 22
Fig. 9 24
Fig. 10 26
Fig. 11 26
Fig. 12 28
Fig. 13 29
Fig. 14 30
Fig. 15 33
Fig. 16 37
Fig. 17 37
Fig. 18 38
Fig. 19 38
Fig. 20 39
Fig. 21 39
Fig. 22 46
Fig. 23 47
Fig. 24 47

- **LISTA DE TABLAS**

Tabla 1 41

Tabla 2 46

- **GLOSARIO**

CNN	**C**onvolutional **N**eural **N**etwork
DNN	**D**eep **N**eural **N**etwork
BBDD	**B**ases de **D**atos
FC	**F**ully **C**onnected
RGB-D	**R**ed **G**reen **B**lue - **D**epth
IA	**I**nteligencia **A**rtificial
GPU	**G**raphics **P**rocessing **U**nit
RELU	**Re**ctified **L**inear **U**nit
SO	**S**istema **O**perativo
SGD	**S**tochastic **G**radient **D**escent

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

En el transcurso del tiempo, diferentes métodos se han establecido para determinar el grado de atención que recibe un anuncio colocado en una pared o edificación. El objetivo de estos es, por un lado, determinar la efectividad del anuncio, ya sea por su contenido o por el lugar de visualización, y por otro lado, poder cobrar con mayor precisión el contenido publicitario. Para que estos métodos sean útiles, tienen que partir de la base de ser lo más baratos y eficientes posibles, para hacerlos competitivos. Además de eso, las mediciones de la atención deben ser no invasivas y adaptables a diferentes ambientes, los cuales suelen estar repletos de objetos.

Algunos de estos métodos parten de colocar una cámara sobre el anuncio, orientada al cliente, para medir si este está mirando activamente hacia el anuncio o no. Esta idea tiene el inconveniente de que se requiere una cámara en cada punto de medición, cosa que hace que el sistema sea pesado y que se pueda ver afectado por oclusiones.

El método que se usa en este proyecto [1], propone medir la audiencia de un anuncio mediante cámaras cenitales, es decir de vista superior, que tengan un ángulo de visión que abarque toda la superficie analizada. Estas cámaras se usarán para medir la trayectoria de las personas y las orientaciones de sus cabezas, con el fin de determinar hacia donde mira la persona, siempre teniendo en cuenta que puede haber posibilidades de error, ya que el sistema calcula la dirección en base a la dirección tomada por la nariz, que suele ser la misma que la de los ojos, aunque puede darse la posibilidad de que la persona mueva los ojos y no se fije en los objetos que tiene en su perpendicular, algo que por otro lado, no es del todo habitual.

En contraste con otros sistemas, este tiene la ventaja de que funciona bien tanto para grandes como pequeñas distancias, es robusto con respecto a la diferencia de estatura entre las personas, y los sujetos analizados pueden estar fijándose en algún punto fuera del campo de visión de la cámara, y aun así poder saber la dirección que toma su cabeza de manera exacta.

Otra mejora que ofrece este método, en comparación con los anteriores es que se trata de un método no invasivo y rentable para calcular mediciones temporales utilizadas para cuantificar la atención recibida por una señal.

1.2. OBJETIVOS

Para desarrollar el método, se van a utilizar algoritmos y técnicas de *Deep Learning*.

La idea principal del proyecto es poder determinar la dirección de visualización de un sujeto, mediante la medición del ángulo de orientación de la cabeza, lo que permite estimar y cuantificar el foco de atención visual que recibe un objeto o un anuncio físico. Para ello se plantean diferentes objetivos:

1. Creación de una base de datos que contendrá un gran volumen de fotogramas, substraídos de grabaciones realizadas previamente, en los cuales aparecerán las cabezas de los sujetos de prueba. Además de las imágenes, la base de datos contendrá las anotaciones de un sensor inercial que tiene la función de *ground truth*, es decir, que marca el ángulo en el que apunta la cabeza.
2. Diseño de un sistema de aprendizaje automático, mediante una red neuronal convolucional, basado en Python, *Keras* [2] y *Tensorflow* [3] como *Backend*.
3. Adecuación de la base de datos, ya sea eliminando tomas incorrectas o normalizando tanto el conjunto de datos como el conjunto de imágenes que la componen.
4. Desarrollo y evaluación de la red neuronal convolucional, para que tenga la capacidad de reconocer la orientación de las cabezas que contienen el conjunto de imágenes de la base de datos, y haga una predicción correcta del ángulo de dirección que toman respecto al eje vertical.

1.3. ALCANCE

Con la obtención de los ángulos de dirección y los de inclinación que tiene la cabeza de un sujeto, sistemas anteriores [1] podrán calcular informaciones relevantes como pueden ser los puntos de interés en los cuales se fija dicho sujeto u otros parámetros muy populares en el mundo de la publicidad, como pueden ser:

- *Tiempo de permanencia (DT)*: Cantidad total de tiempo que un observador pasa en la misma área que la señal que se está evaluando.
- *Tiempo de visualización (IVT)*: Cantidad total de tiempo que el observador está frente al cartel publicitario (no necesariamente prestándole atención).
- *Tiempo de atención/compromiso (AT)*: Cantidad total de tiempo que el observador está mirando activamente la señal. El AT permite cuantificar el grado de atención que ha recibido una señal determinada.

Además de estos valores, también se podrá obtener información adicional, como la distancia de visualización, o el ángulo relativo de visualización, lo que permite un análisis más rico de la escena.

1.4. DECLARACIÓN DE HONOR

Yo declaro que,

El trabajo realizado en esta Tesis de Grado es completamente mi propio trabajo,

Ninguna parte de esta Tesis de Grado está apropiada del trabajo de otras personas sin darles crédito,

Todas las referencias han sido claramente citadas,

Yo entiendo que el infringimiento de esta declaración me deja sujeto a las acciones disciplinarias previstas por la *Universitat Politècnica de Catalunya - BarcelonaTECH*.

Albert Villegas Santos		10/06/2019
Nombre del estudiante	**Firma**	**Fecha**

Título de la tesis: Estudio. Estimación del foco de atención visual mediante trayectorias orientadas.

1.5. REQUERIMIENTOS

Los requerimientos necesarios en la práctica de los procesos descritos en este documento, son pocos, ya que al tratarse del diseño de un prototipo de sistema, el único requerimiento necesario es conseguir unos porcentajes éxitos competitivos, aunque obviamente, el fin de cualquier sistema es conseguir el mayor porcentaje de éxito posible. Otro requerimiento necesario para este tipo de sistemas, es que el tiempo de trabajo del sistema esté lo más cerca posible del tiempo real.

2. ESTADO DEL ARTE

Cada vez es más importante saber qué atrae a la gente, por ello, el grado de atención que recibe un anuncio físico es una de las medidas más importantes para los profesionales del sector, ya que, este dato permite evaluar la efectividad de dicho anuncio.

En el transcurso de los años, tal y como se cita en la introducción, el método usado para la medición de ese grado de atención, consistía en la colocación de una cámara frontal que tuviera la misma orientación que el anuncio, para poder comprobar si una persona miraba de forma activa el anuncio, gracias a la detección de rostros. Este método es muy efectivo, porqué permite analizar la mirada del sujeto, pero tiene el gran inconveniente de que al capturar la cara, el sujeto puede ver violados sus derechos de imagen, además de ser un método que se ve afectado por oclusiones.

Otros métodos que se han usado, que son en los que se basa el nuestro, utilizan cámaras de techo de vista superior. Estos métodos tienen la ventaja de que permiten analizar varios puntos de la superficie de medición, además de resolver inconvenientes del sistema anterior como puede ser el de la privacidad de las personas, pero también presentan algunos inconvenientes, ya que este tipo de medición no permite determinar la edad y el género de los sujetos, unos datos que podrían ser necesarios en estudios de mercado. Los sistemas de medición con cámara cenital, utilizan cámaras RGB-D, es decir *Red Green Blue – Depth*, donde la información *Depth* permite calcular la profundidad del objeto grabado, lo que ha generado la creación de diversos métodos, que parten de esta idea general. Por ejemplo este artículo [16], presenta un sistema que, en base a este método, extrae los esqueletos de los humanos de los datos del sensor de la cámara; los resultados que ofrece este sistema son satisfactorios cuando la aplicación de este es en cámaras en movimiento.

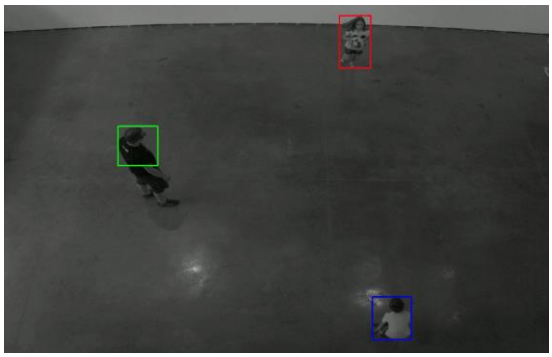

***Fig. 1**: Visión periférica de una cámara cenital. La detección es diferente en comparación con una cámara frontal.*

En cualquier sistema de reconocimiento, la detección es una de las partes fundamentales del proyecto, ya que una buena detección permite al sistema tener una mayor calidad y cantidad en el reconocimiento. En nuestro caso, los objetos a detectar son las personas, más concretamente sus caras y cabezas.

Para cumplir este fin, se han desarrollado diferentes métodos de detección muy notables, como puede ser el caso del Viola-Jones [17], que es el

primer detector de objetos que tuvo una tasa de detección competitiva a tiempo real. Además es un detector que puede ser entrenado para detectar una gran variedad de objetos, aunque principalmente, la motivación de este fue la detección de rostros.

Otro de los métodos de detección más usados es el HOG [18]. Este tipo de detector es en el cual se basa la librería de Python llamada *dlib*. Las siglas HOG vienen de *Histogram of Oriented Gradients*, lo que significa histograma de gradientes orientados. El gradiente nos proporciona información de los cambios de intensidad en una imagen. El HOG es un descriptor global que convierte la información local del gradiente de cada uno de los pixeles, en una representación global de toda la imagen en forma de vector de características que captura la forma del objeto.

Anteriormente se ha comentado que la detección era una de las partes más importantes de un sistema de reconocimiento. La otra parte fundamental de estos sistemas, es la clasificación de estas detecciones, es decir, una de las grandes utilidades que tiene la Inteligencia Artificial.

Dentro del gran grupo de disciplinas engloba la IA, se encuentran el aprendizaje automático de las máquinas, conocido como *Machine Learning*, y el aprendizaje profundo, conocido como *Deep Learning*, que se dedican principalmente, al desarrollo de técnicas y algoritmos que permiten a los sistemas aprender de la experiencia que estos toman, tal y como hacemos los humanos.

Las redes neuronales, una tecnología del campo del *Deep Learning*, se han popularizado por al conseguir resultados muy impresionantes en el reconocimiento de imagen. Esta tecnología está en constante evolución, y ampliando cada vez su radio de acción. Los primeros fundamentos sobre esta tecnología, se empezaron a implementar en la década de los 80, aunque, como es obvio, la limitación tecnológica que había en la época, ya que los conjuntos de datos eran sumamente pequeños, al igual que la potencia de los procesadores, no permitió que esta tecnología tuviera un ámbito en el que ser usada. Es por ello que, la capacidad de generar grandes volúmenes de datos, y las mejoras en los procesadores, juntamente con la llegada de las GPUs, unidades de procesamiento gráfico, ha propiciado la irrupción de las redes neuronales convolucionales en la actualidad, conocidas como CNN, porque hoy en día, sí que tienen muchas utilidades, además de haber mejorado sus prestaciones.

3. MARCO TEÓRICO

3.1. CONCEPTOS PREVIOS

A lo largo de la existencia, los humanos hemos pasado por diferentes etapas, en las cuales el desarrollo de nuestro entorno y de nosotros mismos ha ido evolucionando hasta llegar al día de hoy. Cada una de esas etapas que los seres humanos han vivido, se ha bautizado con un nombre ateniéndose a las circunstancias de la época. Sin duda alguna, uno de los nombres con el que se podría bautizar a la época actual es el de 'La era del dato', y es que en la actualidad, en cualquier aspecto de la vida lo que importa es el dato. Cualquier campo de nuestra vida está regido por datos, des del mundo empresarial, hasta los deportes, pasando por la sanidad.

Ese gran volumen de información a la que se puede acceder hoy en día, mediante las nuevas tecnologías existentes, conocido como *Big Data* [4] ha creado en la sociedad una necesidad de gestionar dichos datos, ya que las herramientas existentes no estaban capacitadas para procesar y gestionar grandes cantidades de datos. A causa de esta necesidad, surgen las tecnologías de aprendizaje automático.

3.1.1. MACHINE LEARNING

El concepto de *Big Data*, analizado previamente, ha creado una necesidad en la sociedad, ya que todos esos volúmenes de datos gigantescos, al fin y al cabo no son más que datos si no se puede trabajar con ellos. El ser humano no tiene la capacidad suficiente como para trabajar con tales cantidades de información en un intervalo de tiempo pequeño, pero en cambio, las máquinas y sistemas sí. Es por ello, que surge el concepto de *Machine Learning* en nuestras vidas.

Machine Learning, o aprendizaje automático, es un subcampo de las ciencias de computación y las neurociencias, y una de las ramas más importantes de la *Inteligencia artificial*, que tiene el objetivo de analizar datos gracias al desarrollo de técnicas y modelos analíticos que permiten a los sistemas aprender de esos datos. En conclusión, lo que se buscaba cuando se empezó a hablar de *Machine Learning*, era un sistema con el cual las personas pudieran trabajar de la mano, pero que llegara a objetivos y conclusiones a las que el humano no puede llegar, para ello, se crearon estos sistemas que tuvieran la capacidad de aprender sin ser explícitamente programados, simulando la capacidad de aprendizaje que tiene el humano.

Existen diferentes métodos o algoritmos de *Machine Learning*, dependiendo del objetivo que tiene el sistema, o del contenido del flujo de datos existente, los más importantes son:

1. **Aprendizaje supervisado:** Los algoritmos que trabajan con este tipo de aprendizaje tienen la característica principal de que son entrenados mediante ejemplos etiquetados, con el fin de asegurar la efectividad y calidad de los datos, es decir, lo que se busca es un algoritmo que establezca una correspondencia entre los datos de entrada y las salidas deseadas del sistema. Con el contenido de las etiquetas, el sistema puede comprobar si su predicción ha sido correcta, y si no ha sido el caso, puede aprender de los errores porque sabe cuál debería ser la respuesta correcta. Este tipo de aprendizaje se usa normalmente en problemas de clasificación, como podría ser sistema que diferencia entre perros y gatos.

   Este tipo de aprendizaje es el usado por la CNN diseñada en este proyecto.

2. **Aprendizaje no supervisado:** A diferencia del tipo anterior, en esta categoría no se usa ninguna etiqueta que permita al sistema comprobar el resultado de su predicción, así que el algoritmo no cuenta ninguna indicación previa. En este tipo de aprendizaje no se le ofrece una respuesta correcta al sistema, sino que la función de este es descubrir lo que contienen los datos, y a partir de ahí, buscar alguna estructura o patrón en el contenido, con el objetivo de etiquetar nuevas entradas. Este tipo de aprendizaje se usa para diversas funciones como puede ser la segmentación de temas de texto, o en problemas de clasificación.

3. **Aprendizaje por refuerzo:** Se trata del tipo más general de las tres categorías, y la característica principal de este tipo de aprendizaje es que se basa en el refuerzo, es decir, la maquina aprende en base a experiencias prueba y error. Los sistemas que utilizan este tipo de aprendizaje, son conocedores des del principio de los resultados que deben obtener, pero no saben cuáles son las mejores decisiones que deben tomar para llegar a ellos. Existen tres componentes principales en este tipo de aprendizaje:

   1. Agente: Es el componente que aprende o toma decisiones.
   2. Entorno: Conjunto de cosas que interactúan con el agente.
   3. Acciones: Funciones que el agente puede hacer.

   La idea para que el sistema aprenda es que el agente elija acciones que maximicen la recompensa deseada en un intervalo de tiempo, el cual será menor si el agente aplica una buena manera de proceder. Si la acción elegida minimiza la recompensa deseada, el sistema aprenderá y tomara otra decisión.

3.1.1.1. CONJUNTO DE TRAIN / VALIDATION / TEST

En problemas tanto de *Machine Learning*, como de *Deep Learning*, habitualmente se divide el volumen de datos en tres conjuntos: datos de entrenamiento (*training*), datos de validación (*validation*) y datos de prueba (*test*).

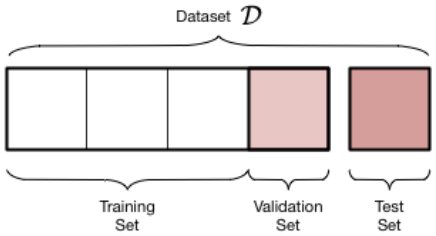

***Fig. 2**: Conjuntos de entrenamiento, validación y test. El conjunto de entrenamiento siempre es el que más datos contiene, mientras que los conjuntos de validación y test tienen en sus conjuntos un pequeño porcentaje del volumen de datos general.*

El conjunto de entrenamiento son los datos que el algoritmo de aprendizaje usa para obtener ciertos parámetros y características del modelo. Cabe la posibilidad de que el entrenamiento del modelo no se esté efectuando de la manera que deseamos, ya que a veces esta etapa puede no ser correcta, o presentar *overfitting*. Si esto ocurre será necesario ajustando ciertos valores de los hiperparámetros del modelo, y una vez ajustados, reiniciar el entrenamiento.

El conjunto de validación tiene una función parecida al conjunto de test, ya que la idea de esta etapa es evaluar el funcionamiento del modelo para un conjunto de datos que no son los de entrenamiento. Cabe la posibilidad también, de que el sistema se aprenda las características del volumen de datos de validación, por este motivo reservamos siempre un conjunto de datos de prueba

El conjunto de test tiene la función principal de evaluar el modelo. Está compuesto por un conjunto de datos totalmente diferentes a los del conjunto de entrenamiento y de validación, lo que provocará que el sistema analice entradas que no conoce. Este conjunto de datos se usarán al final de todo el proceso, cuando consideremos que el modelo está acabado de afinar y ya no modificaremos más ninguno de sus hiperparámetros, con el fin de realizar una evaluación final del sistema.

3.1.2. DEEP LEARNING

El **Deep Learning**, aprendizaje profundo, es un concepto en auge actualmente, y se trata de un conjunto de algoritmos, de clase *Machine Learning*, ideados para el aprendizaje automático de los sistemas. Se trata

de algoritmos automáticos estructurados, que tienen la función de emular o copiar el aprendizaje humano con el fin de obtener ciertos conocimientos, y estos sistemas tienen la característica principal de que no requieren de reglas pre-programadas, sino que el propio sistema es capaz de entrenarse con el fin de aprender por sí mismo. El *Deep Learning* tiene en la actualidad muchas aplicaciones, ya que esta tecnología se usa por ejemplo en traductores inteligentes, reconocimiento de voz, reconocimiento facial, visión computacional, etc.

Este modelo de aprendizaje brilla allí donde haya muchos datos, y problemas complejos que resolver. Tiene diferentes características que lo hacen bueno y singular:

- *Adaptabilidad*: Tiene técnicas que se adaptan a diferentes tipos de datos y a diferentes tipos de problemas
- *Infraestructura común / Lenguaje común*: Todos los modelos usan el mismo tipo de infraestructura y el mismo tipo de lenguaje para describir las cosas, algo que facilita mucho su uso.

La diferencia que existe entre el concepto de *Deep Learning*, y el de *Machine Learning*, es que, pese a tratarse de dos tecnologías que hacen referencia a sistemas que tienen la capacidad de aprender por si solos, el método de aprendizaje del *Deep Learning* es más complejo, más sofisticado y más autónomo, lo que significa que una vez programado el sistema, la intervención del humano es mínima. Es por ello, que el *Deep Learning* no deja de ser una vertiente de *Machine Learning*.

Dentro del conjunto del *Deep Learning* existen diferentes arquitecturas de aprendizaje profundo, algunas de las cuales, serán las usadas en este estudio, con el fin de estimar el foco de atención visual de la personas, mediante trayectorias orientadas. Tenemos redes neuronales profundas, redes de creencia profundas, y redes neuronales convolucionales profundas, que será el tipo de red que se usara en este estudio. Todas estas arquitecturas se aplican a diferentes como puede ser el reconocimiento automático del habla, reconocimiento de señales de audio y música, o la visión por computador, que es nuestro caso.

Previamente a conocer que son las redes neuronales convolucionales, es conveniente saber y conocer que es una red neuronal.

3.1.3. REDES NEURONALES ARTIFICIALES

Una **red neuronal artificial** es un modelo computacional inspirado en el comportamiento observado en el equivalente biológico, es decir los humanos. Consiste en un conjunto de unidades, llamadas neuronas artificiales, que están conectadas entre sí, con la intención de transmitirse señales. Una neurona es la unidad mínima de una red neuronal, y tiene

como entrada una combinación lineal de pesos, a los cuales se les aplica una activación, la cual es una función no lineal, en la salida de la neurona.

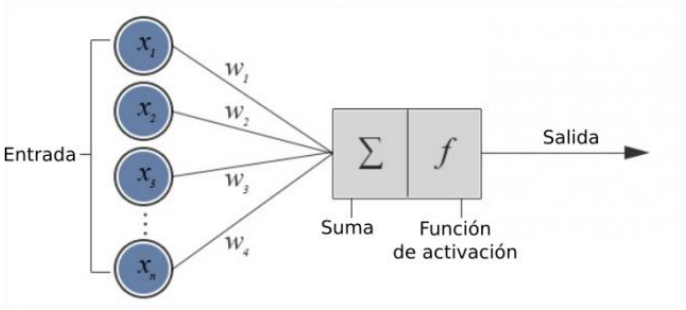

***Fig. 3**: Forma de una neurona.*

Los algoritmos con redes neuronales se suelen usar para problemas de clasificación y regresión, pero realmente tienen un gran potencial para resolver multitud de problemáticas, entre las cuales destaca el reconocimiento de patrones.

Posteriormente, en el apartado 3.2.4, se explicará el funcionamiento de las redes neuronales, en particular el caso de las convolucionales, pero a grandes rasgos, lo que hace es recibir un estímulo del entorno, que en este caso es lo que se consideran las entradas, y la salida que ofrece dicha red neuronal, no es más que una respuesta a ese estímulo. Las neuronas de la red se adaptan al medio, modificando así el peso de sus valores sinápticos.

Este tipo de tecnología, al formar parte la rama del *Deep Learning*, tiene la característica principal, de que para poder configurar y evaluar el modelo correctamente, la fuente de datos deber ser dividida de una manera particular.

3.2. REDES NEURONALES CONVOLUCIONALES

Las **redes neuronales convolucionales** [5], también denominadas CNN, son una clase de redes neuronales artificiales profundas que se han adaptado ampliamente en varias aplicaciones de visión artificial e imágenes visuales, ya que son la herramienta perfecta para trabajar con imágenes de cámara.

3.2.1. CAPAS DE UNA CNN

Las CNN están formadas por múltiples capas diseñadas para requerir un pre-procesamiento relativamente pequeño en comparación con otros algoritmos de clasificación de imágenes. Estas capas se estructuran en un capa de entrada, capa *input*, por varias capas ocultas, capas *hidden*, y finalmente por una capa de salida, capa *output*.

Dentro de las capas *hidden*, en las CNN destacan tres tipos de capas.

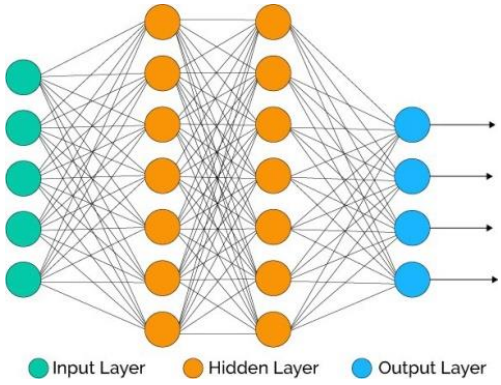

***Fig. 4**: Esquema básico de una CNN.*

3.2.1.1. CAPA CONVOLUCIONAL

La capa convolucional es un conjunto de capas intermedias que forman el núcleo de una CNN, y consisten en un conjunto de filtros, con un pesos ajustados según la necesidad que se tenga, que se desplazan por el conjunto de datos de entrada, a lo ancho y a lo largo en pequeñas ventanas, y que permite a la red neuronal convolucional, identificar patrones en la imagen. A medida que se avanza en la capas del sistema, los datos identificados o reconocidos son más profundos y singulares. Estos filtros se caracterizan por tener una altura y anchura inferiores a los datos del volumen de entrada de la capa, pero sí que comparten la profundidad.

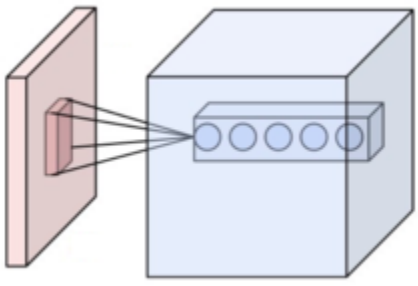

***Fig. 5**: Cada una de las neuronas de la CNN está conectada a una región local de los datos de entrada. La zona roja corresponde a la entrada, y la azul a la capa convolucional.*

La característica principal de este tipo de capas, es que cada neurona está conectada únicamente con una región local, llamada *receptive field*, del volumen de datos de entrada. Al tener solo una conexión local, las neuronas de distintas capas no están conectadas entre ellas, por lo tanto mantienen las características de la región local analizada, es decir, se comparten los pesos entre todas las neuronas que estén englobadas en el mismo *Depth* del volumen de salida.

La capa convolucional tiene una serie de hiperparámetros que son les encargados de la distribución de esta capa:

a) Profundidad o *Depth*: Corresponde al número de filtros diferentes que tiene cada capa convolucional.

b) *Stride*: Cantidad de píxeles que se desplaza el filtro. Por ejemplo, teniendo un stride igual a 2, se desplazaría el filtro a través del volumen de entrada con saltos de 2 píxeles.

c) *Zero-padding*: Conjunto de ceros que se añaden al borde del volumen de entrada. Este parámetro permite controlar las dimensiones (altura y anchura) del volumen de salida.

d) Tamaño del filtro: Corresponde a la anchura y altura del filtro representada en píxeles.

3.2.1.2. CAPA POOLING

La capa de *pooling*, también llamada de reducción, se coloca generalmente después de la capa *convolucional*, y tiene la función principal de reducir las dimensiones espaciales de los mapas de activación que se obtienen de cada filtro, con el objetivo de mantener la información más relevante. Puede parecer contraproducente el hecho de hacer una reducción de la información obtenida por la capa *convolucional*, ya que se puede pensar que la pérdida de información en esta capa, se puede traducir en un descenso de la precisión del modelo, pero nada más lejos de la realidad. Estas pérdidas también pueden ser beneficiosas para el modelo, ya que, al reducir los parámetros en la red, podemos controlar el sobre-enteramiento de esta, además de hacerla más pequeña en memoria.

La reducción se realiza mediante la aplicación de una función de *pooling*, sobre cada *depth slice* del volumen de entrada. Esto provoca una redimensión espacial de la entrada.

Existen diversas funciones de *pooling*, que ofrecen diferentes posibilidades según el objetivo que tengamos con nuestra CNN. Las funciones más comunes son el *average pooling* y el *max pooling*. Tal y como indican los nombres de estas funciones, la primera obtiene el valor medio de cada iteración como representante, y el segundo se queda con el valor máximo de cada iteración por el volumen de entrada.

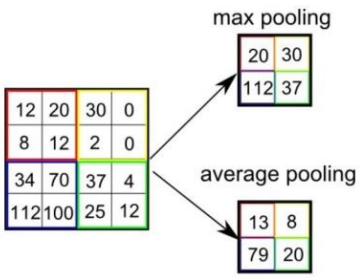

***Fig. 6**: Ejemplos de funciones de la capa pooling. MaxPooling se queda con el valor máximo de cada región, y AveragePolling se queda con el valor medio de cada región.*

Esta capa, al igual que la anterior, tiene una serie de hiperparámetros que son la guía para su manera de actuar:

a) Tamaño del filtro: Corresponde a la anchura y altura del filtro representada en píxeles.

b) *Stride*: Cantidad de píxeles que se desplaza el filtro. Por ejemplo, teniendo un stride igual a 2, se desplazaría el filtro a través del volumen de entrada con saltos de 2 píxeles.

3.2.1.3. CAPA FULLY-CONNECTED

En esta última capa, tal y como indica su nombre, las neuronas de las capas previas se conectarán con las neuronas de la capa actual, con el fin de considerar cada píxel, como una neurona separada.

Este tipo de capas se suelen añadir al final de los modelos de CNN, con el objetivo de poder tratar todas las características extraídas en las etapas anteriores, de la manera que se desee según los objetivos del modelo.

3.2.2. FUNCIONES DE ACTIVACIÓN DE UNA CNN

Las funciones de activación son una de las partes más importantes del funcionamiento de las redes neuronales convolucionales. Una activación es una función no lineal que se encarga de devolver una salida en base a unos valores de entrada. Este conjunto de valores de salida serán de una forma u otra dependiendo del tipo de función de activación usada.

Existen diferentes funciones de activación, o *activation function*, dependiendo de la meta del sistema.

3.2.2.1. RELU

$$f(x) = \max(0, x) = \begin{cases} 0 \; for \;\; x < 0 \\ x \; for \;\; x \geq 0 \end{cases}$$

***Fig. 7**: Fórmula matemática de la función de activación RELU.*

La función *RELU*, siglas de *Rectified Linear Unit*, es la activación más usada en problemas de regresión. Esta función transforma los valores de entrada anulando los valores negativos, y dejando los positivos tal y como se introducen. Este tipo de activación tiene la ventaja de que se comporta bien en imágenes, pero a la vez tiene el inconveniente que al descartar valores negativos, muchas neuronas pueden morir, cosa que dependiendo de la necesidad que se busque cubrir, puede derivar en un problema.

3.2.2.2. SOFTMAX

La función *Softmax*, es la activación más usada en problemas de clasificación. Esta función transforma las salidas a una representación en forma de probabilidades, con el objetivo de que el sumatorio de todas esas probabilidades de 1. Esta función ofrece la ventaja de tener gran rendimiento en las últimas capas de los modelos, de ahí que sea una activación perfecta para problemas de clasificación.

$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

***Fig. 8**: Fórmula matemática de la función de activación Softmax.*

3.2.3. FUNCIONES DE OPTIMIZACIÓN DE UNA CNN

Las funciones de optimización de una CNN también son uno de los parámetros importantes a la hora de entrenar un modelo, ya que es requerimiento indispensable, pasarle al método *fit*, que es el encargado de entrenar un modelo, un parámetro correspondiente a la función *loss*, y otro parámetro, que es el optimizador.

Estas funciones no son más que una serie de algoritmos programados para que la CNN aprenda de manera optimizada y adaptada. Uno de los parámetros que nos ofrece estos algoritmos, es el conocido como *Learning Rate*, es decir, tasa de aprendizaje a la cual va a evolucionar nuestra CNN. La librería *Keras* [2] ofrece diferentes algoritmos de optimización, y los más utilizados son los siguientes.

3.2.3.1. ADAM

Este algoritmo de optimización es una extensión del optimizador SGD, y tiene grandes resultados en aplicaciones de visión por computadora de *Deep Learning*, y también destaca en tecnologías del habla.

A diferencia de la optimización mediante el SGD, *stochastic gradient descent*, que mantiene un *learning rate* para todas las actualizaciones de los *weights* de las neuronas durante el entrenamiento, el optimizador Adam mantiene en un principio el *learning rate* para cada peso de la red, pero a medida que va evolucionando el entrenamiento, este valor se adapta por separado.

Entre las características principales de este optimizador, destaca la eficiencia de este y la sencilla implementación que tiene, además de requerir poca memoria durante su uso.

Los expertos describen a este optimizador como una combinación de las ventajas de otras extensiones del SGD, como son el Adagrad y el RMSprop, de ahí que tanga tan buenas prestaciones. Es por ello que este ha sido el optimizador elegido en el entrenamiento de la CNN.

3.2.3.2. SGD

Este optimizador es el más utilizado en problemas de *Deep Learning*. El descenso de gradiente estocástico, conocido como SGD, es un método iterativo de optimización para una función objetivo derivable. Este optimizador se basa en encontrar mínimos y máximos en cada una de las iteraciones. De esta manera, el gradiente indica la dirección en la cual la función de perdidas tiene el ratio de aumento más pronunciado. Una vez conocida la dirección a tomar, aparece el campo de *learning rate*, que determina la distancia que recorrerá en cada iteración en la dirección del gradiente [19].

Este tipo de optimizador tiene una velocidad de ejecución competitiva, y además puede ser usado para aprendizajes *online*. SGD realiza actualizaciones frecuentes de los *weights* con una alta variación, lo que provoca que la función objetivo fluctúe fuertemente.

3.2.4. FUNCIONAMIENTO DE UNA CNN

Para entender el funcionamiento que tiene una red neuronal convolucional, es necesario conocer el concepto de *forward pass*. Las capas que forman una red neuronal convolucional tienen el objetivo de ir transformando el volumen de activaciones que tienen en su entrada en otro volumen distinto, mediante funciones diferenciables, hasta llegar a la última capa. Este paso es conocido como *forward pass*.

Una red neuronal convolucional, no deja de ser un sistema que recibe un volumen de datos como entrada, a los cuales se aplica diversas operaciones matemáticas, como son las funciones no lineales, mientras van pasando por las diferentes capas.

En la capa convolucional, que es la que da nombre a este tipo de redes neuronales, el *forward pass* se realiza desplazando los diferentes filtros presentes en la capa, a través de la altura y anchura del volumen de entrada. A medida que estos filtros van desplazándose, se realiza una convolución 2D, es decir, se aplican productos escalares entre los distintos coeficientes del filtro y su entrada. Esta convolución creará, por lo tanto, un mapa de activación. Este mapa, será el resultado de las distintas respuestas del filtro en cada posición del espacio. De esta manera, a medida que se entrena la red, se aprenderán coeficientes en el filtro que se activarán en determinadas ocasiones, como por ejemplo con contornos, manchas de color, etc. Con cada filtro se obtiene un mapa de activación distinto que se irá colocando a lo largo de la profundidad del volumen de salida de la capa.

Todo el *forward pass* sirve para calcular unos pesos que deberán ajustar según el fin de la CNN. Este ajuste dependerá de la función de pérdidas del sistema, ya que esta función es la encargada de mostrar la diferencia que

existe entre las predicciones que hace el sistema, y las que debería hacer. Si el valor de esta función es mínimo, significa que el sistema trabaja correctamente y realiza unas predicciones muy acertadas. Para bajar estas pérdidas que pueda tener el sistema se recurre al método de descenso del gradiente, conocido como *gradient descent* [6].

El algoritmo de *Gradient Descent*, permite calcular el gradiente de la función de pérdidas con respecto a los coeficientes de los filtros. Esto permite actualizar los pesos de la CNN a medida que se calcula los nuevos valores.

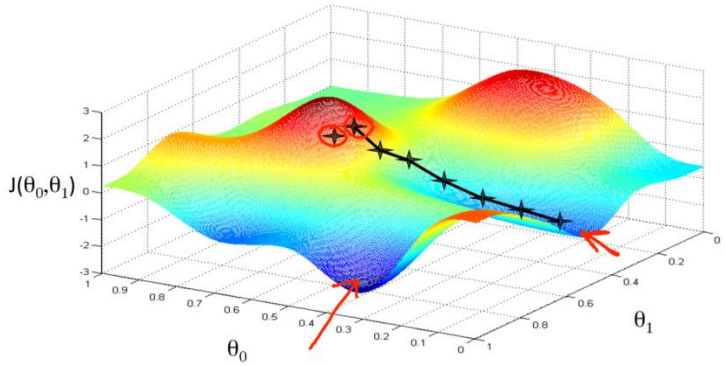

***Fig. 9**: Mapa de activación en el cual trabaja el algoritmo Gradient Descent. Partiendo del pico rojo, la idea del algoritmo es disminuir la pérdida del modelo hasta llegar al valle azul.*

Para finalizar con este el funcionamiento de las CNN, es necesario explicar el método de entrenamiento que estas tienen. Las CNN se entrenan mediante épocas, conocidas como *epochs*. Estas épocas, están compuestas por un paso de *forward* y un paso de *backward*. El paso de *forward* se divide en iteraciones, donde en cada una de ellas, se calcula para un *Batch* diferente: el error de predicción y el gradiente. Una vez se han realizado todos los cálculos sobre todos los *Batch*, se produce el paso de *backward*, dónde se actualizan los pesos de la red. El número de *epochs* es un hiperparámetros de la red, ya que de él depende que la red este sobre-entrenada o infra-entrenada.

4. METODOLOGIA

Para la realización del estudio, se han desarrollado una serie de tareas que serán explicadas en este apartado. El proyecto, a nivel práctico, ha madurado con el trabajo simultáneo en dos tareas principales, que son la *base de datos* y el *modelo de la red neuronal convolucional*. Paralelamente a estas dos tareas, se ha estado trabajando en la creación de programas en Python para la *obtención de los datos del Dataset*, *algoritmo de perdida de la CNN* y un *script general* de trabajo.

4.1. BASE DE DATOS

Las redes neuronales convolucionales tienen una manera de proceder muy simple, ya que una vez el modelo está creado, simplemente es necesario introducirle al sistema un conjunto de datos de entrenamiento, uno de validación, y finalmente uno de test, el cual nos servirá para evaluar nuestro sistema. La creación de la base de datos ha sido una parte fundamental del proyecto, ya que, gracias a las grabaciones que se han ido haciendo durante un período de tiempo, hemos recopilado suficientes imágenes como para poder entrenar al sistema de una manera efectiva.

4.1.1. CREACIÓN

Este proyecto, no es más que una parte de un proyecto de doctorado de mis mentores y profesores, de ahí que la creación de la base de datos ha estado una actividad conjunta con mi cotutor Manuel López Palma.

Para la creación del conjunto de datos, ha sido necesaria, obviamente una cámara cenital que tenga la capacidad de tomar fotogramas en *depth*. La cámara usada en el proyecto ha estado la *Intel RealSense Depth Module D415* [7], una tecnología de alta calidad capaz de ofrecer grandes resultados. Esta estaba colocada en la parte central del techo de la sala de pruebas, que fue el Plató TV del TR1. Una vez comprobado el campo de visión que esta tenia, se colocaron diferentes marcas en el suelo para marcar ese máximo campo de visión, además de diversos objetos que simularían los elementos a los cuales las personas nos fijamos en el algún punto. Con el escenario ya casi preparado, solo faltaba la colocación de una cámara lateral, que tendría la función de chequear las mediciones, para comprobar que todo funcionaba correctamente.

Para la detección de las cabezas, se ha utilizado un sistema de detección que parte de realizar una substracción del fondo mediante la imagen de *depth* y el modelo del fondo, el cual se ha obtenido grabando la escena sin ninguna persona en ella. Entonces se genera una máscara binaria con las zonas que contienen diferencias respecto al fondo. Cabe la posibilidad de que en la escena haya diferentes objetos, los cuales son elementos que no

buscamos, por ello, se eliminan de la máscara binaria todos los pixeles cuyo valor de profundidad sea mayor que el de un umbral determinado, el cual se define mediante la altura mínima de una persona. A continuación se eliminan pequeñas manchas en la imagen, ya que probablemente representan pequeños objetos que no forman parte del cuerpo humano. Finalmente, para determinar el nivel de profundidad de la cabeza, se calcula el histograma mínimo entre los dos picos de profundidad, de los cuales uno corresponde a los hombros, y otro a la cabeza, y se mantienen solo los pixeles que tienen un nivel de profundidad por encima del mínimo.

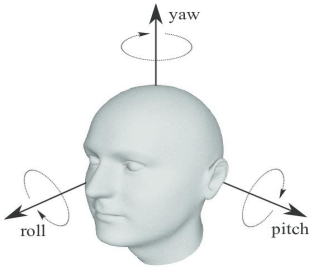

***Fig. 10**: Ángulos yaw, roll y pitch del movimiento de una cabeza humana.*

El procedimiento para la toma de capturas de datos, es decir, los ángulos de posición del sujeto, es muy simple. El sujeto en cuestión parte de una posición de inicio conocida, que después será útil para el cálculo de los ángulos, y se coloca una cámara en la barbilla que contiene un sensor inercial en su estructura. Esta cámara está colocada en la barbilla del sujeto como en la Fig. 11, simulando el mismo campo de visión que tiene el sujeto, lo que provoca que el foco de la cámara esté grabando prácticamente el mismo punto en el que los ojos del sujeto están mirando. Esta estructura, nos permite calcular durante todo el tiempo el ángulo de dirección de la cabeza, conocido como *yaw*, el de elevación, conocido como *pitch*, y el de inclinación, conocido como *roll*.

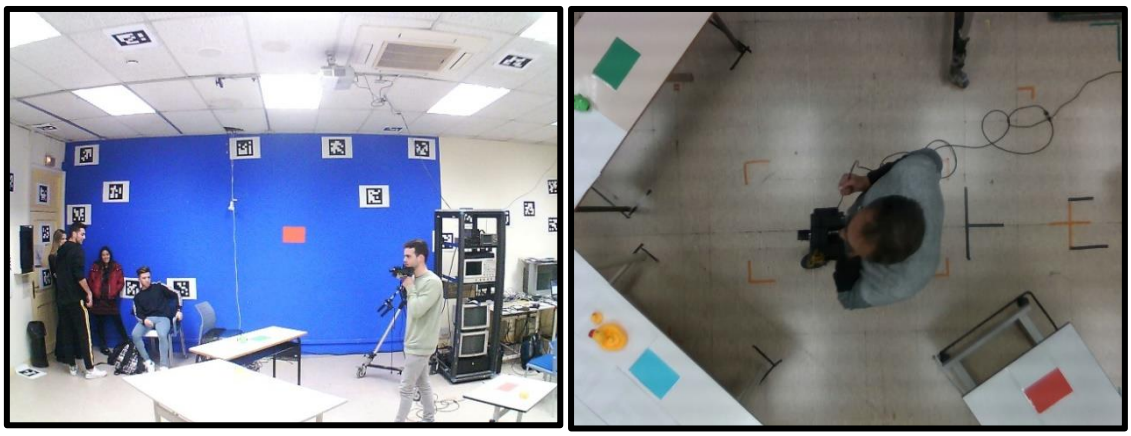

***Fig. 11**: Disposición del Plató del TR1 durante la época de toma de capturas. En la imagen de la izquierda se aprecia la cámara cenital [7] en el techo, y la estructura de cámara + sensor de inercia, en la barbilla del sujeto. En la derecha se aprecia la imagen captada por [7] en ese momento.*

La manera de proceder que debía tener el sujeto era muy simple, ya que solamente debía moverse, durante al menos un minuto, por el espacio marcado en el suelo, que era el correspondiente al campo de visión de la cámara, y debía ir mirando los objetos de muestra colocados a tal fin. Al

mismo tiempo, el sensor de inercia va captando los ángulos ya citados, y mediante ROS [8], se adquieren esos ángulos.

Por tanto, una vez tomados los datos, se pasaba al proceso de estos e inclusión en la base de datos.

La base de datos, conocida como BBDD, tiene una estructura de árbol muy simple. Existen múltiples carpetas, cada una de ellas corresponde a un sujeto diferente. Dentro de cada uno de esos directorios, nos encontramos con un seguido de carpetas que contienen diferentes datos:

1. *Images1*: Este directorio contiene todos los fotogramas capturados en la grabación en formato color, en RGB, y con un tamaño de 640x480 en cada imagen.
2. *Depths1*: Este directorio contiene todos los fotogramas capturados en la grabación en formato *depth*, en escala de grises, y con un tamaño de 640x480 píxeles en cada imagen.
3. *ColorDataBase*: Este directorio contiene cada uno de los fotogramas que forman la carpeta *Images1*, pero estos se encuentran recortados por la zona del campo de visión de la imagen, en la cual aparece la cabeza del sujeto. Estas imágenes redimensionadas, tienen un tamaño de 240x240 píxeles.
4. *DepthDataBase*: Este directorio contiene cada uno de los fotogramas que forman la carpeta *Depths1*, pero estos se encuentran recortados por la zona del campo de visión de la imagen, en la cual aparece la cabeza del sujeto. Estas imágenes redimensionadas, tienen un tamaño de 240x240 píxeles.
5. *Fronts1*: Este directorio contiene cada uno de los fotogramas captados por la cámara frontal, colocada en la barbilla del sujeto, durante la toma de medidas. Las imágenes que contiene esta carpeta simplemente tienen la función de chequeo para el sistema, ya que no cumplen ninguna función a nivel práctico.

El procesamiento de los datos capturados en el entrenamiento, además de estos directorios descritos, añaden a cada carpeta de cada sujeto, un fichero con extensión .txt, que contiene los ángulos, tomados por un sensor inercial, en cada uno de los fotogramas capturados. Este fichero, llamado *dat.txt*, nos permitirá etiquetar las imágenes para entrenar nuestra CNN.

4.1.2. LIMPIEZA

Durante la toma de medidas y el posterior procesamiento, ha habido diversos fotogramas, que el sistema ha recortado pensando que eran cabezas y en las imágenes no aparecía nada. Estos es lo que se llaman falsos positivos, imágenes que el sistema reconoce como correctas pero en realidad no lo son. Lo que se debe hacer con este conjunto de datos obtenidos de manera errónea es eliminarlos de la BBDD, ya que si se

mantienen en ella pueden alterar el rendimiento de la CNN en el entrenamiento. Esta es la parte más costosa en cuanto a la BBDD se refiere, ya que los falsos positivos, hay que eliminarlos manualmente uno a uno de cada uno de los directorios que componen la BBDD, así como del archivo dat.txt que contiene las etiquetas de cada fotograma.

4.1.3. NORMALIZACIÓN

Una vez teníamos nuestra base de datos limpiada y completada, aún faltaba algún paso de adecuación de la BBDD más, para poder trabajar con ella. Y es que nuestra red neuronal convolucional, solamente necesita algunos de los parámetros y datos que la BBDD ofrece, por tanto hay muchos datos de los que esta contiene que no se usan en este estudio, de ahí que haya sido vital hacer una normalización de los datos de la BBDD.

El *Dataset*, que se le va a ofrecer a la CNN es mucho más simple que lo que nos ofrece la BBDD creada. Los campos para entrenar a la CNN son:

1. Conjunto de imágenes en *depth* de los fotogramas, que se corresponde con la carpeta *DepthDataBase*.
2. Archivo *dat.txt*, que se corresponderá a los valores de etiquetado de las imágenes.

Estos dos conjuntos de datos presentan un formato a priori incompatible con la CNN por diferentes motivos. En cuanto al volumen de imágenes, esta todo correcto a excepción de los nombres de cada fotograma, ya que el formato correcto para nuestro proyecto es *x_y.png*, donde x se corresponde con el número de sujeto, e y se corresponde con el número de fotograma.

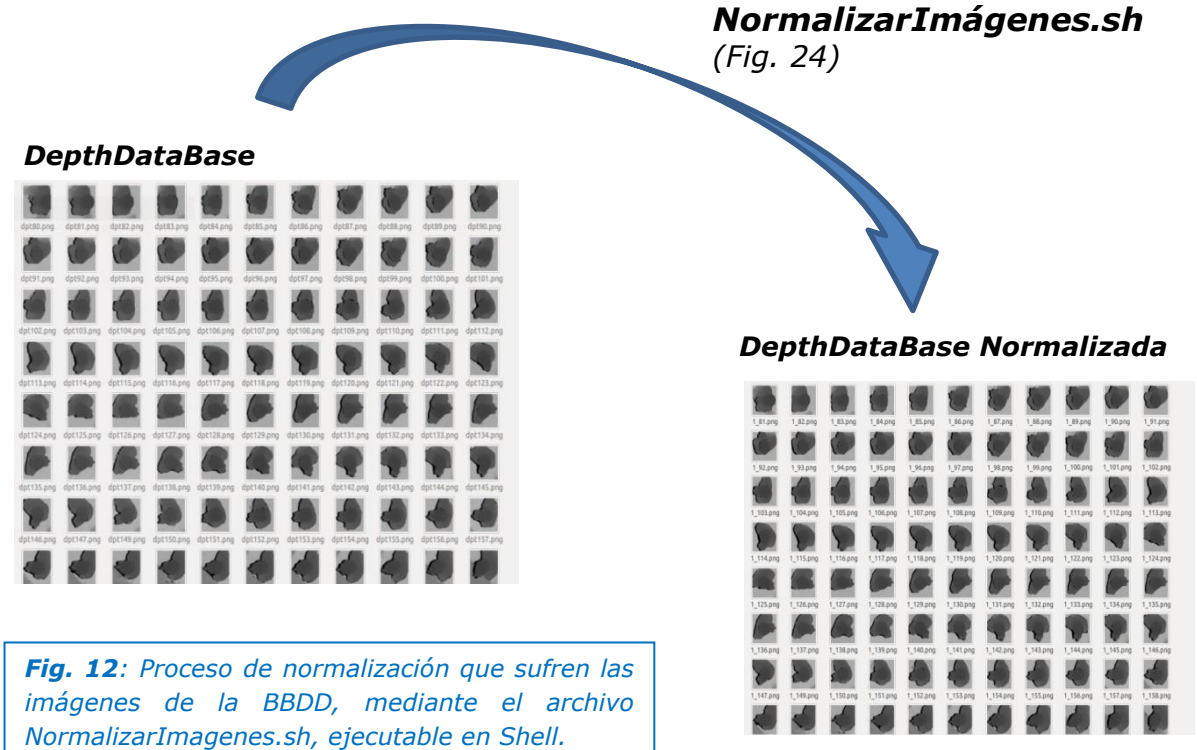

***Fig. 12**: Proceso de normalización que sufren las imágenes de la BBDD, mediante el archivo NormalizarImagenes.sh, ejecutable en Shell.*

El archivo *dat.txt*, ha sido bastante más costoso de normalizar, ya que el formato que este ofrecía era el de la Fig. 13. Los datos que aparecen en el fichero tienen una disposición incompatible con el modelo de una CNN, ya que el formato que aparece no se asemeja en nada al buscado. El formato que se necesita es *"x y z c v"*, donde *x* se corresponde con el número de sujeto, *y* se corresponde con el número de fotograma, *z* se corresponde con el ángulo *yaw*, *c* que se corresponde con el ángulo *pitch* y finalmente *v* que se corresponde con el ángulo *roll*, aunque tanto los ángulos *pitch*, como *roll*, no serán necesarios posteriormente.

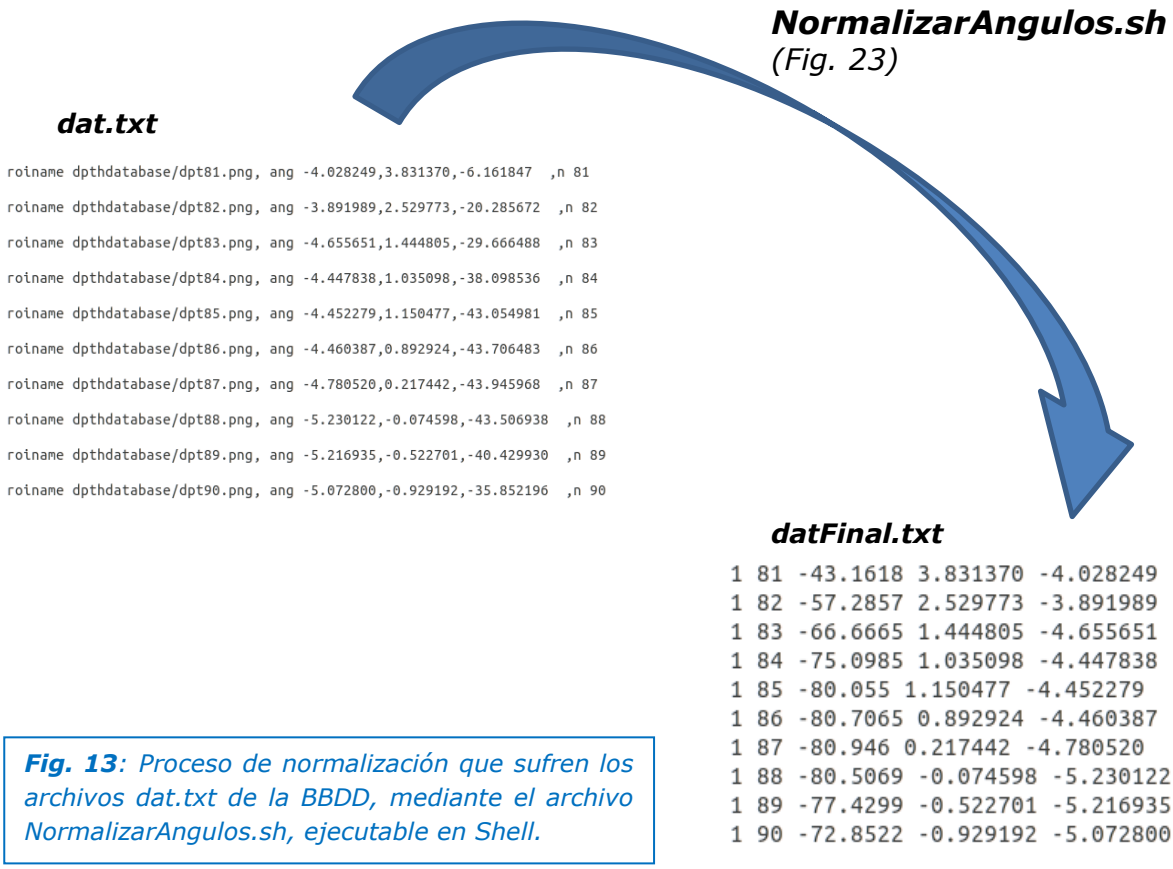

***Fig. 13**: Proceso de normalización que sufren los archivos dat.txt de la BBDD, mediante el archivo NormalizarAngulos.sh, ejecutable en Shell.*

Para la adecuación tanto del bloque de imágenes, como de los datos del archivo *dat.txt*, se ha desarrollado una archivo en Shell, que se ejecuta des del terminal de Linux, y que completa todas las acciones para normalizar los datos en la forma deseada.

4.2. MODELO DE LA CNN

El diseño de modelo de red neuronal convolucional escogido para este estudio no ha partido de cero, sino que ha habido un trabajo previo [9], ya que una parte de la estructura, y el estilo del modelo ha sido obtenido de una red neuronal convolucional ya existente [10], que usaba regresión, y la cual mediante unos parámetros de entrada, como son unas imágenes de

unas casas, juntamente con un conjunto de datos que etiquetaban a cada una de ellas, predecía los valores de adquisición de las viviendas, cuyas imágenes eran el parámetro de entrada de la CNN.

MODEL SUMMARY

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 240, 240, 16)	160
activation_1 (Activation)	(None, 240, 240, 16)	0
batch_normalization_1 (Batch	(None, 240, 240, 16)	64
max_pooling2d_1 (MaxPooling2	(None, 120, 120, 16)	0
conv2d_2 (Conv2D)	(None, 120, 120, 32)	4640
activation_2 (Activation)	(None, 120, 120, 32)	0
batch_normalization_2 (Batch	(None, 120, 120, 32)	128
max_pooling2d_2 (MaxPooling2	(None, 60, 60, 32)	0
conv2d_3 (Conv2D)	(None, 60, 60, 64)	18496
activation_3 (Activation)	(None, 60, 60, 64)	0
batch_normalization_3 (Batch	(None, 60, 60, 64)	256
max_pooling2d_3 (MaxPooling2	(None, 30, 30, 64)	0
flatten_1 (Flatten)	(None, 57600)	0
dense_1 (Dense)	(None, 16)	921616
activation_4 (Activation)	(None, 16)	0
batch_normalization_4 (Batch	(None, 16)	64
dropout_1 (Dropout)	(None, 16)	0
dense_2 (Dense)	(None, 4)	68
activation_5 (Activation)	(None, 4)	0
dense_3 (Dense)	(None, 1)	5

Total params: 945,497
Trainable params: 945,241
Non-trainable params: 256

***Fig. 14**: Sumario del modelo. Se puede apreciar el esquema del modelo, donde aparecen todas las capas con las entradas en cada una de ellas, y el número de parámetros totales con los que se entrena el modelo.*

Para la creación del modelo, hemos desarrollado una función llamada *modelo*, que se encuentra en el archivo *models.py* [20]. Debido al volumen de datos de los cuales se disponían de la base de datos, hemos adaptado el modelo para que funcione de manera óptima acorde con el volumen de datos existentes. Los parámetros de entrada de dicho modelo son las 3 dimensiones de las imágenes, además de un campo booleano, llamado *Regress*, que añadirá al modelo una capa de regresión con una activación

linear, si ese campo se encuentra en estado *True*, aunque por defecto se encuentre en *False*.

El tipo de modelo es secuencial, lo que significa que se trata de un bloque de capas lineal. La estructura del modelo es muy simple, ya que, está compuesto de un conjunto de 3 capas convolucionales, que trabajan en ventanas de 3x3, a las cuales se les aplica una activación *RELU*, y posteriormente se normalizan estas activaciones en cada *batch*, operación llamada *BatchNormalization* [14]. Finalmente se aplica al modelo la capa de *pooling*, o reducción, en la cual trabaja el algoritmo *MaxPooling*. Una vez los datos han pasado por la capa de *pooling*, van dirigidos hacia una nueva capa llamada *Flatten*, esta operación lo que hace es aplanar tensor, es decir, modifica el *shape*, de los valores de entrada, que recordamos que están en 4D, y los convierte en una matriz de elementos 1D. Una vez los datos pasan por la capa de Flatten, pasan por 16 capas Fully-Connected, conocidas como FC, activadas con *RELU*, a las cuales se les aplica un *BatchNormalization* y un *dropout* [15] en la salida. El *dropout* de 0.5 en la salida, a efectos prácticos significa que, de manera aleatoria, se ignoran la mitad de las neuronas del modelo, con el fin de evitar el sobre entrenamiento del modelo, también llamado *overfitting*. Para finalizar, la salida de esta capa de dropout, vuelve a pasar por 4 capas FC, activadas también con la función *RELU*.

4.3. OBTENCIÓN DE LOS DATOS DEL DATASET

La obtención de los datos del D*ataset* es una de las funciones que tienen más importancia en el diseño de la CNN, ya que si la obtención de los datos se realiza de una manera estructurada y eficiente, será mucho más fácil poder trabajar con ellos en el momento del entrenamiento.

En este proyecto el volumen de datos, también llamado *dataset*, se divide en 2 fuentes de información, tal y como se ha explicado en el apartado de la base de datos, las imágenes de los fotogramas, y los ficheros *dat.txt* que contienen toda la información de las imágenes.

Por ello, para la obtención de estos datos, hemos recurrido a la creación de dos funciones [10], una para obtener las imágenes, y otra para obtener los valores de las etiquetas. Ambas se encuentran dentro del archivo *datasetsHeadAngle.py* [20].

Primero se crea una función, llamada *load_images_attributes*, a la cual solo se le pasa un parámetro de entrada, y es el *path* donde se encuentra el archivo *dat.txt*. Esta función lee los datos presentes en el fichero de etiquetas, mediante la librería *pandas*, y los clasifica en columnas, las cuales tienen como nombre los que aparecen en la lista *cols*. Dentro de la función existe una condición, que no permite que el sistema capture datos

del *Dataset*, si el número total de datos que tiene un sujeto es menor que 25, ya que eso podría aportar inestabilidad al sistema. La función retorna todos estos datos en formato *dataframe*.

La segunda función, llamada *load_images*, tiene como parámetro de entrada el path donde se encuentran las imágenes, y también tiene como entrada la salida de la función anterior, es decir, el programa recibirá una lista de los datos que debe cargar, de esta manera evitaremos problemas frecuentes como puede ser el hecho de tener más imágenes que etiquetas, o a la inversa. La función una vez tiene la lista de imágenes que debe procesar, va leyendo las imágenes y las va añadiendo a un *array* de imágenes, que en última instancia es redimensionado, ya que los tensores de la CNN requieren una dimensión 4.

4.4. ALGORITMO DE PÉRDIDAS DE LA CNN

La función de perdida de la CNN, conocido como *loss function* [11], es uno de los parámetros más importantes de una red neuronal. Esta función es la encargada de calcular las pérdidas que tiene el modelo en cada iteración, ateniéndose a las características que buscamos, para luego tener la capacidad de ir optimizando ese modelo, y en consecuencia, reducir las pérdidas.

En nuestro proyecto, las pérdidas que podía tener el modelo, podían venir por el hecho de no predecir correctamente los ángulos *yaw* de los fotogramas. Por ello la *loss function* usada, no es ninguna de las predeterminadas por *Keras*, sino que es una desarrollada por nosotros.

La función de perdida, llamada *lossFunction*, basada en [12], está dentro del archivo de pérdidas del sistema, llamado *lossHeadAngle.py* [20], y como cualquier función de pérdidas, los parámetros de entrada de estas funciones son, un tensor donde aparecen las etiquetas reales de los datos introducidos en la CNN, y un tensor de la misma longitud que el primero, pero que contiene las predicciones del modelo.

Lo que nuestra loss function realiza es un cálculo entre los ángulos reales, que aparece en la entrada etiquetada, y los ángulos predichos. Lo que buscamos es que la diferencia entre ambos sea correcta, pero esto con una resta normal y corriente no nos vale, ya que nuestro rango de ángulos va en el intervalo [-180.0, 180.0], lo que provoca, que por ejemplo, un ángulo, cuyo valor real sea -179.0, y el sistema lo predice como 179.0, deba tener una diferencia de 2 grados, y no de 358, que es lo que nos marcaría el sistema si se hiciera un diferencia simple.

Para conseguir esta idea, hemos creado una función que para calcular la diferencia, calcula la inversa de la tangente de la diferencia entre el ángulo real y el ángulo predicho, tal y como se ve en la Fig. 15. Esta fórmula

matemática funciona pero para ángulos en radianes, en el rango [-π, π], por tanto es necesaria una conversión de los ángulos a radianes. Una vez se tiene la diferencia calculada, el sistema hace una media de todas las diferencias y ese es el parámetro loss que tiene el sistema.

```
def lossFunction (realAngle, predictionAngle):

        realAngle1 = (realAngle * np.pi) / 180
        predictionAngle1 = (predictionAngle * np.pi) / 180

        difference = tf.atan2(tf.sin(realAngle1 -
        predictionAngle1), tf.cos(realAngle1 -
        predictionAngle1))

        lossRadians = (difference * 180) / np.pi
        loss = tf.reduce_mean(tf.abs(lossRadians))


        return loss
```

***Fig. 15**: Función loss utilizada en el sistema. Se aprecia la conversión de ángulos a radianes, para poder calcular la diferencia, y posteriormente se convierten los valores pero a la inversa, es decir, de radianes a grados, para conocer la diferencia en grados.*

El objetivo del modelo es reducir este parámetro en cada iteración, lo que a efectos prácticos significará que los ángulos predichos cada vez se acercan más al valor de las muestras etiquetadas.

4.5. MÉTODO GENERAL

El método general, des del cual se ejecutará todo el proceso, se llama *headAngleCNN.py* [20].

En él, se realiza primero la carga de los datos de entrada mediante las funciones de obtención de datos, y antes de pasarlos al modelo, se les aplica una normalización; los ángulos de las imágenes son colocados en un mismo vector, uno detrás del otro, y a las imágenes se les aplica una normalización muy típica en problemas de *Deep Learning* para mejorar la optimización, que consiste en restarle a cada píxel de la base de datos, el valor medio de todos los pixeles existentes en la BBDD.

Una vez cargados los datos, se pasa a la partición de estos en tres conjuntos necesarios, el de entrenamiento, el de validación y el de test, tal y como se explica anteriormente.

El siguiente paso ya es compilar el modelo para poder entrenarlo posteriormente. En la orden que permite compilar el modelo, se le pasan como parámetros la función de pérdidas que hemos creado, y el optimizador del sistema. Al modelo le especificamos que se usa la regresión, y para entrenarlo le pasamos el volumen de datos de entrenamiento y validación.

Finalmente, se da la orden de evaluar el modelo, pasando como entradas el volumen de datos de test, y la orden de predicción al sistema, lo que nos dará el valor calculado por el sistema, que en nuestro caso son los ángulos.

5. EXPERIMENTOS

Para el cumplimiento de los objetivos planteados en la introducción de la memoria del proyecto, han sido necesarios una serie de experimentos o ensayos, los cuales nos han servido para detectar fallos y errores en el modelo, y que han contribuido en la mejora de la exactitud y eficiencia de la CNN.

5.1. ENTRENAMIENTO 1: OVERFITTING

Lo primero que realizamos para comprobar el funcionamiento del modelo, es un sobre-entrenamiento de la red neuronal convolucional, u *overfitting*. Este procedimiento nos permite comprobar si el funcionamiento del modelo y del algoritmo de pérdidas es correcto o necesita mejorar.

Para ello lo que hemos hecho ha sido hacer una pequeña base de datos, extraída de la fuente de datos que teníamos, con un tamaño igual al del *batch* elegido para el modelo, que en nuestro caso ha sido de 8 imágenes. Estas 8 imágenes, con sus correspondientes 8 etiquetas, son el conjunto de entrenamiento que tendrá la CNN, y a la vez serán el conjunto de validación y test. Lo que se busca con este método es que la CNN, durante el número de *epochs* especificado, sufra un sobre-entrenamiento, y la predicción que haga sobre el conjunto de test sea lo más perfecto posible. Esto se producirá si el diseño de la CNN y de las funciones de pérdidas son correctos, ya que al sobre-entrenar una CNN, y con un *batch* tan pequeño, lo que el sistema hace es aprenderse de memoria las características del conjunto de entrada, por ello, la CNN solo tiene un rendimiento óptimo para ese conjunto de datos, ya que se ha aprendido sus características.

Para conseguir los resultados deseados en este primer entrenamiento, explicados en el apartado 6.1, es necesario hacer diferentes pruebas, para conocer exactamente el número de *epochs* que el sistema necesita para disminuir las pérdidas lo máximo posible. Este número de epochs necesarias, se puede obtener mediante los gráficos *epochs-loss*, que nos muestran a partir de que iteración, el sistema estabiliza sus pérdidas. Una vez es conocido el parámetro epochs óptimo

5.2. ENTRENAMIENTO 2: DATASET

Una vez comprobado el buen funcionamiento de la CNN, mediante el proceso descrito en el apartado anterior, el siguiente paso es realizar un entrenamiento corriente y completo, con el volumen de datos conocido.

Para ello lo que hemos hecho ha sido agrupar todas las imágenes de las que disponemos en la BBDD en un mismo directorio, con el archivo *dat.txt* completo, en el que aparecen todas las etiquetas de todas las imágenes, y

hemos entrenado al sistema con este conjunto de datos como entrada. A diferencia del entrenamiento anterior, como ahora será un entrenamiento normal, no buscaremos *overfitting*, por ello de todas las imágenes de entrada, un pequeño porcentaje serán consideradas imágenes de validación, otro pequeño porcentaje serán imágenes de test, y un gran porcentaje serán imágenes de entrenamiento. Ahora el sistema tiene muchísimas más imágenes que en el caso anterior, para poder detectar patrones, y características del volumen de datos. La única similitud con el entrenamiento 1, descrito en el apartado 5.1, es que en ambos buscamos que las predicciones sean lo más parecidas posibles a los valores de las etiquetas del conjunto de test, pese a saber de qué en este entrenamiento, los resultados serán más imperfectos que en el caso anterior.

La duda en este apartado ha sido el número de filtros usados para la detección de características, ya que en el entrenamiento anterior, con tres bloques de filtros, se obtenían mejores resultados que con cuatro. Es por ello que en este apartado hemos realizado entrenamientos con las dos variantes, para comprobar cual tenía mayor exactitud, y se ha llegado a la conclusión de que con 3 bloques de capas convolucionales, el sistema es más eficiente.

6. RESULTADOS

En este apartado se detallarán los resultados obtenidos en los experimentos descritos en el punto 5 de este documento.

6.1. EXPERIMENTO 1

Tal y como se ha descrito en el apartado 5.1, la idea de este entrenamiento es generar un sobre-entrenamiento en la CNN. Para ello es necesario saber el número de *epochs* necesarias para que el sistema baje el valor de *loss* lo máximo posible.

En este apartado, primero realizamos un entrenamiento con 1000 *epochs*, y basándonos en el gráfico *epoch-loss*, se apreció que el entrenamiento se había quedado corto. Por ello aumentamos el valor de épocas, o *epochs*, hasta 25000. En el gráfico, correspondiente a la Fig. 16, se puede comprobar como hasta la iteración número 1500, el entrenamiento sufre un descenso considerable de la *loss*, y desde ese punto hasta la última iteración, sufre un descenso pero mucho más graduado con respecto al sufrido en la primera etapa mencionada. Si nos fijamos en la Fig. 16, la gráfica correspondiente al entrenamiento, el valor de *train*, es muy inconsistente durante el entrenamiento, ya que oscila en grandes niveles. Esto es debido a un reconocimiento incompleto de las características de las imágenes, lo que provoca que los valores de *loss* durante el entrenamiento, sean mucho más oscilantes que los de *validation.*

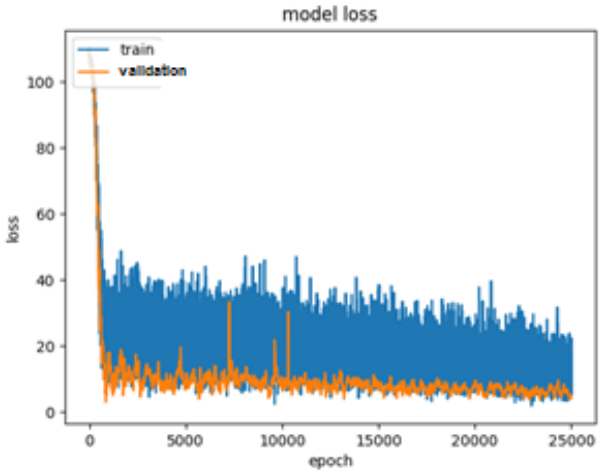

***Fig. 16**: Gráfico epoch-loss del entrenamiento 1.*

Al final del entrenamiento, el error total del entrenamiento fue de **19.9041 º** (grados), mientras que el de validación fue de **6.5980 º**. El sistema obtuvo un valor de *loss* medio de aproximadamente 12, y la media del valor de *loss* en el conjunto de validación ha sido aproximadamente de 12 también. Esto tiene una explicación y es que al estar sobre-entrenando el sistema con las mismas 8 imágenes de entrada, el sistema las trata igual tanto en entrenamiento como en validación y test.

```
Epoch 25000/25000

8/8 [==============================] - 0s 7ms/step - loss: 19.9041 - val_loss: 6.5980
```

***Fig. 17**: Valores de loss y val_loss en la última época del entrenamiento 1.*

Los resultados de la predicción son los que aparecen en la Fig. 18. Como se puede observar, un 87,5% de las predicciones hechas son muy parecidas a los valores reales. Solo hay una imagen que el sistema predice con más de 10 grados de diferencia con respecto al ángulo original.

```
[INFO] predicting head angles...

PREDICCIONES...
[[133.24187   ]
 [156.62611   ]
 [  0.21977782]
 [ 88.0636    ]
 [-39.153896  ]
 [164.2277    ]
 [157.57971   ]
 [ 90.739075  ]]


VALORES REALES...
[142.778   159.881    -1.16006  88.6364  -49.3505  177.034   172.409
  90.9469 ]
```

***Fig. 18**: Predicciones obtenidas en el experimento 1. En la parte inferior se aprecian los valores reales de las predicciones, es decir, los valores que debería predecir el sistema.*

6.2. EXPERIMENTO 2

Tal y como se ha descrito en el apartado 5.2, la idea de este entrenamiento es generar un entrenamiento completo de la CNN, con todos los datos de la BBDD. Debido al volumen de datos usados, no serán necesarias tantas iteraciones, o *epochs*, ya que en cada iteración, el sistema recorrerá más de 1500 imágenes, entre el conjunto de entrenamiento y el de validación, en las cuales detectará múltiples y variables características de las imágenes de entrada. Tal y como ocurre en la gráfica del Entrenamiento 1, aparece mucho ruido en la representación de la *loss* de entrenamiento. Esto es debido a una inconsistencia en la detección de características y patrones.

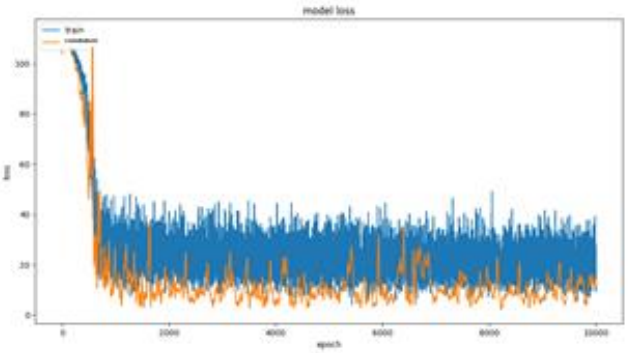

***Fig. 19**: Gráfico epoch-loss del entrenamiento 2.*

Si nos fijamos en la Fig. 21, en la cual aparecen las 30 primeras predicciones de las múltiples que devuelve el sistema, con sus respectivos valores reales, se puede observar como solo aproximadamente un 15% de las predicciones son correctas, y que un gran porcentaje de las predicciones no solo no son correctas, sino que la diferencia entre el valor de la predicción y el valor real es muy amplio. Además los valores de *loss* del entrenamiento y de la validación no son los buscados, ya que son muy altos. Al final del entrenamiento, el error total del entrenamiento fue de **20.8857 º**, mientras que el de validación fue de **29.6557 º**. Tenemos una media de *loss* de 23, y una media de *loss* en la validación de 30 a lo largo del entrenamiento.

Estos errores, en un principio, puede parecer que se deben a un mal diseño del modelo, o a una mala calidad de las imágenes de entrada, pero la realidad es otra, ya que se ha realizado una comprobación de la captura de los datos que forman la BBDD, explicado en el apartado 4.1.1, y se ha concluido con que el etiquetado de muchas de las imágenes que forman la base de datos no ha sido el correcto. Esto ha sido debido a que durante la medición y la captura de datos, ha habido un error de sincronización entre la cámara [7] que captaba las imágenes, y el sensor inercial que captaba los ángulos. Este error de sincronización se ha producido porque el sensor inercial es un teléfono que utiliza Android como SO, o sistema operativo. Android es un SO que produce desfasamiento y desincronización con sistemas que utilizan la red como sincronizador.

Esta desfasamiento en el etiquetado de las imágenes, ha sido más complicado de detectar de lo pensado, porque al tener una BBDD con imágenes de diferentes grabaciones, que se han realizado en diferentes días y a diferentes horas, la desincronización ha sido variable, por tanto, nos hemos encontrado con imágenes cuyo desfasamiento era de 4 etiquetas respecto a la imagen, y en otras el desfasamiento era hasta de 11 fotogramas, es decir, 440 milisegundos de retardo, ya que la captura es a 25 fps.

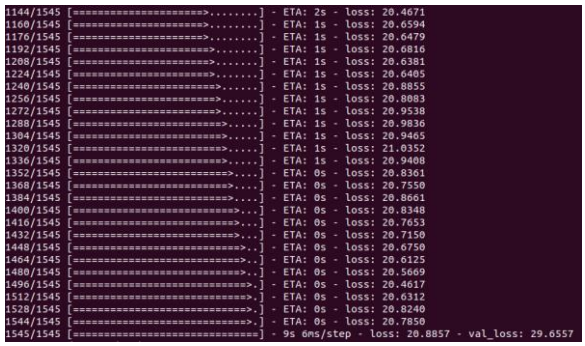

***Fig. 20**: Valores de loss y val_loss en la última época del entrenamiento 2.*

Como consecuencia de este problema en los datos de entrada, el entrenamiento no ha sido correcto en ningún momento, de ahí que los resultados obtenidos no son los esperados, y debido a una falta de tiempo en relación a la entrega del proyecto, ha sido imposible mejorar estos resultados, ya que el problema se ha detectado cuando el estudio o proyecto se encontraba en un estado muy avanzado.

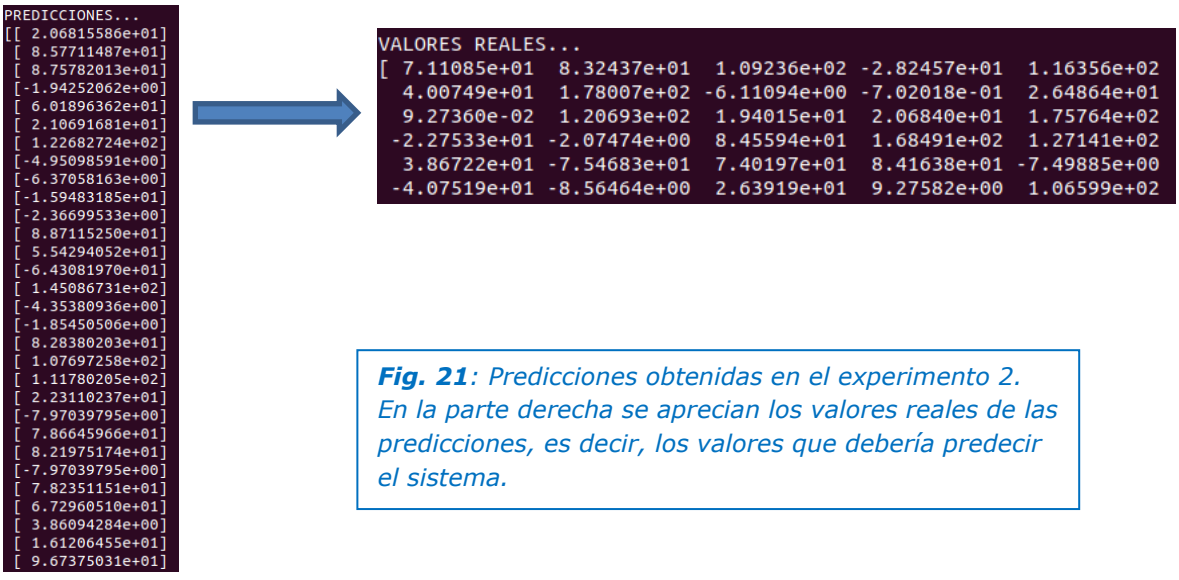

***Fig. 21**: Predicciones obtenidas en el experimento 2. En la parte derecha se aprecian los valores reales de las predicciones, es decir, los valores que debería predecir el sistema.*

7. PRESUPUESTO

El objetivo de este proyecto no es más que realizar un sistema que permita satisfacer las necesidades expuestas en la introducción del proyecto. Al no tener que construir ningún prototipo, o al no tener que realizar un estudio de mercado exhaustivo, tendremos un presupuesto muy simple, cuyos factores serán básicamente, las horas empleadas en el estudio o proyecto más los costes indirectos. En cuanto al software utilizado, en este apartado no constará en ningún momento, ya que Python, *Keras*, *Tensorflow* son *softwares* y librerías de las que puede disponer todo el mundo a coste 0, debido a que son *open-source*. Si bien es cierto que el editor utilizado para la creación del código, llamado *Pycharm* [13], es un programa que tiene una versión profesional de pago anual, pero no constará en este apartado, ya que la versión de este editor usada en este proyecto es la versión gratuita, llamada *community*.

En cuanto al cómputo de horas empleadas en la realización del proyecto, nos basaremos en estadísticas que muestra la tabla de fechas, en la Tabla 2, y el diagrama de Gantt, en la Fig. 22, para el recuento. Dentro de esas estadísticas, estableceremos una media de horas de dedicación al proyecto por semana, ya que ha habido semanas de más trabajo, y semana de menos.

La realización del proyecto en su totalidad se ha extendido 129 días, es decir unas 18 semanas. Considerando que la media de horas a la semana dedicadas a la realización del estudio ha sido de 15 h/semana, y teniendo en cuenta que un Ingeniero Júnior tiene un salario de 12€/hora, un sueldo basado en estadísticas; esto nos hace un total de **3240 €**.

En el proyecto también han intervenido como supervisores y mentores los profesores Josep Ramón Morros y Manuel López Palma, los cuales han tenido una dedicación activa en el proyecto de aproximadamente 3 h/semana. Si consideramos que un Ingeniero Sénior tiene un salario de 60€/hora, basándonos en estadísticas también; esto hace un total de **3240 €** cada uno.

Además de las horas empleadas en la realización y desarrollo del proyecto por parte del alumno, y por parte de los profesores, existen una serie de costes indirectos que también se aplican al presupuesto final. Para la realización de este proyecto es necesario un ordenador, el cual se le estima una tiempo de vida genérico de 3 años, por tanto y dividiendo un presupuesto inicial de 3000 €, entre el tiempo de uso del ordenador, que es de 5 meses, hace un total de **600 €** para este equipamiento tecnológico.

En cuanto al espacio de trabajo utilizado, ya sea un estudio o una oficina, también es necesario contemplarlo en el presupuesto del proyecto. El precio medio de un *coworking* en Barcelona se eleva a 300 € mensuales.

Teniendo en cuenta que el tiempo de trabajo ha sido de 5 meses, el precio asciende hasta los **1500 €** en cuanto al espacio de trabajo nos referimos.

GASTO	**VALOR (€)**
Ingeniero Júnior	3240.00
Ingeniero Sénior (Josep Ramón Morros)	3240.00
Ingeniero Sénior (Manuel López Palma)	3240.00
Equipamiento tecnológico	600.00
Espacio de trabajo	1500.00
TOTAL	**11820.00**

***Tabla 1**: Resumen del presupuesto del estudio.*

8. CONCLUSIONES Y TRABAJO FUTURO

En base a los resultados obtenidos, y en relación de estos con los esperados a priori, el trabajo no concluye con el éxito esperado. Pese a ello, es necesario destacar que la metodología de trabajo, y el sistema creado, han sido correctos, pero se ha padecido el contratiempo con la toma de capturas, un problema que se ha detallado en el apartado de resultados del proyecto, el 6.2.

Del estudio y su desarrollo se pueden extraer conclusiones positivas, ya que el método utilizado, y las técnicas para desarrollarlo son muy efectivas, y subsanan y mejoran ciertas necesidades que existen en el sector. Además se trata de un sistema con poco impacto económico y que puedo reportar grandes beneficios.

No obstante, también se pueden extraer conclusiones menos positivas que invitan a seguir desarrollando el sistema, ya que existen diferentes bloques del proyecto que pueden mejorar. Por ejemplo las imágenes de la base de datos, ya que las imágenes usadas para el entrenamiento del sistema, son imágenes en forma de *depth*, en escala de gris y codificadas con 8 bits, lo que provoca que el rango de la escala de grises no es lo suficientemente amplio como para detectar diferentes profundidades en las imágenes; en una futura versión del proyecto sería conveniente utilizar imágenes codificadas con 10 o más bits. Otro procedimiento a mejorar es la captura de los ángulos de prueba, ya que el método usado en este estudio, ha sido usando un sensor inercial, en concreto un teléfono Android, lo que provoca que las referencias no sean correctas y puedan aparecer, como ha sido el caso, diferencias en la sincronización con las cámaras.

Como trabajo futuro para mejorar este modelo, se plantea diferentes ideas:

1. Añadir al sistema la capacidad de predecir el ángulo *Pitch* de las cabezas analizadas, de esta manera, el foco de atención será calculado a la perfección, ya que sabiendo el ángulo de dirección (*Yaw*), y el ángulo de inclinación (*Pitch*), se calcula de manera exacta la orientación y dirección de la vista de la persona.

   Esta mejora futura, era uno de los objetivos que tenía a priori el estudio, pero debido al error de sincronismo, que se detectó muy tarde, fue imposible adaptar la arquitectura del modelo, diseñada previamente para trabajar solo con 1 ángulo, para que trabajara con 2 ángulos (*Yaw* + *Pitch*) como parámetros de entrada de la CNN.

   Para realizar este concepto, lo que se debe hacer es añadir al diseño del modelo, una capa FC al final del modelo, con la cual el sistema tendrá dos capas finales FC, una para cada ángulo.

2. Regularizar la base de datos creada, para poder arreglar los problemas que han surgido de sincronismo, y poder realizar un entrenamiento más efectivo y comprobar la precisión del modelo.
3. Utilizar una red neuronal pre-entrenada, como puede ser la red ResNet, para comprobar si ajustando los pesos de dicha DNN, los resultados obtenidos son mejores o peores que los registrados con el modelo propio creado.
4. Aumentar el tamaño de los *Batch* de la CNN a la hora de entrenarla con filtros de pequeño tamaño, ya que esto podría derivar en una mejor precisión.
5. Finalmente, otra mejora seria aumentar el número de datos de la BBDD para mejorar, en consecuencia, las prestaciones de la CNN, ya que a más datos, más aprendizaje y mejores resultados.

9. REFERENCIAS

[1] M. López-Palma, Morros, J. R., Gago, J., and Corbalán, M., "Oriented trajectories as a method for audience measurement", in *27th International Symposium on Industrial Electronics (ISIE)*, Cairns, Australia, 2018. [Online]

https://imatge.upc.edu/web/sites/default/files/pub/cLopez-Palma.pdf

[2] «Keras» [Online]

https://keras.io/

[3] «Tensorflow» [Online]

https://www.tensorflow.org/

[4] Jeff Bertolucci, "Big Data: A practical definition", in InformationWeek, 2013. [Online]

https://www.informationweek.com/big-data/big-data-analytics/big-data-a-practical-definition/d/d-id/1111290

[5] «Wikipedia – Redes Neuronales Convolucionales» [Online]

https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales

[6] Niklas Donges, "Gradient descent in a Nutshell", in Towards Data Science, 2018. [Online]

https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0

[7] Camera Properties «Intel – Intel RealSense Depth Camera» [Online]

https://click.intel.com/intelr-realsensetm-depth-camera-d415.html

[8] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics, Kobe, Japan, May 2009.

[9] «Udacity UD730 course» [Online]

https://classroom.udacity.com/courses/ud730

[10] Adrian Rosebrock, "Keras, Regressions and CNNs", in Pyimagesearch, January 2019. [Online]

https://www.pyimagesearch.com/2019/01/28/keras-regression-and-cnns/

[11] «Wikipedia – Loss function» [Online]

https://en.wikipedia.org/wiki/Loss_function

[12] 'Patwie', Stackoverflow – Question 46355068, 2017. [Online]

https://stackoverflow.com/questions/46355068/keras-loss-function-for-360-degree-prediction

[13] «JetBrains – Pycharm» [Online]

https://www.jetbrains.com/pycharm/

[14] Firdaouss Doukkali, "Batch normalization in Neural Networks", in Towards Data Science, 2017. [Online]

https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

[15] Vicente Rodríguez, "Dropout y Batch Normalization", in Vincentblog, November 2018. [Online]

https://vincentblog.xyz/posts/dropout-y-batch-normalization

[16] K. Buys, C. Cagniart, A. Baksheev, T. De Laet, J. De Schutter, and C. Pantofaru, "An adaptable system for RGB-D based human body detection and pose estimation," Journal of Visual Communication and Image Representation, 2014.

[17] «Wikipedia – Viola-Jones object detection framework» [Online]

https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework

[18] Satya Mallick, "Histogram of Oriented Gradients", in Learn OpenCV, December 2016. [Online]

https://www.learnopencv.com/histogram-of-oriented-gradients/

[19] «Wikipedia –Stochastic Gradient Descent» [Online]

https://en.wikipedia.org/wiki/Stochastic_gradient_descent

[20] «GitHub –Código del proyecto» [Online]

https://github.com/AVillegas8/TFG

10. ANEXO

ACTIVIDAD	FECHA INICIAL	DURADA EN DÍAS	FECHA FINAL
Creación de la BBDD	01/02/2019	52	25/03/2019
Limpieza de la BBDD	26/03/2019	4	30/03/2019
Normalización de la BBDD	31/03/2019	10	10/04/2019
Diseño de la CNN	01/02/2019	89	01/05/2019
Entrenamiento de la CNN	01/05/2019	35	05/06/2019
Mejora de la CNN	01/05/2019	35	05/06/2019
Evaluación de la CNN	05/06/2019	5	10/06/2019
Redacción de la memoria	01/02/2019	129	10/06/2019

***Tabla 2**: Tabla de fechas dedicadas a la elaboración del estudio. En base a esta tabla se realza el diagrama de Gantt.*

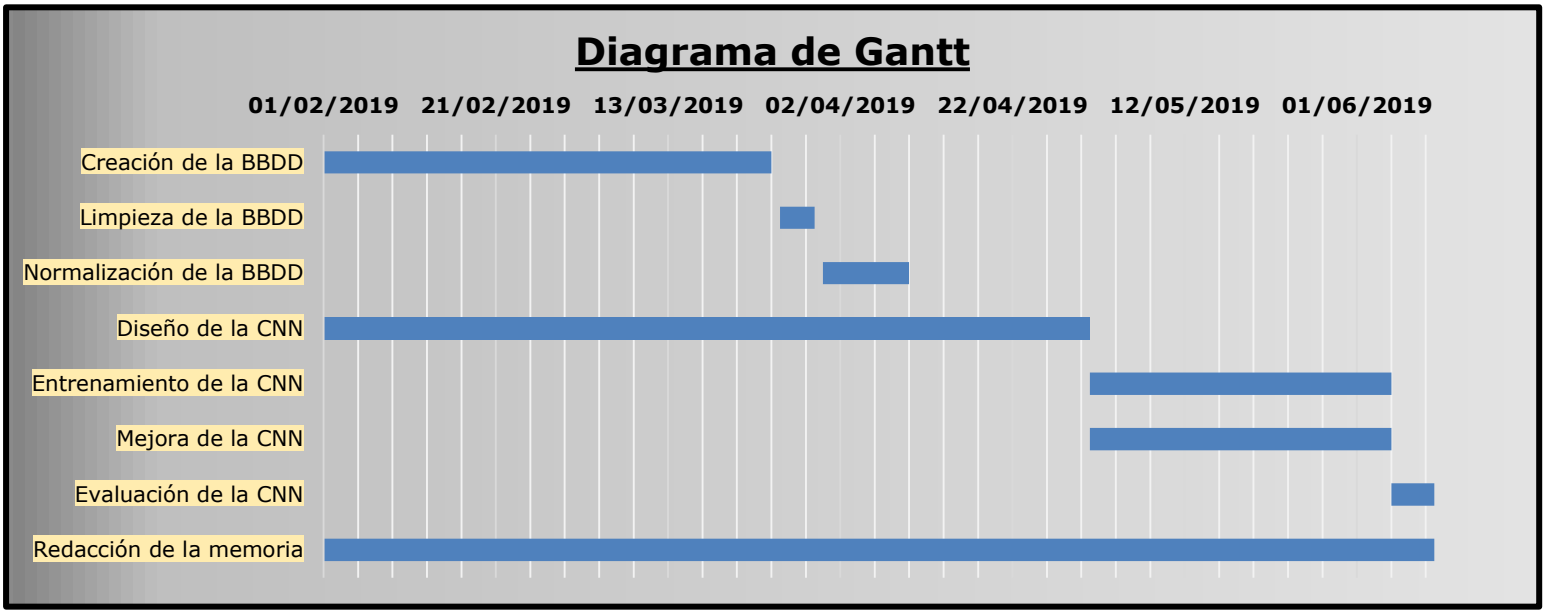

***Fig. 22**: Diagrama de Gantt seguido durante el proyecto.*

```
#! /bin/bash

# Eliminamos las líneas intermedias vacias que aparecen entre tomas.
awk 'NR%2 != 0' dat.txt > dat2.txt

# Modificamos la estructura del archivo, y la colocamos en formato 'dato_dato_dato_dato_dato'.
sed -e 's/.*, ang /1 /g' -e 's/  ,n / /g' -e 's/,/ /g'  dat2.txt > dat3.txt

# De las 5 columnas solo nos quedamos con las columnas 1, 3, 4 y 5 que son las que nos interesan.
# A la columna 4 (angulos 'yaw') le aplicamos la normalización, debido al offset de 37 grados.
awk '{
if ($4 >= -143 && $4 <= 180)
        print $1, $5, $4-37, $3, $2
else
        print $1, $5, $4-37+360, $3, $2

}' dat3.txt > datFinal.txt

# Borramos los archivos .txt creados en el proceso y que no nos interesan
rm dat2.txt dat3.txt
```

***Fig. 23**: Archivo ejecutable NormalizarAngulos.sh. Convierte el archivo dat.txt en el archivo datFinal.txt, que tiene otro formato en la estructura de los datos. Los datos captados por el sensor inercial pasan a tener una estructura: Sujeto_Fotograma_Yaw_Pitch_Roll.*

```
#! /bin/bash

# Nos situamos en el directorio que contiene las imágenes que necesitamos
cd dpthdatabase

# Renombramos todas las imagenes en el formato NumPersona_Fotograma.png
rename 's/dpt/1_/' *.png
```

***Fig. 24**: Archivo ejecutable NormalizarImagenes.sh. Convierte el estilo de los nombres de las imágenes que forman parte de la BBDD. La estructura pasa a ser: Persona_Fotograma.*