

Universidades de Burgos, León y
Valladolid

Máster universitario

Inteligencia de Negocio y Big Data en Entornos Seguros



Trabajo Fin de Máster

**Algo muy largo y no sé que más
puedo hacer aquí**

Presentado por José Miguel Ramírez Sanz
en Universidad de Burgos — 10 de junio
de 2020

Tutor: Dr. José Francisco Díez Pastor
Dr. Álvar Arnaiz González

Universidades de Burgos, León y Valladolid



Máster universitario en Inteligencia de Negocio y Big Data en Entornos Seguros

D. José Francisco Díez Pastor, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. José Miguel Ramírez Sanz, con DNI 71303106R, ha realizado el Trabajo final de Máster en Inteligencia de Negocio y Big Data en Entornos Seguros titulado TFM.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 10 de junio de 2020

Vº. Bº. del Tutor:

Dr. José Francisco Díez Pastor

Vº. Bº. del co-tutor:

Dr. Álvar Arnaiz González

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice general	III
Índice de figuras	VI
Índice de tablas	VIII
Memoria	1
1. Introducción	3
2. Objetivos del proyecto	5
2.1. Objetivos funcionales	5
2.2. Objetivos técnicos	5
2.3. Objetivos personales	6
3. Conceptos teóricos	7
3.1. Vídeo	7
3.2. Visión por computador	8
3.3. Redes neuronales	9
3.4. Puntos y vectores	18
4. Técnicas y herramientas	19
4.1. Visión por computador ~ Detección de movimiento	19
4.2. Estación de trabajo	20
5. Aspectos relevantes del desarrollo del proyecto	21
5.1. Desarrollo FIS-HUBU	21

5.2. Investigación algoritmos de visión por computador	25
5.3. Investigación de <i>Detectron2</i>	26
5.4. Cálculo de características	32
5.5. Primeras versiones comparación de posiciones	36
5.6. Versión 3 de comparación, uso de medias	41
5.7. Error en brazos, nueva posición y comparación	42
5.8. Comprobación de la última versión	49
6. Trabajos relacionados	53
7. Conclusiones y Líneas de trabajo futuras	55
 Apéndices	 56
Apéndice A Plan de Proyecto Software	59
A.1. Introducción	59
A.2. Planificación temporal	59
A.3. Estudio de viabilidad	62
Apéndice B Especificación de Requisitos	63
B.1. Introducción	63
B.2. Objetivos generales	63
B.3. Catalogo de requisitos	63
B.4. Especificación de requisitos	63
Apéndice C Especificación de diseño	65
C.1. Introducción	65
C.2. Diseño de datos	65
C.3. Diseño procedimental	65
C.4. Diseño arquitectónico	65
Apéndice D Documentación técnica de programación	67
D.1. Introducción	67
D.2. Estructura de directorios	67
D.3. Manual del programador	67
D.4. Compilación, instalación y ejecución del proyecto	67
D.5. Pruebas del sistema	67
Apéndice E Documentación de usuario	69
E.1. Introducción	69
E.2. Requisitos de usuarios	69

ÍNDICE GENERAL

v

E.3. Instalación	69
E.4. Manual del usuario	69
Bibliografía	71

Índice de figuras

3.1.	Organización de una red neuronal convolucional [8].	10
3.2.	Ejemplo aplicación de un filtro con <i>stride</i> 1 [8].	12
3.3.	<i>Dilated Convolution</i> con dilatación a 2 [8].	14
3.4.	Ejemplo de capa de agrupación de tamaño 2 [8].	15
3.5.	Funciones de activación más comunes en capas de no linealidad [8].	16
3.6.	Ejemplo bloque residual en ResNet [8].	16
3.7.	Region-Based CNN [8].	17
3.8.	Feature Pyramid Network [10].	18
5.9.	Menú principal de un responsable.	23
5.10.	Menú de estadísticas.	23
5.11.	Ejemplo de la evolución de un paciente vista por un responsable.	24
5.12.	Menú principal de los pacientes.	24
5.13.	Mejora del rendimiento en GB/s con NVlink [7].	28
5.14.	Modelo de tipo COCO Detection with Faster R-CNN, <code>faster_rcnn_R_50_C4_1x</code>	29
5.15.	Modelo de tipo COCO Detection with RetinaNet, <code>retinanet_R_101_FPN_3x</code>	30
5.16.	Modelo de tipo COCO Instance Segmentation Baselines with Mask R-CNN, <code>mask_rcnn_R_50_DC5_3x</code>	31
5.17.	Modelo de tipo COCO Person Keypoint Detection Baselines with Keypoint R-CNN, <code>keypoint_rcnn_R_101_FPN_3x</code>	32
5.18.	Modelo de tipo COCO Panoptic Segmentation Baselines with Panoptic FPN, <code>panoptic_fpn_R_101_3x</code>	33
5.19.	Modelo de tipo LVIS Instance Segmentation Baselines with Mask R-CNN, <code>mask_rcnn_X_101_32x8d_FPN_1x</code>	34
5.20.	Prueba con chaqueta.	35
5.21.	Prueba con un objeto de por medio.	36
5.22.	Prueba con <i>threshold</i> a 0.3.	37
5.23.	Prueba con <i>threshold</i> a 0.99.	38

5.24. Puntos clave predichos con el modelo junto con la etiqueta con su orden.	39
5.25. Ejemplo ángulos sobre los que se calcula la diferencia entre posiciones.	40
5.26. Posición con los codos en 90 grados mirando hacia abajo.	41
5.27. Posición con los codos en 90 grados mirando hacia arriba.	42
5.28. Posición con los hombros en 45 grados mirando hacia abajo.	43
5.29. Posición con los hombros en 45 grados mirando hacia arriba.	44
5.30. Ejemplo de error en la comparación de brazos.	45

Índice de tablas

5.1.	Tabla con el estudio de los modelos de posición ordenado por ratio.	30
5.2.	Tabla con las estadísticas de las diferencias en grados.	50
5.3.	Tabla con las estadísticas de las diferencias en porcentajes de exactitud.	50
5.4.	Tabla con los tiempos de la ejecución.	50

Memoria

Introducción

Descripción del contenido del trabajo y del estrucutra de la memoria y del resto de materiales entregados.

Objetivos del proyecto

En este apartado se va a comentar los objetivos funcionales, técnicos y personales que se tienen en el desarrollo de este proyecto.

2.1. Objetivos funcionales

En este subapartado se va a comentar los distintos objetivos que se querían cumplir con el desarrollo del proyecto:

- Desarrollar una aplicación web para poder realizar y evaluar las rehabilitaciones *online* de los pacientes con Parkinson.
- Desarrollar una aplicación web accesible y fácil de utilizar, sobre todo para los pacientes.
- Crear un herramienta capaz de comparar las rehabilitaciones hechas por los pacientes con los ejercicios bases.

2.2. Objetivos técnicos

En este subapartado se va a detallar los distintos objetivos relacionados con las técnicas y herramientas que se quieren aprender y utilizar:

- Crear una aplicación web para realizar rehabilitaciones *online*.
- Crear una herramienta de comparación de ejercicios a partir de algoritmos de visión por computador y detección de movimientos, que permita comparar los fotogramas de un mismo vídeo.

- Utilizar el lenguaje de programación *Python* para la comparación de ejercicios.
- Realizar el desarrollo del proyecto utilizando un repositorio *Git*, en concreto en *GitHub* para poder controlar las tareas y las versiones del proyecto.
- Utilizar la extensión de *Git* llamada *ZenHub* para controlar el estado de las tareas y la temporalidad de estas, y así seguir el modelo *SCRUM* para desarrollar el proyecto de forma incremental.

2.3. Objetivos personales

Por último, se van a comentar los objetivos personales que se tienen en este proyecto, donde se encuentran objetivos desde probar conocimientos adquiridos en el máster como mejores algunas cualidades personales.

- Poder ayudar a las personas mayores con Parkinson para que puedan realizar las rehabilitaciones necesarias sin necesidad de salir de sus casas. Además, mejorar esta rehabilitación a partir de la comparación de los ejercicios realizados.
- Mejorar mis capacidades comunicativas y de exposición en las diversas presentaciones del proyecto a los responsables.
- Usar los conocimientos adquiridos durante la carrera y durante el máster.
- Mejorar mis conocimientos sobre visión por computador.
- Conocer los distintos algoritmos de comparación y clasificación de instancias para los datos obtenidos sobre los ejercicios.

Conceptos teóricos

En este apartado se van a explicar los distintos conceptos teóricos necesarios para comprender correctamente el proyecto.

3.1. Vídeo

Tipo de datos compuesto por un conjunto de imágenes, y en algunos casos, una señal de audio sincronizada con las imágenes. Existen diversos formatos para videos que dependen de la codificación de los datos que utiliza, pero los más comunes son mp4, avi, . A parte de la compresión de la propia codificación, el tamaño de este tipo de datos depende principalmente de dos factores la cantidad de fotogramas por segundo y la calidad de los fotogramas.

Fotogramas por segundo - fps

Medida esencial en los videos que determina la cantidad de fotogramas que hay por segundo de video. Esta medida está muy relacionada con la finalidad del video, ya que no se necesitan los mismos fotogramas por segundo para una película, que suele ser 24 fps [2], que para ver un documental de deporte. Cabe destacar, que a la hora de visualizar un video es importante la tasa de refresco del monitor, para que se pueda apreciar correctamente la cantidad de fotogramas por segundo del video se necesita una tasa de refresco igual o superior. Esto quiere decir que si un video está a 60 fps lo recomendable para ver correctamente el video es que el monitor tenga al menos 60Hz de tasa de refresco.

Calidad

La calidad de las imágenes del vídeo es otro de los parámetros más importantes, ya que dependiendo de este valor el vídeo pesará más o menos o se verá mejor o peor. Actualmente la proporción más normal de pantalla es 16:9, las calidades más comunes para esta proporción son:

- 480p ->854 x 480.
- 720p HD ->1280 x 720.
- 1080 p Full HD ->1920 x 1080 .
- 1440p QHD ->2560 x 1440.

3.2. Visión por computador

La visión por computador, también llamada visión artificial, es la parte de la ciencia de la computación orientada a la recogida y al tratamiento de imágenes y vídeos [11, 8].

Por la tipología de los datos con los que se trabaja en la visión por computador el procesamiento de éstos es muy costoso. Este elevado coste computacional es causado en gran parte por la calidad de las imágenes y en el caso de los vídeos, además de la calidad de la imagen, por la cantidad de fotogramas por segundo con el que se ha grabado éste.

Estimación de la posición

Una de las principales funciones que tiene la visión artificial es la detección de movimiento, la cual consiste en primero detectar la posición de cada una de las partes del cuerpo (depende del propio algoritmo la división que se quiera realizar sobre el cuerpo) para posteriormente realizar un seguimiento de estos elementos.

Segmentación

Es otra de las funciones más usadas de la visión por computador, en ella se busca analizar la imagen para poder detectar y delimitar el espacio los distintos objetos, personas o animales que hay en la imagen. Este espacio delimitado se obtiene de dos formas principalmente:

- Figura exacta, se delimita la silueta del objeto, persona o animal detectado con un conjunto de puntos que representan la silueta de éste.
- Forma geométrica, es el tipo más usado debido a que requiere menor capacidad de cómputo. La forma más utilizada es el rectángulo, pero se podrían usar otro tipo de formas para delimitar y segmentar el objeto, persona o animal detectado.

3.3. Redes neuronales

Modelos basados en la analogía con la corteza cerebral de los mamíferos, aunque no están cerca, al menos de momento, de ser tan complejos como el cerebro biológico. El funcionamiento de estos modelos es relativamente sencillo, ya que «solo» son un conjunto de unidades básicas, que reciben una entrada y dan una salida. Donde realmente reside su complejidad, al igual que en el cerebro humano, es en las conexiones de estas unidades básicas para formar un modelo complejo capaz de realizar multitud de tareas [8].

Estas unidades interactúan con los datos que les llegan con una función de activación que puede excitar a esta unidad o neurona y así pasar la información modificada a sus conexiones. Además, las distintas conexiones entre las unidades tienen un peso definido, que expresa la fuerza del enlace. Este peso se modifica a lo largo del entrenamiento de la red para poder obtener mejores resultados.

Según como se propague la información por la red, estas pueden dividirse en dos tipos:

- **Feed-forward networks:** redes neuronales donde la información solo avanza por ésta hacia delante, no existe ningún tipo de bucle entre las conexiones de las unidades que haga pasar dos o más veces por una de éstas. En este tipo de redes neuronales se encuentran el perceptrón multicapa y las redes neuronales convolucionales
- **Feed-back networks:** redes neuronales que sí admiten bucles entre sus conexiones, lo que les permite tener capacidad de memorización. Algunos ejemplos de este tipo de redes son las redes neuronales recurrentes o las *Long-Short Term Memory* [13].

Redes neuronales convolucionales

Son de los tipos de redes neuronales más usados en la actualidad, sobre todo cuando se trabaja con datos de alta dimensionalidad, principalmente imágenes y vídeos. Las redes neuronales convolucionales tienen dos partes (figura 3.1):

1. Fase de extracción de características.
2. Fase de clasificación, puede realizarse con un perceptrón multicapa por ejemplo.

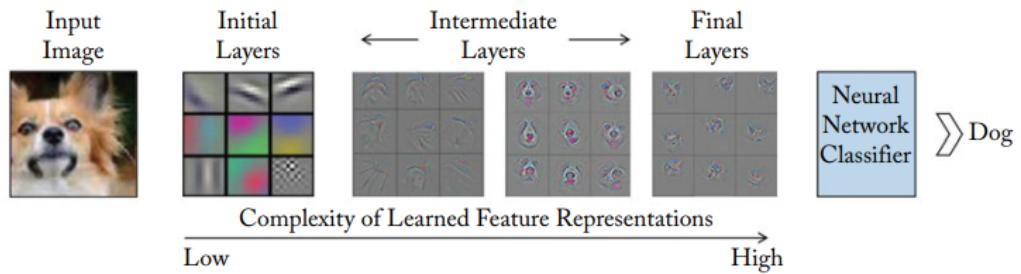


Figura 3.1: Organización de una red neuronal convolucional [8].

Cada fase está relacionada con un paradigma de aprendizaje distinto, ya que en la extracción de características a partir de filtros se realiza de forma no supervisada, mientras que en la fase de clasificación se realiza un aprendizaje supervisado a partir de los datos obtenidos en la fase de extracción de características.

Aun así, el primer paso que siempre se ha de realizar antes de entrar en las distintas capas es realizar un preprocessamiento de la imagen. Éste puede ser por:

- Extracción de la media, para a partir de ésta poder centrar los datos. Para cada dato de entrenamiento x de un conjunto N tal que $x \in \mathbb{R}^{h \times w \times c}$, donde h es el alto de la imagen, w es el ancho y c es color (normalmente 3 dimensiones, RGB), se puede definir la extracción de la media con la siguiente fórmula:

$$x' = x - \hat{x}, \hat{x} = 1/N \sum_{i=1}^N x_i \quad (3.1)$$

- Normalización.
- Blanqueo con PCA para reducir las correlaciones entre las diferentes dimensiones de los datos a partir de un normalización independiente para cada una de ellas.
- *Local Contrast Normalization*, tipo de normalización que permite resaltar los valores más elevados.

Los tipos de preprocesados que se suelen realizar son las extracción de la media y la normalización, que a la vez son los métodos más sencillos.

Tipos de capas en una red neuronal convolucional

Capas convolucionales

Como ya se ha comentado, la funcionalidad de estas capas es la extracción de características a partir de filtros, pero ¿qué es un filtro? Un filtro es una malla con valores discretos, normalmente cuadrada, que representan pesos que se modifican en la fase de entrenamiento de la red. Es con cada uno de estos filtros con los que se recorren los datos para obtener como resultado un resumen de los datos de entrada. Como se ve en la figura 3.2, el resumen se calcula aplicando el filtro a los datos multiplicando los valores de los datos por su posición en el filtro y luego sumando los valores.

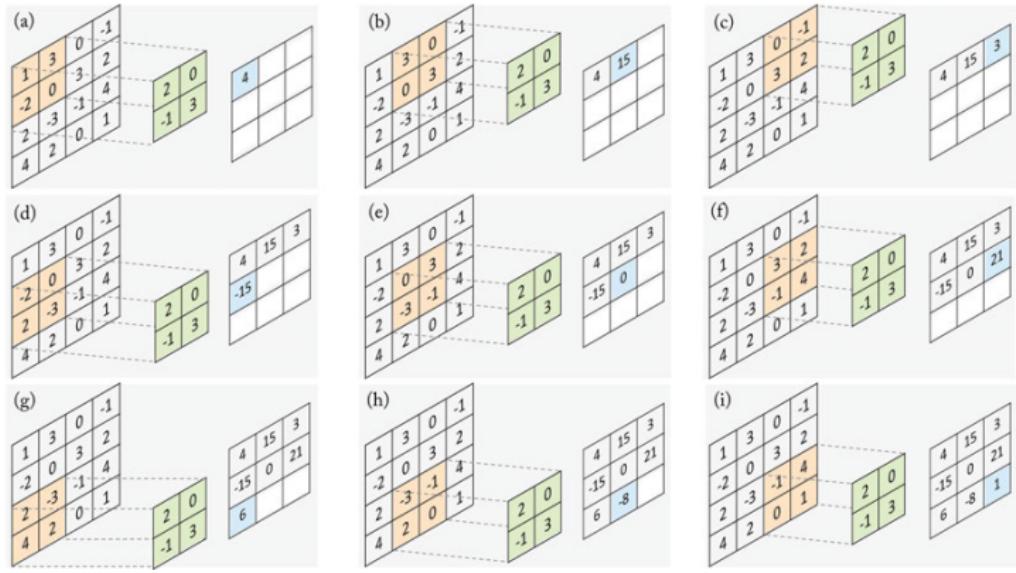
Al aplicar el filtro hay que tener en cuenta el *stride*, que es el movimiento que hace el filtro en cada paso, en la imagen se puede ver como este valor es de 1 por lo que la imagen avanza horizontal y verticalmente de uno en uno. Valores más altos de *stride* producen resúmenes más comprimidos pero puede que menos precisos, lo que se suele llamar un submuestreo de los datos.

Para calcular el tamaño de los datos una vez pasado un filtro se puede usar la siguiente fórmula, donde h es *height* (alto), w es *width* (ancho), s es el *stride* y f es el tamaño del filtro:

$$h' = \left\lfloor \frac{h - f + s}{s} \right\rfloor; w' = \left\lfloor \frac{w - f + s}{s} \right\rfloor \quad (3.2)$$

Siendo esta la función de parte entera o *floor operation*. Si se aplica esta fórmula a los datos de la imagen 3.2:

$$h' = \left\lfloor \frac{4 - 2 + 1}{1} \right\rfloor = 3; w' = \left\lfloor \frac{4 - 2 + 1}{1} \right\rfloor = 3 \quad (3.3)$$

Figura 3.2: Ejemplo aplicación de un filtro con *stride* 1 [8].

Con este tipo de capas existe un problema, ya que hay muchas situaciones que necesitan que se conserve el tamaño de los datos, como por ejemplo en una segmentación, pero si se utiliza este tipo de capas aunque se use el valor mínimo, *stride* 1, se siguen perdiendo datos suavizando sobre todo los bordes. Es por ello que surgió el concepto *zero-padding* que permite mantener el tamaño original de los datos.

El *zero-padding* consiste aadir a los datos de entrada bordes con valores 0 para así cuando se pase el filtro el tamaño de los datos se intente mantener constante. Con el uso del *zero-padding* la fórmula se modifica usando p como el número de aumentos con valores a 0 en cada dimensión [3]:

$$h' = \left\lfloor \frac{h - f + s + p}{s} \right\rfloor; w' = \left\lfloor \frac{w - f + s + p}{s} \right\rfloor \quad (3.4)$$

El uso de *padding* se puede dividir en 3 categorías:

- *Valid Convolution*: es el caso más sencillo, el filtro siempre se mantiene en posiciones válidas. Las dimensiones finales se reducen tanto en el alto (h) como en ancho (w) en el tamaño del filtro (f) menos 1.
- *Same Convolution*: en este caso los datos de entrada y de salida de la capa tienen el mismo tamaño, para conseguirlo se ha de realizar *zero-padding* correctamente, ya que no se puede dar en todos los casos.

- *Full Convolution*: en este caso se utiliza el mayor *padding* posible, que se consigue cuando solo un elemento de los datos de entrada está involucrado en todas las operaciones con el filtro.

Como se puede ver, el *stride* y sobre todo el tamaño del filtro a aplicar es muy importante en este tipo de capas, es por ello que definir un buen valor para estos puede ser esencial en el desarrollo del modelo. Por esta razón existe la función de *Receptive Field* que permite descubrir el valor para el tamaño de los filtros (todos los filtros de todas las capas del mismo tamaño) más efectivo. La fórmula del *Effective Receptive Field* para la capa N de las N capas convolucionales es:

$$RF_{eff}^n = f + n * (f - 1) \quad (3.5)$$

Pero, como se ha visto, los filtros no suelen tener el mismo tamaño ni el mismo *stride*, es por ello que la función de *Receptive Field* para el tamaño de más de un filtro es:

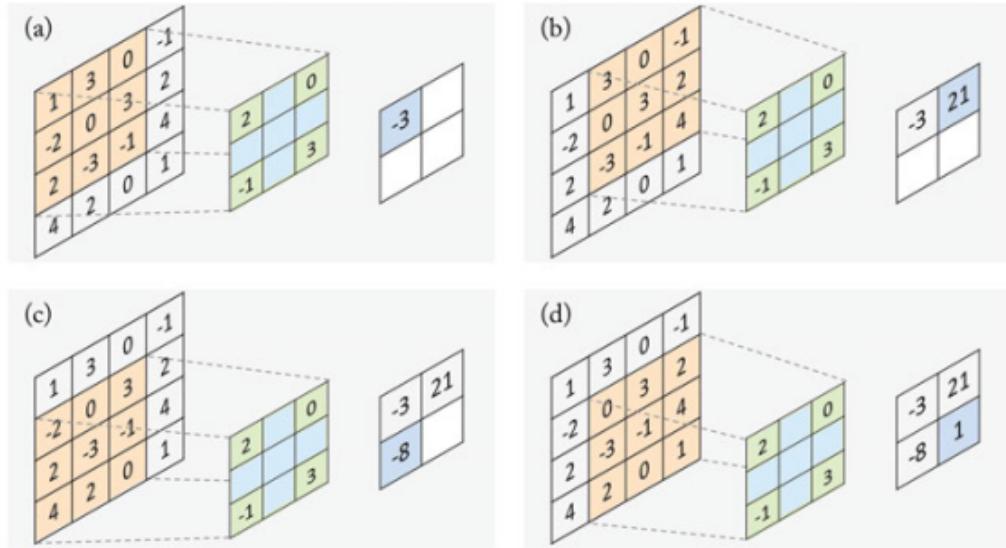
$$RF_{eff}^n = RF_{eff}^{n-1} + ((f_n - 1) * \prod_{i=1}^{n-1} s_i) \quad (3.6)$$

Donde f_n es el tamaño del filtro de la capa n , s_i es el valor de *stride* para la capa i , y RF_{eff}^{n-1} representa el *effective receptive field* de la capa anterior.

Como ya se ha comentado, uno de los principales problemas de las capas convolucionales son los valores de los bordes, ya que al aparecer menos en las operaciones de los bordes se ven menos representados en las salidas de las capas. Es por ello que se propuso una modificación de las capas convolucionales llamada *Dilated Convolution*, la cual tiene una dilatación (d) siendo esta la distancia entre los valores del filtro a tener en cuenta. Como se puede ver en la figura 3.3, donde se obtiene un filtro de tamaño 3 y con una dilatación de 2, los valores de los bordes tienen un mayor peso en la salida de la capa.

Al haber añadido un nuevo parámetro, el tamaño de la salida viene dado por las siguientes fórmulas:

$$\begin{aligned} h' &= \frac{h - f - (d - 1) * (f - 1) + s + 2p}{s} \\ w' &= \frac{w - f - (d - 1) * (f - 1) + s + 2p}{s} \end{aligned} \quad (3.7)$$

Figura 3.3: *Dilated Convolution* con dilatación a 2 [8].

Capas de agrupación

Tipo de capa en la cual se realiza una agrupación por un número fijado de datos. Esta agrupación se puede realizar de diversas formas, como puede ser la media, el mínimo o el máximo. Como en las capas convolucionales se ha de pasar el valor del *stride* para indicar como va a ser el movimiento del segmento de agrupación. Además, se le ha de pasar el tamaño del segmento de agrupación, algo parecido a lo que pasaba con el tamaño del filtro, pero en este caso no hay valores en el filtro, simplemente se agrupan siguiendo una función los datos del segmento. Un ejemplo se puede ver en la figura 3.4, donde se puede observar como los valores se van agrupando acorde con el valor máximo del segmento o bloque comprobado. Los tamaños de las salida de los datos tras pasar por una capa se pueden calcular con las siguientes fórmulas, donde f pasa a ser el tamaño del segmento o bloque:

$$h' = \left\lceil \frac{h - f + s}{s} \right\rceil; w' = \left\lceil \frac{w - f + s}{s} \right\rceil \quad (3.8)$$

Este tipo de capas son muy útiles para compactar la información, para detectar cambios en los datos como puede ser para el movimiento o la posición en una imagen.

Capas totalmente conectadas

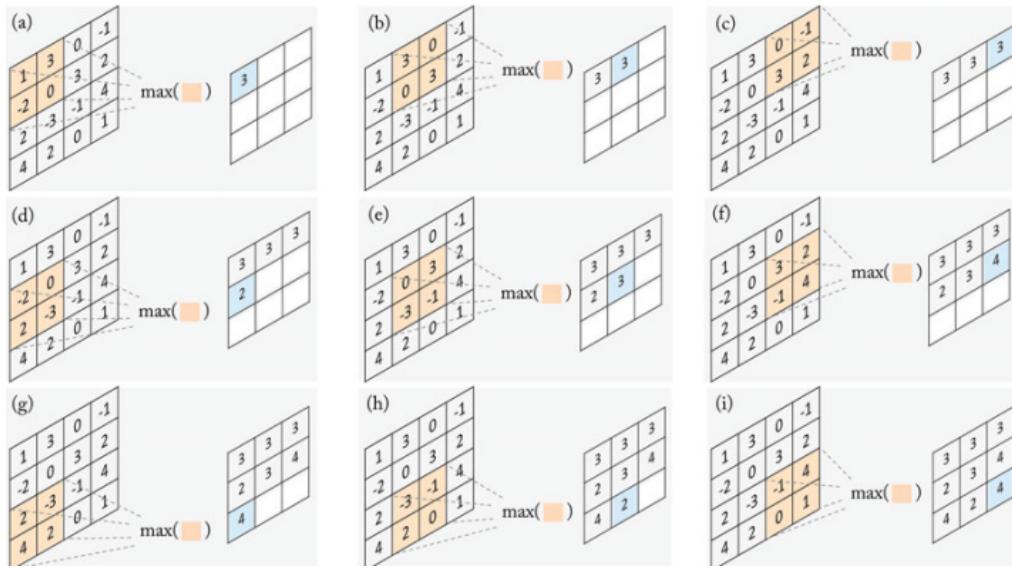


Figura 3.4: Ejemplo de capa de agrupación de tamaño 2 [8].

Tipo de capa en la cual todas sus unidades están conectadas con todas las unidades de la capa anterior. En este tipo de capas se realiza una operación parecida a las capas convolucionales, salvo que en este caso se realiza con una tamaño de filtro de 1, es decir, se realiza la operación dato a dato.

Capas de no linealidad

Capas en las cuales la función de activación de cada unidad toma como valor de entrada un número y da como salida un valor en un rango corto, como puede ser $[0,1]$ o $[-1,1]$. Este tipo de capas suele ir después de las capas «pesadas» como las capas convolucionales o de las capas totalmente conectadas, estas capas son muy importantes utilizarlas después de las capas «pesadas» porque permite mapear espacios no lineales. Algunas de las funciones de activación de esta capa más comunes se pueden ver en la figura 3.5.

ResNet

ResNet o *Residual Network* es un tipo de arquitectura de redes neuronales convolucionales creada por Microsoft [5]. Este tipo de CNN se caracterizan por saltarse las conexiones de identidad en los bloques residuales, para así obtener un modelo mucho más sencillo de entrenar. En un bloque residual, los datos de entrada se duplican, pasando unos por las transformaciones y otros por las conexiones de identidad, estas conexiones, como se puede ver

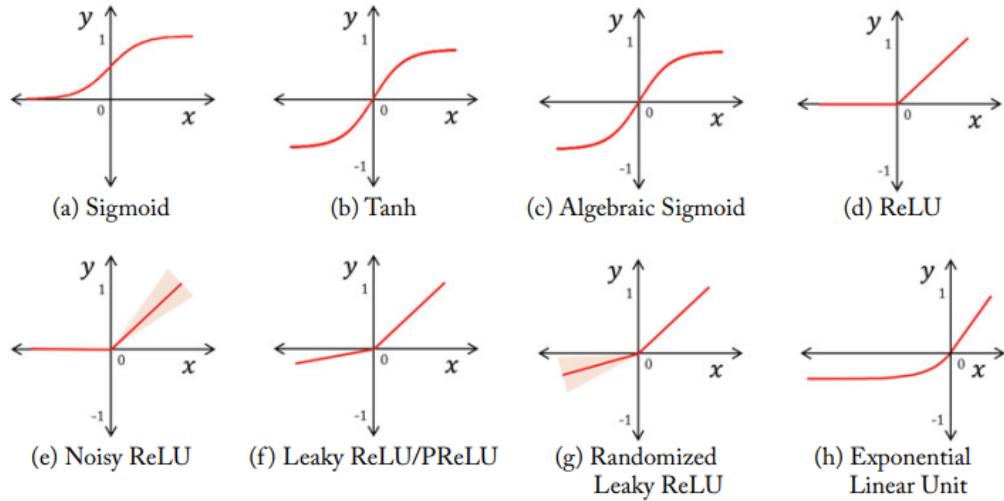


Figura 3.5: Funciones de activación más comunes en capas de no linealidad [8].

en la figura 3.6, sirven para saltarse las capas de transformación, pasando los datos iniciales tal cual llegan.

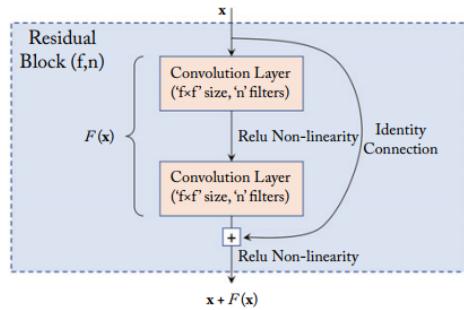


Figura 3.6: Ejemplo bloque residual en ResNet [8].

Region-Based CNN

Las *Region-Based Convolutional Neural Networks* o RCNN son un tipo de redes neuronales convolucionales orientadas al reconocimiento de objetos en imágenes. Este tipo de redes convolucionales, como se puede observar en la figura 3.7 tiene 3 módulos [8]:

- Módulo de extracción de propuestas a región.

- Módulo de extracción de características. Uso de redes neuronales convolucionales para la extracción de características de cada región propuesta.
- Módulo de clasificación de regiones. Se entrena un SVM (Support Vector Machine) por cada clase a predecir, después se usa para clasificar las regiones a partir de las características extraídas en el módulo anterior.

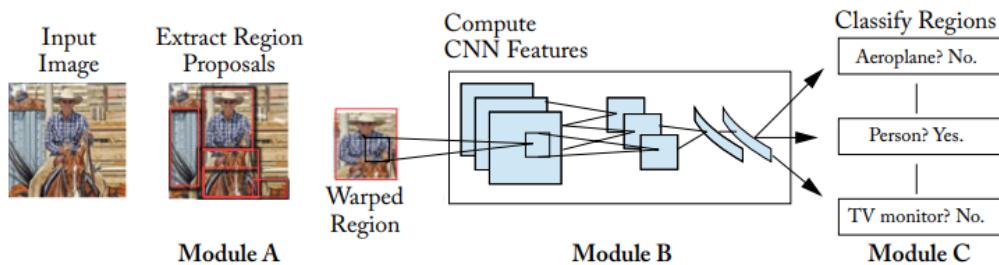


Figura 3.7: Region-Based CNN [8].

FPN

FPN o Feature Pyramid Network es un tipo de red neuronal piramidal que se utiliza junto con las redes neuronales convolucionales para tareas relacionadas con las imágenes, tales como la detección de objetos (usándose junto una RCNN) o la segmentación. Este tipo de redes, como se puede observar en la figura 3.8, tienen dos fases [10]:

- *Bottom-up pathway:* Fase en la cual se utiliza la red neuronal convolucional para extraer características de la imagen, como por ejemplo con ResNet. Cada capa dentro de esta red neuronal convolucional es de la mitad del tamaño que la anterior (por lo que hay que tener en cuenta todos los atributos de las capas de una red neuronal ya comentados). En el ejemplo de ResNet se obtiene la salida de cada capa como el resultado de cada bloque residual.
- *Top-down pathway and lateral connections:* En esta fase, en el primer nivel, se toma como entrada la salida de la última capa de la fase anterior, y se va bajando de nivel pasando al siguiente nivel la predicción del anterior y la salida de la capa de la red neuronal que está al mismo nivel.

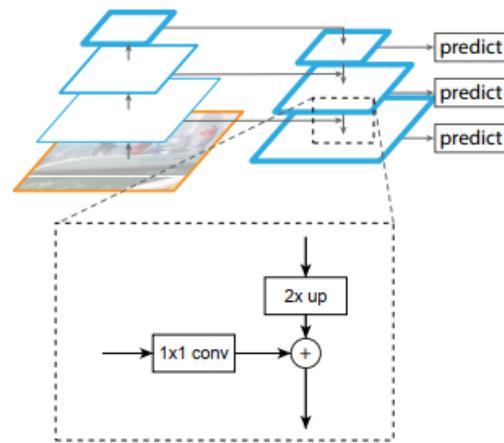


Figura 3.8: Feature Pyramid Network [10].

3.4. Puntos y vectores

Distancia entre vectores, punto medio, ángulo entre vectores

Técnicas y herramientas

En este apartado se van a comentar las distintas técnicas y herramientas utilizadas en el desarrollo del proyecto.

4.1. Visión por computador ~ Detección de movimiento

Existen distintos algoritmos de detección de movimiento, cada uno diferenciado por el uso de distintos lenguajes de programación y diferentes métodos de inteligencia artificial con los que entrenar el modelo. Como en este proyecto se ha querido trabajar en *Python* se han investigado los siguientes algoritmos que están implementados en este lenguaje de programación.

Detectron

Detectron es un proyecto de inteligencia artificial de *Facebook* centrado en la detección de objetos y personas con el uso de algoritmos de *deep learning* a partir de redes neuronales convolucionales [6]. El proyecto se apoya en una de las librerías de inteligencia artificial más usadas en *Python*, *PyTorch*. La mayoría de modelos de *Detectron2* usan redes neuronales convolucionales.

El proyecto cuenta ya con una segunda versión llamada *Detectron2*, en la cual se han mejorado y añadido más modelos [12].

PoseNet

PoseNet es un proyecto de *Google* orientado únicamente al seguimiento del movimiento de las personas centrando su análisis en los puntos claves del

cuerpo humano. El proyecto se basa en la librería de inteligencia artificial de la propia compañía *Google* llamada *TensorFlow*.

4.2. Estación de trabajo

Como ya se ha comentado, el trabajo con vídeos y la visión por computador necesita gran cantidad de capacidad de cálculo. Es por ello que la realización del proyecto en un ordenador personal es casi imposible, o al menos ralentizaría mucho el avance de éste. Es por esta razón que se ha decidido utilizar para el desarrollo del proyecto, sobre todo para la parte de ejecución de los algoritmos, una estación de trabajo de la Universidad de Burgos. Esta estación de trabajo, que se llama Gamma, se encuentra en uno de los despachos del departamento de Ingeniería Informática en la Escuela Politécnica Superior. Las especificaciones de esta estación de trabajo son:

- 3 Nvidia Titan XP.
- Intel Xeon, con 16 núcleos.
- 128GB de RAM.

Aspectos relevantes del desarrollo del proyecto

En este apartado se va a comentar, a manera de resumen temporal, el desarrollo del proyecto. Es en este apartado donde se comentarán las opciones y decisiones tomadas, los problemas surgidos y todos los aspectos importantes. Se ha considerado que la mejor forma de organizar este apartado es en secciones donde se comenta cada apartado del desarrollo.

5.1. Desarrollo FIS-HUBU

FIS-HUBU es proyecto que permite realizar vídeo llamadas entre responsables y pacientes con Parkinson para realizar rehabilitaciones de manera *online*, es decir, sin la necesidad de desplazarse hasta la consulta o el hospital. Este proyecto se ha desarrollado junto con la investigación en la universidad Carlos III, con el proyecto «FUNDACION BURGOS POR LA INVESTIGACION DE LA SALUD COMPLEJO ASISTENCIAL UNIVERSITARIO DE BURGOS PI19/00670 Estudio de factibilidad y coste-efectividad del uso telemedicina con un equipo multidisciplinar para enfermedad de Parkinson».

Esta aplicación, que ha sido desarrollada junto con mi compañero José Luis Garrido Labrador, permite por parte del responsable observar y evaluar la evolución del estado de un paciente, esta evolución también es visible para el paciente que puede ver su propio progreso.

La aplicación puede dividirse en distintas partes que serán comentadas a continuación.

Vídeo llamadas

El punto principal de la aplicación, y por ende su principal uso, son las vídeo llamadas entre pacientes y responsables o terapeutas que permitan sustituir la rehabilitaciones presenciales en consulta por rehabilitaciones *online*, permitiendo así que estás se puedan dar más a menudo y puedan ser accesibles para un mayor número de personas, sobre todo para aquellos pacientes que no se pueden desplazar.

Primero se realizó una investigación sobre las principales plataformas de vídeo llamadas. Lo que se estaba buscando de estas plataformas era:

- Creación de llamadas de manera sencilla y automática. Si es posible a partir de url.
- Vídeo llamada estable sin necesidad de una gran conexión.
- Plataforma que permita grabar la cámara de los pacientes.
- Plataforma gratuita.

Dentro de las plataformas que se investigaron están las más conocidas aplicaciones de este tipo como puede ser *Skype*, pero al final se decidió utilizar *Jitsi* ya que proporciona todas las necesidades anteriormente comentadas, y además permite en un futuro poder crear un servidor propio donde poder modificar parámetros como la calidad de las llamadas.

Responsable

Parte de la aplicación donde los responsables pueden realizar las siguientes tareas:

- Iniciar un vídeo llamada con un paciente.
- Evaluar la evolución de los pacientes.
- Modificar las evaluaciones de los pacientes.
- Comprobar la evolución de los pacientes.

La interfaz de la aplicación para tipo de usuario es sencilla y clara, como se puede ver en el menú principal, figura 5.9, o en el menú de estadísticas, figuras 5.10 y 5.11.



Figura 5.9: Menú principal de un responsable.



Figura 5.10: Menú de estadísticas.

Paciente

Los pacientes que van a utilizar la aplicación, son pacientes de edad avanzada con Parkinson y riesgo de caídas. Esto se ha tenido muy en cuenta tanto en el diseño como en la creación de la parte de la aplicación orientada en los pacientes, se ha intentado que esta parte sea lo más accesible posible, para que todos los pacientes puedan manejarse bien con la aplicación y así poder sacarle el mayor provecho.

Para poder realizar una aplicación lo más accesible posible primero se ha de saber la forma que van a tener los pacientes de interactuar con ésta. En este caso los pacientes van a utilizar un mando de SNES¹ con botones de colores, es por ello que se ha aprovechado estos colores para poder mostrar en la interfaz de la aplicación que botón han de pulsar para realizar que acción. Además, se ha creado un botón de ayuda que carga una página donde se puede ver la acción que realiza cada botón del mando. Un ejemplo de la interfaz de esta aplicación se puede ver en la figura 5.12.

¹SNES: *Super Nintendo Entertainment System*.



Figura 5.11: Ejemplo de la evolución de un paciente vista por un responsable.

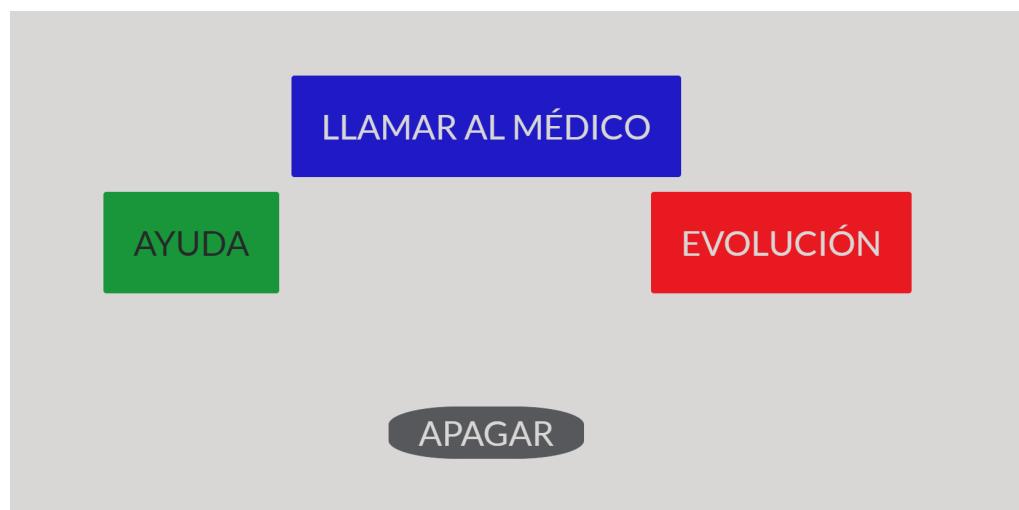


Figura 5.12: Menú principal de los pacientes.

Dispositivos necesarios

Para poder utilizar la aplicación se necesitan distintos dispositivos, que dependiendo del tipo de usuario (responsable o paciente) son unos u otros.

Para los responsables lo único que se necesita es un ordenador con conexión a *Internet* y una cámara conectada a éste.

Por otro lado, se presupone que los pacientes no disponen de ningún dispositivo capaz de cargar la aplicación, es por ello que a cada paciente se le proporciona:

- Dispositivo: MSI - Cubi N 8GL-001BEU N4000 1,10 GHz Western Digital - Green M.2 120 GB Serial ATA III.
- Cámara: Logitech HD Pro WebCam C920.

Conexión

Como ya se ha comentado, el principal uso de la aplicación es las reabilitaciones *online* para pacientes, mayoritariamente de tercera edad, que no se pueden desplazar a las consultas. Muchas de estas personas mayores viven en lugares donde no se tiene contratada ninguna línea de *Internet*, es por ello que además del desarrollo de la aplicación se contrató una serie de *routers* 4G para poder proporcionar conexión a los dispositivos necesarios.

5.2. Investigación algoritmos de visión por computador

Tras haber desarrollado la versión inicial, donde en un futuro se quiere añadir los resultados de este proyecto, se pasó a realizar la primera investigación sobre los distintos algoritmos de visión por computar capaces de detectar y seguir los movimientos de una persona.

De estos algoritmos se necesita:

- Posibilidad de cargar un modelo existente o crear un modelo capaz de detectar a la persona que sale en la imagen.
- Posibilidad de cargar un modelo existente o crear un modelo capaz de detectar la posición de la persona.
- Que el procesado de nuevos fotogramas para detectar a la persona y su posición se realice en poco tiempo.
- Que la salida del procesado de los fotogramas pueda servir para una posterior comparación con el ejercicio base.

Teniendo todos estos factores en cuenta se estudió qué herramientas se pueden usar para esta fase del trabajo. Se buscaron herramientas programadas en *Python* para poder conectarse bien con el resto del proyecto y porque es uno de los lenguajes con los cuales se tiene más soltura, tanto por parte del alumno como por parte de los tutores para resolver dudas y ayudar en los problemas. Las herramientas encontradas fueron:

- *TF-Pose-Estimator*.
- *PoseNet*.
- *Detectron2*.

De cada una de estas herramientas se realizó una investigación y experimentación para probar si cumplían las necesidades requeridas. Al finalizar esta etapa, la única herramienta que permitía realizar todas las necesidades era *Detectron2*. Además, permite con un modelo ya creado por los propios desarrolladores realizar las tareas de predicción de elementos y de la posición de la persona.

Cabe destacar uno de los grandes problemas que surgió en esta fase, y fue justamente con *Detectron2*, la herramienta elegida. El problema era que las versiones de *CUDA* y de *PyTorch* no eran compatibles. El problema se agrandó al estar trabajando en un *workstation* de la universidad como es *Gamma* que utilizan otros investigadores y también por la compleja estructura del propio computador. Tras hablar con el administrador de la computadora, que es uno de los tutores de este trabajo, el doctor Álvar Arnaiz González, se pudo arreglar el problema actualizando la versión de los *drivers* de *CUDA*, y una vez actualizados descargando la versión compatible de *PyTorch*.

5.3. Investigación de *Detectron2*

La fase de investigación de *Detectron2* fue una de las más importantes, y por ende, más costosas en tiempo. Fue en esta fase donde se investigaron las distintas formas que tiene la herramienta para crear o importar modelos con los que poder trabajar. Una vez se seleccionó el modelo se realizó otro estudio para ver qué configuración era la más correcta en el problema tratado.

Al ser una de las fases en las que más tiempo se ha dedicado, también es una de las fases donde surgieron más problemas tanto en la creación de

los modelos, como en la visualización y el almacenamiento de los resultados obtenidos. Todos estos problemas serán comentados en este apartado.

Selección del modelo

Tras haber elegido a *Detectron2* como herramienta de visión por computador para detectar a los pacientes y obtener de estos sus posiciones, elegir qué modelo, dentro de las posibilidades de la herramienta, utilizar para realizar estas tareas.

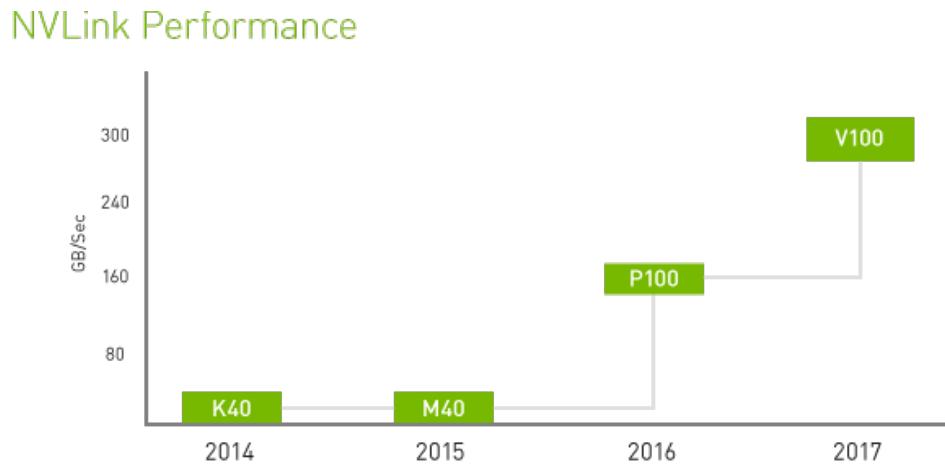
Detectron2 permite dos formas de obtener un modelo:

- Crear un modelo propio a partir de una gran cantidad de datos.
- Importación de modelos ya entrenados.

Ante la falta de datos para la creación de un modelo que diese buenos resultados, y al comprobar en la fase anterior que con la importación de modelos existentes se podían obtener buenos resultados, la investigación se centró en los modelos ya existentes entrenados por los creadores de la herramienta. Estos modelos de redes neuronales fueron entrenados en servidores *Big Basin* de *Facebook*, sucesores de los servidores *Big Sur*, ambos orientados al uso de arquitecturas con varias GPUs potentes para el entrenamiento y uso de algoritmos de inteligencia artificial [4]. En concreto, estos modelos fueron entrenados con 8 Nvidia Tesla V100 con *NVLink*, una tecnología que mejorar las interconexiones entre GPUs proporcionando mayor ancho de banda, haciendo el sistema más escalable [7]. La mejora con otras generaciones de comunicación *inter-GPU* se puede ver en la figura 5.13.

Los modelos creados en *Detectron2* se diferencian en los siguientes tipos:

- COCO Detection with Faster R-CNN (Figura 5.14): modelo que predice los objetos y personas de la imagen.
- COCO Detection with RetinaNet (Figura 5.15): modelo que detecta objetos y personas. Como se puede ver en la imagen de ejemplo la interpretación no es buena aun utilizando un *threshold* alto.
- COCO Detection with RPN and Fast R-CNN: modelos que no se han conseguido probar ya que no siguen la misma estructura que el resto de modelos.



NVLink in Tesla V100 doubles inter-GPU communication bandwidth compared to the previous generation, so researchers can use larger, more sophisticated applications to solve more complex problems.

Figura 5.13: Mejora del rendimiento en GB/s con NVlink [7].

- COCO Instance Segmentation Baselines with Mask R-CNN (Figura 5.16): modelos de detección de objetos y persona, con delimitación en el área ocupada.
- COCO Person Keypoint Detection Baselines with Keypoint R-CNN (Figura 5.17): modelos que detectan a las posiciones y unos puntos claves de ellas, estos puntos son nariz, ojos, orejas, hombros, codos, muñecas, cadera, rodilla y tobillos. Como se puede ver en la imagen de ejemplo, estos modelos predicen bastante bien la posición pero a veces detectan otras cosas como personas, por lo que se posteriormente se vio necesario un estudio sobre el *threshold*.
- COCO Panoptic Segmentation Baselines with Panoptic FPN (Figura 5.18): modelos que segmentación como COCO Instance Segmentation Baselines with Mask R-CNN, obtienen resultados muy parecidos.
- LVIS Instance Segmentation Baselines with Mask R-CNN (Figura 5.19): modelos orientados a la detección de objetos, necesitan un *threshold* bajo para detectar diversos objetos.
- Por último se encuentran modelos como Cityscapes & Pascal VOC Baselines, otras configuraciones y algoritmos de la primera versión de



Figura 5.14: Modelo de tipo COCO Detection with Faster R-CNN, `faster_rcnn_R_50_C4_1x`.

la herramienta *Detectron*. Estos modelos son muy parecidos a los ya mostrados.

Con el estudio de todos los modelos existentes en *Detectron2* se ha comprobado como los únicos modelos que sirven para el problema planteado son COCO Person Keypoint Detection Baselines with Keypoint R-CNN. Dentro de este tipo de modelos existen un total de 4 modelos distintos, para la elección del mejor de estos modelos se realizó otro estudio, esta vez comprobando los tiempos de ejecución, ya que todos los modelos de este tipo realizan unas buenas predicciones (principalmente porque el problema es sencillo, al tener a la persona en el centro de la imagen).

Para realizar este estudio se analizó el tiempo de procesado e impresión de varios vídeos con cada uno de los modelos. Para ello se calculó estos valores para los 4 posibles modelos con un total de 7 vídeos. De este estudio se obtuvieron los resultados que se pueden ver en la tabla 5.1.

Como se puede observar, los datos obtenidos para los modelos son muy parecidos, sobre todo para los dos con menor periodo. Tras observar estos datos se decidió elegir al modelo `keypoint_rcnn_R_50_FPN_3x` (es además el modelo con el que se hicieron las primera pruebas y con el que se decidió a *Detectron2* como herramienta para detectar el movimiento) para utilizarlo

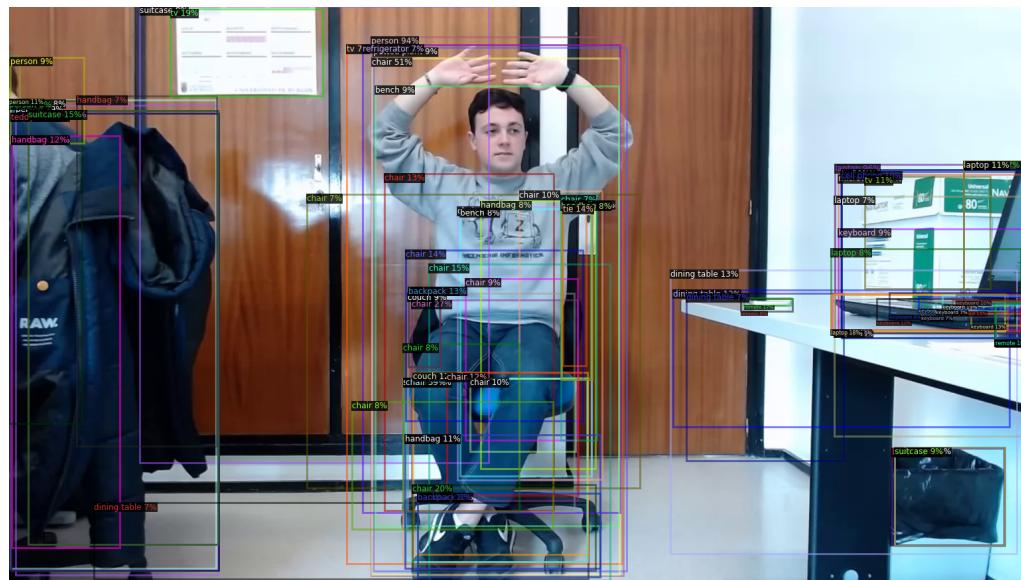


Figura 5.15: Modelo de tipo COCO Detection with RetinaNet, `retinanet_R_101_FPN_3x`.

Modelo	T. Carga (s)	T. Procesamiento (s)	N. Fotogramas	Periodo
keypoint_rcnn_R_50_FPN_3x	9.783479	236.774504	1989	0.119042
keypoint_rcnn_R_50_FPN_1x	8.562254	237.289880	1989	0.119301
keypoint_rcnn_R_101_FPN_3x	13.017239	279.327051	1989	0.140436
keypoint_rcnn_X_101_32x8d_FPN_3x	18.106832	422.024270	1989	0.212179

Tabla 5.1: Tabla con el estudio de los modelos de posición ordenado por ratio.

en el resto del proyecto. Además, en este estudio se ha podido comprobar que los modelos trabajan correctamente sea cual sea el tipo de ropa que lleve la persona o si pasa un objeto por medio de la imagen (normalmente la posición es correcta, pero como se comentará más adelante a veces algunos fotogramas no son correctos), como se pueden ver en la figuras 5.20 y 5.21.

Se puede comentar que el modelo seleccionado usa una red neuronal piramidal de tipo FPN con ResNet y RCNN para la detección de la persona, y dentro de la persona sus puntos más importantes.

Definición del *threshold*

Como se ha comentado en el apartado anterior, es necesario definir un *threshold*, que es un valor entre 0 y 1 que define la sensibilidad del modelo al detectar, es decir, con valores cercanos a 0 el modelo tiende a detectar



Figura 5.16: Modelo de tipo COCO Instance Segmentation Baselines with Mask R-CNN, `mask_rcnn_R_50_DC5_3x`.

más elementos (muchos de ellos erróneos) y cuanto más cercano a 1 menos sensible es, solo detectando los elementos más claros. Sobre este valor se ha realizado un estudio de posibles valores, los valores probados fueron 0.3, 0.5, 0.75 y 0.99.

Con valores bajos del *threshold* se observa que se detectan cosas que no son personas, o se detectan personas que no salen completas en la imagen, como se puede observar en el figura 5.22. Esto ocurre con los valores 0.3, 0.5 y 0.75 que dan la misma salida. Sin embargo, con el valor 0.99, como se puede ver en la figura 5.23, se obtienen muy buenos resultados detectando únicamente a la persona que aparece en el centro de la imagen, con la posición exacta en todos los puntos.

Problemas surgidos

Como se ha comentado a lo largo del apartado, han surgido distintos problemas que se pueden resumir en:

- Problema con los modelos COCO Detection with RPN and Fast R-CNN, que al no tener la misma estructura que el resto de modelos de *Detectron2* no se ha podido probar.



Figura 5.17: Modelo de tipo COCO Person Keypoint Detection Baselines with Keypoint R-CNN, `keypoint_rcnn_R_101_FPN_3x`.

- Se intentó almacenar los vídeos procesados pero por problemas con *OpenCV* no se pudo guardar en esta etapa.

5.4. Cálculo de características

Una vez se seleccionó el modelo que se iba a utilizar durante todo el proyecto y se determinó el parámetro *threshold* de los modelos, el siguiente paso fue la interpretación de las salidas del modelo. Y después de haber interpretado con el modelo de *Detectron2* una posición ser capaces de calcular características que definan de la mejor manera los datos.

Interpretación de las predicciones

Si se analiza la salida tras predecir un imagen con el predictor creado (`DefaultPredictor`), se observa que devuelve un diccionario con una única entrada llamada «instances» donde se almacena toda la información. Dentro de la instancia existen los siguientes apartados:

- `pred_boxes`: Boxes, estructura propia de *Dectectron2* donde se almacenan límites de donde entiende que está la persona en un *tensor*, si se



Figura 5.18: Modelo de tipo COCO Panoptic Segmentation Baselines with Panoptic FPN, `panoptic_fpn_R_101_3x`.

ha detectado más de una persona este `Boxes` estará creado por varios *tensors*.

- **scores:** *tensor* con las probabilidades de que los elementos sean personas. Es este el valor que se compara con el *threshold* para ver si se toma o no como una persona. Si existe más de un valor estos están ordenados de mayor a menor, este es el orden que se sigue en el resto de valores.
- **pred_classes:** *tensor* con el índice de la clase, en este caso todos los elementos son 0 ya que solo detecta personas, que se identifican con este índice.
- **pred_keypoints:** *tensor* con todos los puntos clave de cada elemento (persona) detectada. De cada elemento detectado, si la predicción ha sido correcta, se obtienen un total de 17 puntos, con sus valores *x* e *y*. Después de investigar el posicionamiento de estos puntos se puede decir, como se ve en la figura 5.24, que los puntos representan (teniendo en cuenta que se han detectado todos los puntos y que la persona está mirando de frente al objetivo):
 - 0: nariz.
 - 1 y 2: ojo izquierdo y derecho.



Figura 5.19: Modelo de tipo LVIS Instance Segmentation Baselines with Mask R-CNN, `mask_rcnn_X_101_32x8d_FPN_1x`.

- 3 y 4: oreja izquierda y derecha.
- 5 y 6: hombro izquierdo y derecho.
- 7 y 8: codo izquierdo y derecho.
- 9 y 10: muñeca izquierda y derecha.
- 11 y 12: parte izquierda de la cadera y derecha.
- 13 y 14: rodilla izquierda y derecha.
- 15 y 16: tobillo izquierdo y derecho.

Clase de posición

Una vez se conoce la salida de las predicciones se puede usar esta para obtener más información de la posición de la persona, que en futuras etapas fue usada para la comparación de posiciones. Por ello se creó una clase que almacena todos los puntos detectados además de realizar cálculos para extraer más características.

Para poder obtener más información sobre los puntos se crearon una serie de funciones que permiten el cálculo de:



Figura 5.20: Prueba con chaqueta.

- Cálculo de puntos medios. Este cálculo se realiza para obtener el punto inferior del cuello que se calcula como el punto medio de la recta que une los dos hombros y se utiliza también para calcular el punto central de la cadera a partir de su punto izquierdo y derecho.
- Cálculo de la distancia entre puntos. Este punto se ha realizado sobre todos los pares de puntos unidos. Se pueden utilizar para intuir profundidades.
- Cálculo de los ángulos en grados. Este cálculo se ha realizado en todas las combinaciones posibles (conjunto de 3 puntos consecutivos), ya que se cree que son las características que más información van a aportar debido a que no dependen de ningún otro valor como puede ser la profundidad a la que está el paciente, su altura...

Como se comentó en apartados anteriores, una vez se tienen los puntos, se ha de realizar una serie de comprobaciones para saber si las posiciones son válidas:

- Si existe más de una persona detectada se selecciona la primera, ya que es la que tiene un mayor *score* y por lo tanto el modelo cree que puede ser con mayor probabilidad una persona.

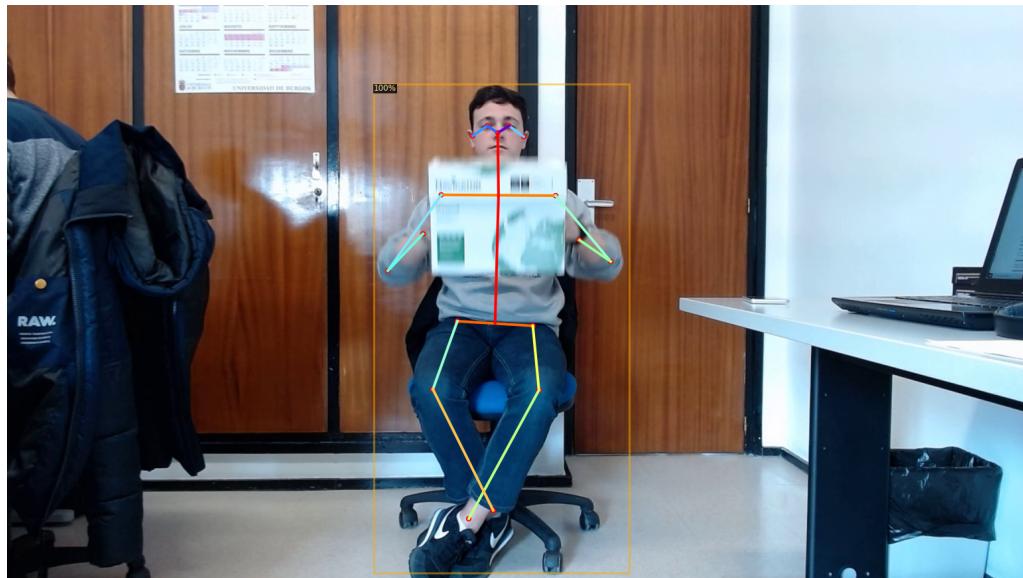


Figura 5.21: Prueba con un objeto de por medio.

- Si una predicción no dispone de todos los puntos no se tiene en cuenta. En la práctica con las pruebas que se han realizado esto solo pasa cuando se pasa un objeto por ciertos puntos del cuerpo, cuando ocurre estos hay algunos fotogramas, muy pocos, que no detectan todos los puntos.

5.5. Primeras versiones comparación de posiciones

Una vez se había comprobado que el sistema de cálculo de posiciones era correcto se pasó a la parte final del desarrollo del proyecto, la comparación de posiciones. Para esta primera versión de la comparación de posiciones se decidió usar los ángulos calculados a partir de los puntos y puntos medios de las posiciones. Se decidió usar los ángulos ya que es la única medida calculada que no depende de la posición del usuario grabado.

División de la comparación

La principal característica, a parte del uso de los ángulos, de la primera versión de la comparación de posiciones fue la división de la comparación en distintas partes para poder dar un peso mayor o menor a cada una de ellas.



Figura 5.22: Prueba con *threshold* a 0.3.

Esto se vio necesario observar los vídeos de ejemplos de los ejercicios que se proporcionaron, ya que había ejercicios donde se centraban en los brazos, otros en las piernas...

La división que se planteó en esta versión es (Figura 5.25):

- Brazos: en esta parte se encuentran únicamente los ángulos de los codos de ambos brazos (5 y 6).
- Torso: ángulos de la cadera con la columna (7 y 8), ángulos de los hombros (3 y 4) y ángulos del cuello con respecto a los hombros (1 y 2).
- Piernas: ángulos de la cadera con el fémur de cada pierna (9 y 10) y ángulos de cada rodilla (11 y 12).

La comparación de cada una de las partes se realiza calculando la suma de los valores absolutos de las restas de los valores entre las dos posiciones.

Problema con los ángulos de los brazos

Tras haber implementado la primera versión se realizaron unas pruebas para comprobar su funcionamiento. En estas pruebas se encontró un fallo

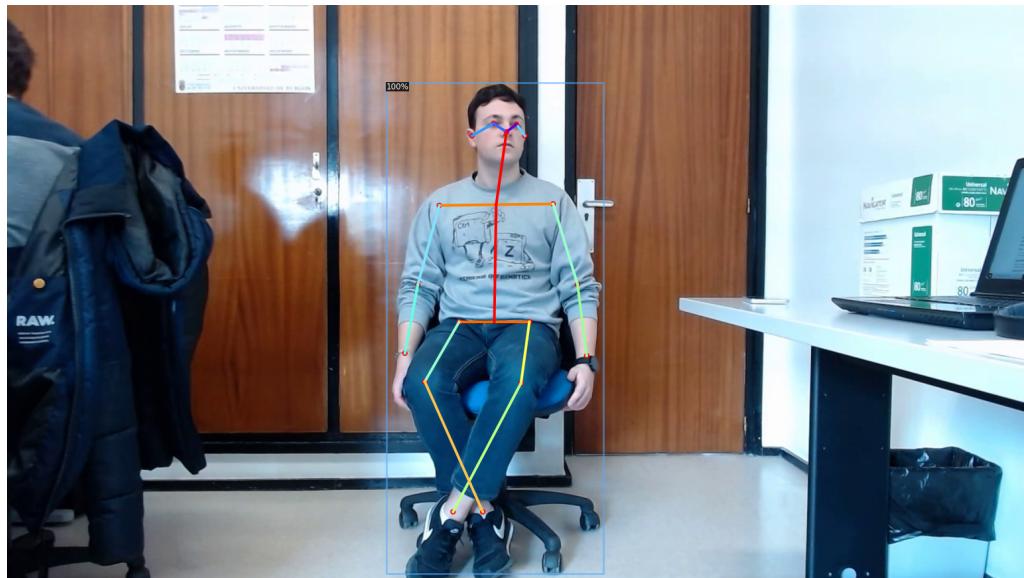


Figura 5.23: Prueba con *threshold* a 0.99.

muy importante, ya que los ángulos que se obtenían eran hasta 180 grados, ya que si se superaba este valor el ángulo se calculaba sobre el otro lado del punto central, es decir, nunca es superior a 180 grados. Es por ello que posiciones como las que se pueden ver en las figuras 5.26 y 5.27 tiene ángulos en ambos codos muy parecidos, aunque su posición sean casi contrarias.

Además, se descubrió que este error con los braozs se daba a demás de en los codos en los ángulos de los hombros, ya que posiciones como las que se pueden ver en las figuras 5.28 y 5.29 obtienen ángulos muy parecidos, por la misma razón que pasaba con los codos.

Para poder solucionar estos problemas lo primero que se hizo fue pasar los ángulos de los hombros que se encontraban en la comparación del torso a la comparación de los brazos. Después, para ambas partes se implemento el siguiente algoritmo para poder penalizar estos casos especiales:

1. Comparar el ángulo (codo y hombro), si este está en un rango de 180 más menos un parámetro se dice que el codo u hombro está estirado y se le da el tipo 0.
2. Si no está en tipo 0 se comprueba la altura del siguiente punto, en el caso del codo el siguiente punto es la muñeca, y en el caso del hombro el punto del codo. Se comprueba entonces si este segundo punto está por encima, y entonces se le da el tipo 1, sino se le da el tipo 2.

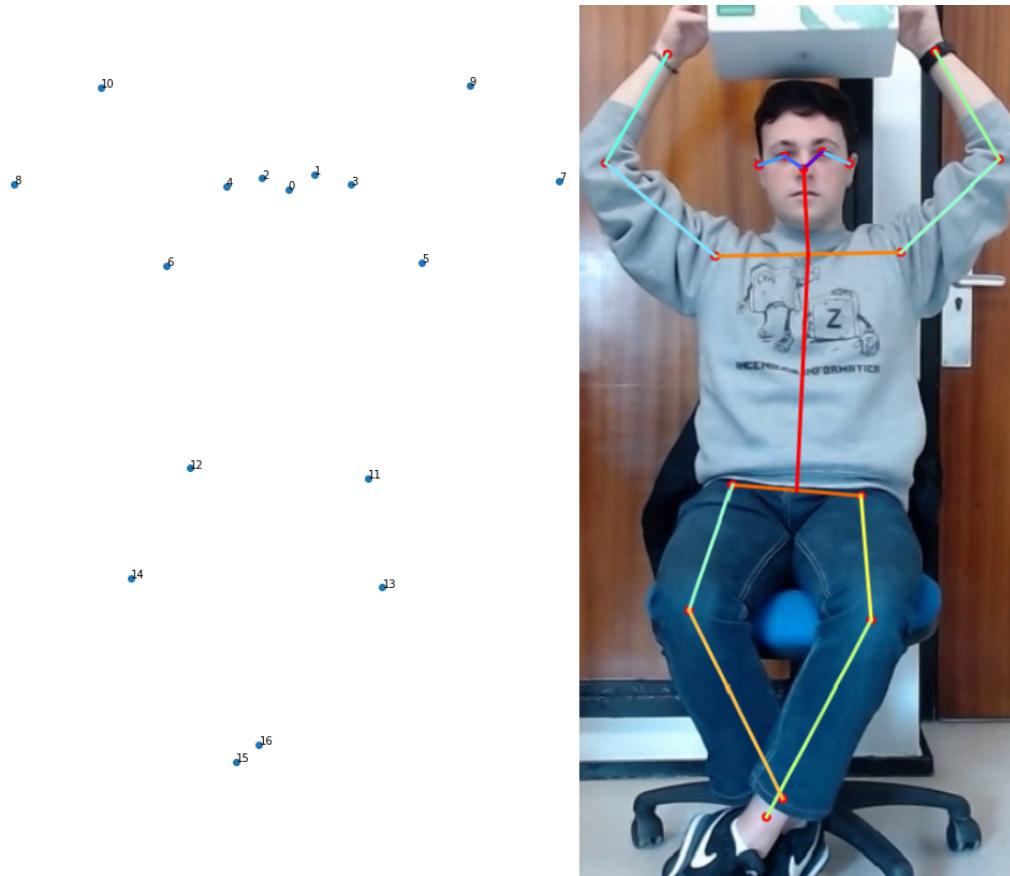


Figura 5.24: Puntos clave predichos con el modelo junto con la etiqueta con su orden.

3. Se comparan los tipos de las dos posiciones que se están comparando. Si cualquiera de las dos posiciones está estirada (tipo 0) entonces no se penaliza sea cual sea el tipo de la otra posición. Por el contrario, si ninguno de las dos posiciones tienen tipo 0, y sus tipos son distintos (uno tiene tipo 1 y le otro tipo 2) se penaliza sumando a la resta del absoluto de los ángulos un valor parametrizado, ya que se encuentran en posiciones muy distintas.

Una vez se implementó esta nueva comparación para los brazos se realizaron una serie de imágenes de prueba para poder confirmar el correcto funcionamiento de la nueva implementación.

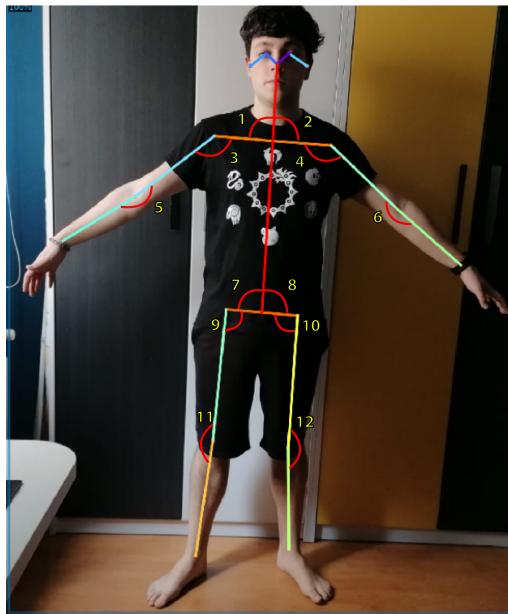


Figura 5.25: Ejemplo ángulos sobre los que se calcula la diferencia entre posiciones.

Umbral y penalización

Como se ha comentado, en esta segunda versión, que se centró en el problema con los brazos, se han utilizado dos parámetros un umbral para definir el rango en el que se considera que un ángulo está estirado o no, y la penalización que se aplica si los tipos son distintos.

Al principio se puso un umbral de 10 grados, que para las imágenes de pruebas que se obtuvieron se recogieron buenos resultados, diferenciando correctamente cuando una de las partes estaba o no estirada. Lo mejor de que este valor sea un parámetro es que se puede modificar dependiendo del ejercicio a realizar.

Por otro lado, al implementar la penalización se utilizó el mismo valor tanto para los codos como para los hombros, con una valor de 180 grados de penalización. Pero en comprobaciones que se realizaron después se observó que penalizar a los hombros con el mismo valor que a los codos no era correcto, ya que la diferencia no era tan grande. Es por ello que se dividió este parámetro de penalización en dos, uno para los codos a 180 y otro para los hombros a 90. Aun así, como pasa con el umbral, estos valores se puede modificar dependiendo del ejercicio a comparar.

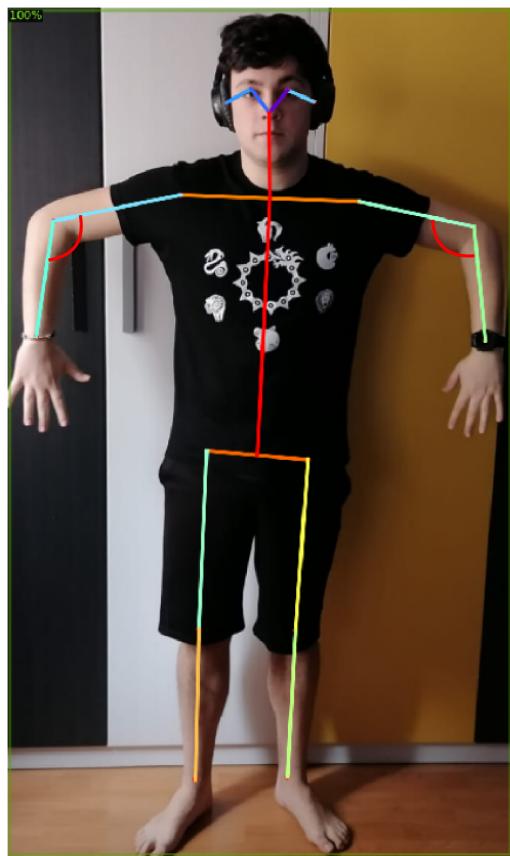


Figura 5.26: Posición con los codos en 90 grados mirando hacia abajo.

5.6. Versión 3 de comparación, uso de medias

Una vez se creía que la comparación de posiciones era correcta había que dar sentido a los valores obtenidos. La forma que se encontró más oportuna para poder realizar esta operación fue el uso de la media de los valores obtenidos en las distintas zonas con las que se trabaja (brazos, piernas y torso).

Además, se implementó el cálculo de la media de tal manera que al aplicar los distintos pesos a las zonas el valor final obtenido siguiese siendo un valor entre 0 y 180 (la distancia máxima que puede haber entre dos ángulos).

Teniendo esto en cuenta el resultado final se calcula con la siguiente

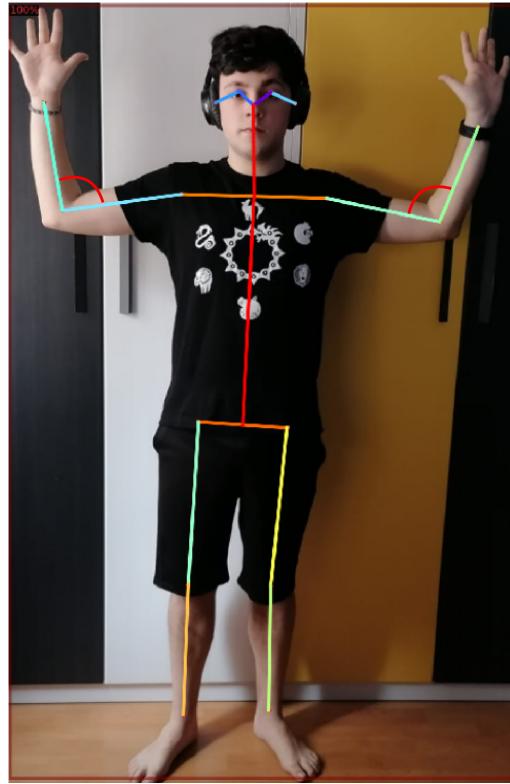


Figura 5.27: Posición con los codos en 90 grados mirando hacia arriba.

fórmula (siendo *total* la suma de los tres pesos):

$$\begin{aligned} resultado = & (pesoBrazos/total) * compBrazos + \\ & (pesoPiernas/total) * compPiernas(pesoTorso/total) * compTorso \end{aligned} \quad (5.9)$$

5.7. Error en brazos, nueva posición y comparación

Una vez se creía que se tenía una versión final para la comparación de posiciones, ya que las pruebas con las imágenes que antes daban problemas se daba un buen resultado, se probó con los vídeos que se habían grabado en la primera recogida de datos de prueba. En esta segunda prueba sobre esta versión se encontró un error muy grave, ya que se penalizaban situaciones en los brazos que nunca se deberían de penalizar.

Estos casos donde se penalizaban eran cuando las manos pasaban de estar

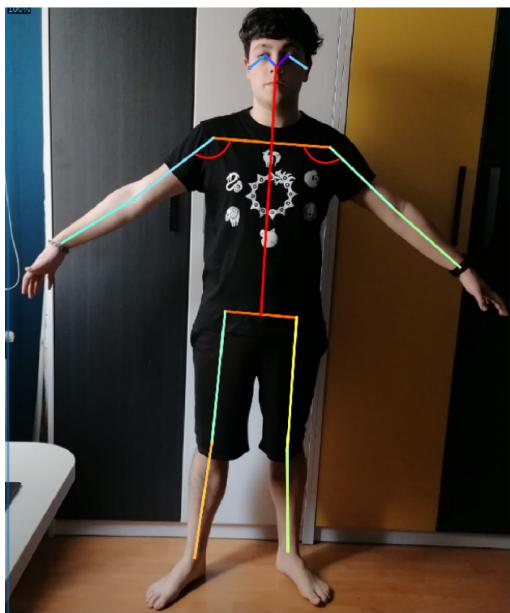


Figura 5.28: Posición con los hombros en 45 grados mirando hacia abajo.

por encima de los codos a pasar por debajo, o viceversa. Un ejemplo de esta situación se puede ver en la figura 5.30, donde se la diferencia en los brazos entre las dos imágenes se debería calcular únicamente con la diferencia de los ángulos de los codos, pero que con la implementación de la versión 3, además de la diferencia entre los ángulos, se sumaría una penalización de 180 grados a la diferencia.

En esta situación, se plantearon una serie de posibles soluciones a este problema ya reincidente. Primero se intentó dar solución al problema manteniendo el sistema de la posición y el resto de comparaciones que daban errores, pero no se ideó una solución plausible que solucionara todos los problemas encontrados en las diversas versiones de la comparación de posiciones. Es por ello que se pasó a hacer una cambio más radical, subiendo a un nivel superior el problema, a la clase de *Python* de la posición.

Nueva versión clase posición

En la primera versión de esta clase, los ángulos se calculaban de forma que nunca eran superiores a 180 grados, lo que daba la necesidad de penalizar ciertas situaciones.

En la segunda versión de la clase se eliminó la condición de que los ángulos fuesen solo de 180 grados (salvo para los ángulos del cuello y de



Figura 5.29: Posición con los hombros en 45 grados mirando hacia arriba.

la cadera con el torso, ya que estos ángulos no pueden ser mayores de 180 por fisionomía humana). Para poder expresar el ángulo en un rango de 360 grados se implementó el siguiente algoritmo que utiliza la función anterior de cálculo del ángulo entre 3 puntos:

```
#p1, p2, p3: puntos a calcular el angulo ,
# p2 debe ser el punto central
#lado: 0 izquierda , 1 derecha
def calcularAngulo( self ,p1,p2,p3,lado ):
    flag=False
    ang = self .calcularAuxAngulo(p1,p2,p3)

    #Cuando la recta que une los dos primeros
    # puntos es una linea vertical
    if (p2[0]-p1[0]) ==0:
        #Lado derecho
        if lado :
            #Si la posicion esta menos a la derecha
            #se ha de cambiar el angulo
```



Figura 5.30: Ejemplo de error en la comparación de brazos.

```

if p3[0]<p2[0]:
    flag=True

#Lado izquierdo
else:
    #Si la posicion esta menos
    #a la izquierda se ha de cambiar el angulo
    if p3[0]>p2[0]:
        flag=True

#Cambio del angulo
if flag:
    ang = 360 - ang
else:
    #Calculo de la ecuacion de la recta sustituyendo
    #el valor de x por el valor del tercer punto
    y = ((p2[1]-p1[1])*(p3[0]-p1[0]))/(p2[0]-p1[0])+p1[1]

#Si esta la tercera parte por encima de la recta que
#hacen las dos primera entonces se cambia el angulo
if p3[1] > y:

```

```
ang = 360 - ang
```

```
return ang
```

Los pasos que sigue este algoritmo es:

1. Calcular el ángulo en un rango de 180 grados, con la implementación de la anterior versión de la clase de la posición.
2. Comprobar si los dos primeros puntos, de los tres estudiados, forman una línea vertical comprobando la resta en sus valores del eje x. Si son iguales ir a 3, sino ir a 4.
3. Si los dos puntos forman una línea vertical no tiene sentido comparar la altura con el tercer punto. Es por ello que lo que se hace es comprobar, dependiendo del lado (izquierda o derecho), la posición del tercer punto analizando si este es más cercano al centro del cuerpo (el ángulo se mantiene igual), o si por el contrario está más al exterior del cuerpo donde se realiza una resta de 360 grados menos el ángulo calculado.
4. Si por el contrario los dos puntos no están en una línea vertical (caso más común) se calcula la ecuación de la recta con estos dos primeros puntos. En esta ecuación de la recta se sustituye el valor de x por el valor de este eje del tercer punto. Por último se comprueba si el valor de y calculado con la ecuación de la recta sustituyendo la x es superior (se mantiene el ángulo) o inferior al valor y del tercer punto donde habría que cambiar el valor del ángulo por 360 menos el ángulo calculado al principio del algoritmo.
5. Se devuelve el valor calculado del ángulo.

Nueva versión comparación

Una vez se había modificado la clase de la posición era necesario hacer una serie de cambios en la comparación de éstas. En este caso, al realizar estos cambios, ya hacía falta «discriminar» la comparación de los brazos, ya que ahora todos se calculan de la misma forma y no se necesita ningún umbral ni penalización. Además, aprovechando que no había que realizar ninguna discriminación, por la zona que se estuviese comparando, se realizó una factorización del código implementado para reducir el código repetido.

Para esta cuarta versión de la comparación de las posiciones se realizaron dos funciones. La primera función permite el cálculo de la media de la

5.7. ERROR EN BRAZOS, NUEVA POSICIÓN Y COMPARACIÓN 47

diferencia entre los ángulos de las zonas pasadas. El código de esta función se puede ver a continuación:

```
#pos1, pos2: posiciones a comparar
#zonas: lista de las zonas a comparar
# zonas sin especificar la parte izq o drch
def comparacionZona(pos1, pos2, zonas):
    #Partes derecha e izquierda
    partes = ["D", "I"]
    #Resultado iniciado a 0
    res = 0.0

    #Recorrer las partes
    for i in partes:
        #Recorrer las zonas
        for j in zonas:
            #aux es la diferencia entre los angulos
            aux = abs(eval("pos1." + j + i) - eval("pos2." + j + i))

            #Si la diferencia es mayor de 180
            # grados se coge el otro lado para
            # mantener la distancia minima
            if aux > 180:
                res += (360 - aux)
            else:
                res += aux

    #Se devuelve el valor final entre el
    #numero de elementos sumados
    return res / (len(partes) * len(zonas))
```

Como se puede ver en el código, esta implementación permite la comparación de todas las zonas de la posición y siempre devuelve una diferencia media entre las zonas comparadas que nunca supera los 180 grados, es decir, siempre devuelve la distancia mínima entre los ángulos.

Pero para poder dar distintos pesos a las distintas zonas a comparar se implementó la segunda función que llama a la función anterior, **comparacionZona**. El código de esta función final que devuelve la diferencia media aplicando los pesos se puede ver a continuación:

```
#pos1, pos2 posiciones a comparar
#pesos: diccionario con los pesos de
```

```

# cada zona
def compararPosiciones (pos1 , pos2 , pesos={ "brazos" :1 ,

zonas={ "brazos" :[ "angCodo" , "angHombro" ] ,
        "piernas" :[ "angRodilla" , "angCadera" ] ,
        "torso" :[ "angCaderaTorso" , "angCuelloSup" ] }
res=0
total=0

#Se recoge el peso total
for i in pesos:
    #Si no esta en las keys de la zona se lanza
    # una excepcion
    if i not in zonas:
        raise Exception( "Excepcion" )
    total+=pesos [ i ]

#Se recorren los distintos tipos de zonas
# y se les aplica el peso a la comparacion
for i in zonas:
    res+=(pesos [ i ] / total)*
                comparacionZona( pos1 , pos2 , zonas [ i ] )

#Calculo del porcentaje entorno a 180
porcentaje = res*100/180

return res ,100-porcentaje

```

Como se puede ver en el código, la implementación recorre las distintas zonas de las cuales calcula la comparación entre las posiciones y le aplica el peso considerado. Además, una vez calculada la diferencia final, al estar está entre 0 y 180 grados por utilizar las medias, se calcula el porcentaje inverso de exactitud entre las dos posiciones. Este porcentaje se interpreta como el porcentaje de exactitud entre las posiciones, siendo el 100 % el valor máximo donde las dos posiciones son totalmente iguales.

5.8. Comprobación de la última versión

Una vez se terminó de implementar estas nuevas versiones se probaron con las imágenes de la sección 5.5 que la implementación funcionaba correctamente con los datos que las primeras dos versiones tenían grandes problemas. Después, para poder probar el funcionamiento y sobre todo el rendimiento de la implementación se desarrolló unas funciones que permiten recorrer todos los vídeos de una carpeta y de cada vídeo comparar un fotograma con el siguiente de ese mismo vídeo (objetivo principal del proyecto). Cabe comentar en que esta comparación se ha dado el mismo peso a todas las zonas comparadas. Gracias a estas funciones se pudo obtener los siguientes datos de cada uno de los vídeos:

- Media de la diferencia de los grados en todo el vídeo.
- Diferencia máxima en grados en todo el vídeo.
- Diferencia mínima en grados en todo el vídeo.
- Desviación típica entre las diferencias de los grados.
- Media del porcentaje de exactitud.
- Porcentaje máximo de exactitud.
- Porcentaje mínimo de exactitud.
- Desviación típica de los porcentajes.
- Número de fotogramas en el vídeo.
- Tiempo de obtención de las posiciones del vídeo.
- Tiempo de comparación de las posiciones del vídeo.
- Tiempo de cálculo de estadísticas de las comparaciones del vídeo.
- Tiempo total.
- Tiempo total empleado por fotograma.

Los resultados obtenidos en esta fase se pueden ver en:

- Resultados diferencia en grados: tabla 5.2.
- Resultados diferencias en porcentajes: tabla 5.3.

Vídeo	MediaGrados	GradosMáximos	GradosMínimos	DesviaciónGrados
depie	5,26	34,89	0,46	5,93
sentado1	2,68	26,07	0,05	3,01
sentado6-camiseta	2,36	5,27	0,32	0,96
sentado2-cruzado-480	2,74	14,59	0,58	1,72
sentado4-remangado	2,78	23,80	0,22	2,96
sentado2-cruzado	2,79	15,35	0,40	1,81
sentado3-caja	2,62	34,32	0,15	4,13
sentado5-chaqueta-abierta	2,96	17,29	0,42	1,94

Tabla 5.2: Tabla con las estadísticas de las diferencias en grados.

Vídeo	PorcMedio	PorcMáximo	PorcMínimo	DesvPorcentaje
depie	97,08	99,74	80,61	3,29
sentado1	98,51	99,97	85,52	1,67
sentado6-camiseta	98,69	99,82	97,07	0,53
sentado2-cruzado-480	98,48	99,68	91,89	0,95
sentado4-remangado	98,46	99,88	86,78	1,65
sentado2-cruzado	98,45	99,78	91,47	1,01
sentado3-caja	98,54	99,92	80,93	2,29
sentado5-chaqueta-abierta	98,35	99,76	90,39	1,08

Tabla 5.3: Tabla con las estadísticas de las diferencias en porcentajes de exactitud.

Vídeo	NFrames	TiempoObtPos (s)	TiempoComp (s)	TiempoEst (s)	TiempoTotal (s)	Tiempo/Frame
depie	258	26,704819	0,045619	0,000168	26,750605	0,103685
sentado1	445	49,113378	0,075400	0,000146	49,188924	0,110537
sentado6-camiseta	194	21,433898	0,027136	0,000100	21,461134	0,110624
sentado2-cruzado-480	286	31,532470	0,039944	0,000111	31,572524	0,110393
sentado4-remangado	219	23,847520	0,030736	0,000101	23,878356	0,109034
sentado2-cruzado	286	32,729511	0,040728	0,000121	32,770360	0,114582
sentado3-caja	299	32,811822	0,042313	0,000112	32,854246	0,109880
sentado5-chaqueta-abierta	281	30,815606	0,039650	0,000108	30,855364	0,109806

Tabla 5.4: Tabla con los tiempos de la ejecución.

- Resultados tiempos de ejecución: tabla 5.4.

Como se puede observar, los resultados comparando los fotogramas de un mismo vídeo dan muy buenos resultados, con medias de porcentaje de exactitud que no bajan del 97 %, lo que significa que ninguna comparación tiene una media superior a 5,4° de diferencia.

Si se observan los porcentajes máximos de exactitud y los grados mínimos de diferencia se observan valores muy positivos, siendo los porcentajes muy cercanos al 100 % y los grados muy similares a 0°, como cabría de esperar ya que en todo el vídeo es muy probable que haya al menos dos fotogramas

contiguos que se difieran muy poco al no haber casi movimiento.

Otro dato importante, que principalmente puede servir para detectar fallos en la posición obtenida por la red neuronal, es el porcentaje mínimo de exactitud y el correspondiente grado máximo de diferencia. Observando estos datos se puede ver como los grados máximos de diferencia entre dos fotogramas de todos los vídeos es de unos 35° , que ocurren en el vídeo donde se está de pie. Tras observar en que fotogramas se haya este máximo se vio como este valor se debía a un pequeño fallo en la detección de la posición por parte de la red neuronal. Pero este «problema» no es importante debido a que este tipo de movimientos no se realizan en los ejercicios de ejemplos de las rehabilitaciones reales a personas con *Parkinson*, y a que no afecta en gran medida (debido al gran número de fotogramas) a las otras estadísticas, sobre todo a la más importante, la media.

Por último, tras haber comprobado que los algoritmos daban unos resultados bastante positivos, se analizaron los tiempos de ejecución, ya que no había que olvidarse que este proyecto se iba a juntar con el José Luis Garrido Labrador, para poder realizar estas operaciones en un flujo de imágenes. Es por ello que los algoritmos, además de usar la mínima capacidad de memoria posible, debían de ser rápidos. Este tipo de estudios ya se habían realizado para obtener el mejor modelo de red neuronal que proporcionase los mejores resultados posibles en el menor tiempo posible, por lo que ahora había que comprobar el tiempo de ejecución de todo el proceso, es decir, el cálculo de la posición a partir de la salida de la red neuronal, después la comparación de las posiciones obtenidas y por último el cálculo de estadísticas que permitan obtener un resultado legible y comprensible.

Observando los tiempos de ejecución, tabla 5.4, se puede ver como sin lugar a duda la ejecución del modelo de la red neuronal y el posterior cálculo de la posición es donde se invierte más tiempo (aun teniendo el modelo ya cargado en memoria como se haría en el despliegue final con la estructura de flujos del compañero). Si se mira al resto de tiempos se puede ver como la implementación de la comparación de posiciones es muy óptima, siendo el tiempo invertido en esta tarea en el vídeo más largo tan solo de 0,07 segundos. Por último, el cálculo de estadísticas sobre las comparaciones realizada también es muy rápida, no superando en ninguno de los vídeos los 0,00017 segundos.

Como se ha comentado, tras realizar todas las pruebas, se concluyó en que estas versiones del cálculo de posición y de comparación de posiciones serían las definitivas. Es por ello que interesaba saber el tiempo total invertido para realizar el proceso de obtener la posición de un fotograma, comparar

esta posición con la siguiente y obtener las estadísticas de esta comparación. Como se puede ver en la tabla 5.4, el tiempo medio de procesamiento de cada fotograma es de 0,10981756537573605 segundos, un resultado muy bueno si se tiene en cuenta todo el proceso que lleva detrás.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final de máster no parece tan obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Apéndices

Apéndice A

Plan de Proyecto Software

A.1. Introducción

La planificación de un proyecto es una fase esencial en desarrollo de éste, ya que permite comprobar y guiar el proyecto según las necesidades actuales y futuras.

Dentro de este plan para el desarrollo del proyecto se pueden diferenciar dos puntos sobre los que se puede enfocar este estudio:

- **Temporal:** enfoque en el que se analiza la evolución del proyecto en el tiempo. Al utilizar una metodología SCRUM, se ha dividido el desarrollo en *sprints* de 2 semanas.
- **Viabilidad:** comprobación de la adecuación del proyecto tanto desde el ámbito económico como desde el ámbito legal.

A.2. Planificación temporal

Como ya se ha comentado, la planificación temporal se ha dividido en distintos *sprints* de 2 semanas cada uno, en los cuales se realizaron distintas tareas y reuniones. Cabe destacar que antes de realizar el primer *sprint* se desarrolló la aplicación FIS sobre la cual será implementado en el futuro el proyecto desarrollado.

Sprint0

En este *sprint* se creó la aplicación para el Hospital Universitario de Burgos llamada FIS-HUBU. Esta aplicación, como se ha visto en el apartado 5.1, sirve para realizar vídeo llamadas entre médicos y los pacientes con Parkinson en las cuales se realizan rehabilitaciones para mejorar su calidad de vida. Como se ha explicado, esta aplicación cuenta con dos partes, una para los médicos o responsables y otra para los pacientes. Las tareas realizadas en este *sprint* son las siguientes:

- Investigación sobre plataformas de vídeo llamadas.
- Creación de la estructura base de la página web.
- Diseño e implementación de la base de datos.
- Creación del login por usuario, guardando una *cookie* para mantener la sesión.
- *Backend* de las páginas de los pacientes, donde se incluye la creación de las llamadas, el calculo de la evolución a partir de los datos en la base de datos, la grabación de la cámara del paciente y subida a un servidor propio de la universidad...
- *Backend* de las páginas de los responsables.

Tras desarrollar la aplicación se empezaron con las tareas de explotación de la aplicación, en donde se cogieron los dispositivos para los paciente y se les instalaron todo el *software* necesario para funcionar correctamente, y con ello poder grabar los vídeos y subirlos a un servidor propio. Debido a la aparición del COVID-19 y el posterior confinamiento sufrido y las consiguientes medidas de seguridad no se pudo empezar a instalar los dispositivos en las casa de los pacientes.

Sprint 1: 17/02/2020 - 26/02/2020

Este es el primer *sprint* real del desarrollo de la parte del proyecto de visión artificial, en el se realizaron las primeras tareas de creación del repositorio en *Github* y la creación de la estructura del mismo. Además, se creo el documento *LATEX* a partir de la plantilla de la asignatura. Las tareas más relevantes en este primer *sprint* fueron las relacionadas con la configuración de *Gamma* y los primeros pasos en la investigación de algoritmos de visión por computador y detección de movimiento, como son *Detecron2* y *PoseNet*.

Sprint 2: 27/02/2020 - 11/03/2020

En este *sprint* se realizó una tarea fundamental dentro del desarrollo del proyecto, que es la investigación y elección del algoritmo de visión artificial para la obtención de la postura, que se usará en los fotogramas de los vídeos recogidos con la aplicación FIS-HUBU y compararlos con otros vídeos base para saber la exactitud del ejercicio. Esta tarea de investigación de los distintos algoritmos dio como resultado que el *Detectron2* (apartado 4.1) es el algoritmo que más se amoldaba al problema del proyecto.

Además, en este *sprint* se realizaron las tareas de documentación necesarias sobre los algoritmos candidatos y sobre la selección del mejor.

Sprint 3: 12/03/2020 - 02/05/2020

En este tercer *sprint* se profundizó en la investigación de los distintos algoritmos y modelos de *Detectron2*. En este *sprint* se realizaron las siguientes tareas:

- Documentación de los primeros objetivos del proyecto.
- Estudio de la herramienta *Detectron2*.
- Estudio de los modelos ya creados en *Detectron2*.
- Estudio sobre los modelos de detección de posición de *Detectron2*.
- Estudio del *threshold*.
- Interpretación de la salida obtenida.
- Creación de la clase de posición y extracción de características a partir de los datos devueltos por el modelo.
- Documentación de los aspectos relevantes.

Sprint 4: 03/05/2020 - 05/06/2020

Este *sprint* se realizaron tareas muy importantes tanto en la documentación como en la implementación, ya que se documentó el concepto teórico más importante, las redes neuronales convolucionales, y se implementó las primeras versiones de la comparación de posiciones. Las principales tareas que se realizaron en este *sprint* son:

- Documentación de los conceptos teóricos, donde destaca la documentación teórica de las redes neuronales convolucionales que tiene el mayor peso teórico del proyecto.
- Documentación de las técnicas y herramientas usadas hasta el momento.
- Cambios en la memoria de la primera revisión por parte de los tutores.
- Desarrollo de la primera versión de comparación de posiciones.
- Implementación de la corrección del *bug* en la comparación de los brazos.
- Creación de imágenes para probar la comparación de posiciones.
- Test con las imágenes obtenidas.
- Documentación de los aspectos relevantes ocurridos en este *sprint*.

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

- B.1. Introducción
- B.2. Objetivos generales
- B.3. Catalogo de requisitos
- B.4. Especificación de requisitos

Apéndice C

Especificación de diseño

- C.1. Introducción
- C.2. Diseño de datos
- C.3. Diseño procedimental
- C.4. Diseño arquitectónico

Apéndice D

Documentación técnica de programación

- D.1. Introducción**
- D.2. Estructura de directorios**
- D.3. Manual del programador**
- D.4. Compilación, instalación y ejecución
del proyecto**
- D.5. Pruebas del sistema**

Apéndice E

Documentación de usuario

- E.1. Introducción**
- E.2. Requisitos de usuarios**
- E.3. Instalación**
- E.4. Manual del usuario**

Bibliografía

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jo-nathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] Formación audiovisual. El cine y los fotogramas por segundo. <https://www.formacionaudiovisual.com/blog/cine-y-tv/el-cine-y-los-fotogramas-por-segundo-fps/>, jan 2017.
- [3] DeepLizard. Zero padding in convolutional neural networks explained. https://deeplizard.com/learn/video/qSTv_m-KFk0.
- [4] Facebook Engineering. Introducing big basin: Our next-generation ai hardware. <https://engineering.fb.com/data-center-engineering/introducing-big-basin-our-next-generation-ai-hardware/>, mar 2017.
- [5] Baki Er. Microsoft presents : Deep residual networks. <https://medium.com/@bakiiii/microsoft-presents-deep-residual-networks-d0ebd3fe5887>, aug 2016.

- [6] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. <https://github.com/facebookresearch/detectron>, 2018.
- [7] Nvidia. Nvlink y nvswitch: Los elementos fundamentales de la comunicación multi-gpu avanzada. <https://www.nvidia.com/es-es/data-center/nvlink/>.
- [8] Syed Afaq Ali Shah Mohammed Bennamoun Salman Khan, Hossein Rahmani. *A Guide to Convolutional Neural Networks for Computer Vision*. Gérard Medioni & Sven Dickinson. Morgan & Claypool.
- [9] TensorFlow. Pose estimation. https://www.tensorflow.org/lite/models/pose_estimation/overview, feb 2020.
- [10] Ross Girshick Kaiming He Bharath Hariharan Tsung-Yi Lin, Piotr Dollár and Serge Belongie. Feature pyramid networks for object detection. http://openaccess.thecvf.com/content_cvpr_2017/html/Lin_Feature_Pyramid_Networks_CVPR_2017_paper.html, 2017.
- [11] Wikipedia. Visión artificial — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Visi%C3%B3n_artificial&oldid=120821222, 2019.
- [12] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. <https://github.com/facebookresearch/detectron2>, 2019.
- [13] Sepp Hochreiter y Jürgen Schmidhuber. Long short-term memory, nov 1997.